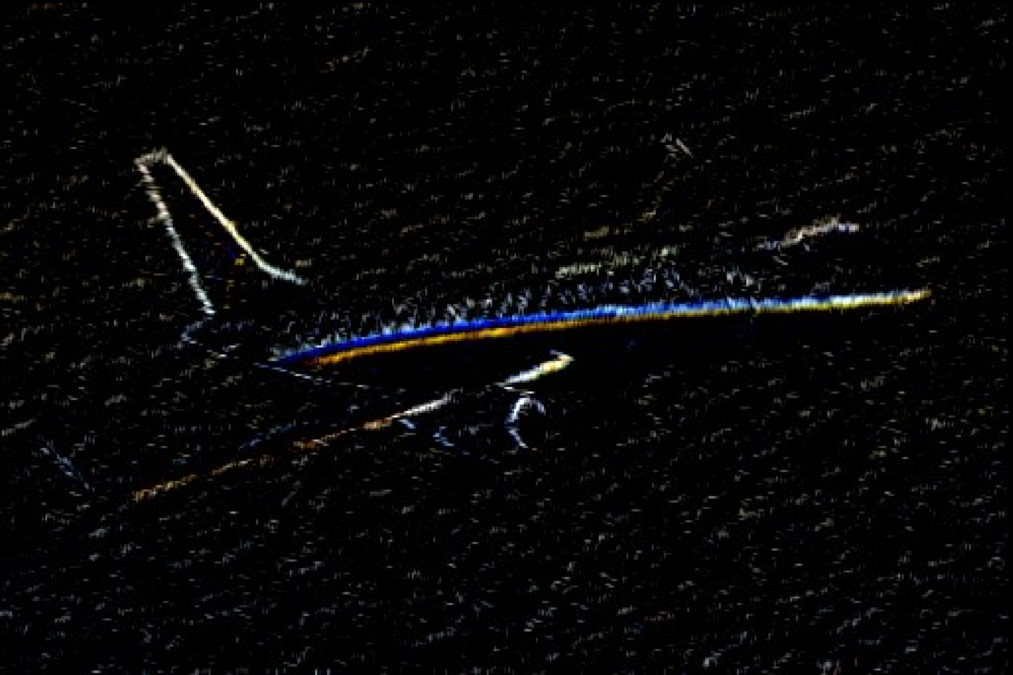
Biased random-key genetic algorithms with applications to optimization problems in telecommunications

Talk given at U. Fed. de São Paulo (UNIFESP)
São José dos Campos (SP) Brazil ✤ March 27, 2013



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

mgcr@research.att.com

Summary

- Biased random-key genetic algorithms
- Three applications in telecommunications
  - Routing in IP networks
  - Design of survivable IP networks with composite links
  - Redundant server location for content distribution
- Concluding remarks

Reference

M.G.C.R., "Biased random-key genetic algorithms with applications in telecommunications," TOP, vol. 20, pp. 120-153, 2012.

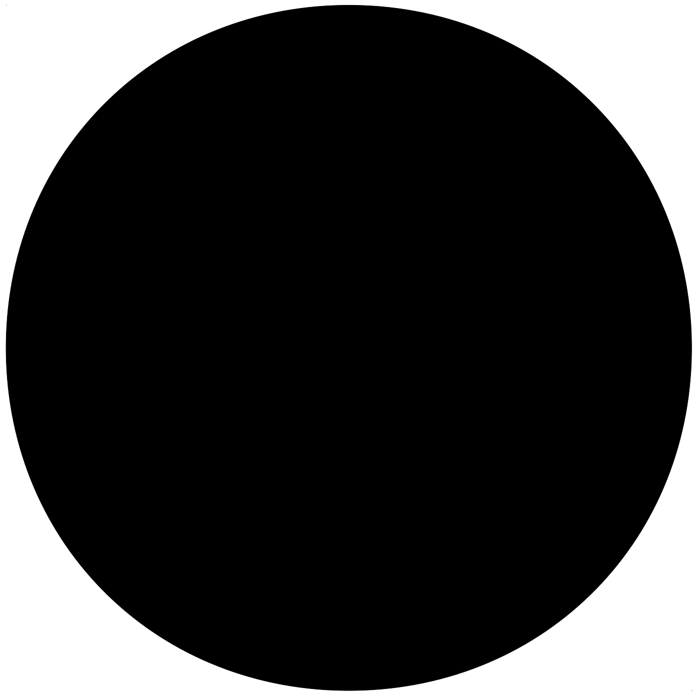
Tech report version:

http://www2.research.att.com/~mgcr/doc/brkga-telecom.pdf

Biased random-key genetic algorithms

Genetic algorithms

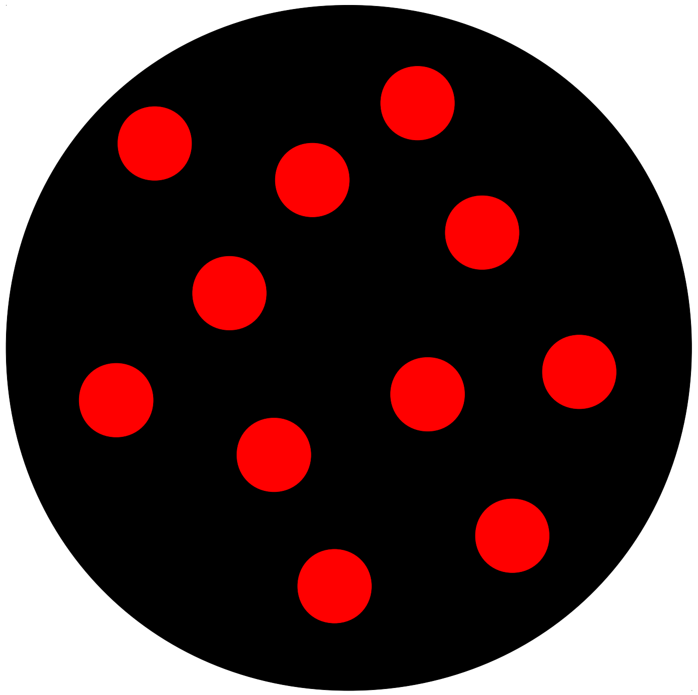
Holland (1975)



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

Individual: solution ●

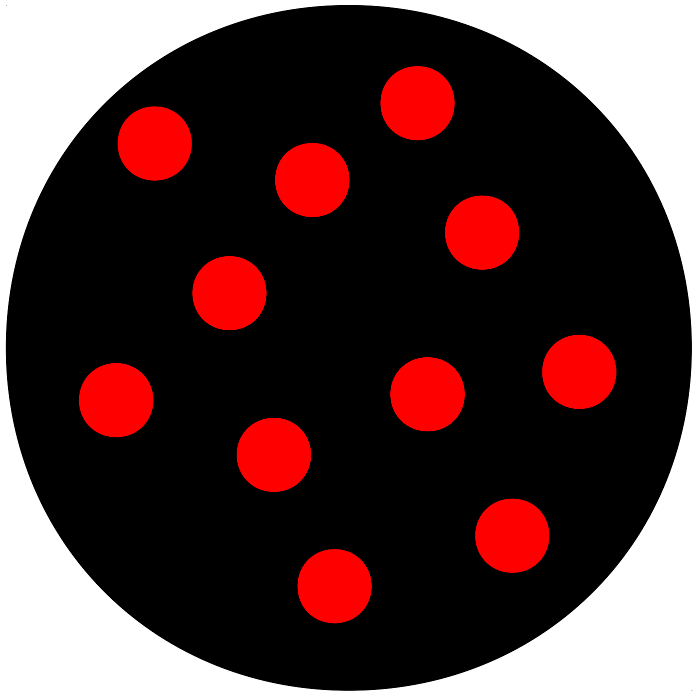
Genetic algorithms



Individual: solution (chromosome = string of genes)

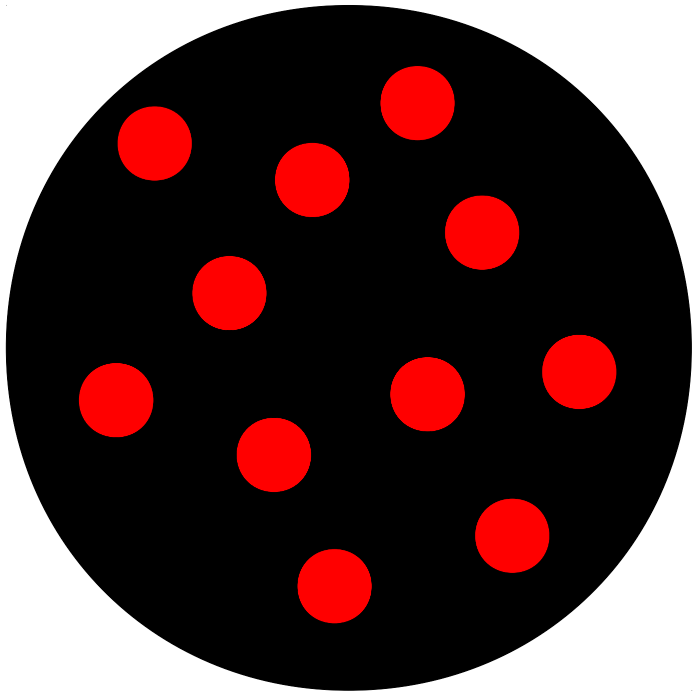
Population: set of fixed number of individuals

Genetic algorithms



Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

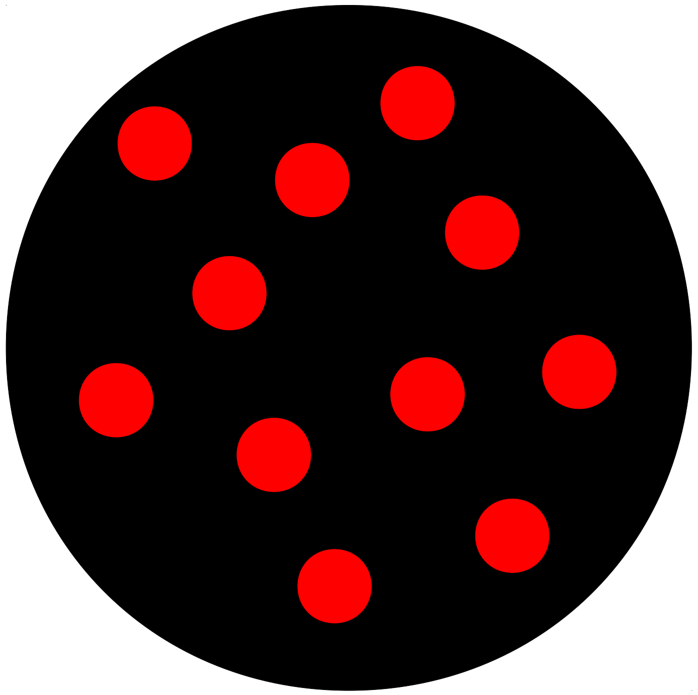
Genetic algorithms



Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

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

Genetic algorithms

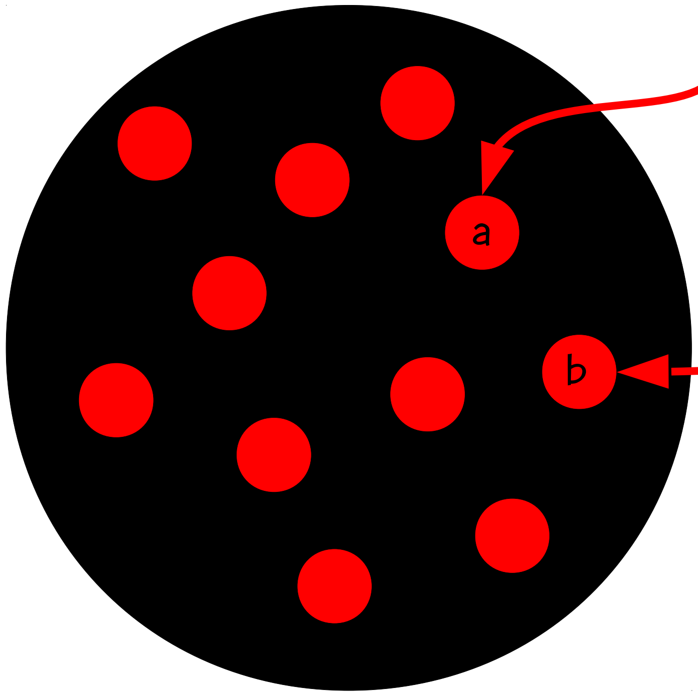


Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

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

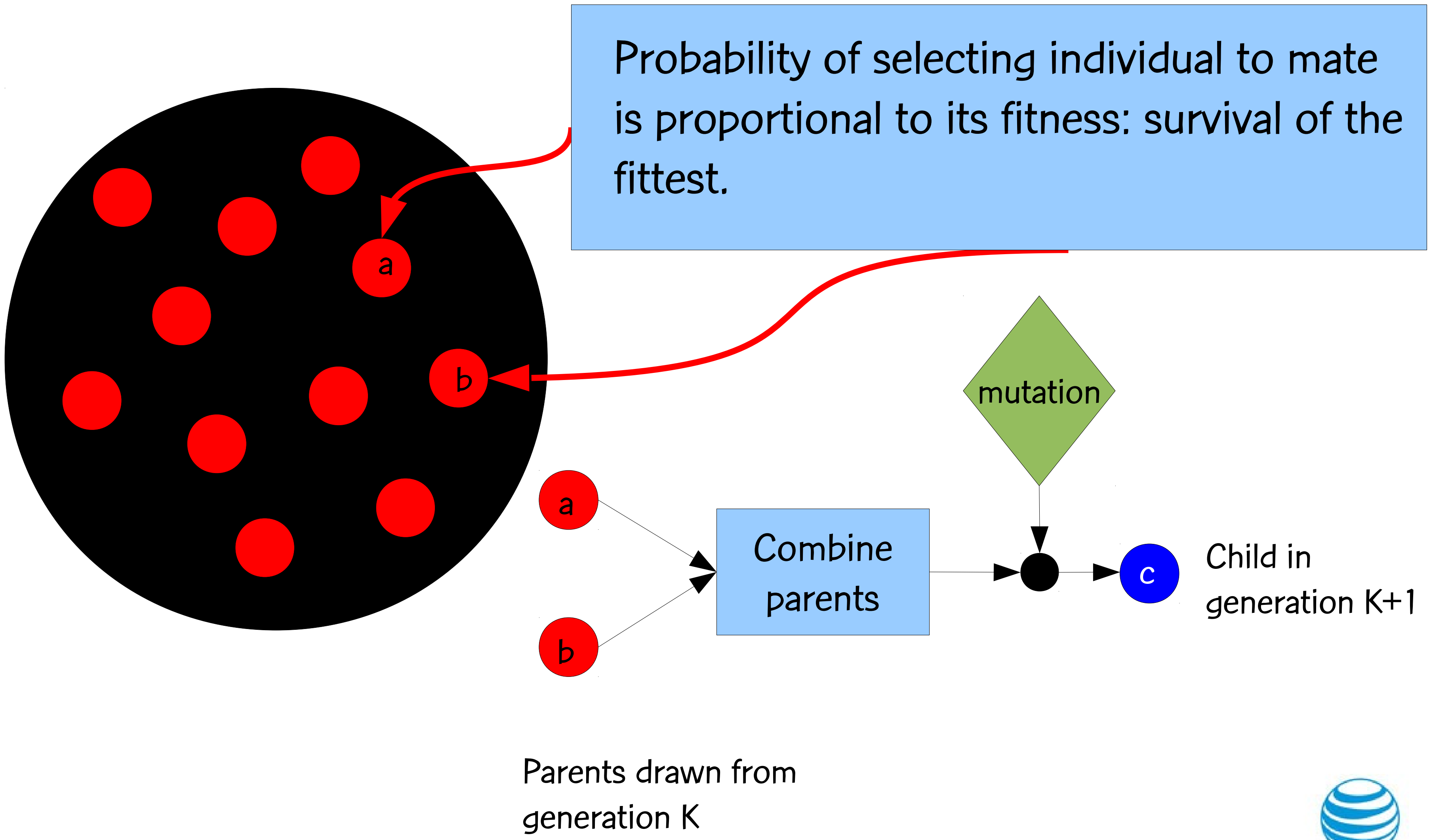
Individuals from one generation are combined to produce offspring that make up next generation.

Genetic algorithms

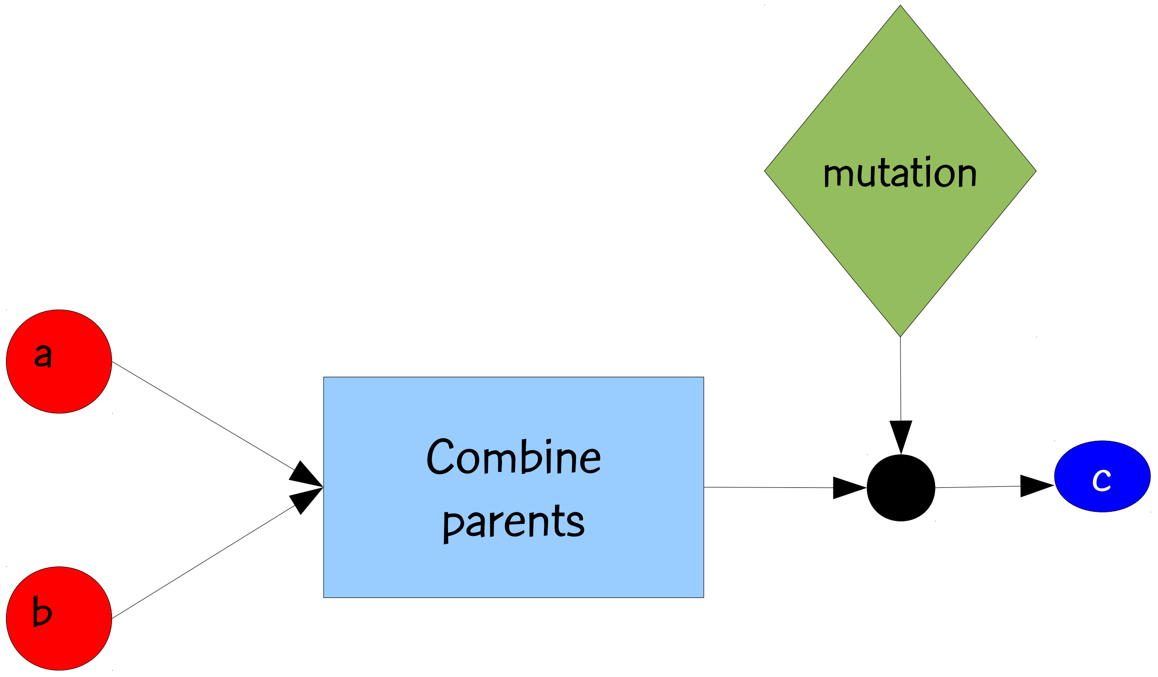


Probability of selecting individual to mate is proportional to its fitness: survival of the fittest.

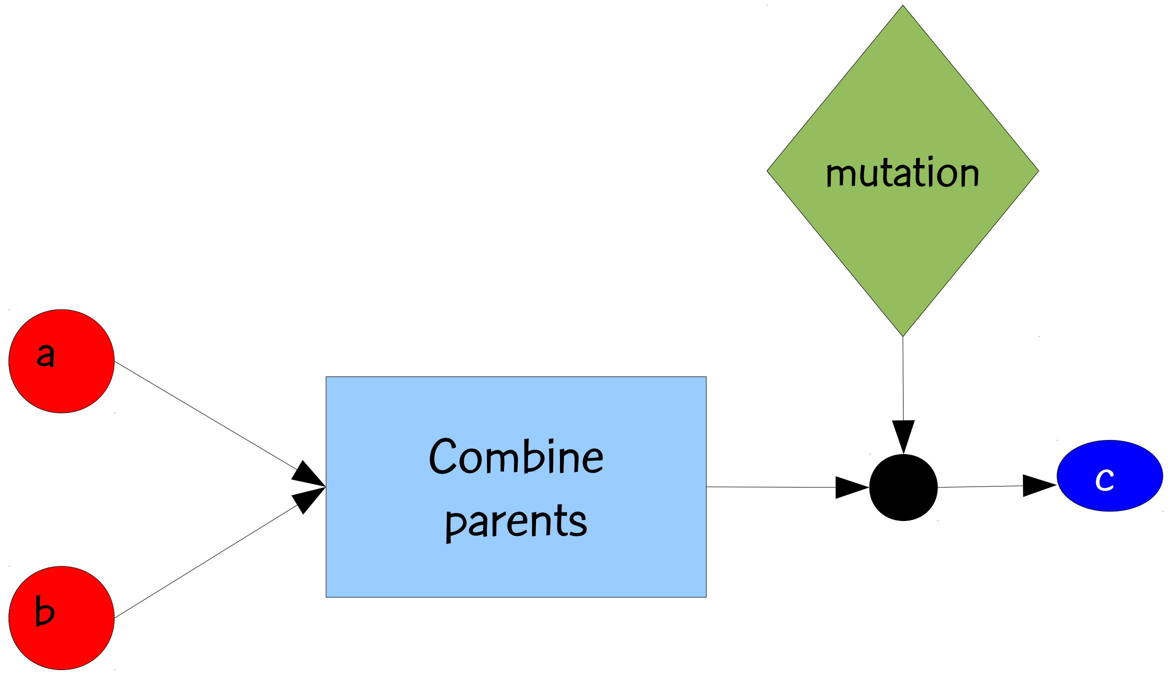
Genetic algorithms


Probability of selecting individual to mate is proportional to its fitness: survival of the fittest.
a
b
a
b
Combine parents
mutation
c
Child in generation K+1
Parents drawn from generation K

Crossover and mutation



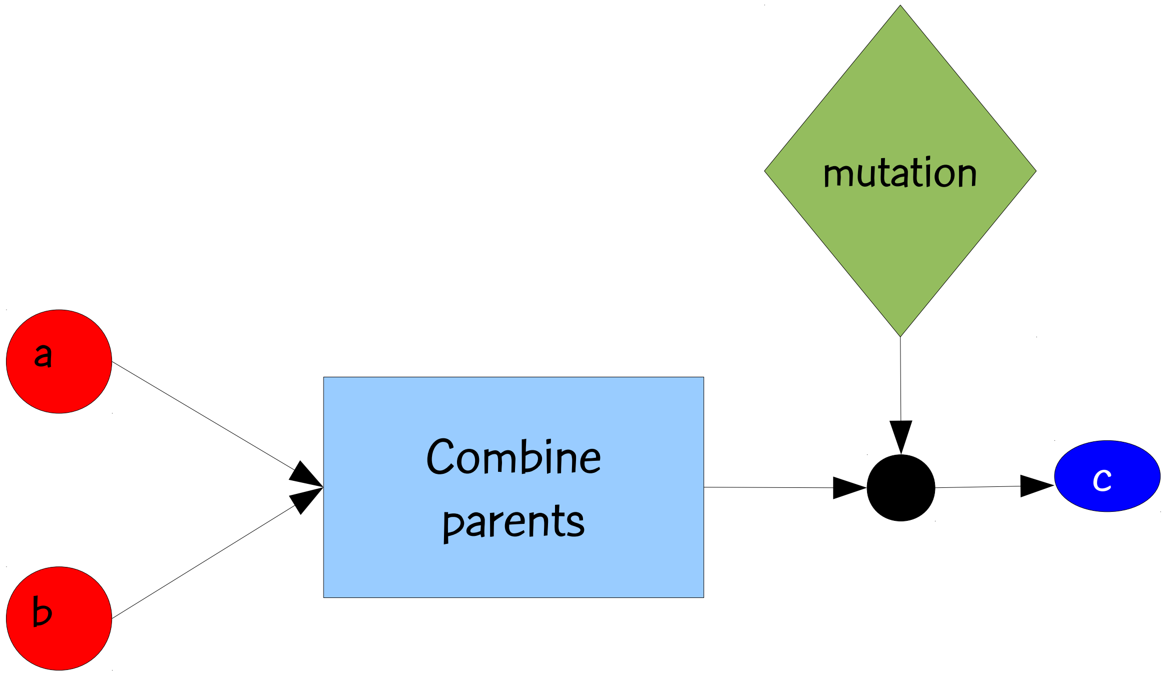
Crossover and mutation



Crossover: Combines parents ... passing along to offspring characteristics of each parent ...

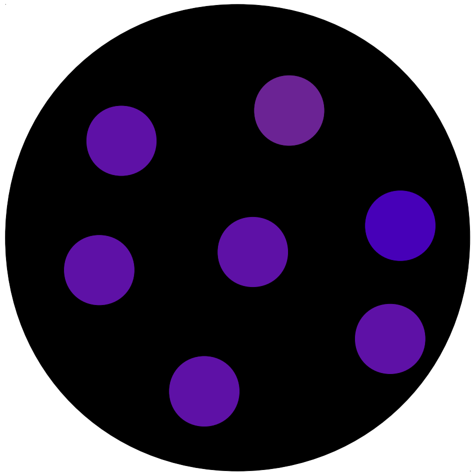
Intensification of search

Crossover and mutation

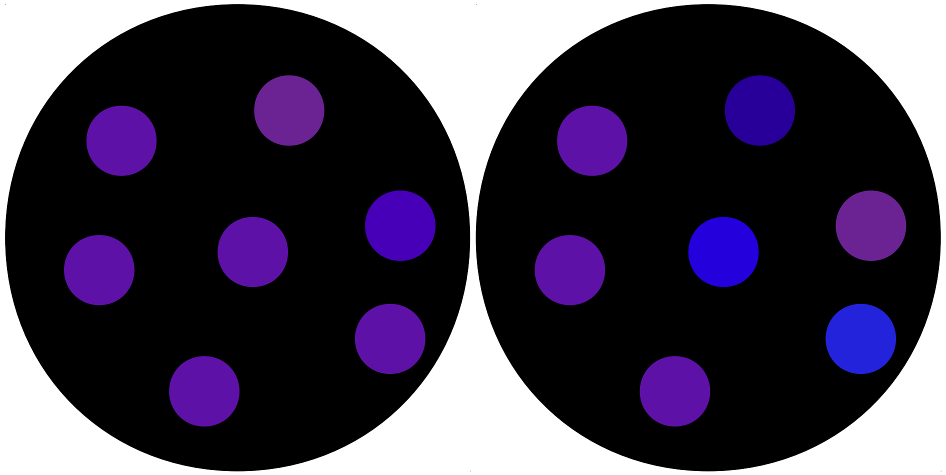


Mutation: Randomly changes chromosome of offspring ...
Driver of evolutionary process ...
Diversification of search

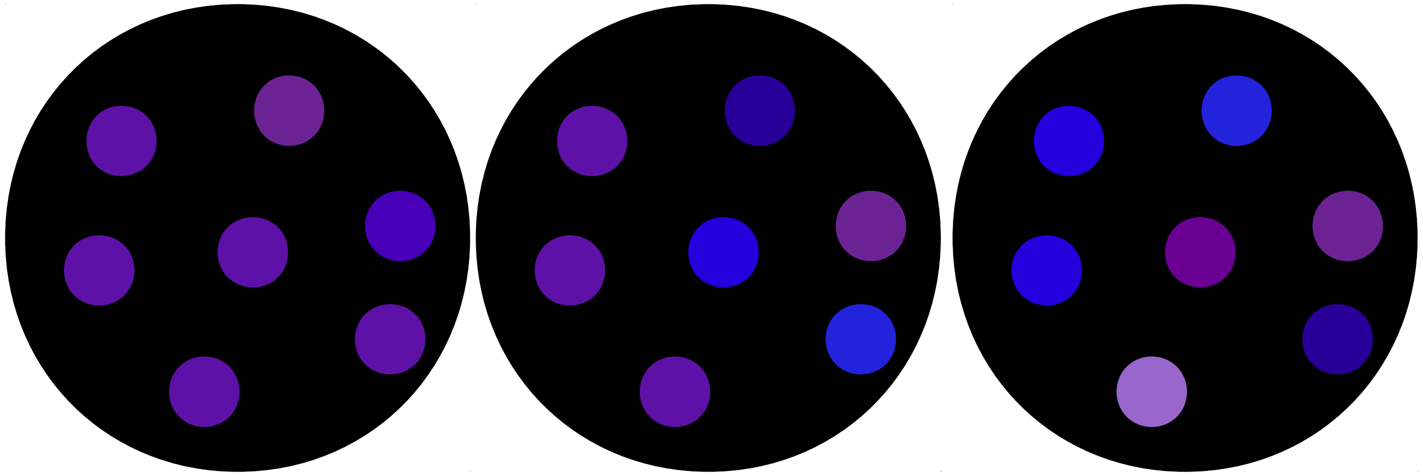
Evolution of solutions



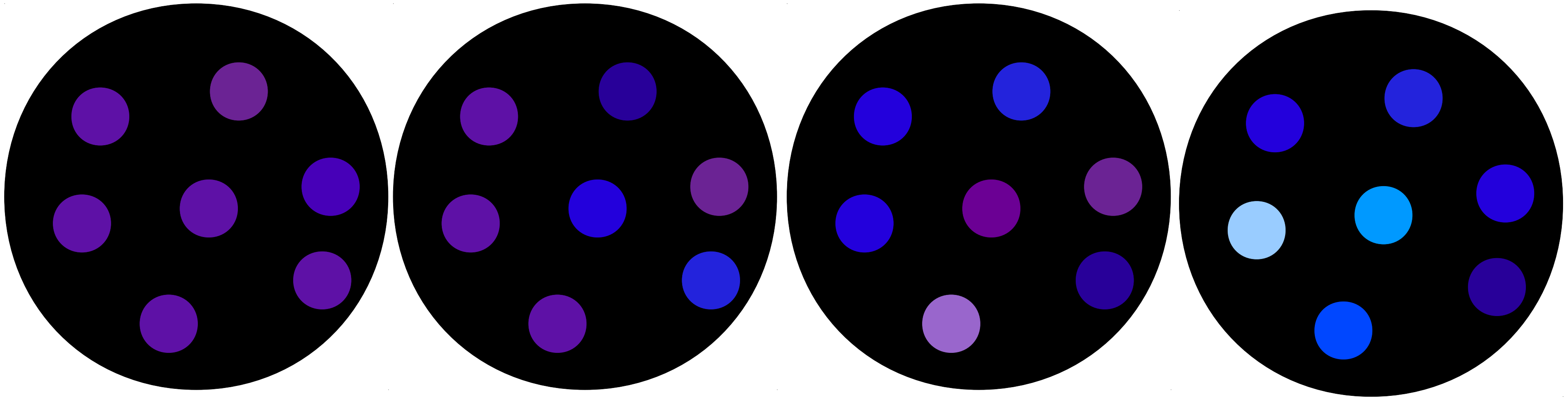
Evolution of solutions



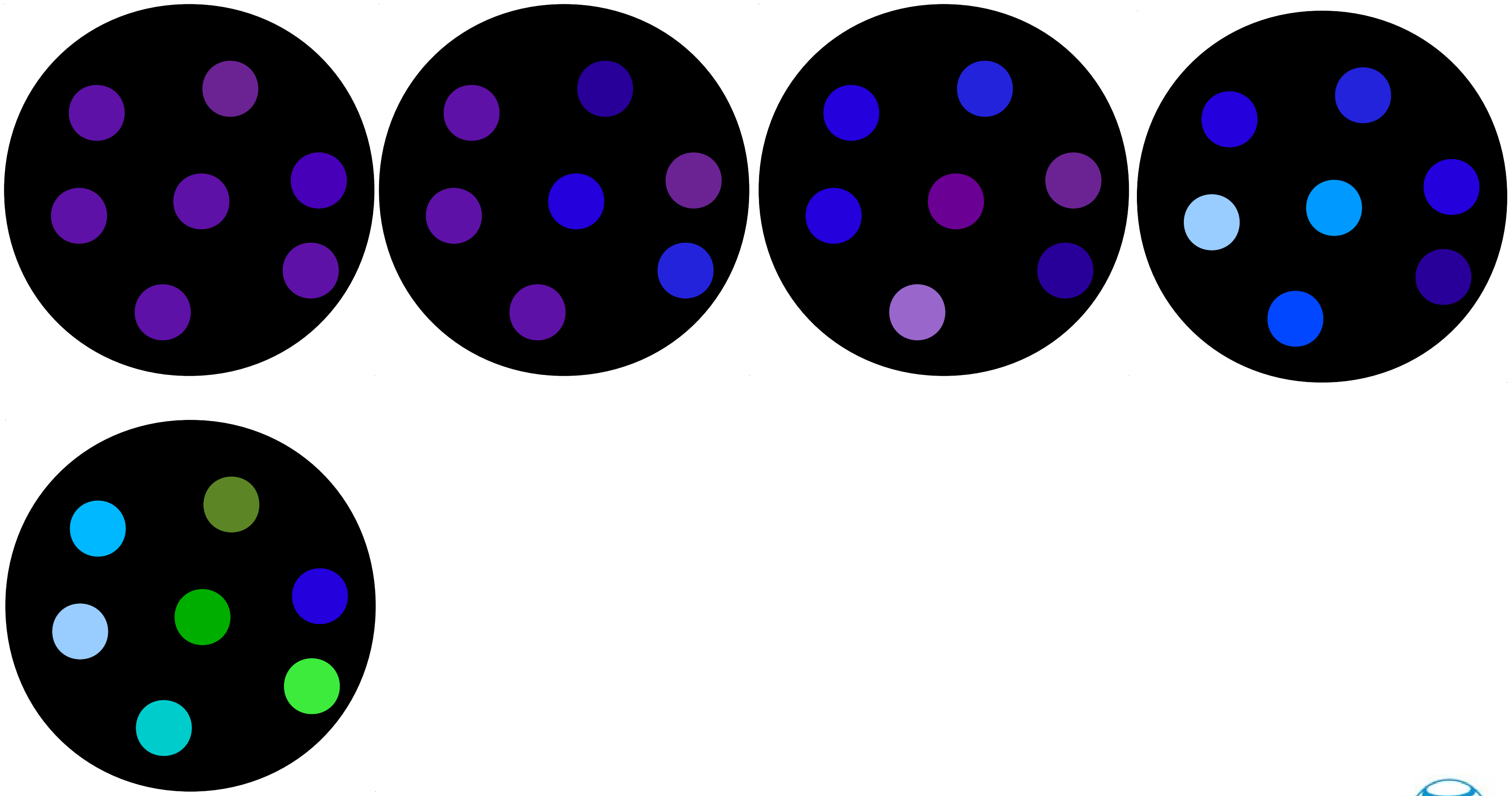
Evolution of solutions



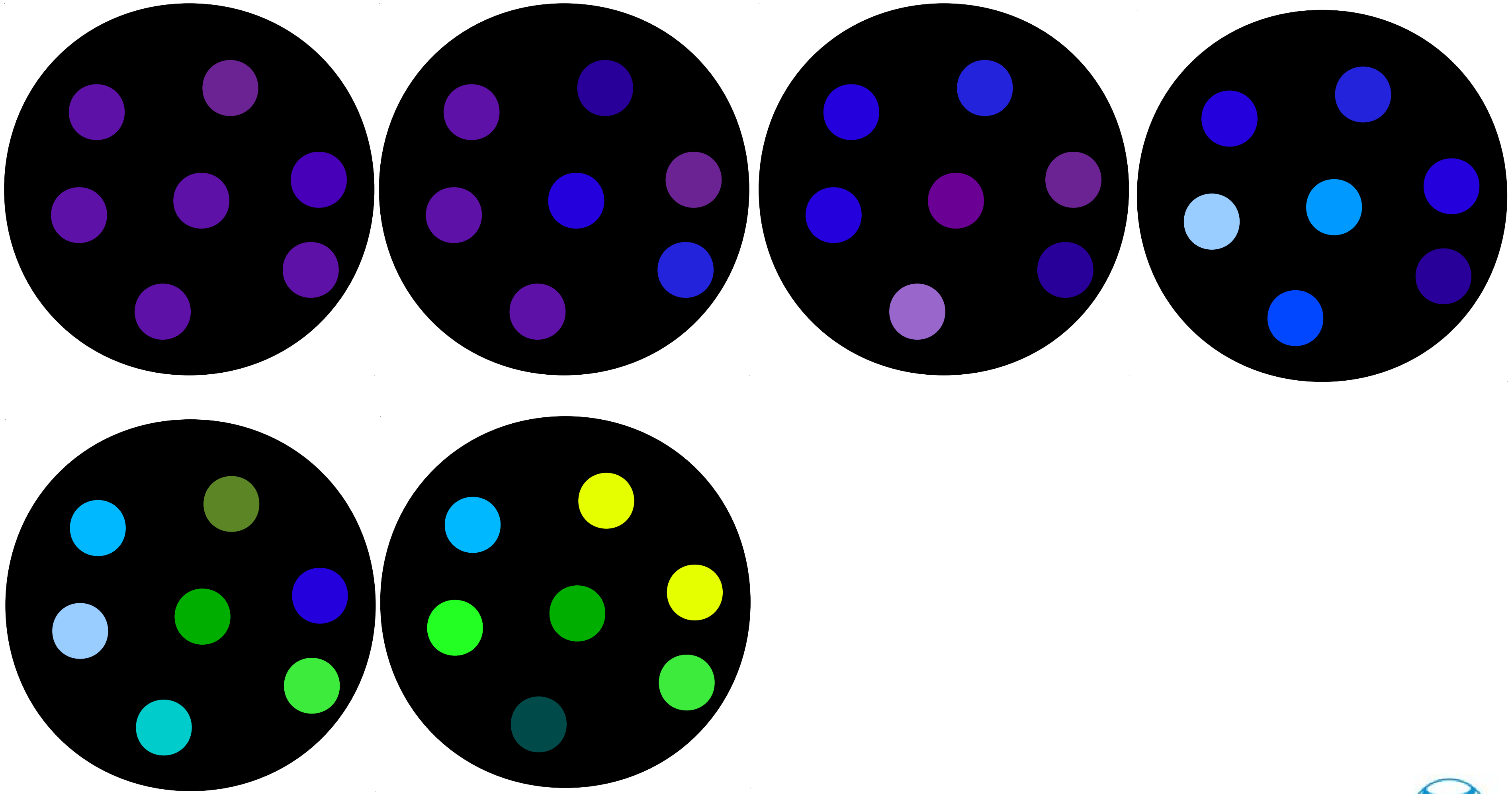
Evolution of solutions



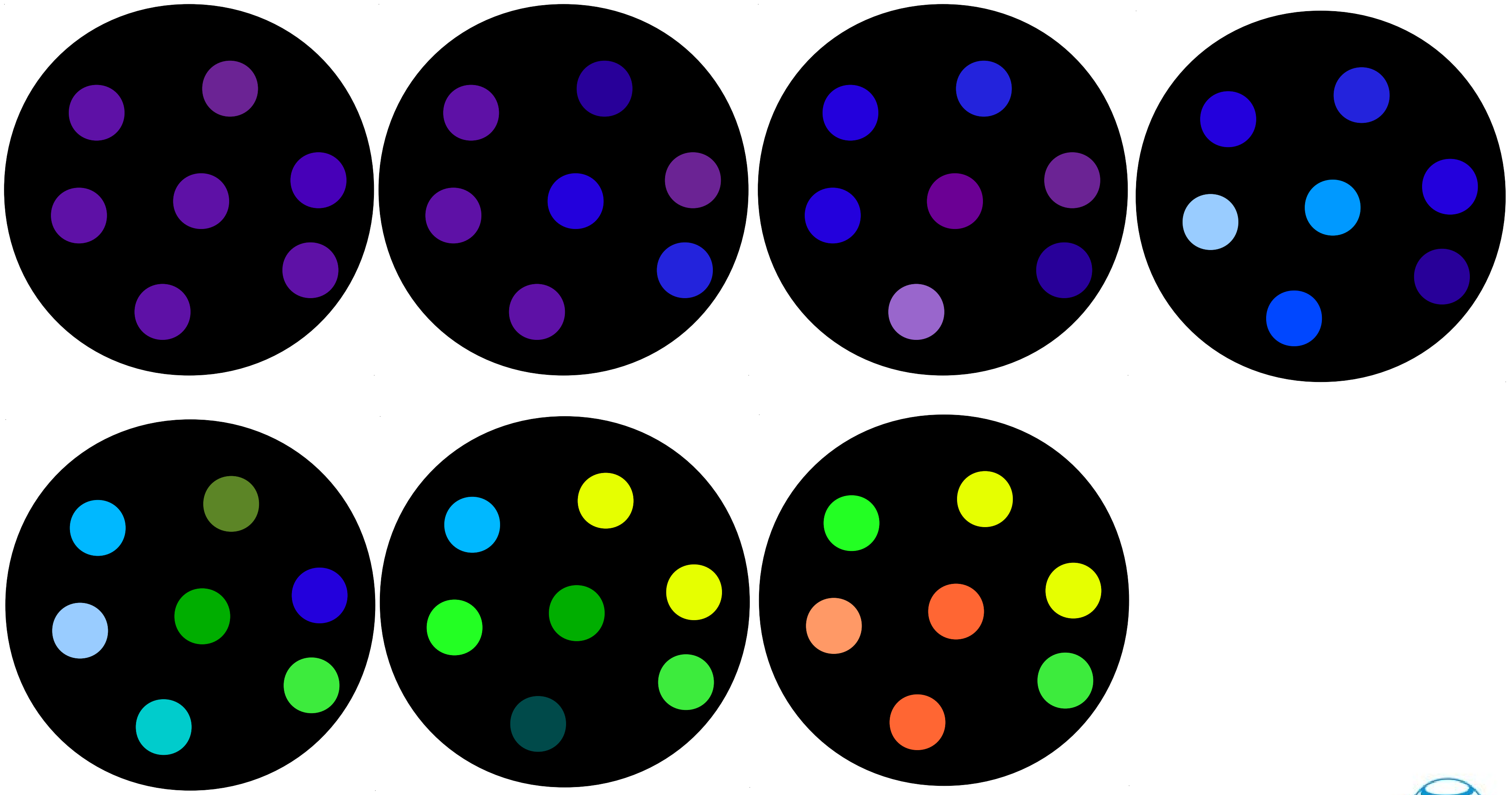
Evolution of solutions



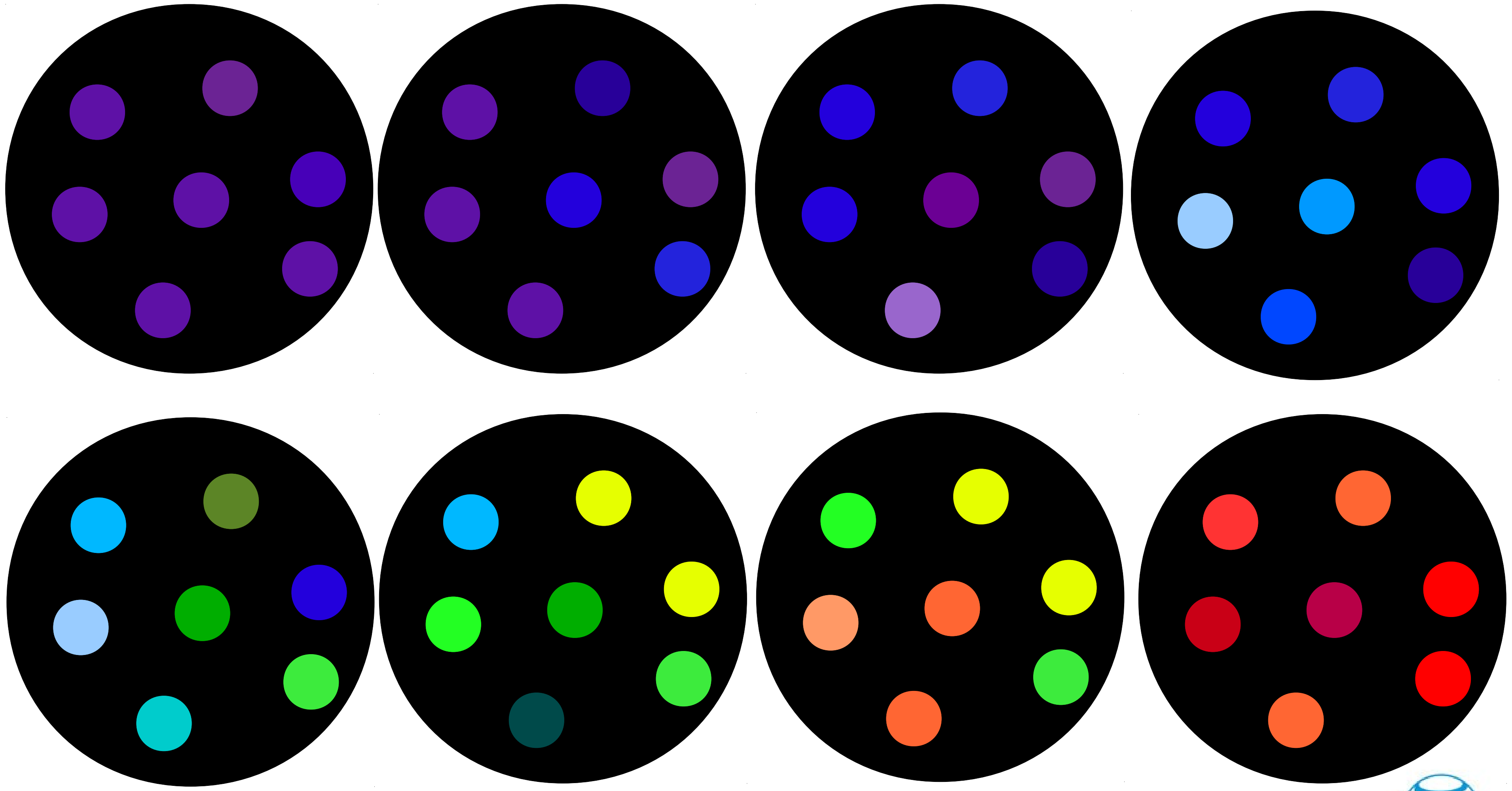
Evolution of solutions



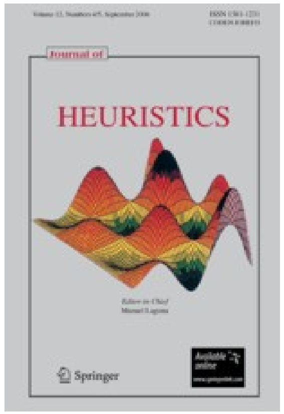
Evolution of solutions



Evolution of solutions



Reference



J.F. Gonçalves and M.G.C.R., "Biased random-key genetic algorithms for combinatorial optimization," J. of Heuristics, vol.17, pp. 487-525, 2011.

Tech report version:

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

Encoding solutions with random keys

Encoding with random keys

- *A* random key is a real random number in the continuous interval [0,1).

Encoding with random keys

- A random key is a real random number in the continuous interval [0,1).
- A vector X of random keys, or simply random keys, is an array of n random keys.

Encoding with random keys

- A random key is a real random number in the continuous interval [0,1).
- A vector X of random keys, or simply random keys, is an array of n random keys.
- Solutions of optimization problems can be encoded by random keys.

Encoding with random keys

- A random key is a real random number in the continuous interval [0,1).
- A vector X of random keys, or simply random keys, is an array of n random keys.
- Solutions of optimization problems can be encoded by random keys.
- A decoder is a deterministic algorithm that takes a vector of random keys as input and outputs a feasible solution of the optimization problem.

Encoding with random keys: Sequencing

Encoding

[1, 2, 3, 4, 5]

X = [0.099, 0.216, 0.802, 0.368, 0.658]

Encoding with random keys: Sequencing

Encoding

$$\begin{array}{rl} & [\ 1,\quad 2,\quad 3,\quad 4,\quad 5\ ] \\ X = & [\ 0.099,\ 0.216,\ 0.802,\ 0.368,\ 0.658\ ] \end{array}$$

Decode by sorting vector of random keys

$$\begin{array}{rl} & [\ 1,\quad 2,\quad 4,\quad 5,\quad 3\ ] \\ X = & [\ 0.099,\ 0.216,\ 0.368,\ 0.658,\ 0.802\ ] \end{array}$$

Encoding with random keys: Sequencing

Therefore, the vector of random keys:

X = [0.099, 0.216, 0.802, 0.368, 0.658]

encodes the sequence: 1 – 2 – 4 – 5 – 3

Encoding with random keys: Subset selection (select 3 of 5 elements)

Encoding

$$\begin{array}{ccccc} [\ 1, & 2, & 3, & 4, & 5\ ] \\ X = [\ 0.099, & 0.216, & 0.802, & 0.368, & 0.658\ ] \end{array}$$

Encoding with random keys: Subset selection (select 3 of 5 elements)

Encoding

$$\begin{array}{cccccc} & [\ 1, & 2, & 3, & 4, & 5\ ] \\ X = & [\ 0.099, & 0.216, & 0.802, & 0.368, & 0.658\ ] \end{array}$$

Decode by sorting vector of random keys

$$\begin{array}{cccccc} & [\ 1, & 2, & 4, & 5, & 3\ ] \\ X = & [\ 0.099, & 0.216, & 0.368, & 0.658, & 0.802\ ] \end{array}$$

Encoding with random keys: Subset selection (select 3 of 5 elements)

Therefore, the vector of random keys:

X = [0.099, 0.216, 0.802, 0.368, 0.658]

encodes the subset: {1, 2, 4 }

Encoding with random keys: Assigning integer weights ∈ [0,10] to a subset of 3 of 5 elements

Encoding

```
      [    1,    2,    3,    4,    5 |     1,     2,     3,     4,     5 ]
X = [ 0.099, 0.216, 0.802, 0.368, 0.658 | 0.4634, 0.5611, 0.2752, 0.4874, 0.0348 ]
```

Encoding with random keys: Assigning integer weights $\in [0,10]$ to a subset of 3 of 5 elements

Encoding

[1, 2, 3, 4, 5 | 1, 2, 3, 4, 5]

X = [0.099, 0.216, 0.802, 0.368, 0.658 | 0.4634, 0.5611, 0.2752, 0.4874, 0.0348]

Decode by sorting the first 5 keys and assign as the weight the value $W_i =$ **floor** $[\ 10\ X_{5+i}\ ] + 1$ to the 3 elements with smallest keys X_i, for $i = 1,\ldots,5$.

Encoding with random keys: Assigning integer weights ∈ [0,10] to a subset of 3 of 5 elements

Therefore, the vector of random keys:

X = [0.099, 0.216, 0.802, 0.368, 0.658 | 0.4634, 0.5611, 0.2752, 0.4874, 0.0348]

encodes the weight vector W = (5,6,–,5,–)

Genetic algorithms and 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).

$$S = (\underbrace{0.25}_{s(1)}, \underbrace{0.19}_{s(2)}, \underbrace{0.67}_{s(3)}, \underbrace{0.05}_{s(4)}, \underbrace{0.89}_{s(5)})$$

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.

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

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

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 (key, or value of gene) 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 (key, or value of gene) to the child.

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

GAs and random keys

- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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

GAs and random keys

- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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

GAs and random keys

- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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

GAs and random keys

- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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

GAs and random keys

- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) 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

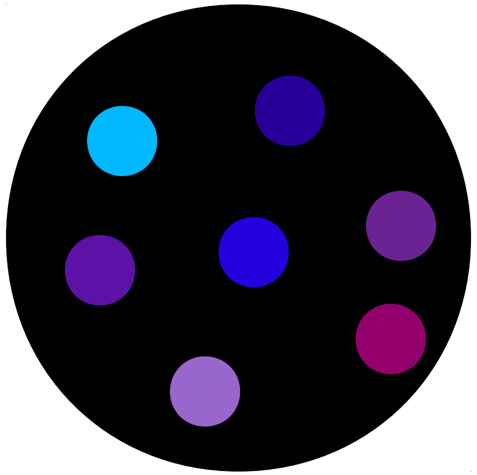
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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.

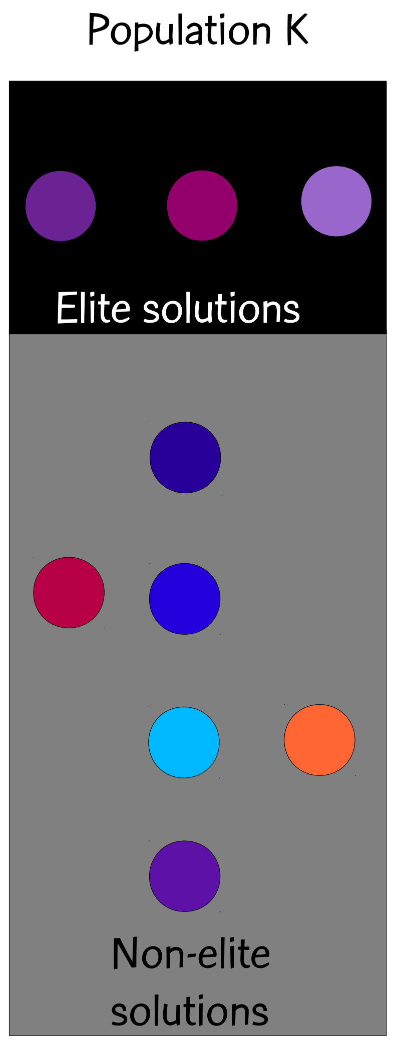
GAs and random keys

Initial population is made up of P random-key vectors, each with N keys, each having a value generated uniformly at random in the interval [0,1).



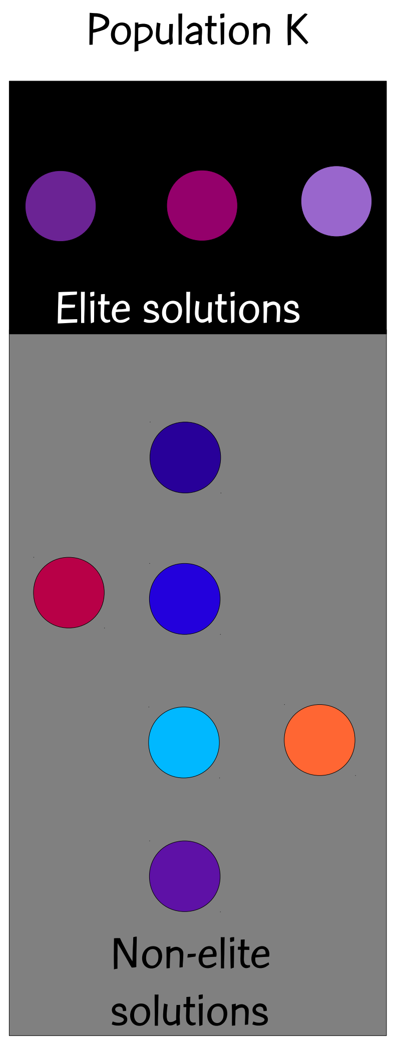
GAs and random keys

At the K-th generation, compute the cost of each solution ...



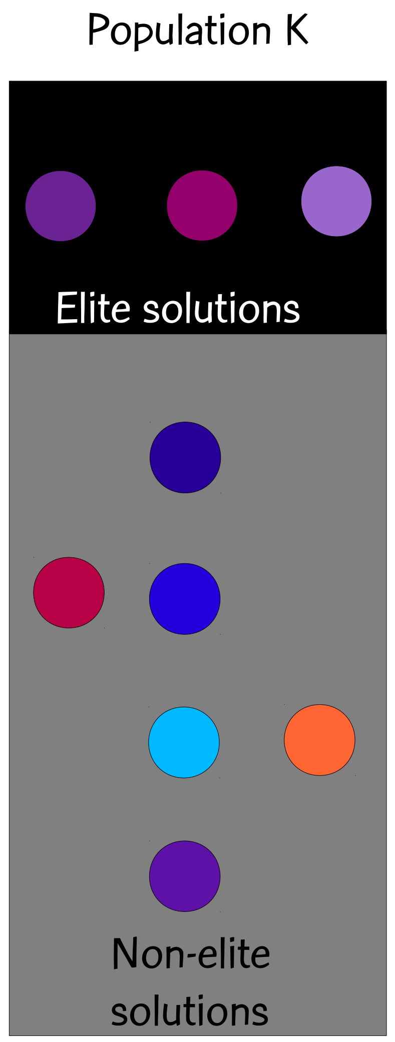
GAs and random keys

At the K-th generation, compute the cost of each solution and partition the solutions into two sets:



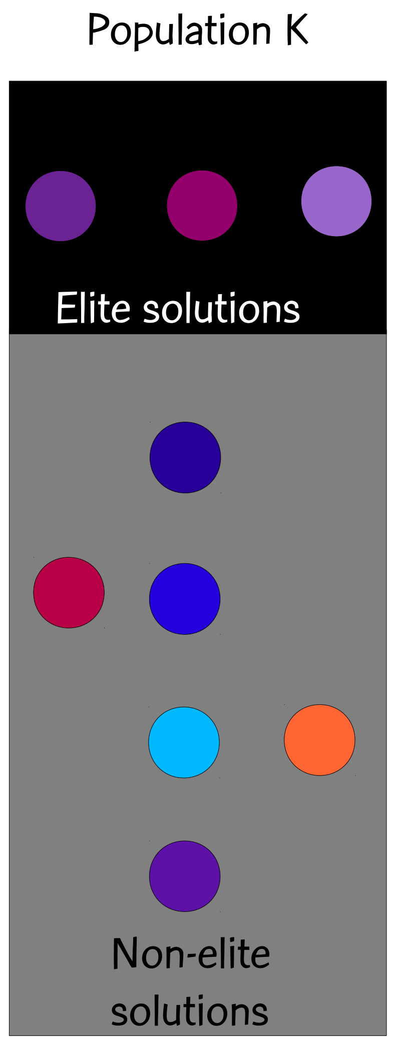
GAs and random keys

At the K-th generation, compute the cost of each solution and partition the solutions into two sets: elite solutions and non-elite solutions.



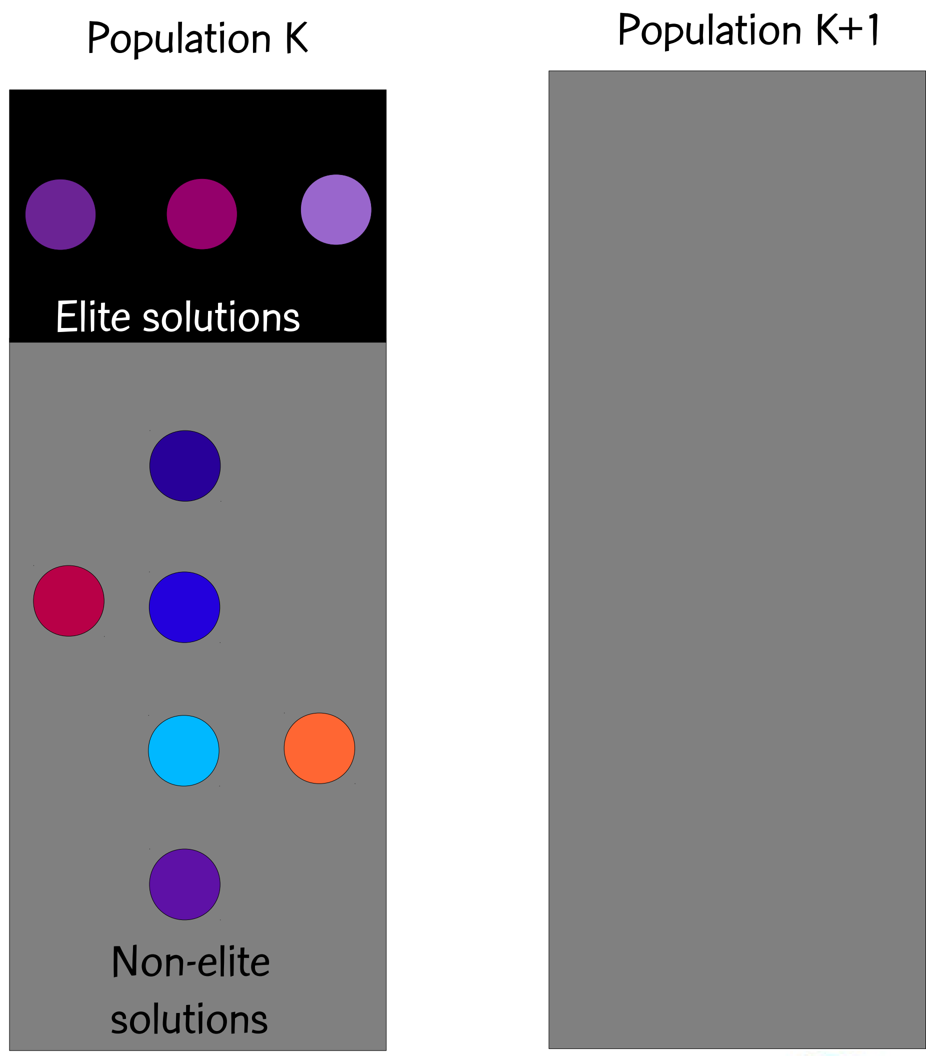
GAs and random keys

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



GAs and random keys

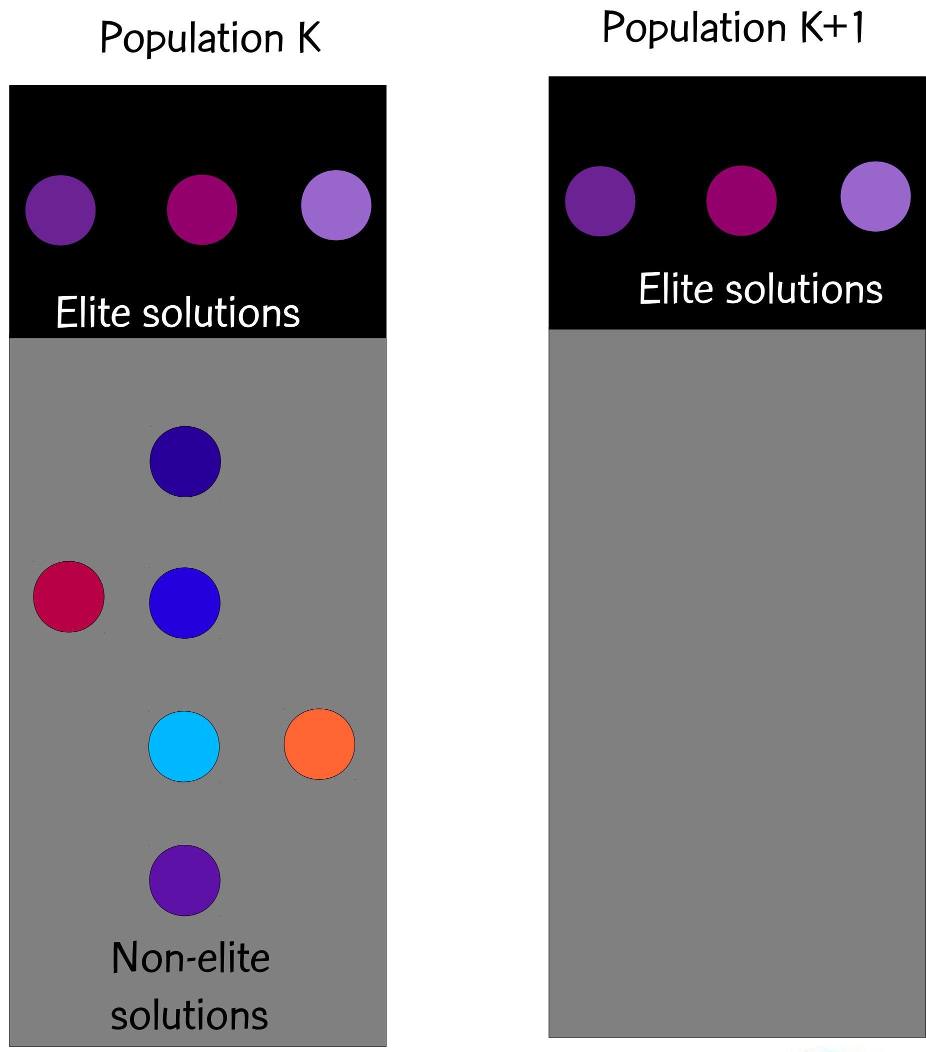
Evolutionary dynamics



GAs and random keys

Evolutionary dynamics

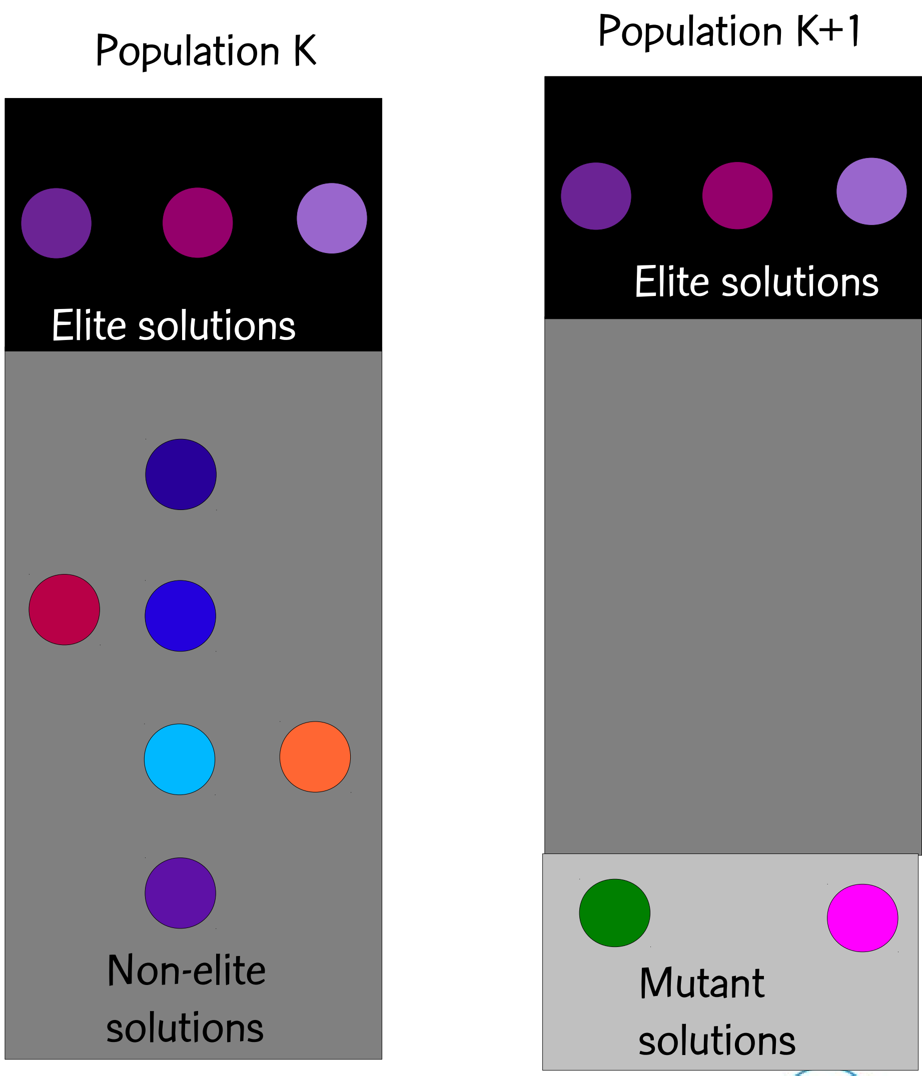
- Copy elite solutions from population K to population K+1



GAs and random keys

Evolutionary dynamics

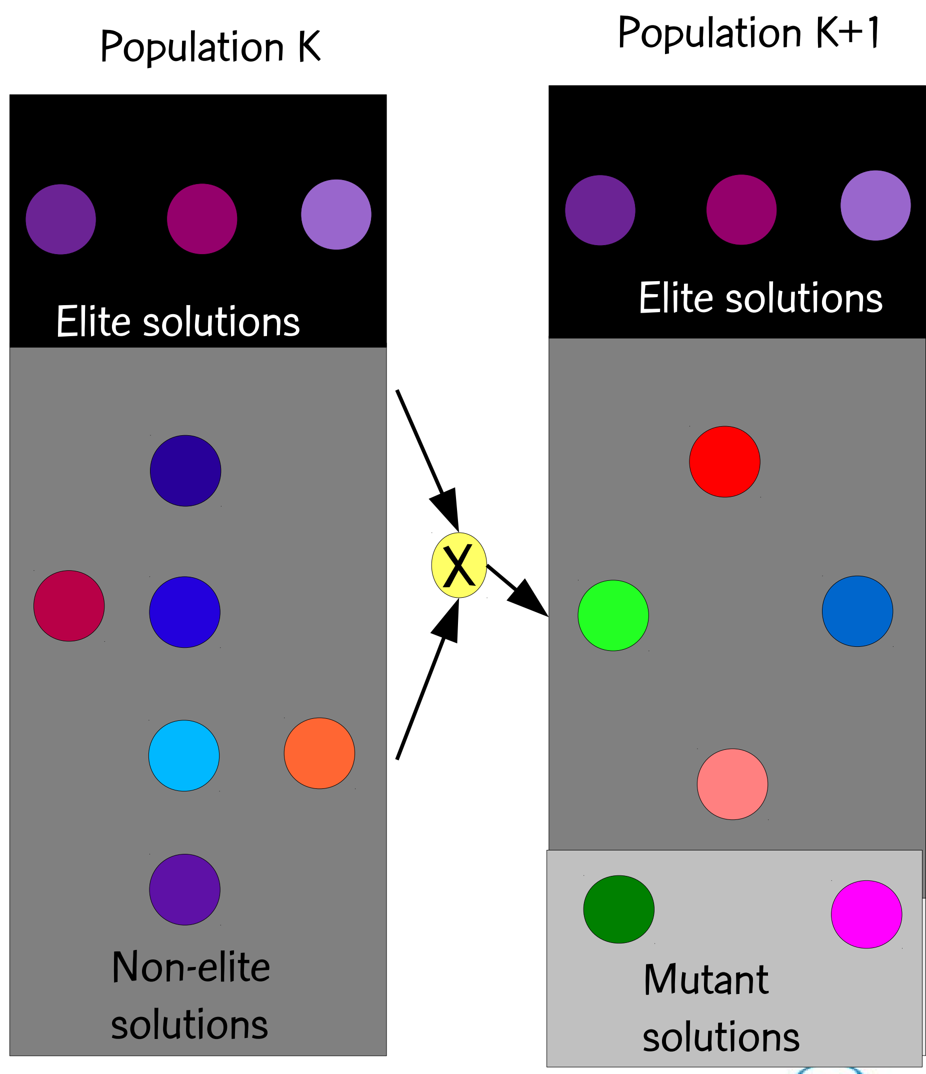
- Copy elite solutions from population K to population K+1
- Add R random solutions (mutants) to population K+1



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
- While K+1-th population < P
  - RANDOM-KEY GA: Use any two solutions in population K to produce child in population K+1. Mates are chosen at random.



Biased random key genetic algorithm

- A biased random key genetic algorithm (BRKGA) is a random key genetic algorithm (RKGA).

Biased random key genetic algorithm

- A biased random key genetic algorithm (BRKGA) is a random key genetic algorithm (RKGA).
- BRKGA and RKGA differ in how mates are chosen for crossover and how parametrized uniform crossover is applied.

How RKGA & BRKGA differ

RKGA	BRKGA
both parents chosen at random from entire population	

How RKGA & BRKGA differ

RKGA

both parents chosen at random from entire population

BRKGA

both parents chosen at random but one parent chosen from population of elite solutions

How RKGA & BRKGA differ

RKGA	BRKGA
both parents chosen at random from entire population	both parents chosen at random but one parent chosen from population of elite solutions
either parent can be parent A in parametrized uniform crossover	

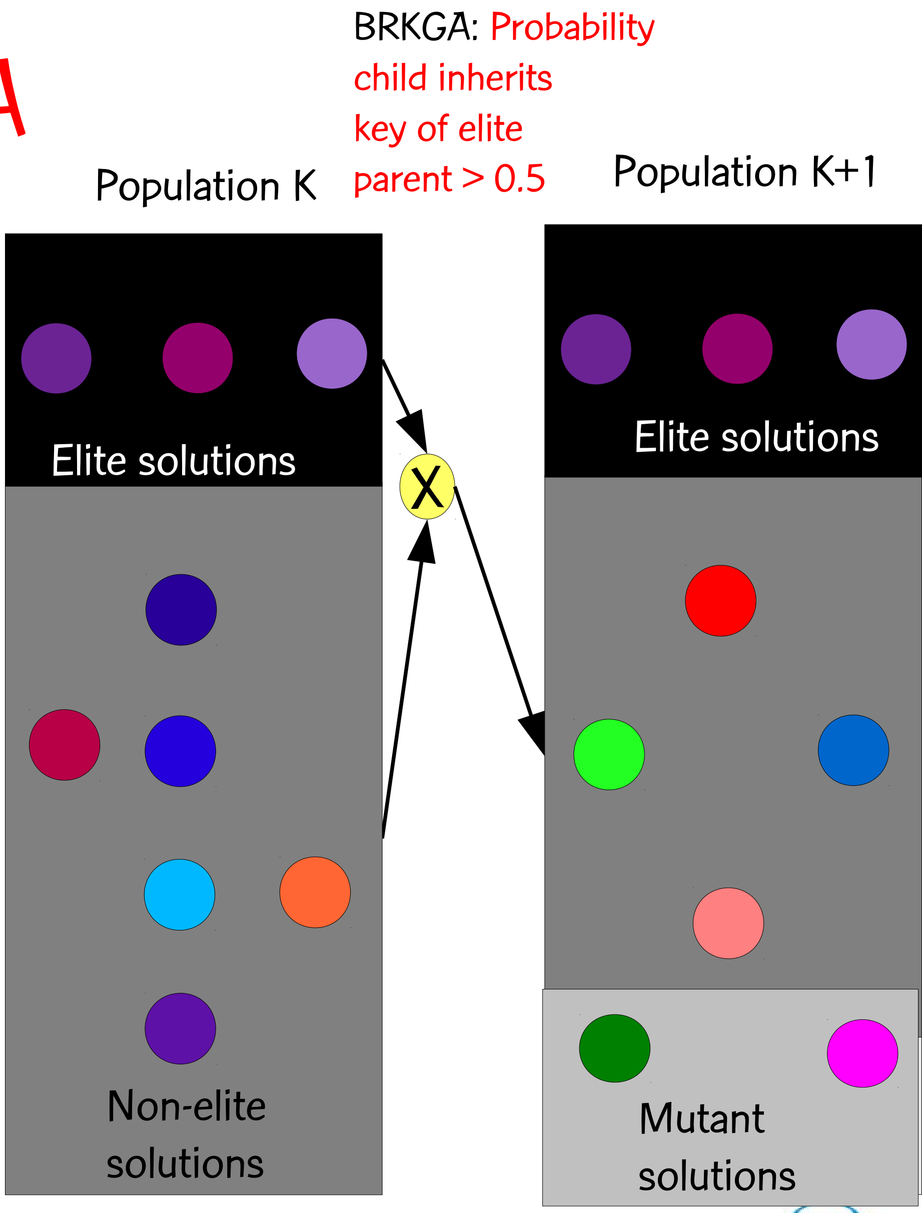
How RKGA & BRKGA differ

RKGA	**BRKGA**
both parents chosen at random from entire population	both parents chosen at random but one parent chosen from population of elite solutions
either parent can be parent A in parametrized uniform crossover	best fit parent is parent A in parametrized uniform crossover

Biased random key GA

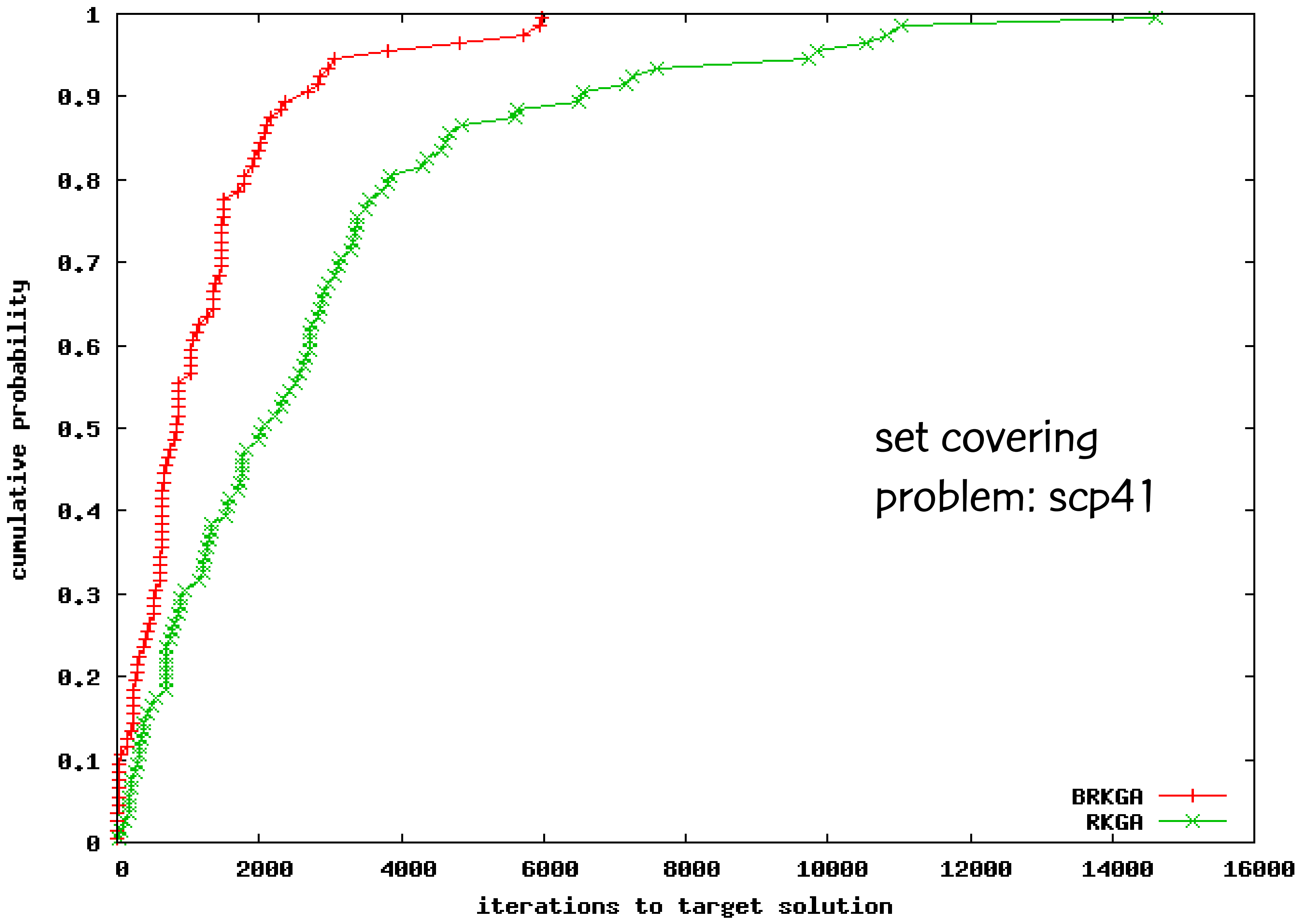
Evolutionary dynamics

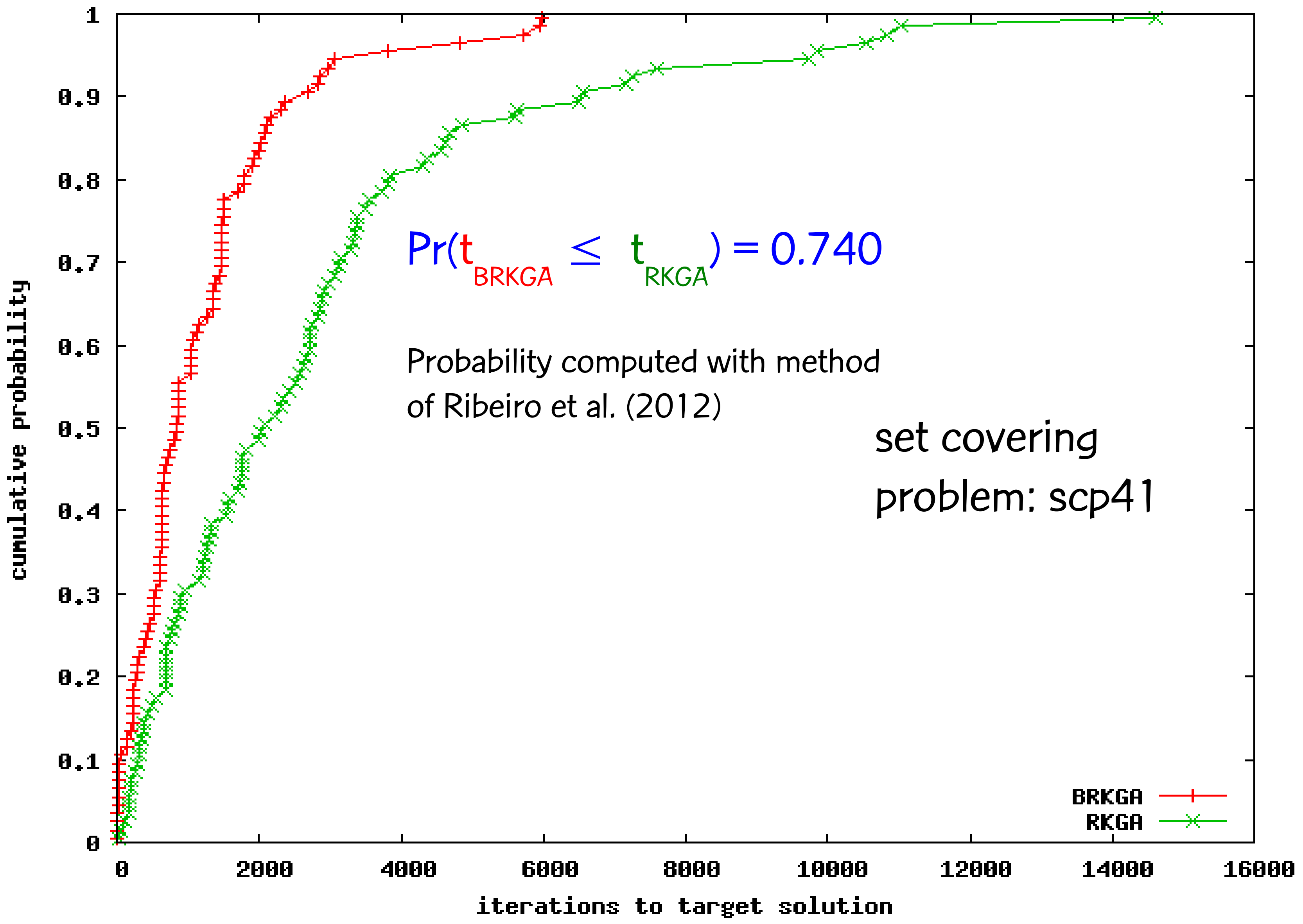
- Copy elite solutions from population K to population K+1
- Add R random solutions (mutants) to population K+1
- While K+1-th population < P
    - RANDOM-KEY GA: Use any two solutions in population K to produce child in population K+1. Mates are chosen at random.
    - BIASED RANDOM-KEY GA: Mate elite solution with other solution of population K to produce child in population K+1. Mates are chosen at random.



Paper comparing BRKGA and Bean's Method

Gonçalves, R., and Toso, "Biased and unbiased random-key genetic algorithms: An experimental analysis", AT&T Labs Research Technical Report, Florham Park, December 2012.

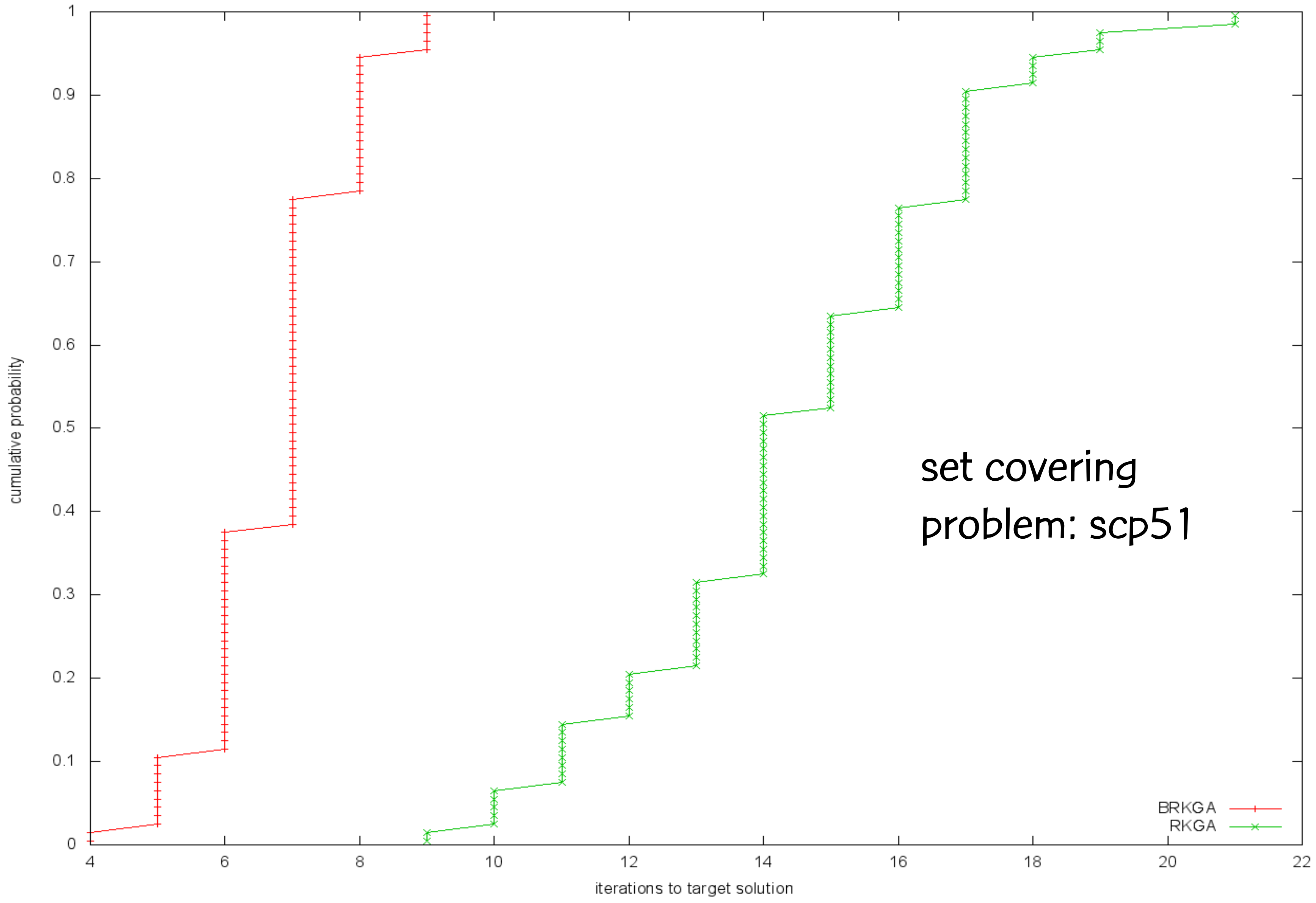

cumulative probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
12000
14000
16000
iterations to target solution
set covering
problem: scp41
BRKGA
RKGA



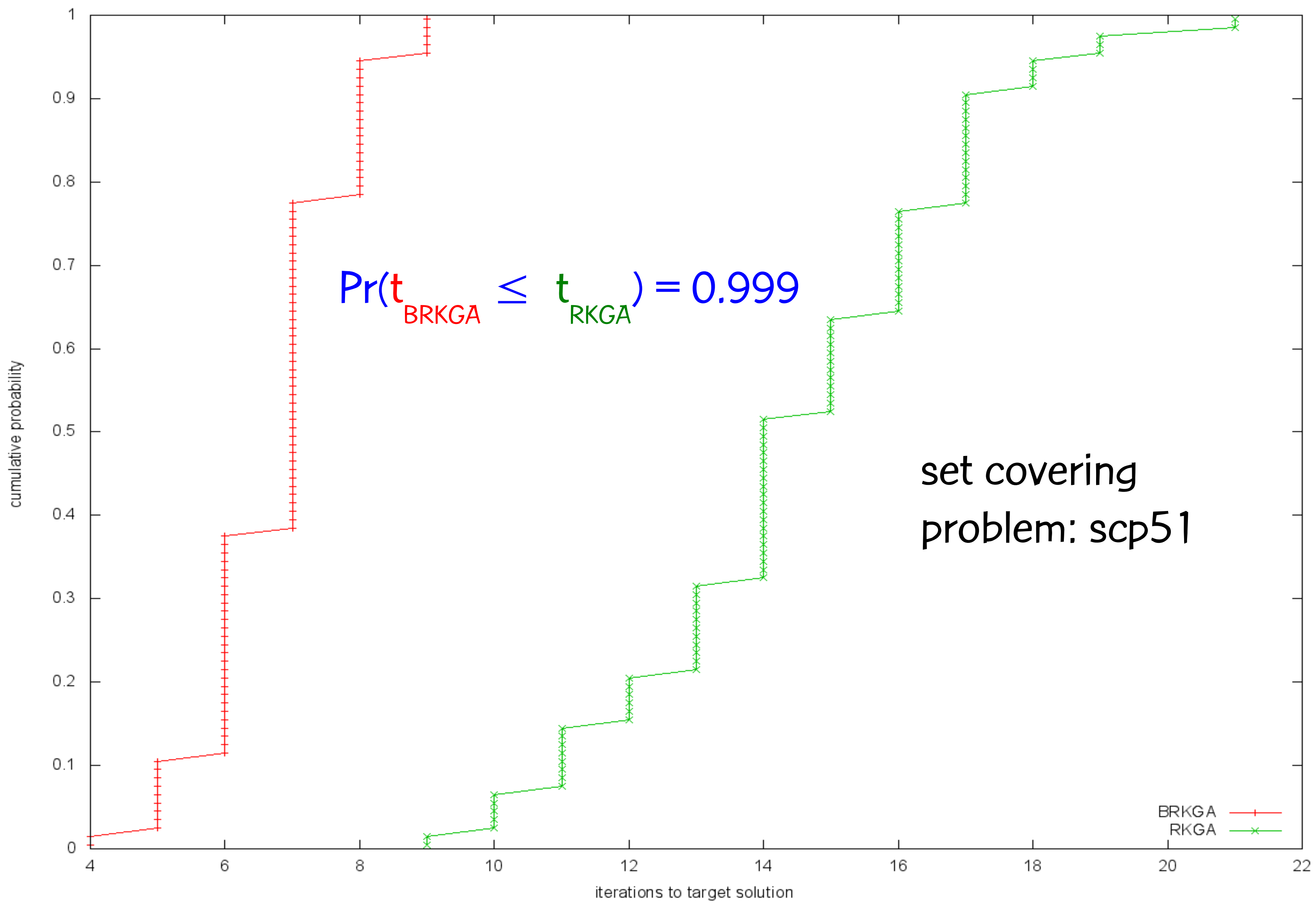
$\Pr(t_{\text{BRKGA}} \leq t_{\text{RKGA}}) = 0.740$

Probability computed with method of Ribeiro et al. (2012)

set covering problem: scp41

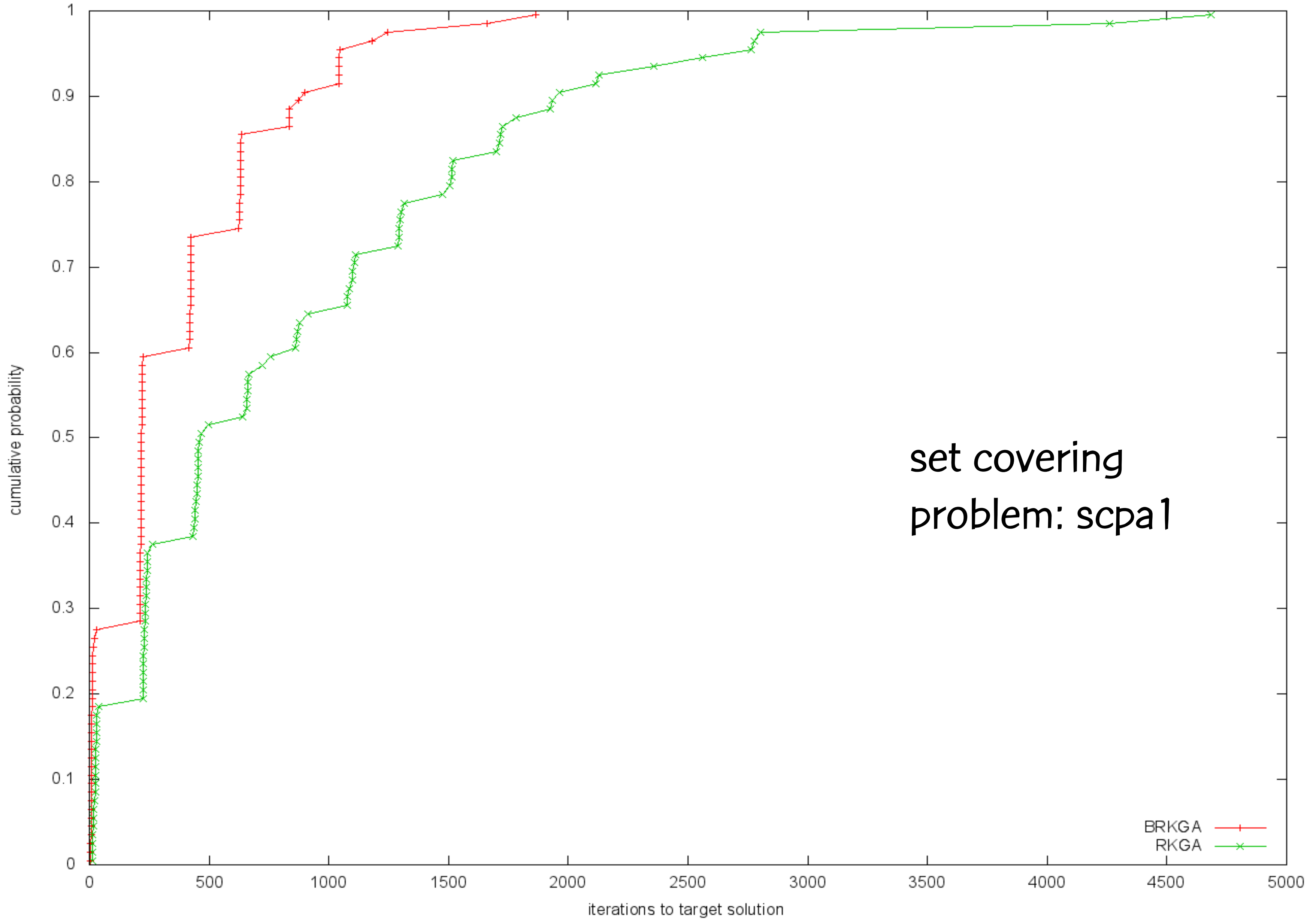


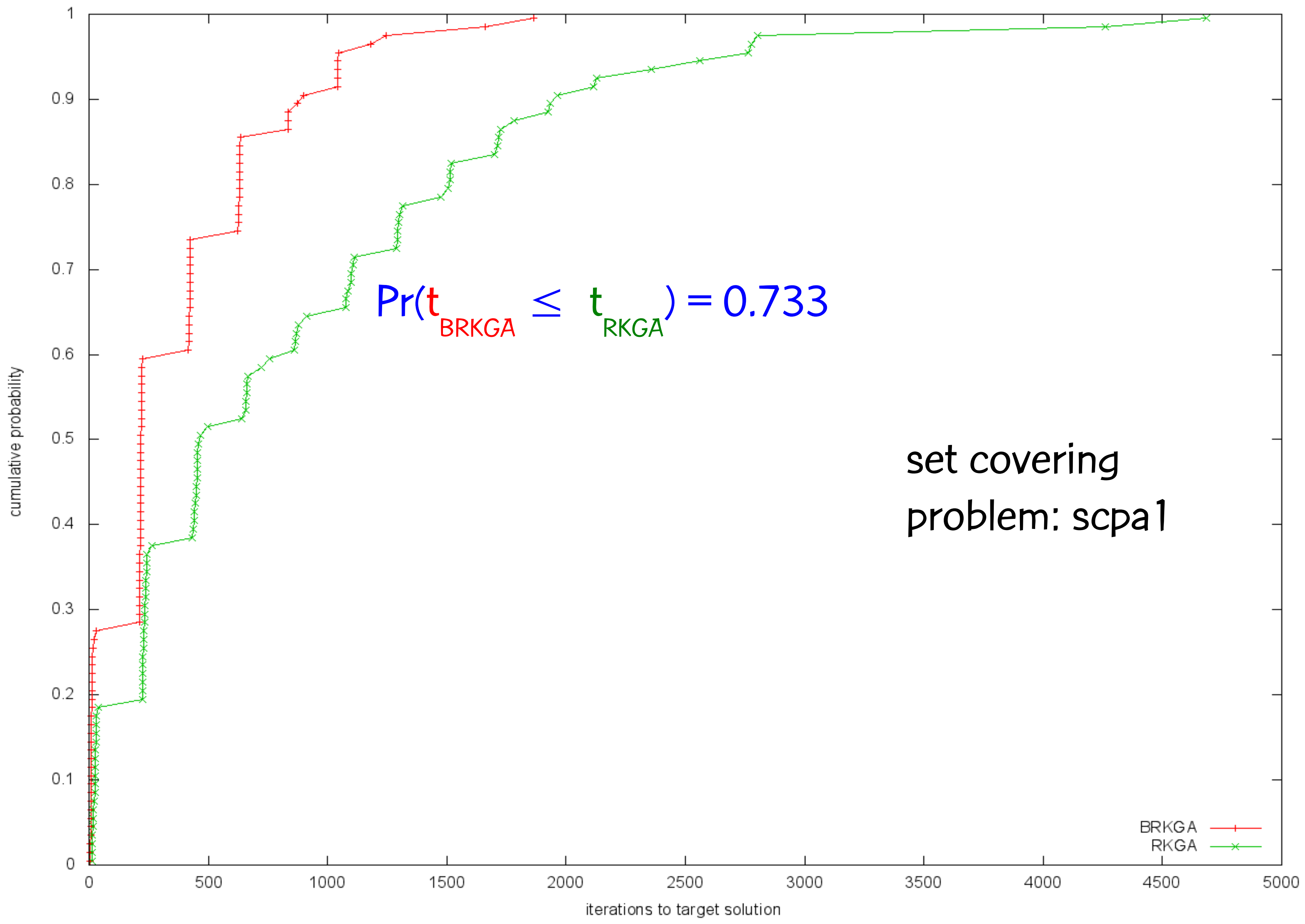
set covering problem: scp51



$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.999$

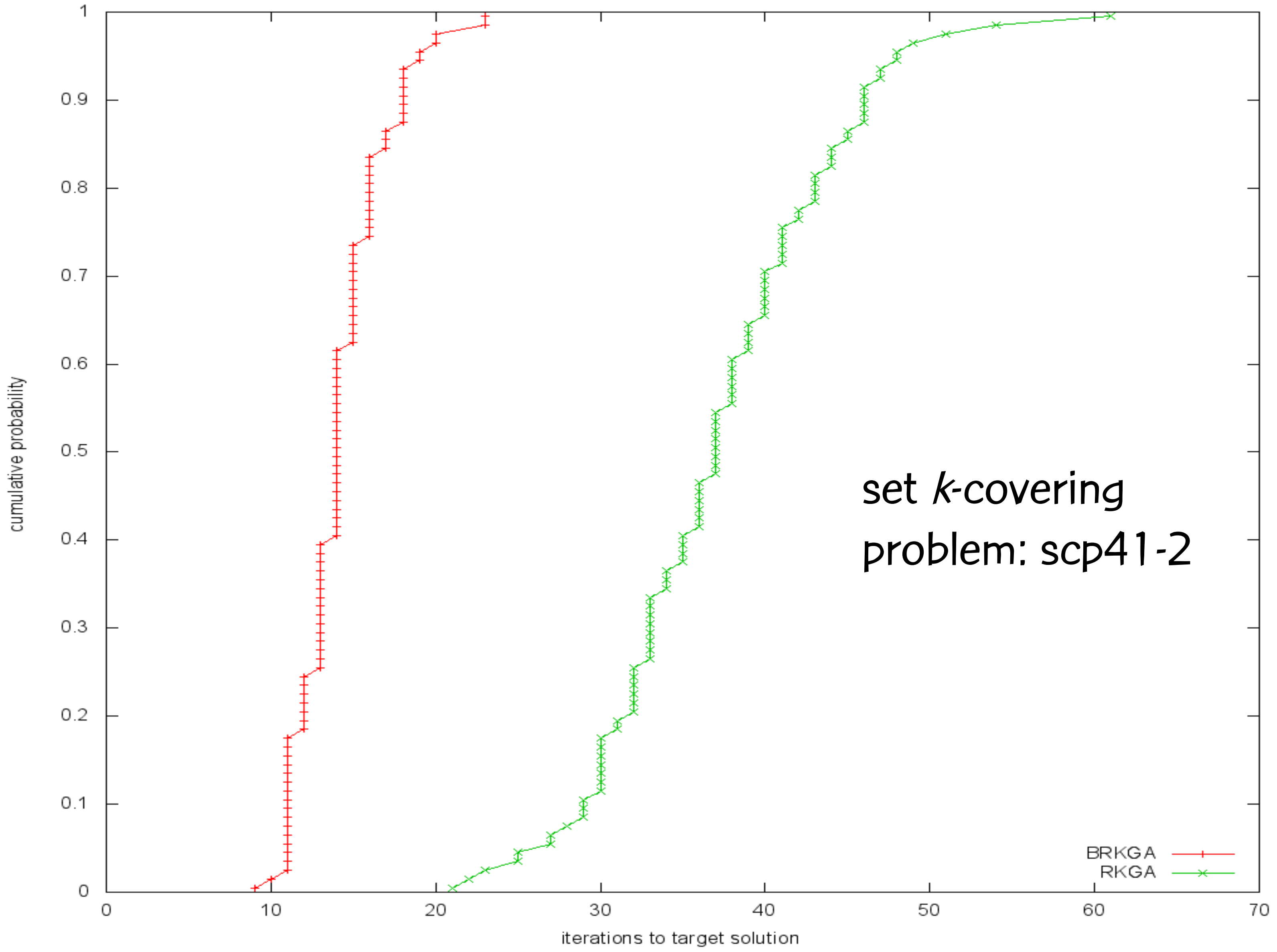
set covering problem: scp51


cumulative probability
iterations to target solution
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
BRKGA
RKGA
set covering problem: scpa1

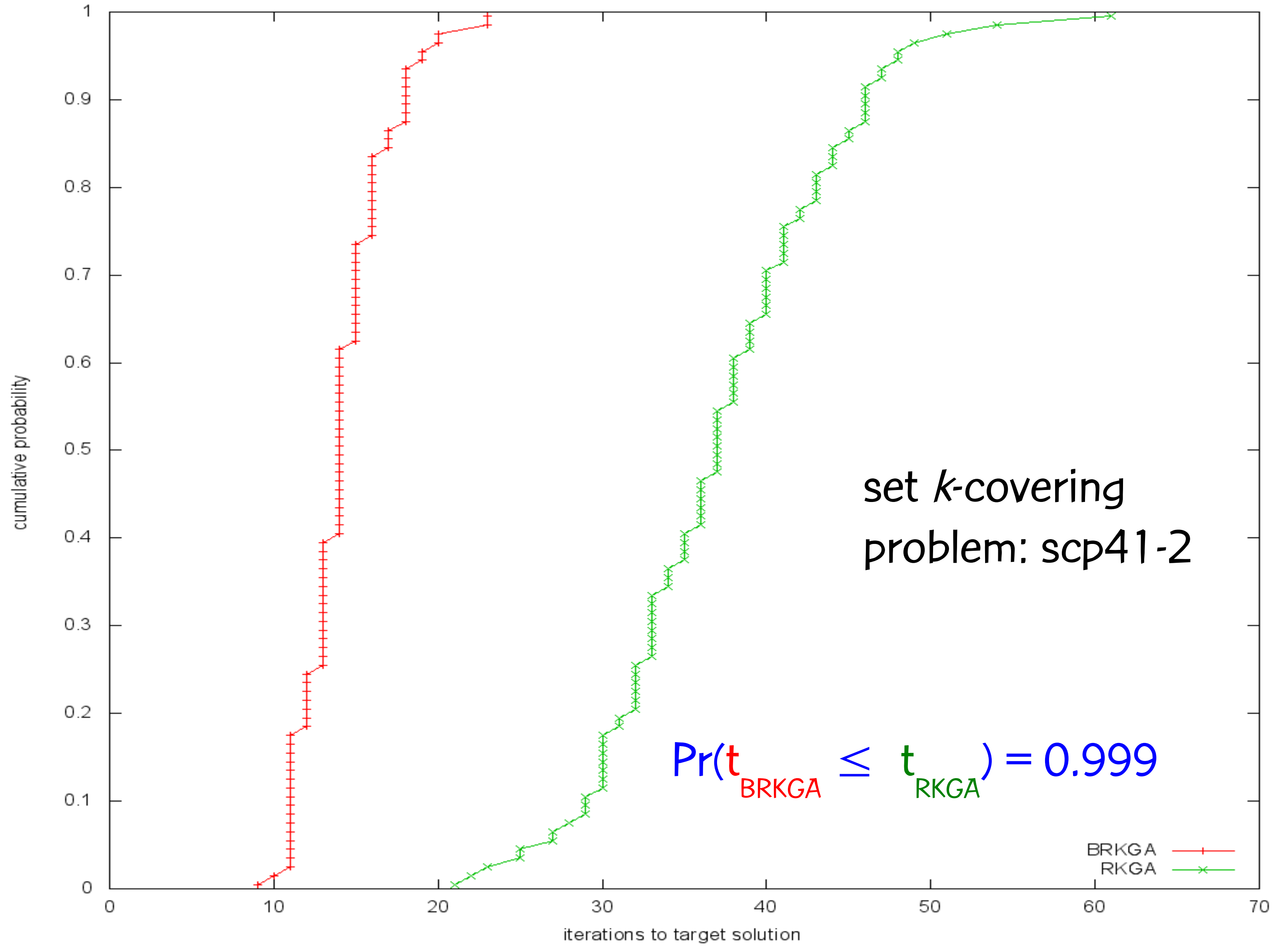


$\Pr(t_{BRKGA} \leq t_{RKGA}) = 0.733$

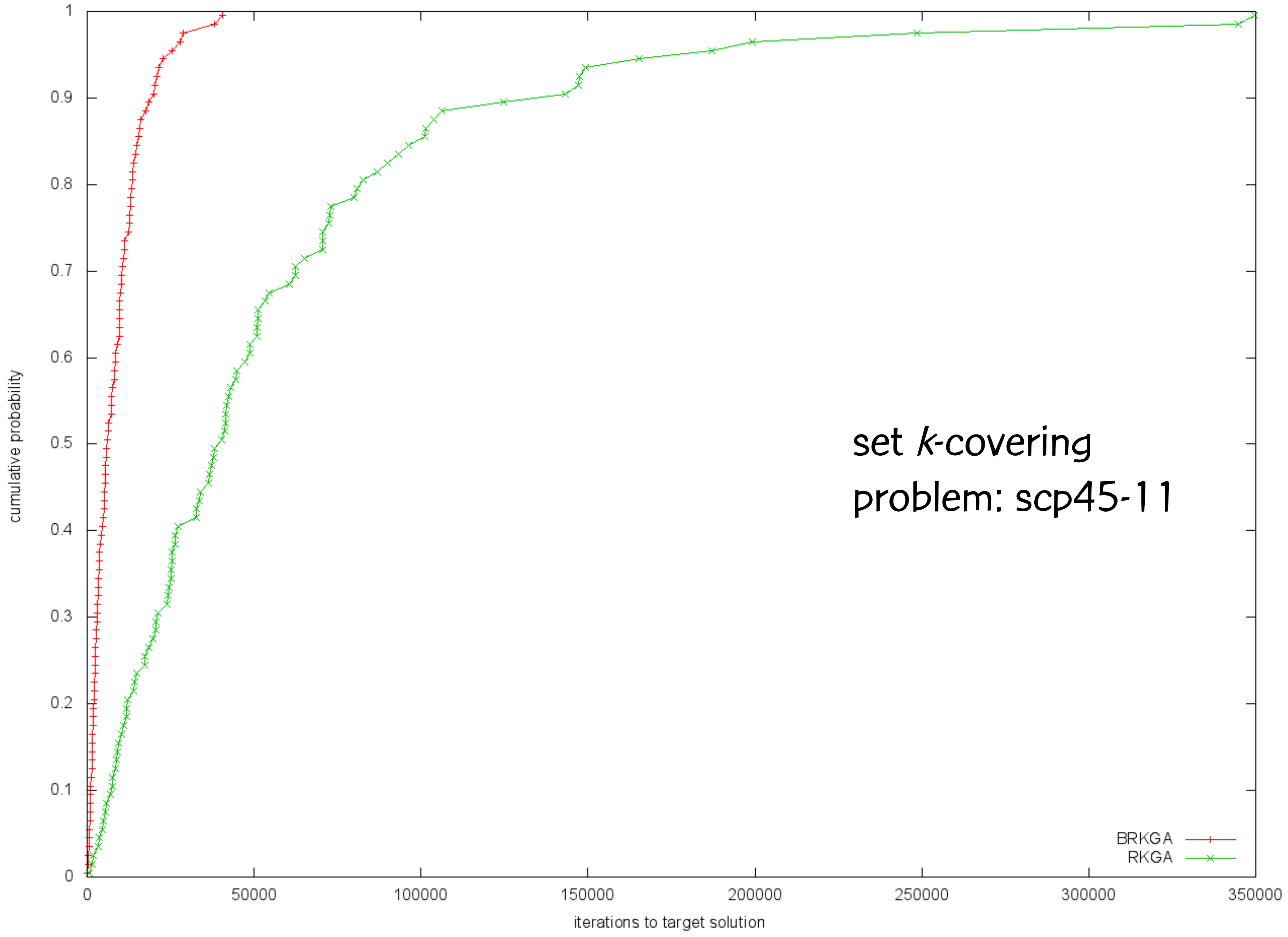
set covering problem: scpa1



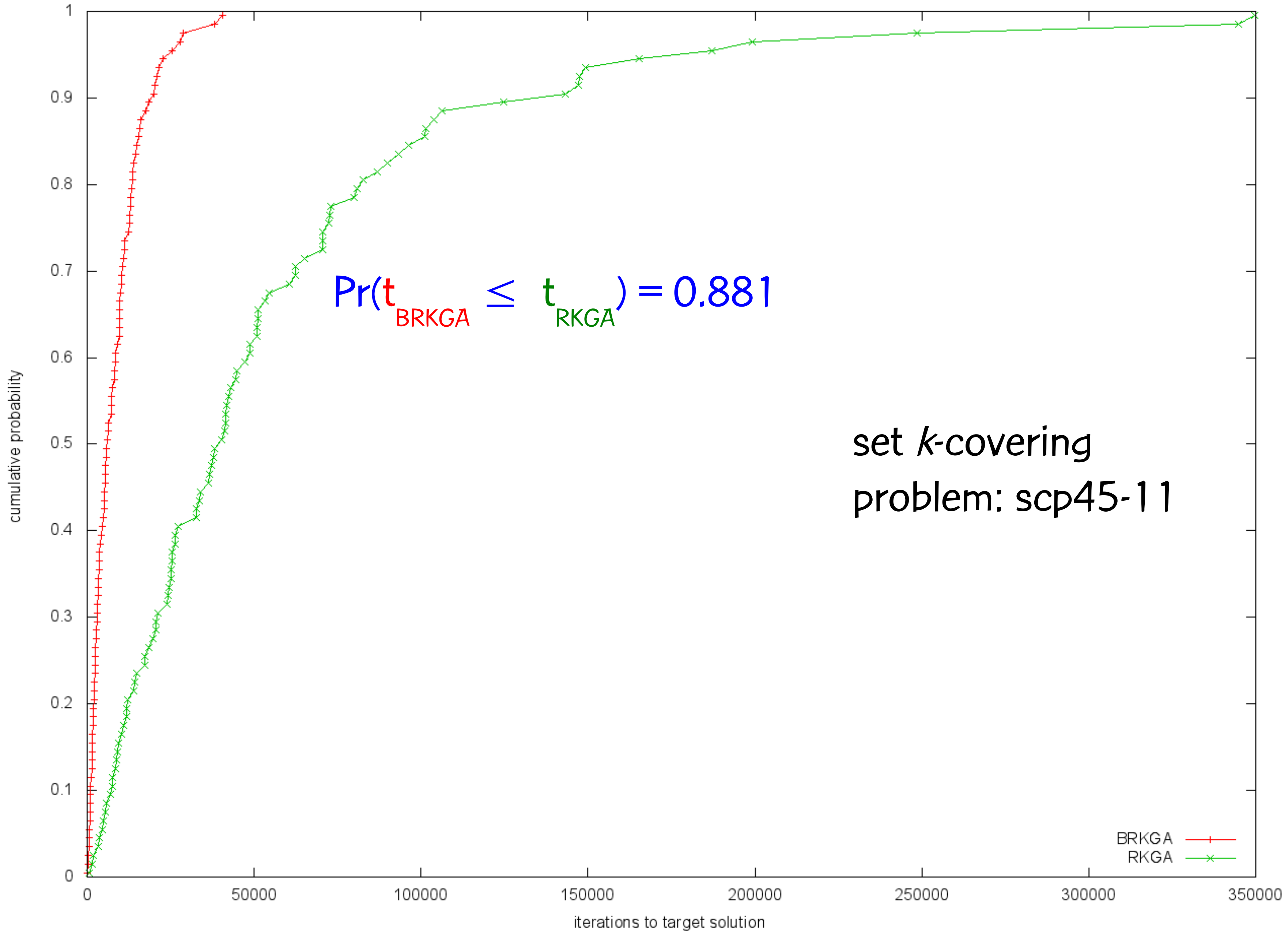
set *k*-covering problem: scp41-2



cumulative probability
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70
iterations to target solution
BRKGA
RKGA
set *k*-covering problem: scp41-2
$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.999$

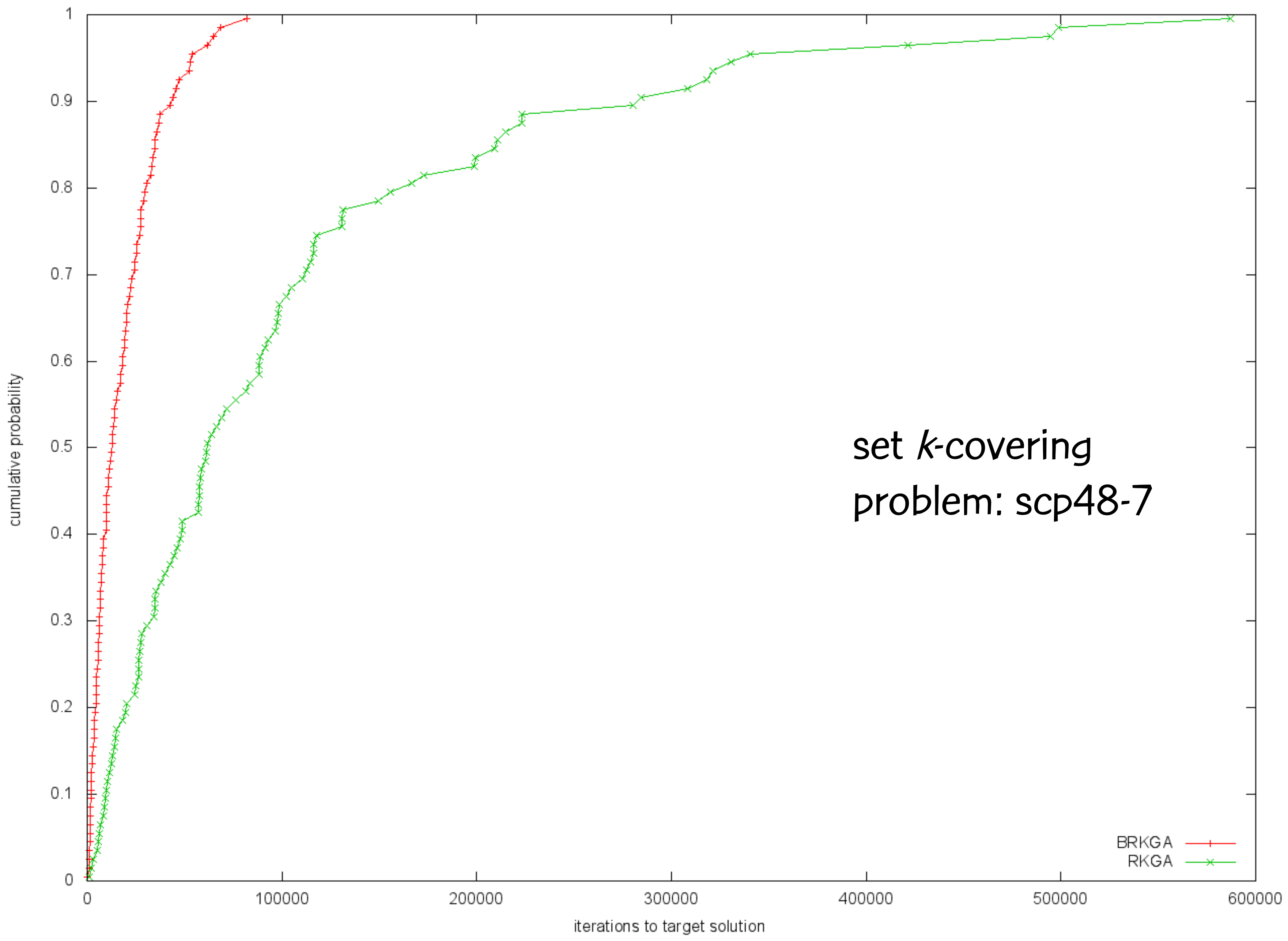


set *k*-covering problem: scp45-11

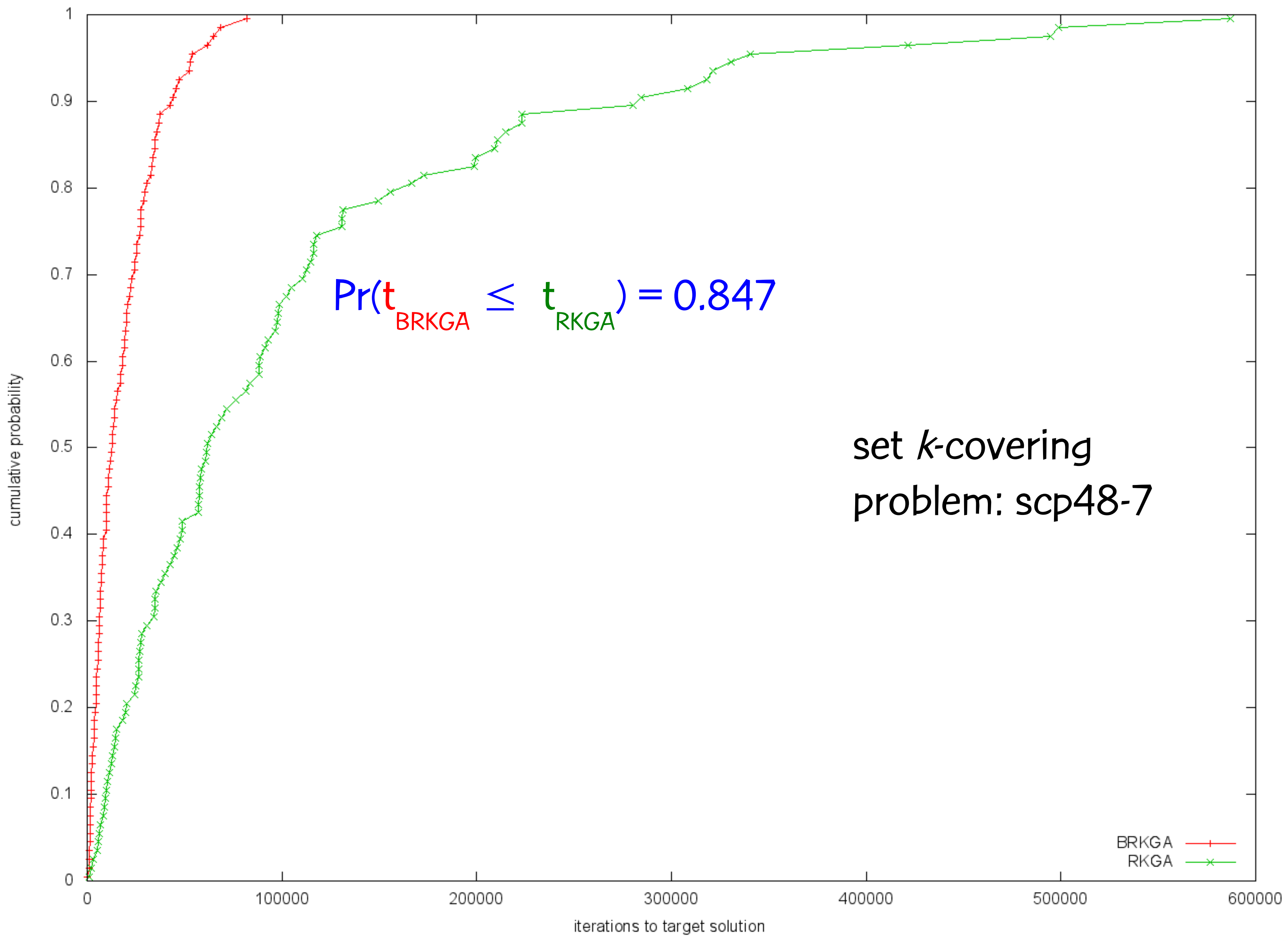


$\Pr(t_{\text{BRKGA}} \leq t_{\text{RKGA}}) = 0.881$

set *k*-covering problem: scp45-11



set *k*-covering problem: scp48-7



$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.847$

set *k*-covering problem: scp48-7

Observations

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

Observations

- Random method: keys are randomly generated so solutions are always vectors of random keys
- Elitist strategy: best solutions are passed without change from one generation to the next (incumbent is kept)

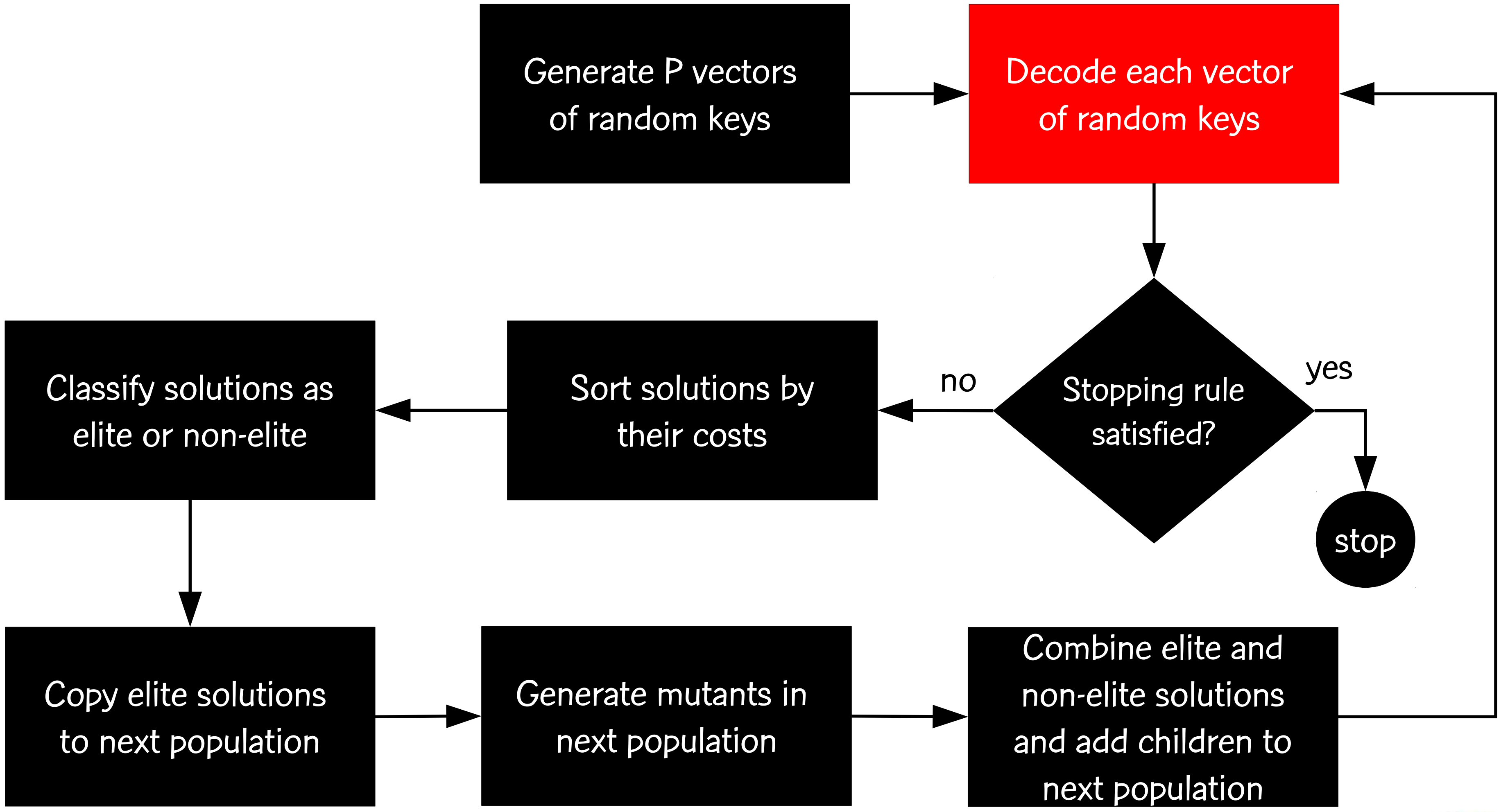
Observations

- Random method: keys are randomly generated so solutions are always vectors of random keys
- Elitist strategy: best solutions are passed without change from one generation to the next (incumbent is kept)
- 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 Not so in the RKGA of Bean.

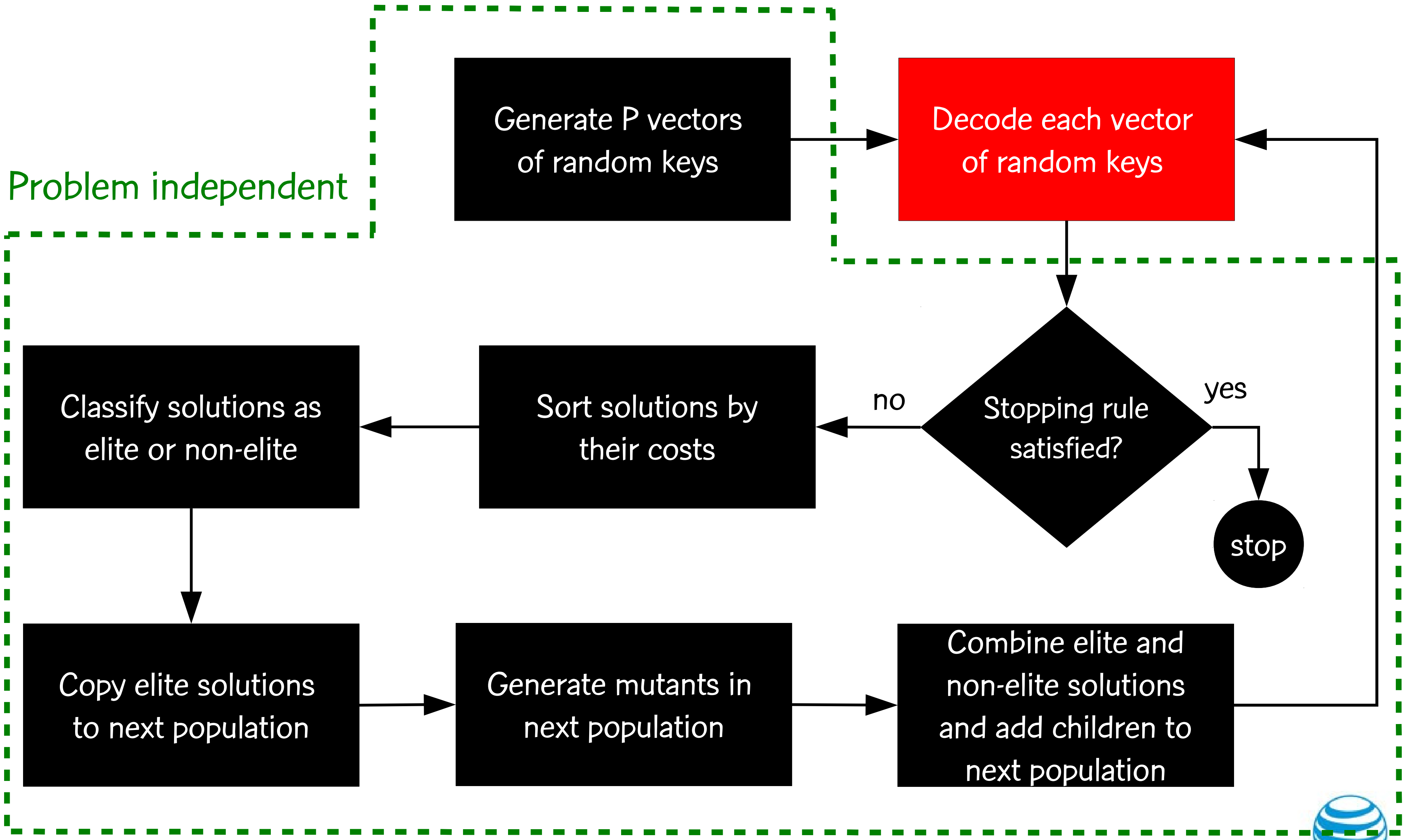
Observations

- Random method: keys are randomly generated so solutions are always vectors of random keys
- Elitist strategy: best solutions are passed without change from one generation to the next (incumbent is kept)
- 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 Not so in the RKGA of Bean.
- No mutation in crossover: mutants are used instead (they play same role as mutation in GAs … help escape local optima)

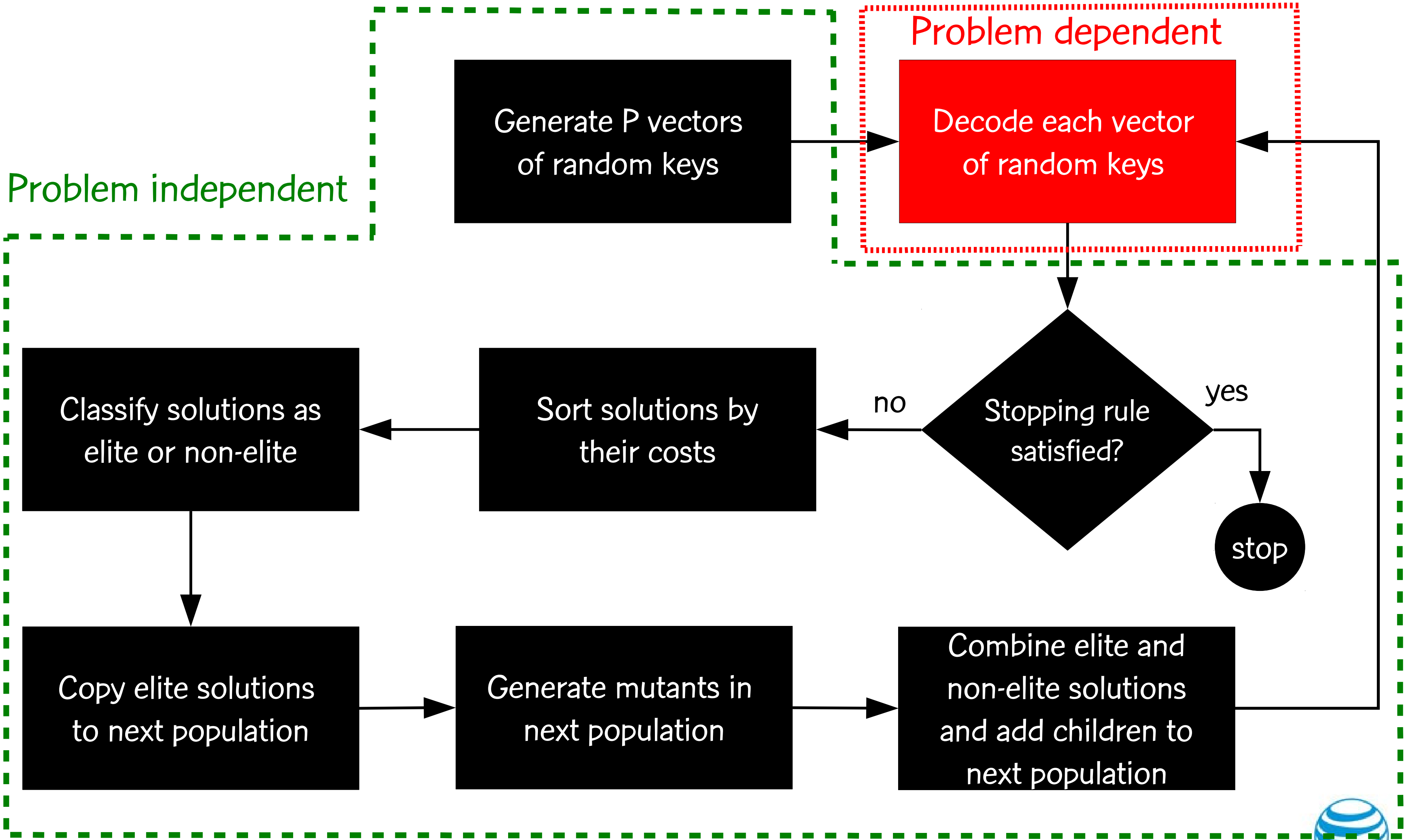
Framework for biased random-key genetic algorithms


Generate P vectors of random keys
Decode each vector of random keys
Stopping rule satisfied?
yes
stop
no
Sort solutions by their costs
Classify solutions as elite or non-elite
Copy elite solutions to next population
Generate mutants in next population
Combine elite and non-elite solutions and add children to next population

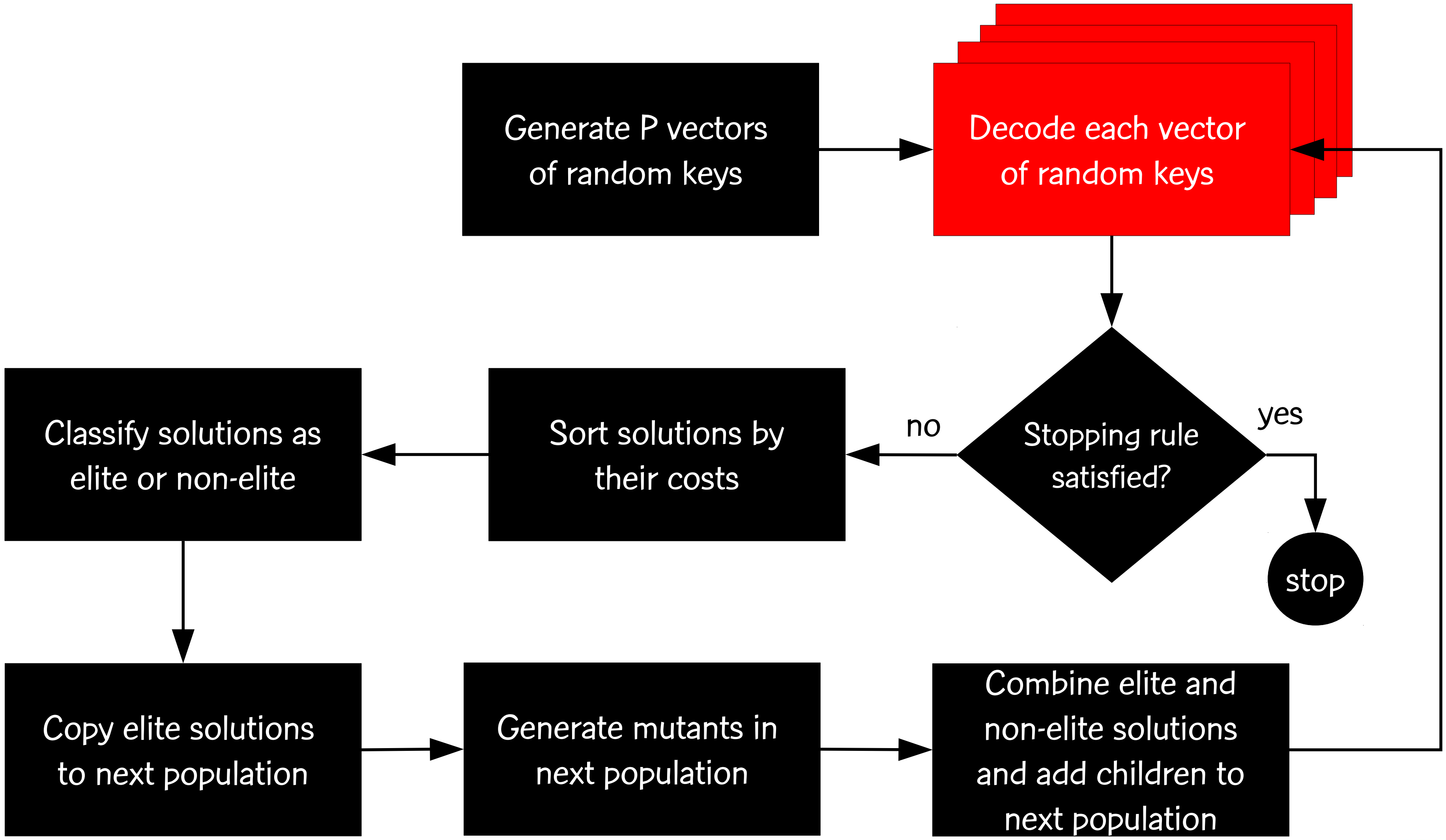
Framework for biased random-key genetic algorithms


Problem independent
Generate P vectors of random keys
Decode each vector of random keys
Stopping rule satisfied?
yes
no
stop
Sort solutions by their costs
Classify solutions as elite or non-elite
Copy elite solutions to next population
Generate mutants in next population
Combine elite and non-elite solutions and add children to next population

Framework for biased random-key genetic algorithms


Problem independent
Problem dependent
Generate P vectors of random keys
Decode each vector of random keys
Stopping rule satisfied?
no
yes
stop
Sort solutions by their costs
Classify solutions as elite or non-elite
Copy elite solutions to next population
Generate mutants in next population
Combine elite and non-elite solutions and add children to next population

Decoding of random key vectors can be done in parallel


Generate P vectors of random keys
Decode each vector of random keys
Stopping rule satisfied?
yes
stop
no
Sort solutions by their costs
Classify solutions as elite or non-elite
Copy elite solutions to next population
Generate mutants in next population
Combine elite and non-elite solutions and add children to next population

brkgaAPI: A C++ API for BRKGA

- Efficient and easy-to-use object oriented application programming interface (API) for the algorithmic framework of BRKGA.

brkgaAPI: A C++ API for BRKGA

- Efficient and easy-to-use object oriented application programming interface (API) for the algorithmic framework of BRKGA.
- Cross-platform library handles large portion of problem independent modules that make up the framework, e.g.
  - population management
  - evolutionary dynamics

brkgaAPI: A C++ API for BRKGA

- Efficient and easy-to-use object oriented application programming interface (API) for the algorithmic framework of BRKGA.
- Cross-platform library handles large portion of problem independent modules that make up the framework, e.g.
  - population management
  - evolutionary dynamics
- Implemented in C++ and may benefit from shared-memory parallelism if available.

brkgaAPI: A C++ API for BRKGA

- Efficient and easy-to-use object oriented application programming interface (API) for the algorithmic framework of BRKGA.
- Cross-platform library handles large portion of problem independent modules that make up the framework, e.g.
  - population management
  - evolutionary dynamics
- Implemented in C++ and may benefit from shared-memory parallelism if available.
- User only needs to implement problem-dependent decoder.

brkgaAPI: A C++ API for BRKGA

Paper: Rodrigo F. Toso and M.G.C.R., "A C++ Application Programming Interface for Biased Random-Key Genetic Algorithms," AT&T Labs Technical Report, Florham Park, August 2011.

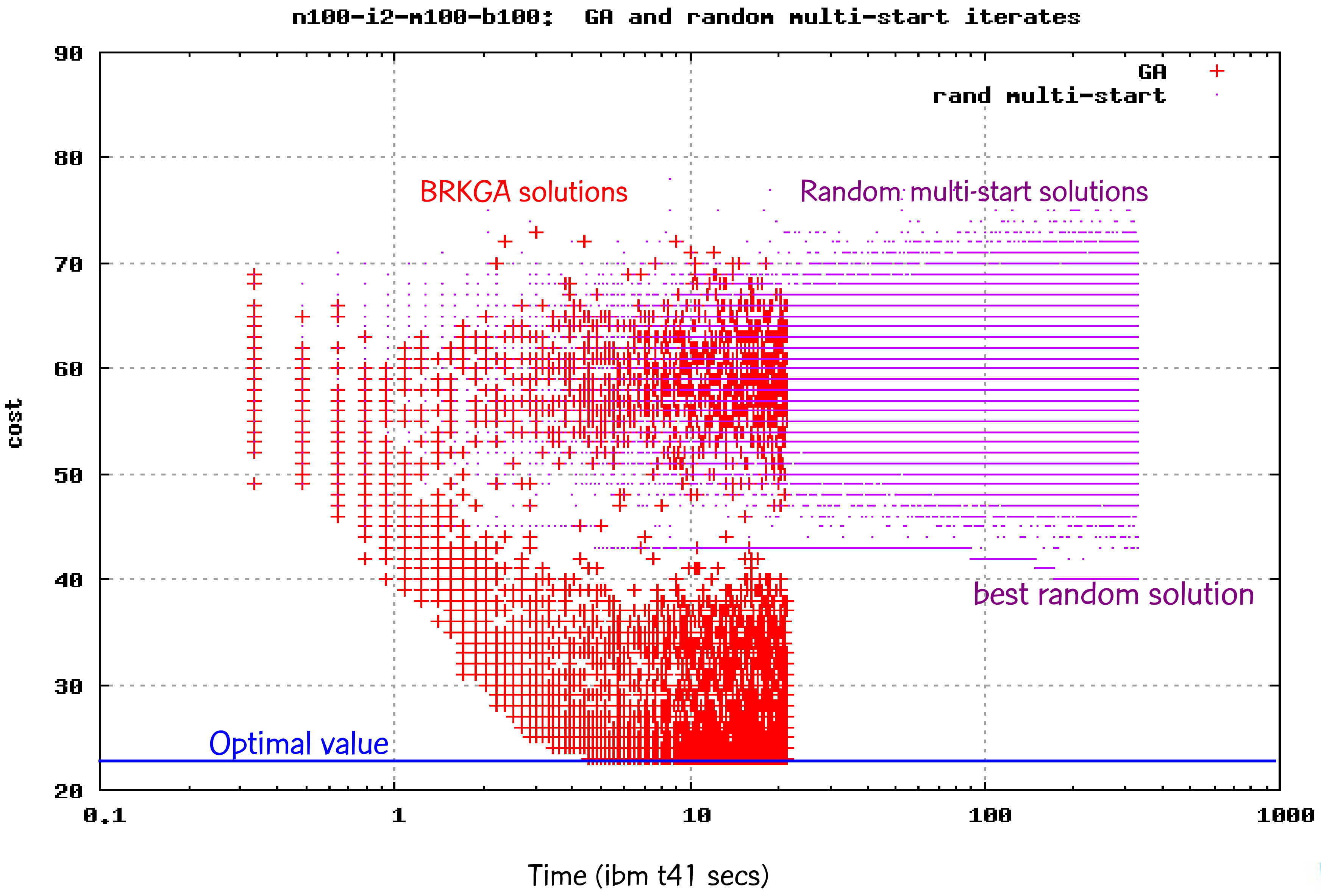
Software: http://www.research.att.com/~mgcr/src/brkgaAPI

Is a BRKGA any different from applying the decoder to random keys?

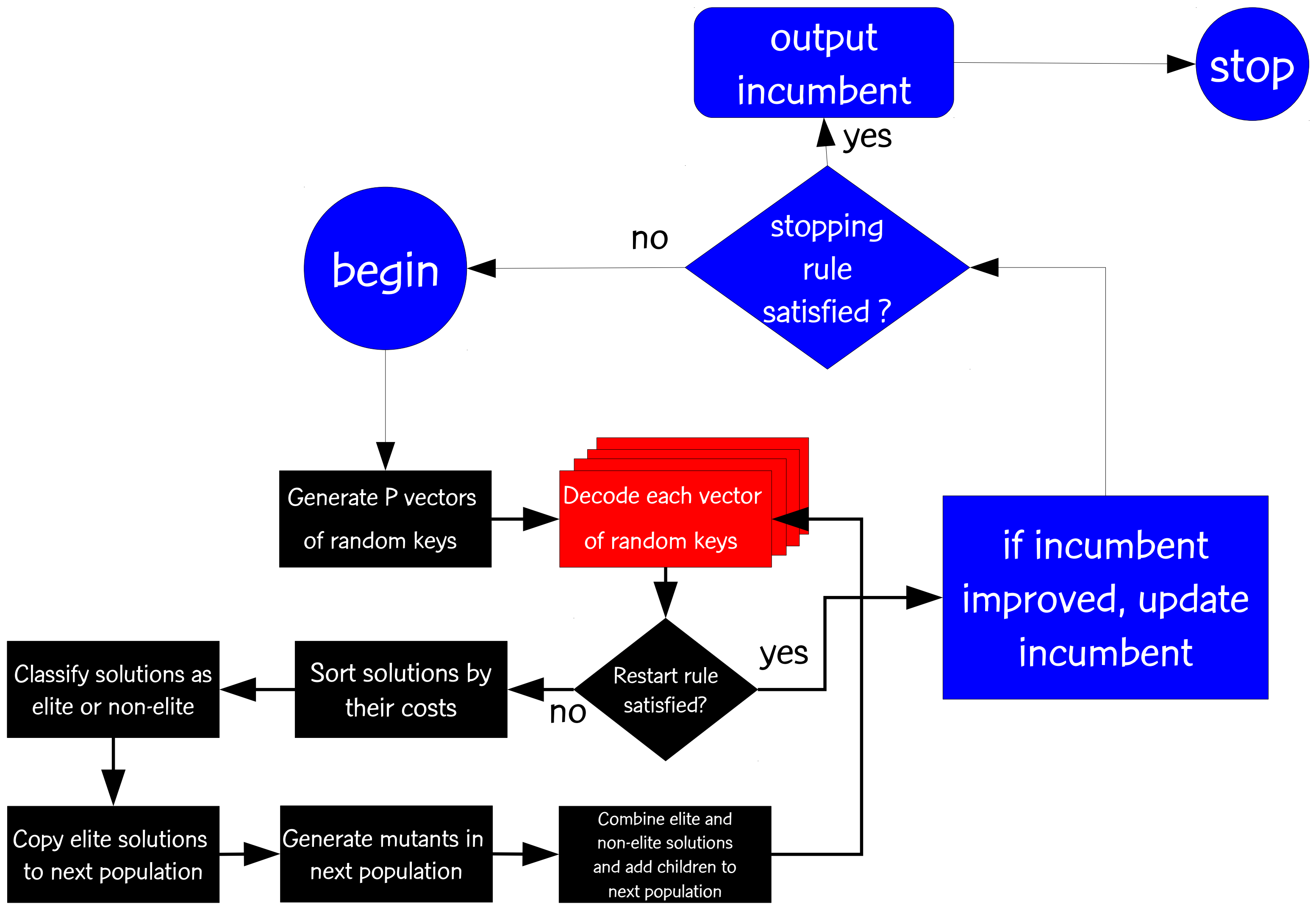
- Simulate a random multi-start decoding method with a BRKGA by setting size of elite partition to 1 and number of mutants to P–1
- Each iteration, best solution is maintained in elite set and P–1 random key vectors are generated as mutants … no mating is done since population already has P individuals

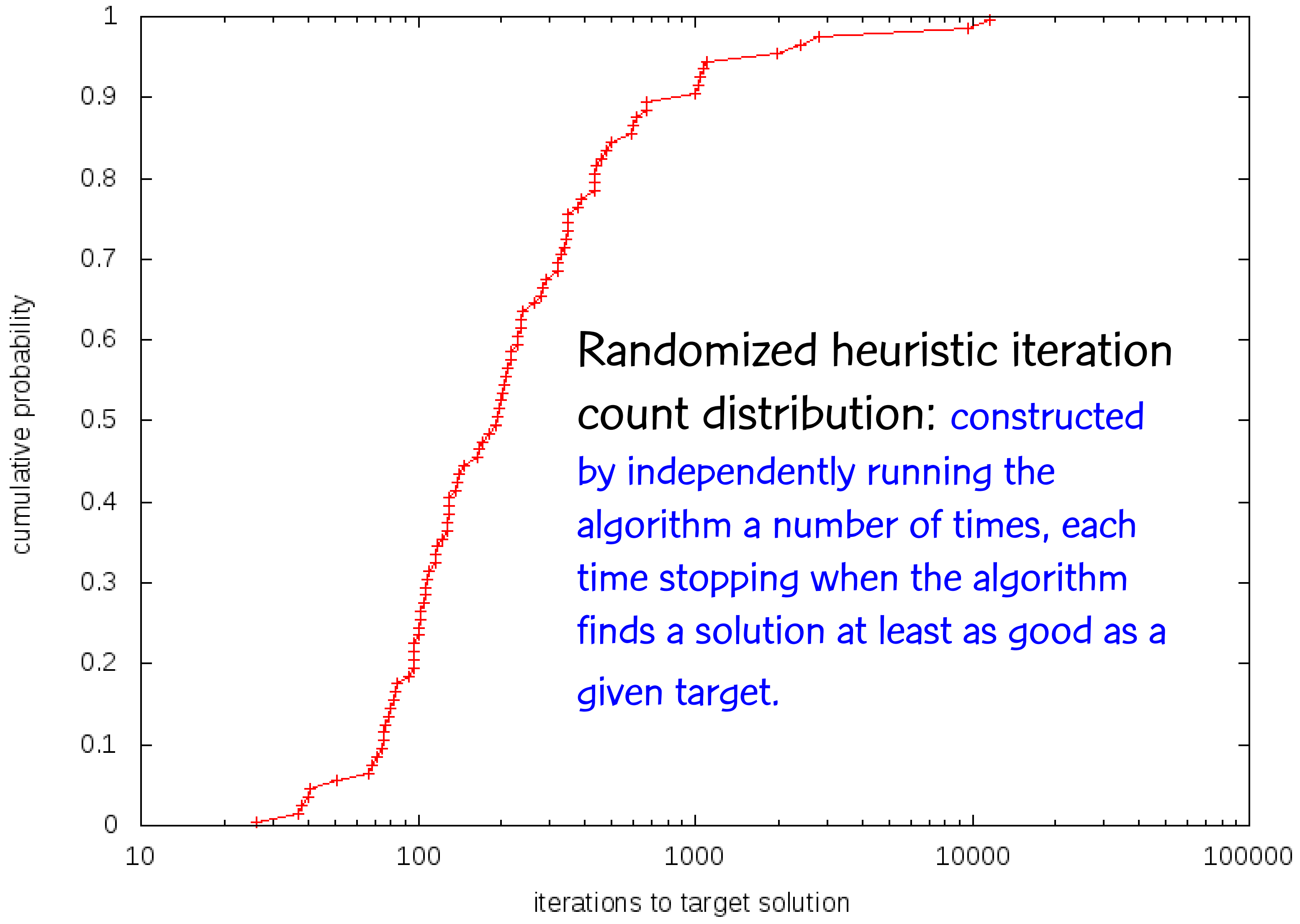
Network monitor location problem (opt = 23)

solution

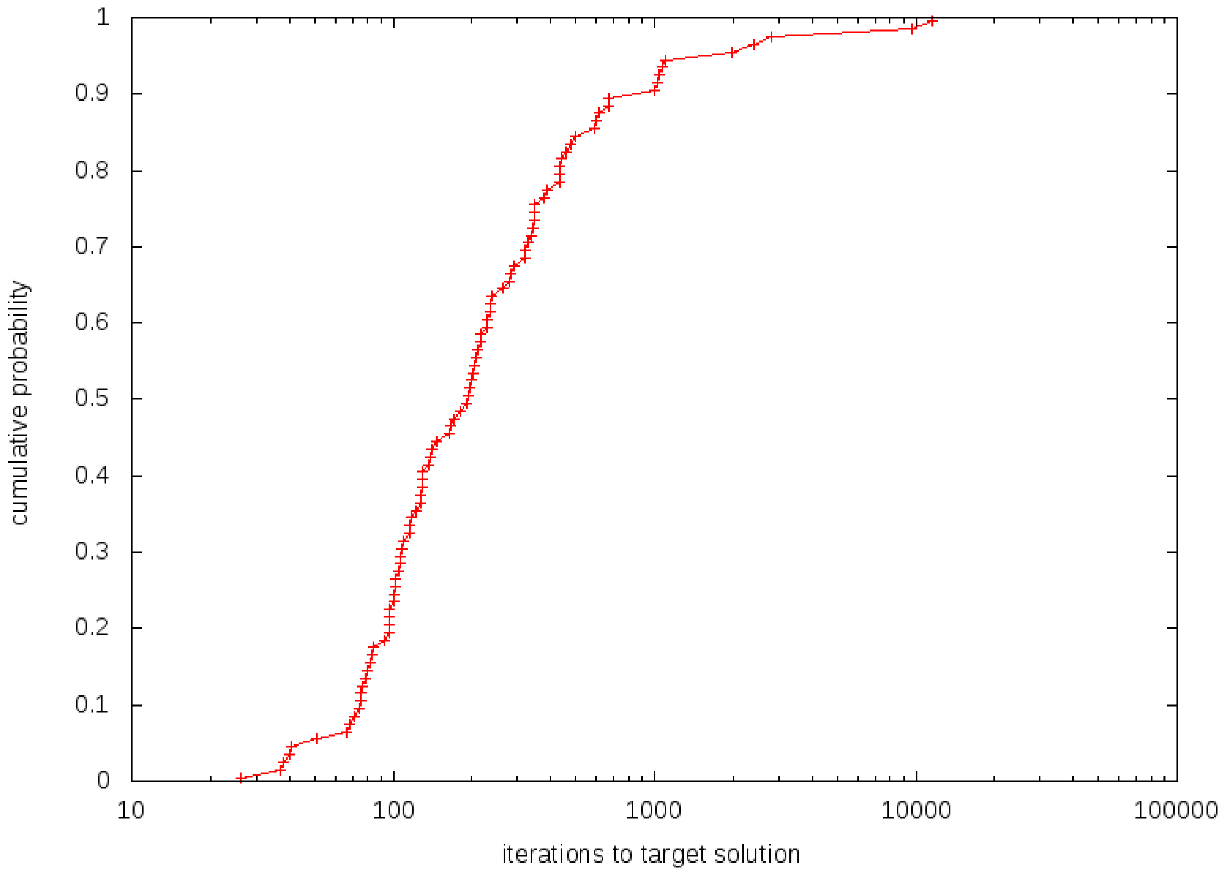


BRKGA in multi-start strategy

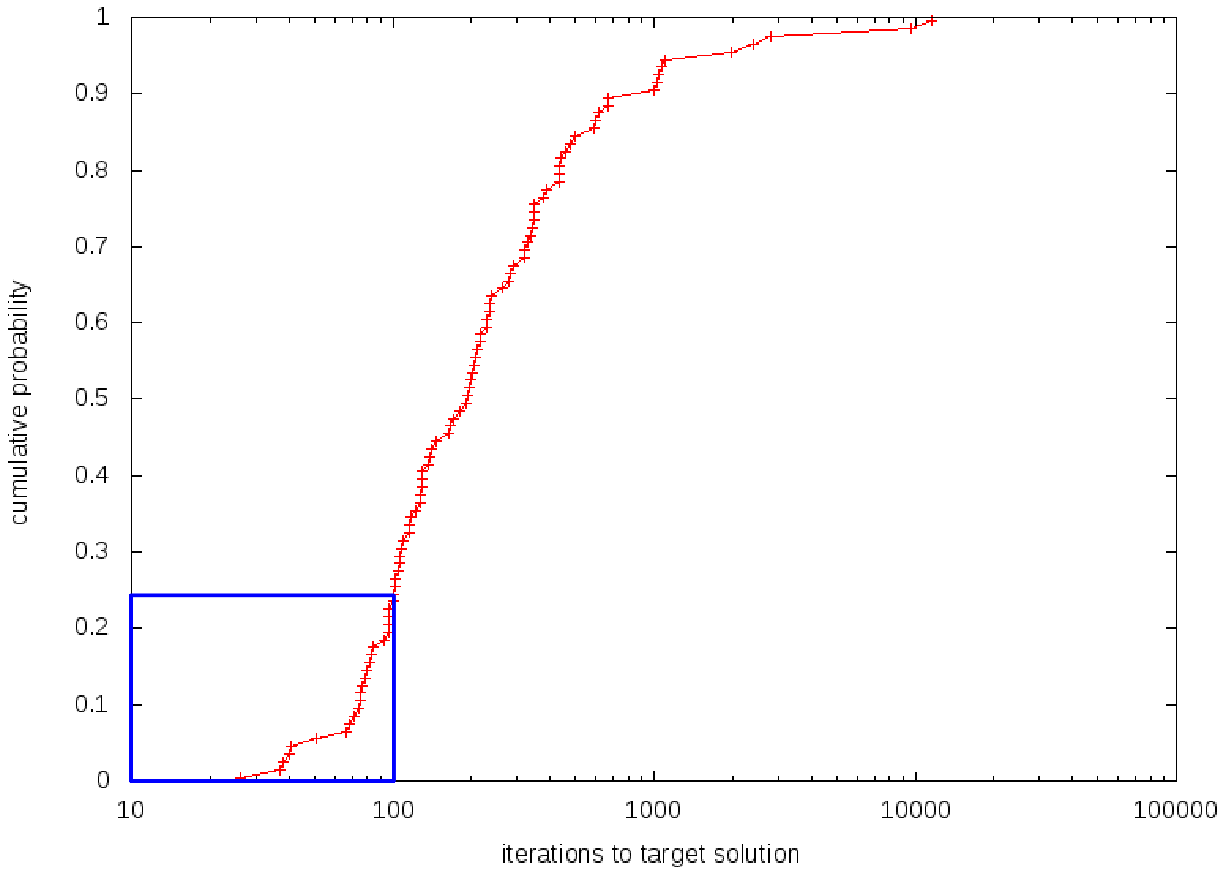

output incumbent
stop
yes
no
stopping rule satisfied ?
begin
Generate P vectors of random keys
Decode each vector of random keys
if incumbent improved, update incumbent
yes
Restart rule satisfied?
no
Sort solutions by their costs
Classify solutions as elite or non-elite
Copy elite solutions to next population
Generate mutants in next population
Combine elite and non-elite solutions and add children to next population



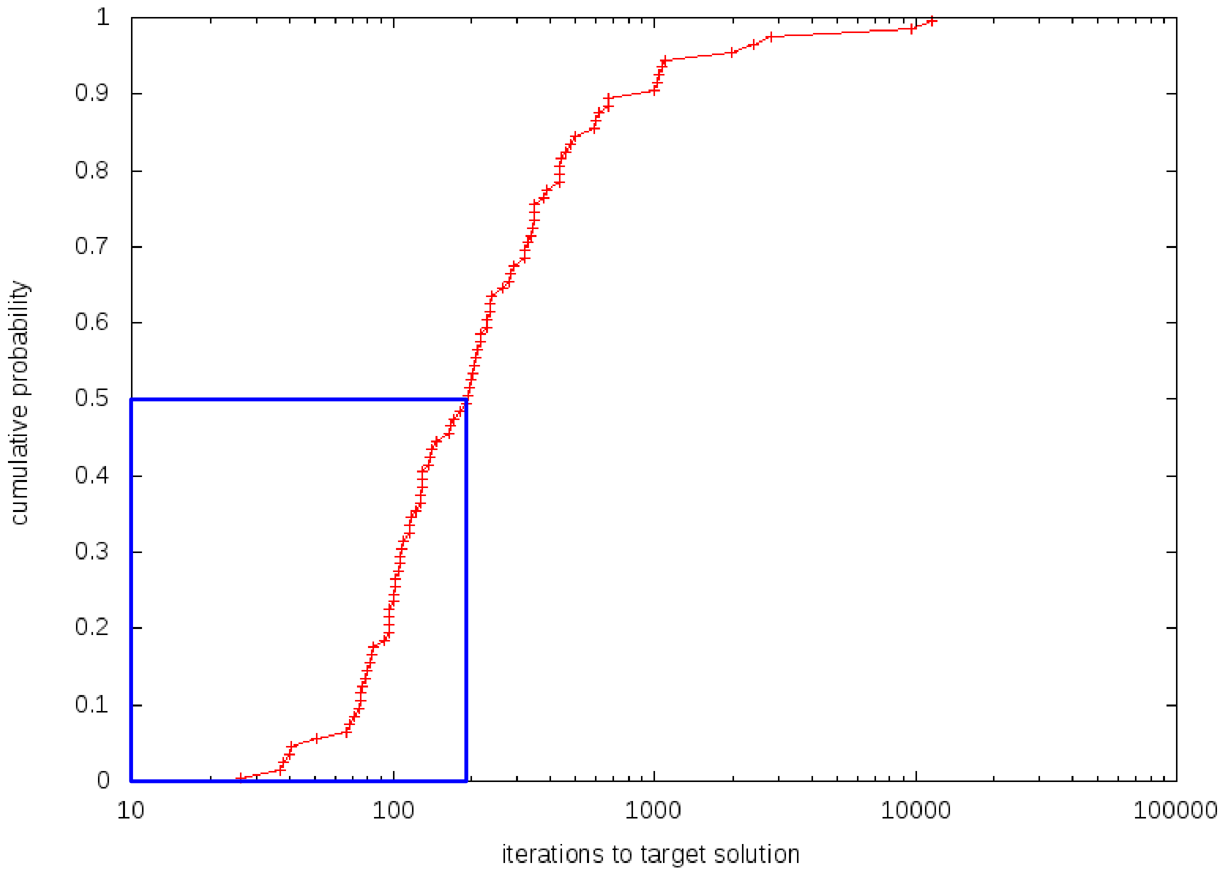
Randomized heuristic iteration count distribution: constructed by independently running the algorithm a number of times, each time stopping when the algorithm finds a solution at least as good as a given target.



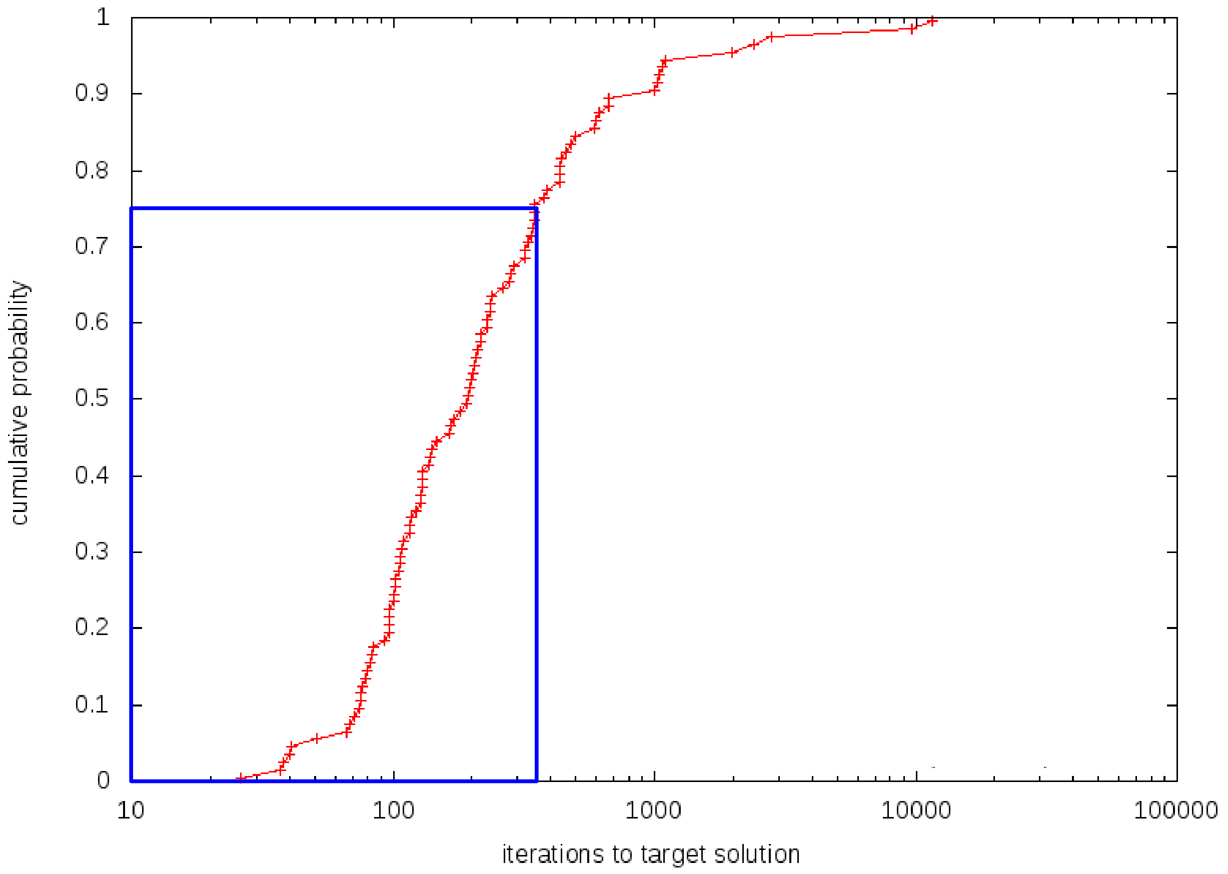
In most of the independent runs, the algorithm finds the target solution in relatively few iterations:



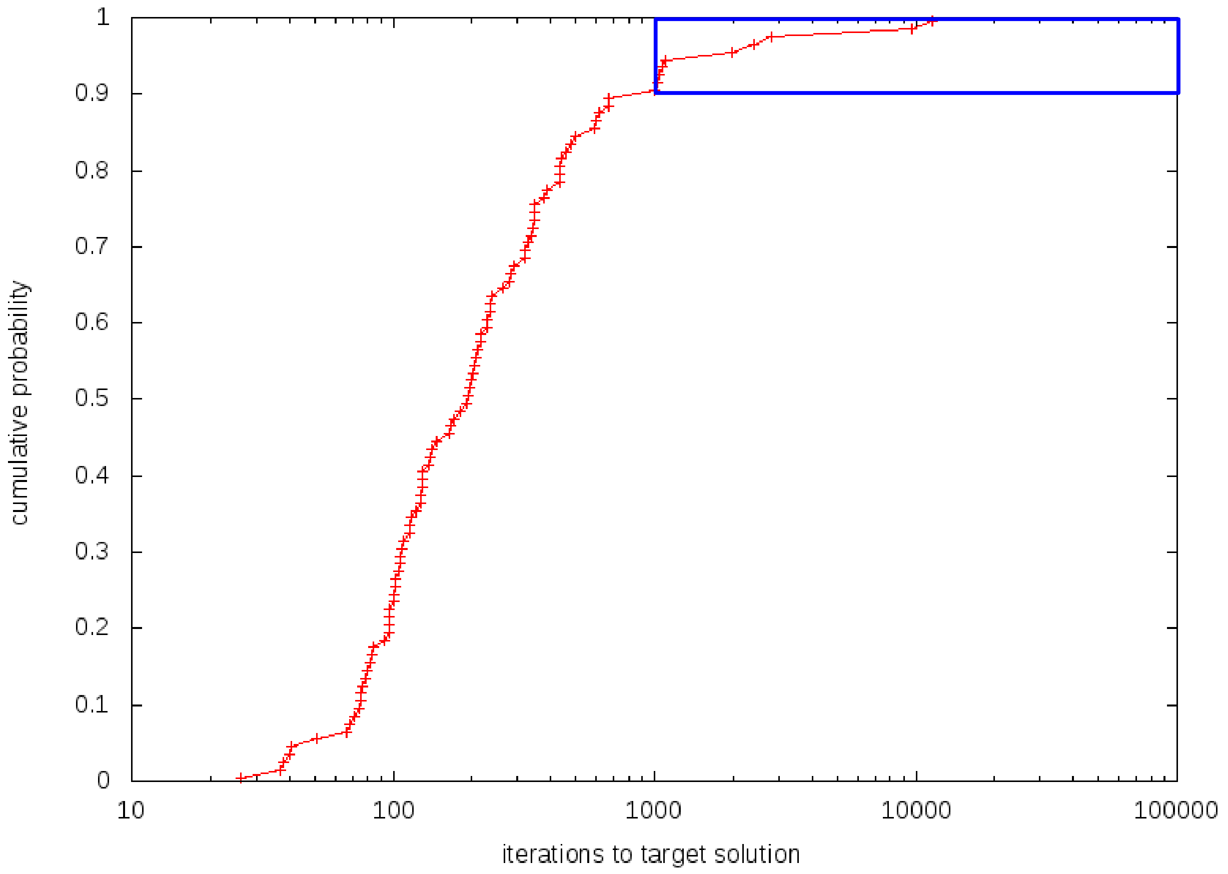
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 25% of the runs take fewer than 101 iterations



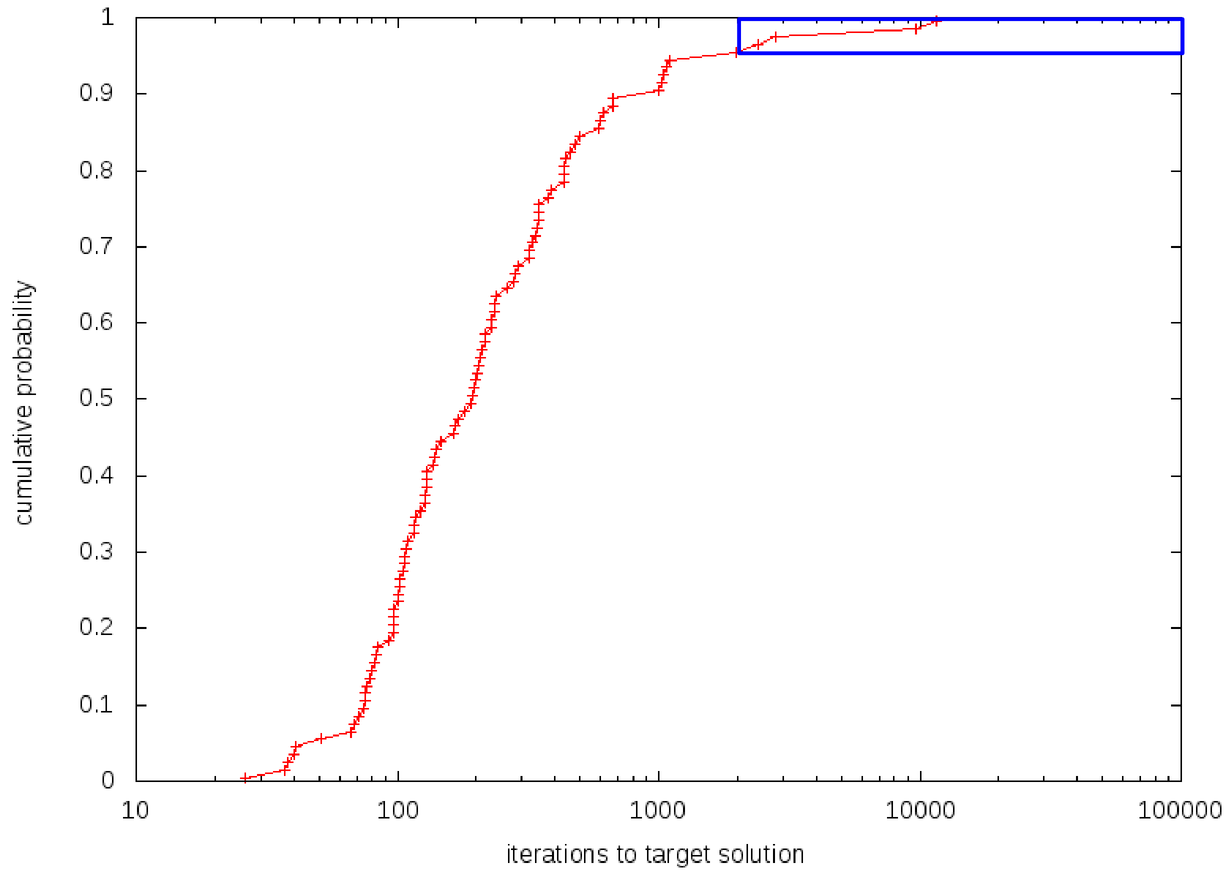
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 50% of the runs take fewer than 192 iterations



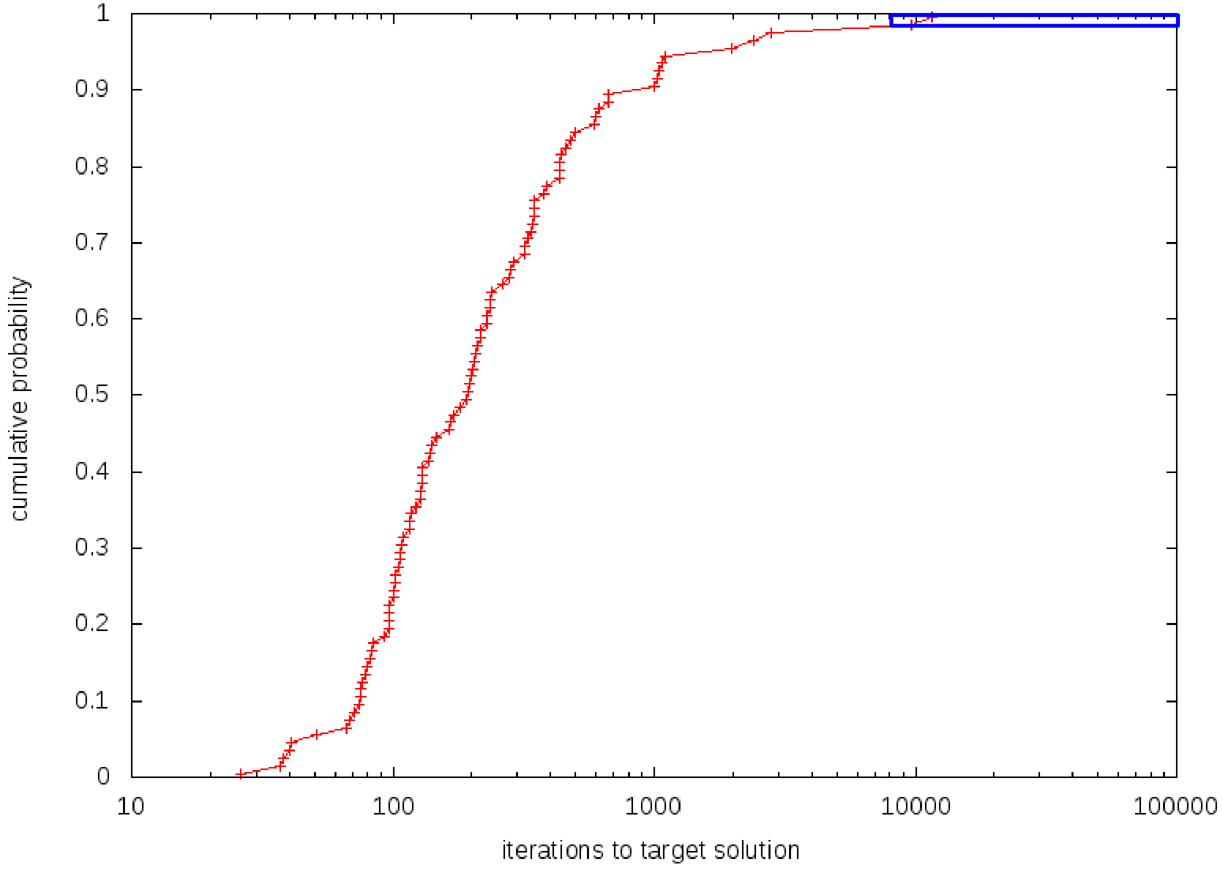
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 75% of the runs take fewer than 345 iterations



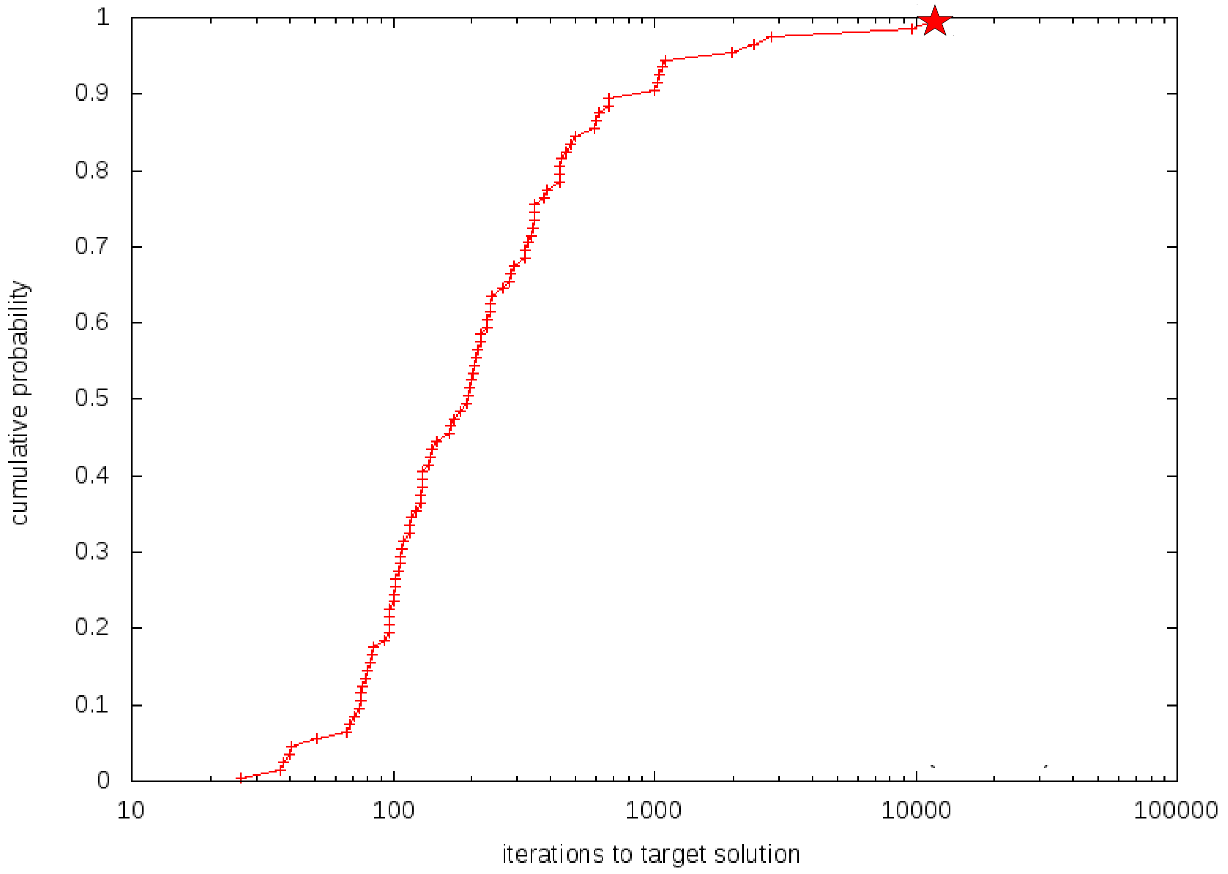
However, some runs take much longer: 10% of the runs take over 1000 iterations



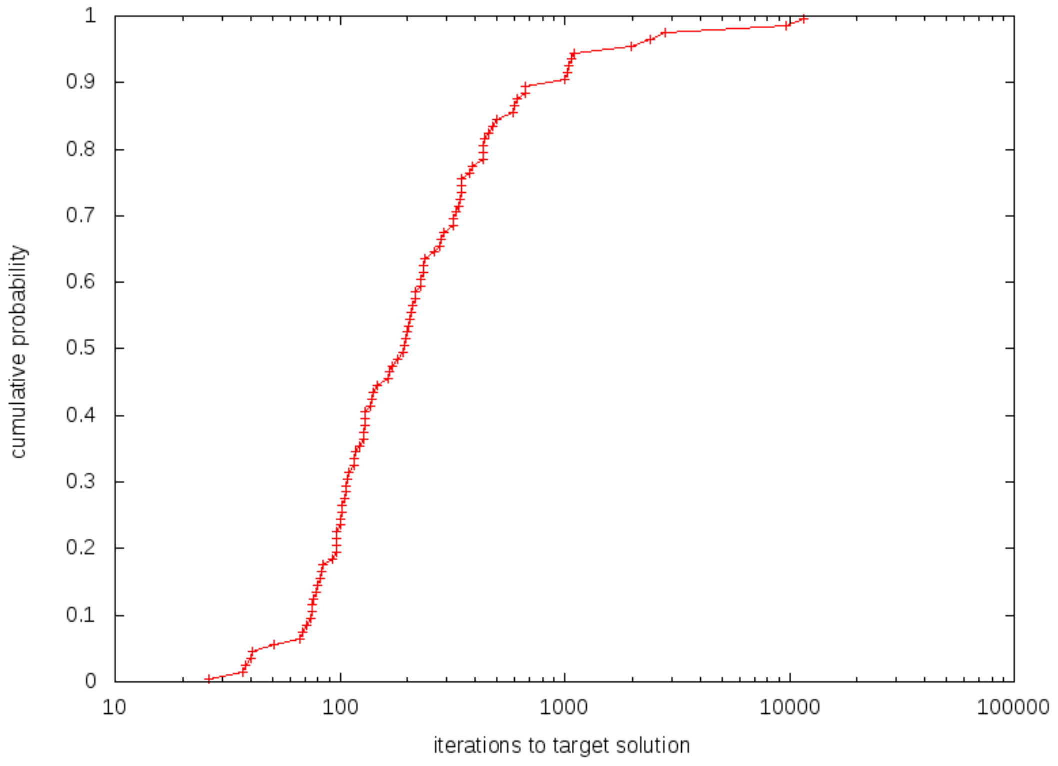
However, some runs take much longer: 5% of the runs take over 2000 iterations



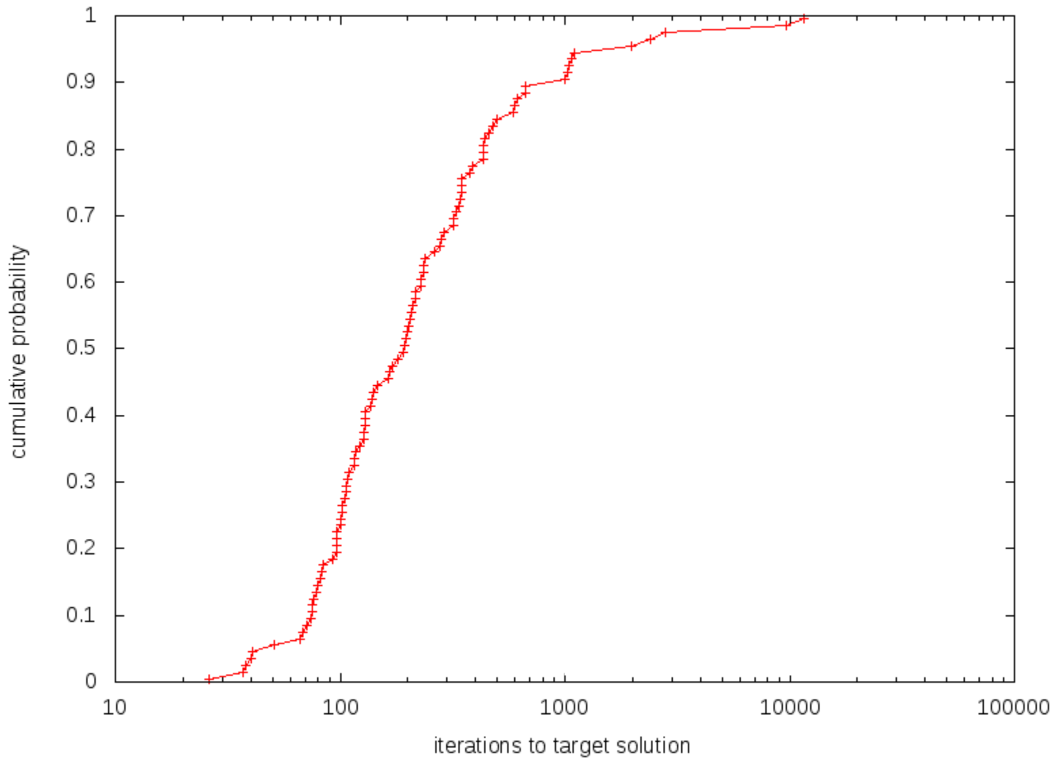
However, some runs take much longer: 2% of the runs take over 9715 iterations



However, some runs take much longer: the longest run took 11607 iterations



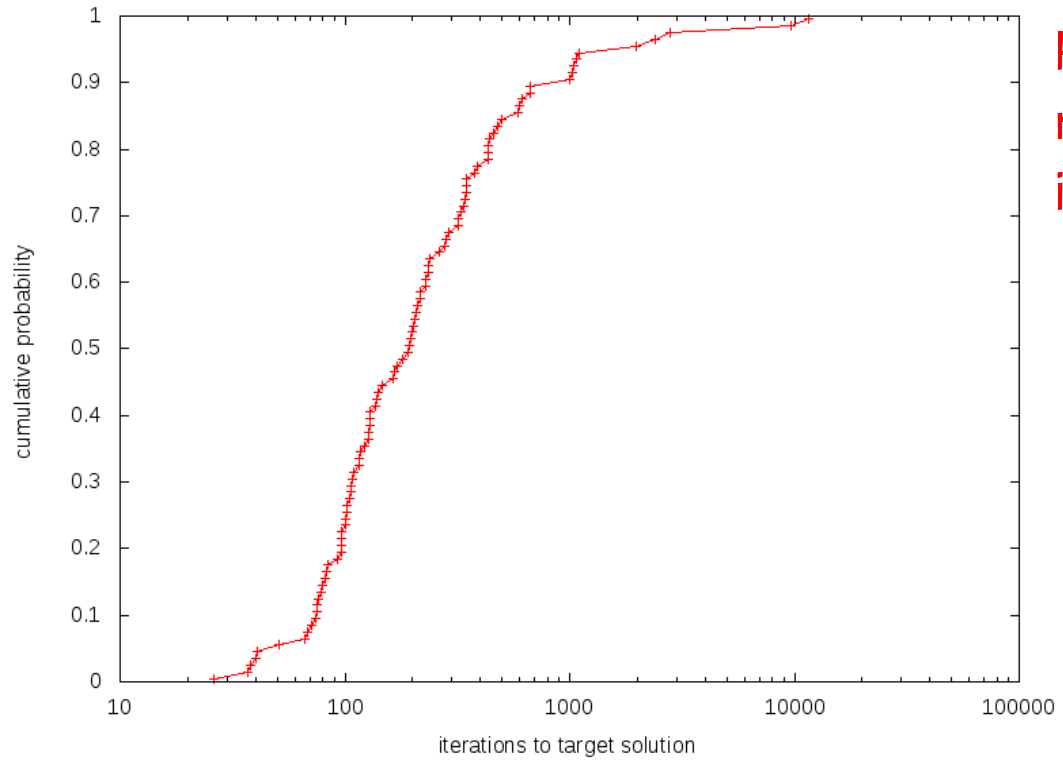
Probability that algorithm will take over 345 iterations: 25% = 1/4



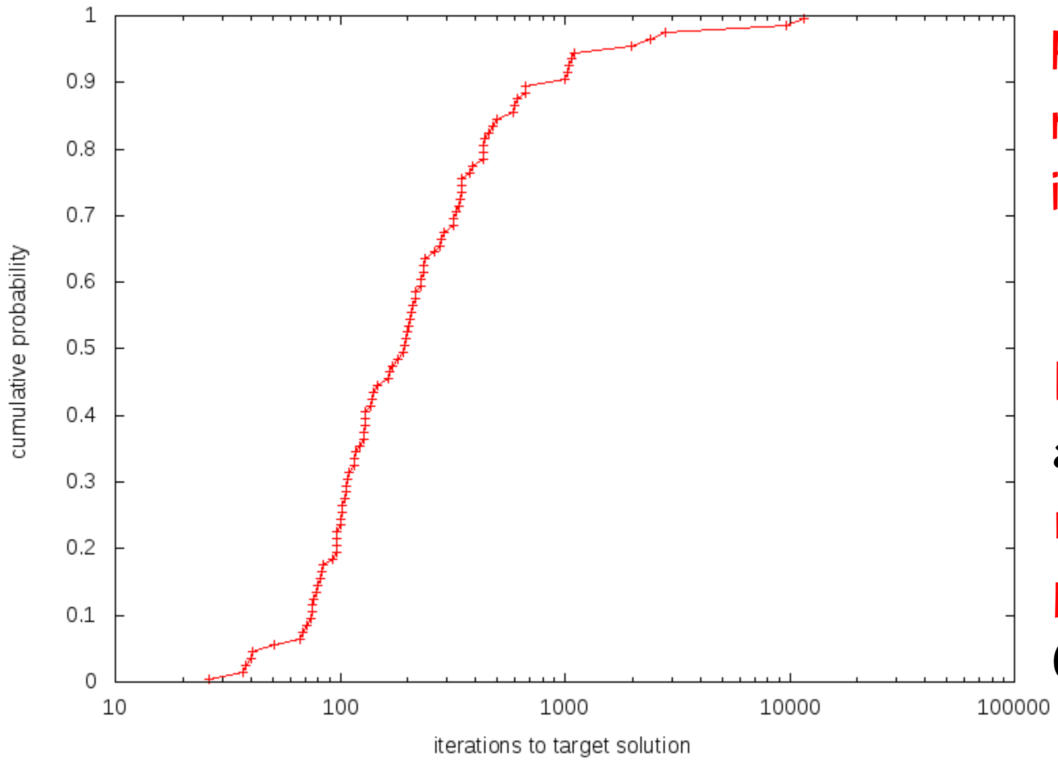
Probability that algorithm will take over 345 iterations: 25% = 1/4

By restarting algorithm after 345 iterations, probability that new run will take over 690 iterations: 25% = 1/4

Probability that algorithm with restart will take over 690 iterations: probability of taking over 345 X probability of taking over 690 iterations given it took over 345 = ¼ x ¼ = $1/4^2$

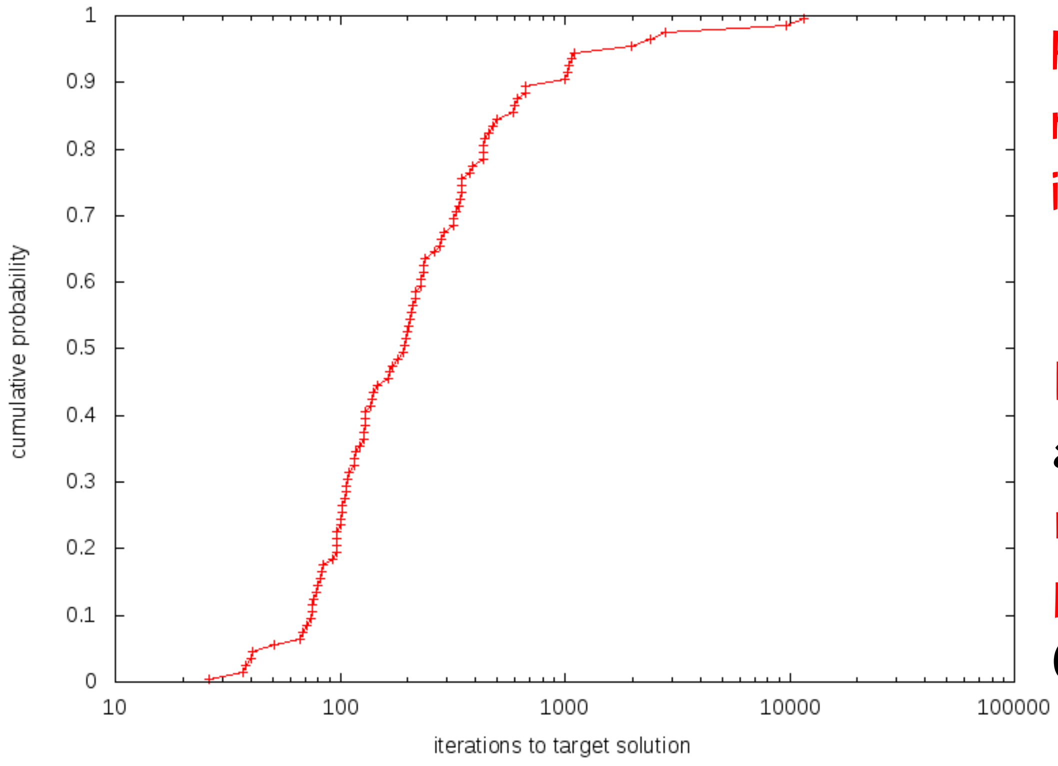


Probability that algorithm will still be running after K periods of 345 iterations: $1/4^K$



Probability that algorithm will still be running after K periods of 345 iterations: $1/4^K$

For example, probability that algorithm with restart will still be running after 1725 iterations (5 periods of 345 iterations): $1/4^5 \cong 0.0977\%$



Probability that algorithm will still be running after K periods of 345 iterations: $1/4^K$

For example, probability that algorithm with restart will still be running after 1725 iterations (5 periods of 345 iterations): $1/4^5 \cong$ 0.0977%

This is much less than the 5% probability that the algorithm without restart will take over 2000 iterations.

Restart strategies

- First proposed by Luby et al. (1993)
- They define a restart strategy as a finite sequence of time intervals $S = \{\tau_1, \tau_2, \tau_3, \dots\}$ which define epochs τ_1, $\tau_1+\tau_2$, $\tau_1+\tau_2+\tau_3$, ... when the algorithm is restarted from scratch.
- Luby et al. (1993) prove that the optimal restart strategy uses $\tau_1 = \tau_2 = \tau_3 = \cdots = \tau^*$, where τ^* is a constant.

Restart strategies

- Luby et al. (1993)
- Kautz et al. (2002)
- Palubeckis (2004)
- Sergienko et al. (2004)
- Nowicki & Smutnicki (2005)
- D'Apuzzo et al. (2006)
- Shylo et al. (2011a)
- Shylo et al. (2011b)
- Resende & Ribeiro (2011)

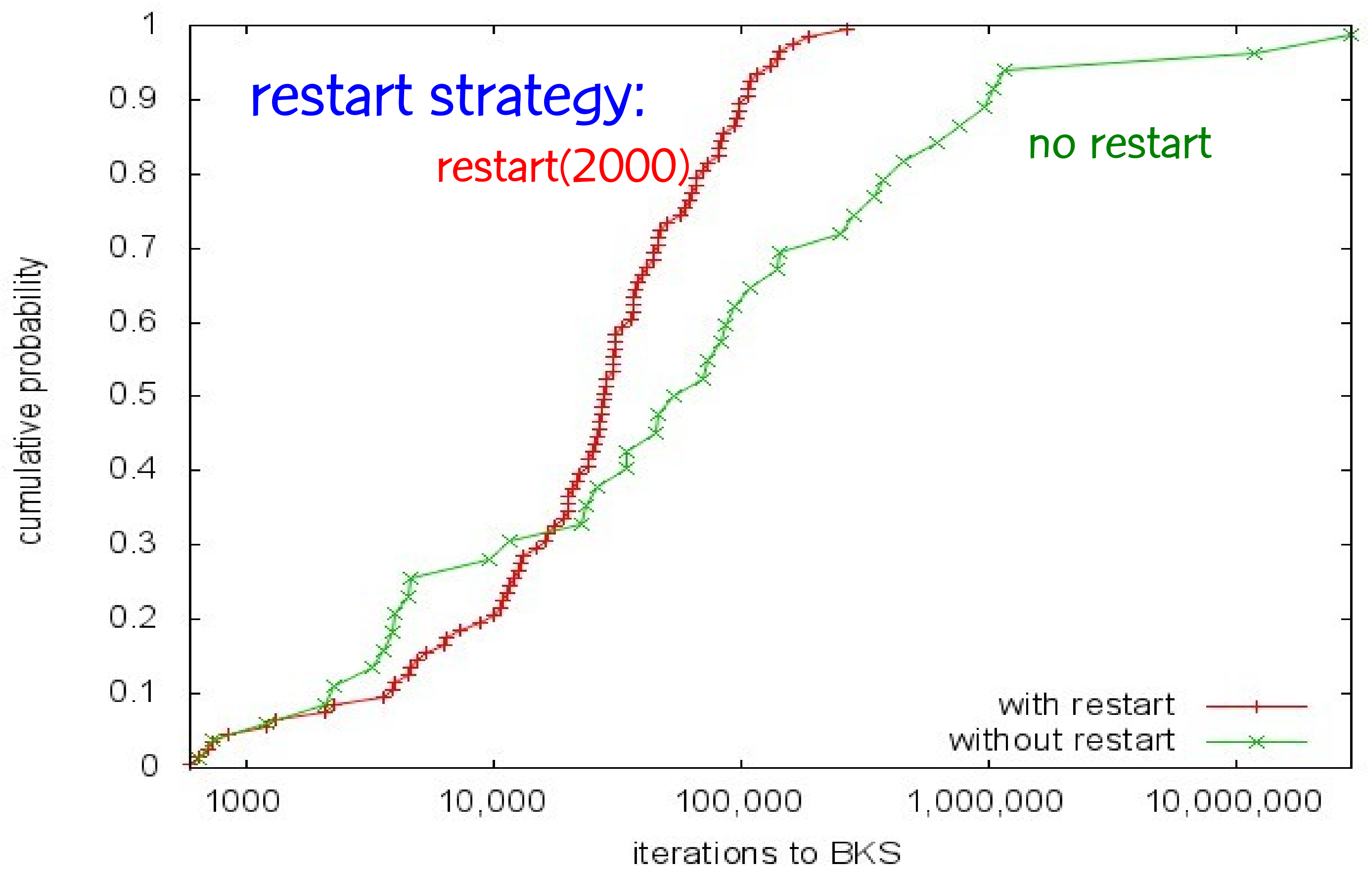
Restart strategy for BRKGA

- Recall the restart strategy of Luby et al. where equal time intervals $\tau_1 = \tau_2 = \tau_3 = \cdots = \tau^*$ pass between restarts.
- Strategy requires τ^* as input.
- Since we have no prior information as to the runtime distribution of the heuristic, we run the risk of:
  - choosing τ^* too small: restart variant may take long to converge
  - choosing τ^* too big: restart variant may become like no-restart variant

Restart strategy for BRKGA

- We conjecture that number of iterations between improvement of the incumbent (best so far) solution varies less w.r.t. heuristic/ instance/ target than run times.
- We propose the following restart strategy: Keep track of the last generation when the incumbent improved and restart BRKGA if K generations have gone by without improvement.
- We call this strategy restart(K)

Example of restart strategy for BRKGA: Load balancing



Specifying a BRKGA

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)

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)

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

Specifying a biased random-key GA

Parameters:

- Size of population
- Parallel population parameters
- Size of elite partition
- Size of mutant set
- Child inheritance probability
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters
- Size of elite partition
- Size of mutant set
- Child inheritance probability
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, $p = 3$, $v = 2$, and $x = 200$
- Size of elite partition
- Size of mutant set
- Child inheritance probability
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, $p = 3$, $v = 2$, and $x = 200$
- Size of elite partition: 15-25% of population
- Size of mutant set
- Child inheritance probability
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, $p = 3$, $v = 2$, and $x = 200$
- Size of elite partition: 15-25% of population
- Size of mutant set: 5-15% of population
- Child inheritance probability
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, p = 3 , v = 2, and x = 200
- Size of elite partition: 15-25% of population
- Size of mutant set: 5-15% of population
- Child inheritance probability: > 0.5, say 0.7
- Restart strategy parameter
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, p = 3 , v = 2, and x = 200
- Size of elite partition: 15-25% of population
- Size of mutant set: 5-15% of population
- Child inheritance probability: > 0.5, say 0.7
- Restart strategy parameter: a function of N, say 2N or 10N
- Stopping criterion

Specifying a biased random-key GA

Parameters:

- Size of population: a function of N, say N or 2N
- Parallel population parameters: say, $p = 3$, $v = 2$, and $x = 200$
- Size of elite partition: 15-25% of population
- Size of mutant set: 5-15% of population
- Child inheritance probability: > 0.5, say 0.7
- Restart strategy parameter: a function of N, say 2N or 10N
- Stopping criterion: e.g. time, # generations, solution quality, # generations without improvement

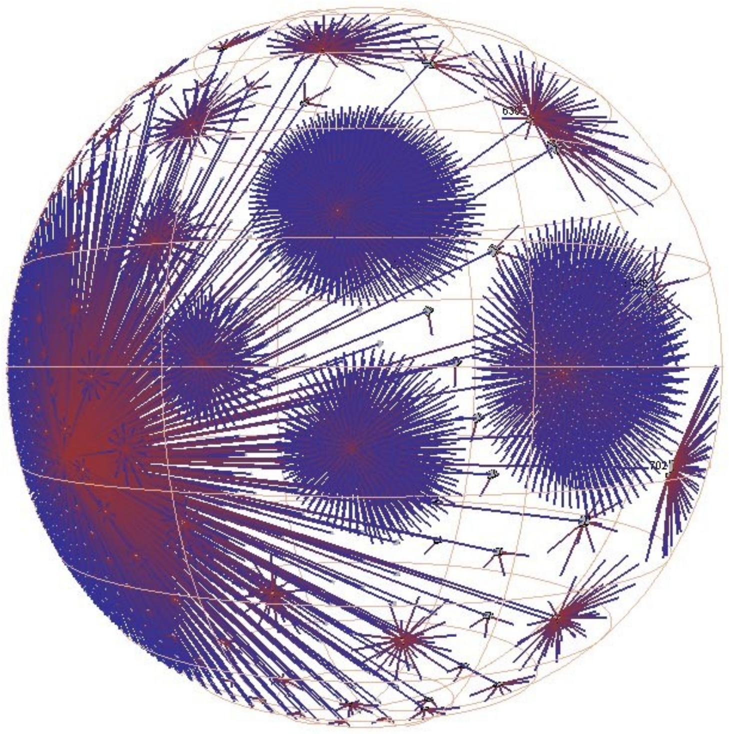
Applications in telecommunications

Three applications in telecommunications

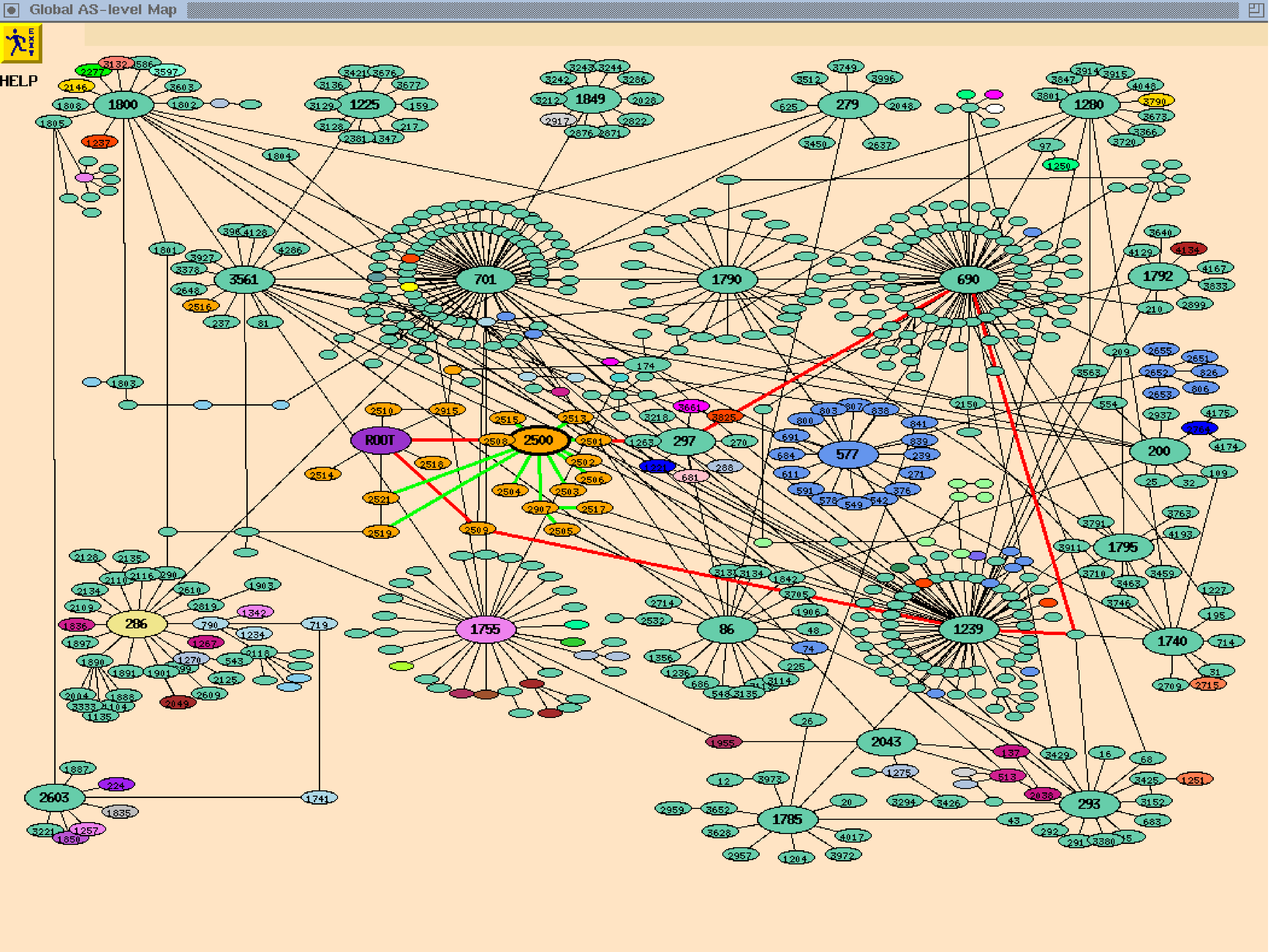
- Routing in IP networks
- Design of survivable IP networks
- Redundant server location for content distribution

OSPF routing in IP networks

The Internet



- The Internet is composed of many (inter-connected) autonomous systems (AS).
- An AS is a network controlled by a single entity, e.g. ISP, university, corporation, country, ...


Global AS-level Map
EXIT
HELP
1800
1225
1849
279
1280
3561
701
1790
690
1792
ROOT
2500
297
577
200
1795
286
1755
86
1239
1740
2043
2603
1785
293
2508
2501
2510
2915
2515
2513
2518
2514
2521
2519
2509
2504
2503
2502
2506
2907
2517
2505
2516
3825
3661
1221
681
288
1263
270
3218
174
803
807
838
800
841
691
839
684
239
611
271
591
578
549
542
376
3132
3586
3597
2277
2146
3603
1802
1808
1805
1237
1804
1801
1803
3421
3676
3136
3677
3129
159
3128
217
2381
1347
3243
3244
3242
3286
3212
2028
2917
2822
2876
2871
3749
3512
3996
625
2048
3450
2637
3914
3915
3847
4048
3801
3790
3673
3366
3720
97
1250
396
4128
4286
3927
3378
2648
237
81
3640
4134
4129
4167
3833
2899
210
209
2655
2651
2652
826
2653
806
3563
554
2150
2937
4175
2764
4174
25
32
109
3763
3791
4193
3911
3710
3463
3459
1227
3746
195
714
31
2709
2715
2128
2135
2110
2116
290
2134
2610
1903
2109
2819
1342
1836
790
719
1234
1897
1267
2118
1890
1270
543
1891
1901
399
2125
2004
1888
2609
3333
4104
2049
1135
2714
2532
3131
3134
1842
3705
1906
48
74
1356
225
1236
686
548
3135
3114
3113
26
1955
137
3429
16
68
1275
513
3425
1251
2038
3152
3294
3426
683
43
292
291
3380
15
12
3973
2959
3652
20
3628
4017
2957
1204
3972
1887
224
1741
1835
3221
1257
1850

Routing

- A packet is sent from a origination router S to a destination router T.
- S and T may be in
  - same AS:
  - different ASes:

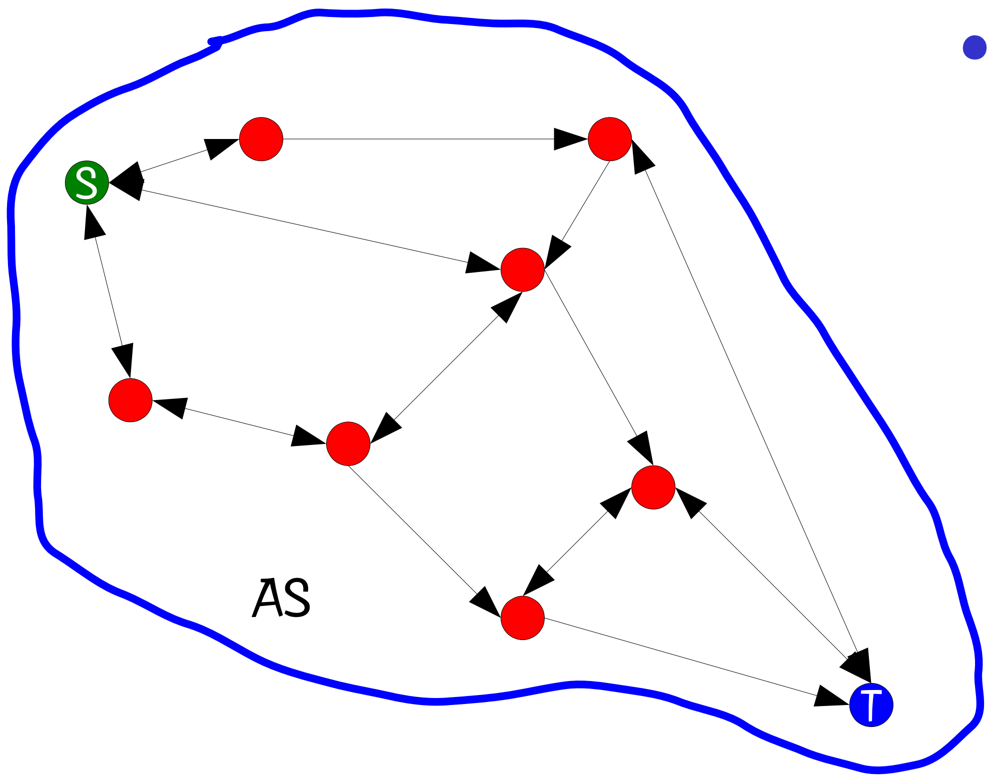
Routing

- A packet is sent from a origination router S to a destination router T.
- S and T may be in
  - same AS: IGP routing
  - different ASes:

Routing

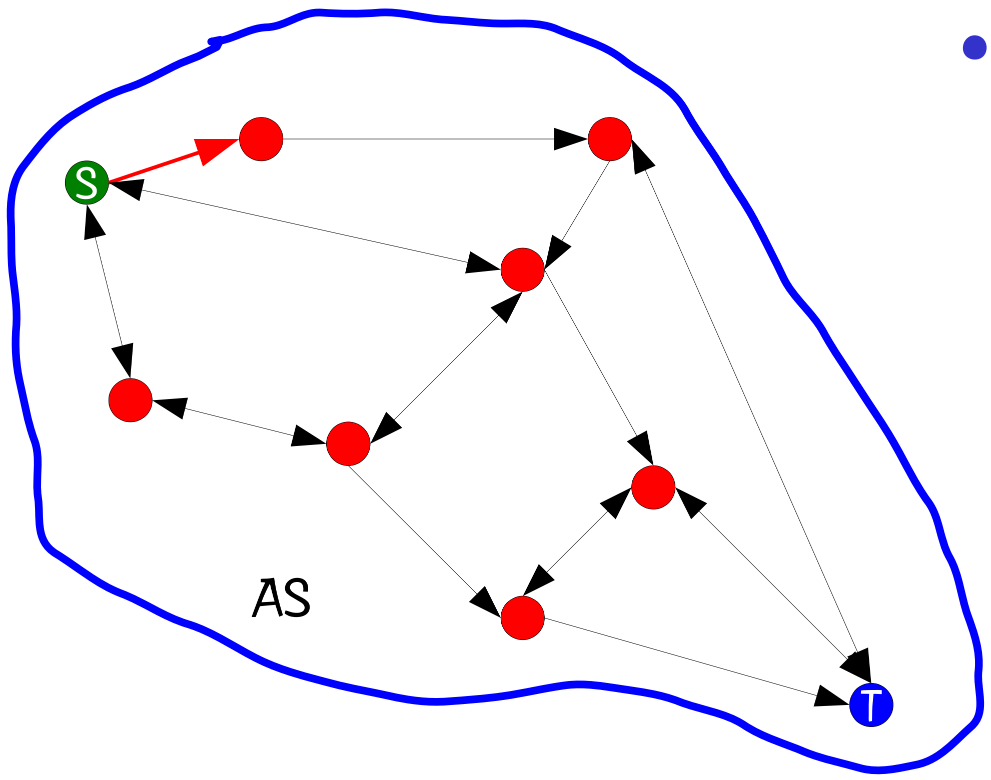
- A packet is sent from a origination router S to a destination router T.
- S and T may be in
  - same AS: IGP routing
  - different ASes: BGP routing

IGP Routing



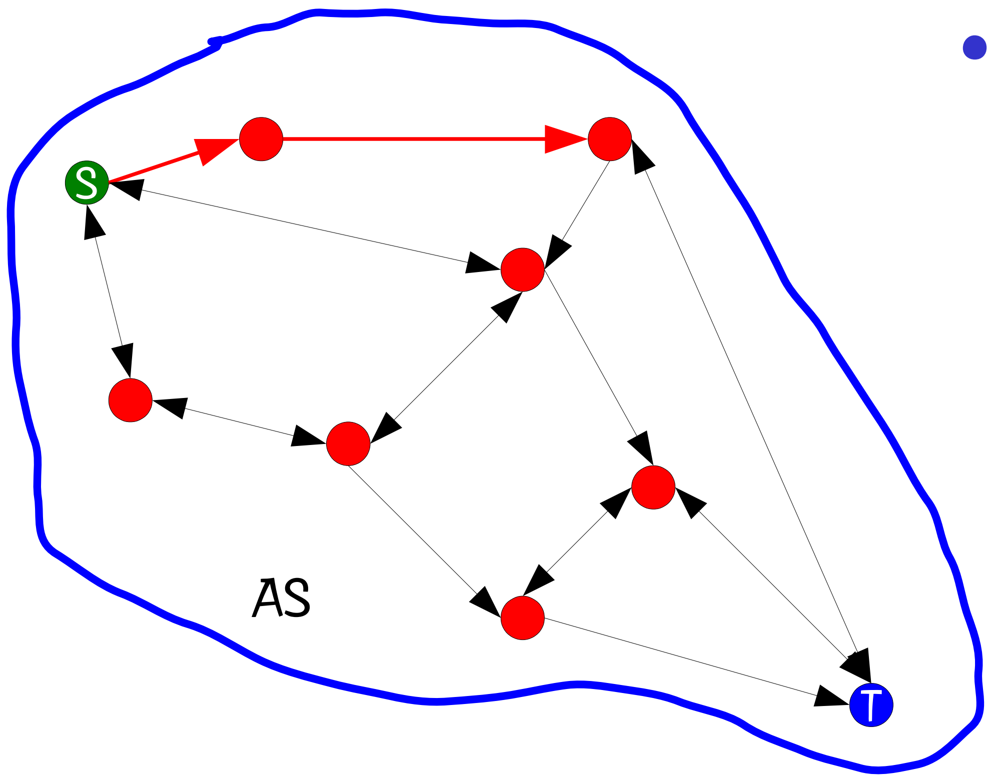
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



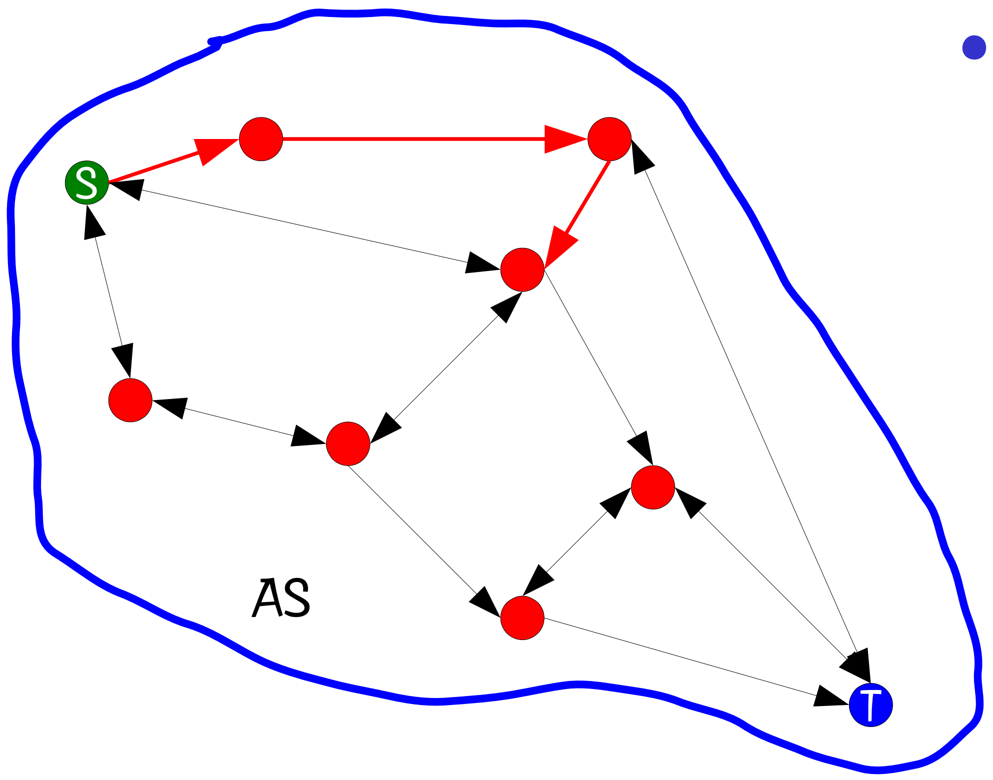
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



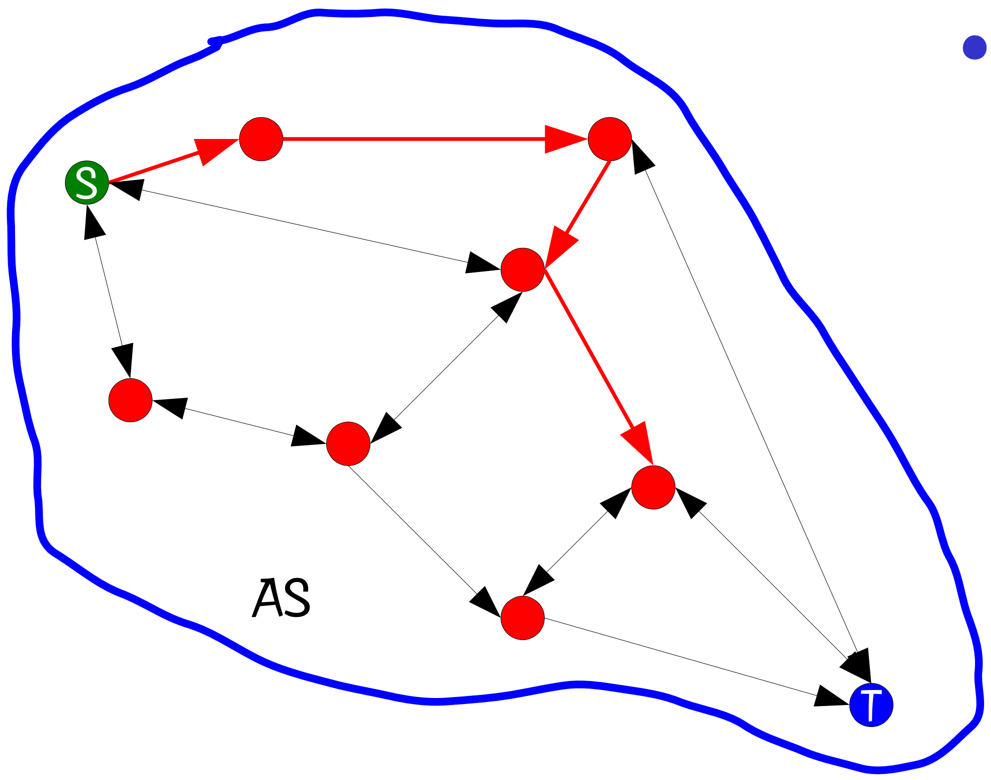
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



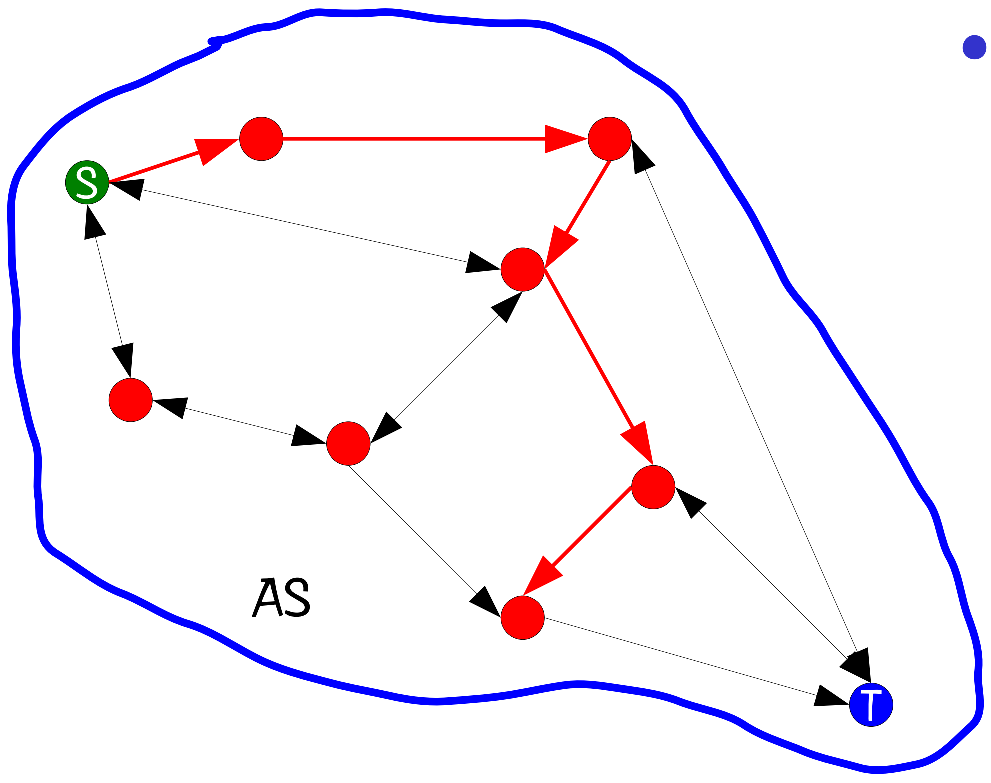
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



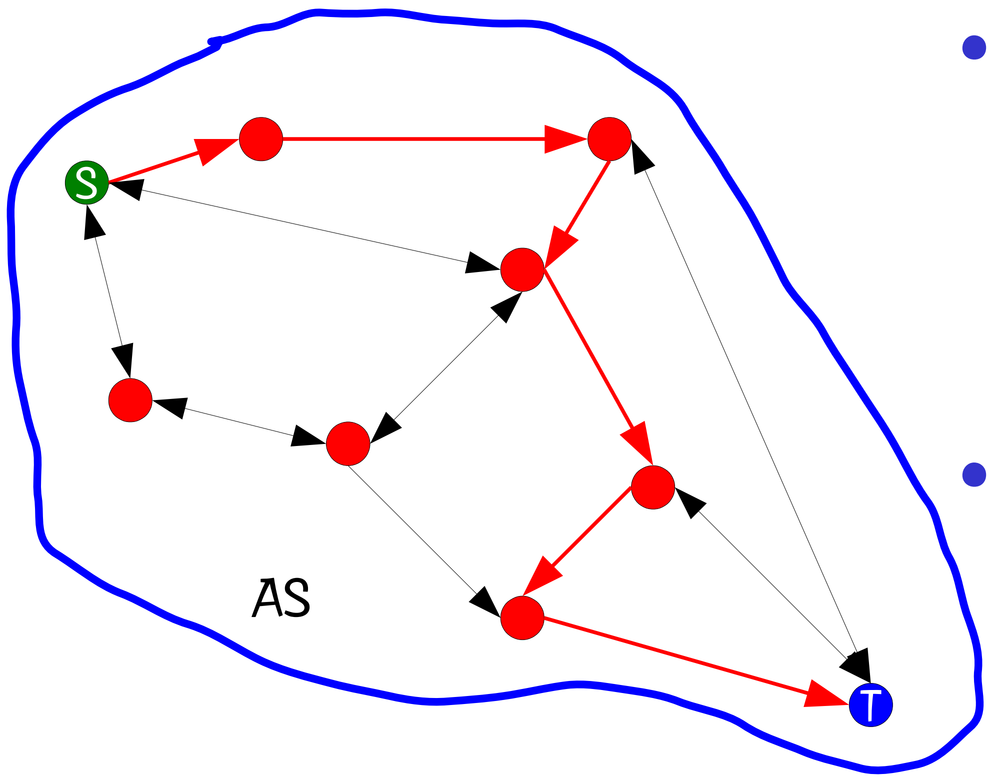
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



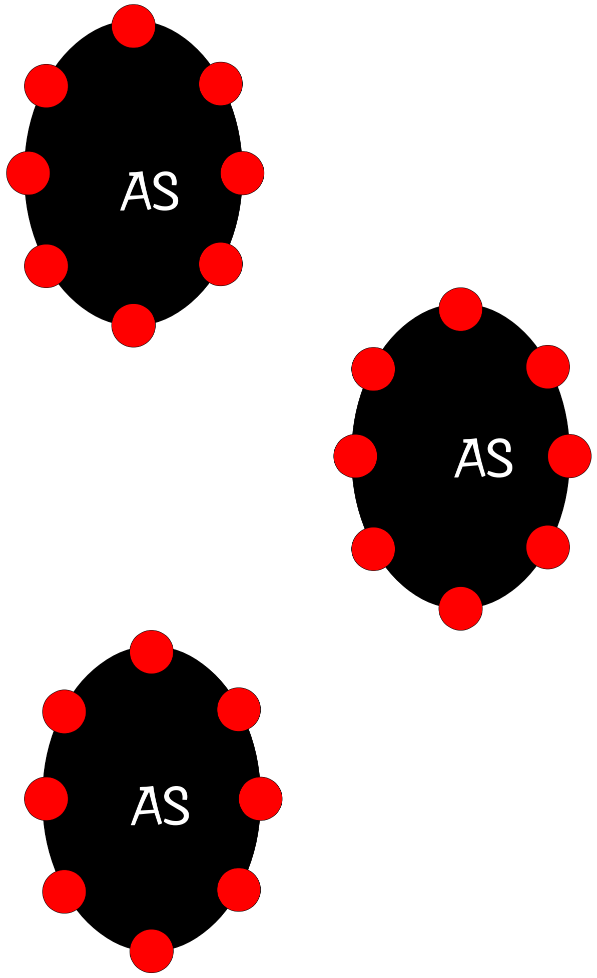
- IGP (interior gateway protocol) routing is concerned with routing within an AS.

IGP Routing



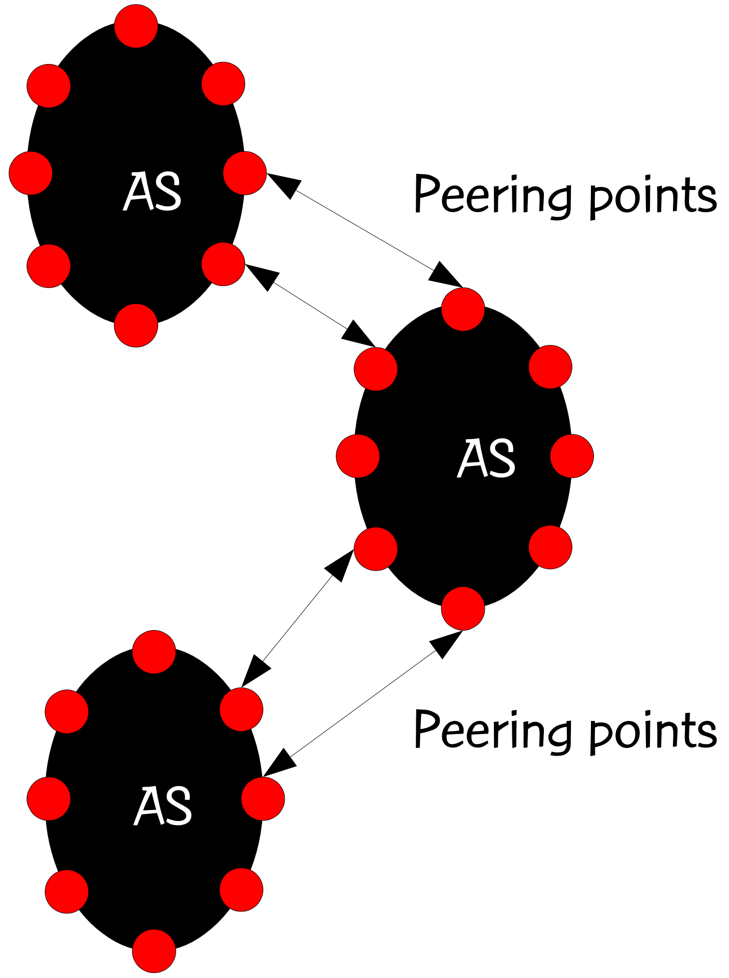
- IGP (interior gateway protocol) routing is concerned with routing within an AS.
- Routing decisions are made by AS operator.

BGP Routing



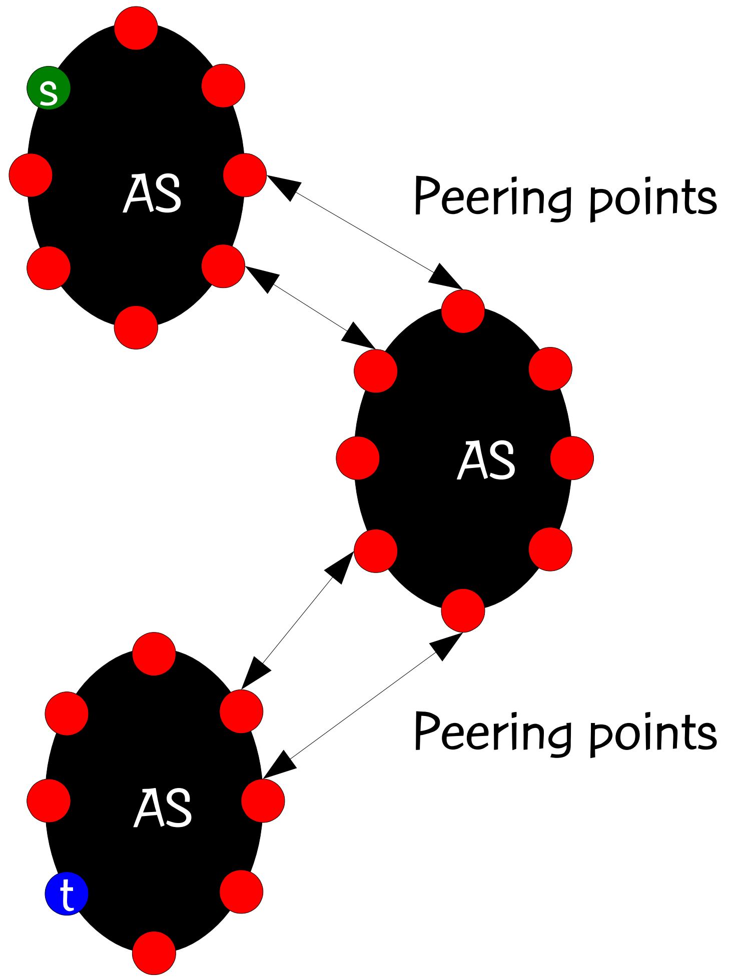
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



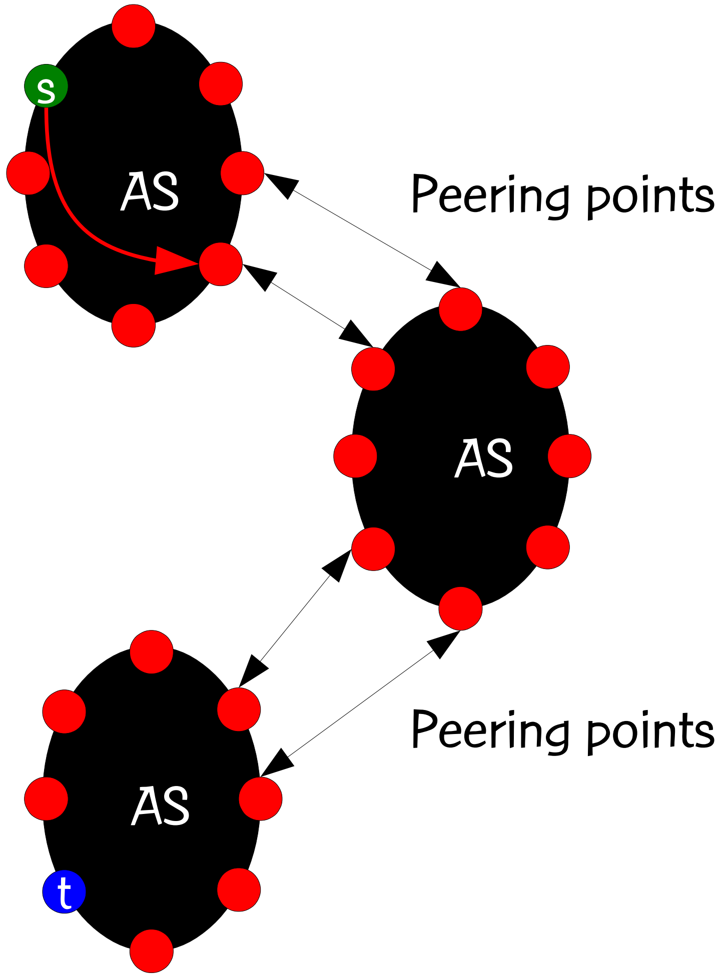
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



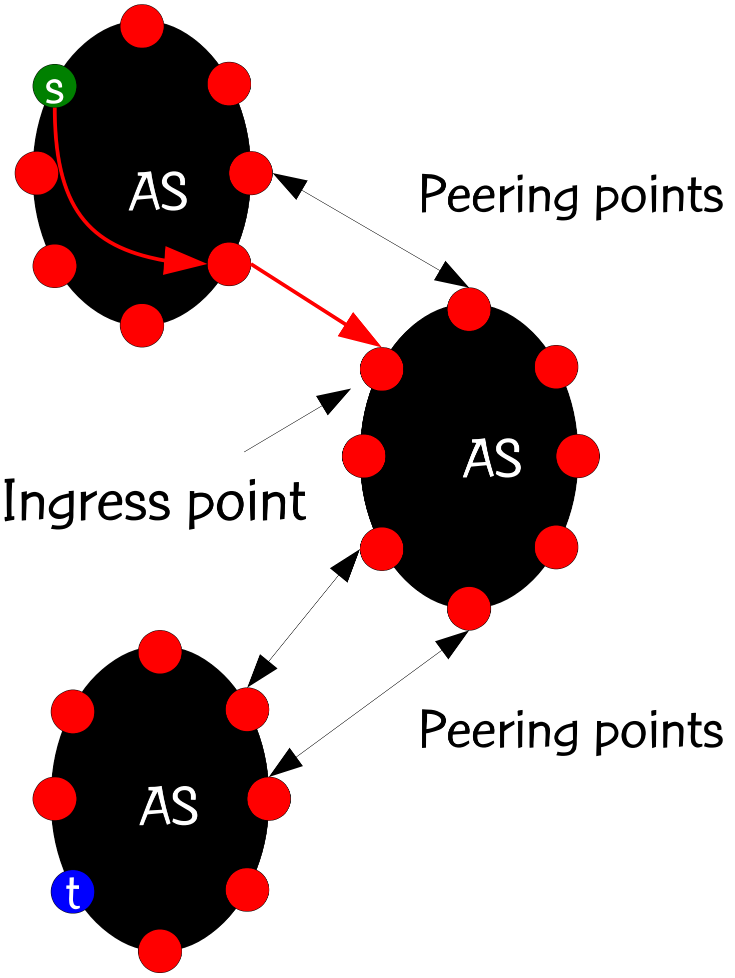
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



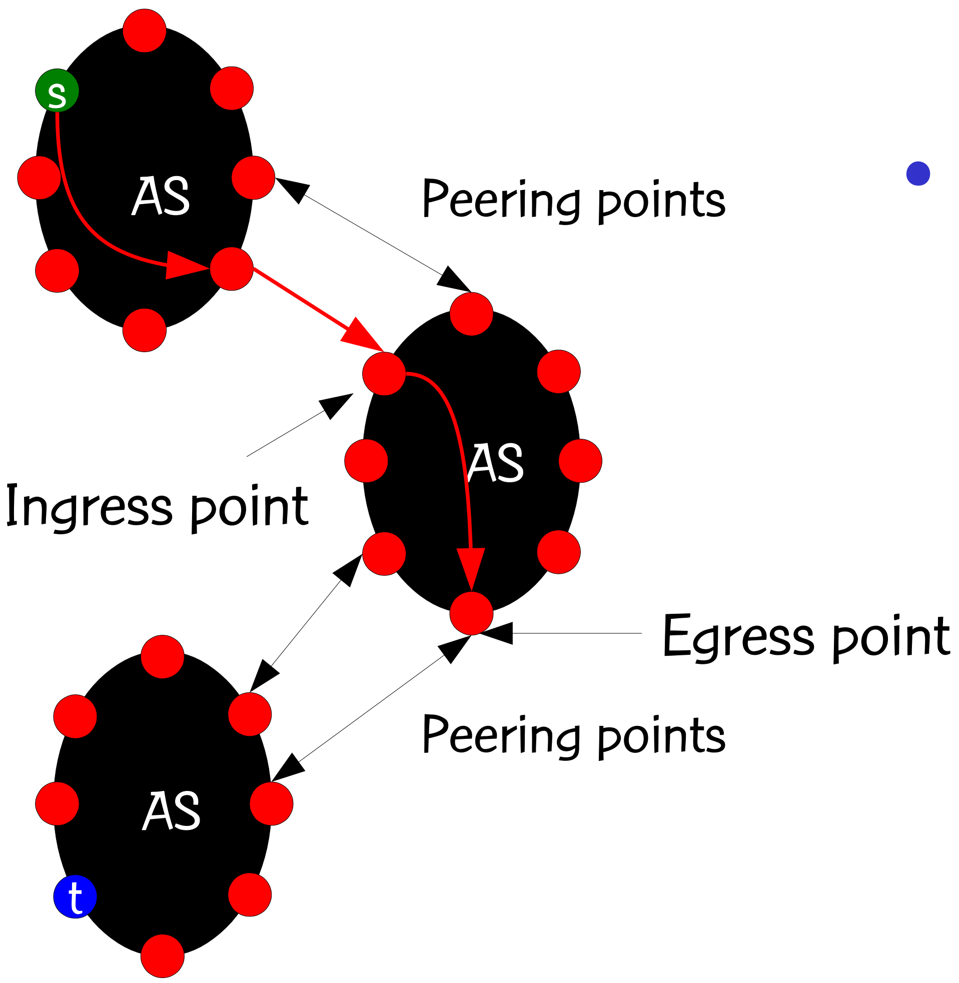
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



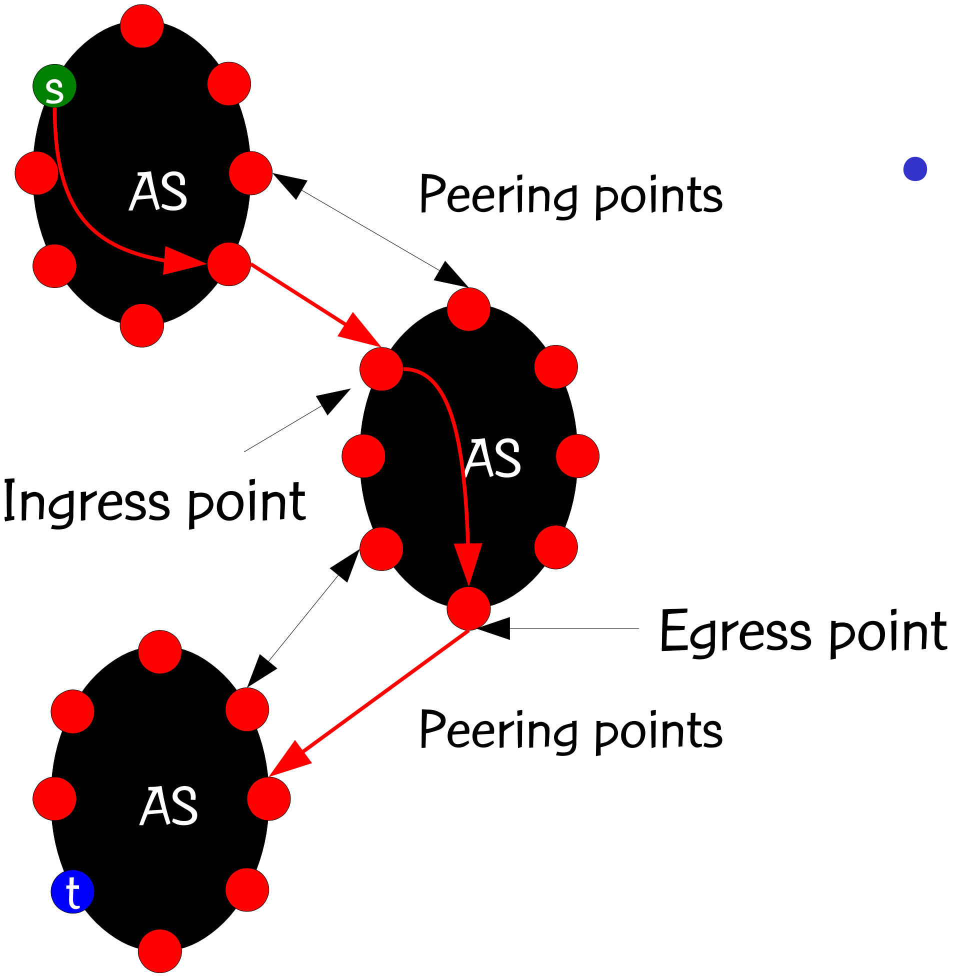
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



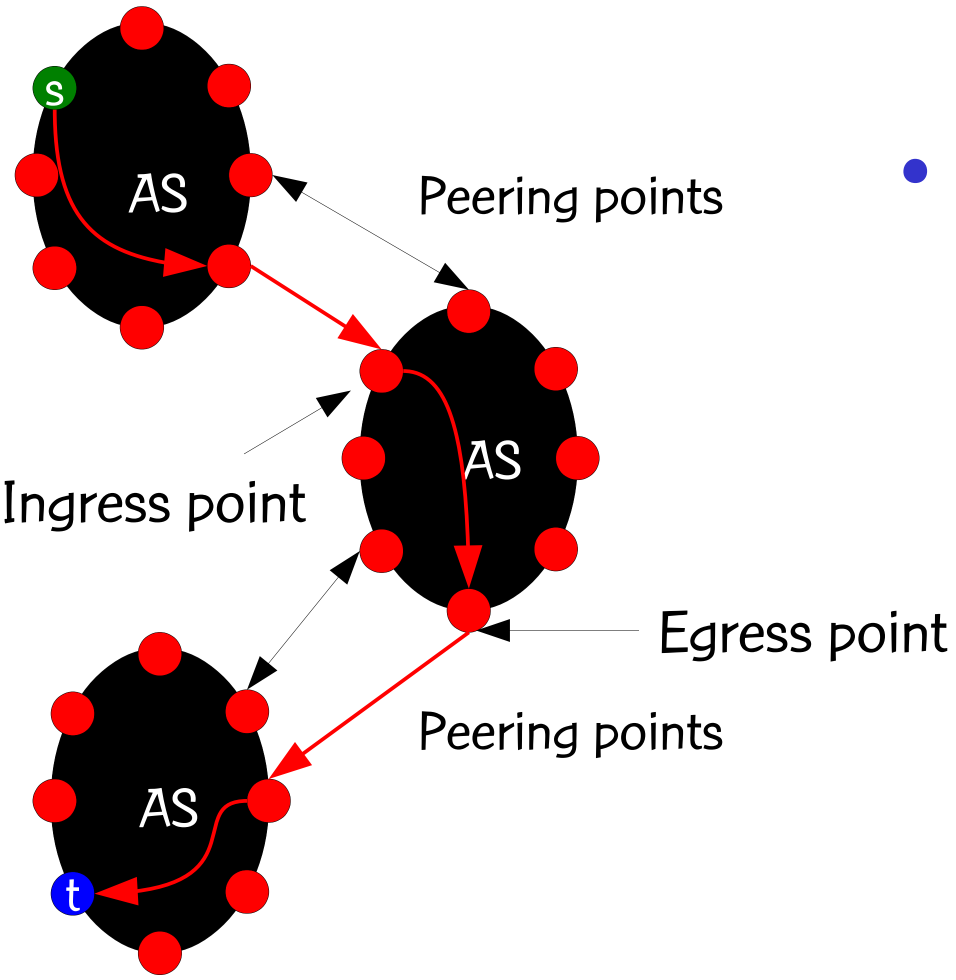
- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



- BGP (border gateway protocol) routing deals with routing between different ASes.

BGP Routing



- BGP (border gateway protocol) routing deals with routing between different ASes.

IGP Routing

OSPF routing

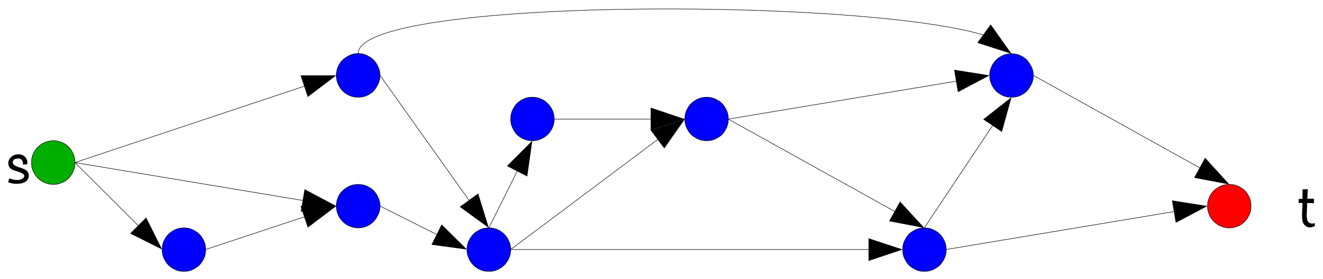
- Given a network $G = (N,A)$, where N is the set of routers and A is the set of links.

OSPF routing

- Given a network $G = (N,A)$, where N is the set of routers and A is the set of links.
- The OSPF (open shortest path first) routing protocol assumes each link a has a weight w(a) assigned to it so that a packet from a source router s to a destination router t is routed on a shortest weight path from s to t.

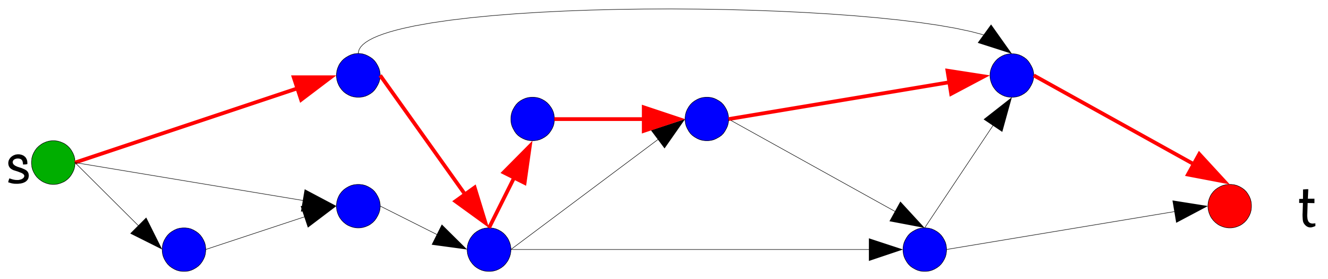
OSPF routing

- Given a network $G = (N,A)$, where N is the set of routers and A is the set of links.
- The OSPF (open shortest path first) routing protocol assumes each link a has a weight w(a) assigned to it so that a packet from a source router s to a destination router t is routed on a shortest weight path from s to t.



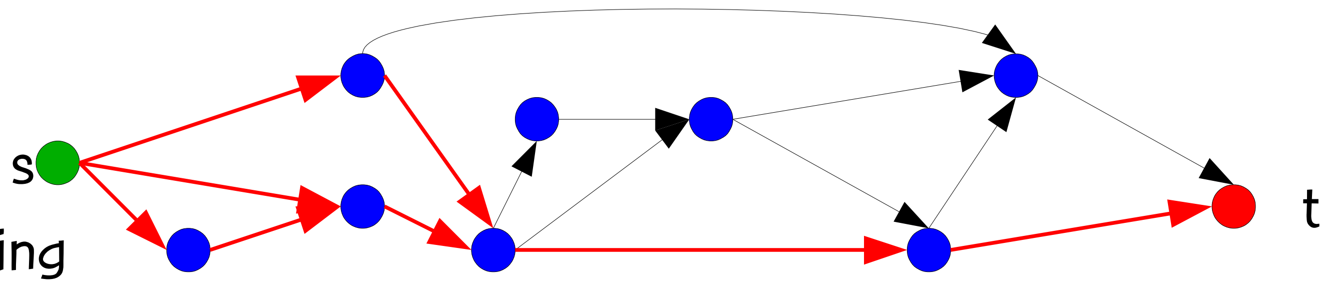
OSPF routing

- Given a network $G = (N,A)$, where N is the set of routers and A is the set of links.
- The OSPF (open shortest path first) routing protocol assumes each link a has a weight w(a) assigned to it so that a packet from a source router s to a destination router t is routed on a shortest weight path from s to t.



OSPF routing

- Given a network $G = (N,A)$, where N is the set of routers and A is the set of links.
- The OSPF (open shortest path first) routing protocol assumes each link a has a weight w(a) assigned to it so that a packet from a source router s to a destination router t is routed on a shortest weight path from s to t.



Traffic splitting

OSPF routing

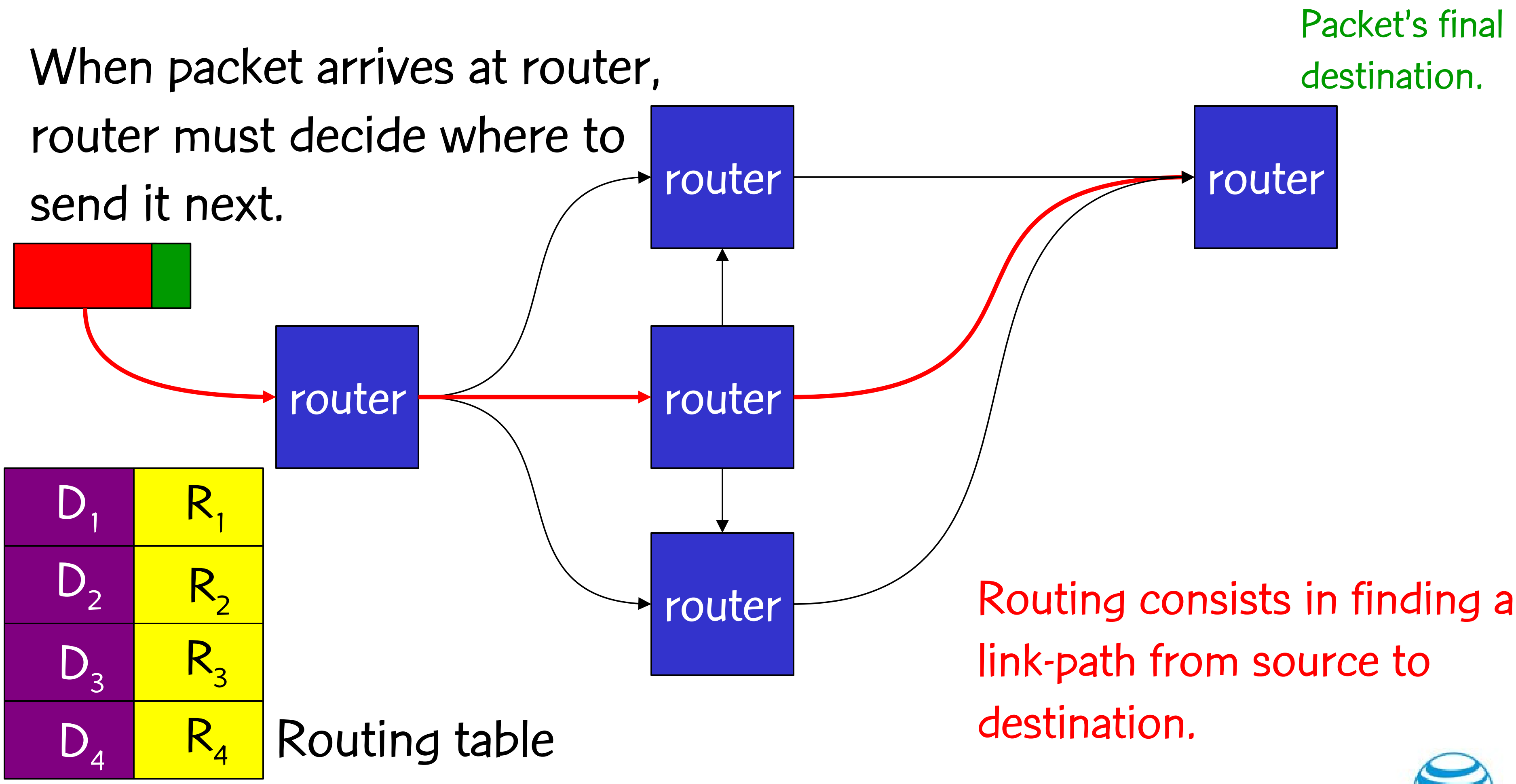
- By setting OSPF weights appropriately, one can do traffic engineering, i.e. route traffic so as to optimize some objective (e.g. minimize congestion, maximize throughput, etc.).
- Some recent papers on this topic:
  - Fortz & Thorup (2000, 2004)
  - Ramakrishnan & Rodrigues (2001)
  - Sridharan, Guérin, & Diot (2002)
  - Fortz, Rexford, & Thorup (2002)
  - Ericsson, Resende, & Pardalos (2002)
  - Buriol, Resende, Ribeiro, & Thorup (2002, 2005)
  - Reis, Ritt, Buriol, & Resende (2011)

OSPF routing

- By setting OSPF weights appropriately, one can do traffic engineering, i.e. route traffic so as to optimize some objective (e.g. minimize congestion, maximize throughput, etc.).
- Some recent papers on this topic:
  - Fortz & Thorup (2000, 2004)
  - Ramakrishnan & Rodrigues (2001)
  - Sridharan, Guérin, & Diot (2002)
  - Fortz, Rexford, & Thorup (2002)
  - Ericsson, Resende, & Pardalos (2002)
  - Buriol, Resende, Ribeiro, & Thorup (2002, 2005)
  - Reis, Ritt, Buriol & Resende (2011)

Packet routing

When packet arrives at router, router must decide where to send it next.



D_1	R_1
D_2	R_2
D_3	R_3
D_4	R_4

Routing table

Routing consists in finding a link-path from source to destination.

OSPF routing

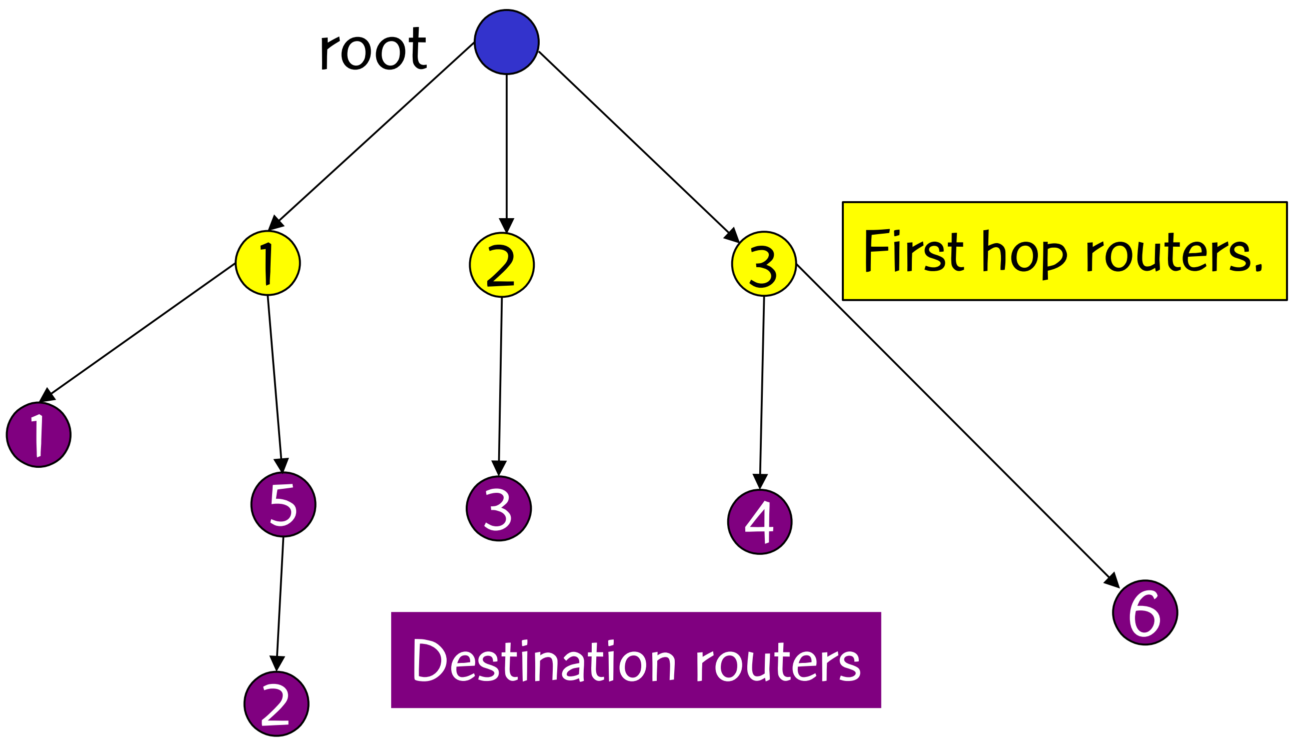
- Assign an integer weight $\in [1, w_{max}]$ to each link in *AS*. In general, $w_{max} = 65535 = 2^{16} - 1$.
- Each router computes tree of shortest weight paths to all other routers in the *AS*, with itself as the root, using Dijkstra's algorithm.

OSPF routing

Routing table

D_1	R_1
D_2	R_1
D_3	R_2
D_4	R_3
D_5	R_1
D_6	R_3

Routing table is filled
with first hop routers
for each possible destination.

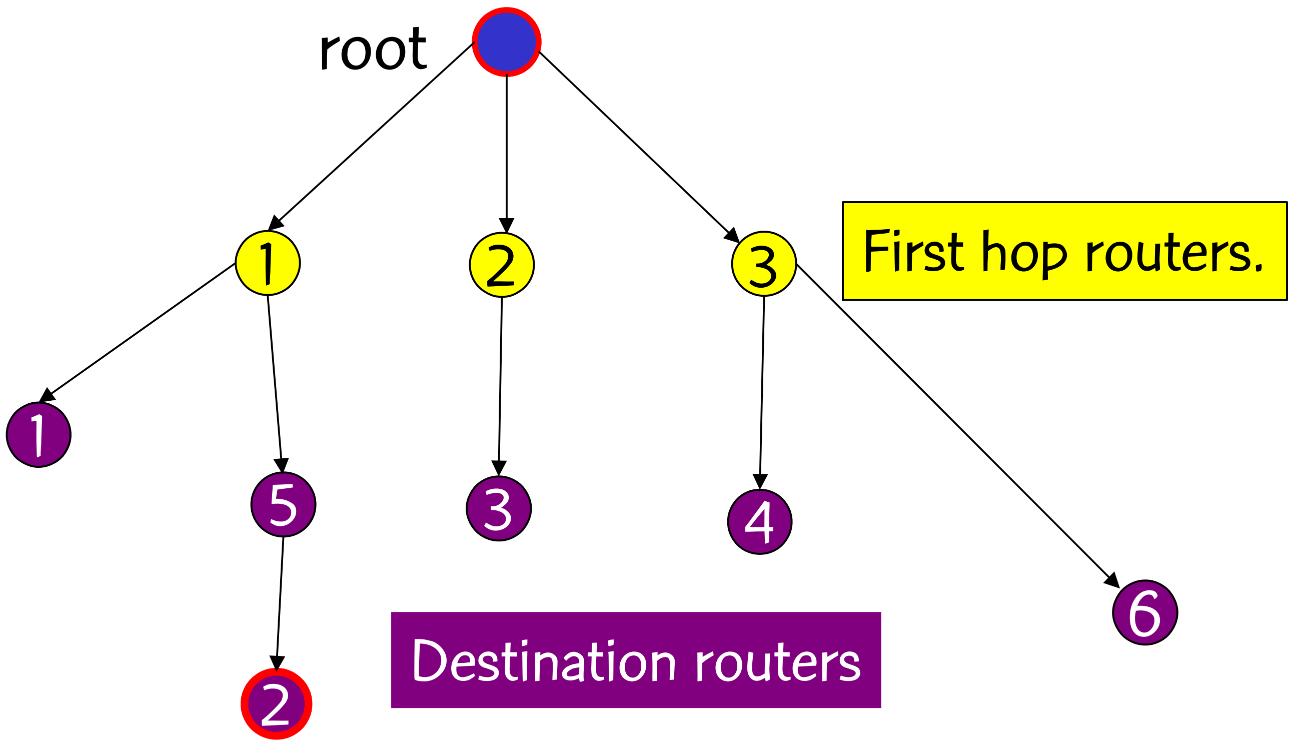


OSPF routing

Routing table

D_1	R_1
D_2	R_1
D_3	R_2
D_4	R_3
D_5	R_1
D_6	R_3

Routing table is filled with first hop routers for each possible destination.

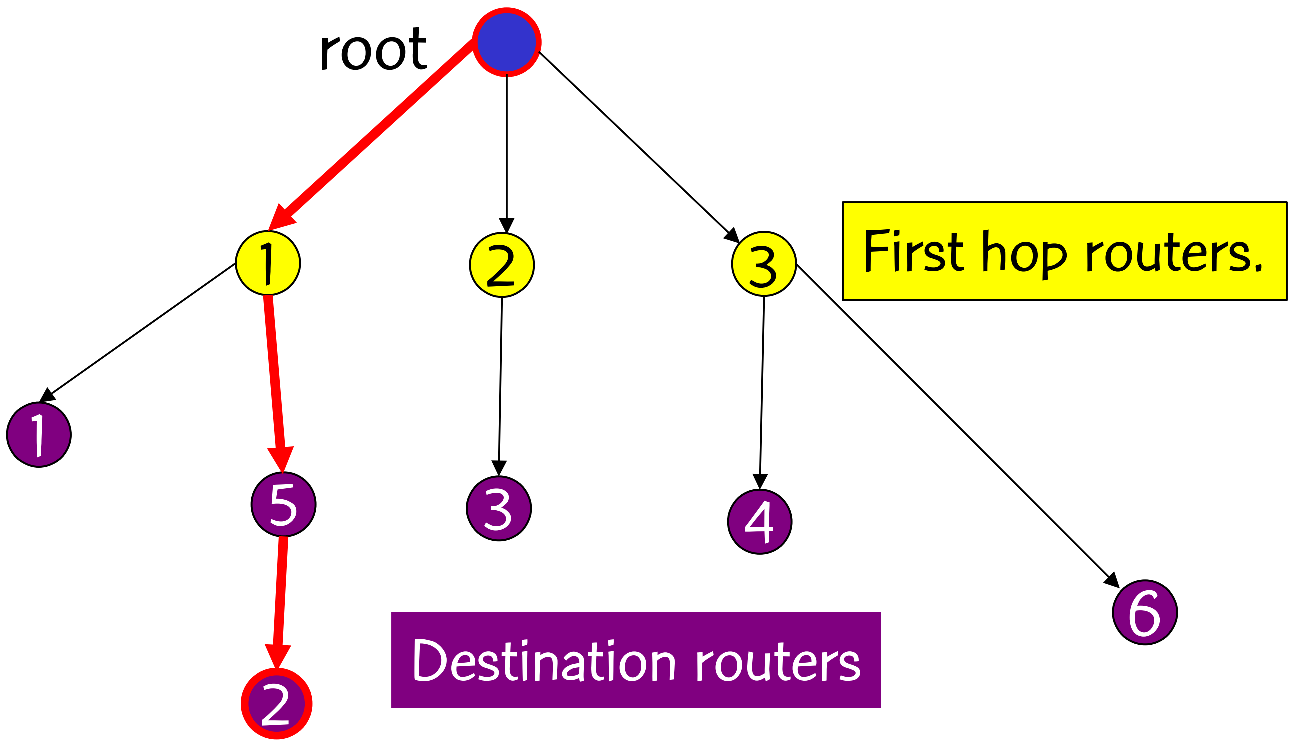


OSPF routing

Routing table

D_1	R_1
D_2	R_1
D_3	R_2
D_4	R_3
D_5	R_1
D_6	R_3

Routing table is filled
with first hop routers
for each possible destination.

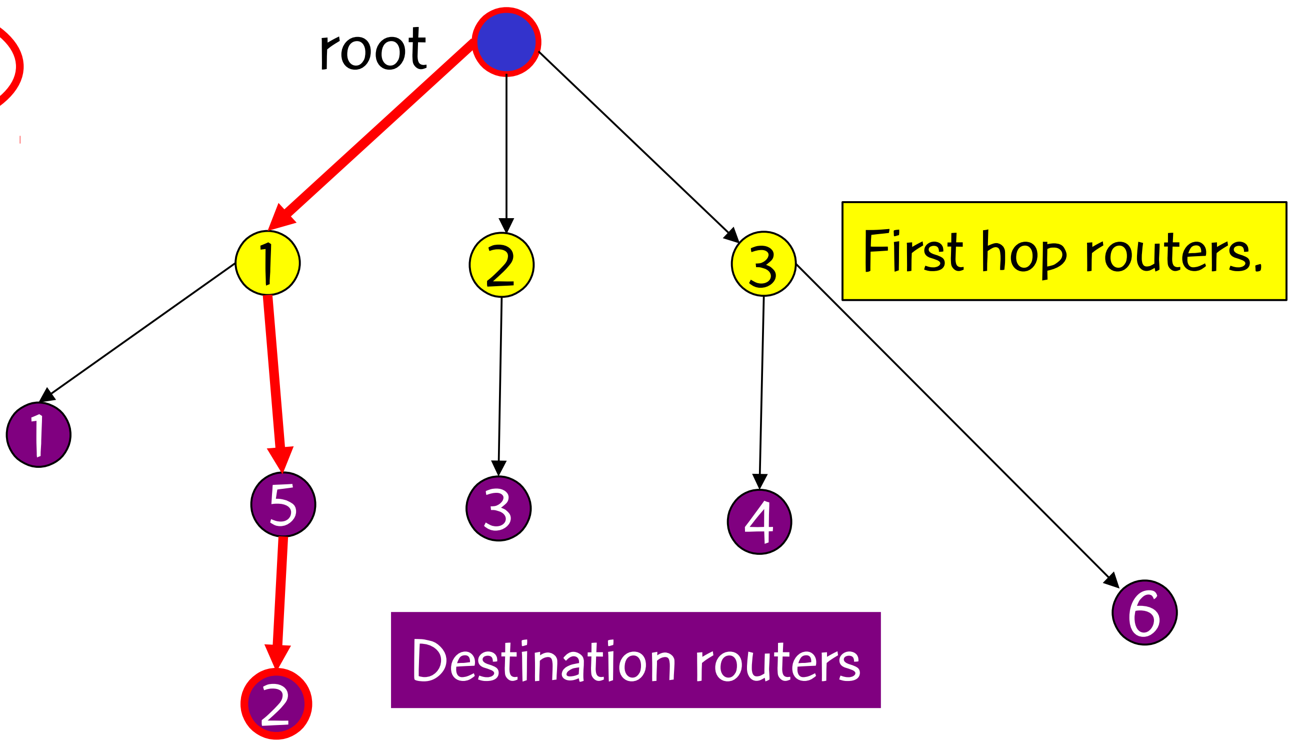


OSPF routing

Routing table is filled
with first hop routers
for each possible destination.

Routing table

D_1	R_1
D_2	R_1
D_3	R_2
D_4	R_3
D_5	R_1
D_6	R_3



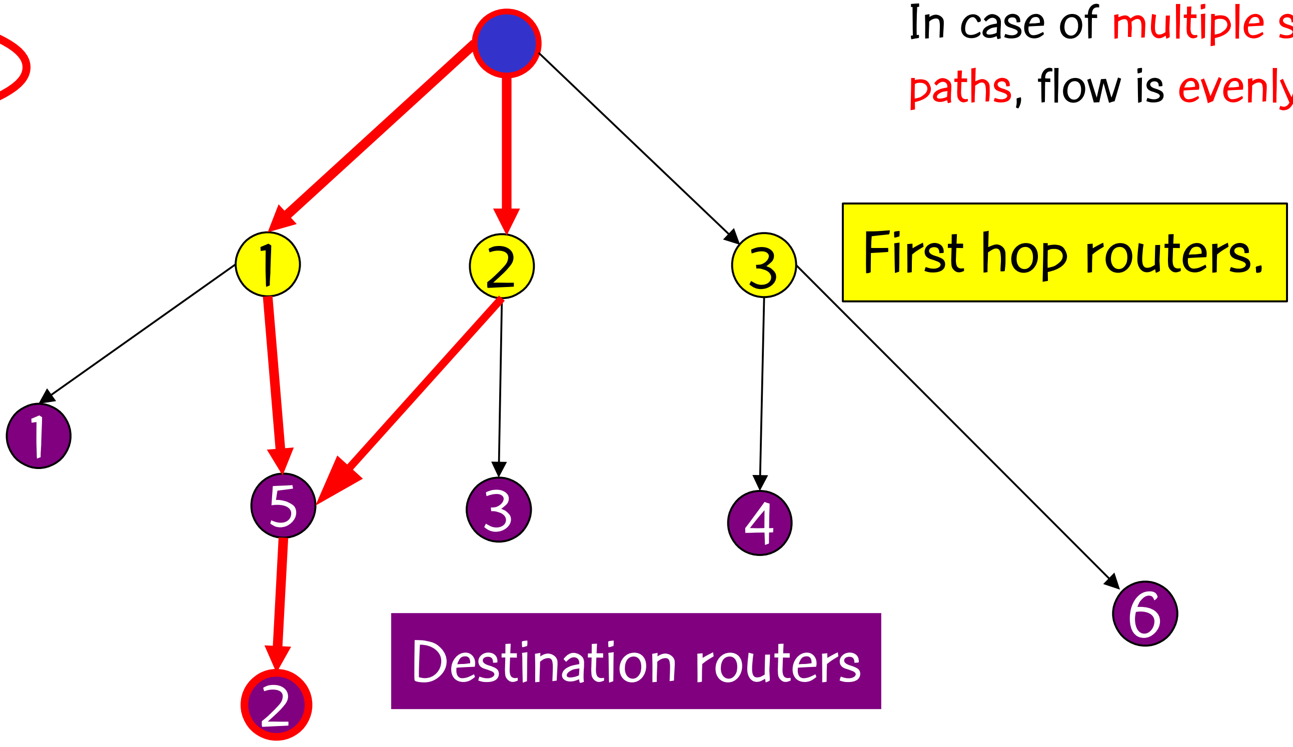
OSPF routing

Routing table

D_1	R_1
D_2	R_1, R_2
D_3	R_2
D_4	R_3
D_5	R_1
D_6	R_3

Routing table is filled with first hop routers for each possible destination.

In case of multiple shortest paths, flow is evenly split.



OSPF weight setting

- OSPF weights are assigned by network operator.
  - CISCO assigns, by default, a weight proportional to the inverse of the link bandwidth (Inv Cap).
  - If all weights are unit, the weight of a path is the number of hops in the path.
- We propose two BRKGA to find good OSPF weights.

Minimization of congestion

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

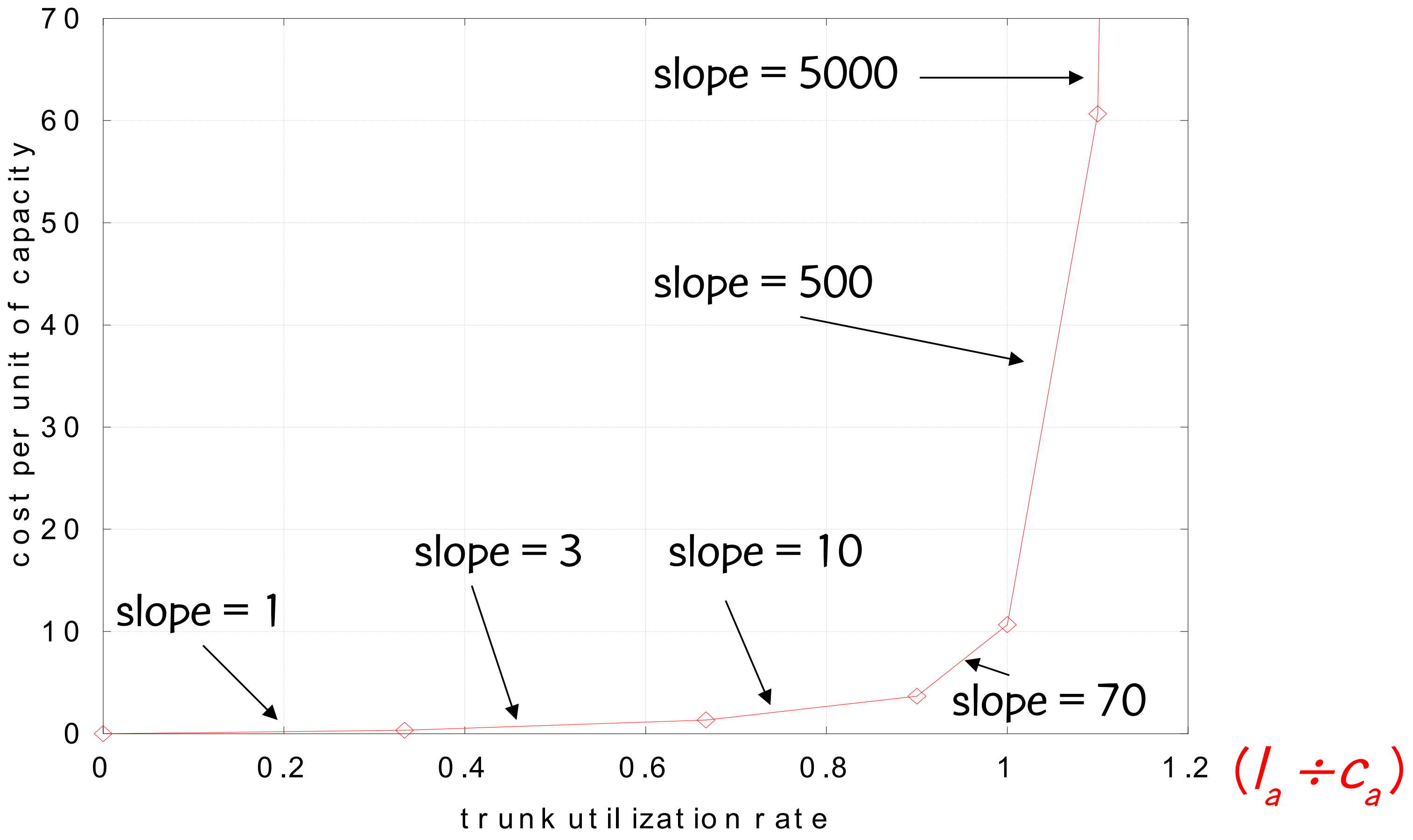
$$\Phi = \Phi_1(l_1) + \Phi_2(l_2) + \ldots + \Phi_{|A|}(l_{|A|})$$

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

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

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

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



OSPF weight setting problem

- Given a directed network $G = (N, A)$ with link capacities $c_a \in A$ and demand matrix $D = (d_{s,t})$ specifying a demand to be sent from node s to node t:
  - Assign weights $w_a \in [1, w_{max}]$ to each link $a \in A$, such that the objective function Φ is minimized when demand is routed according to the OSPF protocol.

BRKGA for OSPF routing in IP networks

M. Ericsson, M.G.C.R., & P.M. Pardalos, "A genetic algorithm for the weight setting problem in OSPF routing," J. of Combinatorial Optimization, vol. 6, pp. 299–333, 2002.

Tech report version:

http://www2.research.att.com/~mgcr/doc/gaospf.pdf

BRKGA for OSPF routing in IP networks

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

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.

BRKGA for OSPF routing in IP networks

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

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoding:

BRKGA for OSPF routing in IP networks

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

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoding:
  - For $i = 1, \ldots, N$: set $w(i) = \text{ceil}\,(\,X(i) \times w_{max}\,)$

BRKGA for OSPF routing in IP networks

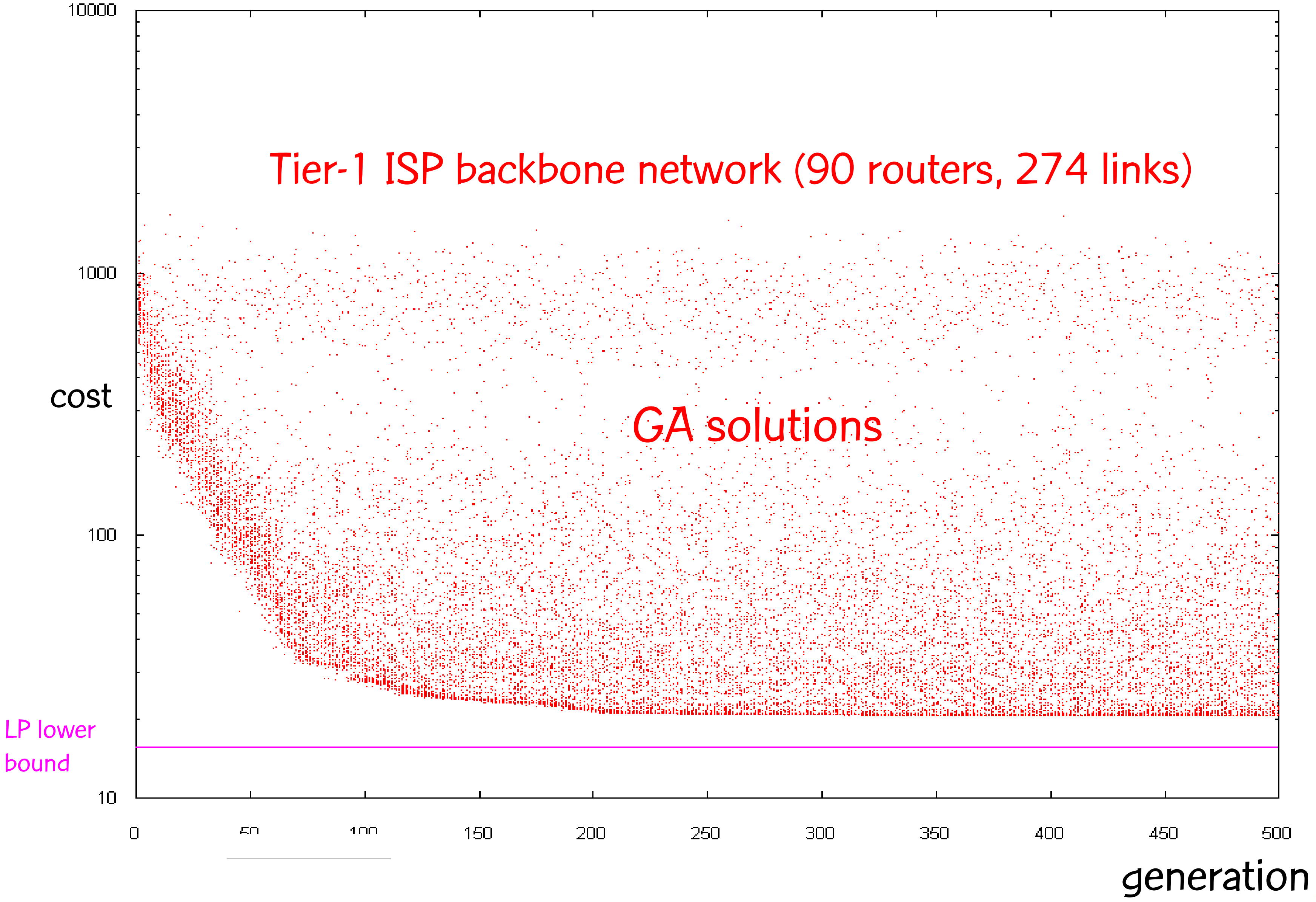
Ericsson, R., & Pardalos (J. Comb. Opt., 2002)

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoding:
  - For $i = 1, \ldots, N$: set $w(i) = \text{ceil}\,(X(i) \times w_{\max})$
  - Compute shortest paths and route traffic according to OSPF.

BRKGA for OSPF routing in IP networks

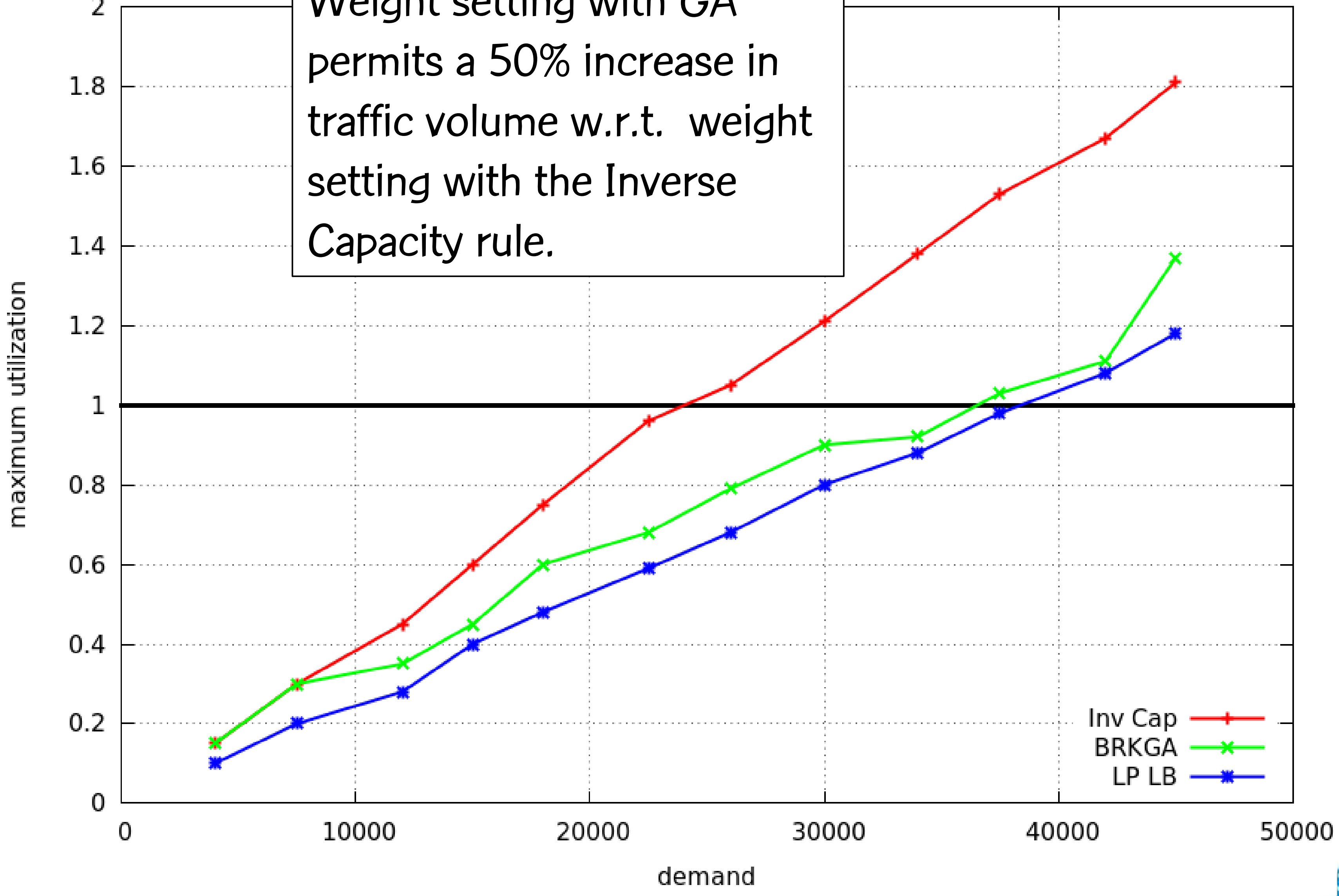
Ericsson, R., & Pardalos (J. Comb. Opt., 2002)

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

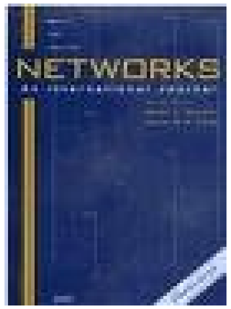

Tier-1 ISP backbone network (90 routers, 274 links)
GA solutions
cost
LP lower bound
generation
10000
1000
100
10
0
50
100
150
200
250
300
350
400
450
500

Tier-1 ISP backbone network (90 routers, 274 links)

Weight setting with GA permits a 50% increase in traffic volume w.r.t. weight setting with the Inverse Capacity rule.



Improved BRKGA for OSPF routing in IP networks



L.S. Buriol, M.G.C.R., C.C. Ribeiro, and M. Thorup, "A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing," Networks, vol. 46, pp. 36–56, 2005.

Tech report version:

http://www2.research.att.com/~mgcr/doc/hgaospf.pdf

Improved BRKGA for OSPF routing in IP networks

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

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.

Improved BRKGA for OSPF routing in IP networks

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

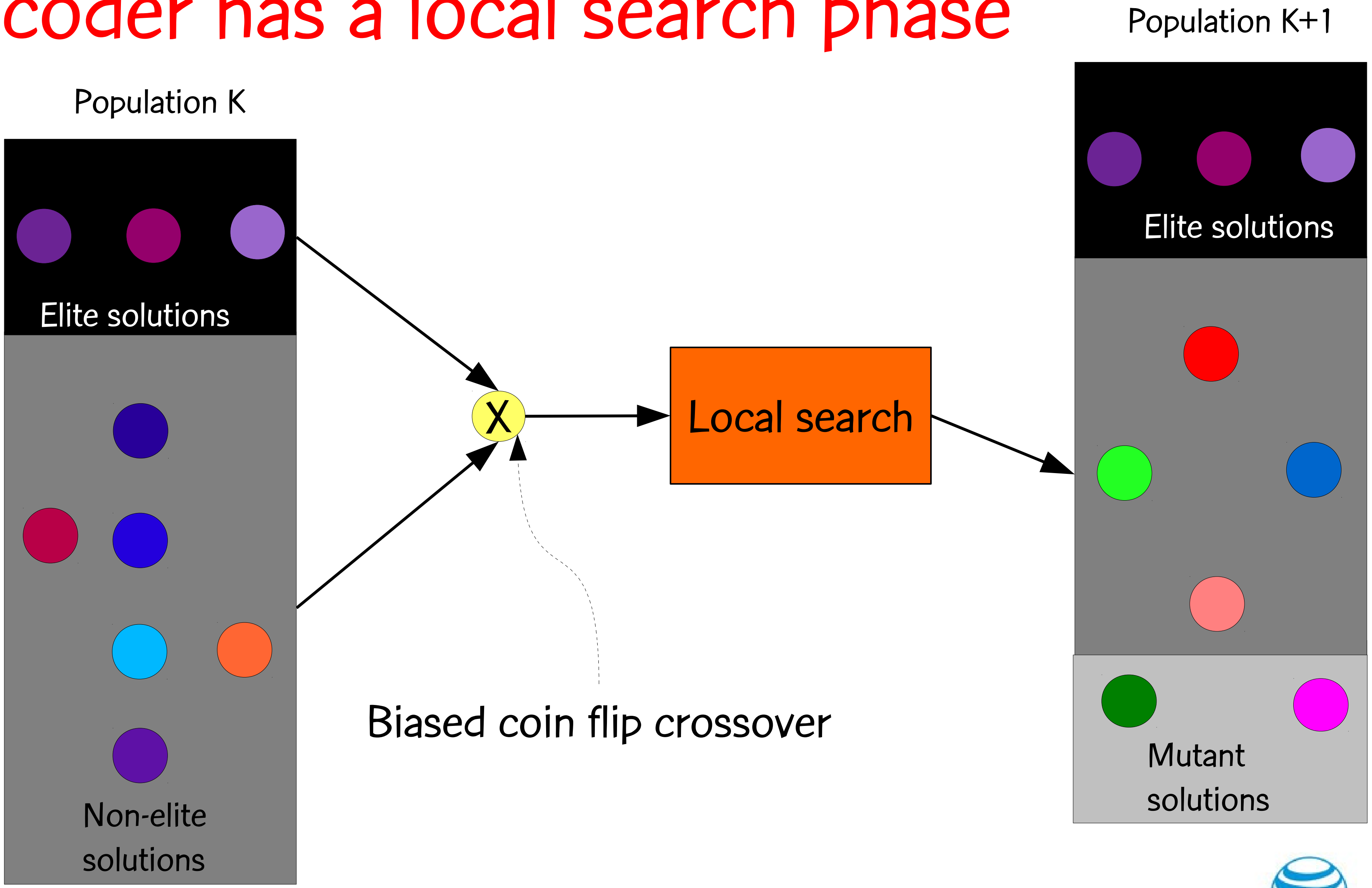
- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For i = 1, ..., N: set $w(i) = \text{ceil}\,(\,X(i) \times w_{max}\,)$
  - Compute shortest paths and route traffic according to OSPF.
  - Compute load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

Improved BRKGA for OSPF routing in IP networks

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

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

Decoder has a local search phase



Fast local search

- Let A^* be the set of five arcs $a \in A$ having largest Φ_a values.

Fast local search

- Let A^* be the set of five arcs $a \in A$ having largest Φ_a values.
- Scan arcs $a \in A^*$ from largest to smallest Φ_a:

Fast local search

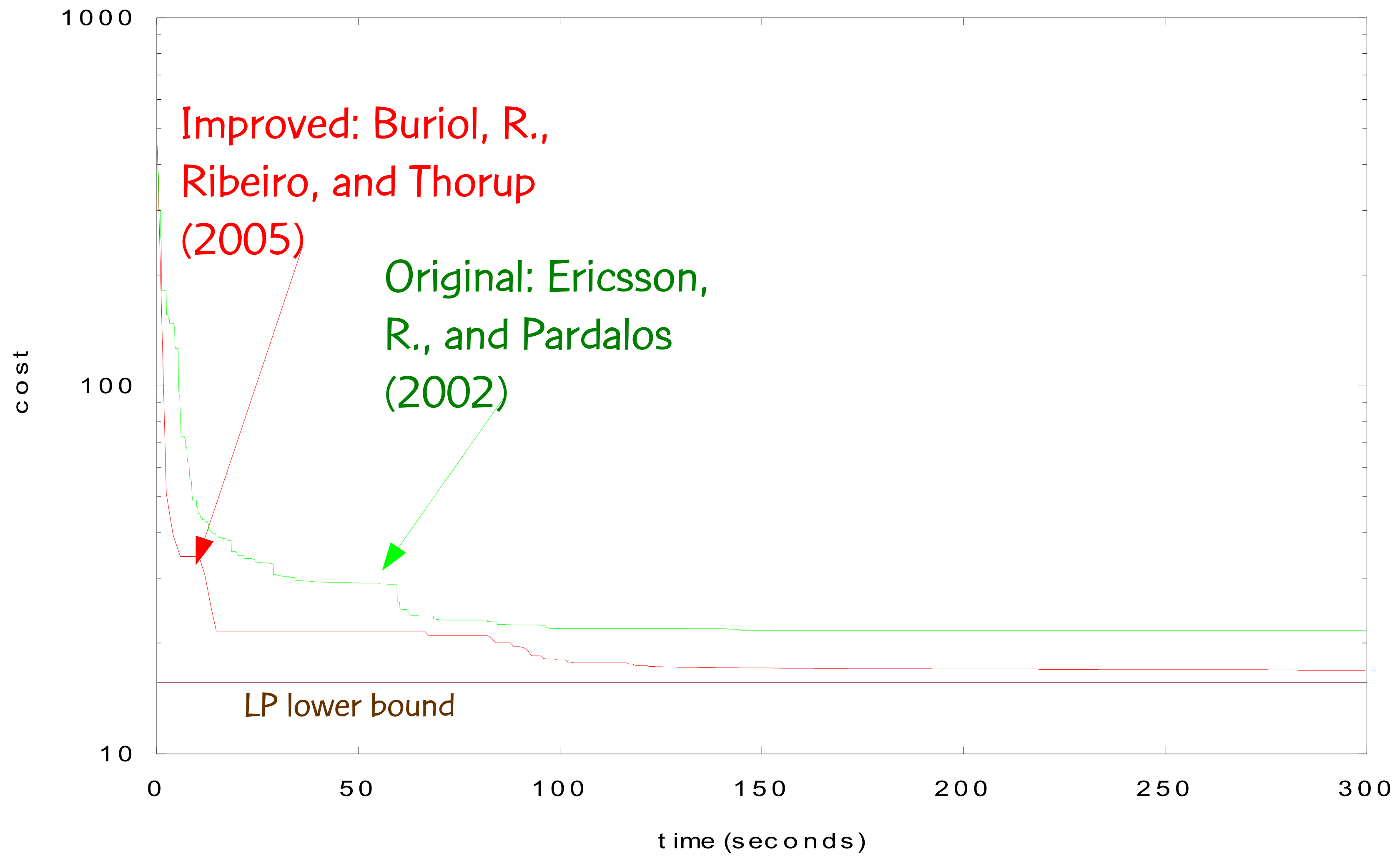
- Let A^* be the set of five arcs $a \in A$ having largest Φ_a values.
- Scan arcs $a \in A^*$ from largest to smallest Φ_a:
  - Increase arc weight, one unit at a time, in the range

$$[w_a , w_a + \lceil (w_{max} - w_a)/4 \rceil]$$

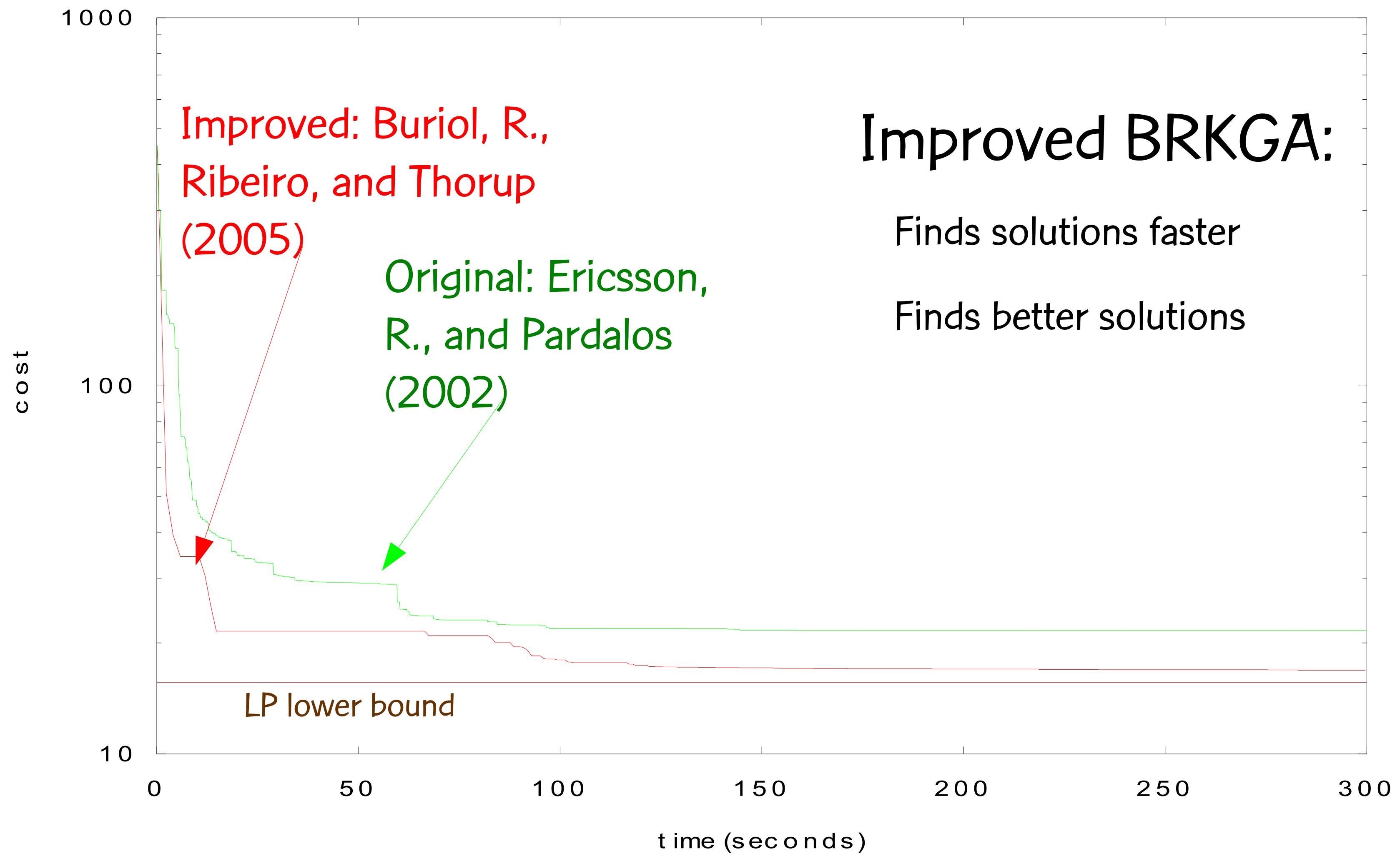
Fast local search

- Let A^* be the set of five arcs $a \in A$ having largest Φ_a values.
- Scan arcs $a \in A^*$ from largest to smallest Φ_a:
  - Increase arc weight, one unit at a time, in the range
    $$[w_a\,,\, w_a + \lceil (w_{max} - w_a)/4 \rceil\,]$$
  - If total cost Φ is reduced, restart local search.

Effect of decoder with fast local search



Effect of decoder with fast local search



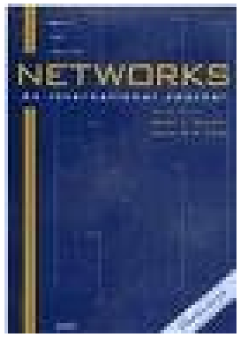
Improved BRKGA:

Finds solutions faster

Finds better solutions

Survivable IP network design

Survivable IP network design



L.S. Buriol, M.G.C.R., and M. Thorup, "Survivable IP network design with OSPF routing," Networks, vol. 49, pp. 51–64, 2007.

Tech report version:

http://www2.research.att.com/~mgcr/doc/gamult.pdf

Survivable IP network design

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

- Given

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,
  - a cost K(a) to lay fiber on arc a

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,
  - a cost $K(a)$ to lay fiber on arc a
  - a capacity increment C for the fiber.

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,
  - a cost K(a) to lay fiber on arc a
  - a capacity increment C for the fiber.
- Determine

Survivable IP network design

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

- Given
  - directed graph G = (N,A), where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair (s,t) ∈ N×N, specifies the demand D(s,t) between s and t,
  - a cost K(a) to lay fiber on arc a
  - a capacity increment C for the fiber.
- Determine
  - OSPF weight w(a) to assign to each arc a ∈ A,

Survivable IP network design

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

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

Survivable IP network design

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

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

Survivable IP network design

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

- Given
  - directed graph $G = (N,A)$, where N is the set of routers, A is the set of potential arcs where capacity can be installed,
  - a demand matrix D that for each pair $(s,t) \in N \times N$, specifies the demand $D(s,t)$ between s and t,
  - a cost $K(a)$ to lay fiber on arc a
  - a capacity increment C for the fiber.

- Determine
  - OSPF weight $w(a)$ to assign to each arc $a \in A$,
  - which arcs should be used to deploy fiber and how many units (multiplicities) $M(a)$ of capacity C should be installed on each arc $a \in A$,
- such that all the demand can be routed on the network even when any single arc fails.
- Min total design cost $= \sum_{a \in A} M(a) \times K(a)$.

Survivable IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.

Survivable IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:

Survivable IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For $i = 1, \ldots, N$: set $w(i) = \text{ceil}\,(X(i) \times w_{max})$

Survivable IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For $i = 1, \ldots, N$: set $w(i) = \text{ceil}\,(\,X(i) \times w_{\max}\,)$
  - For each failure mode: route demand according to OSPF and for each arc $a \in A$ determine the load on arc a.

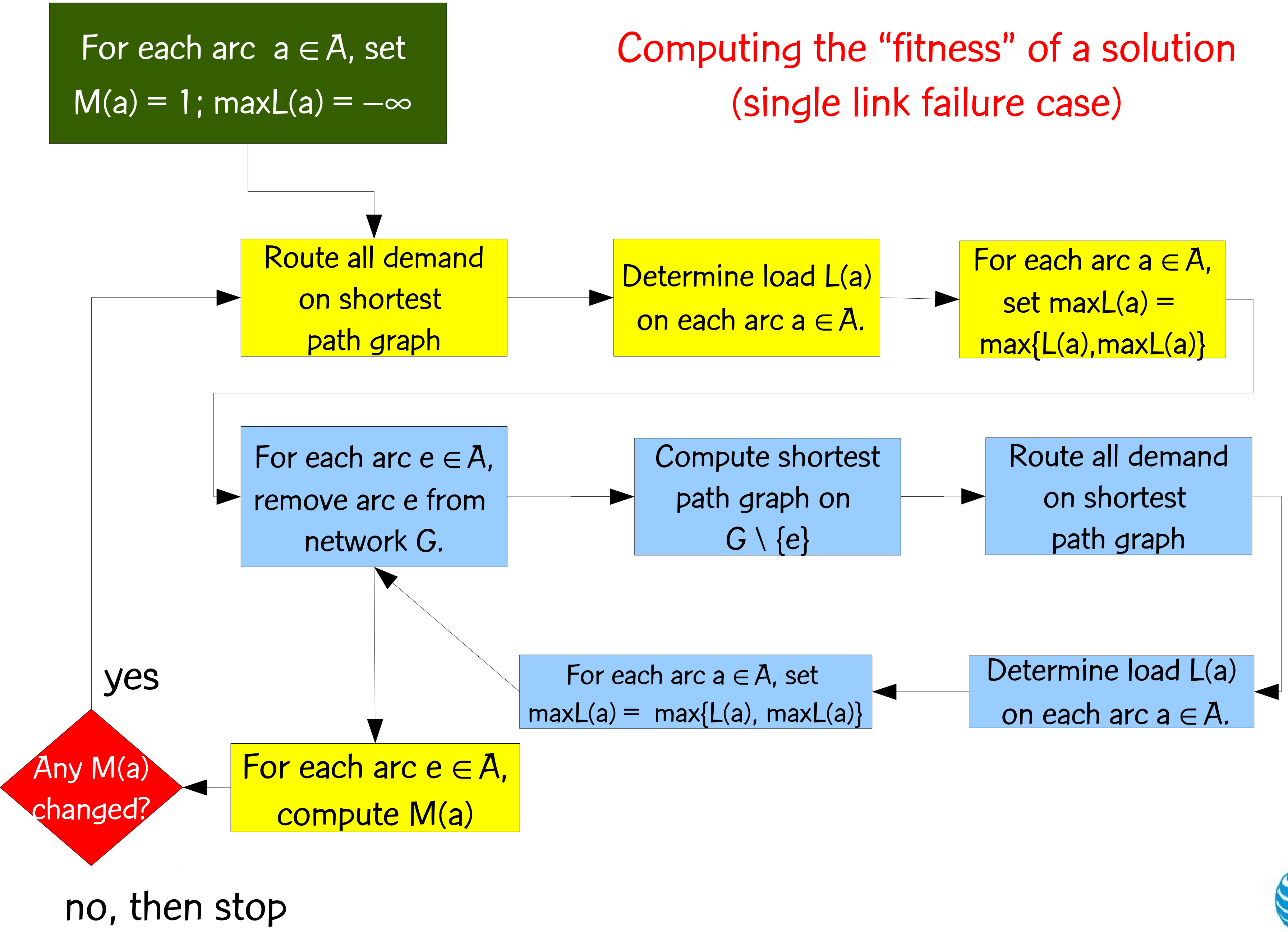
Survivable IP network design

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

Survivable IP network design

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

Computing the "fitness" of a solution (single link failure case)


For each arc a ∈ A, set M(a) = 1; maxL(a) = −∞
Route all demand on shortest path graph
Determine load L(a) on each arc a ∈ A.
For each arc a ∈ A, set maxL(a) = max{L(a),maxL(a)}
For each arc e ∈ A, remove arc e from network G.
Compute shortest path graph on G \ {e}
Route all demand on shortest path graph
Determine load L(a) on each arc a ∈ A.
For each arc a ∈ A, set maxL(a) = max{L(a), maxL(a)}
For each arc e ∈ A, compute M(a)
Any M(a) changed?
yes
no, then stop

Composite-link design

- In Buriol, R., and Thorup (2006)
  - links were all of the same type,
  - only the link multiplicity had to be determined.
- Now consider composite links. Given a load L(a) on arc a, we can compose several different link types that sum up to the needed capacity $c(a) \geq L(a)$:
  - $c(a) = \sum_{t \text{ used in arc } a} M(t) \times \gamma(t)$, where
  - M(t) is the multiplicity of link type t
  - $\gamma(t)$ is the capacity of link type t

Composite-link design

- In Buriol, Resende, and Thorup (2006)
  - links were all of the same type,
  - only the link multiplicity had to be determined.
- Now consider composite links. Given a load $L(a)$ on arc a, we can compose several different link types that sum up to the needed capacity $c(a) \geq L(a)$:
  - $c(a) = \sum_{t \text{ used in arc } a} M(t) \times \gamma(t)$, where
  - $M(t)$ is the multiplicity of link type t
  - $\gamma(t)$ is the capacity of link type t

Composite-link design

D.V. Andrade, L.S. Buriol, M.G.C.R., and M. Thorup, "Survivable composite-link IP network design with OSPF routing," The Eighth INFORMS Telecommunications Conference, Dallas, Texas, April 2006.

Tech report:

http://www2.research.att.com/~mgcr/doc/composite.pdf

Composite-link design

- Link types = $\{ 1, 2, \ldots, T \}$
- Capacities = $\{ c(1), c(2), \ldots, c(T) \} : c(i) < c(i+1)$
- Prices / unit length = $\{ p(1), p(2), \ldots, p(T) \}: p(i) < p(i+1)$
- Assumptions:
  - $[p(T)/c(T)] < [p(T-1)/c(T-1)] < \cdots < [p(1)/c(1)]$, i.e. price per unit of capacity is smaller for links with greater capacity
  - $c(i) = \alpha \times c(i-1)$, for $\alpha \in N$, $\alpha > 1$, i.e. capacities are multiples of each other by powers of α

Composite-link design

- Link types = $\{ 1, 2, \ldots, T \}$
- Capacities = $\{ c(1), c(2), \ldots, c(T) \} : c(i) < c(i+1)$
- Prices / unit length = $\{ p(1), p(2), \ldots, p(T) \}: p(i) < p(i+1)$
- Assumptions:
  - $[p(T)/c(T)] < [p(T-1)/c(T-1)] < \cdots < [p(1)/c(1)]$, i.e. price per unit of capacity is smaller for links with greater capacity
  - $c(i) = \alpha \times c(i-1)$, for $\alpha \in N$, $\alpha > 1$, i.e. capacities are multiples of each other by powers of α

Composite-link design

- Link types = $\{ 1, 2, \ldots, T \}$
- Capacities = $\{ c(1), c(2), \ldots, c(T) \} : c(i) < c(i+1)$
- Prices / unit length = $\{ p(1), p(2), \ldots, p(T) \}: p(i) < p(i+1)$
- Assumptions:
  - $[p(T)/c(T)] < [p(T-1)/c(T-1)] < \cdots < [p(1)/c(1)]$: economies of scale
  - $c(i) = \alpha \times c(i-1)$, for $\alpha \in N$, $\alpha > 1$, e.g. $c(OC192) = 4 \times c(OC48)$; $c(OC48) = 4 \times c(OC12)$; $c(OC12) = 4 \times c(OC3)$;

OC3	OC12	OC48	OC192	
155 Mb/s	622 Mb/s	2.5 Gb/s	10 Gb/s	$\alpha = 4$

Survivable composite link IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.

Survivable composite link IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:

Survivable composite link IP network design

- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoder:
  - For $i = 1, \ldots, N$: set $w(i) = \text{ceil}\ (\ X(i) \times w_{max}\ )$

Survivable composite link IP network design

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

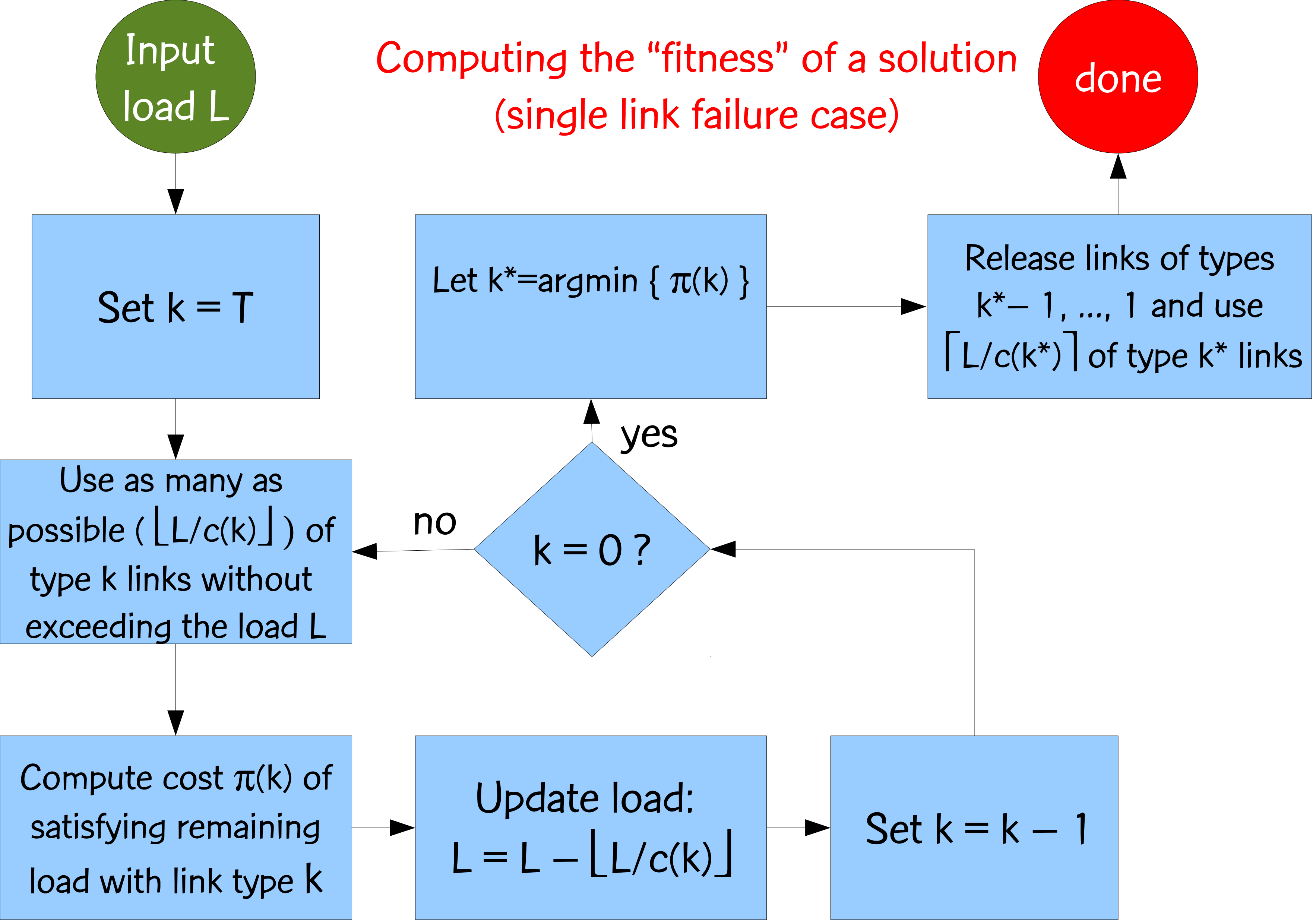
Survivable composite link IP network design

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

Survivable composite link IP network design

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

Computing the "fitness" of a solution (single link failure case)


Input load L
done
Set k = T
Let k*=argmin { π(k) }
Release links of types k*– 1, ..., 1 and use ⌈L/c(k*)⌉ of type k* links
yes
no
k = 0 ?
Use as many as possible (⌊L/c(k)⌋) of type k links without exceeding the load L
Compute cost π(k) of satisfying remaining load with link type k
Update load: L = L – ⌊L/c(k)⌋
Set k = k – 1

Redundant content distribution

Reference:

ALENEX11

Workshop on
Algorithm Engineering & Experiments

January 22, 2011
Holiday Inn San Francisco Golden Gateway
San Francisco, California USA

L. Breslau, I. Diakonikolas, N. Duffield, Y. Gu, M. Hajiaghayi, D.S. Johnson, H. Karloff, M.G.C.R., and S. Sen, "Disjoint-path facility location: Theory and practice," Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX11), SIAM, San Francisco, pp. 60–74, January 22, 2011

Tech report version:

http://www2.research.att.com/~mgcr/doc/monitoring-alenex.pdf

Redundant content distribution (RCD)

- Suppose a number of users located at nodes in a network demand content.

Redundant content distribution

- Suppose a number of users located at nodes in a network demand content.
- Copies of content are stored throughout the network in data warehouses.

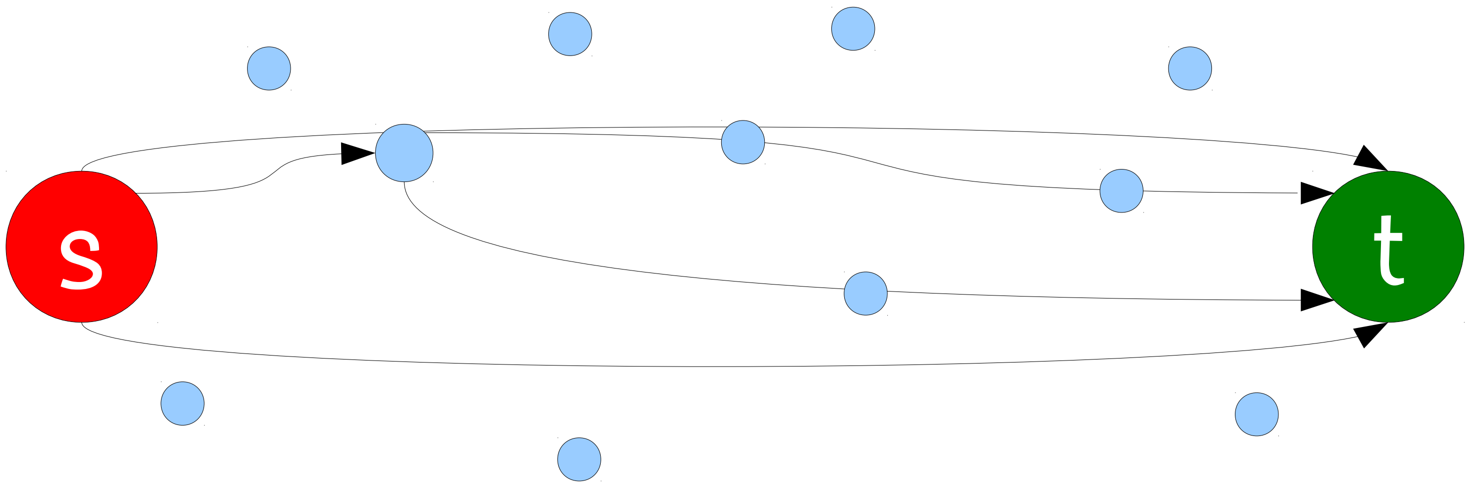
Redundant content distribution

- Suppose a number of users located at nodes in a network demand content.
- Copies of content are stored throughout the network in data warehouses.
- Content is sent from data warehouse to user on routes determined by OSPF.

Redundant content distribution

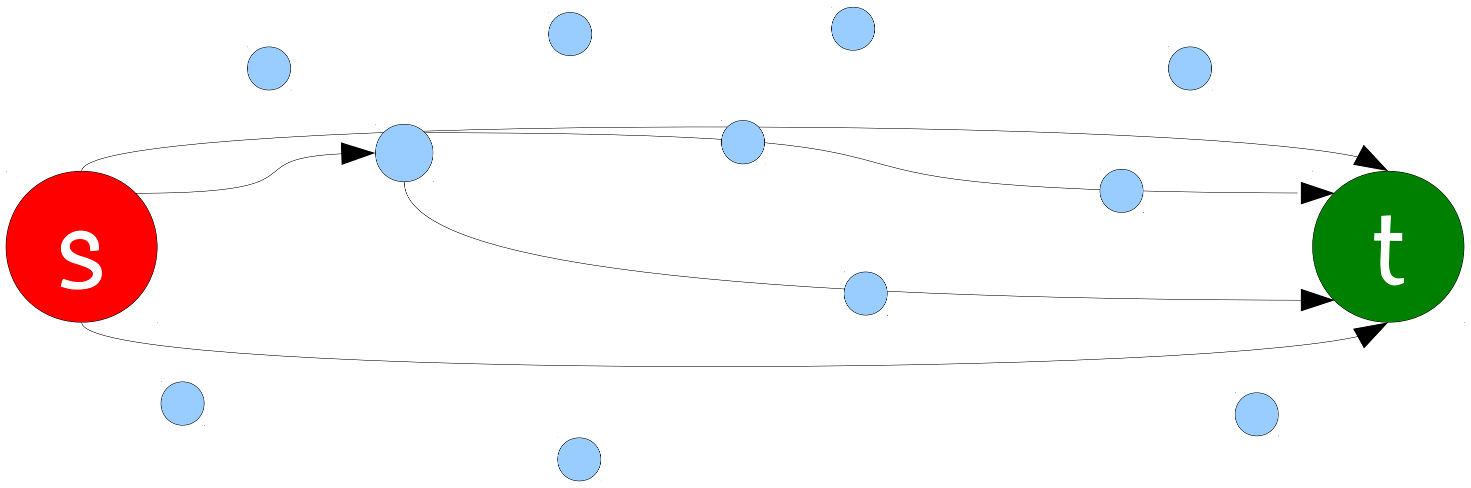
- Suppose a number of users located at nodes in a network demand content.
- Copies of content are stored throughout the network in data warehouses.
- Content is sent from data warehouse to user on routes determined by OSPF.
- Problem: Locate minimum number of warehouses in network such all users get their content even in presence of edge failures.

Redundant content distribution



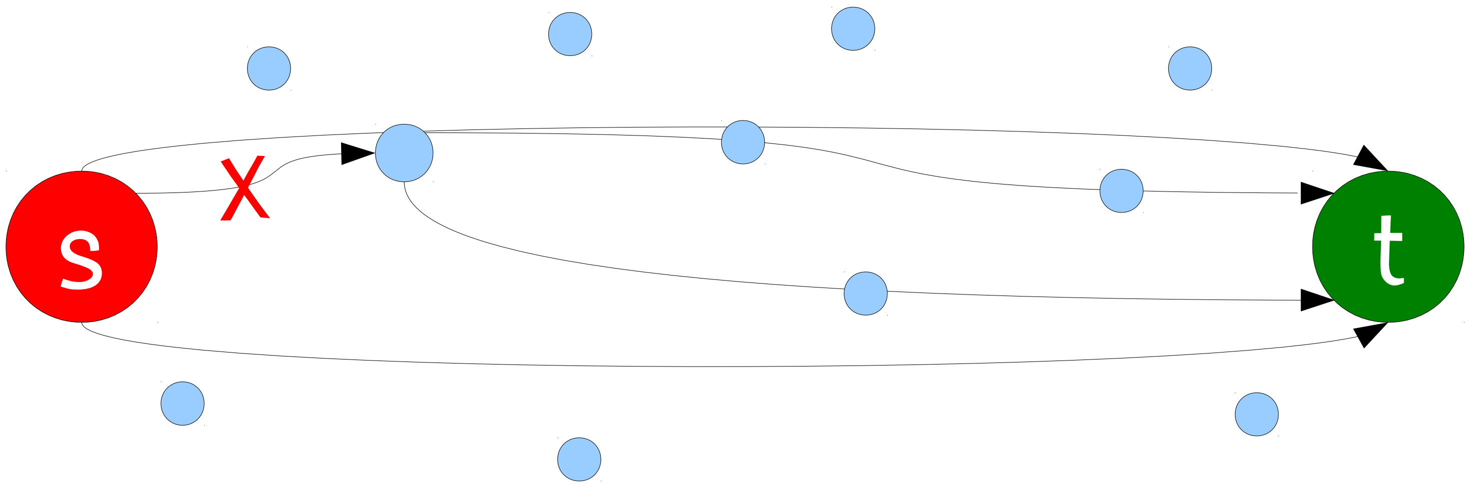
Traffic from node s to node t flows on paths defined by OSPF.

Redundant content distribution

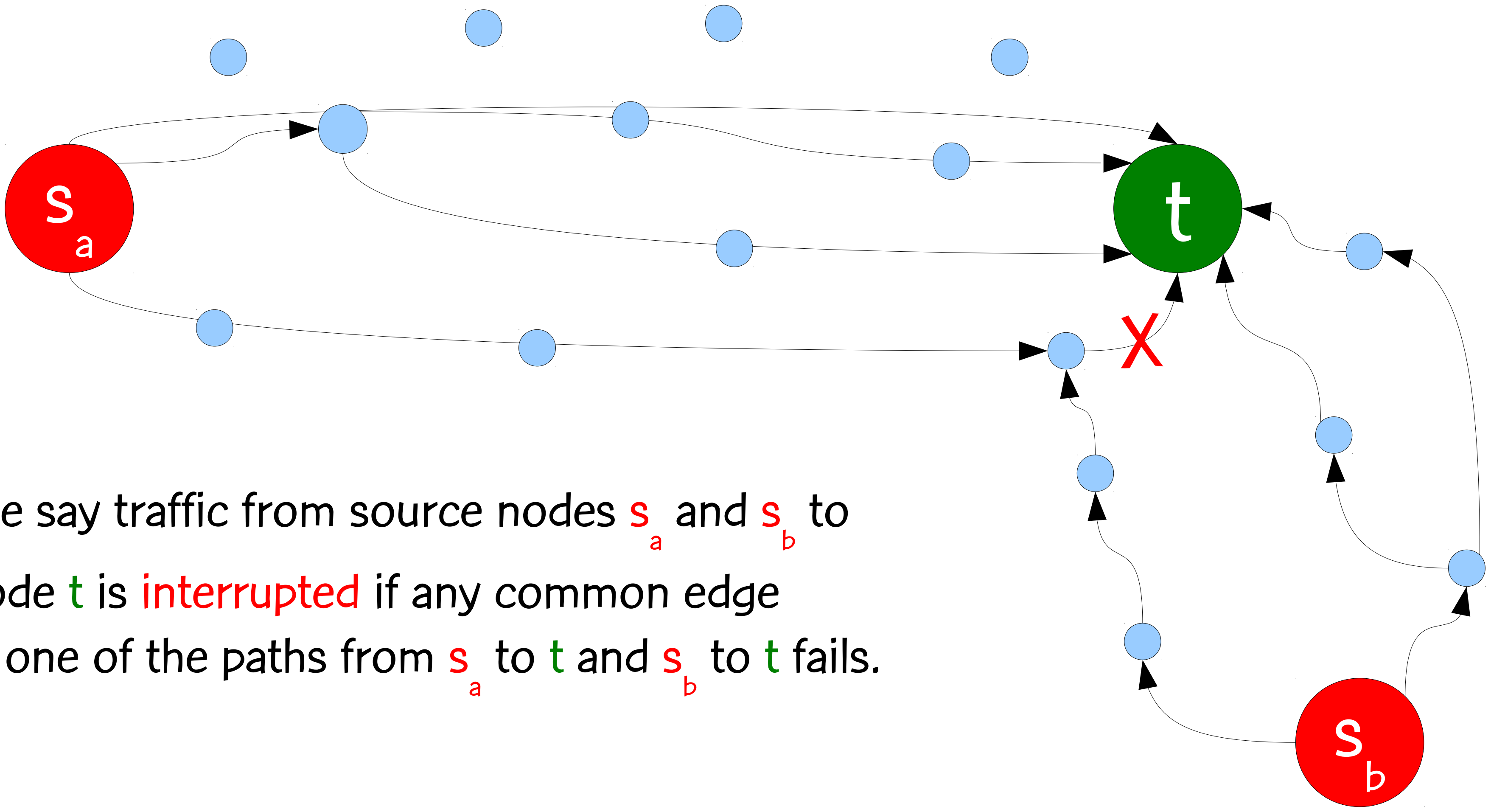


We don't know on which path a particular packet will flow.

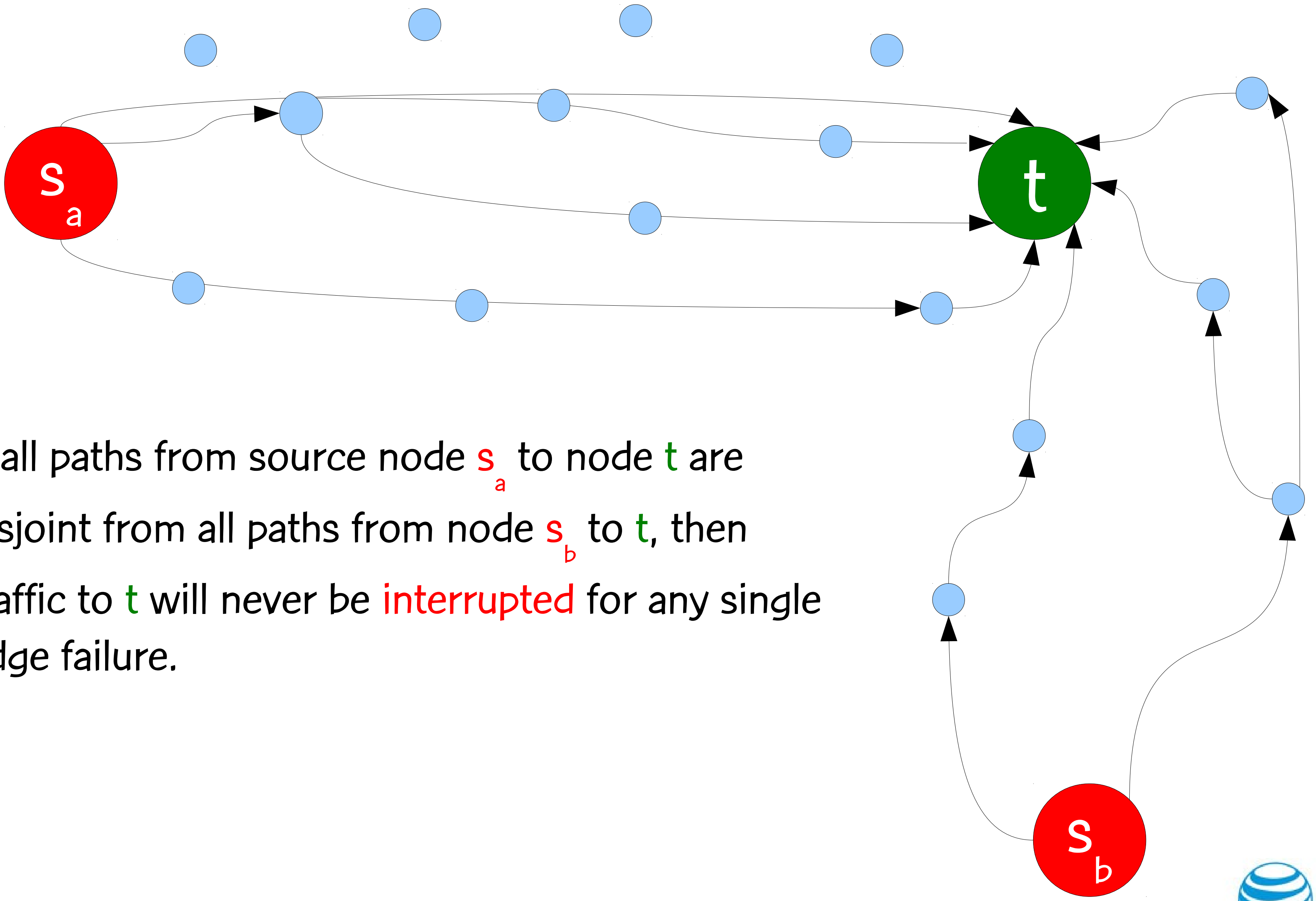
Redundant content distribution



We say traffic from node s to node t is interrupted if any edge in one of the paths from s to t fails.



We say traffic from source nodes s_a and s_b to node t is interrupted if any common edge in one of the paths from s_a to t and s_b to t fails.



If all paths from source node s_a to node t are disjoint from all paths from node s_b to t, then traffic to t will never be interrupted for any single edge failure.

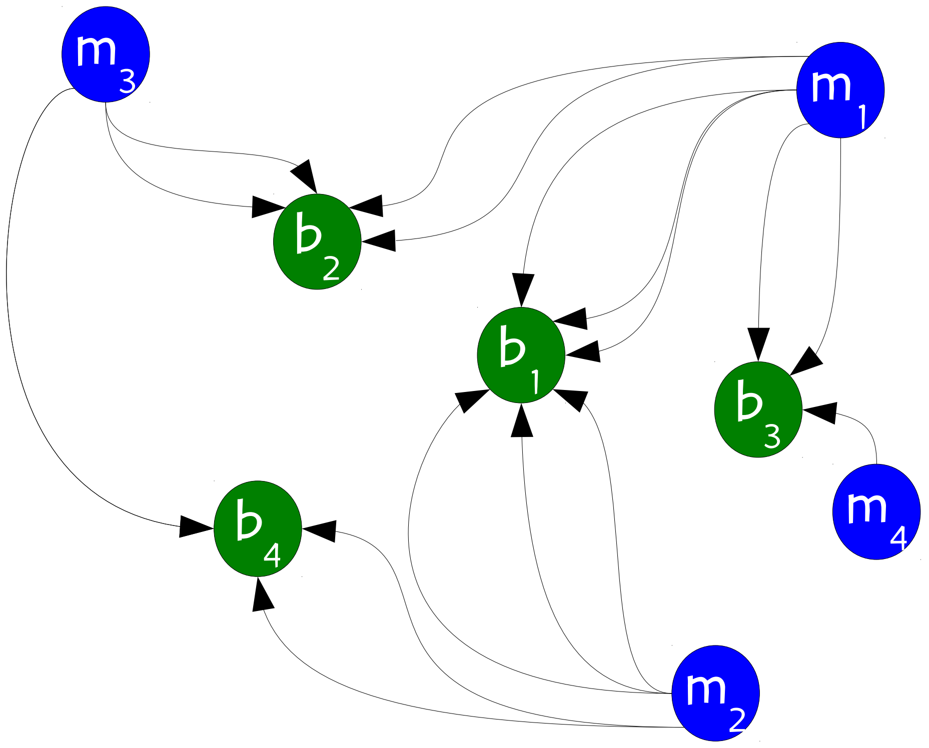
Redundant content distribution

Suppose nodes b_1, b_2, ... want some content (e.g. video).

We want the smallest set **S** of servers such that:

for every b_i there exist m_1, $m_2 \in$ **S** both of which can provide content to b_i

and all paths $m_1 \to b$ are disjoint with all paths $m_2 \to b$



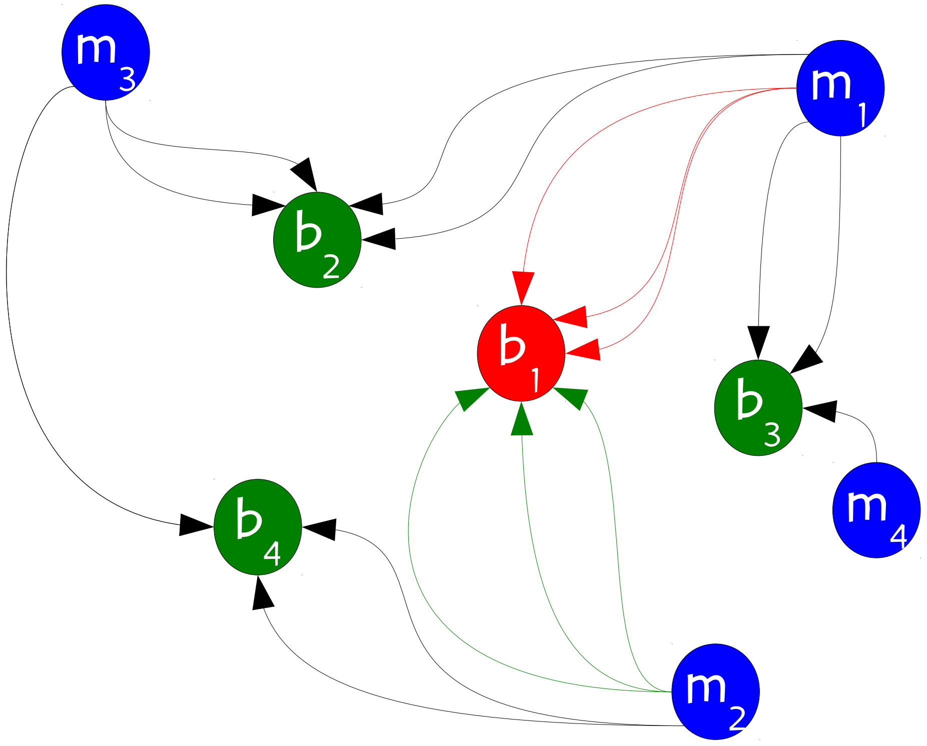
Redundant content distribution

Suppose nodes b_1, b_2, ... want some content (e.g. video).

We want the smallest set **S** of servers such that:

for every b_i there exist m_1, m_2 ∈ **S** both of which can provide content to b_i

and all paths $m_1 \to b$ are disjoint with all paths $m_2 \to b$



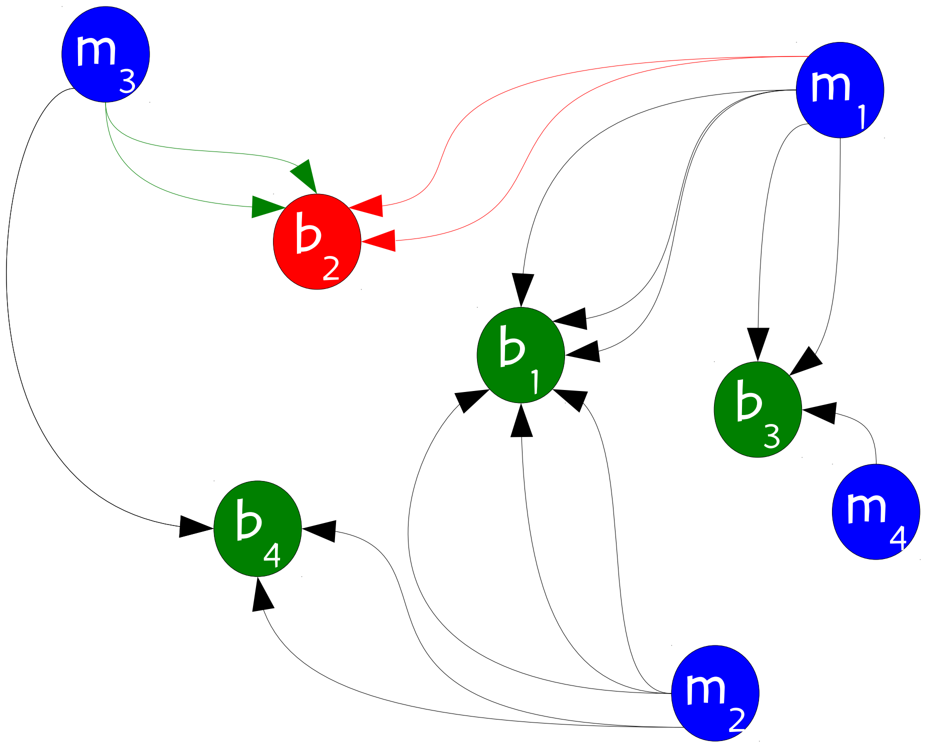
Redundant content distribution

Suppose nodes b_1, b_2, ... want some content (e.g. video).

We want the smallest set **S** of servers such that:

for every b_i there exist m_1, $m_2 \in$ **S** both of which can provide content to b_i

and all paths $m_1 \to b$ are disjoint with all paths $m_2 \to b$



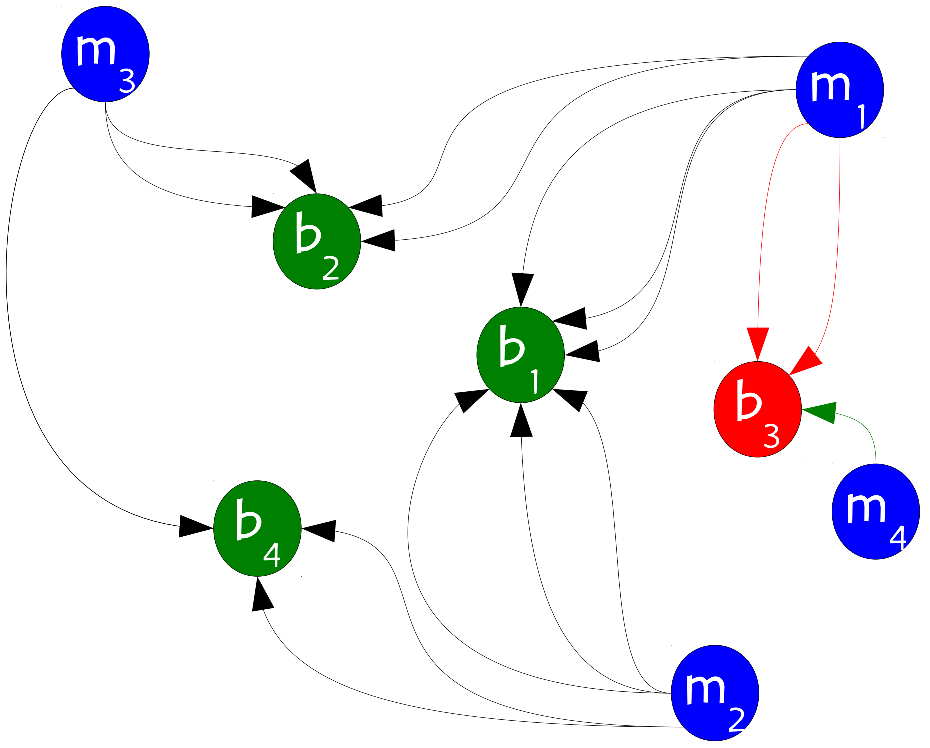
Redundant content distribution

Suppose nodes b_1, b_2, ... want some content (e.g. video).

We want the smallest set **S** of servers such that:

for every b_i there exist m_1, $m_2 \in$ **S** both of which can provide content to b_i

and all paths $m_1 \to b$ are disjoint with all paths $m_2 \to b$



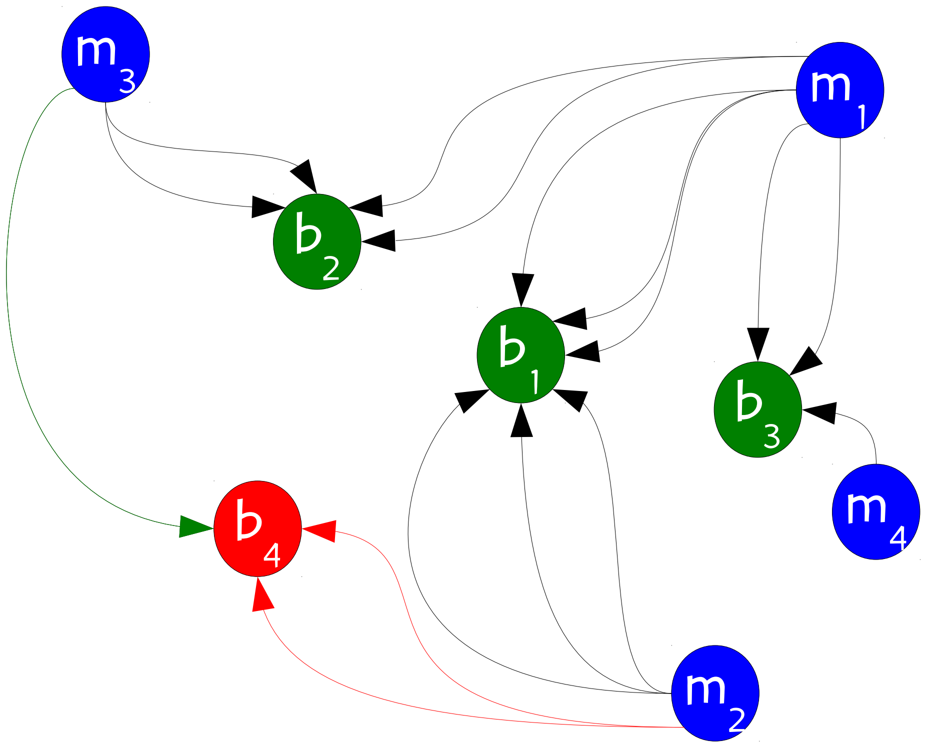
Redundant content distribution

Suppose nodes b_1, b_2, ... want some content (e.g. video).

We want the smallest set **S** of servers such that:

for every b_i there exist m_1, m_2 ∈ **S** both of which can provide content to b_i

and all paths $m_1 \to b$ are disjoint with all paths $m_2 \to b$



Redundant content distribution

- Given:
  - A directed network $G = (V, E)$;
  - A set of nodes $B \subseteq V$ where content-demanding users are located;
  - A set of nodes $M \subseteq V$ where content warehouses can be located;
  - The set of all OSPF paths from m to b, for $m \in M$ and $b \in B$.

Redundant content distribution

- Compute:
  - The set of triples $\{ m_1, m_2, b \}^i$, $i = 1, 2, \ldots, T$, such that all paths from m_1 to b and from m_2 to b are disjoint, where $m_1, m_2 \in M$ and $b \in B$.
  - Note that if $B \cap M \neq \emptyset$, then some triples will be of the type $\{ b, b, b \}$, where $b \in B \cap M$, i.e. a data warehouse that is co-located with a user can provide content to the user by itself.

Redundant content distribution

- Solve the covering by pairs problem:
  - Find a smallest-cardinality set $M^* \subseteq M$ such that for all $b \in B$, there exists a triple $\{ m_1, m_2, b \}$ in the set of triples such that $m_1, m_2 \in M^*$.

Greedy algorithm for covering by pairs

- initialize partial cover $M^* = \{\ \}$

Greedy algorithm for covering by pairs

- initialize partial cover M* = { }
- while M* is not a cover do:

Greedy algorithm for covering by pairs

- initialize partial cover $M^* = \{\ \}$
- while M^* is not a cover do:
  - find $m \in M \setminus M^*$ such that $M^* \cup \{m\}$ covers a maximum number of additional user nodes (break ties by vertex index) and set $M^* = M^* \cup \{m\}$

Greedy algorithm for covering by pairs

- initialize partial cover $M^* = \{\ \}$
- while M^* is not a cover do:
  - find $m \in M \setminus M^*$ such that $M^* \cup \{m\}$ covers a maximum number of additional user nodes (break ties by vertex index) and set $M^* = M^* \cup \{m\}$
  - if no $m \in M \setminus M^*$ yields an increase in coverage, then choose a pair $\{m_1, m_2\} \in M \setminus M^*$ that yields a maximum increase in coverage and set $M^* = M^* \cup \{m_1\} \cup \{m_2\}$

Greedy algorithm for covering by pairs

- initialize partial cover $M^* = \{\ \}$
- while M^* is not a cover do:
  - find $m \in M \setminus M^*$ such that $M^* \cup \{m\}$ covers a maximum number of additional user nodes (break ties by vertex index) and set $M^* = M^* \cup \{m\}$
  - if no $m \in M \setminus M^*$ yields an increase in coverage, then choose a pair $\{m_1,m_2\} \in M \setminus M^*$ that yields a maximum increase in coverage and set $M^* = M^* \cup \{m_1\} \cup \{m_2\}$
  - if no pair exists, then the problem is infeasible

BRKGA for redundant content distribution

BRKGA for the RCD problem

- Encoding:
  - A vector X of N keys randomly generated in the real interval (0,1], where N = |M| is the number of potential data warehouse nodes. The i-th random key corresponds to the i-th potential data warehouse node.

BRKGA for the RCD problem

- Encoding:
  - A vector X of N keys randomly generated in the real interval (0,1], where N = |M| is the number of potential data warehouse nodes. The i-th random key corresponds to the i-th potential data warehouse node.
- Decoder:

BRKGA for the RCD problem

- Encoding:
  - A vector X of N keys randomly generated in the real interval $(0,1]$, where $N = |M|$ is the number of potential data warehouse nodes. The i-th random key corresponds to the i-th potential data warehouse node.
- Decoder:
  - For $i = 1, \ldots, N$: if $X(i) > ½$, add i-th data warehouse node to solution

BRKGA for the RCD problem

- Encoding:
  - A vector X of N keys randomly generated in the real interval (0,1], where $N = |M|$ is the number of potential data warehouse nodes. The i-th random key corresponds to the i-th potential data warehouse node.
- Decoder:
  - For $i = 1, \ldots, N$: if $X(i) > \frac{1}{2}$, add i-th data warehouse node to solution
  - If solution is feasible, i.e. all users are covered: STOP

BRKGA for the RCD problem

- Encoding:
  - A vector X of N keys randomly generated in the real interval $(0,1]$, where $N = |M|$ is the number of potential data warehouse nodes. The i-th random key corresponds to the i-th potential data warehouse node.
- Decoder:
  - For $i = 1, \ldots, N$: if $X(i) > 1/2$, add i-th data warehouse node to solution
  - If solution is feasible, i.e. all users are covered: STOP
  - Else, apply greedy algorithm to cover uncovered user nodes.

BRKGA for the RCD problem

- Size of population: N (number of monitoring nodes)
- Size of elite set: 15% of N
- Size of mutant set: 10% of N
- Biased coin probability: 70%
- Stop after N generations without improvement of best found solution

Another application: Host placement for end-to-end monitoring

- Internet service provider (ISP) delivers virtual private network (VPN) service to customers.

Another application: Host placement for end-to-end monitoring

- Internet service provider (ISP) delivers virtual private network (VPN) service to customers.
- The ISP agrees to send traffic between locations specified by the customer and promises to provide certain level of service on the connections.

Another application: Host placement for end-to-end monitoring

- Internet service provider (ISP) delivers virtual private network (VPN) service to customers.
- The ISP agrees to send traffic between locations specified by the customer and promises to provide certain level of service on the connections.
- A key service quality metric is packet loss rate.

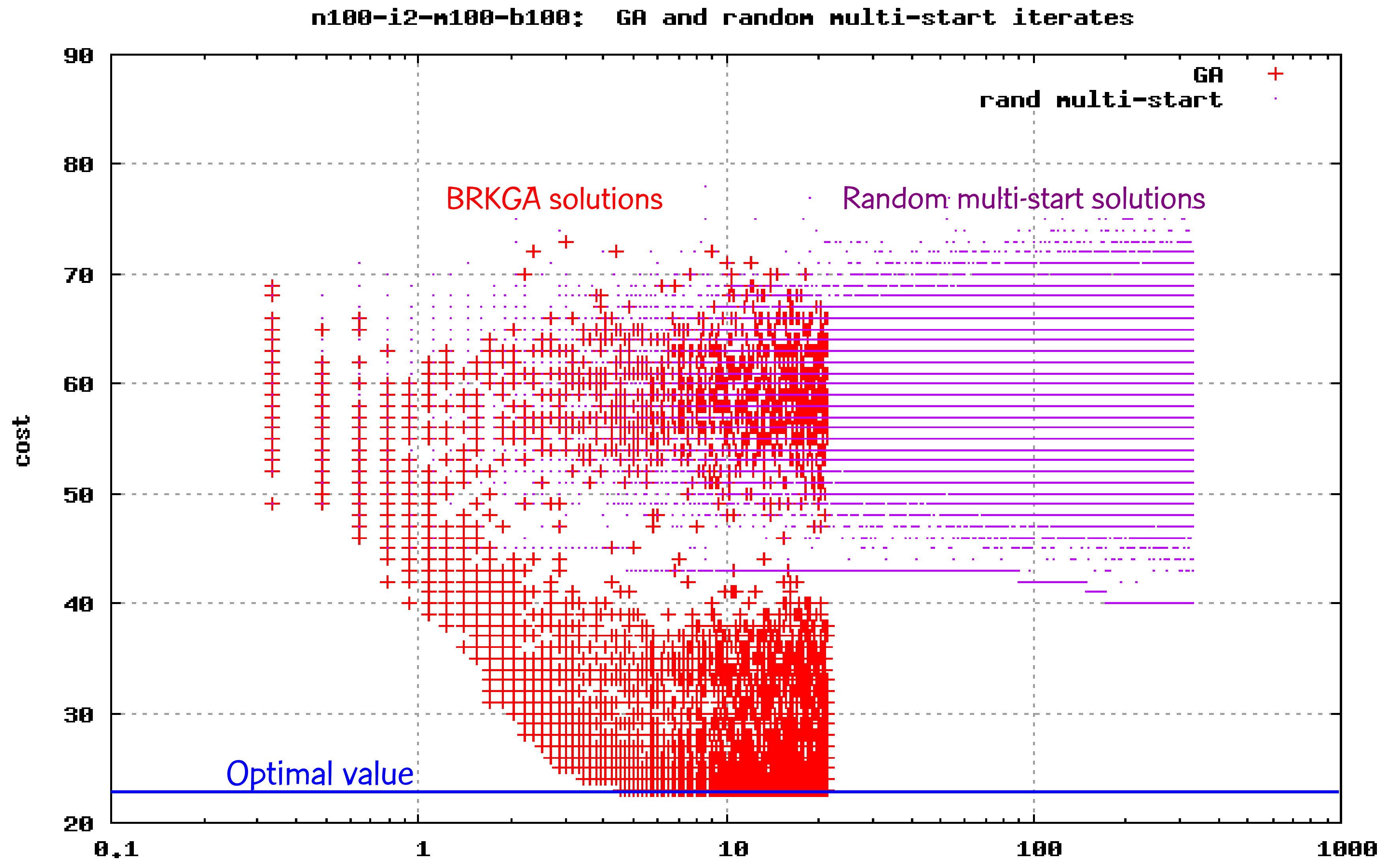
Another application: Host placement for end-to-end monitoring

- Internet service provider (ISP) delivers virtual private network (VPN) service to customers.
- The ISP agrees to send traffic between locations specified by the customer and promises to provide certain level of service on the connections.
- A key service quality metric is packet loss rate.
- We want to minimize the number of monitoring equipment placed in the network to measure packet loss rate: This is a type of covering by pairs problem.

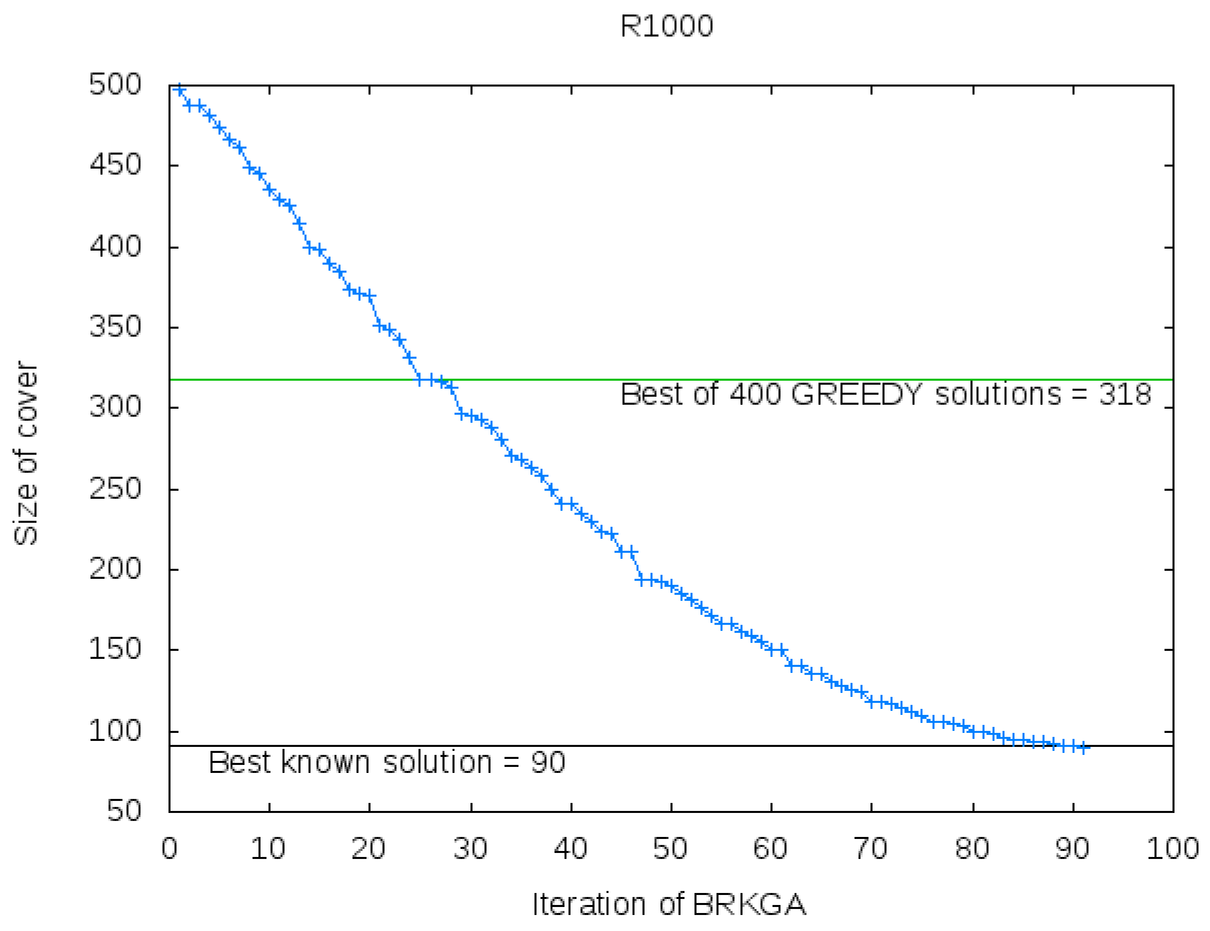
at&t
Your world. Delivered.

n100-i2-m100-b100 (opt = 23)

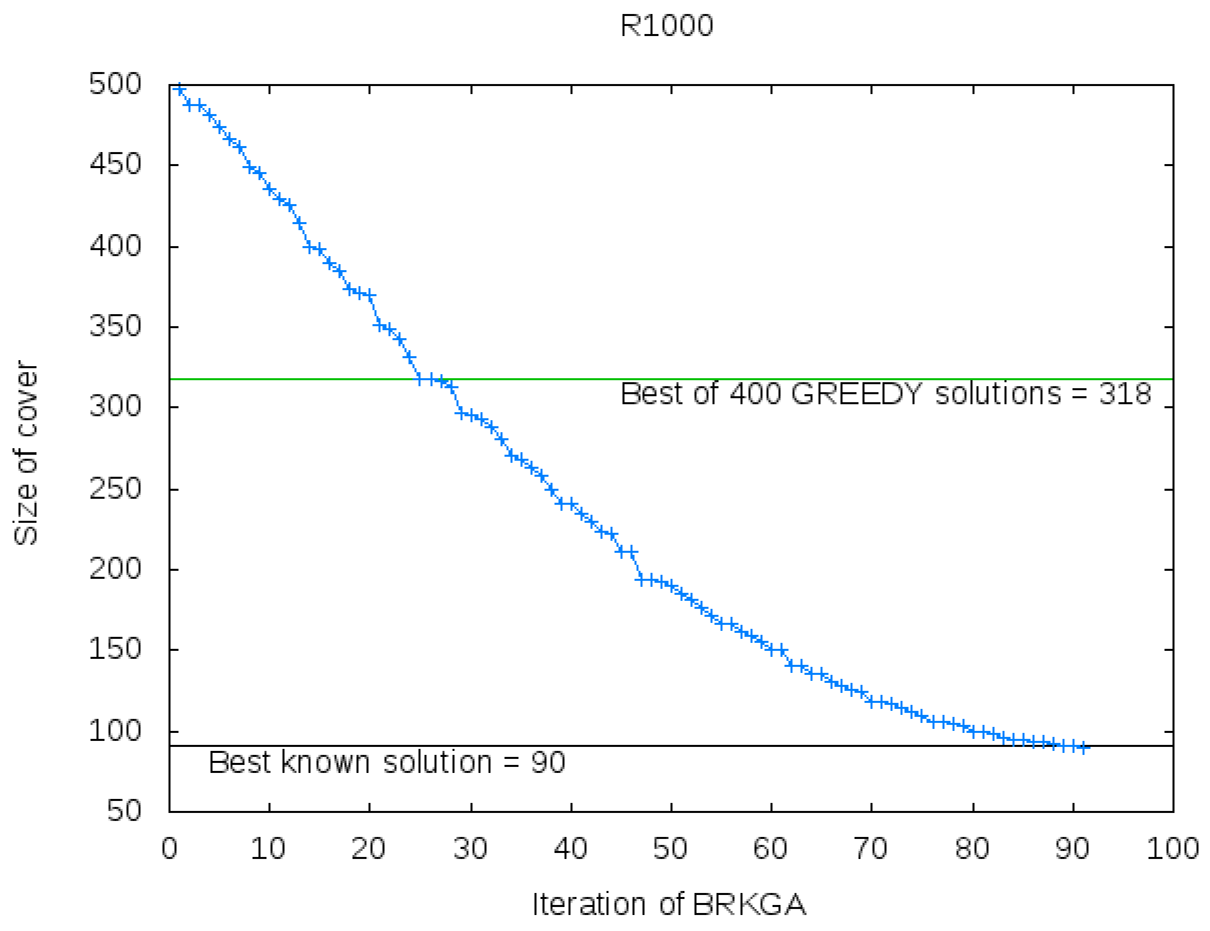
solution



Time (ibm t41 secs)



Real-world instance derived from a proprietary Tier-1 Internet Service Provider (ISP) backbone network using OSPF for routing.



Size of network: about 1000 nodes, where almost all can store content and about 90% have content-demanding users. Over 45 million triples.

Concluding remarks

Concluding remarks

- A small modification of Bean's RKGA results in a BRKGA.

Concluding remarks

- A small modification of Bean's RKGA results in a BRKGA.
- Though small, this modification, leads to significant performance improvements.

Concluding remarks

- A small modification of Bean's RKGA results in a BRKGA.
- Though small, this modification, leads to significant performance improvements.
- BRKGA are true metaheuristics: they coordinate simple heuristics and produce better solutions than the simple heuristics alone.

Concluding remarks

- A small modification of Bean's RKGA results in a BRKGA.
- Though small, this modification, leads to significant performance improvements.
- BRKGA are true metaheuristics: they coordinate simple heuristics and produce better solutions than the simple heuristics alone.
- Problem independent module of a BRKGA needs to be implemented once and can be reused for a wide range of problems. User can focus on problem dependent module.

Concluding remarks

- A small modification of Bean's RKGA results in a BRKGA.
- Though small, this modification, leads to significant performance improvements.
- BRKGA are true metaheuristics: they coordinate simple heuristics and produce better solutions than the simple heuristics alone.
- Problem independent module of a BRKGA needs to be implemented once and can be reused for a wide range of problems. User can focus on problem dependent module.
- BRKGA heuristics are highly parallelizable. Calls to decoder are independent.

Concluding remarks

- BRKGA have been applied in a wide range of application areas, including scheduling, packing, cutting, tollbooth assignment, ...

Concluding remarks

- BRKGA have been applied in a wide range of application areas, including scheduling, packing, cutting, tollbooth assignment, ...
- We have had only a small glimpse at BRKGA applications to problems arising in telecommunications.

Concluding remarks

- BRKGA have been applied in a wide range of application areas, including scheduling, packing, cutting, tollbooth assignment, ...
- We have had only a small glimpse at BRKGA applications to problems arising in telecommunications.
- The BRKGAs described in this talk are all state-of-the-art heuristics for these applications

Concluding remarks

- BRKGA have been applied in a wide range of application areas, including scheduling, packing, cutting, tollbooth assignment, …
- We have had only a small glimpse at BRKGA applications to problems arising in telecommunications.
- The BRKGAs described in this talk are all state-of-the-art heuristics for these applications
- We are currently working on a number of other applications in telecommunications, including the degree-constrained and the capacitated spanning tree problems and a metropolitan network design problem.

Thanks!

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

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