GRASP heuristics for discrete and continuous global optimization

Tutorial given at the Spring School in Advances in Operations Research, Higher School of Economics
Nizhny Novgorod, Russia ✤ May 3, 2011

Mauricio G. C. Resende
AT&T Labs Research
Florham Park, New Jersey

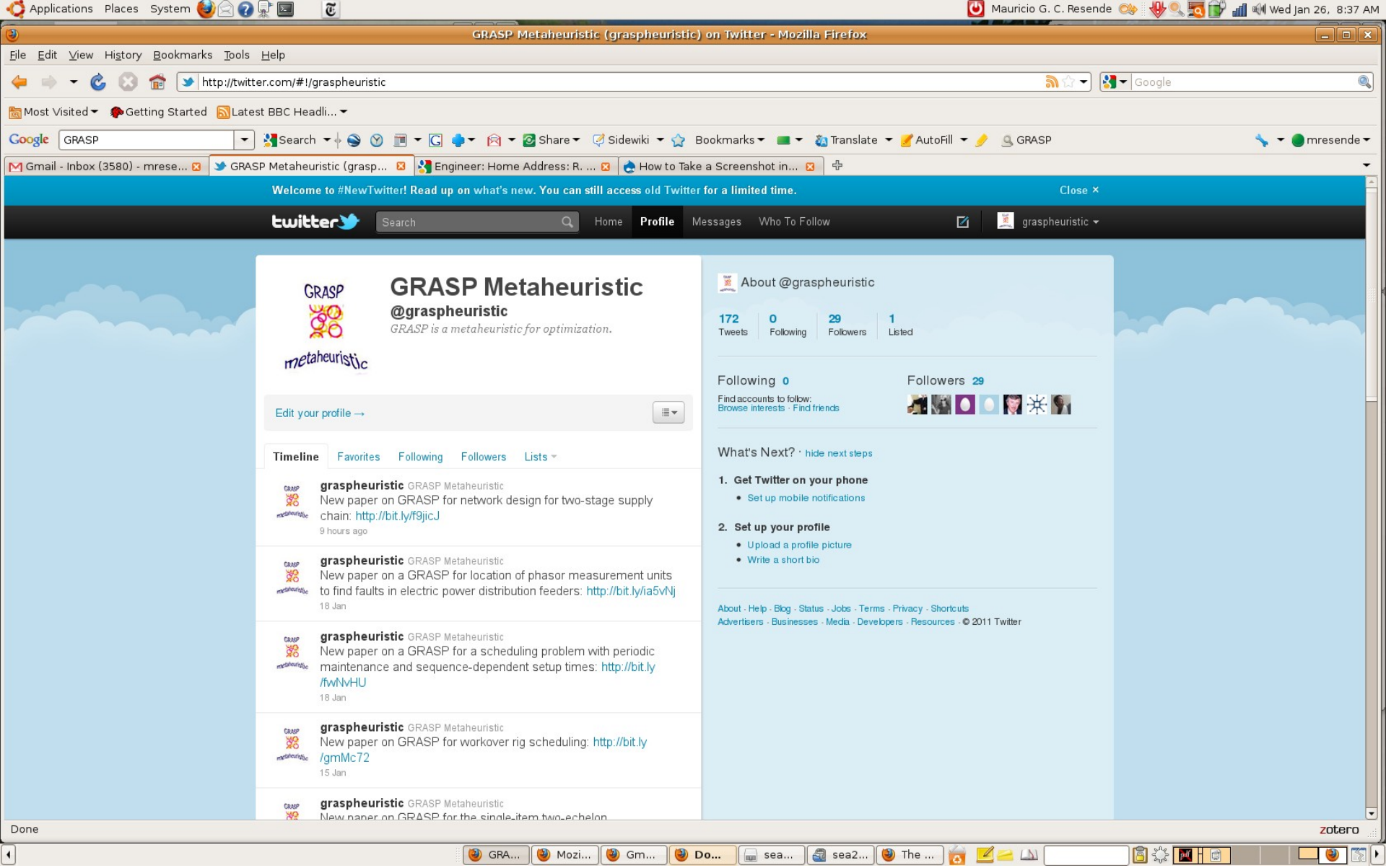
mgcr@research.att.com
http://www2.research.att.com/~mgcr

Google Scholar Search: "greedy randomized adaptive search"
(http://scholar.google.com)

year	cumul. papers	year	Cumul. papers
1990	1	2001	402
1991	7	2002	533
1992	11	2003	661
1993	16	2004	803
1994	34	2005	1,010
1995	54	2006	1,220
1996	89	2007	1,470
1997	126	2008	1,770
1998	196	2009	2,130
1999	256	2010	2,440
2000	308	2011 (to Apr. 26th)	3,400

Annotated bibliographies of GRASP

- P. Festa and M.G.C. Resende, *GRASP: An annotated bibliography*, Essays and Surveys on Metaheuristics, C.C. Ribeiro and P. Hansen, Eds., Kluwer Academic Publishers, pp. 325-367, 2002
- P. Festa and M.G.C. Resende, *An annotated bibliography of GRASP–Part I: Algorithms*, International Transactions in Operational Research, vol. 16, pp. 1-24, 2009.
- P. Festa and M.G.C. Resende, *An annotated bibliography of GRASP–Part II: Applications*, International Transactions in Operational Research, vol. 16, pp. 131-172, 2009.



Follow GRASP on Twitter: http://twitter.com/graspheuristic

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

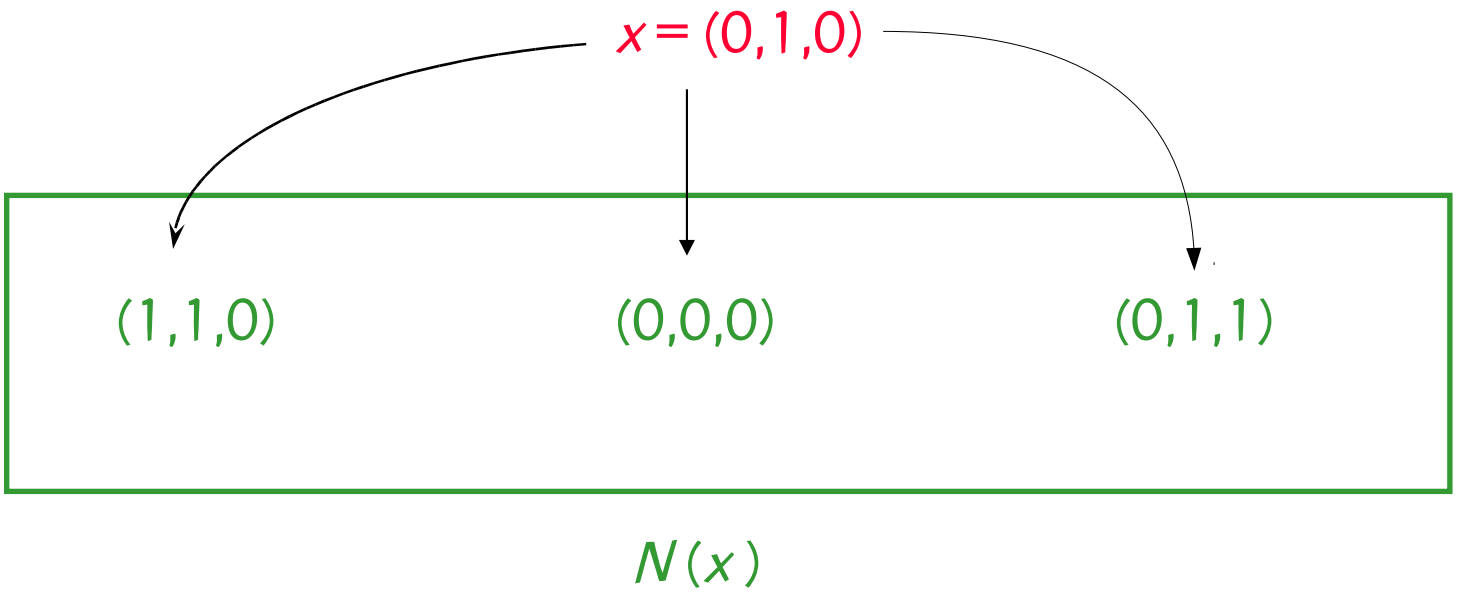
Local Search

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.

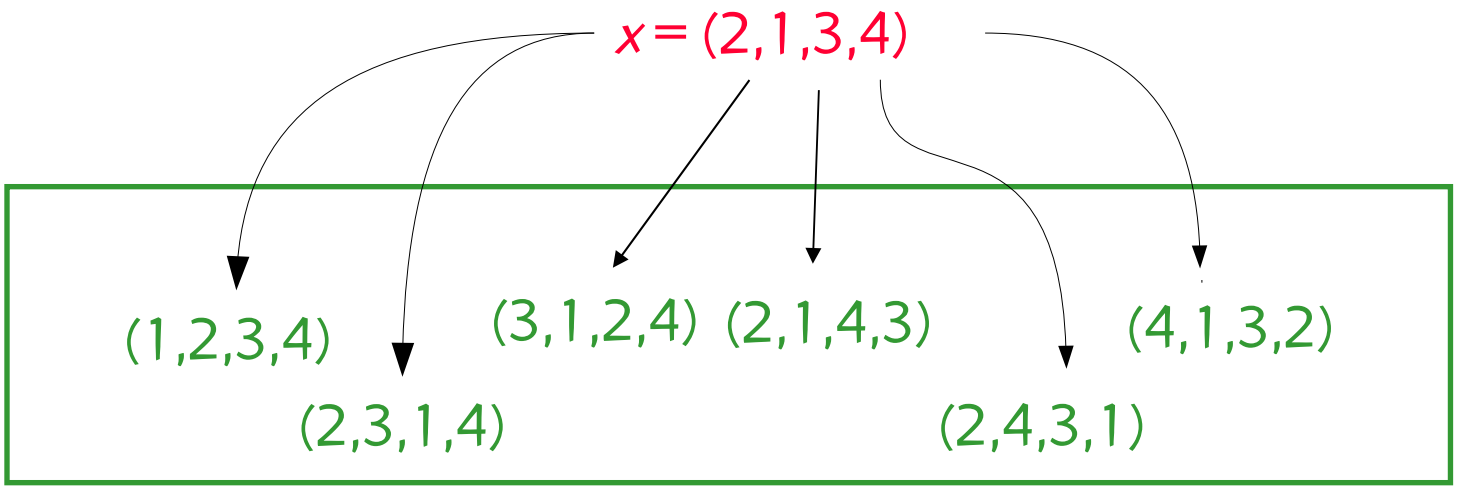
Local Search Neighborhoods

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:

$x = x_0$;

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

    $x = y$;

}

check for better solution in neighborhood of

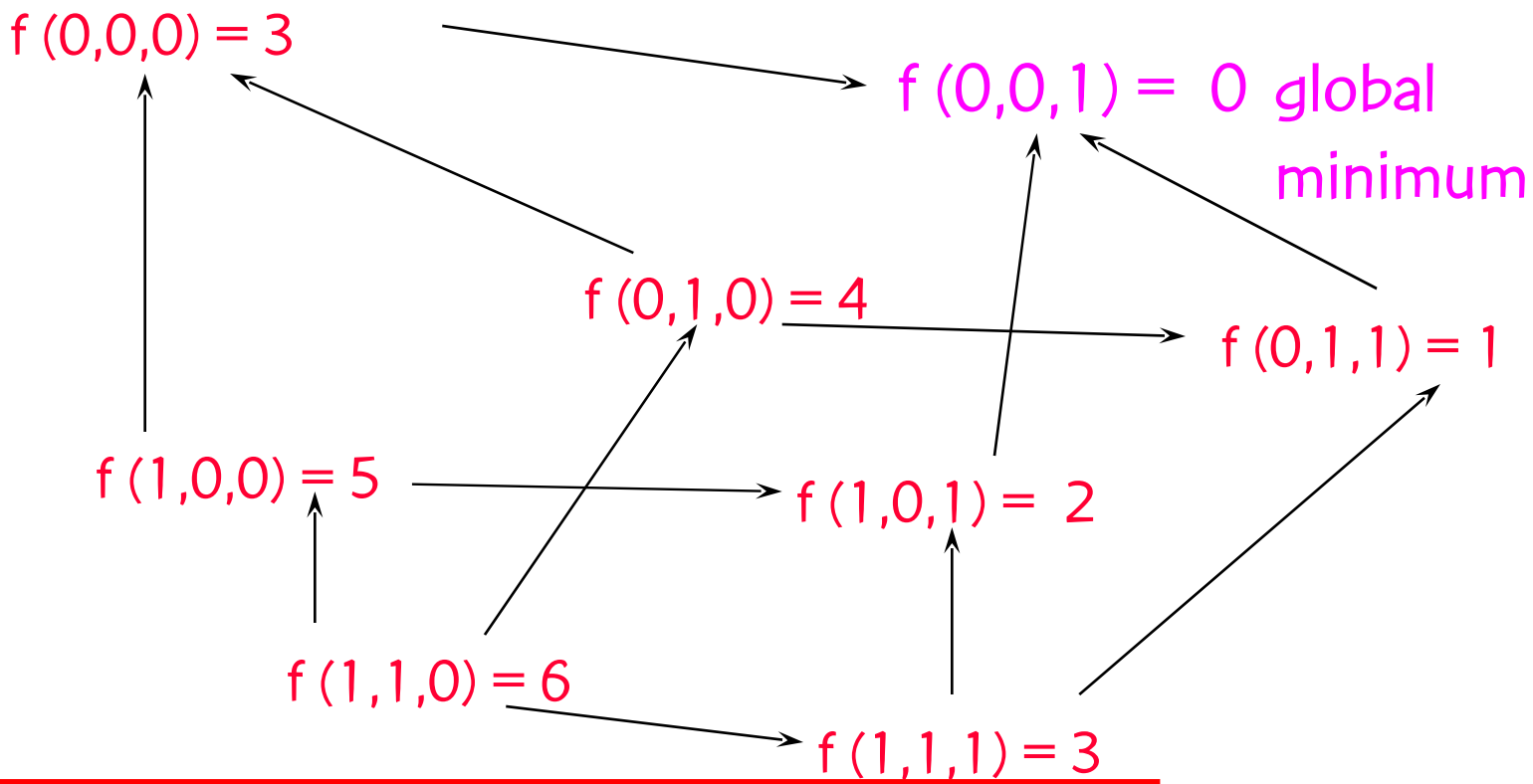
move to better solution *y*

Time complexity of local search can be exponential.

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

Local Search

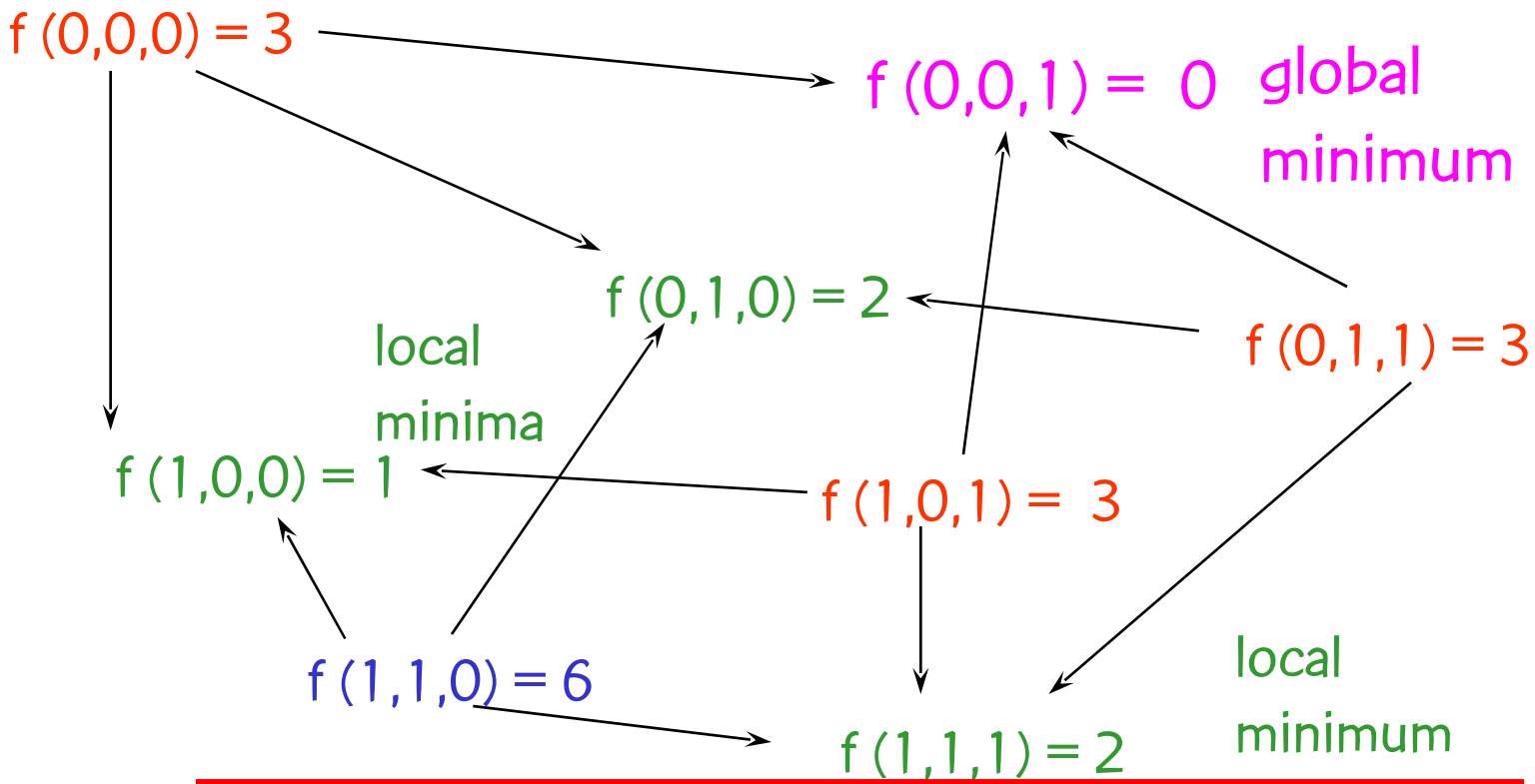
(ideal situation)



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.

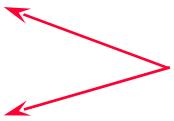
Local Search

Effectiveness of local search depends on several factors:

neighborhood structure

function to be minimized

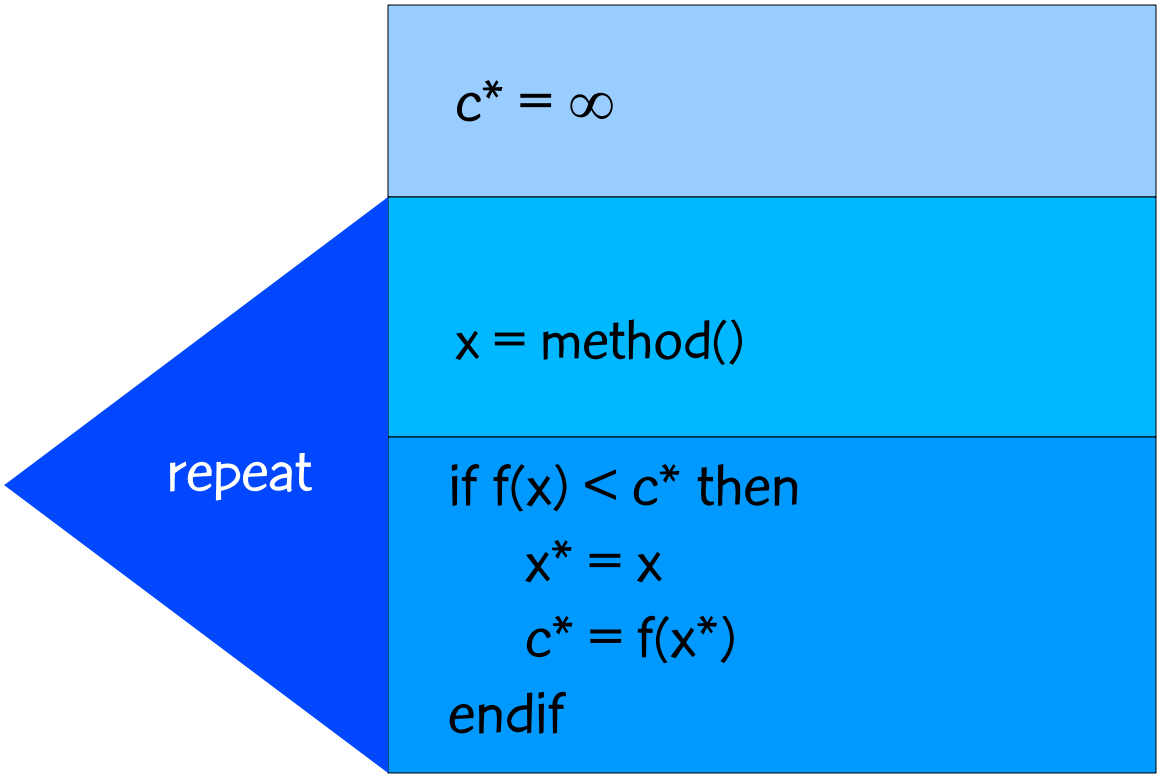
starting solution



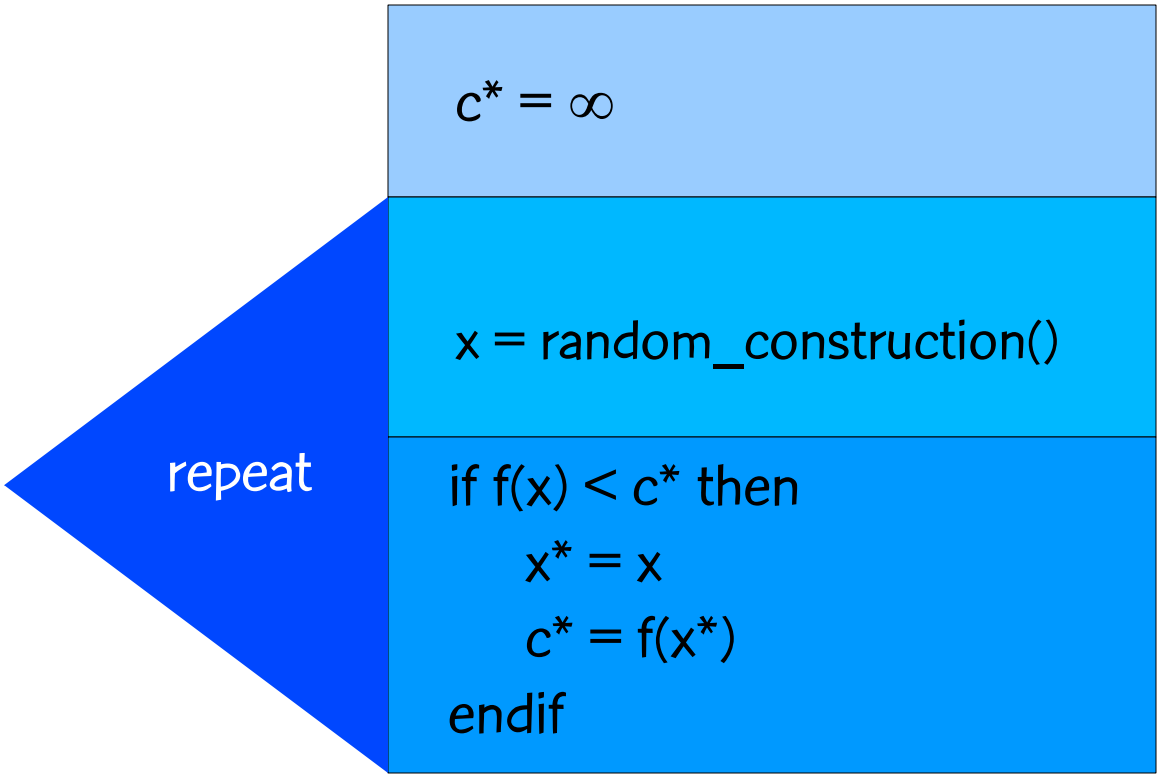
usually pre-determined

usually easier to control

Multi-start method



Random multi-start



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

N:	10	15	20	25	30
K:					
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

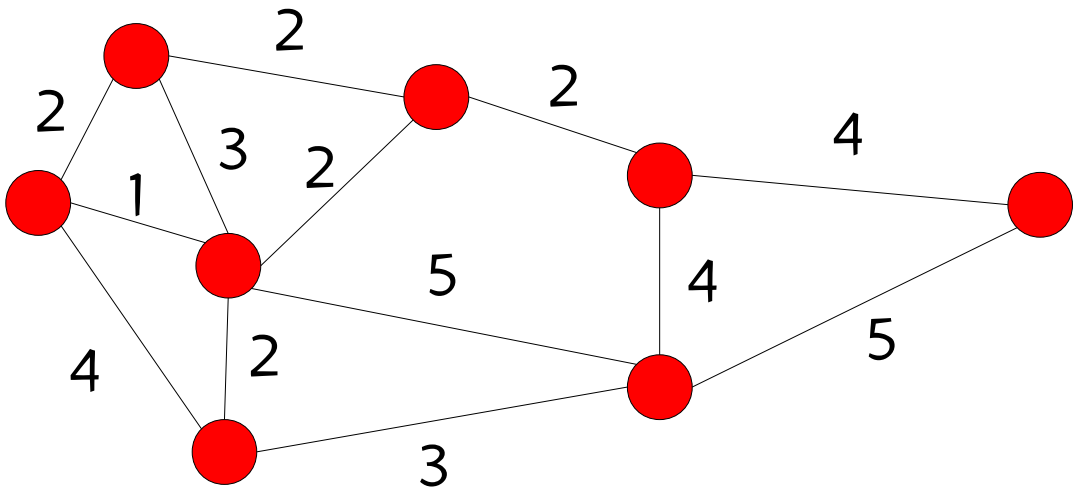
Constructs a solution, one element at a time:

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.

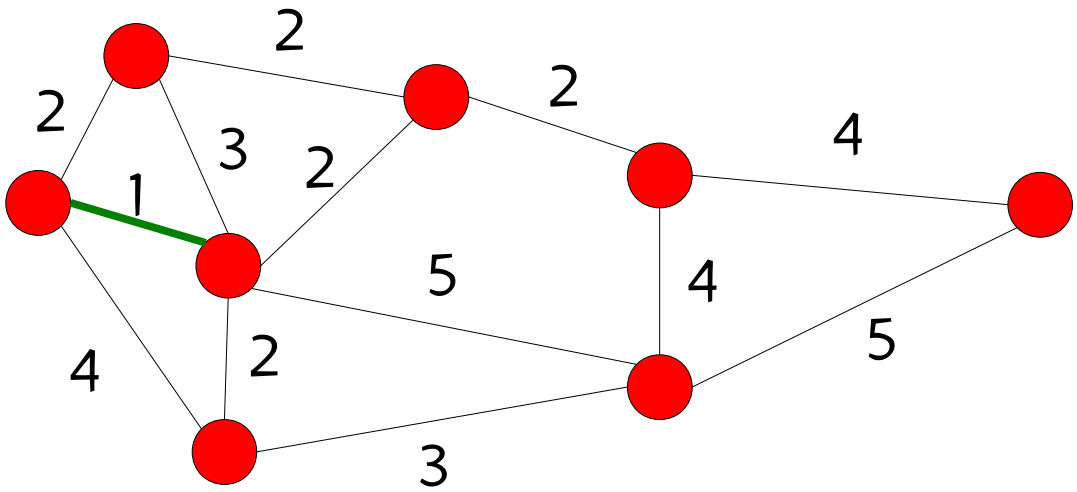
The greedy algorithm

An example: minimum weight spanning tree



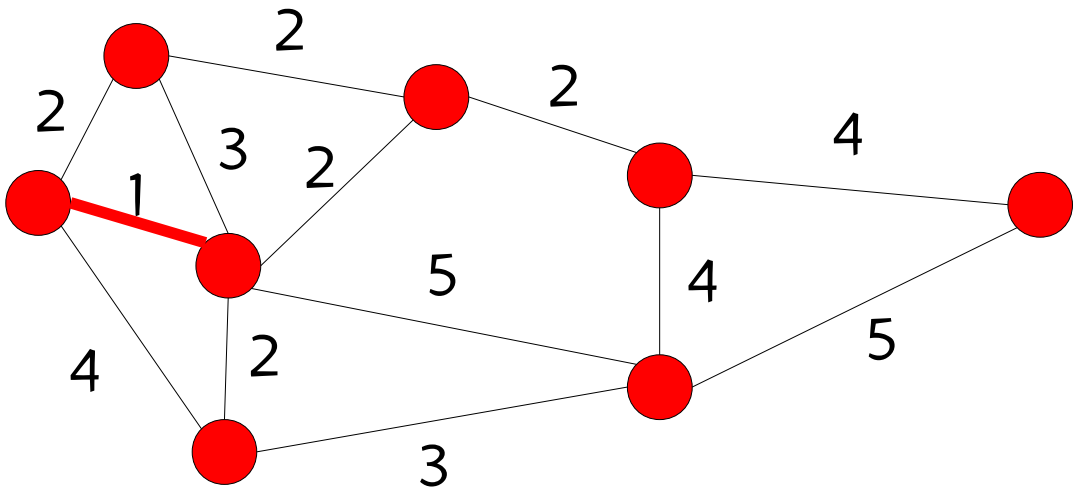
The greedy algorithm

An example: minimum weight spanning tree



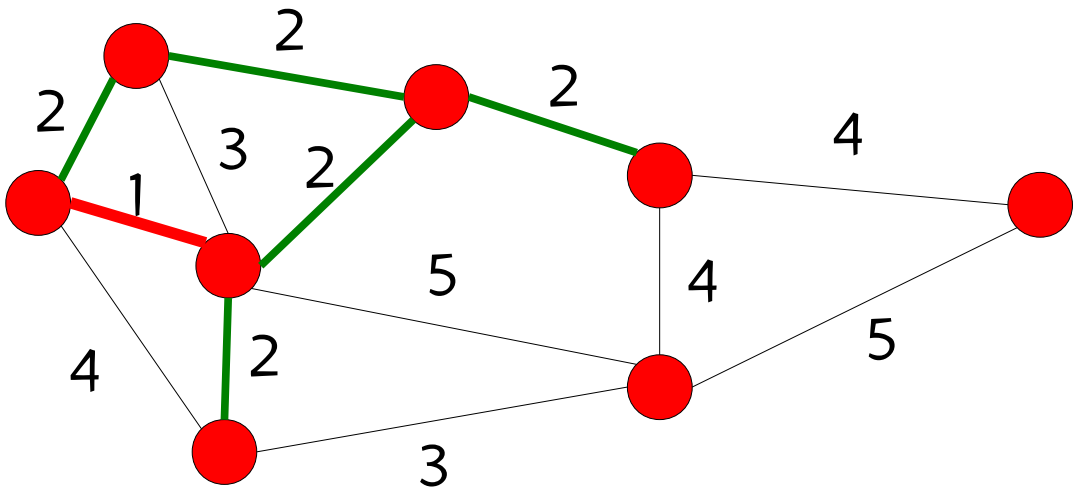
The greedy algorithm

An example: minimum weight spanning tree



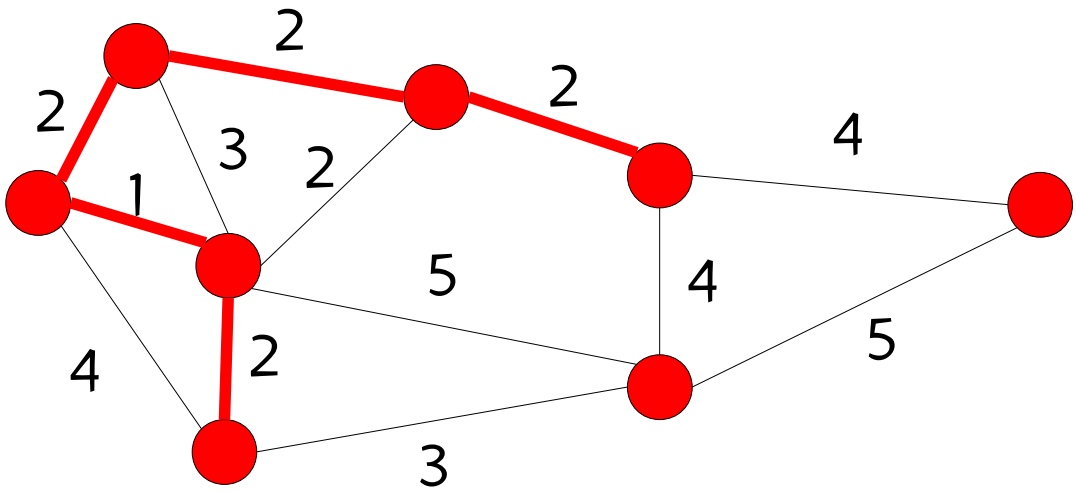
The greedy algorithm

An example: minimum weight spanning tree



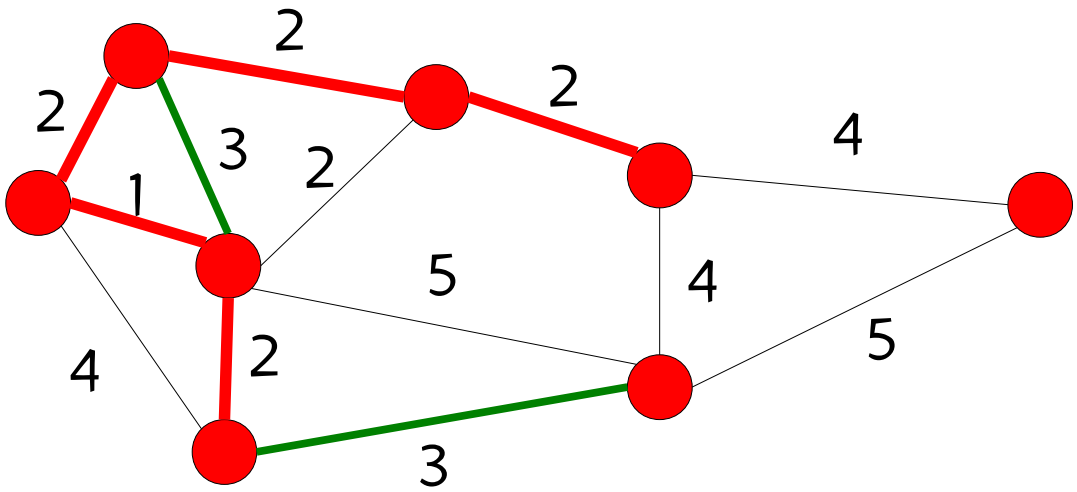
The greedy algorithm

An example: minimum weight spanning tree



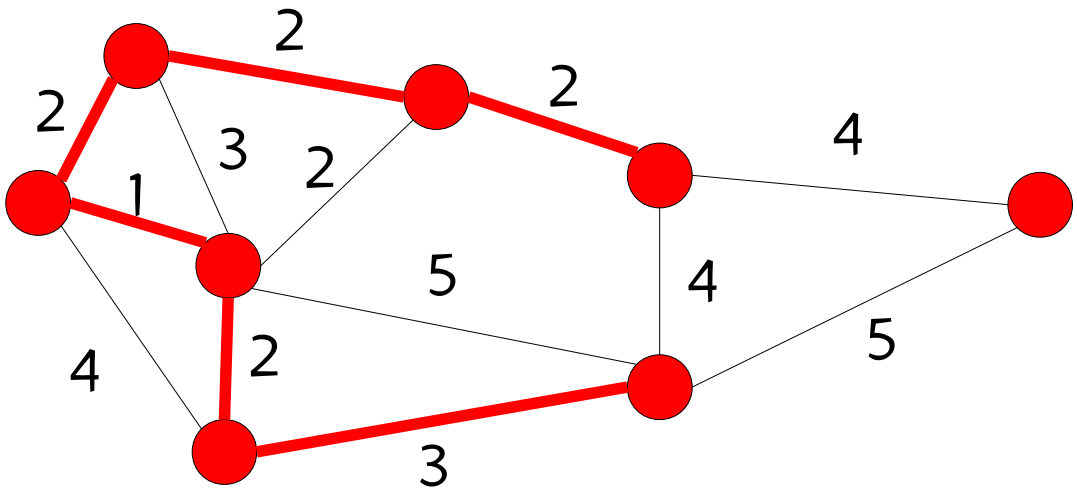
The greedy algorithm

An example: minimum weight spanning tree



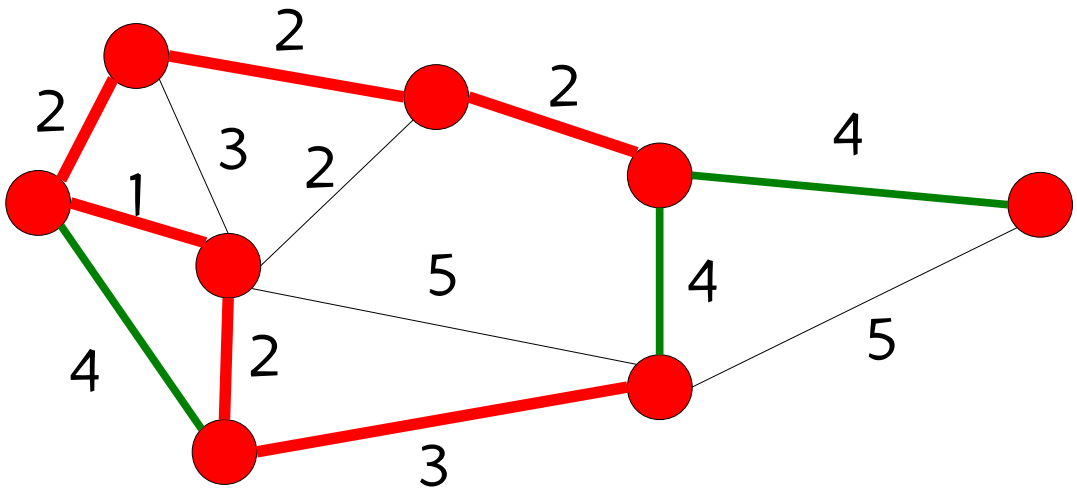
The greedy algorithm

An example: minimum weight spanning tree



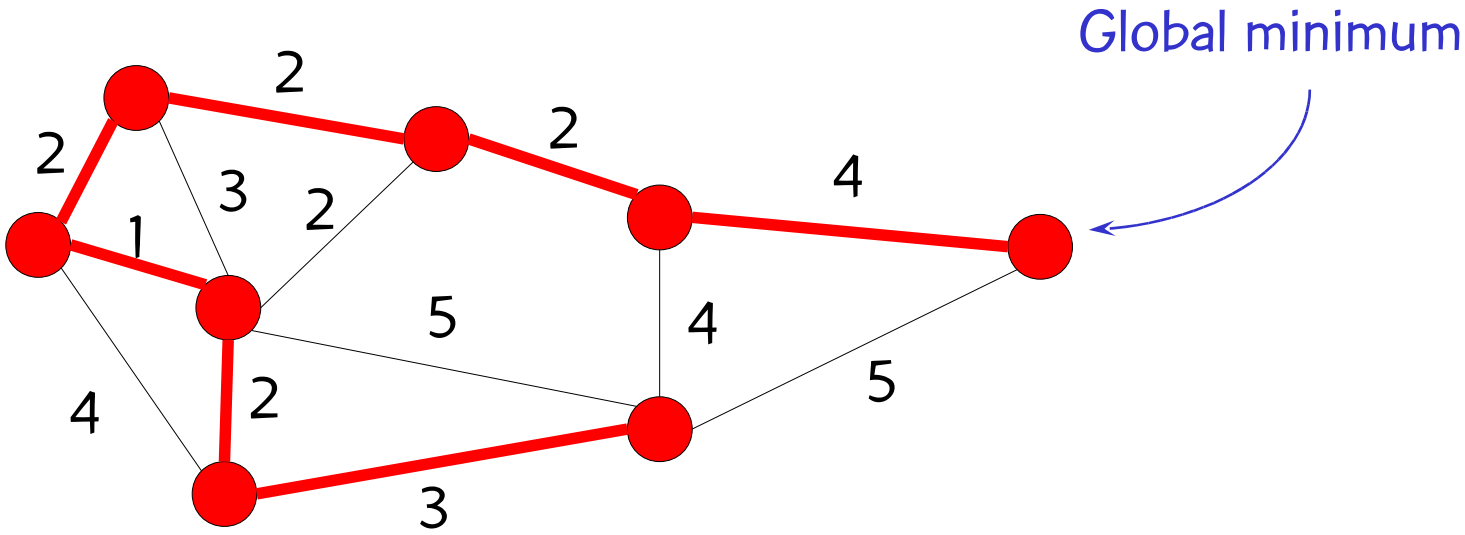
The greedy algorithm

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.

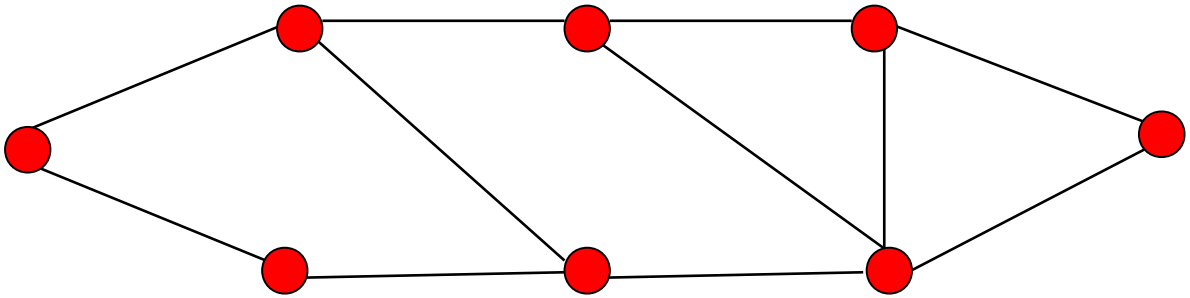
greedy algorithm builds solution, one element (vertex) at a time

candidate set: unselected vertices adjacent to all selected vertices

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

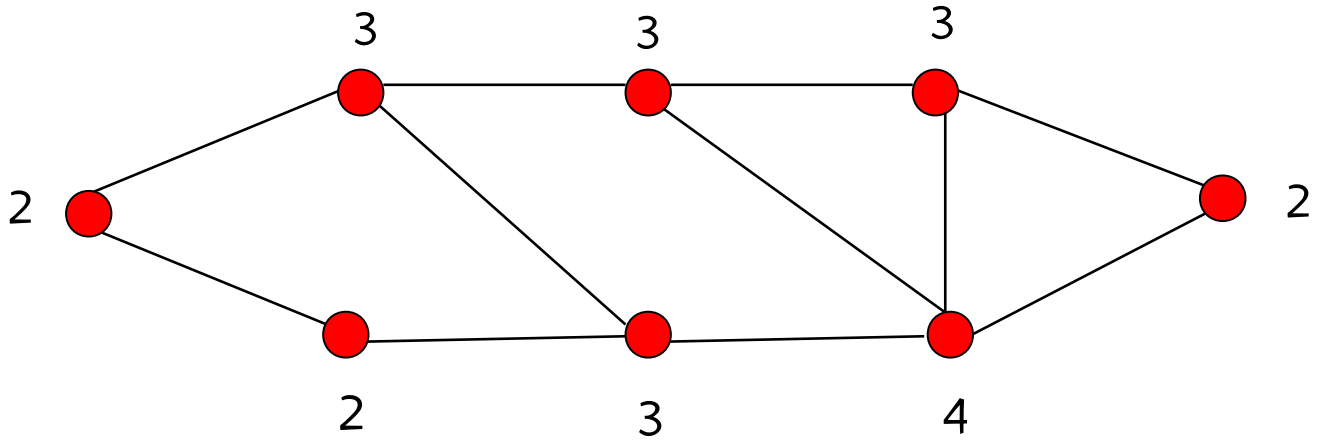
The greedy algorithm

Another example: Maximum clique



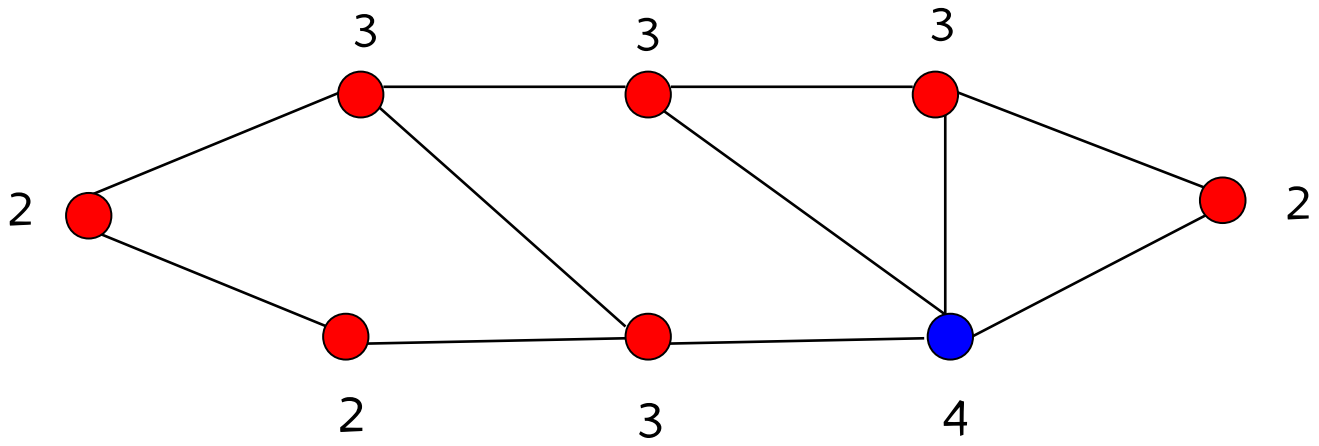
The greedy algorithm

Another example: Maximum clique



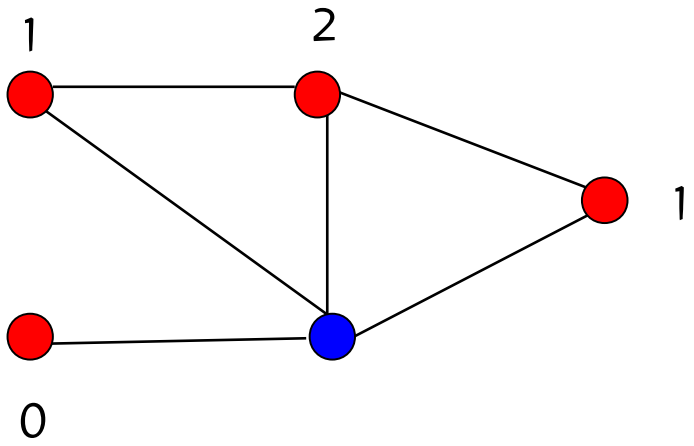
The greedy algorithm

Another example: Maximum clique



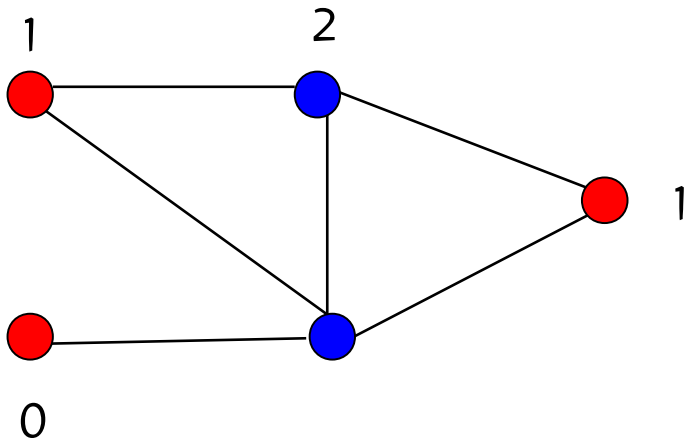
The greedy algorithm

Another example: Maximum clique



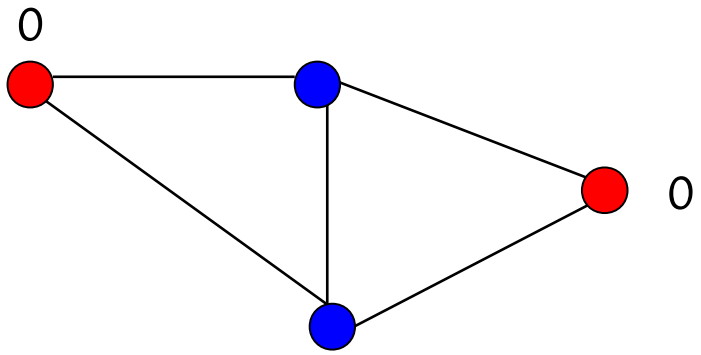
The greedy algorithm

Another example: Maximum clique



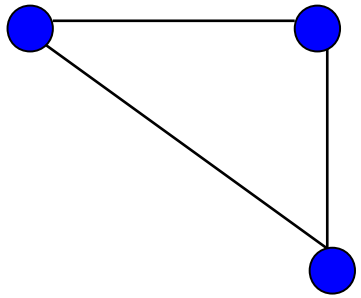
The greedy algorithm

Another example: Maximum clique



The greedy algorithm

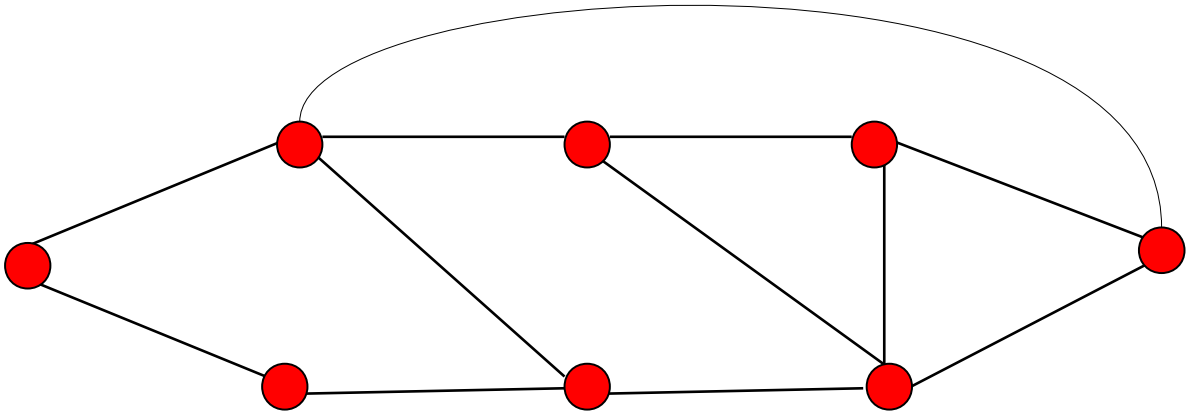
Another example: Maximum clique



global maximum

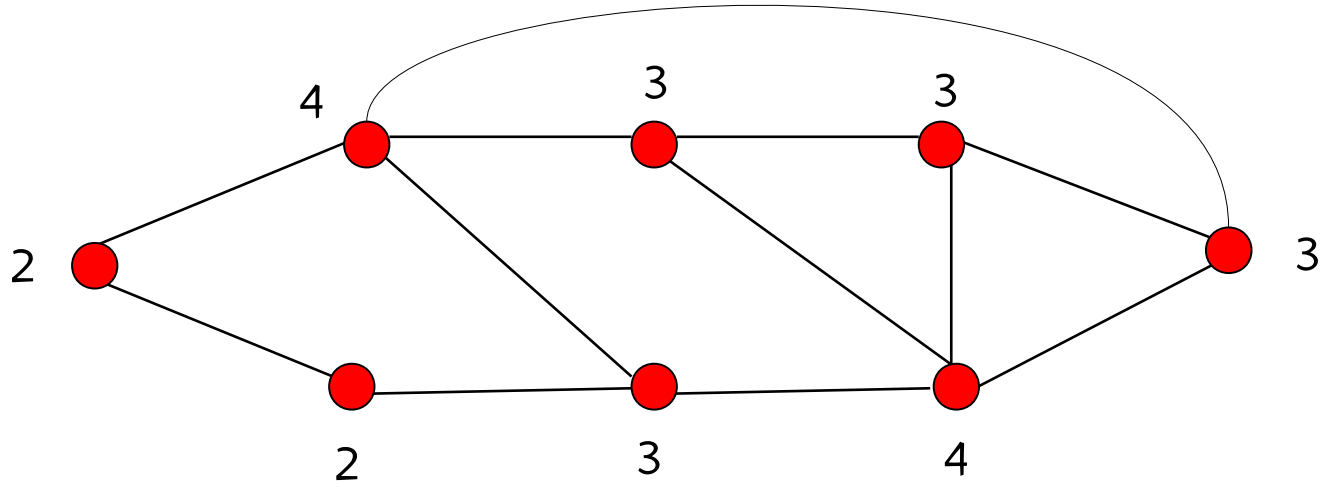
The greedy algorithm

Another example: Maximum clique



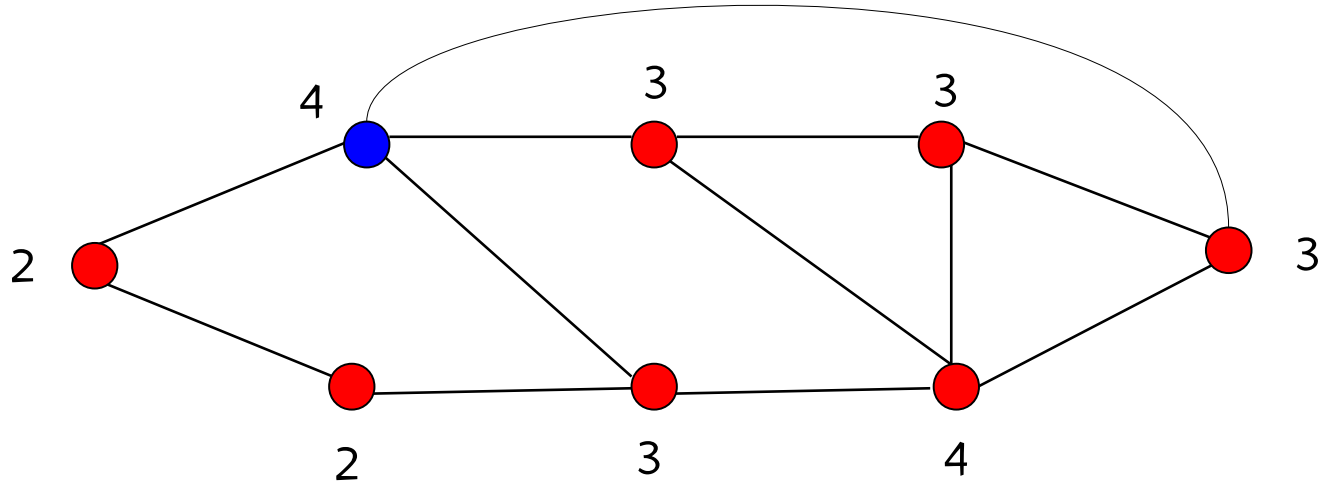
The greedy algorithm

Another example: Maximum clique



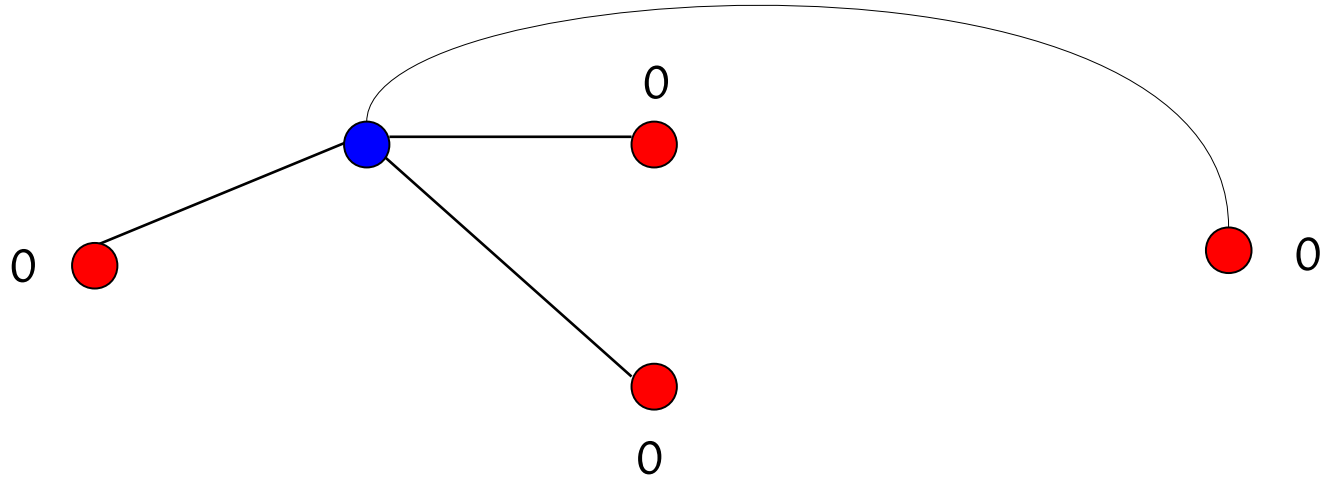
The greedy algorithm

Another example: Maximum clique



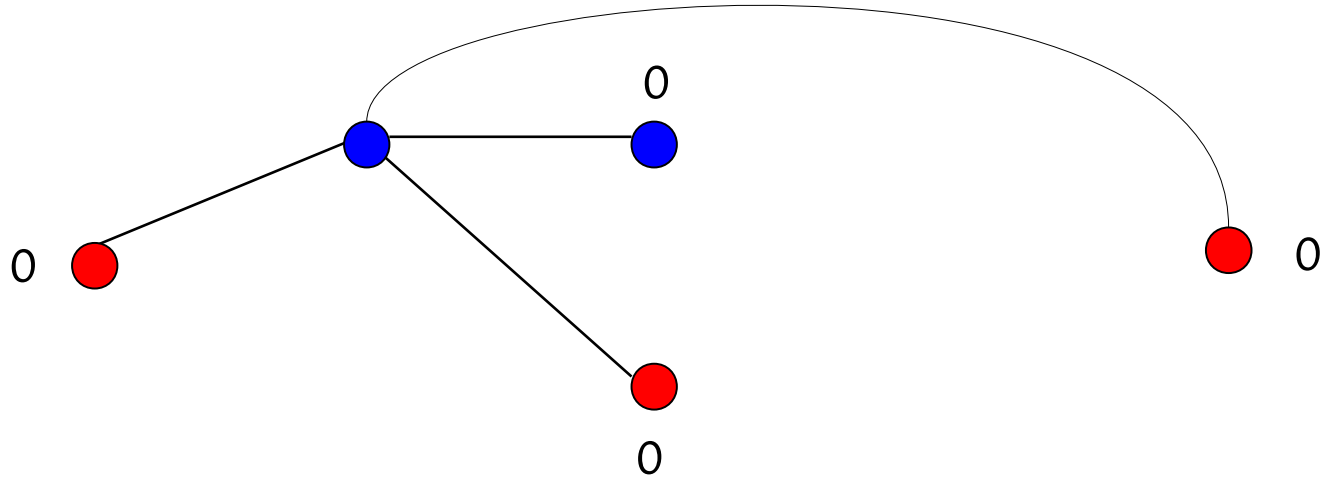
The greedy algorithm

Another example: Maximum clique



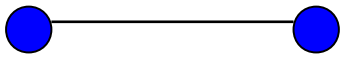
The greedy algorithm

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

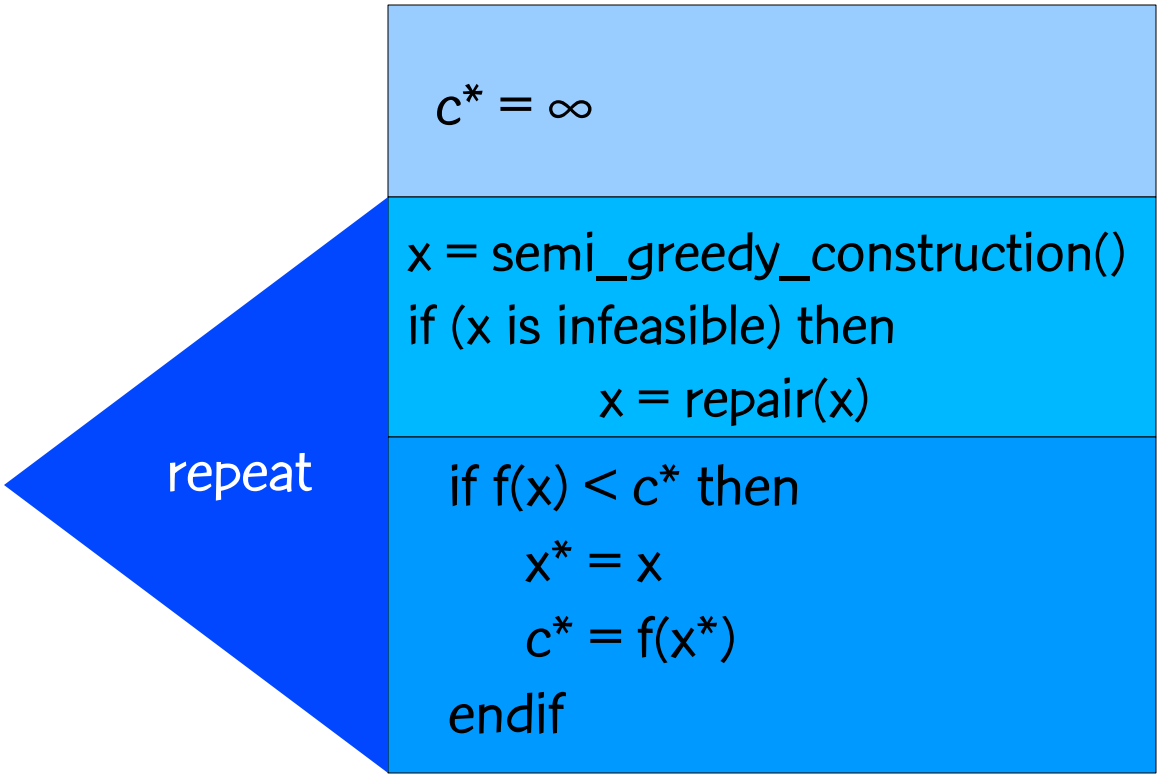
Hart & Shogan (1987) propose two mechanisms for building the RCL:

Cardinality based: place k best candidates in RCL

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

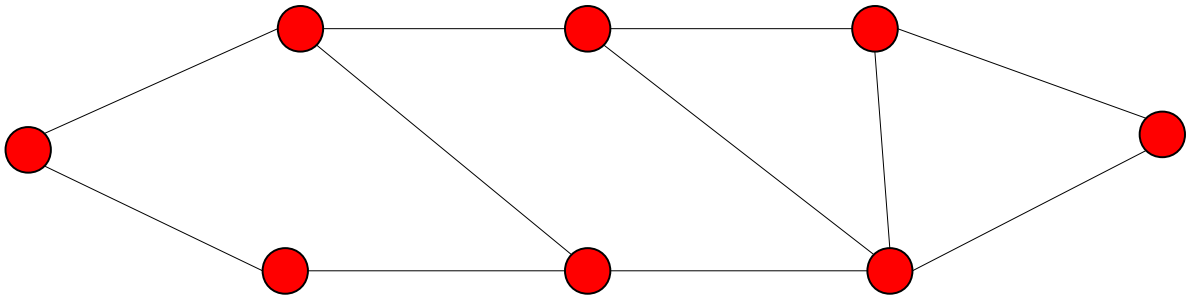
Feo & Resende (1989) proposed semi-greedy construction as a basic component of GRASP.

Hart-Shogan Algorithm



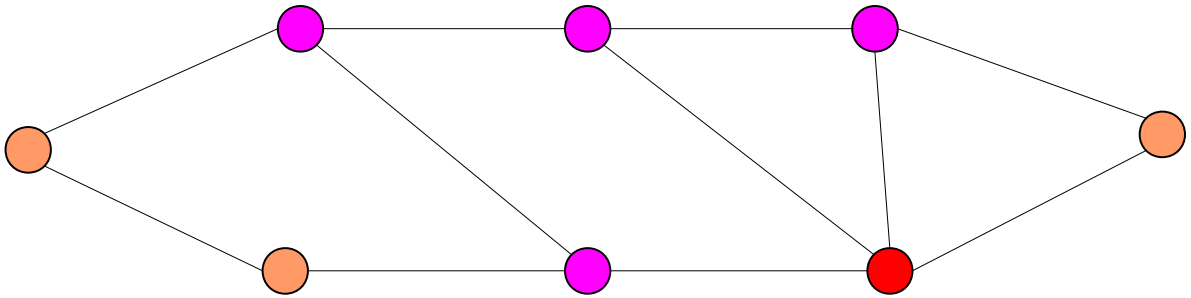
The semi-greedy algorithm

Maximum clique example



The semi-greedy algorithm

Maximum clique example

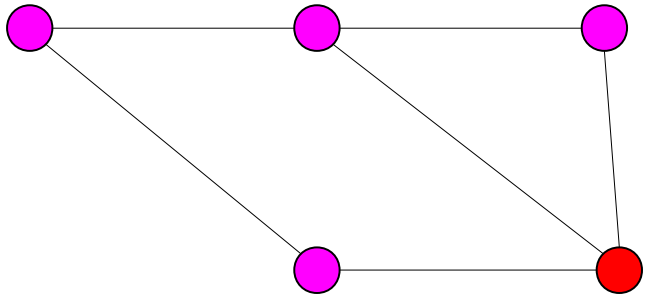


Build clique, one node at a time.

Candidates: nodes adjacent to clique.

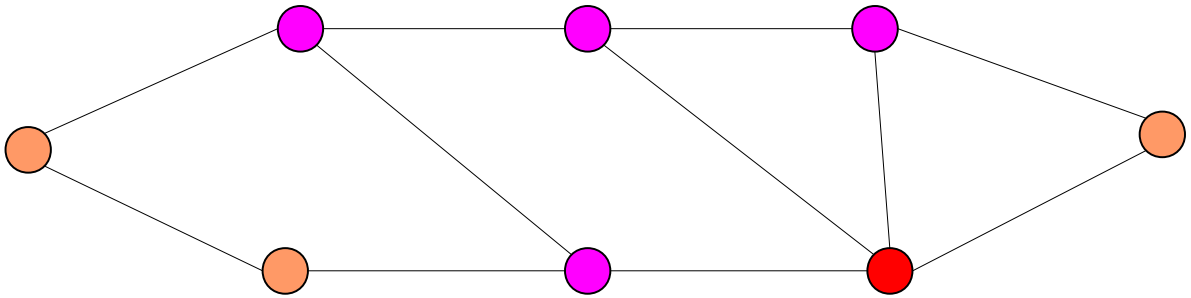
Greedy function: degree with respect to candidate nodes.

RCL =

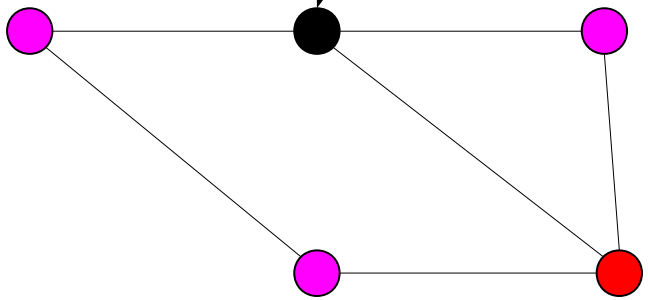


The semi-greedy algorithm

Maximum clique example



Choose at random



RCL =

Build clique, one node at a time.

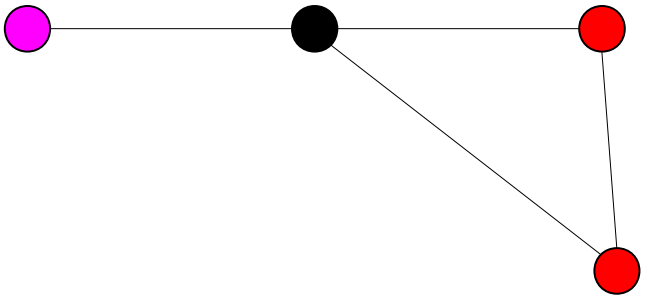
Candidates: nodes adjacent to clique.

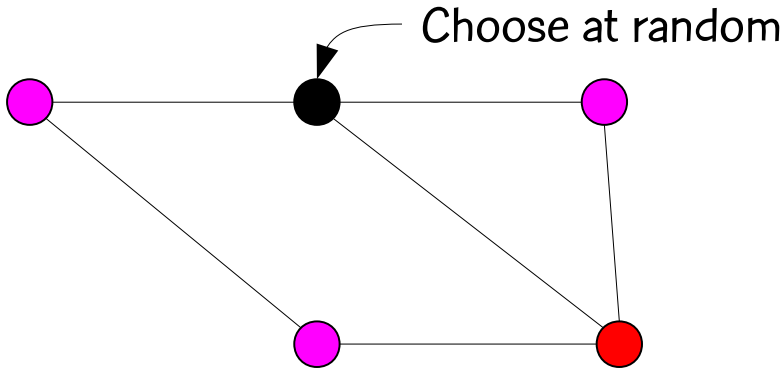
Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 1

The semi-greedy algorithm

Maximum clique example





RCL =

Build clique, one node at a time.

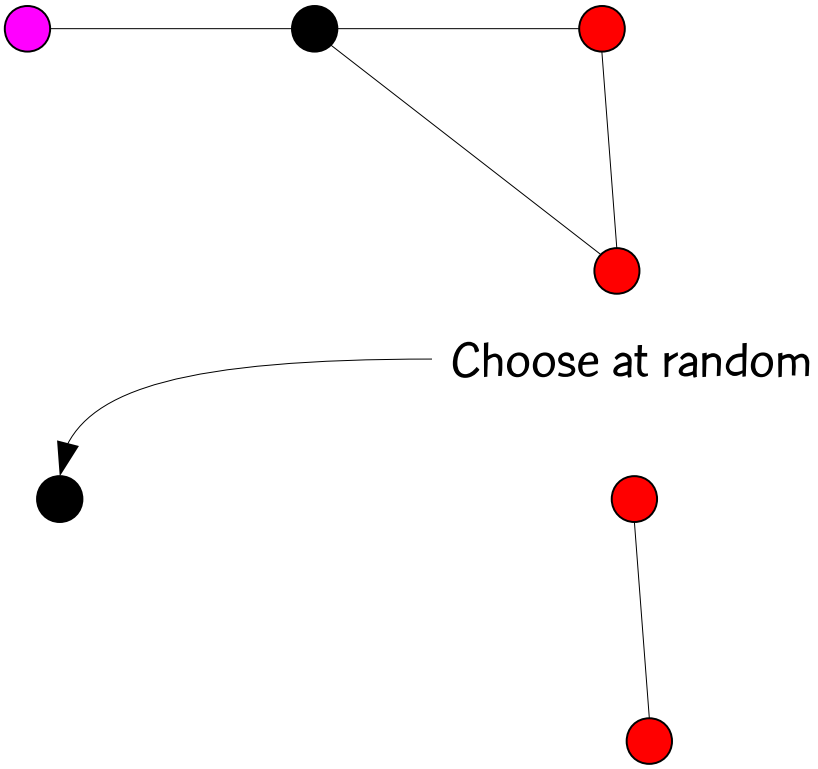
Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 1

The semi-greedy algorithm

Maximum clique example



Build clique, one node at a time.

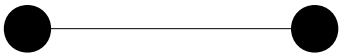
Candidates: nodes adjacent to clique.

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.

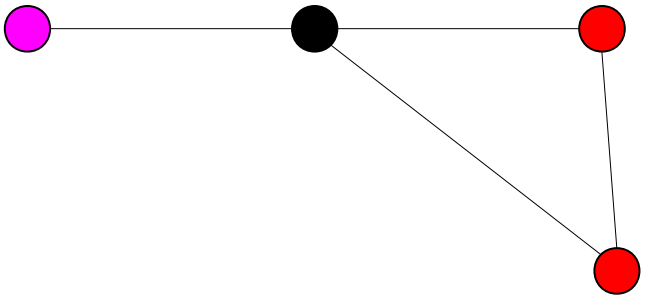
Choose at random

RCL =

Semi-greedy iteration 1

The semi-greedy algorithm

Maximum clique example



Instead, choose at random

RCL =

Build clique, one node at a time.

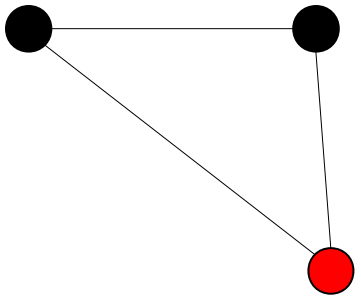
Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 2

The semi-greedy algorithm

Maximum clique example



Instead, choose at random

RCL =

Build clique, one node at a time.

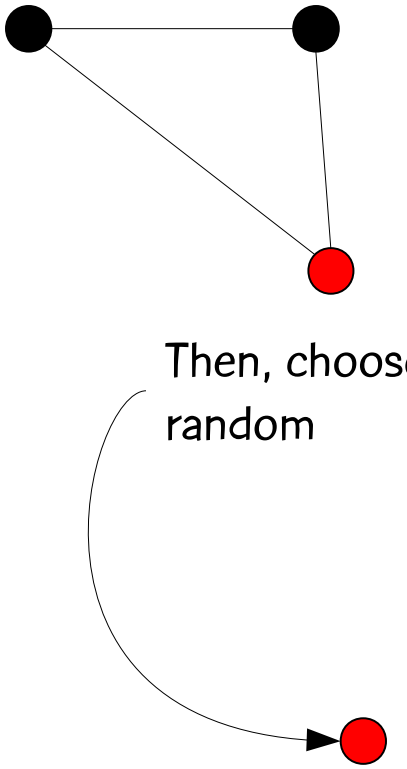
Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 2

The semi-greedy algorithm

Maximum clique example



Build clique, one node at a time.

Candidates: nodes adjacent to clique.

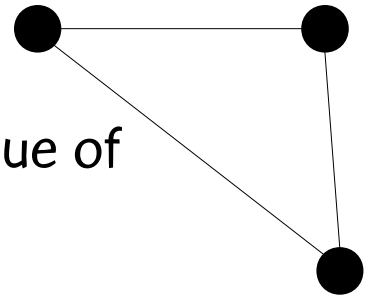
Greedy function: degree with respect to candidate nodes.

RCL =

Semi-greedy iteration 2

The semi-greedy algorithm

Maximum clique example



Optimal clique of size 3

Then, choose at random

RCL =

Build clique, one node at a time.

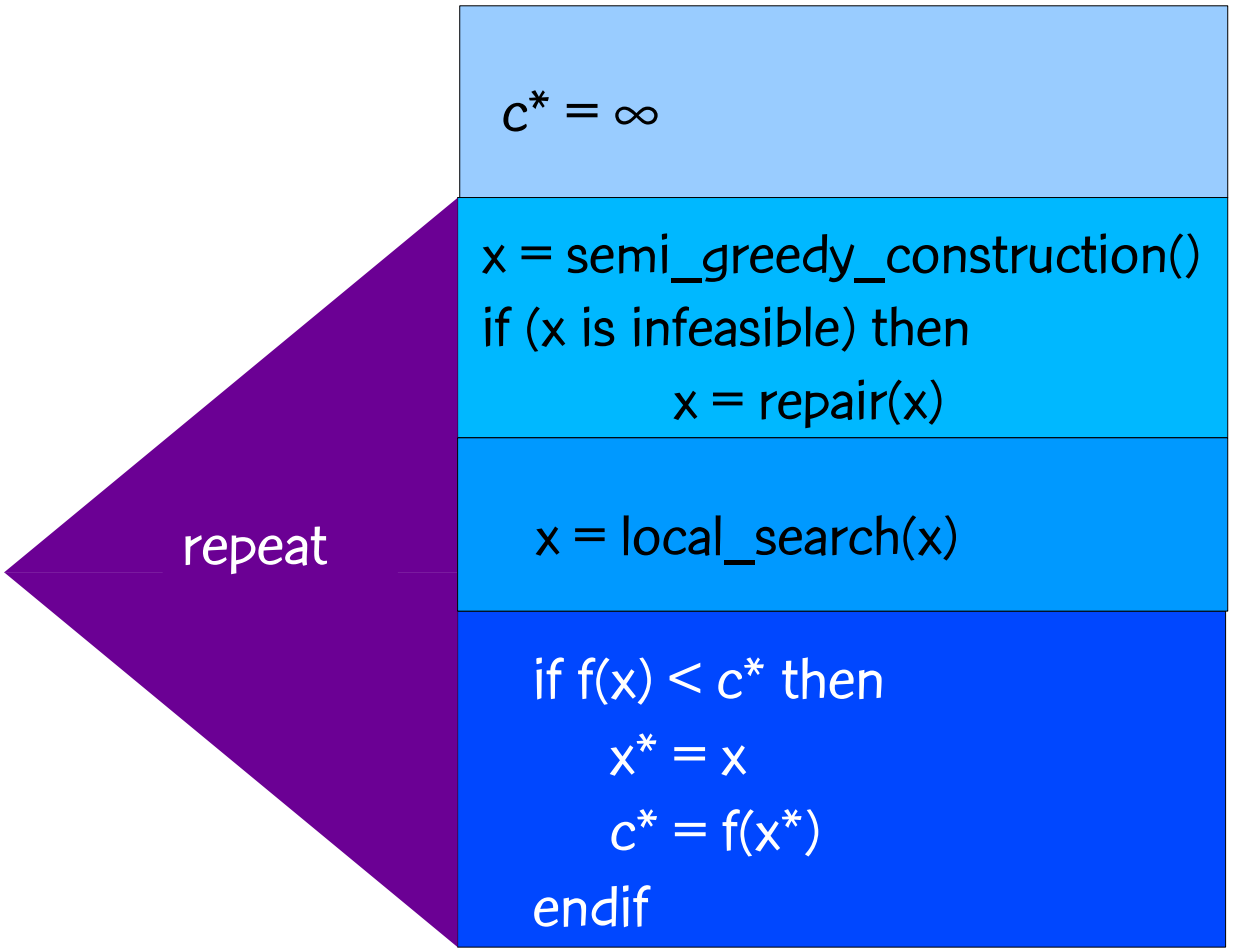
Candidates: nodes adjacent to clique.

Greedy function: degree with respect to candidate nodes.

Semi-greedy iteration 2

GRASP

GRASP: Basic algorithm



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

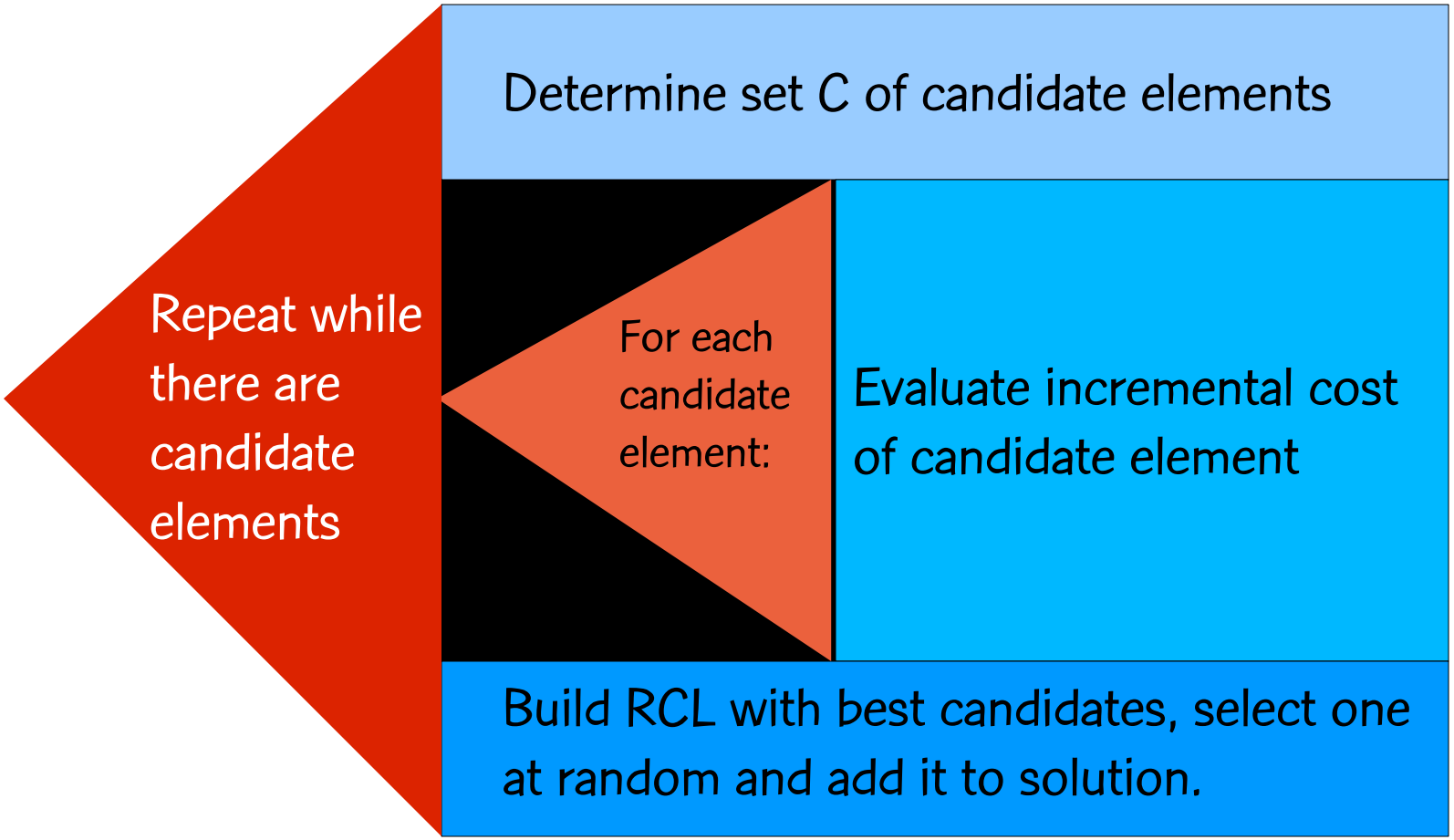
Solutions generated by the construction procedure are not necessarily optimal:

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 α defines the quality of the elements in RCL.

RCL contains elements with incremental cost

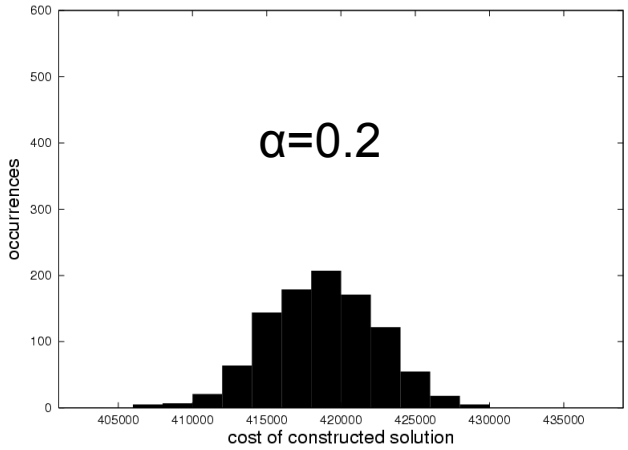
$c^{min} \leq c(e) \leq c^{min} + \alpha\,(c^{max} - c^{min})$

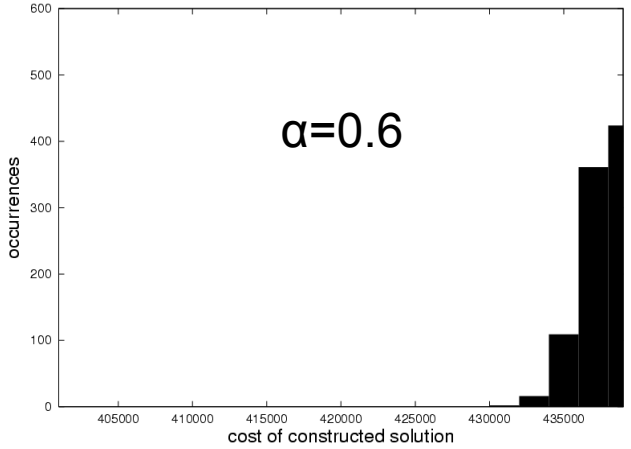
$\alpha = 0$: pure greedy construction

$\alpha = 1$: pure randomized construction

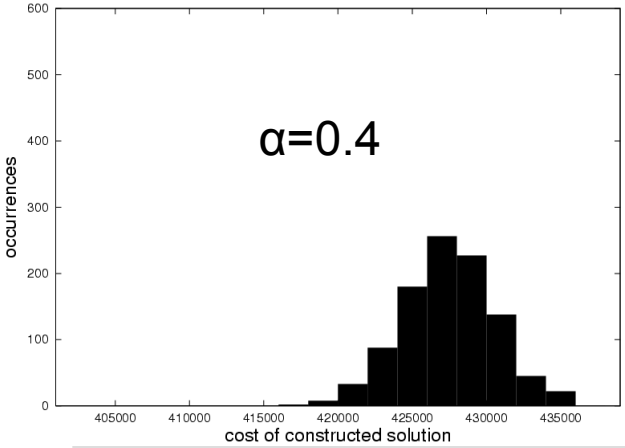
Select at random from RCL using uniform probability distribution

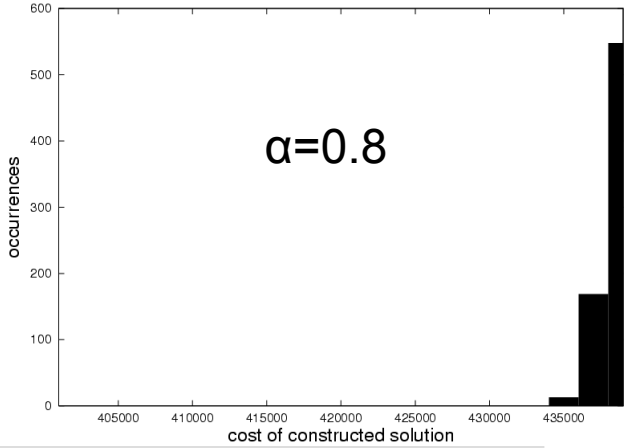
Illustrative results: RCL parameter





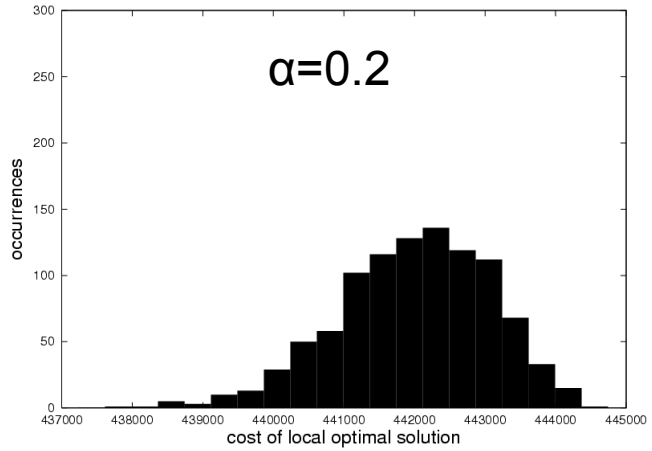
Construction phase only

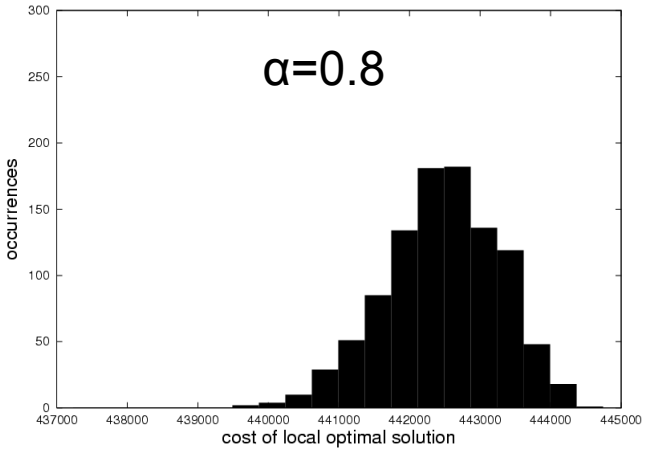




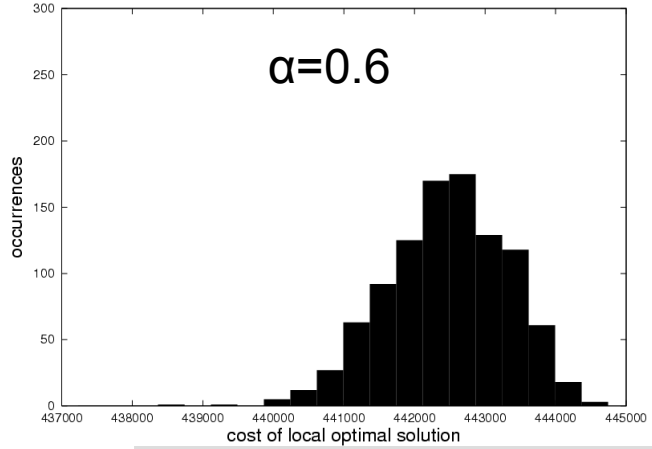
weighted MAX-SAT instance, 1000 GRASP iterations

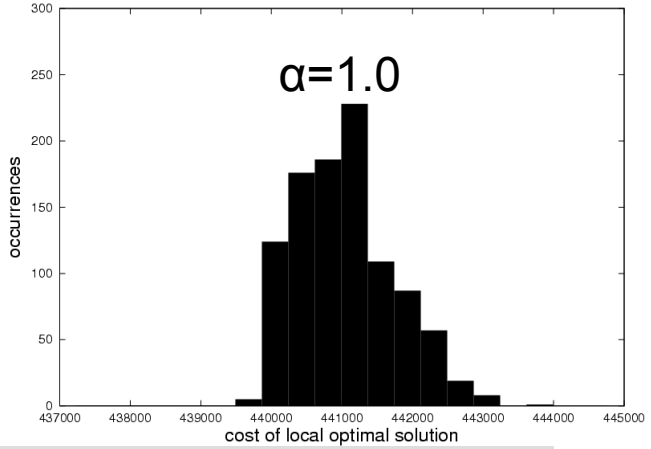
Illustrative results: RCL parameter





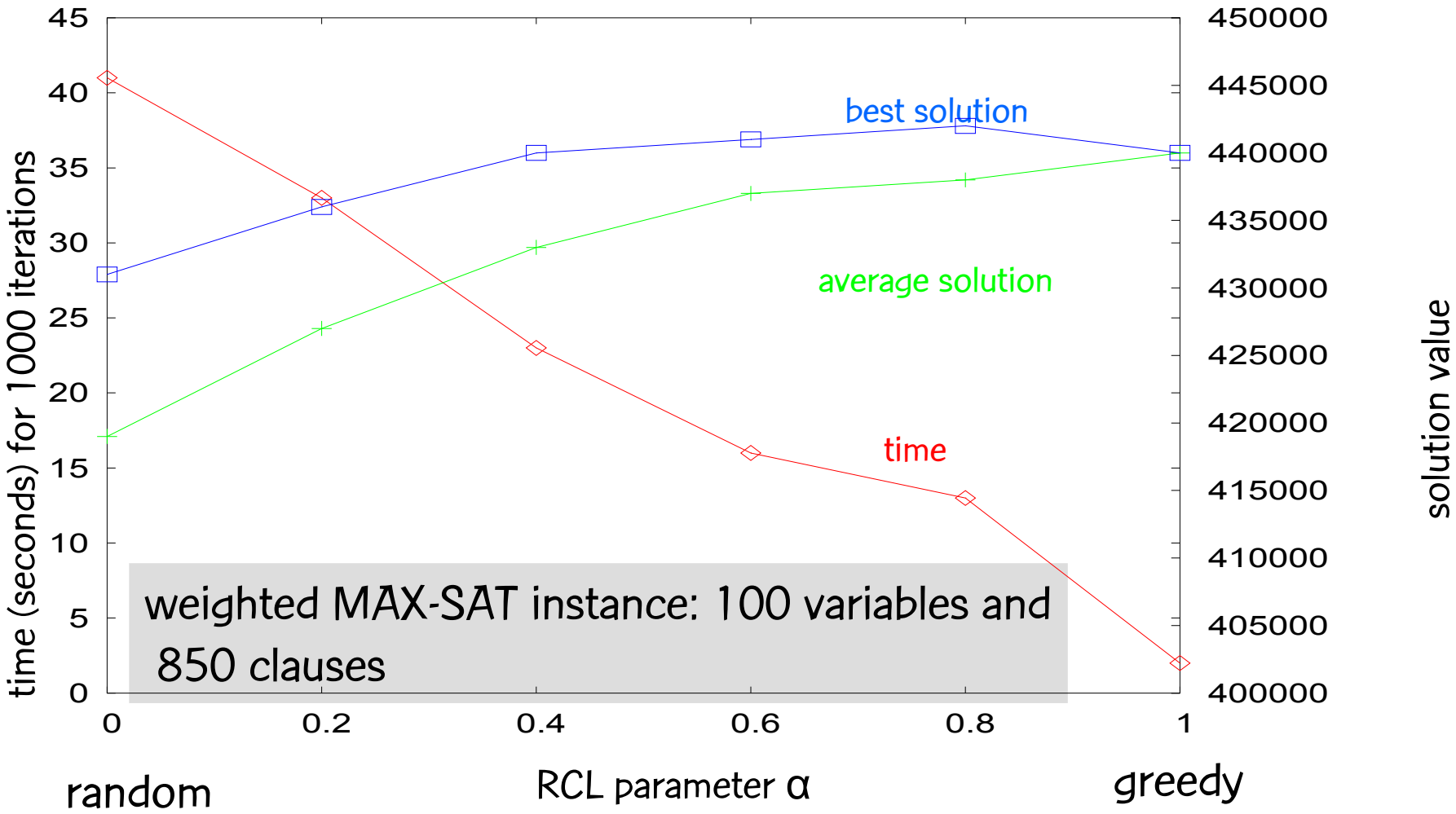
Construction + local search





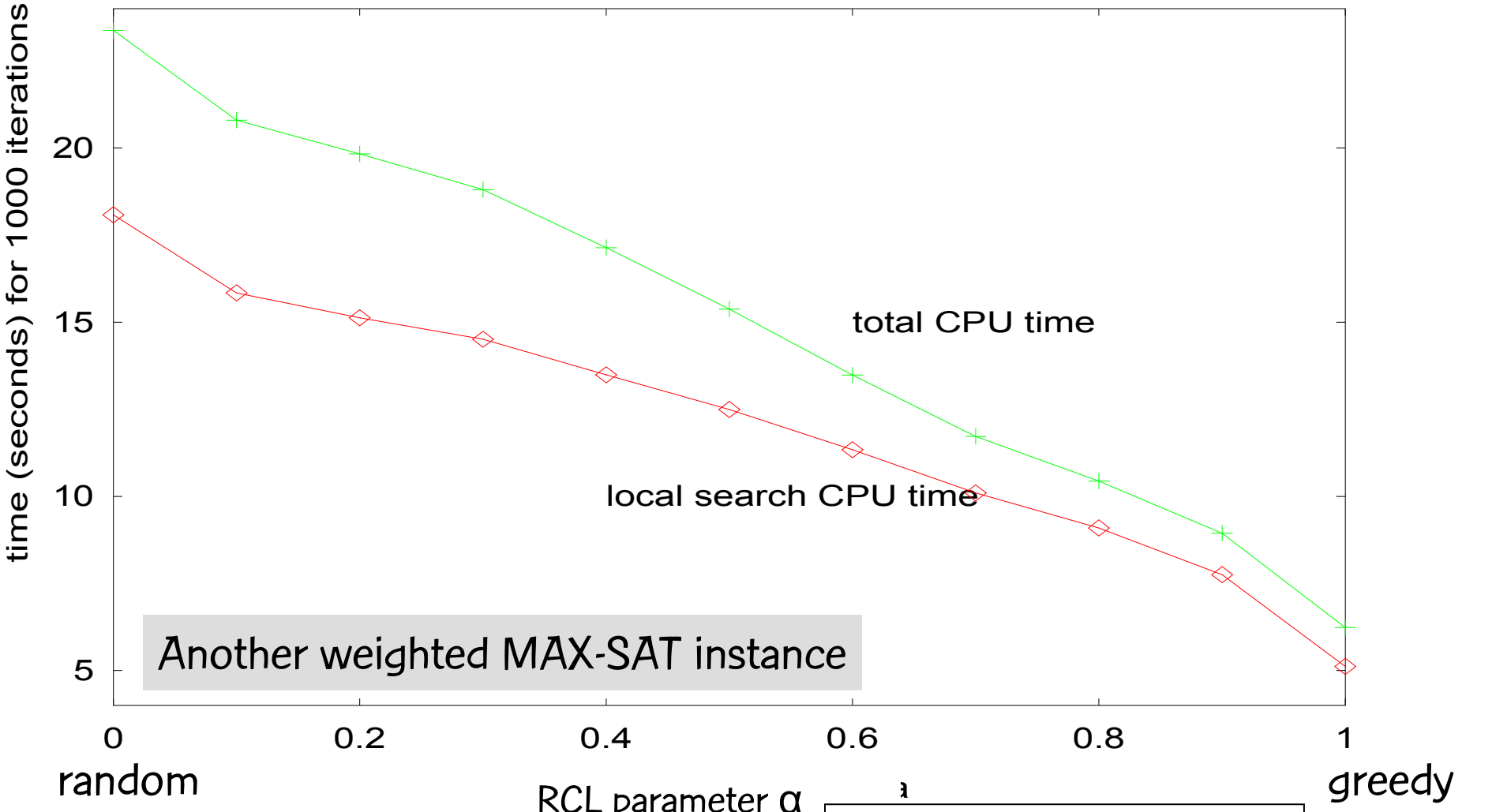
weighted MAX-SAT instance, 1000 GRASP iterations

Illustrative results: RCL parameter



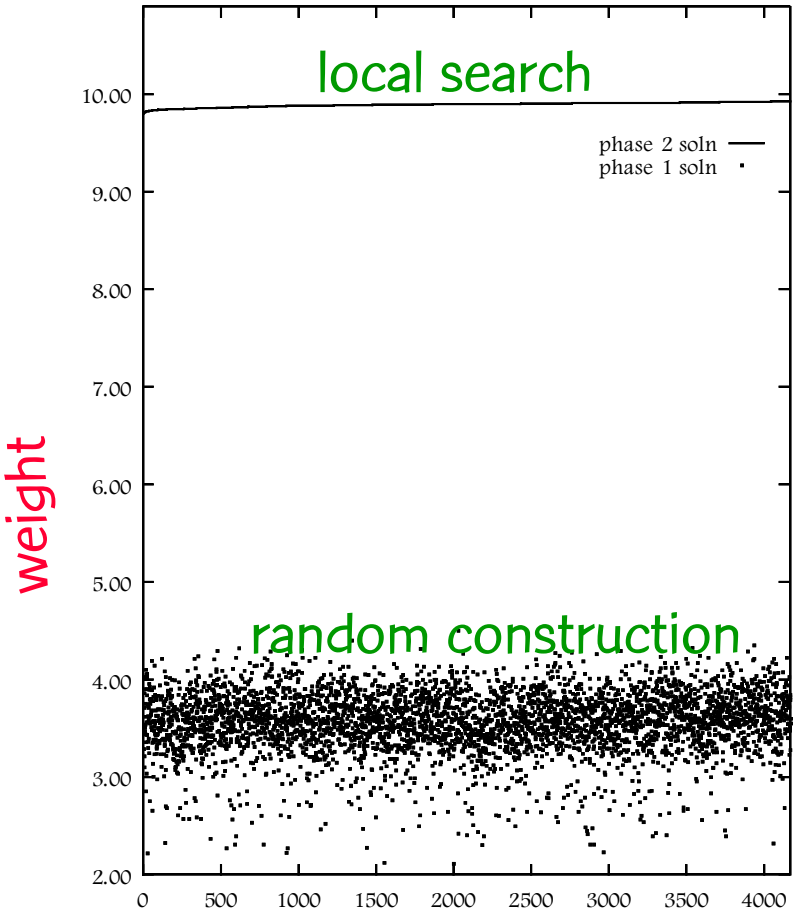
SGI Challenge 196 MHz

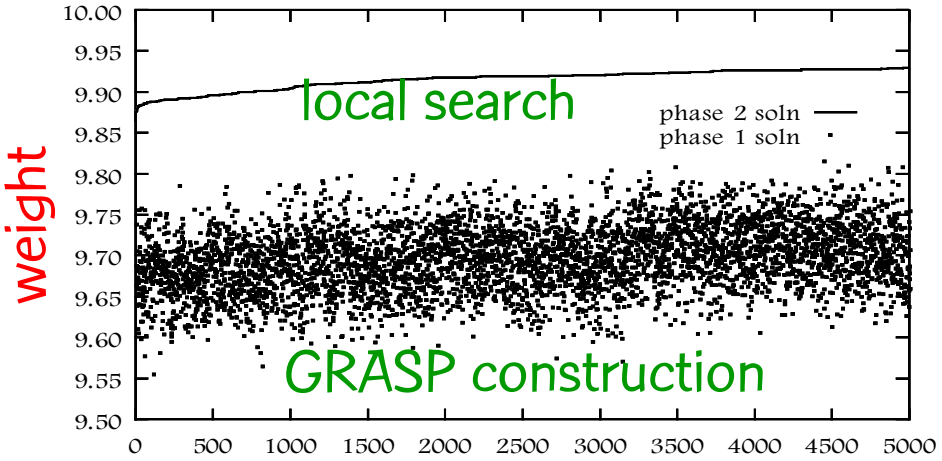
Illustrative results: RCL parameter



SGI Challenge 196 MHz

GRASP: Basic algorithm





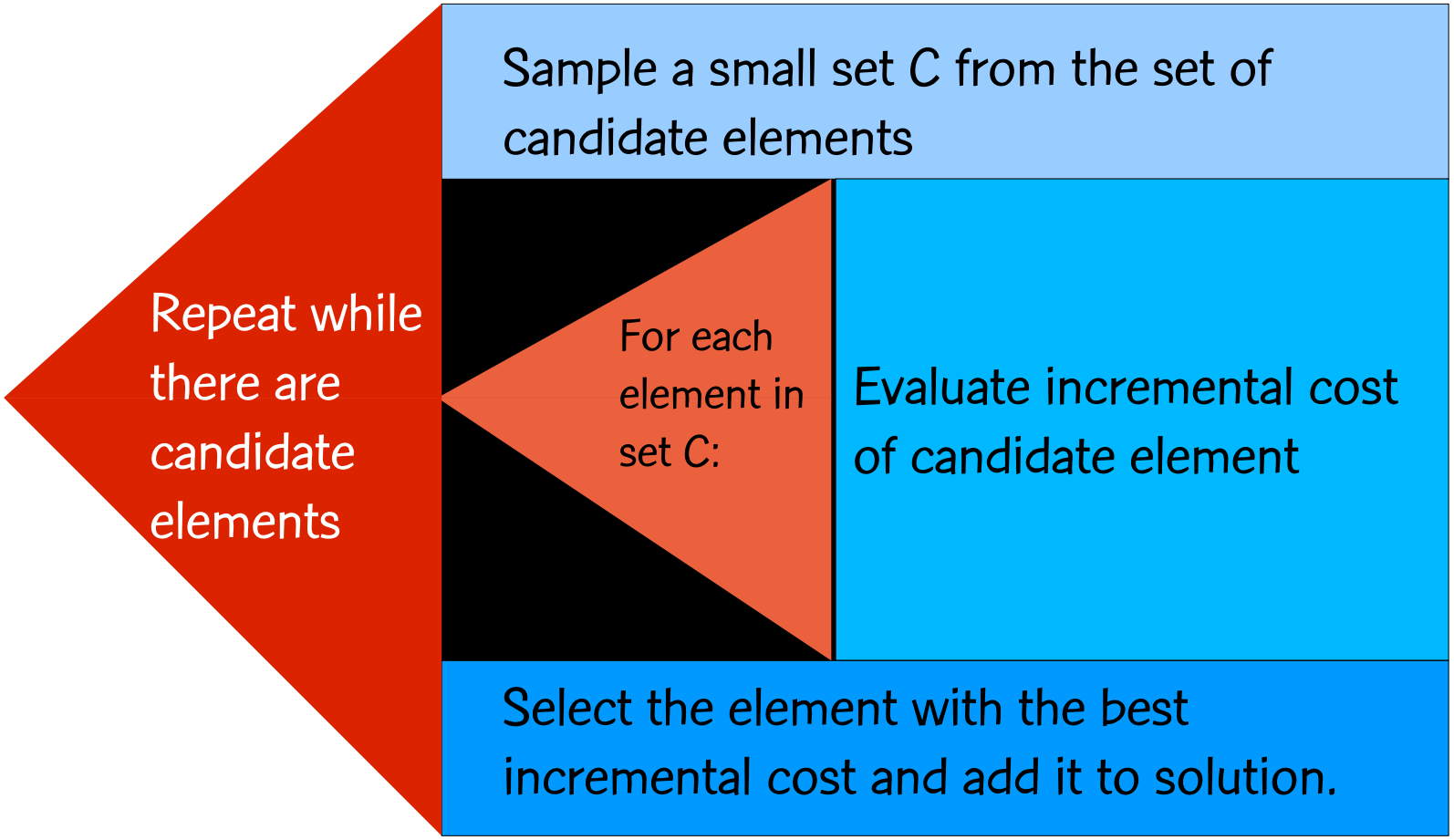
Effectiveness of greedy randomized vs purely randomized construction:

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

Hybrid construction schemes

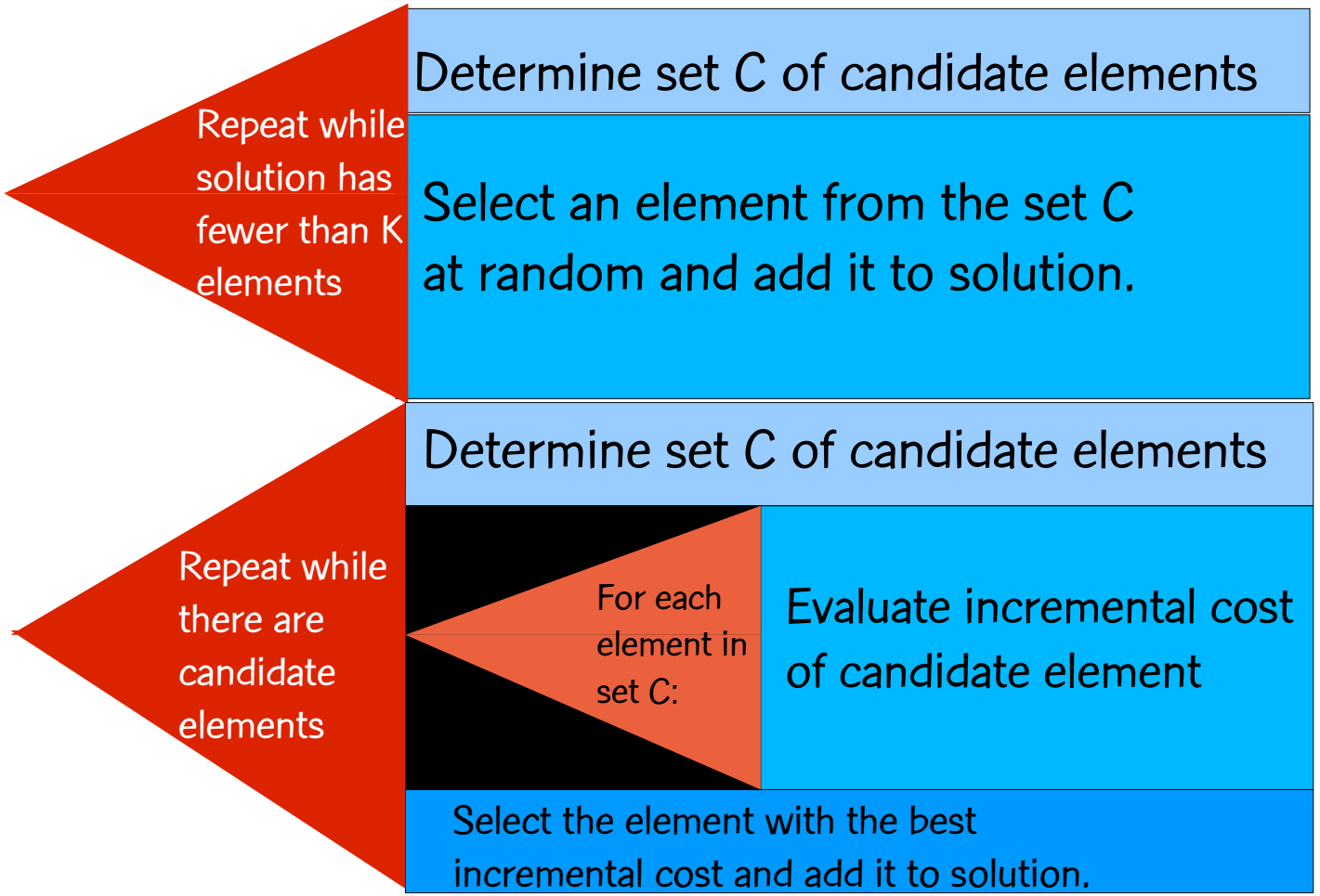
Construction phase: sampled greedy

[Resende & Werneck, 2004]

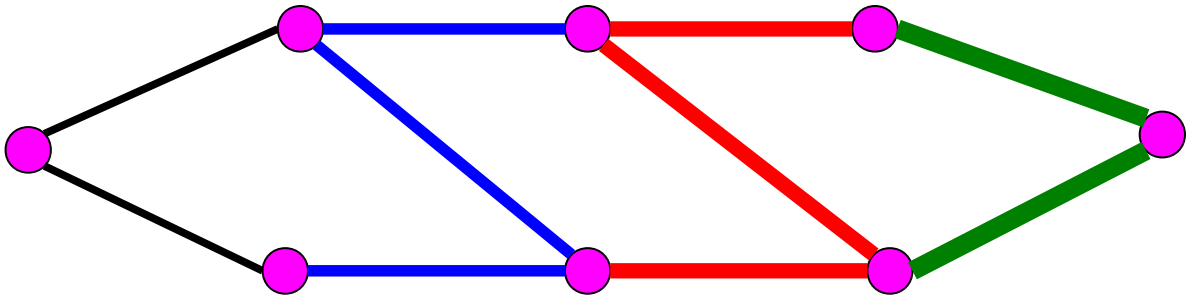


Construction phase: random+greedy

[Resende & Werneck, 2004]



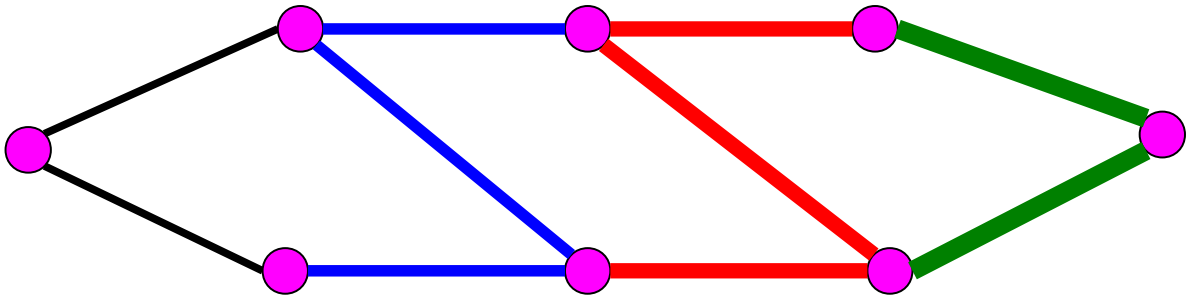
Construction with cost perturbation



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

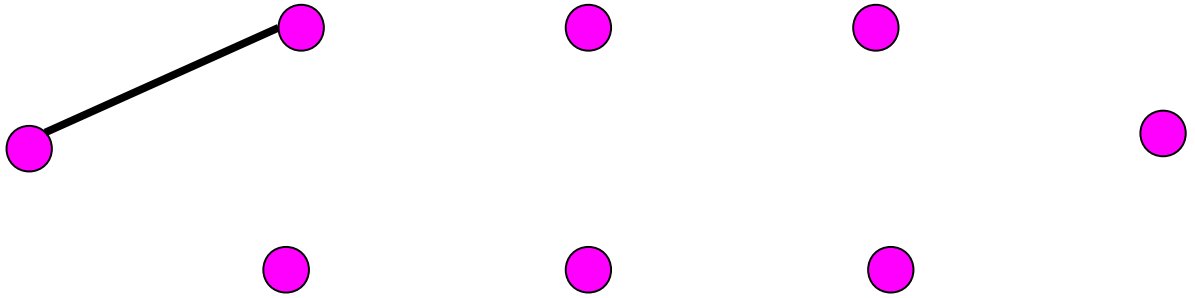
Perturb with costs increasing from top to bottom.

Construction with cost perturbation

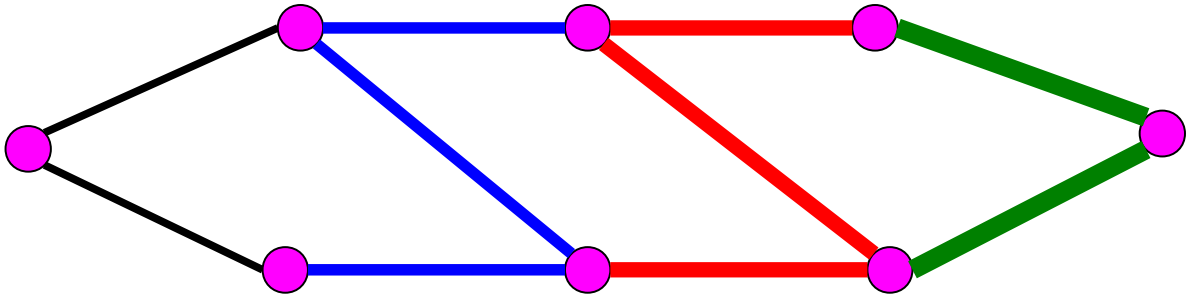


W( ) < W( ) < W( ) < W( )

Perturb with costs increasing from top to bottom.

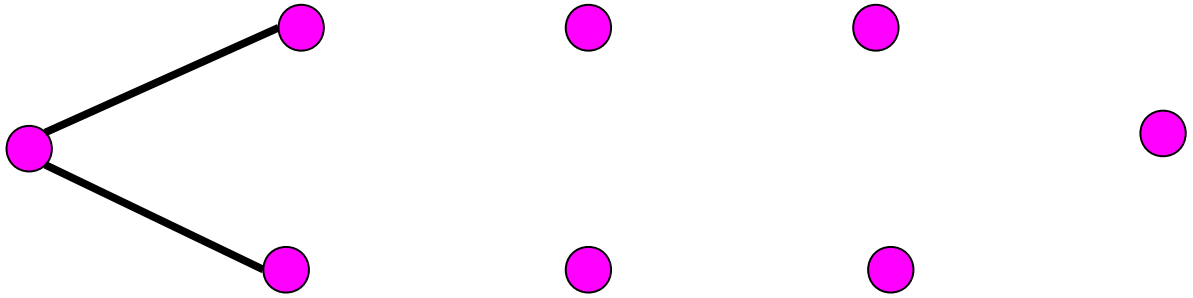


Construction with cost perturbation

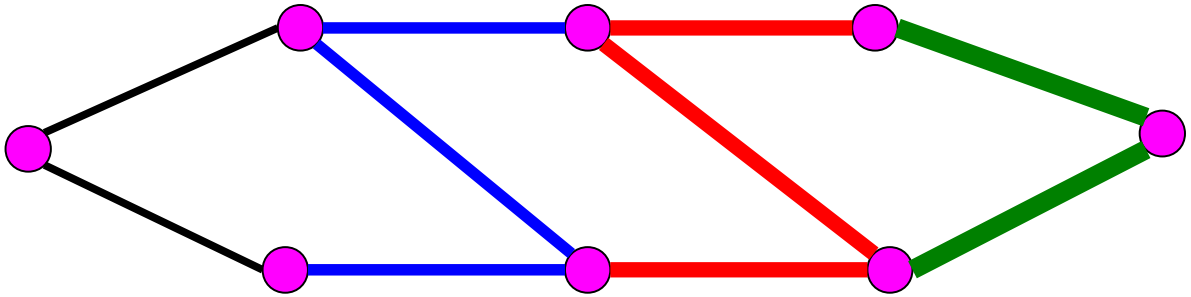


Perturb with costs increasing from top to bottom.

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

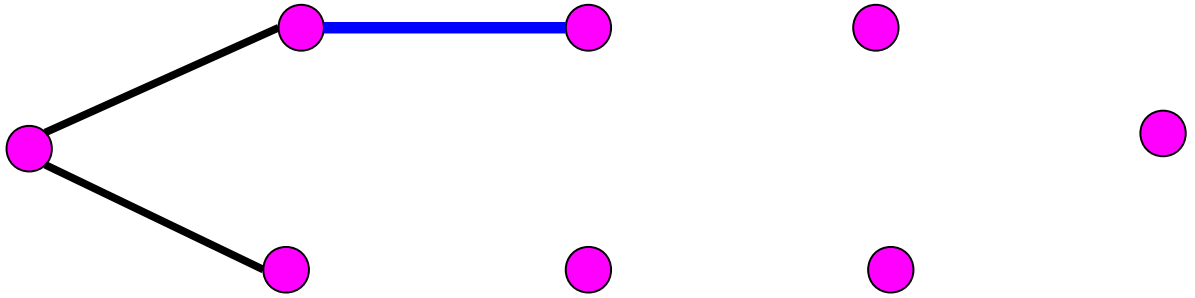


Construction with cost perturbation

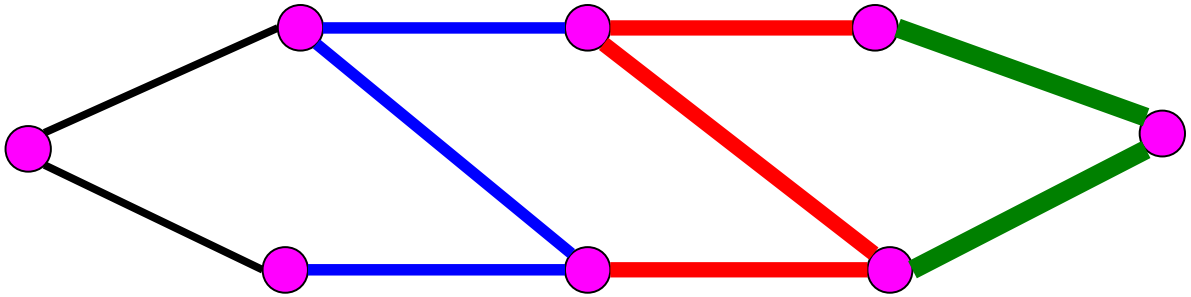


Perturb with costs increasing from top to bottom.

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

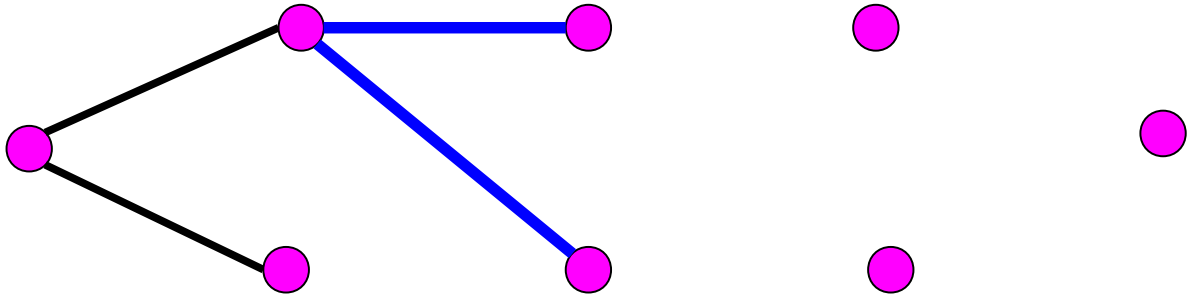


Construction with cost perturbation

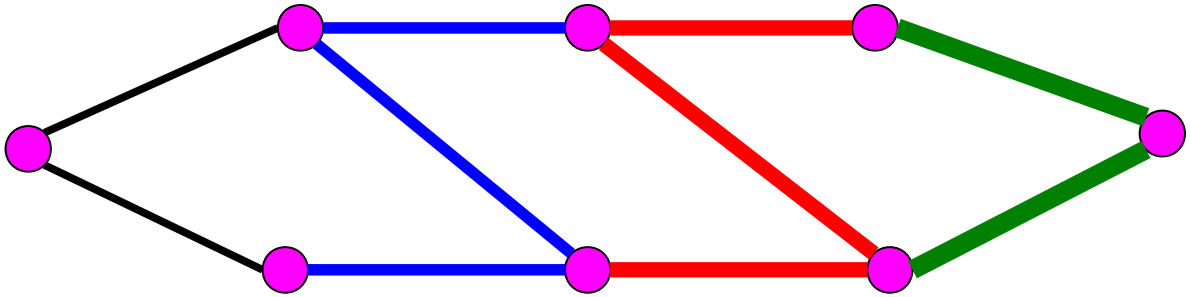


Perturb with costs increasing from top to bottom.

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

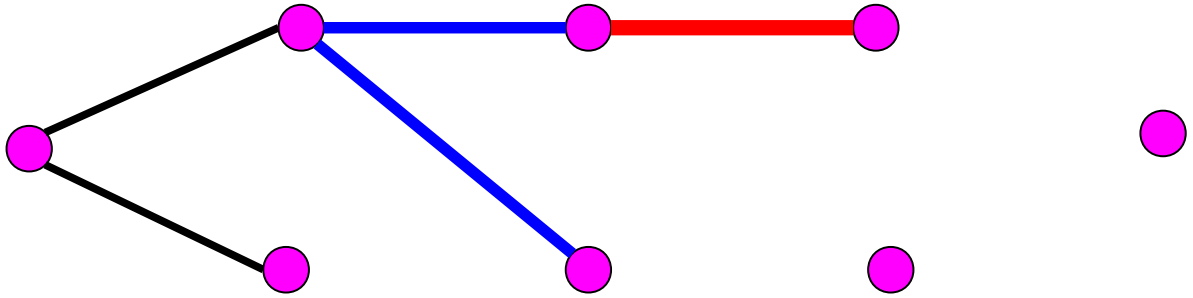


Construction with cost perturbation

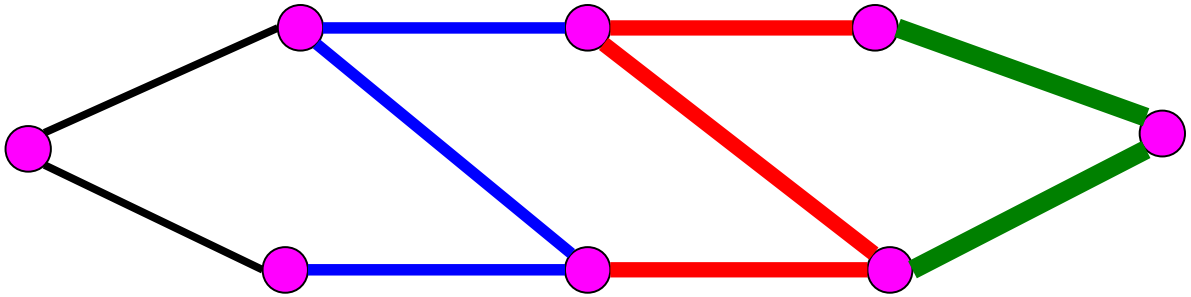


W( ) < W( ) < W( ) < W( )

Perturb with costs increasing from top to bottom.

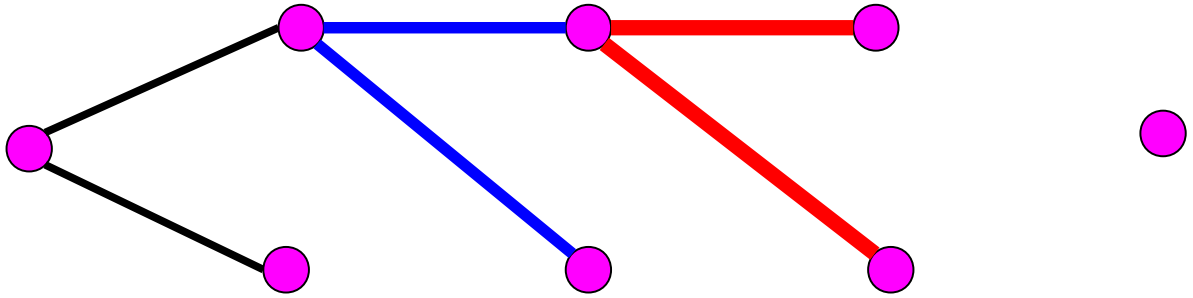


Construction with cost perturbation

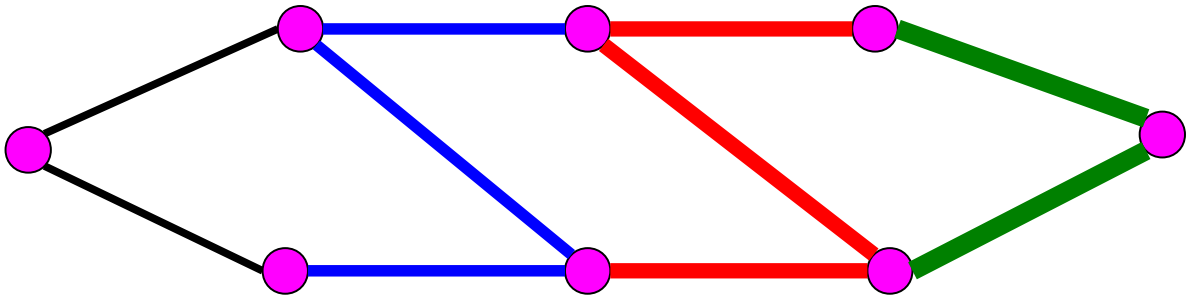


W( ) < W( ) < W( ) < W( )

Perturb with costs increasing from top to bottom.

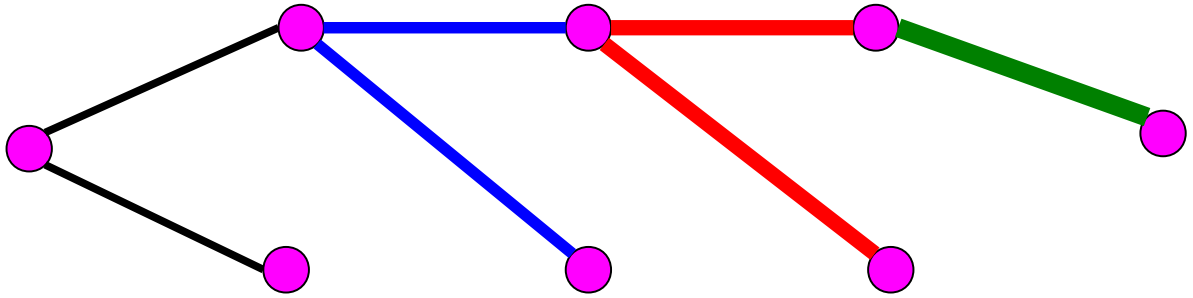


Construction with cost perturbation

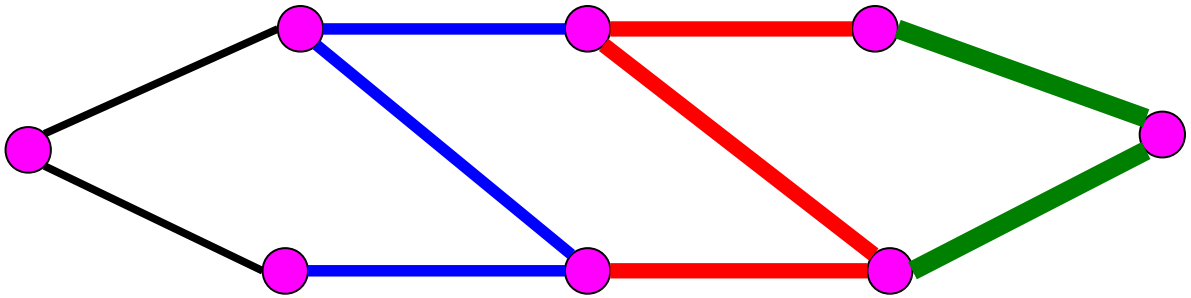


Perturb with costs increasing from top to bottom.

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



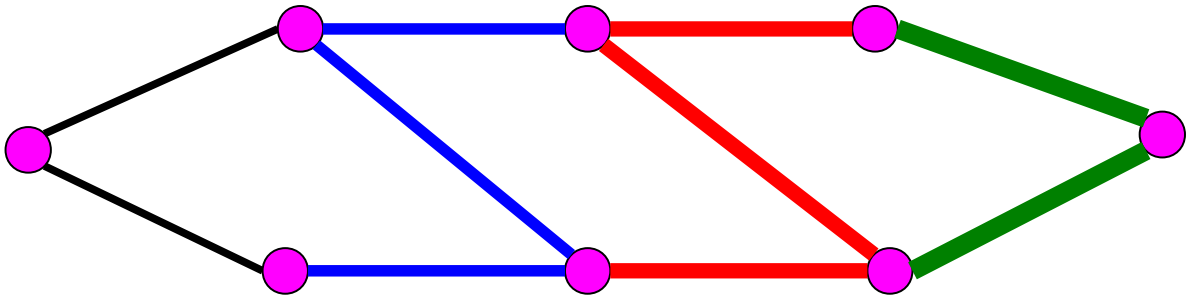
Construction with cost perturbation



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

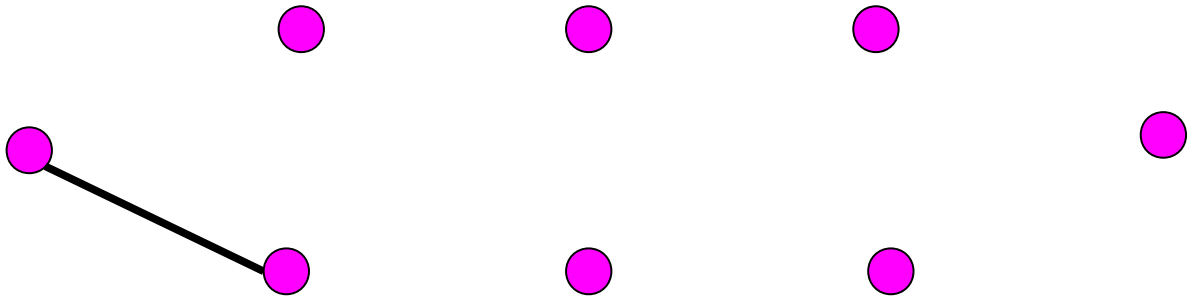
Perturb with costs increasing from bottom to top.

Construction with cost perturbation

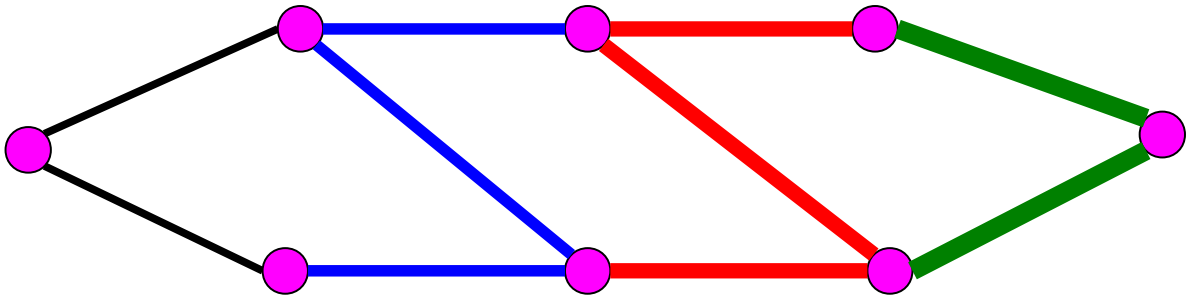


W( ) < W( ) < W( ) < W( )

Perturb with costs increasing from bottom to top.

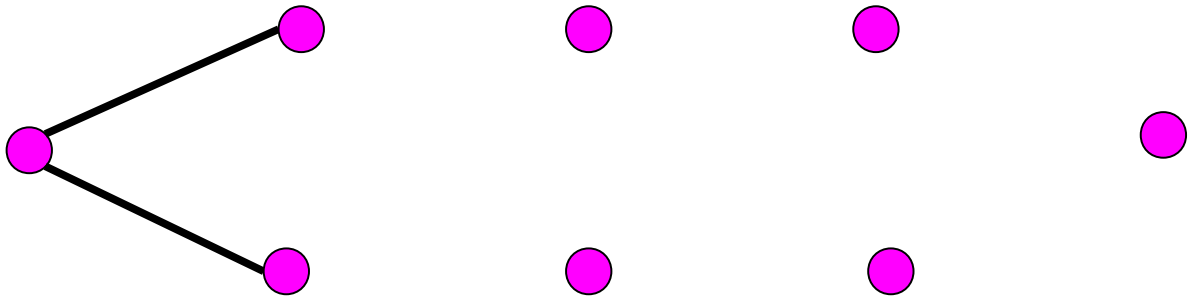


Construction with cost perturbation

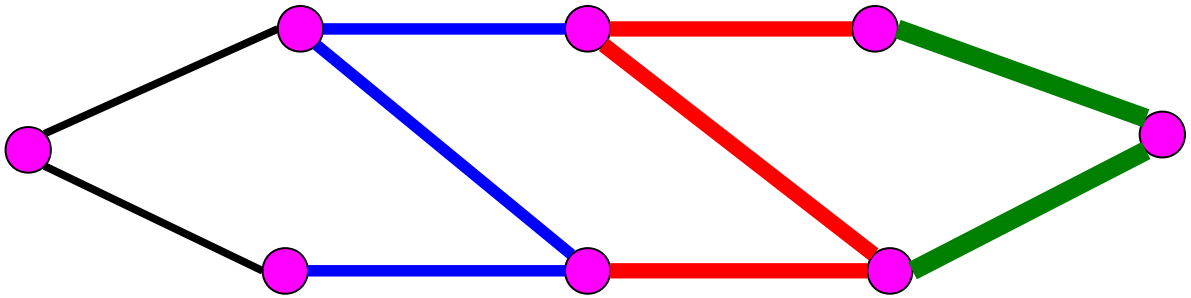


Perturb with costs increasing from bottom to top.

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

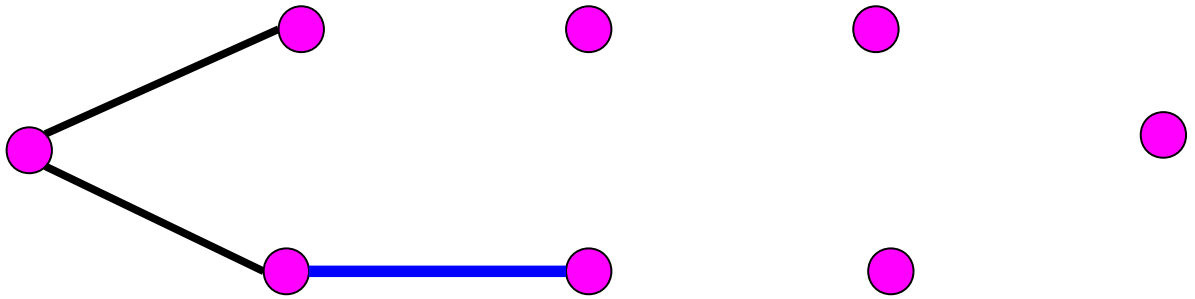


Construction with cost perturbation

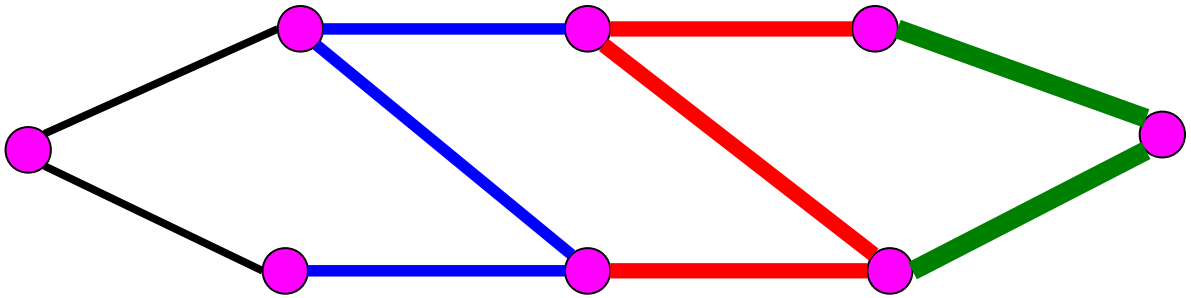


Perturb with costs increasing from bottom to top.

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

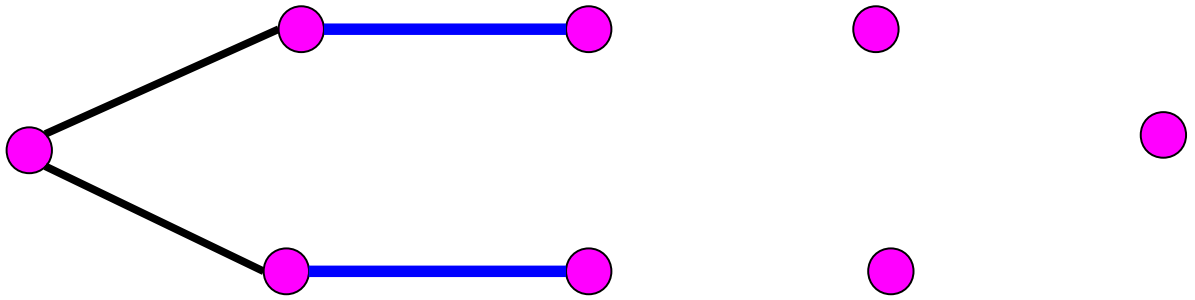


Construction with cost perturbation

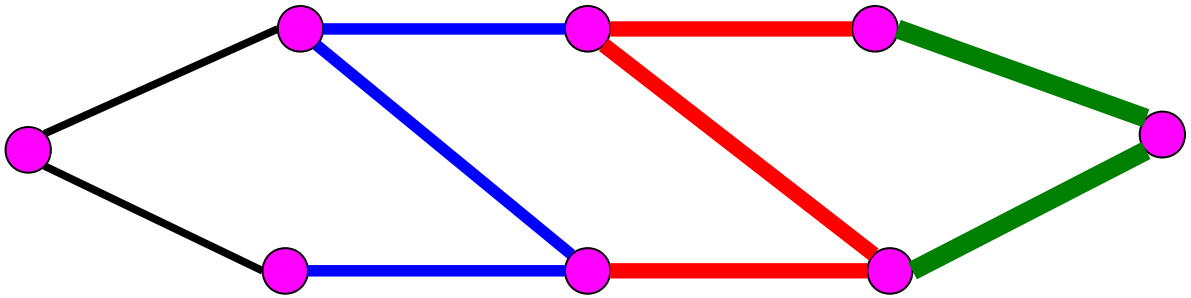


Perturb with costs increasing from bottom to top.

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

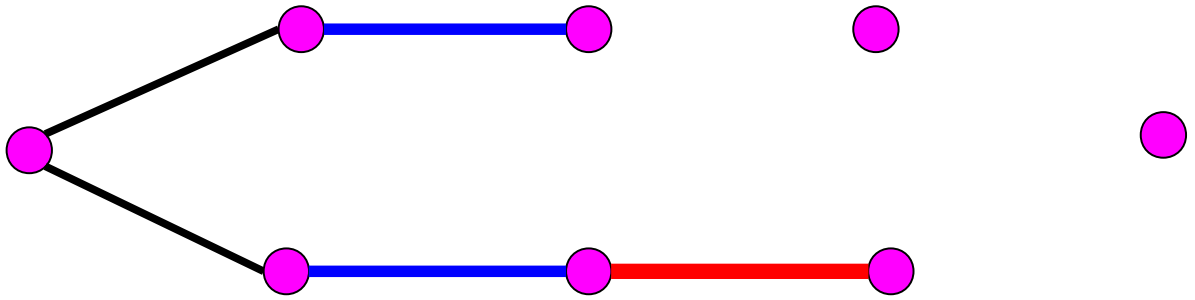


Construction with cost perturbation

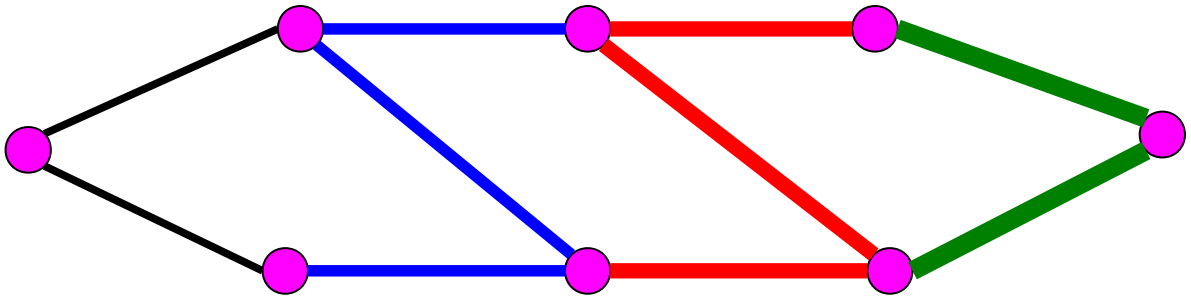


Perturb with costs increasing from bottom to top.

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

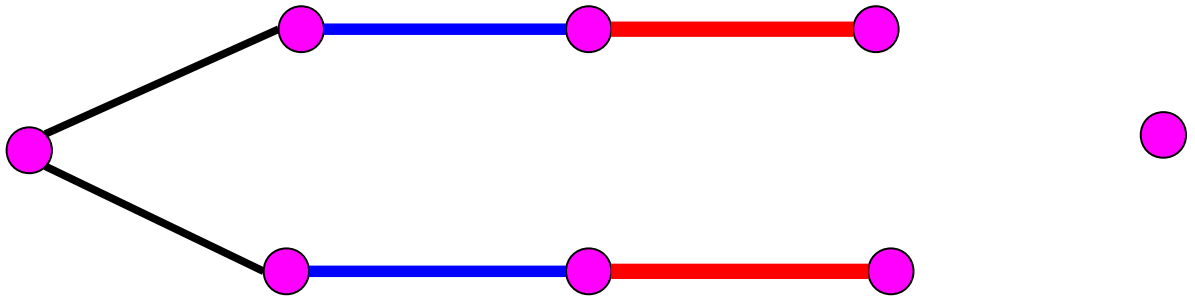


Construction with cost perturbation

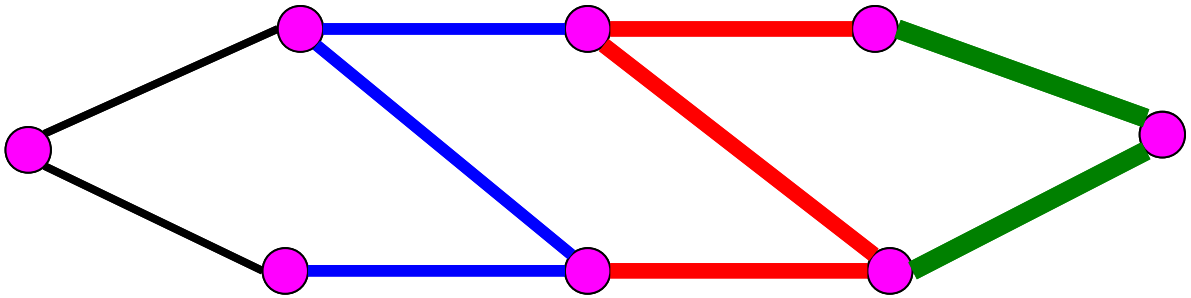


Perturb with costs increasing from bottom to top.

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

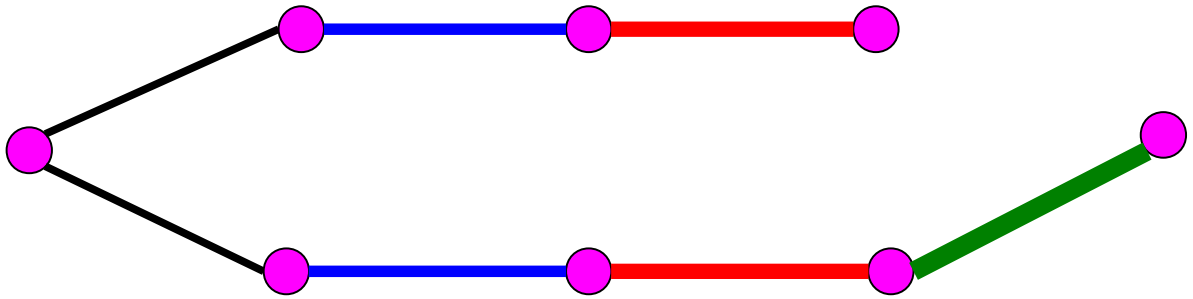


Construction with cost perturbation

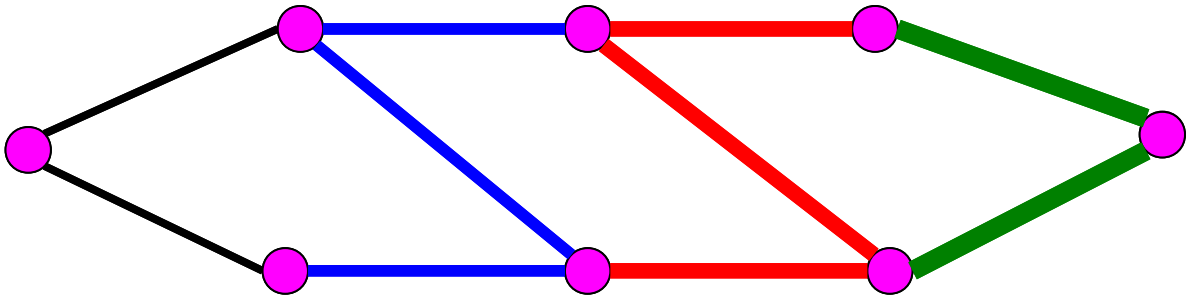


Perturb with costs increasing from bottom to top.

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

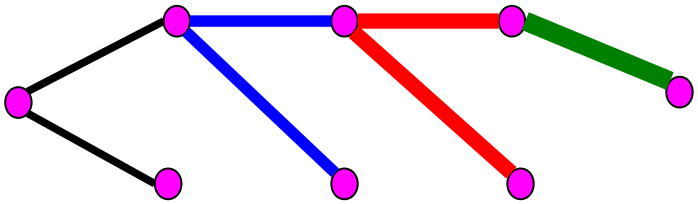


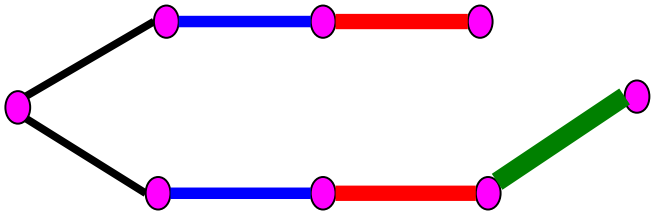
Construction with cost perturbation



Greedy heuristic generates two different spanning trees.

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





Reactive GRASP

Prais & Ribeiro (2000)

When building RCL, what α to use?

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

Choose α at random (uniformly) at each GRASP iteration.

Another approach reacts to search ...

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

The probability that α_k is selected is p_k.

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

Reactive GRASP

Prais & Ribeiro (2000)

Reactive GRASP for minimization ...

Initially $p_k = 1/m$, for $k = 1,\ldots,m$. (α'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 α_k

Every N_α 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 $k = 1,...,m$. (α'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 α_k

Every N_α GRASP iterations, compute

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

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

The more suitable is α_k, the larger is q_k, and consequently p_k, making α_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 $y \in N(x)$ | $f(y) < f(x)$) do

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

end while

return x;

first improving

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 ( ∃ y ∈ N(x) | f(y) < f(x) ) do
    x = y;
end while
return x;
```

first improving

```
x = x^0;
y = argmin { f(z) | z ∈ N(x) };
while ( f(y) < f(x) ) do
    x = y;
    y = argmin { f(z) | z ∈ 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 ( ∃ y ∈ N(x) | f(y) < f(x) ) do
    x = y;
end while
return x;
```

first improving

```
x = x^0;
y = argmin { f(z) | z ∈ N(x) };
while ( f(y) < f(x) ) do
    x = y;
    y = argmin { f(z) | z ∈ 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 ( ∃ y ∈ N(x) | f(y) < f(x) ) do
    x = y;
end while
return x;
```

first improving

```
x = x^0;
y = argmin { f(z) | z ∈ N(x) };
while ( f(y) < f(x) ) do
    x = y;
    y = argmin { f(z) | z ∈ 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 ( ∃ y ∈ N(x) | 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 = argmin { f(z) | z ∈ N(x) };
  while ( f(y) < f(x) ) do
      x = y;
      y = argmin { f(z) | z ∈ 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, ..., 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.

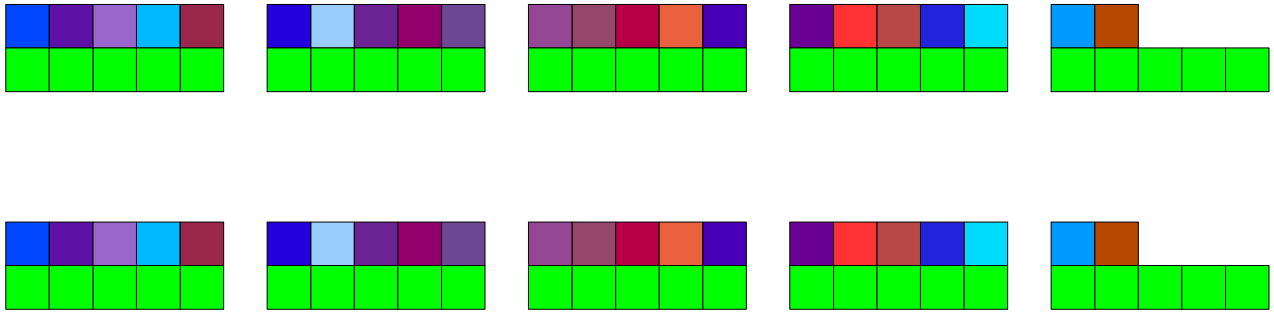
solution 

GRASP VND local search

example: scheduling of multi-grouped units

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

solution

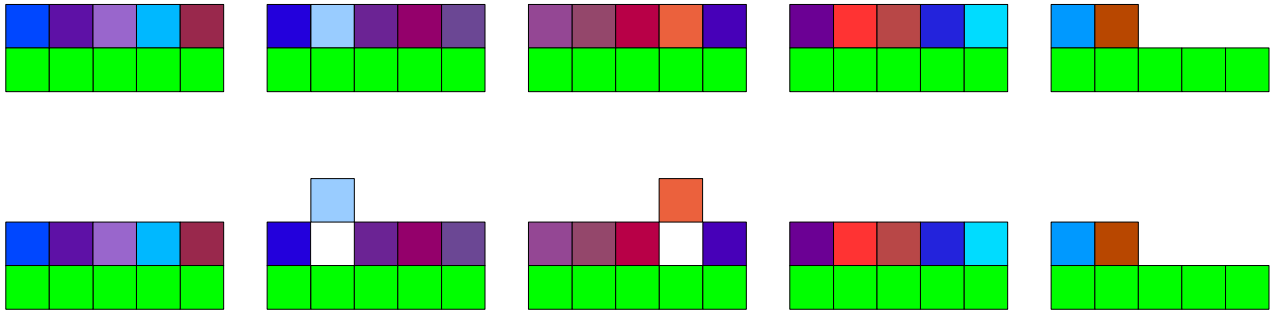


GRASP VND local search

example: scheduling of multi-grouped units

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

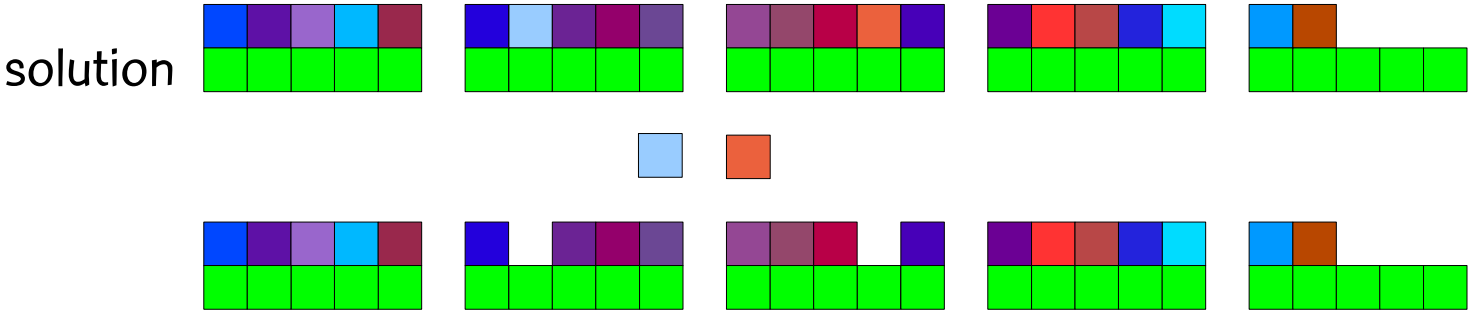
solution



GRASP VND local search

example: scheduling of multi-grouped units

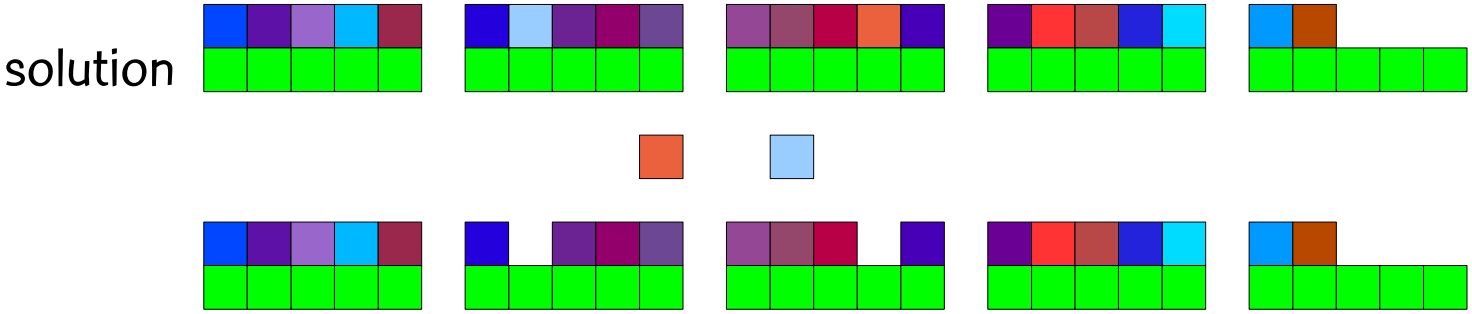
Swap units neighborhood: Swaps places of two units assigned to distinct periods.



GRASP VND local search

example: scheduling of multi-grouped units

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

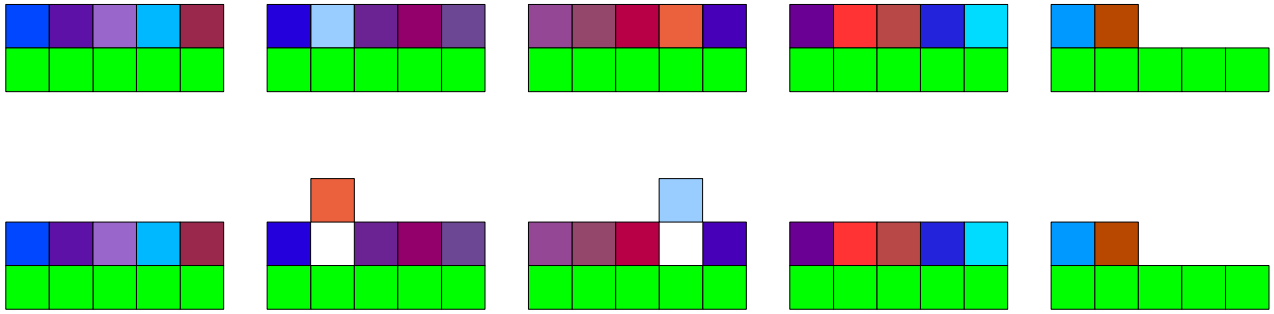


GRASP VND local search

example: scheduling of multi-grouped units

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

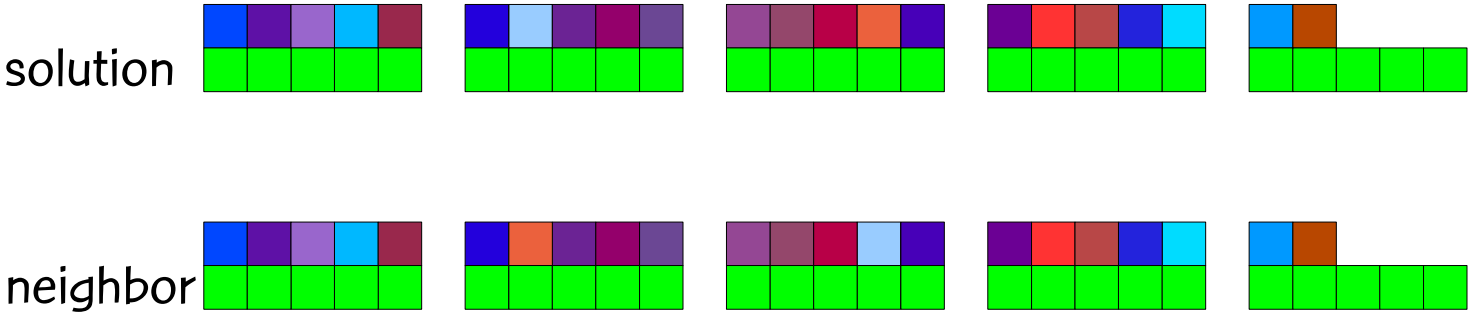
solution



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.

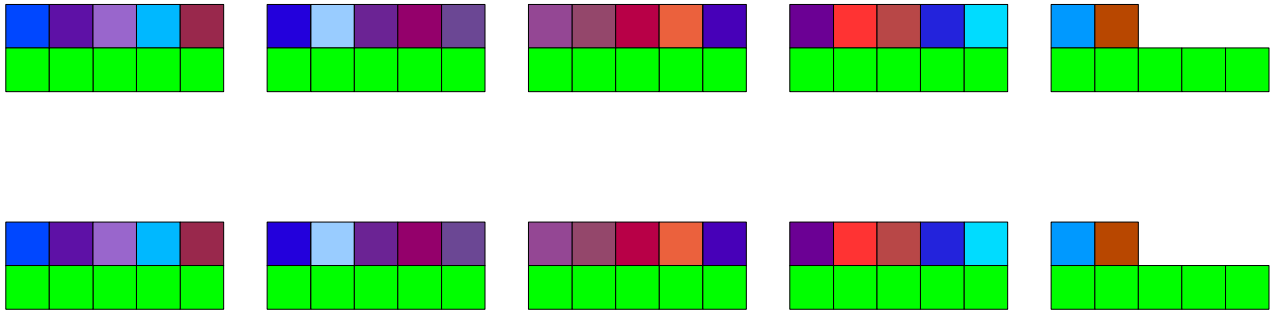
solution 

GRASP VND local search

example: scheduling of multi-grouped units

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

solution

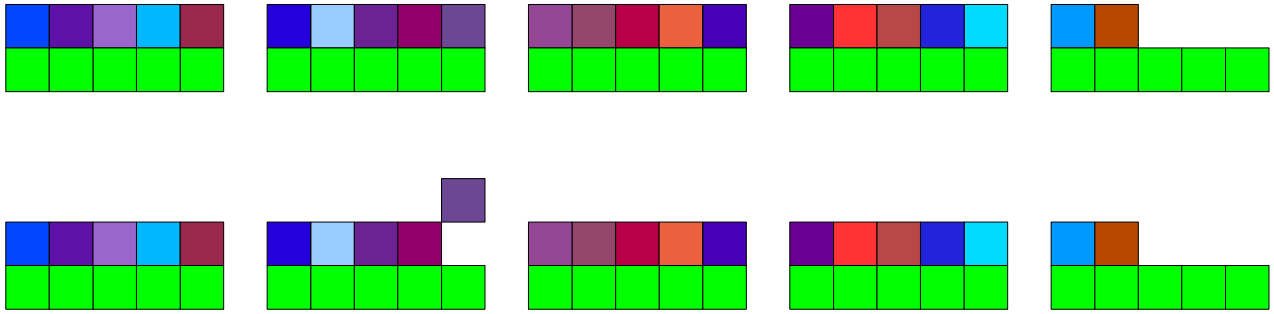


GRASP VND local search

example: scheduling of multi-grouped units

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

solution

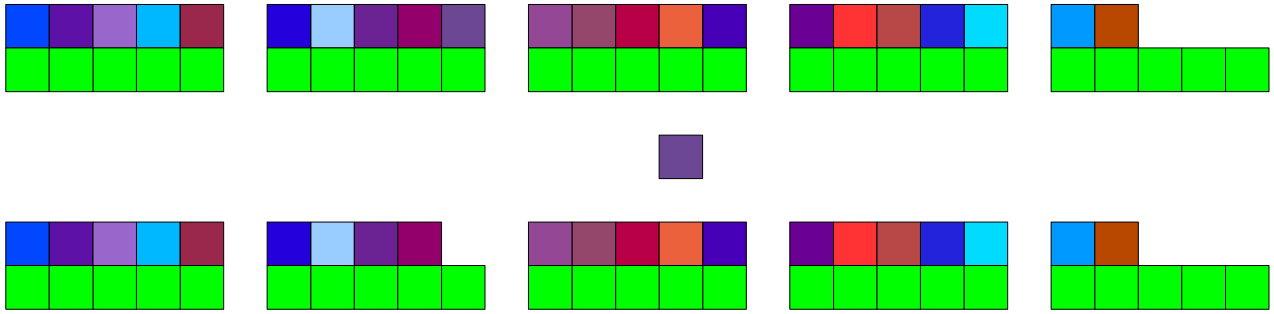


GRASP VND local search

example: scheduling of multi-grouped units

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

solution

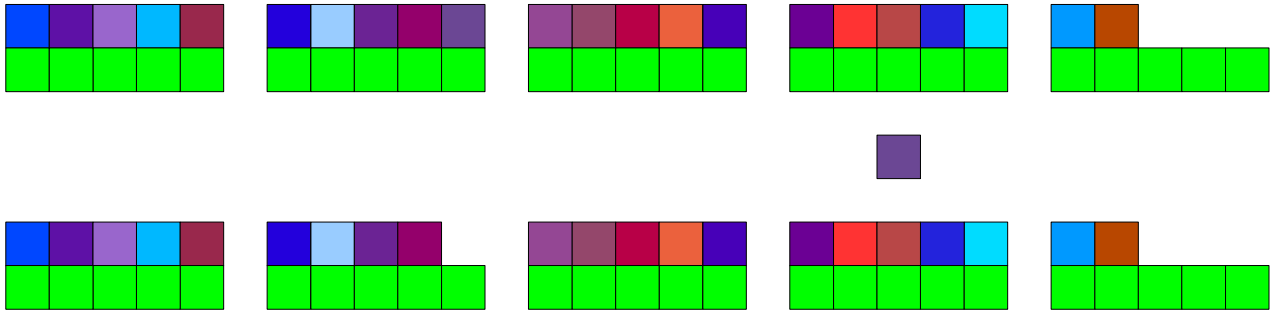


GRASP VND local search

example: scheduling of multi-grouped units

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

solution

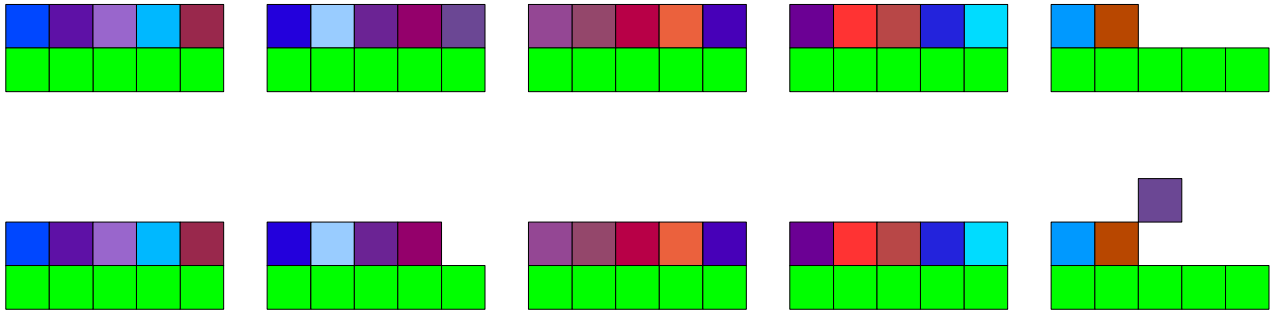


GRASP VND local search

example: scheduling of multi-grouped units

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

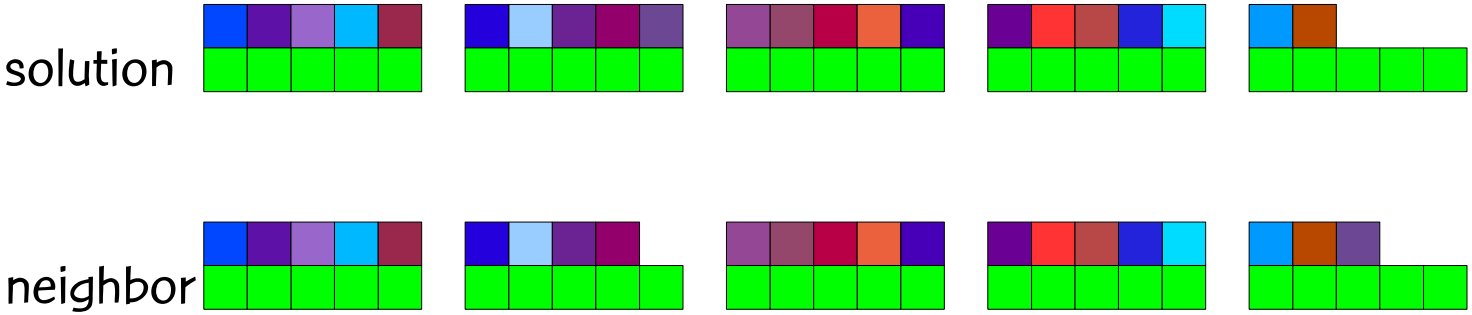
solution



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

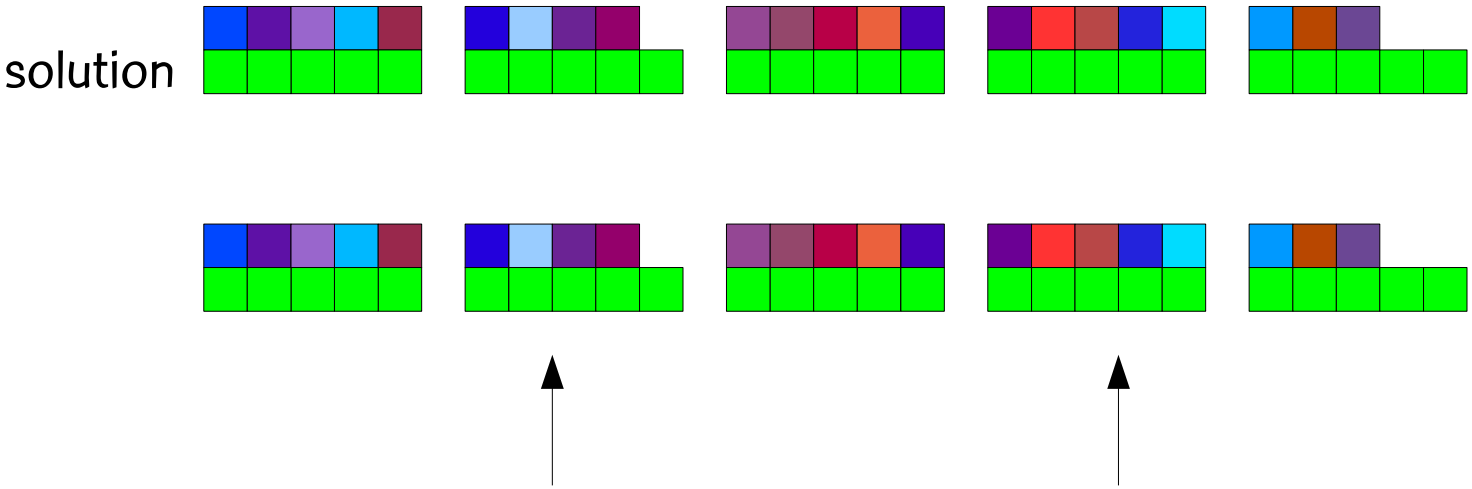
Swap periods neighborhood: Swap all units in period i with all units in period j.

solution 

GRASP VND local search

example: scheduling of multi-grouped units

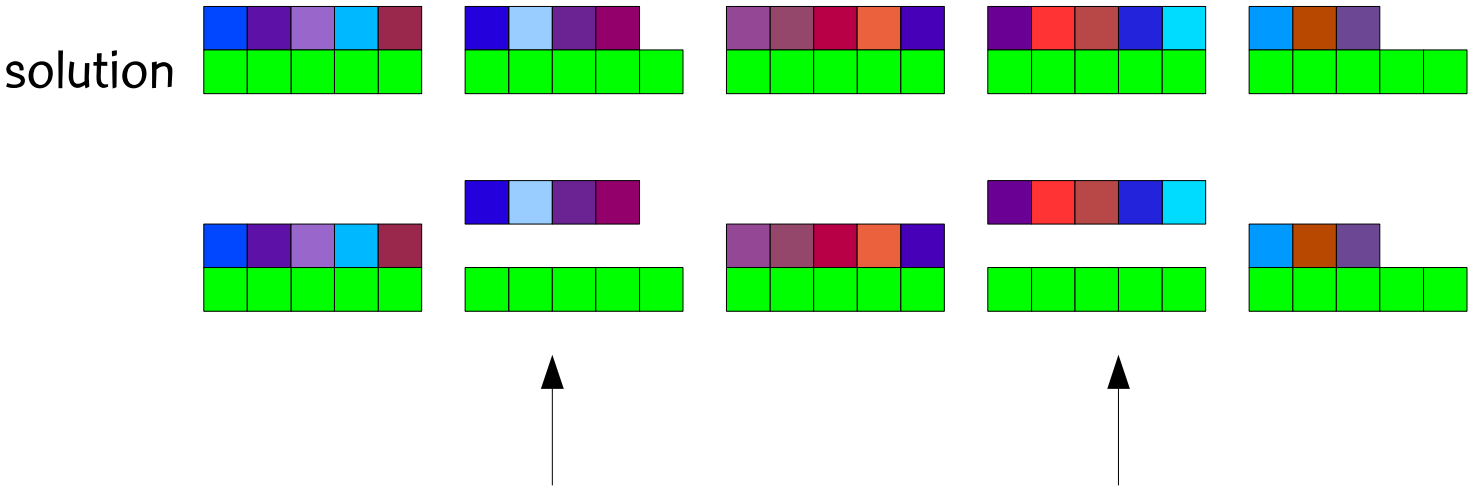
Swap periods neighborhood: Swap all units in period i with all units in period j.



GRASP VND local search

example: scheduling of multi-grouped units

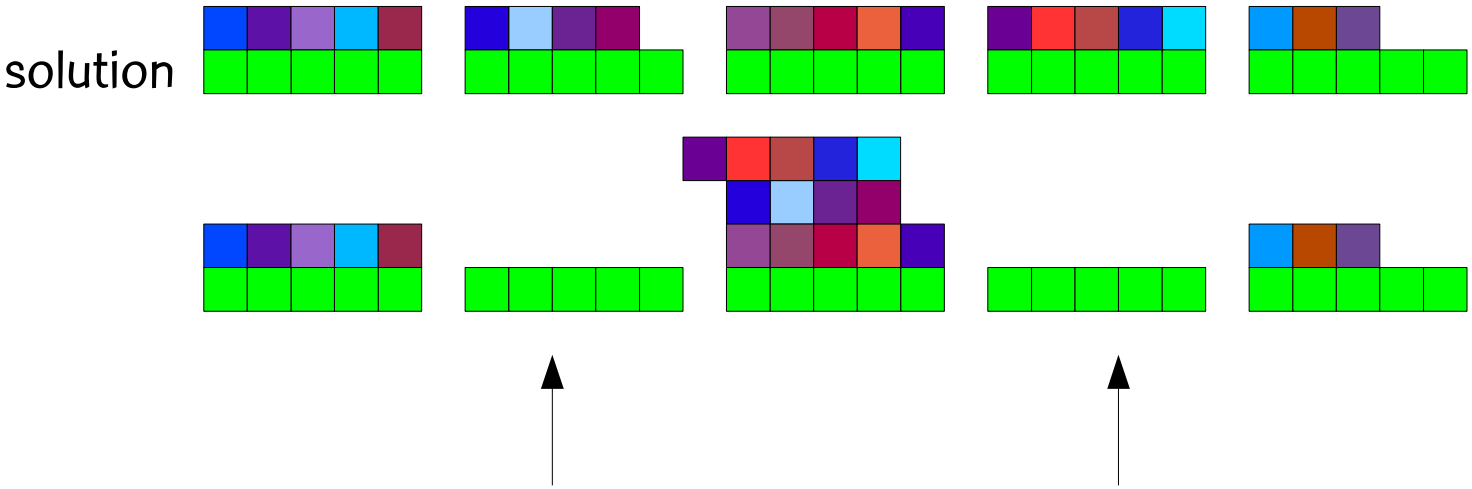
Swap periods neighborhood: Swap all units in period i with all units in period j.



GRASP VND local search

example: scheduling of multi-grouped units

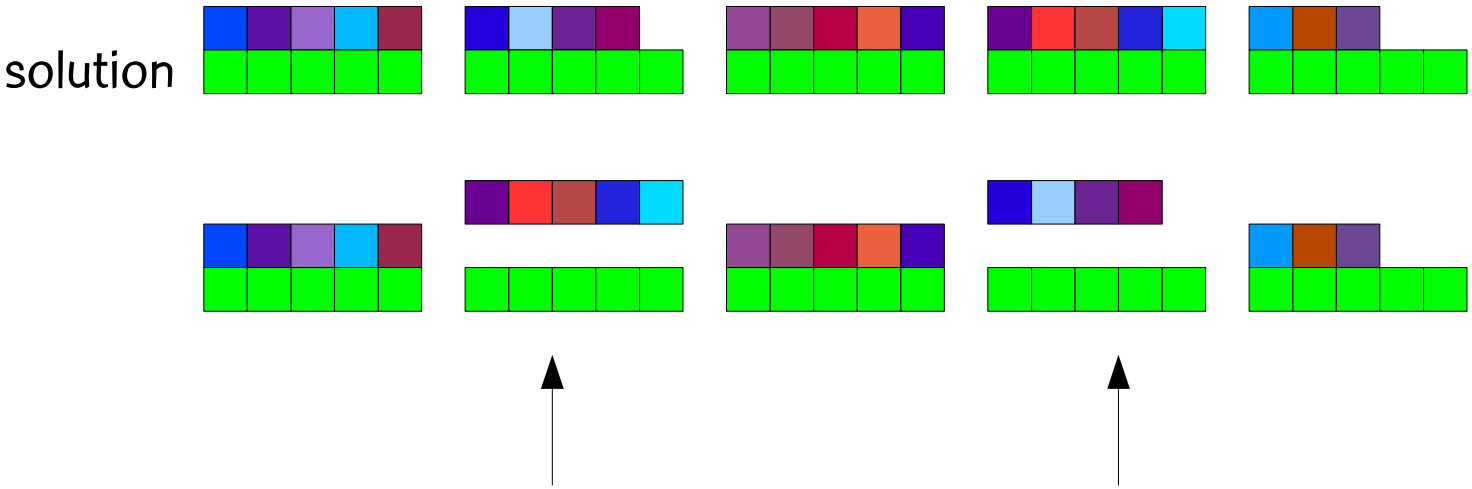
Swap periods neighborhood: Swap all units in period i with all units in period j.



GRASP VND local search

example: scheduling of multi-grouped units

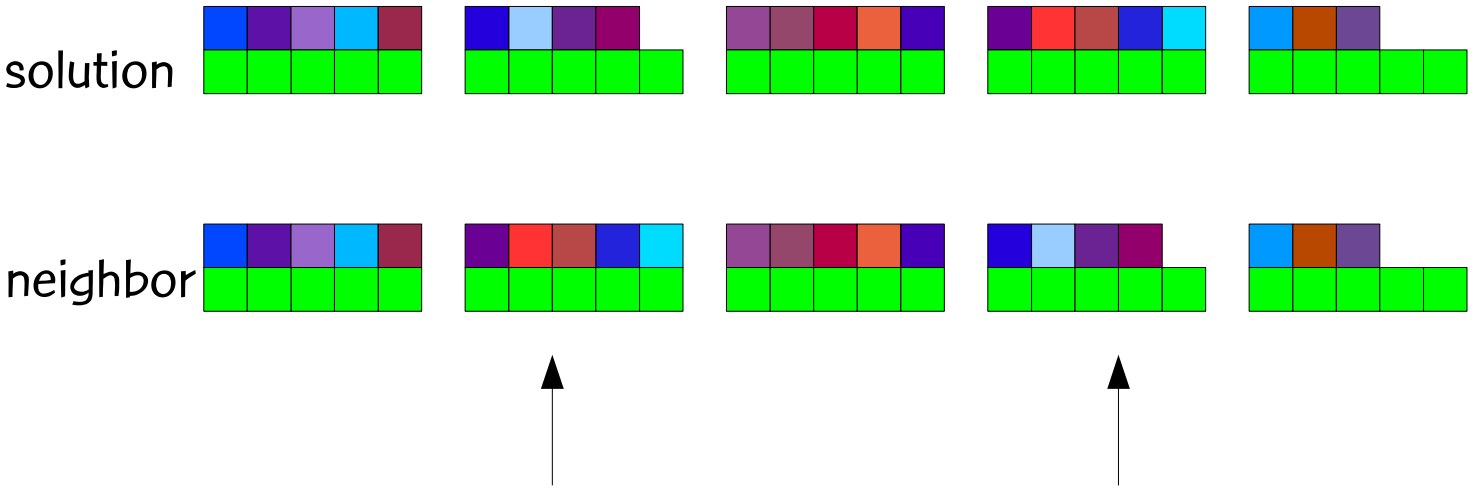
Swap periods neighborhood: Swap all units in period i with all units in period j.



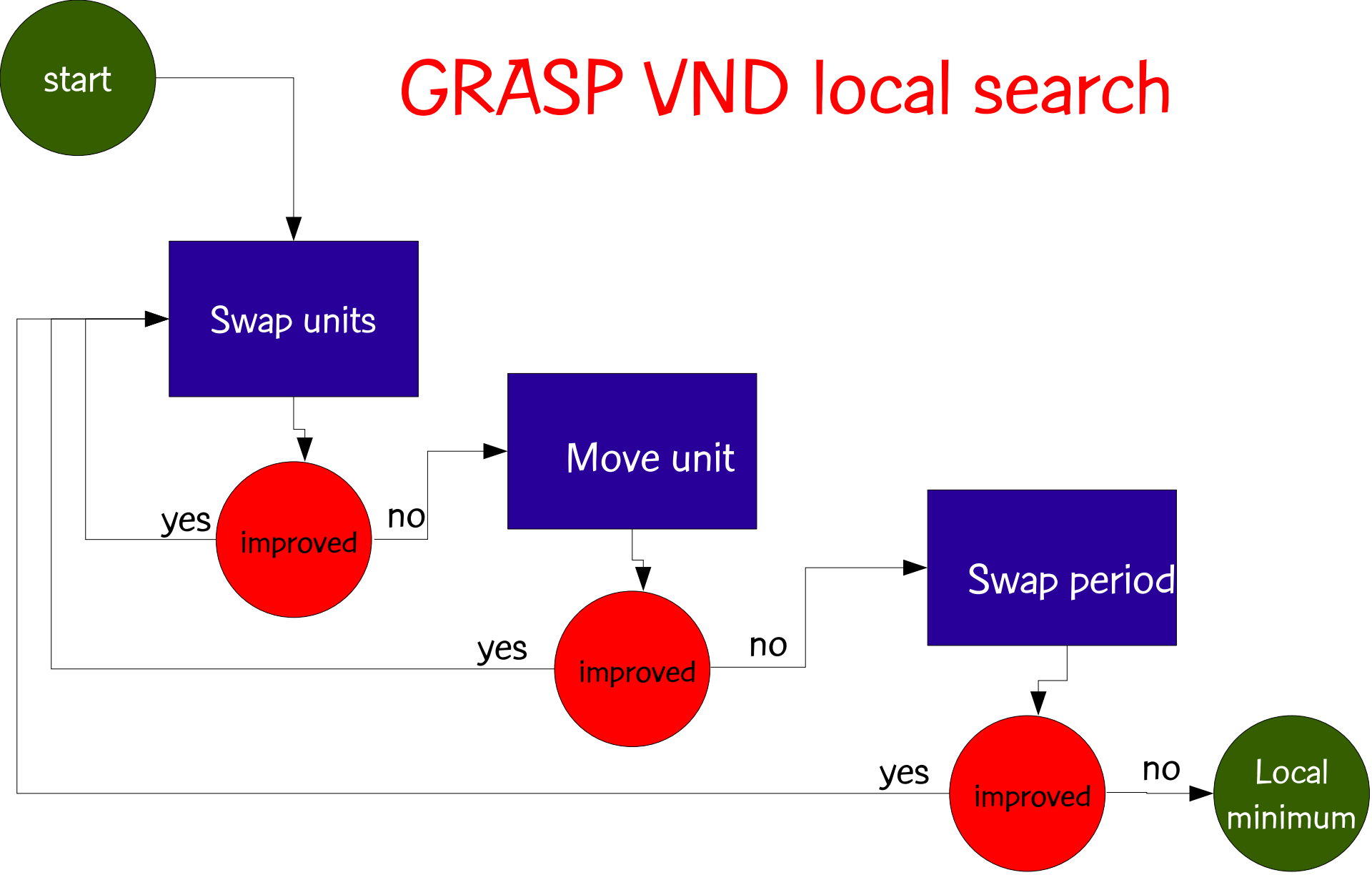
GRASP VND local search

example: scheduling of multi-grouped units

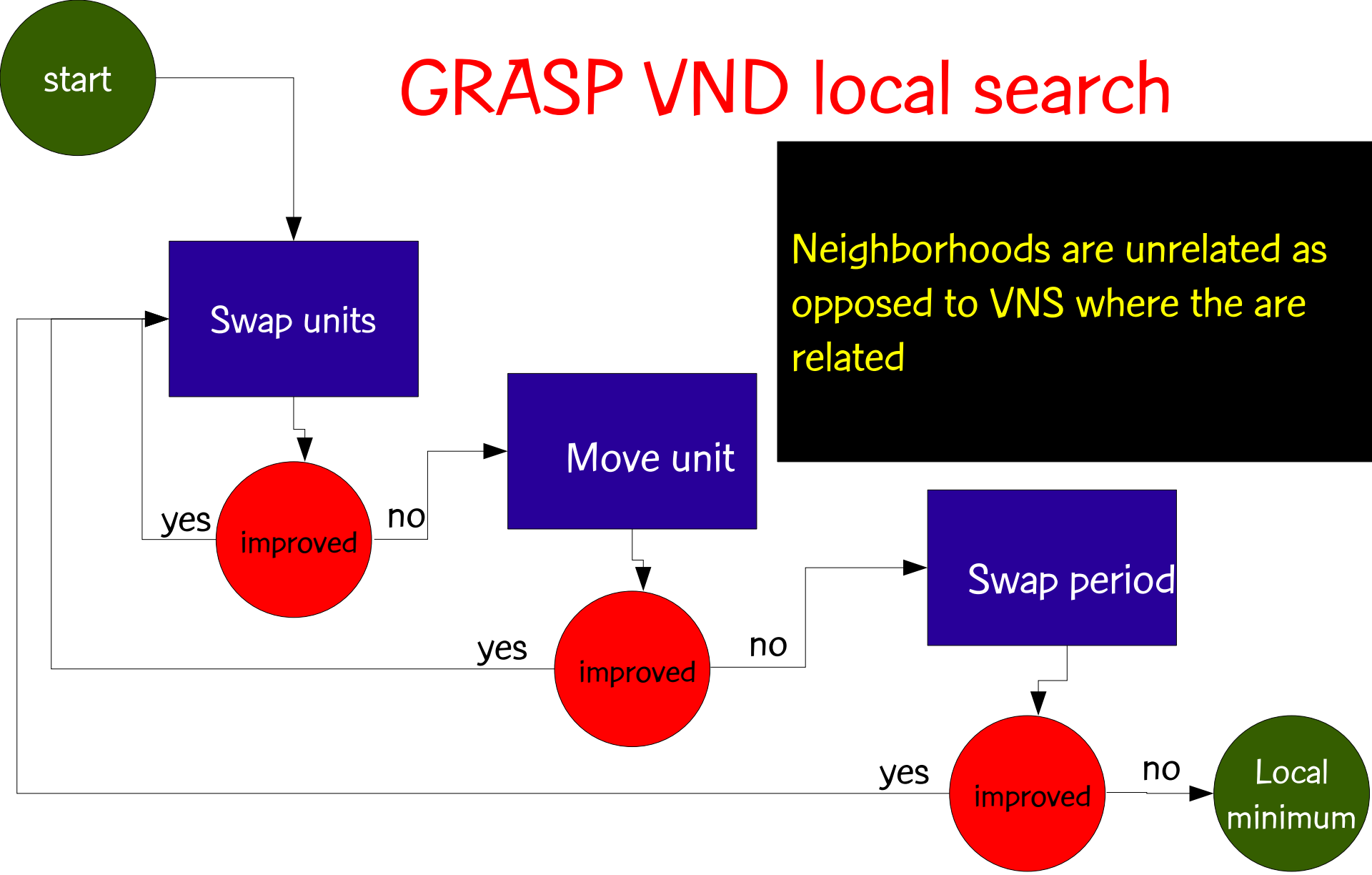
Swap periods neighborhood: Swap all units in period i with all units in period j.



GRASP VND local search

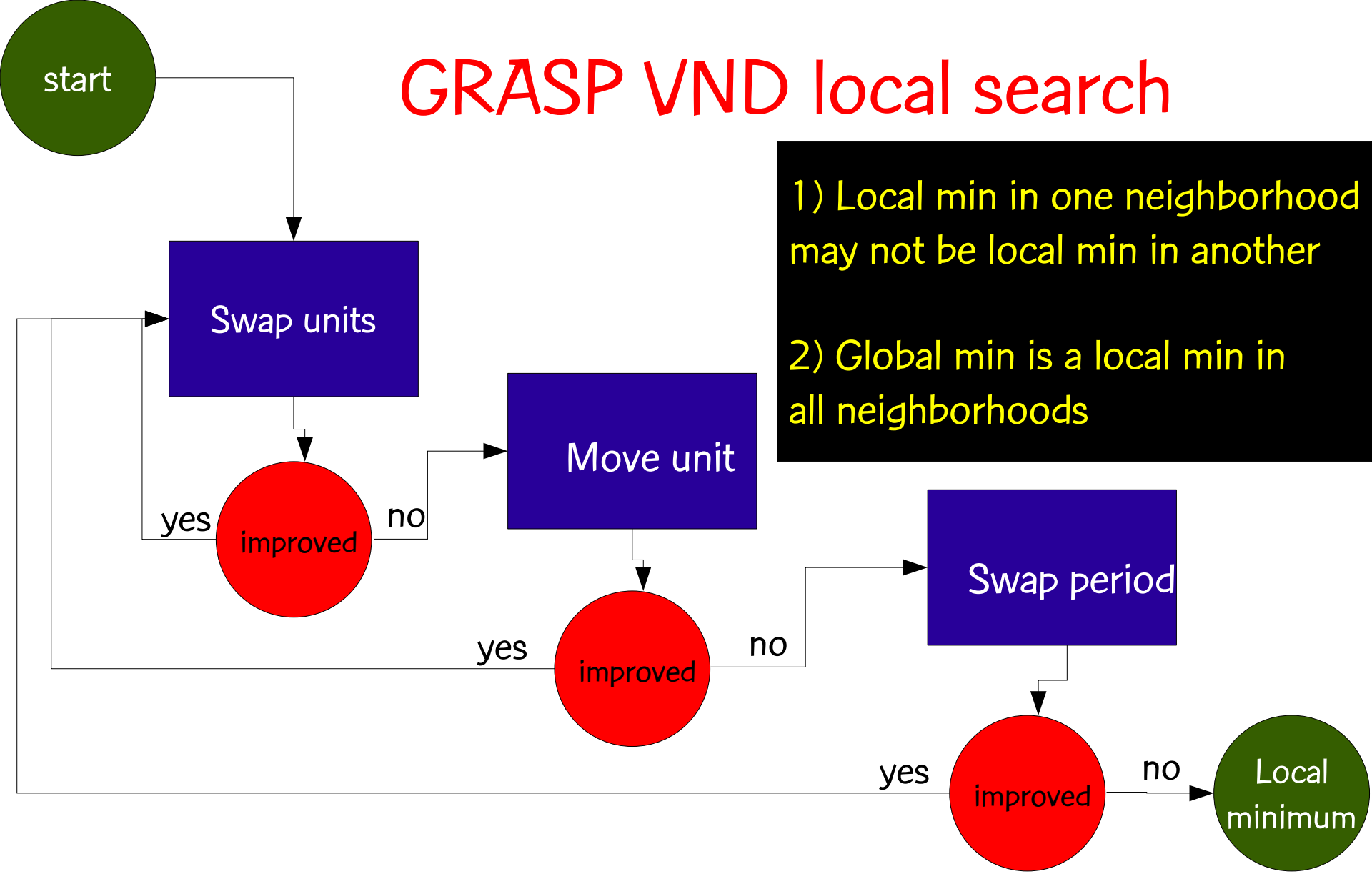


GRASP VND local search



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

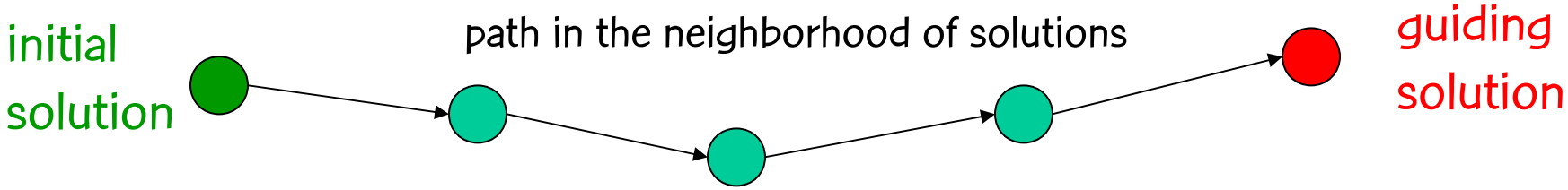
Intensification strategy exploring trajectories connecting elite solutions (Glover, 1996)

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

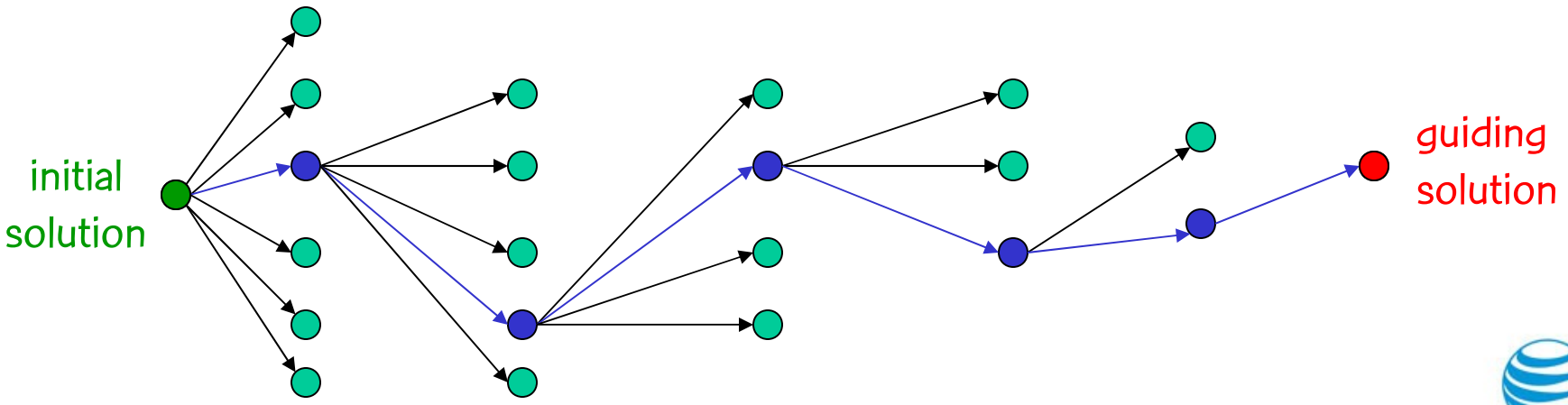
Exploration of trajectories that connect high quality (elite) solutions:



Path-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:

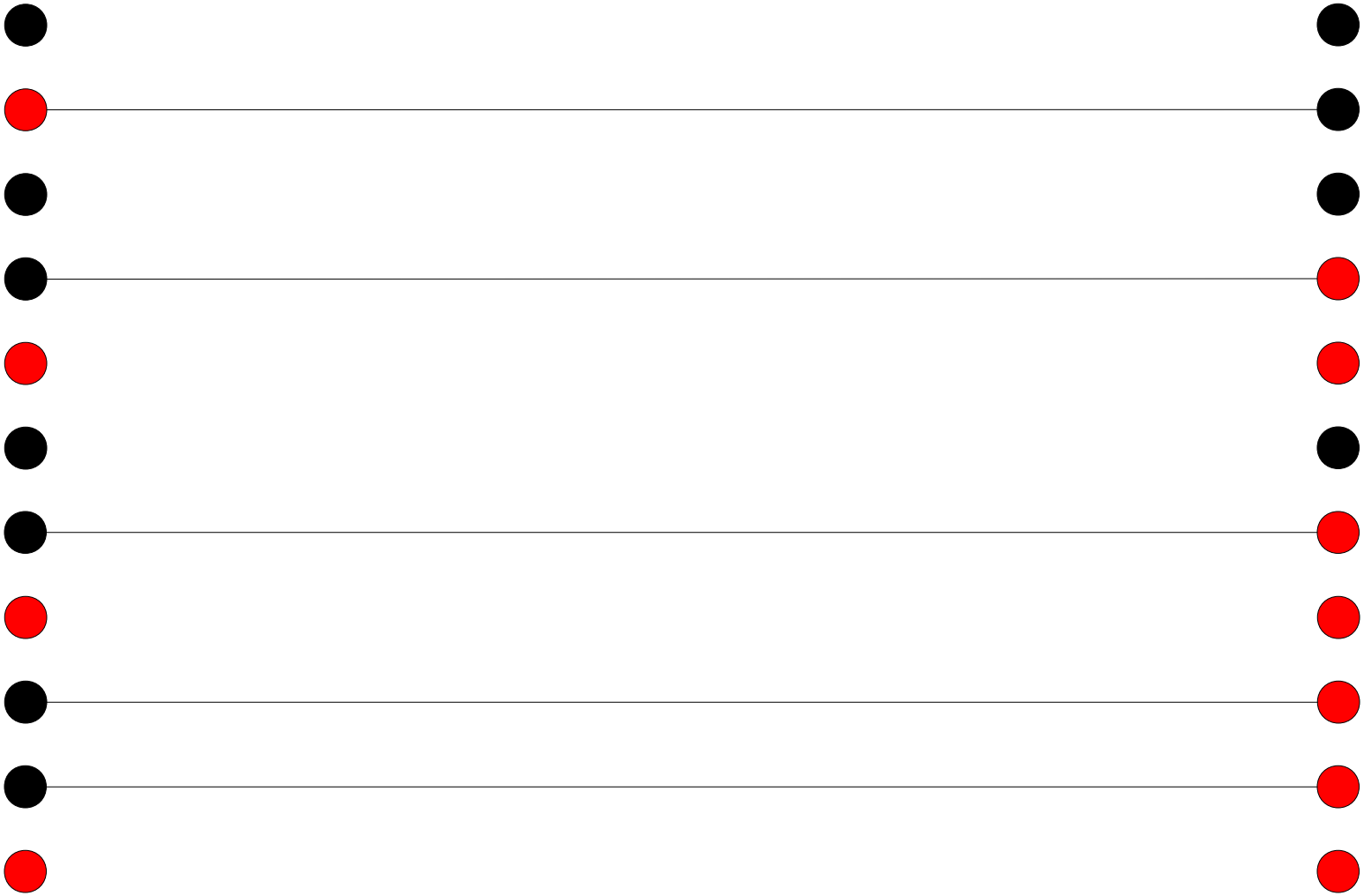


starting solution

PR example

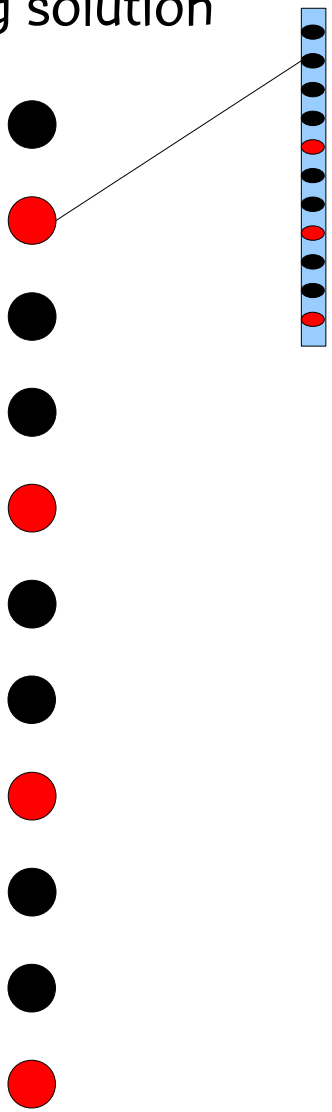
guiding solution

starting solution x PR example guiding solution y



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

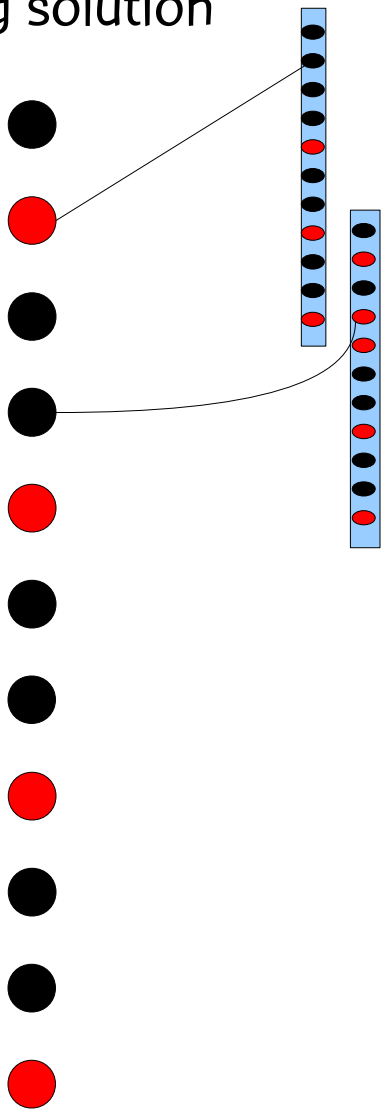
starting solution



PR example

guiding solution

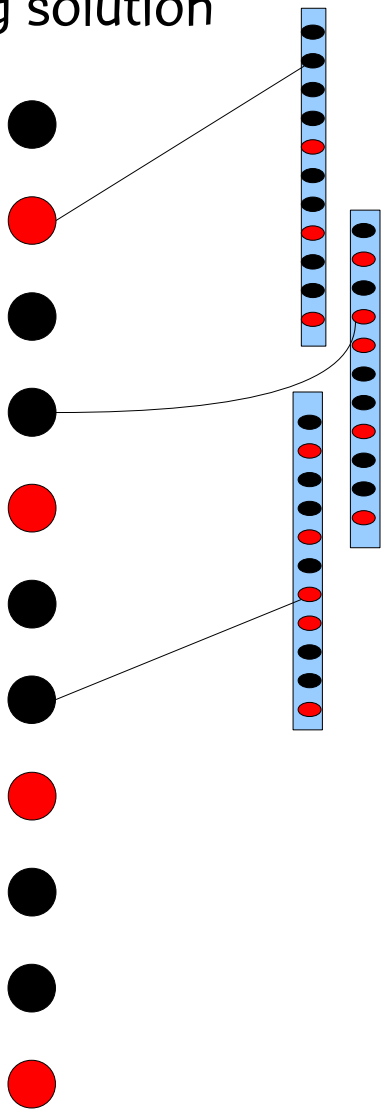
starting solution



PR example

guiding solution

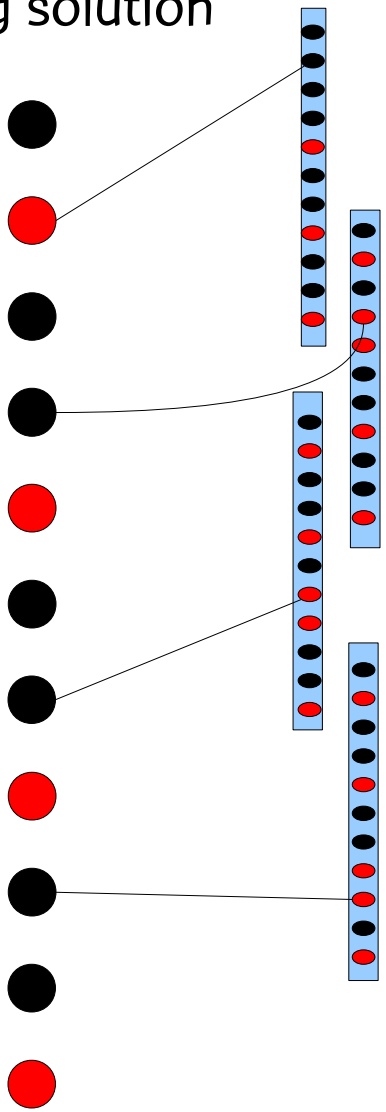
starting solution



PR example

guiding solution

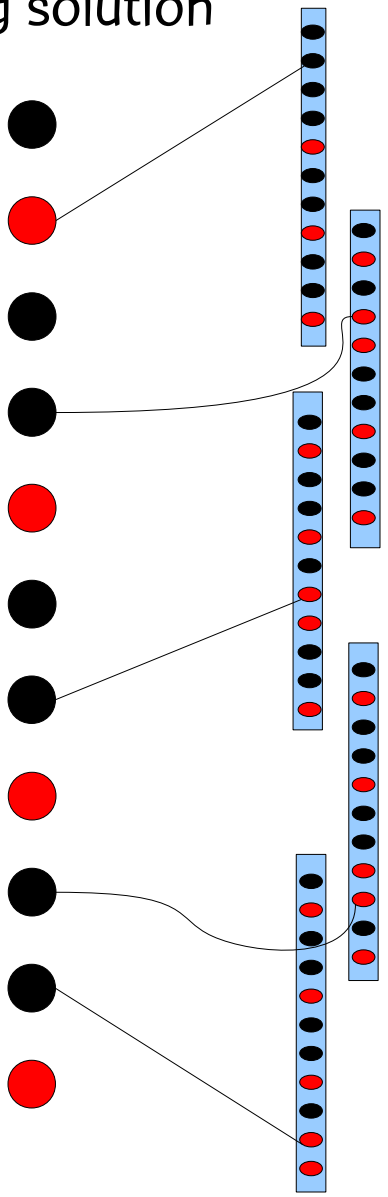
starting solution



PR example

guiding solution

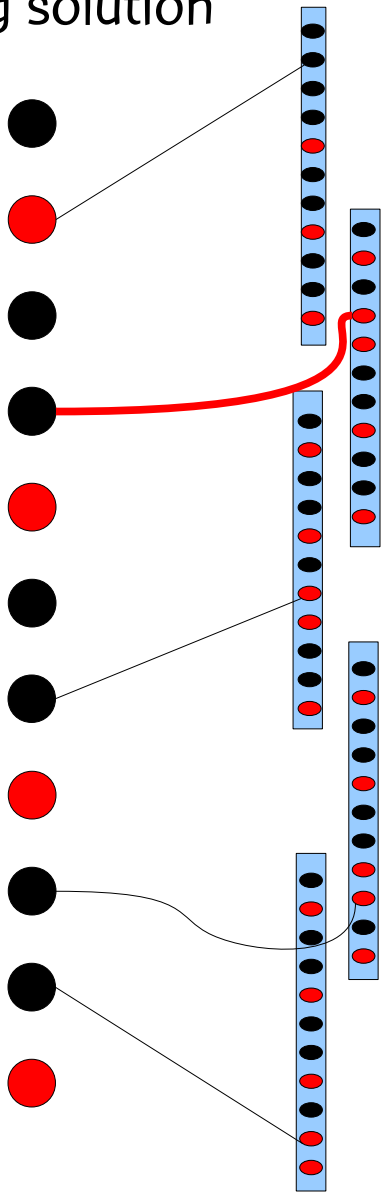
starting solution



PR example

guiding solution

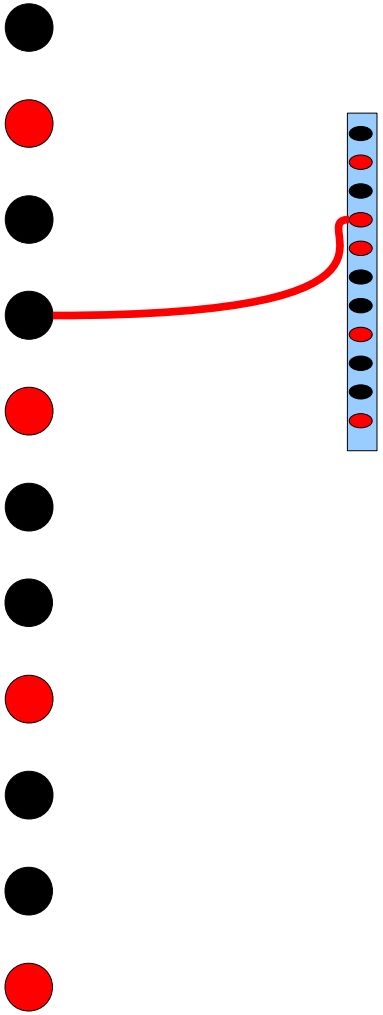
starting solution



PR example

guiding solution

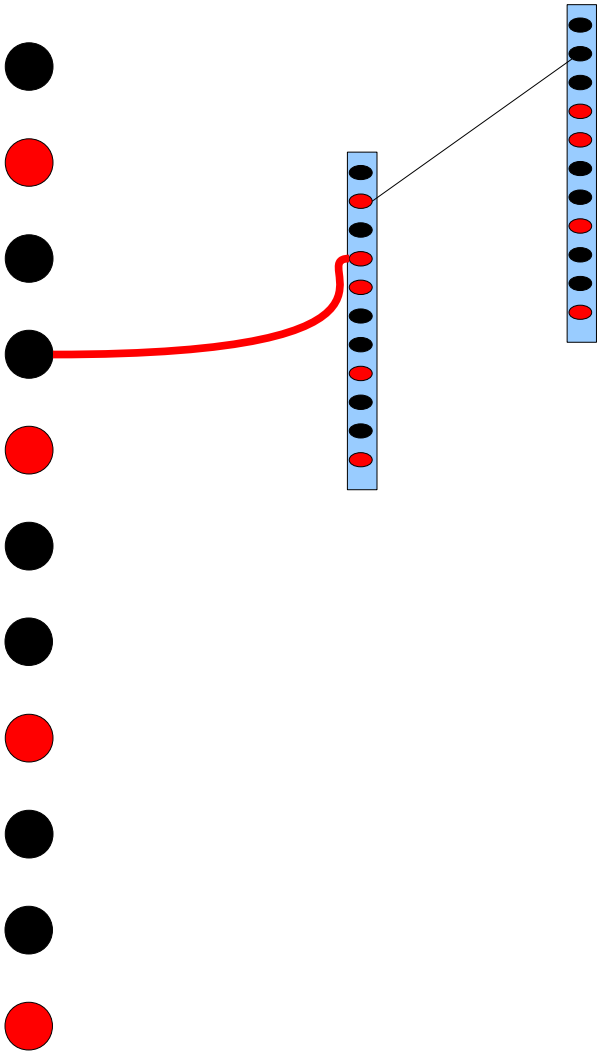
starting solution



PR example

guiding solution

starting solution



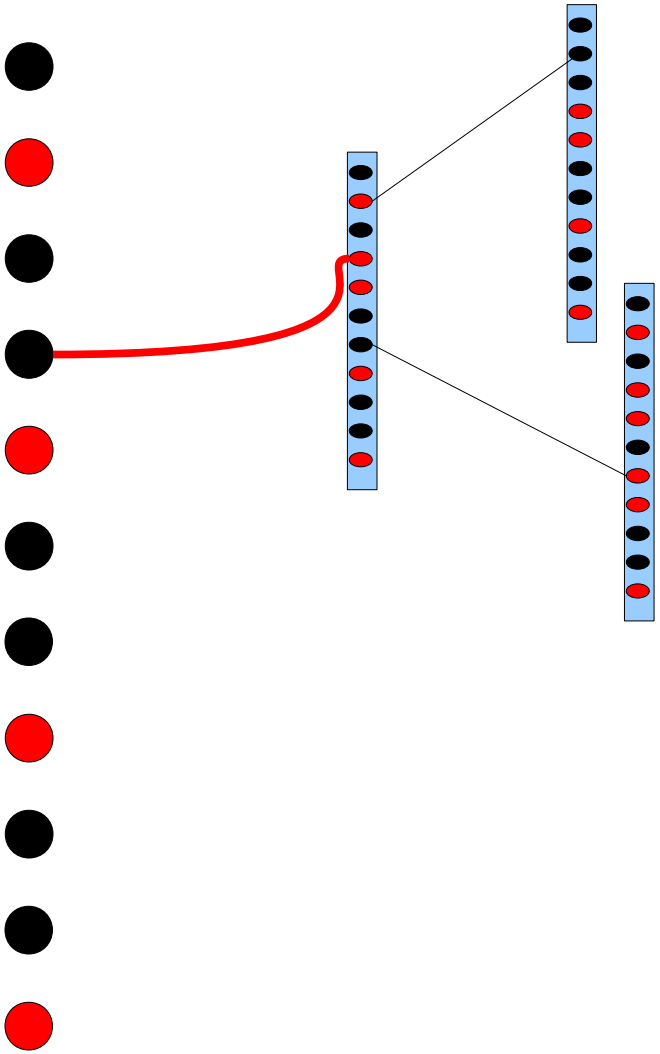
PR example

guiding solution

starting solution

PR example

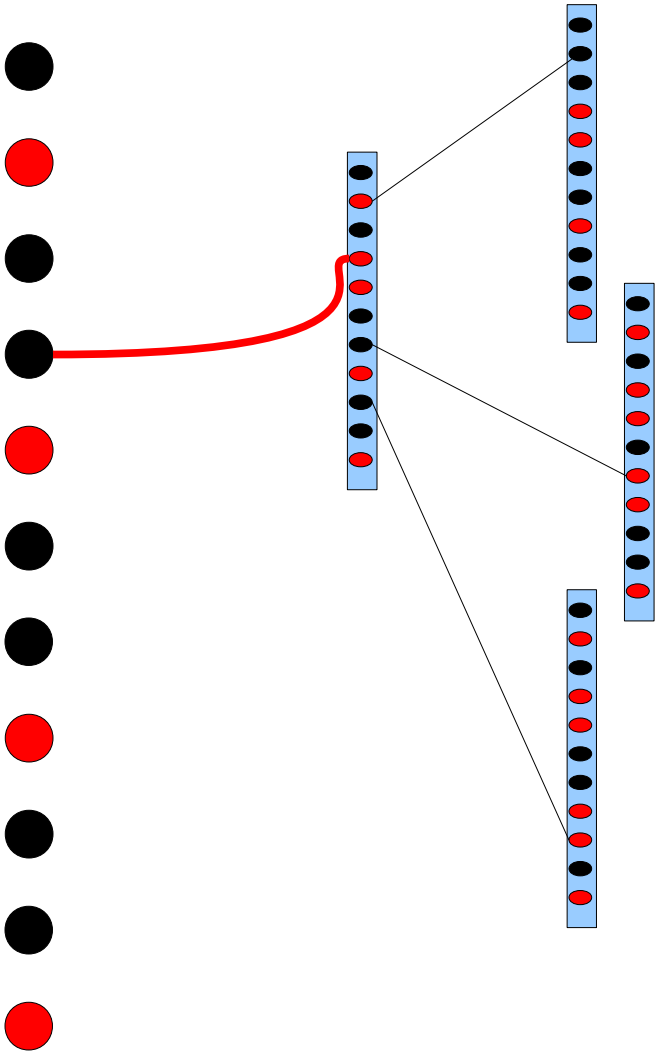
guiding solution



starting solution

PR example

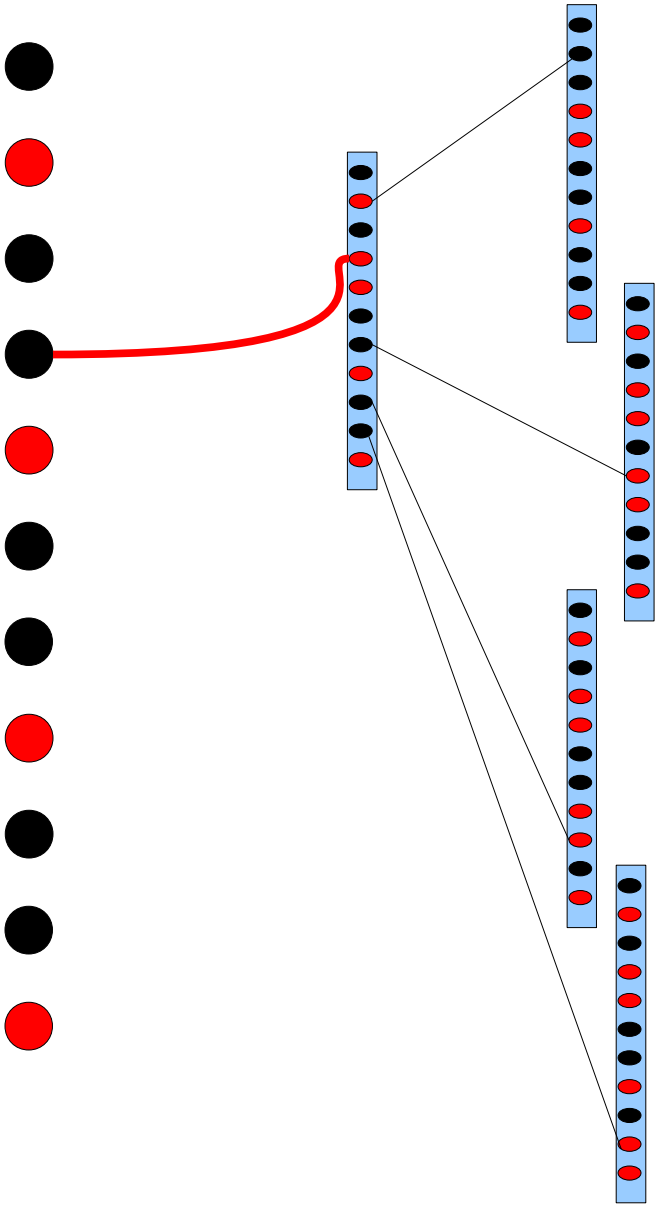
guiding solution



starting solution

PR example

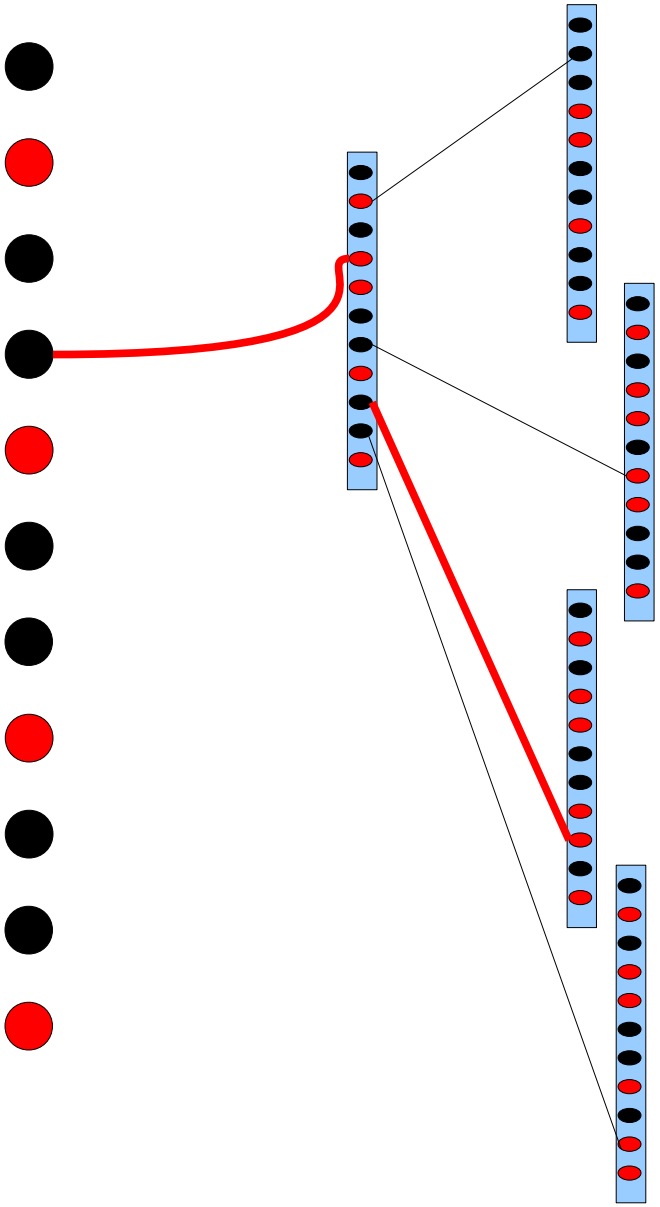
guiding solution



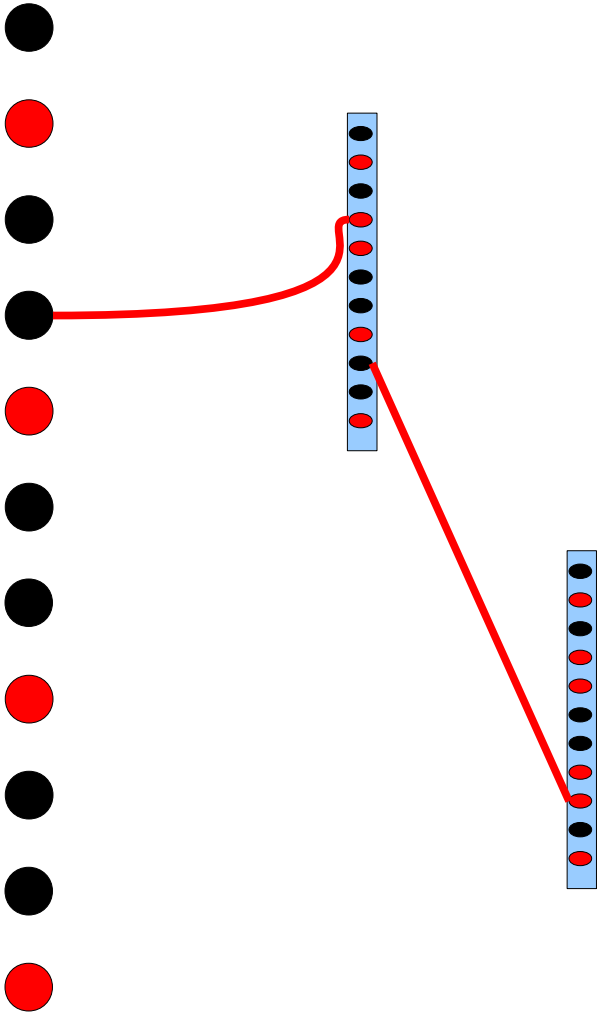
starting solution

PR example

guiding solution



starting solution



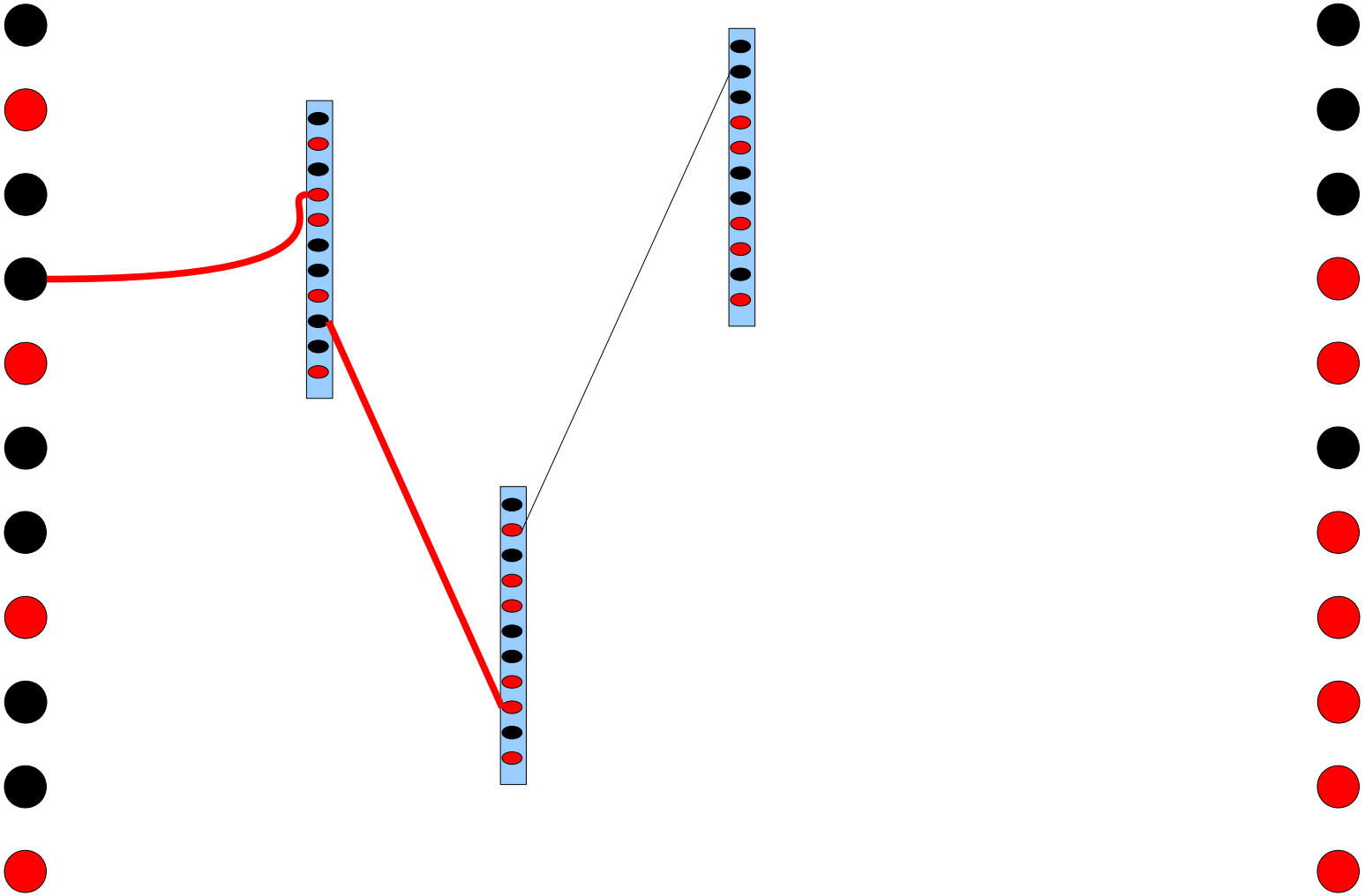
PR example

guiding solution

starting solution

PR example

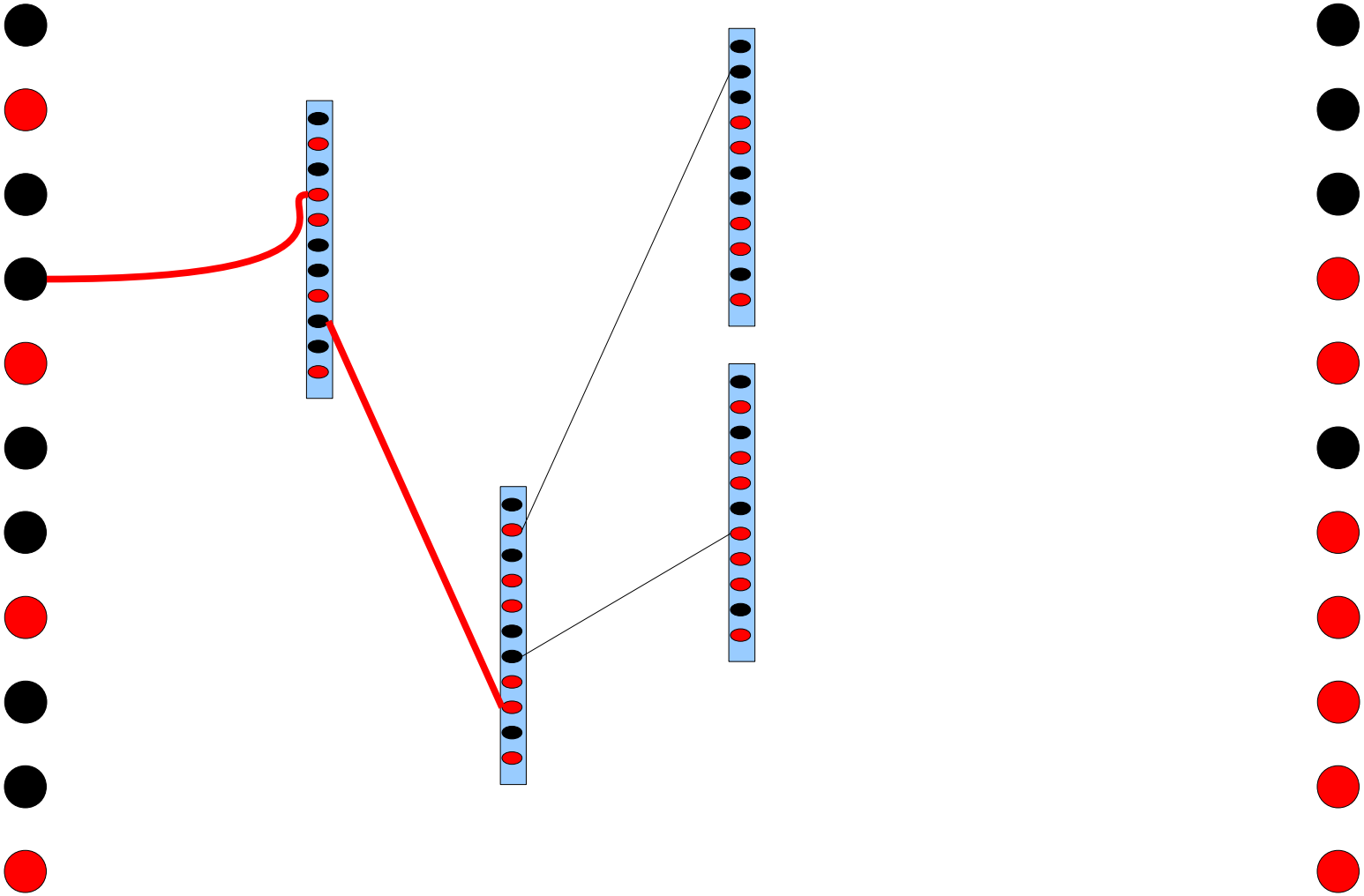
guiding solution



starting solution

PR example

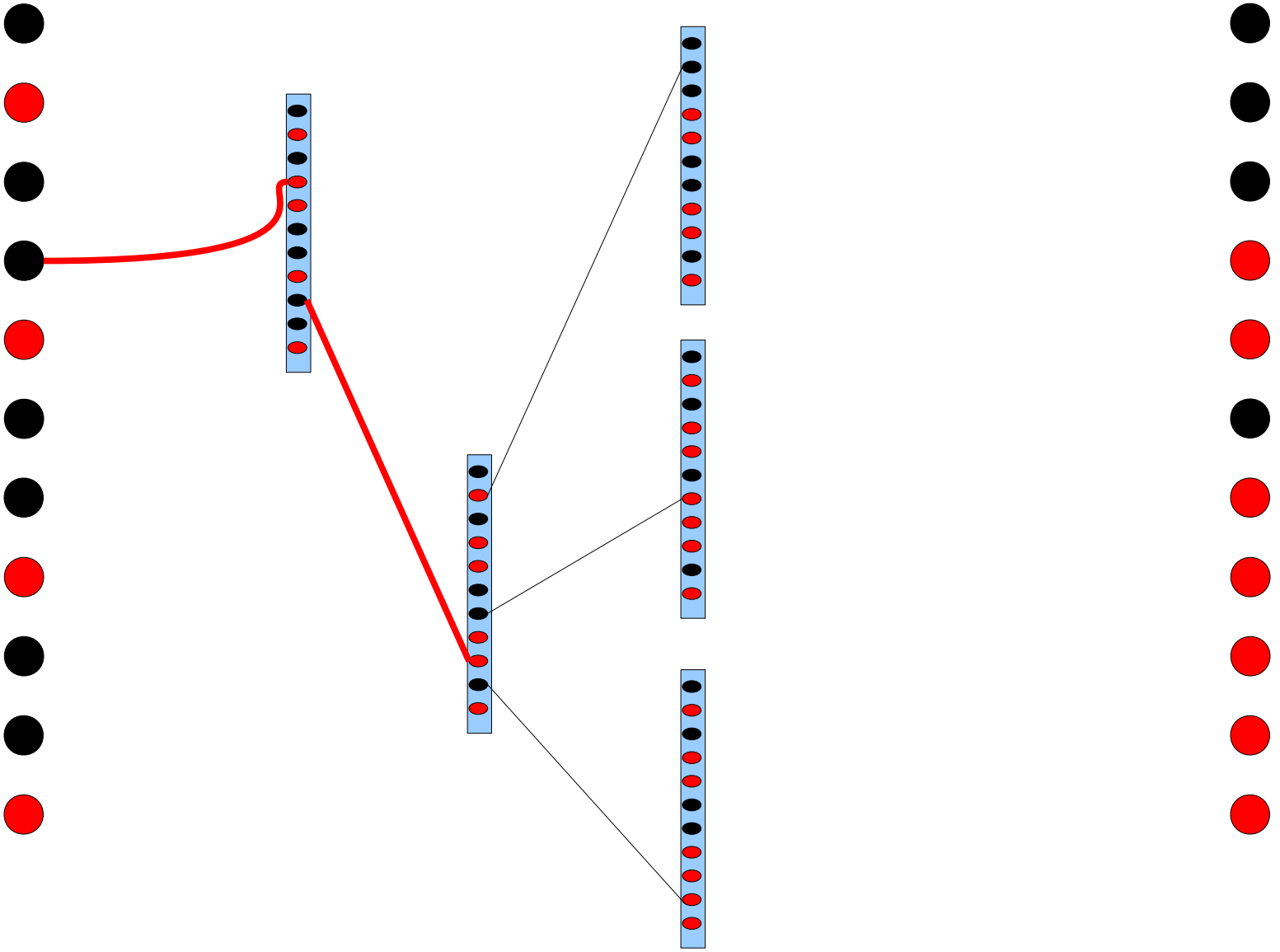
guiding solution



starting solution

PR example

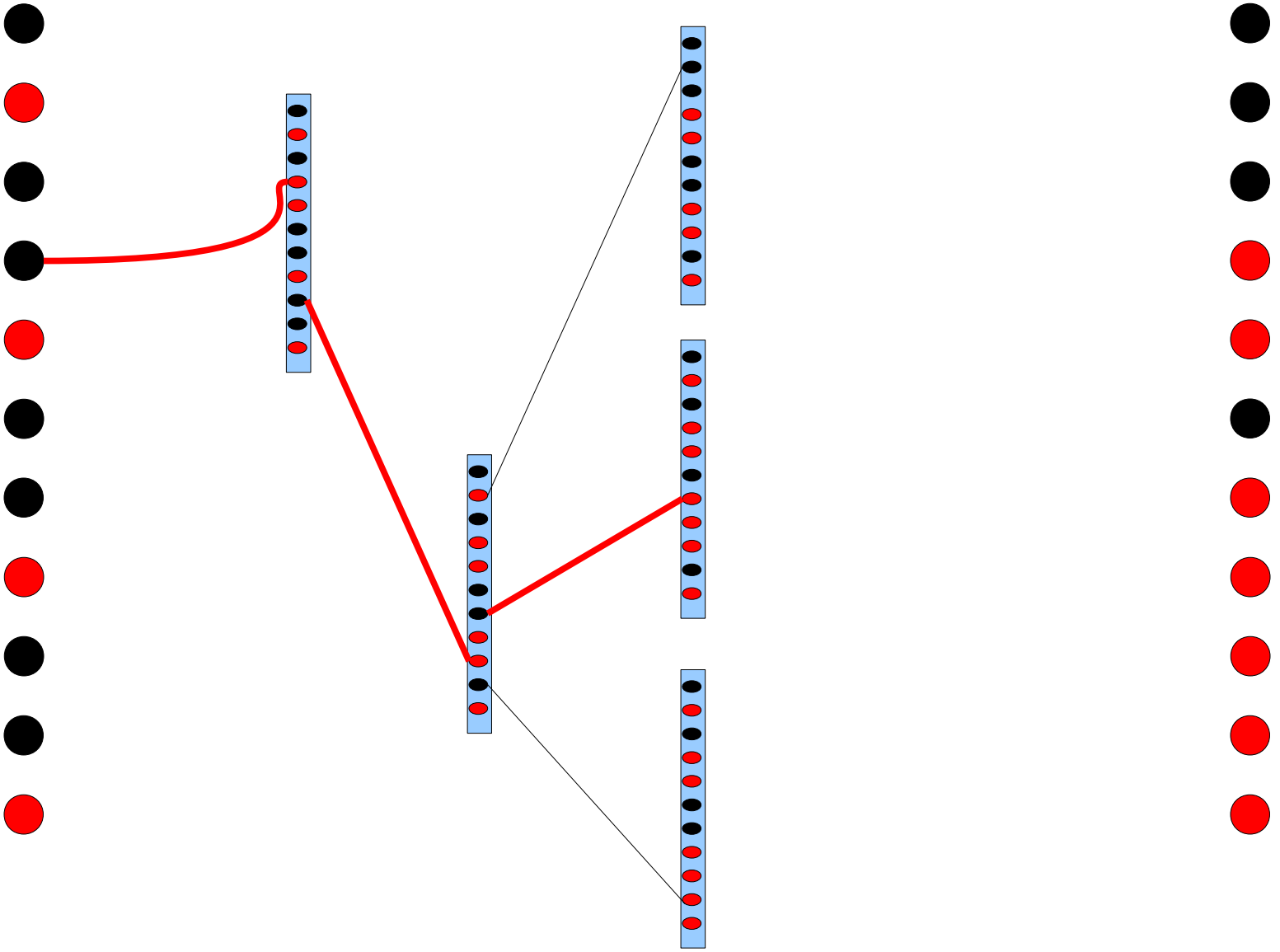
guiding solution



starting solution

PR example

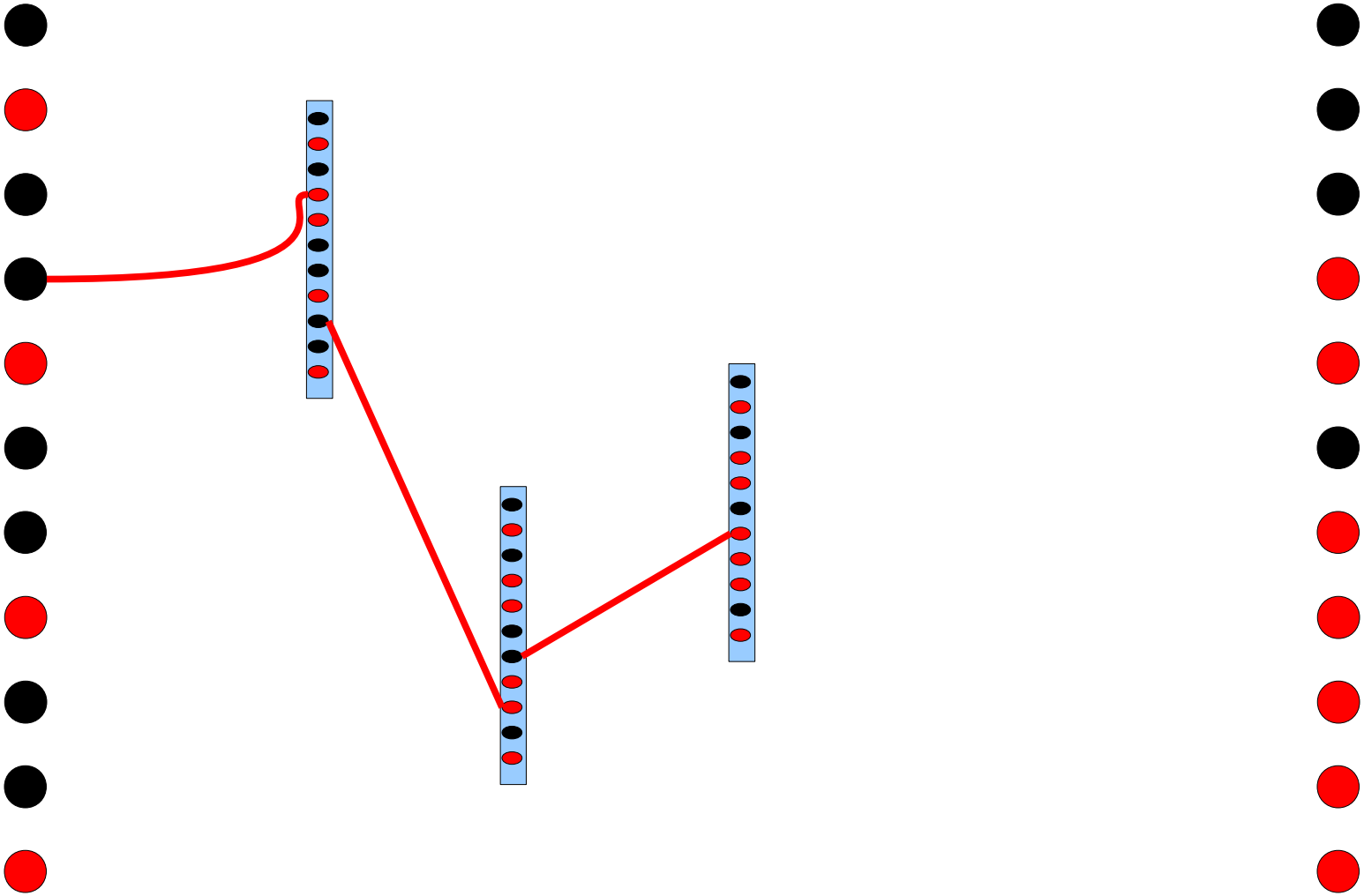
guiding solution



starting solution

PR example

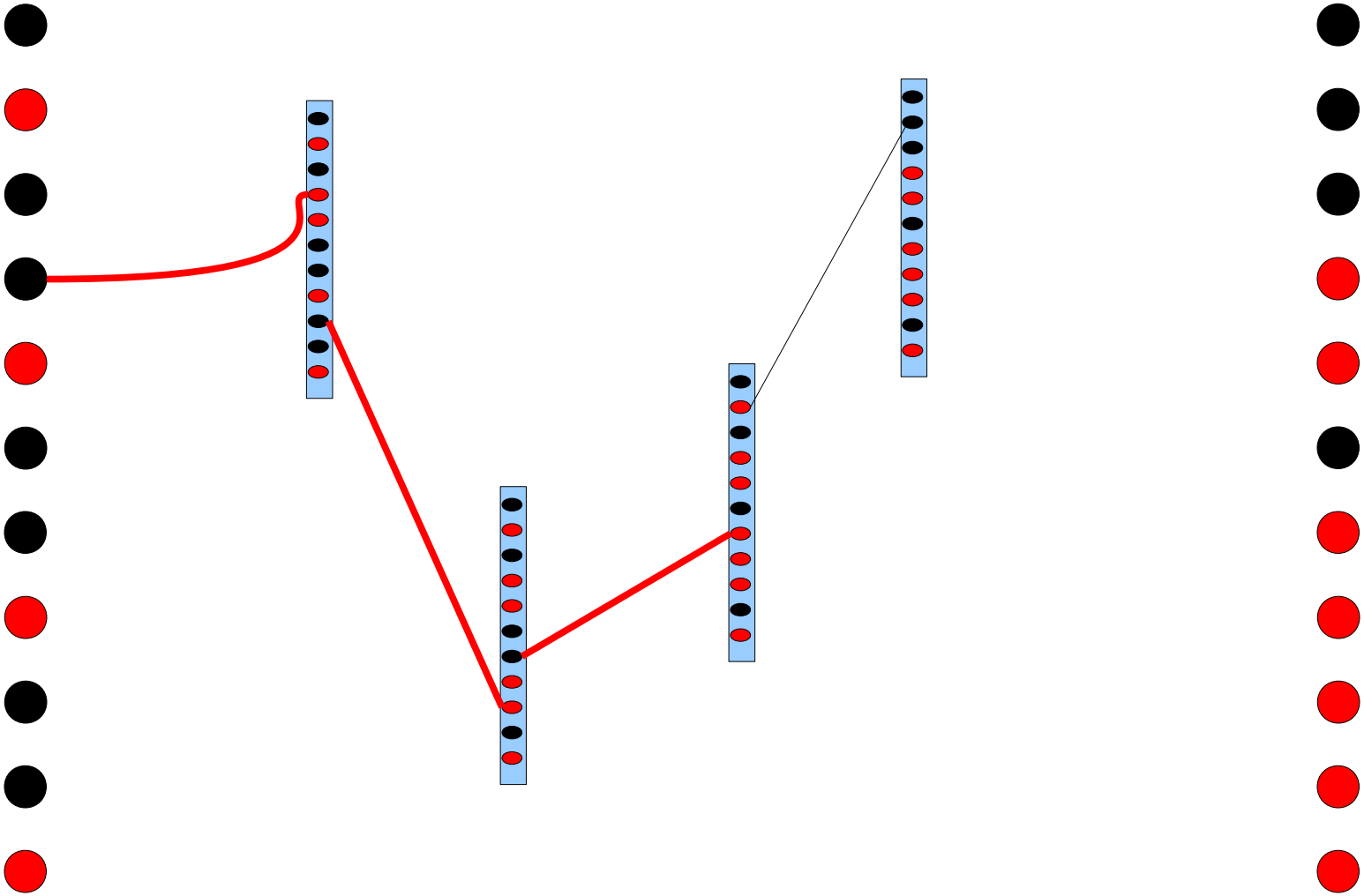
guiding solution



starting solution

PR example

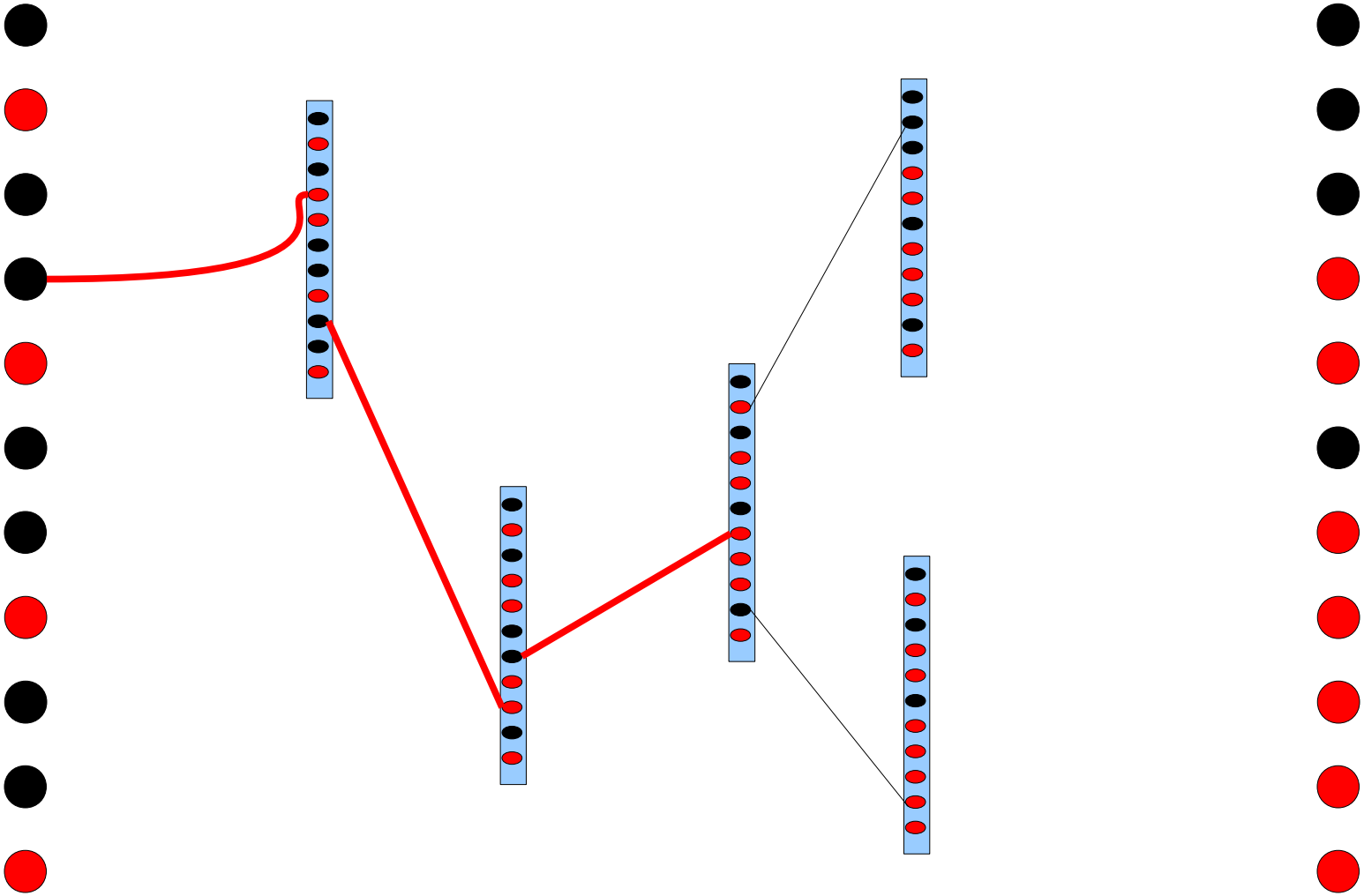
guiding solution



starting solution

PR example

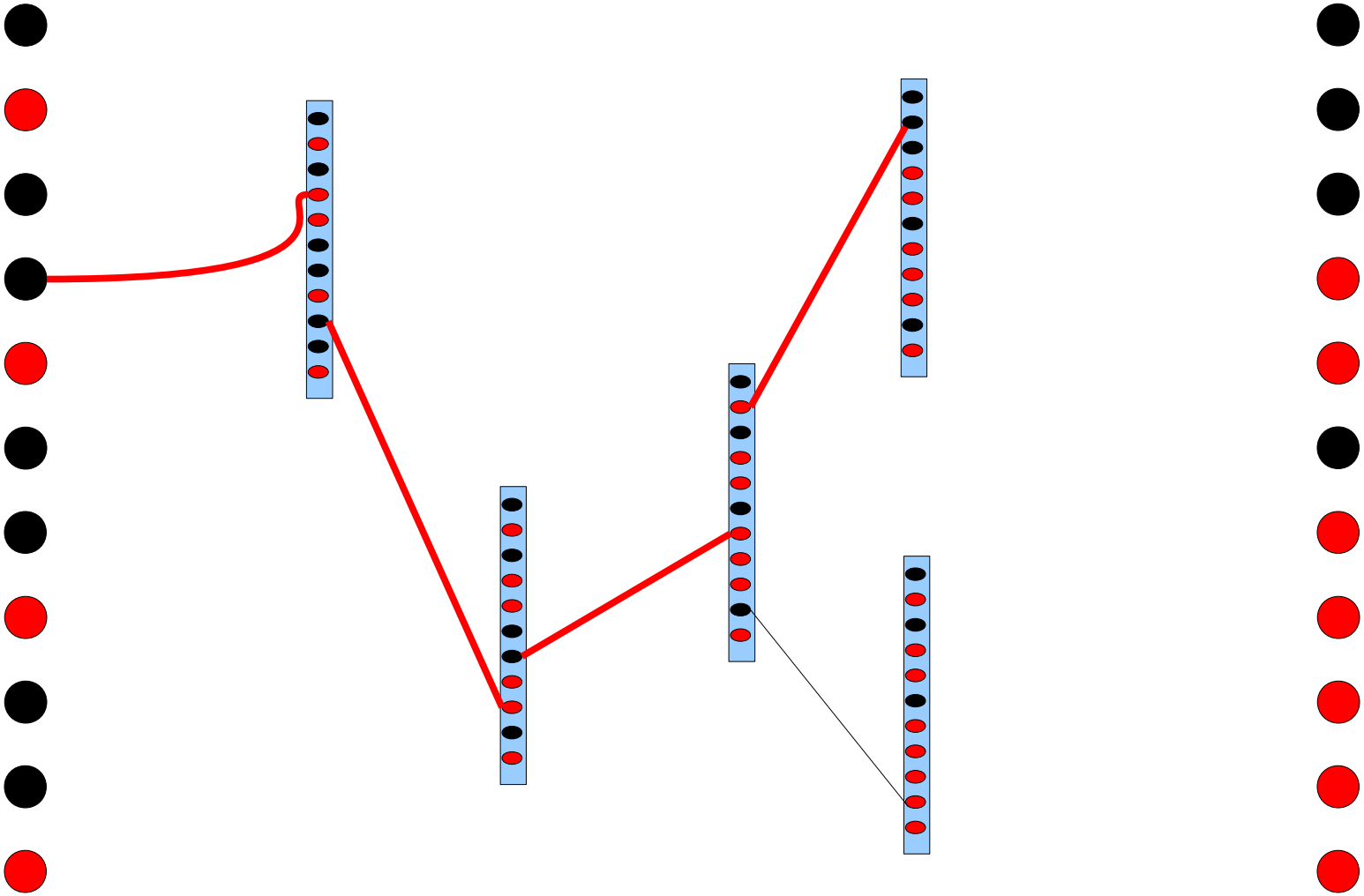
guiding solution



starting solution

PR example

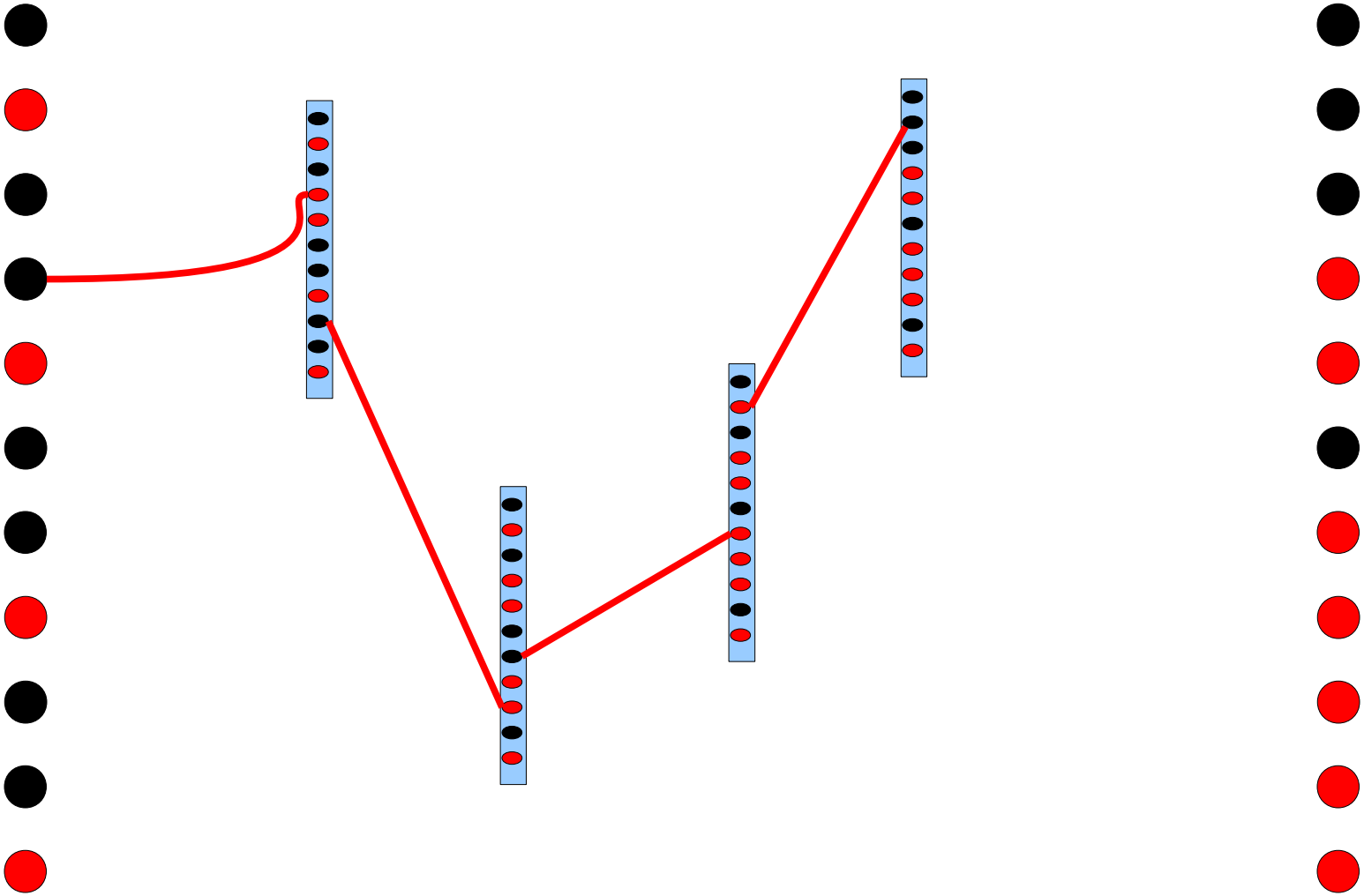
guiding solution



starting solution

PR example

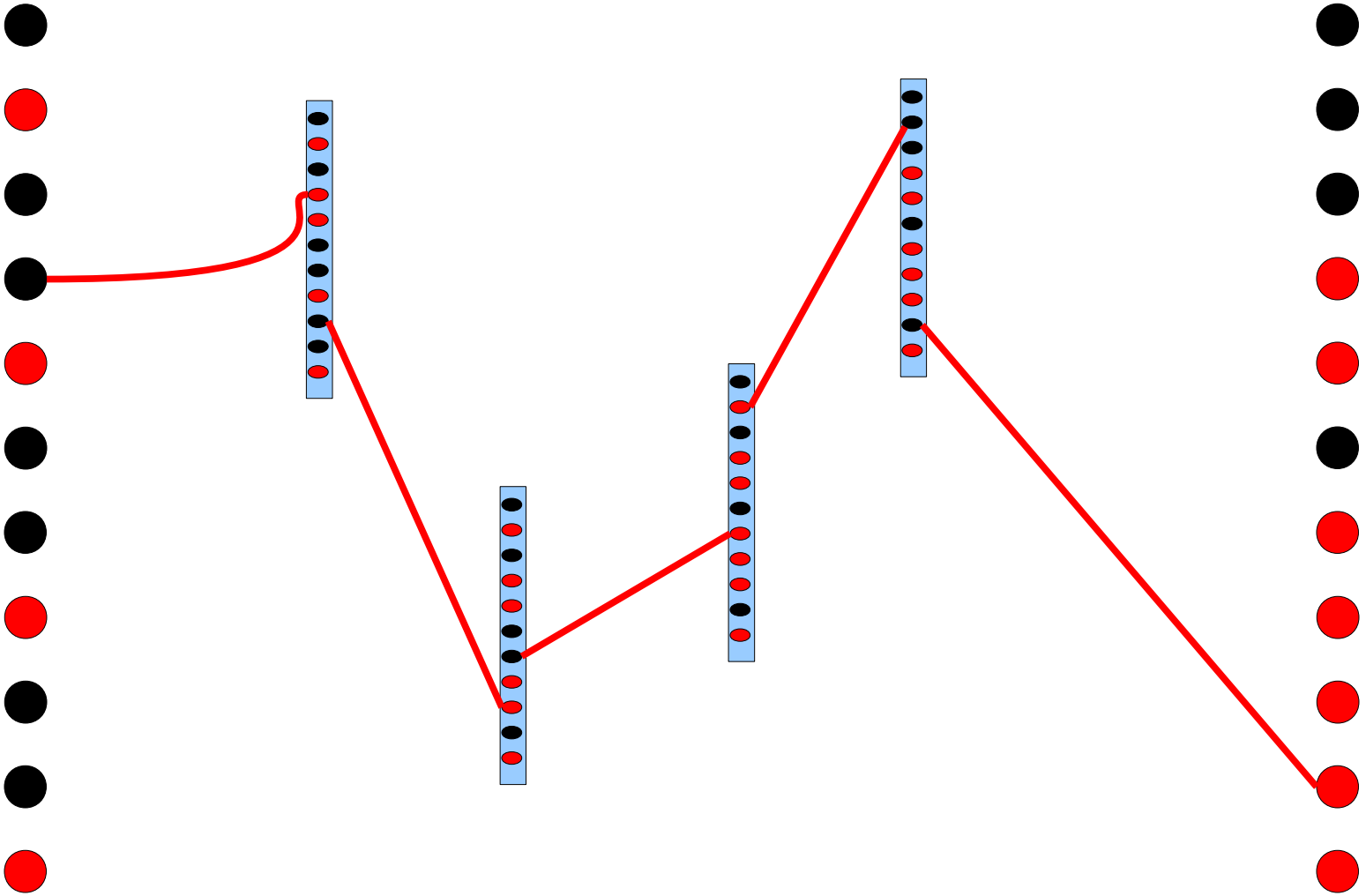
guiding solution

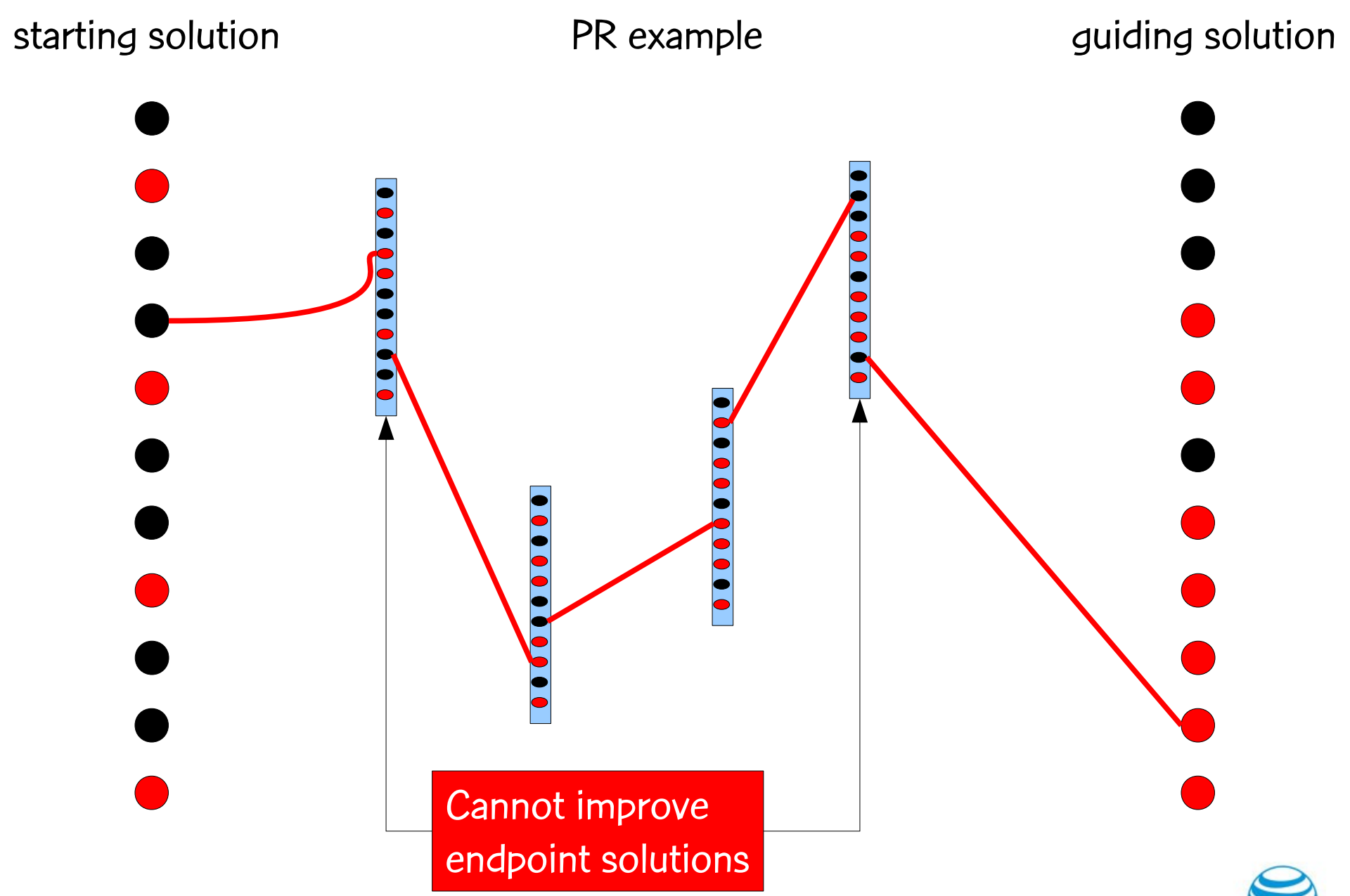


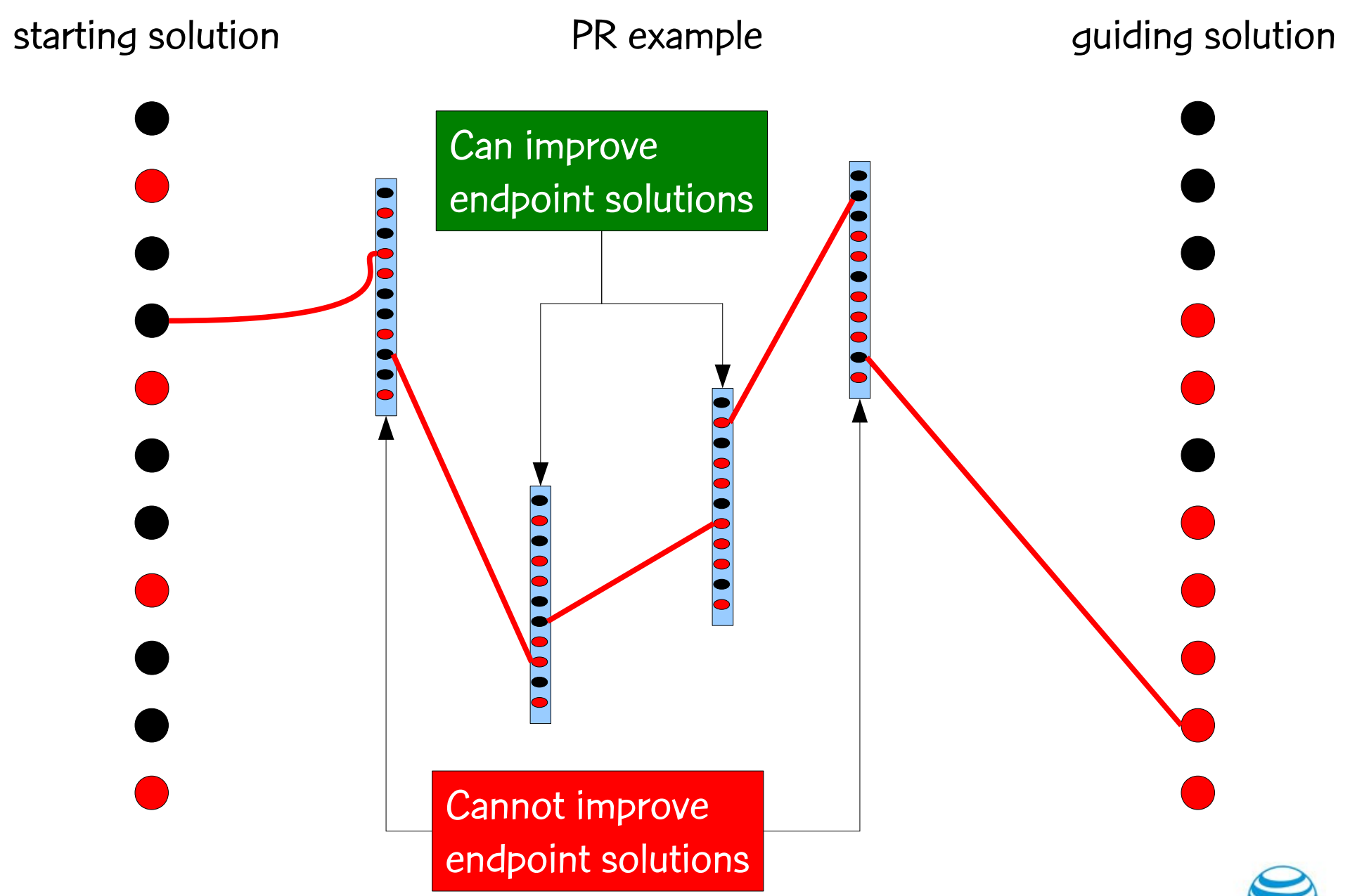
starting solution

PR example

guiding solution



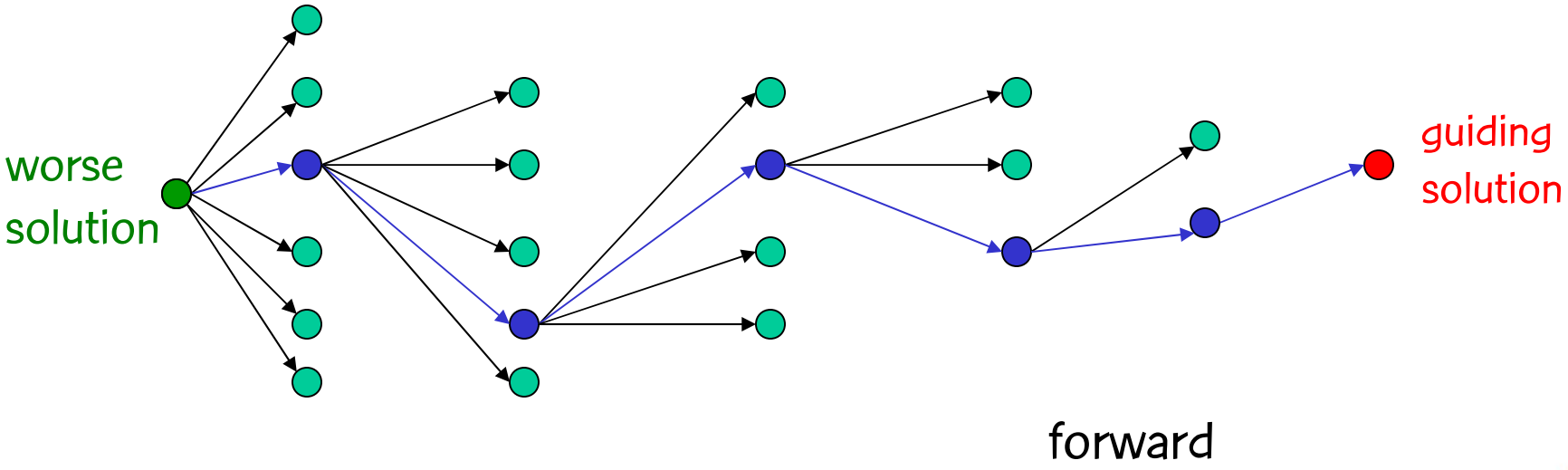




Forward path-relinking

Variants: trade-offs between computation time and solution quality

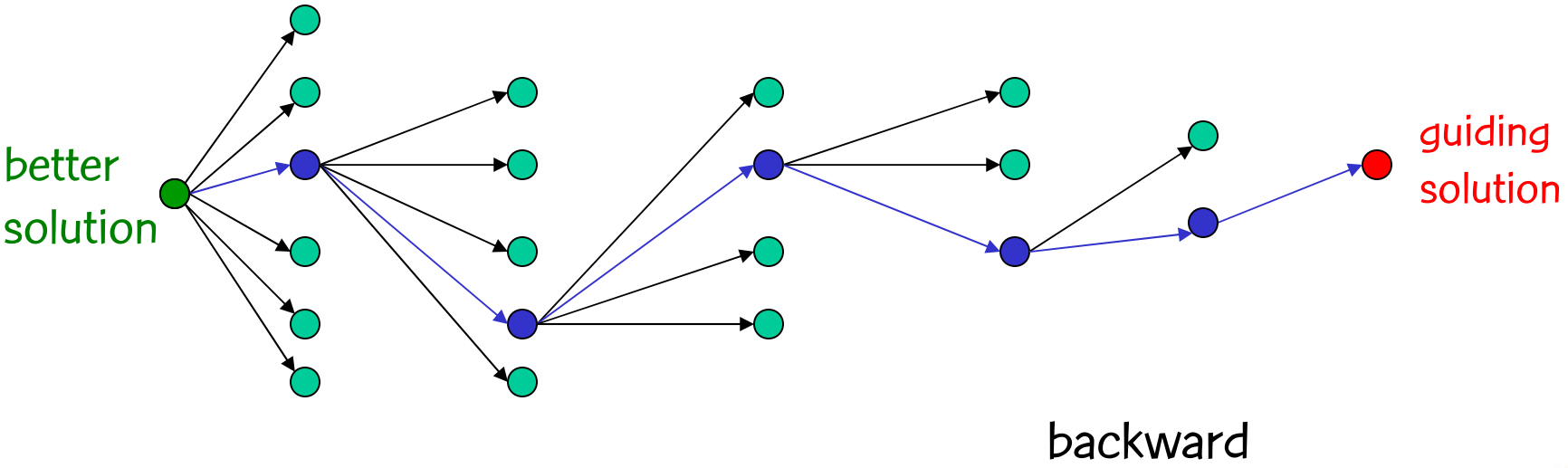
Forward PR adopts as initial solution the worse of the two input solutions and uses the better solution as the guide.



Backward path-relinking

Variants: trade-offs between computation time and solution quality

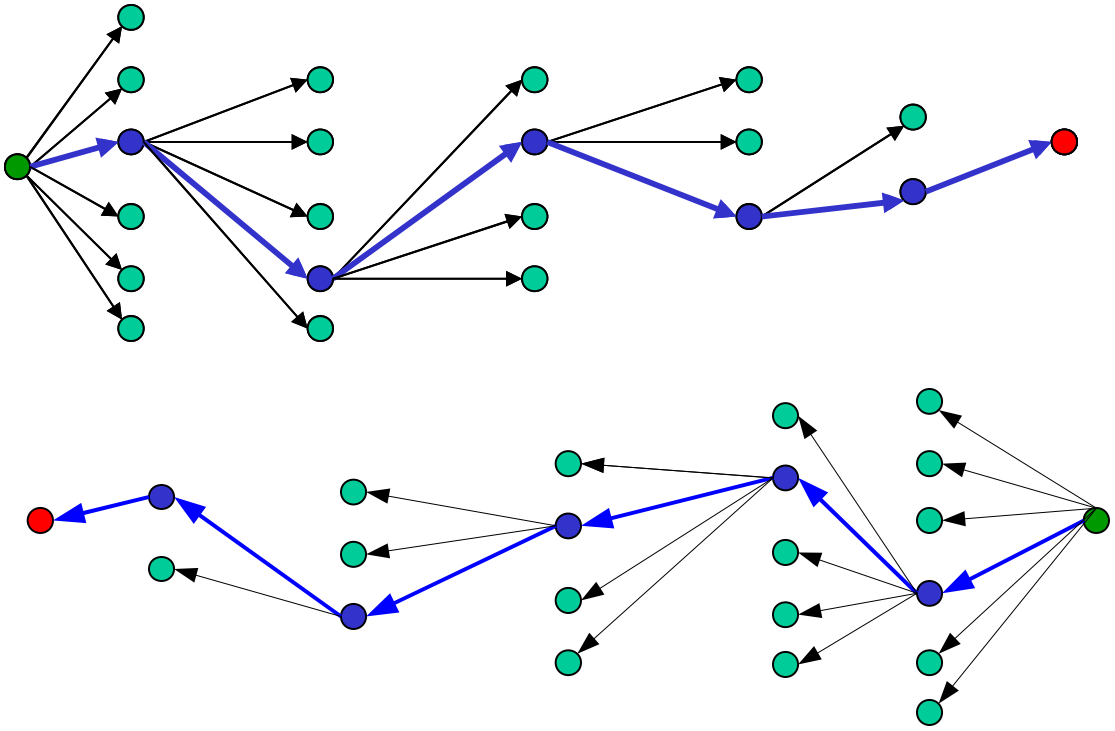
Backward PR usually does better: Better to start from the best of the two input solutions, neighborhood of the initial solution is explored more than of the guide!



Back and forth path-relinking

Variants: trade-offs between computation time and solution quality

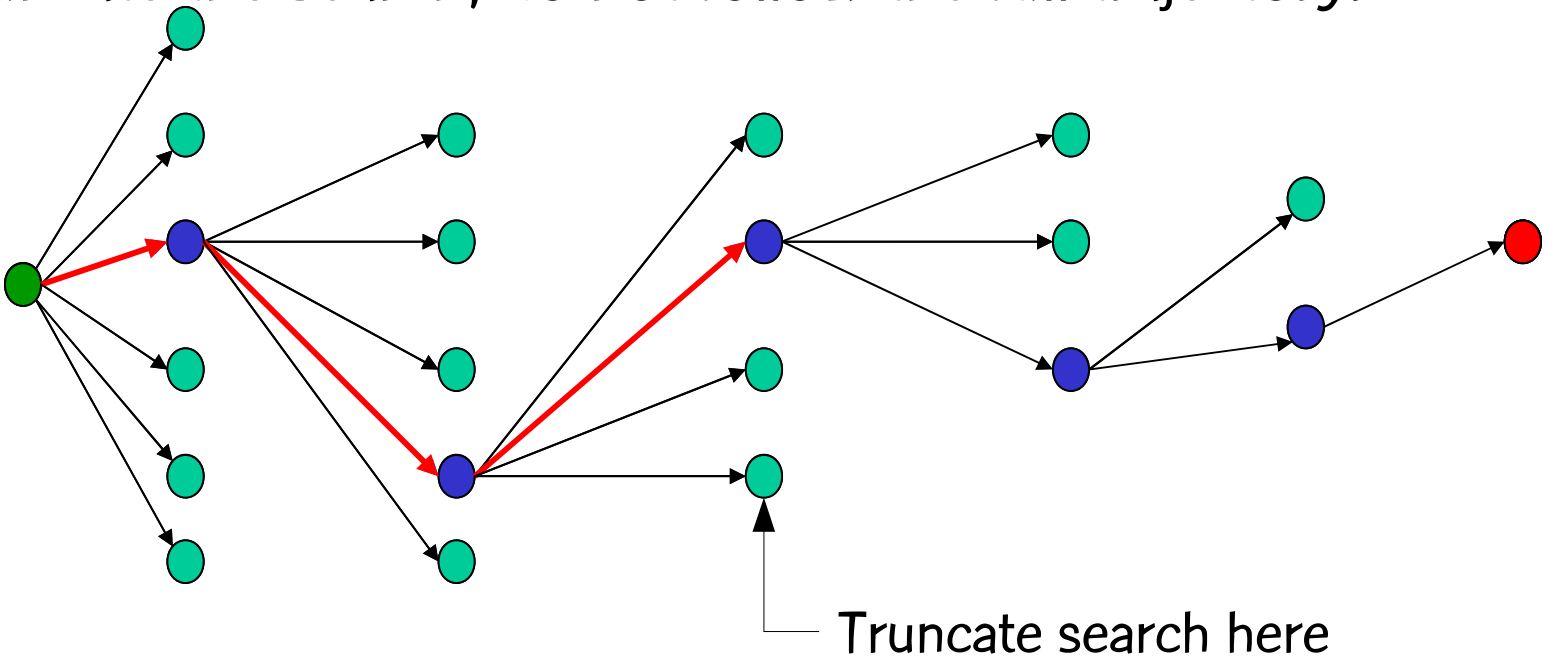
Explore both trajectories: twice as much time, often with only marginal improvements!



Truncated path-relinking

Variants: trade-offs between computation time and solution quality

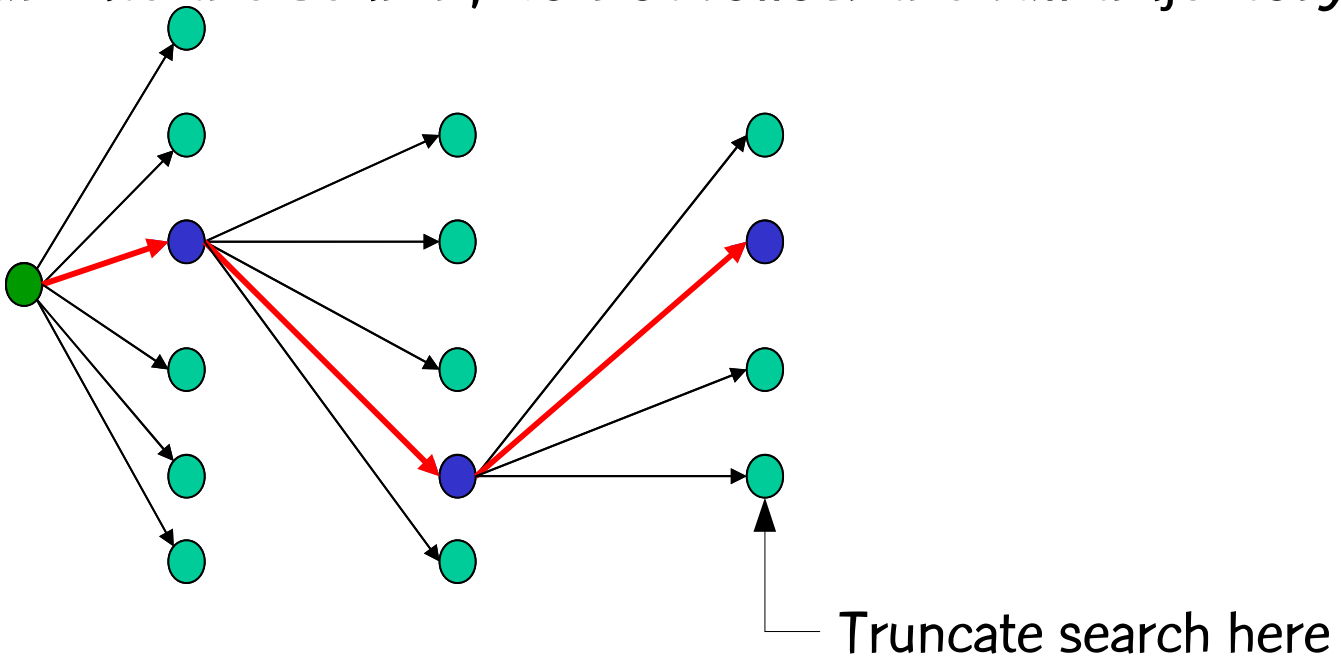
Truncate the search, do not follow the full trajectory.



Truncated path-relinking

Variants: trade-offs between computation time and solution quality

Truncate the search, do not follow the full trajectory.



Mixed path-relinking

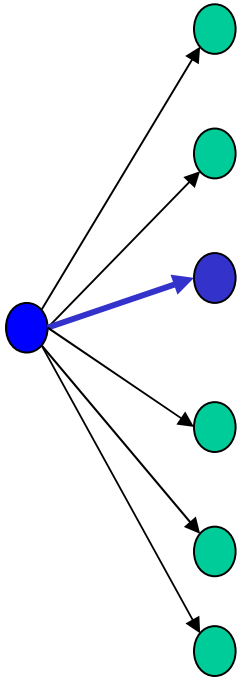
Variants: trade-offs between computation time and solution quality

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

Mixed path-relinking

Variants: trade-offs between computation time and solution quality

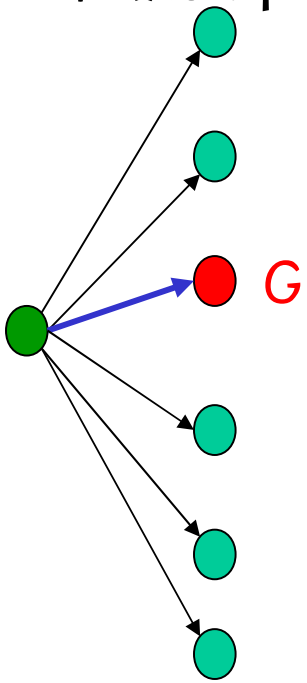
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

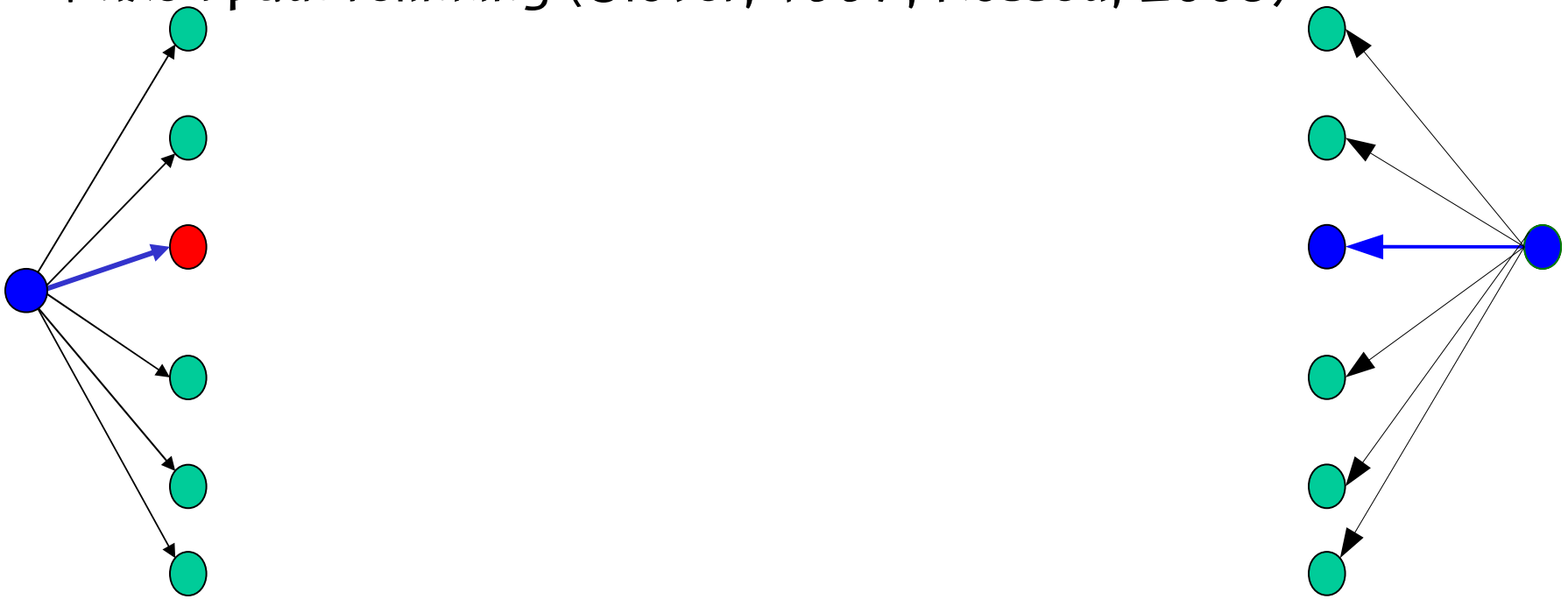
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

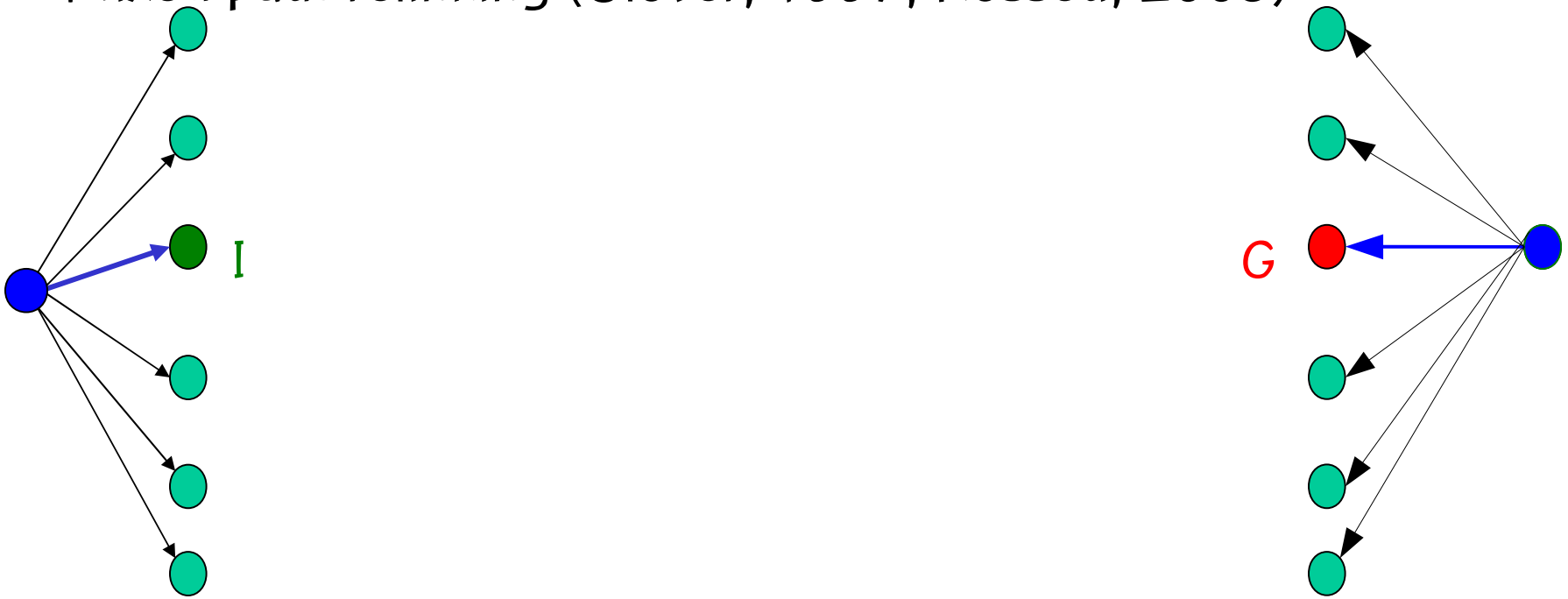
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

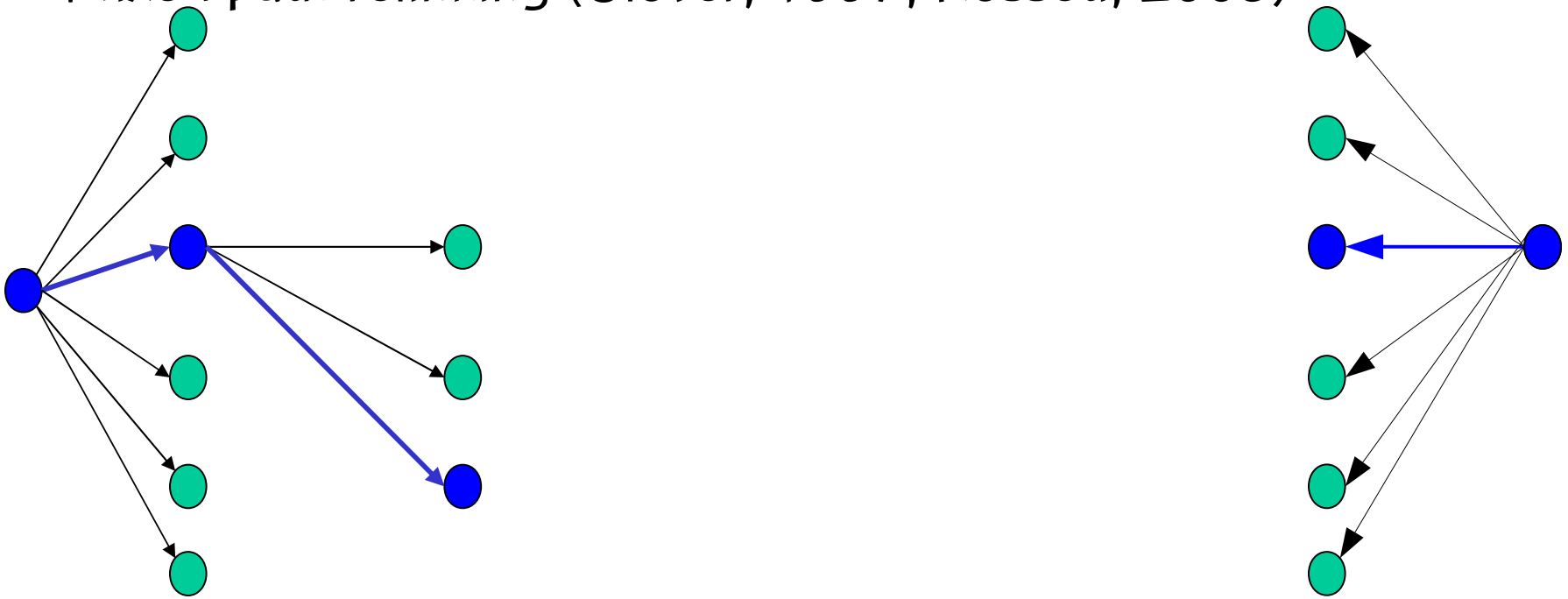
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

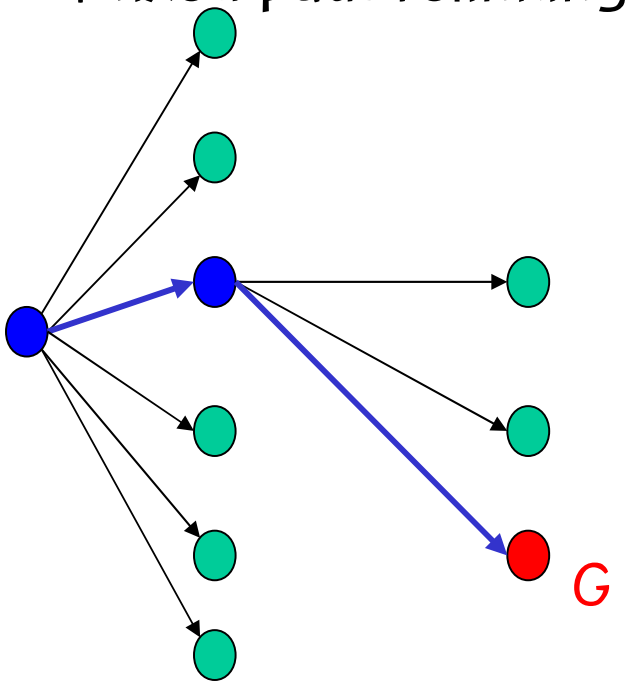
Mixed path-relinking (Glover, 1997; Rosseti, 2003)

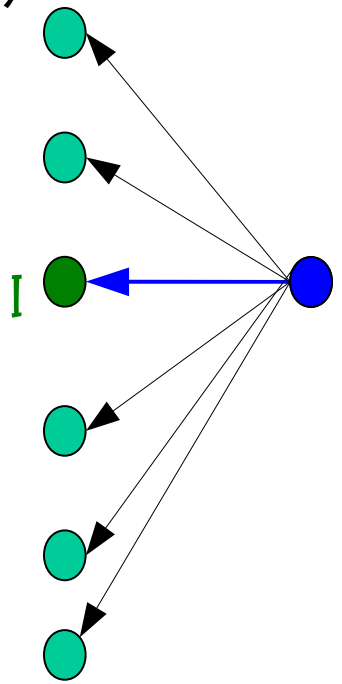


Mixed path-relinking

Variants: trade-offs between computation time and solution quality

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

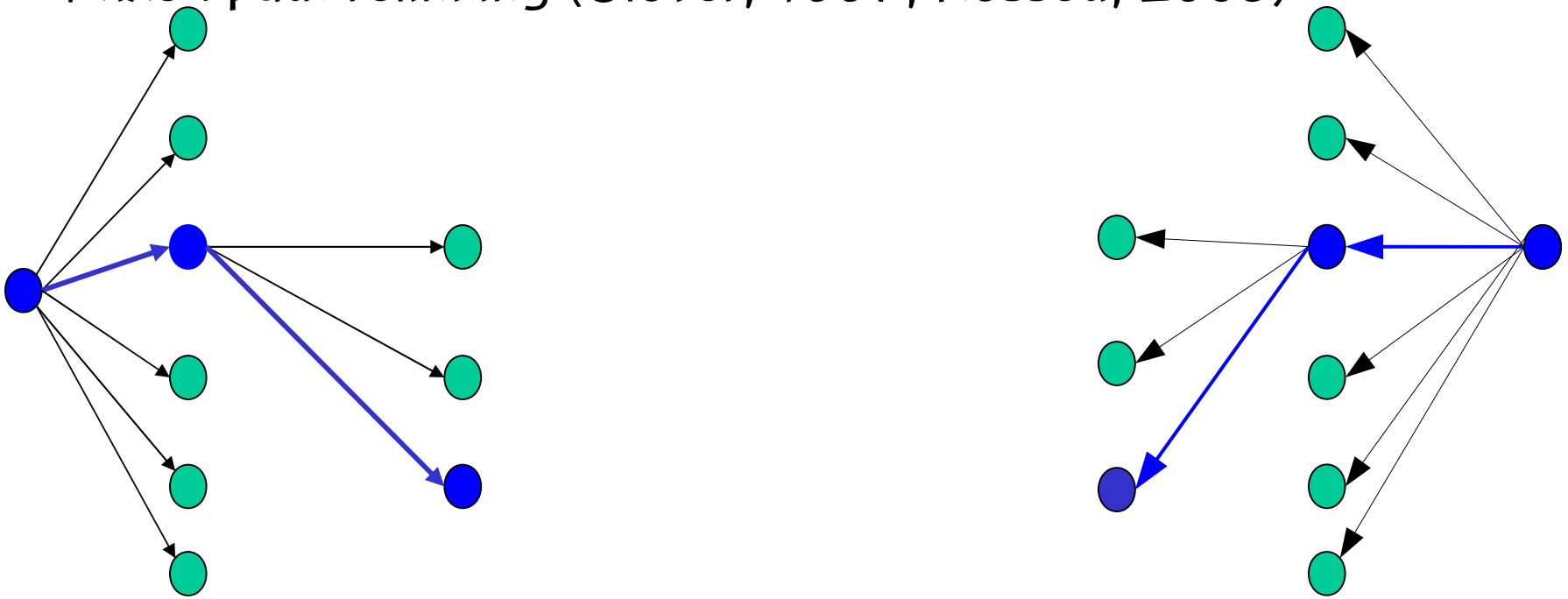




Mixed path-relinking

Variants: trade-offs between computation time and solution quality

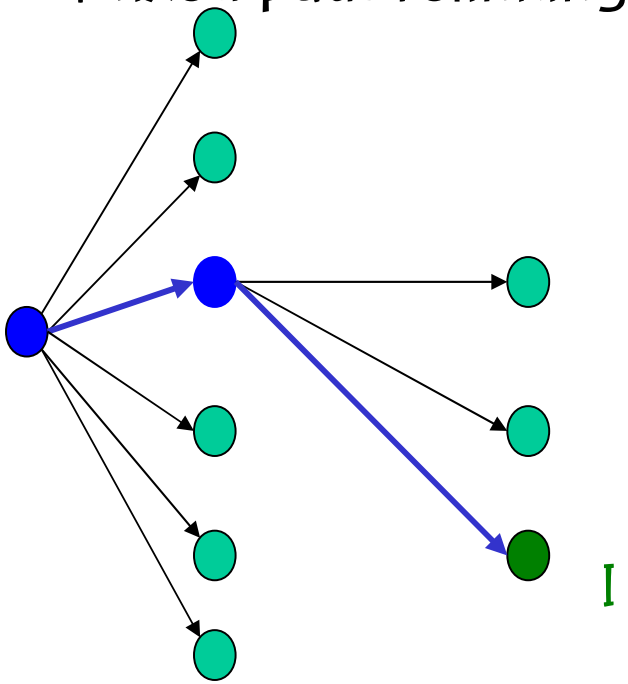
Mixed path-relinking (Glover, 1997; Rosseti, 2003)

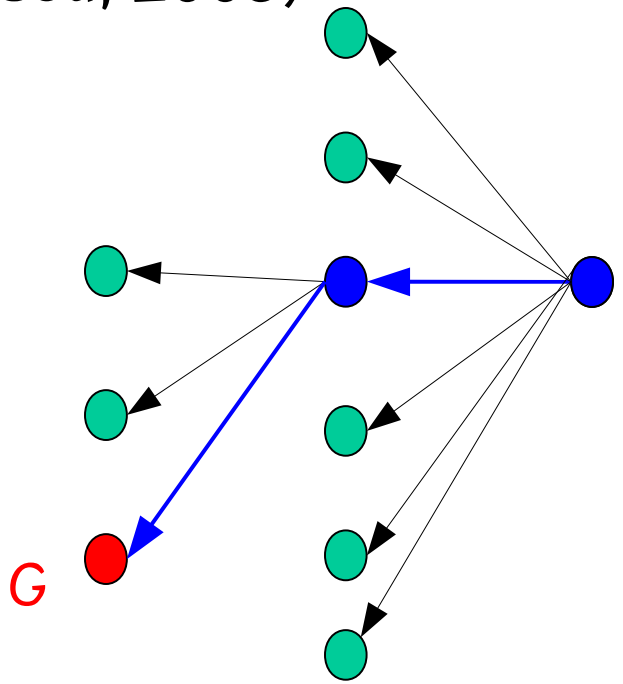


Mixed path-relinking

Variants: trade-offs between computation time and solution quality

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

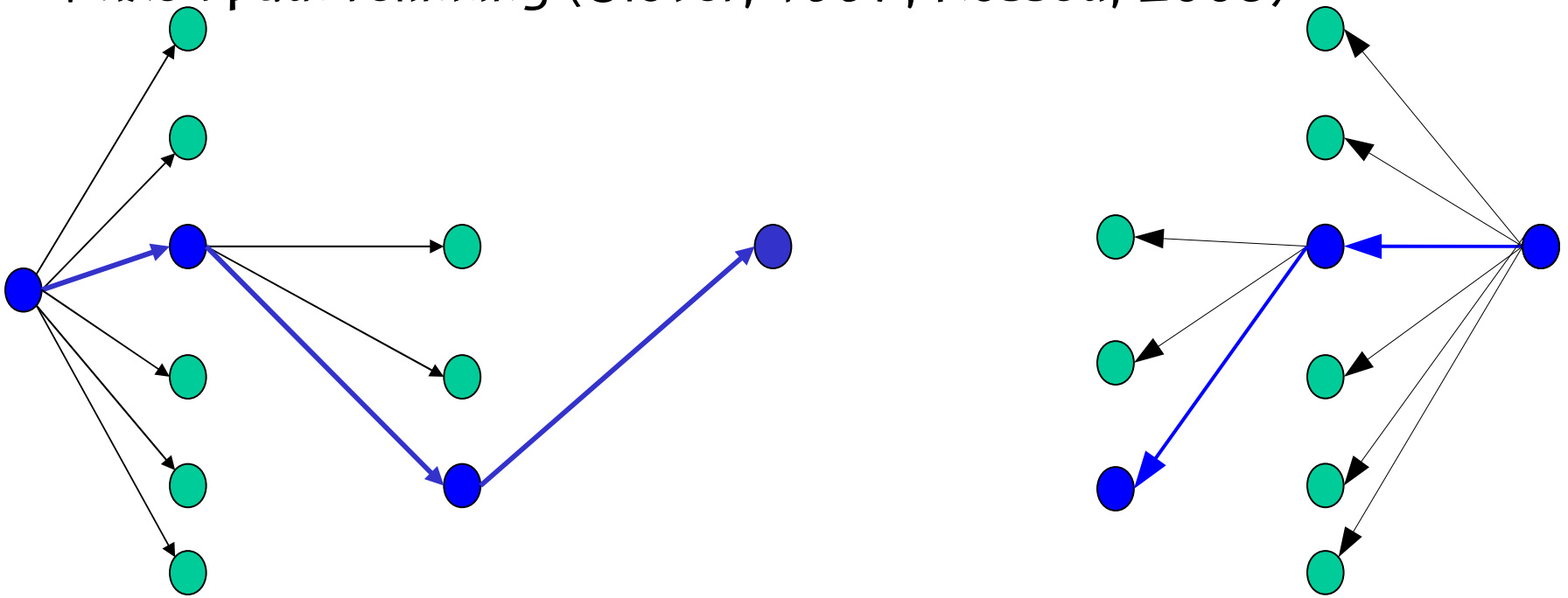




Mixed path-relinking

Variants: trade-offs between computation time and solution quality

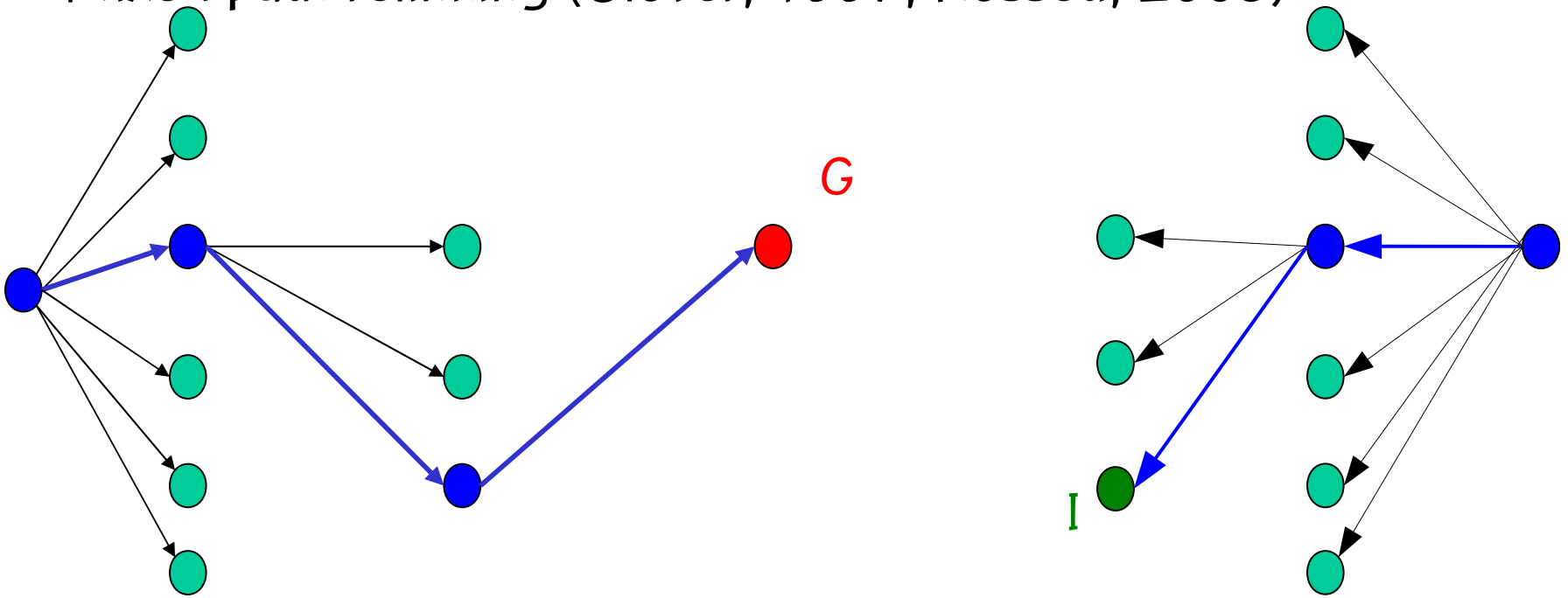
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

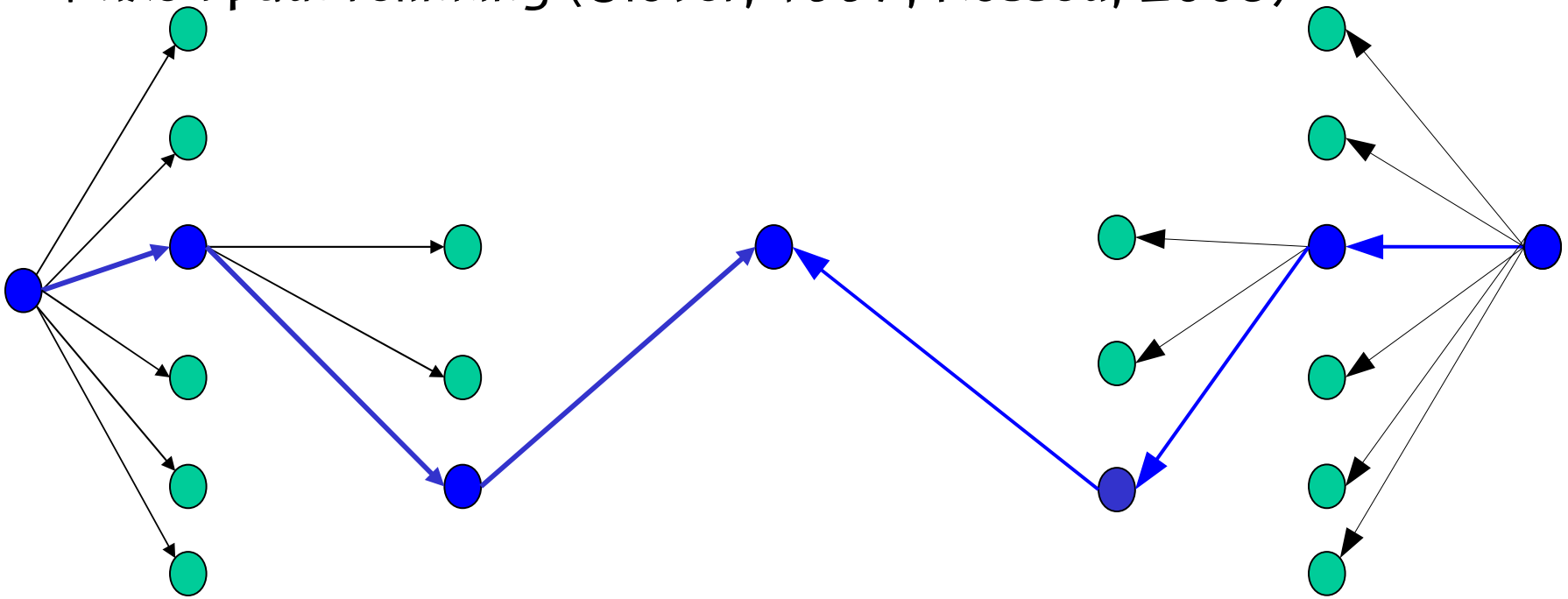
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

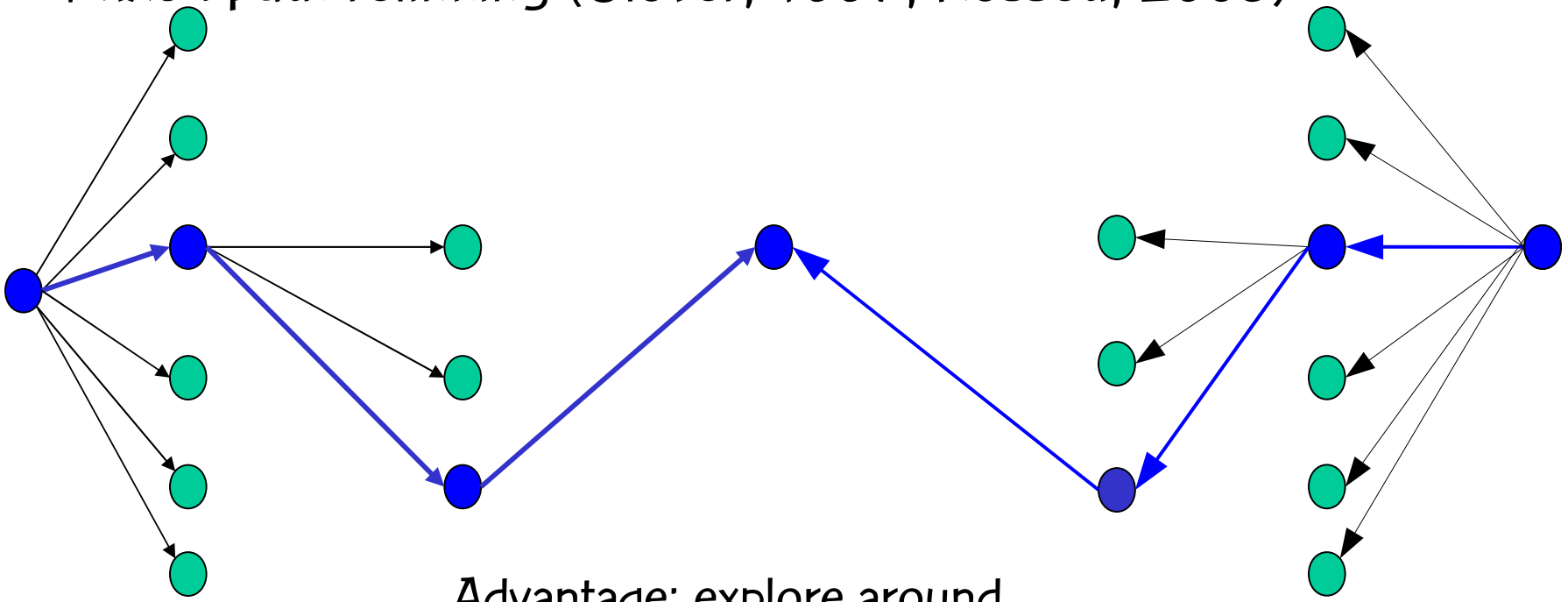
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

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

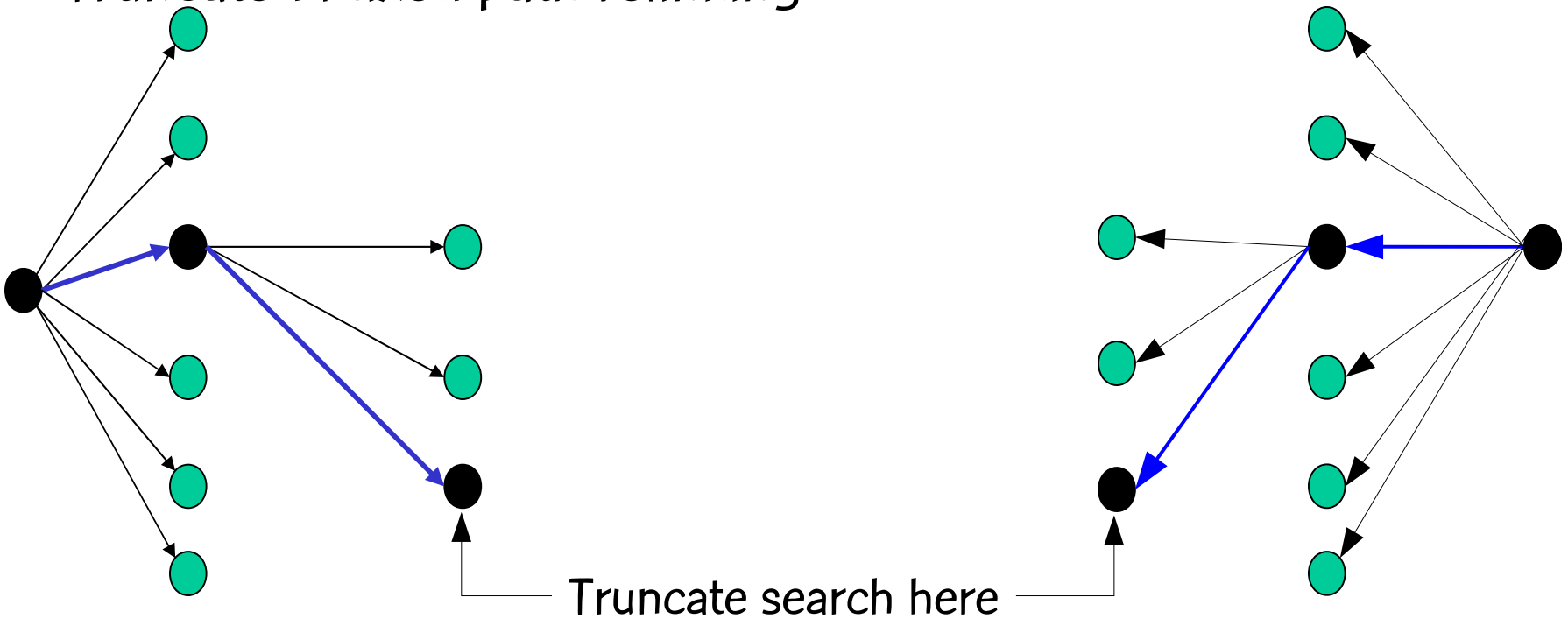


Advantage: explore around neighborhoods of both input solutions.

Truncated mixed path-relinking

Variants: trade-offs between computation time and solution quality

Truncated mixed path-relinking

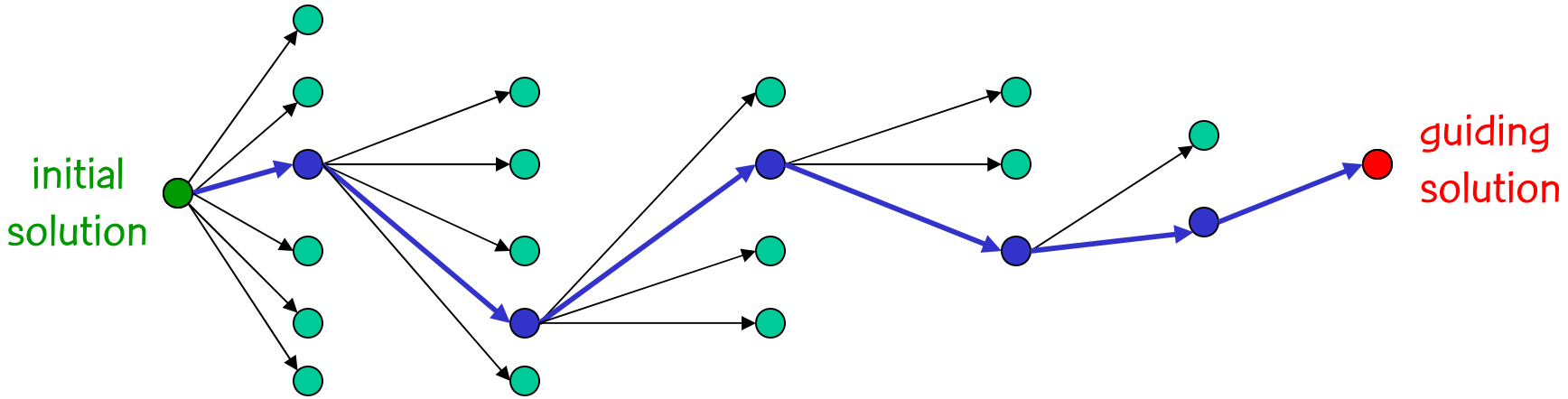


Greedy randomized adaptive path-relinking

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

Incorporates semi-greediness into PR.

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

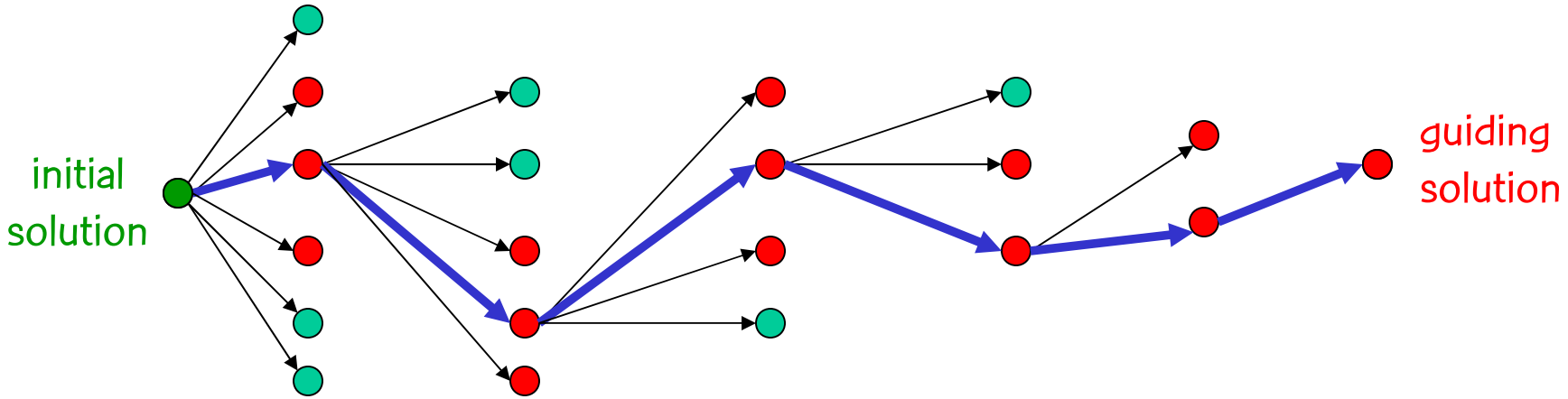


Greedy randomized adaptive path-relinking

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

Incorporates semi-greediness into PR.

graPR creates RCL with best moves: samples several paths

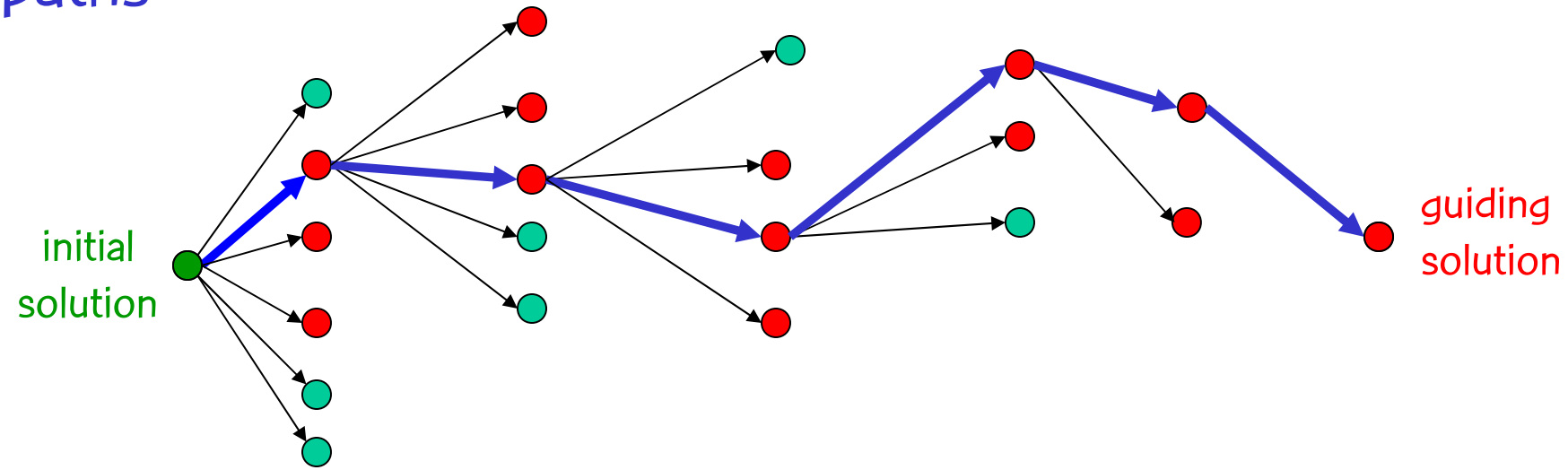


Greedy randomized adaptive path-relinking

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

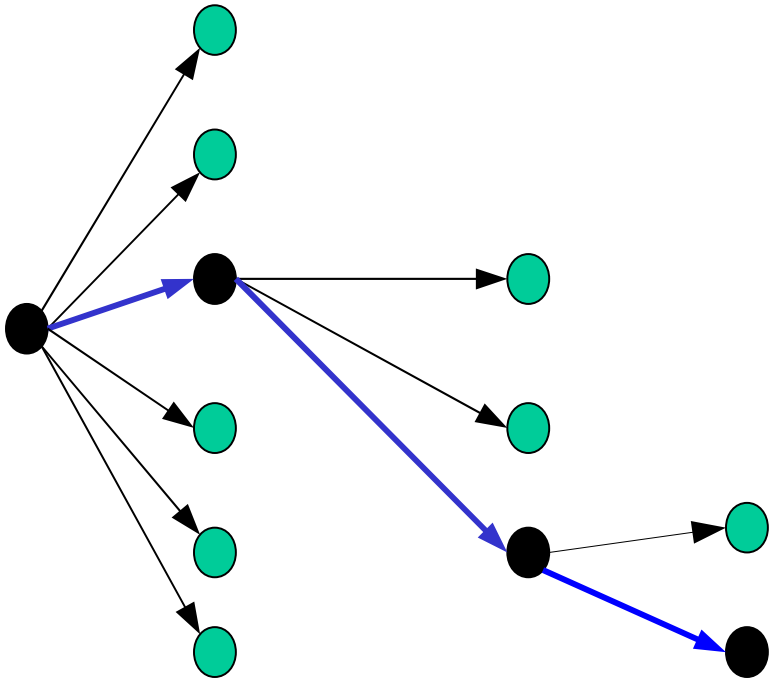
Incorporates semi-greediness into PR.

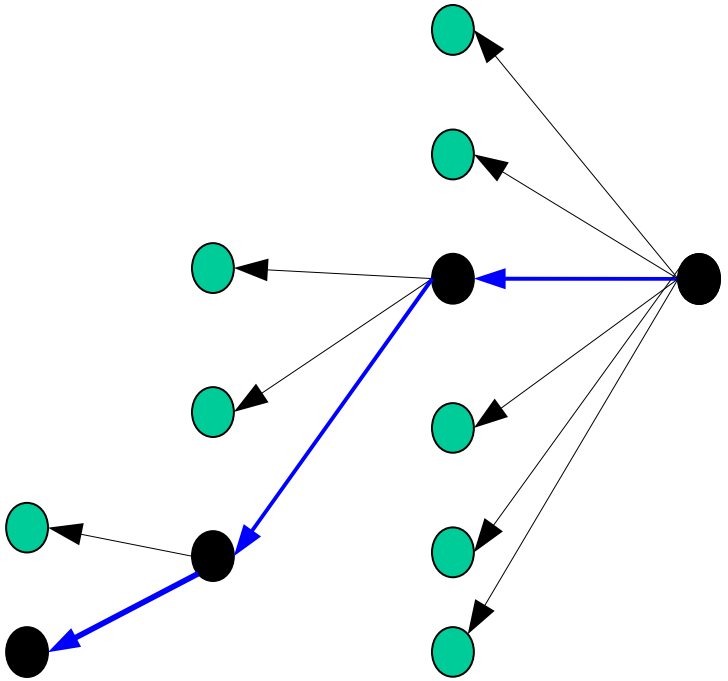
graPR creates RCL with best moves: samples several paths



Truncated mixed graPR

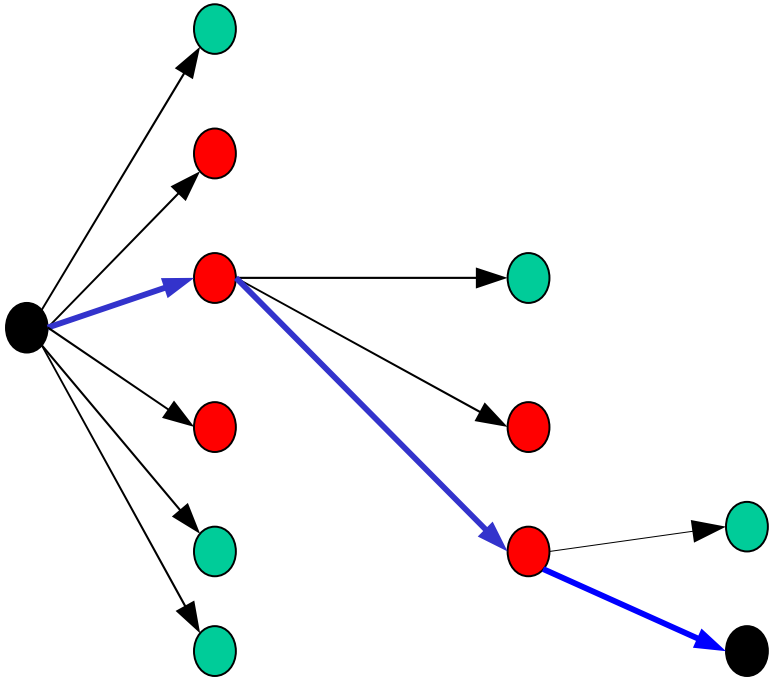
When applied to a given pair of solutions truncated mixed PR explores one of exponentially many path segments each time it is executed.

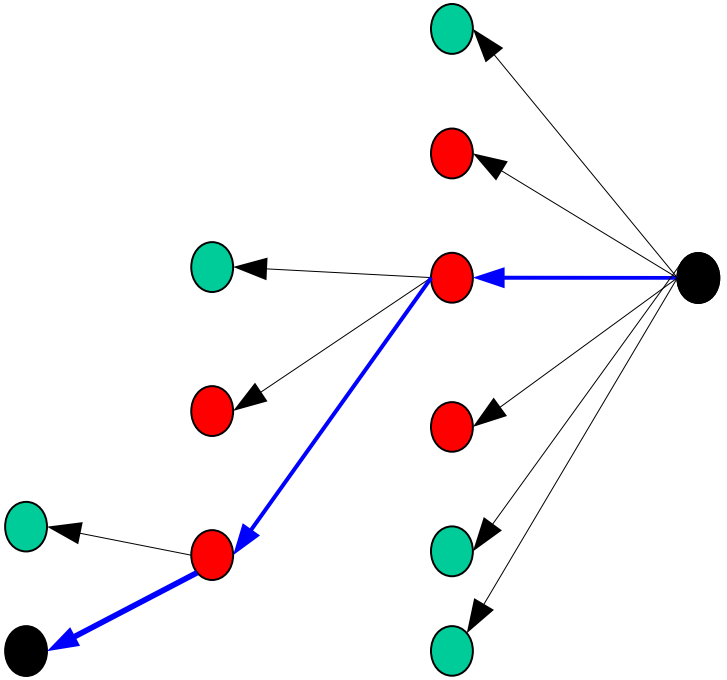




Truncated mixed graPR

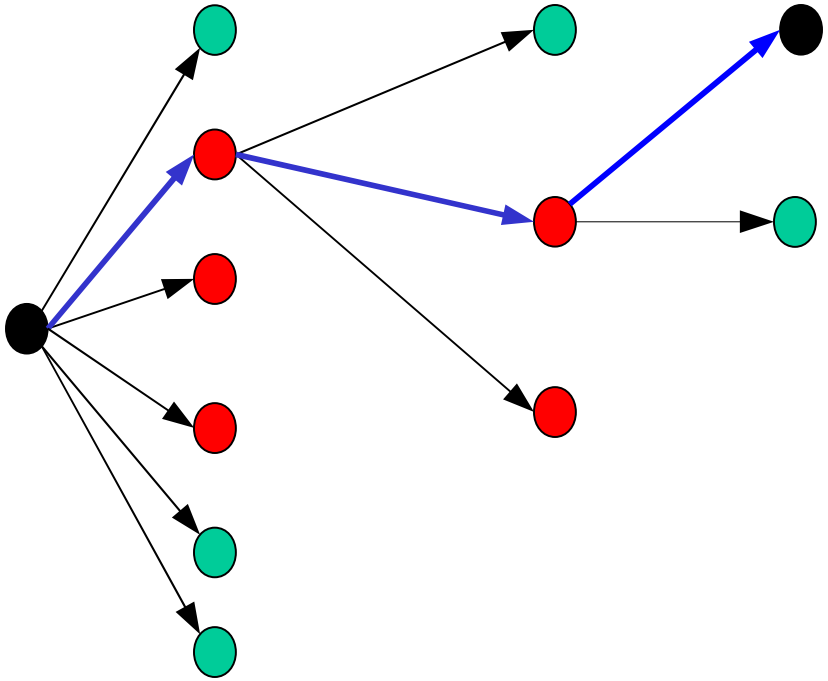
With high probability, truncated mixed graPR explores different path segments each time it is executed between the same pair of solutions.

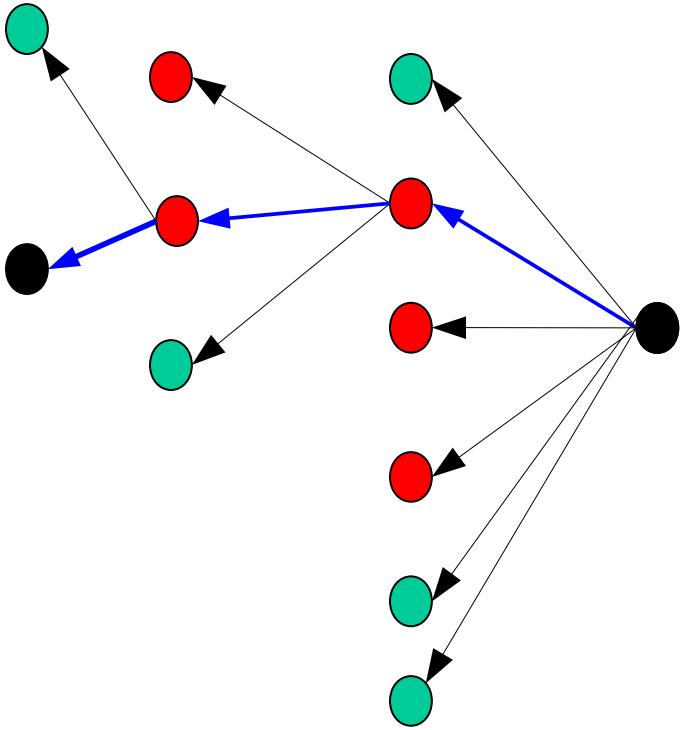




Truncated mixed graPR

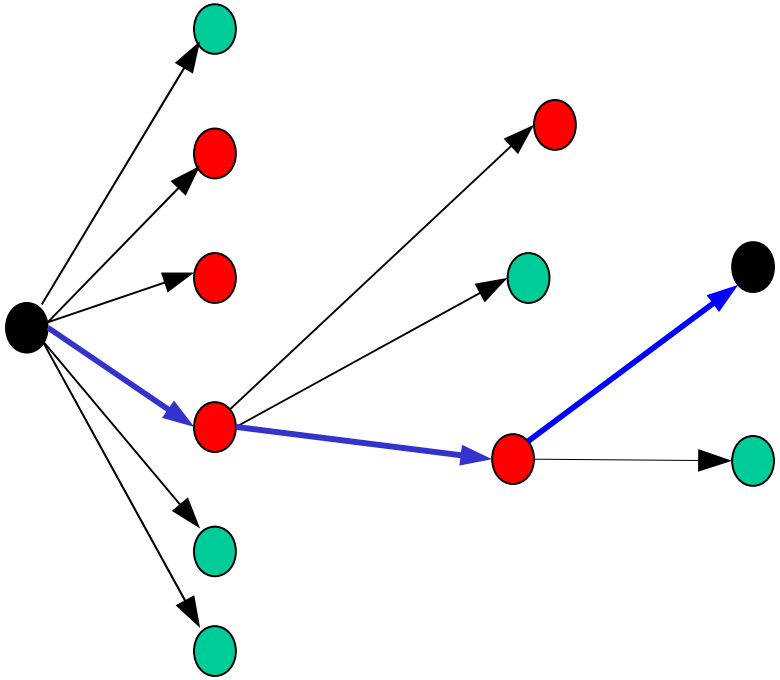
With high probability, truncated mixed graPR explores different path segments each time it is executed between the same pair of solutions.

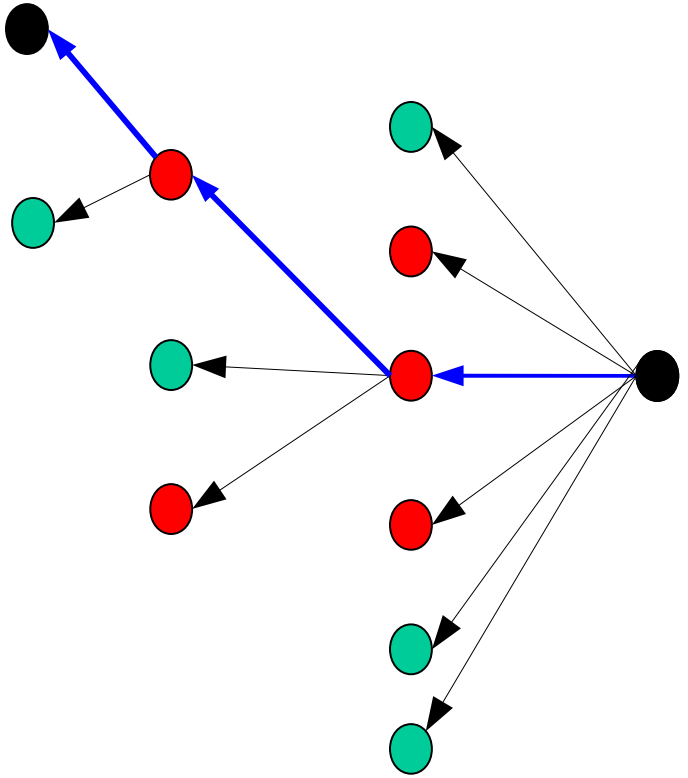




Truncated mixed graPR

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.

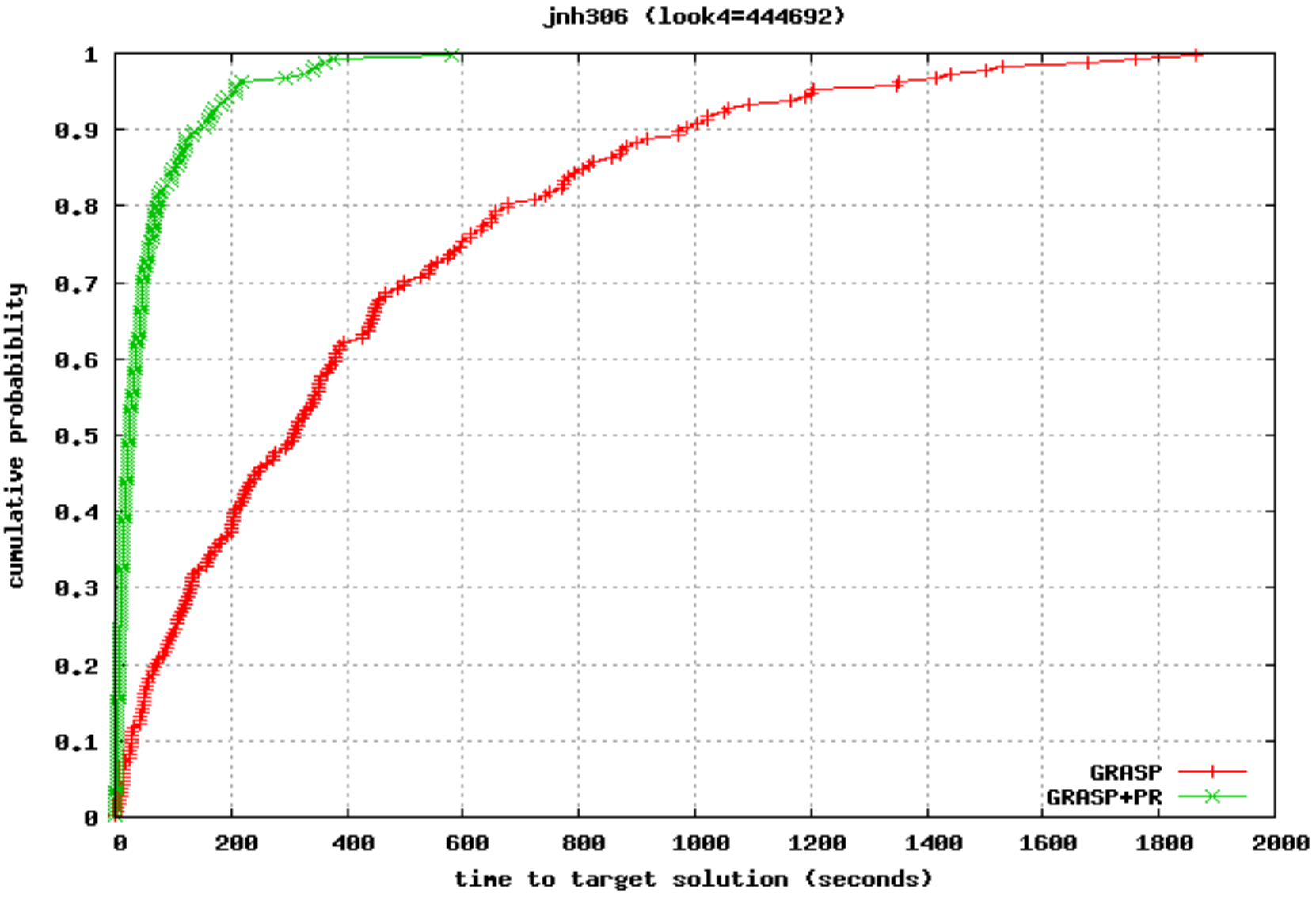
GRASP with path-relinking

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

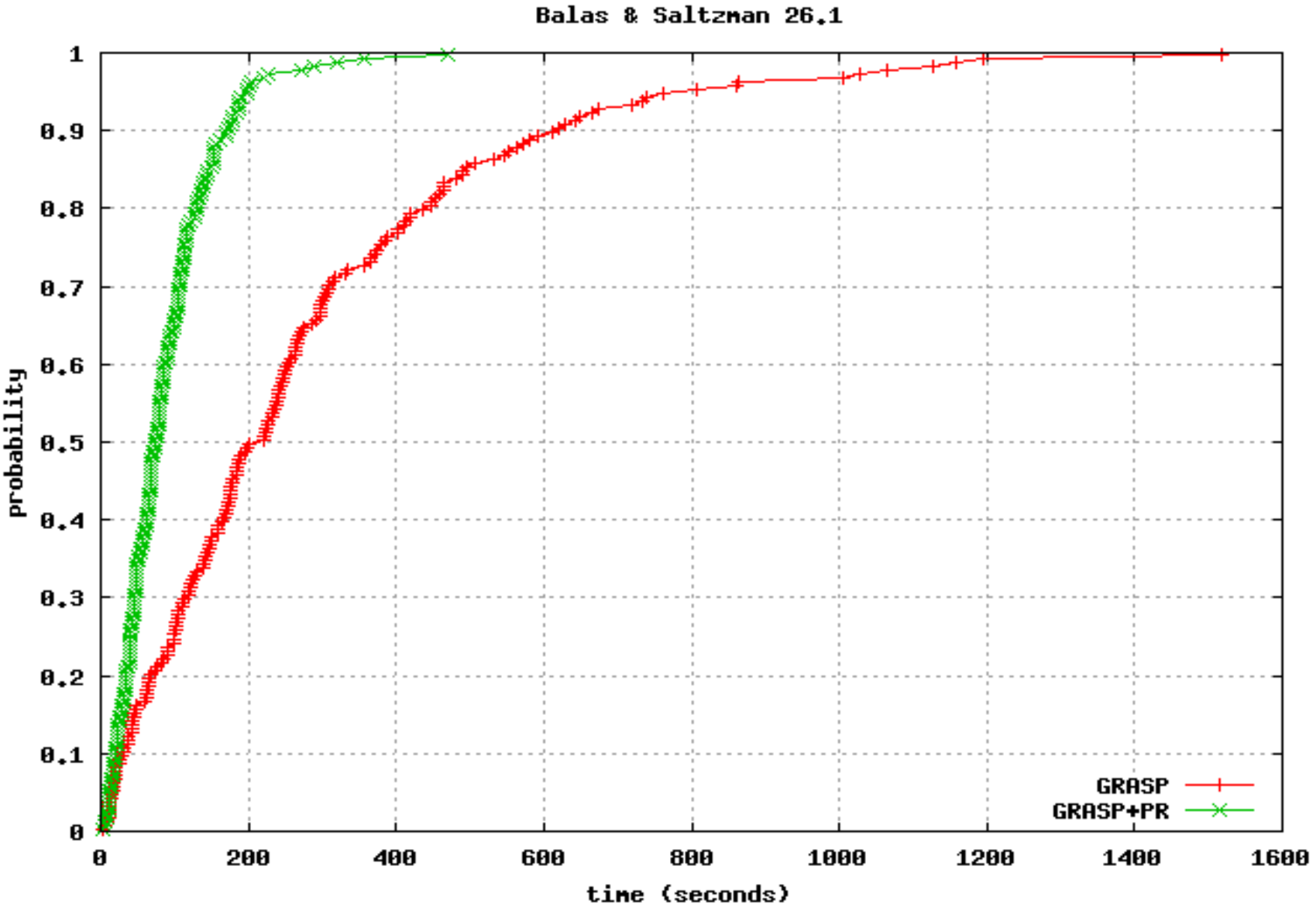
Survey by Resende & Ribeiro in MIC 2003 book of Ibaraki, Nonobe, and Yagiura (2005).

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

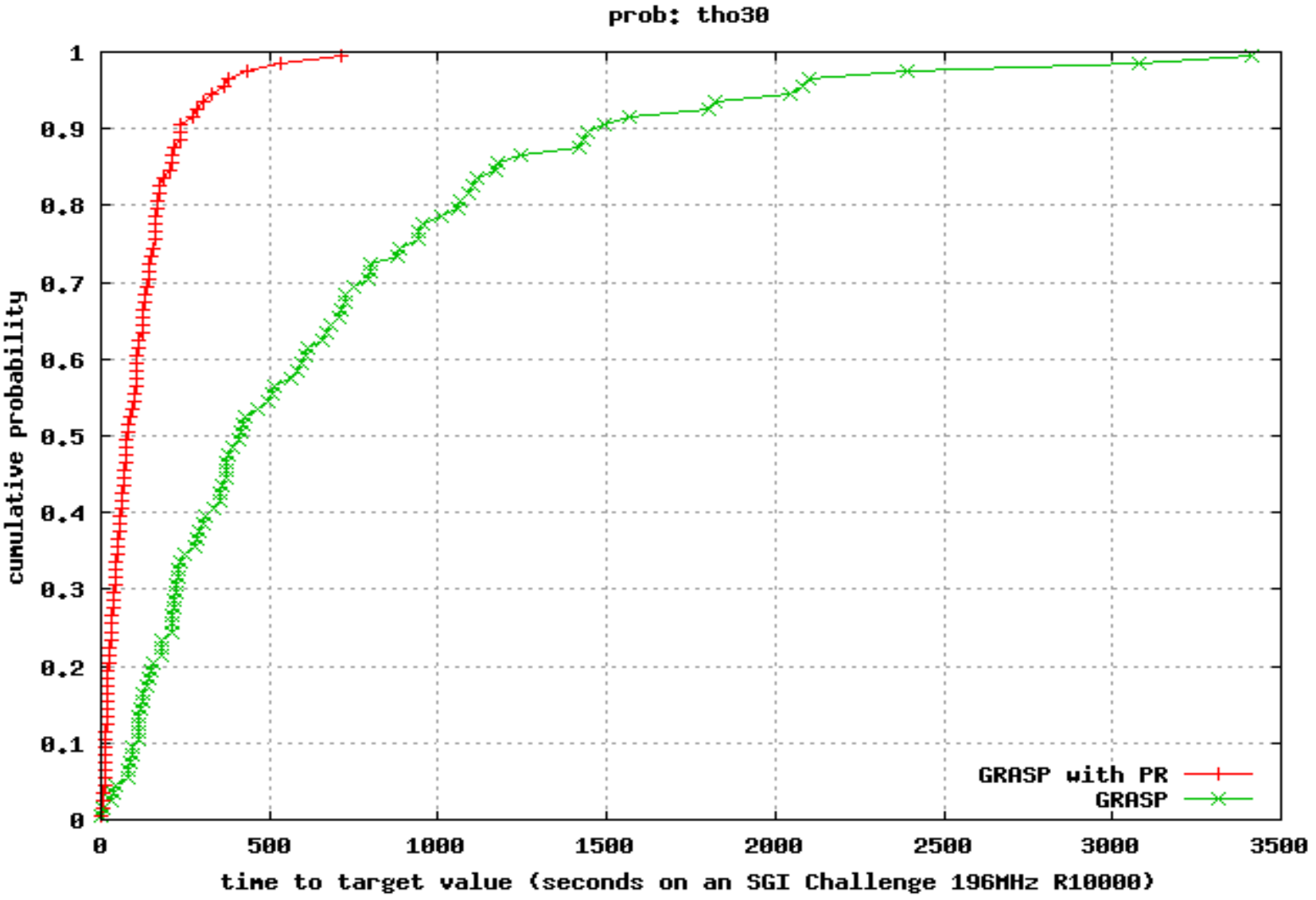
MAX-SAT (Festa, Pardalos, Pitsoulis, and Resende, 2006)



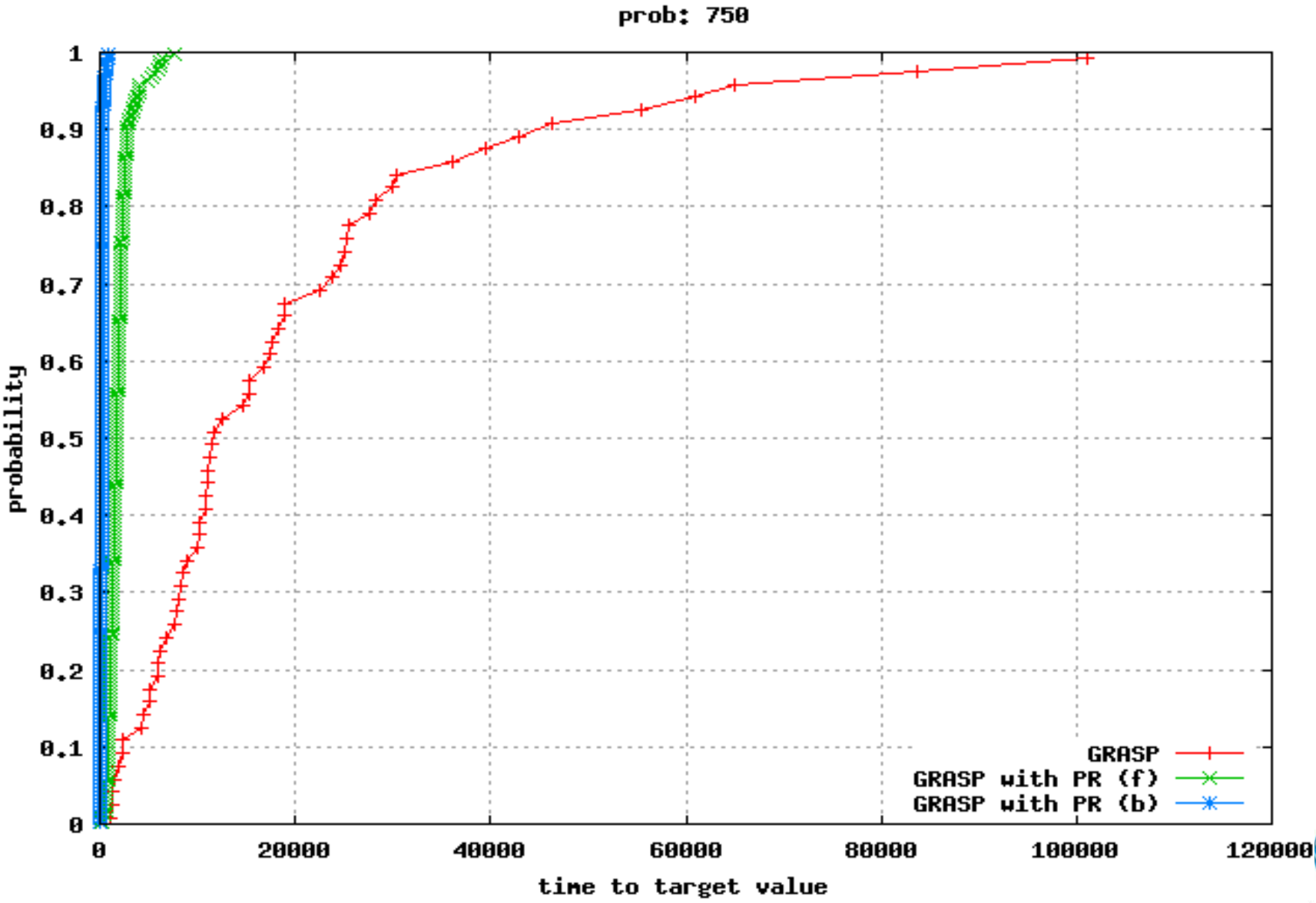
3-index assignment (Aiex, Resende, Pardalos, & Toraldo, 2005)



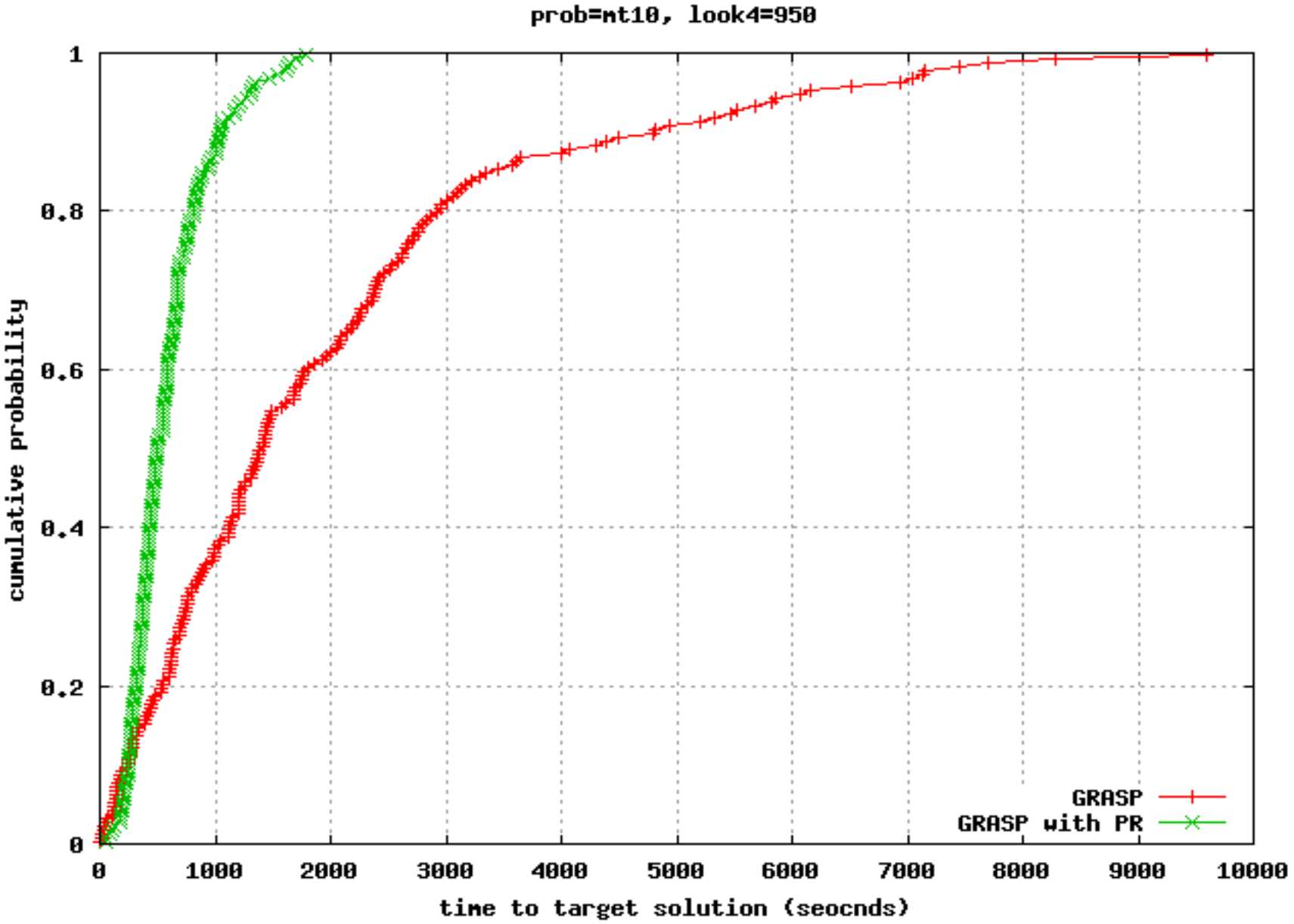
QAP (Oliveira, Pardalos, and Resende, 2004)



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 2) Y = local search to improve X 3) Path-relinking between Y and pool solution Z 4) Update pool

Evolutionary path-relinking (EvPR)

Evolutionary path-relinking

(Resende & Werneck, 2004, 2006)

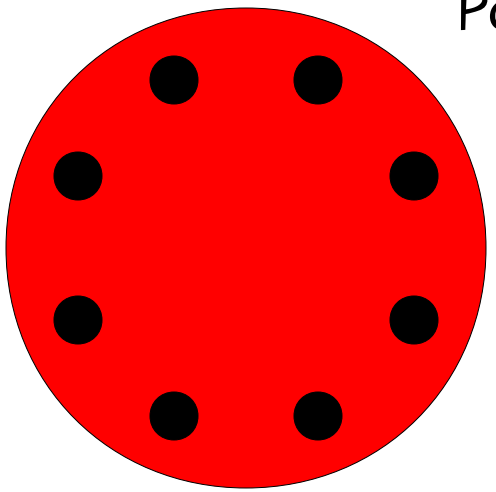
Evolutionary path-relinking "evolves" the pool, i.e. transforms it into a pool of diverse elements whose solution values are better than those of the original pool.

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)

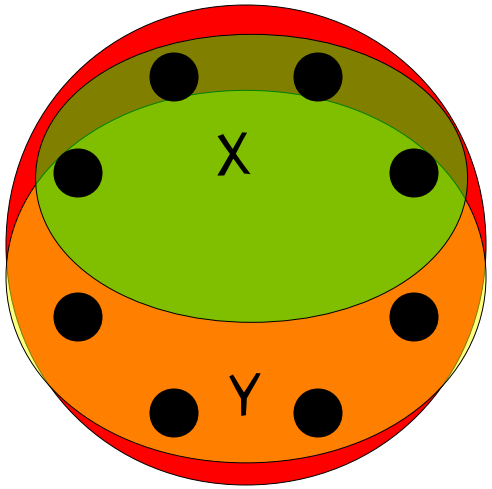


Population P(0)

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

Population P(0) is the current elite set.

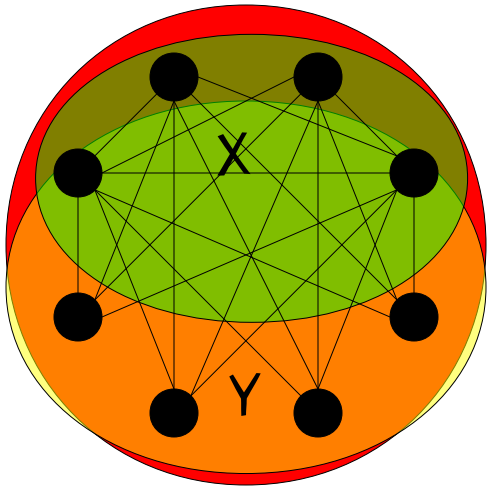
Evolutionary path-relinking (EvPR)



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.

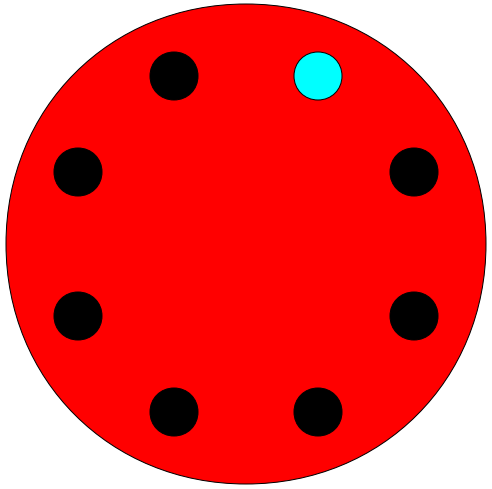
Evolutionary path-relinking (EvPR)



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.

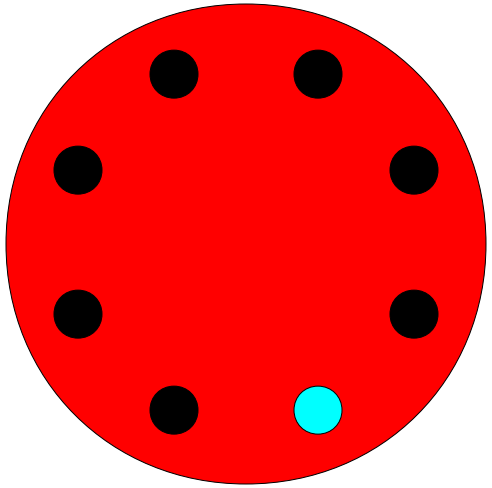
Evolutionary path-relinking (EvPR)



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.

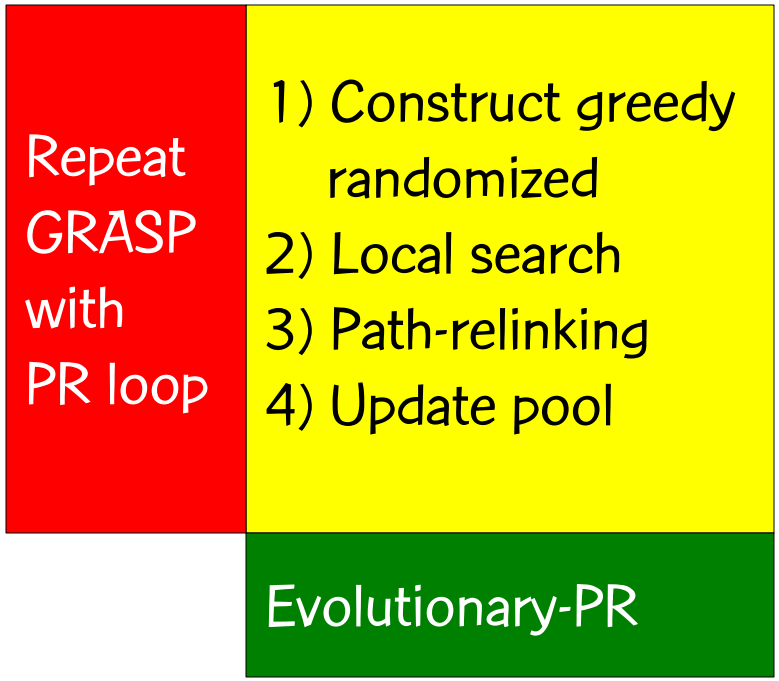
Else K=K+1 and repeat.



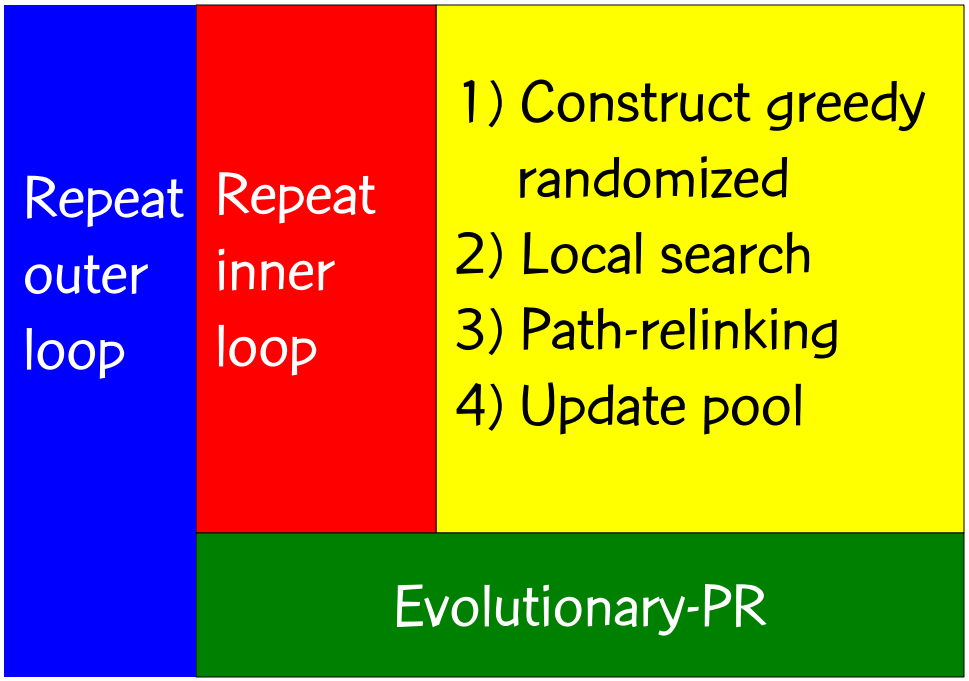
Population P(K+1)

GRASP with evolutionary path-relinking

As post-optimization



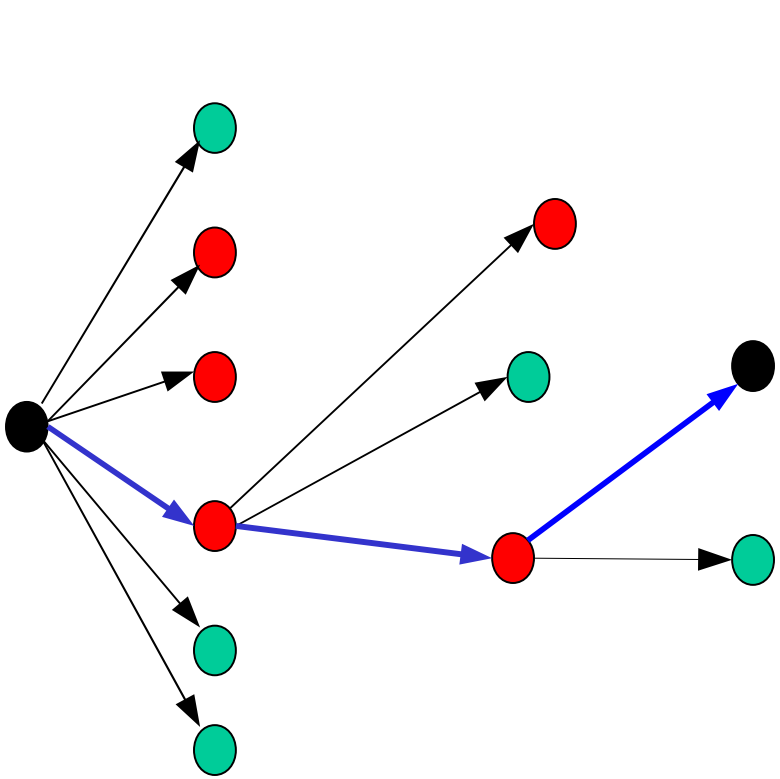
During GRASP + PR

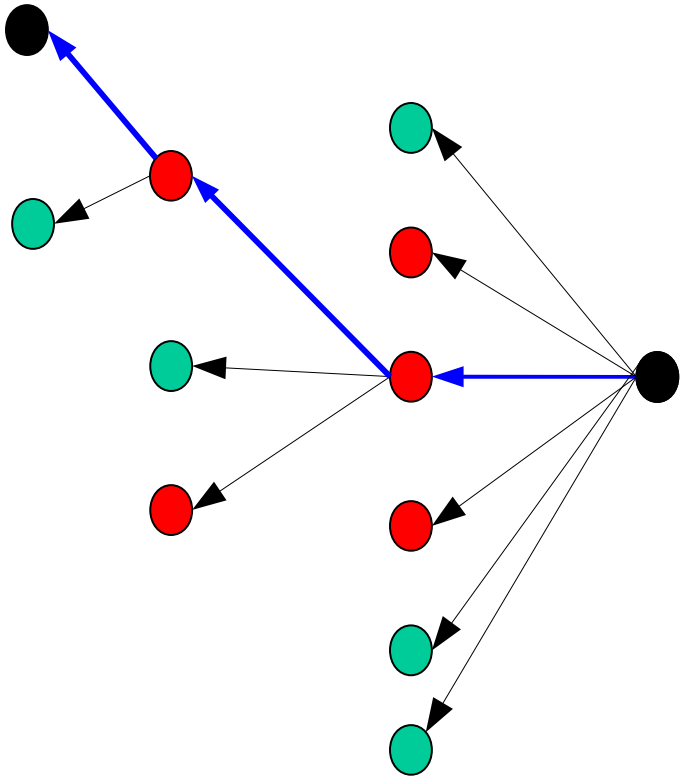


(Resende & Werneck, 2004, 2006)

GRASP with EvPR: Implementation ideas

Truncated mixed graPR





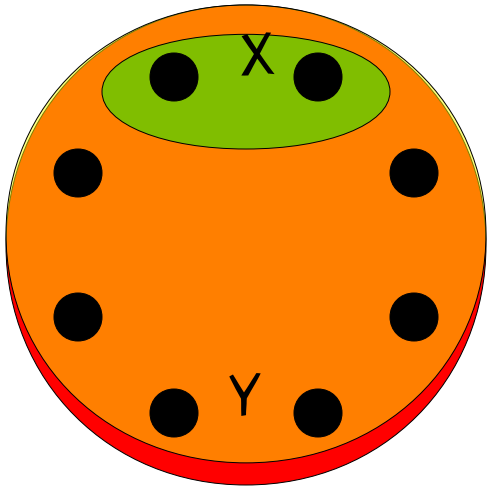
In PR and EvPR, apply one iteration of graPR.

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.

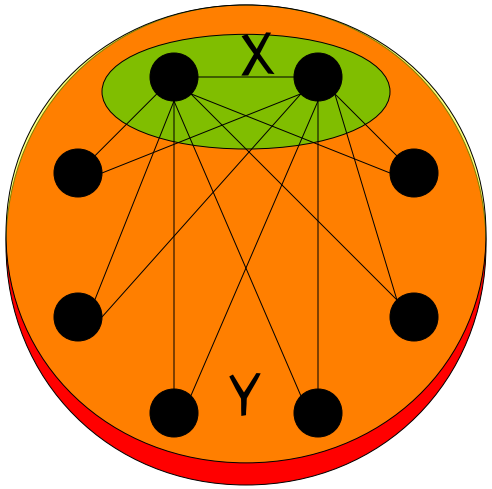
Speeds up EvPR.

Avoids unfruitful calls to graPR(x,y)

GRASP with EvPR: Implementation ideas

Make set X small and with best pool solutions.

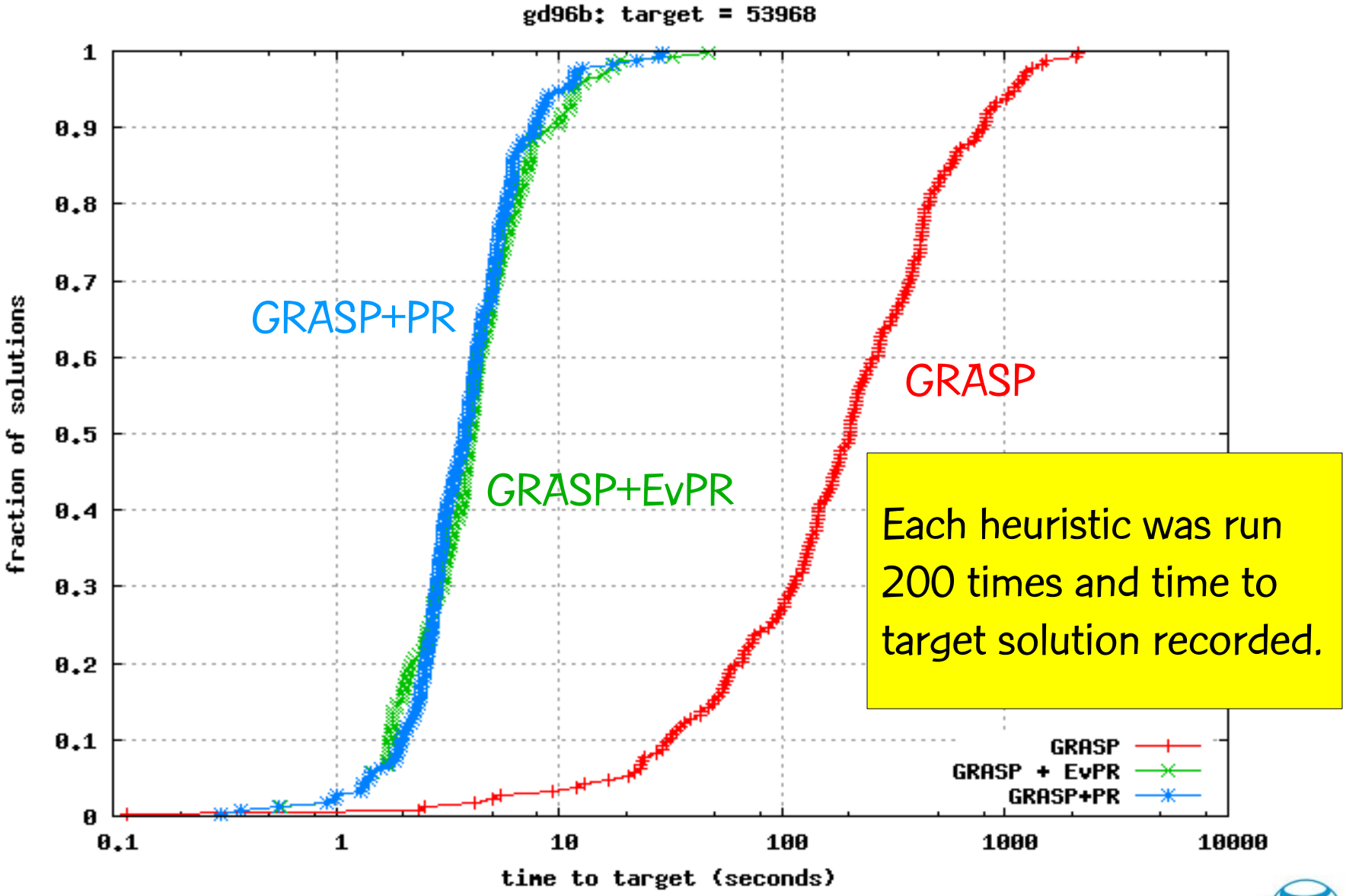
Make set Y be entire pool.



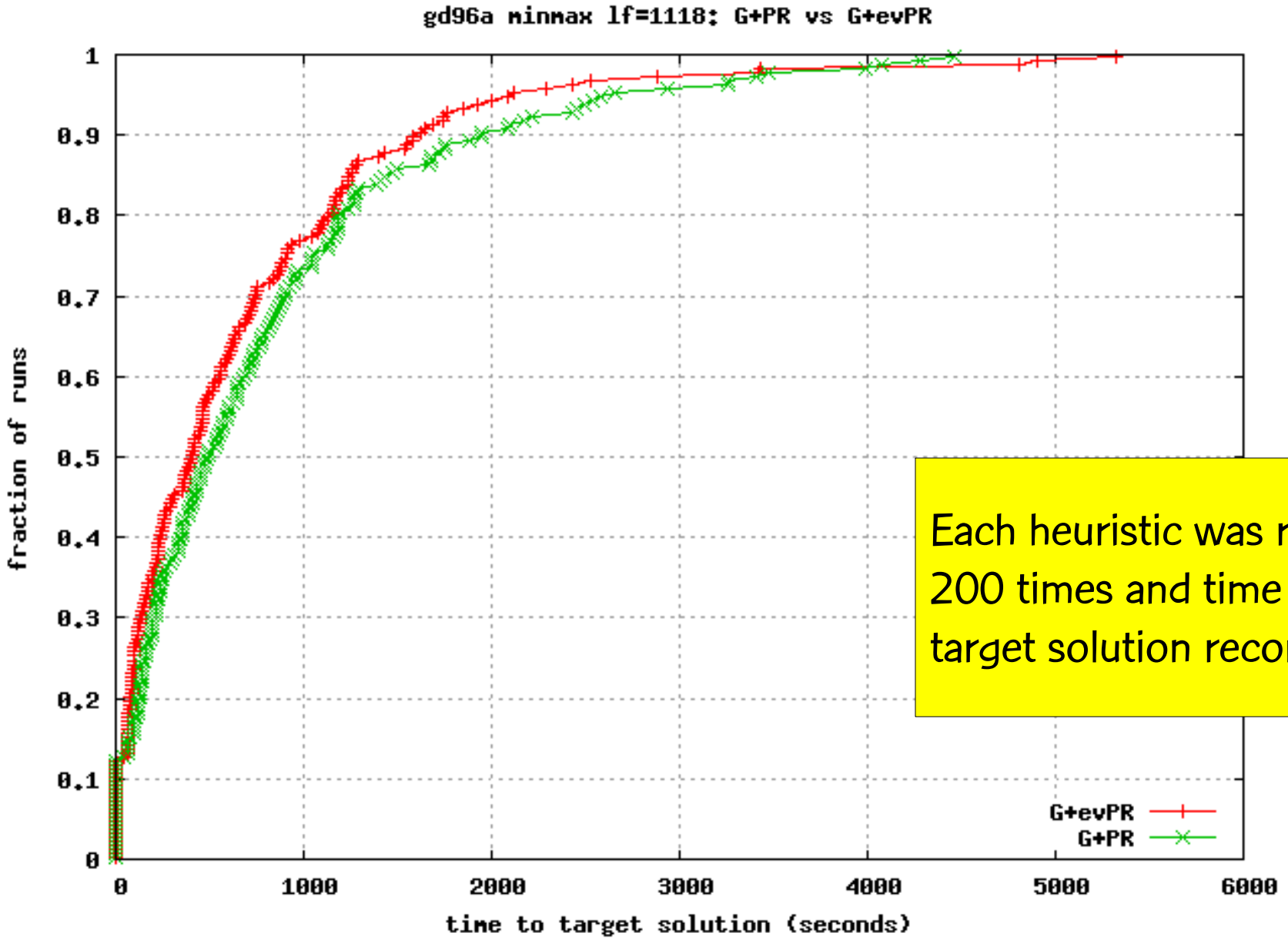
Use set X of size 1 or 2.

Speeds up EvPR.

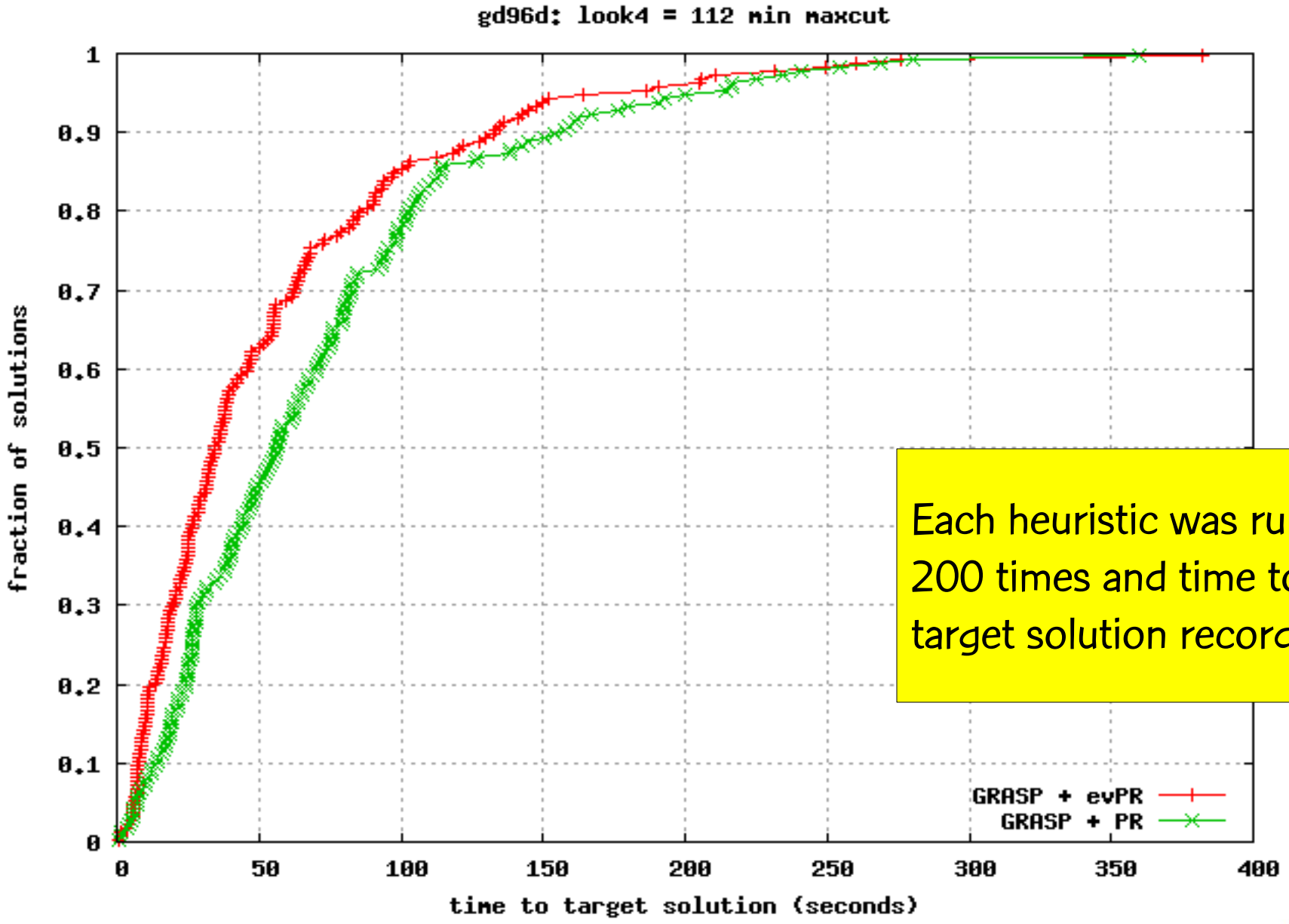
Avoids unfruitful calls to graPR(x,y)



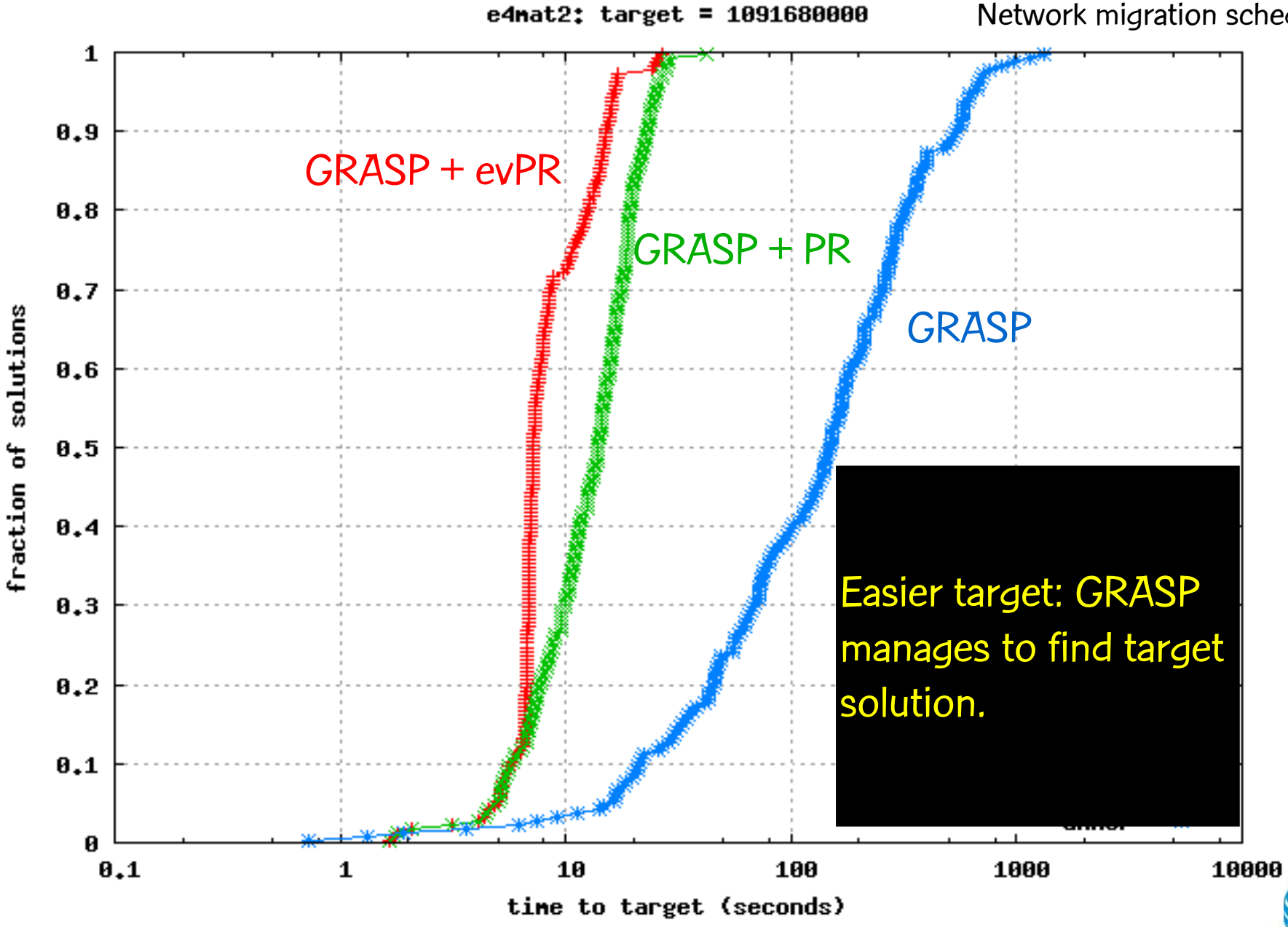
Each heuristic was run 200 times and time to target solution recorded.

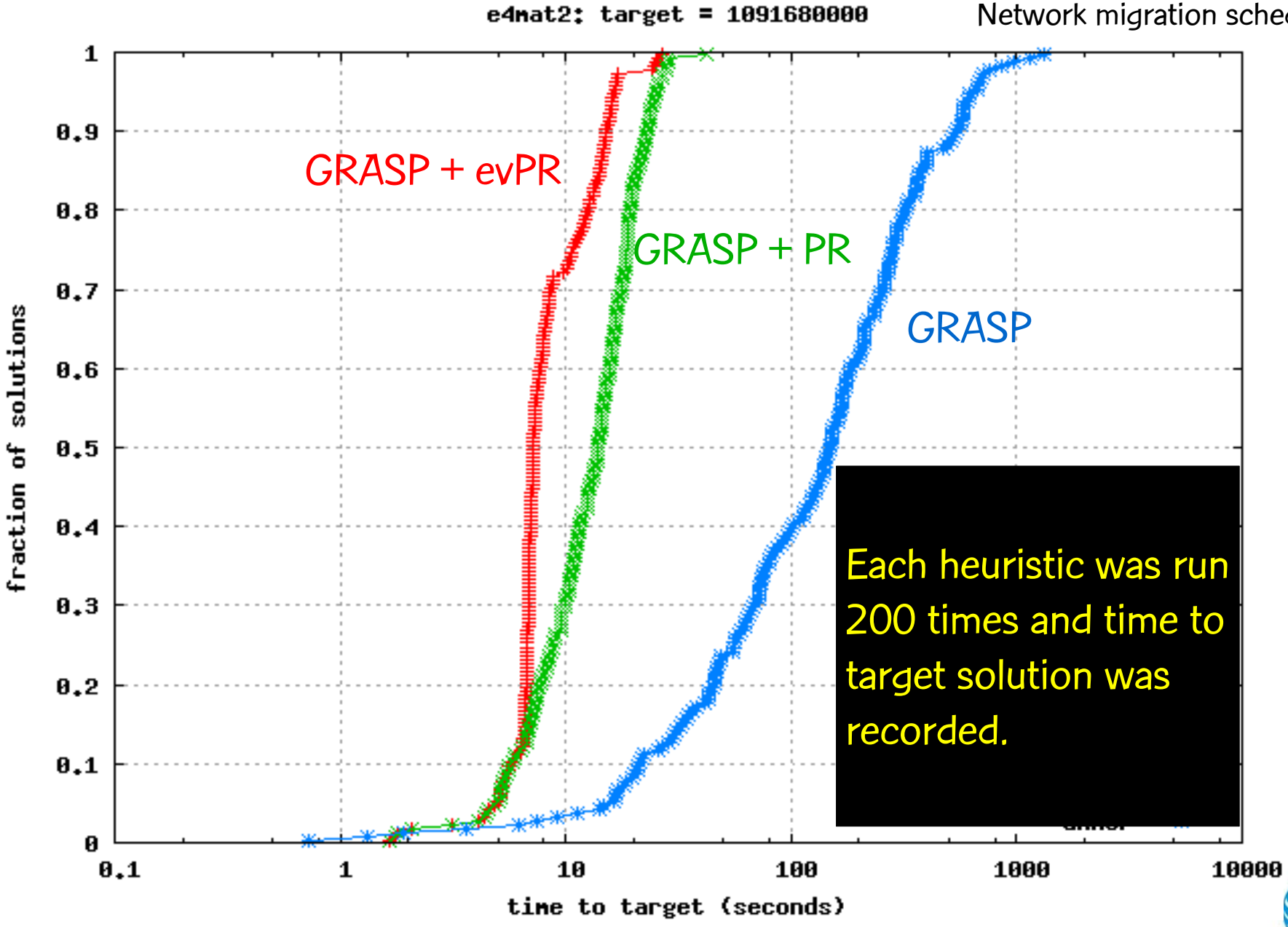


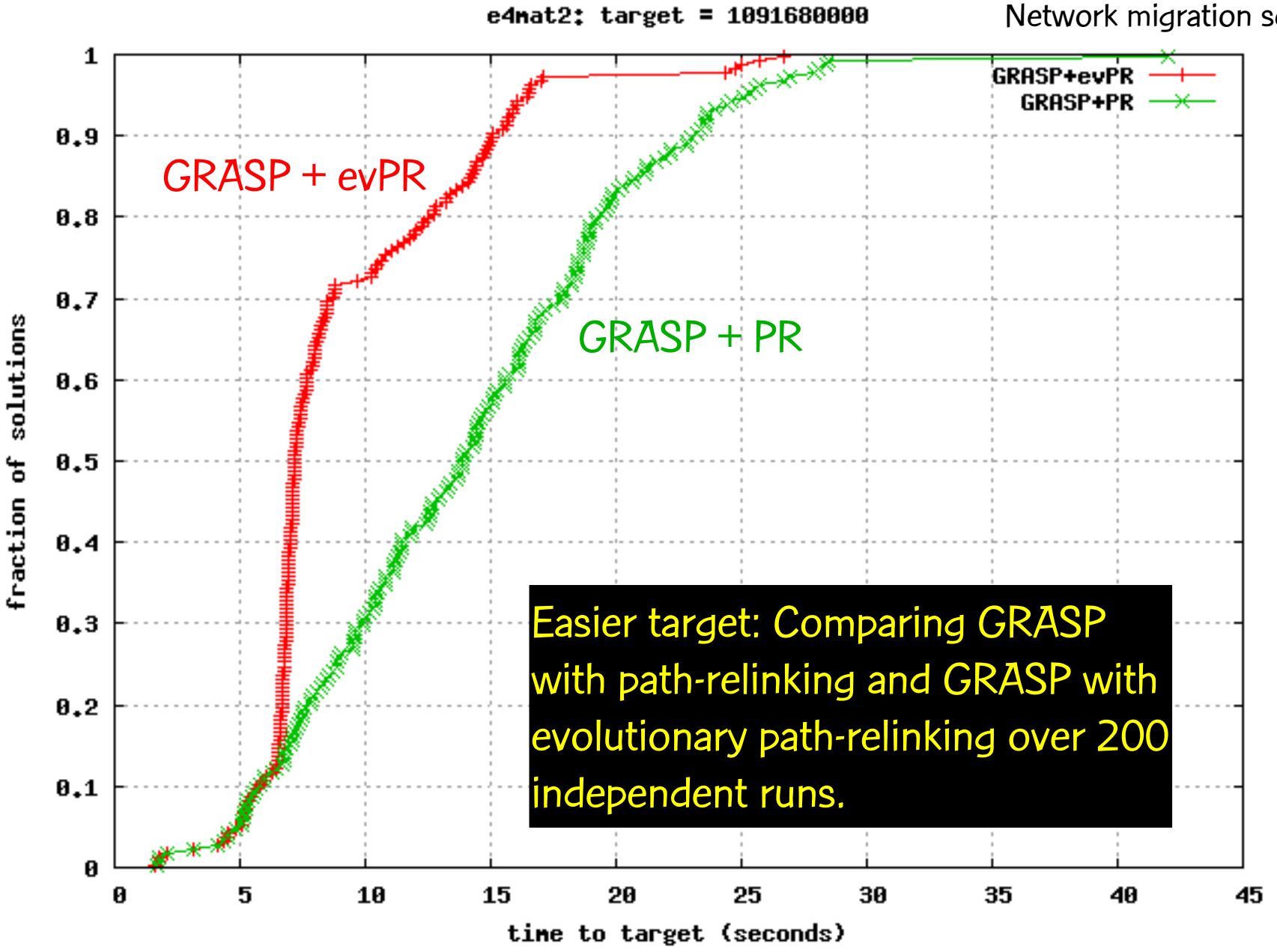
Each heuristic was run 200 times and time to target solution recorded.



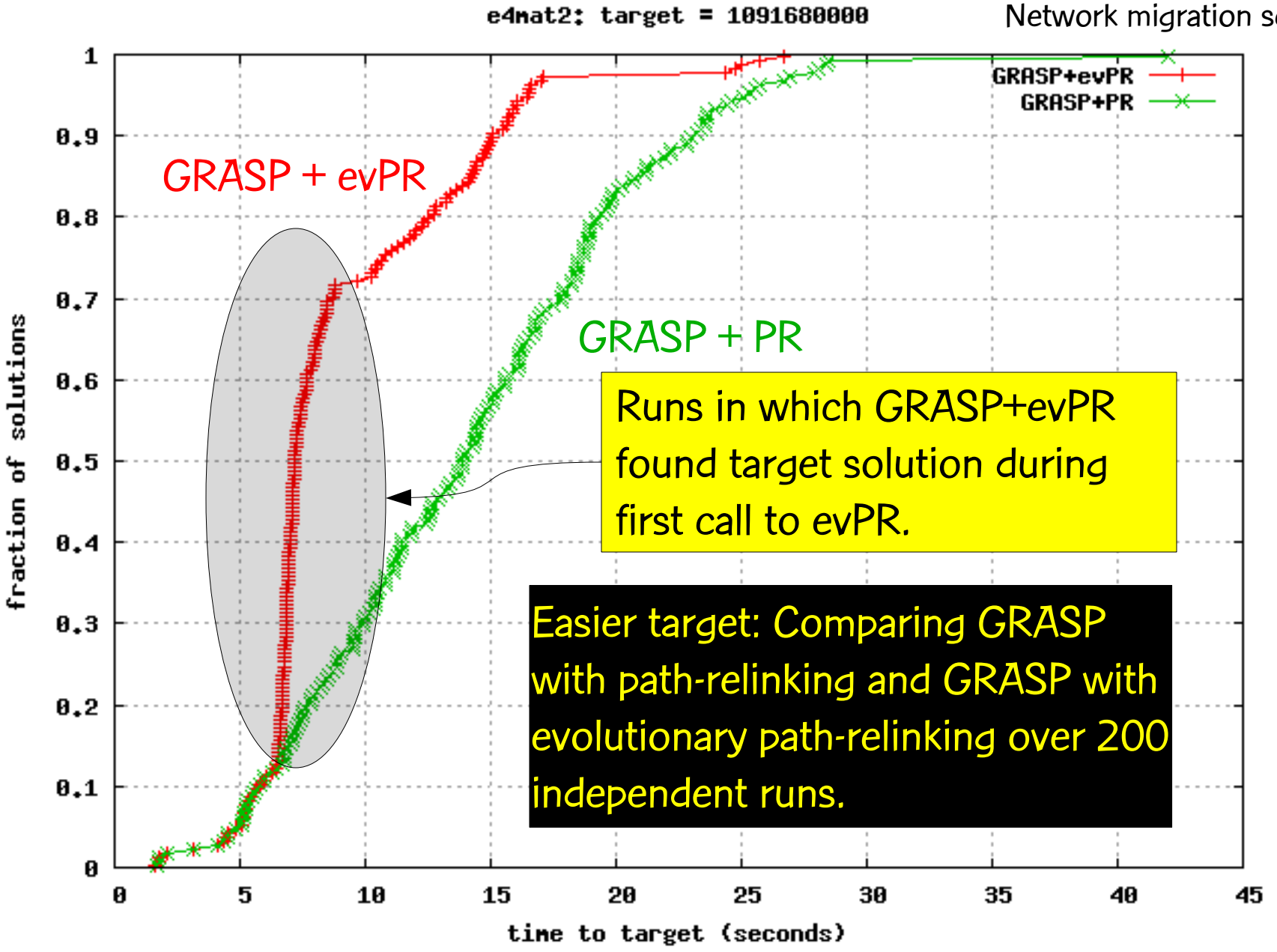
Each heuristic was run 200 times and time to target solution recorded.

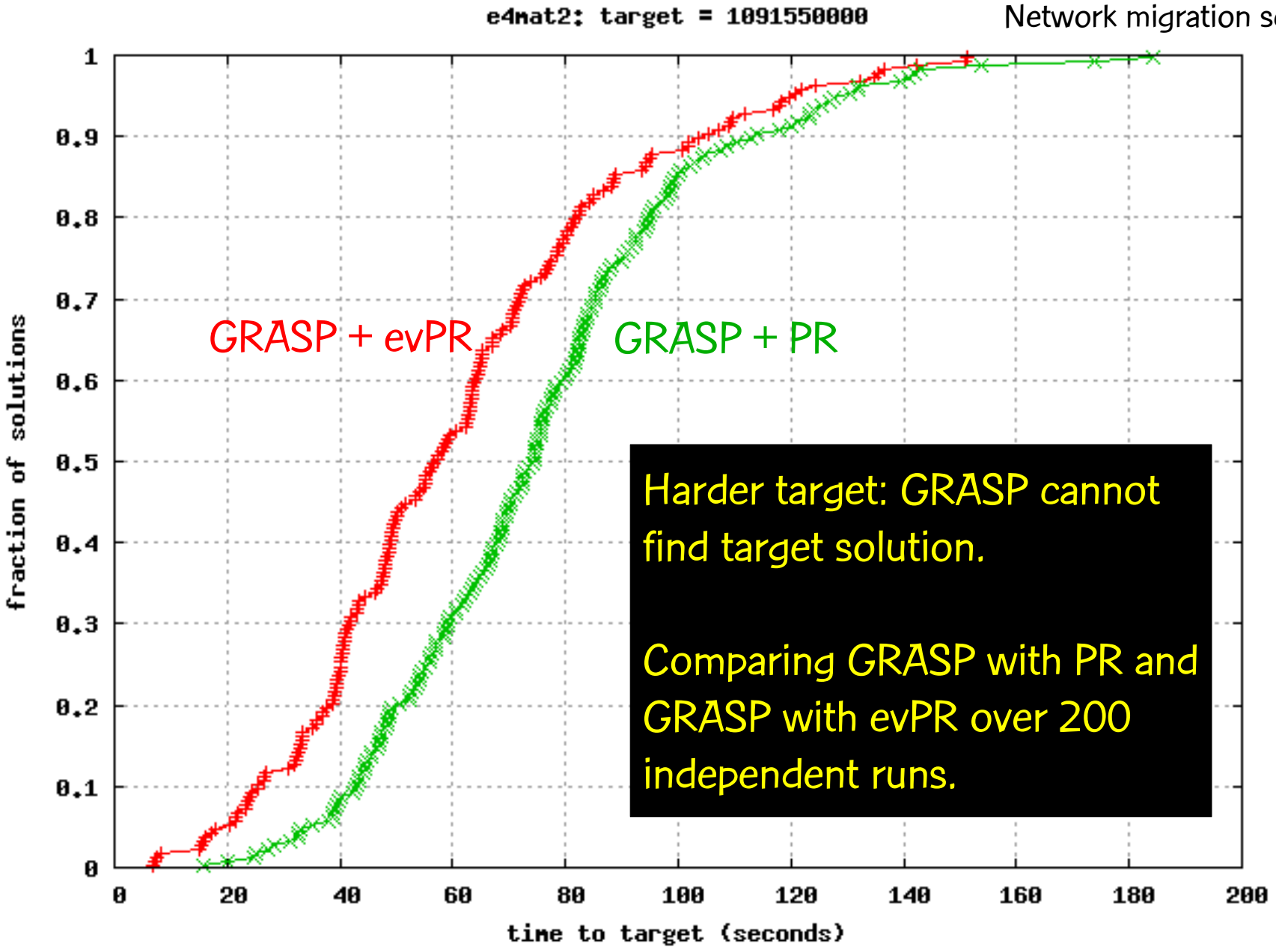






Easier target: Comparing GRASP with path-relinking and GRASP with evolutionary path-relinking over 200 independent runs.





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
- 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.

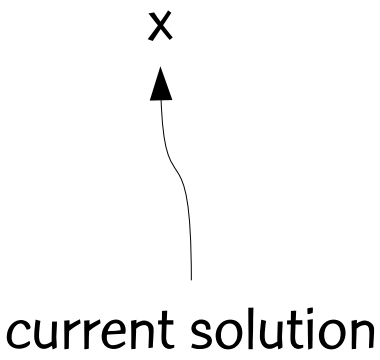
C-GRASP

- 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.

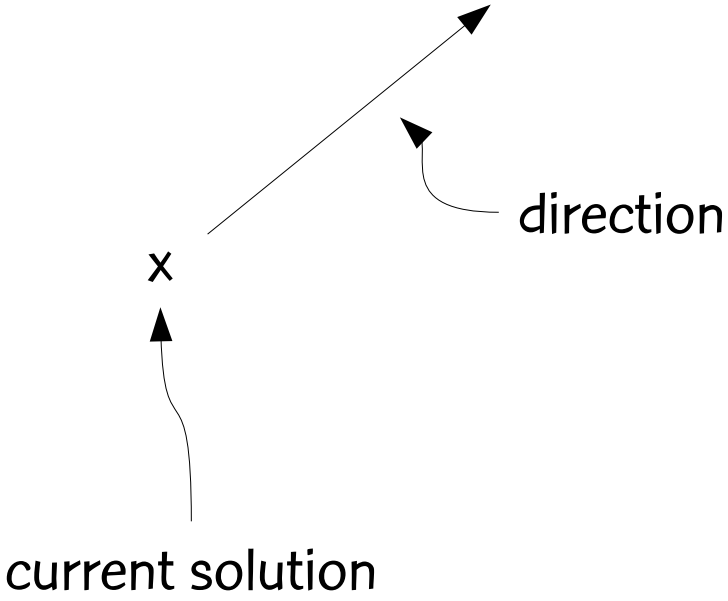
C-GRASP

```
f* = ∞
while (stopping criterion not satisfied) do
    x = random[L,U]; h = h(start);
    while ( h ≥ h(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*
```

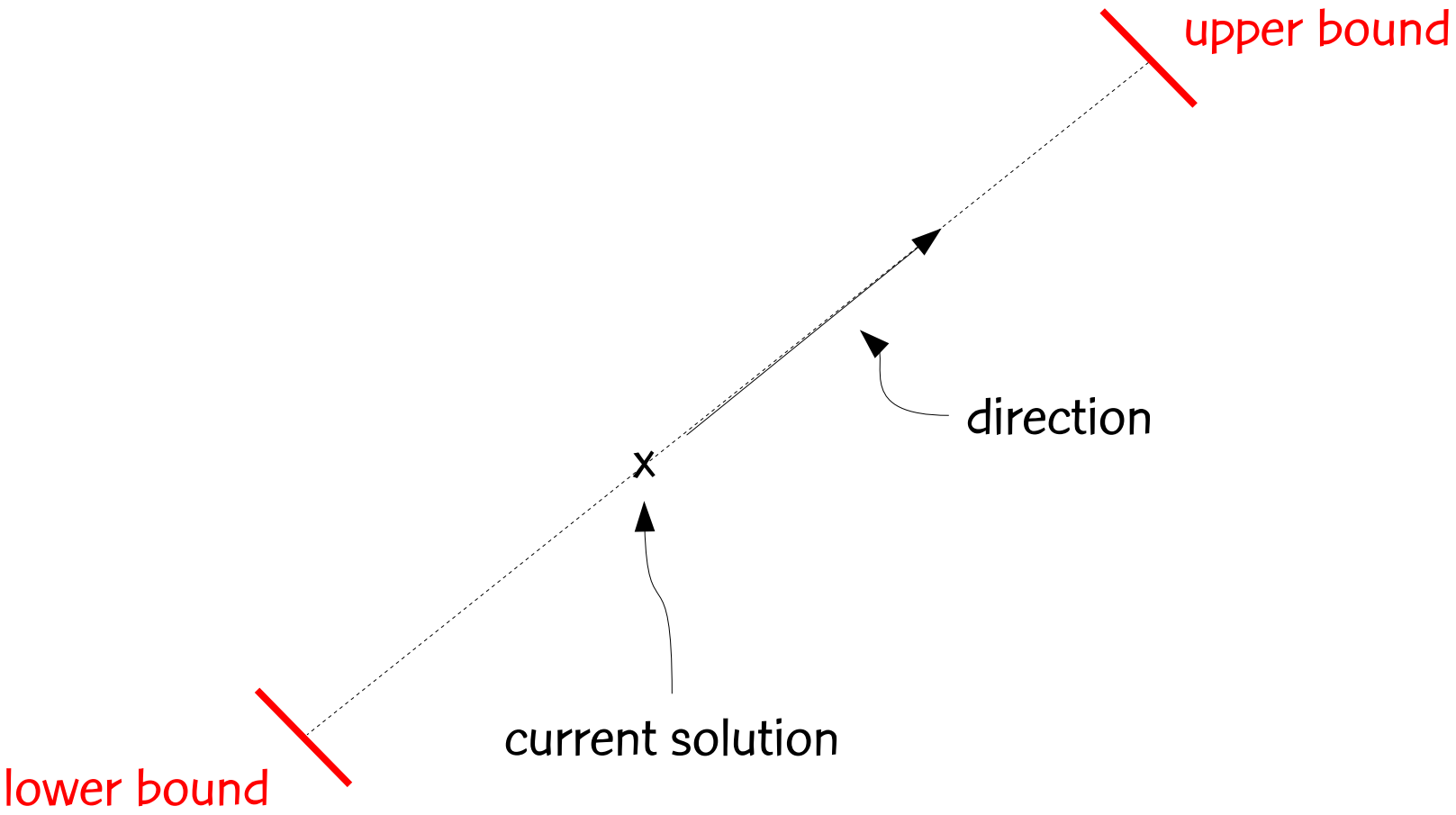
C-GRASP line search



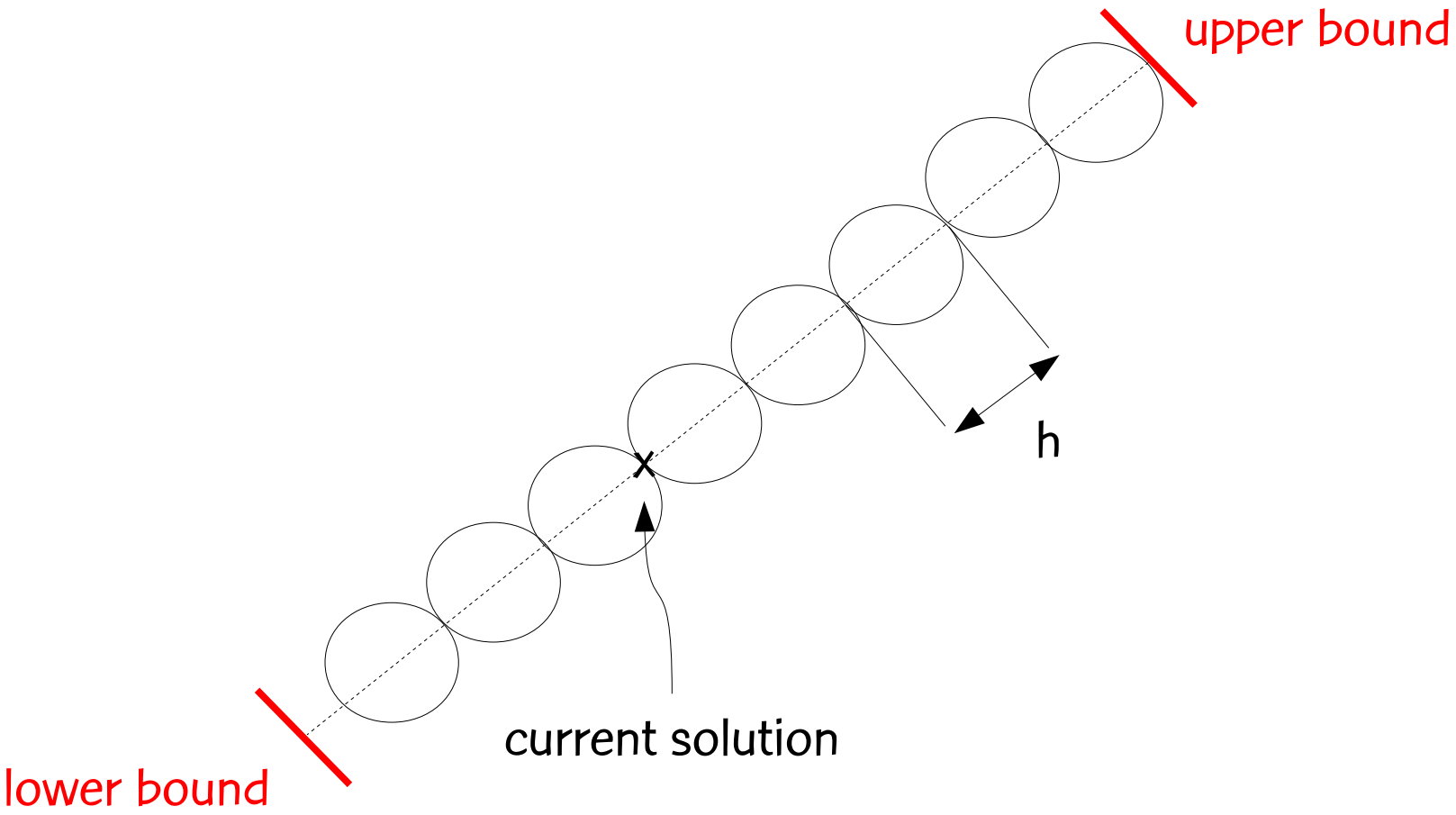
C-GRASP line search



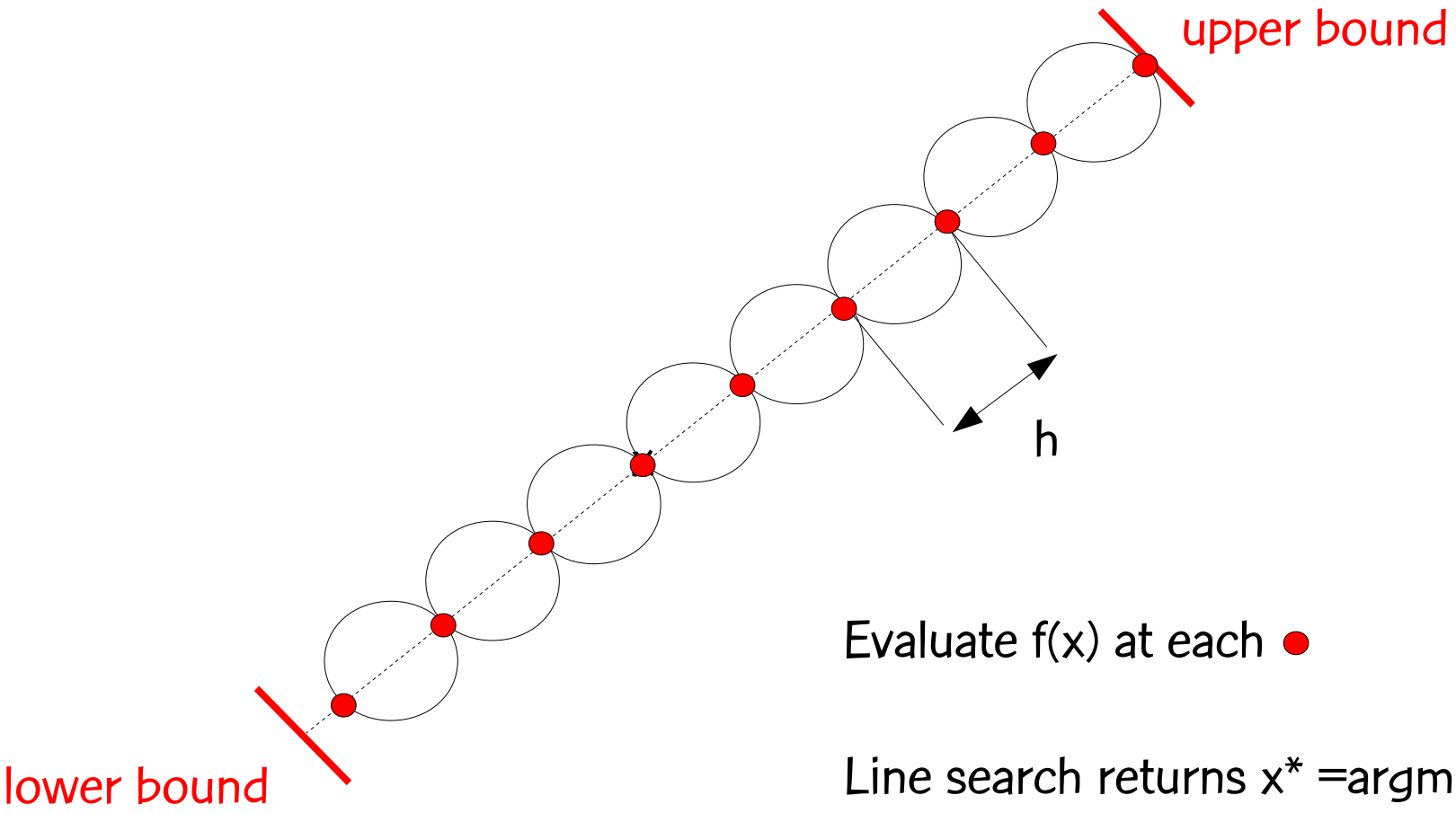
C-GRASP line search



C-GRASP line search



C-GRASP line search



Evaluate f(x) at each ●

Line search returns x* =argmin{ f(x) }

C-GRASP greedy randomized construction

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

for (k = 1, 2, ..., n) **do**

for (all i ∈ unset) **do**

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

i-th component

end for

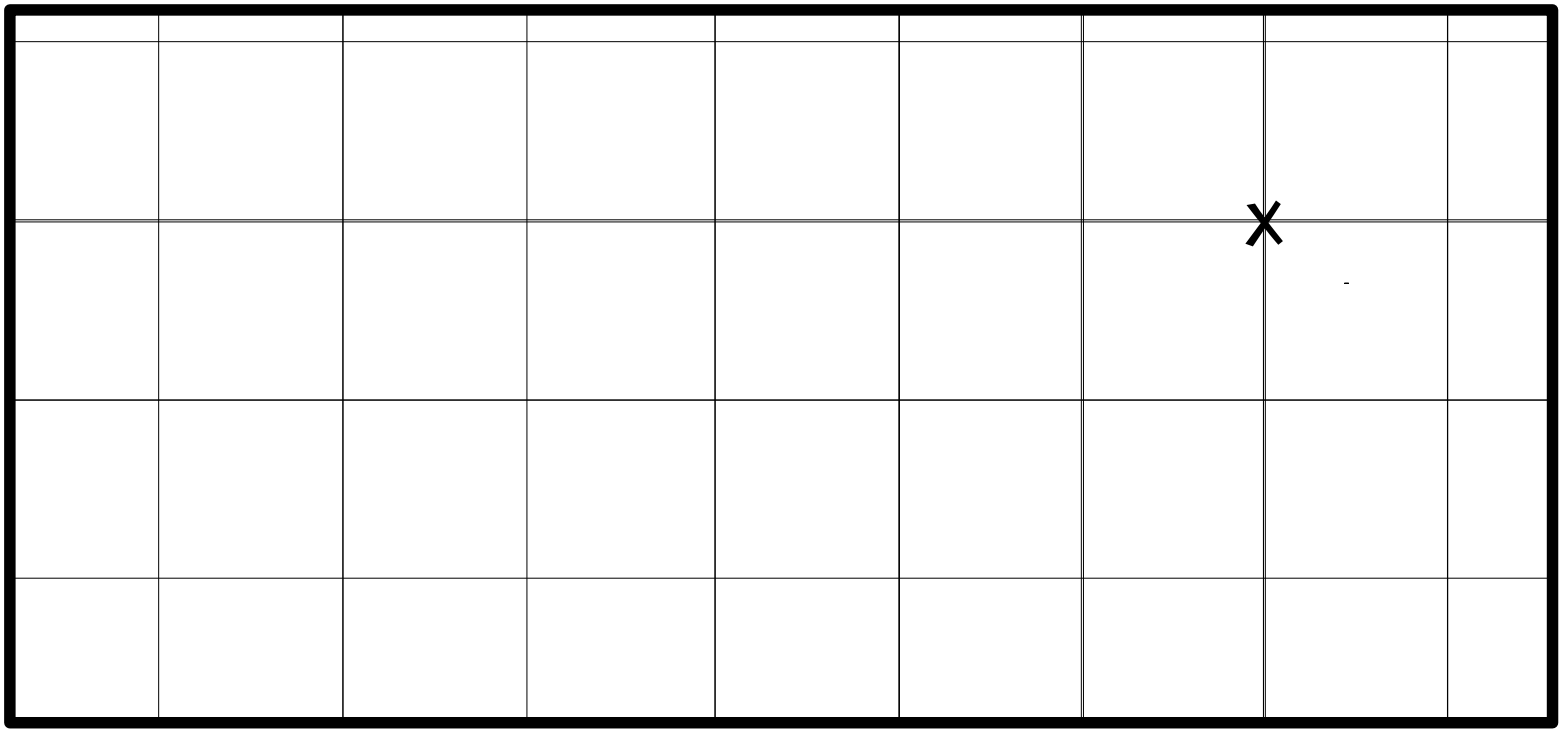
RCL = { i ∈ unset | $f(z_i) < CUTOFF$ }

Select at random i* ∈ RCL

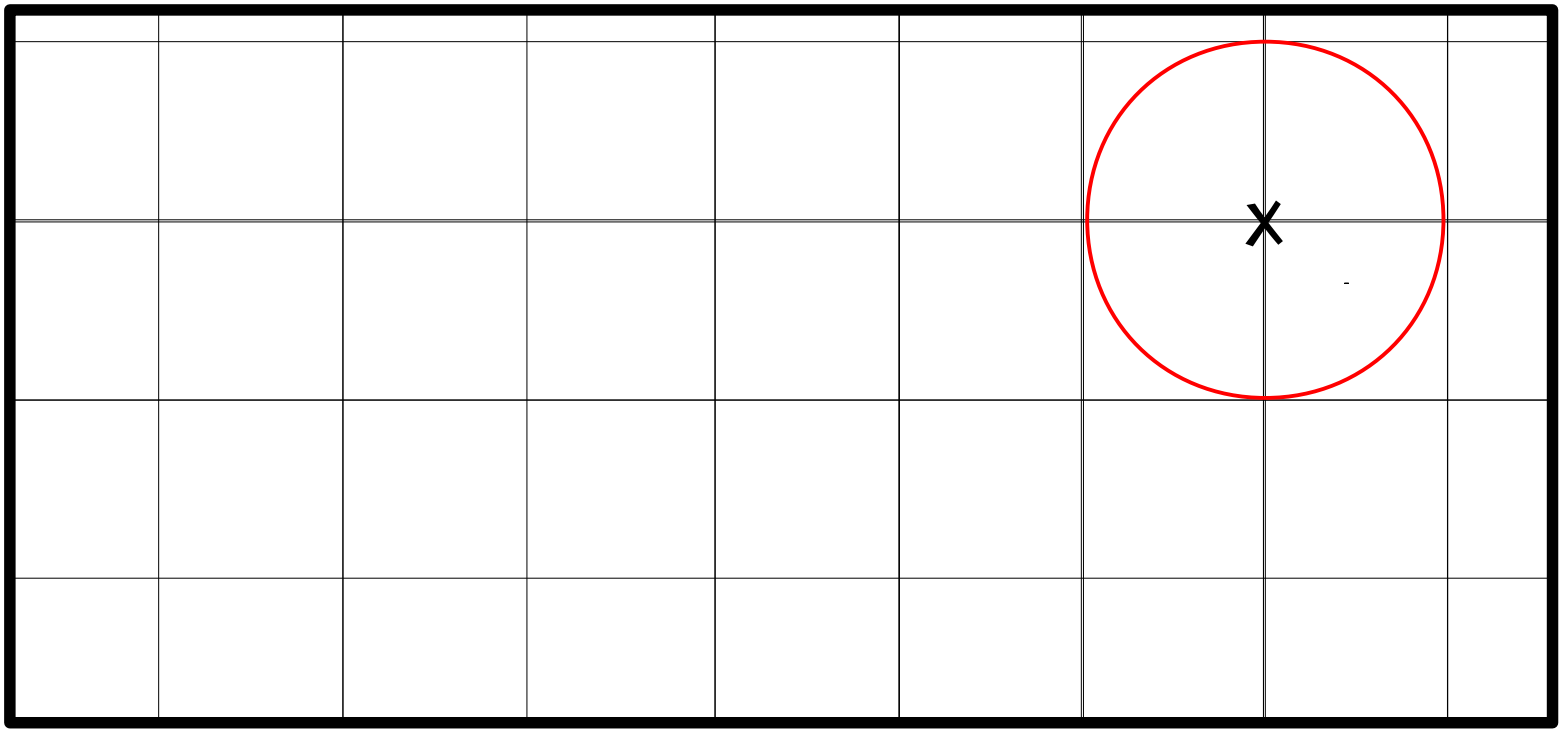
Set $x_{i^*} = z_{i^*}$; unset = unset \ {i*}

end for

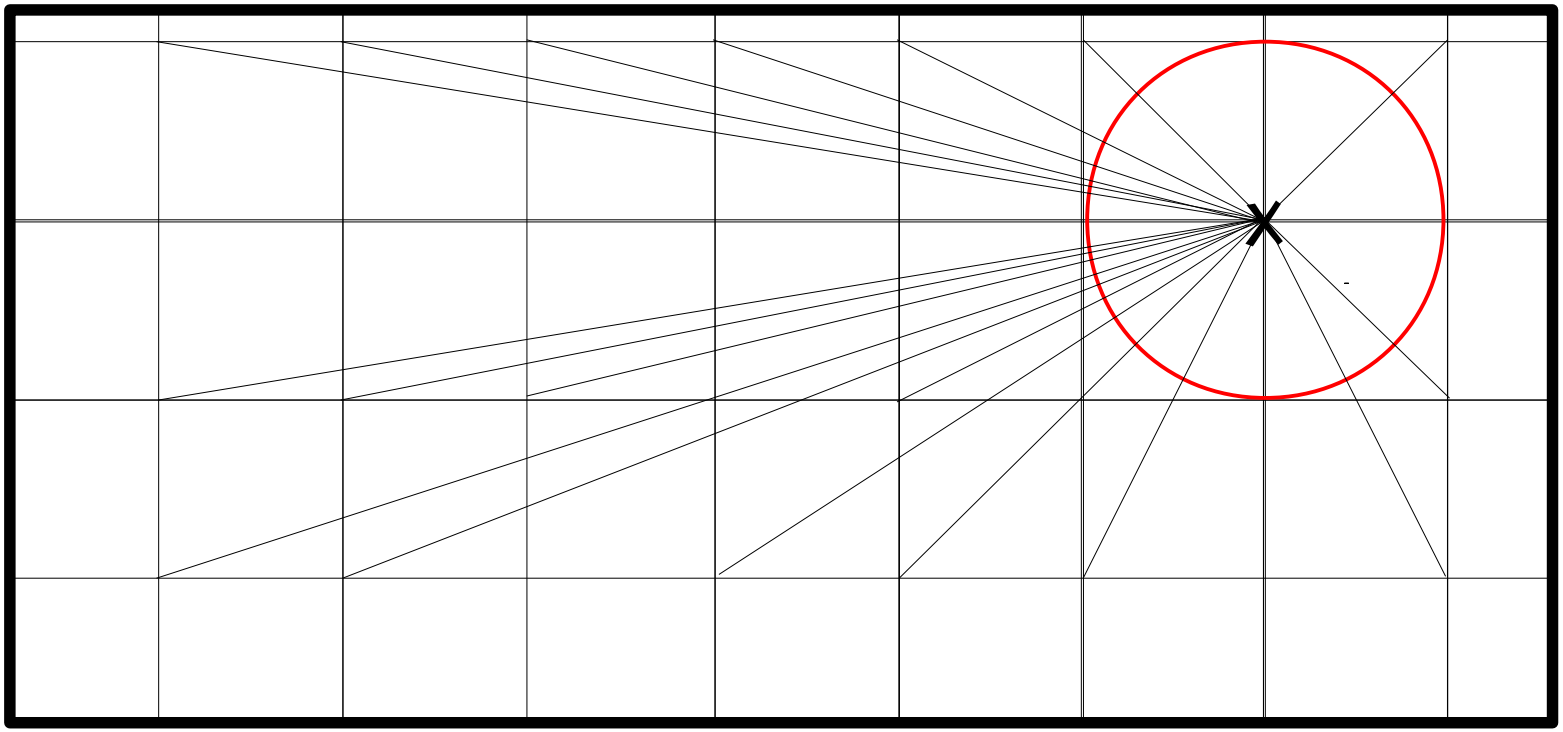
C-GRASP local improvement



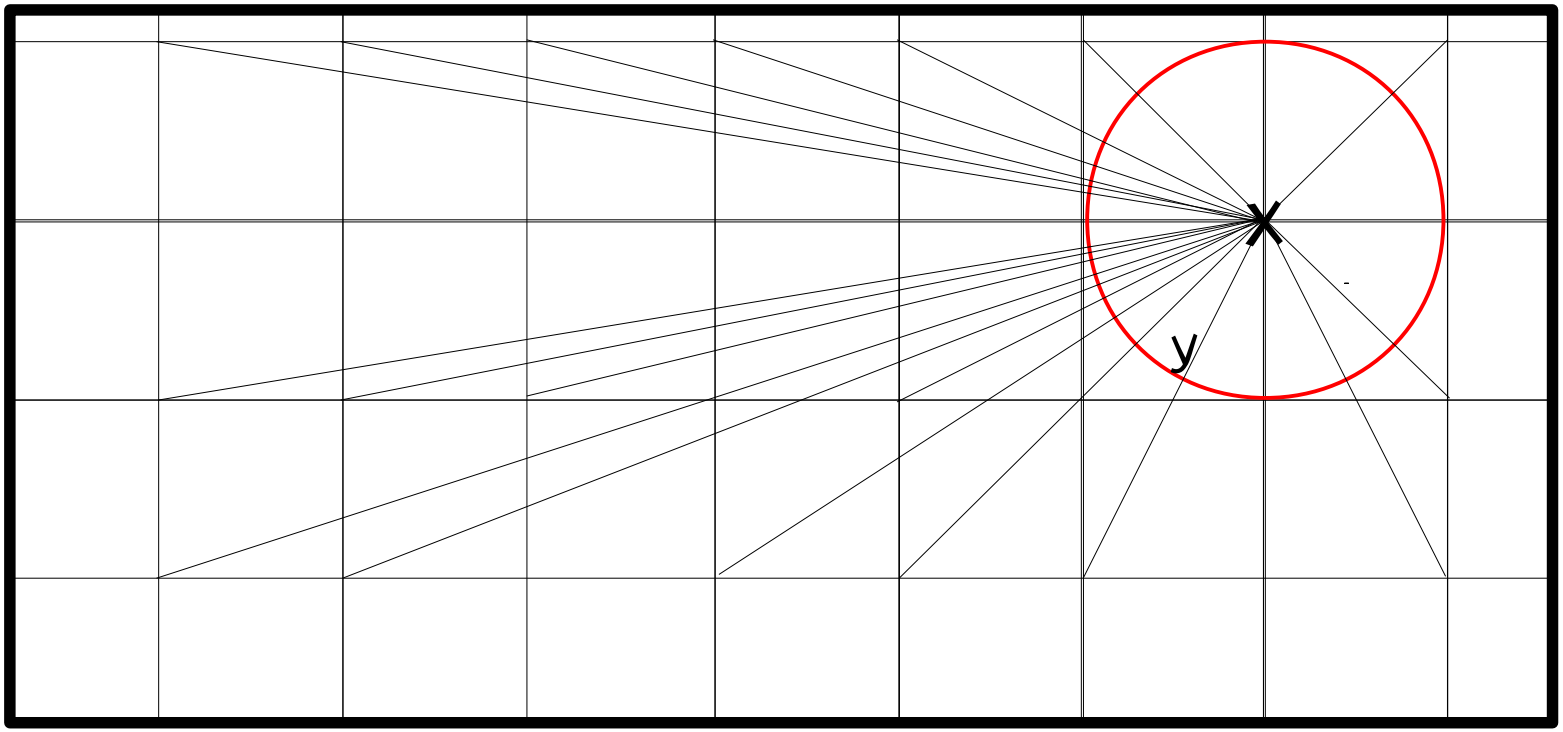
C-GRASP local improvement



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 been granted 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

Detecting cliques in massive sparse graphs

Tutorial given at the Spring School in Advances in Operations Research, Higher School of Economics
Nizhny Novgorod, Russia ✤ May 3, 2011

Maurício G. C. Resende
AT&T Labs Research
Florham Park, New Jersey

mgcr@research.att.com
http://www2.research.att.com/~mgcr

Joint work with J. Abello, P.M. Pardalos, and S. Sudarsky

Summary of talk

- Data explosion
- Massive graphs arising from telephone call detail database
- Structure of call detail graph
- Searching for large cliques and bicliques
- Some experimental results

Data explosion

(Abello, Pardalos, & R., Eds., "Handbook of Massive Data Sets," Kluwer, 2001)

- Proliferation of massive data sets brings with it computational challenges
- Data avalanche arises in a wide range of scientific and commercial applications
- Today's data sets are of high dimension and are made up of huge numbers of observations:
  - More often they overwhelm rather than enlighten
- Outstripped the capabilities of traditional data measurement, data analysis, and data visualization tools

Data explosion

- A variety of massive data sets can be modeled as a very large multi-digraph
  - Special set of edge attributes represent special characteristics of application
- WWW: nodes are pages, edges are links pointing from one page to another
- Telephone call graph is another example …

Call detail

- Every phone call placed on AT&T network generates a record (~ 200 bytes) with:
  - Originating & terminating numbers
  - Start time & duration of call
  - Other billing information
- The collection of these records is known as the Call Detail Database

Call detail

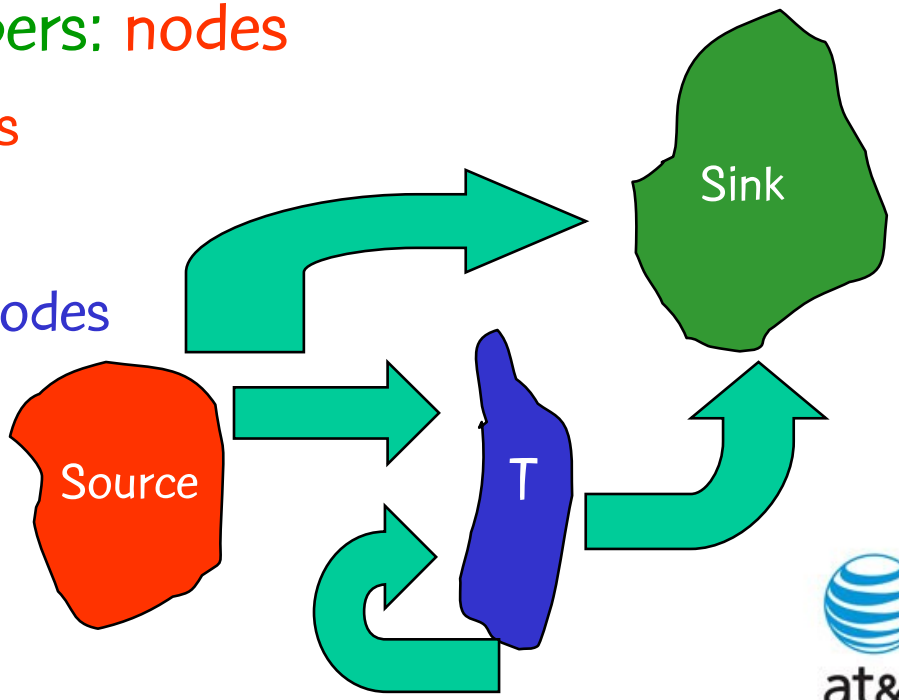
- AT&T system (currently) generates:
  - 250 million records per day (on average)
  - 320 million records on busy day
  - 18 terabytes of data per year
- Data is accessed for:
  - Billing & customer inquiries
  - Marketing & traffic engineering

Call detail graph

- $G = (V, E)$ is a directed graph:
  - V is the set of phone numbers
  - E is the set of phone calls
    - $(u,v) \in E$ implies that phone u called phone v
- G quickly grows into a huge graph
  - Hundreds of millions of nodes and billions of edges
  - Our goal is to work with one year of data (~ 1 Tb)

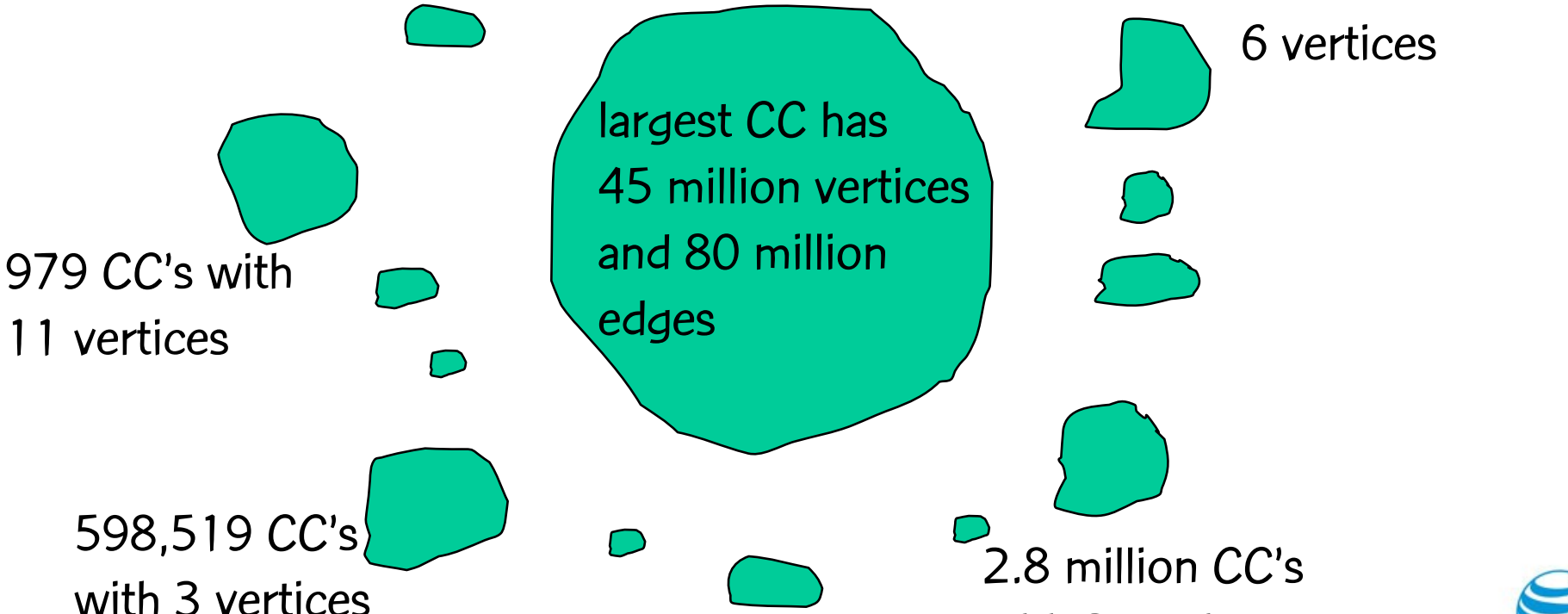
Structure of call detail graph

- Consider a 12-hour call detail graph
  - 123 million records: edges
  - 53 million phone numbers: nodes
    - 21 million source nodes
    - 22 million sink nodes
    - 10 million transmittal nodes

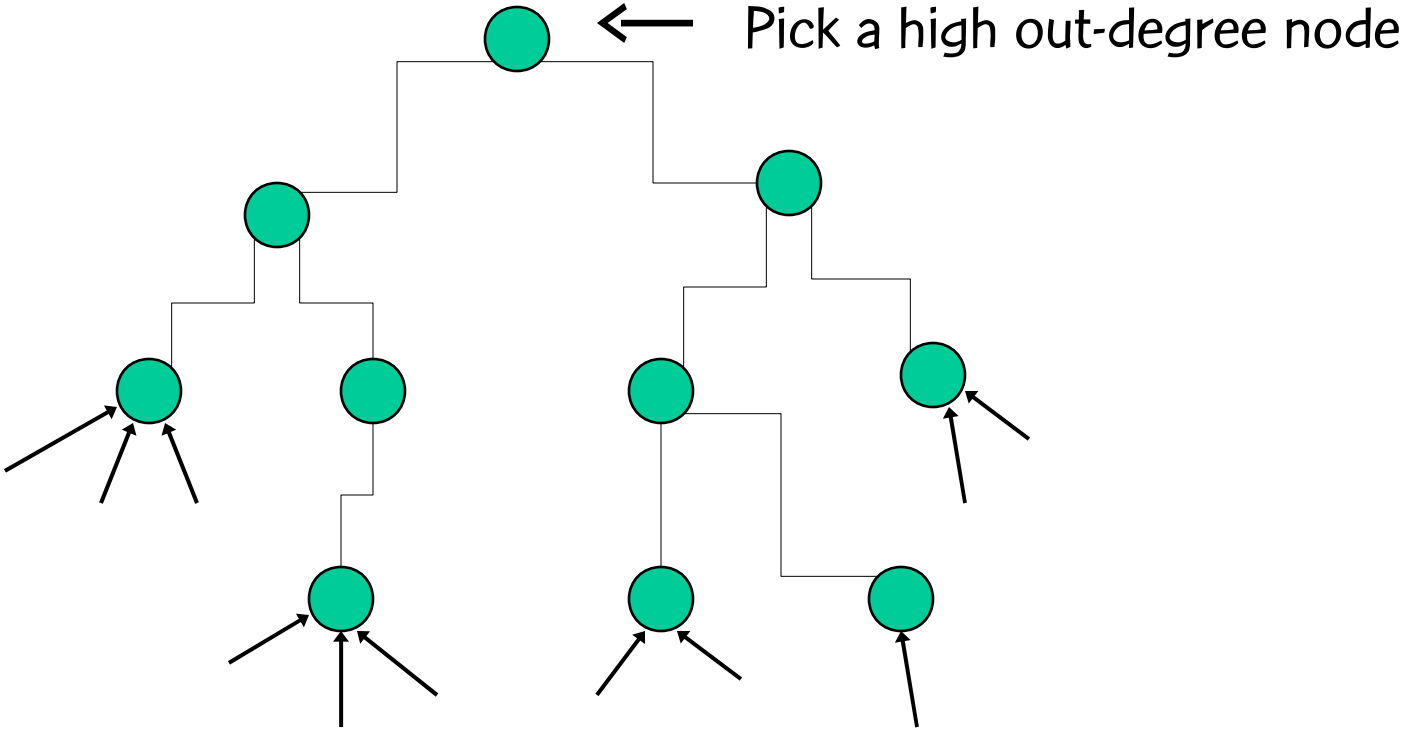


Connected components

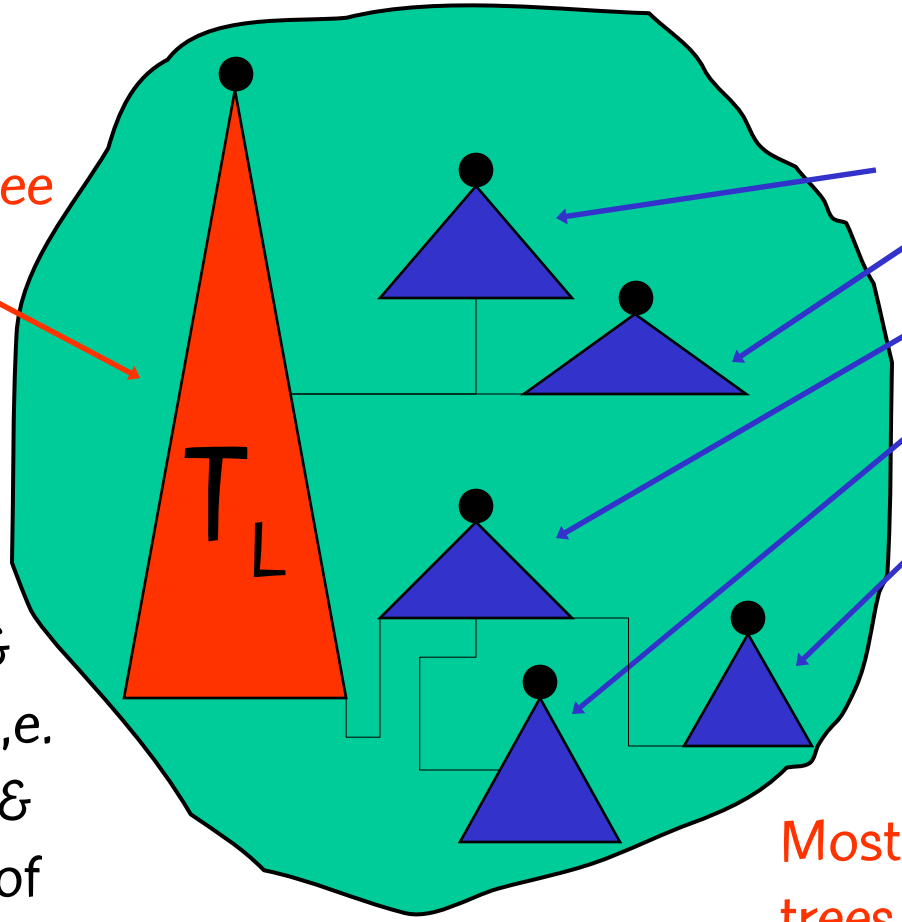
3.6 million connected components



Depth first search (DFS) tree



DFS trees in largest CC

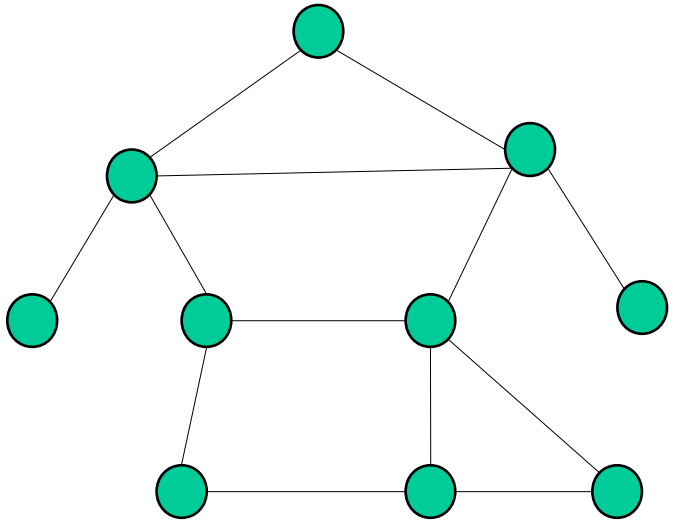


$G(T_L)$ has
10 million nodes &
19 million edges, i,e.
22% of the nodes &
24% of the edges of
the CC

18 million DFS trees

Most edges are within trees.

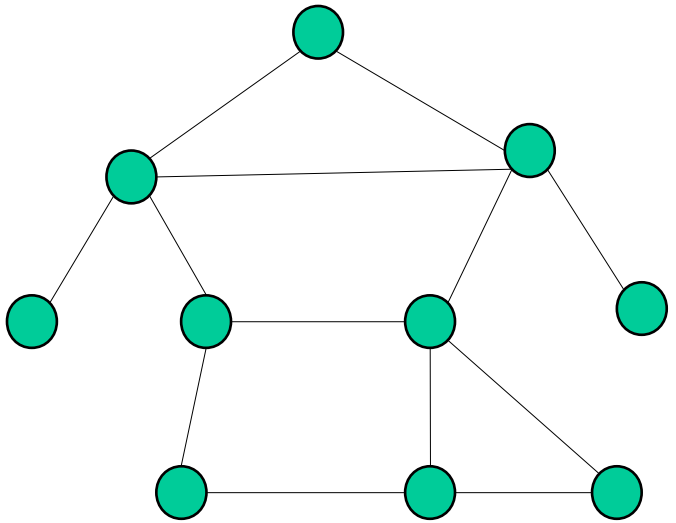
Subgraph induced by DFS tree nodes



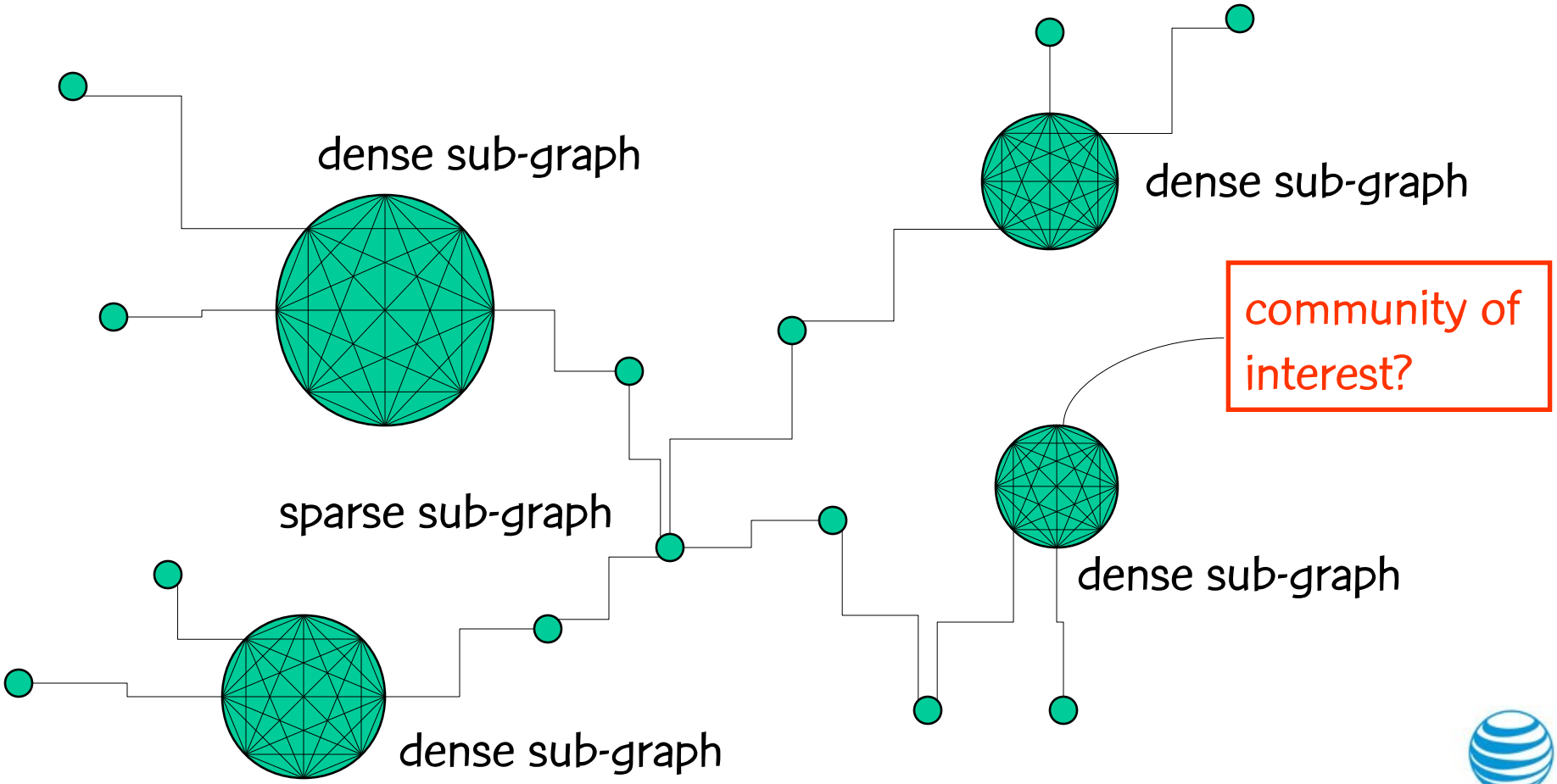
- Most subgraphs induced by DFS tree nodes are very sparse: $|E| < \log(|V|)$
- Few are dense: $|E| > \text{sqrt}(|V|)$ with at most 32 nodes

Dense subgraphs

- Dense subgraphs could be
  - within *G* (DFS tree)
  - among different *G* (DFS tree)
- Counting edges:
  - most are within *G* (DFS tree)
  - leaves few edges between different *G* (DFS tree)

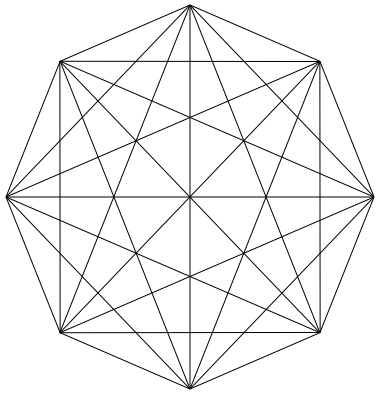


Macro structure of call detail graph

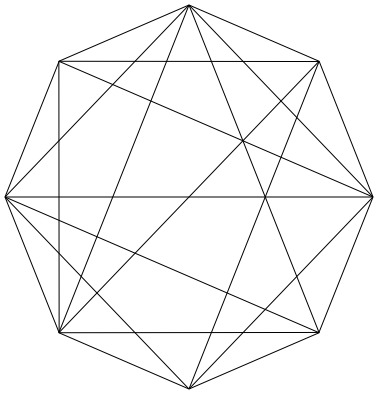


Searching for dense subgraphs

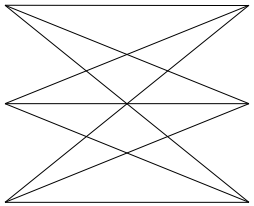
- We look for two types of subgraphs
  - cliques or quasi-cliques
  - bicliques or quasi-bicliques



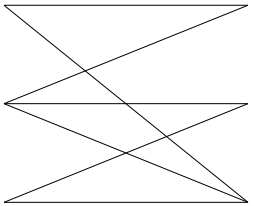
clique



quasi-clique



biclique



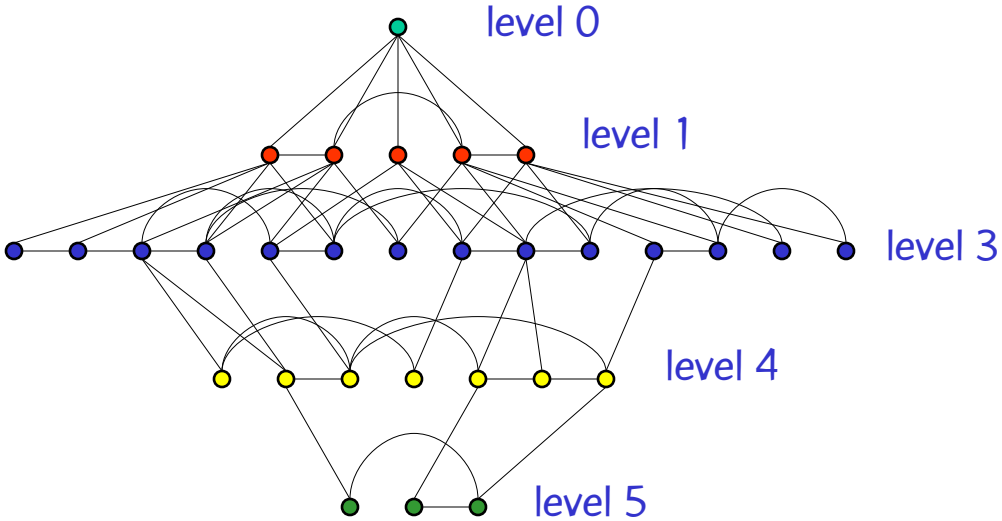
quasi-biclique

Clique case

- We illustrate the approach with the clique case.
  - We work on connected component of transmittal nodes (no cliques in sources or sinks)
  - Breadth first search decomposition
  - Peeling off vertices to focus in on large cliques
  - Finding cliques in a subgraph

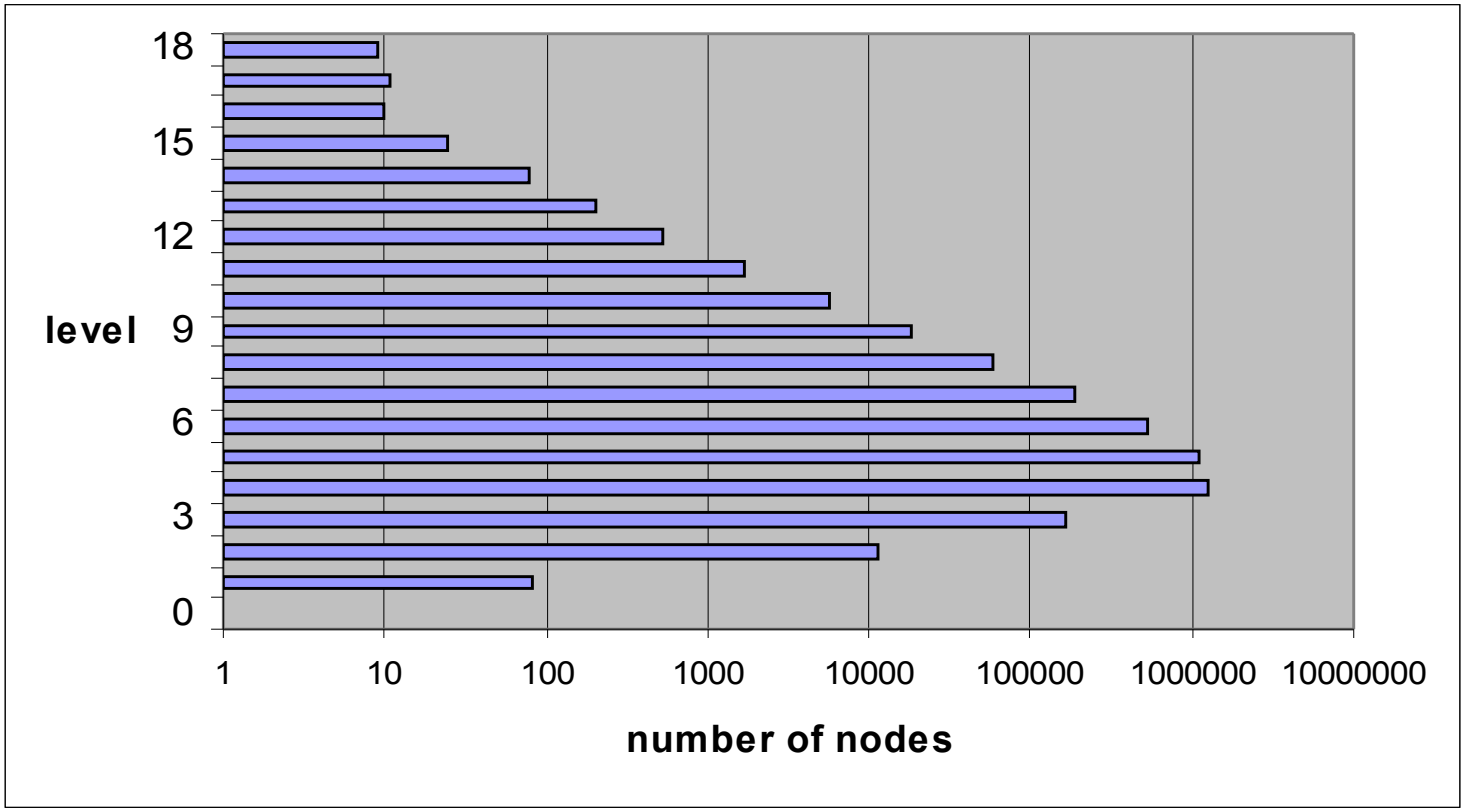
Breadth first search decomposition

- Given a graph G one can decompose its vertices into levels



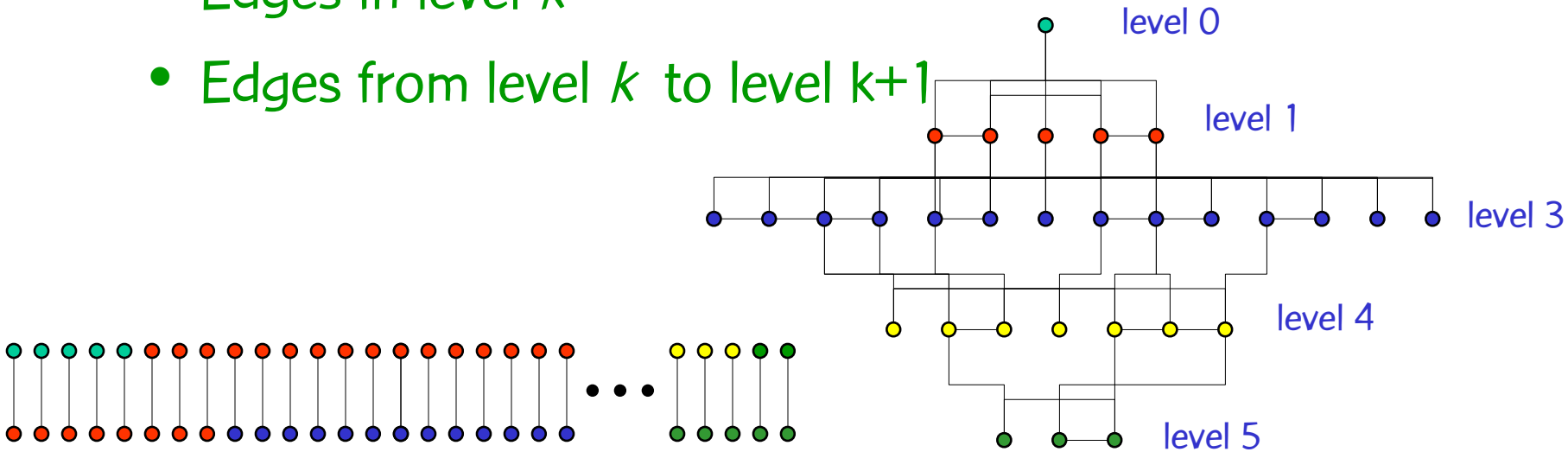
There are no cliques spanning three or more levels.

BFS: distribution of nodes per level



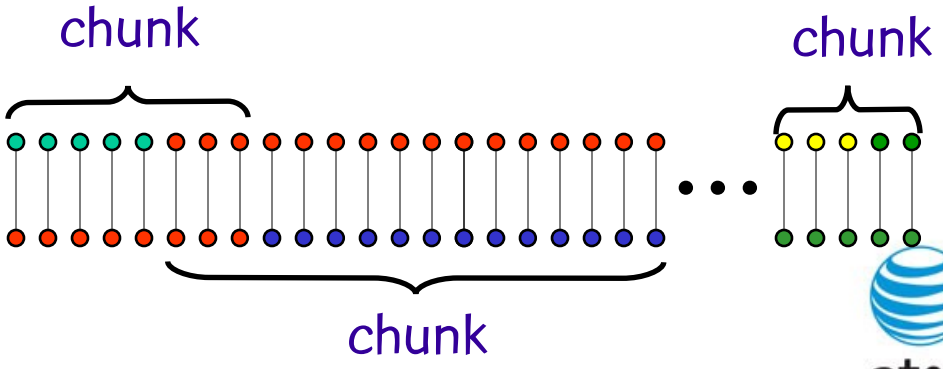
Edge ordering

- Use levels to order edges ($k = 0,1,2,\ldots$)
  - Edges in level k
  - Edges from level k to level k+1



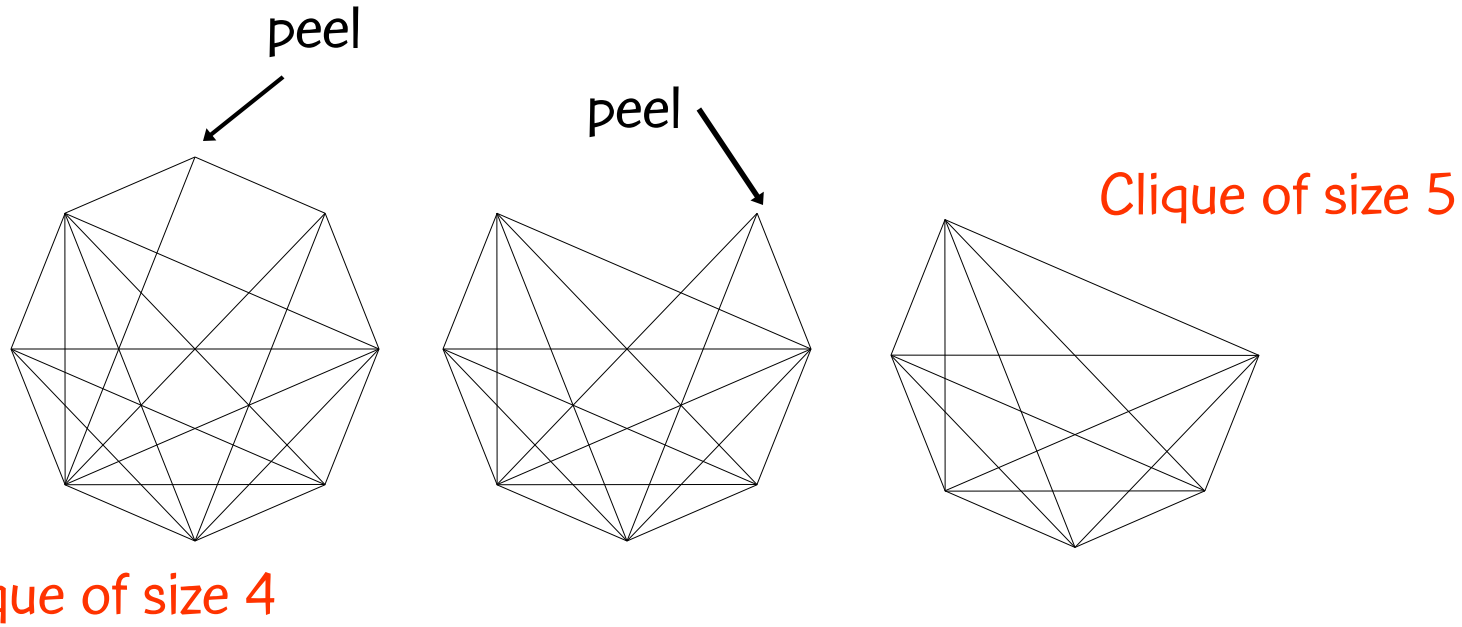
Chunking & peeling

- Start with all edges in E (set is massive)
- Repeat
  - Create a subgraph G' with one or more chunks
  - Find large clique (of size c') in G'
  - Peel from G all vertices v with $\deg(v) < c'$
  - $E = E(G)$



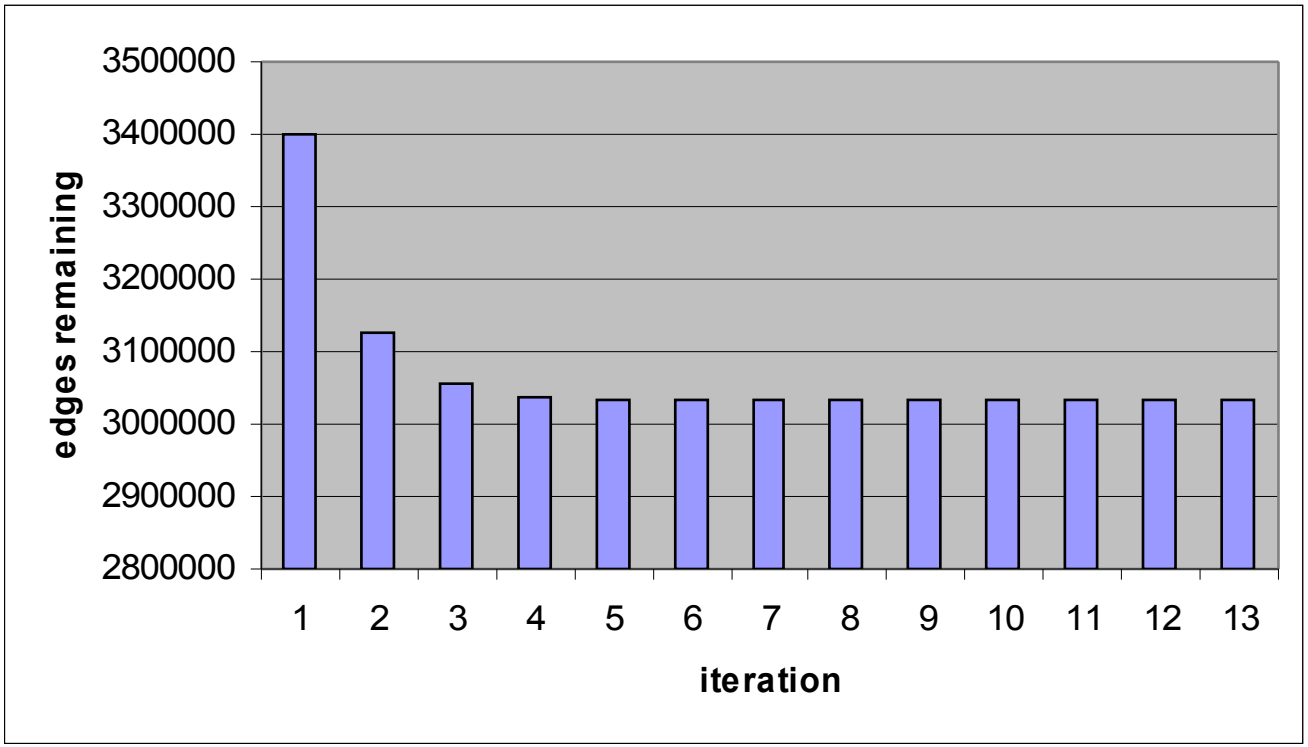
Peeling

- Peeling is applied recursively



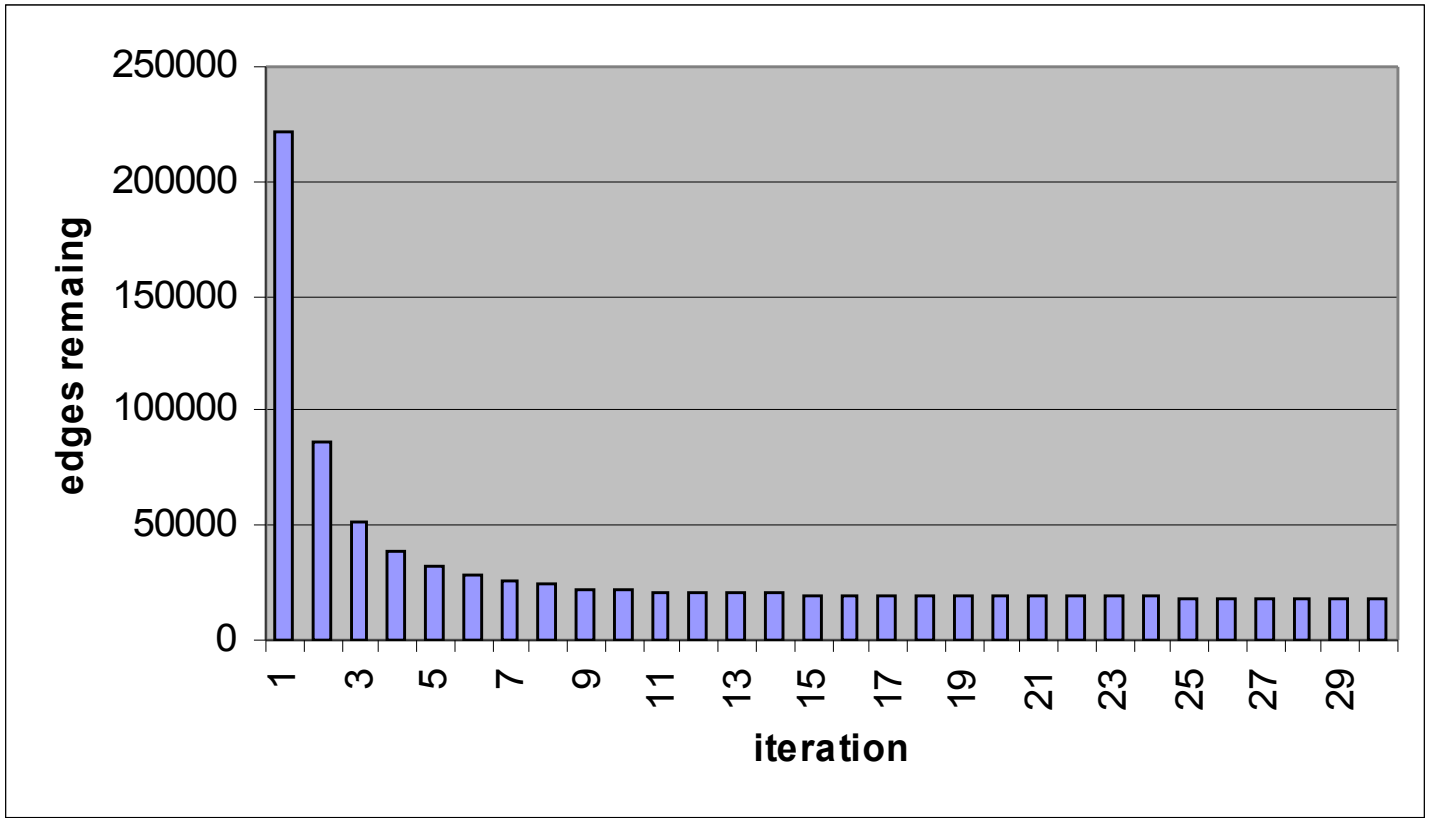
Peeling with degree = 2

reduction from 3.4 M edges to 3.0 M edges



Peeling with degree = 14

reduction from 3.0 M edges to 18.3 K edges

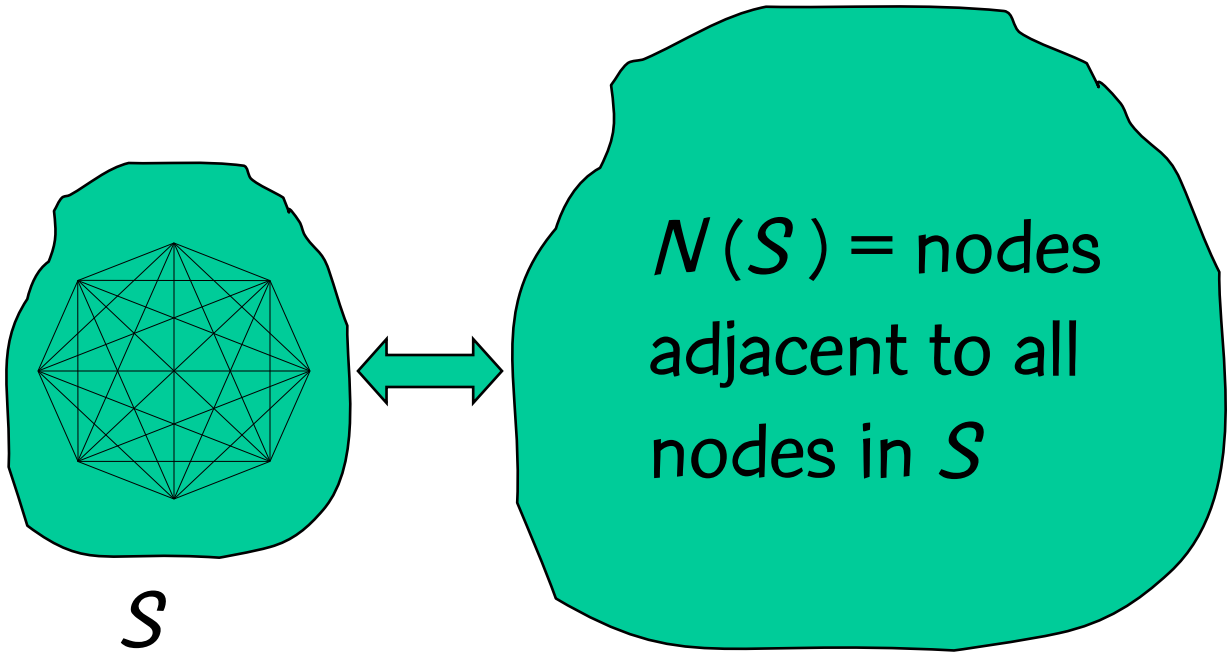


Finding cliques

- GRASP for max clique
  - multi-start
    - construct clique using randomized greedy algorithm
    - attempt to improve clique using 2-exchange local search
    - store all cliques found in construction & local search

Greedy vertex choice

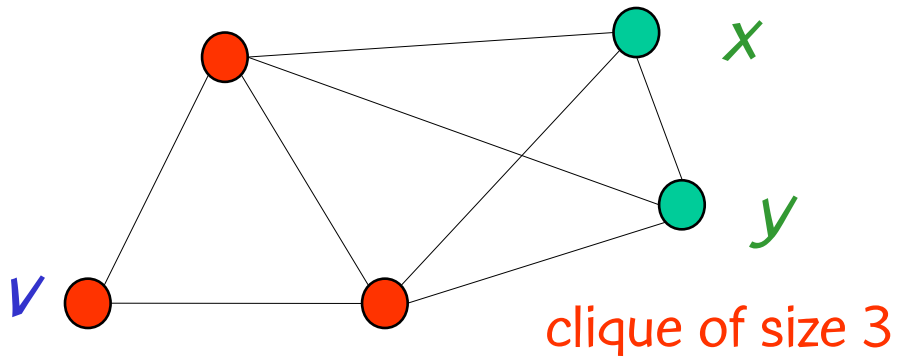
- Choose $v \in N(S)\}$ with max $deg_{N(S)}$ $\{v \in N(S)\}$.

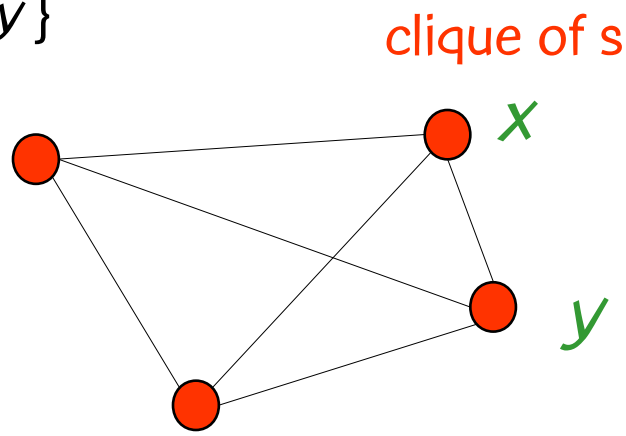


(2,1) exchange local search

- for each vertex v in clique S
  - while $\exists$ an edge $(x, y) \in E$ with x and y adjacent to all vertices in $S \setminus \{v\}$
    - remove v from S and add x and y to S:

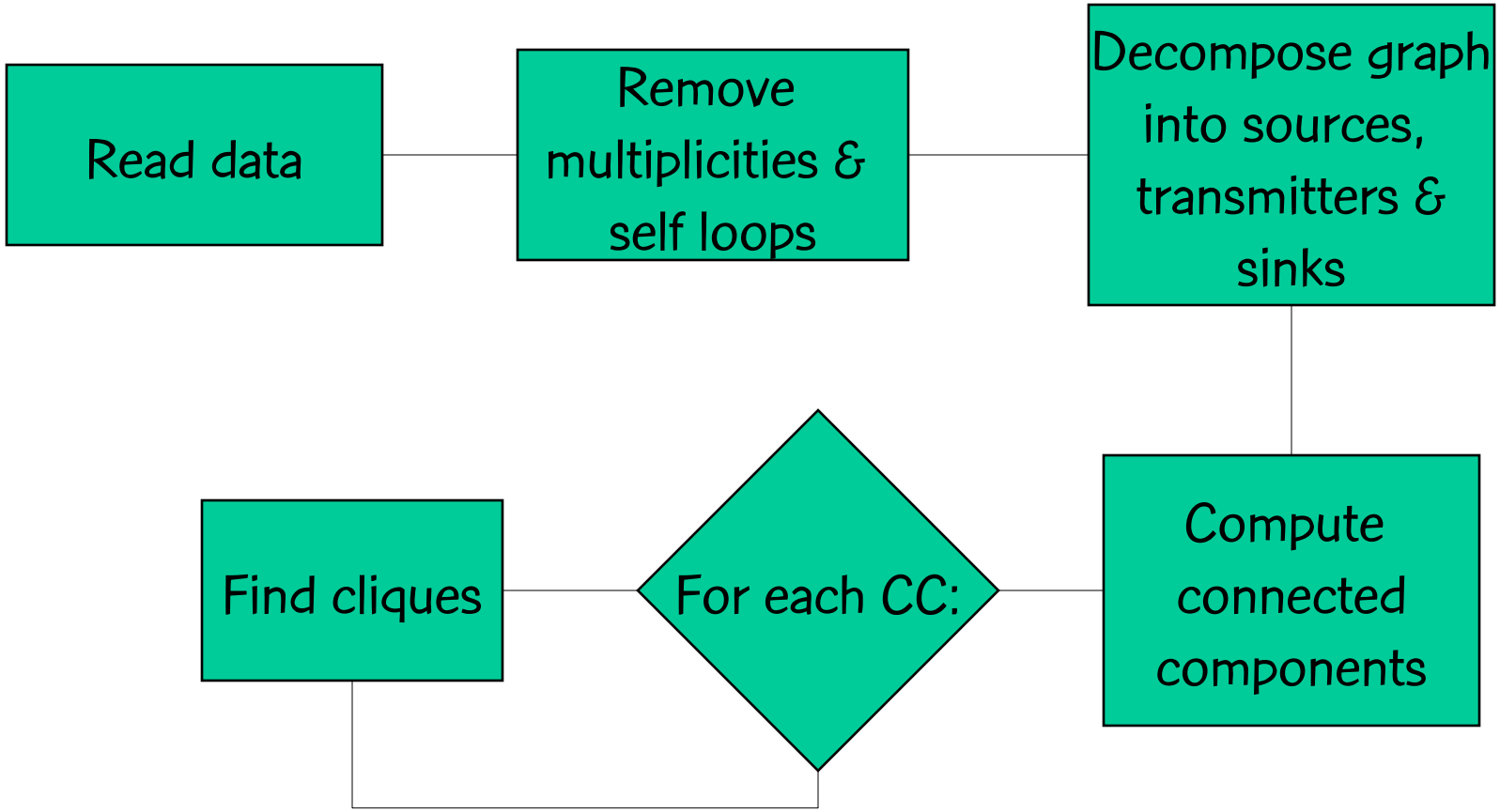
$$S = S \setminus \{v\} \cup \{x\} \cup \{y\}$$





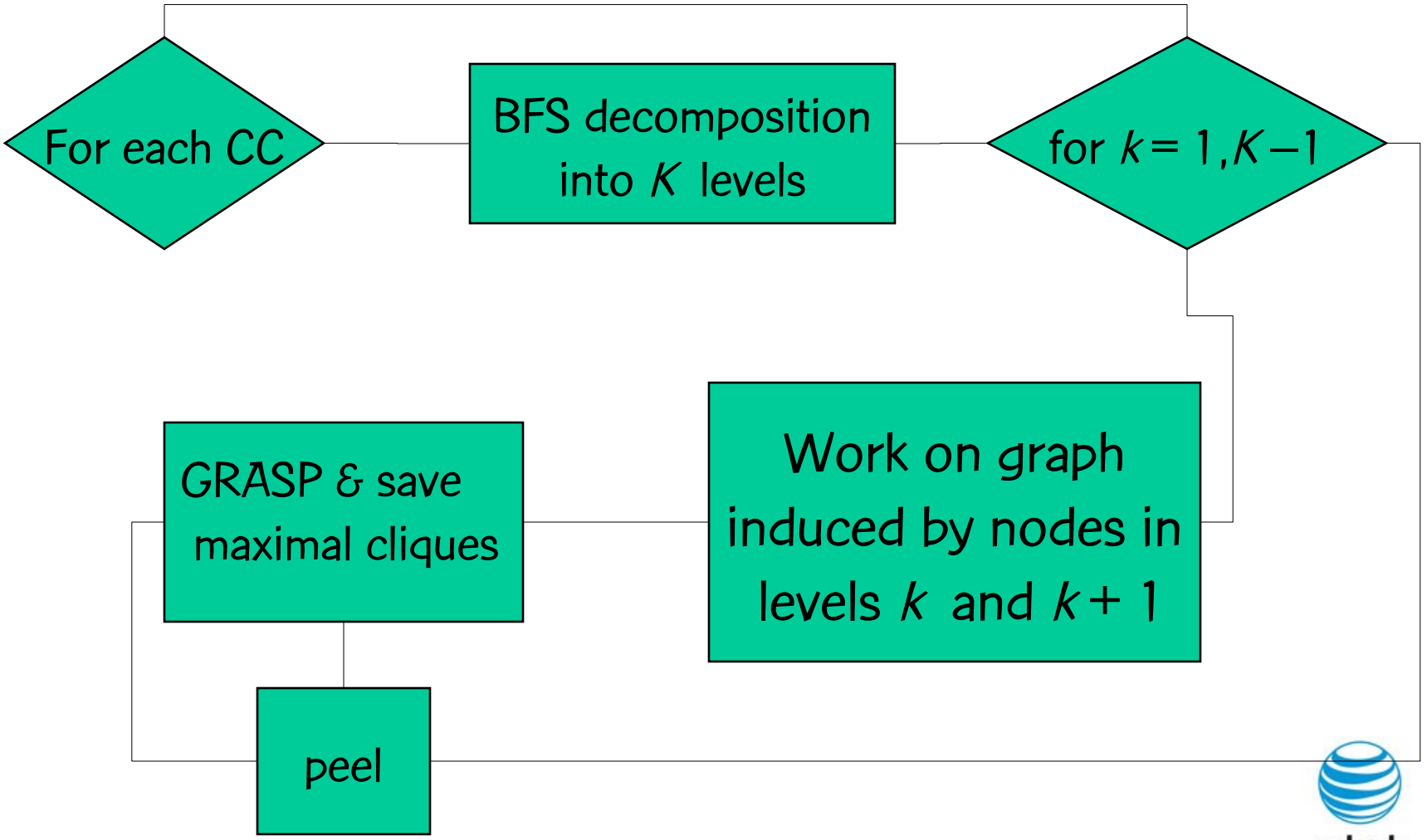
Software platform

external & semi-external memory algorithms



Software platform

computing cliques



Mining for cliques

examples

- 12 hours of calls
  - 53M nodes, 170M edges
  - 3.6M connected components (only 302K had more than three nodes)
    - 255 self loops, 2.7M pairs, and 598K triplets
  - Giant CC has 45M nodes
  - Found cliques of size up to 30 nodes in giant CC.
  - Found quasi-cliques of size 44 (90% density), 57 (80%), 65 (70%), and 98 (50%) in giant CC.

Concluding remarks

- We developed algorithms and systems for mining dense subgraphs is massive graphs.
- Subgraphs currently handled:
  - Cliques and quasi-cliques
  - Bicliques and quasi-bicliques
- We have explored data sets up to one week of calls, but aim to handle one year.
- Parallelization under way to speed up computations.

GRASP with evolutionary path-relinking for the antibandwidth problem

Tutorial given at the Spring School in Advances in Operations Research, Higher School of Economics
Nizhny Novgorod, Russia ✤ May 3, 2011

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

Joint work with A. Duarte, R. Martí, & R. Silva

Summary

- Antibandwidth
- Integer programming formulation
- GRASP construction
- Local search
- GRASP with evolutionary path-relinking
- Experimental results
- Concluding remarks

Paper

A. Duarte, R. Martí, , M.G.C. Resende, and R.M.A. Silva, "GRASP with path relinking heuristics for the antibandwidth problem," Networks, published online 22 December 2010.

Tech report:

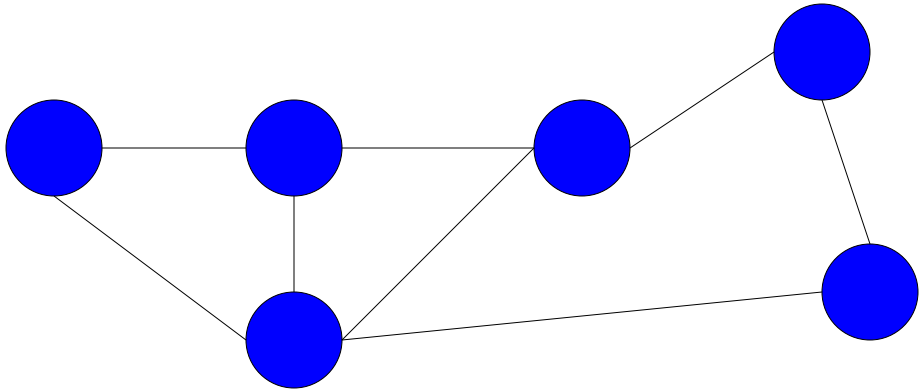
http://www2.research.att.com/~mgcr/doc/gpr-antiband.pdf

Antibandwidth problem

- Given an undirected graph $G = (V,E)$, where
  - V is the set of nodes ($n = |V|$)
  - E is the set of edges ($m = |E|$)
- A labeling f of V is a one-to-one mapping of $\{1,2,\ldots,n\}$ onto V.
  - Each vertex $v \in V$ has a unique label $f(v) \in \{1,2,\ldots,n\}$

Antibandwidth problem

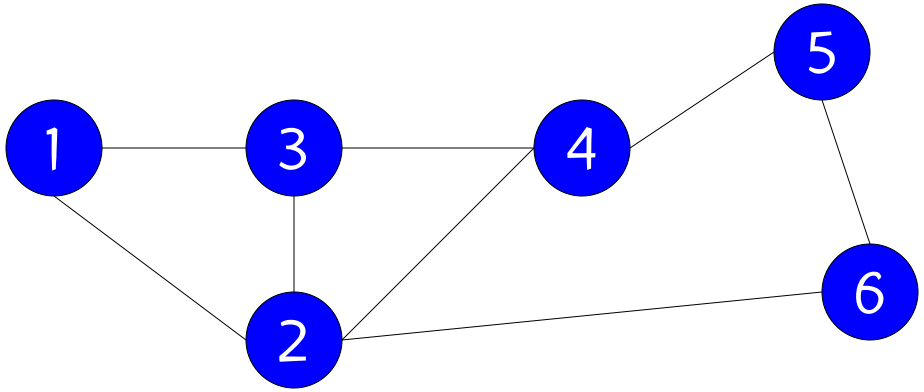
- Given an undirected graph $G = (V,E)$, where
  - V is the set of nodes ($n = |V|$)
  - E is the set of edges ($m = |E|$)
- A labeling f of V is a one-to-one mapping of $\{1,2,\ldots,n\}$ onto V.
  - Each vertex $v \in V$ has a unique label $f(v) \in \{1,2,\ldots,n\}$



undirected graph
$G = (V,E)$

Antibandwidth problem

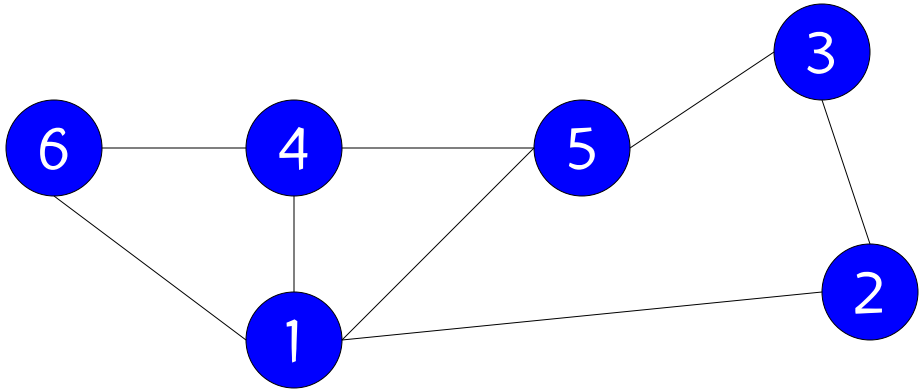
- Given an undirected graph $G = (V,E)$, where
  - V is the set of nodes ($n = |V|$)
  - E is the set of edges ($m = |E|$)
- A labeling f of V is a one-to-one mapping of $\{1,2,\ldots,n\}$ onto V.
  - Each vertex $v \in V$ has a unique label $f(v) \in \{1,2,\ldots,n\}$



undirected graph $G = (V,E)$ with a labeling f

Antibandwidth problem

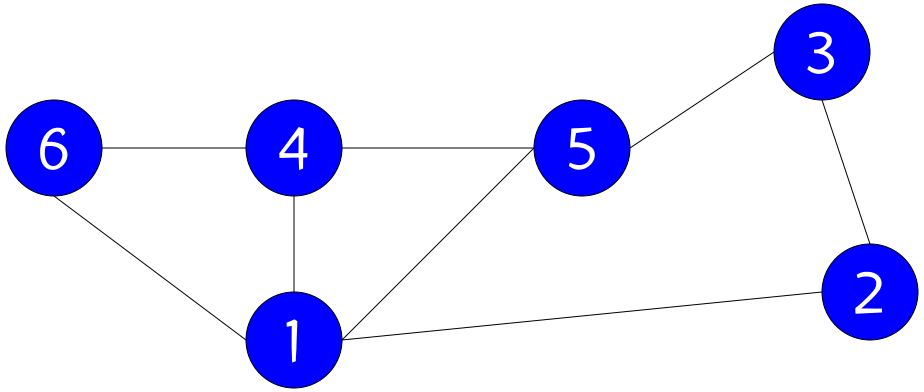
- Given an undirected graph $G = (V,E)$, where
  - V is the set of nodes ($n = |V|$)
  - E is the set of edges ($m = |E|$)
- A labeling f of V is a one-to-one mapping of $\{1,2,\ldots,n\}$ onto V.
  - Each vertex $v \in V$ has a unique label $f(v) \in \{1,2,\ldots,n\}$



undirected graph $G = (V,E)$ with another labeling f

Antibandwidth problem

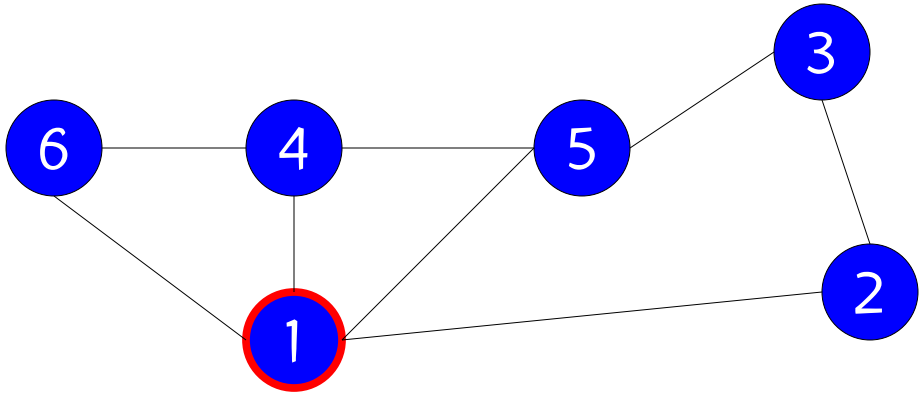
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between $f(v)$ and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where $N(v)$ is the set of nodes adjacent to v



undirected graph $G = (V,E)$ with a labeling f

Antibandwidth problem

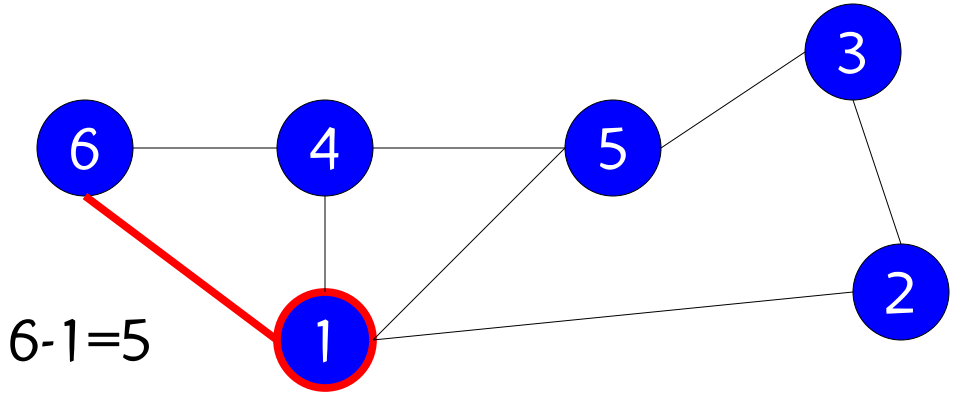
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between $f(v)$ and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where $N(v)$ is the set of nodes adjacent to v



undirected graph $G = (V,E)$ with a labeling f

Antibandwidth problem

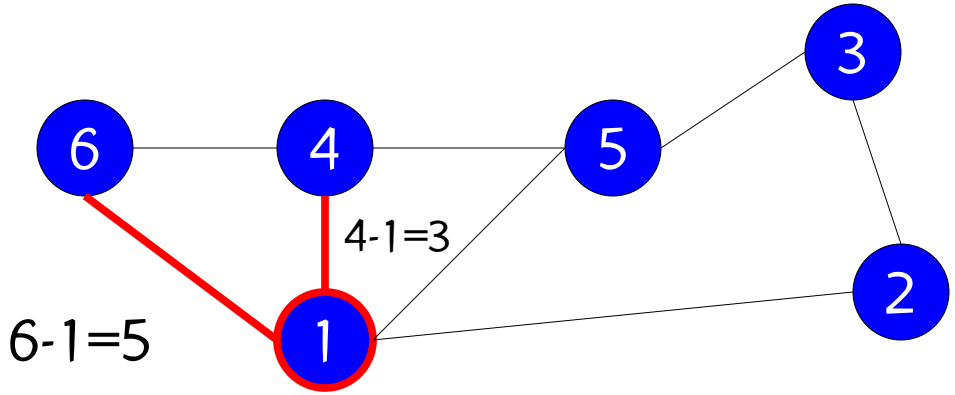
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between $f(v)$ and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where $N(v)$ is the set of nodes adjacent to v



undirected graph $G = (V,E)$ with a labeling f

Antibandwidth problem

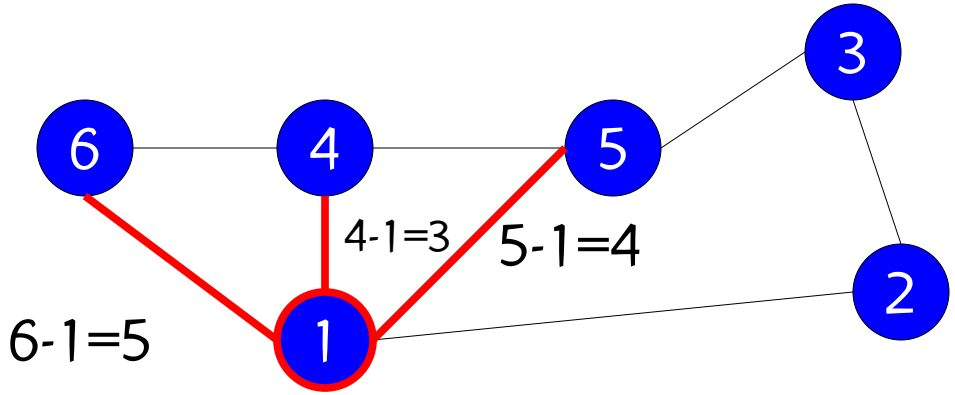
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between $f(v)$ and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where $N(v)$ is the set of nodes adjacent to v



undirected graph $G = (V,E)$ with a labeling f

Antibandwidth problem

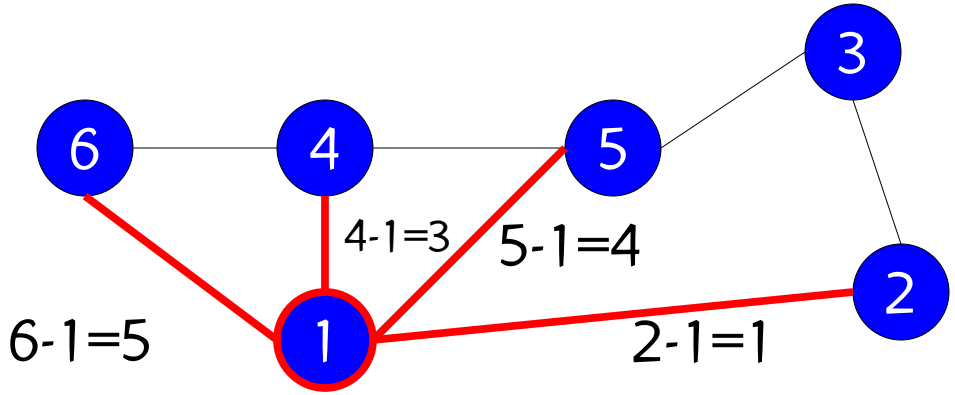
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between f(v) and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where N(v) is the set of nodes adjacent to v



undirected graph G = (V,E) with a labeling f

Antibandwidth problem

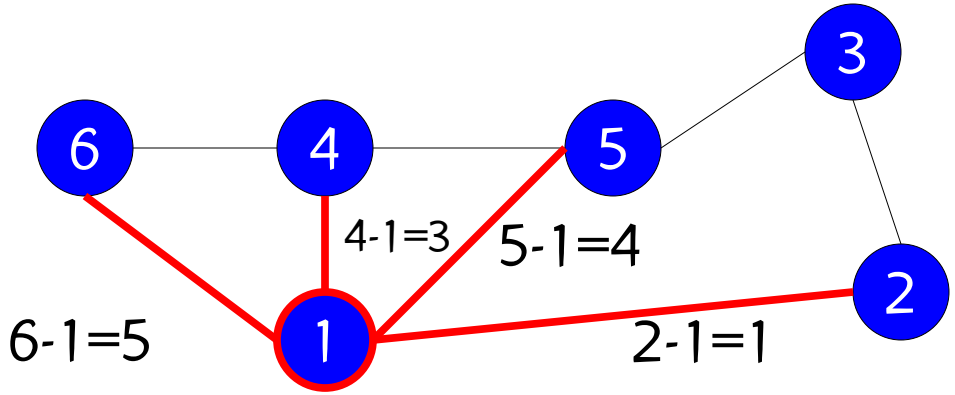
- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between f(v) and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where N(v) is the set of nodes adjacent to v



undirected graph G = (V,E) with a labeling f

Antibandwidth problem

- Given G and f, the antibandwidth $AB_f(v)$ of node v is smallest difference between f(v) and the labels of all of the nodes adjacent to v, i.e.
  - $AB_f(v) = \min \{ |f(v) - f(u)| : u \in N(v) \}$
  - where N(v) is the set of nodes adjacent to v

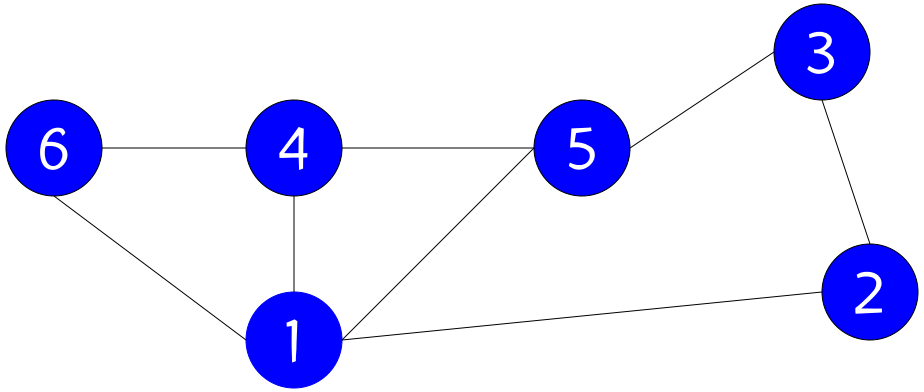


undirected graph G = (V,E) with a labeling f

$AB_f(1) = \min\{5,3,4,1\} = 1$

Antibandwidth problem

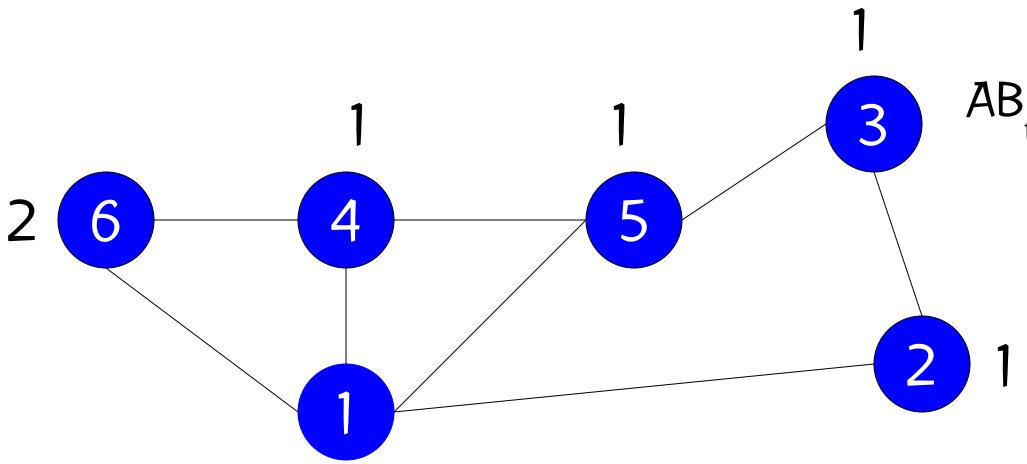
- Given G and f, the antibandwidth $AB_f(G)$ of f is smallest antibandwidth over all nodes in V, i.e.
  - $AB_f(G) = \min \{ AB_f(v) : v \in V \}$
  - where N(v) is the set of nodes adjacent to v



undirected graph G = (V,E) with a labeling f

Antibandwidth problem

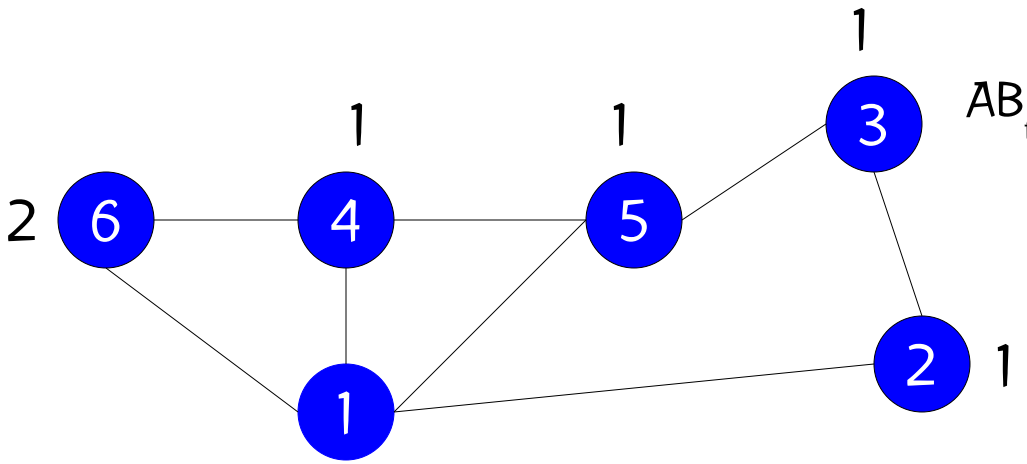
- Given G and f, the antibandwidth $AB_f(G)$ of f is smallest antibandwidth over all nodes in V, i.e.
  - $AB_f(G) = \min \{ AB_f(v) : v \in V \}$
  - where N(v) is the set of nodes adjacent to v



$AB_f(G) = \min \{ 1, 1, 1, 1, 2 \} = 1$

Antibandwidth problem

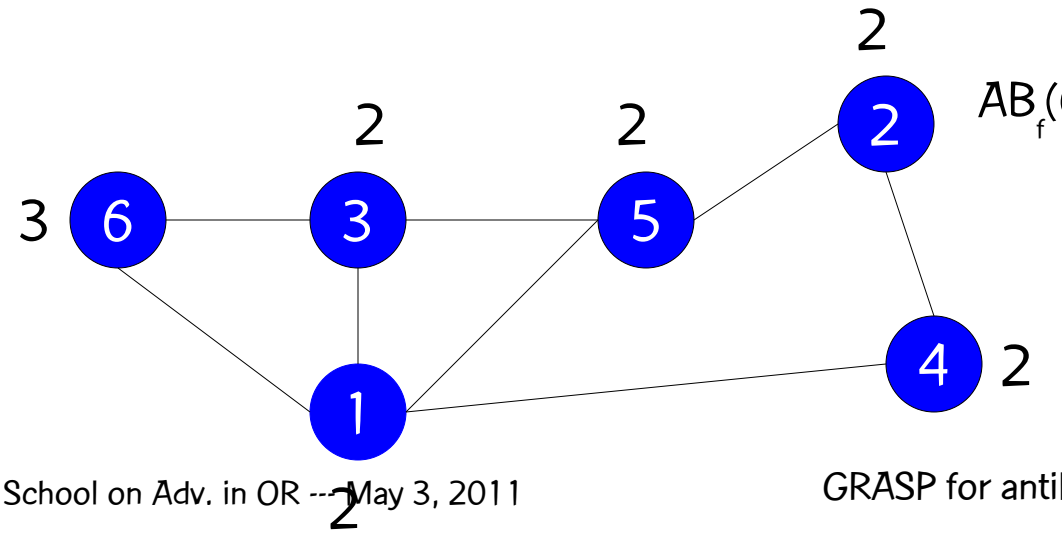
- Given G, the antibandwidth $AB(G)$ of G is largest antibandwidth over all possible labelings, i.e.
  - $AB(G) = \max \{ AB_f(G) : f \in \Pi_n \}$
  - where Π_n is the set of all permutations of {1, 2, ..., n}



$AB_f(G) = \min \{ 1, 1, 1, 1, 2 \} = 1$

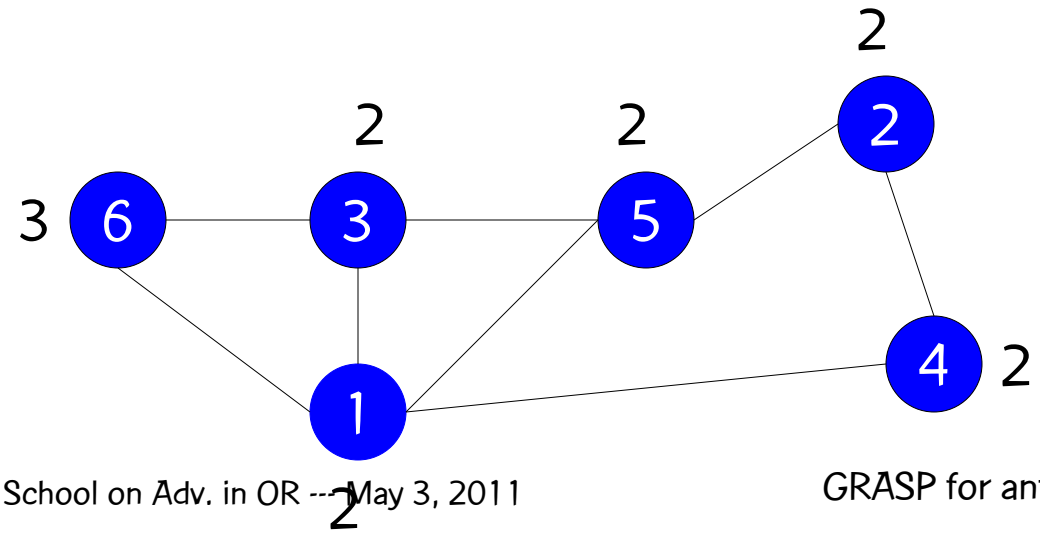
Antibandwidth problem

- Given G, the antibandwidth $AB(G)$ of G is largest antibandwidth over all possible labelings, i.e.
  - $AB(G) = \max \{ AB_f(G) : f \in \Pi_n \}$
  - where Π_n is the set of all permutations of {1, 2, ..., n}



Antibandwidth problem

- Given G, the antibandwidth $AB(G)$ of G is largest antibandwidth over all possible labelings, i.e.
  - $AB(G) = \max \{ AB_f(G) : f \in \Pi_n \}$
  - where Π_n is the set of all permutations of {1, 2, ..., n}



$AB(G) = 2$

Antibandwidth problem

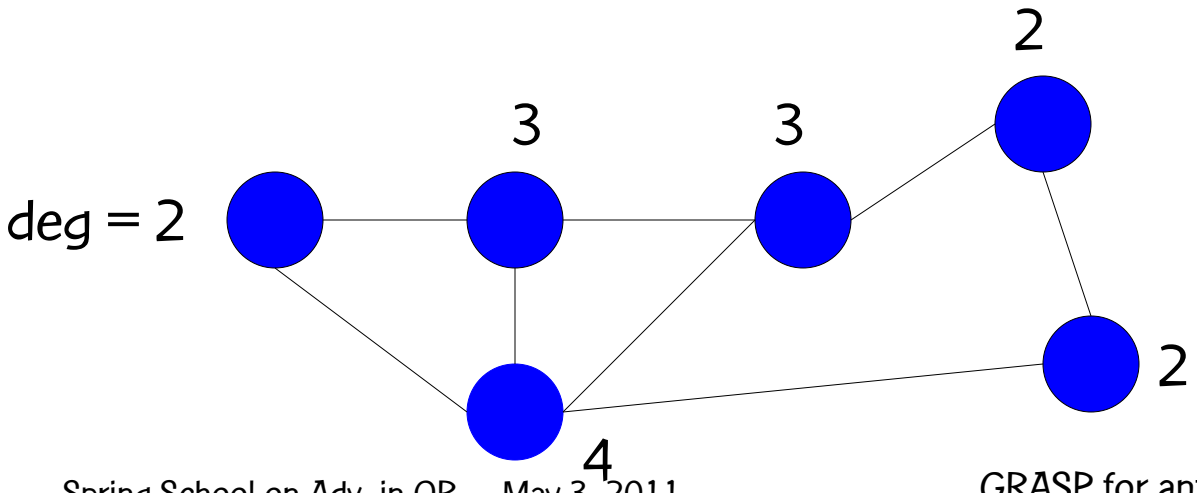
- NP-hard (Leung et al., 1984)
- Special cases can be solved in polynomial time, e.g. complements of intervals, arborescent comparability, and on threshold graphs (Raspaud et al., 2008)

Antibandwidth problem

- Yixum and Jinjiang (2003) proposed the upper bound: min { floor($(n - \delta + 1)/2$), $n - \Delta$ }, where
  - δ is the smallest degree over all $v \in V$
  - Δ is the largest degree over all $v \in V$

Antibandwidth problem

- Yixum and Jinjiang (2003) proposed the upper bound: min { floor($(n - \delta + 1)/2$), $n - \Delta$ }, where
  - δ is the smallest degree over all $v \in V$
  - Δ is the largest degree over all $v \in V$



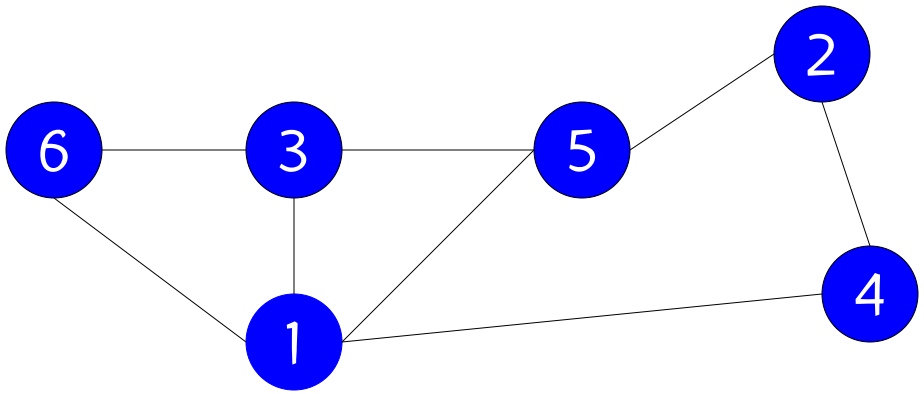
$\delta = 2$

$\Delta = 4$

UB = min{ 2, 2 } = 2

Antibandwidth problem

- Yixum and Jinjiang (2003) proposed the upper bound: min { floor((n – δ + 1)/2), n – Δ }, where
  - δ is the smallest degree over all v ∈ V
  - Δ is the largest degree over all v ∈ V



$\delta = 2$

$\Delta = 4$

UB = min{ 2, 2 } = 2

$AB_f(G) = 2$ is opt

$AB(G) = 2$

Antibandwidth problem: IP formulation

- Let x_{ik} be a binary variable that takes on the value 1 if and only if $f(i) = k$, i.e. node i takes label k.
- Define $l_i = f(i) \in \{1, 2, \ldots, n\}$ to be the label of node i.
- Finally, let $b = AB_f(G) = \min\{|f(u) - f(v)| : (u, v) \in E\}$ be the antibandwidth of labeling f.
- In the antibandwidth problem we want to determine the labeling f^* that maximizes b.

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - One label is assigned to each node:

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \forall\, k = 1, \ldots, n$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - One node is assigned to each label:

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \forall\, i = 1,\ldots,n$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - Each label l_i is a function of the binary variables x_{ik}:

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \quad \forall\, i = 1, \ldots, n$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - Require that $b = \min\{|l_i - l_j| : (i,j) \in E\}$:

$$b \leq |l_i - l_j|, (i,j) \in E$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - Binary variables x_{ik} can only take values 0 or 1 :

$$x_{ik} \in \{0,1\}, \forall\ i,k = 1,\ldots,n$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints:
  - Labels l_i can only take on values {1, ..., n} :

$$l_i \in \{1, 2, \ldots, n\}, \forall\ i = 1, \ldots, n$$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints $b \le |l_i - l_j|, (i,j) \in E$ are nonlinear:
  - If $l_i \ge l_j$ then $b \le l_i - l_j$
  - Otherwise, $b \le -(l_i - l_j)$
  - Introduce two binary variables to indicate case:
    - If $l_i \ge l_j$ then $y_{ij} = 0$ and $z_{ij} = 1$
    - Otherwise, $y_{ij} = 1$ and $z_{ij} = 0$

Antibandwidth problem: IP formulation

- Objective: maximize b
- Constraints $b \le |l_i - l_j|, (i,j) \in E$ become:

$$b - (l_i - l_j) \le 2y_{ij}(n-1), \forall (i,j) \in E$$

$$b + (l_i - l_j) \le 2z_{ij}(n-1), \forall (i,j) \in E$$

$$y_{ij} + z_{ij} = 1, \forall (i,j) \in E$$

$$b \ge 1$$

$l_i \ge l_j$ then $y_{ij} = 0$ $z_{ij} = 1$

$l_i < l_j$ then $y_{ij} = 1$ $z_{ij} = 0$

Antibandwidth problem: IP formulation

$$\max \quad b$$

$$b \geq 1$$

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \forall\, k = 1, \ldots, n$$

$$x_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \forall\, i = 1, \ldots, n$$

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \quad \forall\, i = 1, \ldots, n$$

$$l_i \in \{1, 2, \ldots n\}, \forall\, i = 1, 2, \ldots n$$

$$b - (l_i - l_j) \leq 2 y_{ij}(n-1), \forall\, (i,j) \in E$$

$$y_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

$$b + (l_i - l_j) \leq 2 z_{ij}(n-1), \forall\, (i,j) \in E$$

$$z_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

$$y_{ij} + z_{ij} = 1, \forall\, (i,j) \in E$$

Antibandwidth problem: IP formulation

$$\max \quad b$$

$$b \geq 1$$

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \forall\, k = 1, \ldots, n$$

$$x_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \forall\, i = 1, \ldots, n$$

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \quad \forall\, i = 1, \ldots, n$$

$$l_i \in \{1, 2, \ldots n\}, \forall\, i = 1, 2, \ldots n$$

$$b - (l_i - l_j) \leq 2y_{ij}(n-1), \forall\, (i,j) \in E$$

$$y_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

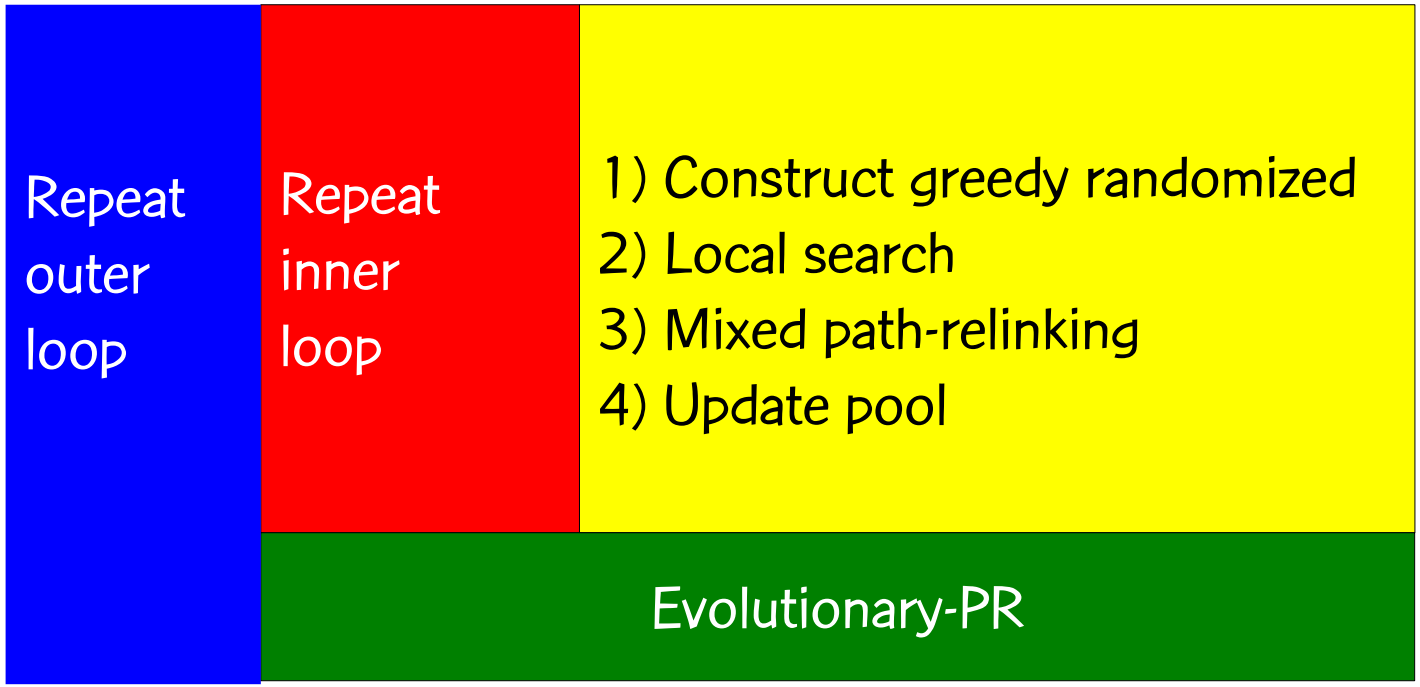
$$b + (l_i - l_j) \leq 2z_{ij}(n-1), \forall\, (i,j) \in E$$

$$z_{ik} \in \{0,1\}, \forall\, (i,k) \in E$$

$$y_{ij} + z_{ij} = 1, \forall\, (i,j) \in E$$

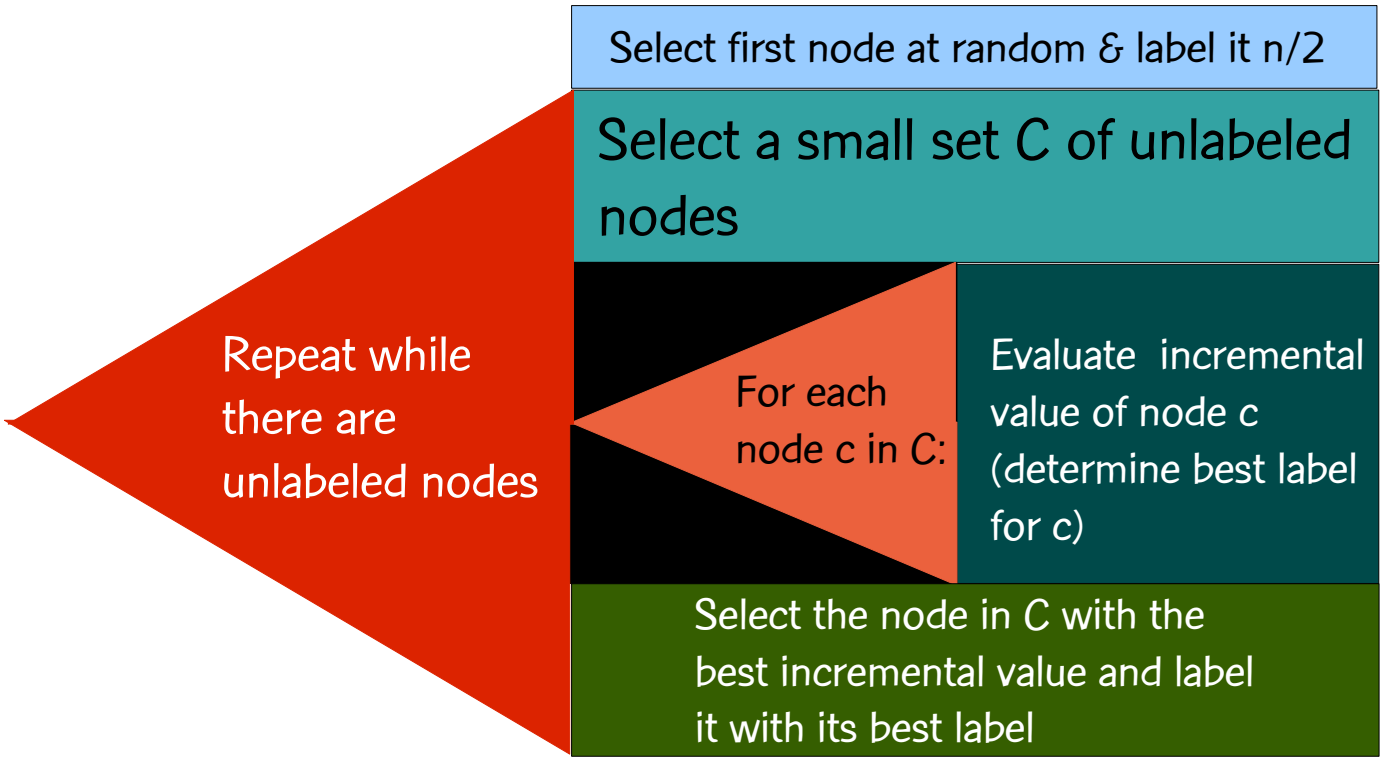
IP has $O(n^2)$ variables and $O(n^2)$ constraints

GRASP with evolutionary path-relinking



GRASP construction procedure

- We use the sampled greedy construction scheme of R. & Werneck (2004)



GRASP construction procedure:

Selecting a small set C of unlabeled nodes

- The set CL of candidate nodes is made up of nodes adjacent to labeled nodes
- The small set C of candidate nodes is a set of $\alpha \times |CL|$ randomly sampled nodes from CL, where α is a random real number $\in [0,1]$
- The value of α does not change during construction
- Values of $\alpha \approx 1$ makes sampled greedy resemble a greedy construction, while values of $\alpha \approx 0$ makes it behave like a random construction

GRASP construction procedure:

Determine the best label for a candidate node c

- Let $\breve{l}_c$ and $\hat{l}_c$ be, respectively, the smallest and largest assigned labels to the the nodes adjacent to c
- The "best" label for c is

$$l^* = \text{argmax}\{\min(|l - \hat{l}_c|, |l - \breve{l}_c|): l = 1, \ldots, n\}$$

- The closest available label to l^* is assigned to c

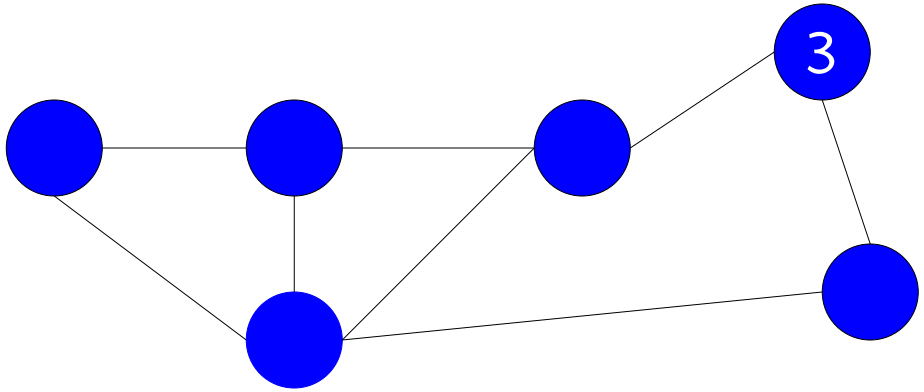
GRASP construction procedure:

Determine the best label for a candidate node

- Let $\breve{l}_c$ and $\hat{l}_c$ be, respectively, the smallest and largest assigned labels to the the nodes adjacent to c
- The "best" label for c is

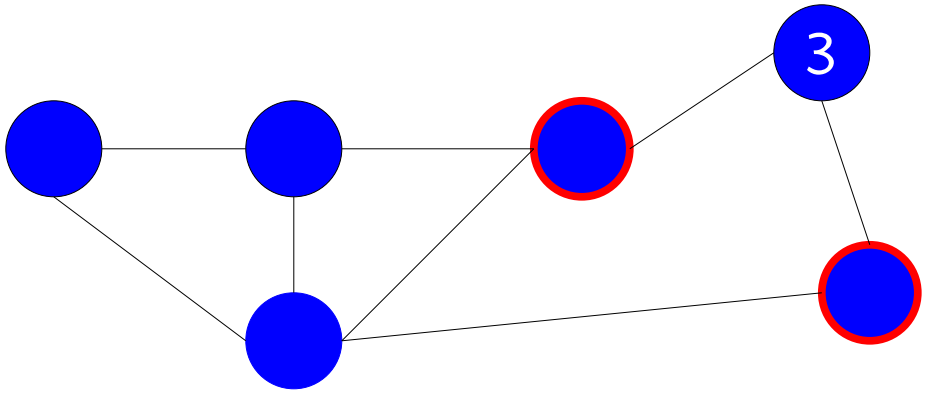
$$l^* = \text{argmax}\{\min(|l - \hat{l}_u|, |l - \breve{l}_u|): l = 1, \ldots, n\}$$

- The closest available label to l^* is assigned to c



Choose first node at random and label it n/2 = 6/2 = 3

GRASP construction procedure

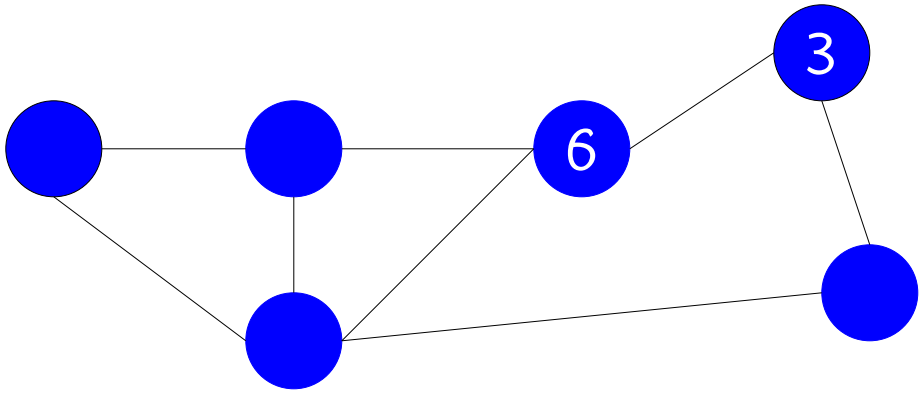


Best label for both candidates is 6.

Label one of the nodes with a 6

Candidate node

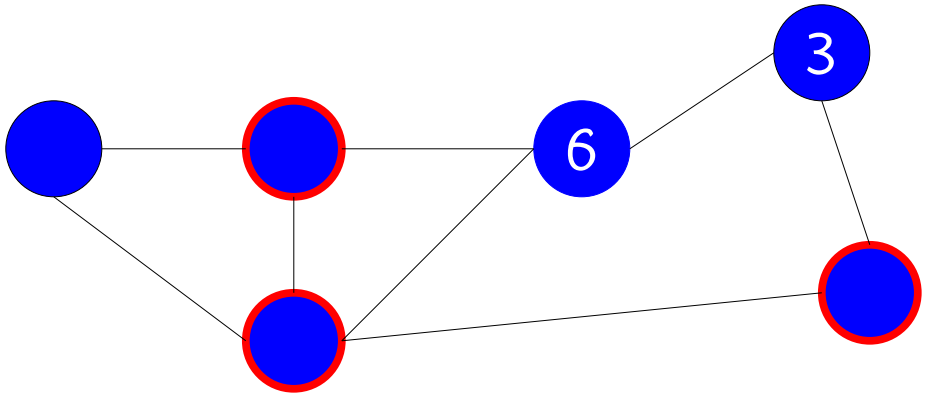
GRASP construction procedure



Best label for both candidates on left is 1 and on right is 6.

Label node on right with a 5 (closest available label to 6)

GRASP construction procedure

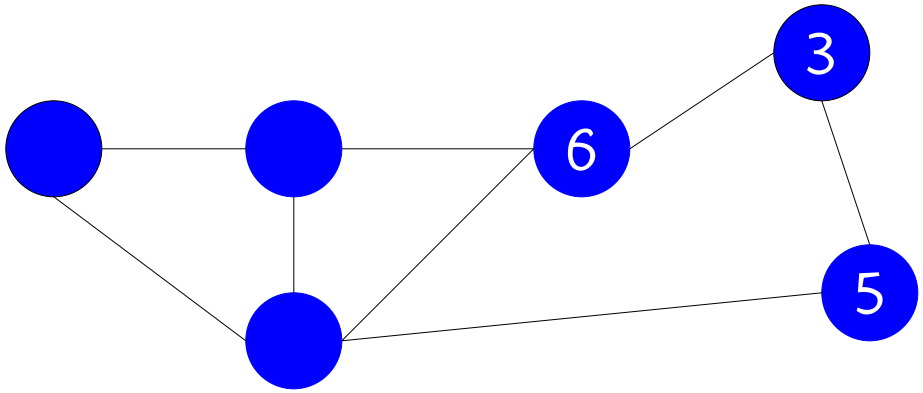


Best label for both candidates on left is 1 and on right is 6.

Label node on right with a 5 (closest available label to 6)

Candidate node

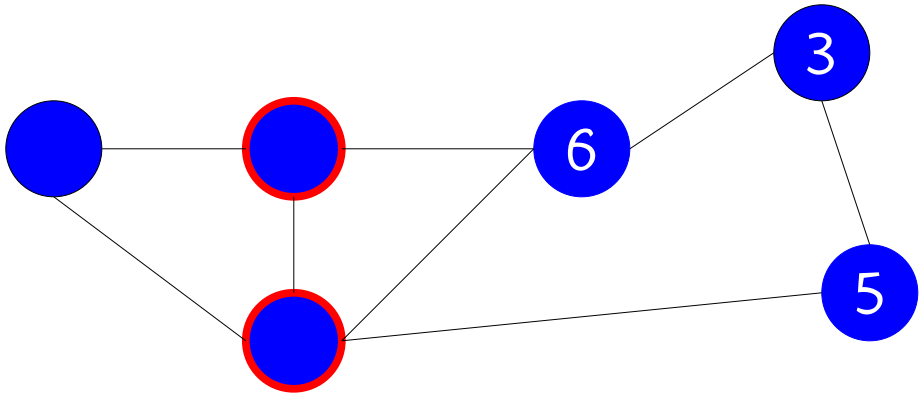
GRASP construction procedure



Best label for both candidates on left is 1 and on right is 6.

Label node on right with a 5 (closest available label to 6)

GRASP construction procedure

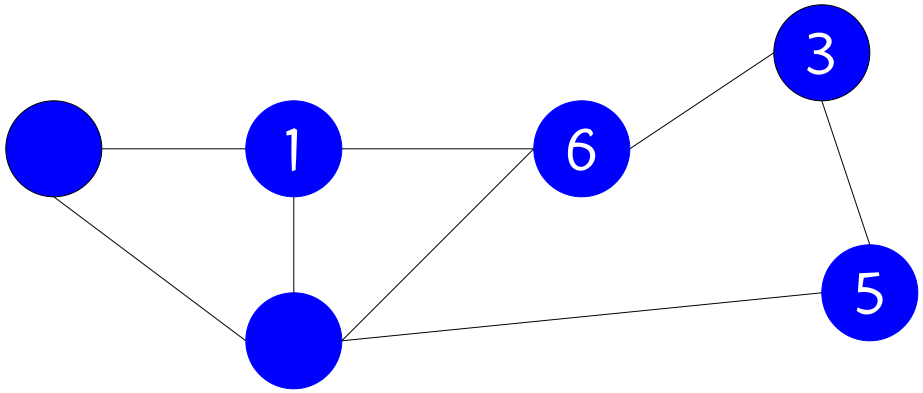


Best label for both candidates is 1.

Label node on top with a 1.

Candidate node

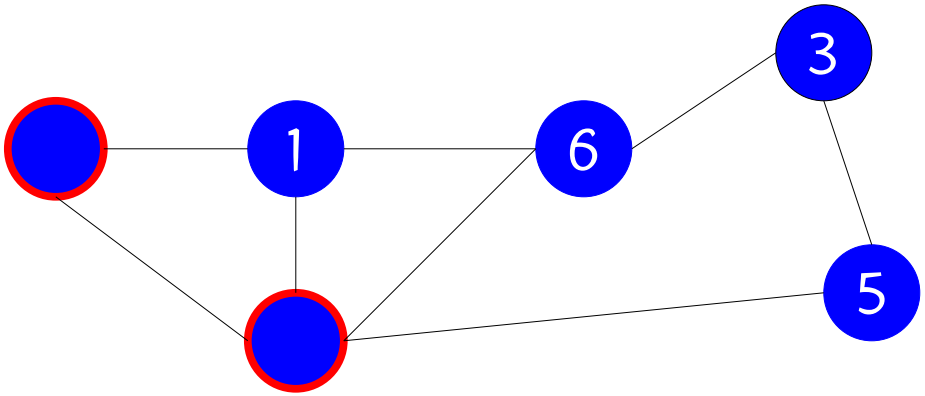
GRASP construction procedure



Best label for both candidates is 1.

Label node on top with a 1.

GRASP construction procedure

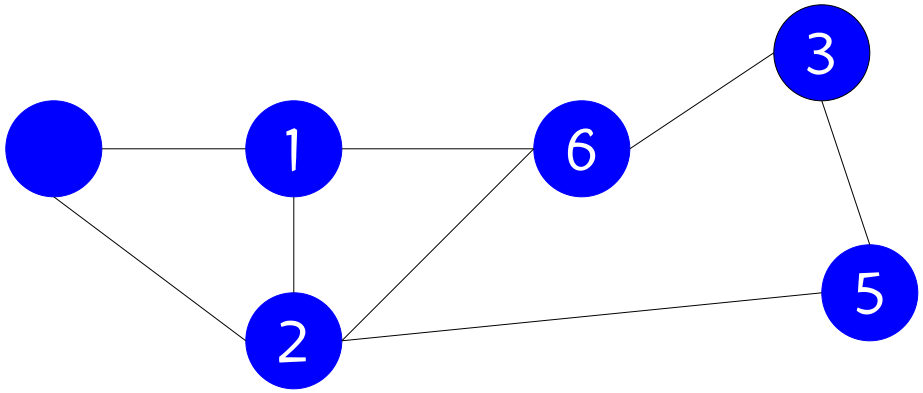


Best label for node on left is 6 and for node on bottom is 3.

Label node on bottom with a 2.

Candidate node

GRASP construction procedure

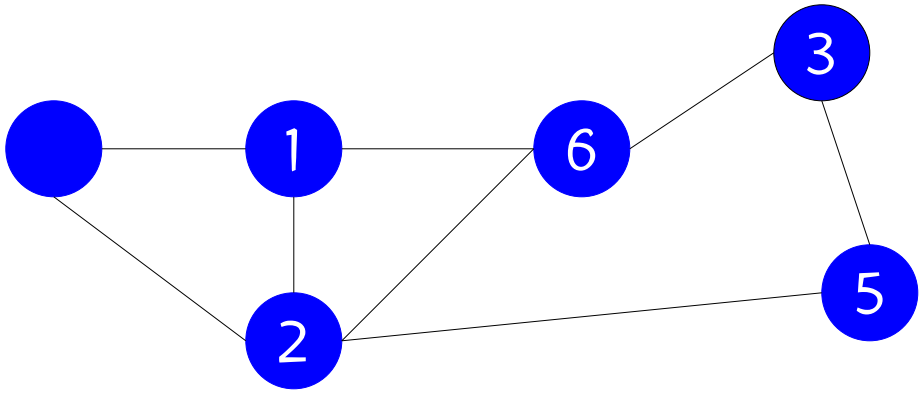


Best label for node on left is 6 and for node on bottom is 2 or 3.

Label node on bottom with a 2.

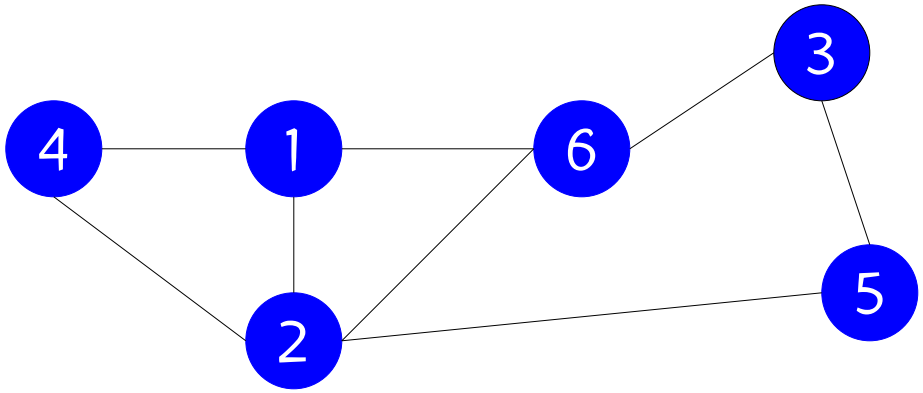
Candidate node

GRASP construction procedure



Remaining node must be labeled with a 4.

GRASP construction procedure



Remaining node must be labeled with a 4.

$AB_f(G) = 1$

Local Search

GRASP local search procedure

- Antibandwidth problem has a flat landscape: many solutions have same cost
- For a given labeling f, there may be multiple nodes u such that $AB_f(u) = AB_f(G)$
- Therefore, in local search, a move (swap of labels of a pair of nodes) that improves $AB_f(u)$ does not necessarily change the value of the solution $AB_f(G)$

GRASP local search procedure

- Nodes u with unequal $AB_f(u)$ values but that are close to $AB_f(G)$ can be crucial in future iterations (swaps) of the local search, even though they cannot affect the value of the current labeling
- Define the set of crucial vertices of a labeling f to be

$$C(f) = \{u \in V : AB_f(u) \leq \beta \cdot AB_f(G)\}$$

$$(1 \leq \beta \leq 2)$$

GRASP local search procedure

- Given a labeling f, operator move(u,v) assigns the label $f(u)$ to node v and the label $f(v)$ to node u, resulting in a new labeling f'
- Local search scans nodes u in $C(f)$, changing their labels to increase their antibandwidths
- Let $\breve{l}_u$ and $\hat{l}_u$ be, respectively, the smallest and largest assigned labels to the the nodes adjacent to u
- The best label for u is

$$l_u^* = \text{argmax}\{\min(|l - \hat{l}_u|, |l - \breve{l}_u|): l = 1, \ldots, n\}$$

GRASP local search procedure

- Once we determine the best label l^* for u, we determine the node v with this label to evaluate move(u,v)
- We know that label l^* is good for u, but we need to determine whether label $f(u)$ is good for node v
- We extend the search for a good label for u not only to node v with label l^*, but also to nodes with labels close to l^*
- The set $N'(u)$ of suitable swapping nodes for u depends on the relationship between l^*, $\breve{l}_u$, and $\hat{l}_u$

GRASP local search procedure

- If $l_u^* < \breve{l}_u$ then $N'(u) = \{v \in V : l_u^* \leq f(v) \leq \breve{l}_u - AB_f(G)\}$
- If $l_u^* > \hat{l}_u$ then $N'(u) = \{v \in V : \hat{l}_u + AB_f(G) \leq f(v) \leq l_u^*\}$
- If $\breve{l}_u \leq l_u^* \leq \hat{l}_u$ then

$$N'(u) = \{v \in V : \hat{l}_u + AB_f(G) \leq f(v) \leq \hat{l}_u - AB_f(G)\}$$

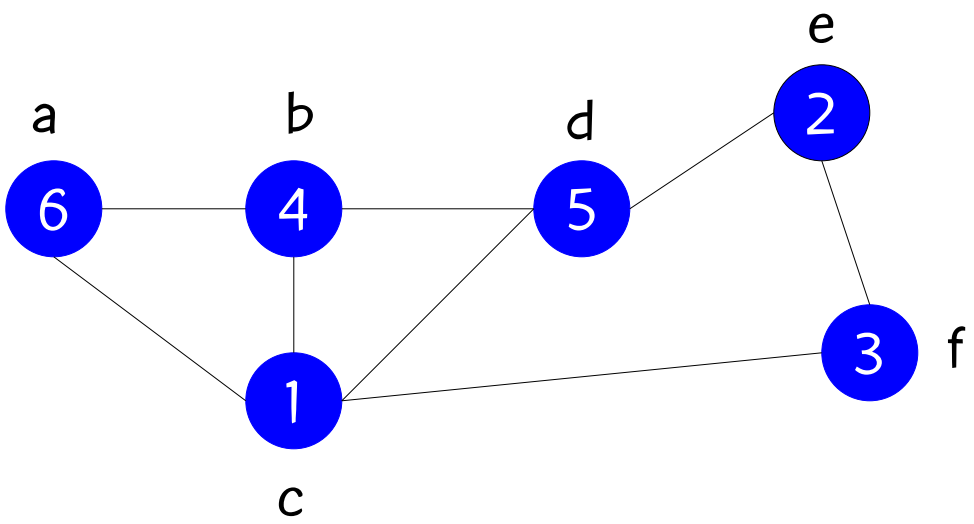
GRASP local search procedure

- If $l_u^* < \breve{l}_u$ then $N'(u) = \{v \in V : l_u^* \leq f(v) \leq \breve{l}_u - AB_f(G)\}$
- If $l_u^* > \hat{l}_u$ then $N'(u) = \{v \in V : \hat{l}_u + AB_f(G) \leq f(v) \leq l_u^*\}$
- If $\breve{l}_u \leq l_u^* \leq \hat{l}_u$ then

$$N'(u) = \{v \in V : \breve{l}_u + AB_f(G) \leq f(v) \leq \hat{l}_u - AB_f(G)\}$$

If N'(u) = ∅, then $AB_f(u)$ cannot be increased in a single step by changing the current label of u.

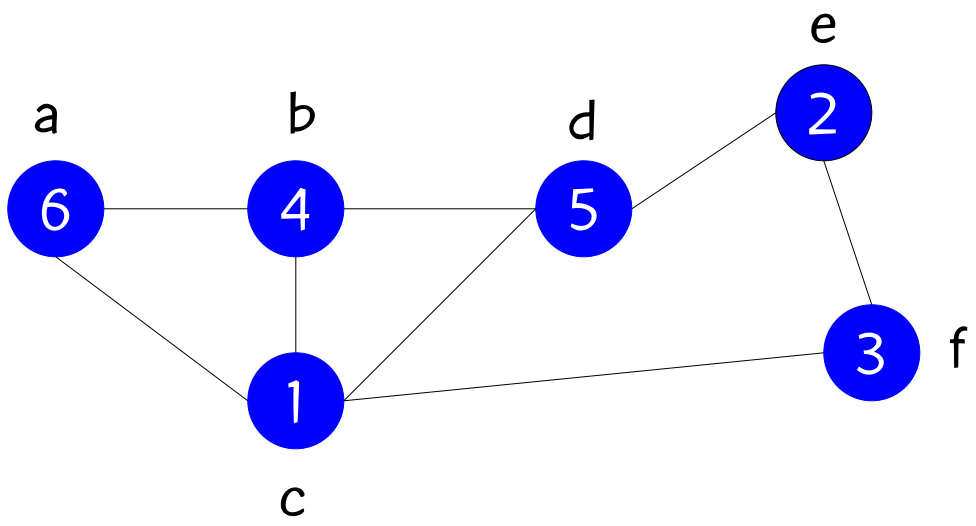
GRASP local search procedure



$AB_f(G) = 1$

v	$AB_f(v)$
a	2
b	1
c	2
d	1
e	1
f	1

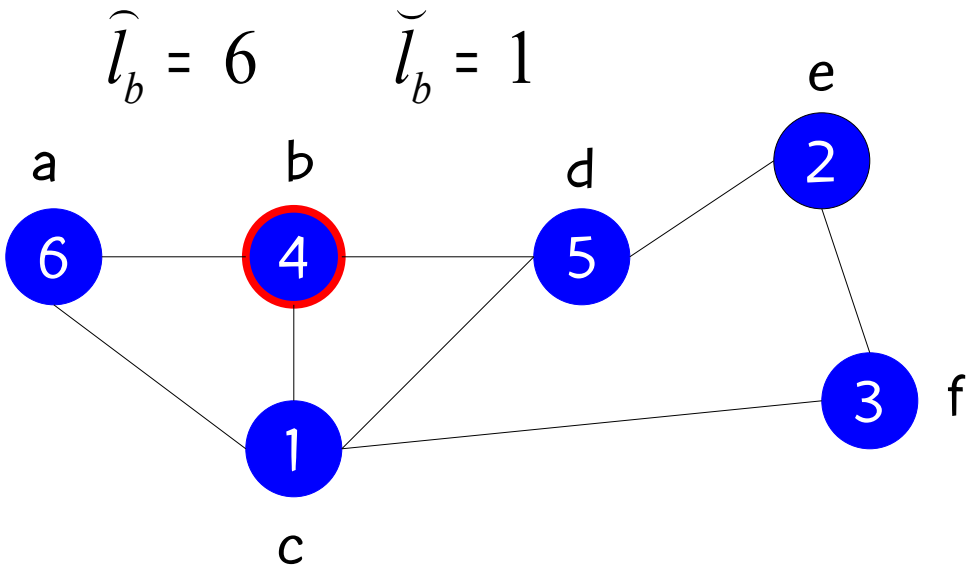
GRASP local search procedure



$AB_f(G) = 1$

v	$AB_f(v)$
a	2
b	1 crucial
c	2
d	1 crucial
e	1 crucial
f	1 crucial

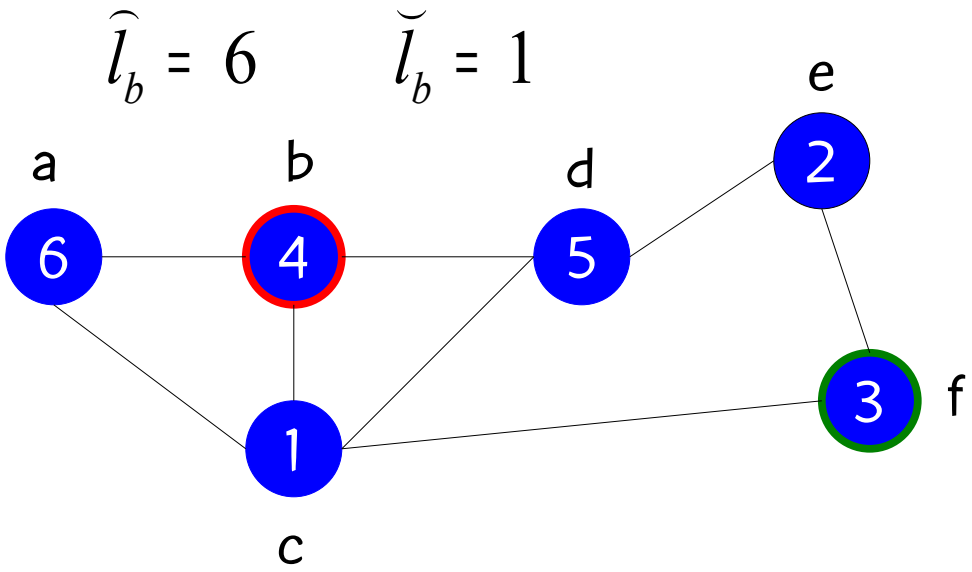
GRASP local search procedure



$AB_f(G) = 1$

l	$\lvert l - \hat{l}_u \rvert$	$\lvert l - \check{l}_u \rvert$
1	5	0
2	4	1
3	3	2
4	2	3
5	1	4
6	0	5

GRASP local search procedure

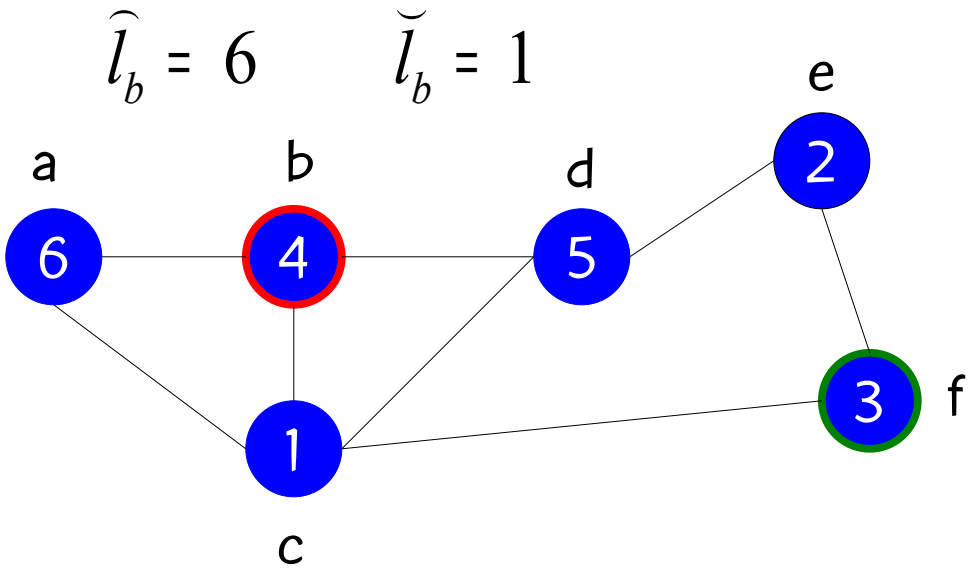


$$l_u^* = \text{argmax}\{\min(|l - \hat{l}_u|, |l - \check{l}_u|): l = 1,\ldots,n\} = 3$$

$AB_f(G) = 1$

l	$\lvert l - \hat{l}_u \rvert$	$\lvert l - \check{l}_u \rvert$
1	5	0
2	4	1
3	3	2
4	2	3
5	1	4
6	0	5

GRASP local search procedure



$AB_f(G) = 1$

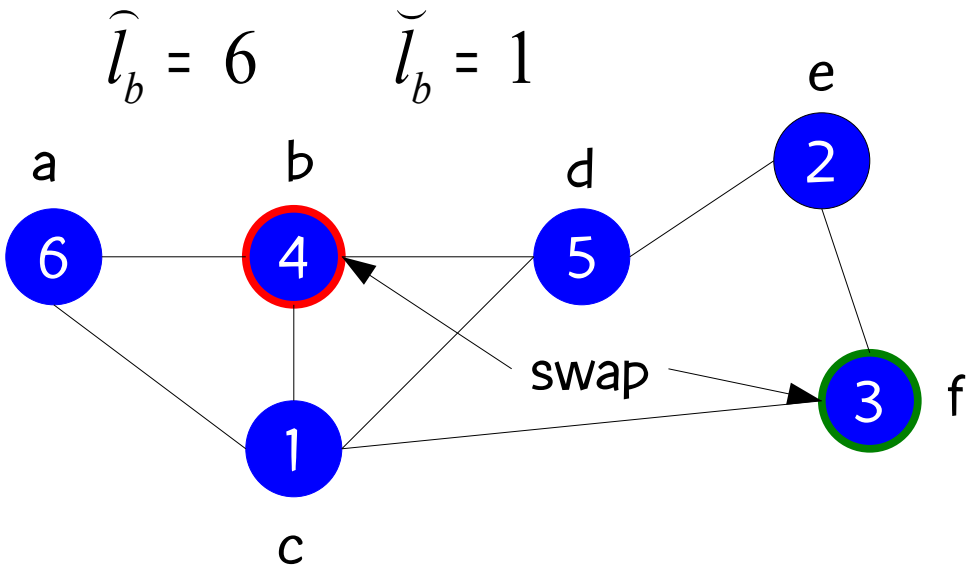
l	$\lvert l - \overset{\frown}{l}_u \rvert$	$\lvert l - \overset{\smile}{l}_u \rvert$
1	5	0
2	4	1
3	3	2
4	2	3
5	1	4
6	0	5

$$l_u^* = \text{argmax}\{\min(|l - \overset{\frown}{l}_u|, |l - \overset{\smile}{l}_u|): l = 1, \ldots, n\} = 3$$

Since $1 = \overset{\smile}{l}_u \le l_u^* \le \overset{\frown}{l}_u = 6$ then

$$N'(u) = \{v \in V : \overset{\smile}{l}_u + AB_f(G) \le f(v) \le \overset{\frown}{l}_u - AB_f(G)\} = \{d, e, f\}$$

GRASP local search procedure



$AB_f(G) = 1$

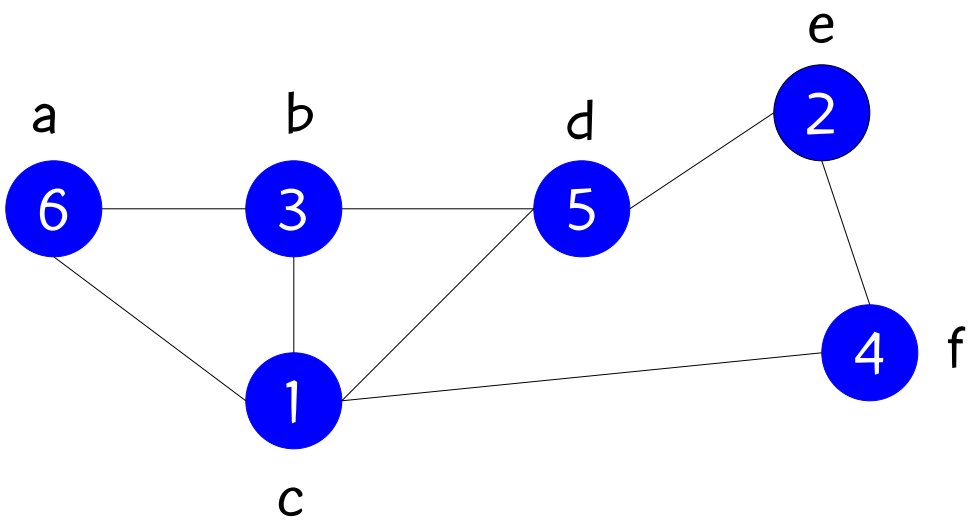
l	$\lvert l - \overset{\frown}{l}_u \rvert$	$\lvert l - \overset{\smile}{l}_u \rvert$
1	5	0
2	4	1
3	3	2
4	2	3
5	1	4
6	0	5

$$l_u^* = \text{argmax}\{\min(|l - \overset{\frown}{l}_u|, |l - \overset{\smile}{l}_u|): l = 1,\ldots,n\} = 3$$

Since $1 = \overset{\smile}{l}_u \le l_u^* \le \overset{\frown}{l}_u = 6$ then

$$N'(u) = \{v \in V : \overset{\smile}{l}_u + AB_f(G) \le f(v) \le \overset{\frown}{l}_u - AB_f(G)\} = \{d, e, f\}$$

GRASP local search procedure



$AB_f(G) = 2$

optimal!

GRASP local search procedure

- Value of a move:
  - Common practice is to define it as change in objective function value
  - In antibandwidth, change in objective function provides little information
- Given node u and node $v \in C(u)$, we define value of move(u,v) to be the difference in the antibandwidth of u.

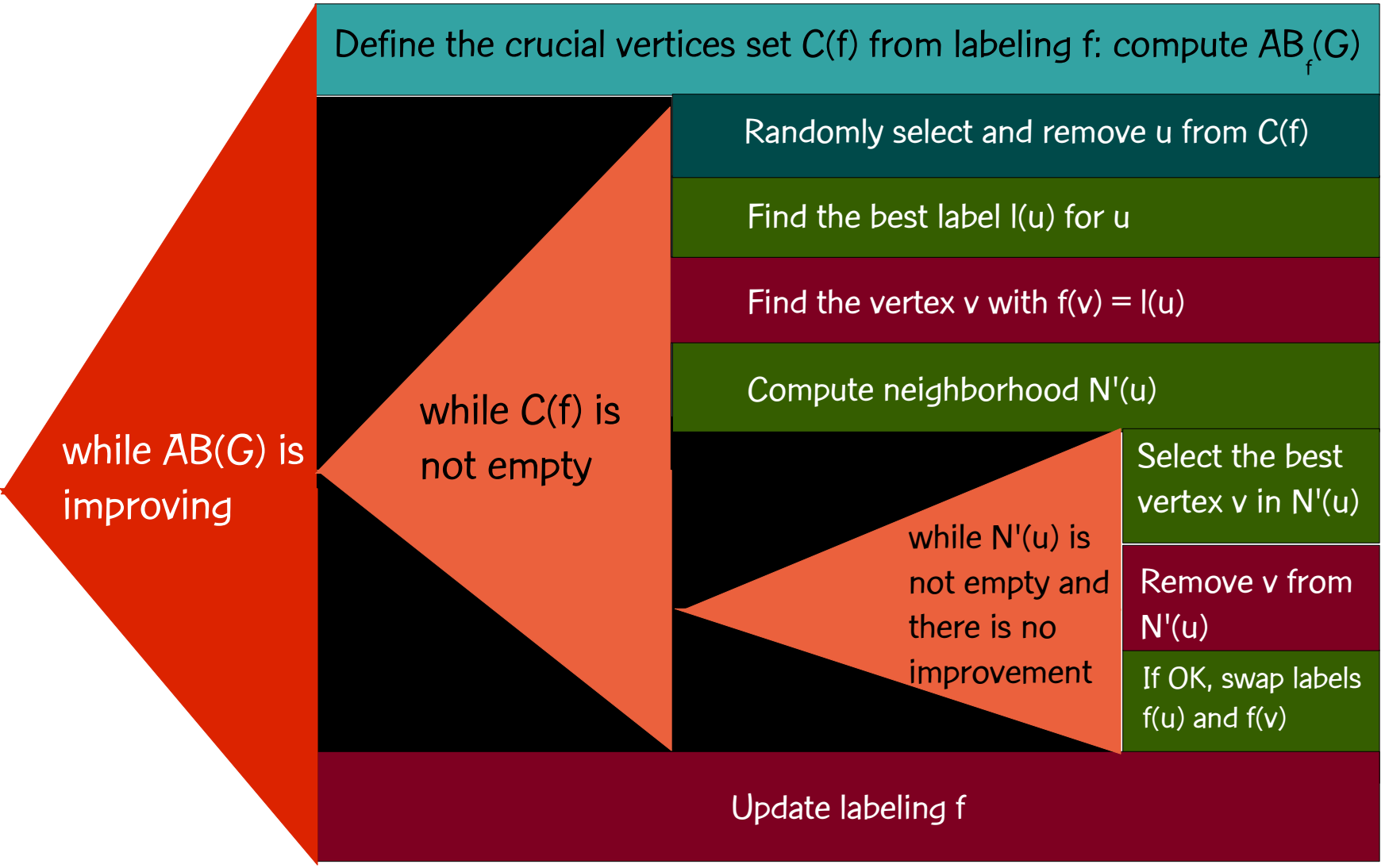
GRASP local search procedure

- If f is the original labeling and f' is the resulting labeling after move(u,v), then

$$\text{moveValue}(u,v) = AB_{f'}(u) - AB_{f}(u)$$

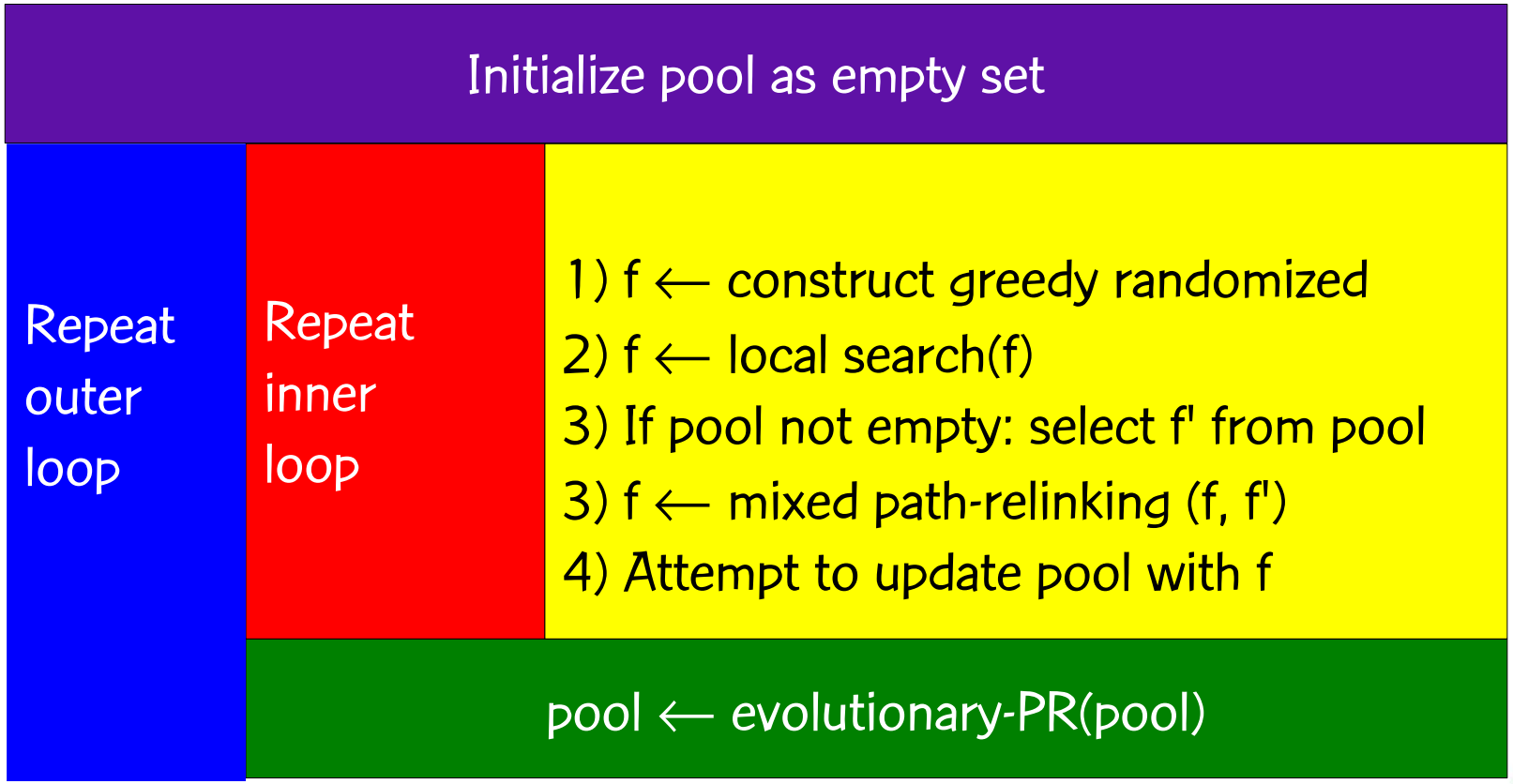
- Perform move(u,v) only if moveValue(u,v) > 0 and $AB_{f'}(v) \geq AB_{f}(G)$
- Computation of $AB_{f}(G)$ is expensive: requires examination of all vertices in graph
  - $AB_{f}(G)$ is not updated after each move, only when C(f) is computed (a la Glover & Laguna (1997))

GRASP local search procedure



GRASP with evolutionary path-relinking

GRASP with evolutionary path-relinking



Mixed path-relinking

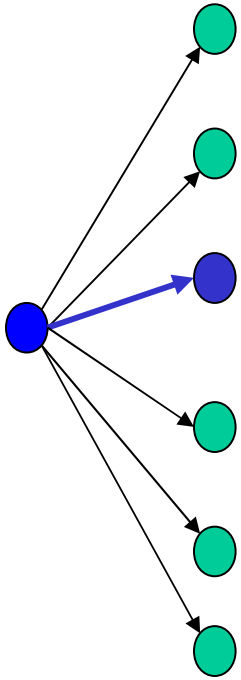
Variants: trade-offs between computation time and solution quality

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

Mixed path-relinking

Variants: trade-offs between computation time and solution quality

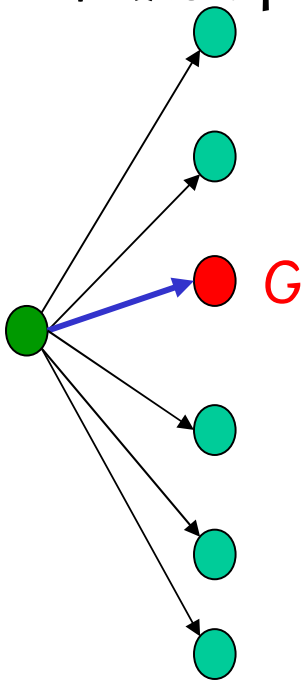
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

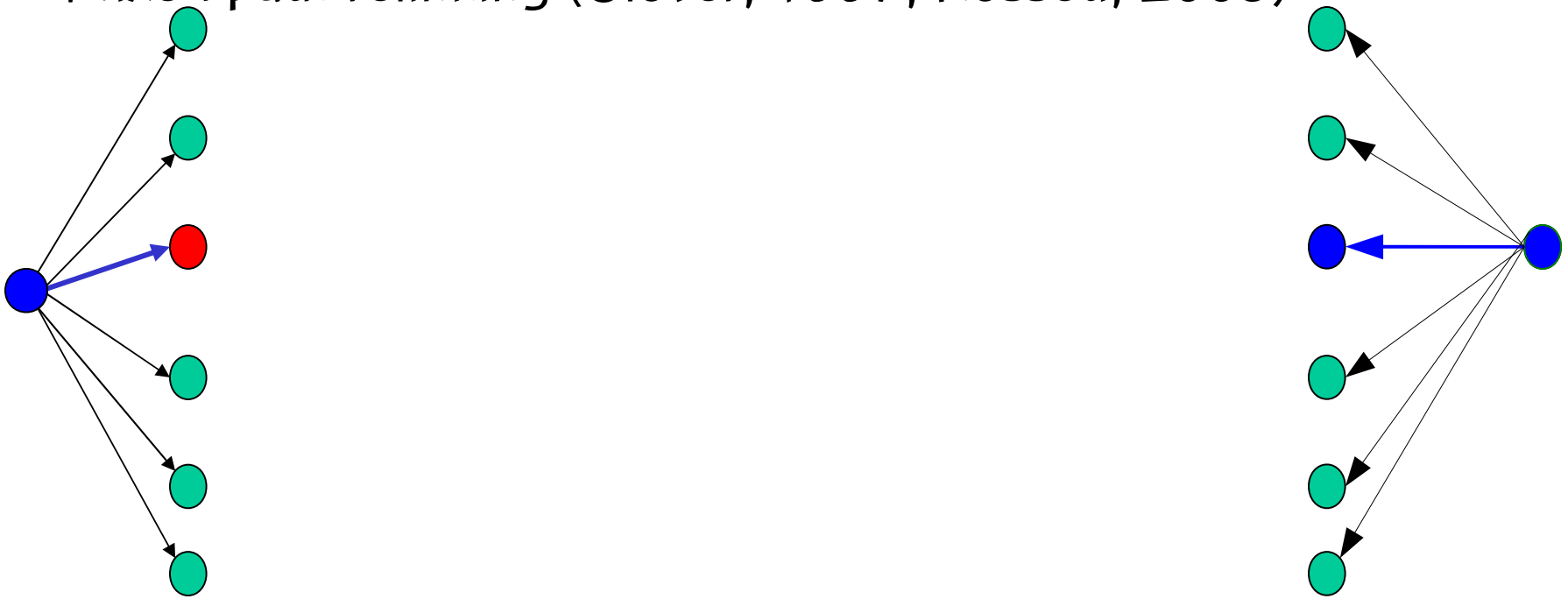
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

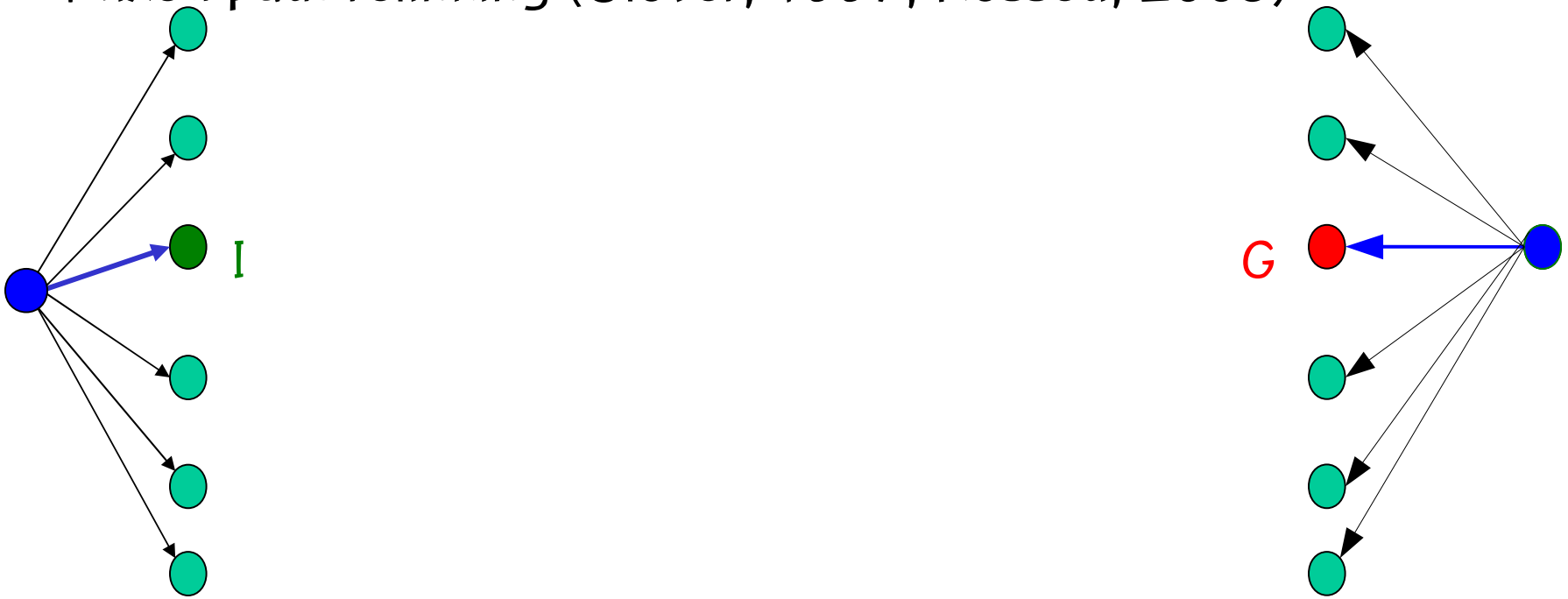
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

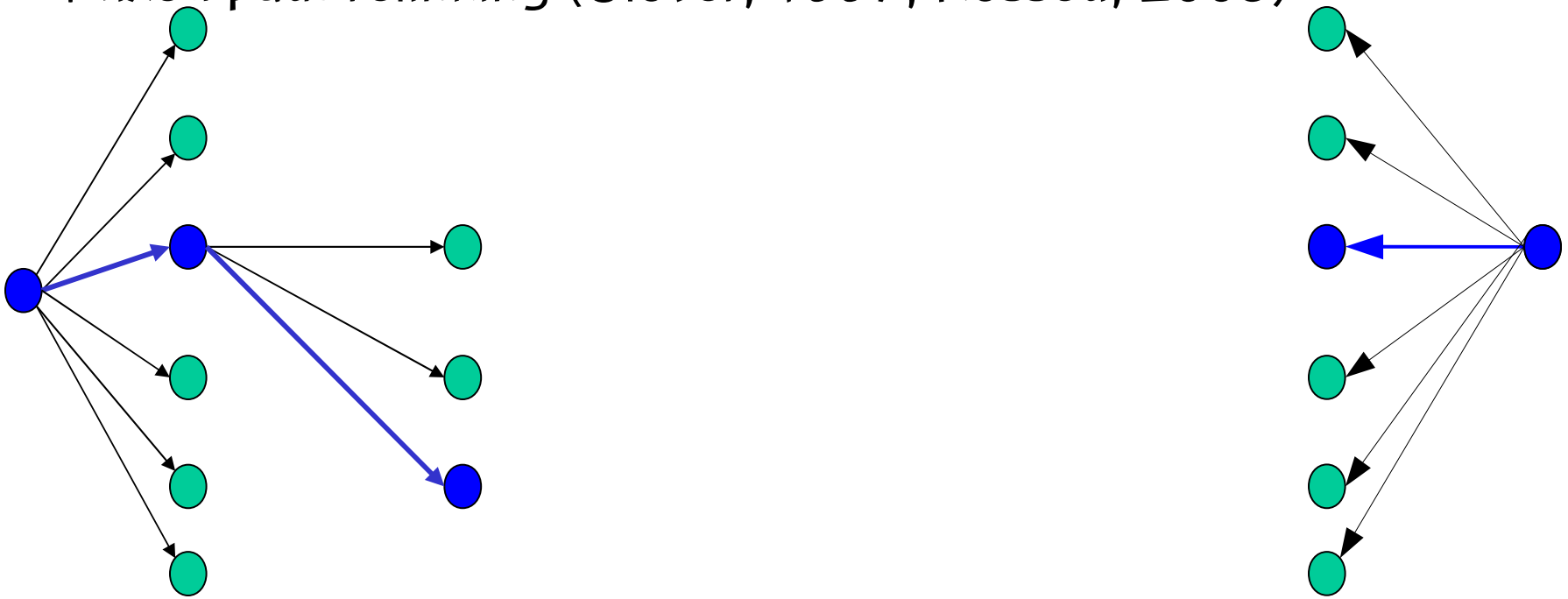
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

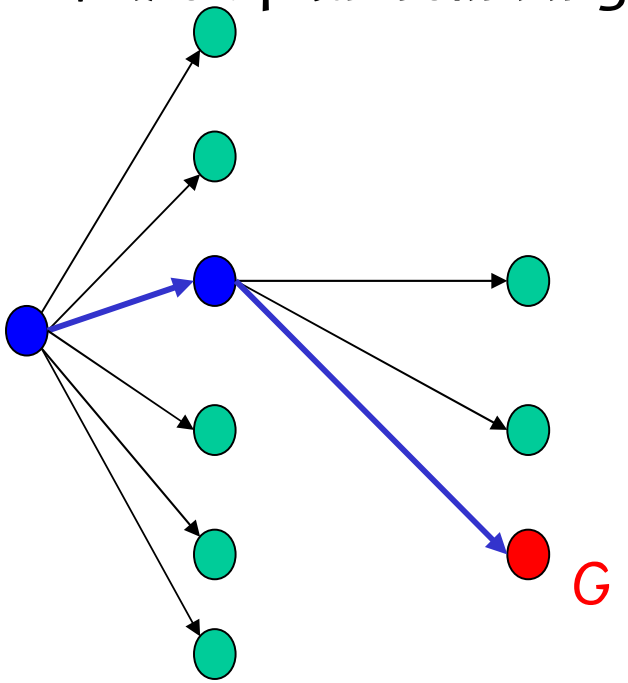
Mixed path-relinking (Glover, 1997; Rosseti, 2003)

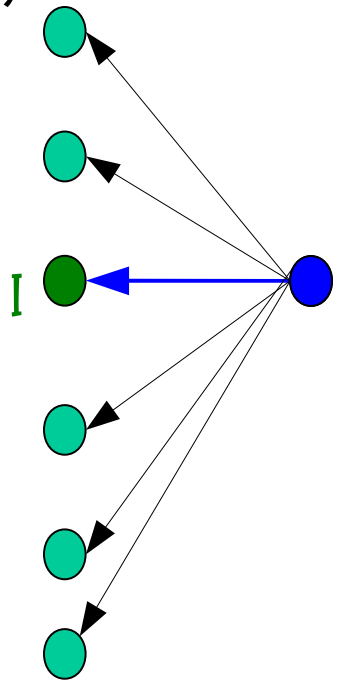


Mixed path-relinking

Variants: trade-offs between computation time and solution quality

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

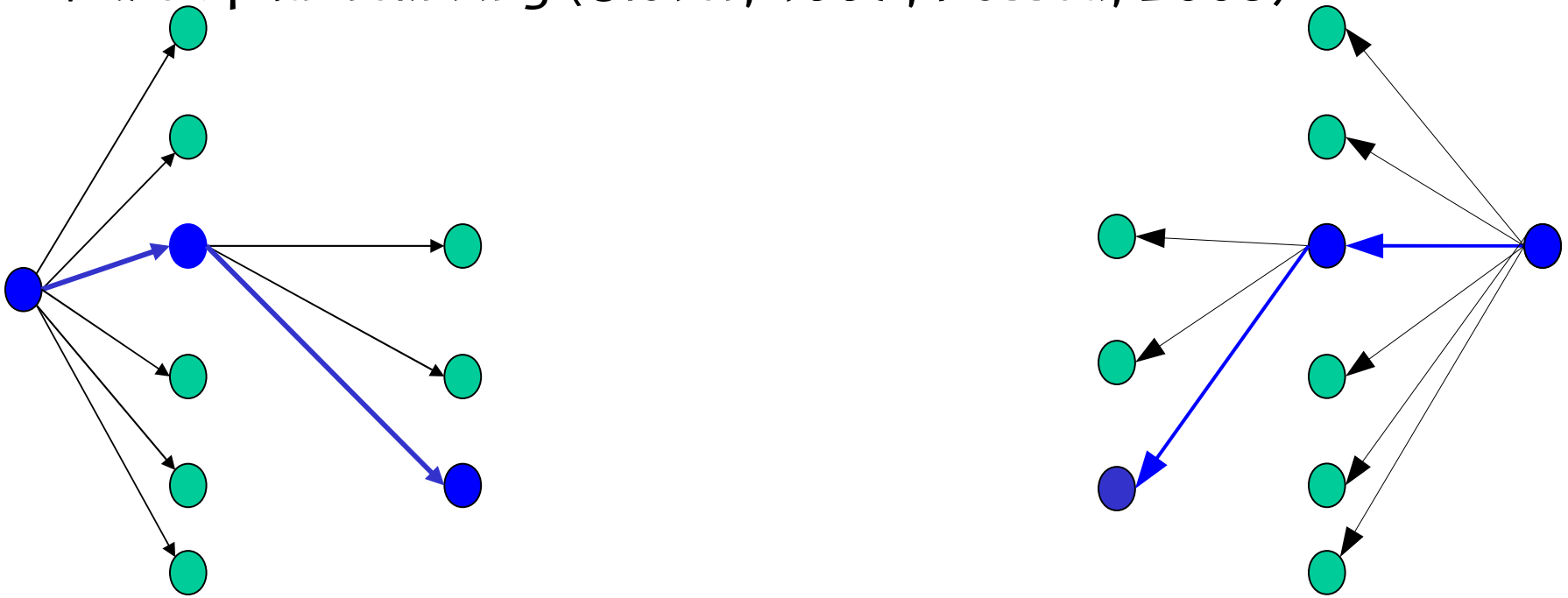




Mixed path-relinking

Variants: trade-offs between computation time and solution quality

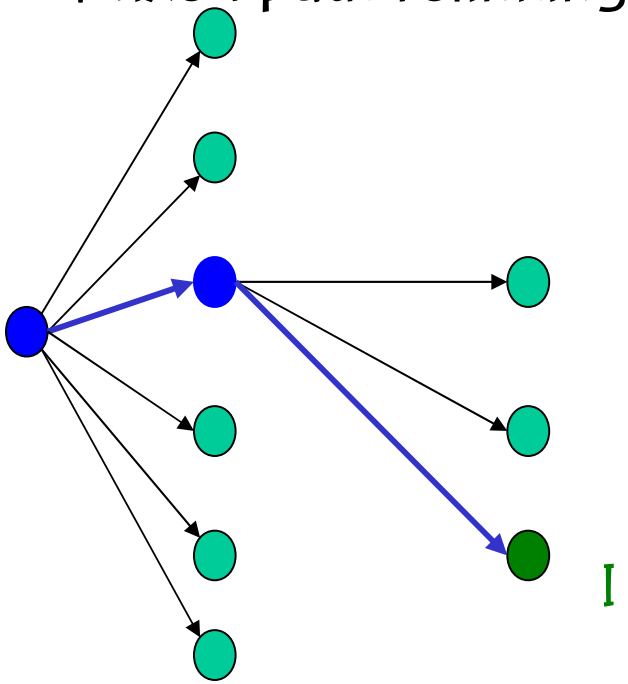
Mixed path-relinking (Glover, 1997; Rosseti, 2003)

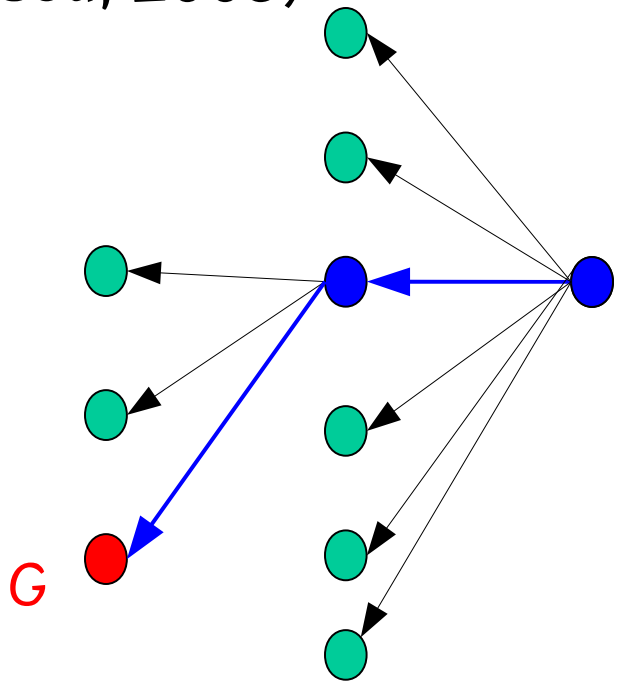


Mixed path-relinking

Variants: trade-offs between computation time and solution quality

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

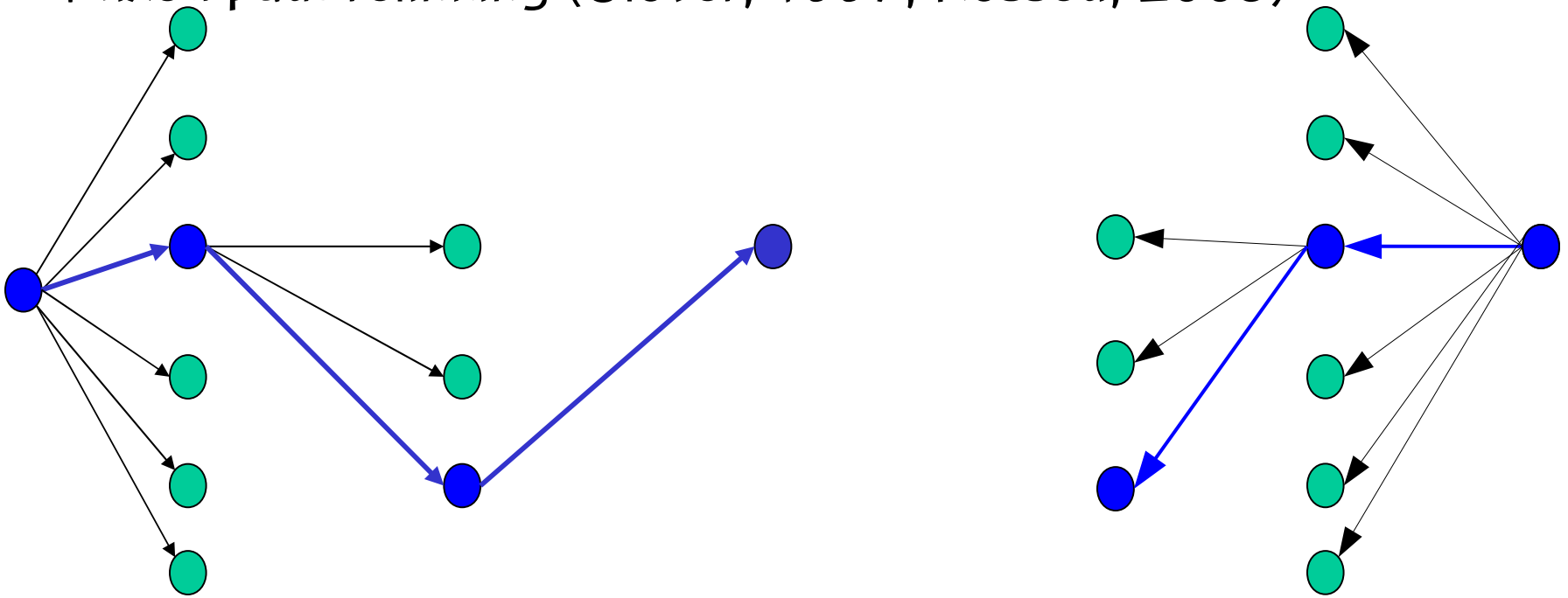




Mixed path-relinking

Variants: trade-offs between computation time and solution quality

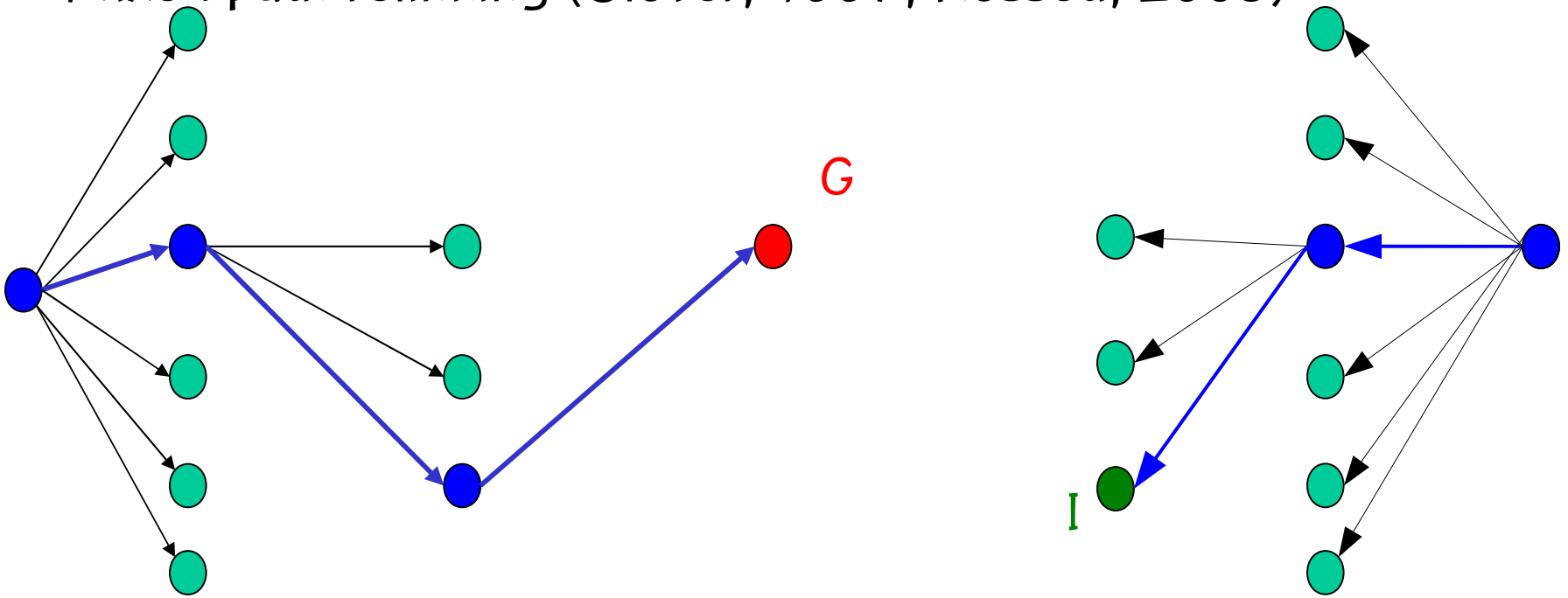
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

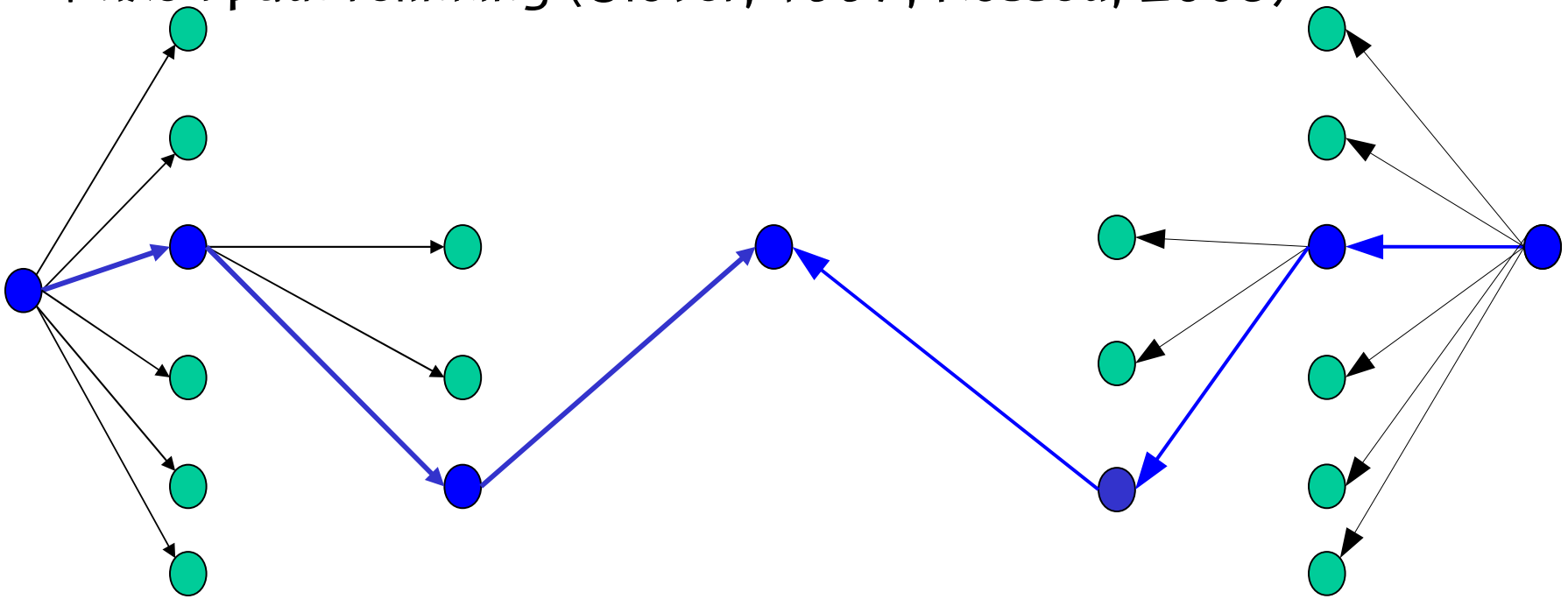
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



Mixed path-relinking

Variants: trade-offs between computation time and solution quality

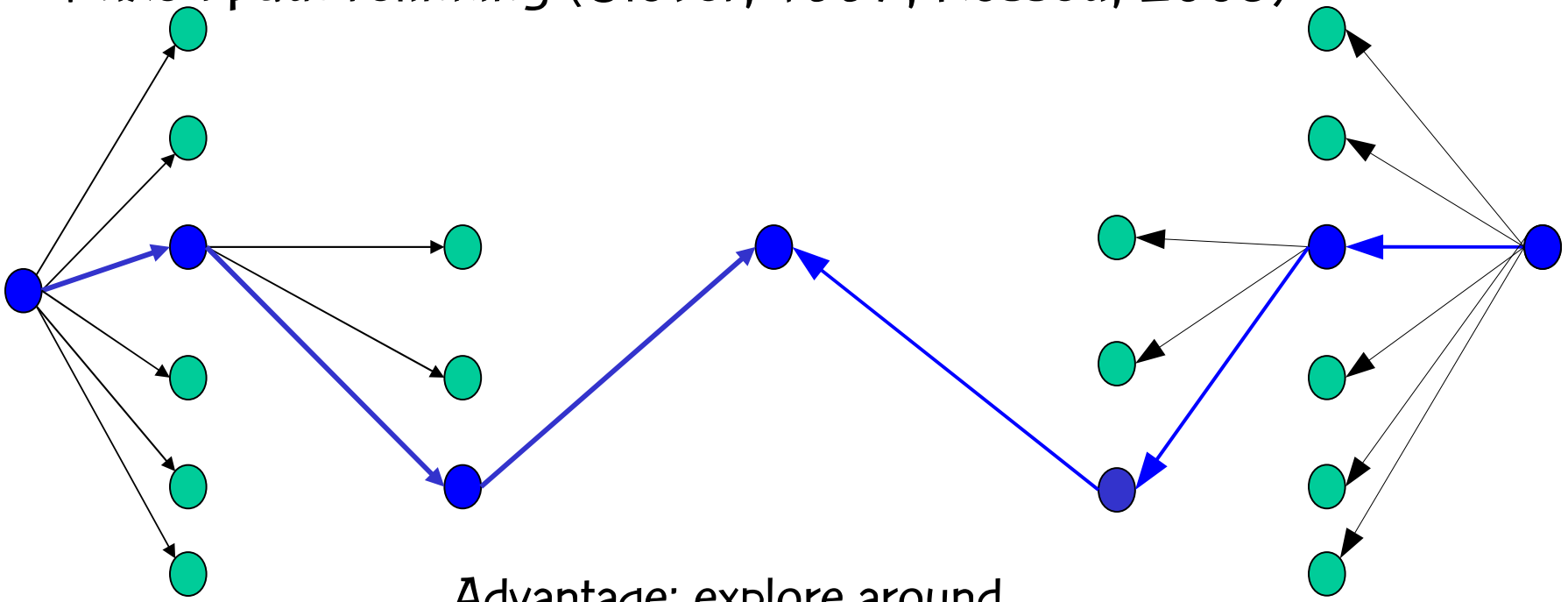
Mixed path-relinking (Glover, 1997; Rosseti, 2003)



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.

Pool management

- Pool has at most p (e.g. $p = 10$ elements) ordered from best {f(1)} to worst {f(p)}.
- Let $AB_{f(1)}(G)$ be the antibandwidth of the best labeling {f(1)} in the pool
- Labeling f is accepted to the pool if $AB_f(G) > AB_{f(1)}(G)$ or if $AB_f(G) > AB_{f(p)}(G)$ and $\Delta(f, pool) > \delta$, where

$$\Delta(f, \text{pool}) = \min\{\sum_{k=1}^{n} |f(k) - f^i(k)| : i \in \text{pool}\}$$

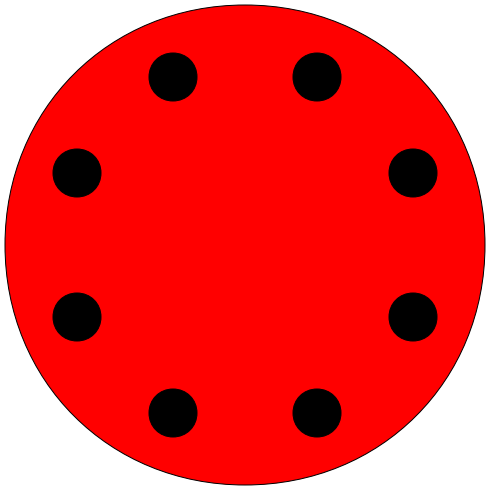
- If the pool is full and f is accepted into the pool: among all labelings f' such that $AB_{f'}(G) < AB_f(G)$ we remove from the pool the labeling closest to f.

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.

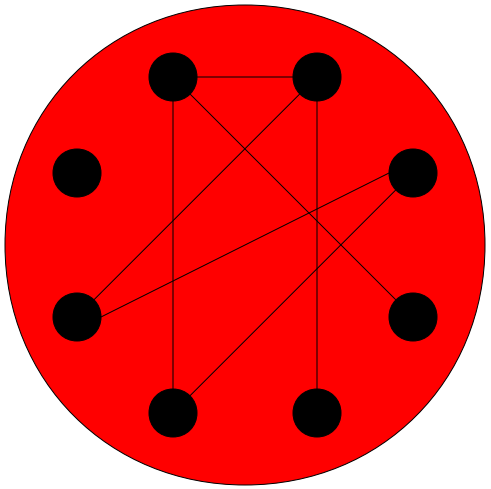
Evolutionary path-relinking (EvPR)



We use a variant of EvPR introduced in Resende, Martí, Gallego, & Duarte (2008)

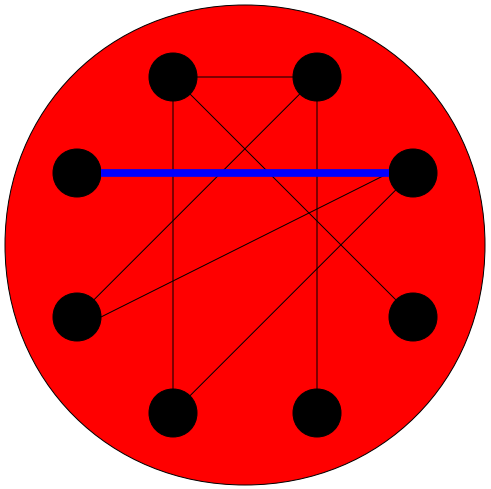
Start with the pool of elite solutions

Evolutionary path-relinking (EvPR)



While there exists a pair of pool solutions that have not yet been relinked:

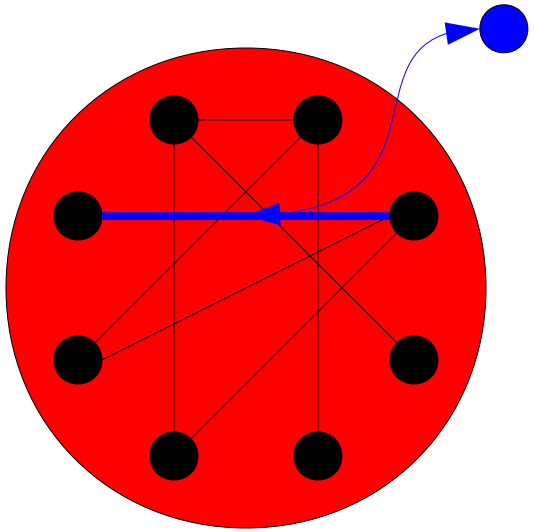
Evolutionary path-relinking (EvPR)



While there exists a pair of pool solutions that have not yet been relinked:

Apply mixed path-relinking between pair

Evolutionary path-relinking (EvPR)

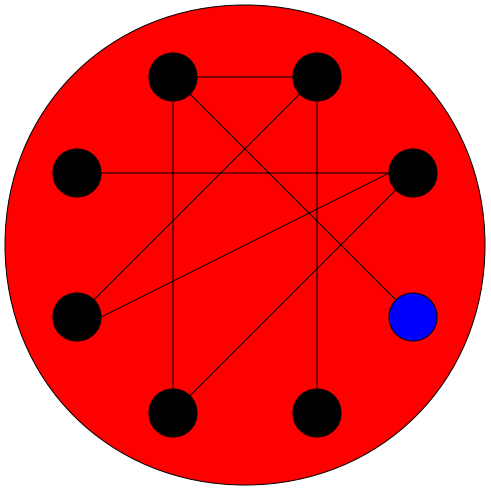


While there exists a pair of pool solutions that have not yet been relinked:

Apply mixed path-relinking between pair

Solution of path-relinking is candidate to enter the pool: if accepted, it replaces closest solution with smaller antibandwidth

Evolutionary path-relinking (EvPR)

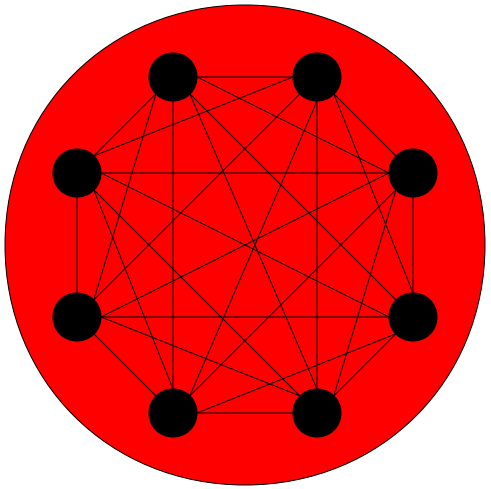


While there exists a pair of pool solutions that have not yet been relinked:

Apply mixed path-relinking between pair

Solution of path-relinking is candidate to enter the pool: if accepted, it replaces closest solution with smaller antibandwidth

Evolutionary path-relinking (EvPR)



EvPR ends when all pairs of pool solutions have been relinked and resulting labelings are not accepted to enter the pool.

Preliminary experimental results

Experiments

- Heuristics were coded in C and testing was done on a 3.0 GHz Pentium 4 PC with 3 Gb of memory
- CPLEX 11.1 was used to solve the integer program on a 1.6 GHz Itanium 2 computer with 256 Gb of memory
- Four sets of test problems serve as our benchmark

Experiments

- Test problems derived from the Harwell-Boeing Sparse Matrix Collection
  - 12 small instances (having between 30 and 100 vertices)
  - 12 large instances (having between 400 and 900 vertices)
- 2-dim meshes with optimal solutions known by construction (Raspaud et al., 2008)
  - 12 small instances (having between 90 and 120 vertices)
  - 12 large instances (having between 900 and 1200 vertices)

Experiments

All instances are available at http://www.uv.es/rmarti

- Test problems from the Harwell-Boeing Sparse Matrix Collection
  - 12 small instances (having between 30 and 100 vertices)
  - 12 large instances (having between 400 and 900 vertices)
- 2-dim meshes with optimal solutions known by construction (Raspaud et al., 2008)
  - 12 small instances (having between 90 and 120 vertices)
  - 12 large instances (having between 900 and 1200 vertices)

Integer programming: small Harwell-Boeing instances

bcspwr01	209	1607	4931	74.8	3.7	14641	**17**	**17**
bcspwr02	265	2510	7675	426.6	8.9	>24h	21	22
ibm32	276	1147	3792	27.5	0.5	5709	**9**	**9**
pores1	296	1034	3524	352.6	16.9	>24h	6	8
curtis54	410	3095	9740	219.2	7.0	>24h	10	13
will57	425	3434	10763	216.0	3.8	>24h	12	14
bcsstk01	496	2529	8320	219.9	4.9	>24h	6	11
dwt234	675	13969	42363	91.9	1.7	>24h	23	58
ash85	693	7530	23427	116.2	3.8	>24h	12	27
bcspwr03	712	14222	43204	75.3	1.7	>24h	22	57
impcol.b	739	3822	12691	148.8	2.5	>24h	5	11
nos4	794	10348	31976	99.6	2.8	>24h	10	48

Integer programming: large Harwell-Boeing instances

494bus	2654	245117	736	4.52	949	>24h	12	247
662bus	3798	439813	1321	1.35	408	>24h	16	331
685bus	4619	471193	1417	1.53	10	>24h	3	342
bcsstk06	8700	180541	559	5.97	406	>24h	1	210
bcsstk07	8700	180541	559	5.85	401	>24h	1	210
can445	4699	200153	608	3.35	321	>24h	1	221
can715	8095	514916	1557	1.89	16	>24h	1	357
dwt503	7033	256275	781	2.40	103	>24h	1	250
dwt592	6288	353313	1069	3.38	84	>24h	2	295
impcold	3809	182318	552	4.59	466	>24h	2	212
nos6	4605	457591	1377	2.37	48	>24h	4	337
sherman	4320	300004	905	3.16	107	>24h	5	272

Experiments with GRASP

- For each of the 48 instances, we apply G+evPR and G+PR 30 times
- G+evPR: 25 iterations of inner loop and 4 iterations of the outer loop (total of 100 GRASP iterations)
- G+PR: 250 iterations
- Size of elite set is 10

Deviation w.r.t. best or optimum

Small grids	G+PR	2.9 %	5.9 %	3.8 %
	G+evPR	2.2 %	4.8 %	3.4 %
Large grids	G+PR	2.4 %	3.8 %	3.3 %
	G+evPR	2.2 %	3.6 %	3.0 %
Small H-B	G+PR	0.6 %	5.9 %	3.8 %
	G+evPR	0.0 %	5.9 %	3.1 %
Large H-B	G+PR	1.0 %	3.9 %	2.7 %
	G+evPR	0.0 %	3.4 %	2.1 %

CPU time (seconds)

Small grids	G+PR	2.4	2.7	2.6
	G+evPR	4.0	5.4	4.7
Large grids	G+PR	1009.0	1081.6	1046.8
	G+evPR	2479.3	3281.1	2822.1
Small H-B	G+PR	1.0	1.1	1.1
	G+evPR	3.9	4.9	4.3
Large H-B	G+PR	194.3	200.9	197.6
	G+evPR	588.7	790.2	668.5

Number of best (optimal) solutions

Small grids	G+PR	0	30	23%
	G+evPR	0	30	25%
Large grids	G+PR	0	0	0%
	G+evPR	0	0	0%
Small H-B	G+PR	0	30	51%
	G+evPR	1	30	57%
Large H-B	G+PR	0	30	12%
	G+evPR	1	30	17%

Number of best (optimal) solutions

Small grids	G+PR	0	30	23%
	G+evPR	0	30	25%
Large grids	G+PR	0	0	0%
	G+evPR	0	0	0%
Small H-B	G+PR	0	30	51%
	G+evPR	1	30	57%
Large H-B	G+PR	0	30	12%
	G+evPR	1	30	17%

A minimum of 0 implies at least one instance (of the 12) for which all 30 runs failed to find the best/opt

Number of best (optimal) solutions

Small grids	G+PR	0	30	23%
	G+evPR	0	30	25%
Large grids	G+PR	0	0	0%
	G+evPR	0	0	0%
Small H-B	G+PR	0	30	51%
	G+evPR	1	30	57%
Large H-B	G+PR	0	30	12%
	G+evPR	1	30	17%

A maximum of 30 implies at least one instance (of the 12) for which all 30 runs found the best/opt

Number of best (optimal) solutions

Small grids	G+PR	0	30	23%
	G+evPR	0	30	25%
Large grids	G+PR	0	0	0%
	G+evPR	0	0	0%
Small H-B	G+PR	0	30	51%
	G+evPR	1	30	57%
Large H-B	G+PR	0	30	12%
	G+evPR	1	30	17%

%best = total number of runs that found best/opt / (12 × 30)

Small Harwell-Boeing instances (solution values)

		max	min	avg	max	min	avg
bcspwr01	**17**	17	16	16.13	17	16	16.40
bcspwr02	21	21	20	20.97	21	20	20.93
ibm32	**9**	9	8	8.30	9	8	8.27
pores1	6	6	6	6	6	6	6
curtis54	10	12	12	12	12	12	12
will57	12	13	12	12.3	13	12	12.43
bcsstk01	6	8	8	8	8	8	8
dwt234	23	51	49	49.5	51	49	49.67
ash85	12	21	19	19.87	22	19	20.30
bcspwr03	22	39	39	39	39	39	39
impcol.b	5	8	7	7.4	8	7	7.63
nos4	10	34	31	32.6	35	31	33.03

Large Harwell-Boeing instances (solution values)

		max	min	avg	max	min	avg
494bus	12	227	224	225.43	228	224	225.73
662bus	16	220	219	219.33	220	219	219.57
685bus	3	136	136	136.00	136	136	136.00
bcsstk06	1	32	31	31.2	33	31	31.57
bcsstk07	1	32	31	31.03	33	31	31.57
can445	1	82	75	78.2	85	78	80.67
can715	1	115	112	113.73	127	115	115.97
dwr503	1	53	51	51.97	58	51	53.73
dwr592	2	108	99	103.03	112	102	106.10
impcol.d	2	104	100	102.03	105	101	102.90
nos6	4	326	324	325.4	328	325	326.47
sherman	5	261	260	260.1	261	260	261.1

Concluding remarks

- We described a GRASP with evolutionary path-relinking for the antibandwidth problem.
- The antibandwidth problem has an important application in frequency assignment in cellular telephony.
- To complete the experiments, we will derive run time distributions for the heuristics. Preliminary results indicate that G+PR and G+evPR have similar run time distributions.
- We will also conclude the CPLEX runs on the mesh instances. Preliminary results indicate that CPLEX cannot solve optimally even the smallest of the mesh instances.
- Our current G+evPR implementation can be made more efficient, resulting in a reduction in the number of path-relinking operations in the evolutionary path-relinking procedure.

Coauthors

Ricardo M. A. Silva

Abraham Duarte

Rafael Martí

The End

These slides and a technical report
can be downloaded from my homepage:
http://www.research.att.com/~mgcr

Hospital layout optimization using GRASP with path-relinking

Tutorial given at the Spring School in Advances in Operations Research, Higher School of Economics
Nizhny Novgorod, Russia ✤ May 3, 2011

Mauricio G. C. Resende
AT&T Labs Research
Florham Park, New Jersey

mgcr@research.att.com

Joint work with Ricardo M. A. Silva

Summary

- Modeling hospital layout via quadratic assignment
- Modeling hospital layout via generalized quadratic assignment
- Generalized quadratic assignment problem (GQAP)
- GRASP with path-relinking for GQAP
  - GRASP construction
  - Local search
  - Path-relinking
- Experimental results

Hospital layout as a QAP [Elshafei, 1977]

- Assign N facilities (surgery, ICU, recovery, ...) to N locations in the hospital
  - Each facility is assigned to a unique location
  - Each location has only one facility assigned to it
- Given:
  - Number of patients that move between each pair (i,j) of facilities (in some time period): P(i,j)
  - Distance between each pair of locations: D(k,l)
- Minimize average distance traveled by patients

Hospital layout as a QAP [Elshafei, 1977]

- assignment array π:
  - $[\pi(i) = j \Leftrightarrow$ facility i is assigned to location j]
- $P[i,j] \times D[\pi(i), \pi(j)]$
  - Total distance traveled by patients between facilities i and j that are assigned to locations $\pi(i)$ and $\pi(j)$, respectively

Hospital layout as a QAP [Elshafei, 1977]

Find the assignment vector π from all possible permutations Π_N of {1, 2, ..., N} that minimizes:

$$\sum_{i,j} P[i,j] \times D[\pi(i), \pi(j)]$$

Hospital layout as a QAP [Elshafei, 1977]

Find the assignment vector π from all possible permutations Π_N of {1, 2, ..., N} that minimizes:

$$\sum_{i,j} P[i,j] \times D[\pi(i), \pi(j)]$$

QAP's are one of the most computationally difficult classes of combinatorial optimization problems: Instances of size N=20 are considered challenging for exact methods.

Hospital layout as a QAP [Elshafei, 1977]

Find the assignment vector π from all possible permutations Π_N of {1, 2, ..., N} that minimizes:

$$\sum_{i,j} P[i,j] \times D[\pi(i), \pi(j)]$$

Heuristics are optimization methods that find good, though not provably optimal solutions to combinatorial optimization problems like the QAP.

Hospital layout as a QAP [Elshafei, 1977]

Find an assignment vector π from all possible permutations Π_N of {1, 2, ..., N} that minimizes:

$$\sum_{i,j} P[i,j] \times D[\pi(i), \pi(j)]$$

Since the 1990s, many effective heuristics have been developed for the QAP. Examples: simulated annealing, tabu search, genetic algorithms, ant colony optimization, and GRASP.

Hospital layout as a QAP [Elshafei, 1977]

The main drawback of the QAP model is that it assumes that it does not take into account that facilities have different dimensions and that they must be assigned to locations that can accommodate them.

Hospital layout as a QAP [Elshafei, 1977]

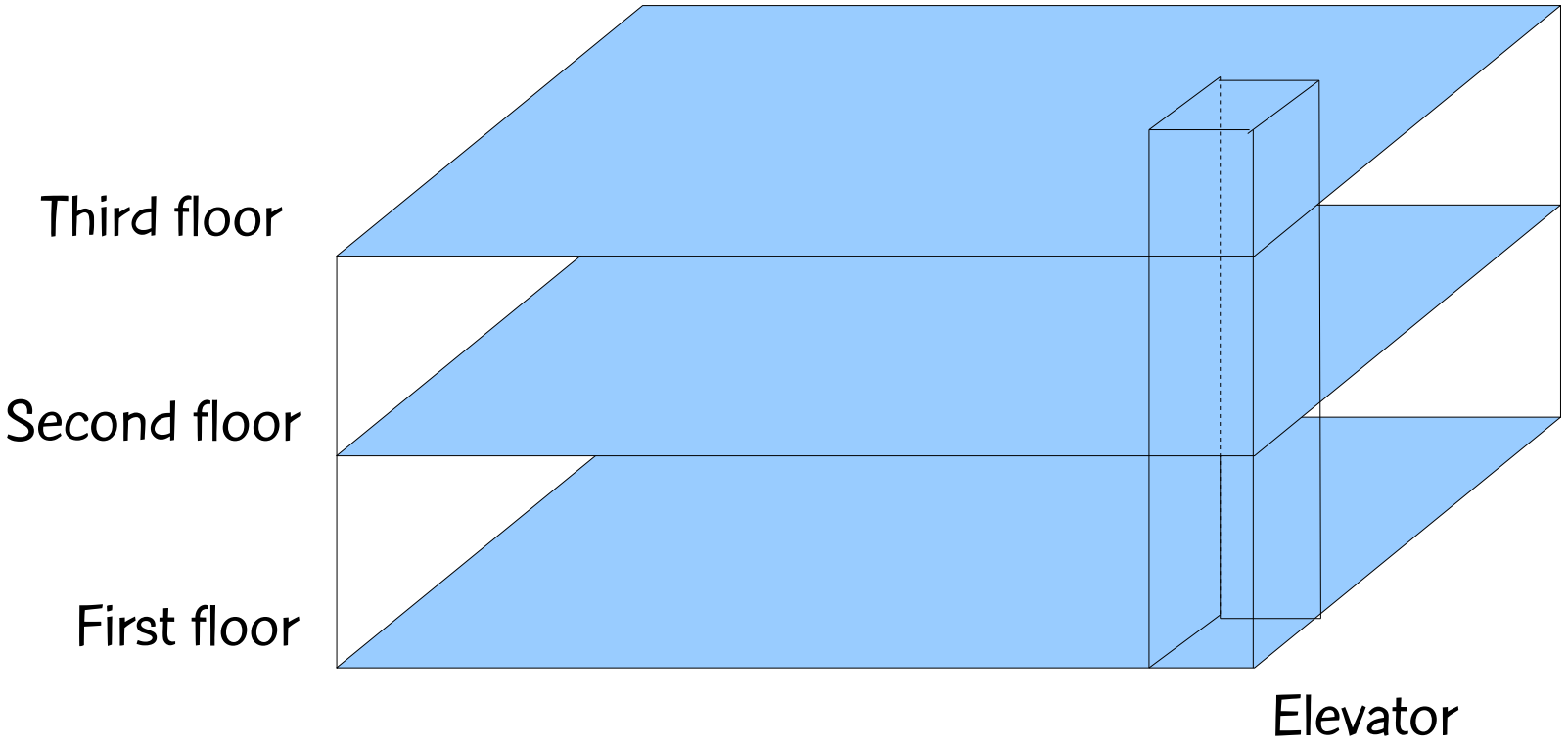
The main drawback of the QAP model is that it assumes that it does not take into account that facilities have different dimensions and that they must be assigned to locations that can accommodate them.

The generalized QAP model addresses this.

Hospital layout as a GQAP

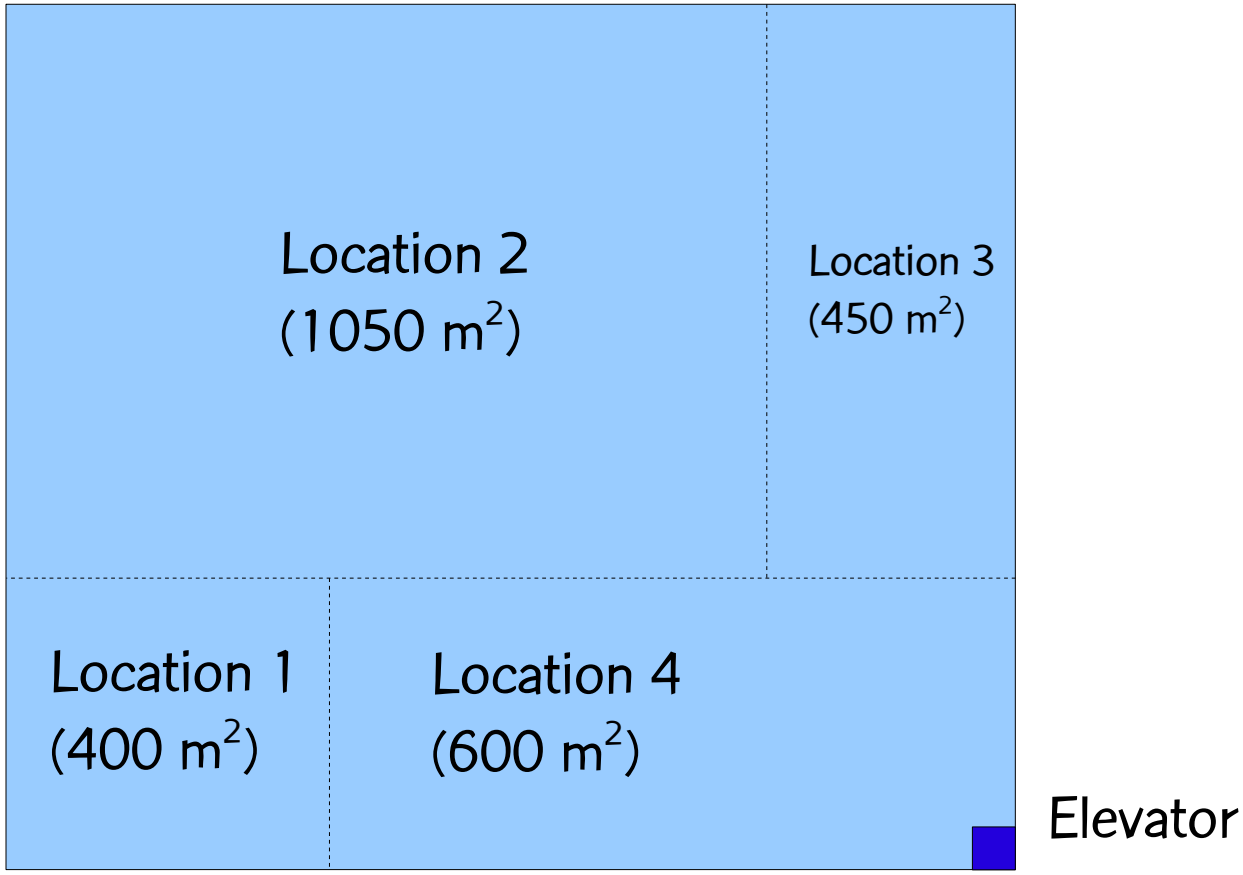
- The GQAP is similar to the QAP except that
  - Facilities have an associated area
  - Locations have an associated total available area
- Assign facilities to locations minimizing the average distance traveled by patients such that
  - Sum of areas of facilities assigned to a location does not exceed the total available area of the location
  - More than one facility can be assigned to a location.
  - No facility can be be assigned to a location.

Hospital layout as a GQAP

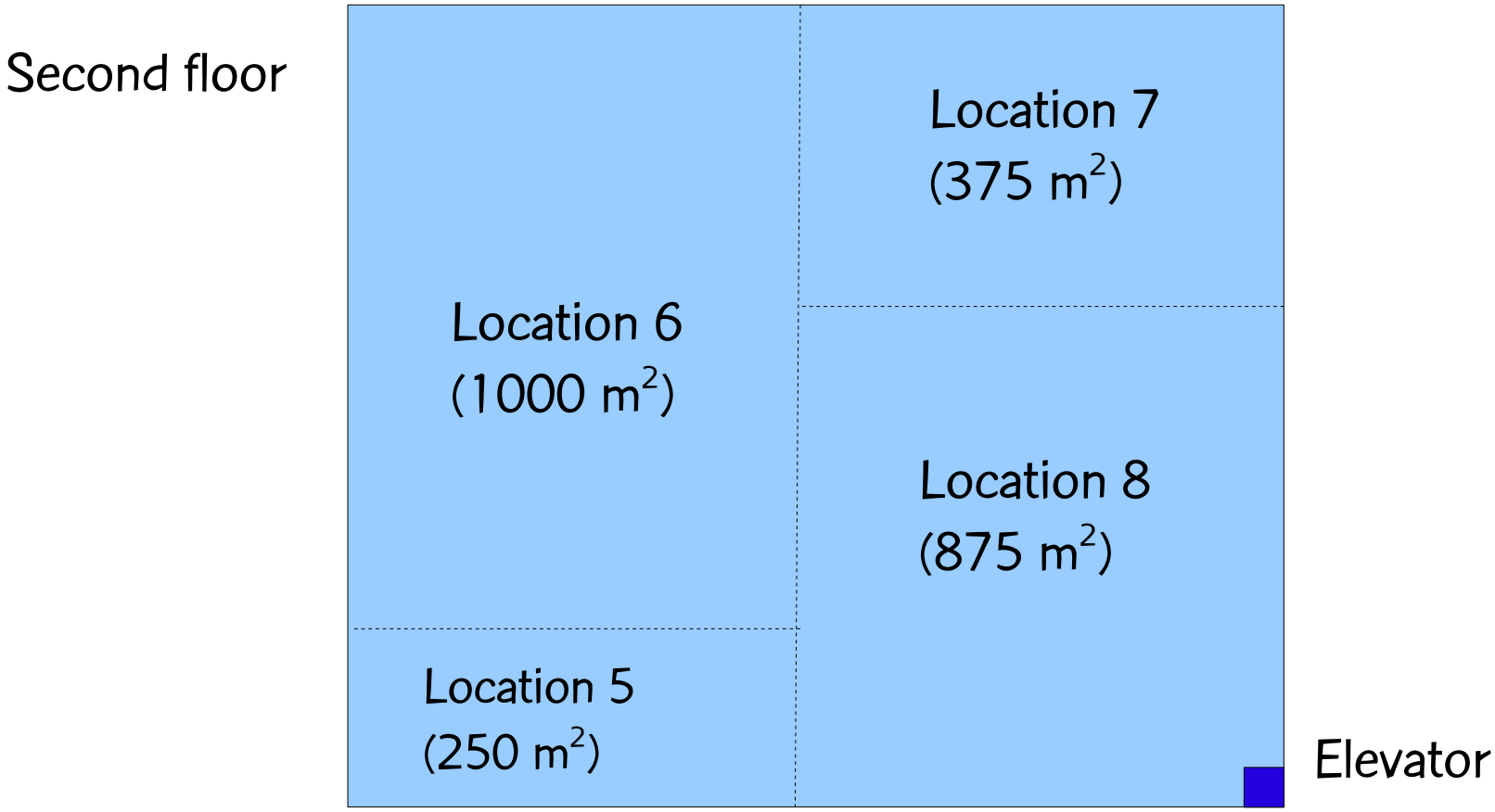


Hospital layout as a GQAP

First floor

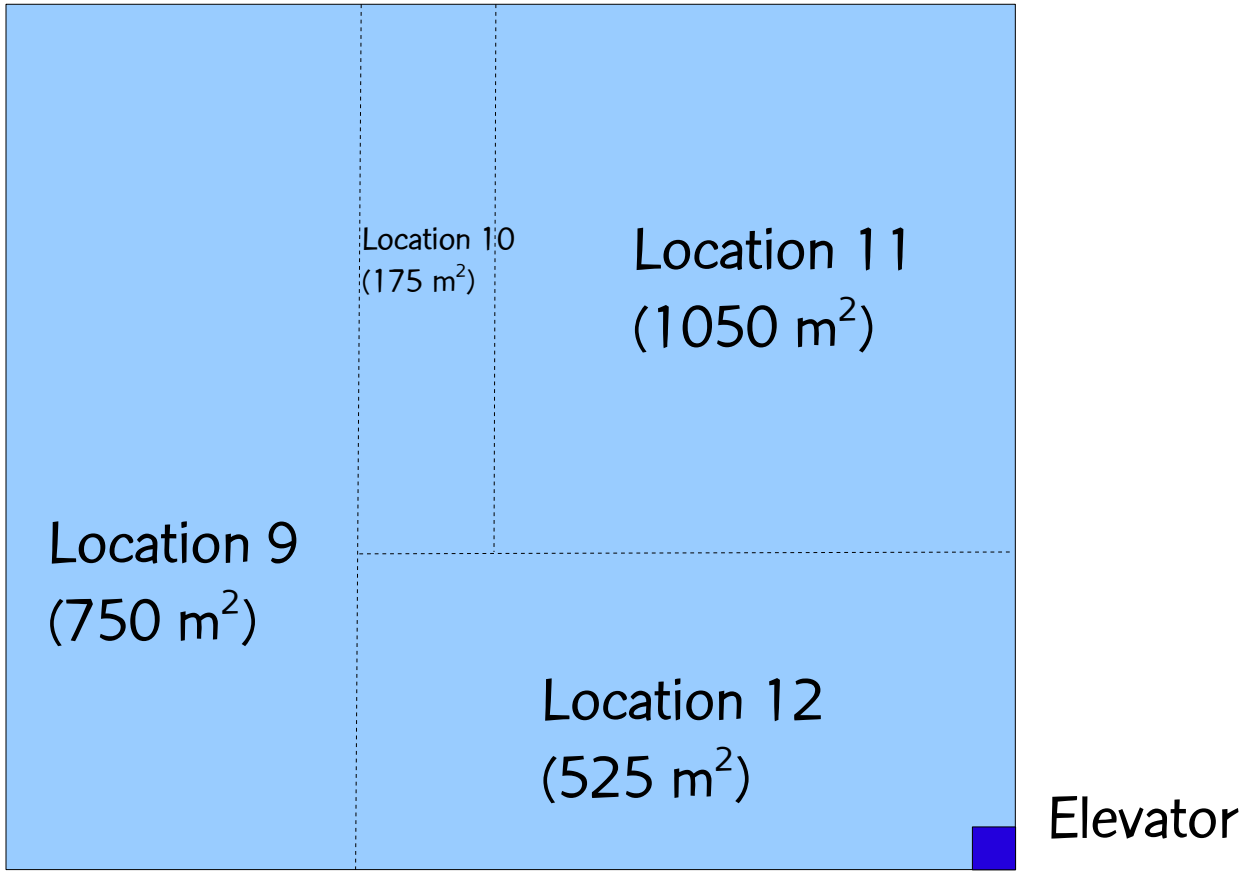


Hospital layout as a GQAP



Hospital layout as a GQAP

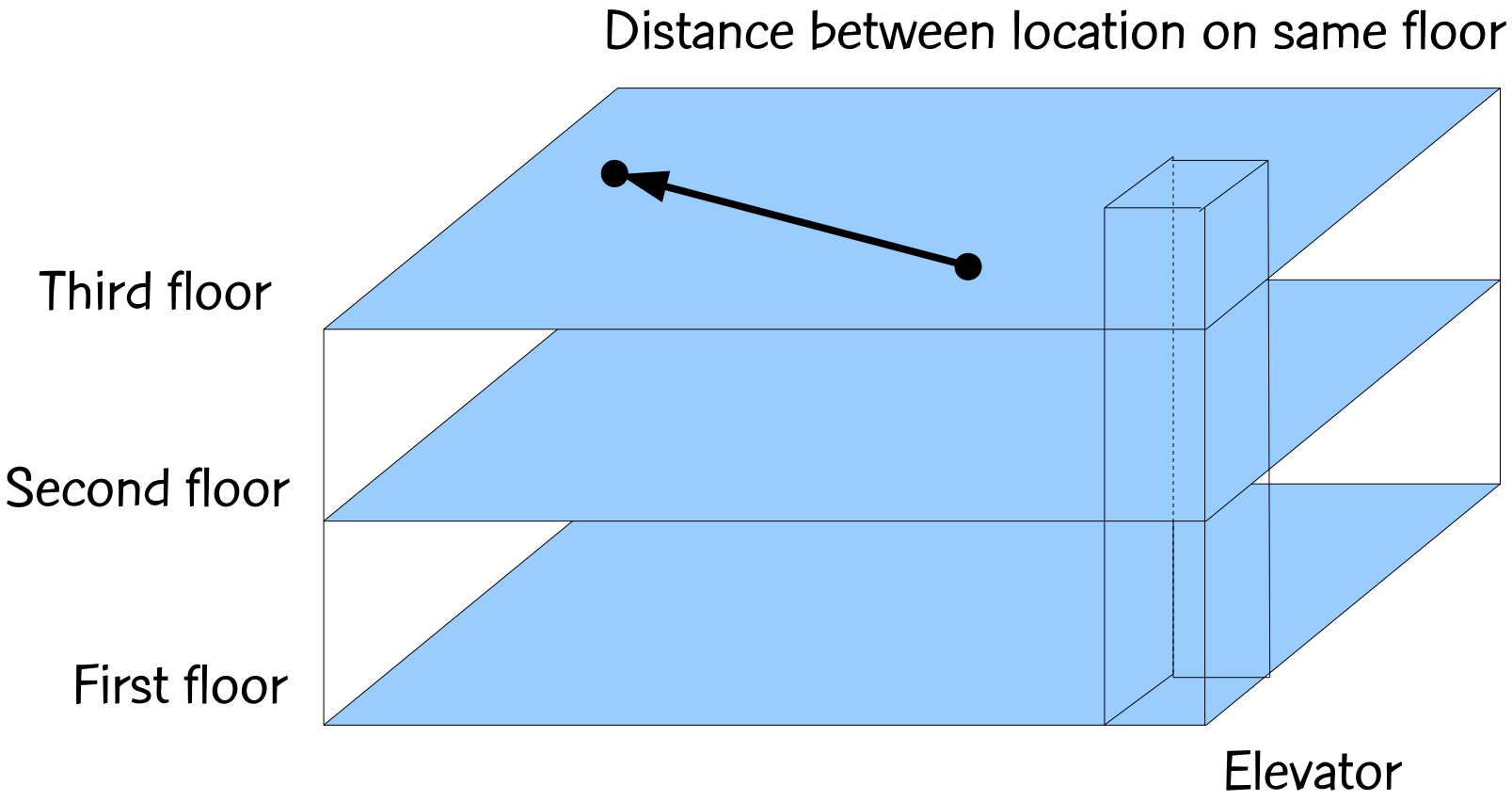
Third floor



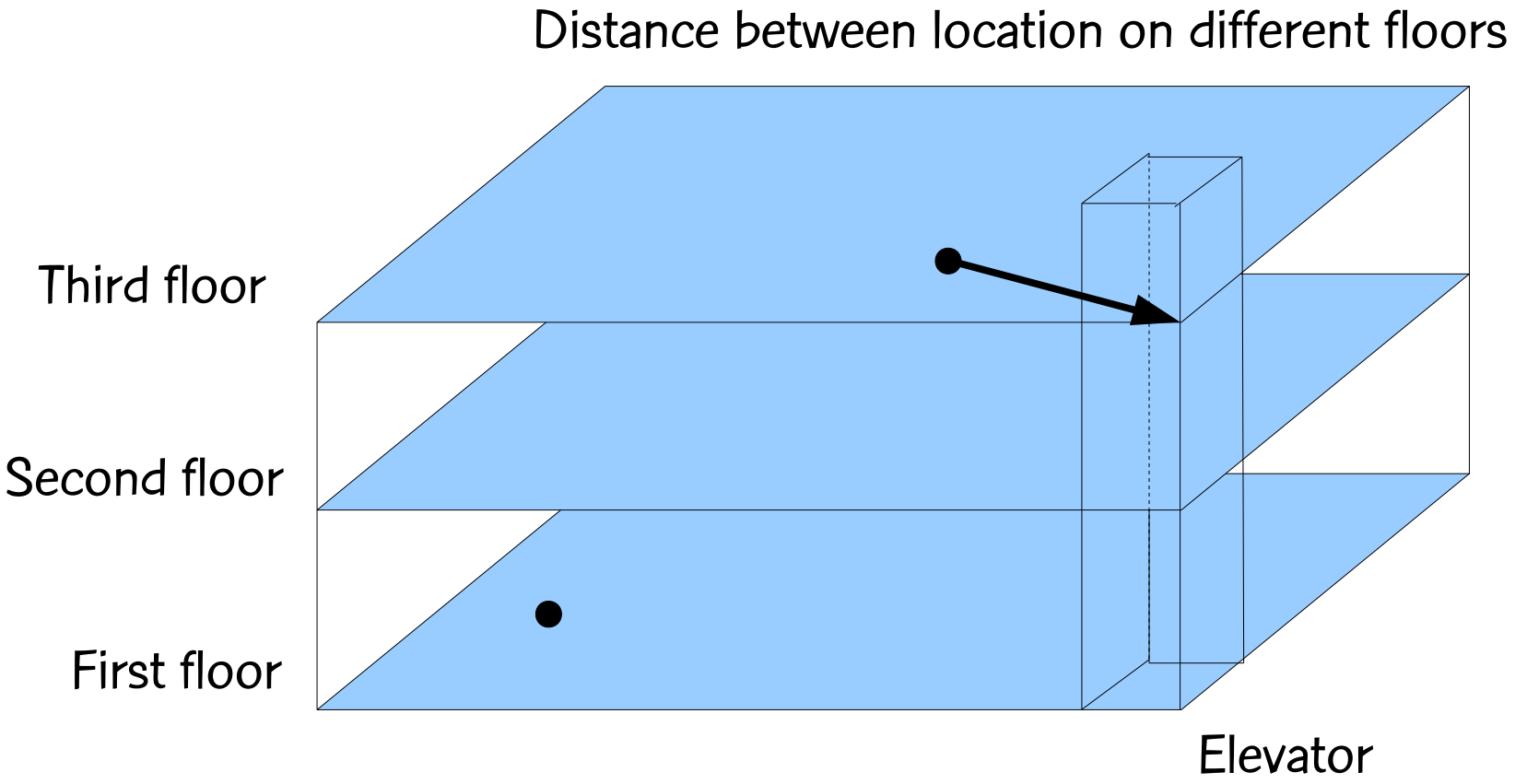
Hospital layout as a GQAP

- Distances between locations on same floor are just the Euclidean distances between the centers of the locations.
- Distances between locations on different floors are the sums of the Euclidean distance between the center of the the first location to the elevator on that floor, the distance traveled by elevator (penalized), and the Euclidean distance between the elevator on the other floor and the center of the second location.

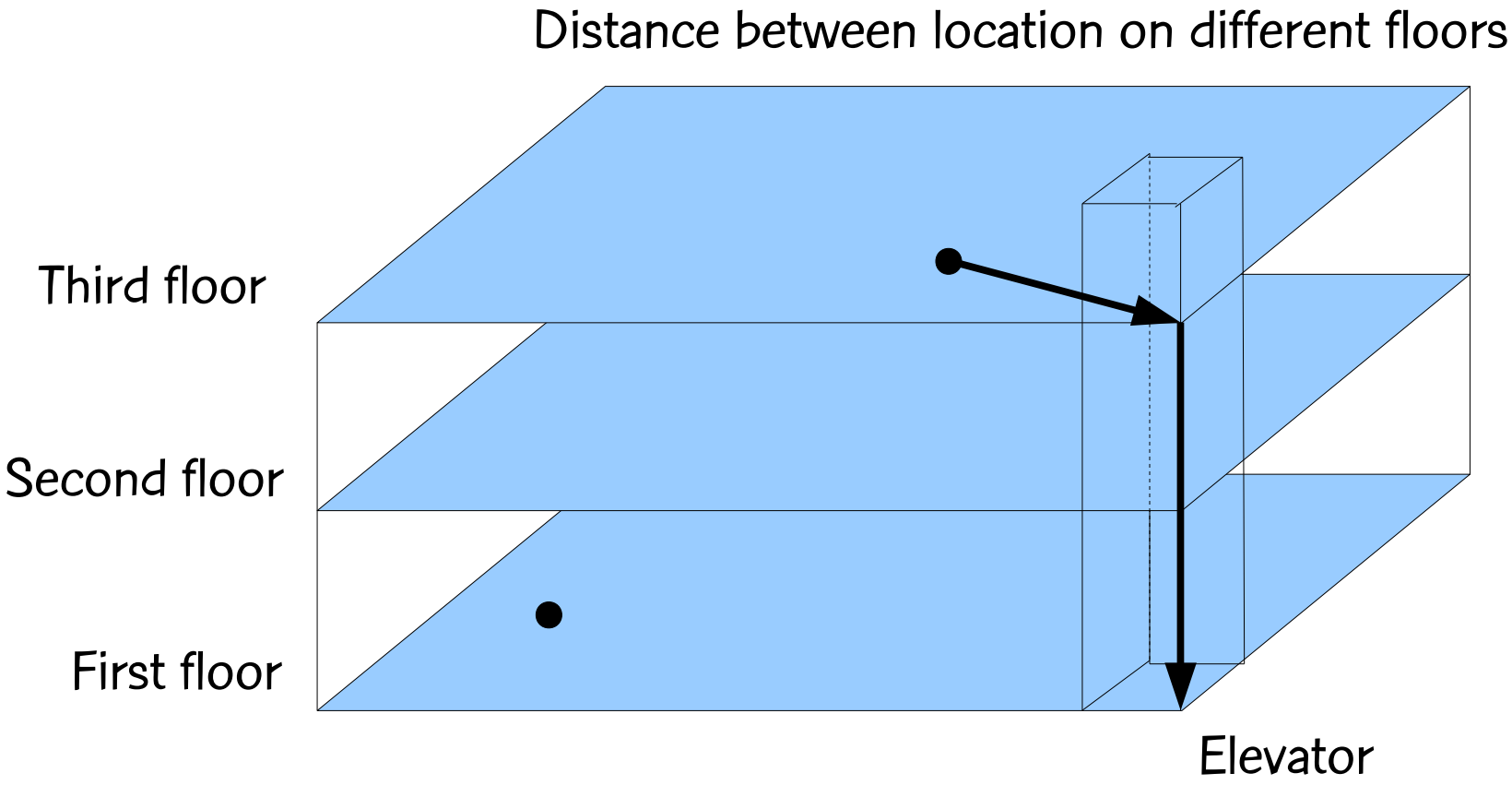
Hospital layout as a GQAP



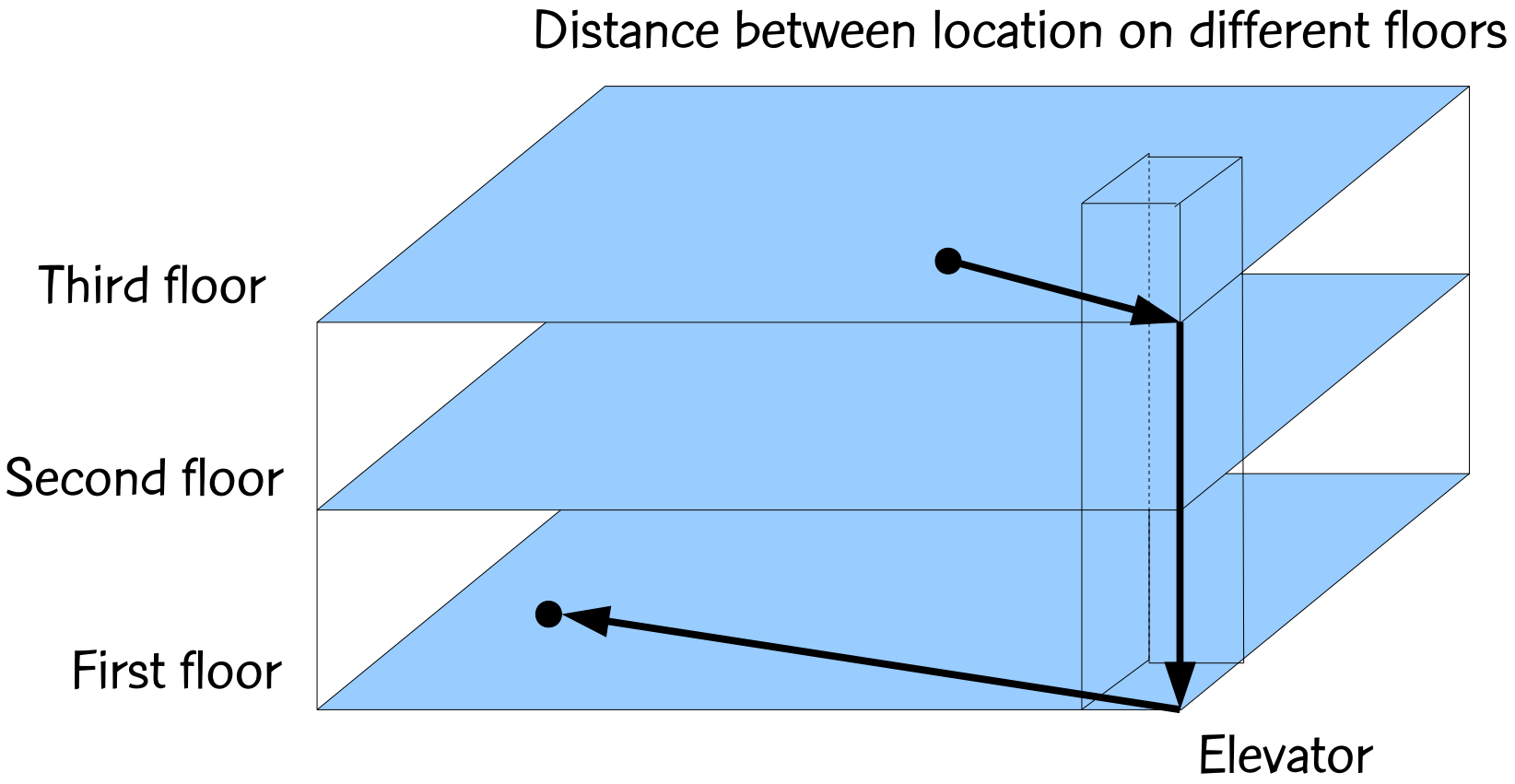
Hospital layout as a GQAP



Hospital layout as a GQAP

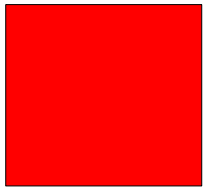


Hospital layout as a GQAP

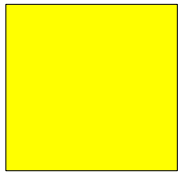


Hospital layout as a GQAP

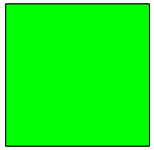
ICU ::: quantity: 3 ::: area 135 m^2



Pediatric ICU ::: quantity: 6 ::: area 110 m^2



Operating room ::: quantity: 12 ::: area 90 m^2



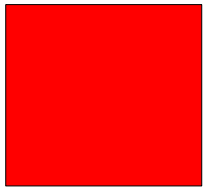
Radiology ::: quantity: 12 ::: area 65 m^2

Physician's office ::: quantity: 15 ::: area 45 m^2

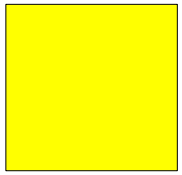
Hospital layout as a GQAP

ICU ::: quantity: 3 ::: area 135 m^2

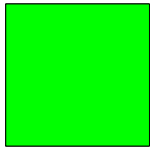
Inter-facility traffic is given



Pediatric ICU ::: quantity: 6 ::: area 110 m^2



Operating room ::: quantity: 12 ::: area 90 m^2



Radiology ::: quantity: 12 ::: area 65 m^2

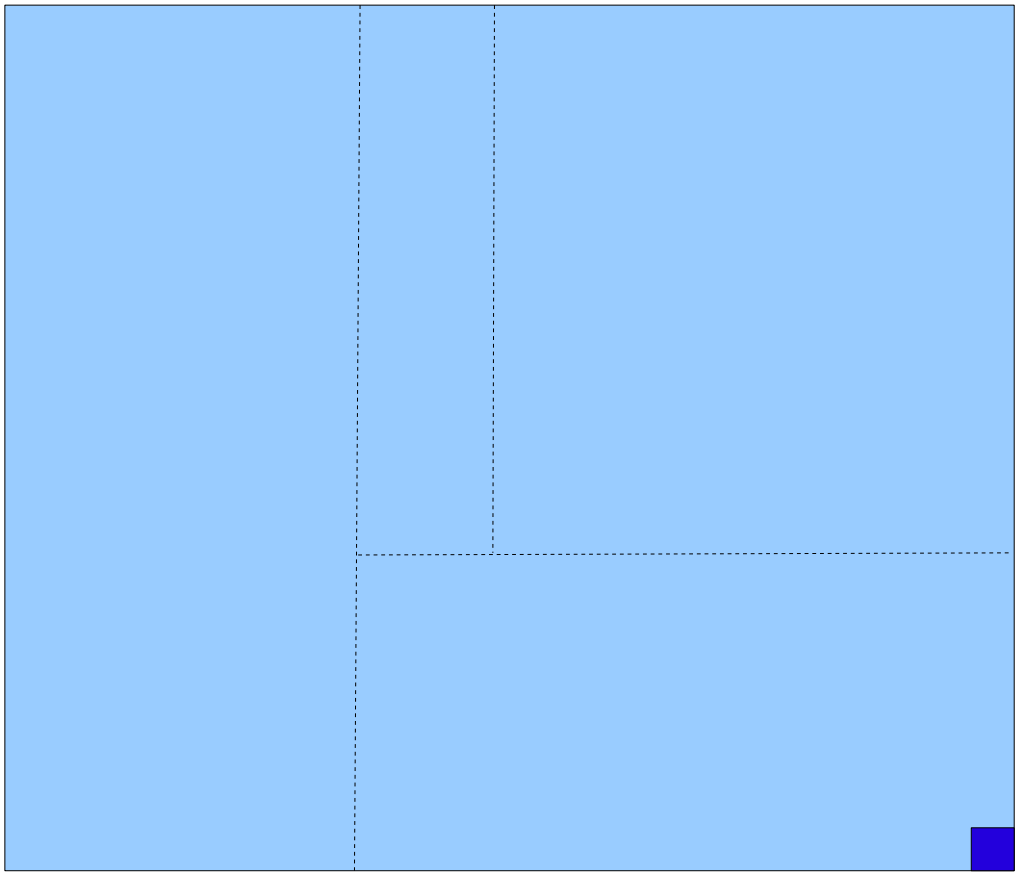
Physician's office ::: quantity: 15 ::: area 45 m^2

Hospital layout as a GQAP

- Applying GRASP with path-relinking heuristic, the following assignment was found in 1898.4 secs on a 2.6 Ghz machine.

Hospital layout as a GQAP

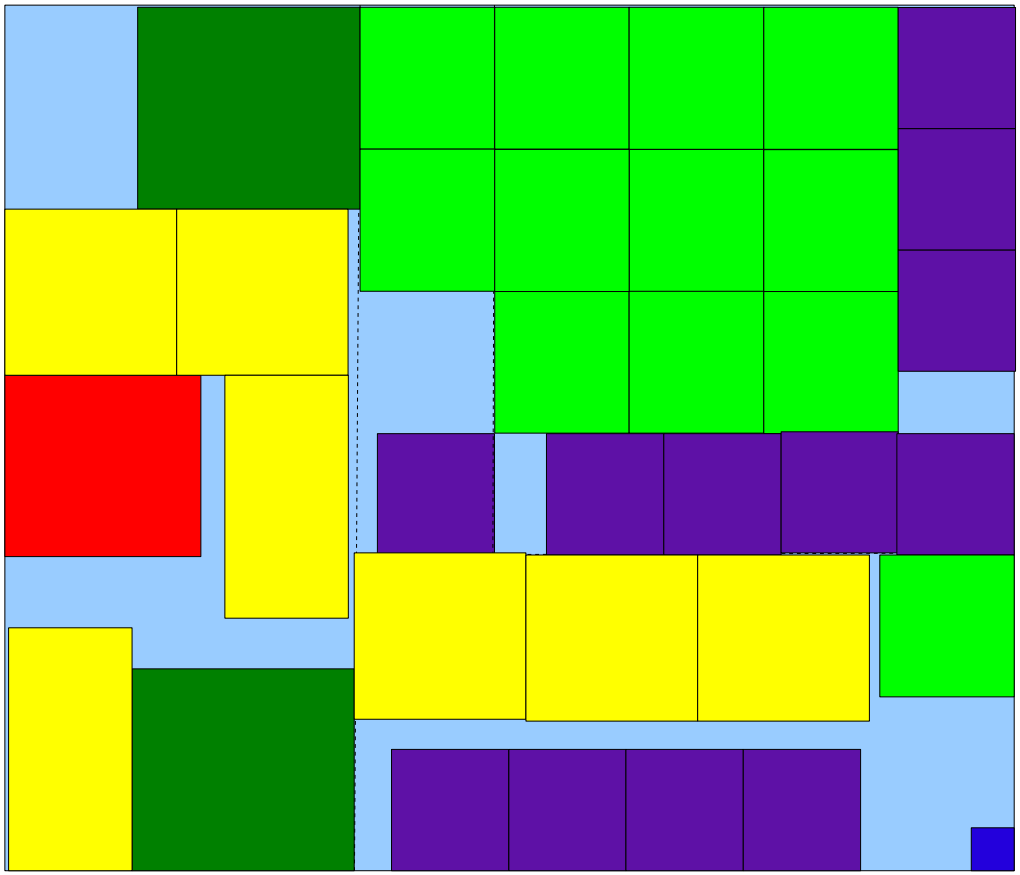
Third floor



Elevator

Hospital layout as a GQAP

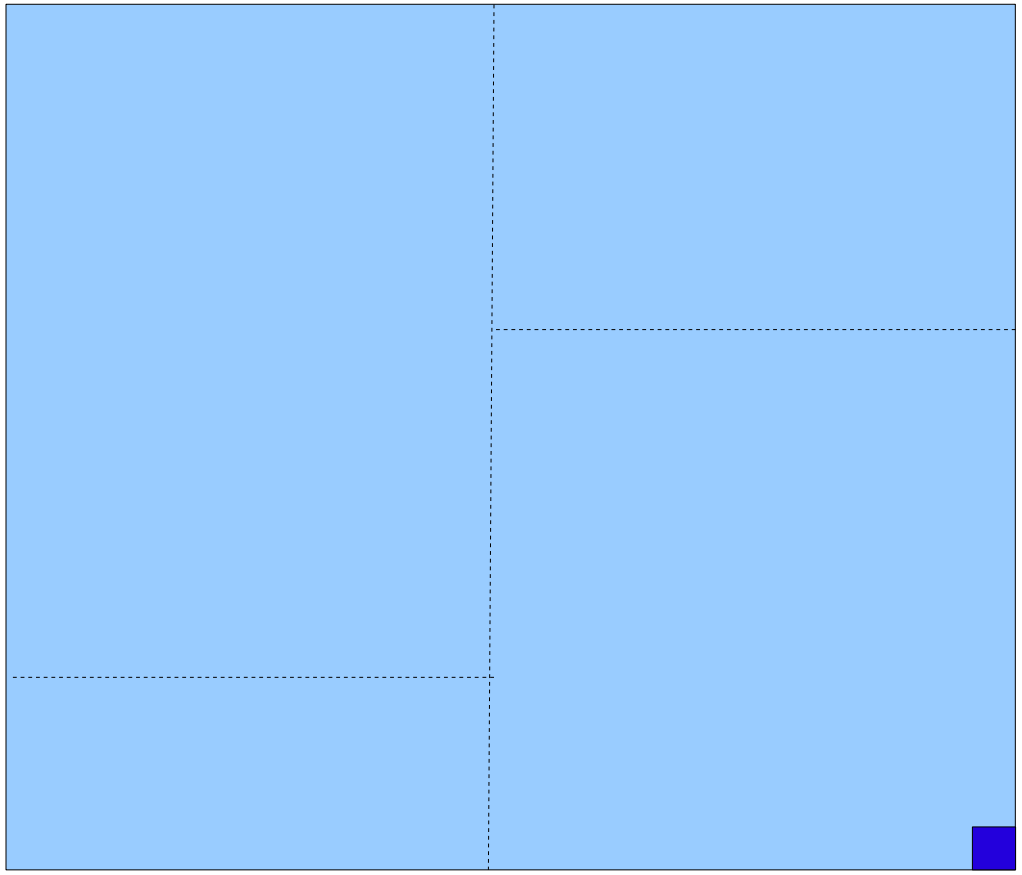
Third floor



Elevator

Hospital layout as a GQAP

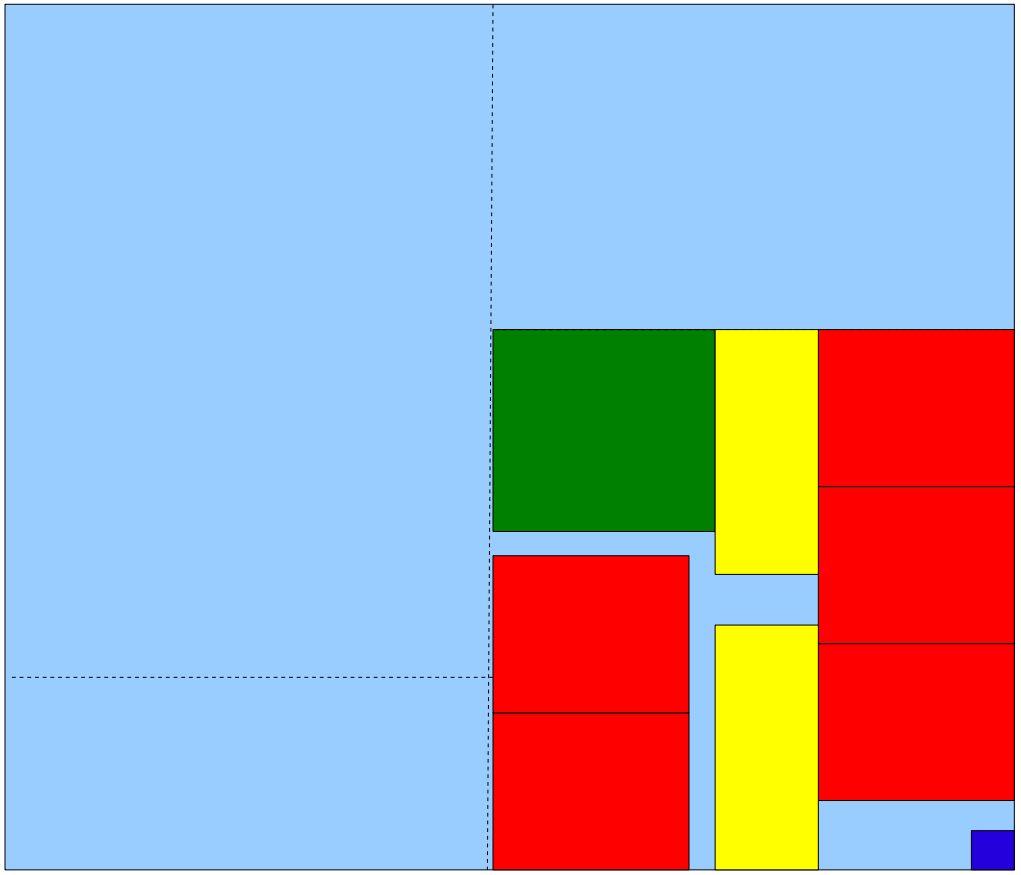
Second floor



Elevator

Hospital layout as a GQAP

Second floor



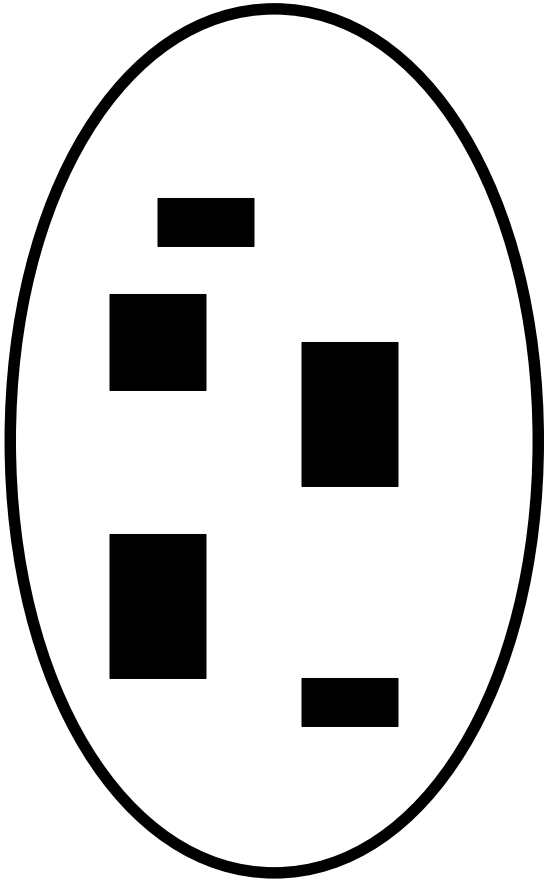
Elevator

Generalized quadratic assignment problem

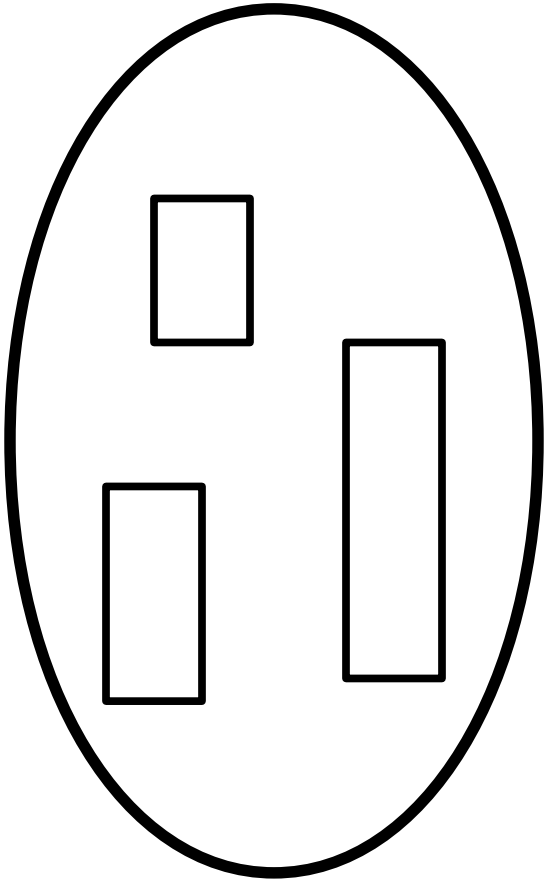
Generalized quadratic assignment

- The GQAP is NP-hard.
- It is a generalization of the quadratic assignment problem (QAP).
- Multiple facilities can be assigned to a single location as long as the capacity of the location allows.

N: set of n facilities



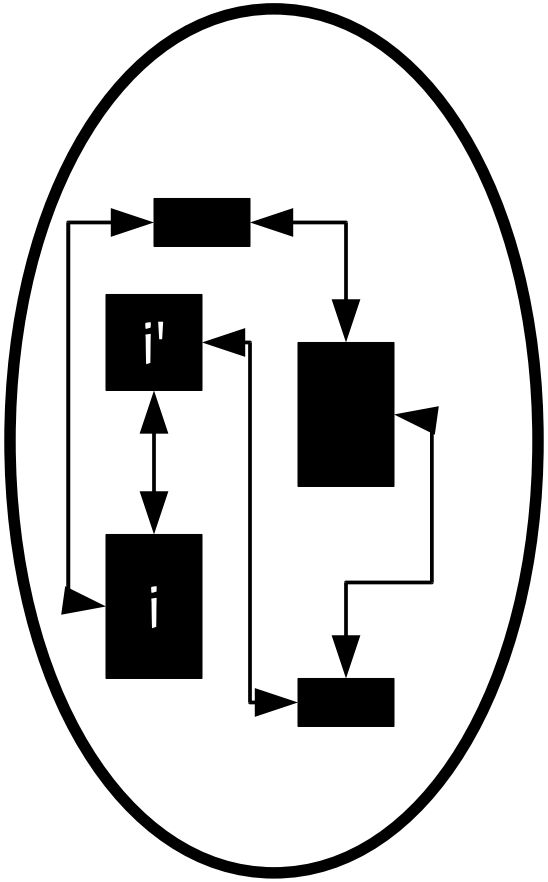
M: set of m locations



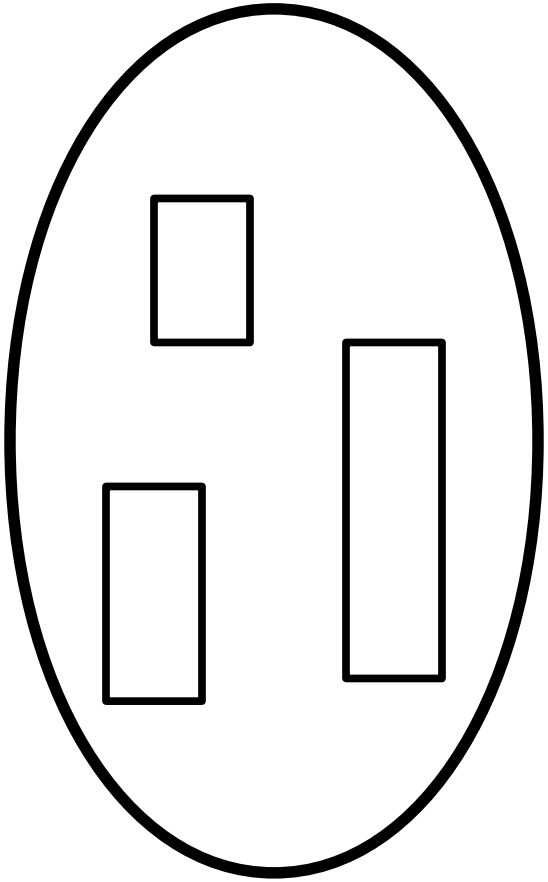
d_i : capacity demanded by facility $i \in N$

Q_j : capacity of location $j \in M$

N: set of n facilities

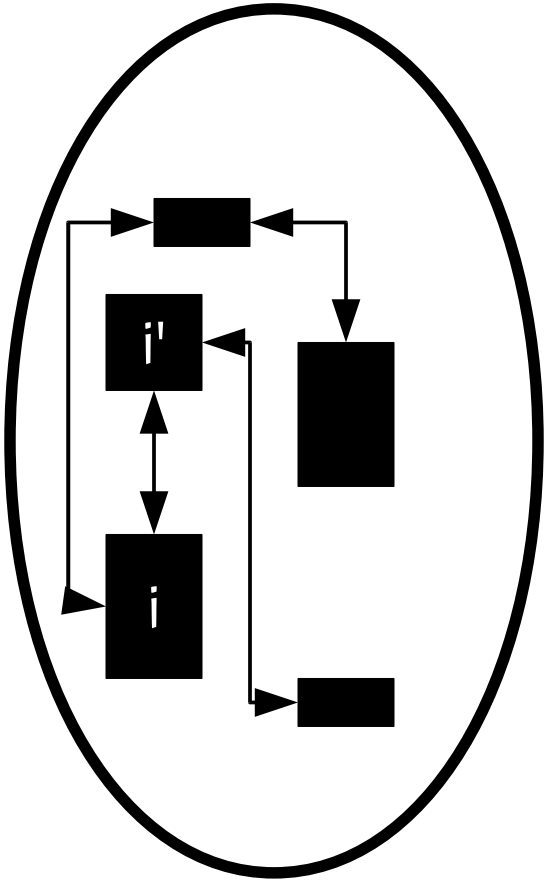


M: set of m locations

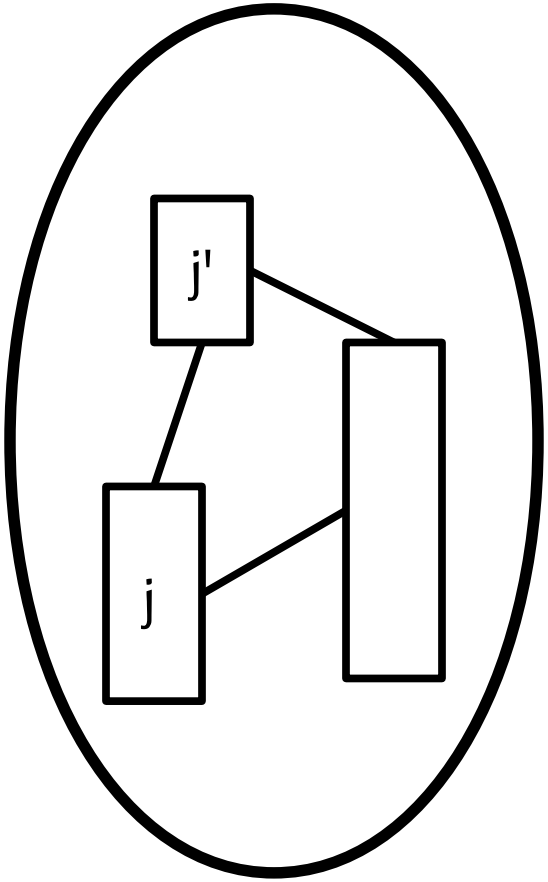


$A_{nxn} = (a_{ii'})$: flow between facilities

N: set of n facilities



M: set of m locations

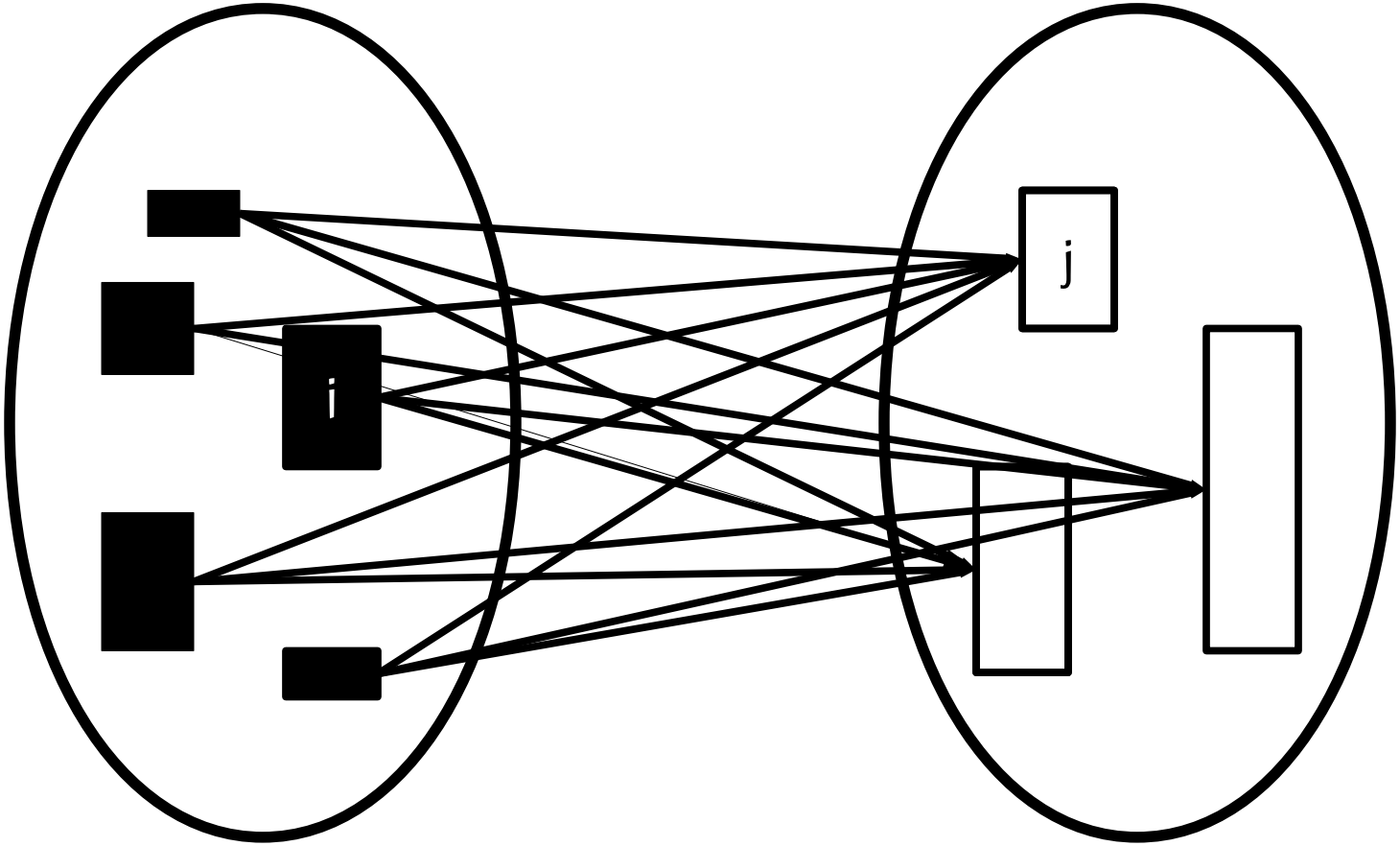


$A_{nxn} = (a_{ii'})$: flow between facilities

$B_{mxm} = (b_{jj'})$: distance between locations

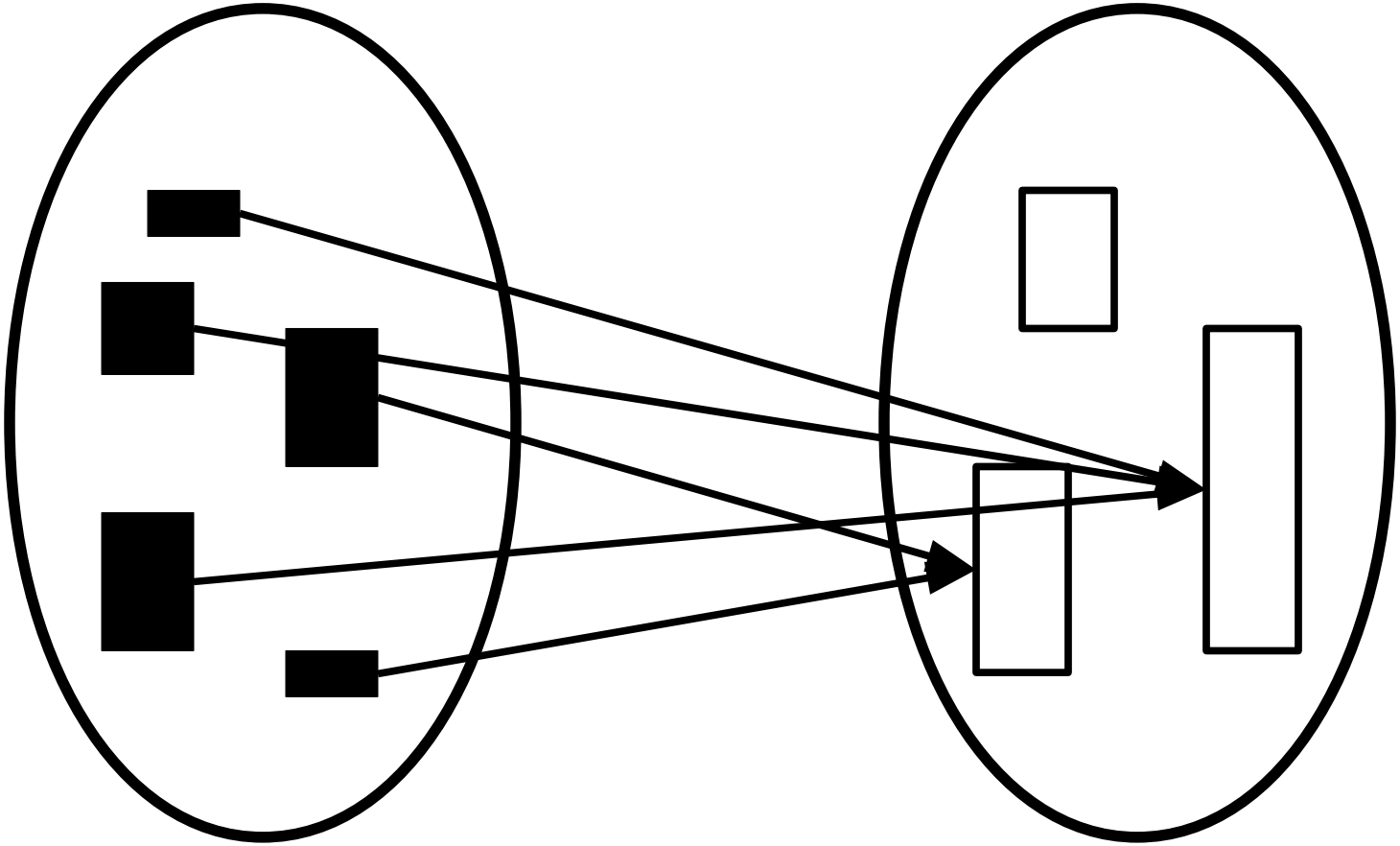
N: set of n facilities

M: set of m locations



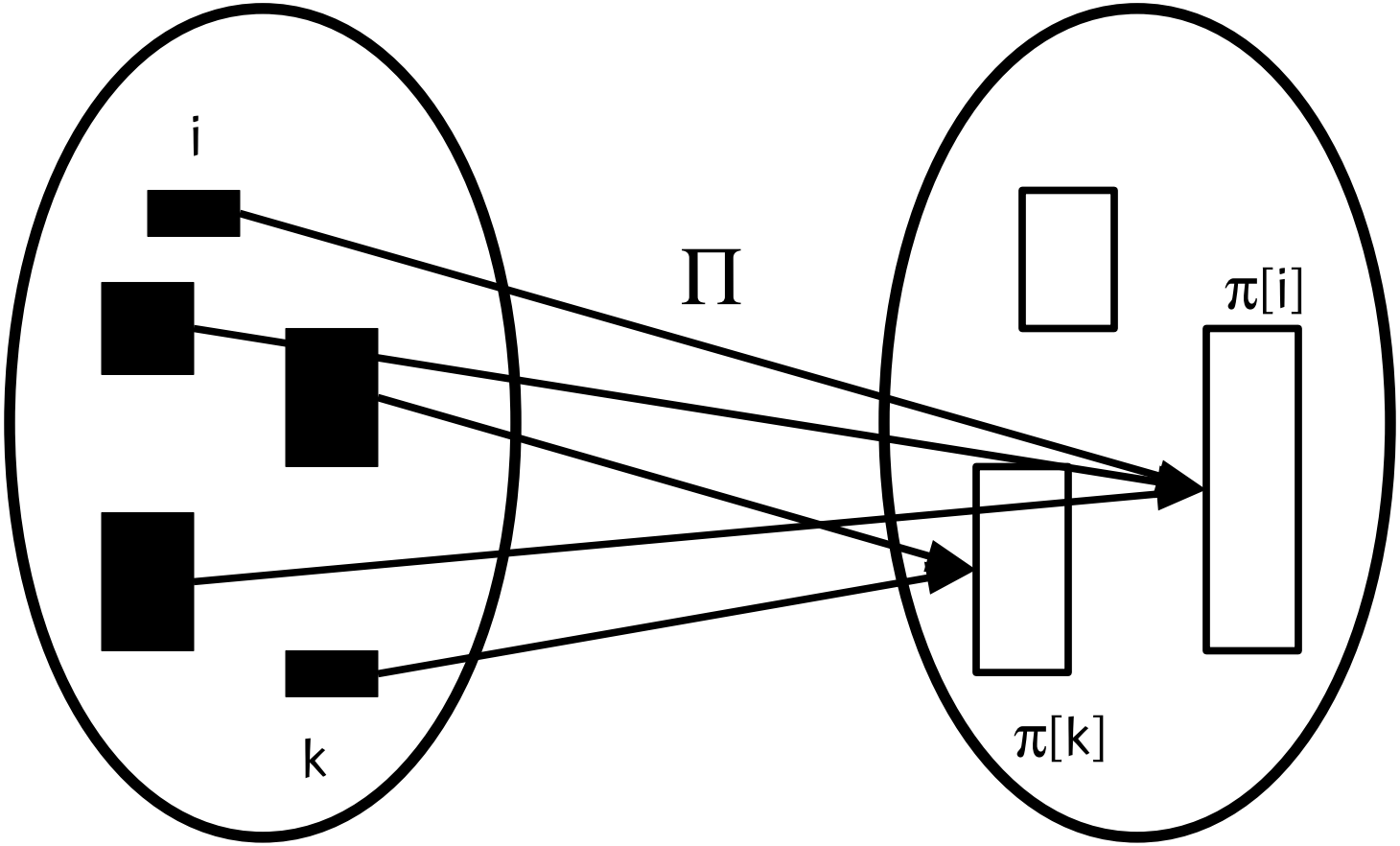
$C_{nxm} = (c_{ij})$: cost of assigning facility $i \in N$ to location $j \in M$

The generalized quadratic assignment problem



GQAP seeks a assignment, without violating the capacities of locations, that minimizes the sum of products of flows and distances in addition to a linear total cost of assignment.

The generalized quadratic assignment problem



$$cost[\Pi] = \sum_{i=1,n} c[i,\pi[i]] + \sum_{i=1,n} \sum_{i \neq k=1,n} F[i,k]*D[\pi[i],\pi[k]]$$

Solution method

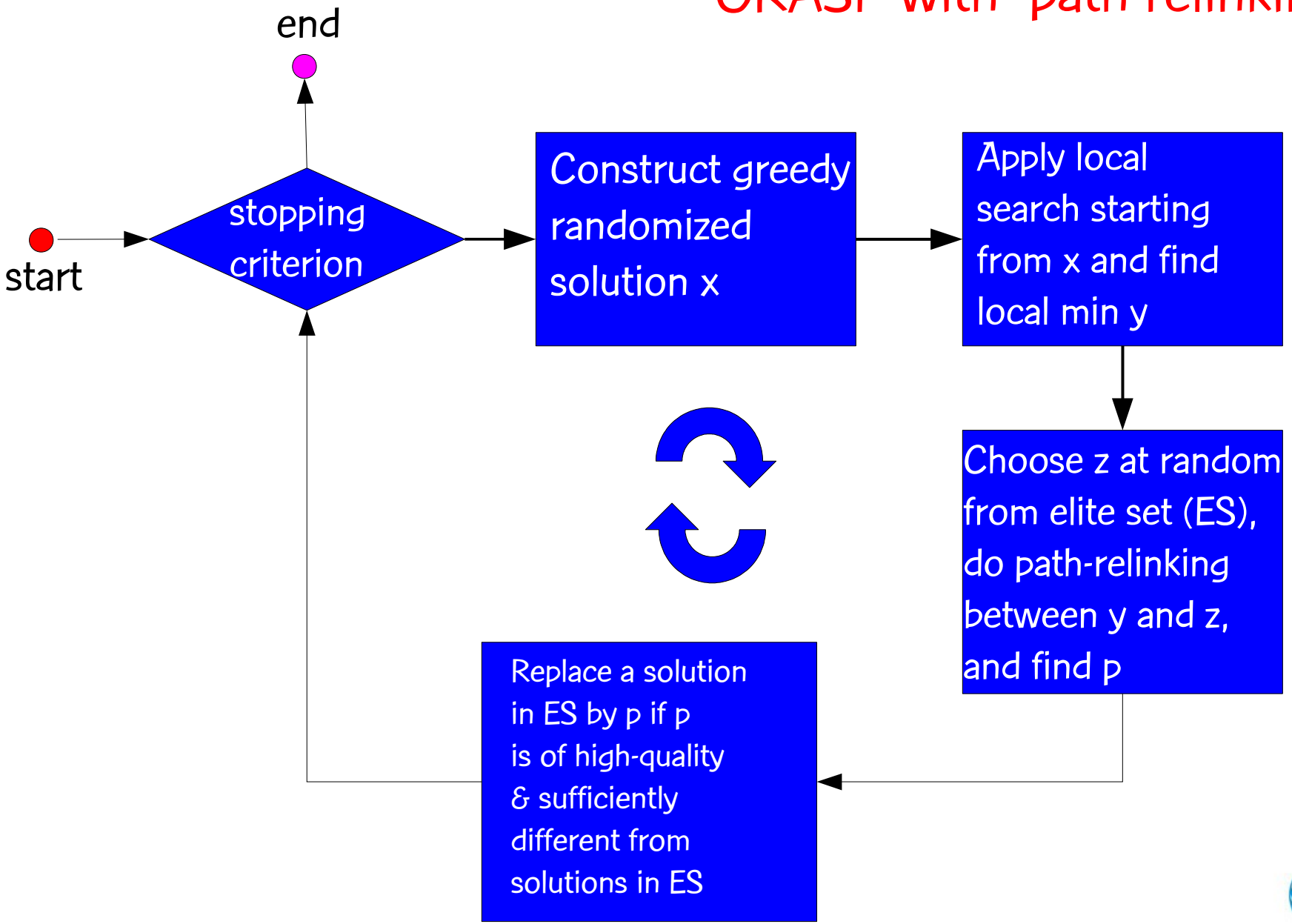
Paper

G.R. Mateus, R.M.A. Silva, and M.G.C. Resende, "GRASP with path-relinking for the generalized quadratic assignment problem," J. of Heuristics, published online 1 September 2010.

Tech report:

http://www2.research.att.com/~mgcr/doc/gpr-gqap.pdf

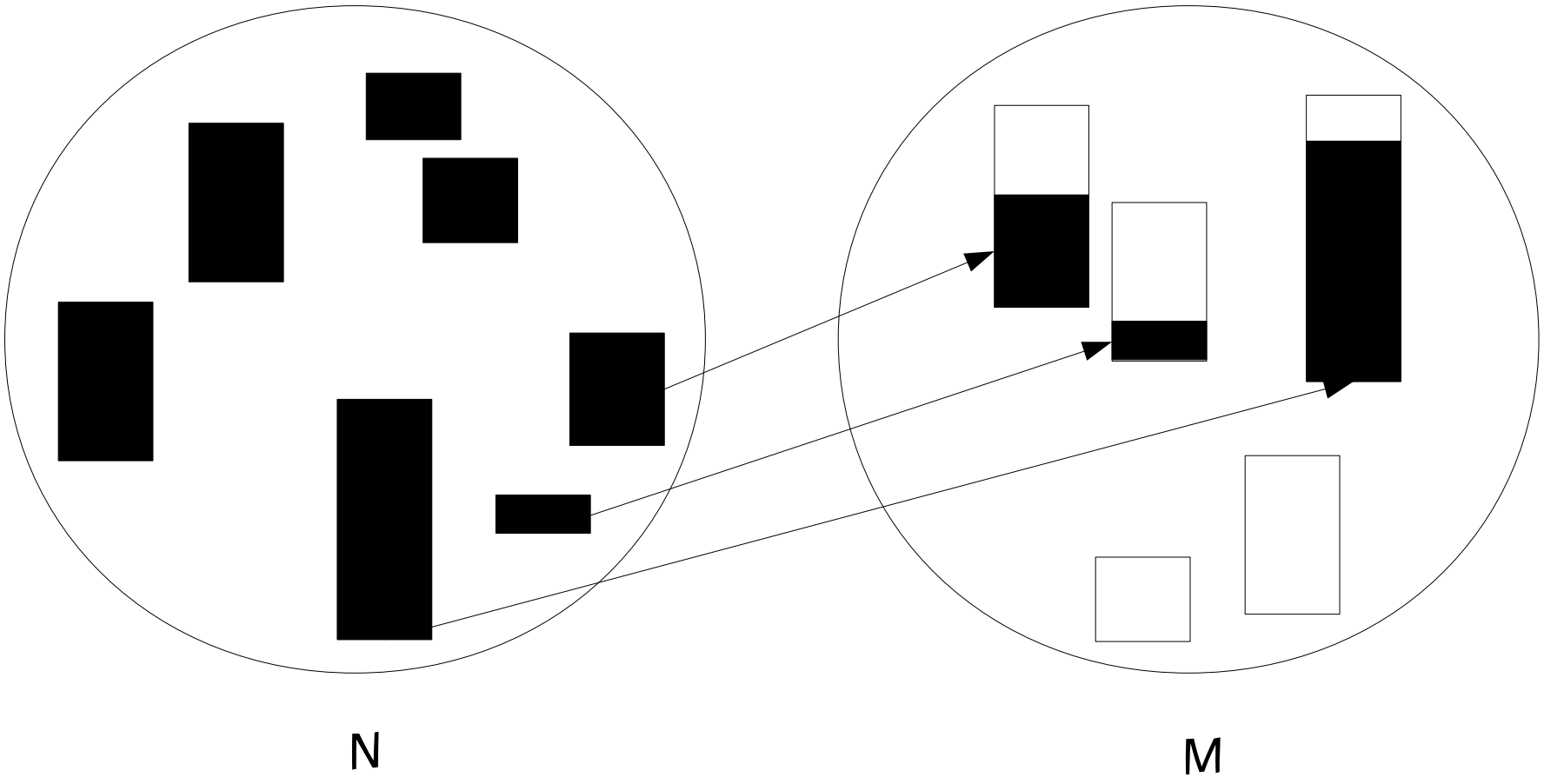
GRASP with path-relinking



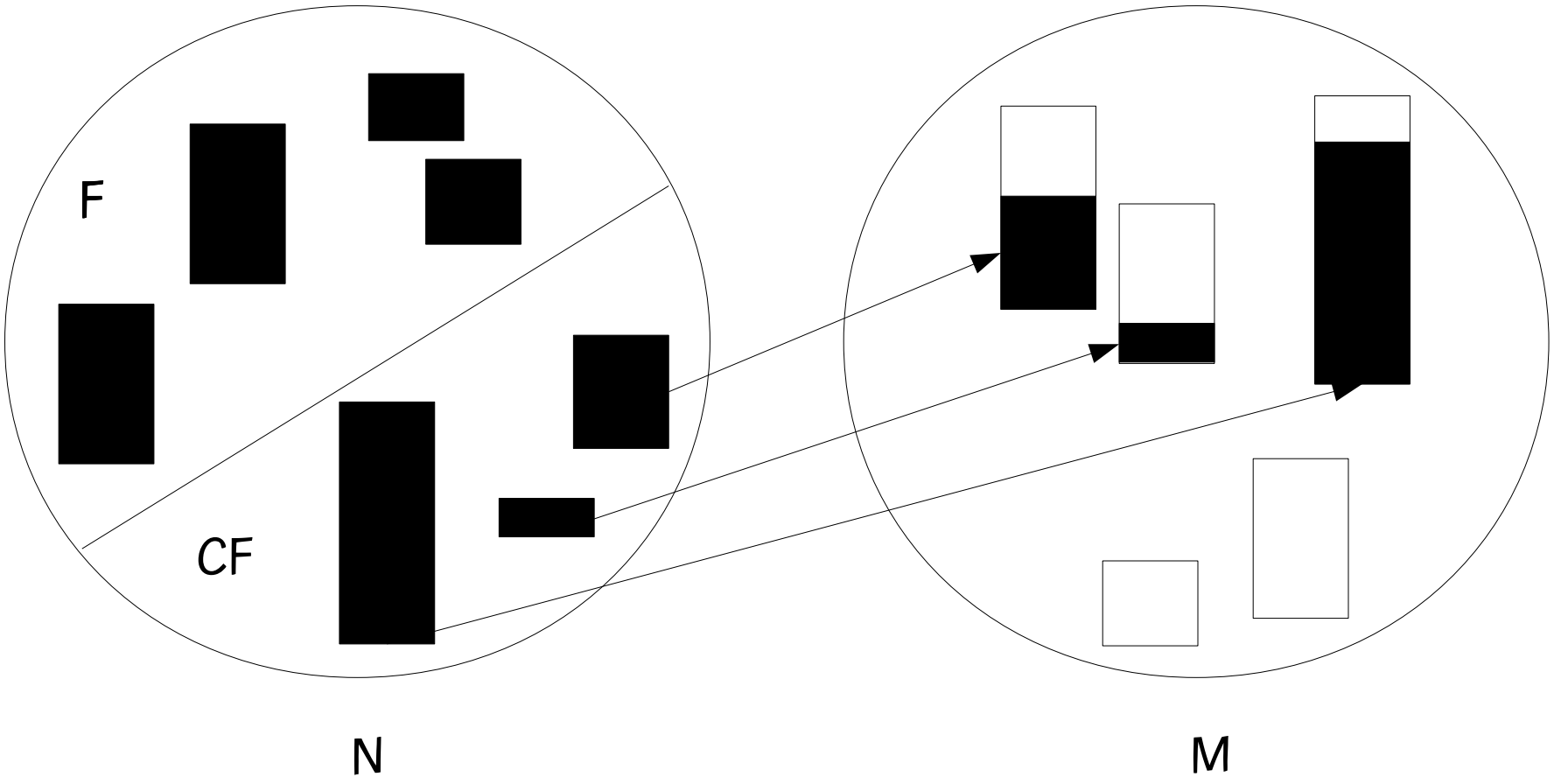
Components

- Construction of greedy randomized solution
- Local search
- Path-relinking

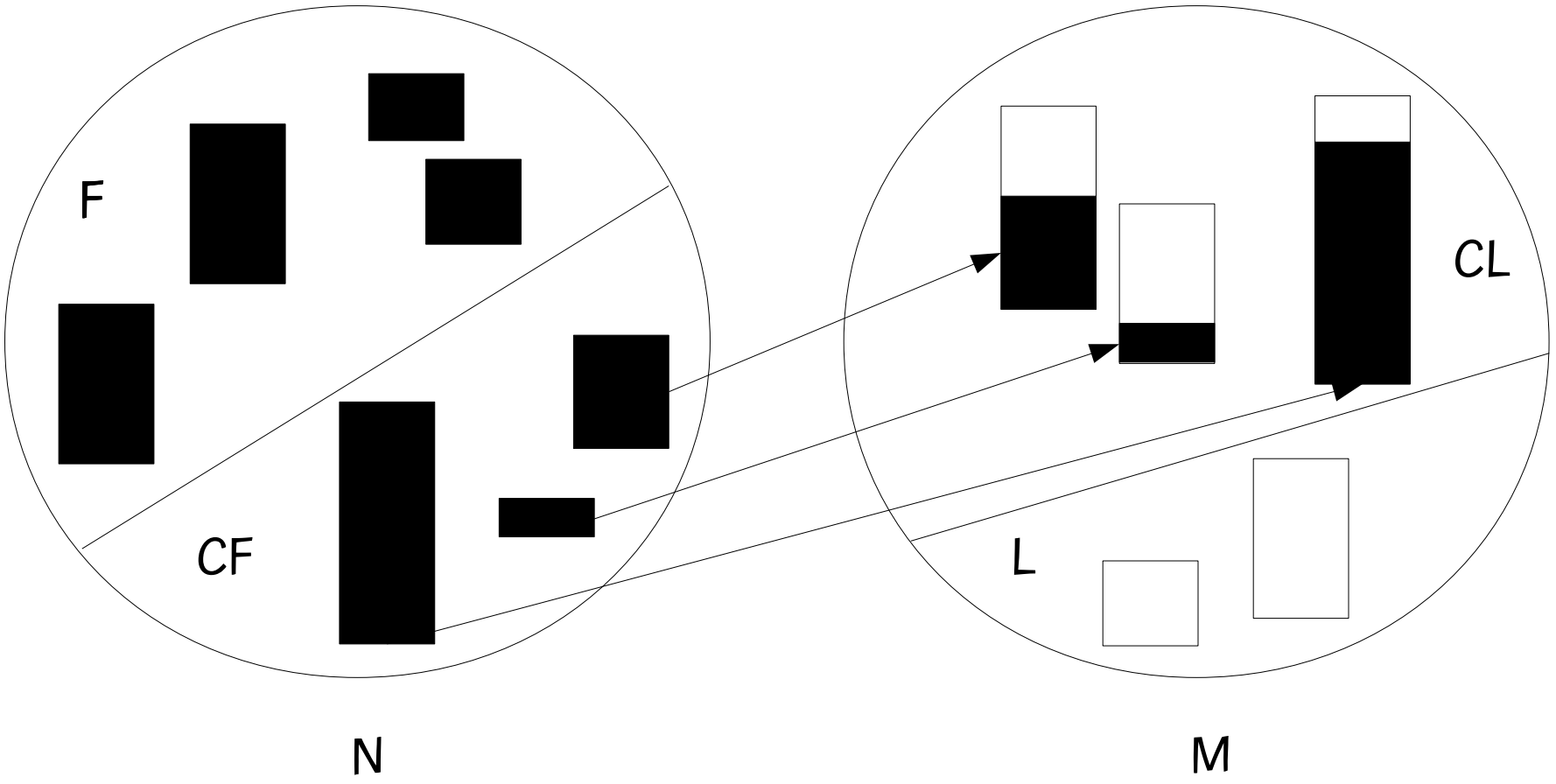
GRASP construction



Suppose a number of assignments have already been made

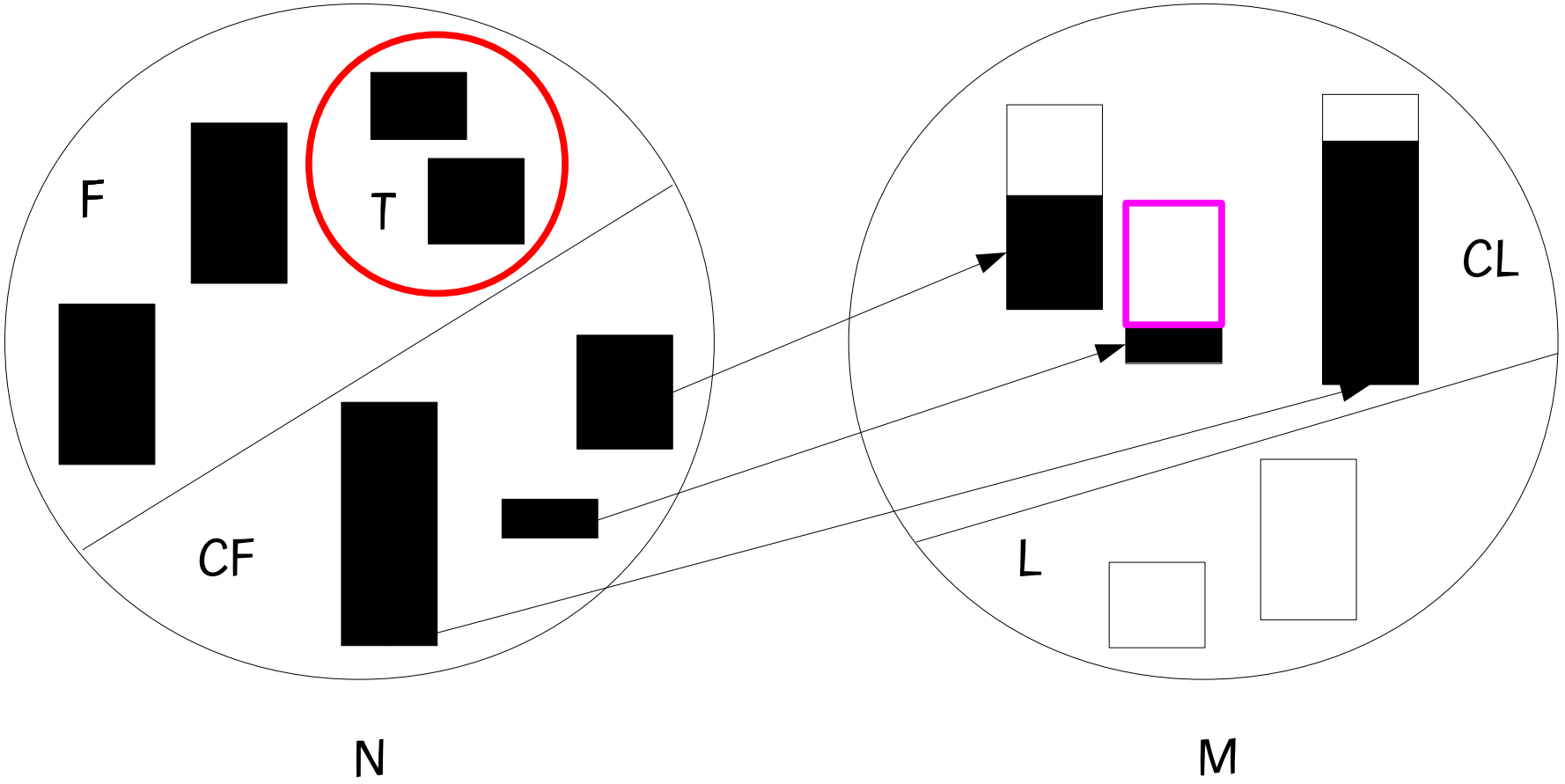


N = F ∪ CF, where CF is the set of assigned facilities and F the set of facilities not yet assigned to some location



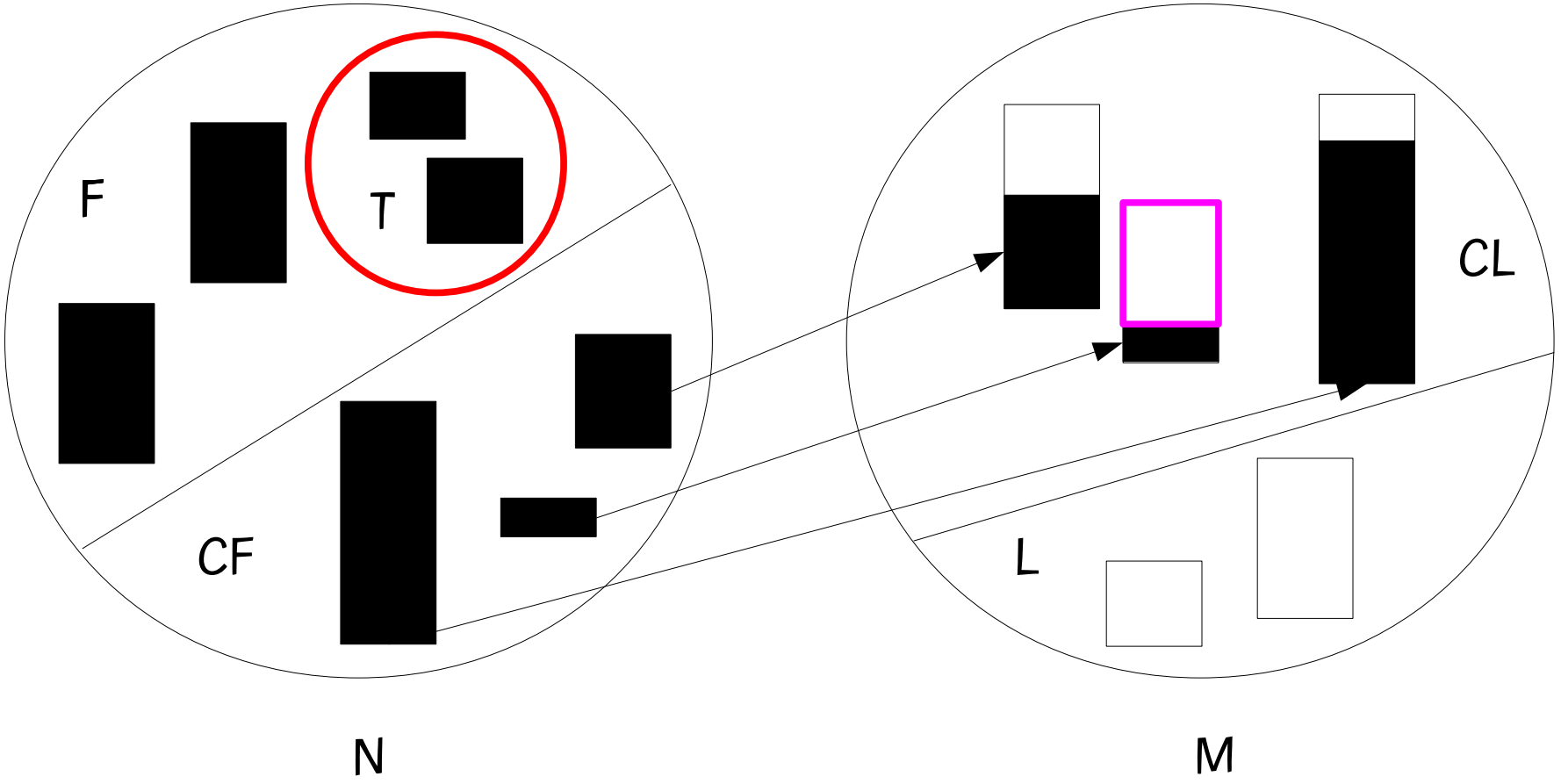
$M = L \cup CL$, where CL is the set of previously chosen locations and L the set of unselected locations.

Procedure to select a new location from set L



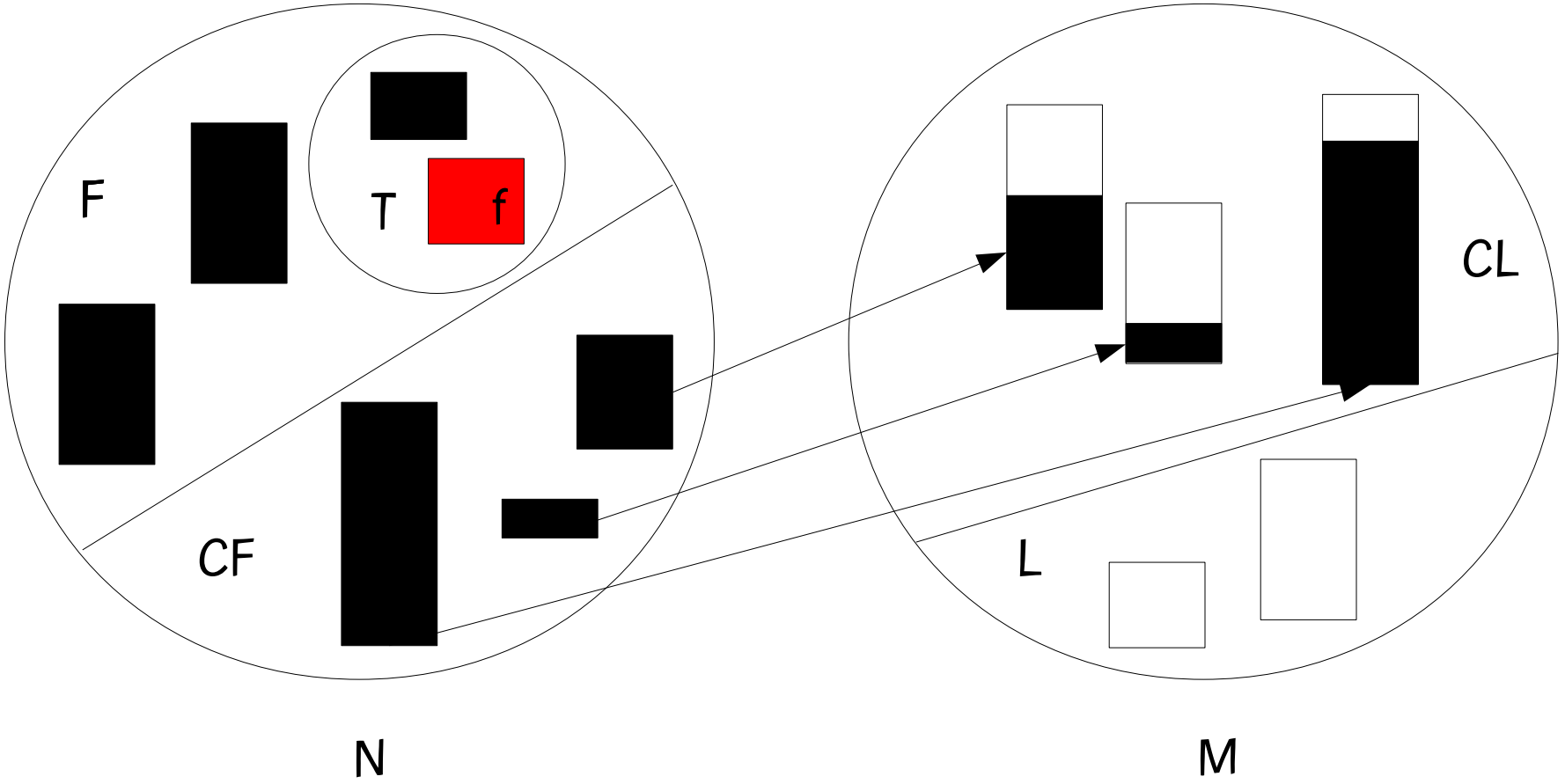
With probability 1– (|T|/|F|), randomly select a new location l from L, where the set T consists of all unassigned facilities with demands less than or equal to the maximum available capacity of locations in CL and move location l to CL

Procedure to select a new location from set L



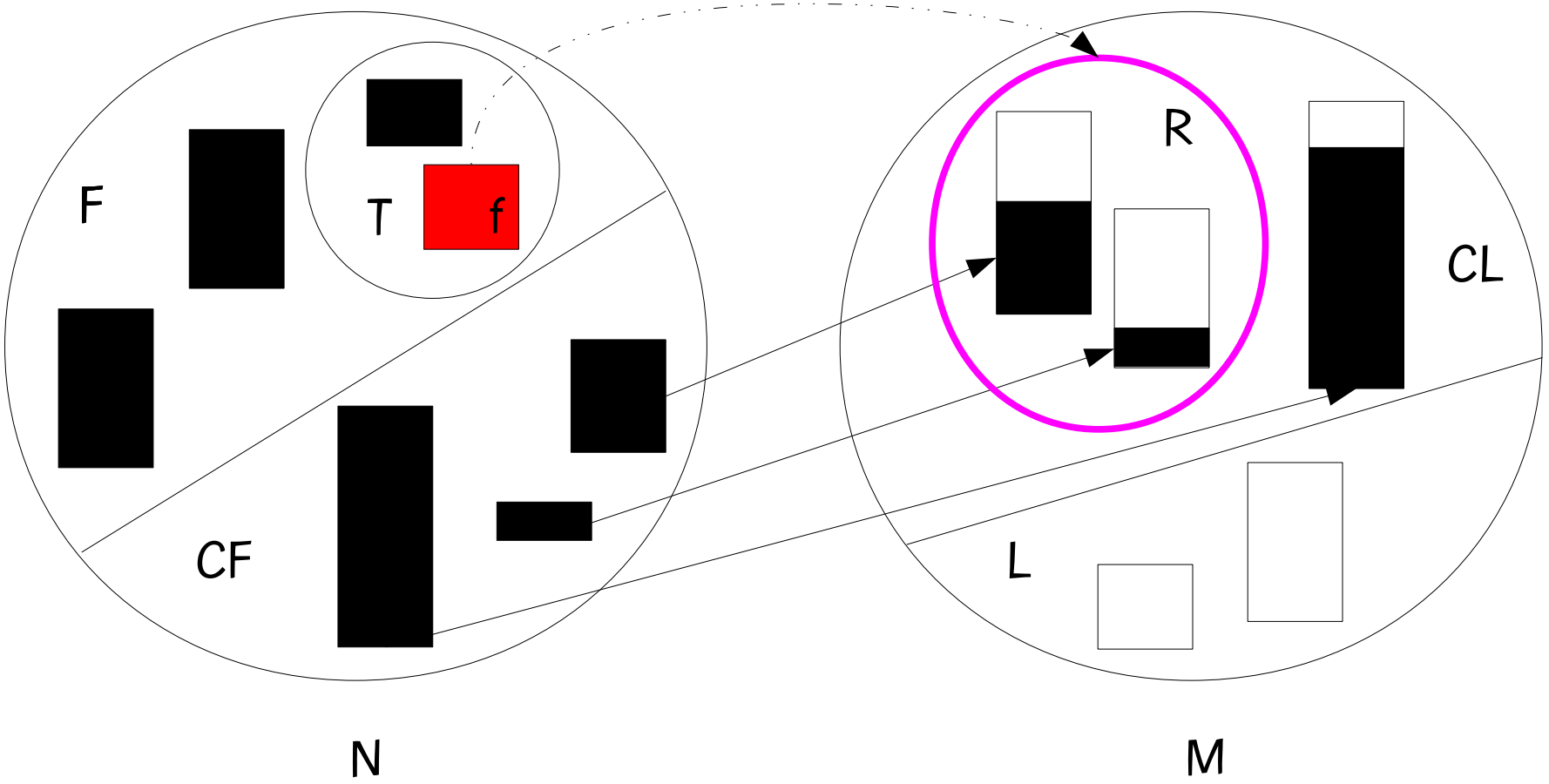
Favor locations in L that have high available capacity and that are close to all locations in CL

Facility selection procedure



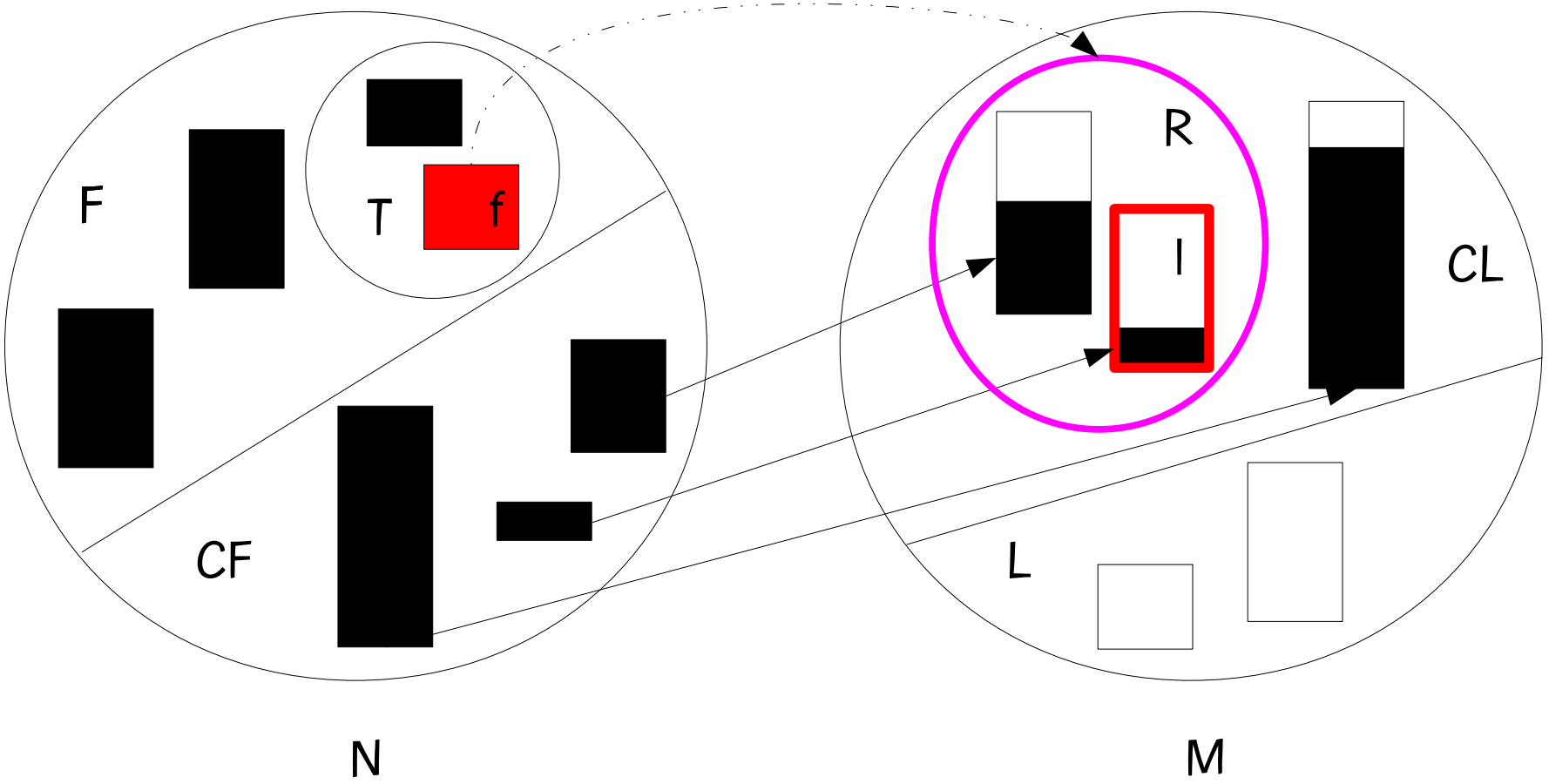
Randomly select a facility $f \in T$ favoring facilities that have high demand and high flows to other facilities.

Procedure to select a location from CL (step 1)



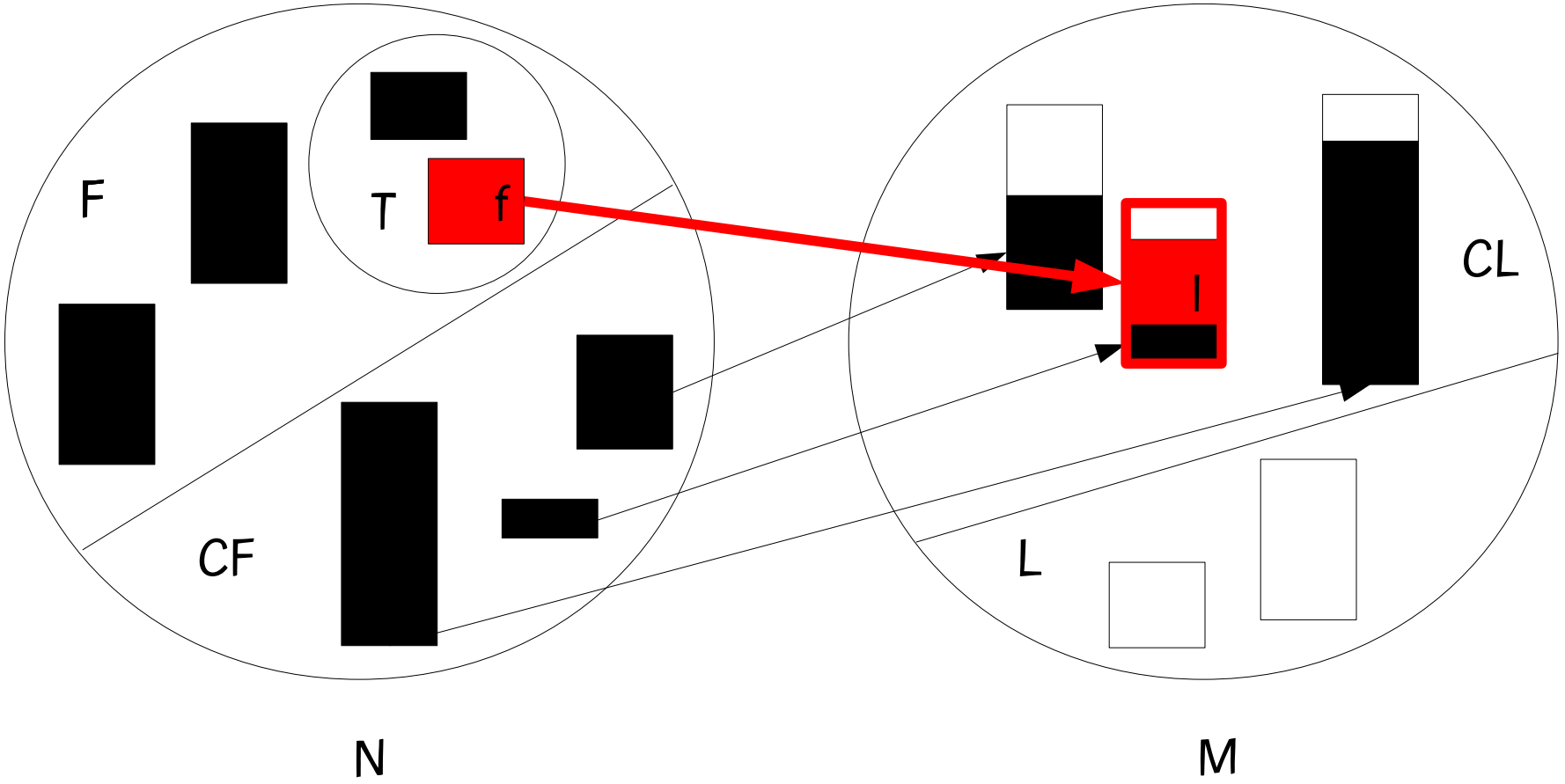
1. Let set R to be all locations in CL having slack greater than or equal to demand of facility f;

Procedure to select a location from CL (step 2)



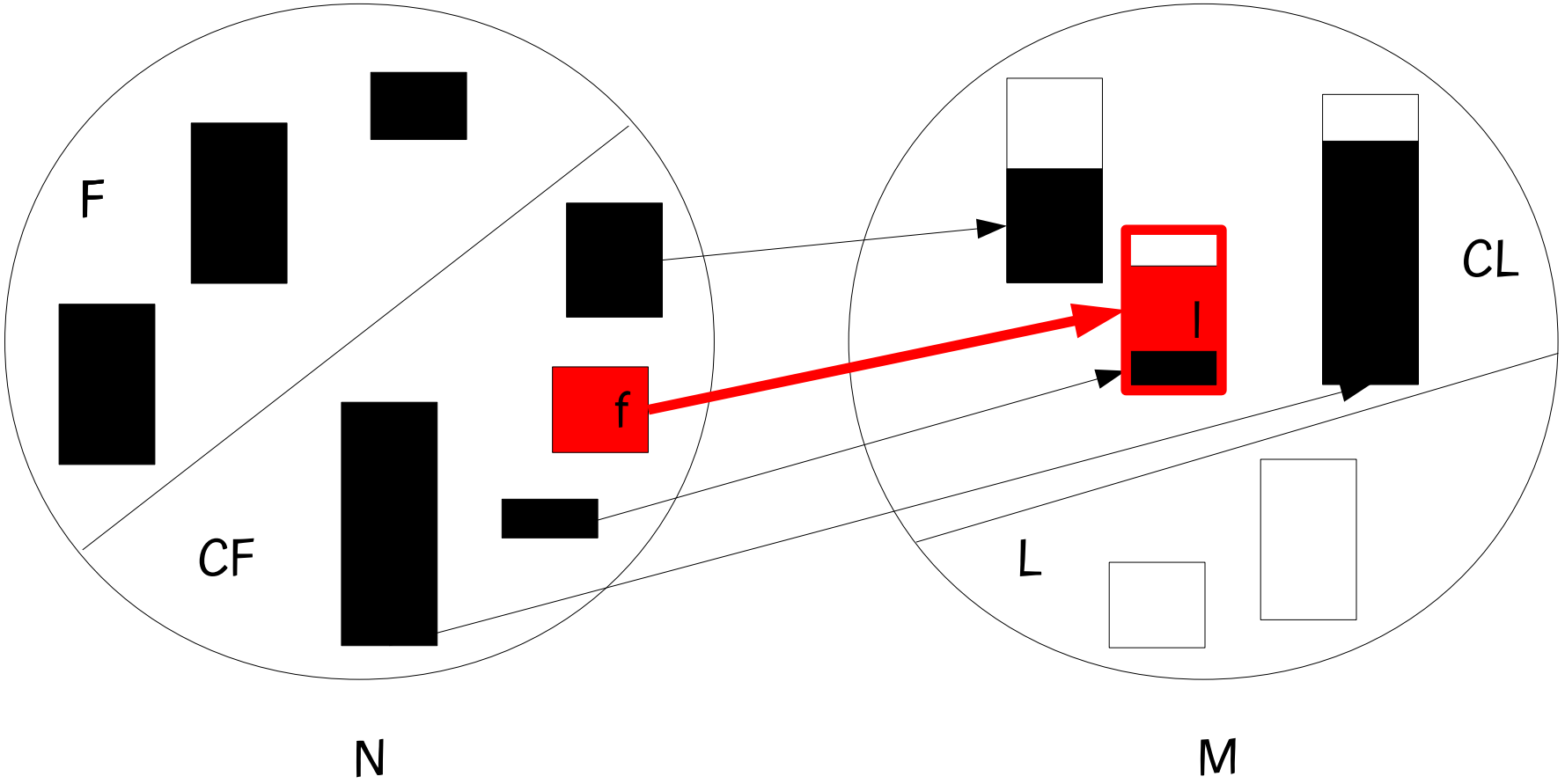
2. Randomly select a location $l \in R$ favoring those having high available capacity and those close to high-capacity locations in CL;

Assignment procedure



Assign facility f to location l

Assignment procedure



Update sets F, CF, and slack of location l

Considerations about the construction procedure

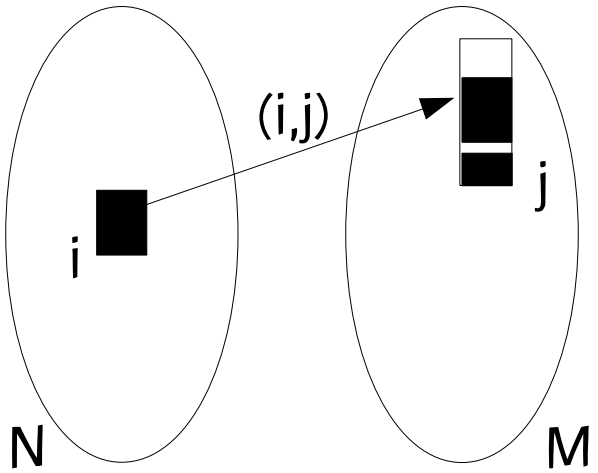
- The procedure is not guaranteed to produce a feasible solution.
- To address this difficulty, the construction procedure is repeated a maximum number of times or until all facilities are assigned (i.e. until $F=\emptyset$).
- At start of construction, a location l from L is selected with probability proportional to its capacity. Location l is placed in CL.

Local search

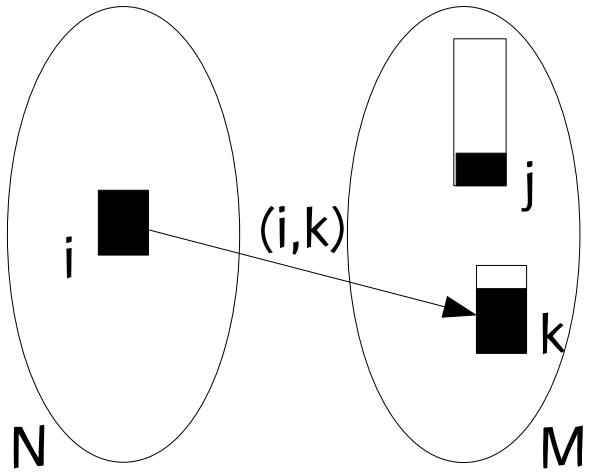
Local search

1-move and 2-move neighborhoods from solution p are used in our local search.

1-move: changing one facility-to-location assignment in p



solution p



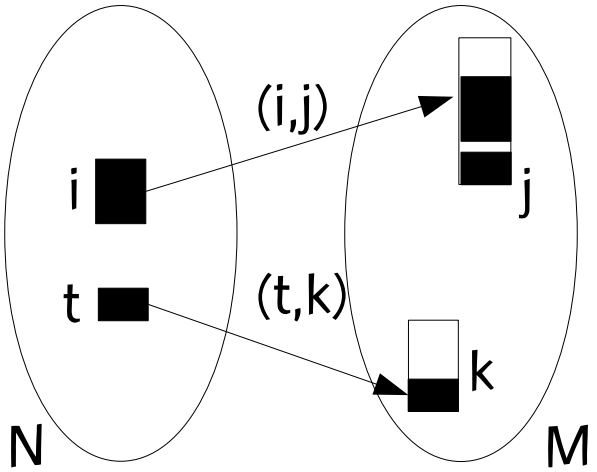
1-move neighbor of p

Local search

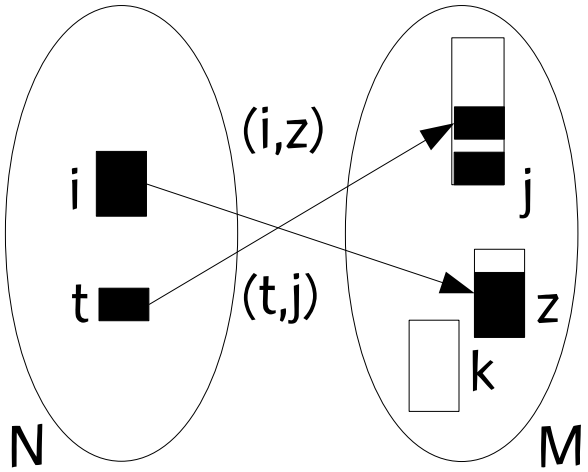
1-move and 2-move neighborhoods from solution p are used in our local search.

1-move: changing one facility-to-location assignment in p

2-move: changing two facility-to-location assignment in p.

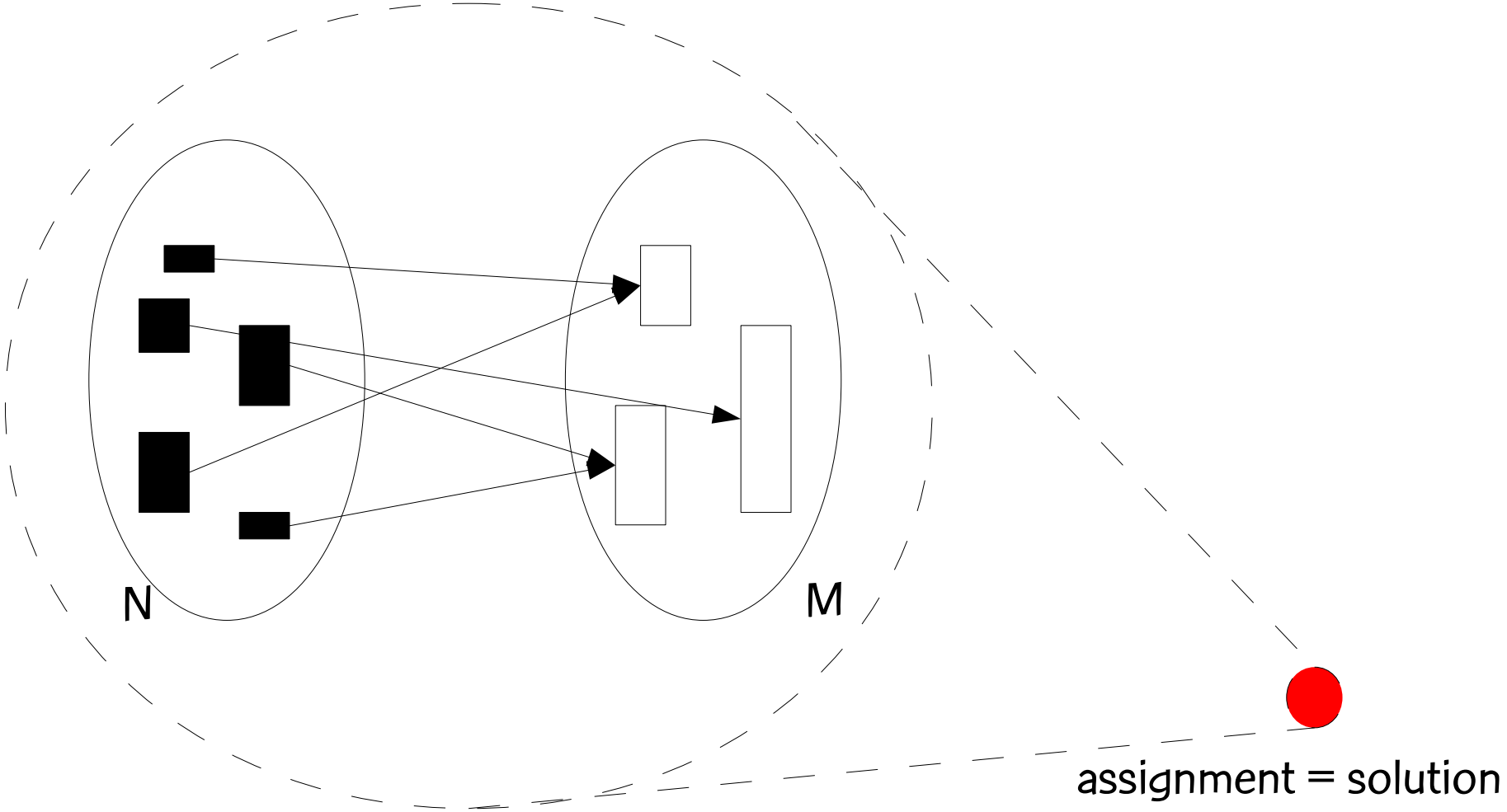


solution p

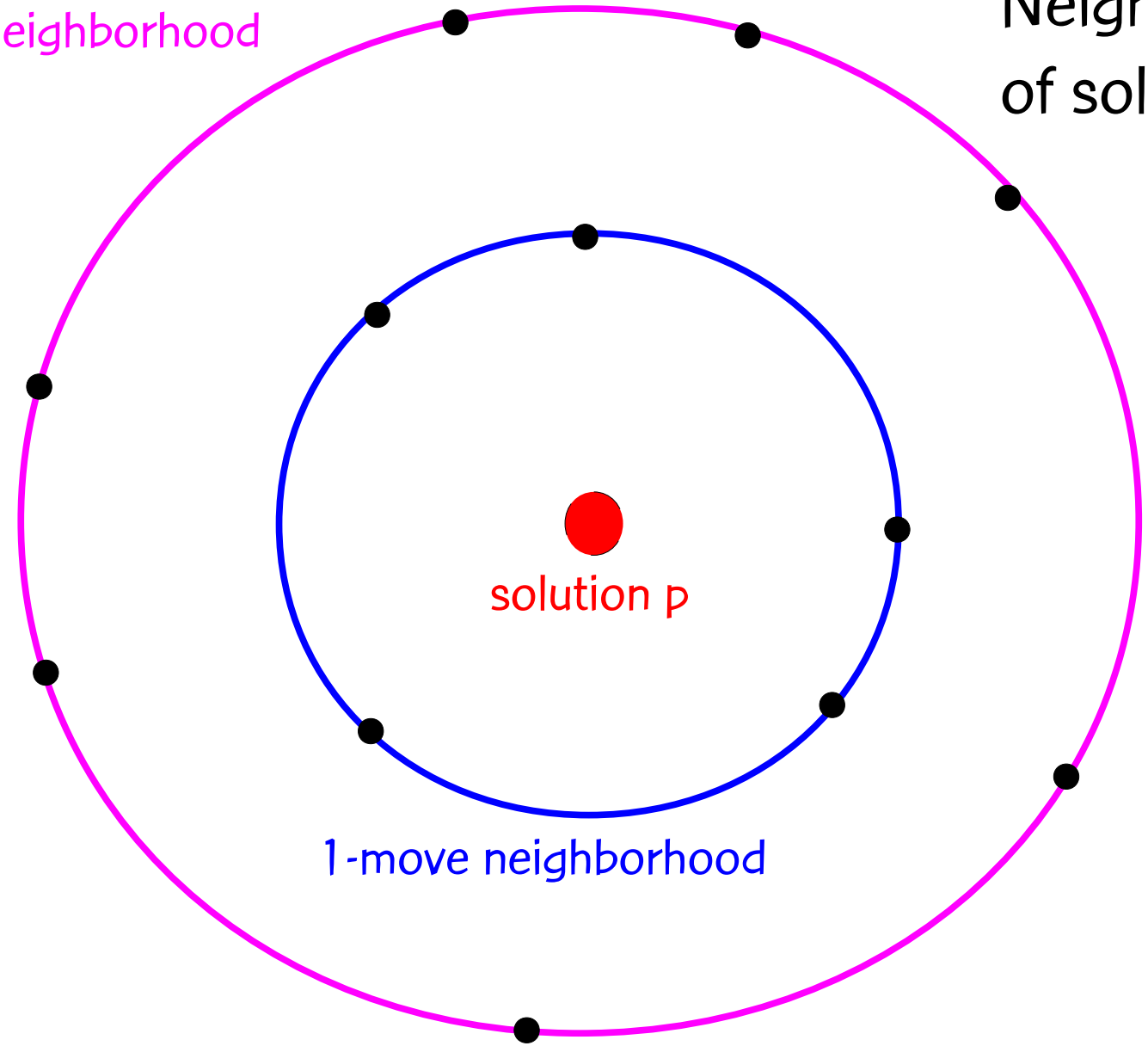


2-move neighbor of p

Assignment representation



Neighborhood of solution p



Traditional local search approaches

Best improving approach:

Evaluate all 1-move and 2-move neighborhood solutions and select the best improving solution

First improving approach:

1: From solution p, to evaluate its 1-move neighbors until the first improving solution q is found.

2: If q does not exist, continue search in the 2-move neighborhood.

3: If q does not exist in the 2-move neighborhood, stop. Otherwise, assign p = q and go to step 1.

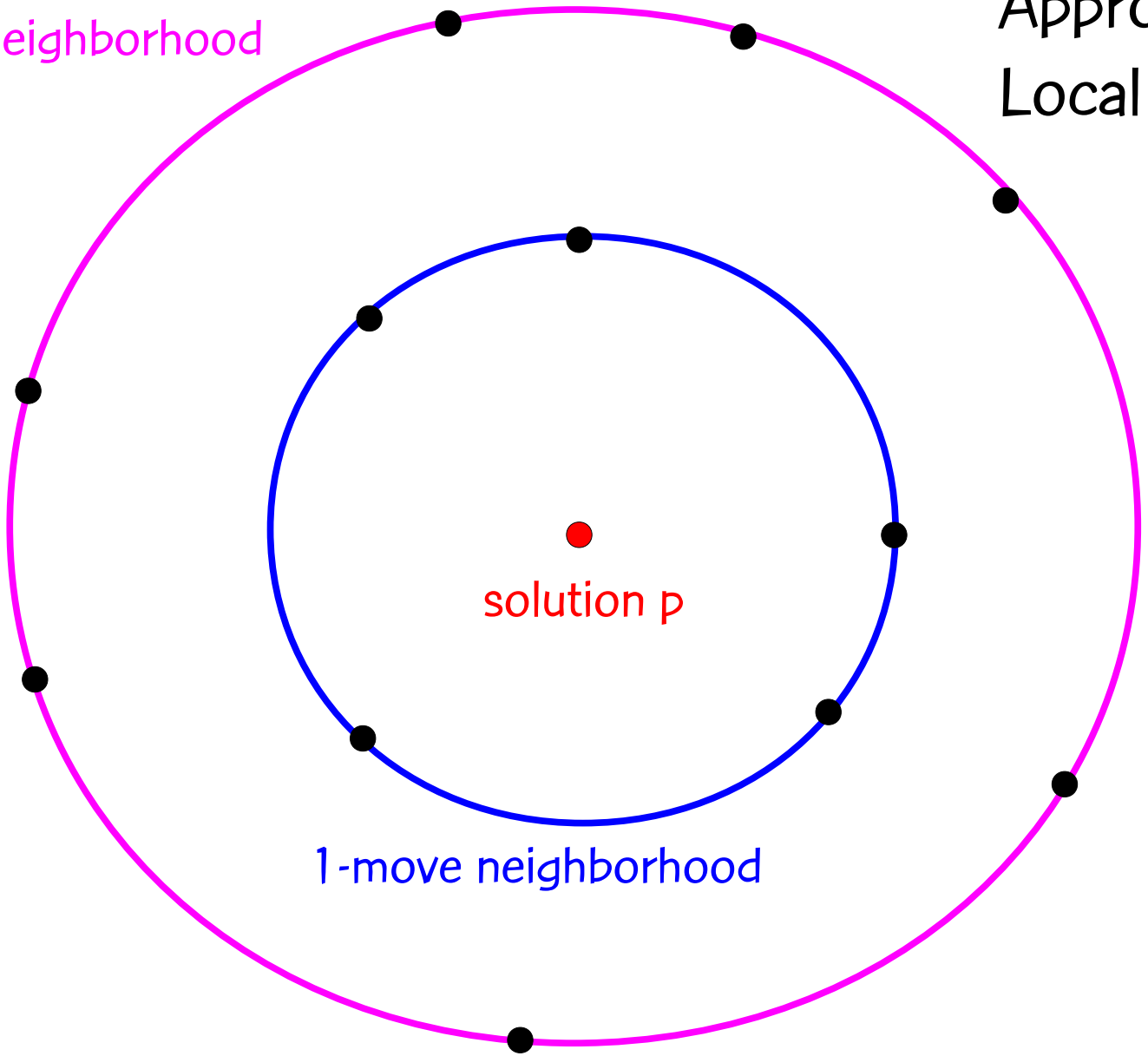
Approximate local search

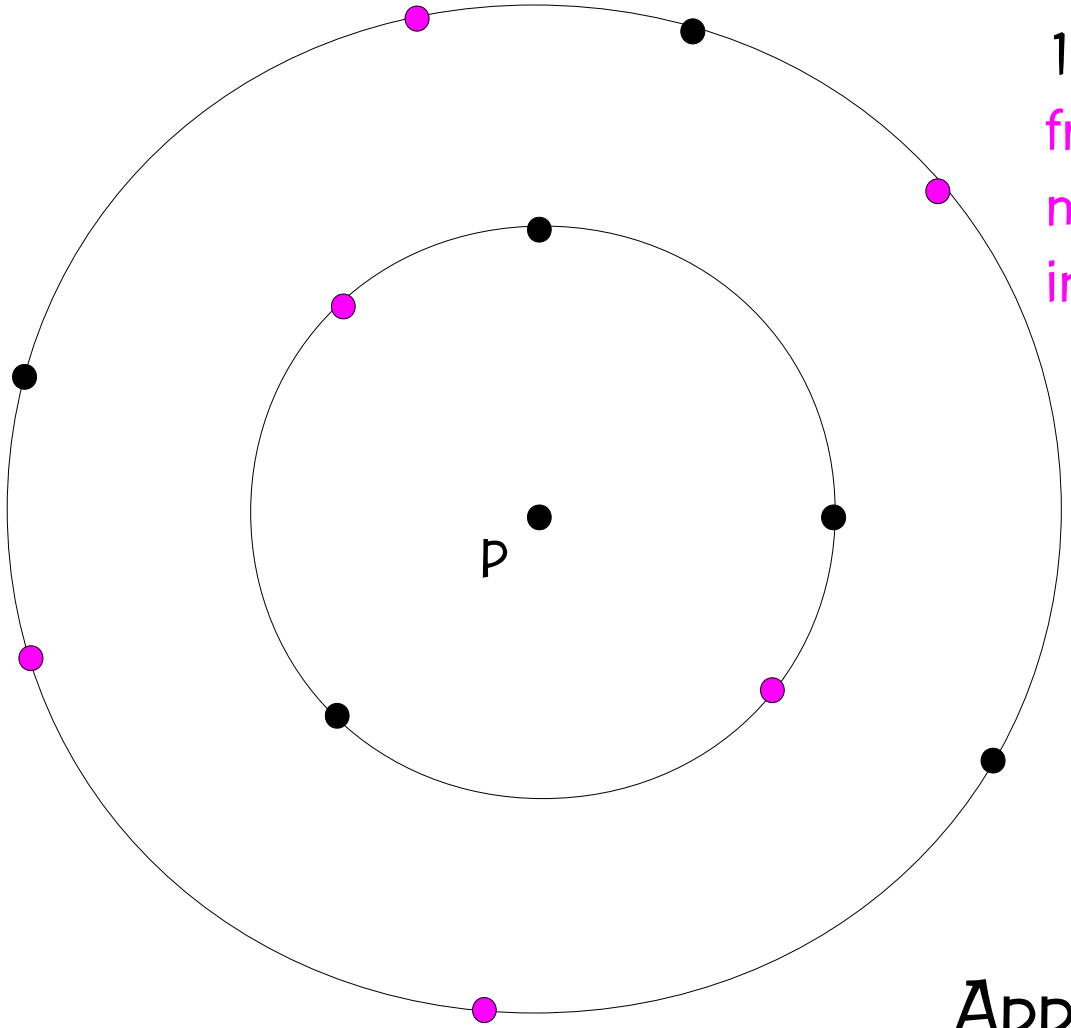
Neighborhoods can be very large for best improvement

Local search can take very long

Tradeoff between best & first improvement: sample the neighborhood of solution p.

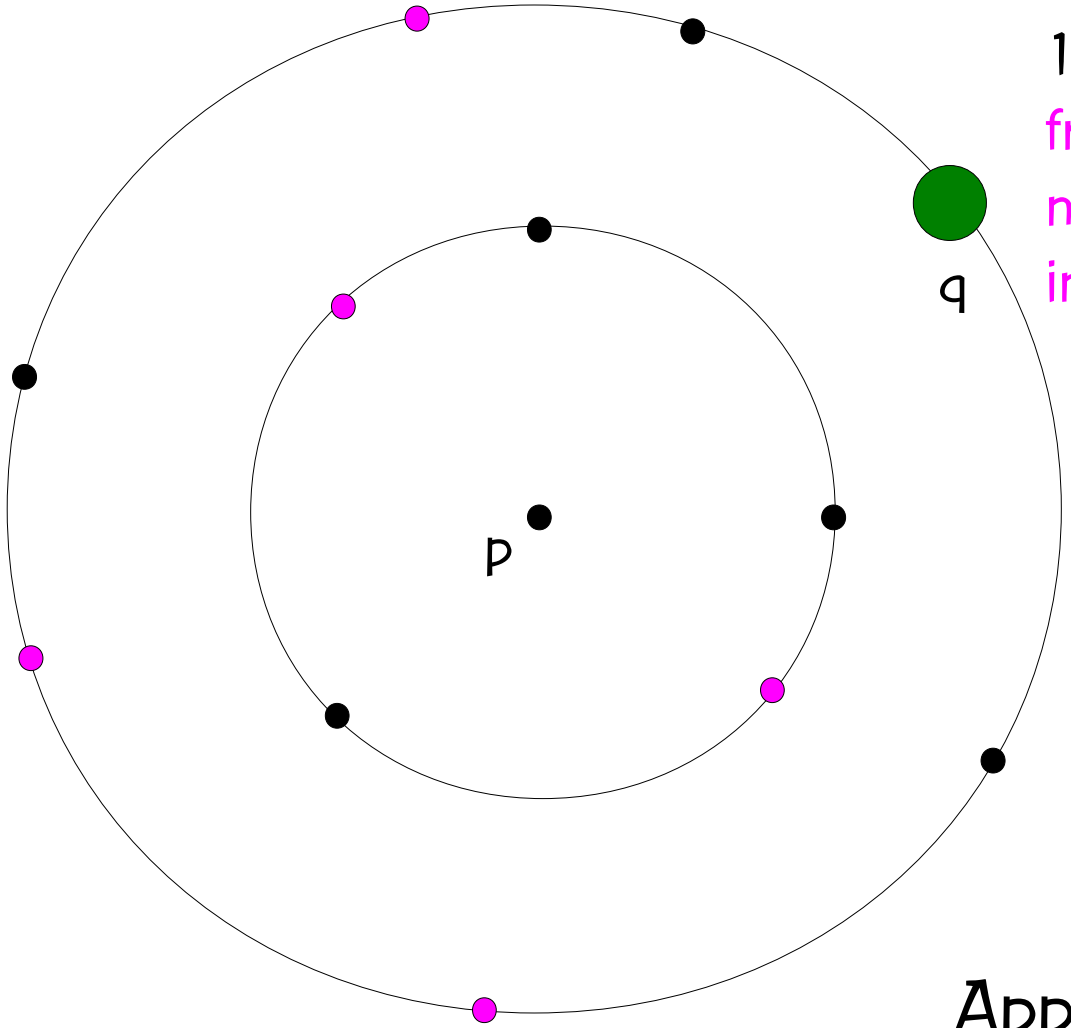
Approximate Local Search





1. Sample k improving solutions from 1-move and 2-move neighborhood of p and place them in an elite set E.

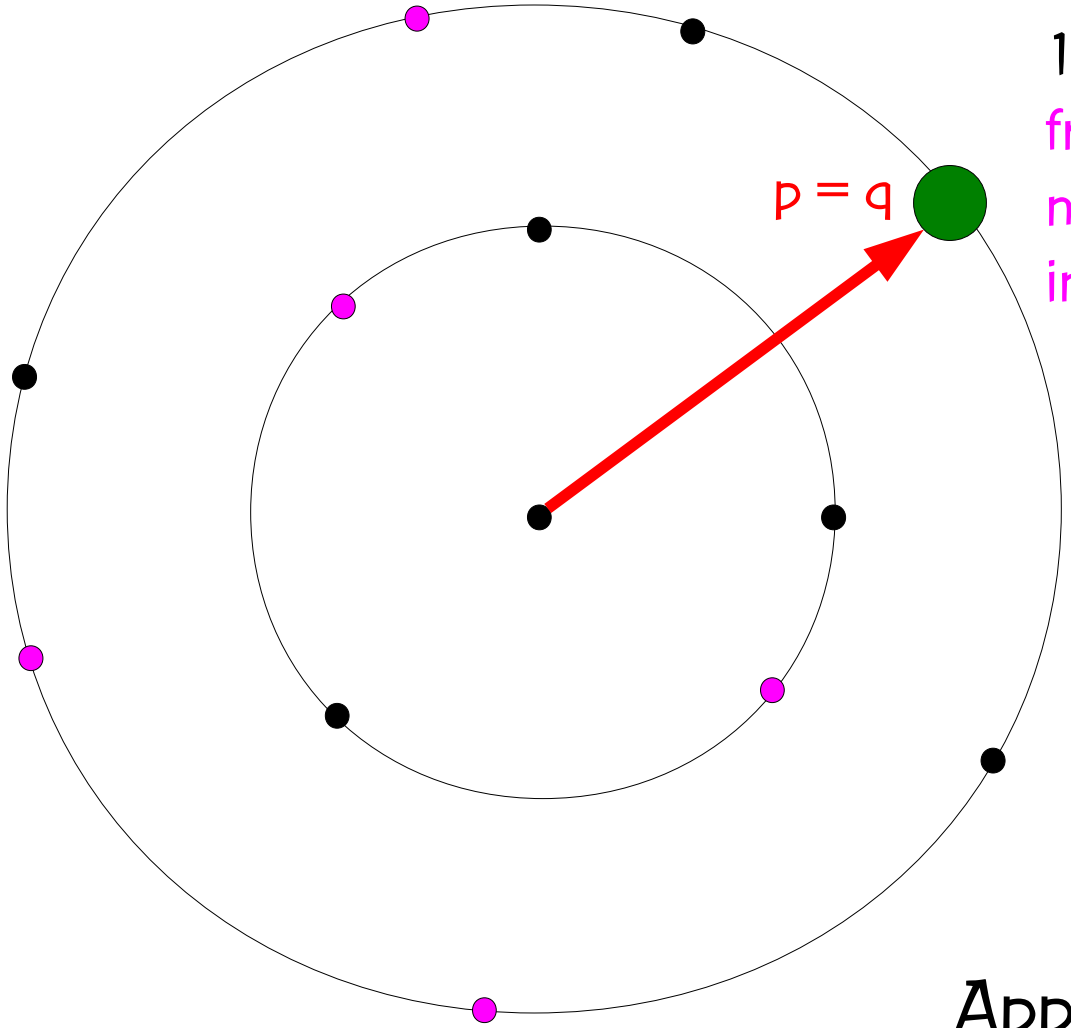
Approximate Local Search



1. Sample k improving solutions from 1-move and 2-move neighborhood of p and place them in an elite set E.

2. Select the best solution **q** from elite set E.

Approximate Local Search



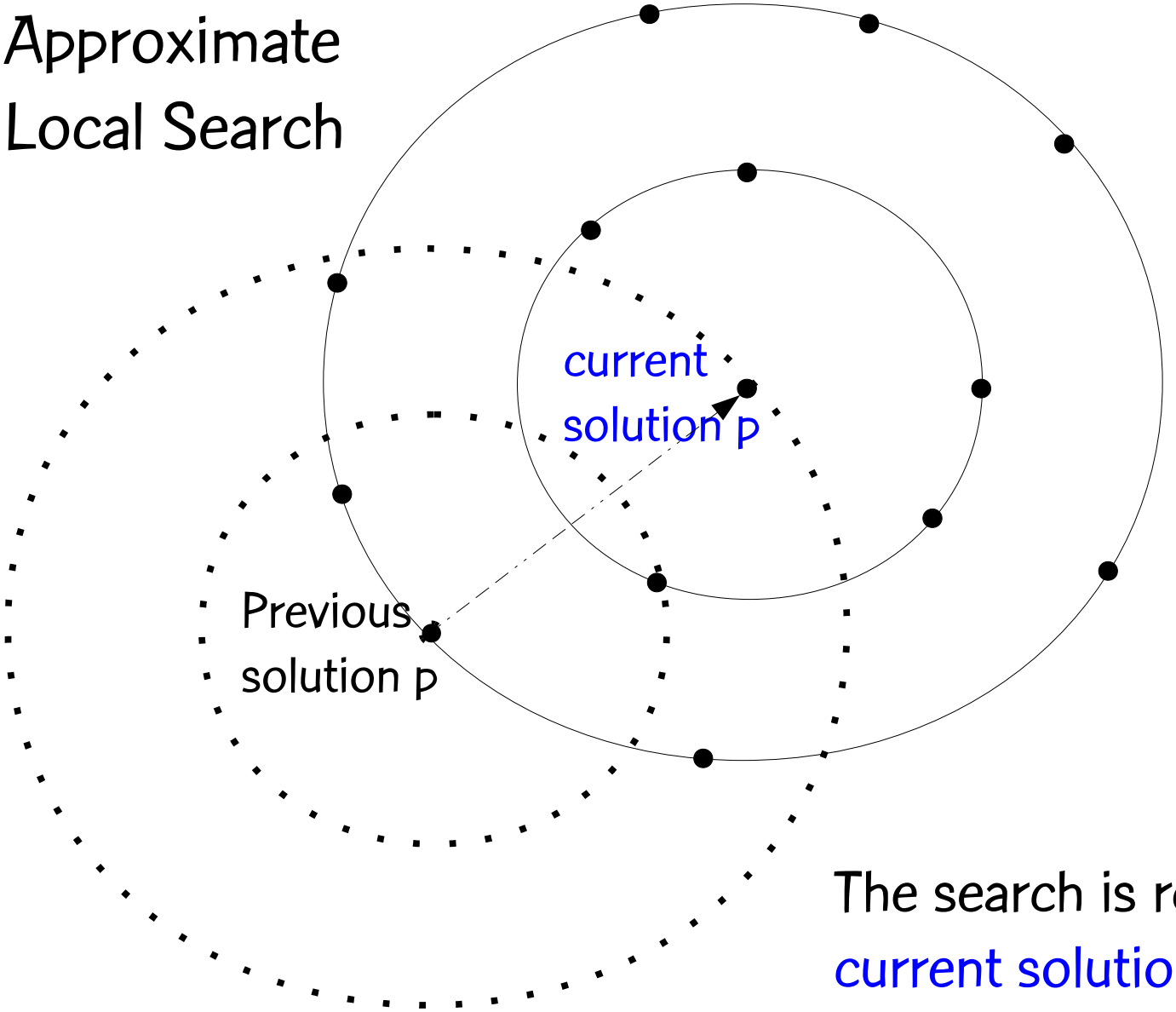
1. Sample k improving solutions from 1-move and 2-move neighborhood of p and place them in an elite set E.

2. Select the best solution **q** from elite set E.

3. Update p = q

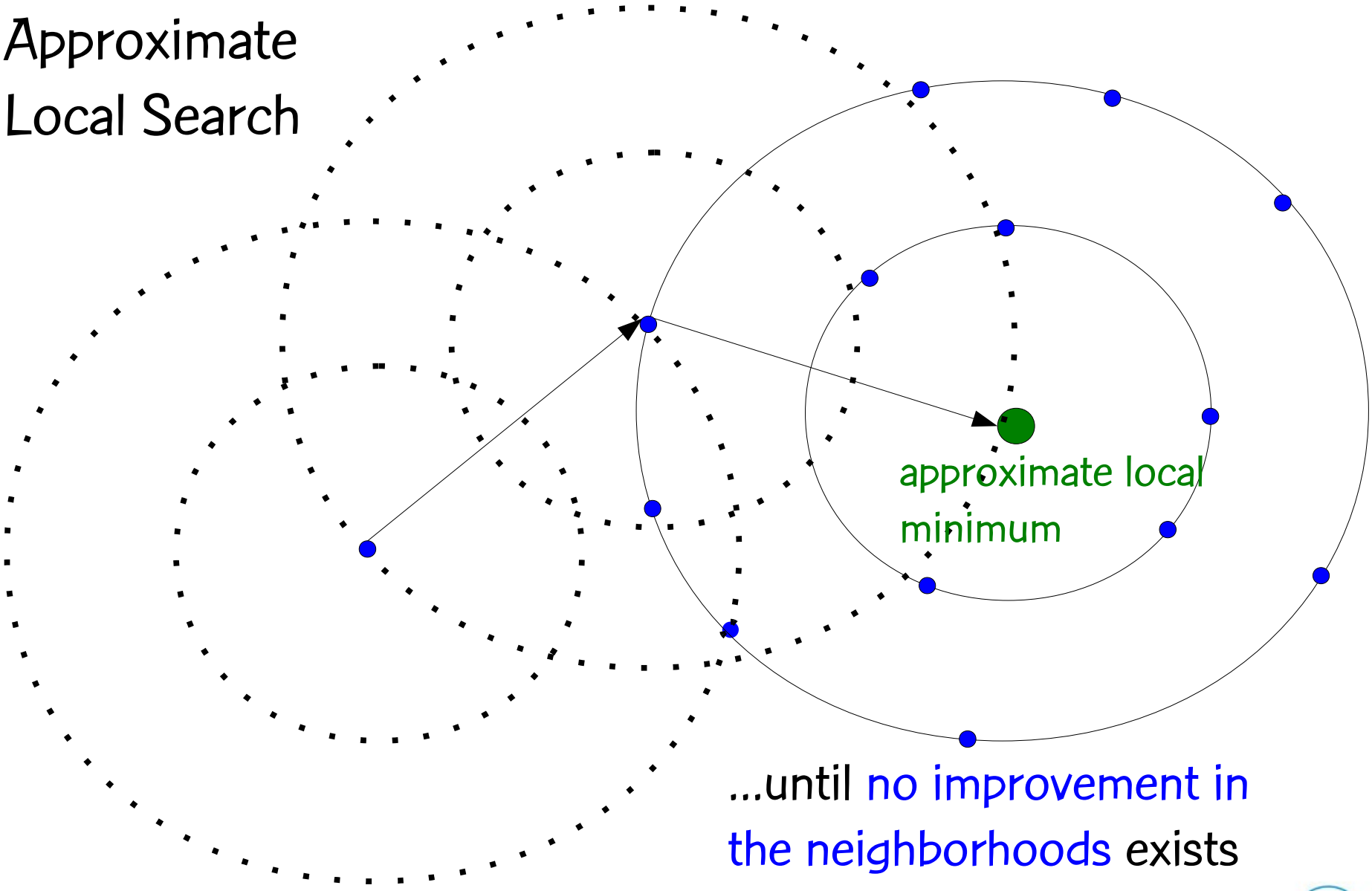
Approximate Local Search

Approximate Local Search



The search is repeated from current solution p until ....

Approximate Local Search

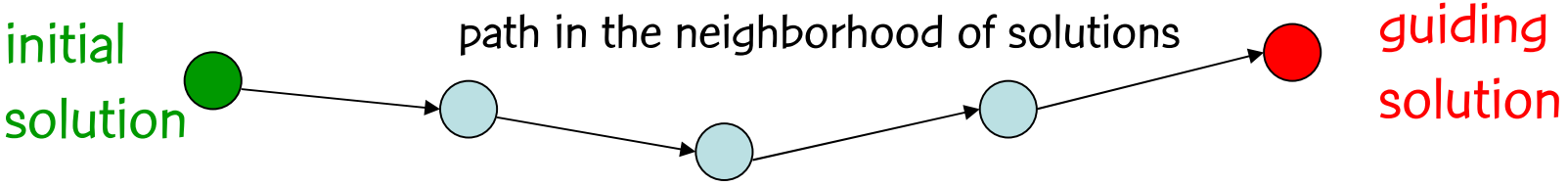


...until no improvement in the neighborhoods exists

Path-relinking

Path-relinking (Glover, 1996)

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



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:

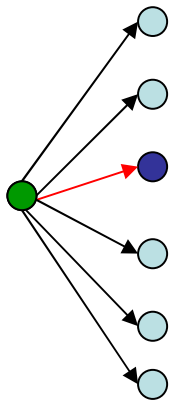
initial solution

guiding solution

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:



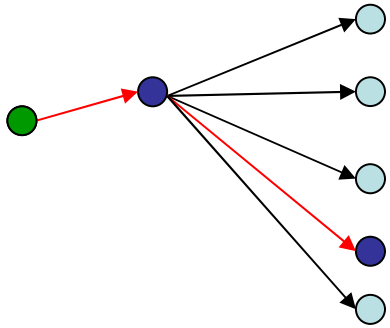
initial solution

guiding solution

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:

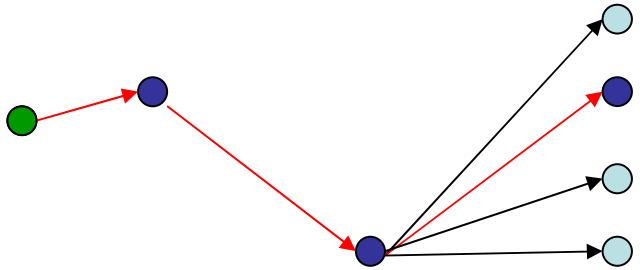


guiding solution

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:

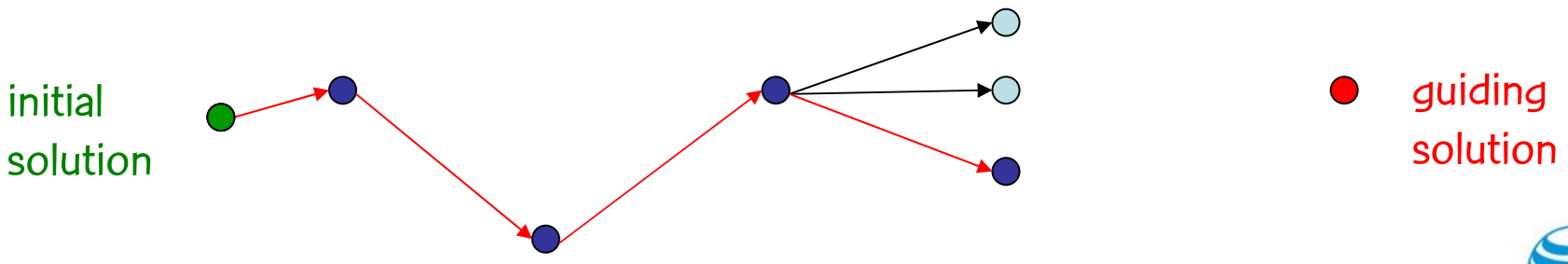


guiding solution

Path-relinking

Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.

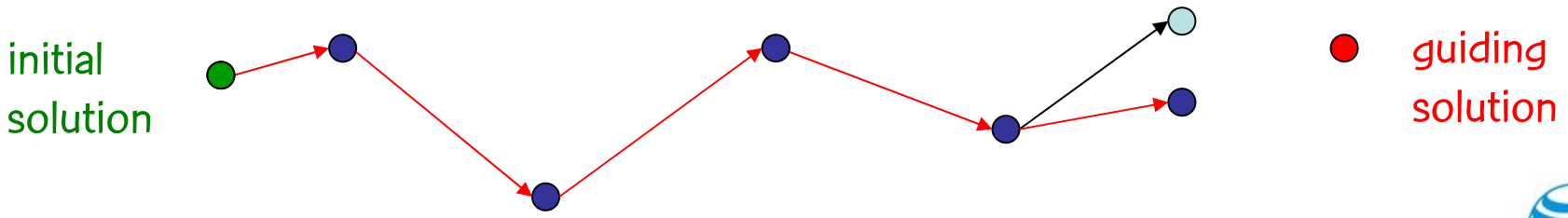
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



Path-relinking

Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.

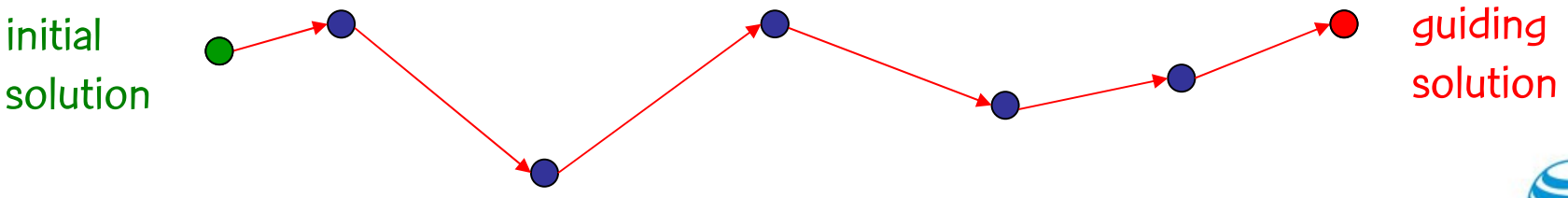
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



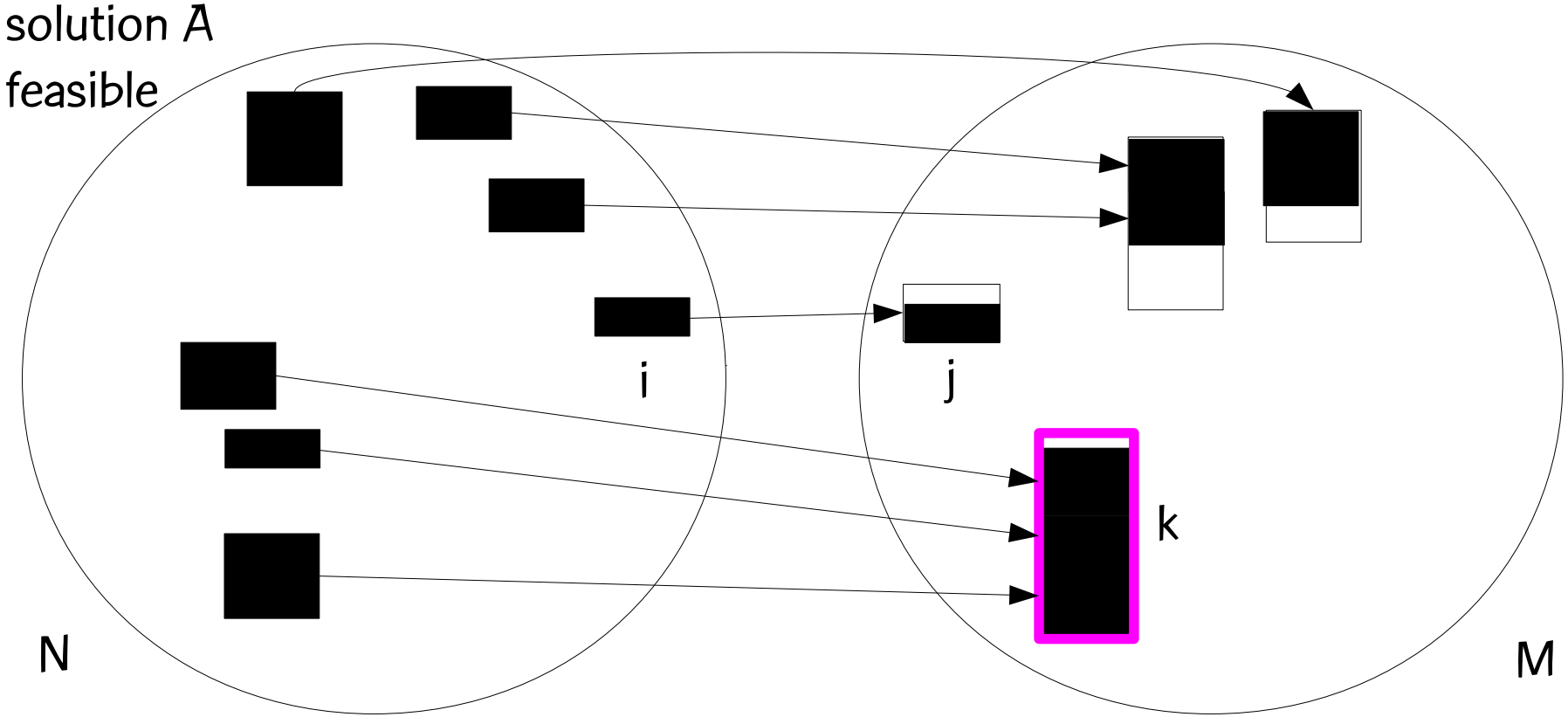
Path-relinking

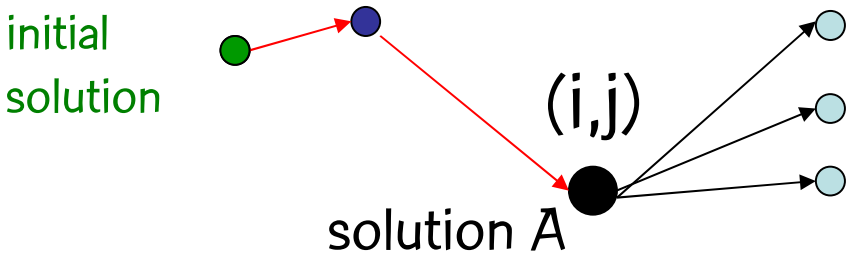
Path is generated by selecting moves that introduce in the initial solution attributes of the guiding solution.

At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



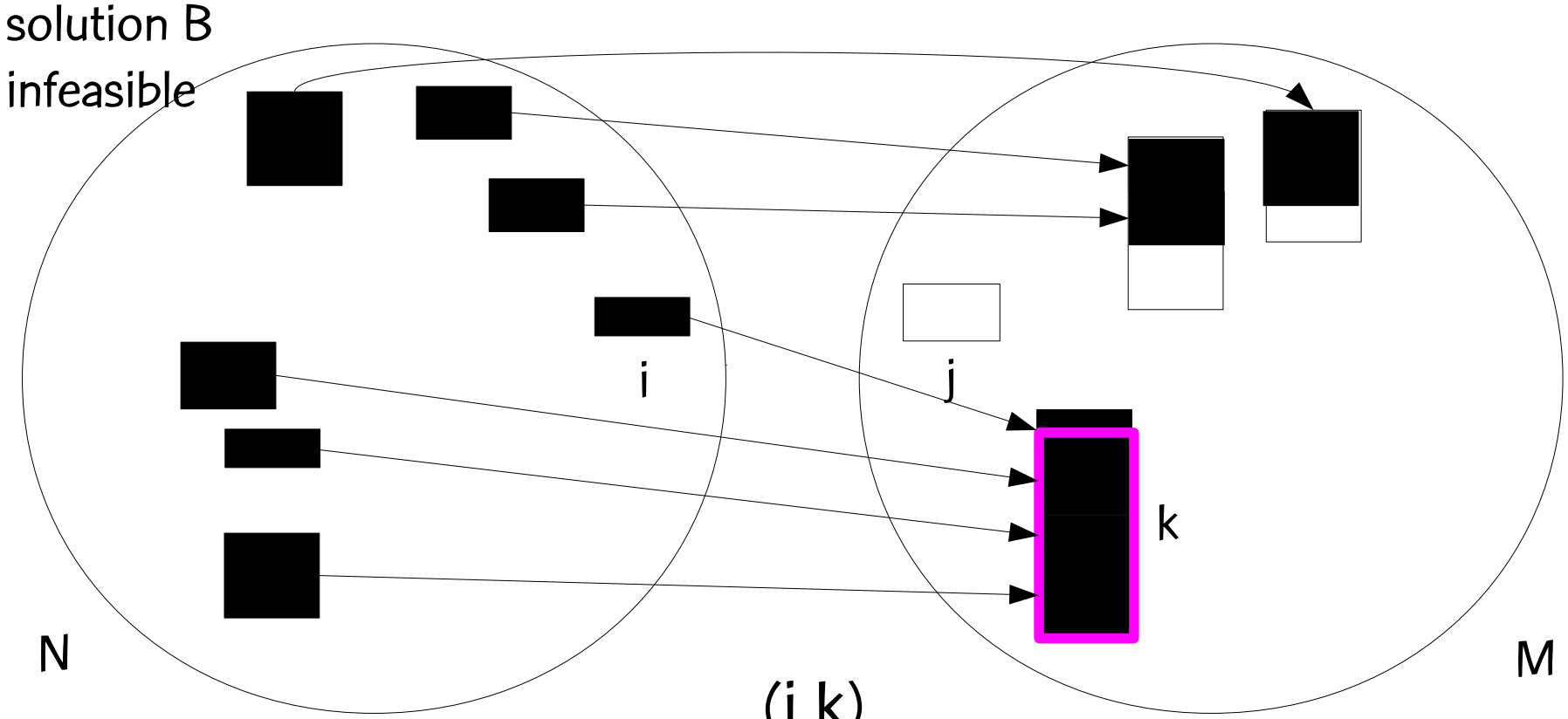
Infeasibility in path-relinking for GQAP

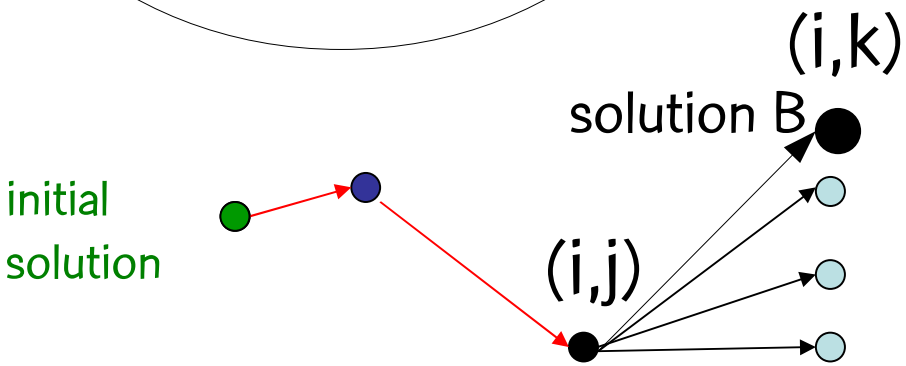




guiding solution (i,k)

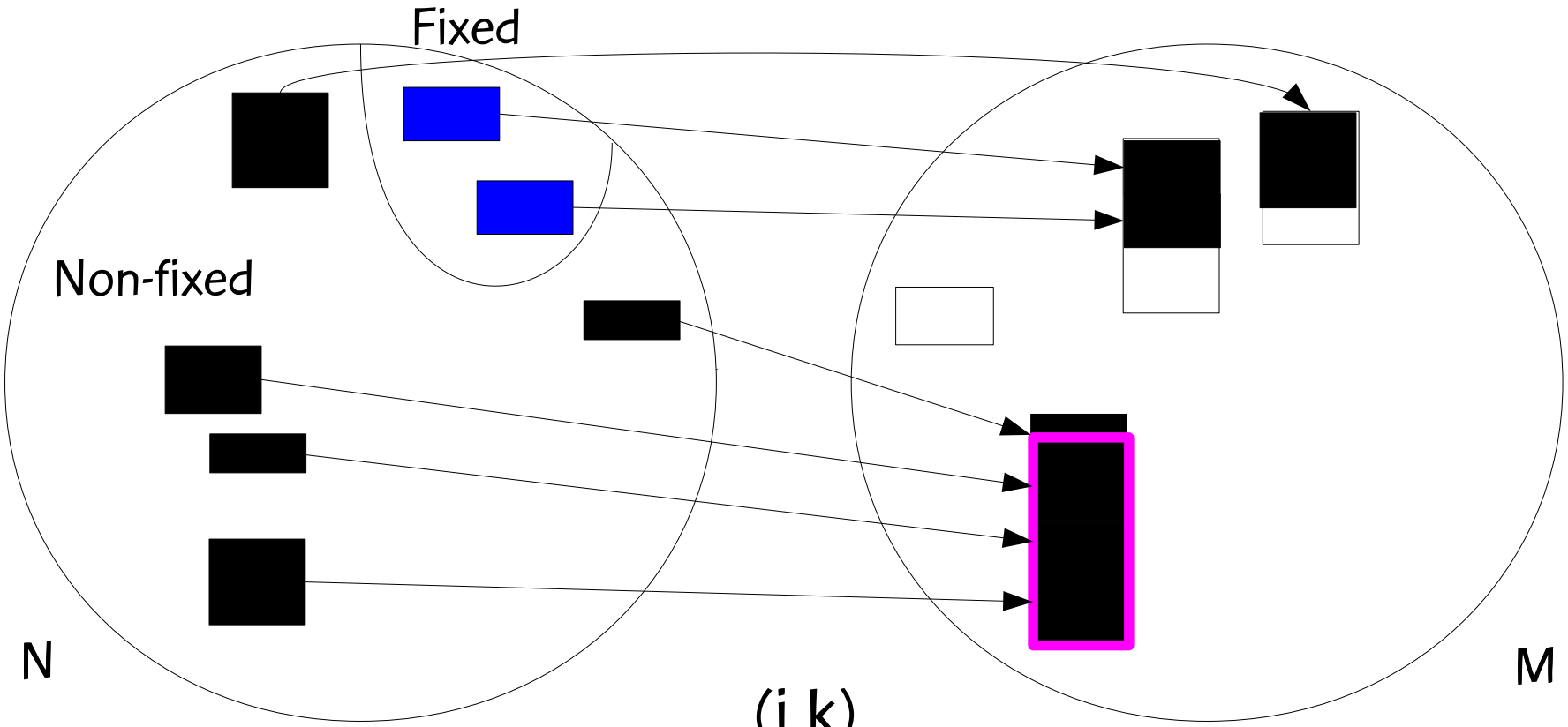
Infeasibility in path relinking for GQAP

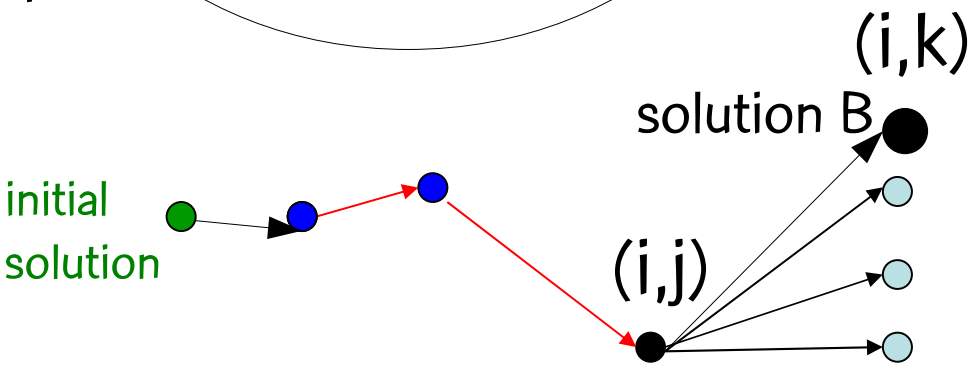




guiding solution (i,k)

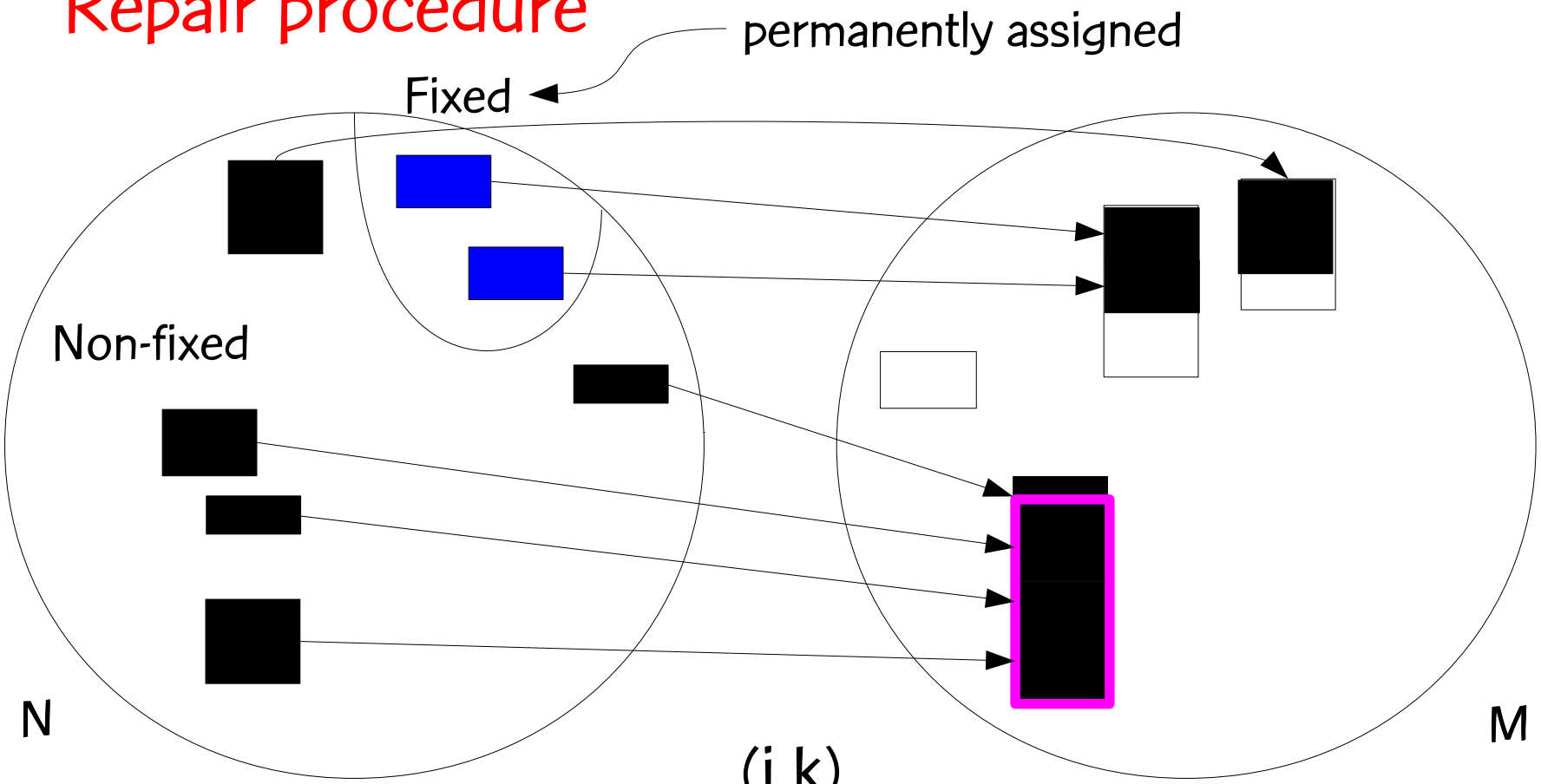
Repair procedure

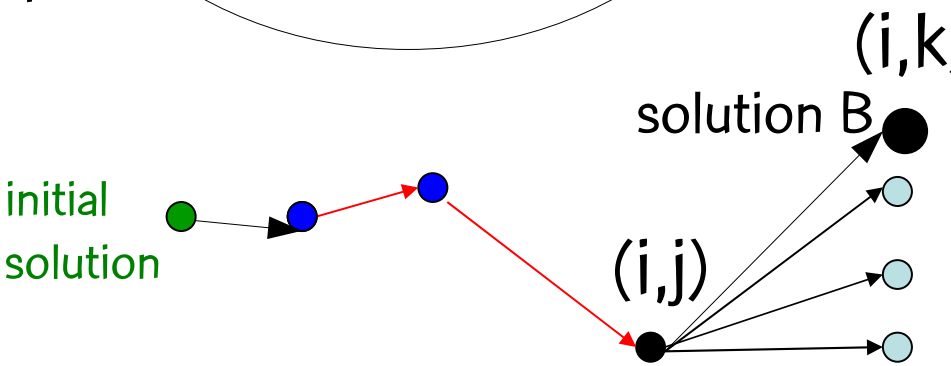




guiding solution

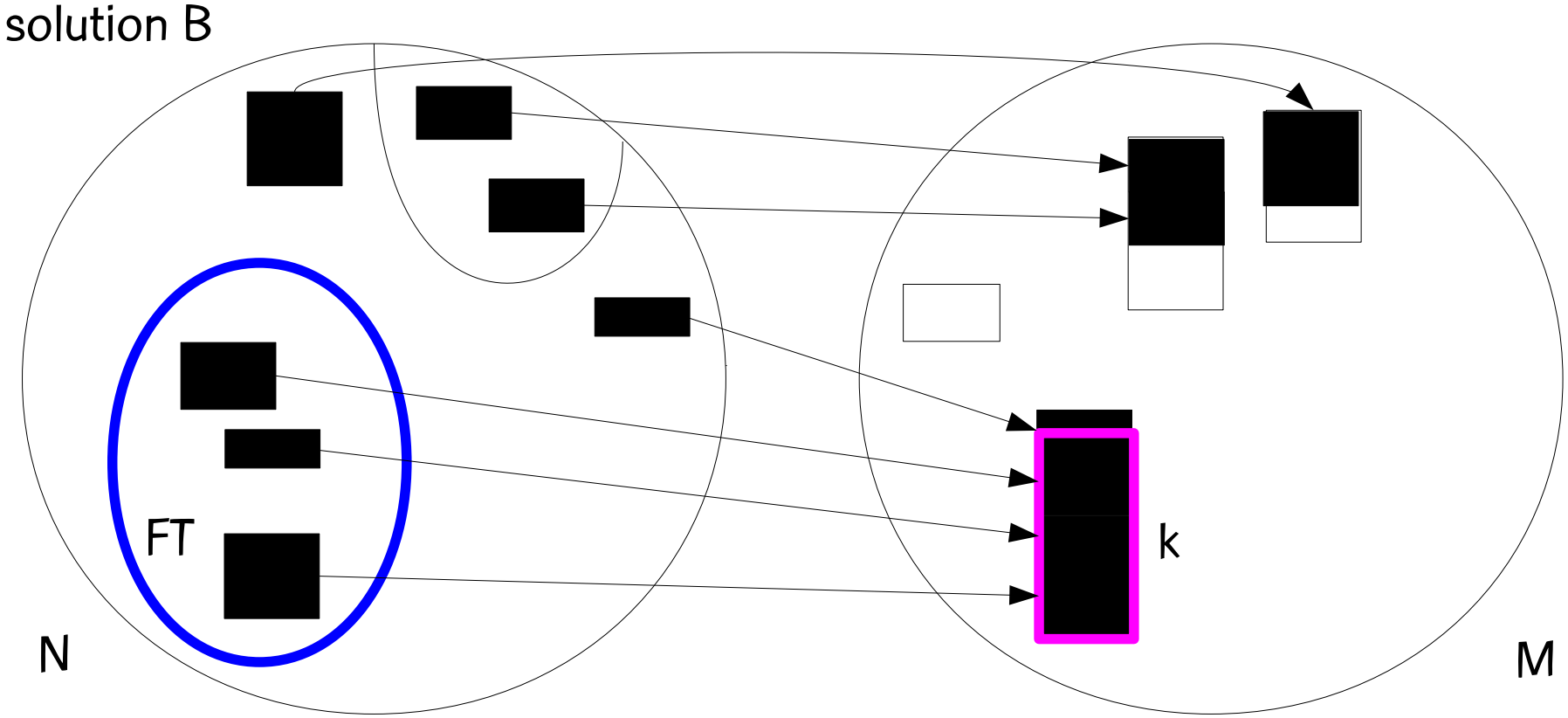
Repair procedure





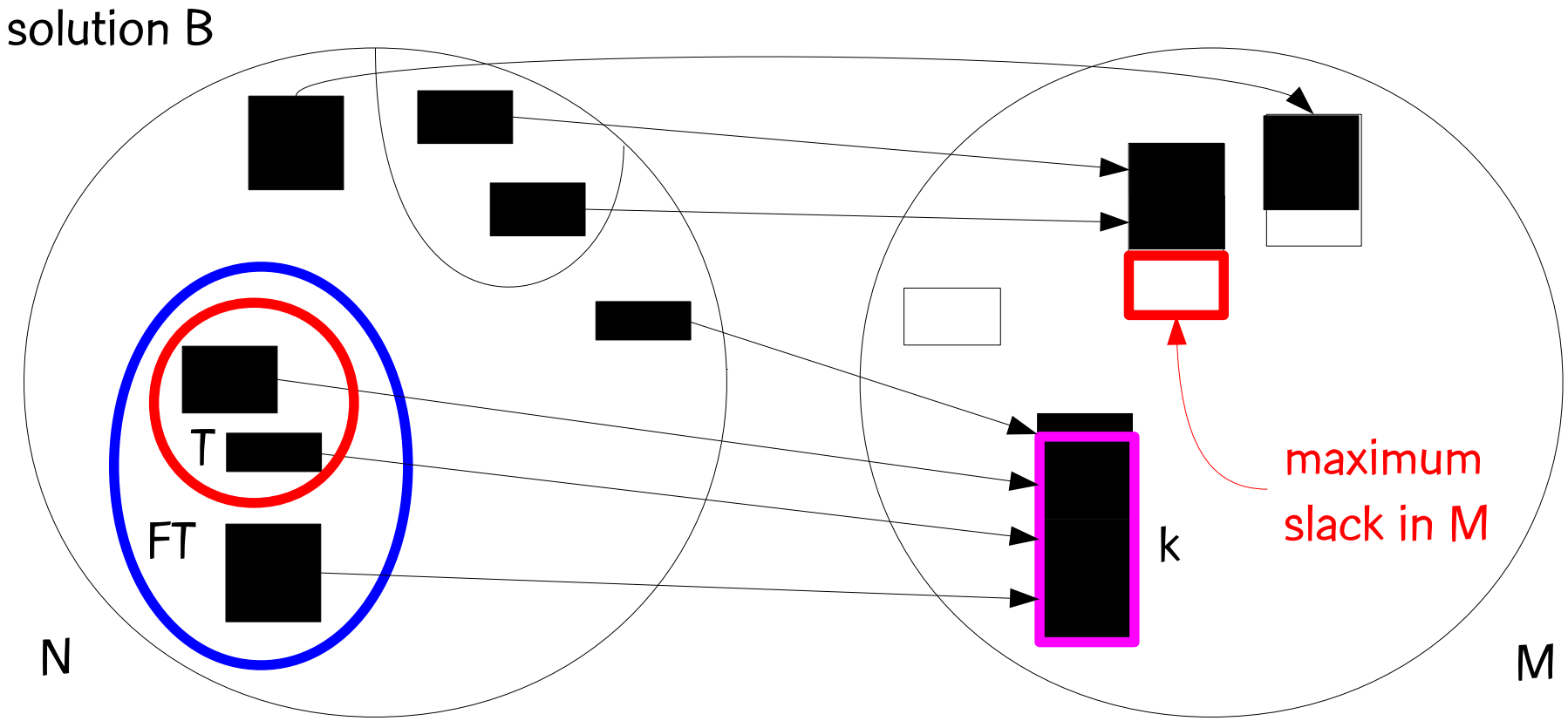
guiding solution

Repair procedure



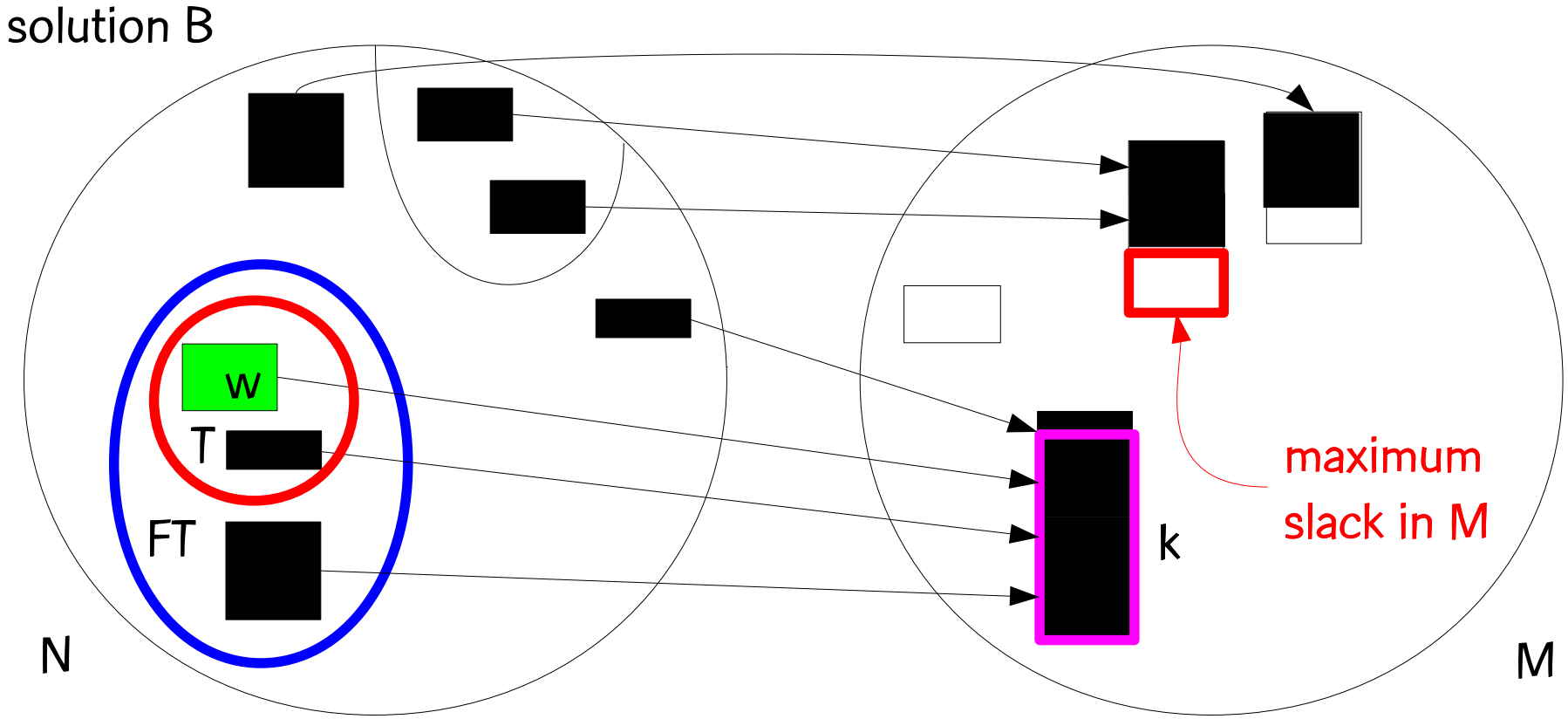
1. Set FT $\subseteq$ non-Fixed: all facilities in solution B assigned to location k

Repair procedure

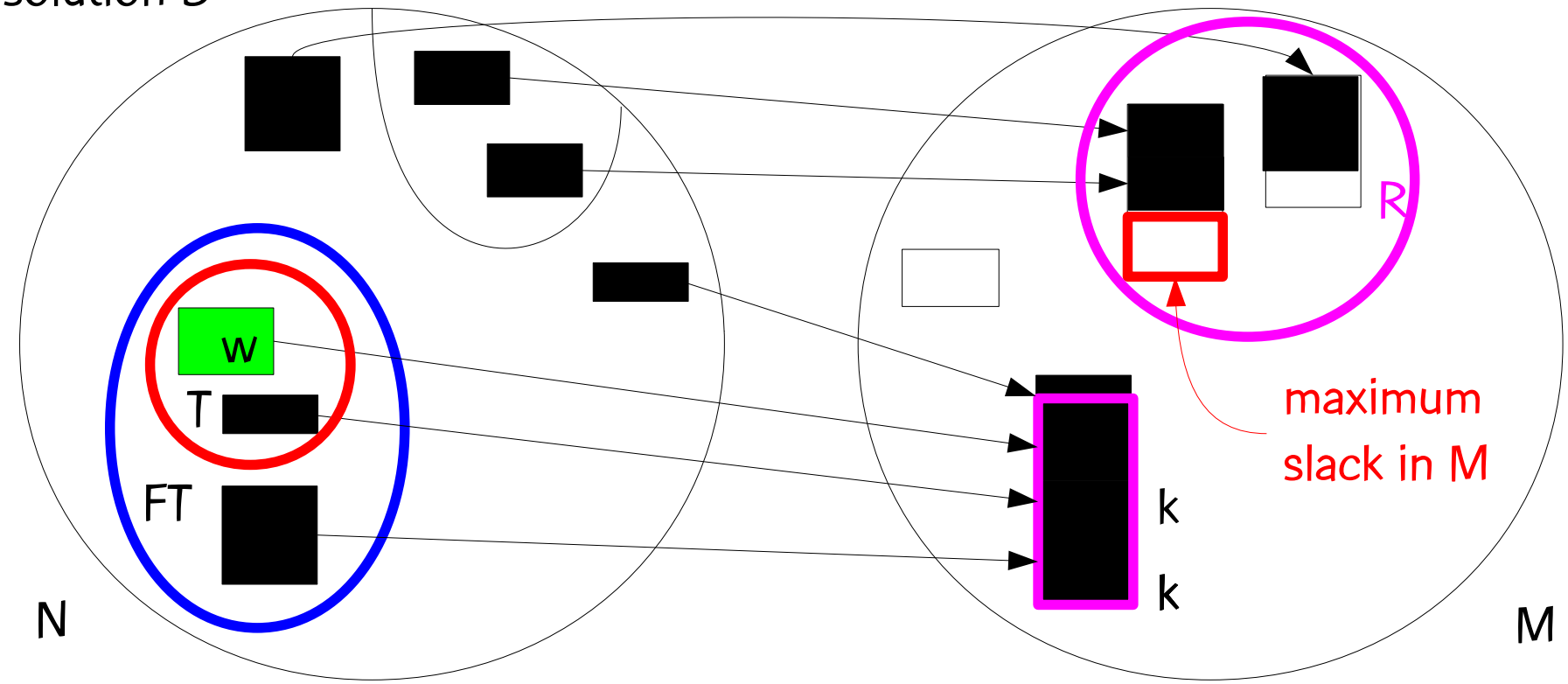


1. Set FT ⊆ non-Fixed: all facilities in solution B assigned to location k
2. Set T ⊆ FT: all facilities in B with demand ≤ maximum slack in M

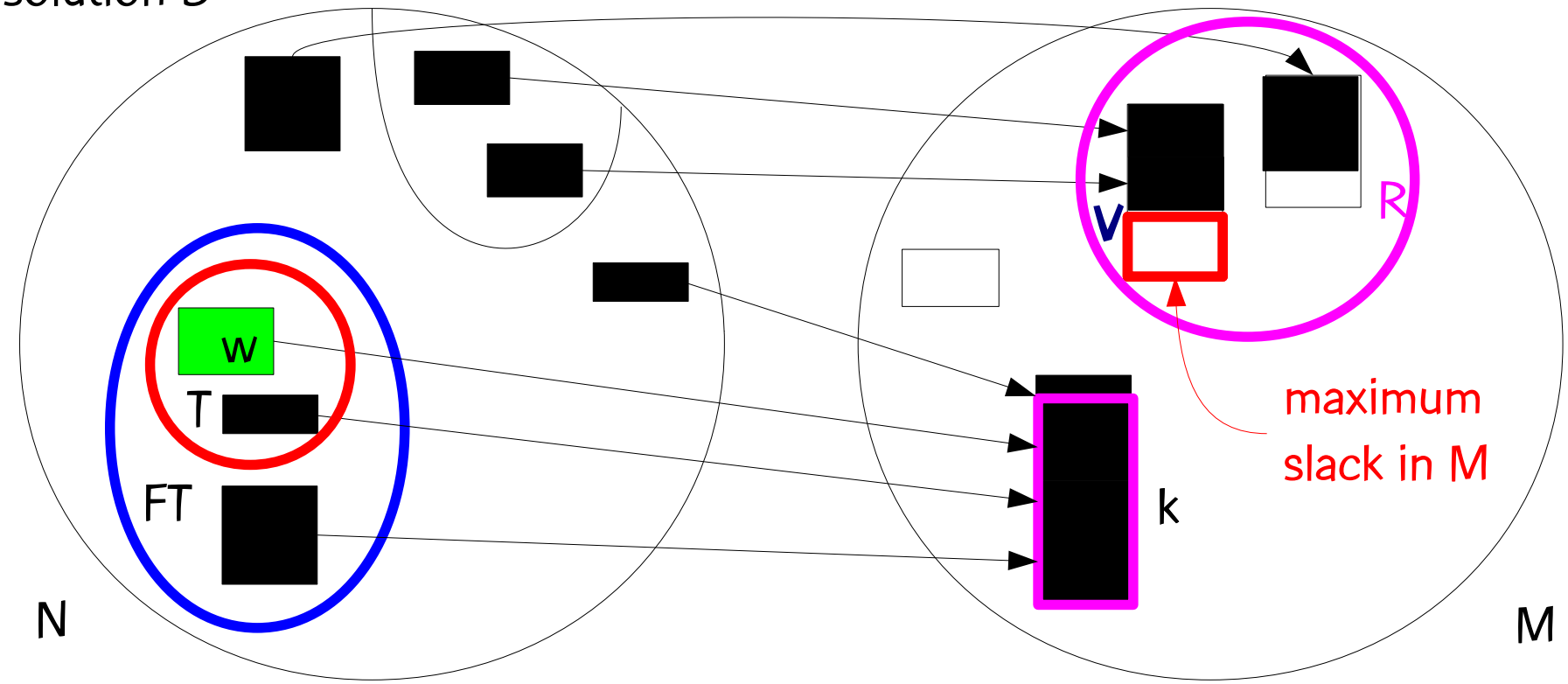
Repair procedure



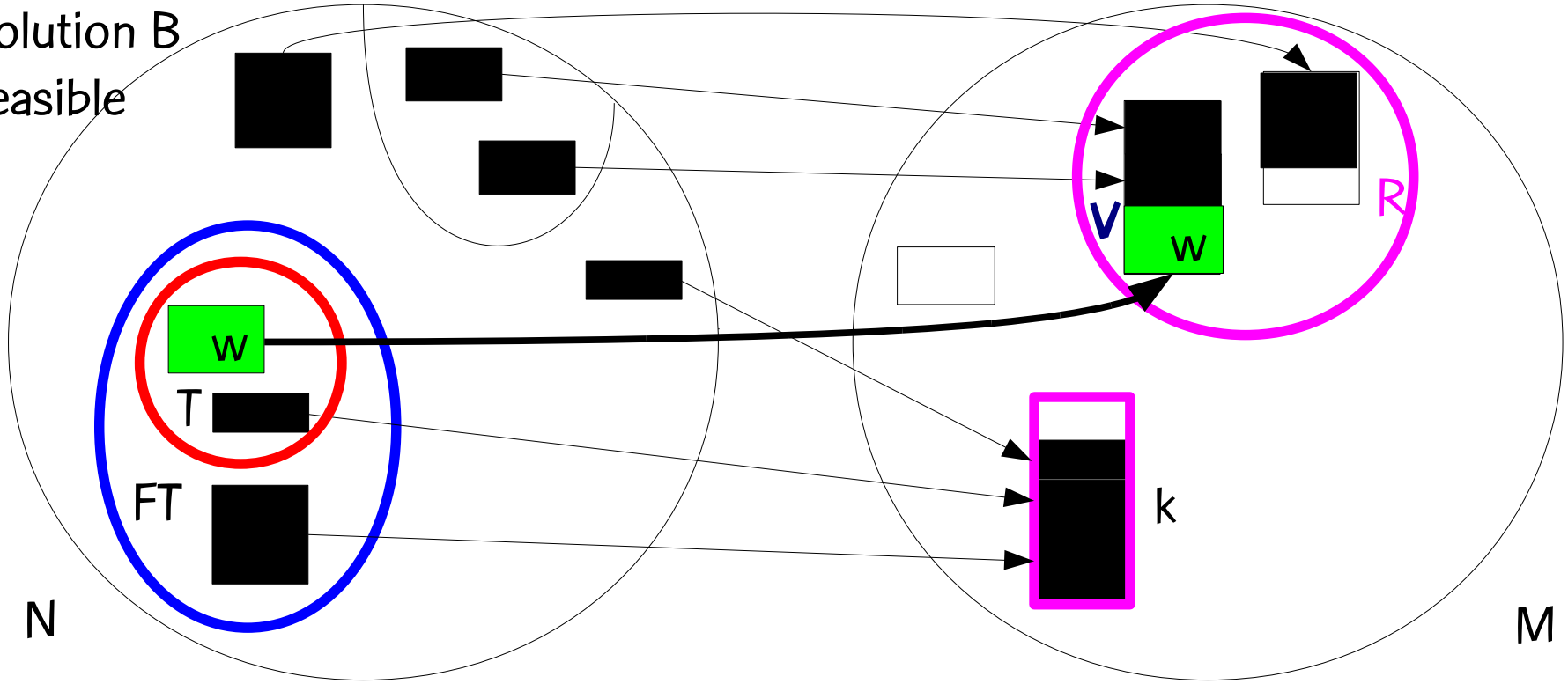
1. Set FT $\subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set T $\subseteq$ FT: all facilities in B with demand $\leq$ maximum slack in M
3. Randomly select a facility w $\in$ T favoring those with higher demand



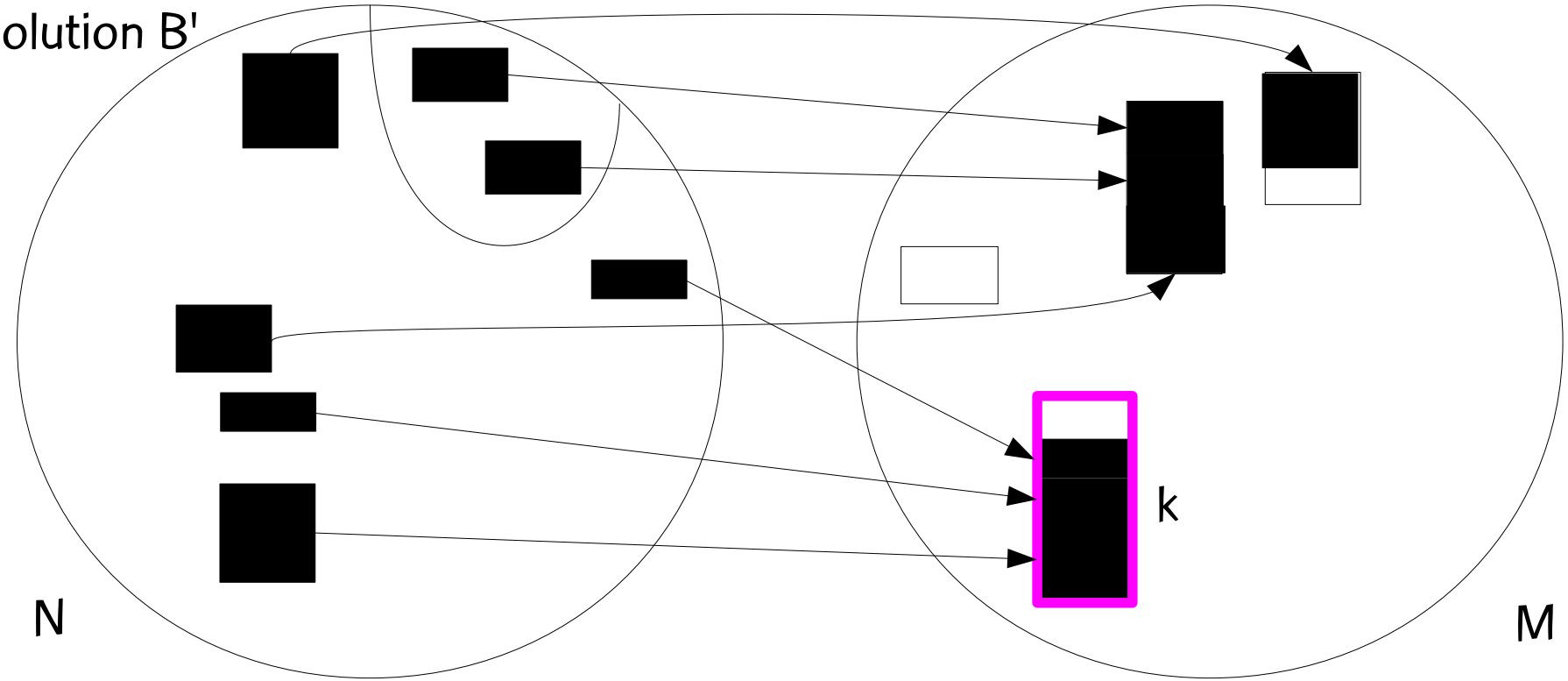
1. Set FT $\subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set T $\subseteq$ FT: all facilities in B with demand $\leq$ maximum slack in M
3. Randomly select a facility w $\in$ T favoring those with higher demand
4. Set R $\subseteq$ M: all locations having slack $\geq$ demand of facility w

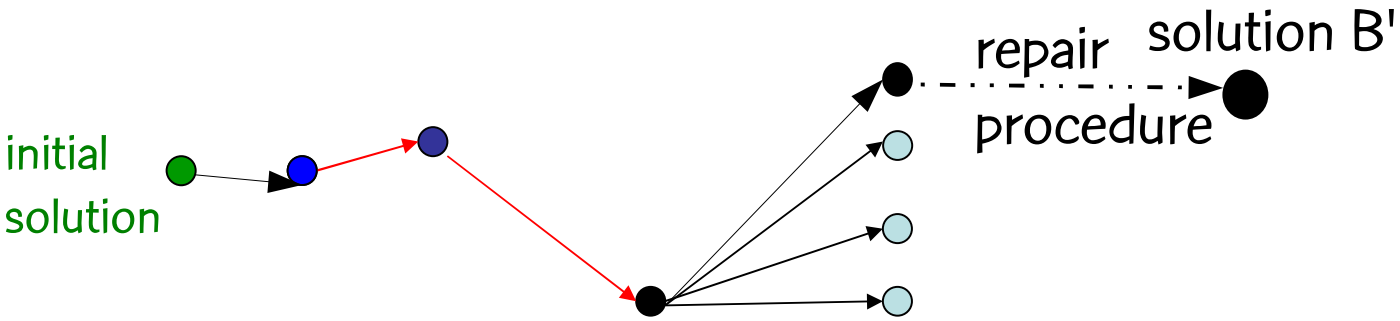


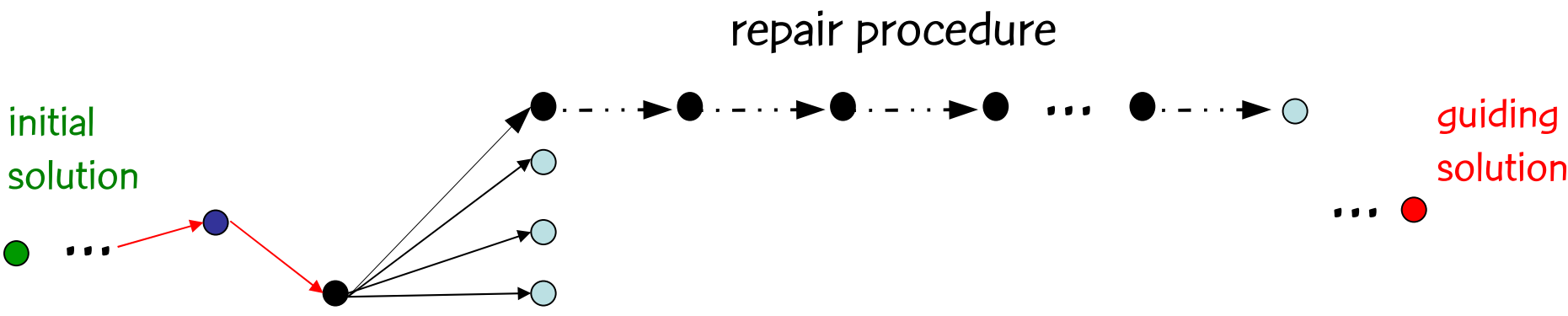
1. Set FT ⊆ non-Fixed: all facilities in solution B assigned to location k
2. Set T ⊆ FT: all facilities in B with demand ≤ maximum slack in M
3. Randomly select a facility w ∈ T favoring those with higher demand
4. Set R ⊆ M: all locations having slack ≥ demand of facility w
5. Randomly select a location v ∈ R (equal probability)



1. Set FT $\subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set T $\subseteq$ FT: all facilities in B with demand $\leq$ maximum slack in M
3. Randomly select a facility w $\in$ T favoring those with higher demand
4. Set R $\subseteq$ M: all locations having slack $\geq$ demand of facility w
5. Randomly select a location v $\in$ R (equal probability)
6. Assign facility w to location v

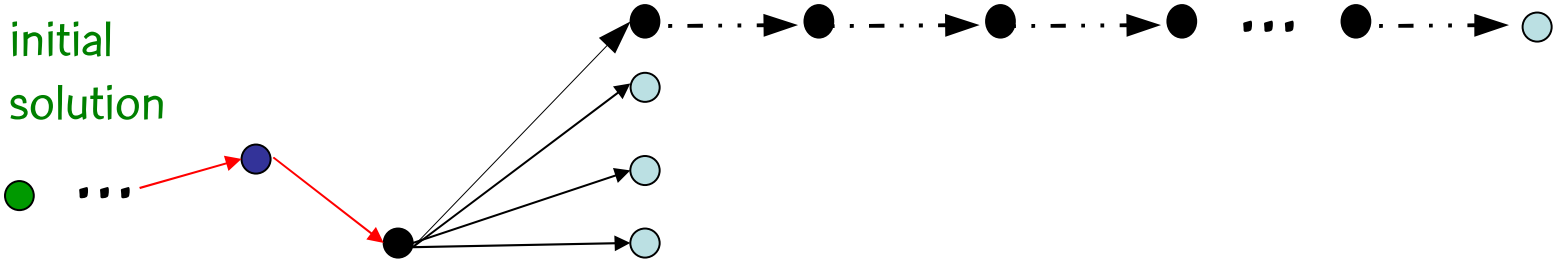




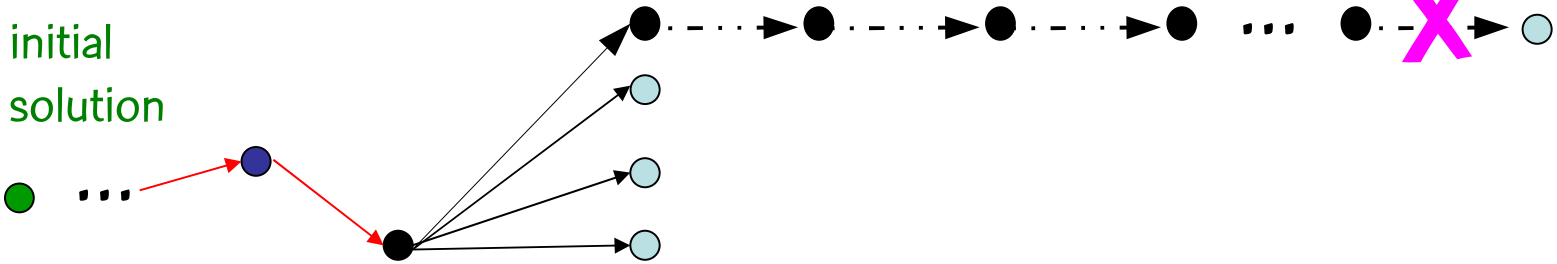


Possible outcomes

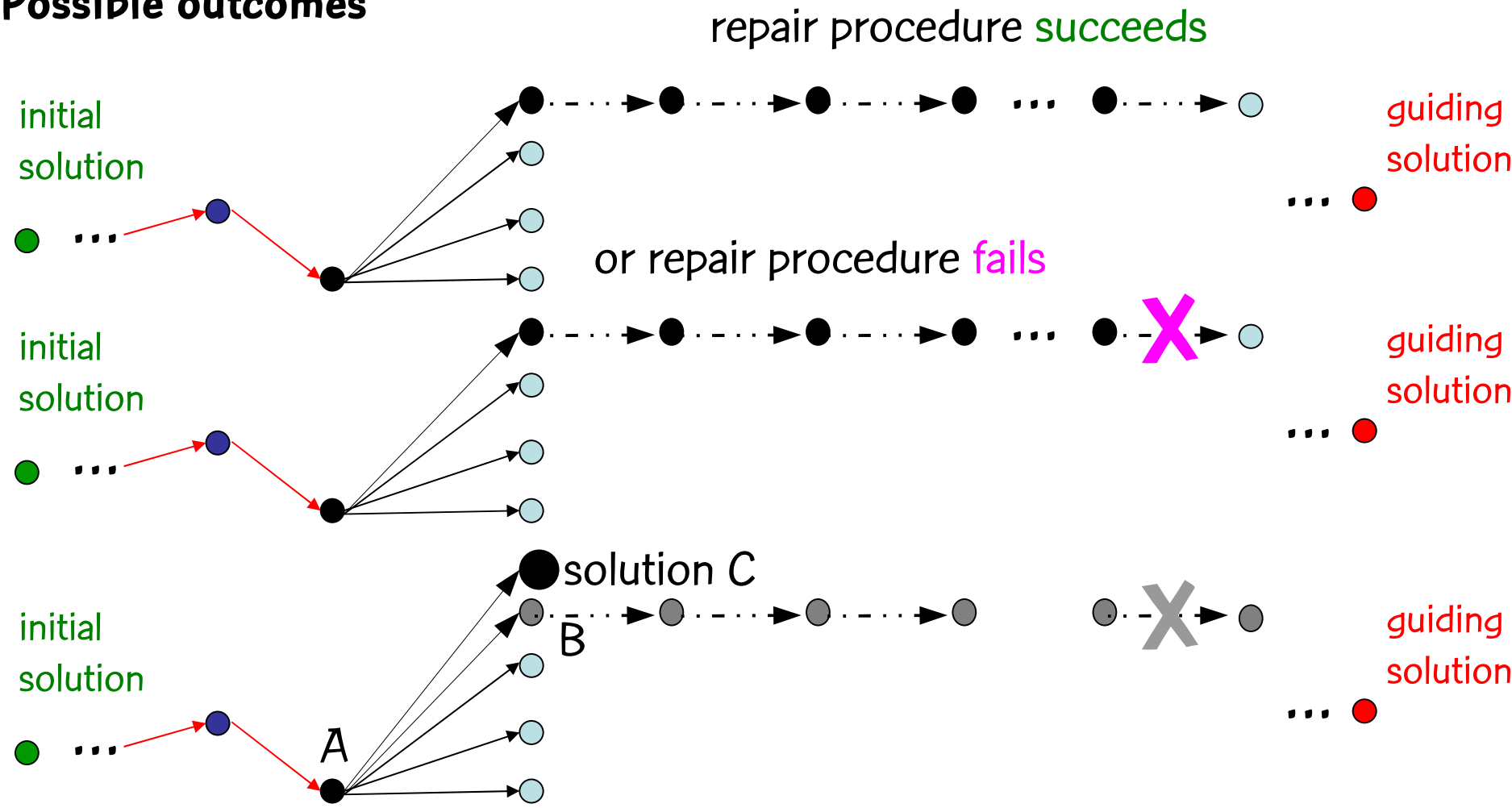
repair procedure succeeds



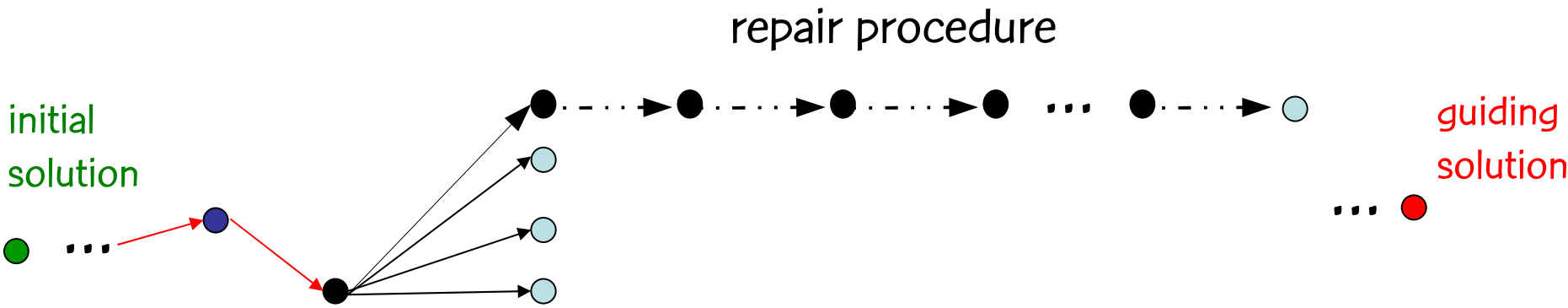
or repair procedure fails



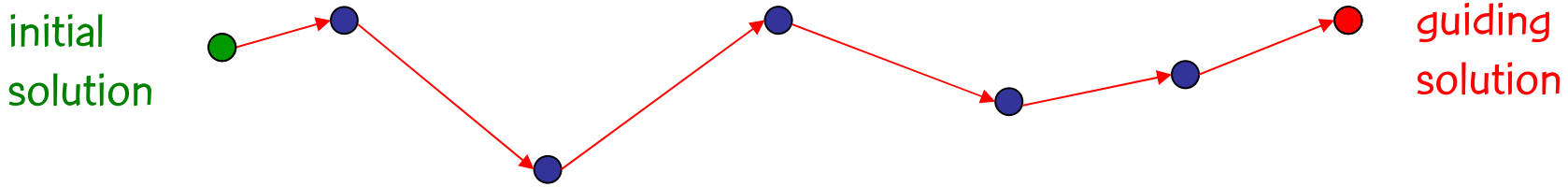
Possible outcomes

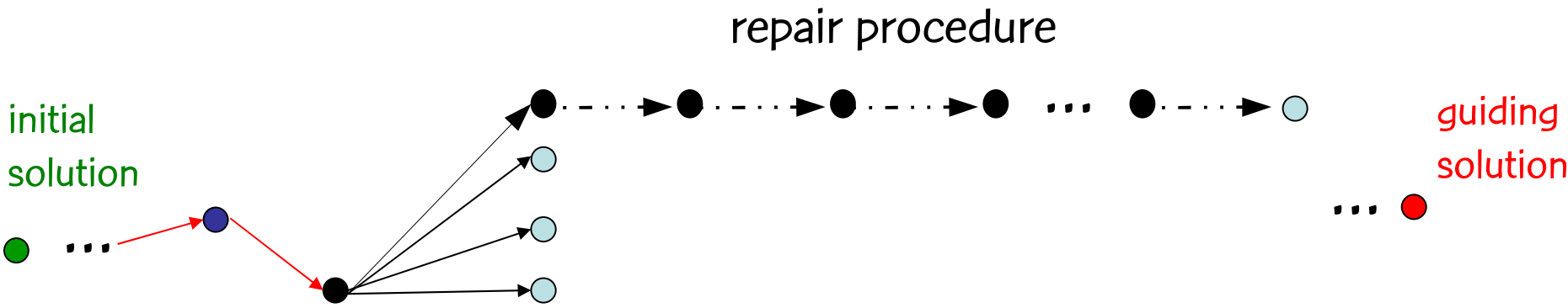


Repeat the repair procedure on solution B a maximum number of times. If a feasible solution is not found, discard B and move to solution C

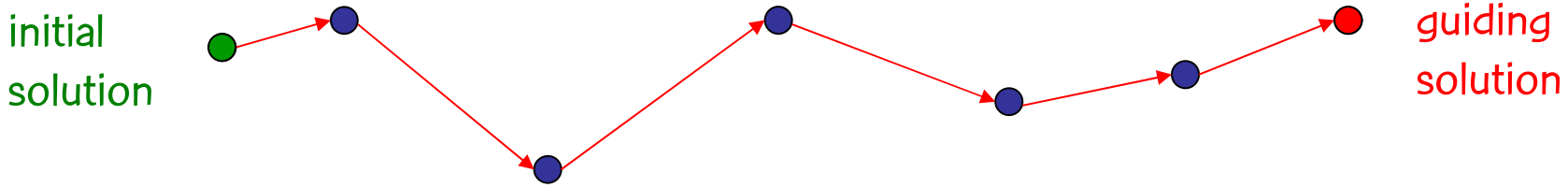


So, instead of a path with feasible solution in one single step ...

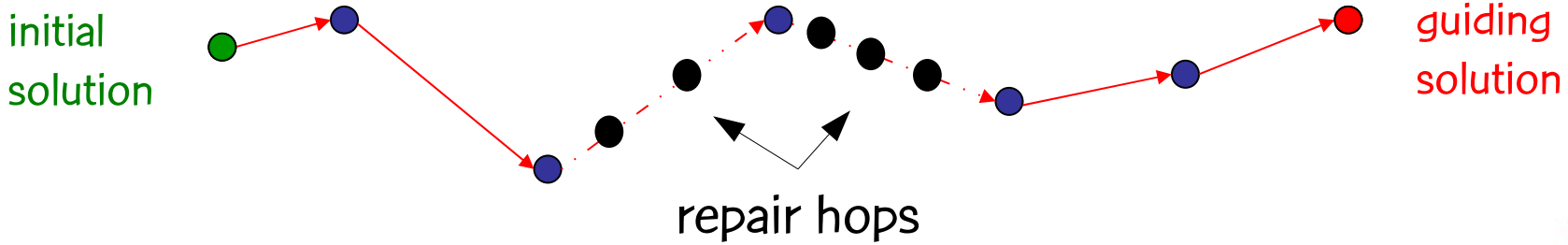




So, instead of a path with feasible solution in one single step ...



We have now a path with eventual intermediate repair hops



Experimental results

Test environment

Dell PE1950 computer with a dual quad core 2.66 GHz Intel Xeon processors an 16 GB of Memory

Red Hat Linux version 5.1.19.6

Java language, Javac compiler ver.1.6.0-05

Random-number generator: Mersenne Twister algorithm (Matsumoto and Nishimura, 1998) from the COLT library

Test environment

Instances:

From Elloumi et al. (2003), Lee and Ma (2005), and Cordeau et al. (2006): 10 to 50 facilities and 3 to 20 locations.

Experimental Design:

For each instance we made 200 independent runs of GRASP-PR. Each run stopped when a solution value as good as the best in the literature was found.

Statistics:

Minimum, maximum, average times, and standard deviation.

Time for 95% of the runs to find solutions as good as the literature.

Parameter tuning for GRASP-PR

Instance: 50-10-95 (Cordeau et al., 2006).

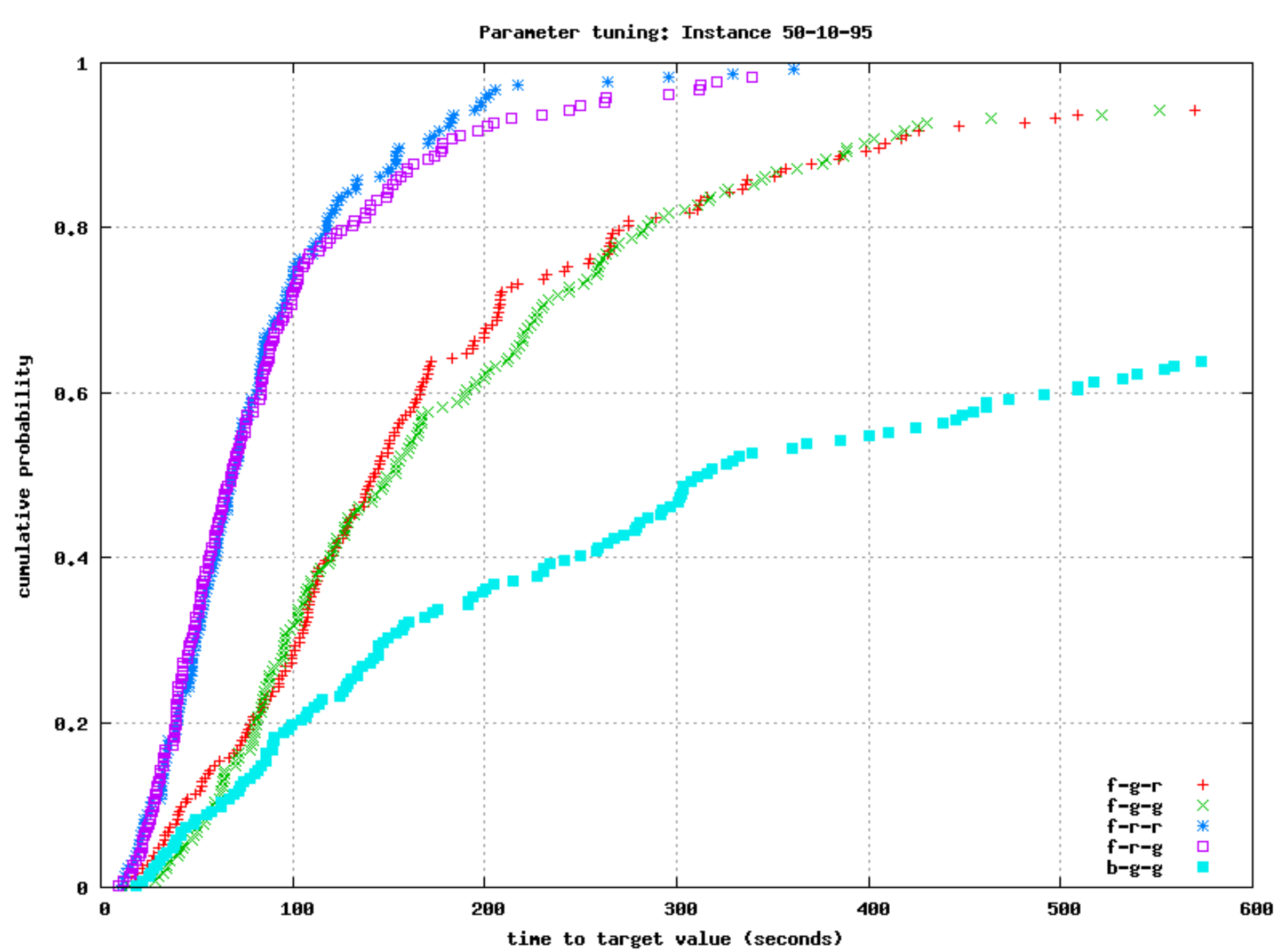
Strategies tested:

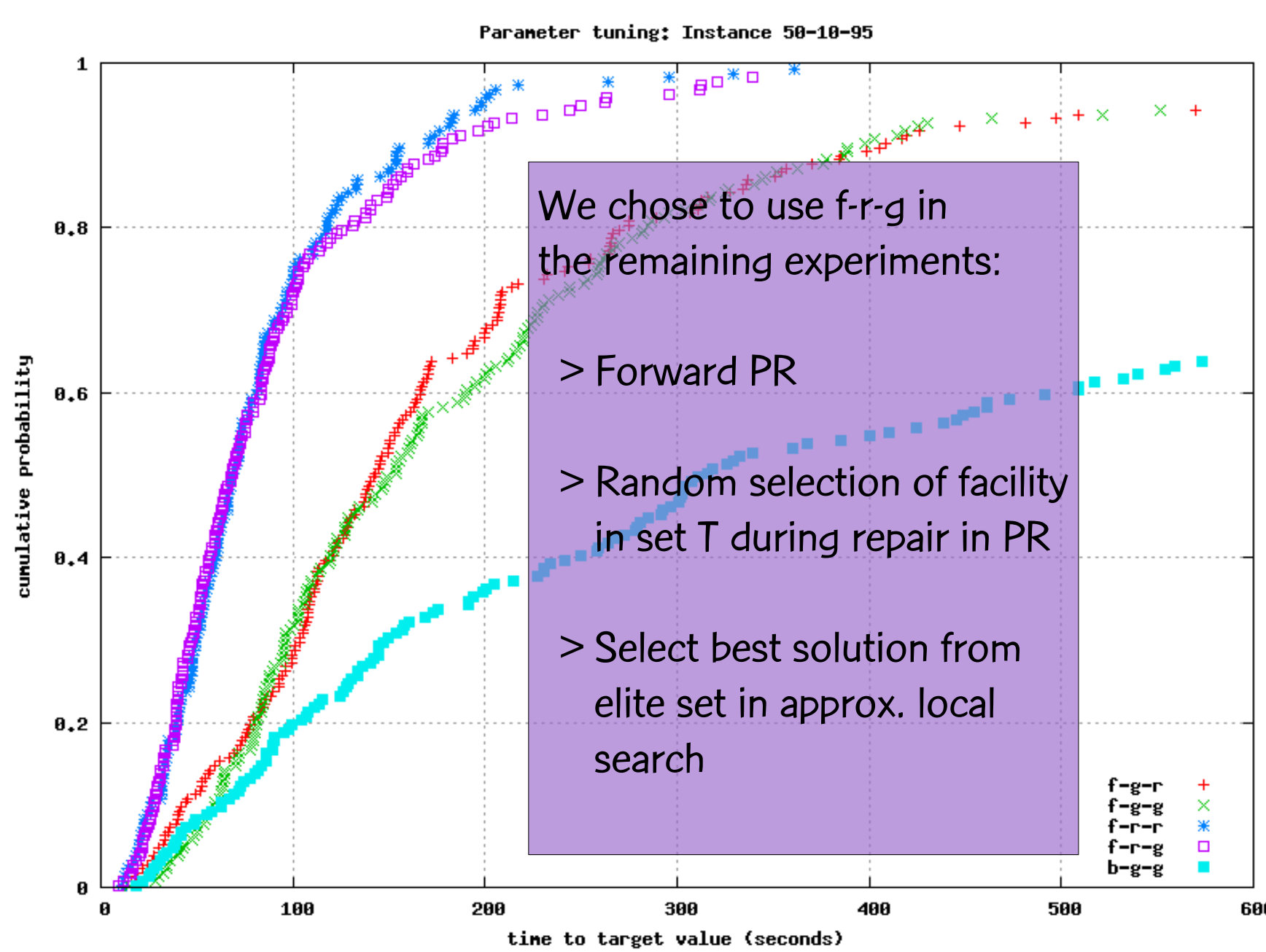
Path-relinking direction: forward (f) or backward (b);

Criteria to select a facility from set T in the repairing procedure: randomly (r) or greedily (g)

Criteria to select a solution from elite set in the approximate local search: randomly (r) or greedily (g).

Combinations: $2^3 = 8$





We chose to use f-r-g in the remaining experiments:

> Forward PR

> Random selection of facility in set T during repair in PR

> Select best solution from elite set in approx. local search

Comparison with other algorithms

Elloumi et al. (2003)

Lee and Ma (2005)

Cordieu et al. (2006)

Hahn et al. (2007)

Pessoa et al. (2008)

Comparison with Elloumi et al (2003):

Method(s): Three linearization methods (L1, L2, and L3), three semidefinite programming formulations (S0, S1, and S2) and a Lagrangian decomposition (D0).

Instances (Elloumi (1991) and Roupin (2004)): For each one of eight types [four configurations (A, B, C, and D) with two classes of instances], five instances with 10 facilities and three locations, and five instances with 20 facilities and five locations. Total of 80 instances

Comparison: GRASP-PR achieved the target values on all instances, with an AVERAGE performance improvement varying between a factor of 7.3 and over 5000 in relation to the BEST average time of the methods of Elloumi et al (2003)

Comparison with Lee and Ma (2005):

Method(s): Three linearization methods (F-Y, K-B, and L3), based on the work of Frieze and Yadegar (1983), Kaufman and Broeckx (1978), and Padberg and Rijal (1996) and a branch and bound method (B&B) based on the work of Burkard (1991).

Instances: Suite of test problems with 10 to 16 facilities and 3 to 8 locations. Total of 25 instances.

Comparison: GRASP-PR found the target value on all 200 runs for each of the instances, with an AVERAGE performance improvement varying between a factor of 11.2 and 1004.6 in relation to the BEST average time of the methods of Lee and Ma (2005)

Comparison with Cordeau et al. (2006):

Method: memetic algorithm.

Instances: problems with 20 to 50 facilities and 6 to 20 locations.
Total = 21 instances

Comparison: GRASP-PR found the target value on all 200 runs for each of the instances, with an AVERAGE performance improvement varying between a factor of 1.5 and 59.2 in relation to the BEST average time of the memetic algorithm, except for instances 30-20-95, 35-15-95, and 50-10-75.

However, for the last two instances the FASTEST GRASP-PR running times were FAR LESS than those of the memetic algorithm.

For instance 30-20-95, the GRASP-PR heuristic found the best solution found by the memetic algorithm but in 44 hours and 47 minutes.

Comparison with Hahn et al. (2007):

Method(s): Level-1 reformulation-linearization technique (RLT) dual ascent procedure in a branch-and-bound scheme.

Instances: Four instances from Elloumi et al. (2003), three instances from Lee and Ma (2005), and one instance from Cordeau et al. (2006). Total of eight instances.

Comparison: GRASP-PR found the target value on all 200 runs for each of the instances, with an AVERAGE performance improvement varying between a factor of 8.8 and over 69,000 w.r.t. the BEST average time of the method of Hahn et al. (2007).

Comparison with Pessoa et al. (2008):

Method: Combination of Hahn et al. (2007) dual ascent procedure with the general-purpose volume algorithm of Barahona and Anbil (2000).

Instances: Four instances from Elloumi et al. (2003), three instances from Lee and Ma (2005), and 12 instances from Cordeau et al. (2006). Total of 24 instances.

Comparison: GRASP-PR found the target value on all 200 runs for each of the instances, with an AVERAGE performance improvement varying between a factor of 132.7 and over 100,000 w.r.t. the BEST average time of the method of Pessoa et al. (2008), except for instance 30-20-95.

Concluding remarks

Concluding remarks

Reviewed hospital layout optimization via QAP

Introduced hospital layout optimization via generalized QAP

Described several heuristics that can be applied to solve this layout problem:

> Greedy

> Randomized greedy

> Local search

> Path-relinking

> GRASP

> GRASP with path-relinking

Coauthor

Ricardo M.A. Silva
Fed. U. of Lavras,
Brazil. Visiting scholar
at AT&T Research (2008-2010)

The End

Slides and full paper can be downloaded from
http://mauricioresende.com

GRASP for the regenerator location problem

Tutorial given at the Spring School in Advances in Operations Research, Higher School of Economics
Nizhny Novgorod, Russia ✤ May 3, 2011

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

Joint work with A. Duarte, R. Martí, and R.M.A. Silva

Summary

- Regenerator location problem (RLP)
- Solution construction procedures
- Local improvement procedure
- GRASP for the RLP
- Experimental results
- Concluding remarks

Regenerator location problem

Reference

A. Duarte, R. Martí, M.G.C.R., and R.M.A. Silva, "Randomized heuristics for the regenerator location problem," AT&T Labs Research Technical Report, Florham Park, NJ, July 13, 2010.

Tech report:

http://www2.research.att.com/~mgcr/doc/gpr-regenloc.pdf

Signal regeneration

- Telecommunication systems use optical signals to transmit information

Signal regeneration

- Telecommunication systems use optical signals to transmit information
- Strength of signal deteriorates and loses power as it gets farther from source

Signal regeneration

- Telecommunication systems use optical signals to transmit information
- Strength of signal deteriorates and loses power as it gets farther from source
- Signal must be regenerated periodically to reach destination

Signal regeneration

- Telecommunication systems use optical signals to transmit information
- Strength of signal deteriorates and loses power as it gets farther from source
- Signal must be regenerated periodically to reach destination: Regenerators

Signal regeneration

- Telecommunication systems use optical signals to transmit information
- Strength of signal deteriorates and loses power as it gets farther from source
- Signal must be regenerated periodically to reach destination: Regenerators
- Regenerators are expensive: minimize the number of regenerators in the network

Regenerator location problem (RLP)

- Given:
  - Graph $G=(V,E)$, where V are vertices, E are edges, where edge (i,j) has a real-valued length $d(i,j) > 0$
  - $D > 0$ is the maximum length that a signal can travel before it must be regenerated

Regenerator location problem (RLP)

- Find:
  - Paths that connect all pairs of nodes in $V \times V$
  - Set of nodes where it is necessary to locate single regenerators
- Minimize number of deployed regenerators

Regenerator location problem (RLP)

- Path between $\{s,t\} \in V \times V$
  - $\{ (s,v[1]), (v[1],v[2]), \ldots,(v[k],t) \}$ is formed by one or more path segments
- Path segment is sequence of consecutive edges
  - $\{ (v[i],v[i+1]), (v[i+1],v[i+2]), \ldots,(v[q-1],v[q]) \}$ in the path satisfying the condition

$$d(v[i],v[i+1]) + d(v[i+1],v[i+2]) + \cdots + d(v[q-1],v[q]) \leq D$$

Regenerator location problem (RLP)

- Path between $\{s,t\} \in V \times V$
  - $\{ (s,v[1]), (v[1],v[2]), \ldots,(v[k],t) \}$ is formed by one or more path segments
- Path segment is sequence of consecutive edges
  - $\{ (v[i],v[i+1]), (v[i+1],v[i+2]), \ldots,(v[q-1],v[q]) \}$ in the path satisfying the condition

$$d(v[i],v[i+1]) + d(v[i+1],v[i+2]) + \cdots + d(v[q-1],v[q]) \leq D$$

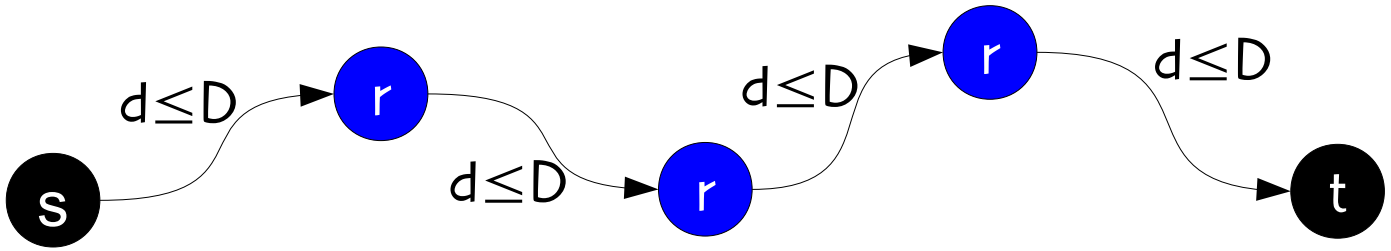
Path segment must not be longer than D

Regenerator location problem (RLP)

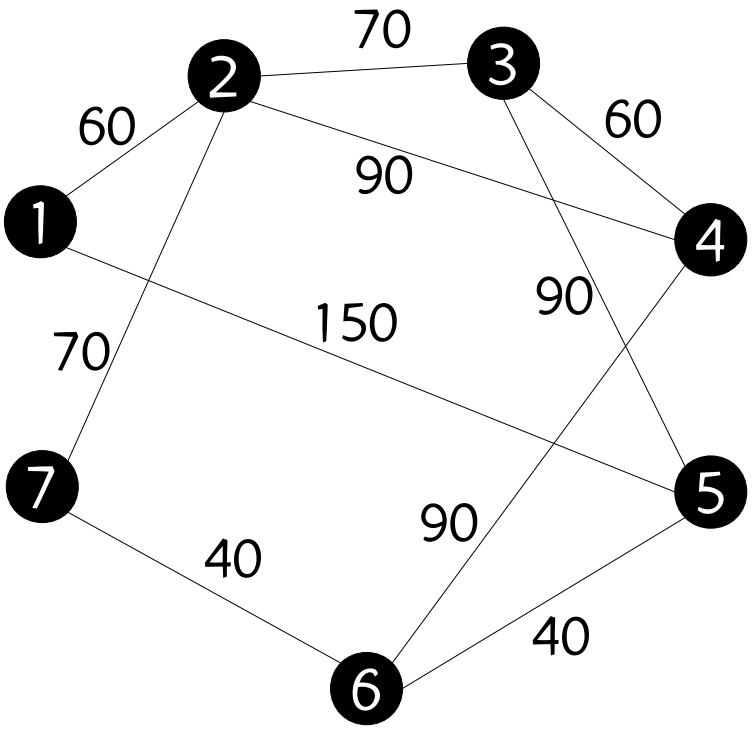
- If total length of path is no more than D, then path consists of a single path segment
- Otherwise, it consists of two or more segments
  - Regenerators will be located in the internal nodes of the path

Regenerator location problem (RLP)

- If total length of path is no more than D, then path consists of a single path segment
- Otherwise, it consists of one or more segments
  - Regenerators will be located in the internal nodes of the path



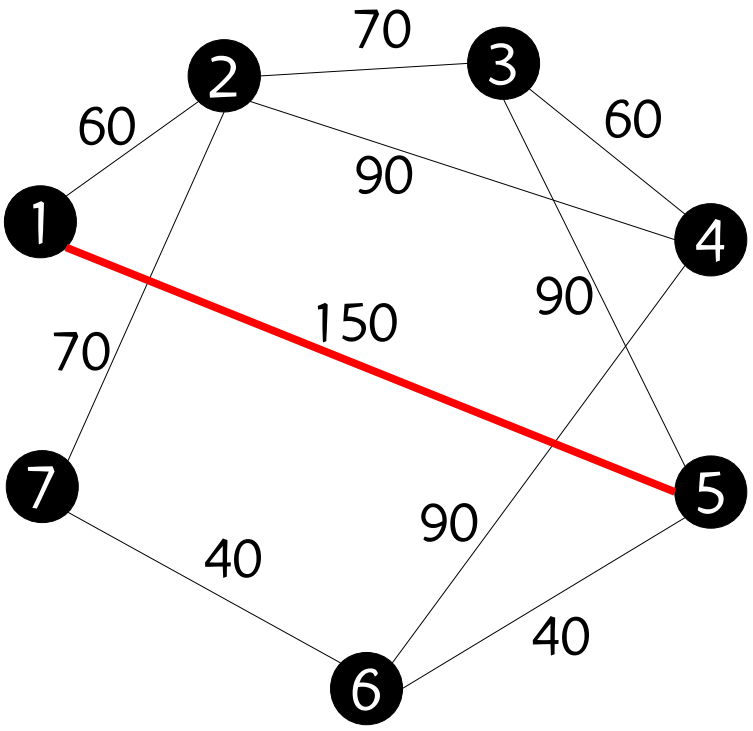
Regenerator location problem (RLP)



D = 100

7-node graph with D = 100

Regenerator location problem (RLP)

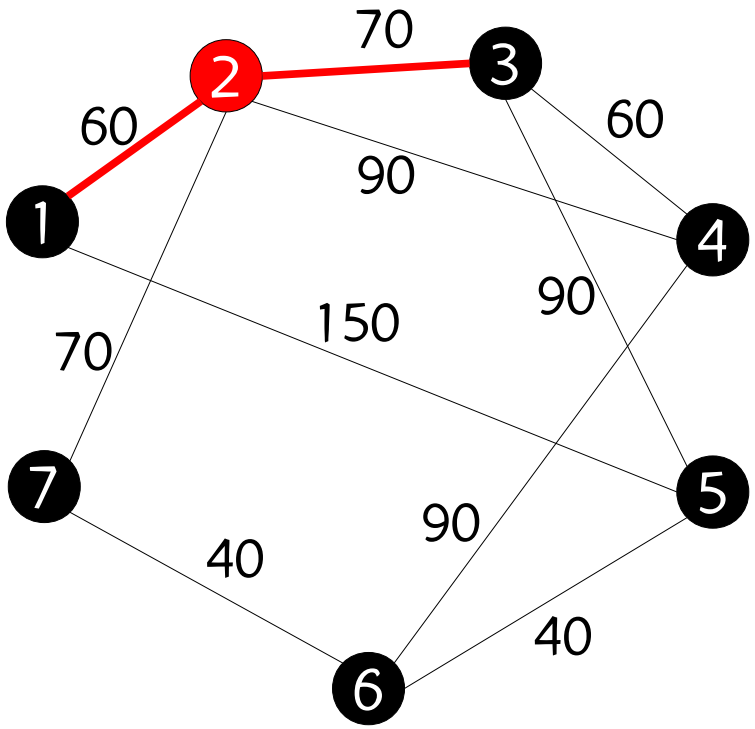


D = 100

(1) Note that:

- D(1,5) = 150 > 100 = D
- Edge (1,5) cannot be part of any path

Regenerator location problem (RLP)



D = 100

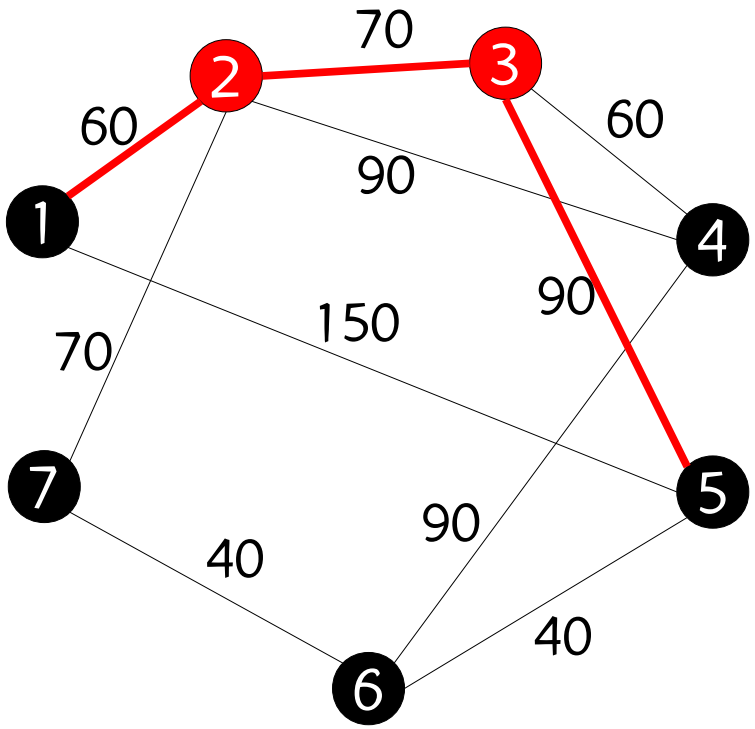
(2) Note that:

- Shortest path from 1 to 3 is { (1,2), (2,3) } with total length

  60 + 70 = 130 > 100 = D

- Must be decomposed into two path segments { (1,2) } and { (2,3) } with a regenerator in node 2

Regenerator location problem (RLP)

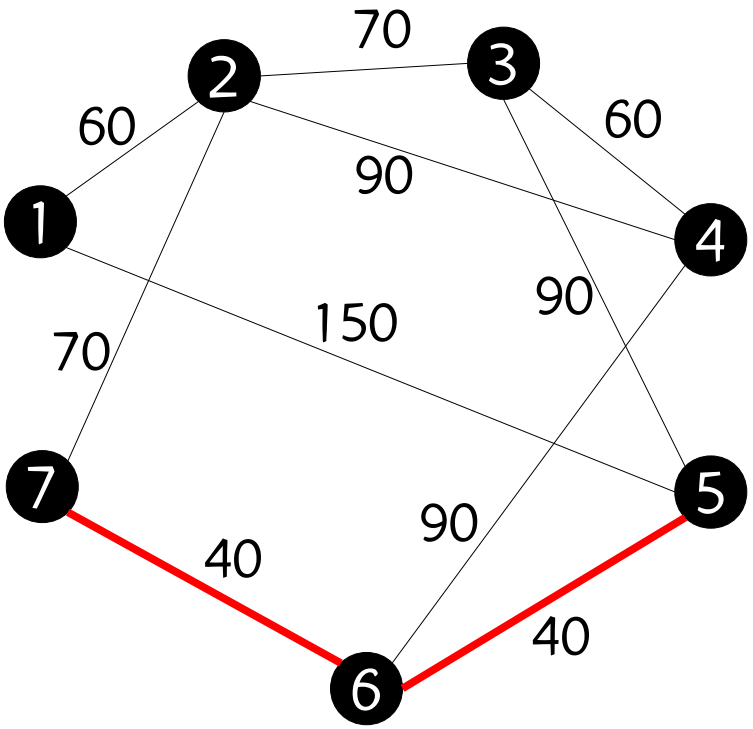


D = 100

(3) Note that:

- Shortest feasible path from 1 to 5 is { (1,2), (2,3), (3,5) } with total length
  60 + 70 + 90 = 220 > 100 = D
- Must be decomposed into three path segments { (1,2) }, { (2,3) }, and { (3,5) } with regenerators in nodes 2 and 3

Regenerator location problem (RLP)

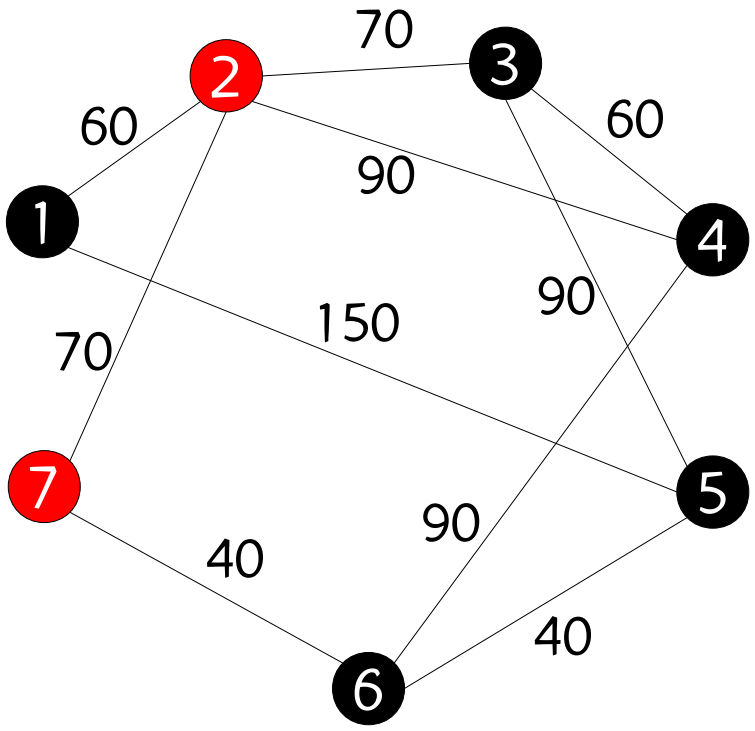


D = 100

(4) Note that:

- Shortest feasible path from 5 to 7 is { (5,6), (6,7) } with total length 40 + 40 = 80 ≤ 100 = D
- No regenerator is needed to connect nodes 5 and 7

Regenerator location problem (RLP)



D = 100

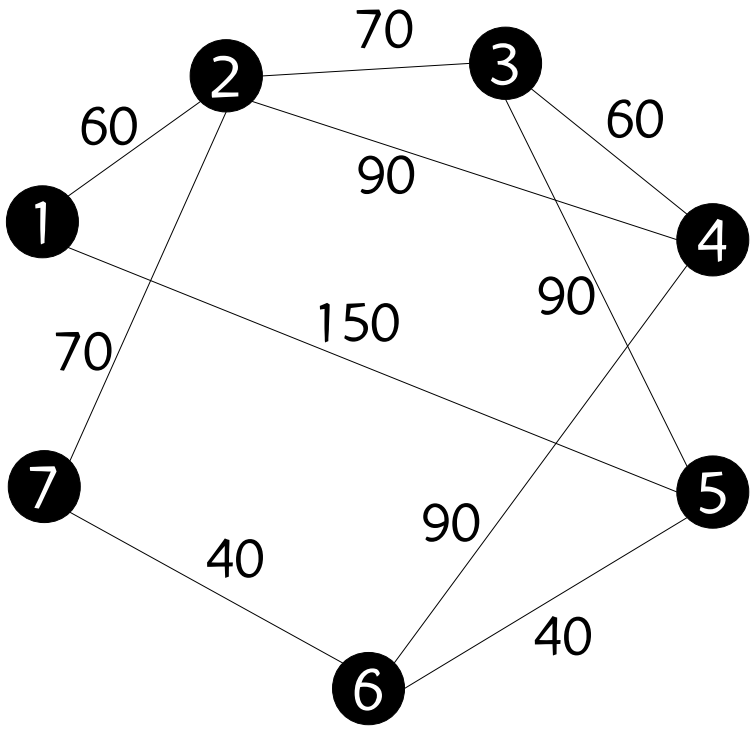
(5) Note that:

- Placing regenerators in nodes 2 and 7 allows for communication between all pairs of nodes in the graph

Related work

- Yetginer & Karasan (2003): regenerator placement in context of traffic engineering with restoration
- Gouveia et al. (2003): network design problem that forbids path segments between components that are longer than a maximum length
- Chen & Raghavan (2007): introduce RLP & greedy heuristic
- Chen et al. (2010): introduce branch & cut scheme and new heuristics; prove NP-hardness
- Flammini et al. (2009): prove NP-hardness

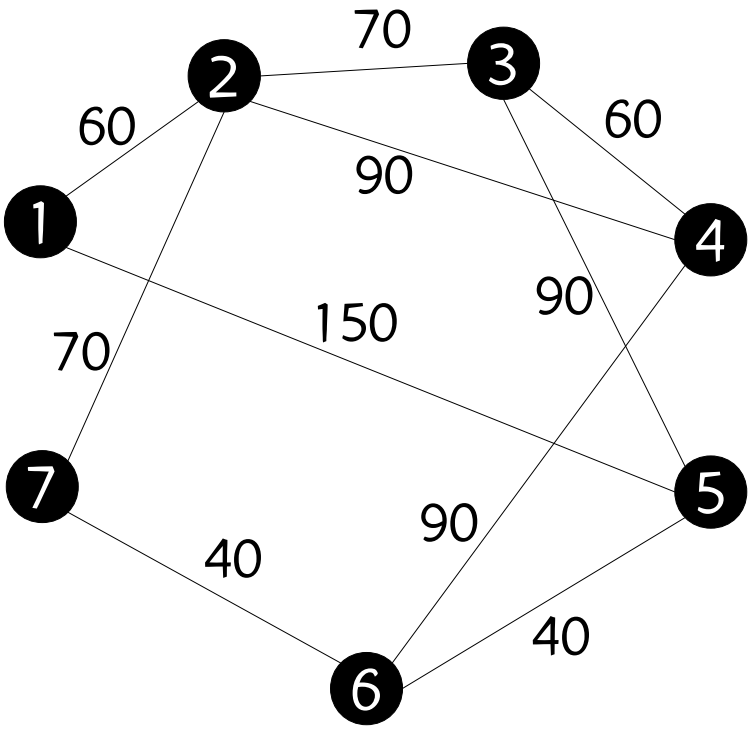
Communication graph (Chen et al., 2010)



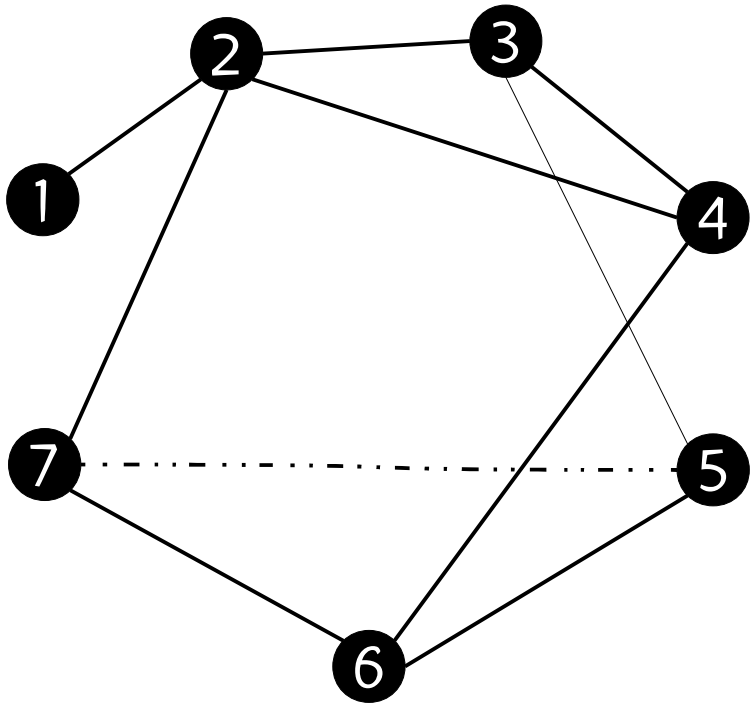
G = (V,E)
D = 100

- Given weighted graph G
  - Delete all edges having length greater than D
  - For all non-adjacent nodes, add an edge between them of length equal to the corresponding shortest path in G if it is no longer than D
  - Disregard all length info

Communication graph (Chen et al., 2010)

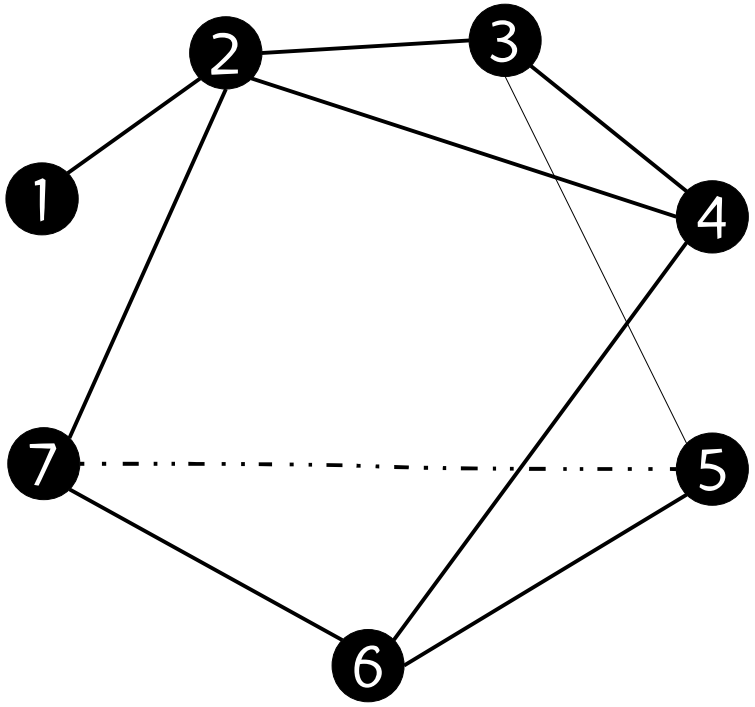


G = (V,E)
D = 100



M = (V,E')

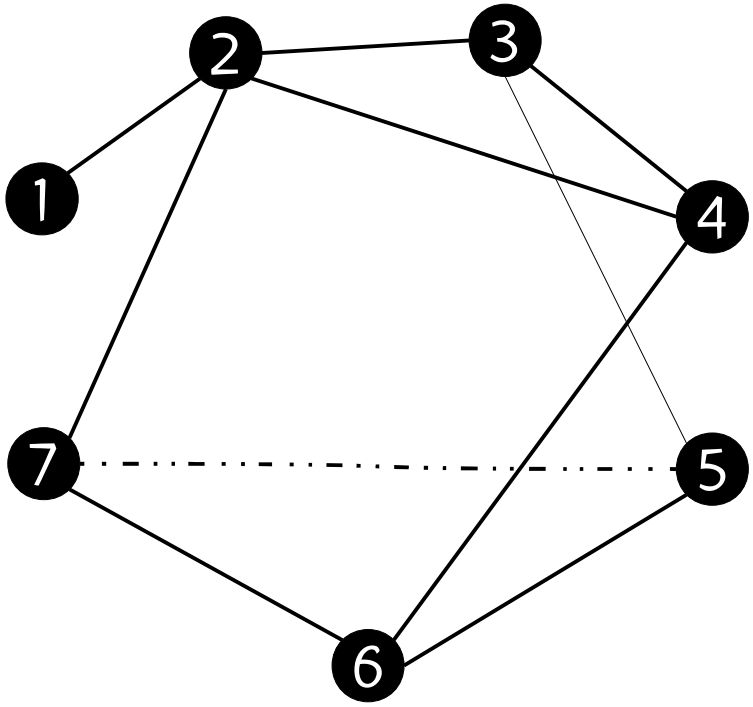
Communication graph (Chen et al., 2010)



M = (V,E')

- If M is complete, then there is no need for regenerators
- If M is not connected, then the problem is infeasible
- Otherwise, one or more regenerators are needed

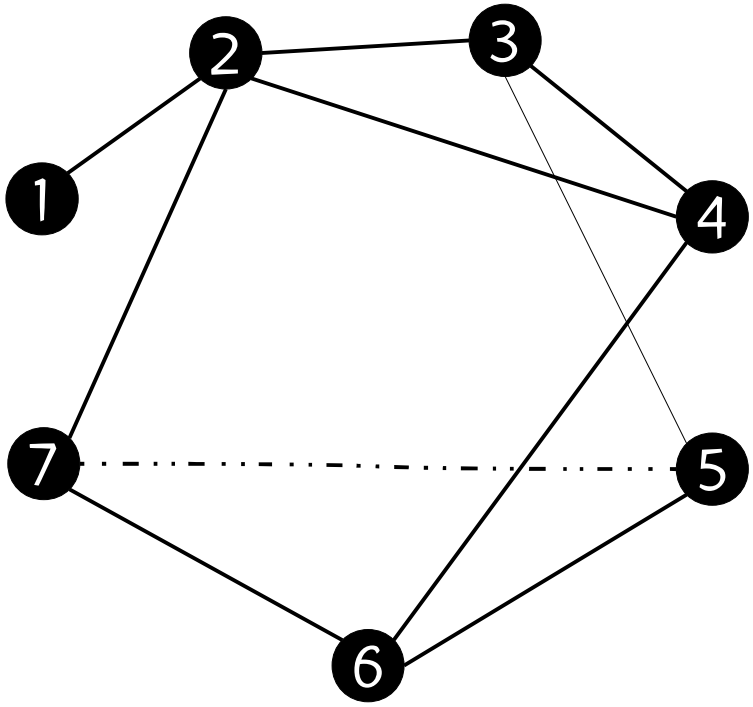
Greedy algorithm (Chen et al., 2010)



M = (V,E')

- Works on communication graph M
- Input: set of nodes not directly connected (NDC) in M and builds a set R of regenerator nodes
- At each step the procedure determines a node u* whose inclusion in R enables the connection of the largest number g(u*) of yet unconnected pairs X(u*) in M
- Node u* is added to R and M is updated

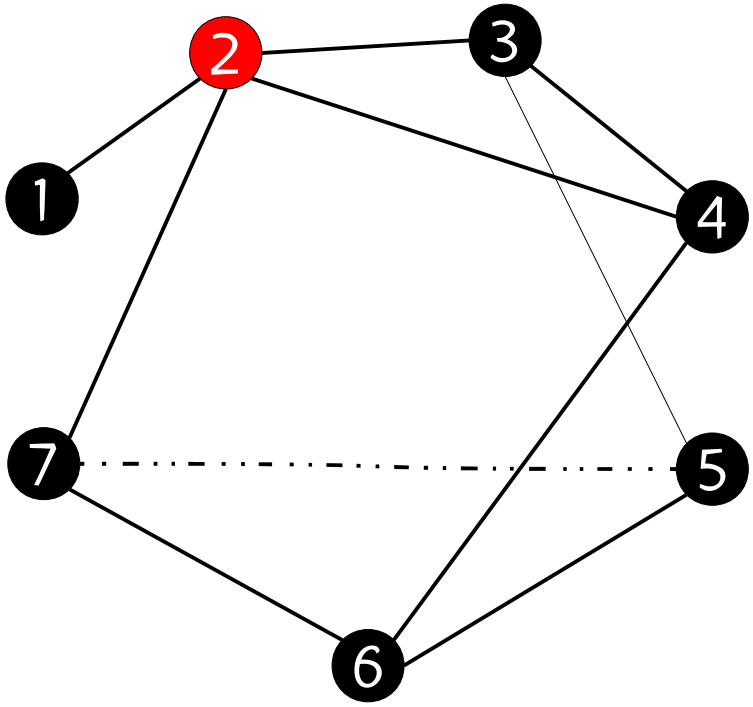
Greedy algorithm (Chen et al., 2010)



M = (V,E')

u	X(u)	g(u)
1	∅	0
2	{ (1,3),(1,4),(1,7),(3,7),	5
3	{ (4,5),(2,5) }	2
4	{ (2,6),(3,6) }	2
5	{ (3,7),(3,6) }	2
6	{ (4,7),(4,5) }	2
7	{ (2,6),(2,5) }	2

Greedy algorithm (Chen et al., 2010)

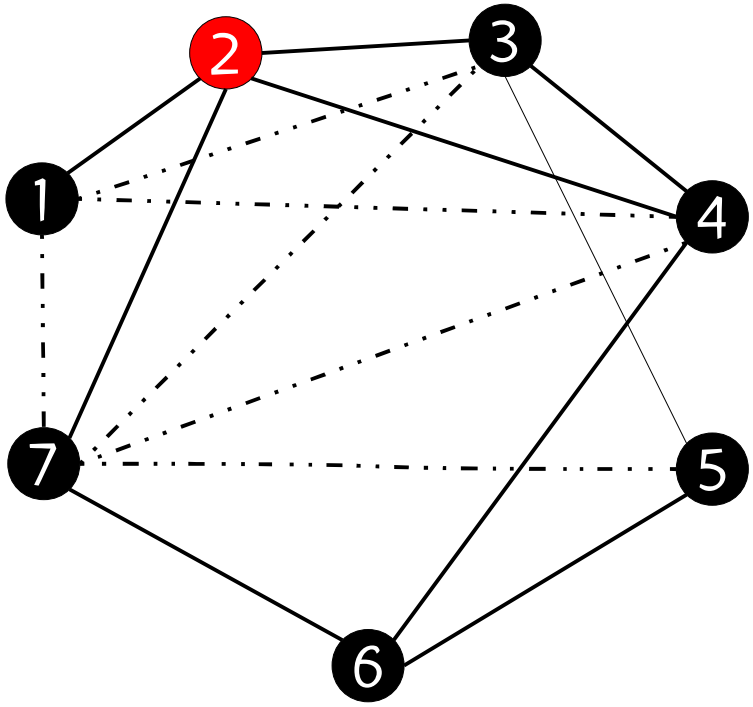


M = (V,E')

u	X(u)	g(u)
1	∅	0
2	**{ (1,3),(1,4),(1,7),(3,7),**	**5**
3	{ (4,5),(2,5) }	2
4	{ (2,6),(3,6) }	2
5	{ (3,7),(3,6) }	2
6	{ (4,7),(4,5) }	2
7	{ (2,6),(2,5) }	2

Add regenerator to node 2

Greedy algorithm (Chen et al., 2010)

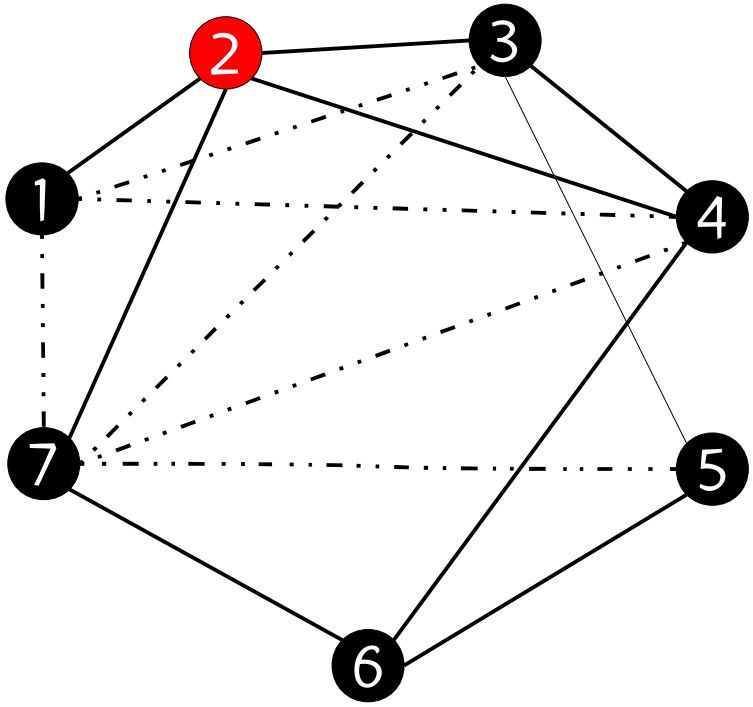


M = (V,E')

u	X(u)	g(u)
1	∅	0
2	**{ (1,3),(1,4),(1,7),(3,7),**	**5**
3	{ (4,5),(2,5) }	2
4	{ (2,6),(3,6) }	2
5	{ (3,7),(3,6) }	2
6	{ (4,7),(4,5) }	2
7	{ (2,6),(2,5) }	2

Update M to account for regenerator in node 2

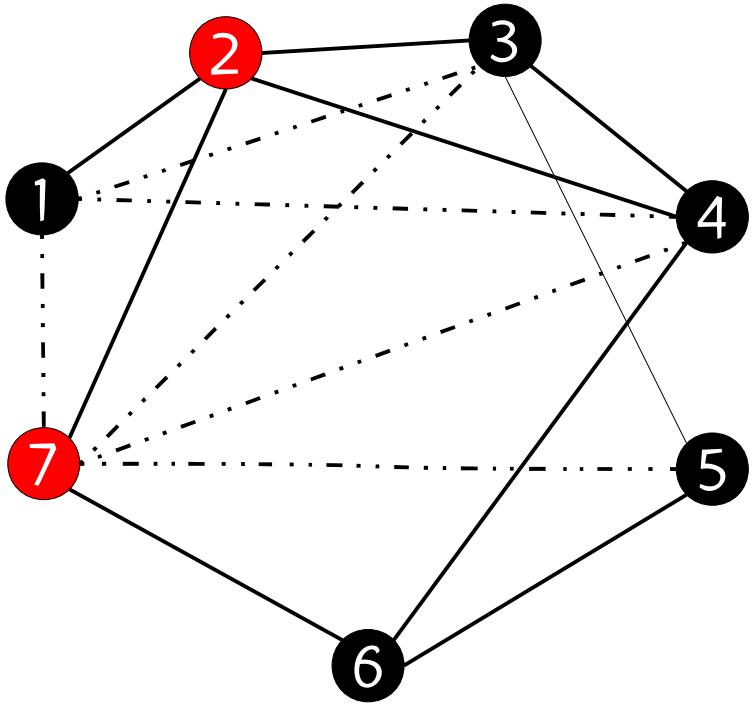
Greedy algorithm (Chen et al., 2010)



M = (V,E')

u	X(u)	g(u)
1	∅	0
-	-	-
3	{ (1,5),(2,5),(4,5) }	3
4	{ (1,6),(2,6),(3,6) }	3
5	{ (3,6) }	1
6	{ (4,5) }	1
7	{ (1,5),(1,6),(2,5),(2,6),(3,6),(4,5) }	6

Greedy algorithm (Chen et al., 2010)

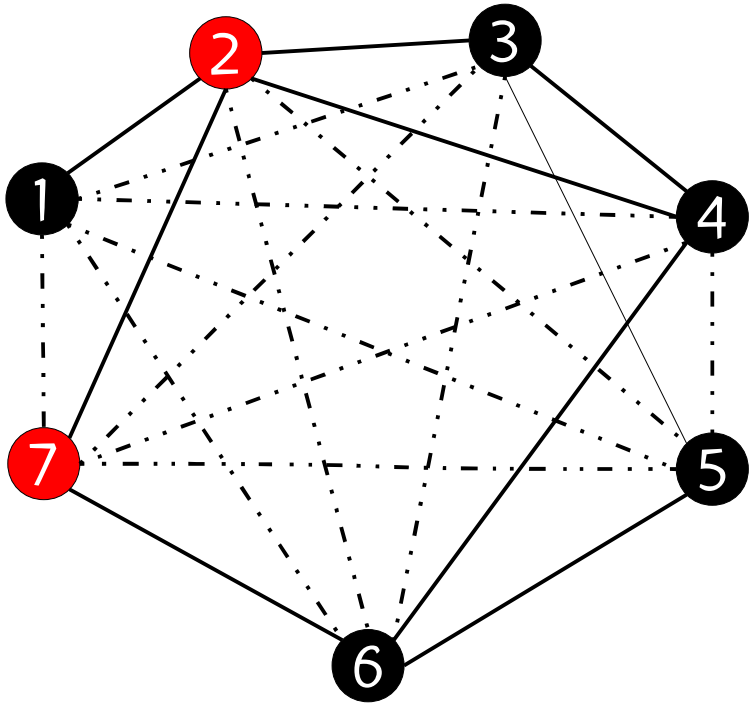


M = (V,E')

u	X(u)	g(u)
1	∅	0
-	-	-
3	{ (1,5),(2,5),(4,5) }	3
4	{ (1,6),(2,6),(3,6) }	3
5	{ (3,6) }	1
6	{ (4,5) }	1
7	**{ (1,5),(1,6),(2,5),(2,6),(3,6),(4,5) }**	**6**

Add regenerator to node 7

Greedy algorithm (Chen et al., 2010)

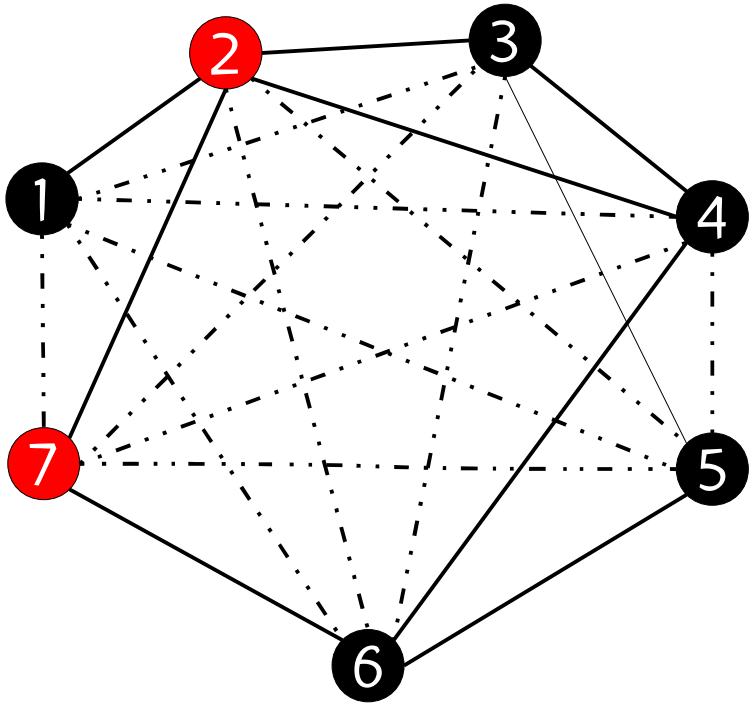


M = (V,E')

u	X(u)	g(u)
1	∅	0
-	-	-
3	{ (1,5),(2,5),(4,5) }	3
4	{ (1,6),(2,6),(3,6) }	3
5	{ (3,6) }	1
6	{ (4,5) }	1
7	**{ (1,5),(1,6),(2,5),(2,6),(3,6),(4,5) }**	**6**

Update M to account for regenerator in node 7

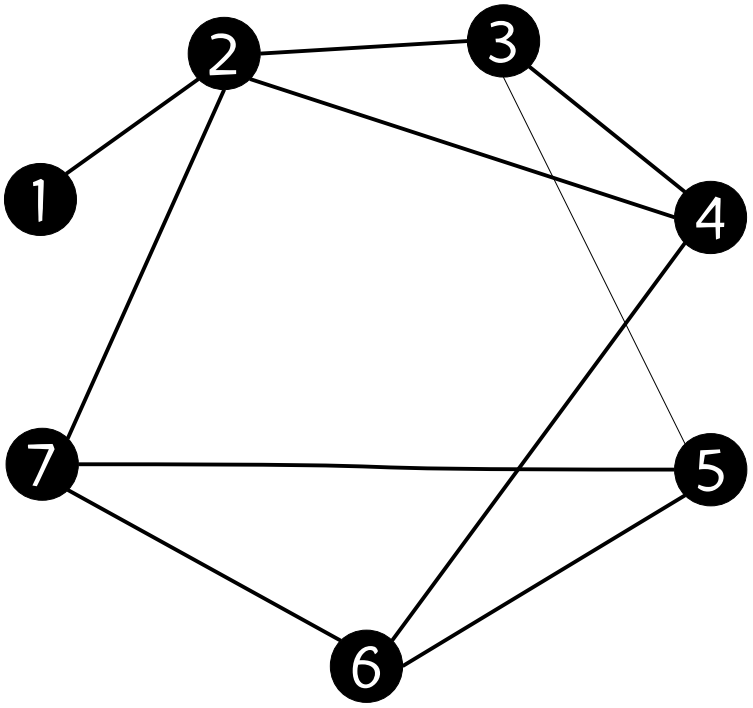
Greedy algorithm (Chen et al., 2010)



M = (V,E')

Since M is complete, all pairs can communicate and solution R = {2,7}

H1 heuristic (Chen et al., 2010)

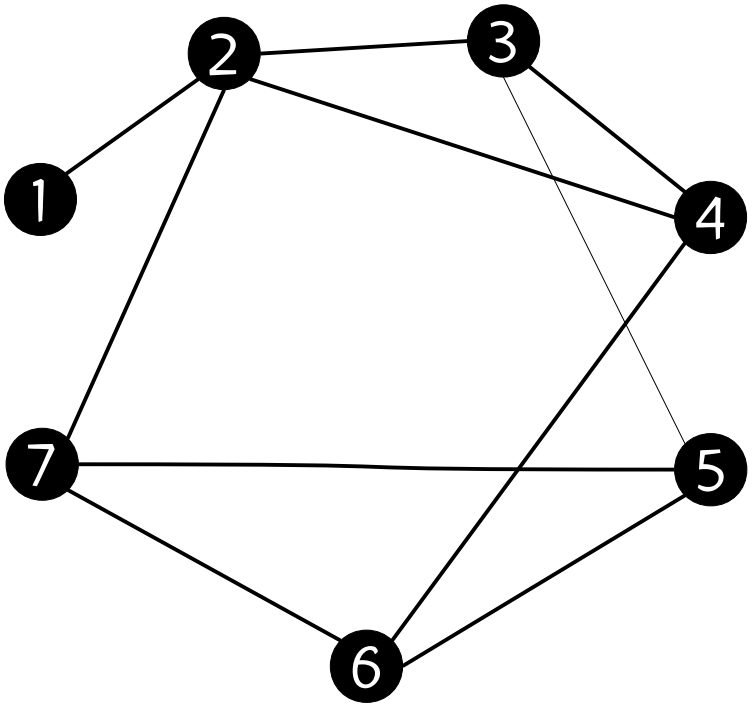


Communication graph M

Aim of H1 is find a spanning tree in M having the maximum number of leaves, thus minimizing the number of internal nodes.

Regenerators are assigned to the internal nodes.

H1 heuristic (Chen et al., 2010)



Communication graph M

$u^* = \text{argmin} \{ \deg(u): u \in V \}$

$S \leftarrow \{ u^* \}$ (spanning tree)

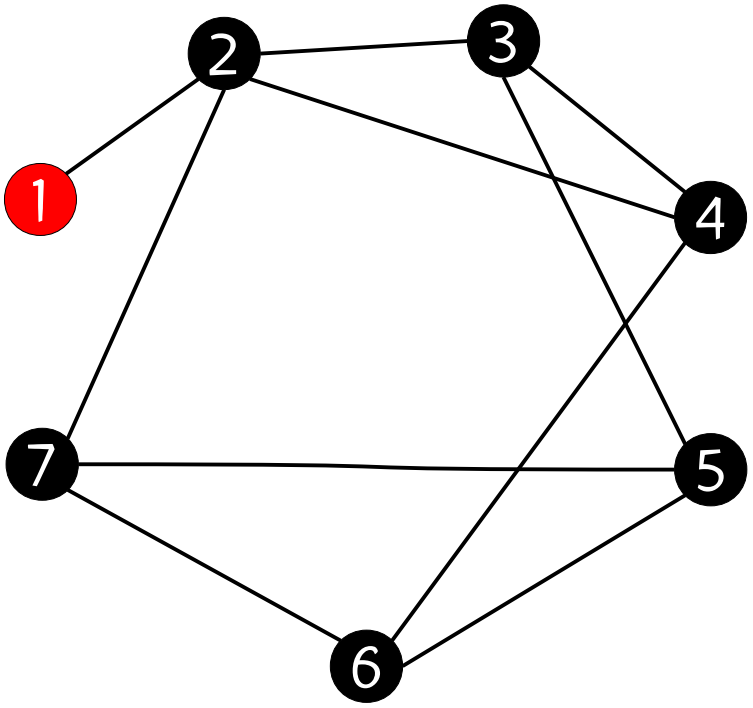
$\overline{S} \leftarrow V \setminus \{ u^* \}$

$R \leftarrow \emptyset$ (regenerators)

Call recursive function

$\text{Tree}(u^*, S, \overline{S}, R)$

H1 heuristic (Chen et al., 2010)



Communication graph M

$u^* = \text{argmin} \{ \deg(u): u \in V \}$

$S \leftarrow \{ u^* \}$ (spanning tree)

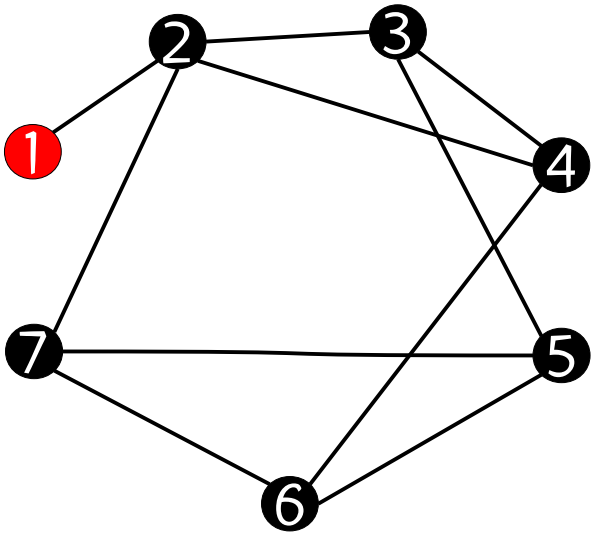
$\overline{S} \leftarrow V \setminus \{ u^* \}$

$R \leftarrow \emptyset$ (regenerators)

Call recursive function

$\text{Tree}(u^*, S, \overline{S}, R)$

H1 heuristic (Chen et al., 2010)



$u = 1$; $S = \{1\}$; $\bar{S} = \{2,3,4,5,6,7\}$

$U(1) = \{ 2 \}$

$S = \{1,2\}$; $\bar{S} = \{3,4,5,6,7\}$

$deg_{\bar{S}}(2) = 3$

$u^* = 2$

$R \leftarrow R \cup \{ 2 \} = \{ 2 \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S| > 0$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

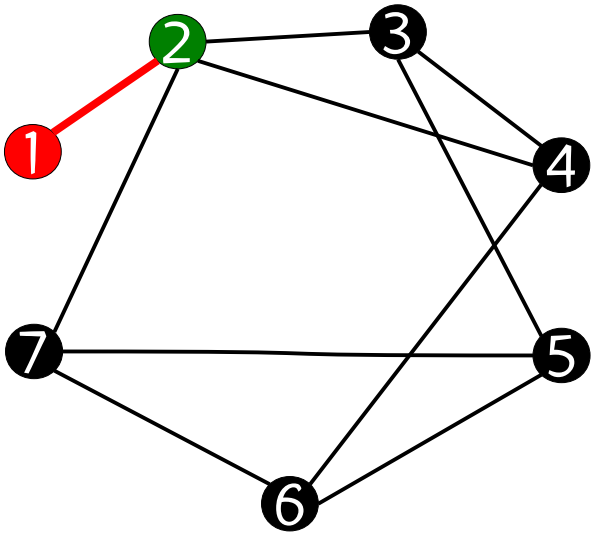
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H1 heuristic (Chen et al., 2010)



$u = 1$; $S = \{1\}$; $\bar{S} = \{2,3,4,5,6,7\}$

$U(1) = \{ 2 \}$

$S = \{1,2\}$; $\bar{S} = \{3,4,5,6,7\}$

$deg_{\bar{S}}(2) = 3$

$u^* = 2$

$R \leftarrow R \cup \{ 2 \} = \{ 2 \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S| > 0$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

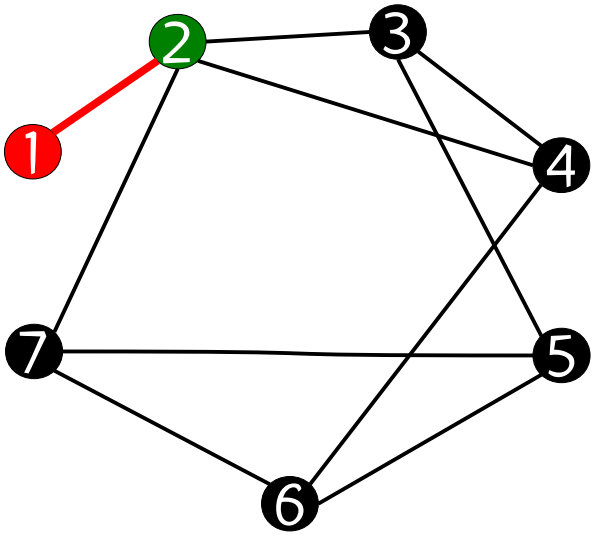
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H1 heuristic (Chen et al., 2010)



$u = 2$; $S = \{1,2\}$; $\bar{S} = \{3,4,5,6,7\}$

$U(2) = \{ 3,4,7 \}$

$S = \{1,2,3,4,7\}$; $\bar{S} = \{5,6\}$

$deg_{\bar{S}}(3) = 1$

$deg_{\bar{S}}(4) = 1$

$deg_{\bar{S}}(7) = 2$

$u^* = 7$

$R \leftarrow R \cup \{ 7 \} = \{ 2,7 \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S| > 0$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

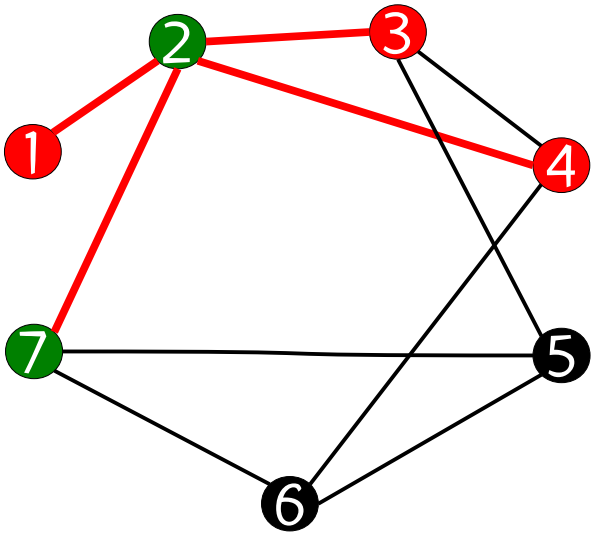
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H1 heuristic (Chen et al., 2010)



$u = 2$; $S = \{1,2\}$; $\bar{S} = \{3,4,5,6,7\}$

$U(2) = \{ 3,4,7 \}$

$S = \{1,2,3,4,7\}$; $\bar{S} = \{5,6\}$

$deg_{\bar{S}}(3) = 1$

$deg_{\bar{S}}(4) = 1$

$deg_{\bar{S}}(7) = 2$

$u^* = 7$

$R \leftarrow R \cup \{ 7 \} = \{ 2,7 \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S| > 0$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

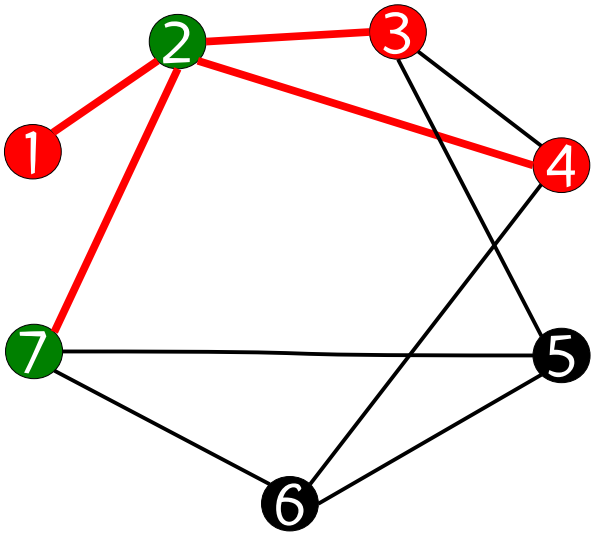
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H1 heuristic (Chen et al., 2010)



$u = 7$; $S = \{1,2,3,4,7\}$; $\bar{S} = \{5,6\}$
$U(7) = \{ 5,6 \}$
$S = \{1,2,3,4,5,6,7\}$; $\bar{S} = \{ \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S|>1$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

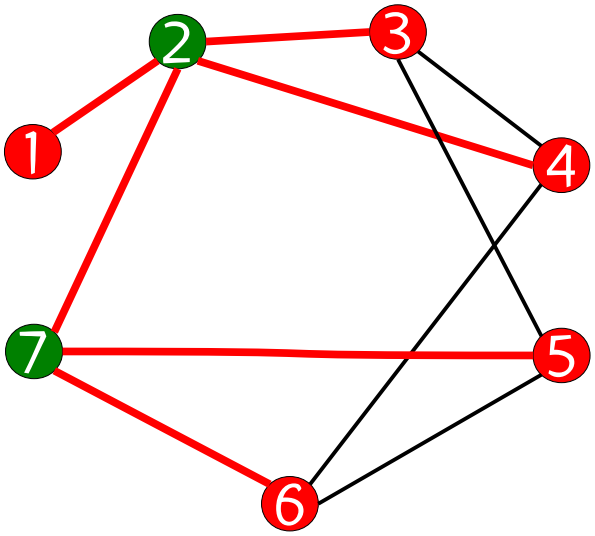
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H1 heuristic (Chen et al., 2010)



$u = 7$; $S = \{1,2,3,4,7\}$; $\bar{S} = \{5,6\}$
$U(7) = \{ 5,6 \}$
$S = \{1,2,3,4,5,6,7\}$; $\bar{S} = \{ \}$

Tree(u,S,$\bar{S}$,R)

$U(u) \leftarrow N(u) \cap \bar{S}$

if ($|S|>1$) then

  $S \leftarrow S \cup U(u)$

  $\bar{S} \leftarrow V \setminus S$

end if

while $U(u) \neq 0$ do

  Compute $deg_{\bar{S}}(v)$ for all $v \in U(u)$

  $u^* \leftarrow$ argmax { $deg_{\bar{S}}(v)$: $v \in U(u)$ }

  if $deg_{\bar{S}}(u^*) > 0$ then

    $R \leftarrow R \cup \{ u^* \}$

    Tree(u^*,S,$\bar{S}$,R)

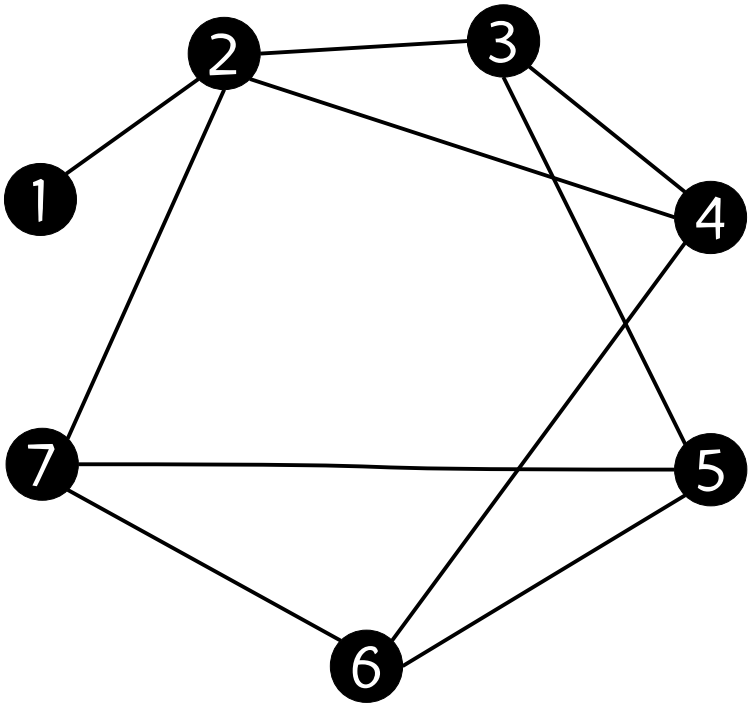
  end if

  $U(u) \leftarrow N(u) \setminus \{ u^* \}$

end while

return

H2 heuristic (Chen et al., 2010)

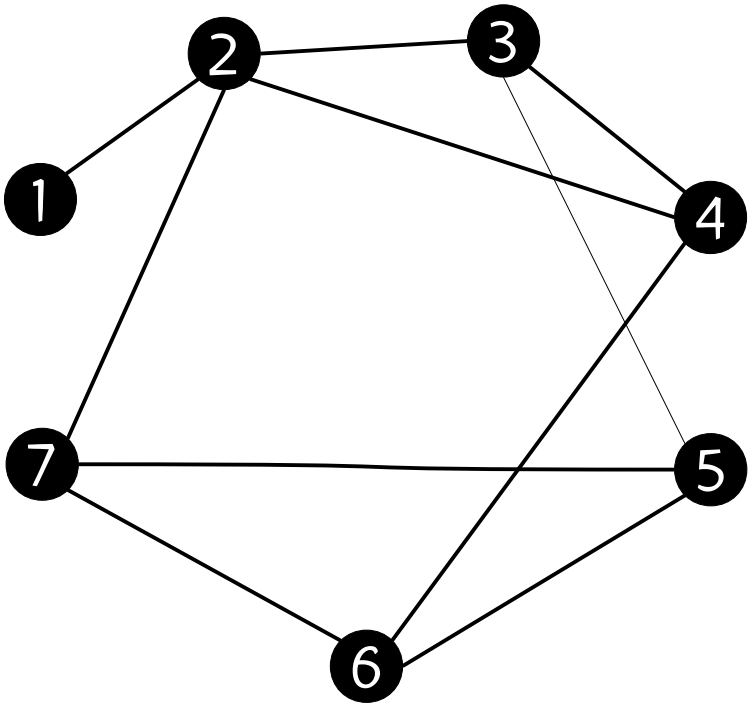


Communication graph M

while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

H2 heuristic (Chen et al., 2010)

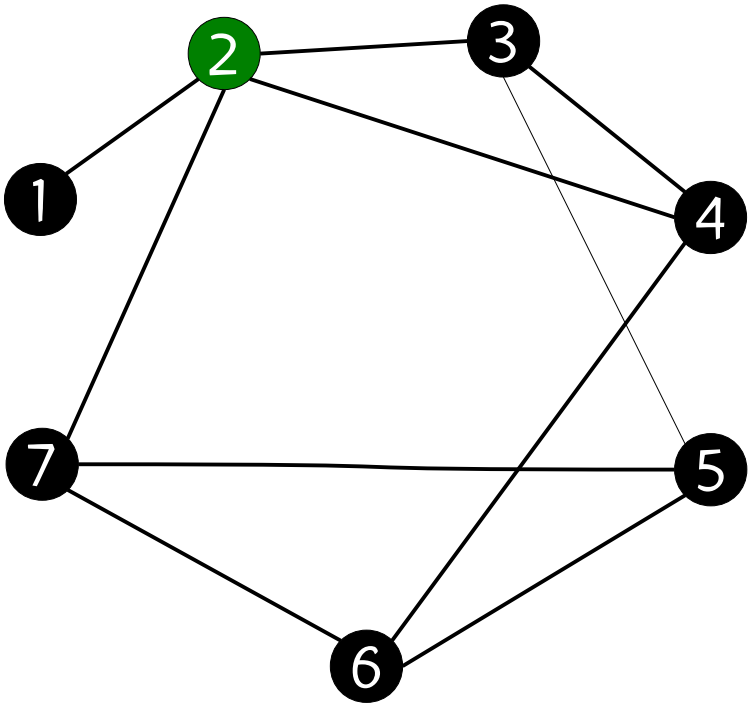


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {1}

H2 heuristic (Chen et al., 2010)

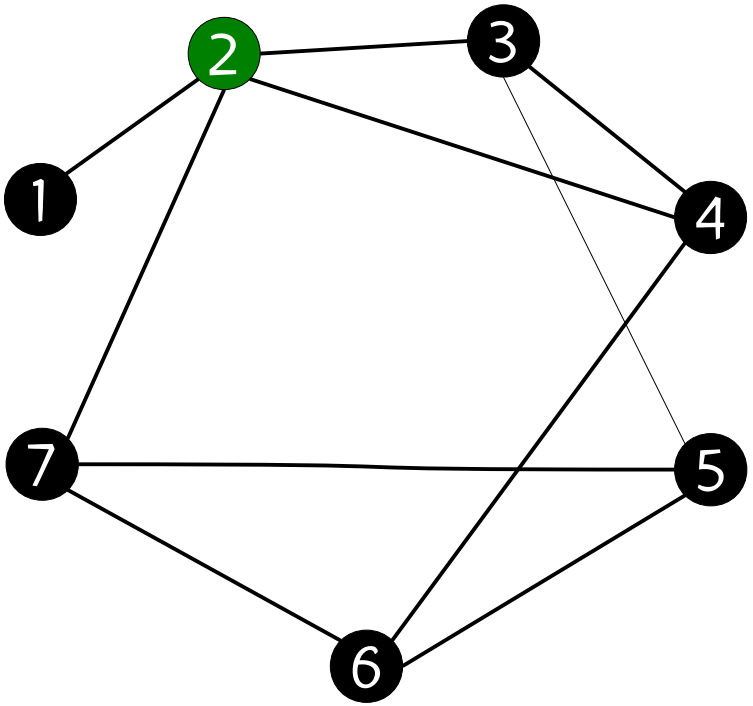


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {1}
Max degree neighbor of {1} is {2}
Add regenerator to {2}

H2 heuristic (Chen et al., 2010)

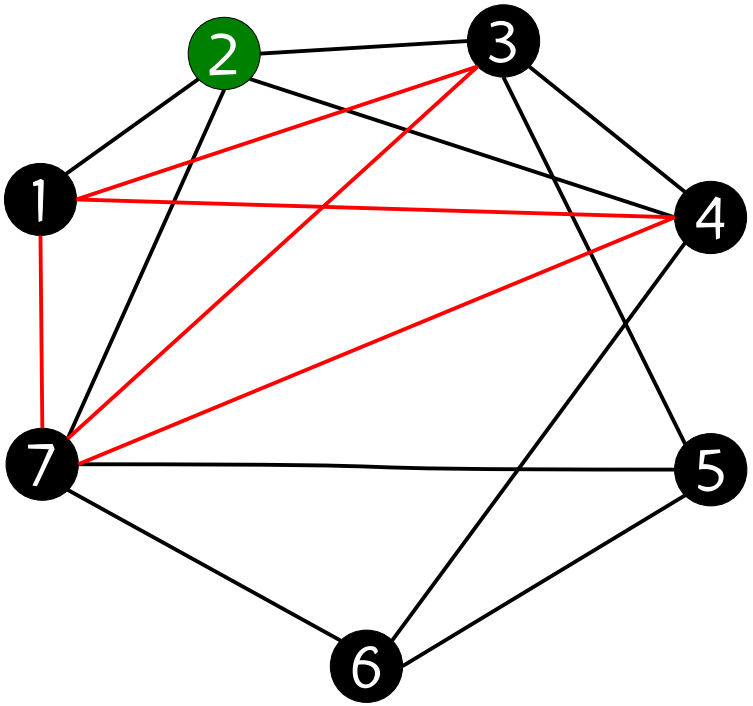


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {1}
Max degree neighbor of {1} is {2}
Add regenerator to {2}

H2 heuristic (Chen et al., 2010)

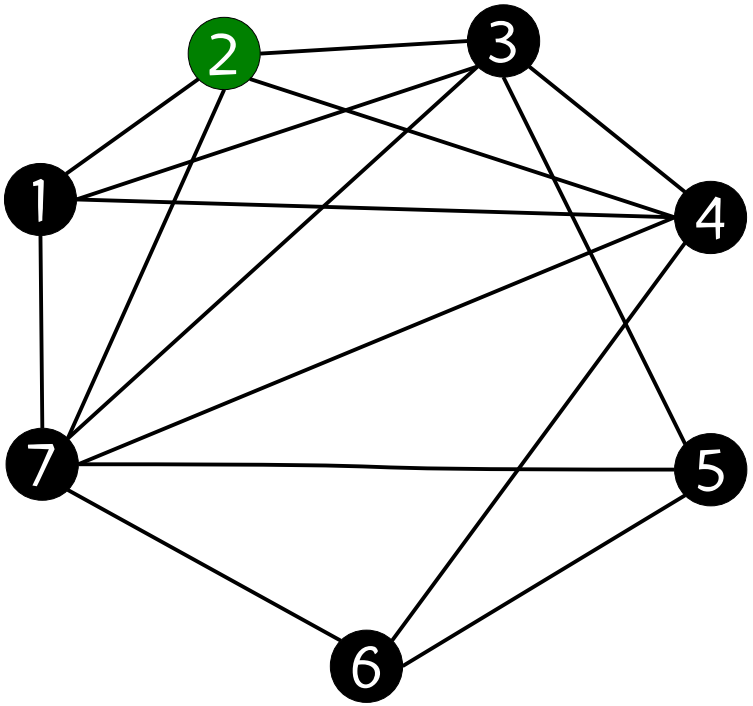


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {1}
Max degree neighbor of {1} is {2}
Add regenerator to {2}
Update M: add { (1,3), (1,4), (1,7), (3,7), (4,7) }

H2 heuristic (Chen et al., 2010)

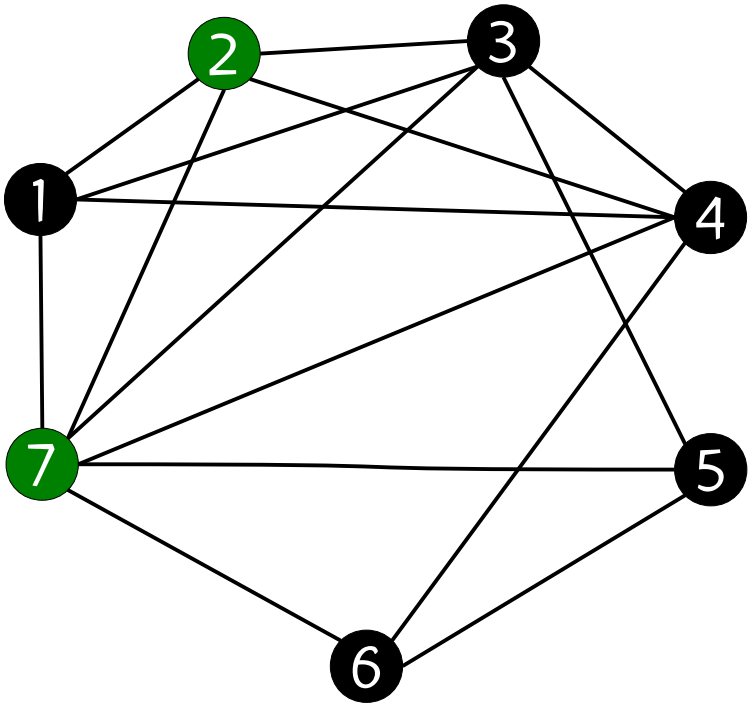


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {5}
Max degree neighbors of {5} are {3,7}

H2 heuristic (Chen et al., 2010)

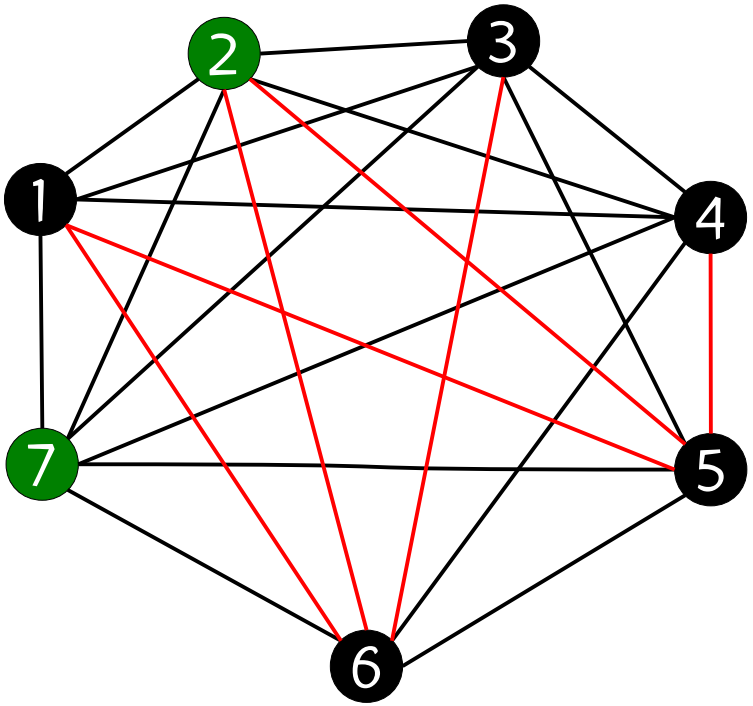


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {5} or {6}. Pick {5}.
Max degree neighbors of {5} is {7}.
Add regenerator to {7}

H2 heuristic (Chen et al., 2010)

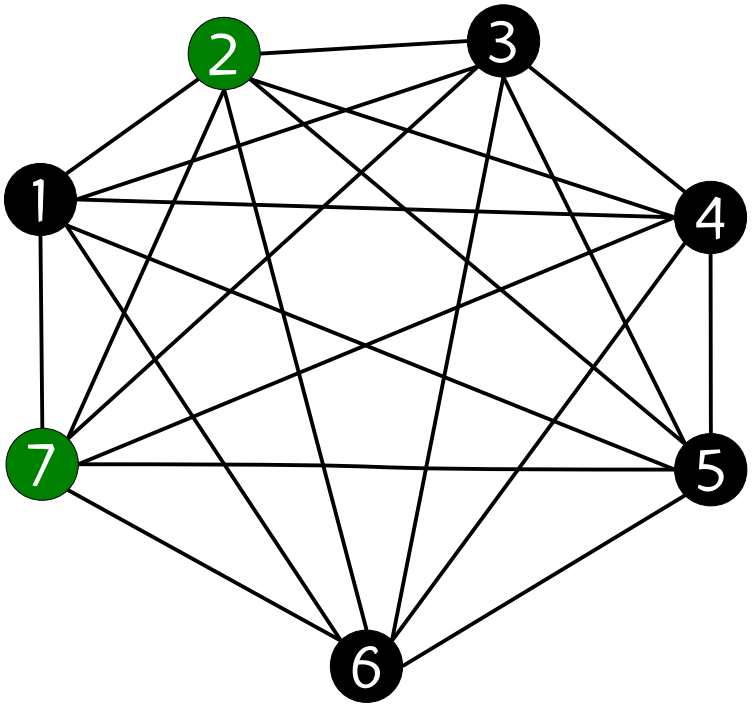


while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

Min degree node is {5}
Max degree neighbors of {5} are {3,7}
Add regenerator to {7}
Update M: add { (1,5), (1,6), (2,5), (2,6), (3,6), (4,5) }

H2 heuristic (Chen et al., 2010)



while M is not complete do

Find lowest degree node u* in M
Assign regenerator to neighbor v* of u*
having max degree
Update M adding links that can now
communicate because of v*

M is complete!

R = { 2, 7 }

GRASP heuristics

GREEDY, H1, and H2 are greedy heuristics.

- In GREEDY: pick the node u^* which maximizes the number of unconnected pairs that become connected if a regenerator is added at u^*.
- In H1: pick the node u^* that maximizes $deg_{\bar{S}}(v)$ for $v \in U(u)$.
- In H2: pick the maximum degree node u^* adjacent to the node v^* of minimum degree in M.

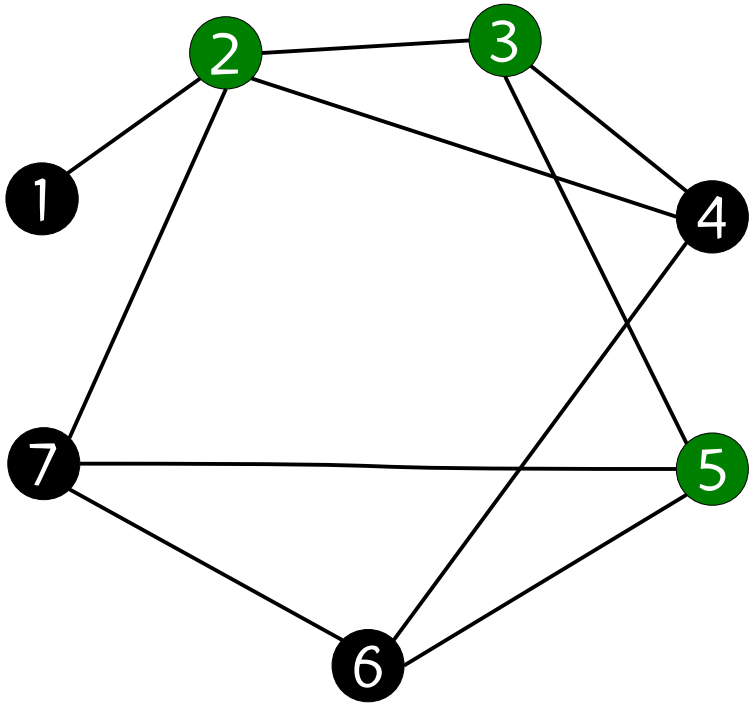
GRASP heuristics (Feo & M.G.C.R., 1989, 1995)

- GRASP constructs a solution, one regenerator at a time.
- Randomized greedy: instead of making the greedy choice, randomized greedy builds a restricted candidate set (RCL) of semi-greedy elements and selects one at random to add to the solution. A real-valued parameter $\alpha \in [0,1]$ controls the amount of randomness and greediness of the semi-greedy method. This is repeated until a solution is on hand.
- Local search: after solution is constructed, local improvement attempts to decrease the number of regenerators.

GRASP heuristics (Feo & M.G.C.R., 1989, 1995)

- We propose randomized versions of GREEDY, H1, and H2, which we call, respectively, CG, C1, and C2.
- Local search (Chen et al., 2010) attempts to remove regenerators { i,j } from R and replace them with one not currently in R.

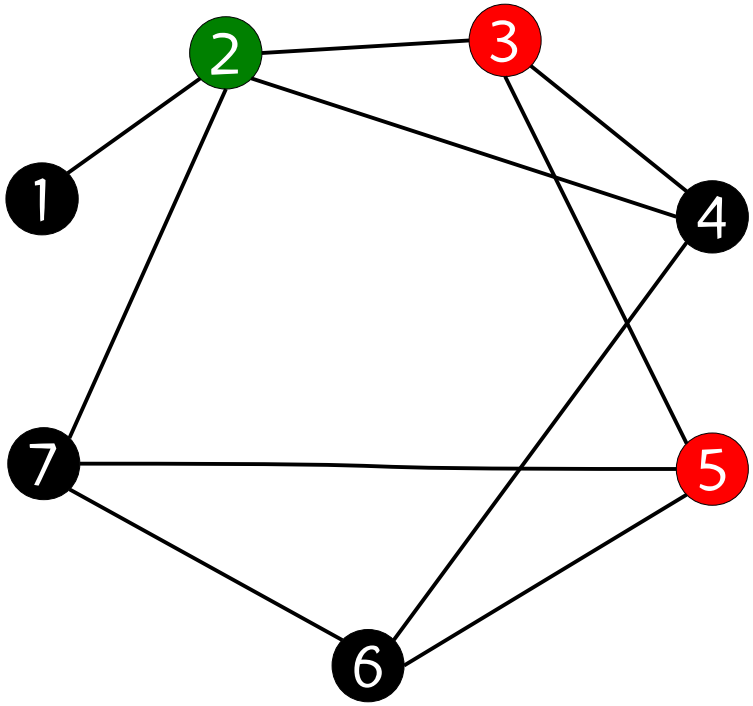
Local search



M = (V,E')

Local search

Attempt to remove {3,5}

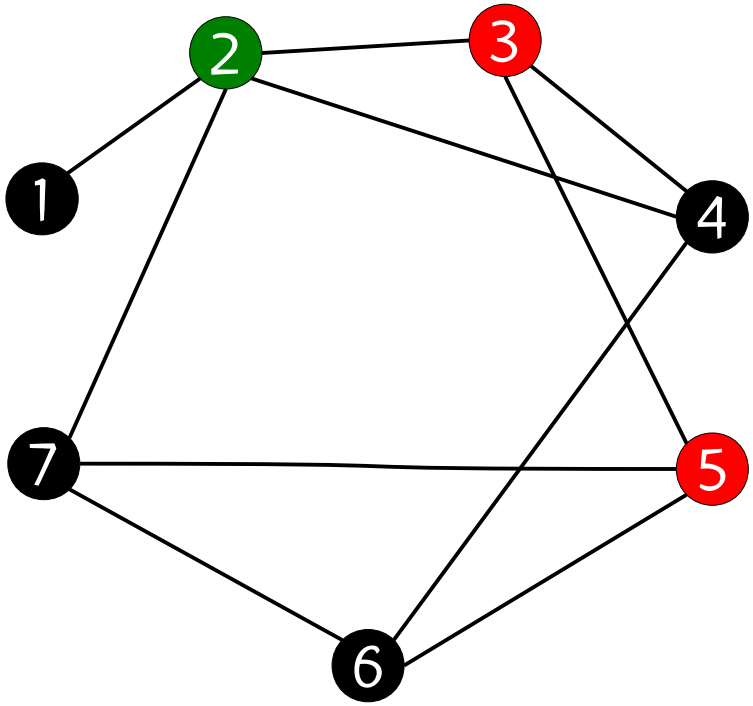


M = (V,E')

Local search

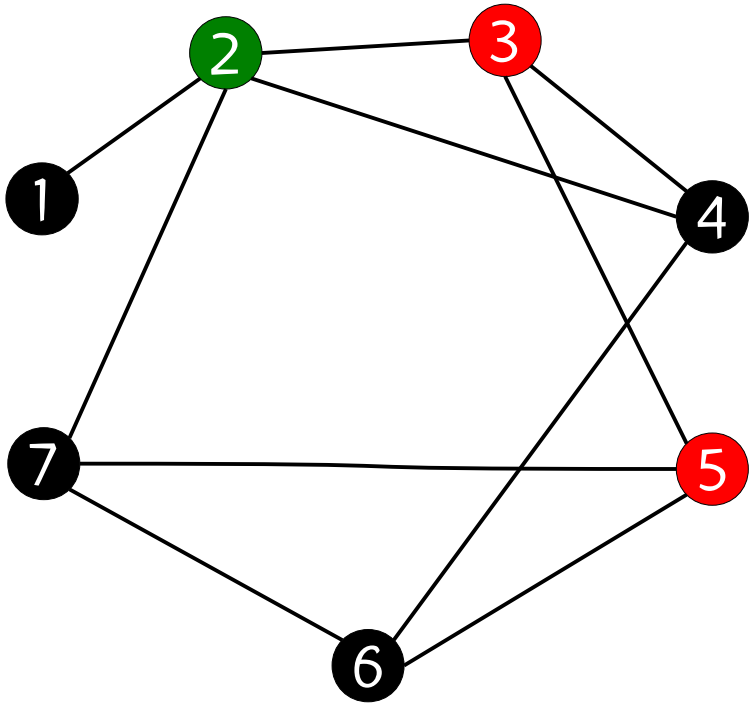
Attempt to remove {3,5}

R' = R \ { 3, 5 } = { 2 }



M = (V,E')

Local search



M = (V,E')

Attempt to remove {3,5}

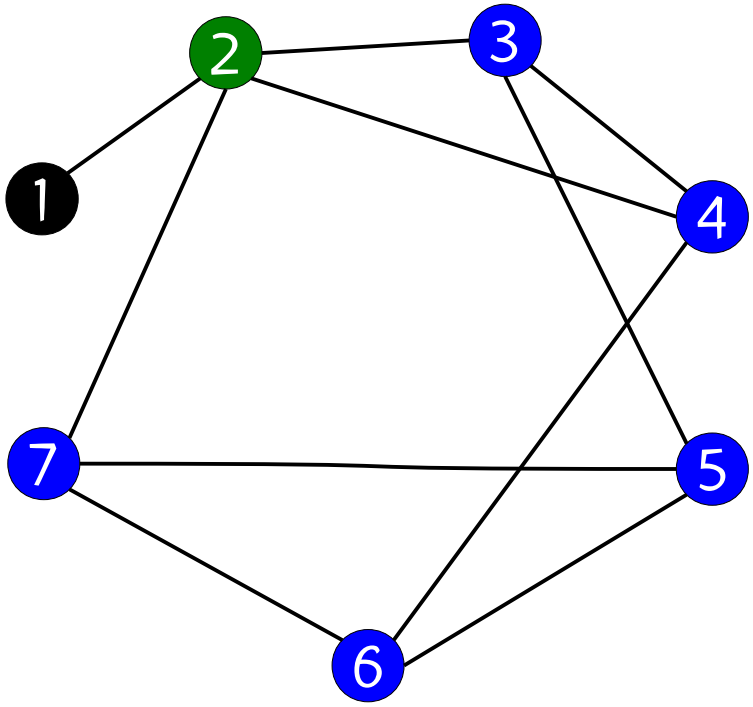
R' = R \ { 3, 5 } = { 2 }

C = { 1, 3, 4, 5, 6, 7 } candidates

Analyze non-regen neighbors of { 3, 5 }:

u = { 3, 4, 5, 6, 7} to be added to R'

Local search



M = (V,E')

Attempt to remove {3,5}

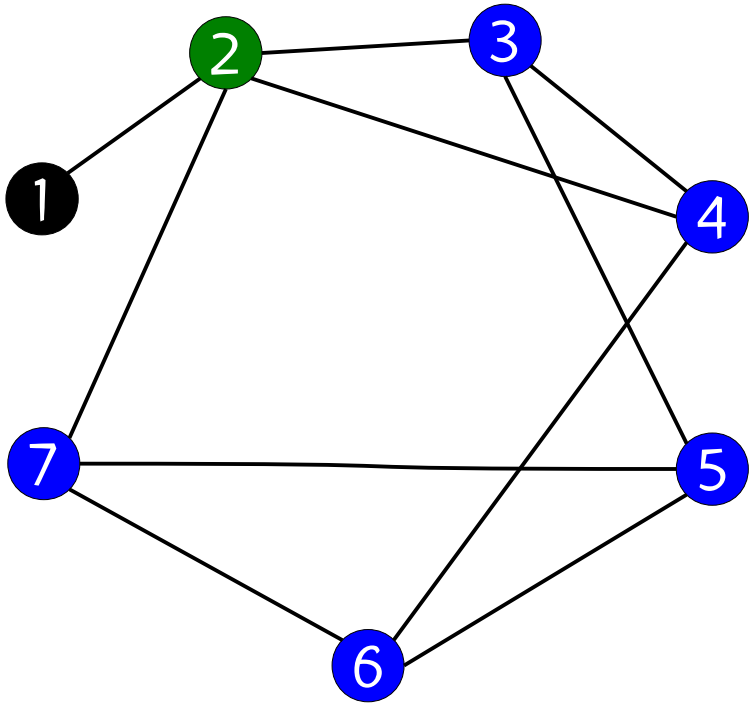
R' = R \ { 3, 5 } = { 2 }

C = { 1, 3, 4, 5, 6, 7 } candidates

Analyze non-regen neighbors of { 3, 5 }:

u = { 3, 4, 5, 6, 7} to be added to R'

Local search



M = (V,E')

Attempt to remove {3,5}

R' = R \ { 3, 5 } = { 2 }

C = { 1, 3, 4, 5, 6, 7 } candidates

Analyze non-regen neighbors of { 3, 5 }:

u = { 3, 4, 5, 6, 7} to be added to R'

For each u: if $N(u) \cap R' = \emptyset$ then

$C = C \cap N(u)$

endif

$N(3) \cap R' = \{ 2 \}$

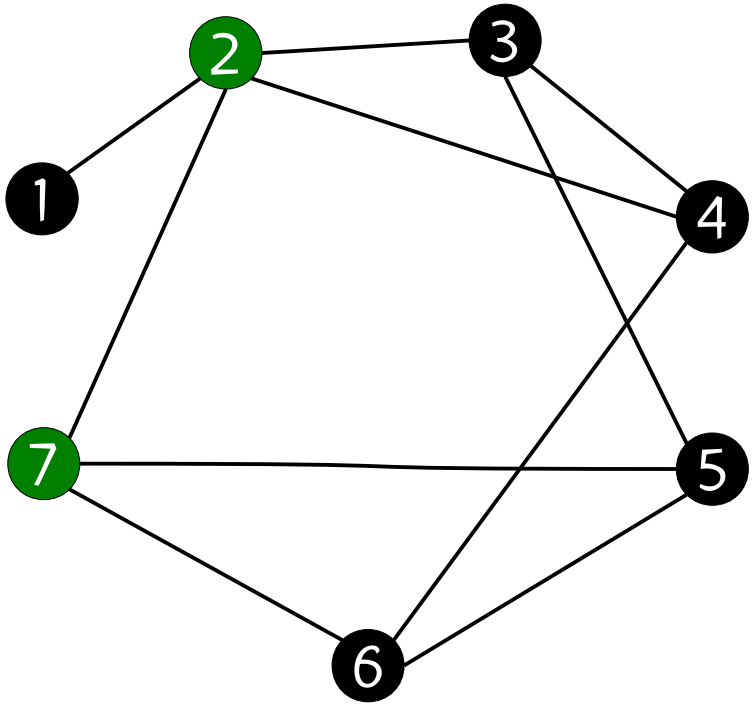
$N(4) \cap R' = \{ 2 \}$

$N(5) \cap R' = \emptyset$: $C = C \cap \{3,6,7\} = \{3,6,7\}$

$N(6) \cap R' = \emptyset$: $C = C \cap \{4,5,7\} = \{7\}$

$N(7) \cap R' = \{ 2 \}$

Local search



M = (V,E')

Attempt to remove {3,5}

R' = R \ { 3, 5 } = { 2 }

C = { 1, 3, 4, 5, 6, 7 } candidates

Analyze non-regen neighbors of { 3, 5 }:

u = { 3, 4, 5, 6, 7} to be added to R'

For each u: if $N(u) \cap R' = \varnothing$ then

$C = C \cap N(u)$

endif

C = {7}

Add { 7 } to R' = { 2, 7 }

Experimental results

Design

- We compare effectiveness & efficiency of the procedures proposed with those in Chen et al. (2010)
- Use 280 instances shared with us by Chen et al. (2010), with 40, 60, 80, and 100 nodes:
  - 200 instances are M-graphs, generated directly
  - 80 are instances in which edges are generated randomly and from which the corresponding M-graphs are computed
- All methods implemented in Java SE 6
- All experiments done on a 3 GHz Pentium 4 computer with 2 Gb of memory

Design

- In each experiment we compute
  - The overall best solution (BestValue) found for each instance by all executions of the methods considered
  - The relative percentage deviation from BestValue for each method on each instance
  - The average deviation (Dev) across all instances in each experiment
  - For each method, the number of instances (#Best) in which the BestValue solution was obtained

Design

- Tuning on a set of 20 instances randomly selected of size n = 80, 100.
- We study the value of the RCL parameter α in constructive methods *CG*, *C1*, and *C2*
- We tested three values for α: 0.3, 0.6, 0.9
- *CG*, *C1*, and *C2* were each run independently 100 times on each instance

Tuning the RCL parameter α

	Value	#Best	% dev	CPU (s)
CG				
α = 0.3	7.00	7	25.77	12935.5
α = 0.6	5.65	13	5.92	10001.0
α = 0.9	5.35	19	0.45	7958.1
C1				
α = 0.3	6.80	8	20.42	767.75
α = 0.6	6.10	10	10.79	747.25
α = 0.9	6.25	11	12.10	761.55
C2				
α = 0.3	6.30	7	16.59	573.85
α = 0.6	6.10	11	10.64	562.40
α = 0.9	5.95	12	8.51	543.65

Tuning the RCL parameter α

	Value	#Best	% dev	CPU (s)
CG				
$\alpha = 0.3$	7.00	7	25.77	12935.5
$\alpha = 0.6$	5.65	13	5.92	10001.0
$\alpha = 0.9$	5.35	19	0.45	7958.1
C1				
$\alpha = 0.3$	6.80	8	20.42	767.75
$\alpha = 0.6$	6.10	10	10.79	747.25
$\alpha = 0.9$	6.25	11	12.10	761.55
C2				
$\alpha = 0.3$	6.30	7	16.59	573.85
$\alpha = 0.6$	6.10	11	10.64	562.40
$\alpha = 0.9$	5.95	12	8.51	543.65

Clearly, the best outcomes are for method CG with RCL parameter $\alpha = 0.9$

However, CG's running times are longer than those of C1 and C2

Comparing constructive methods with local search

- On the same 20 instances from the previous experiment we compare the constructive methods with local search of Chen et al. (2010) with their GRASP counterparts
- Chen et al. methods are: Greedy+LS, H1+LS, and H2+LS
- GRASP methods are CG+LS, C1+LS, and C2+LS and are run for 50 iterations
- We use the best value of α for each method according to the previous experiment (corresponding to minimum % deviation)
- RCL parameter α was set to 0.9, 0.6, and 0.9 for CG, C1, and C2, respectively
- We report Value, #Best, %dev, and CPU time

Comparing constructive methods with local search

	Value	#Best	%dev	CPU(s)
Greedy+LS	5.65	11	8.96	176.20
H1+LS	5.70	11	9.96	48.95
H2+LS	5.55	14	7.50	14.05
GC(0.9)+LS	5.25	18	1.45	8193.05
C1(0.6)+LS	5.15	20	0.00	2238.25
C2(0.9)+LS	5.25	18	1.70	1093.55

Comparing constructive methods with local search

	Value	#Best	%dev	CPU(s)
Greedy+LS	5.65	11	8.96	176.20
H1+LS	5.70	11	9.96	48.95
H2+LS	5.55	14	7.50	14.05
GC(0.9)+LS	5.25	18	1.45	8193.05
C1(0.6)+LS	5.15	20	0.00	2238.25
C2(0.9)+LS	5.25	18	1.70	1093.55

Three new GRASP methods improve upon previous methods based on construction with local search

Since GRASP is a multi-start method and previous methods are deterministic and run only once, GRASP running times are higher

Comparing best GRASP with best method of Chen et al. (2010) and BRKGA

- We now compare C1+LS with H2+LS and the BRKGA
- GRASP was run for 100 iterations and the BRKGA was run for 100 generations with a population of size 100
- We now report Value, #Best, %dev, and CPU time for the entire set of 280 test instances (separated by problem size)

C1+LS, H2+LS, and BRKGA on all 70 n = 40 instances

	Value	#Best	%dev	CPU (s)
H2+LS	3.96	59	4.46	3.61
C1(0.6)+LS	3.77	70	0.00	253.69
BRKGA	3.83	66	1.71	1445.57

C1+LS, H2+LS, and BRKGA on all 70 n = 40 instances

	Value	#Best	%dev	CPU (s)
H2+LS	3.96	59	4.46	3.61
C1(0.6)+LS	3.77	70	0.00	253.69
BRKGA	3.83	66	1.71	1445.57

GRASP was best in terms of solution quality

GRASP running times longer than deterministic method

BRKGA beat H2+LS but running times were the longest

C1+LS, H2+LS, and BRKGA on all 70 n = 60 instances

	Value	#Best	%dev	CPU (s)
H2+LS	4.37	62	1.74	6.43
C1(0.6)+LS	4.26	70	0.00	646.46
BRKGA	4.34	64	1.48	4462.64

C1+LS, H2+LS, and BRKGA on all 70 n = 60 instances

	Value	#Best	%dev	CPU (s)
H2+LS	4.37	62	1.74	6.43
C1(0.6)+LS	4.26	70	0.00	646.46
BRKGA	4.34	64	1.48	4462.64

GRASP was best in terms of solution quality

GRASP running times longer than deterministic method

BRKGA beat H2+LS but running times were the longest

C1+LS, H2+LS, and BRKGA on all 70 n = 80 instances

	Value	#Best	%dev	CPU (s)
H2+LS	4.90	46	8.25	10.26
C1(0.6)+LS	4.50	70	0.00	1356.20
BRKGA	4.67	58	4.39	9742.37

C1+LS, H2+LS, and BRKGA on all 70 n = 80 instances

	Value	#Best	%dev	CPU (s)
H2+LS	4.90	46	8.25	10.26
C1(0.6)+LS	4.50	70	0.00	1356.20
BRKGA	4.67	58	4.39	9742.37

GRASP was best in terms of solution quality

GRASP running times longer than deterministic method

BRKGA beat H2+LS but running times were the longest

C1+LS, H2+LS, and BRKGA on all 70 n = 100 instances

	Value	#Best	%dev	CPU (s)
H2+LS	5.27	50	5.69	16.20
C1(0.6)+LS	4.91	70	0.00	2393.73
BRKGA	5.00	62	2.13	20169.06

C1+LS, H2+LS, and BRKGA on all 70 n = 100 instances

	Value	#Best	%dev	CPU (s)
H2+LS	5.27	50	5.69	16.20
C1(0.6)+LS	4.91	70	0.00	2393.73
BRKGA	5.00	62	2.13	20169.06

GRASP was best in terms of solution quality

GRASP running times longer than deterministic method

BRKGA beat H2+LS but running times were the longest

Concluding remarks

Concluding remarks

- We introduce several new randomized heuristics for the regenerator location problem
- GRASP heuristics are based on the three greedy algorithms of Chen et al. (2010)
- BRKGA uses a decoder based on the greedy algorithm GREEDY of Chen et al. (2010)

Concluding remarks

- Experiments show that our heuristic C1+LS with RCL parameter $\alpha = 0.6$ consistently produces the best solutions with smaller %dev and larger #Best values than the other heuristics
- Deterministic heuristic H2+LS of Chen et al. (2010) is able to obtain relatively good solutions in short computational time

My coauthors

Holmdel, New Jersey
September 2009

The End

Slides of this talk as well as all papers cited in the talk can be downloaded from my homepage:

http://www2.research.att.com/~mgcr