

Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm

ALIO INFORMS Joint International Meeting
Buenos Aires, Argentina * June 6-9, 2010



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

Summary

- Biased random-key genetic algorithms (BRKGA)
- GRASP with path-relinking for the generalized quadratic assignment problem
- Two-phase hybrid heuristic
 - Tuning phase
 - Solution phase
- Computational results
- Concluding remarks

Reference

P. Festa, J.F. Gonçalves, M.G.C.R., and R.M.A. Silva, "Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm," SEA 2010, LNCS, Springer, 2010

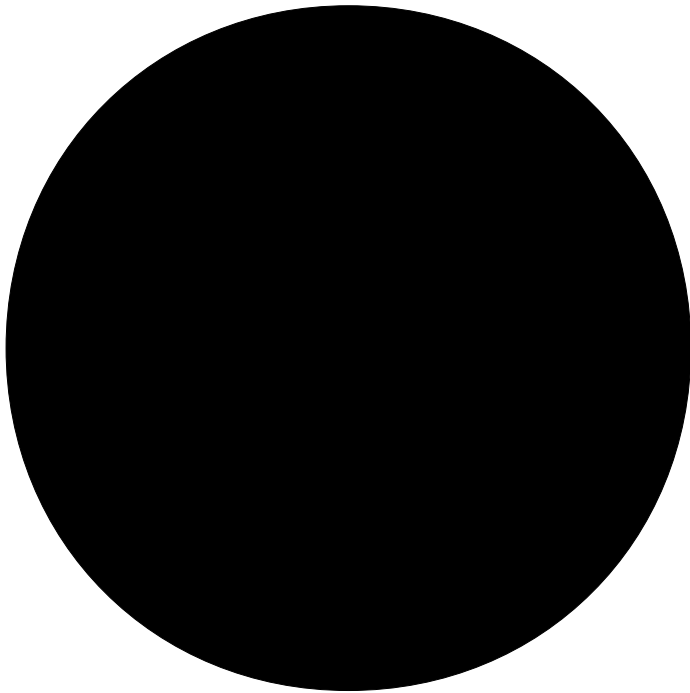
<http://www.research.att.com/~mgcr/doc/brkga-gp-gqap.pdf>

Biased random-key genetic algorithms

Genetic algorithms

Holland (1975)

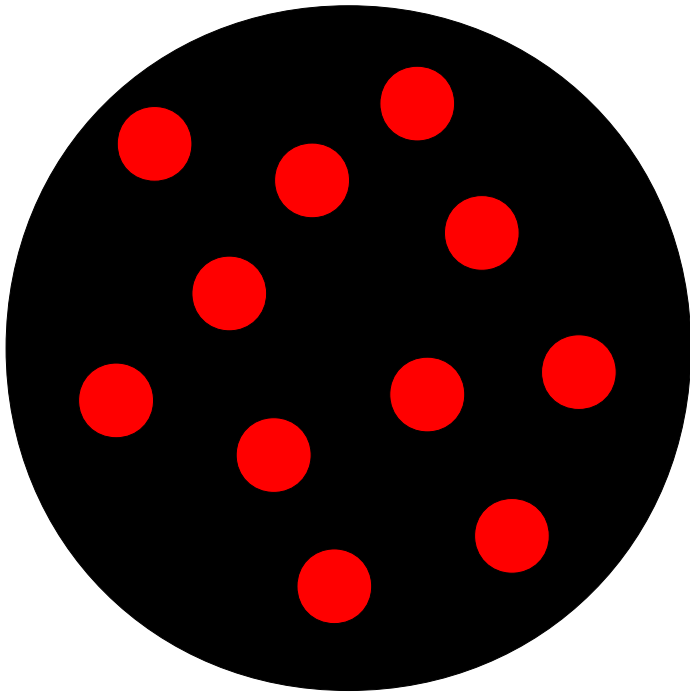
Adaptive methods that are used to solve search and optimization problems.



Individual: solution



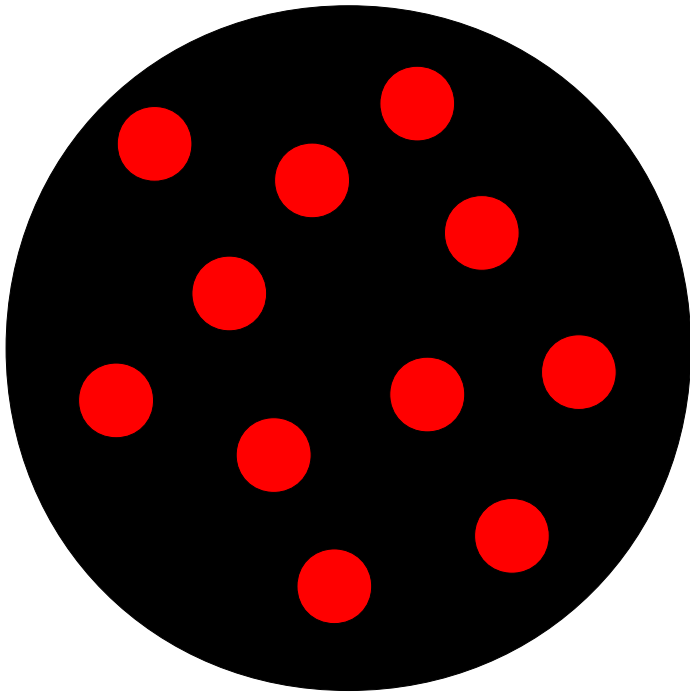
Genetic algorithms



Individual: solution

Population: set of fixed number of individuals

Genetic algorithms

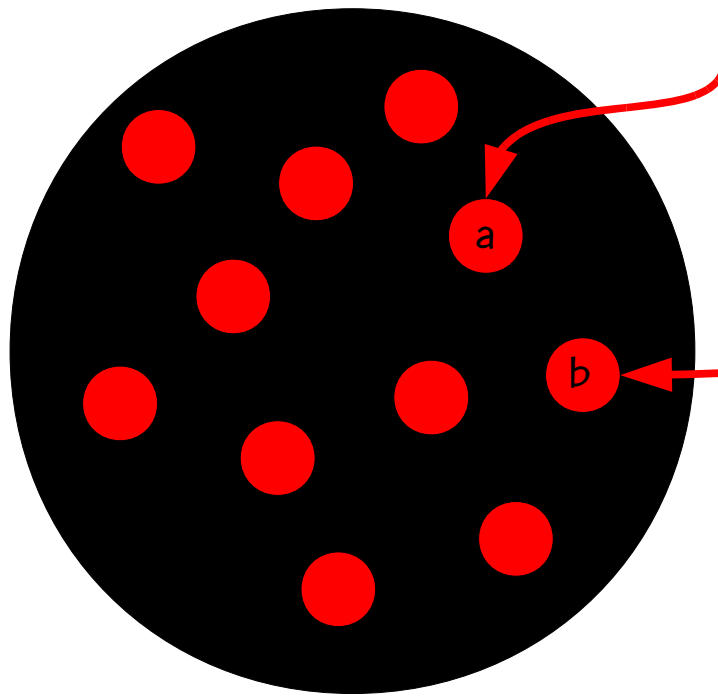


Genetic algorithms evolve population applying the principle of survival of the fittest.

A series of generations are produced by the algorithm. The most fit individual of last generation is the solution.

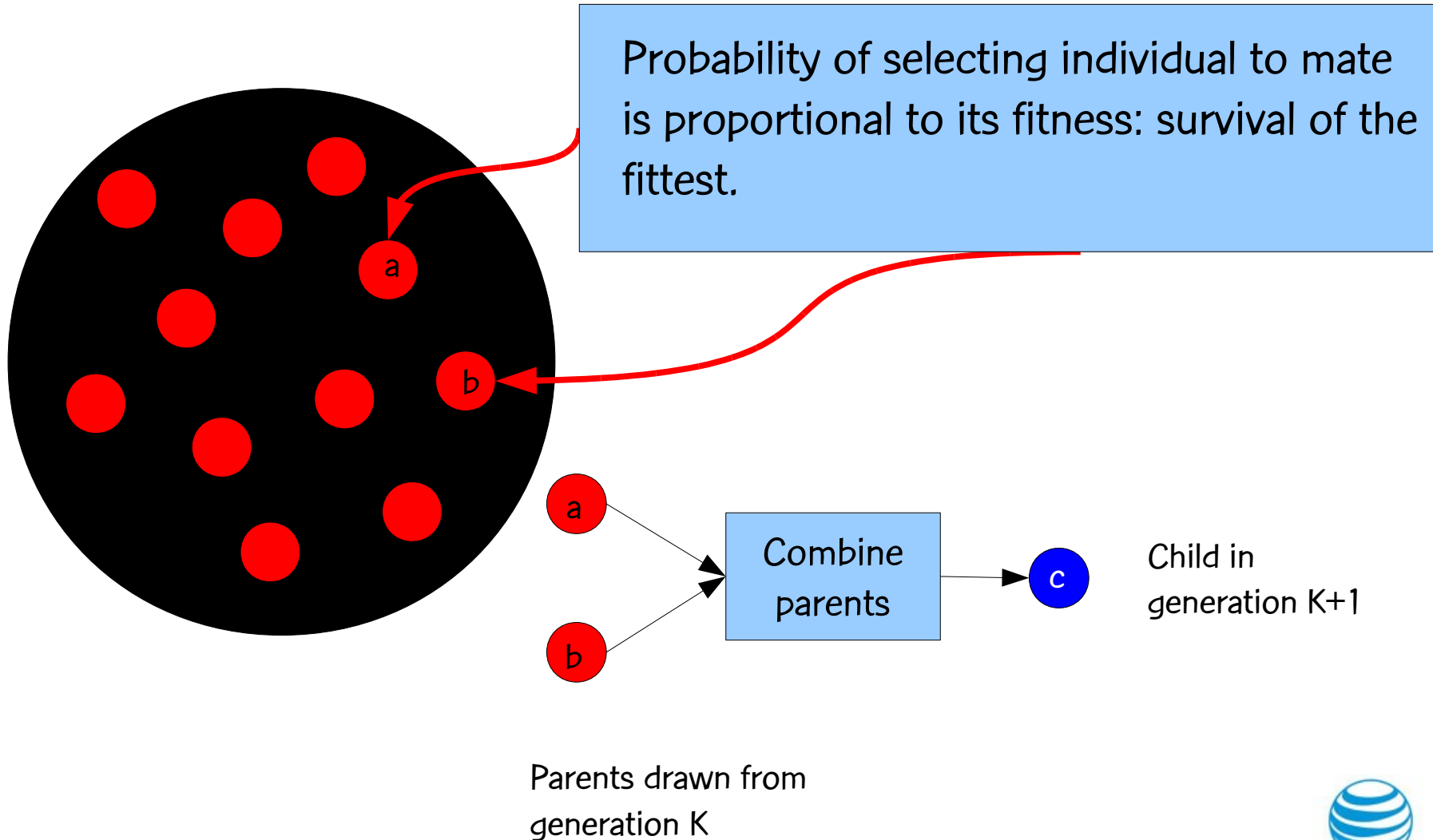
Individuals from one generation are combined to produce offspring that make up next generation.

Genetic algorithms

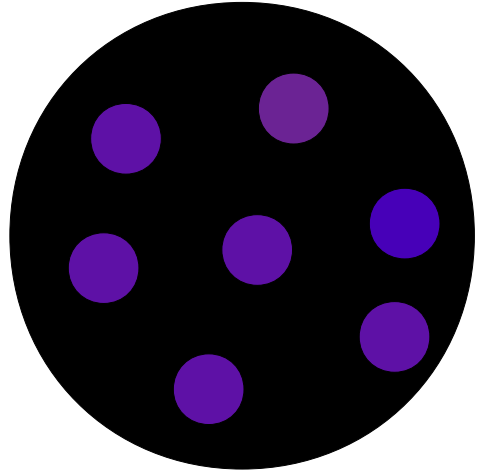


Probability of selecting individual to mate is proportional to its fitness: survival of the fittest.

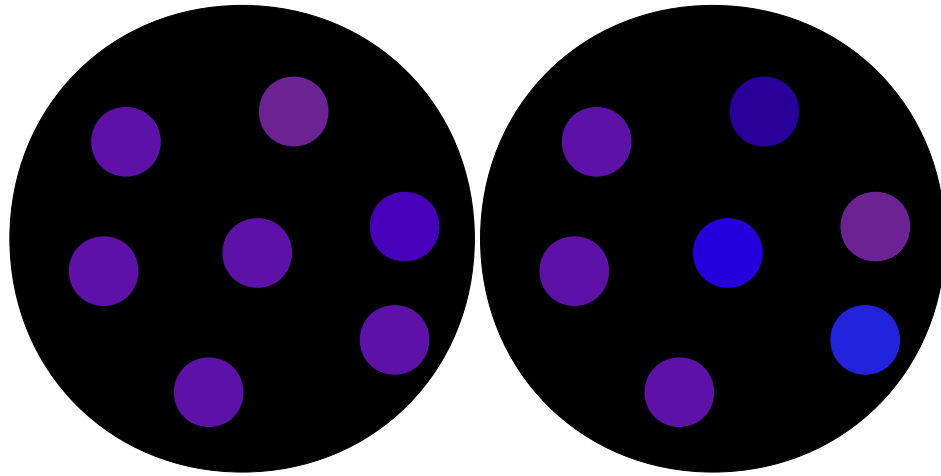
Genetic algorithms



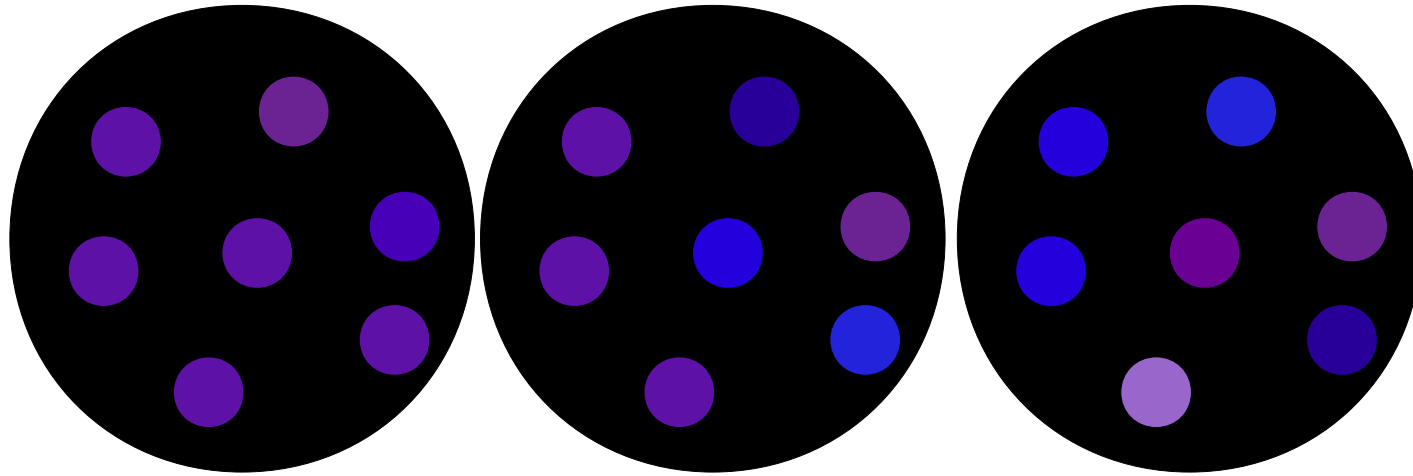
Evolution of solutions



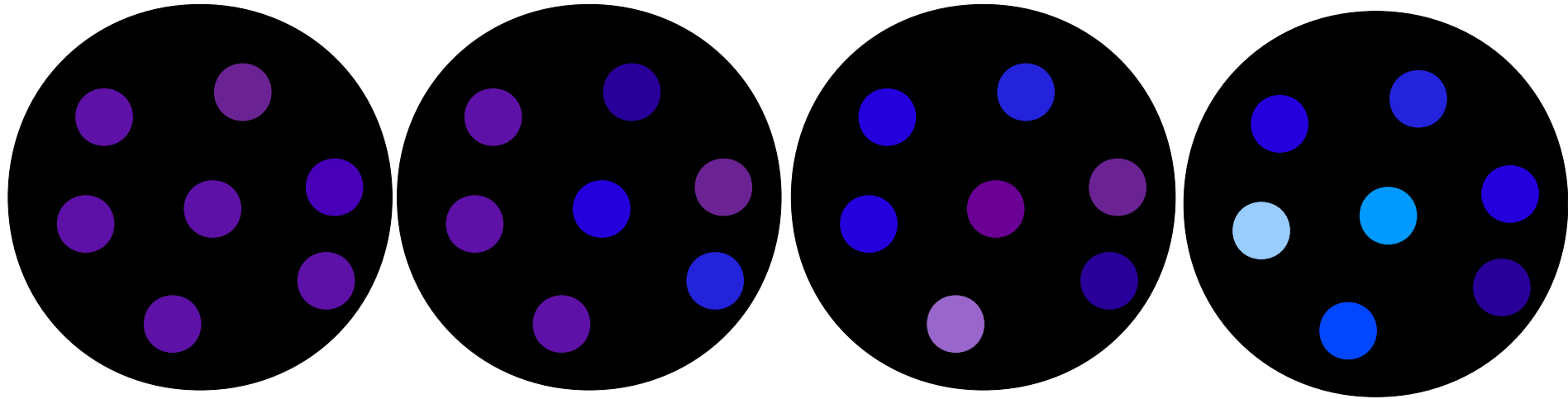
Evolution of solutions



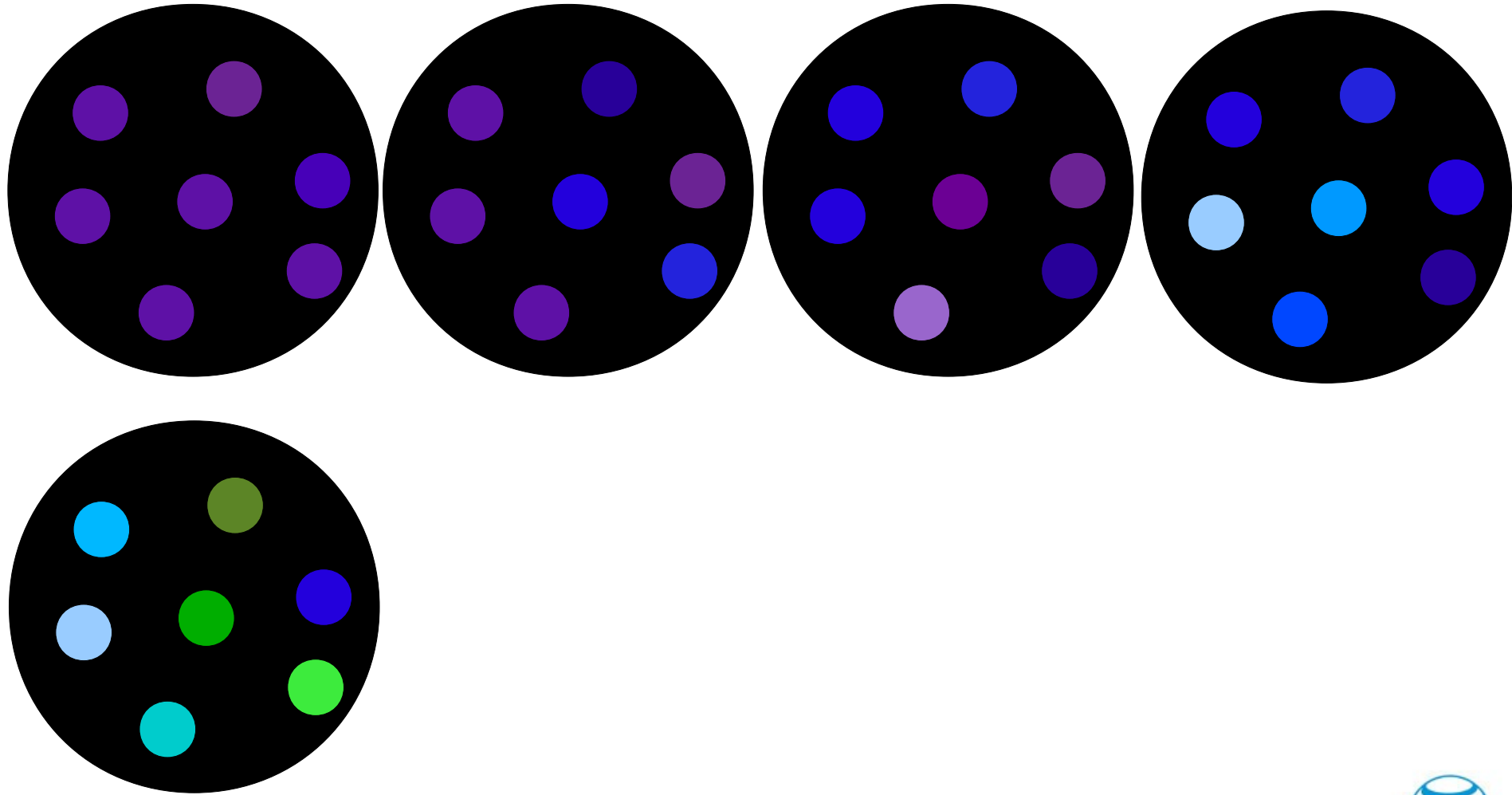
Evolution of solutions



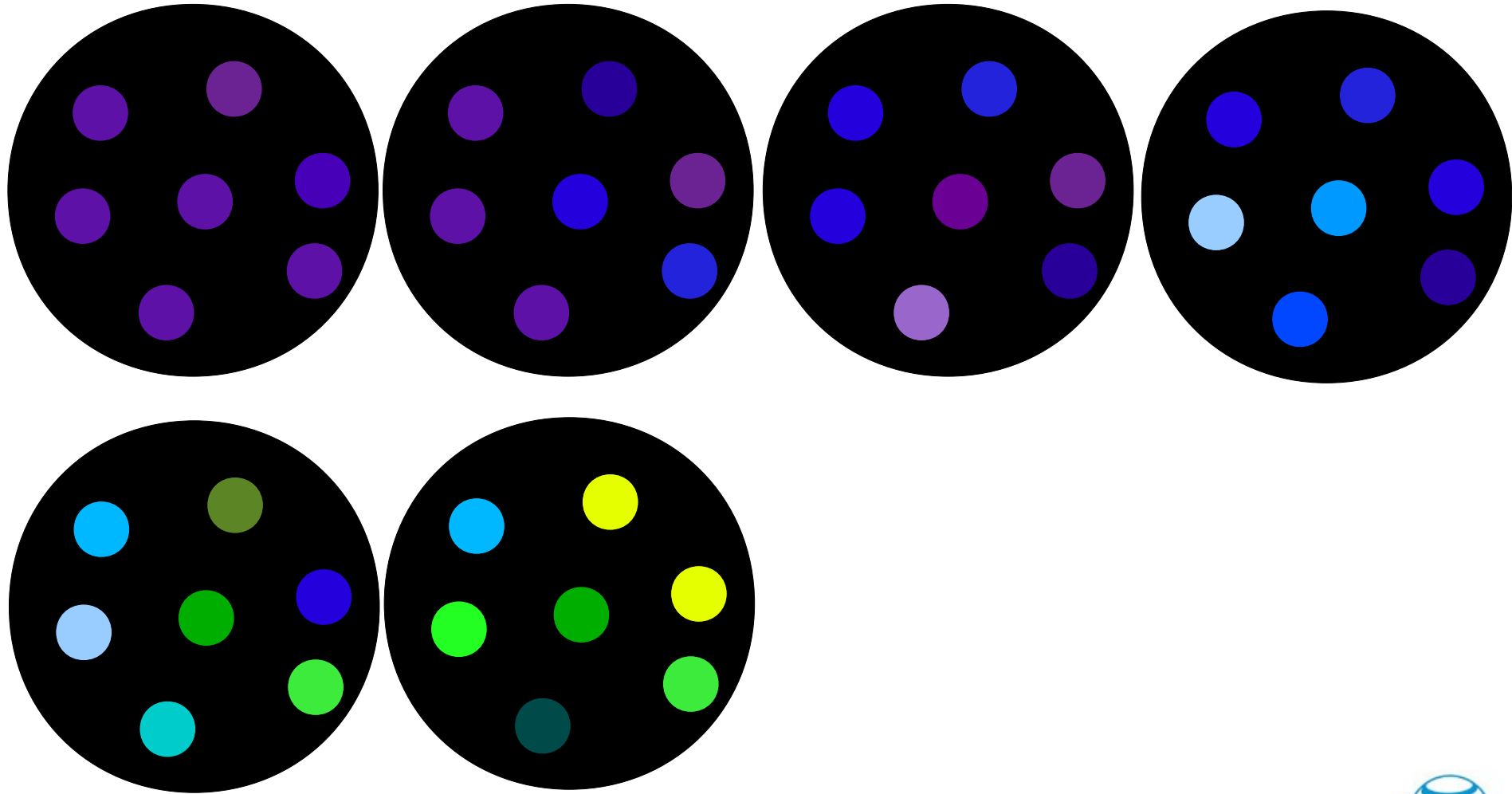
Evolution of solutions



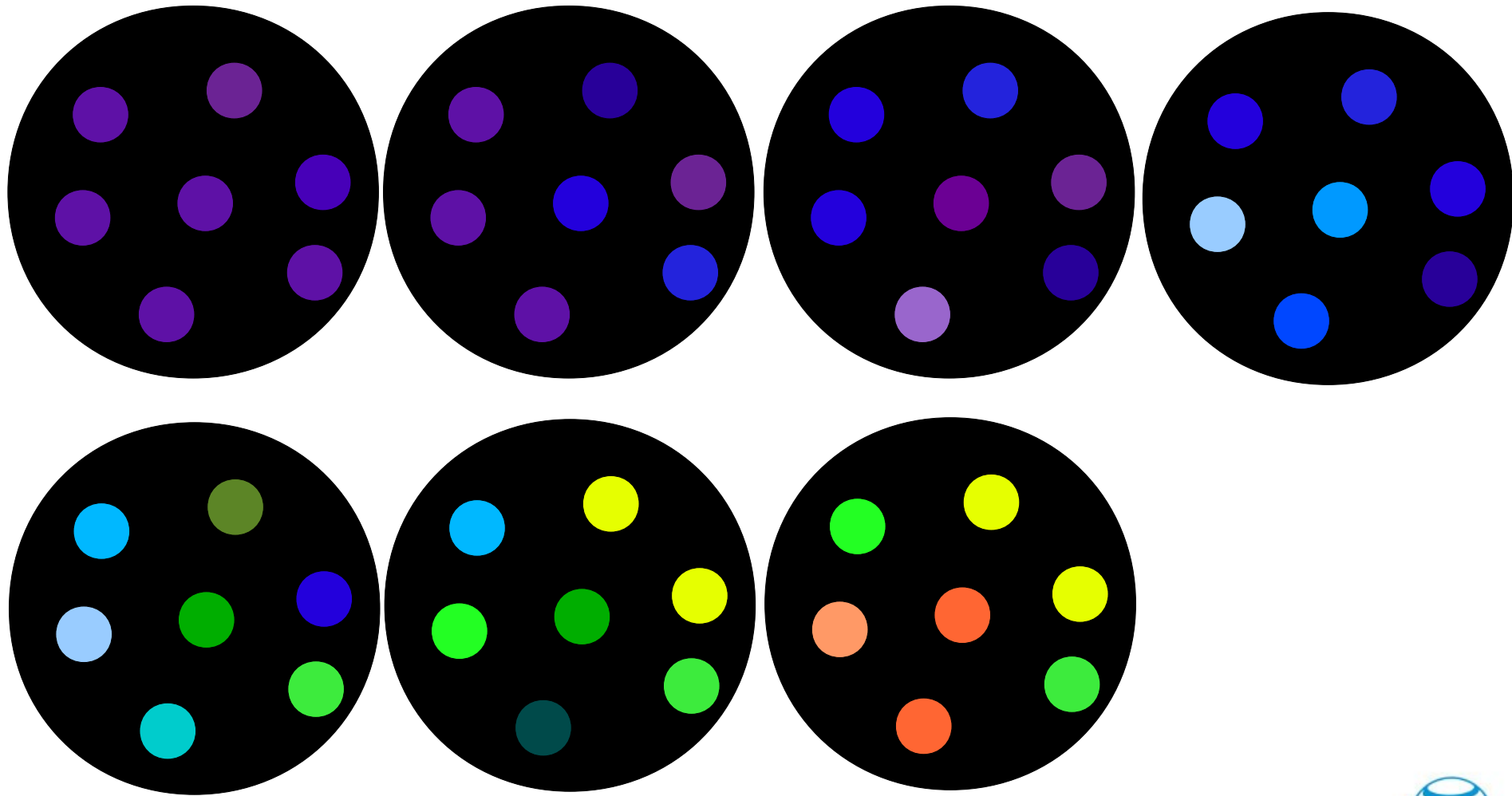
Evolution of solutions



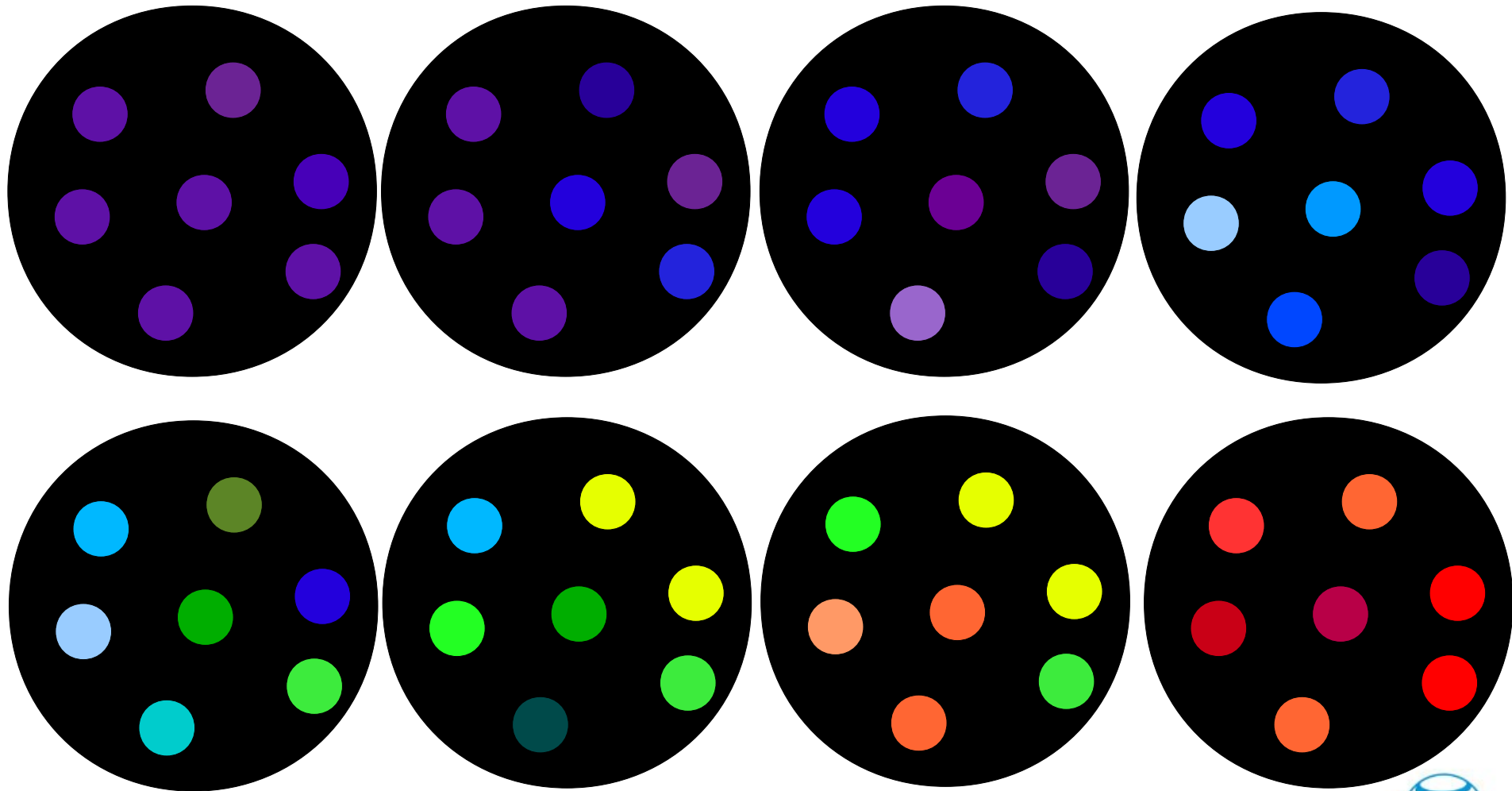
Evolution of solutions



Evolution of solutions



Evolution of solutions



Genetic algorithms with random keys

GAs and random keys

Introduced by Bean (1994)
for sequencing problems.

GAs and random keys

Introduced by Bean (1994)
for sequencing problems.

Individuals are strings of real-valued numbers (random keys) in the interval $[0,1]$.

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

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

GAs and random keys

Introduced by Bean (1994)
for sequencing problems.

Individuals are strings of real-valued numbers (random keys) in the interval $[0,1]$.

Sorting random keys results in a sequencing order.

$$S = (\begin{matrix} 0.25 & 0.19 & 0.67 & 0.05 & 0.89 \end{matrix}) \\ \begin{matrix} s(1) & s(2) & s(3) & s(4) & s(5) \end{matrix}$$

$$S' = (\begin{matrix} 0.05 & 0.19 & 0.25 & 0.67 & 0.89 \end{matrix}) \\ \begin{matrix} s(4) & s(2) & s(1) & s(3) & s(5) \end{matrix}$$

Sequence: 4 – 2 – 1 – 3 – 5

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

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

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

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

$c = (\quad \quad \quad \quad \quad)$

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

$$\begin{aligned} a &= (0.25, 0.19, 0.67, 0.05, 0.89) \\ b &= (0.63, 0.90, 0.76, 0.93, 0.08) \\ c &= (0.25, 0.90, 0.76 \end{aligned}$$

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

$$\begin{aligned} a &= (0.25, 0.19, 0.67, 0.05, 0.89) \\ b &= (0.63, 0.90, 0.76, 0.93, 0.08) \\ c &= (0.25, 0.90, 0.76, 0.05 \quad \quad) \end{aligned}$$

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

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

$c = (0.25, 0.90, 0.76, 0.05, 0.89)$

GAs and random keys

Mating is done using
parametrized uniform
crossover (Spears & DeJong , 1990)

For each gene, flip a biased
coin to choose which parent
passes the allele to the child.

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

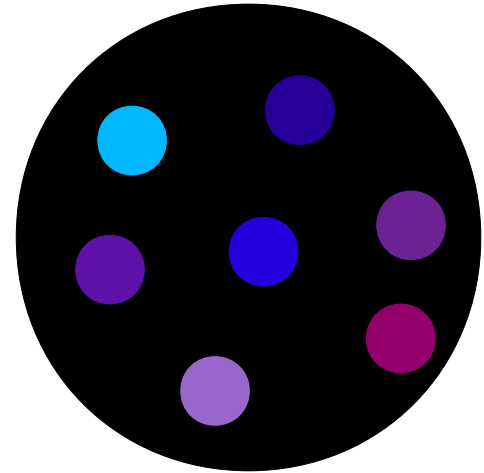
$b = (0.63, 0.90, 0.76, 0.93, 0.08)$

$c = (0.25, 0.90, 0.76, 0.05, 0.89)$

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

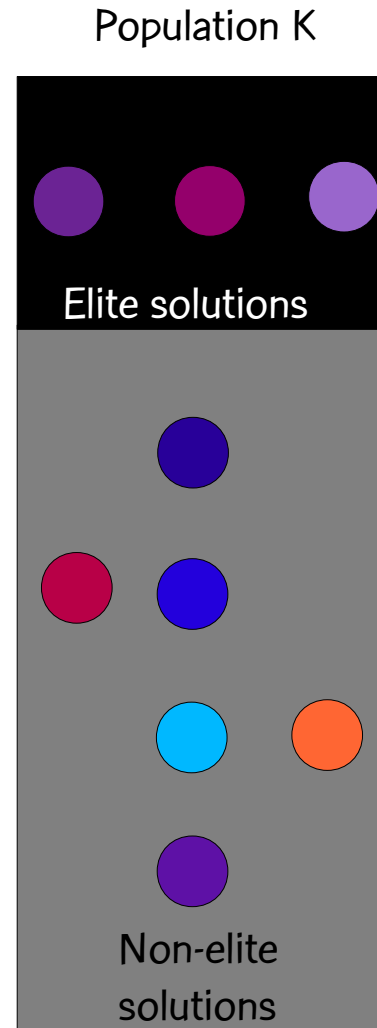
GAs and random keys

Initial population is made up of P chromosomes, each with N genes, each having a value (allele) generated uniformly at random in the interval $[0,1]$.



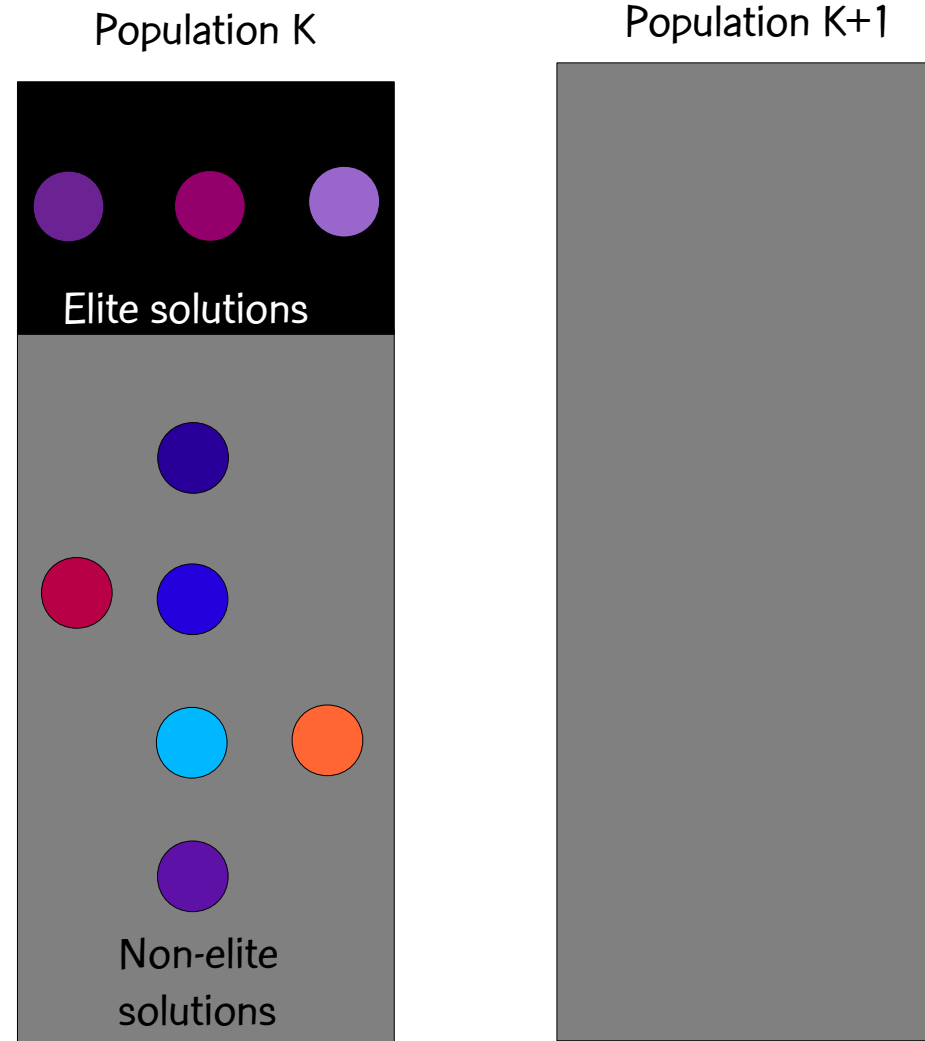
GAs and random keys

At the K-th generation, compute the cost of each solution and partition the solutions into two sets: elite solutions, non-elite solutions. Elite set should be smaller of the two sets and contain best solutions.



GAs and random keys

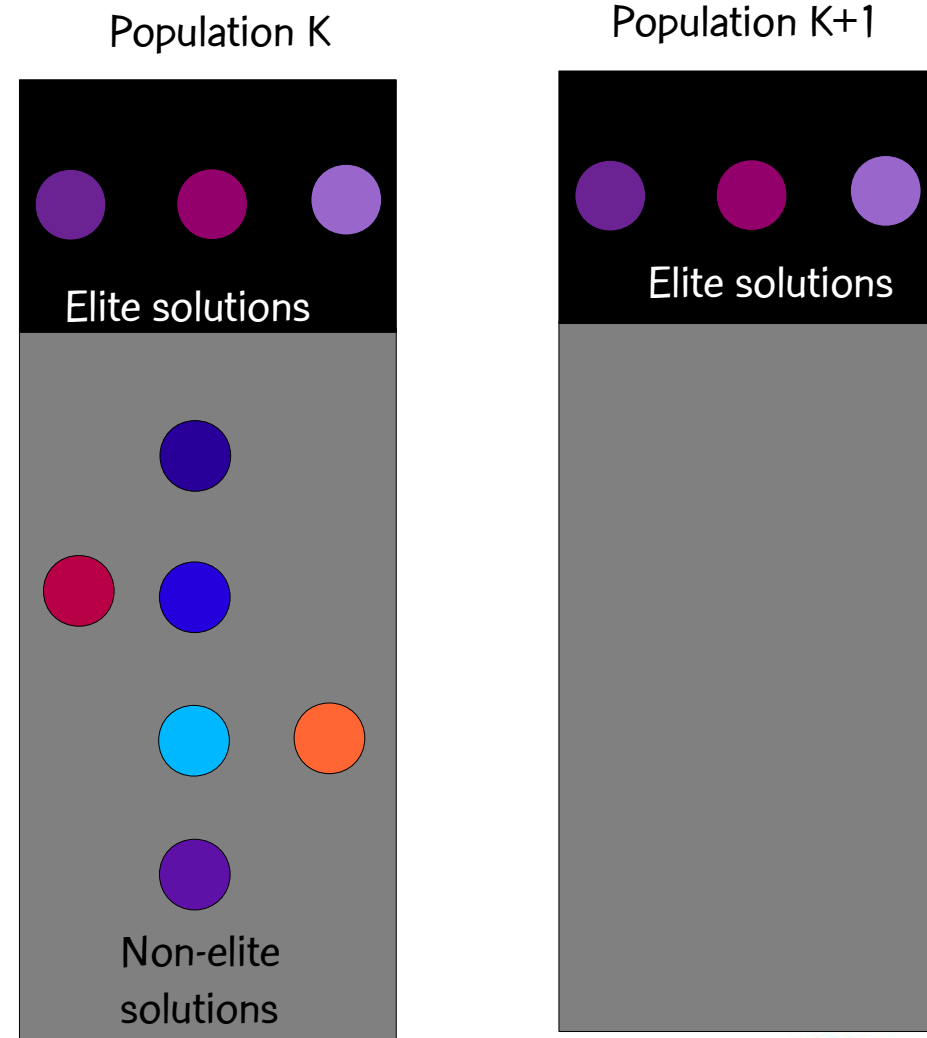
Evolutionary dynamics



GAs and random keys

Evolutionary dynamics

Copy elite solutions from population K to population K+1

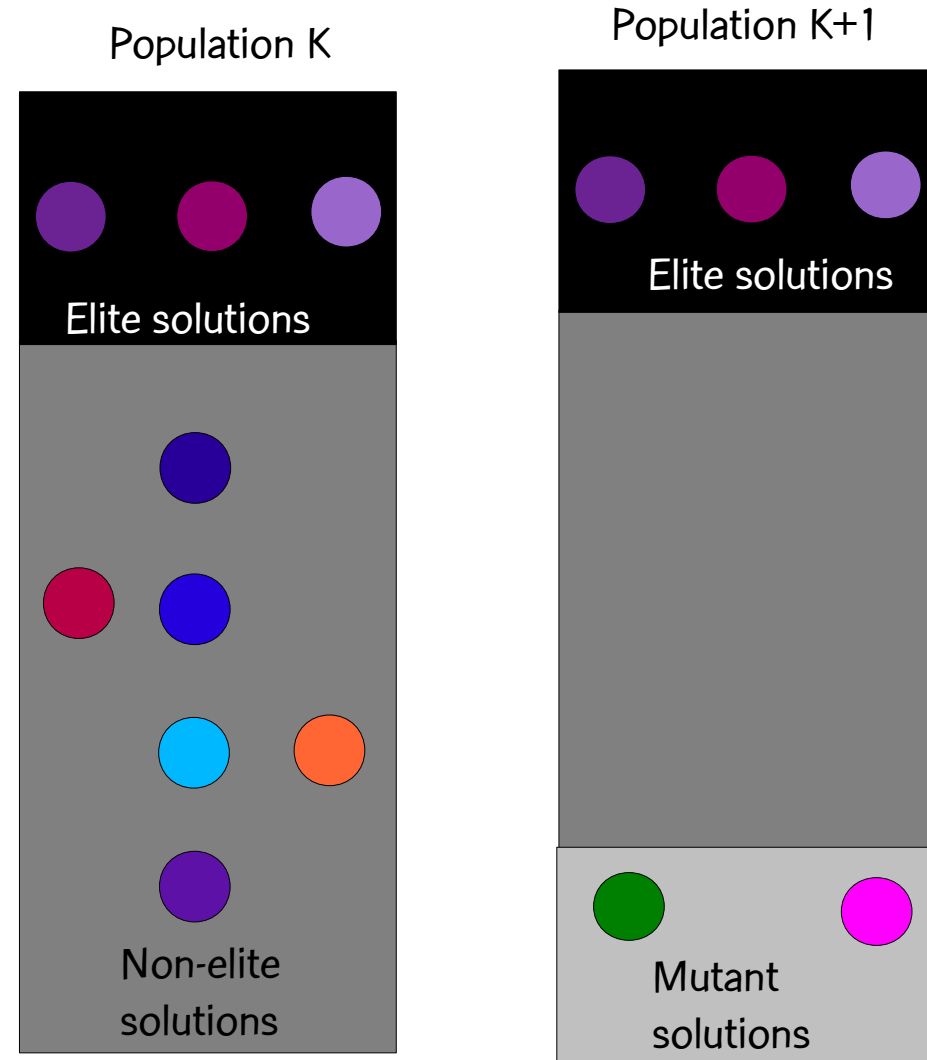


GAs and random keys

Evolutionary dynamics

Copy elite solutions from population K to population K+1

Add R random solutions (mutants) to population K+1



Biased random key GA

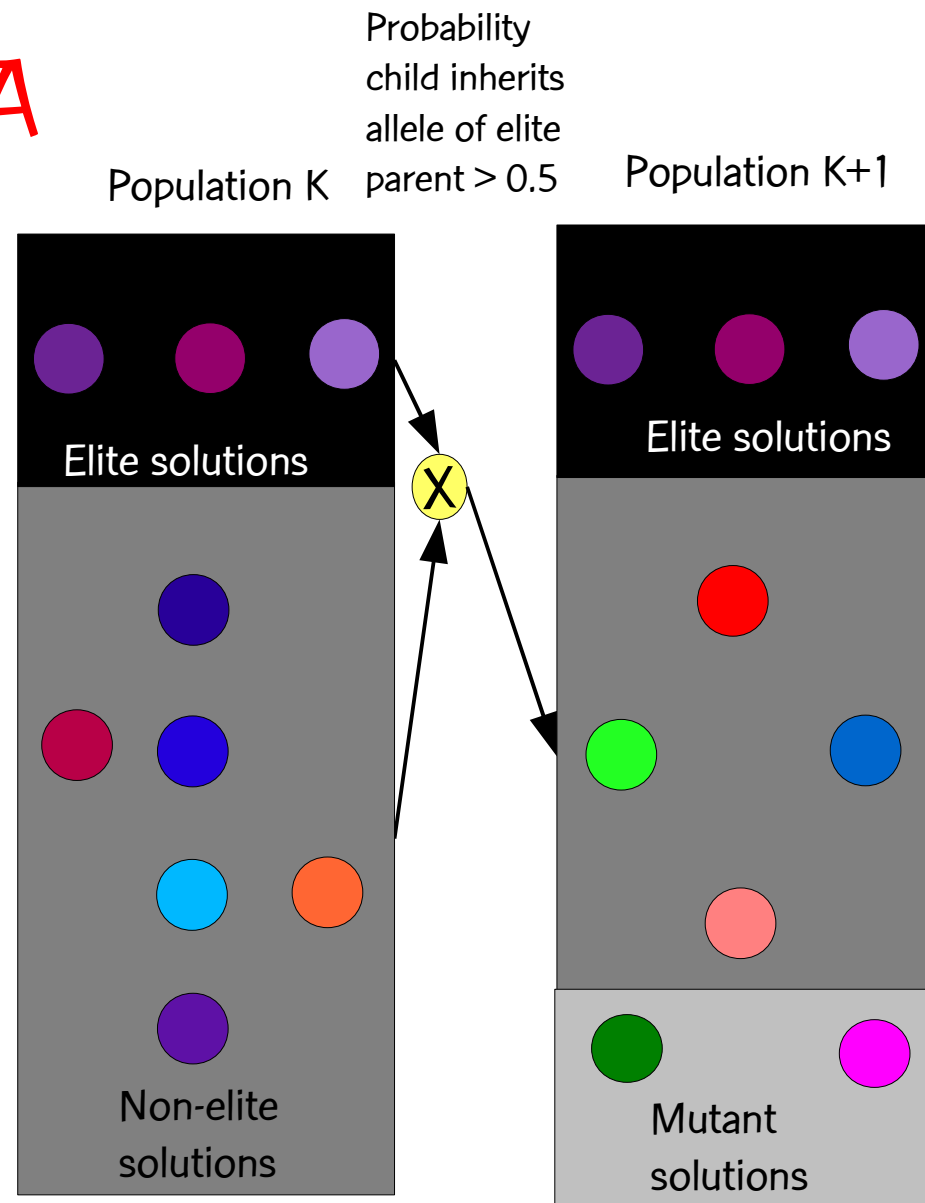
Evolutionary dynamics

Copy elite solutions from population K to population K+1

Add R random solutions (mutants) to population K+1

While K+1-th population $< P$

BIASED RANDOM KEY GA: Mate elite solution with non elite to produce child in population K+1. Mates are chosen at random.



Observations

Random method: keys are randomly generated so solutions are always random vectors

Elitist strategy: best solutions are passed without change from one generation to the next

Child inherits more characteristics of elite parent: one parent is always selected (with replacement) from the small elite set and probability that child inherits key of elite parent > 0.5

No mutation in crossover: mutants are used instead

Random-keys vs biased random-keys

How do random-key GAs (Bean, 1994) and biased random-key GAs differ?

A random-key GA selects both parents at random **from the entire population** for crossover: some pairs may not have any elite solution

A biased random-key GA always has an **elite parent** during crossover

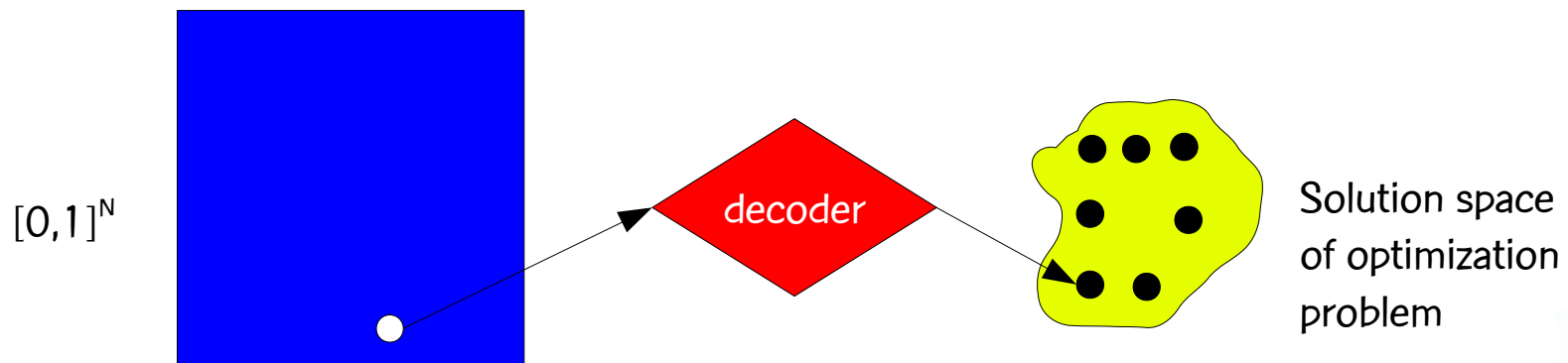
Parametrized uniform makes it **more likely that child inherits characteristics of elite parent** in biased random-key GA while it does not in random-key GA (survival of the fittest)

Decoders

A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.

Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.

A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.

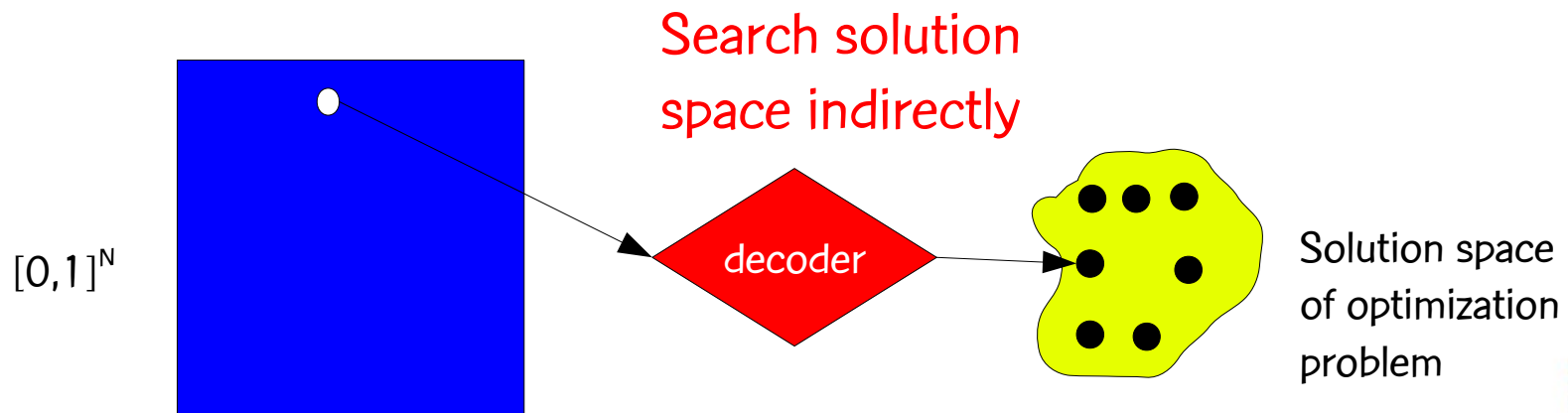


Decoders

A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.

Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.

A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.

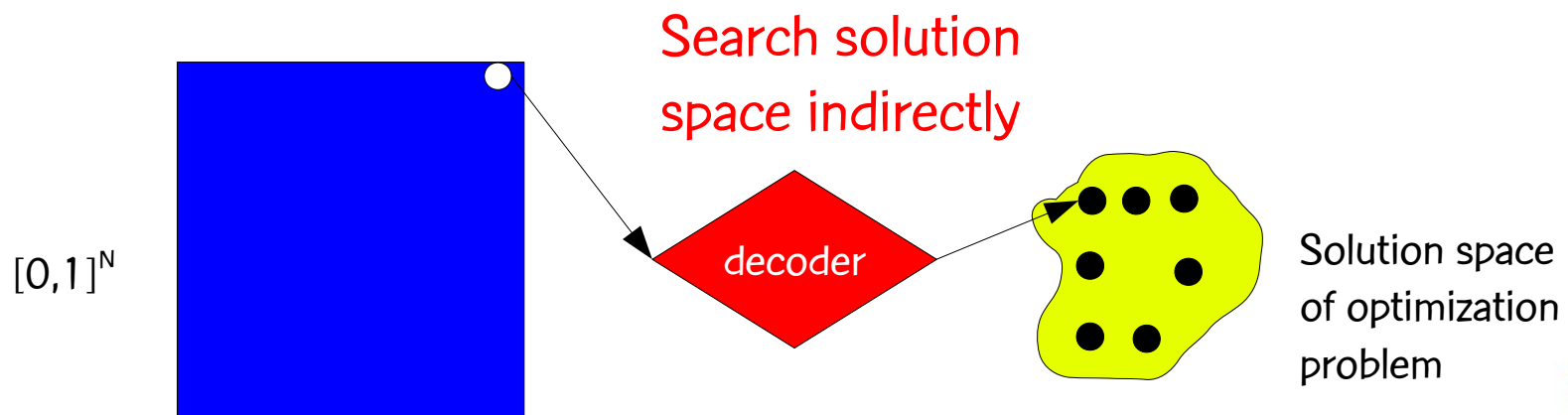


Decoders

A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.

Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.

A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.

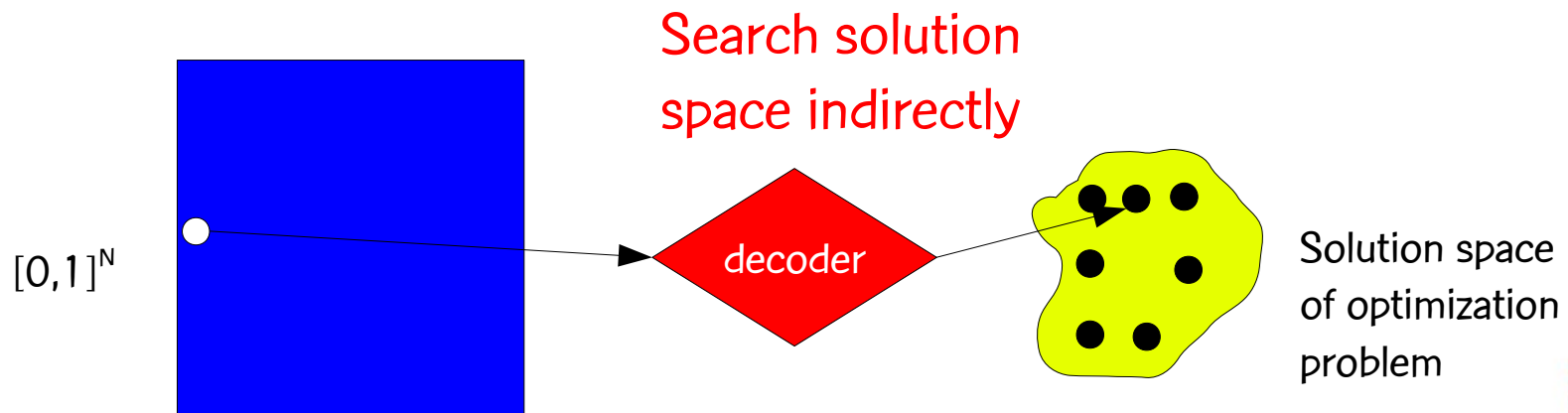


Decoders

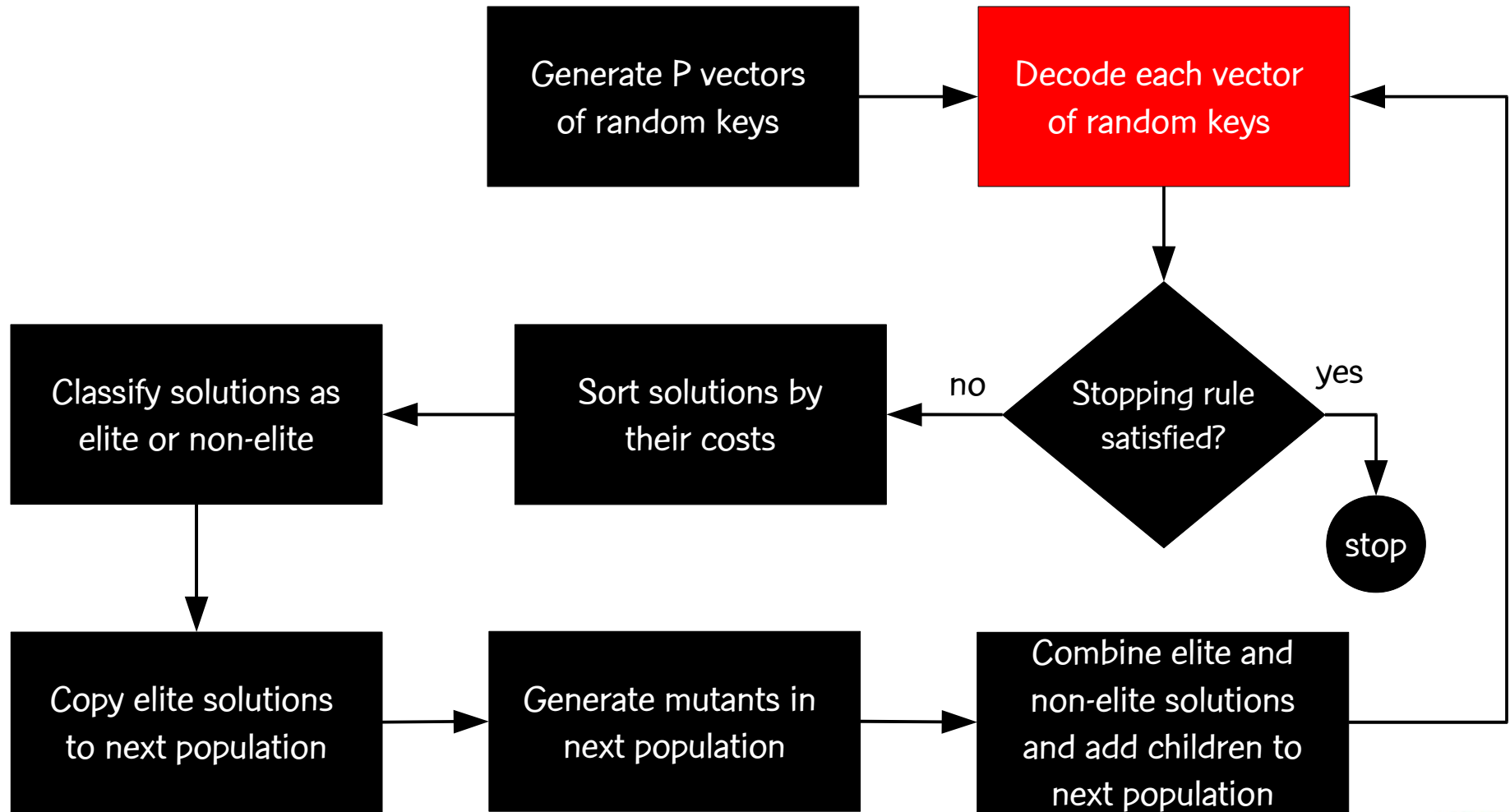
A decoder is a deterministic algorithm that takes as input a random-key vector and returns a feasible solution of the optimization problem and its cost.

Bean (1994) proposed decoders based on sorting the random-key vector to produce a sequence.

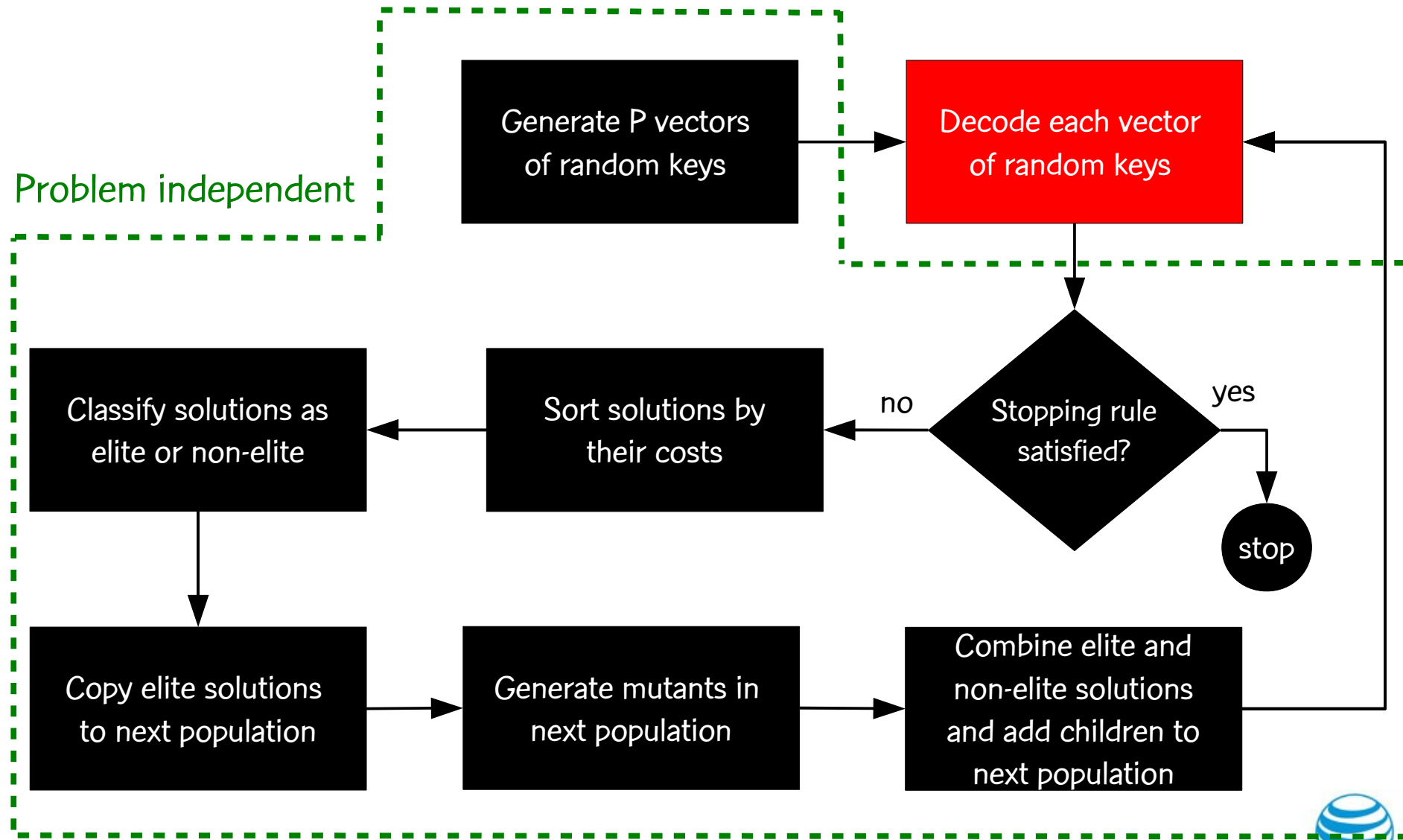
A random-key GA searches the solution space indirectly by searching the space of random keys and using the decoder to evaluate fitness of the random key.



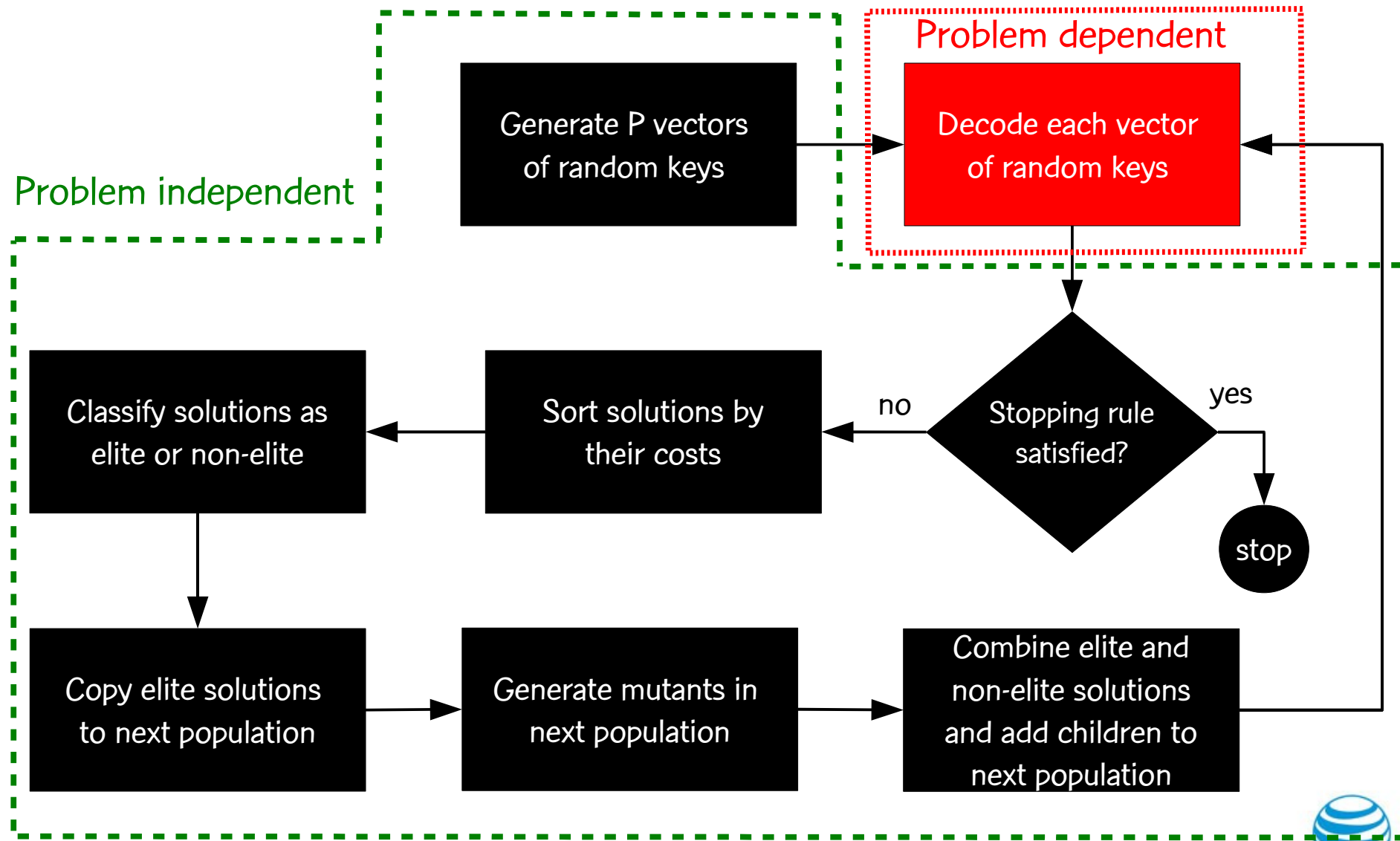
Framework for biased random-key genetic algorithms



Framework for biased random-key genetic algorithms



Framework for biased random-key genetic algorithms



Specifying a biased random-key GA

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

- Size of population

- Size of elite partition

- Size of mutant set

- Child inheritance probability

- Stopping criterion

Specifying a biased random-key algorithm

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

- Size of population: a function of N , say N or $2N$

- Size of elite partition

- Size of mutant set

- Child inheritance probability

- Stopping criterion

Specifying a biased random-key algorithm

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

Size of population: a function of N , say N or $2N$

Size of elite partition: 15-30% of population

Size of mutant set

Child inheritance probability

Stopping criterion

Specifying a biased random-key algorithm

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

Size of population: a function of N , say N or $2N$

Size of elite partition: 15-30% of population

Size of mutant set: 5-20% of population

Child inheritance probability

Stopping criterion

Specifying a biased random-key algorithm

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

Size of population: a function of N , say N or $2N$

Size of elite partition: 15-30% of population

Size of mutant set: 5-20% of population

Child inheritance probability: > 0.5 , say 0.7

Stopping criterion

Specifying a biased random-key algorithm

Encoding is always done the same way, i.e. with a vector of N random-keys (parameter N must be specified)

Decoder that takes as input a vector of N random-keys and outputs the corresponding solution of the combinatorial optimization problem and its cost (this is usually a heuristic)

Parameters:

Size of population: a function of N , say N or $2N$

Size of elite partition: 15-30% of population

Size of mutant set: 5-20% of population

Child inheritance probability: > 0.5 , say 0.7

Stopping criterion: e.g. time, # generations, solution quality,
generations without improvement

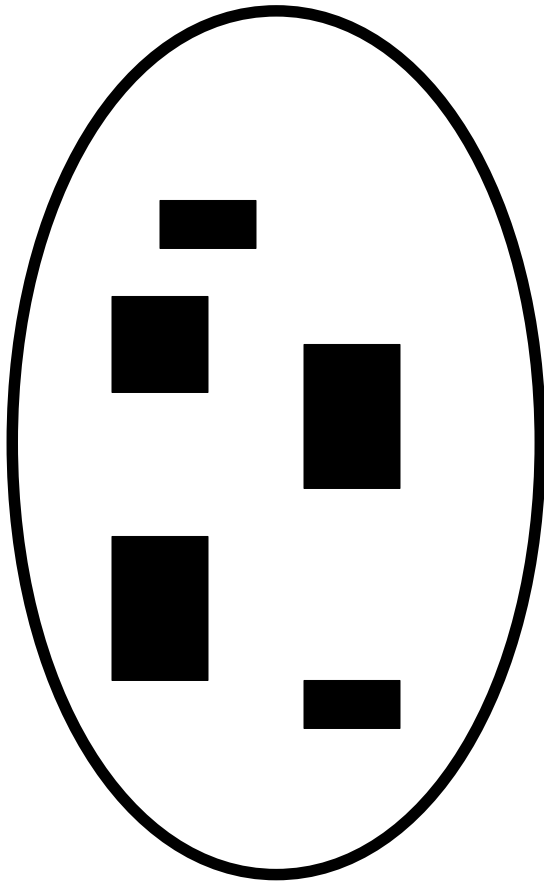
Reference

J.F. Gonçalves and M.G.C.R., "Biased random-key genetic algorithms for combinatorial optimization," AT&T Labs Research Technical Report, Florham Park, New Jersey, Oct. 2009

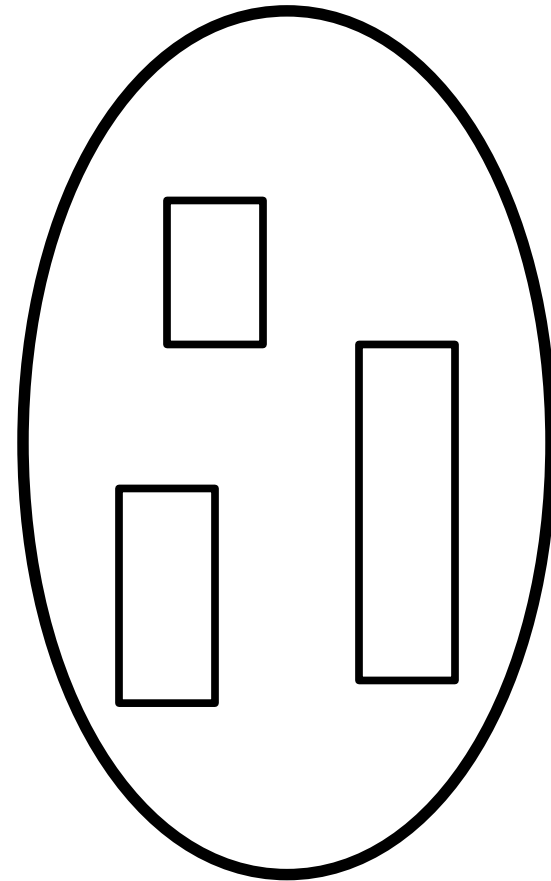
<http://www.research.att.com/~mgcr/doc/srkga.pdf>

Generalized quadratic assignment problem

N : set of n facilities



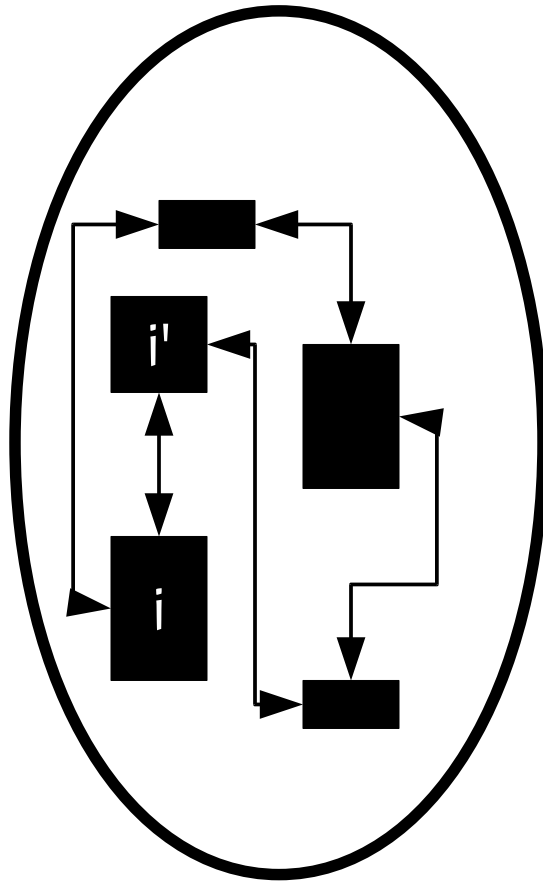
M : set of m locations



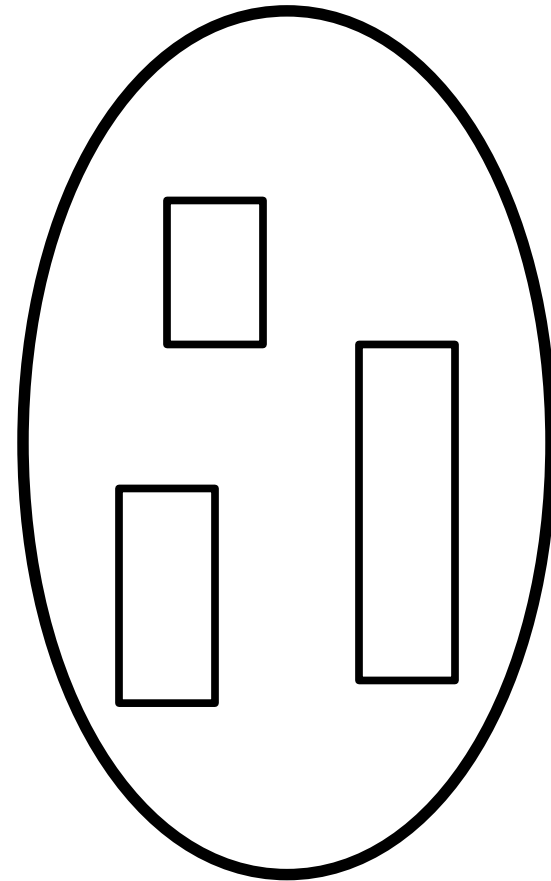
d_i : capacity demanded by facility $i \in N$

Q_j : capacity of location $j \in M$

N: set of n facilities

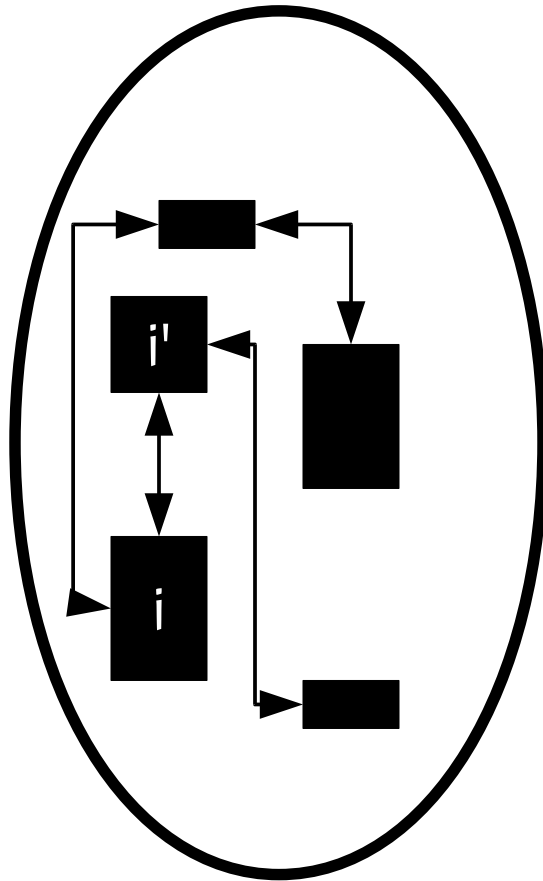


M: set of m locations

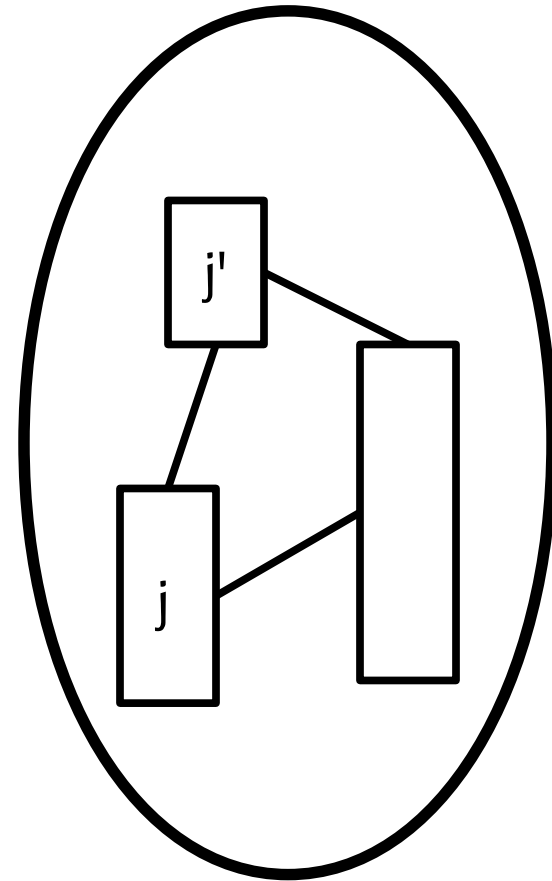


$A_{n \times n} = (a_{ij})$: flow between facilities

N: set of n facilities



M: set of m locations

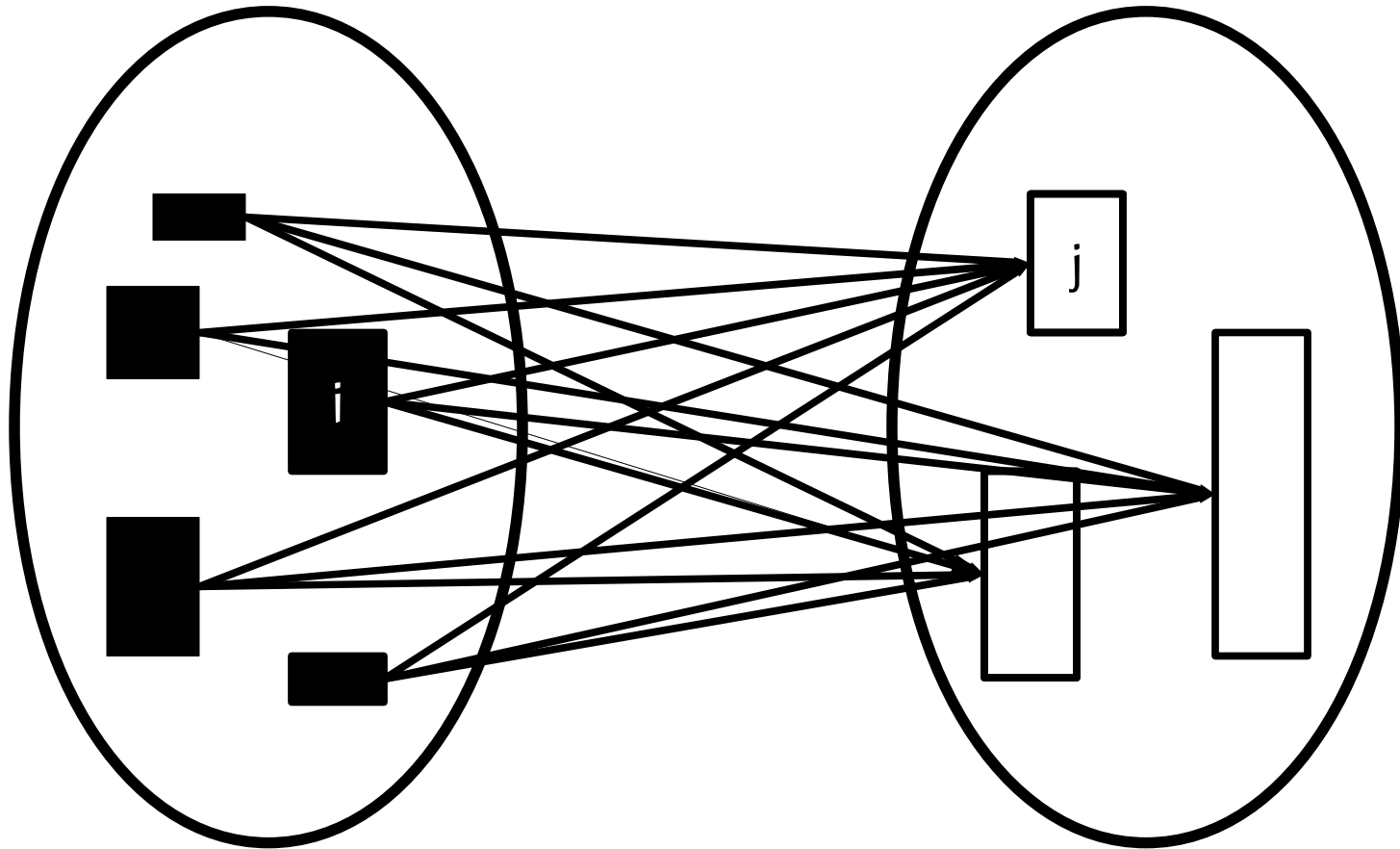


$A_{n \times n} = (a_{ij})$: flow between facilities

$B_{m \times m} = (b_{jj'})$: distance between locations

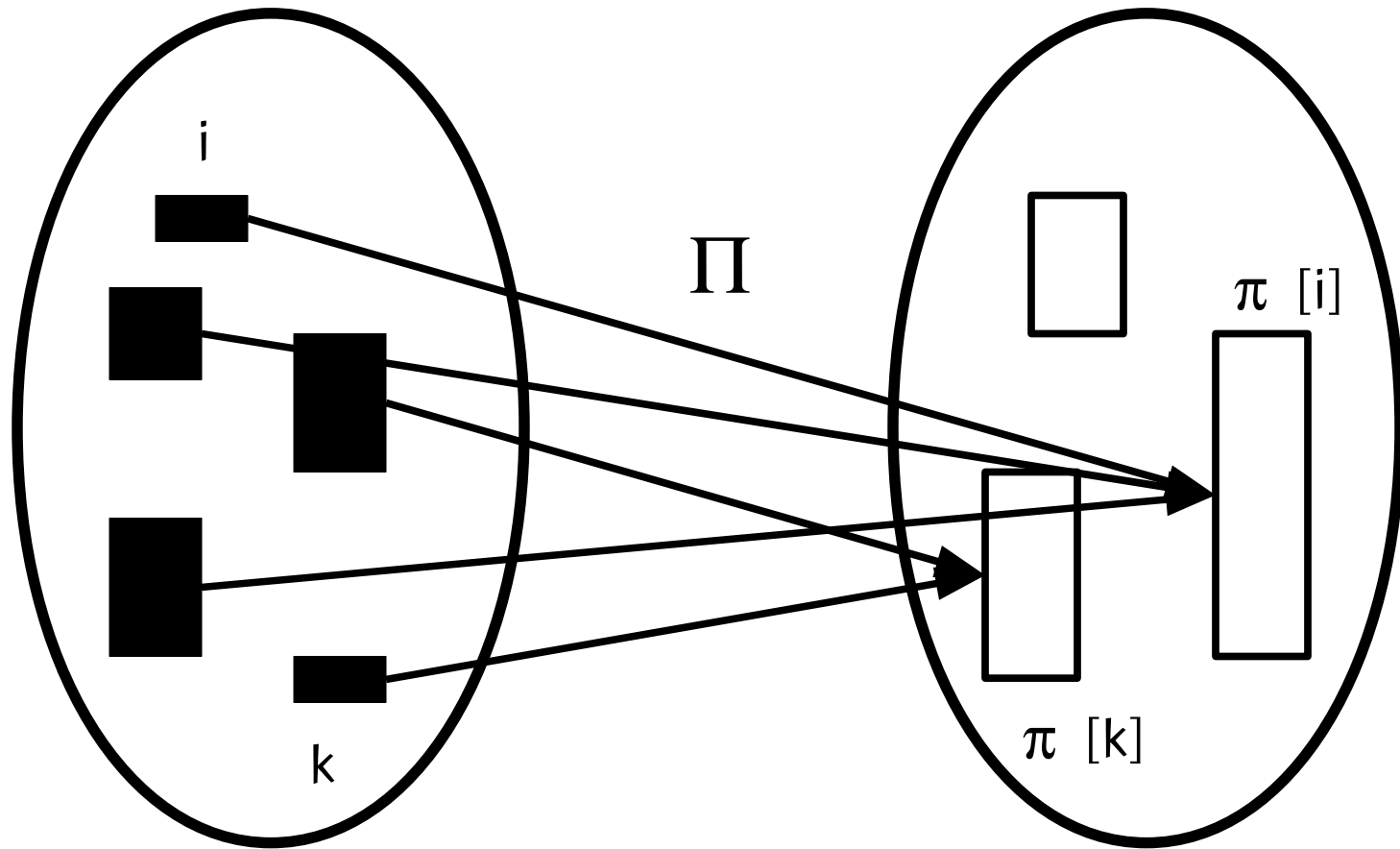
N: set of n facilities

M: set of m locations



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

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



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.

Generalized quadratic assignment

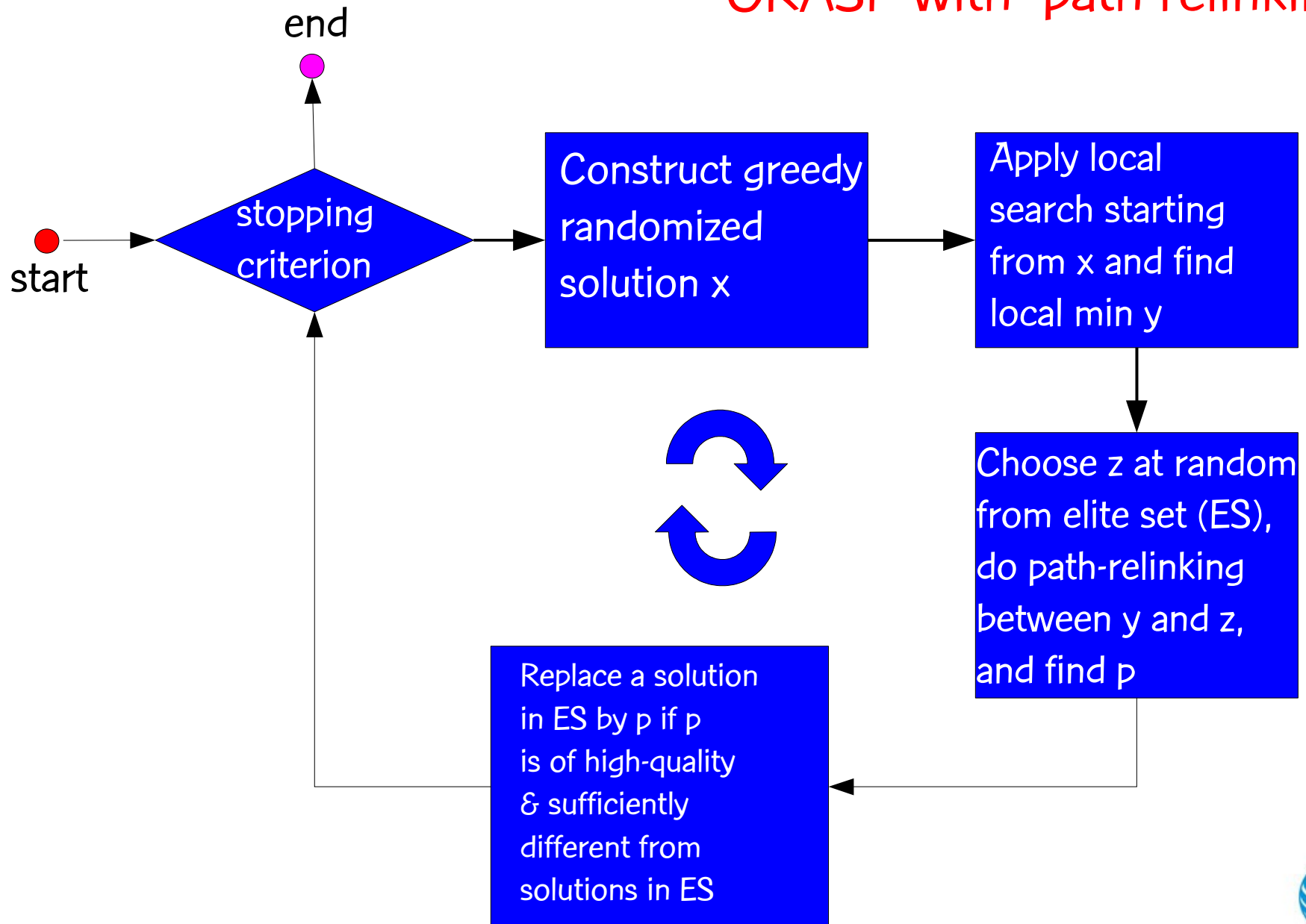
The GQAP is NP-hard.

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.

Solution method

GRASP with path-relinking



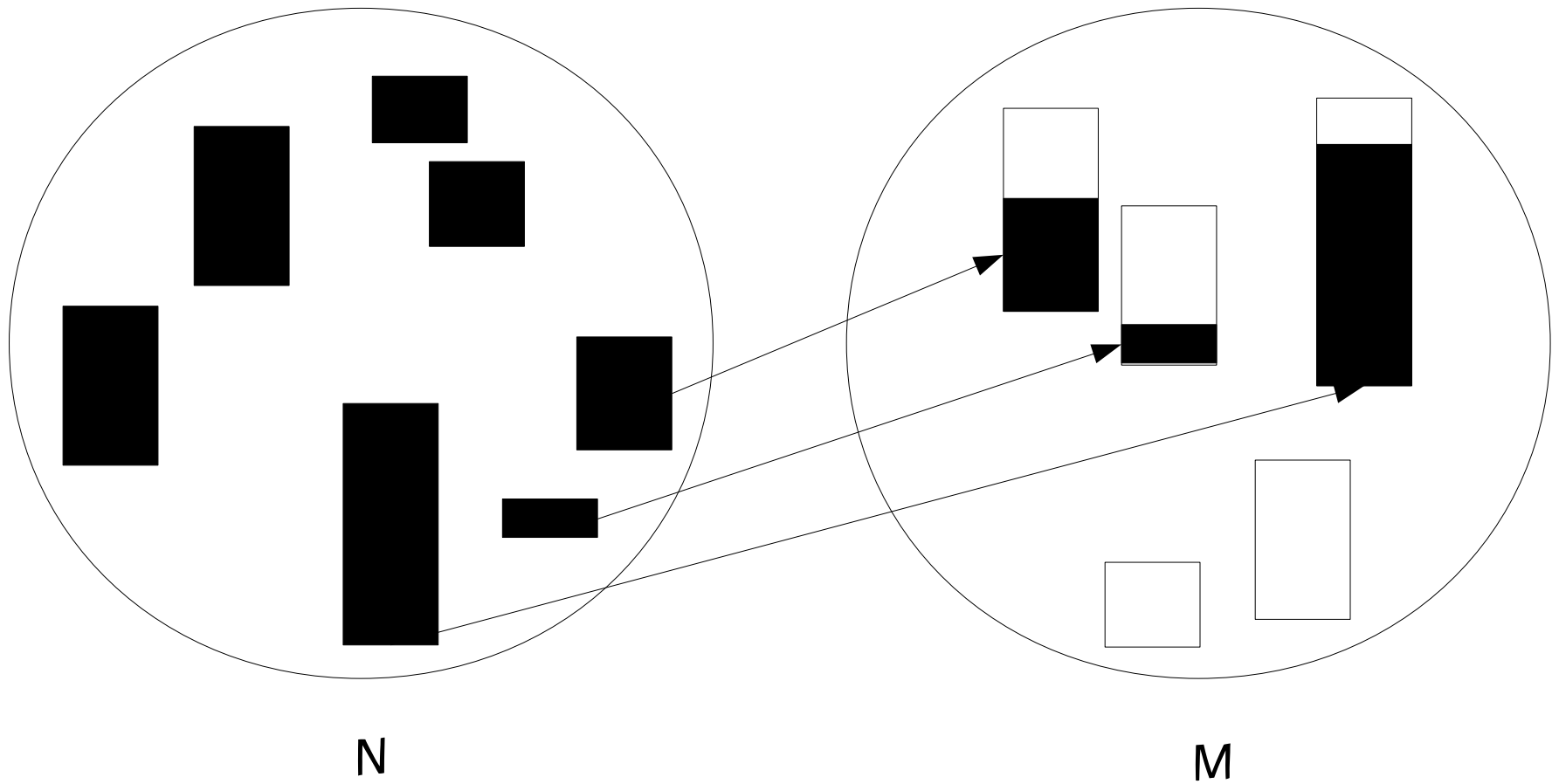
Components

Construction of greedy randomized solution

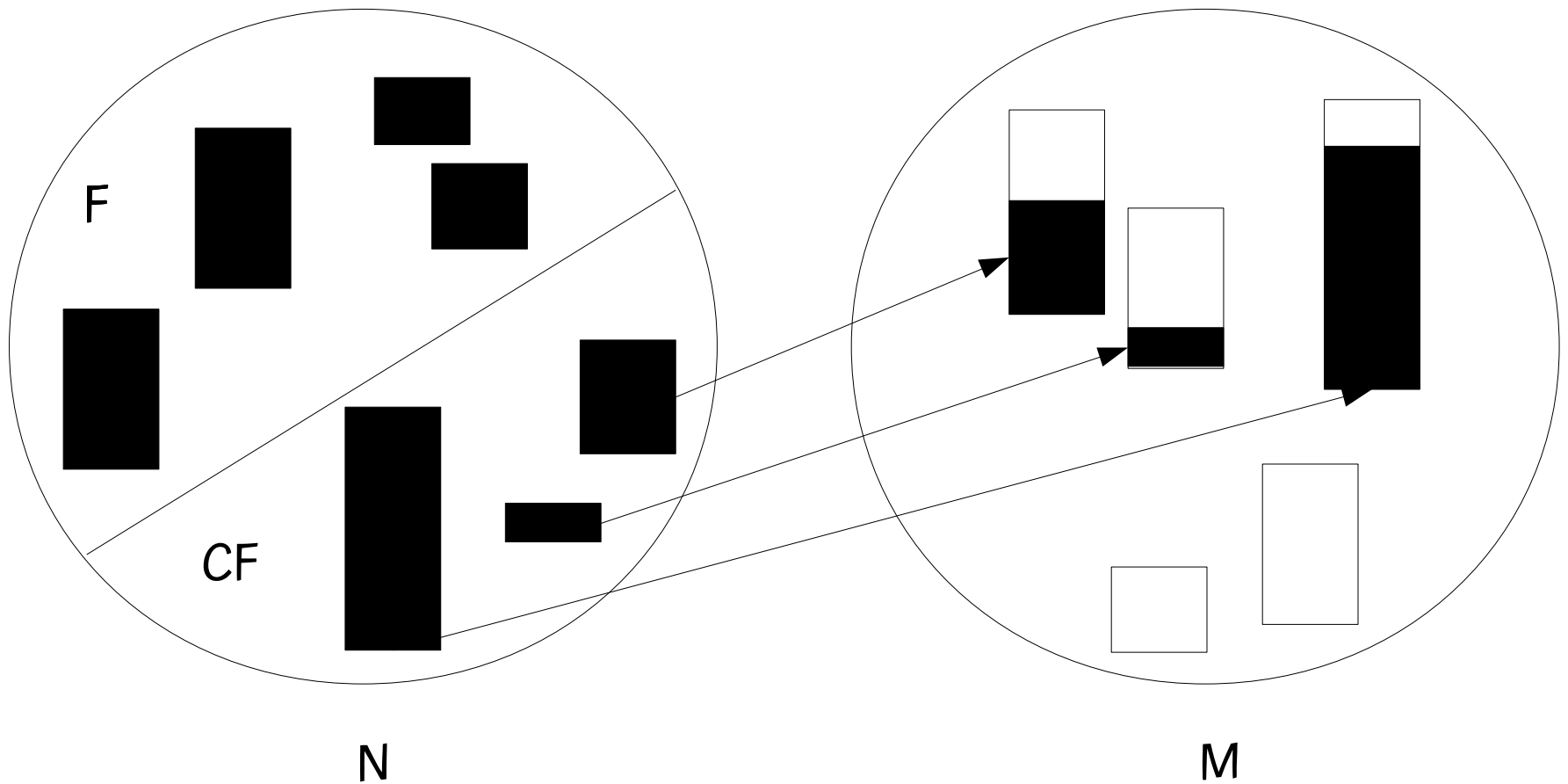
Local search

Path-relinking

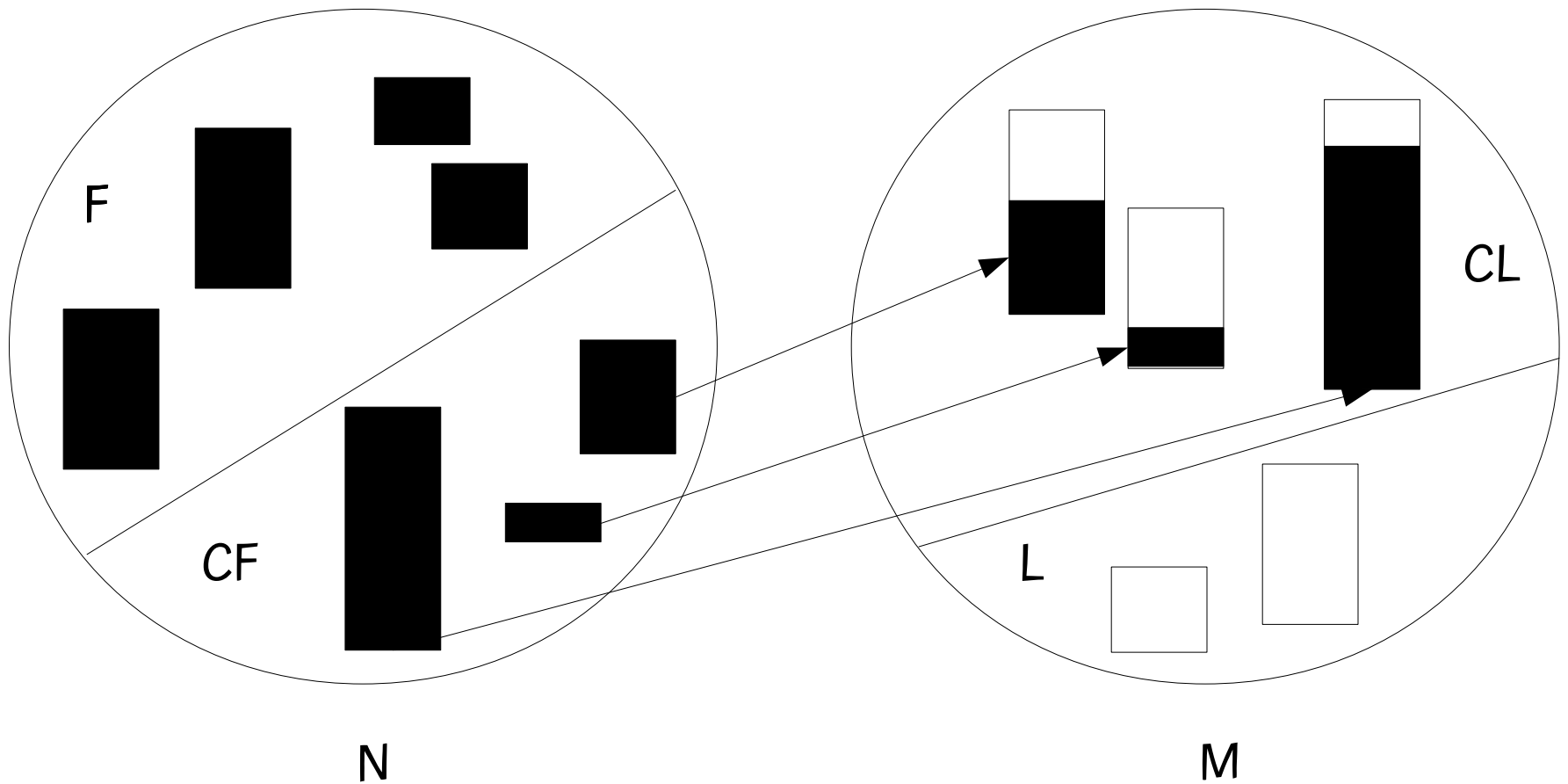
GRASP construction



Suppose a number of assignments have already been made

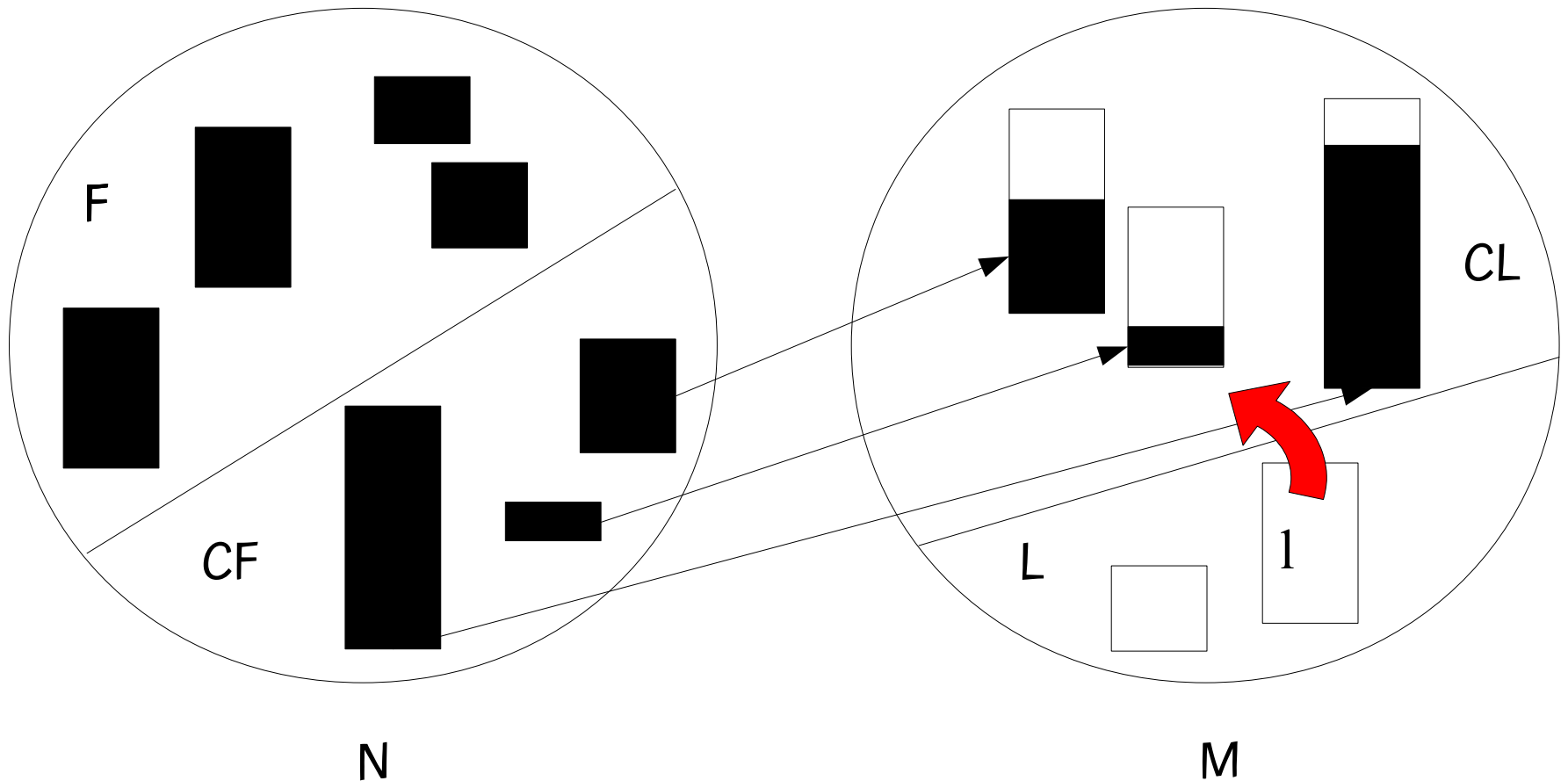


$N = F \cup CF$, where CF is the set of assigned facilities and F the set of facilities not yet assigned to some location



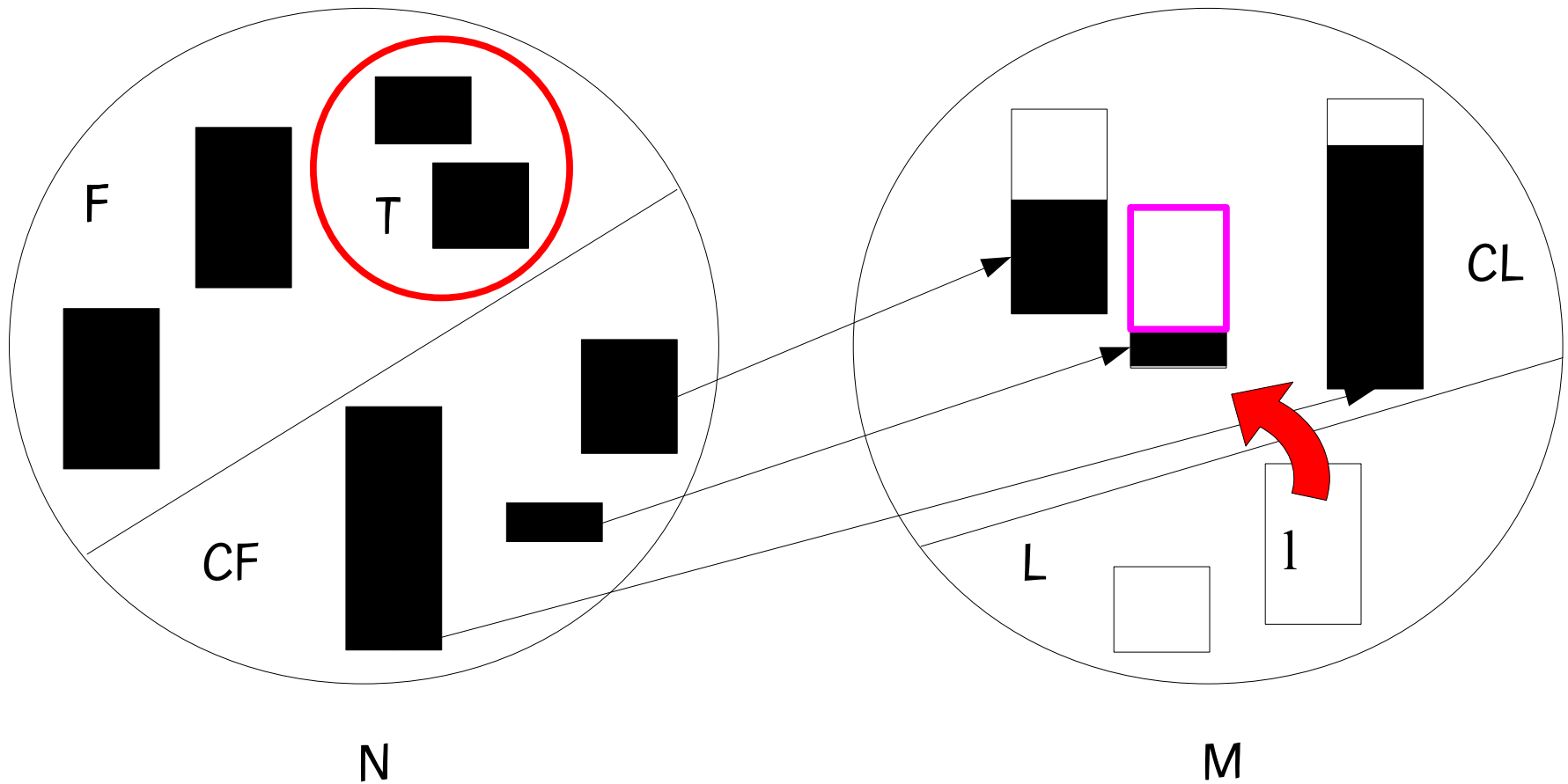
$M = L \cup CL$, where CL is the set of previously chosen locations and L the set of unselected locations.

Procedure to select a NEW location from set L



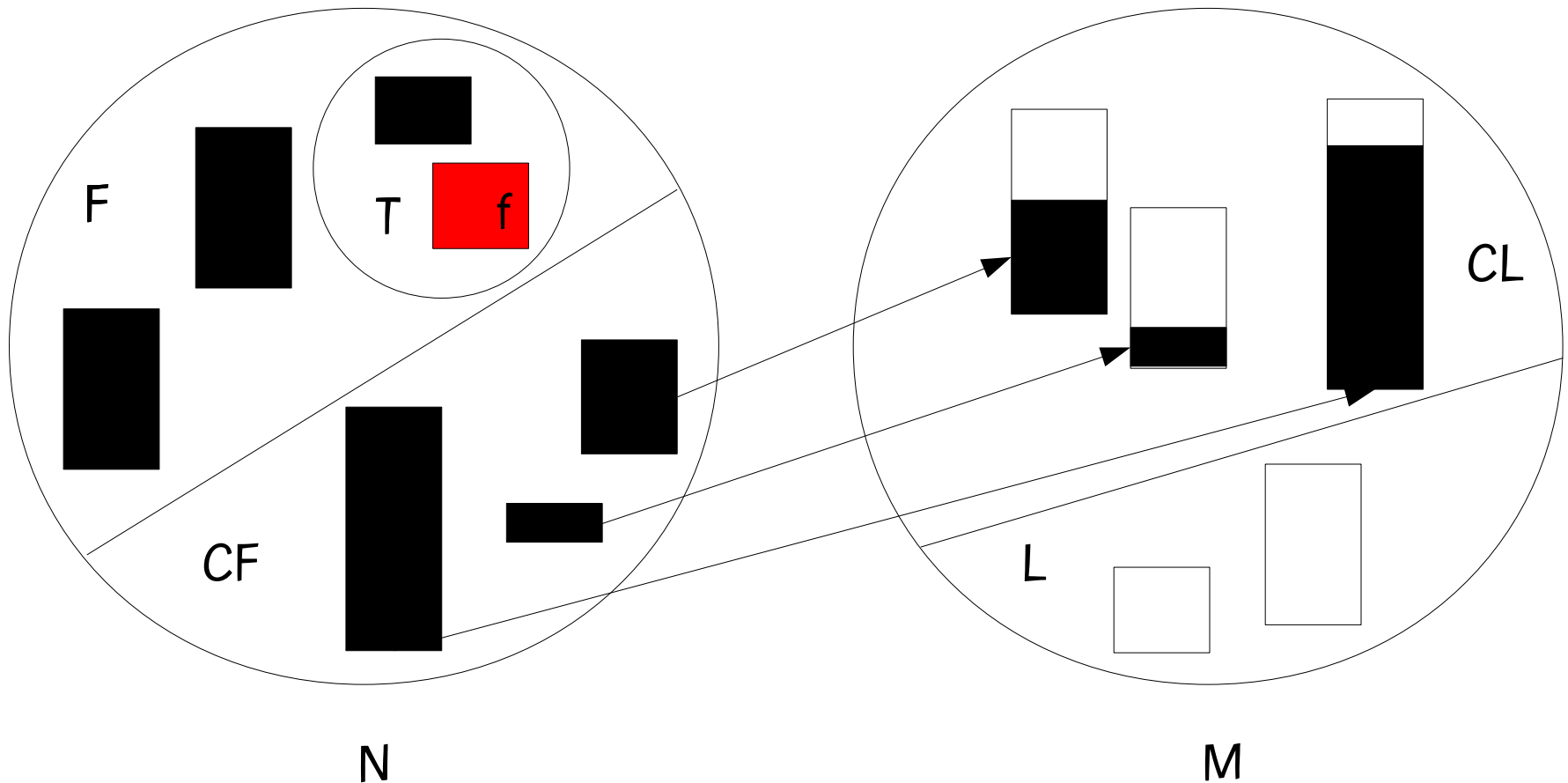
With probability P , randomly select a new location I from L , favoring those having **high capacity** and those **close to all locations in CL** , and move location I to CL .

Procedure to select a new location from set L



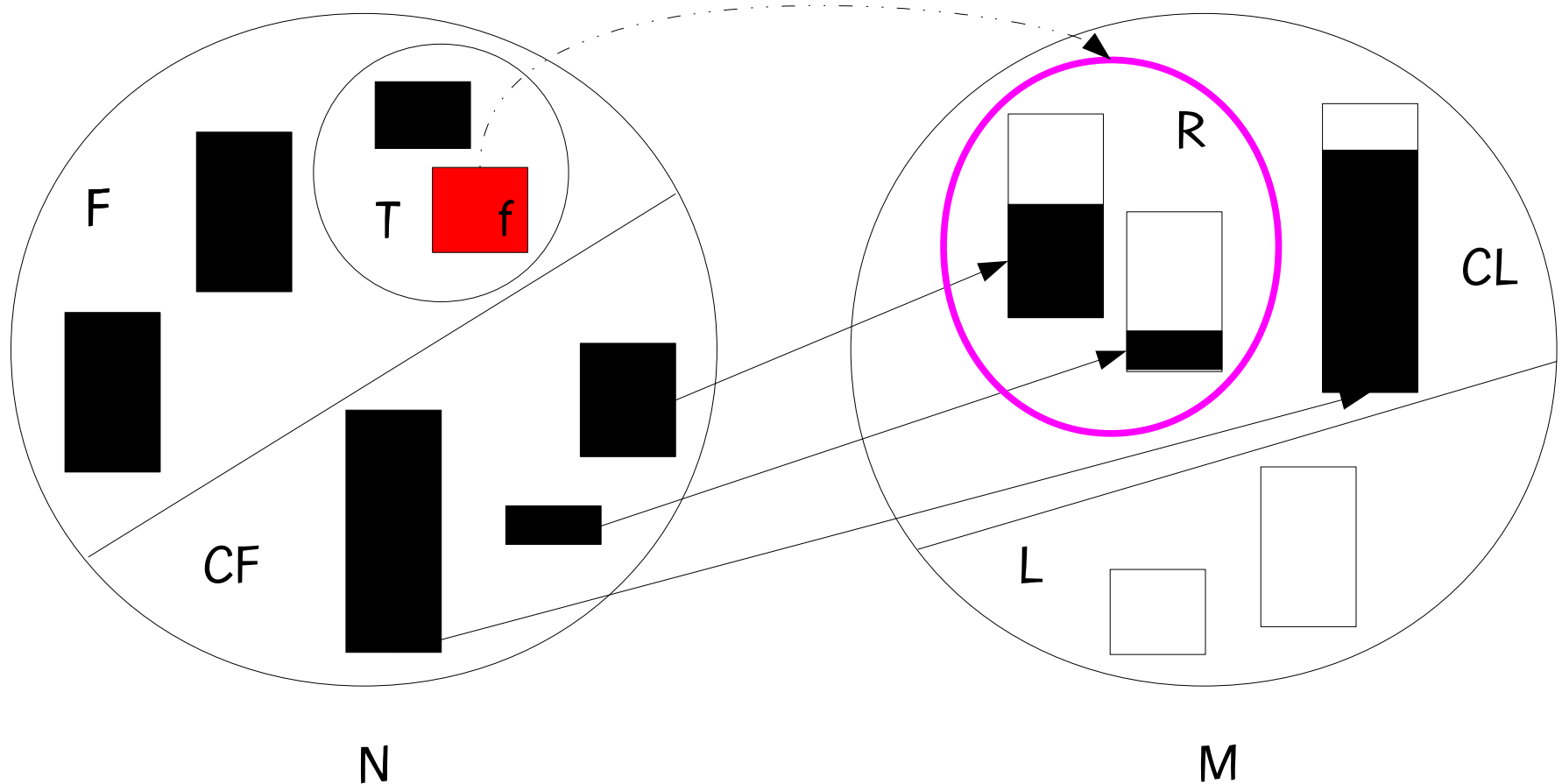
The probability P is equal to $1 - (|T| / |F|)$, where the **set T consists of all unassigned facilities with demands less than or equal to the maximum available capacity of locations in CL.**

Facility selection procedure



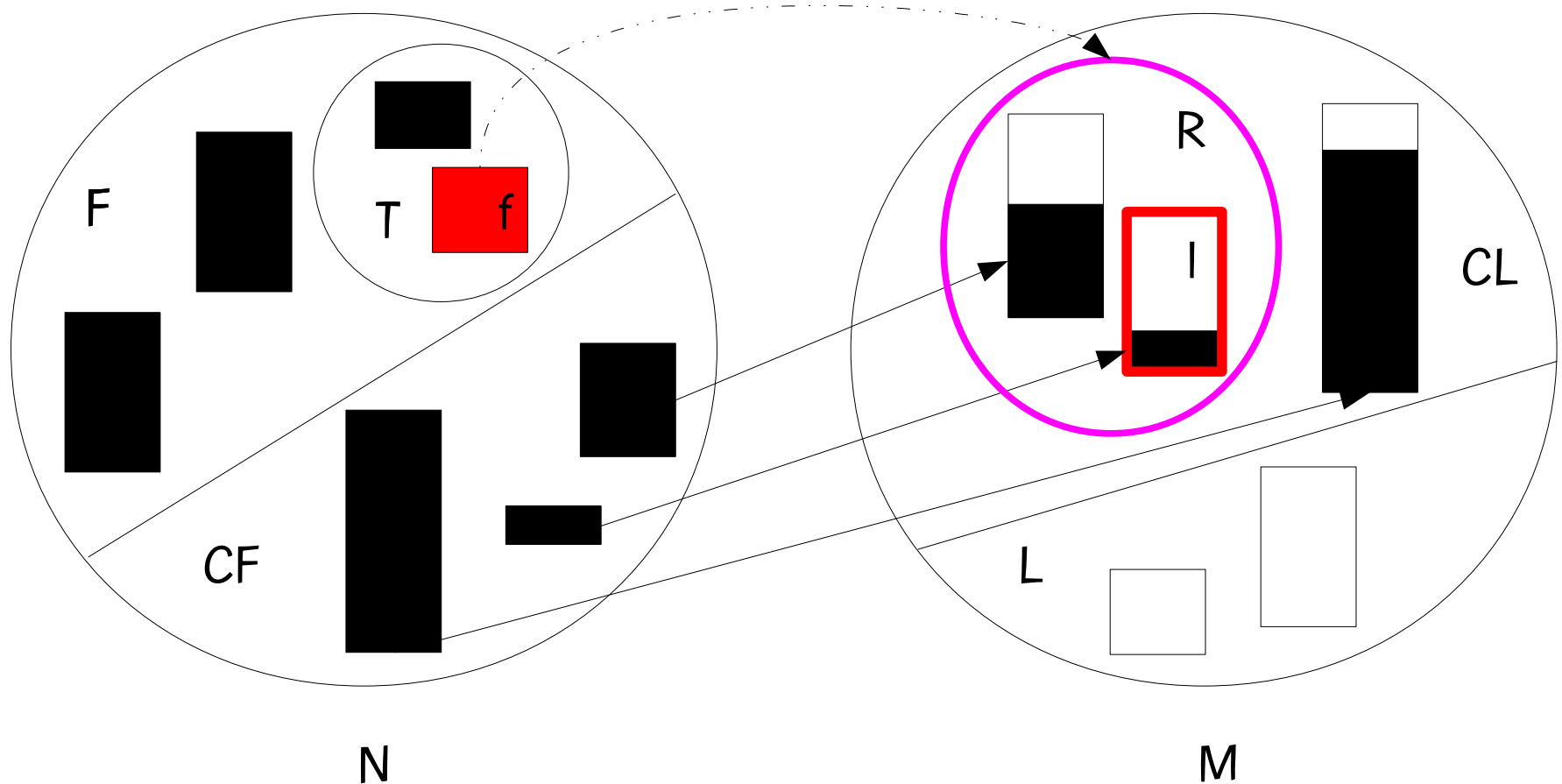
Randomly select a **facility $f \in T$** favoring facilities that have high demand and high flows to other facilities.

Procedure to select a location from CL (step 1)



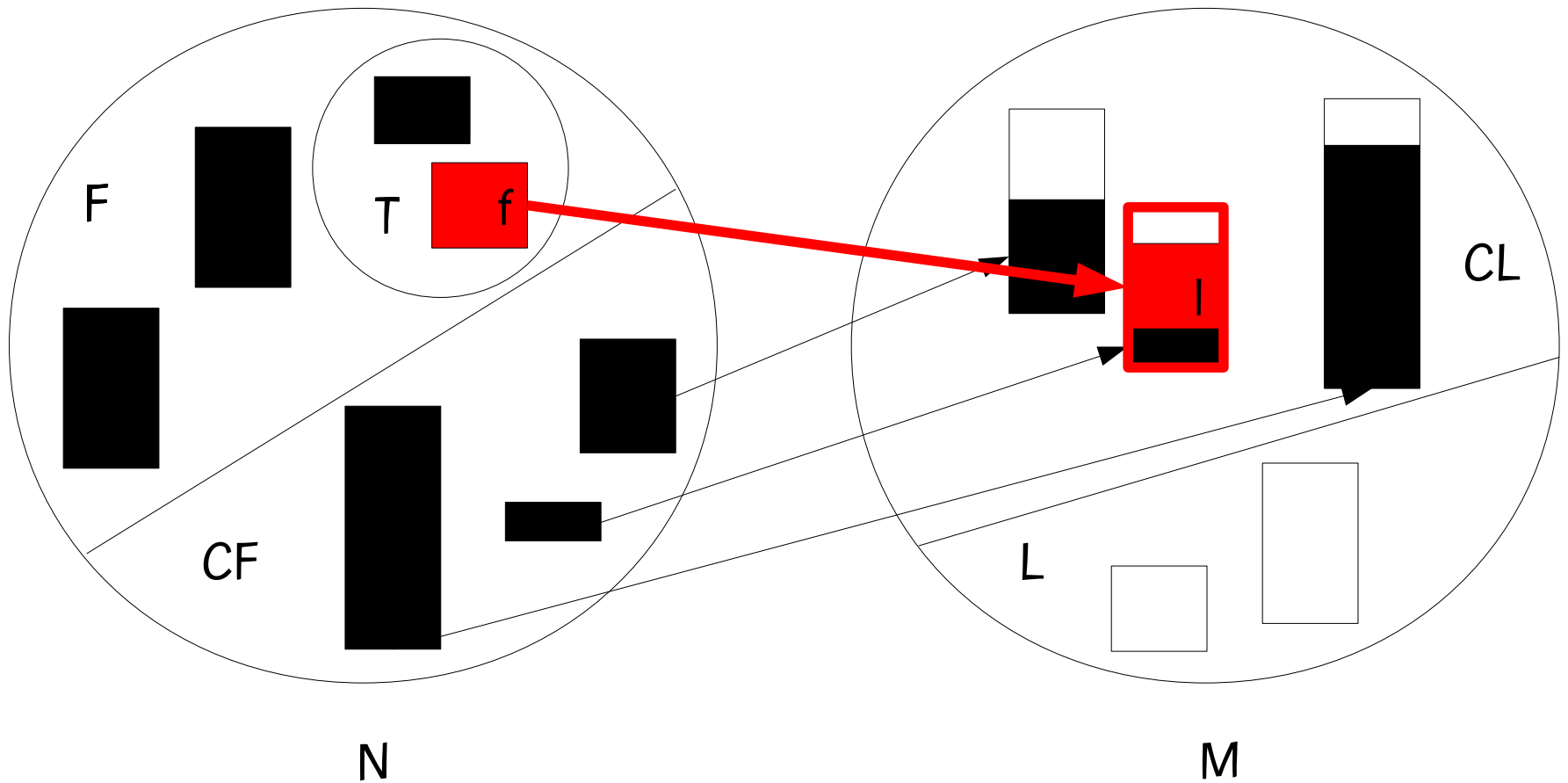
1. Let set R to be all locations in CL having slack greater than or equal to demand of facility f ;

Procedure to select a location from CL (step 2)



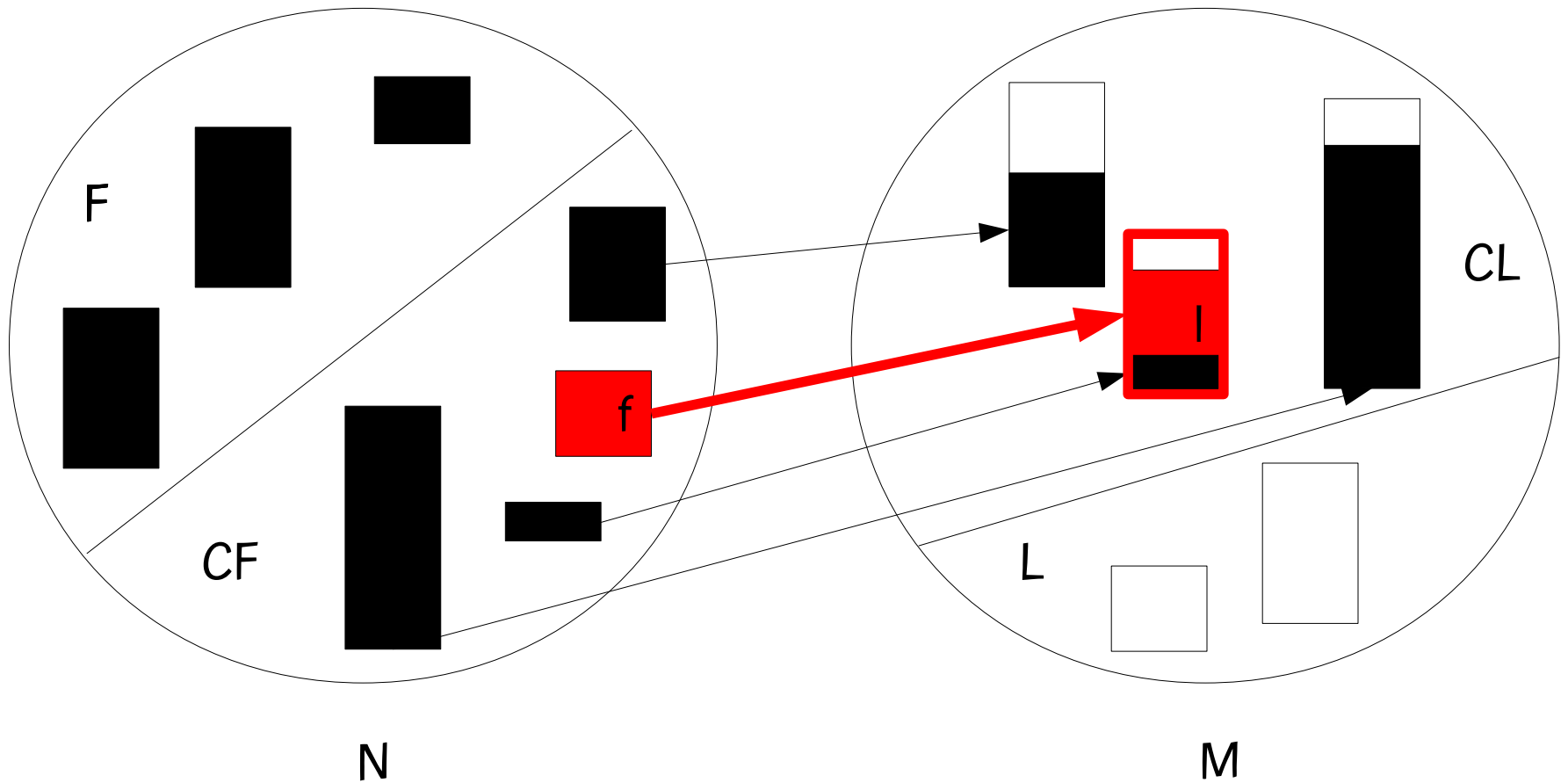
2. Randomly select a location $I \in R$ favoring those having high available capacity and those close to high-capacity locations in CL;

Assignment procedure



Assign facility f to location l

Assignment procedure



Update sets F, CF, and slack of location I

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 $F=\emptyset$).

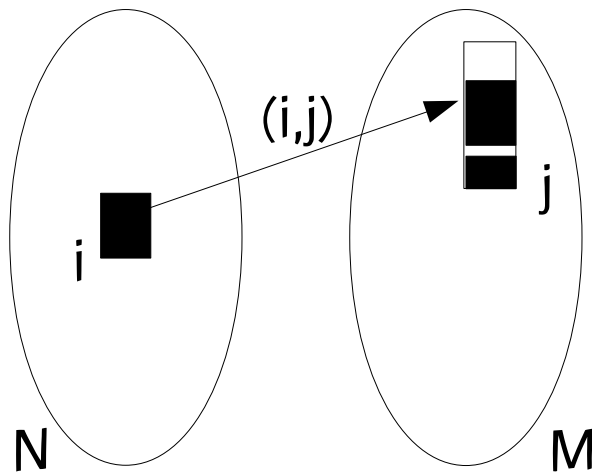
At start of construction, a location l is selected from the set L with probability proportional to its capacity. Location l is placed in CL .

Local search

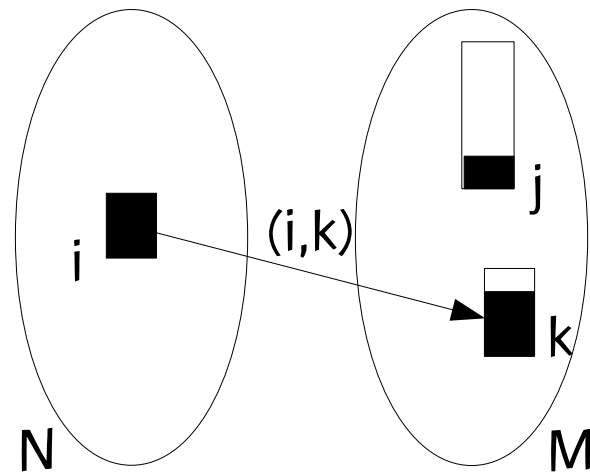
Local search

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



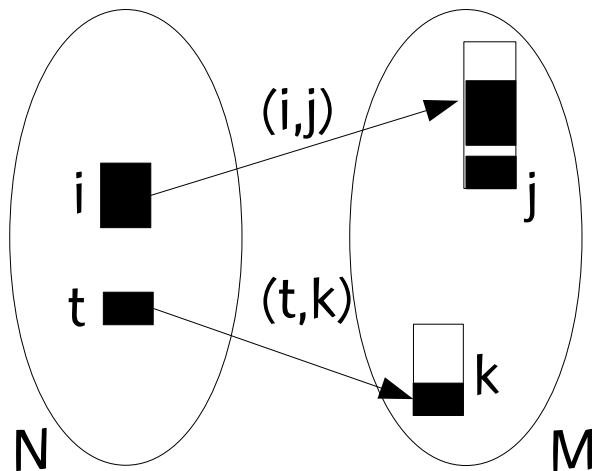
1-move neighbor of p

Local search

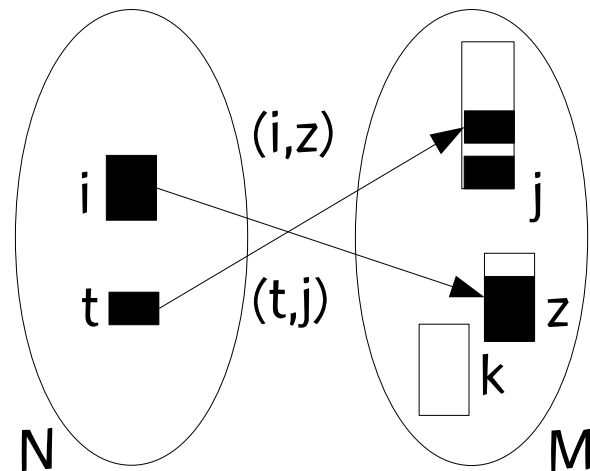
1-move and 2-move neighborhoods from solution p are used in our local search.

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

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

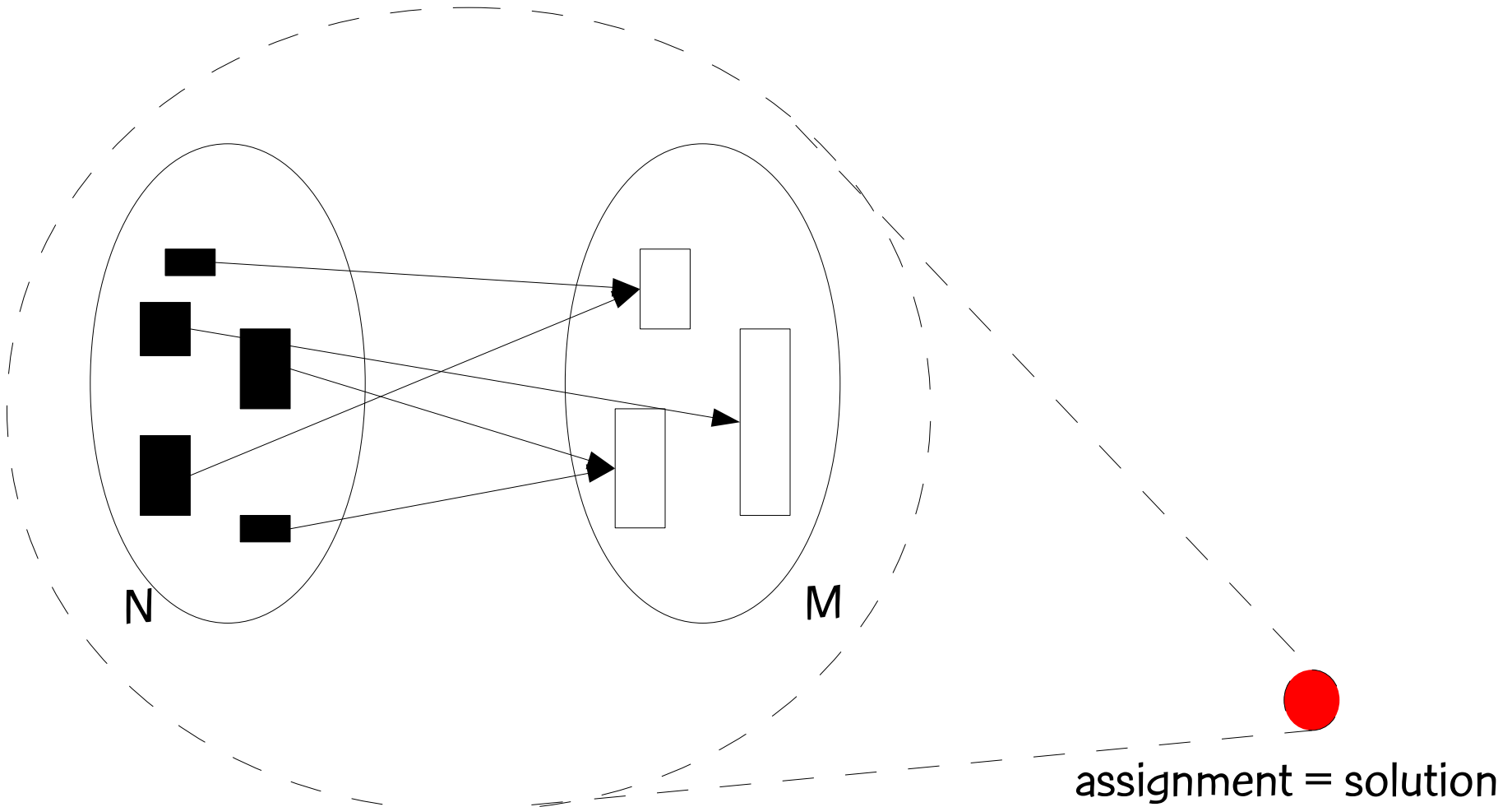


solution p



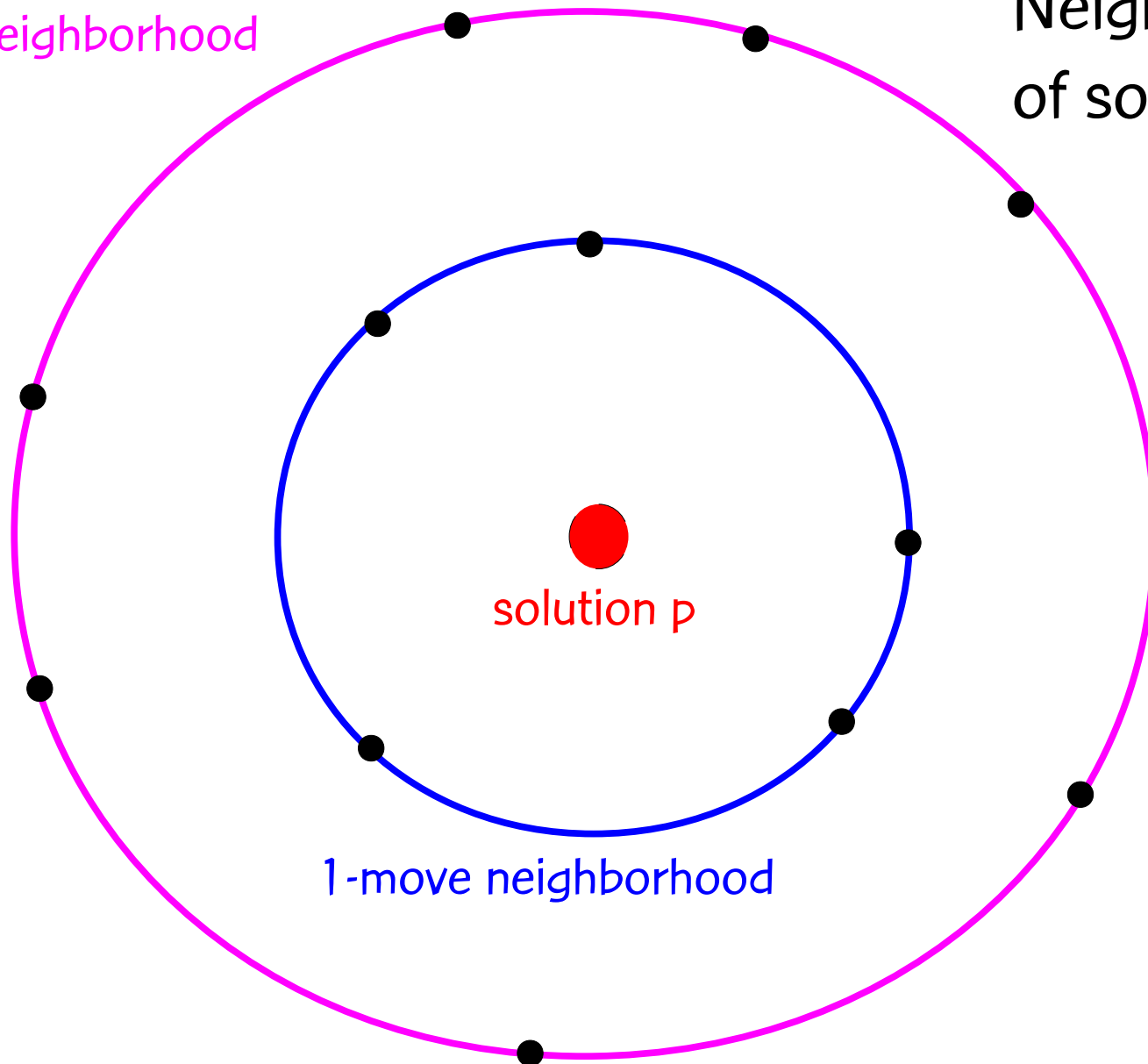
2-move neighbor of p

Assignment representation



2-move neighborhood

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:

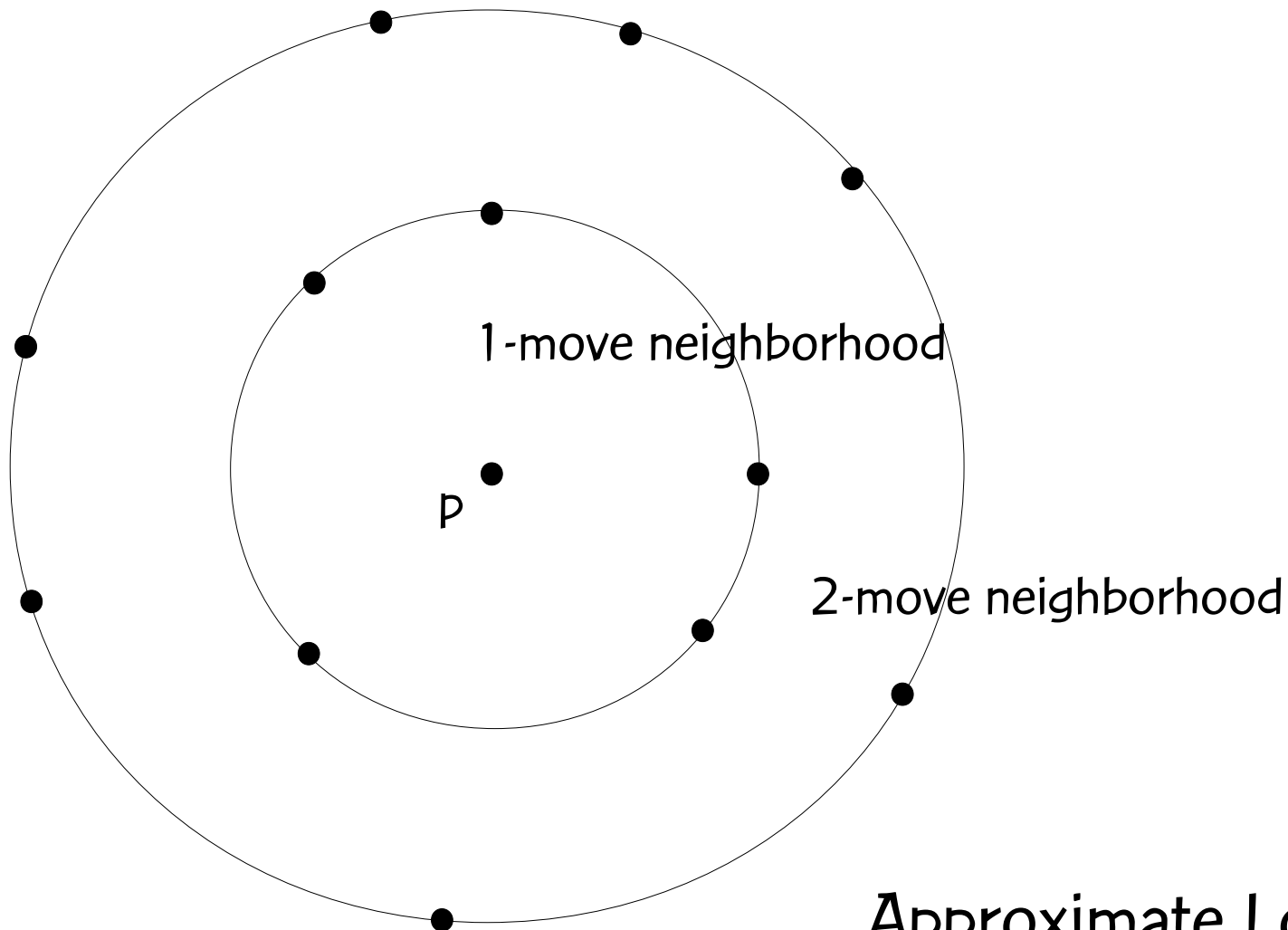
- 1: From solution p , to evaluate its 1-move neighbors until the first improving solution q is found.
- 2: If q does not exist, continue search in the 2-move neighborhood.
- 3: If q does not exist in the 2-move neighborhood, stop. Otherwise, assign $p = q$ and go to step 1.

Approximate local search

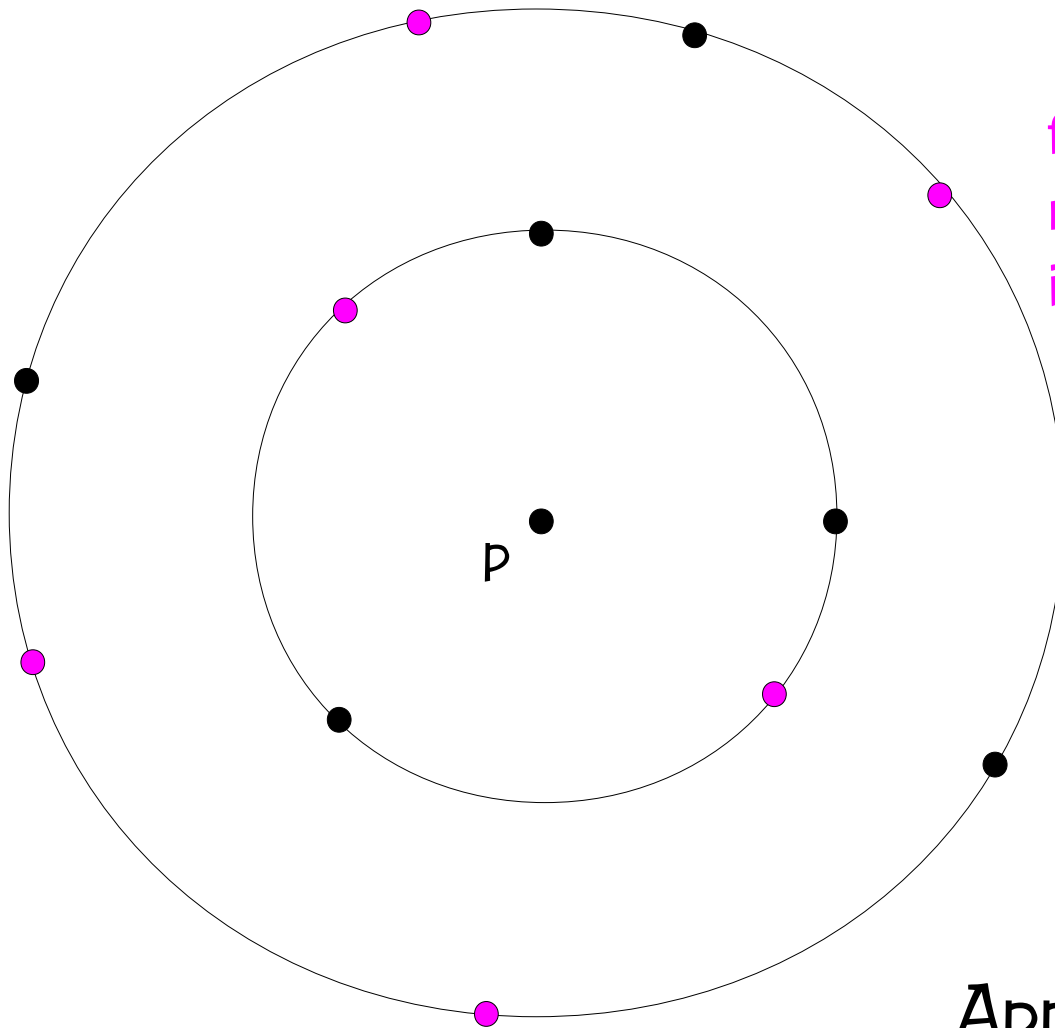
Tradeoff between best & first improvement: sample the neighborhood of solution p .

Neighborhoods can be very large for best improvement

Local search can take very long

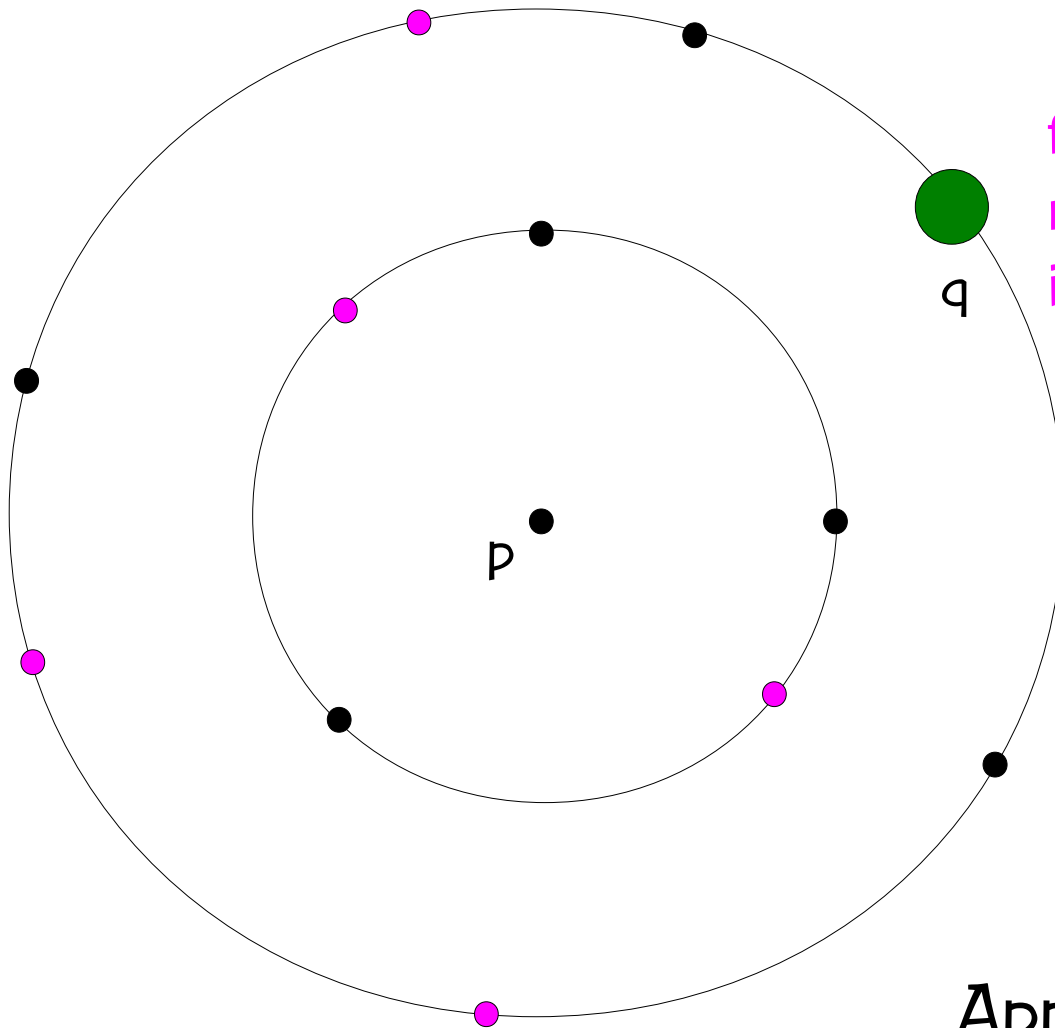


Approximate Local Search



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

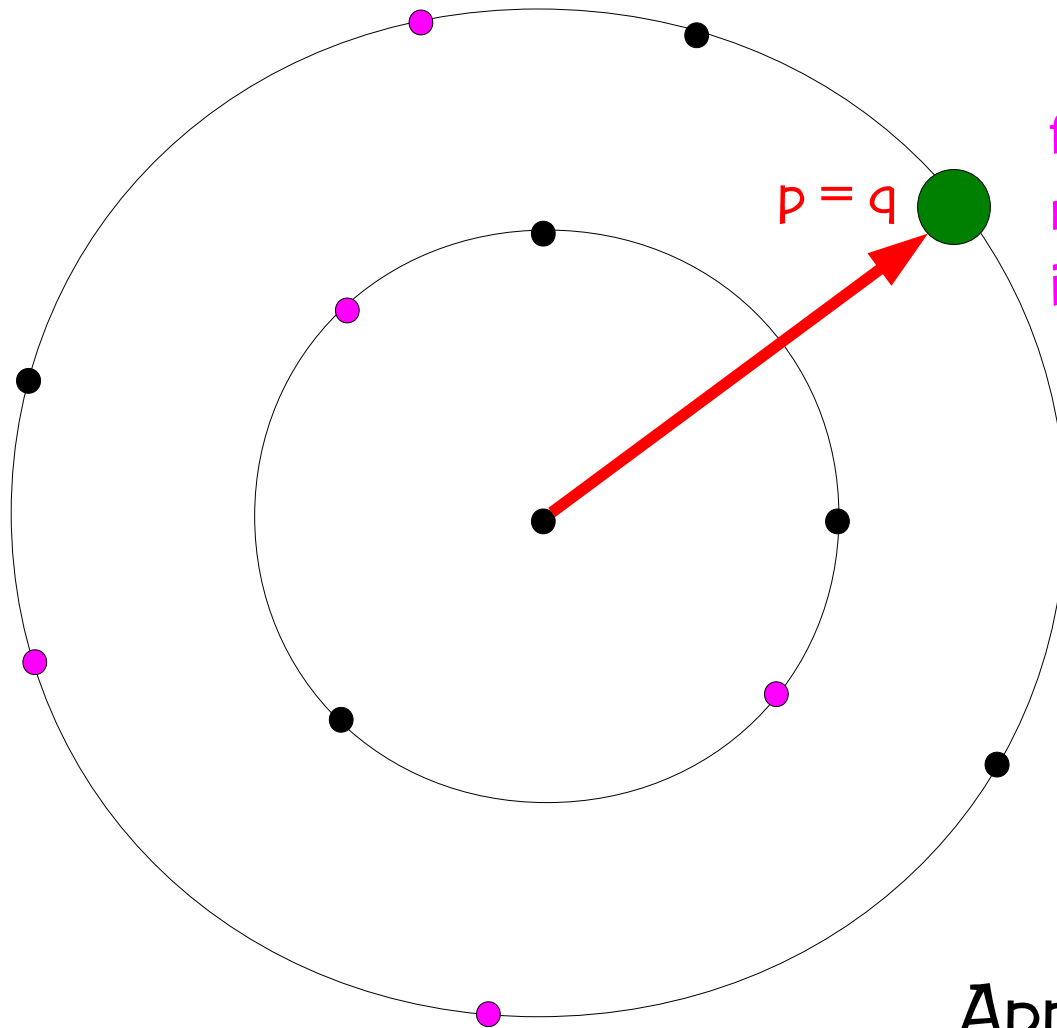
Approximate Local Search



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



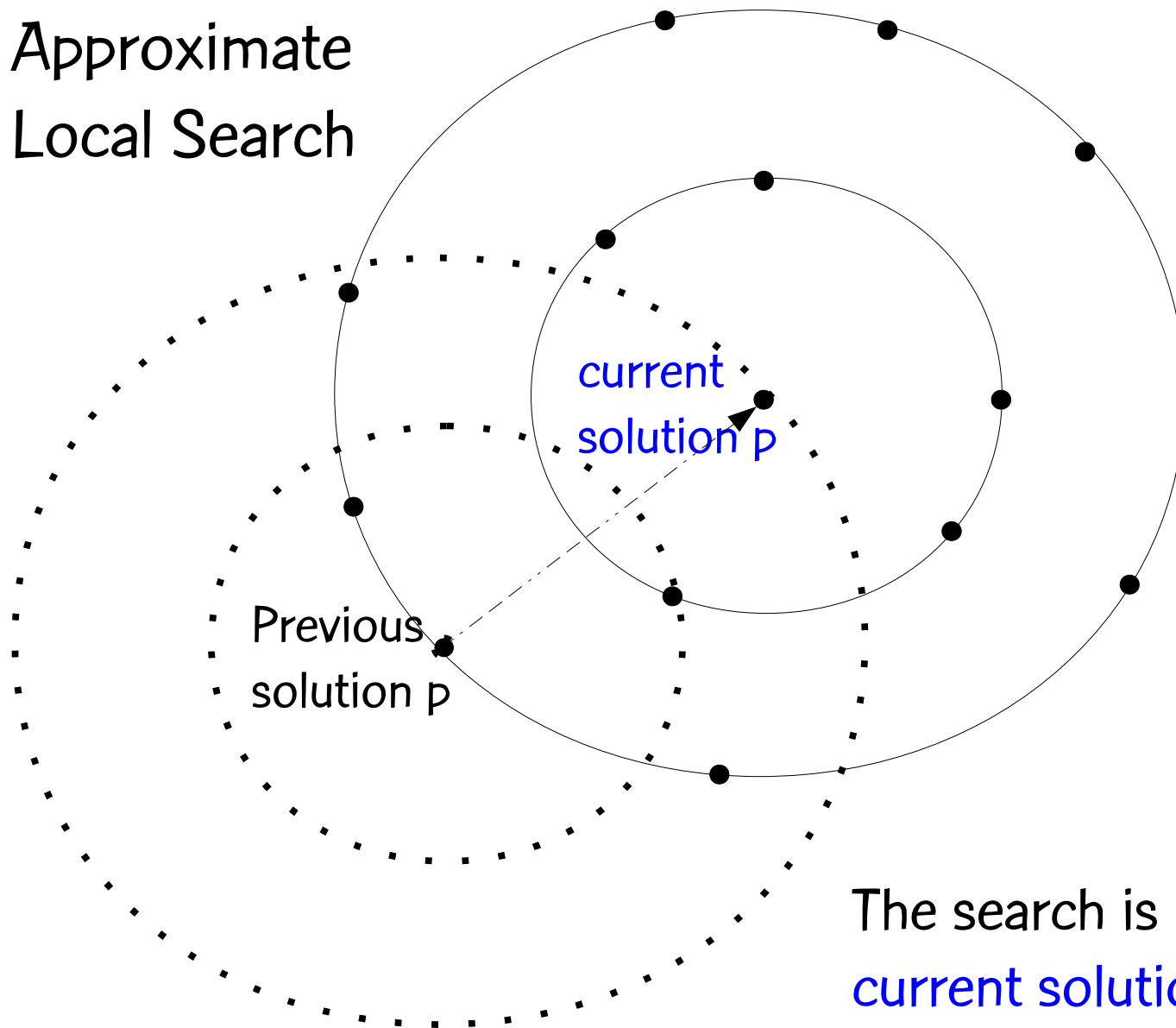
1. Sample k improving solutions from 1-move and 2-move neighborhood of p and place them in an elite set E .

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

3. Update $p = q$

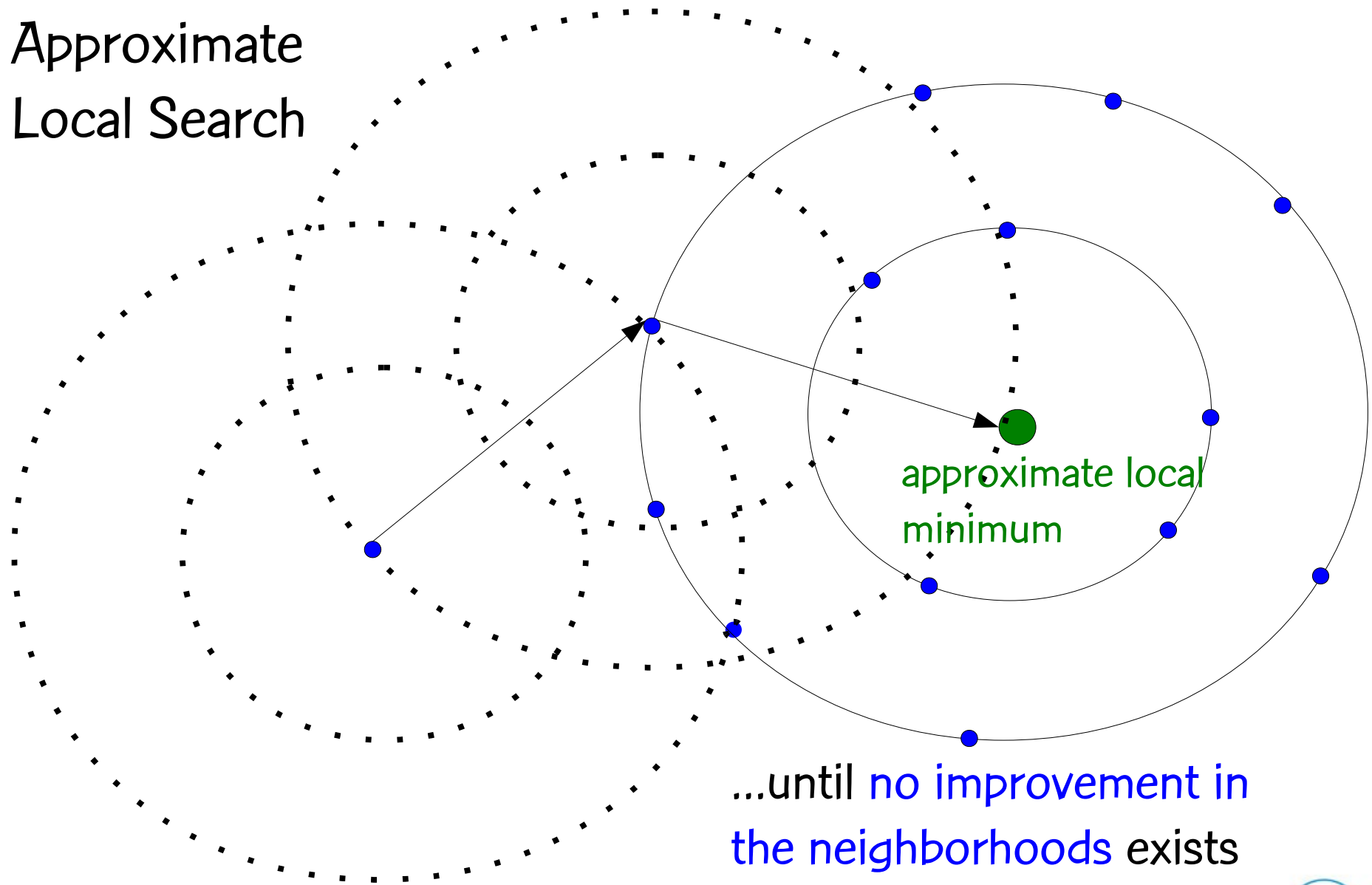
Approximate Local Search

Approximate Local Search



The search is repeated from
current solution p until

Approximate Local Search

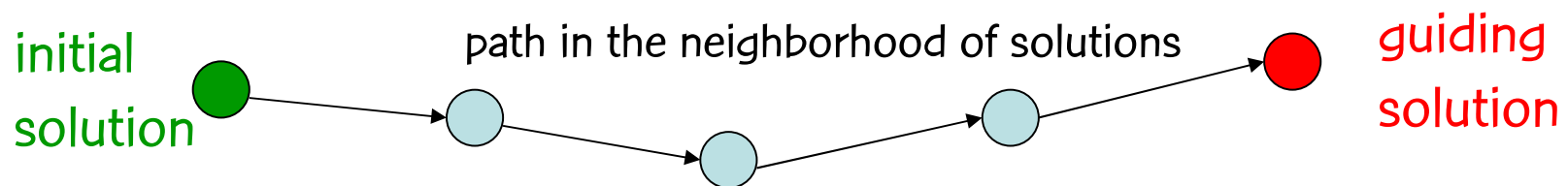


...until no improvement in
the neighborhoods exists

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:



Path-relinking

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

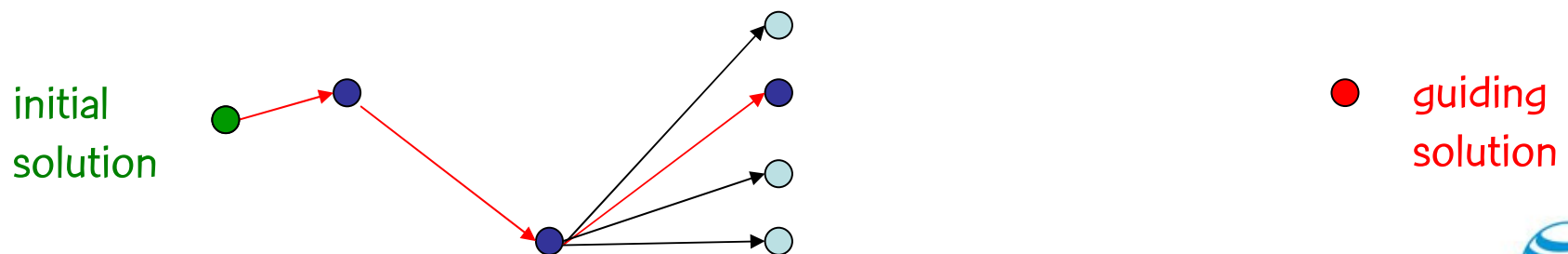
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



Path-relinking

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

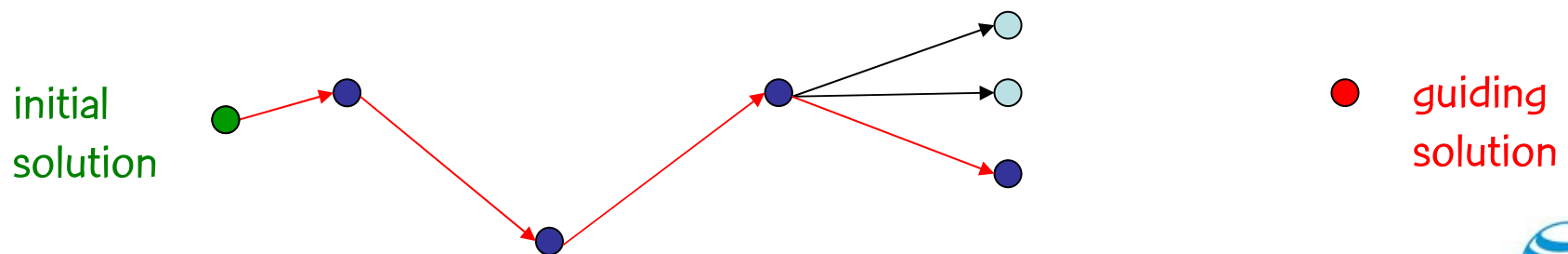
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



Path-relinking

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

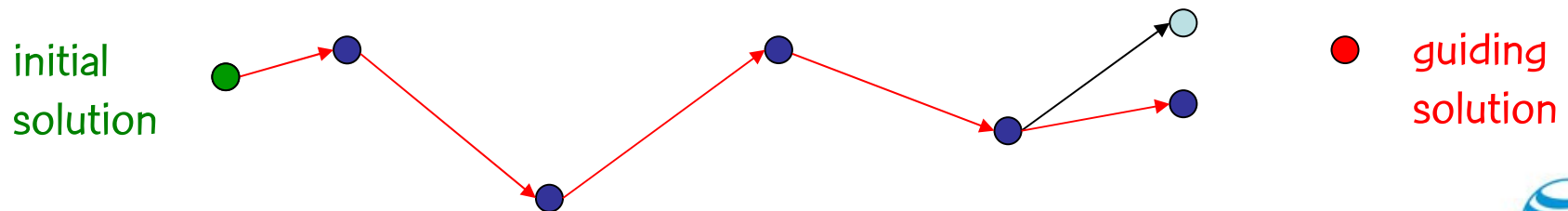
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:



Path-relinking

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

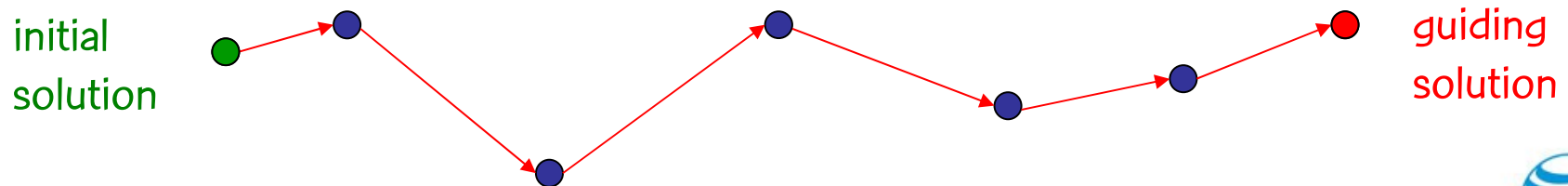
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:

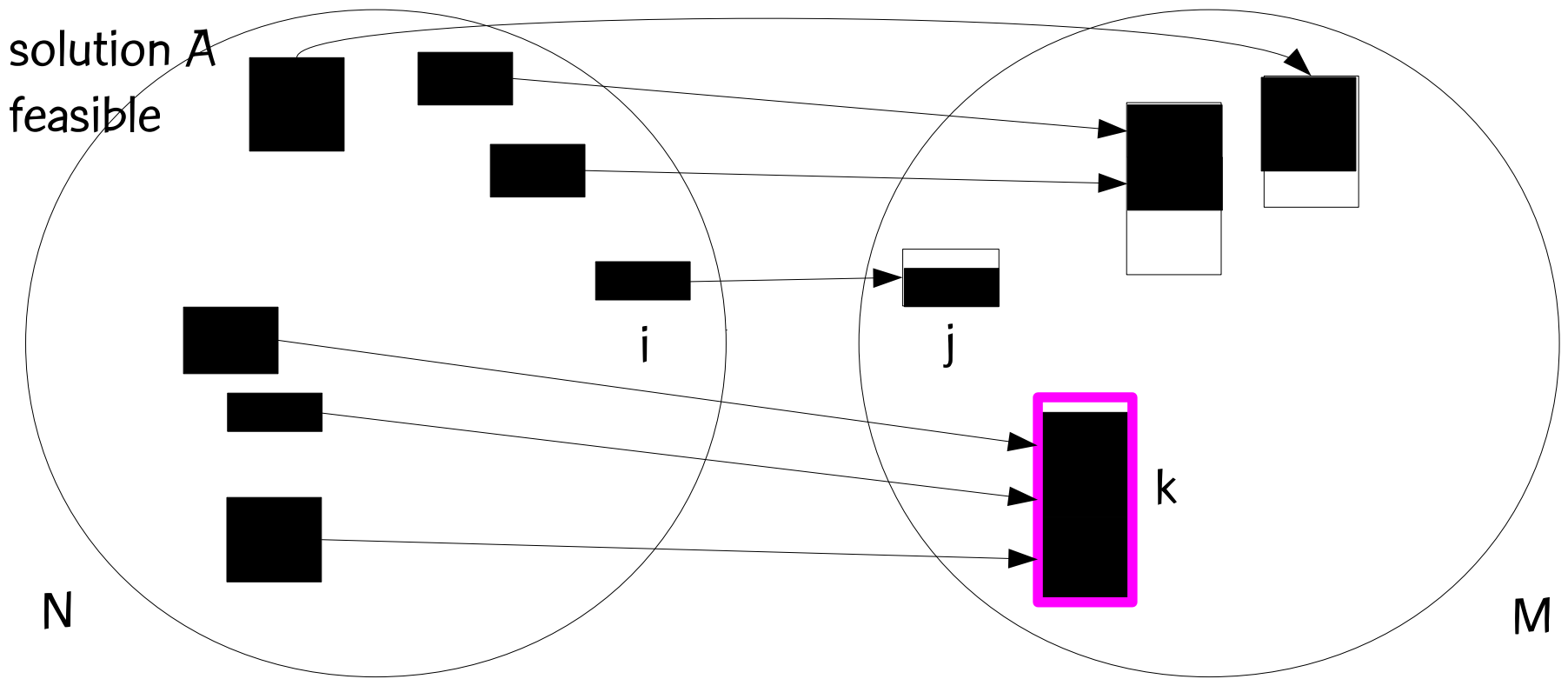


Path-relinking

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

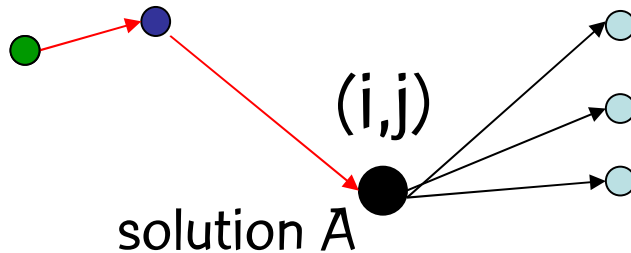
At each step, all moves that incorporate attributes of the guiding solution are evaluated and the best move is selected:





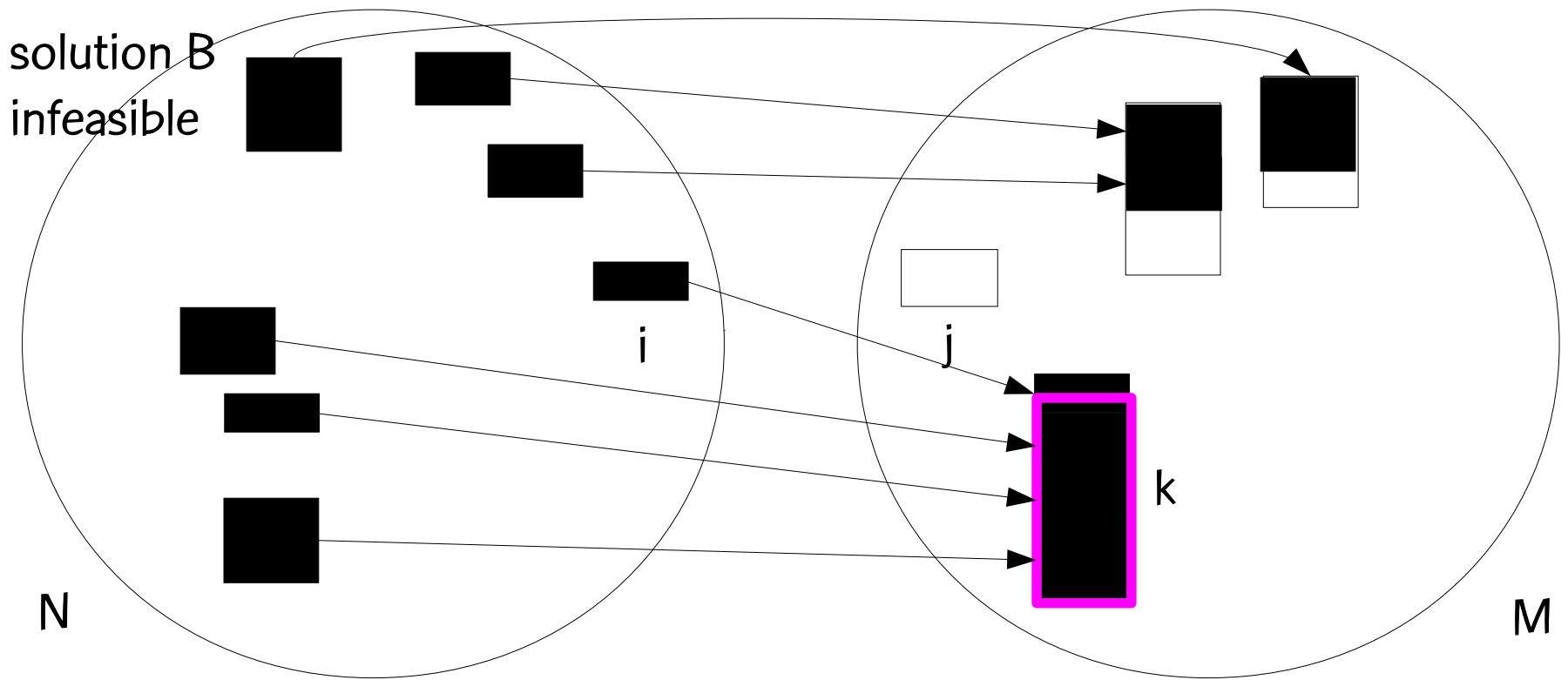
Infeasibility in path-relinking for GQAP

initial
solution

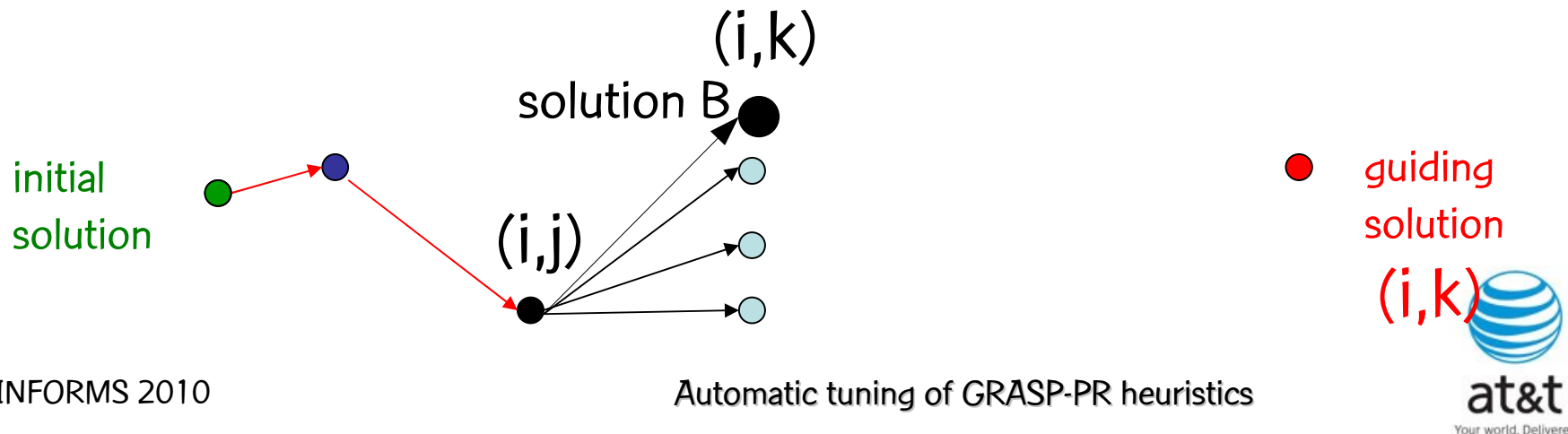


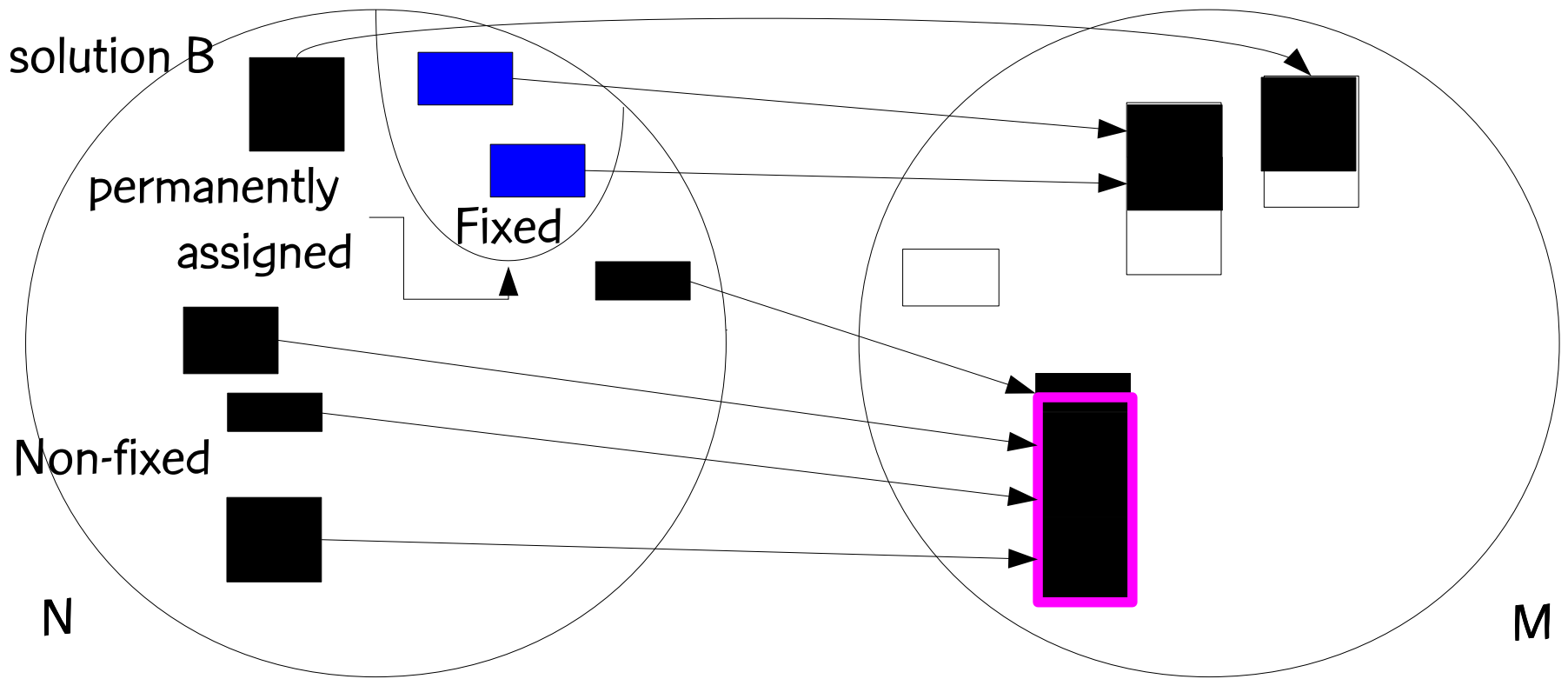
● guiding
solution
 (i,k)



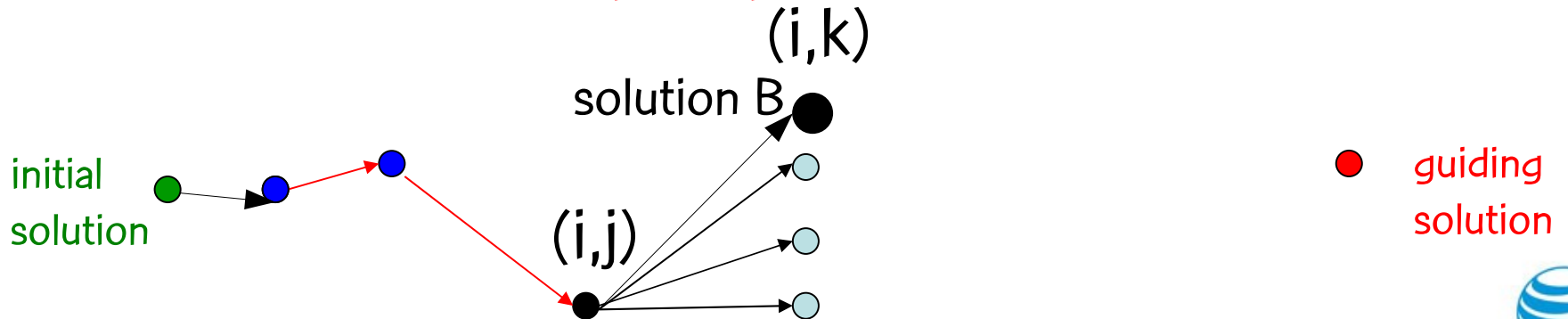


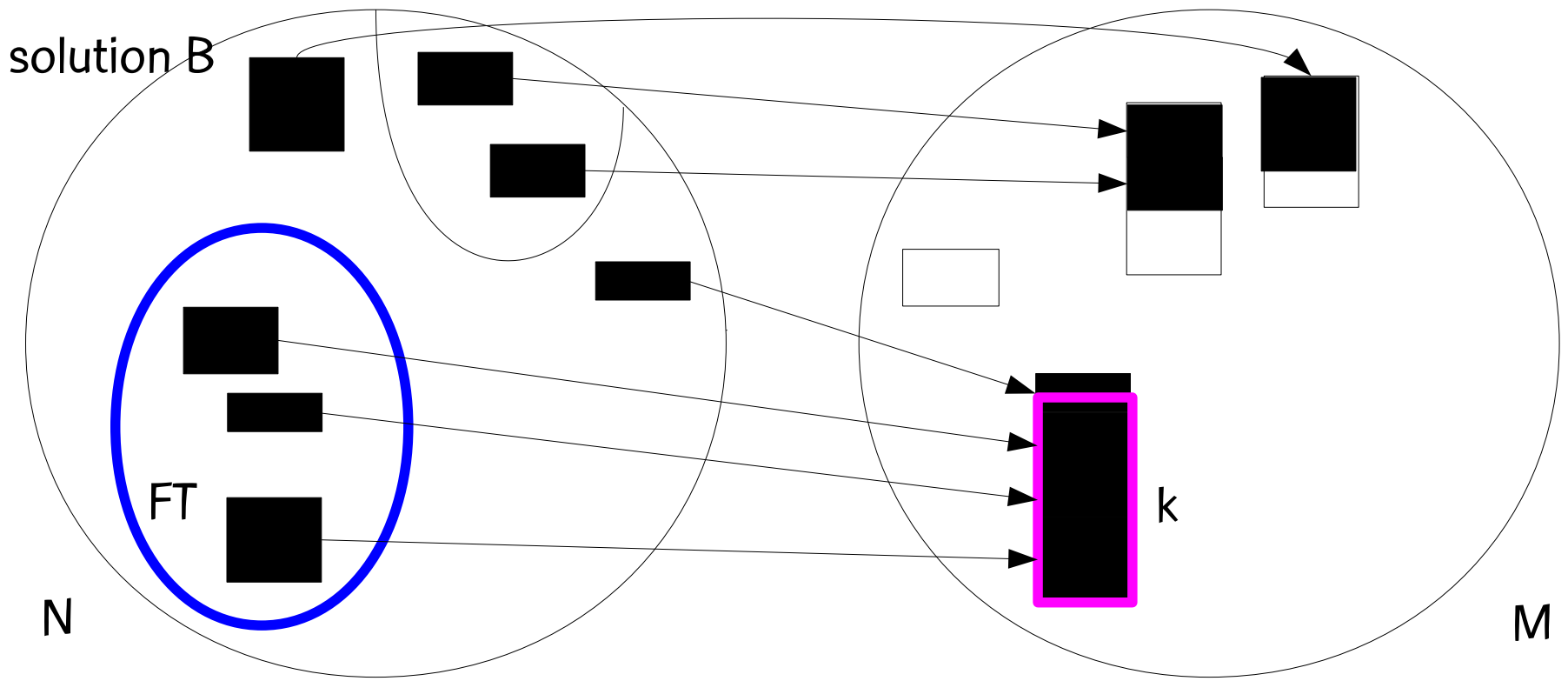
Infeasibility in path-relinking for GQAP



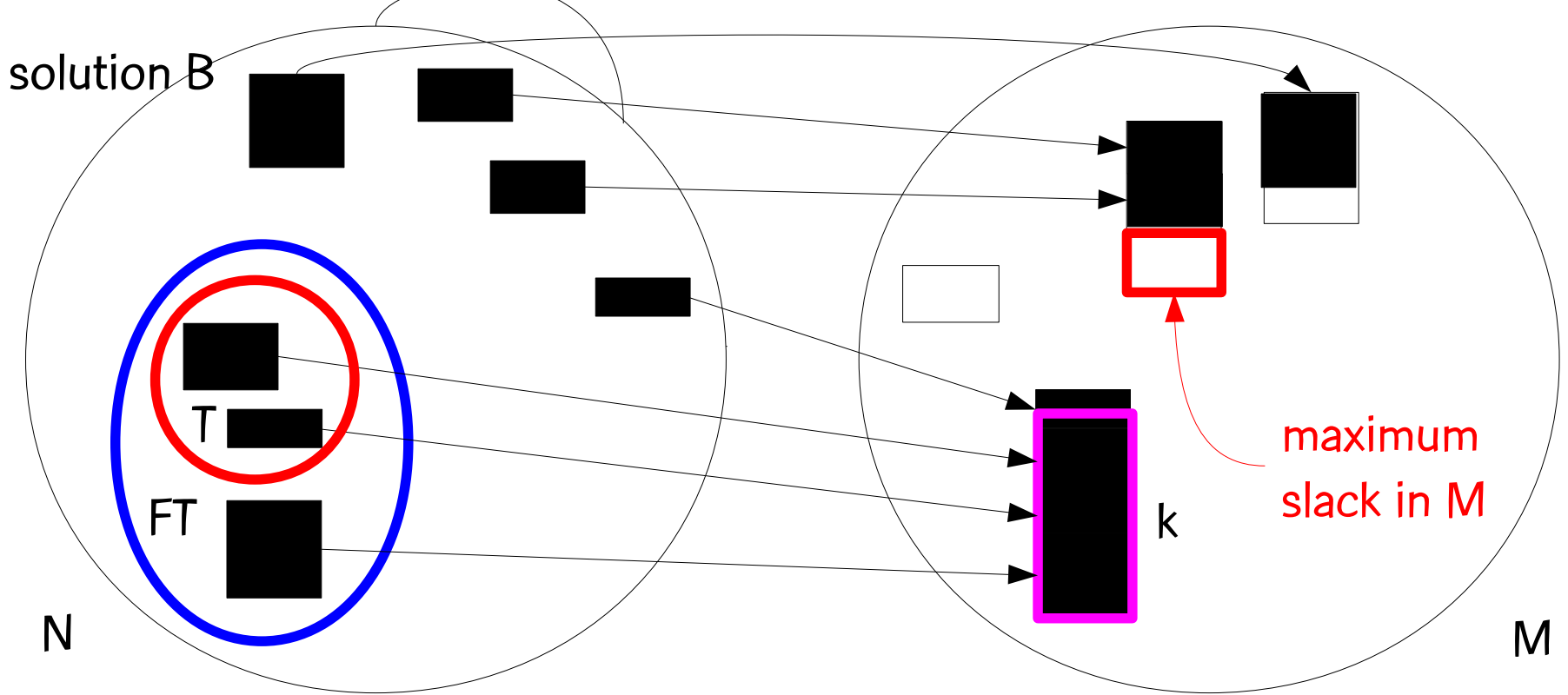


Repair procedure

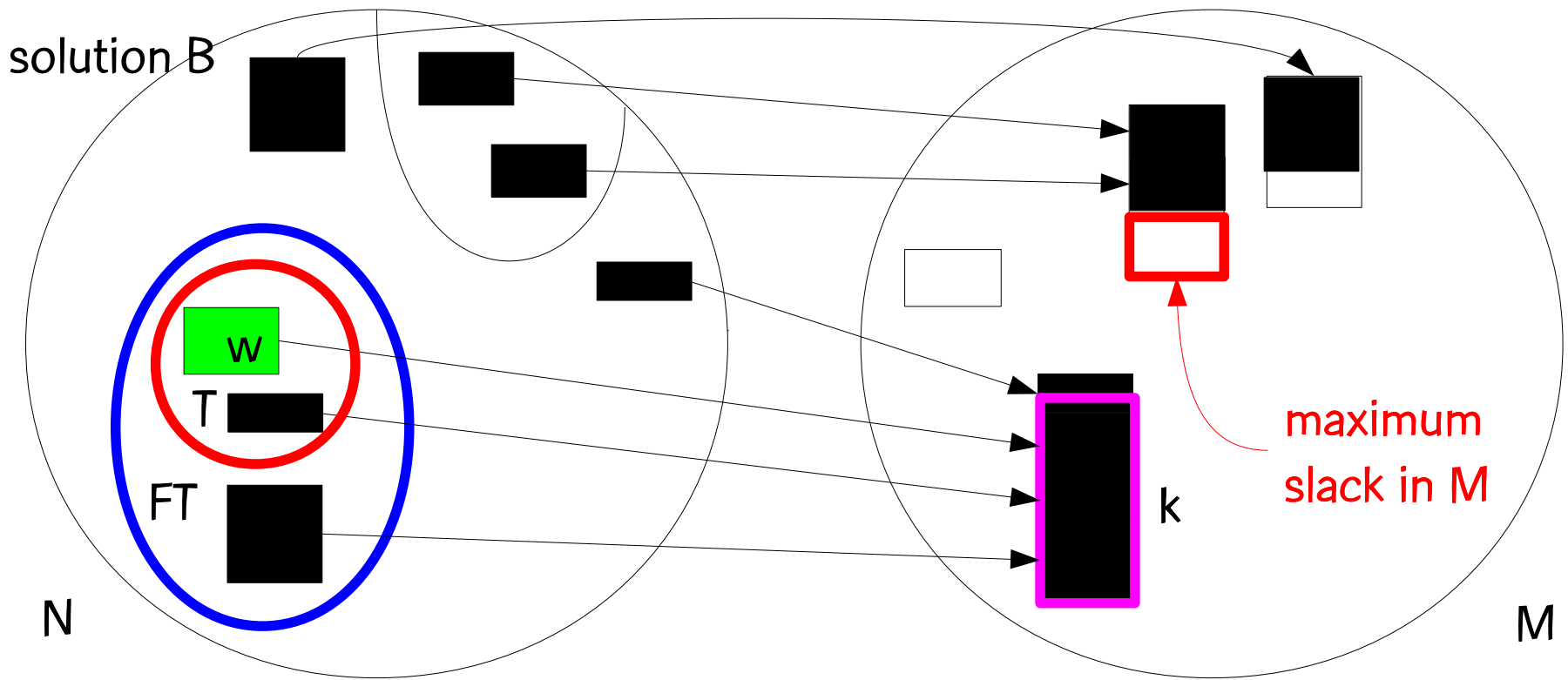




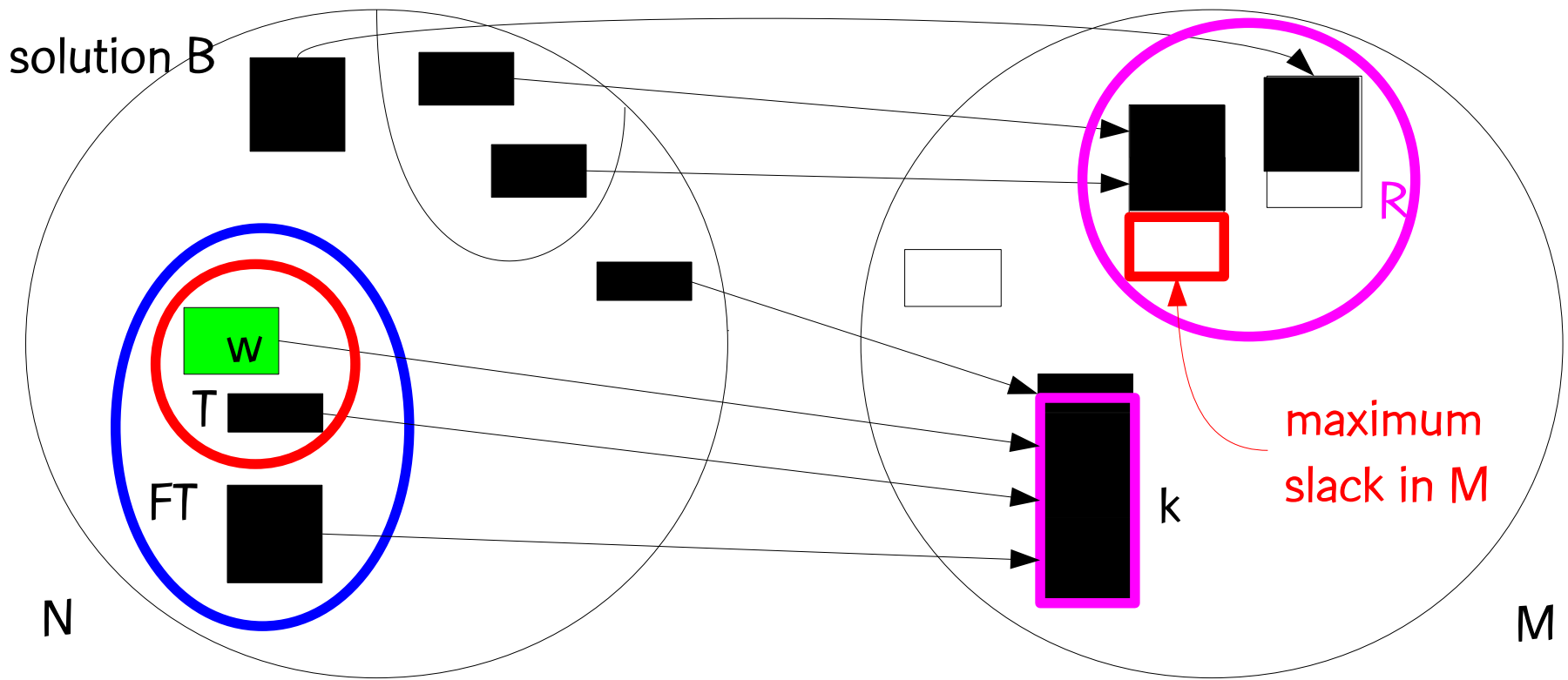
1. Set $FT \subseteq \text{non-Fixed}$: all facilities in solution B assigned to location k



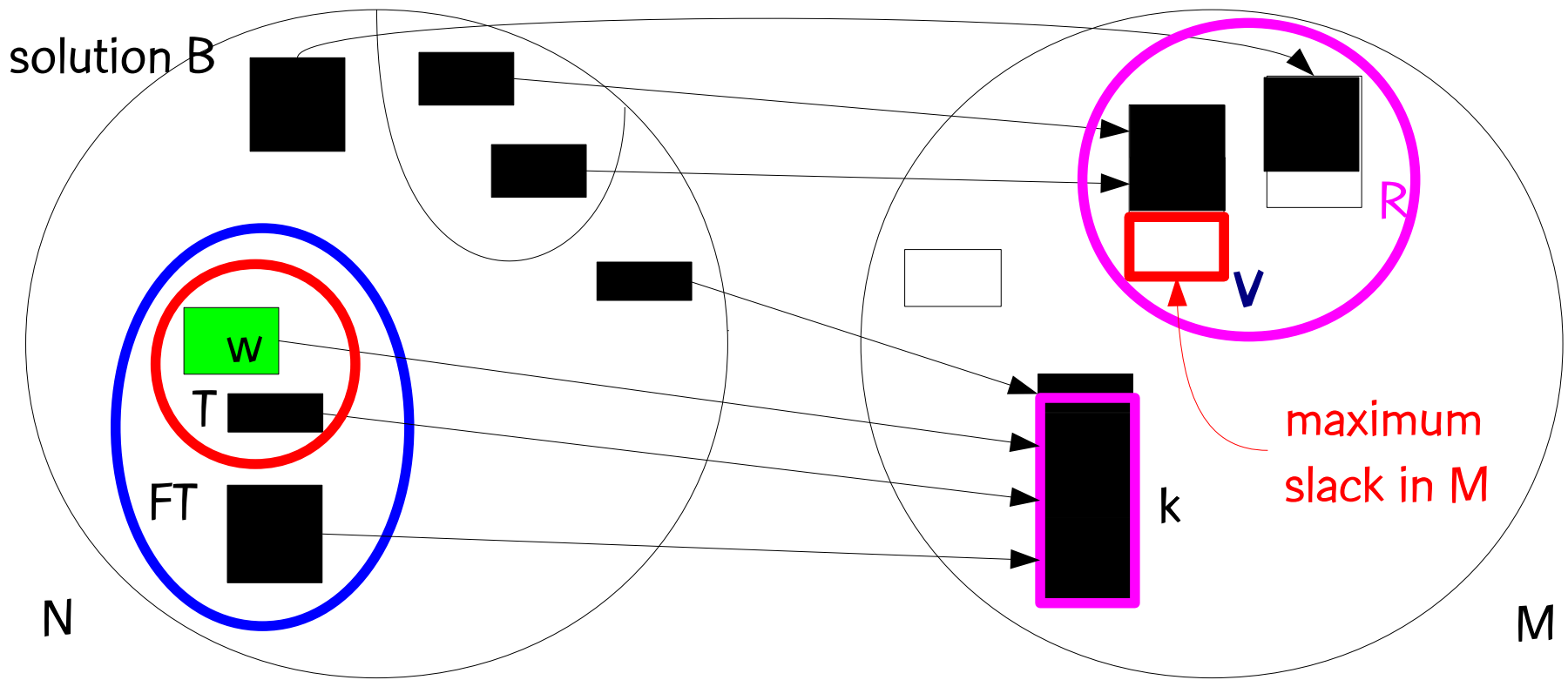
1. Set $FT \subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set $T \subseteq FT$: all facilities in B with demand \leq maximum slack in M



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

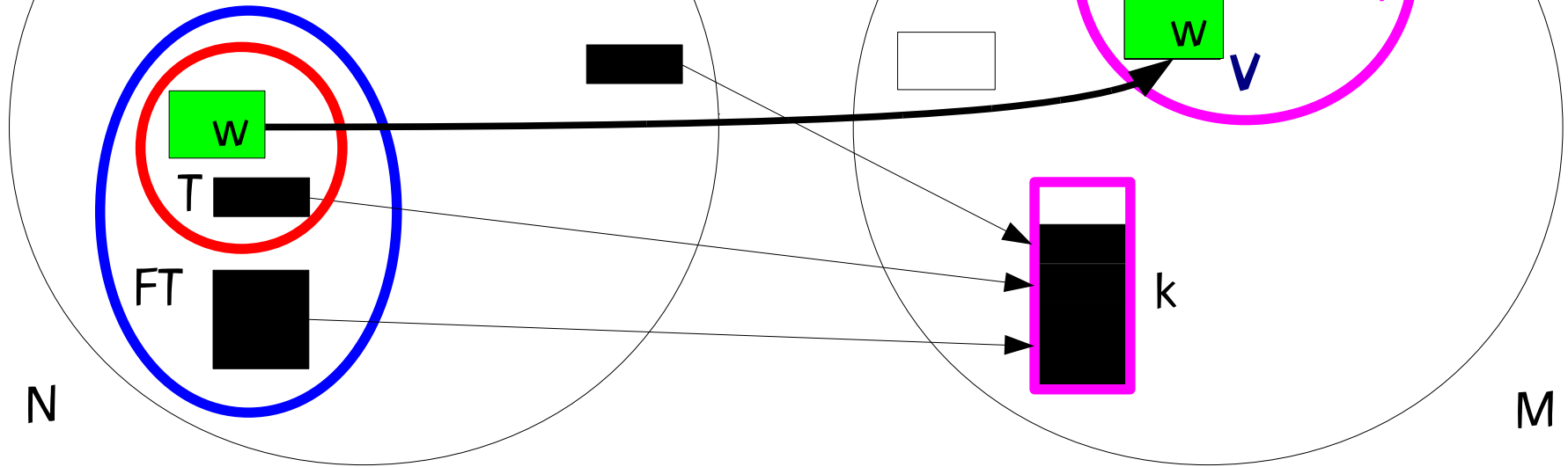


1. Set $FT \subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set $T \subseteq FT$: all facilities in B with demand \leq maximum slack in M
3. Randomly select a facility $w \in T$ favoring those with higher demand
4. Set $R \subseteq M$: all locations having slack \geq demand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

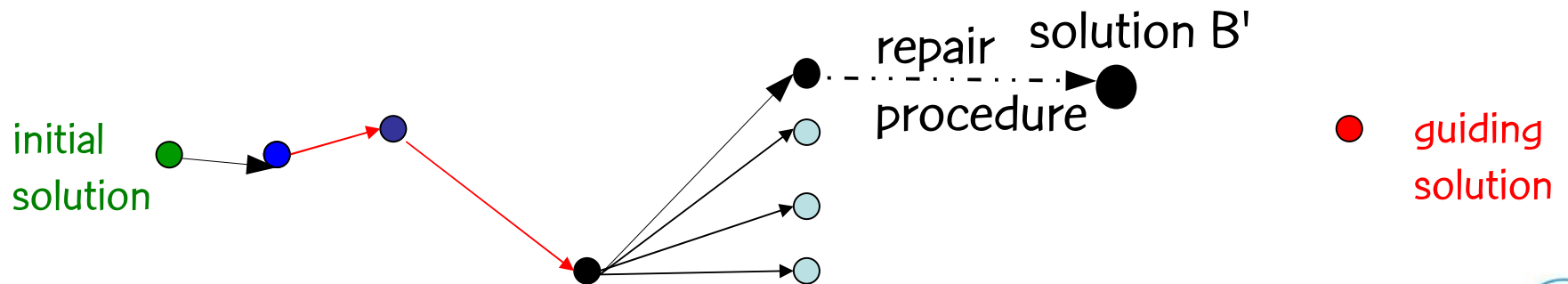
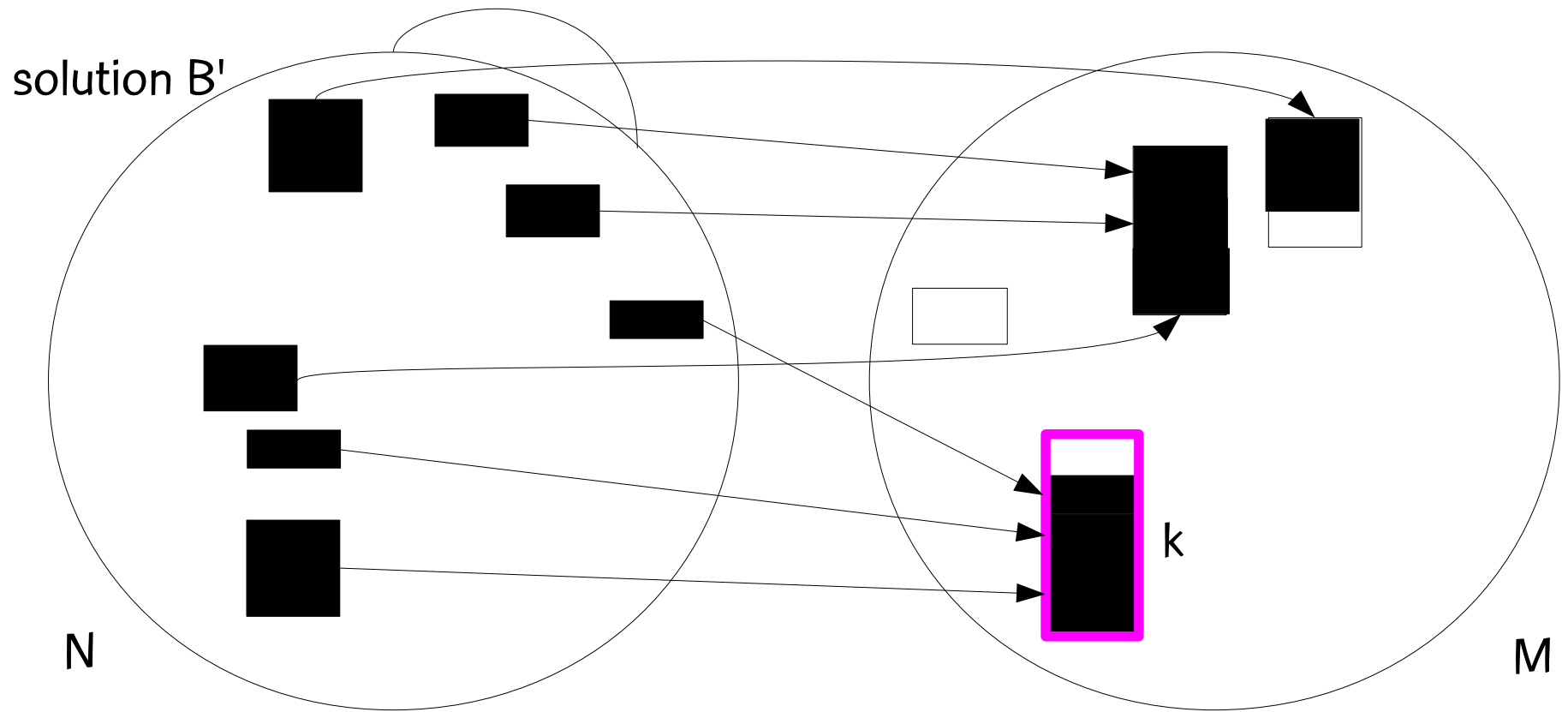


1. Set $FT \subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set $T \subseteq FT$: all facilities in B with demand \leq maximum slack in M
3. Randomly select a facility $w \in T$ favoring those with higher demand
4. Set $R \subseteq M$: all locations having slack \geq demand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

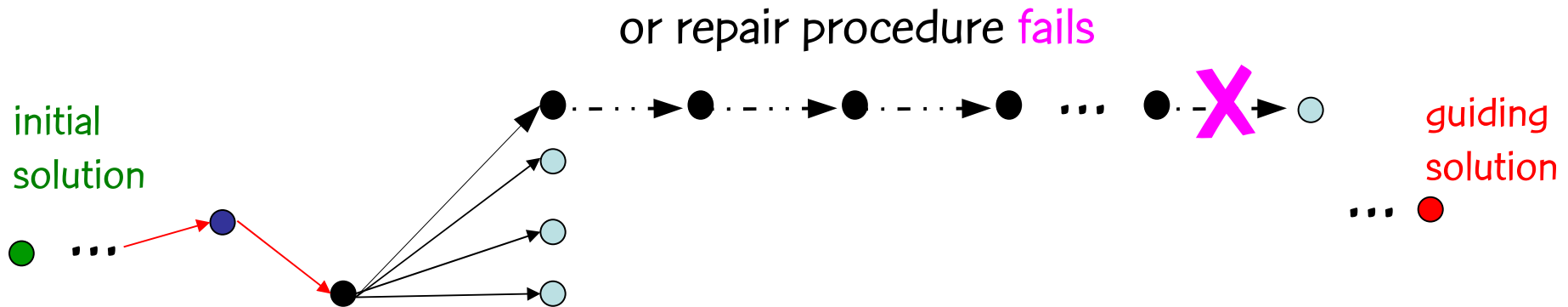
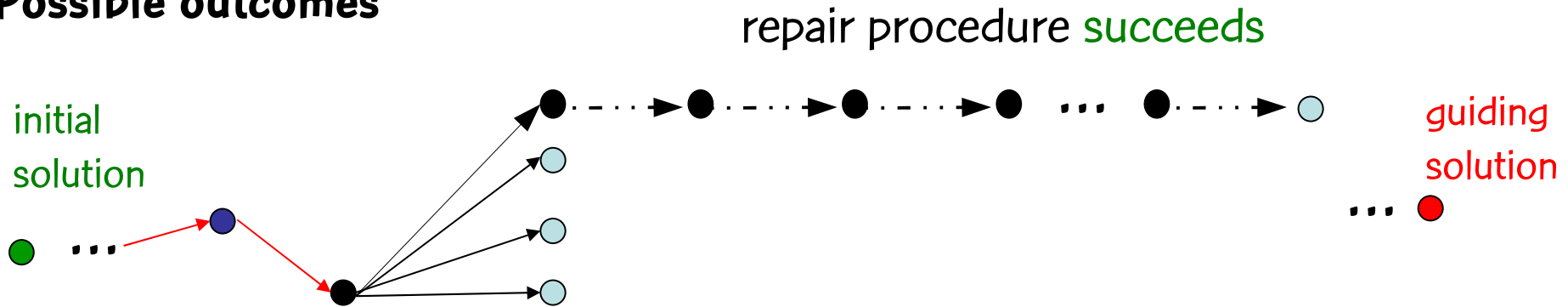
solution B
feasible



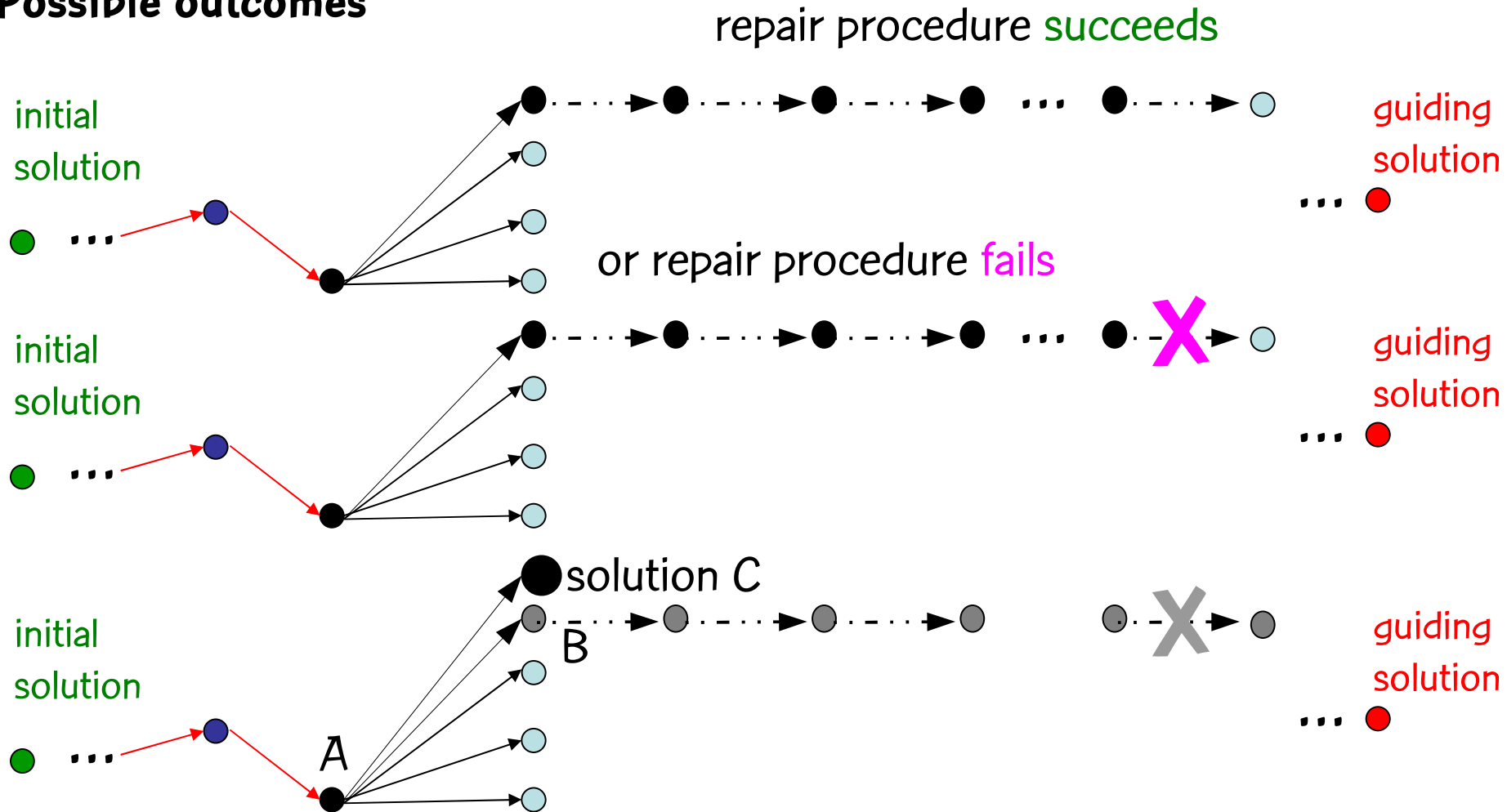
1. Set $FT \subseteq$ non-Fixed: all facilities in solution B assigned to location k
2. Set $T \subseteq FT$: all facilities in B with demand \leq maximum slack in M
3. Randomly select a facility $w \in T$ favoring those with higher demand
4. Set $R \subseteq M$: all locations having slack \geq demand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



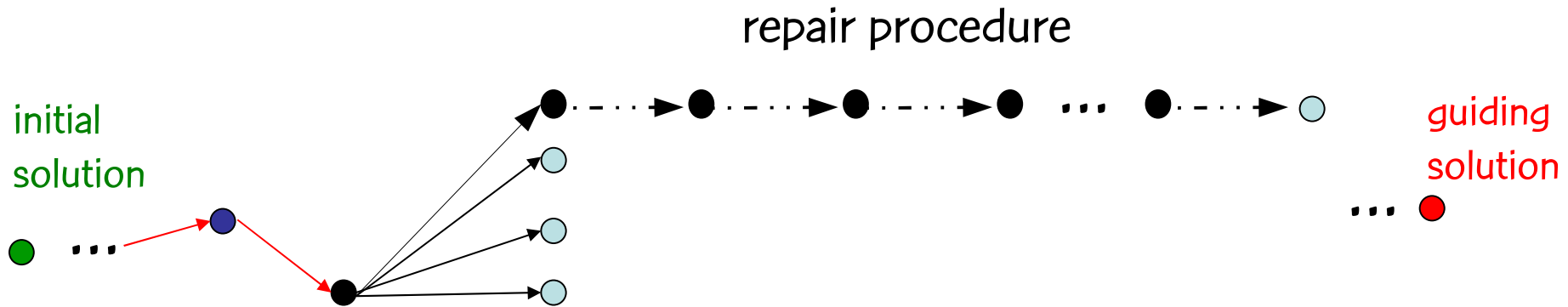
Possible outcomes



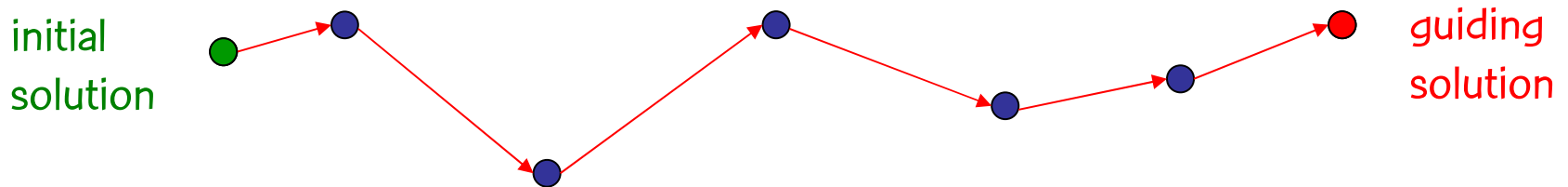
Possible outcomes

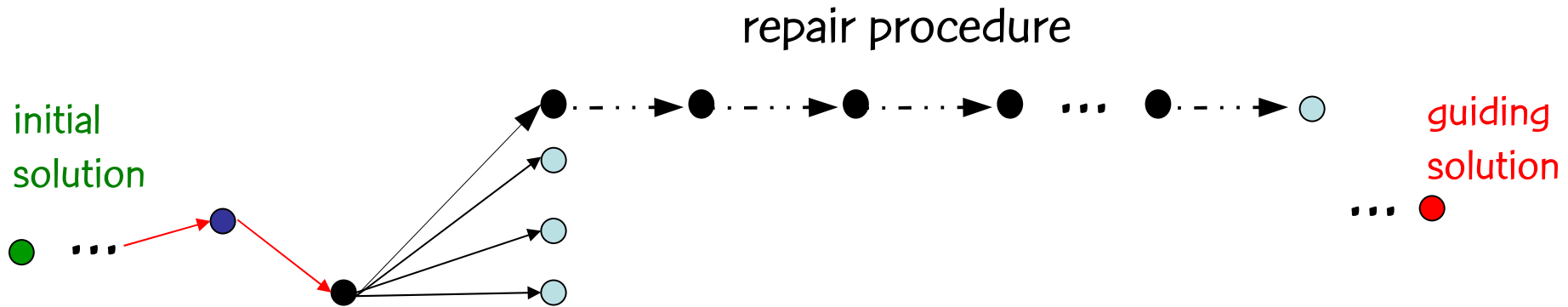


Repeat the repair procedure on solution B a maximum number of times. If a feasible solution is not found, discard B and move to solution C

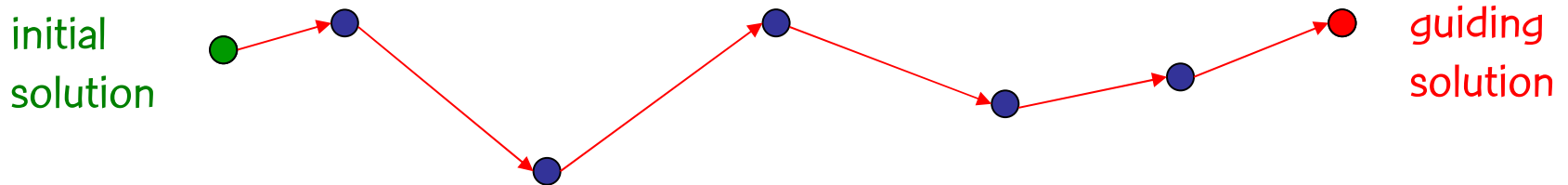


So, instead of a path with feasible solution in one single step ...

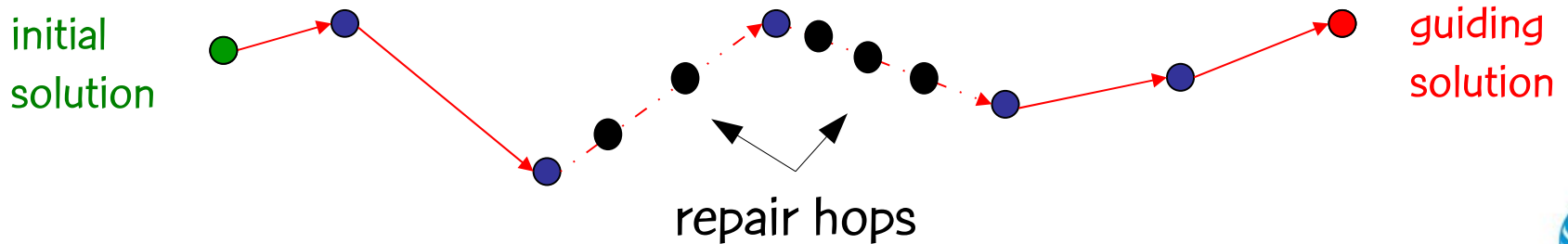




So, instead of a path with feasible solution in one single step ...



We have now a path with eventual intermediate repair hops



Reference

G.R. Mateus, M.G.C.R., and R.M.A. Silva,
“GRASP with path-relinking for the generalized
quadratic assignment problem,” AT&T Labs
Research Technical Report, Florham Park, New
Jersey, Dec. 2008 (revised Apr. 2010)

<http://www.research.att.com/~mgcr/doc/gpr-gqap.pdf>

Two-phase hybrid heuristic

Tuning phase

BRKGA explores the GRASP+PR parameter space.

Encoding: the random-key solution vector x has n $x(i)$ components generated in the real interval $[0,1]$, one for each tunable parameter.

Decoder: if a parameter $i=1, \dots, n$ is in the

a) real interval $[l,u]$, then $x(i)$ is decoded as $l+x(i) \cdot (u-l)$

b) discrete interval $[l,u]$, then $x(i)$ is decoded as

$$\text{ceil}\{ l - 1/2 + x(i) \cdot (u-l) \}$$

Tuning phase

In case of GRASP+PR for GQAP, we have the following tunable user-defined parameters:

- a) 12 parameters from construction procedure;
- b) 6 parameters from aprox. local search procedure;
- c) 8 parameters from path-relinking procedure;
- d) 4 parameters from main procedure;

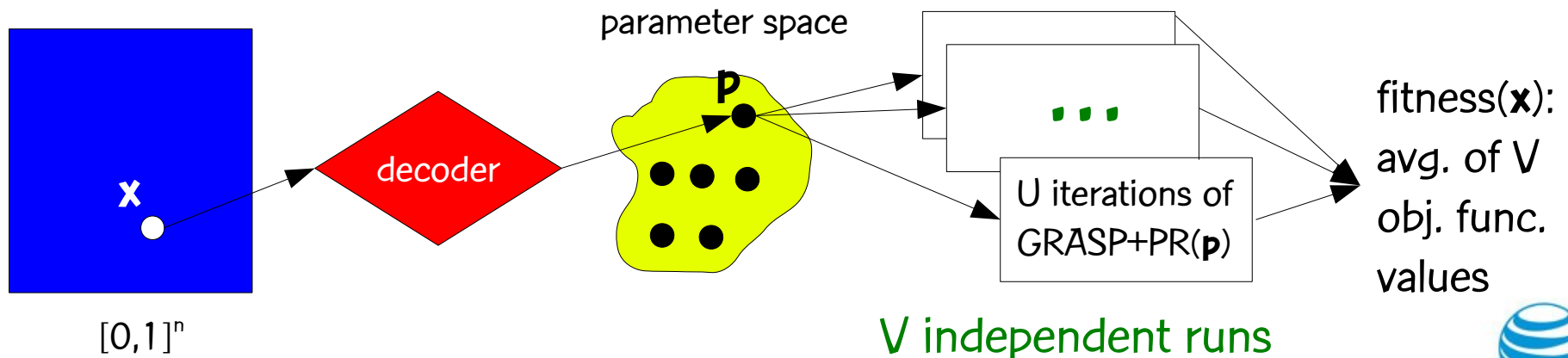
Total of $n=30$ tunable parameters.

Tuning phase

Fitness:

average objective function value of the V independent runs of the GRASP+PR heuristic using the parameters decoded from the solution vector, each run for U GRASP+PR iterations.

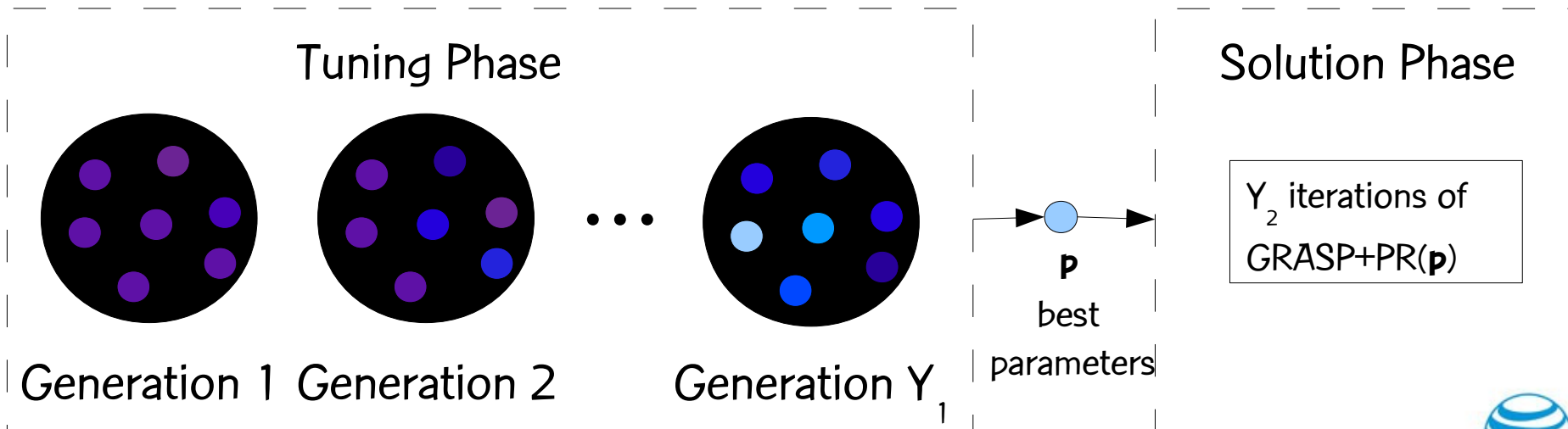
Problem-dependent scheme of tuning phase



Solution phase

Using the best parameters determined after Y_1 BRKGA generations in the tuning phase, GRASP+PR heuristic is run for Y_2 iterations to explore the GQAP solution space, seeking an optimal or near optimal assignment of facilities to locations.

Two-phase hybrid heuristic



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: Five instances from Cordeau et al. (2006): 20-15-35, 20-15-55, 20-15-75, 30-07-75, and 30-08-55.

OBS: Instance **f-l-t** has **f** facilities, **l** locations and **t** controls the tightness of the problem constraints. The higher the value of **t**, the greater the tightness of the constraints, and harder it is to find a feasible solution.

Test environment

Experimental Design:

Tuning phase:

- size of individuals: 30 parameters;
- size of population: 15 individuals;
- size of elite partition: 30% of population;
- size of mutant set: 20% of population;
- child inheritance probability: 0.7;
- stopping criterion: $Y1 = 10$ generations, $V = 30$ independent runs of GRASP+PR heuristic, each one for $U = 100$ iterations.

Test environment

Experimental Design:

Solution phase:

GRASP+PR found the best known value on all 200 runs for each of the five instances from Cordeau et al. (2006): 20-15-35, 20-15-55, 20-15-75, 30-07-75, and 30-08-55.

Test environment

Statistics:

Minimum, maximum, average times, and standard deviation.

Comparison of a GRASP+PR for GQAP with manually and automatically tuned parameters

Problem	Manually tuned (times in seconds)				Automatically tuned (times in seconds)			
	min	max	avg	sdev	min	max	avg	sdev
20-15-35	1.16	845.29	147.09	146.53	0.59	71.30	9.62	9.19
20-15-55	0.63	83.52	17.04	16.43	0.36	33.03	7.17	6.15
20-15-75	0.92	166.30	8.47	14.04	0.78	552.19	47.55	82.88
30-08-55	0.35	11.67	2.26	1.54	0.07	3.42	0.96	0.61
30-07-75	9.22	26914.03	716.08	2027.75	1.27	228.01	28.63	29.75

Problem	Manually tuned				Automatically tuned			
	min	max	avg	sdev	min	max	avg	sdev
20-15-35	1.16	845.29	147.09	146.53	0.59	71.30	9.62	9.19
20-15-55	0.63	83.52	17.04	16.43	0.36	33.03	7.17	6.15
20-15-75	0.92	166.30	8.47	14.04	0.78	552.19	47.55	82.88
30-08-55	0.35	11.67	2.26	1.54	0.07	3.42	0.96	0.61
30-07-75	9.22	26914.03	716.08	2027.75	1.27	228.01	28.63	29.75

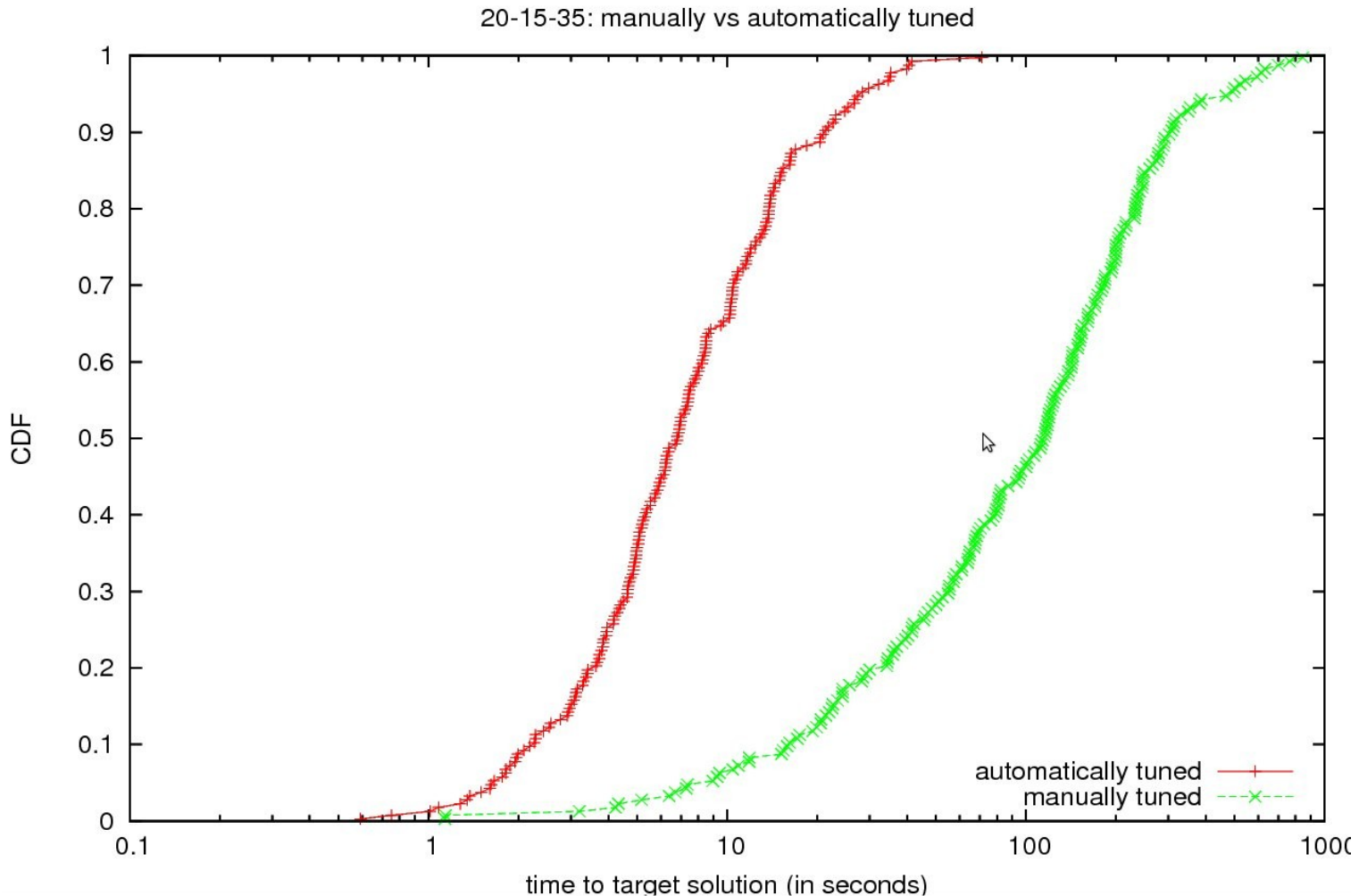
On all instances, except 20-15-75, the automatically tuned variant proved to find the best known solution in **less time** than the manually variant



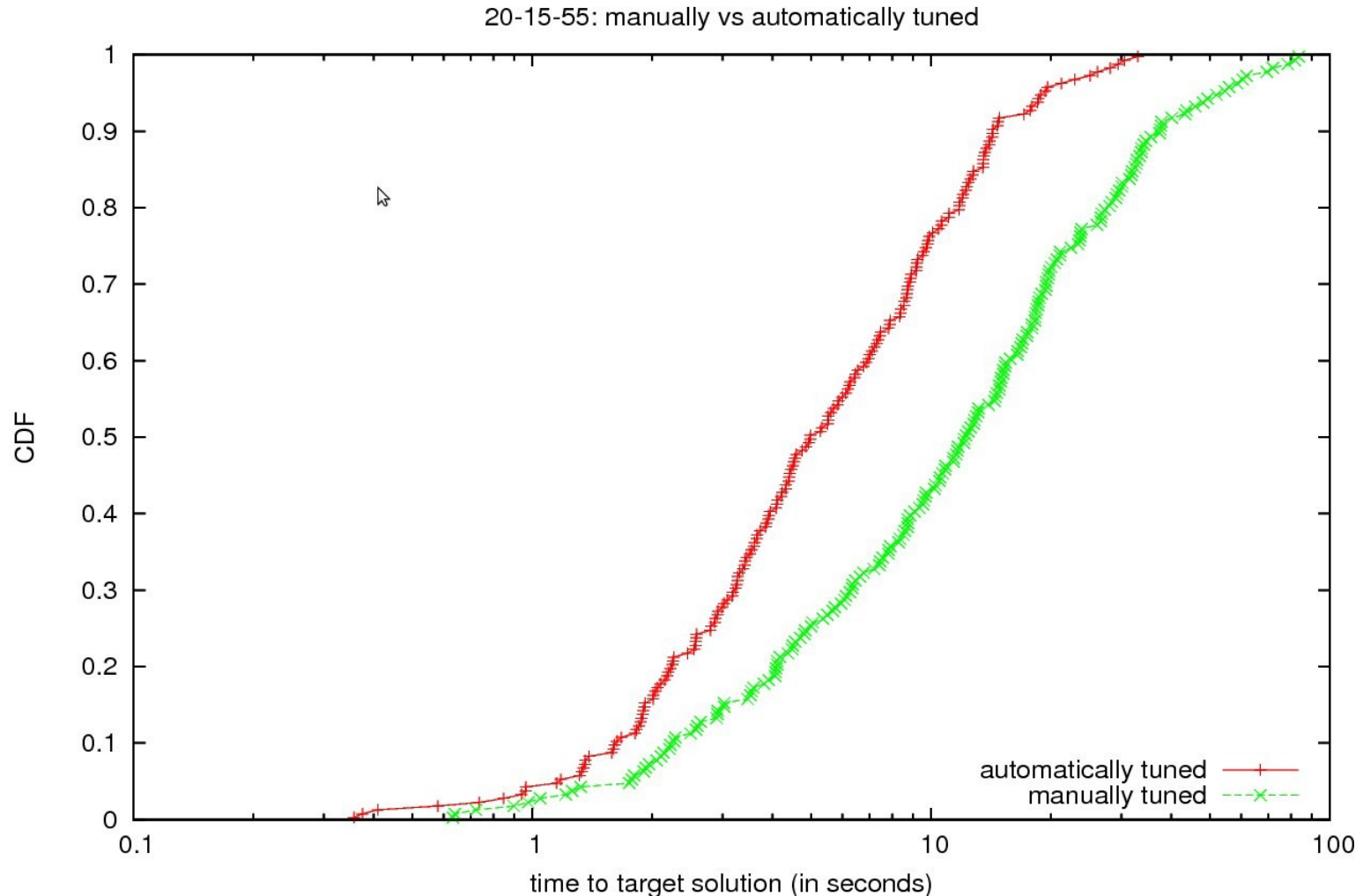
Problem	Manually tuned				Automatically tuned			
	min	max	avg	sdev	min	max	avg	sdev
20-15-35	1.16	845.29	147.09	146.53	0.59	71.30	9.62	9.19
20-15-55	0.63	83.52	17.04	16.43	0.36	33.03	7.17	6.15
20-15-75	0.92	166.30	8.47	14.04	0.78	552.19	47.55	82.88
30-08-55	0.35	11.67	2.26	1.54	0.07	3.42	0.96	0.61
30-07-75	9.22	26914.03	716.08	2027.75	1.27	228.01	28.63	29.75

In the most difficult instance (30-07-75), the automatically tuned variant was on average about 25 times faster than the manually tuned variant. The ratio of maximum running times on this instance was over 118, in favor of the automatically tuned variant.

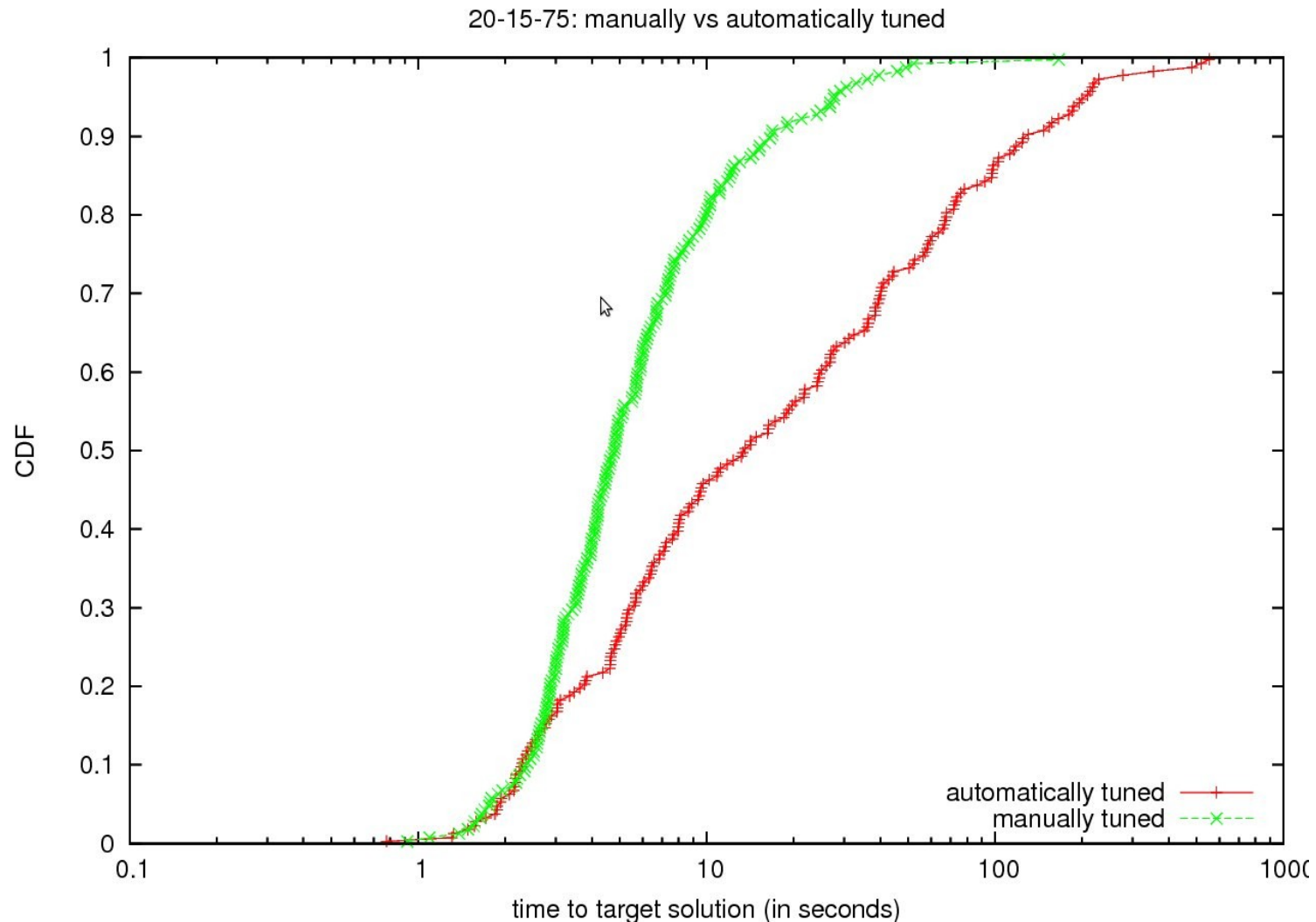
Runtime distributions for manually and automatically tuned GRASP+PR heuristics for the GQAP on instance 20-15-35



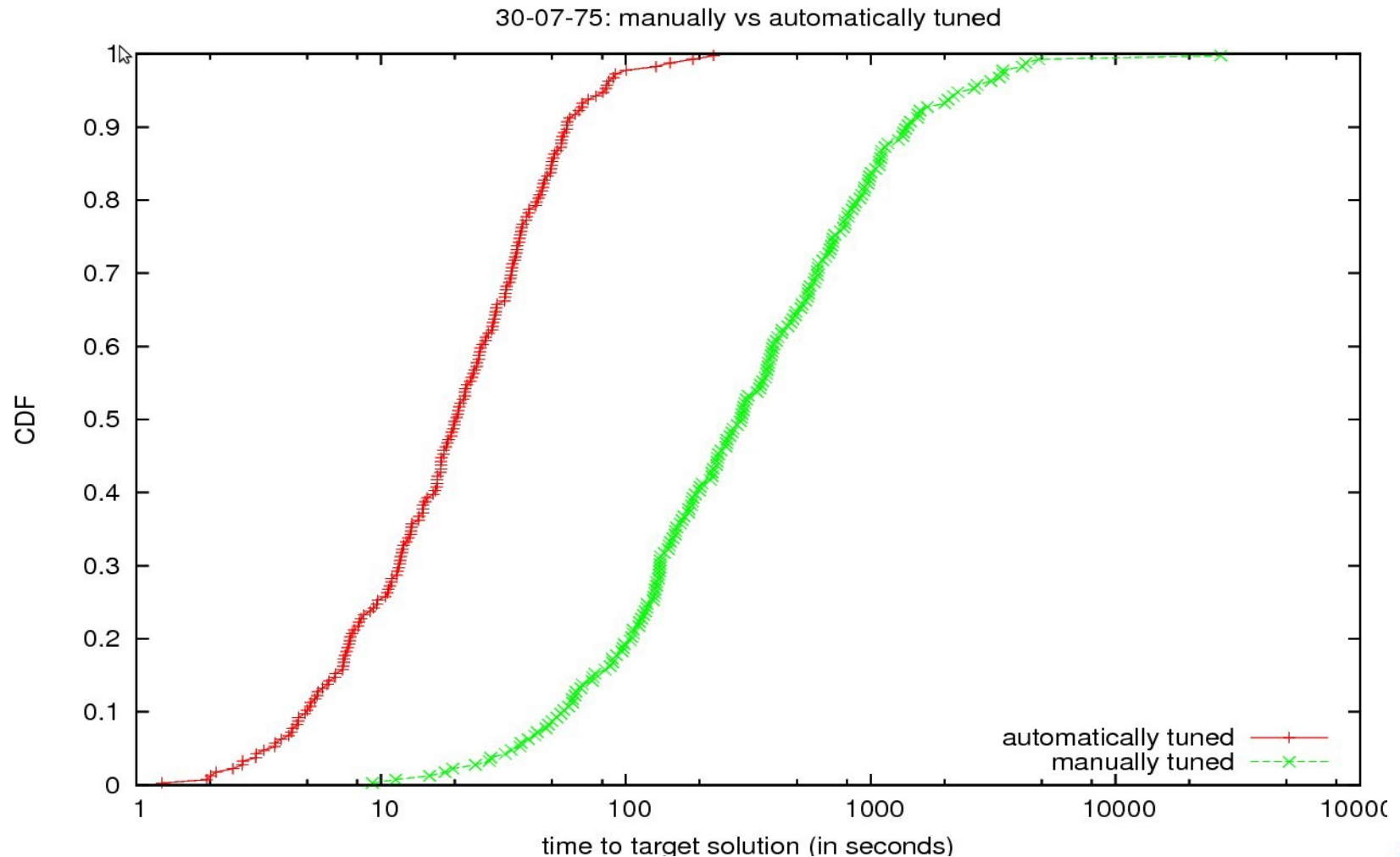
Runtime distributions for manually and automatically tuned GRASP+PR heuristics for the GQAP on instance 20-15-55



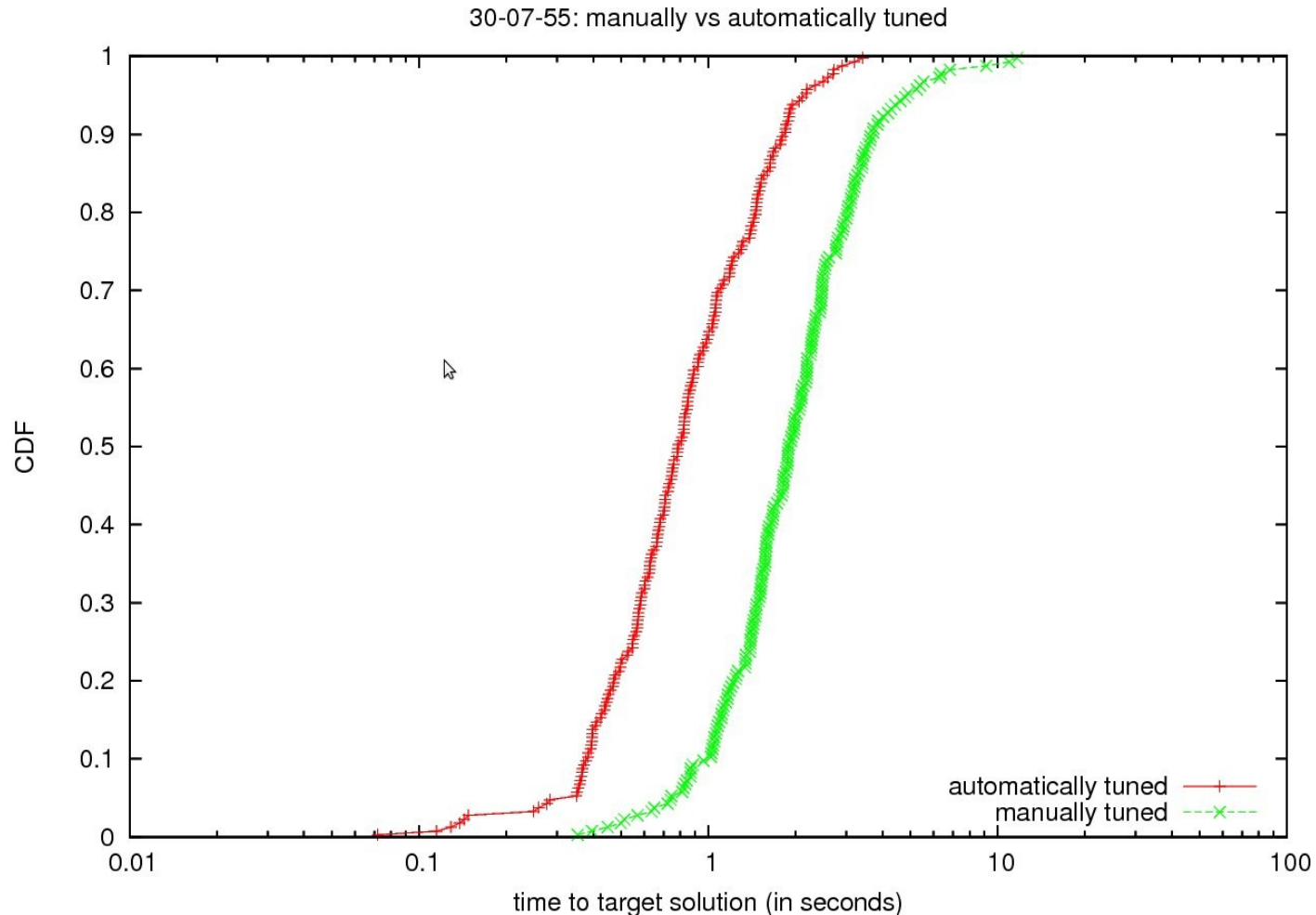
Runtime distributions for manually and automatically tuned GRASP+PR heuristics for the GQAP on instance 20-15-75



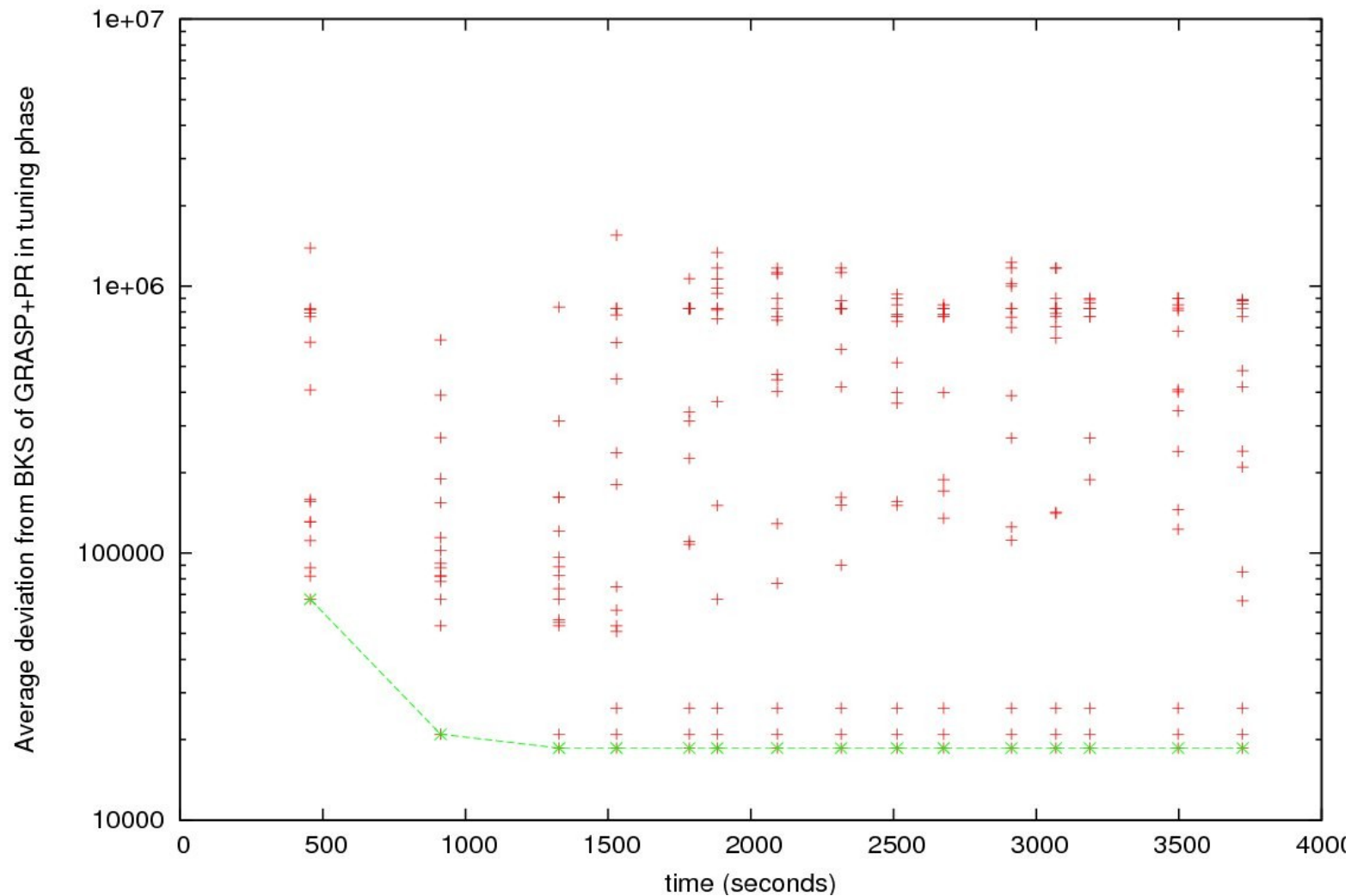
Runtime distributions for manually and automatically tuned GRASP+PR heuristics for the GQAP on instance 30-07-75



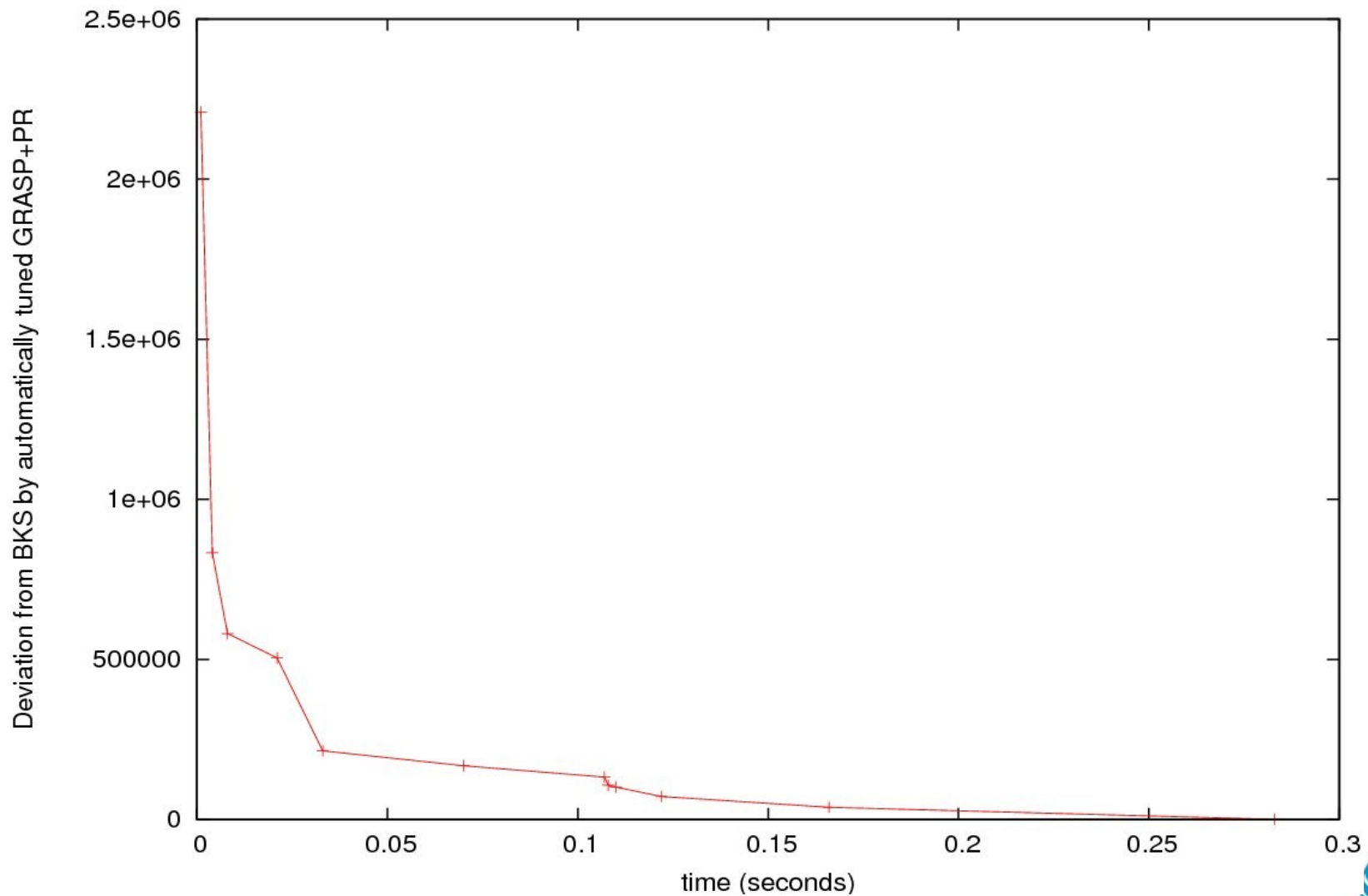
Runtime distributions for manually and automatically tuned GRASP+PR heuristics for the GQAP on instance 30-08-55



Average deviation from BKS of GRASP+PR in tuning phase



Deviation from BKS by automatically tuned GRASP+PR



Considerations

- Times in Table and Figures are limited to GRASP+PR and do not include the time taken by the BRKGA to automatically tune the parameters.
- Tuning times were, respectively, 10,739.2, 7,551.2, 3,690.3, 21,909.1, and 14,386.5 seconds for instances 20-15-35, 20-15-55, 20-15-75, 30-07-75, and 30-08-55.

Considerations

- These times could be reduced considerably with a parallel implementation of the BRKGA as well as with the imposition of a maximum running time for the GRASP+PR heuristic run in the process of computing the fitness of the parameter settings.

Considerations

- Poor settings often lead to configurations that struggle to find feasible assignments, thus leading to long running times.
- On the other hand, the times for the manually tuned heuristic do not reflect the weeks that it took for us to do the manual tuning.

Concluding remarks

Concluding remarks

- We have studied a new two-phase automatic parameter tuning procedure for GRASP+PR heuristics based on a biased random-key genetic algorithm.
- The robustness of the procedure was illustrated with a GRASP+PR for the generalized quadratic assignment problem (GQAP) with $n=30$ tunable parameters on five difficult GQAP instances from Cordeau et al. (2006).
- In the near future, we plan to apply this automatic tuning procedure on GRASP with path-relinking heuristics for other NP-hard problems.

The End

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

<http://www.research.att.com/~mgcr>