Biased random-key genetic algorithms

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

mgcr@research.att.com

Talk given at XLV Symposium of the Brazilian Operational Research Society (XLV SBPO)
Natal, RN, Brazil ✤ September 16-19, 2013

Summary

- Metaheuristics and basic concepts of genetic algorithms
- Random-key genetic algorithm of Bean (1994)
- Biased random-key genetic algorithms (BRKGA)
  - Encoding / Decoding
  - Initial population
  - Evolutionary mechanisms
  - Problem independent / problem dependent components
  - Multi-start strategy
  - Specifying a BRKGA
  - Application programming interface (API) for BRKGA

Metaheuristics

Metaheuristics are heuristics to devise heuristics.

Metaheuristics

Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone.

Metaheuristics

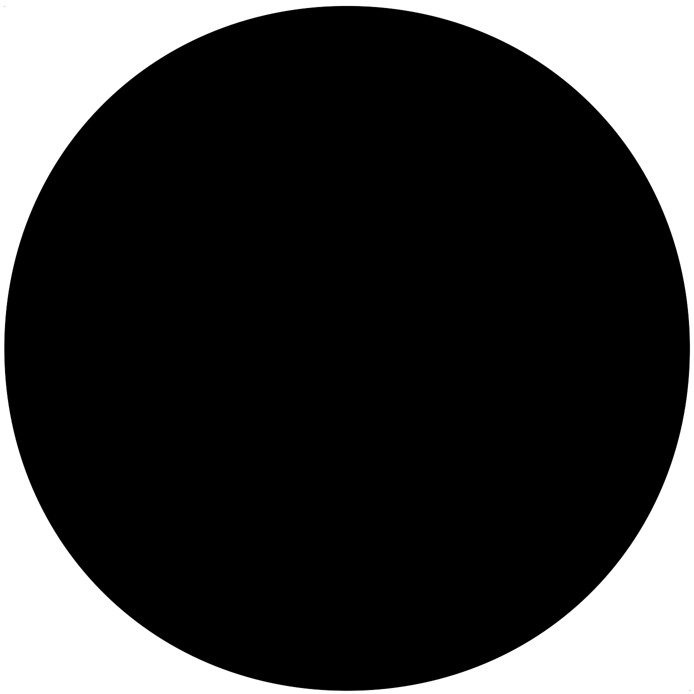
Metaheuristics are high level procedures that coordinate simple heuristics, such as local search, to find solutions that are of better quality than those found by the simple heuristics alone.

Examples: GRASP and C-GRASP, simulated annealing, genetic algorithms, tabu search, scatter search, ant colony optimization, variable neighborhood search, and biased random-key genetic algorithms (BRKGA).

Genetic algorithms

Genetic algorithms

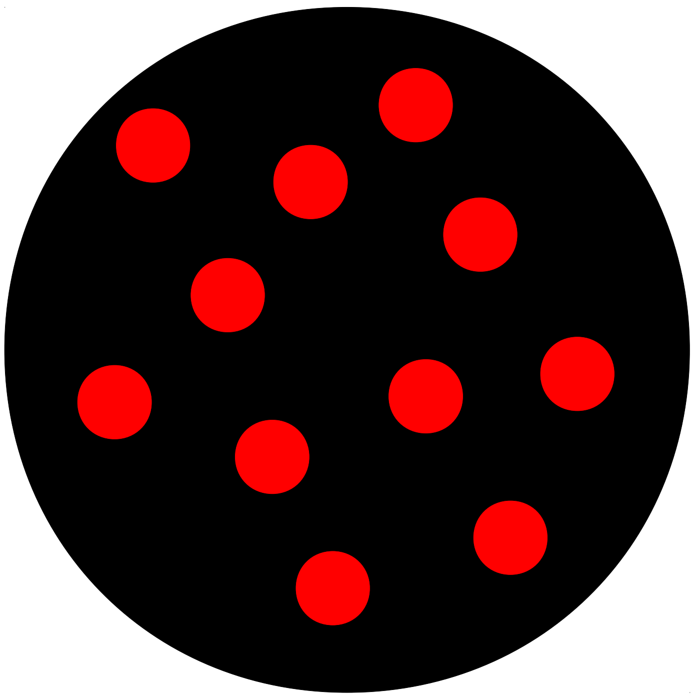
Holland (1975)



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

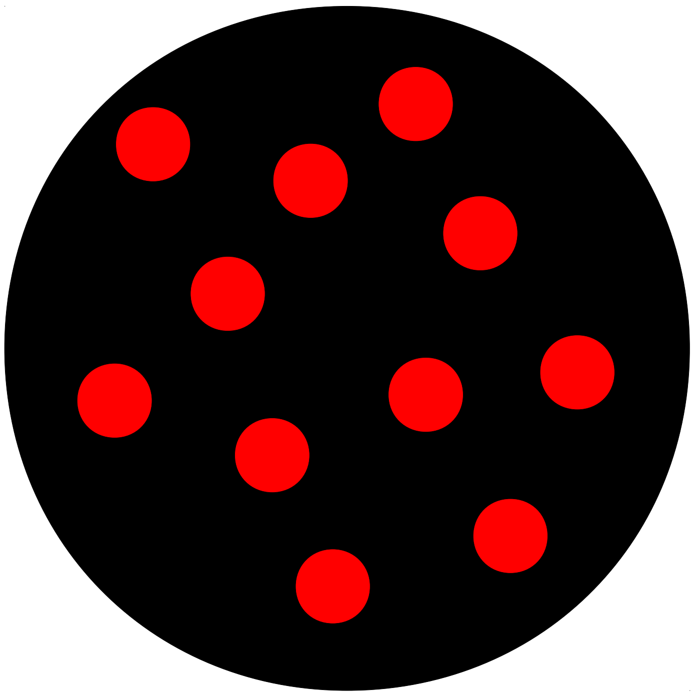
Individual: solution

Genetic algorithms



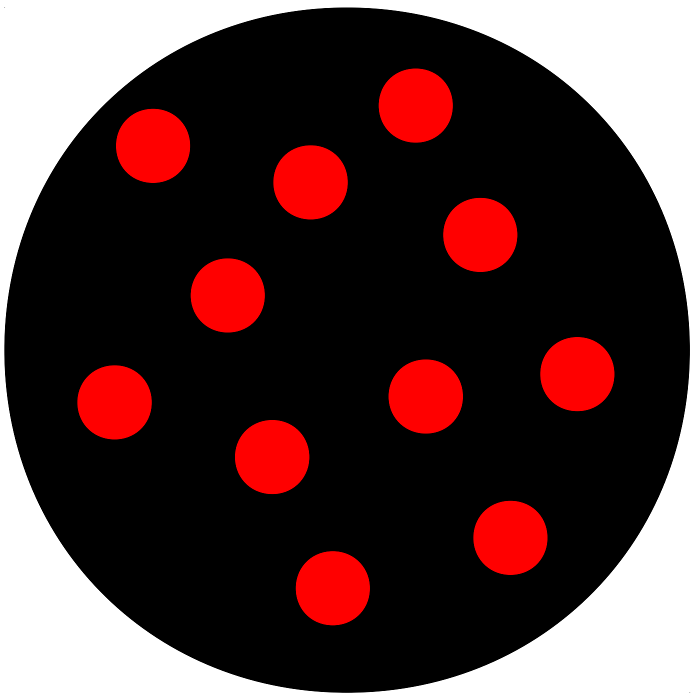
Individual: solution (chromosome = string of genes)
Population: set of fixed number of individuals

Genetic algorithms



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

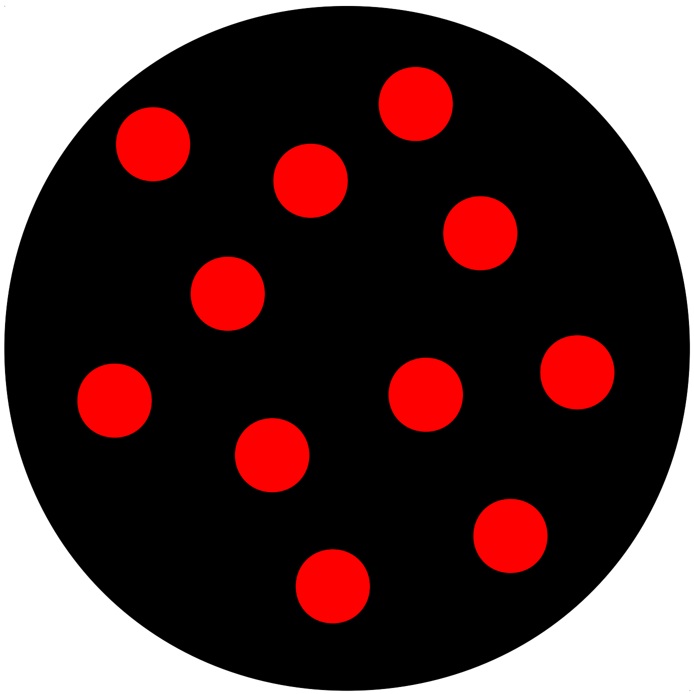
Genetic algorithms



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

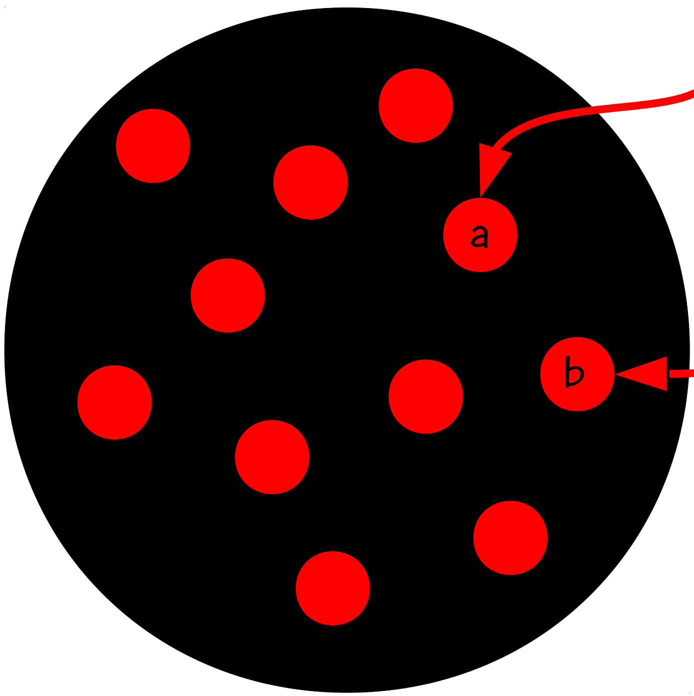


Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

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

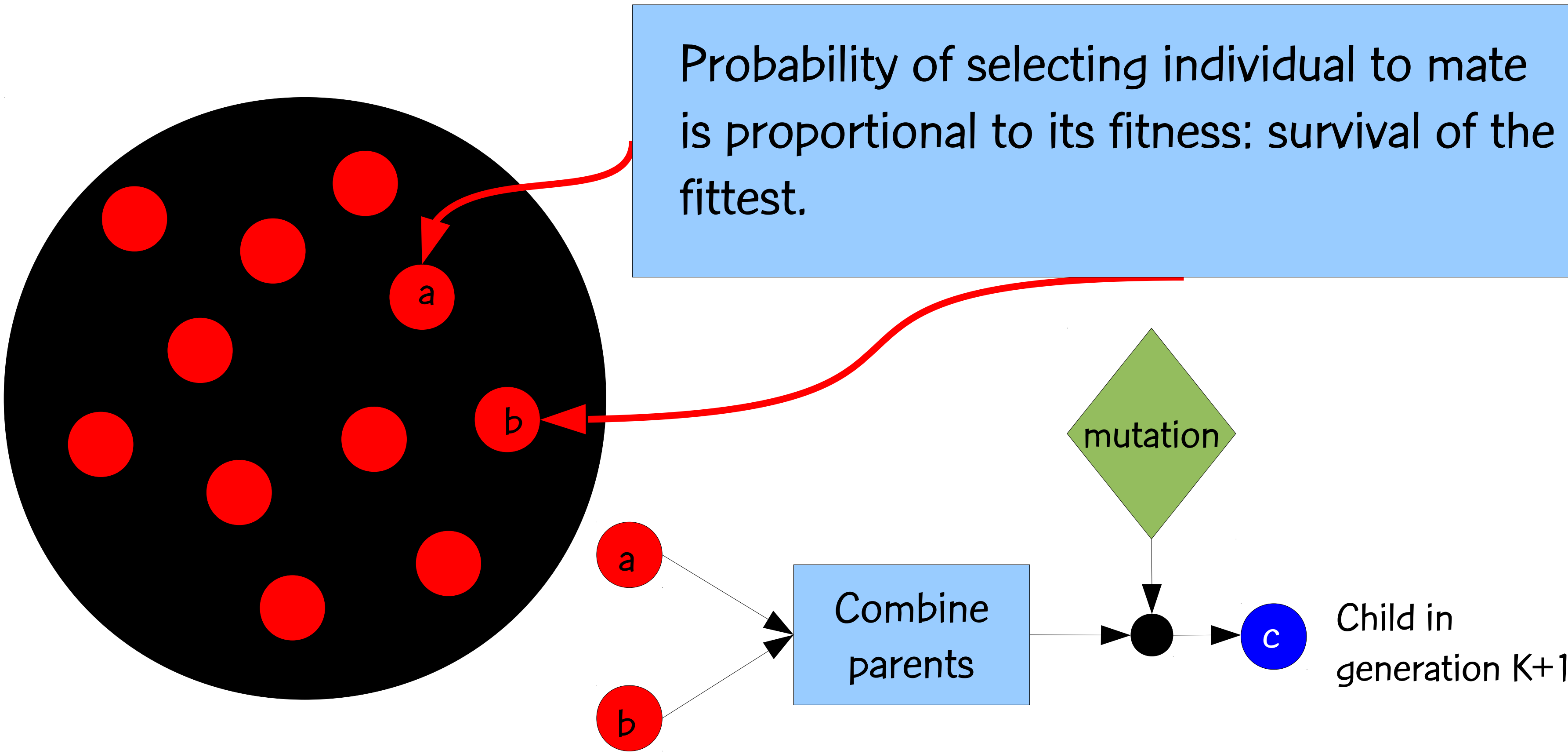
Individuals from one generation are combined to produce offspring that make up next generation.

Genetic algorithms



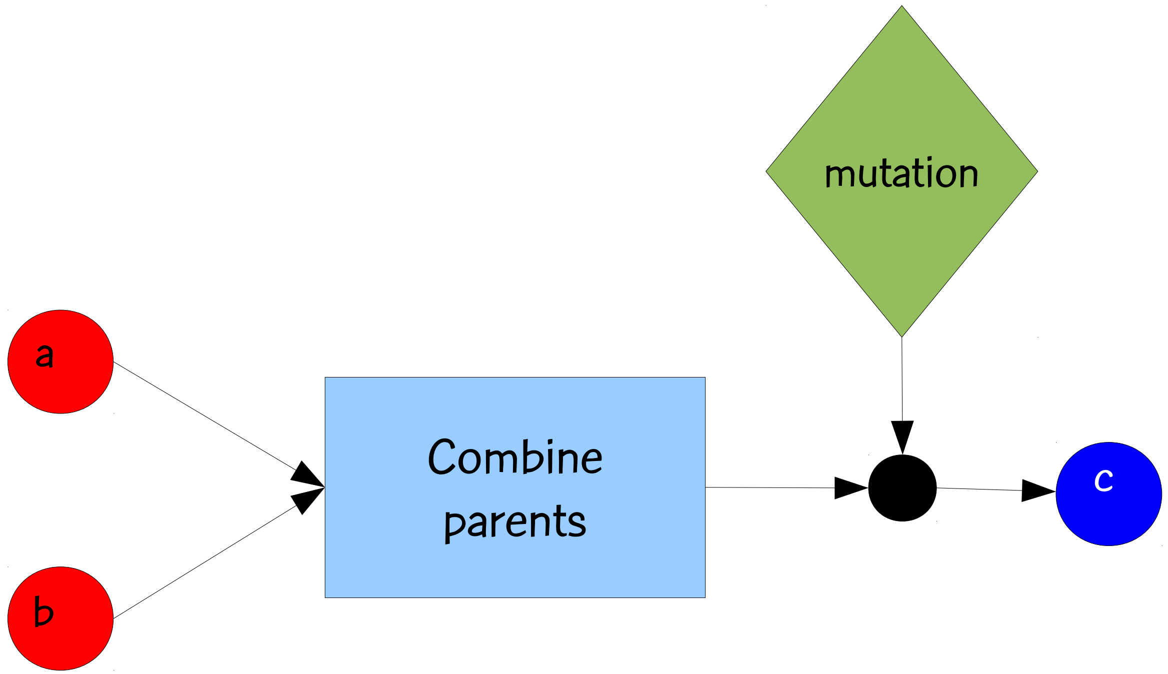
Probability of selecting individual to mate is proportional to its fitness: survival of the fittest.

Genetic algorithms

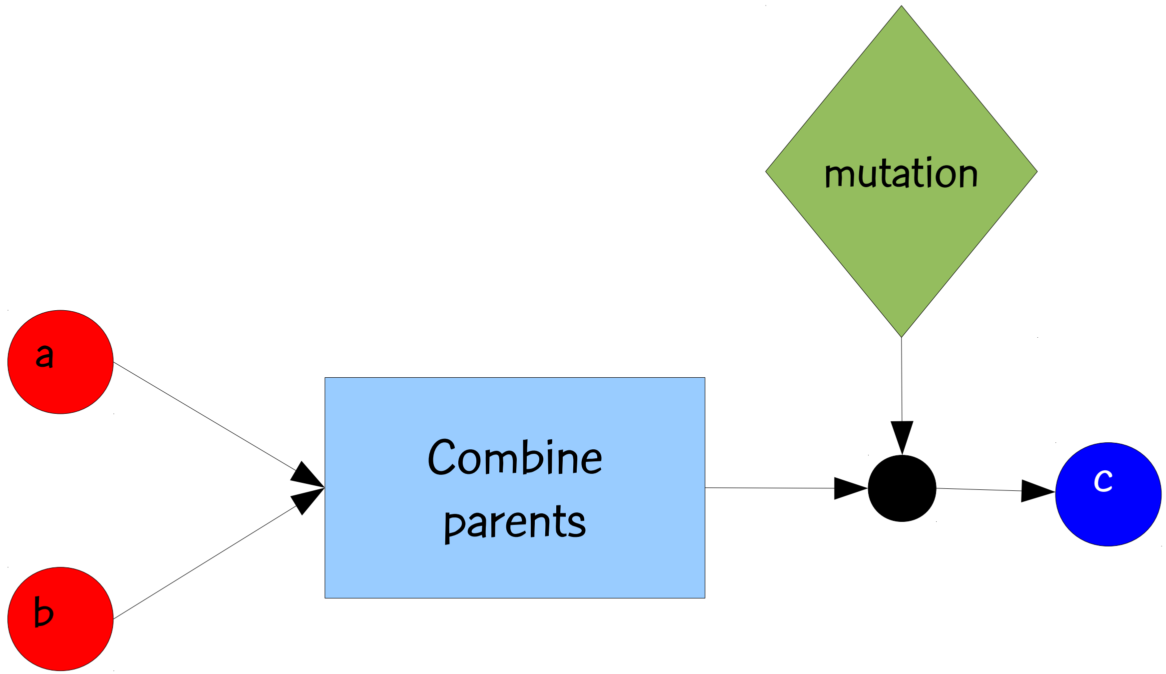


Parents drawn from generation K

Crossover and mutation



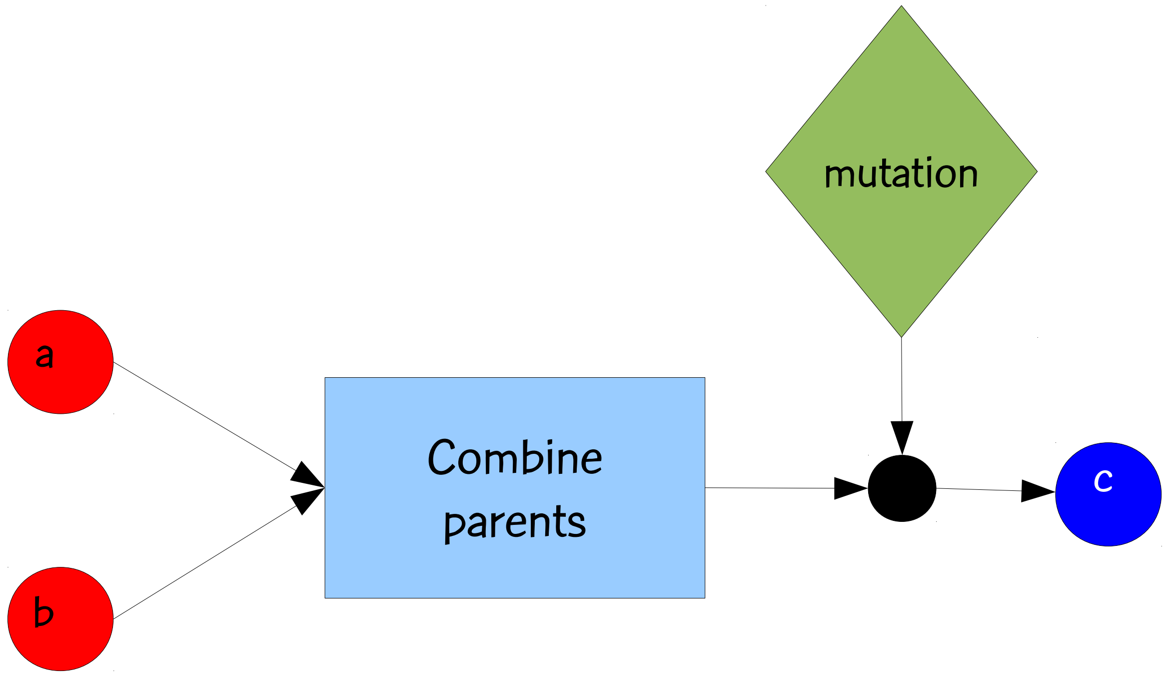
Crossover and mutation



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

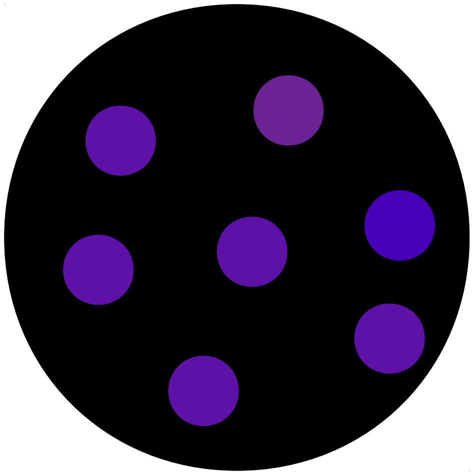
Intensification of search

Crossover and mutation

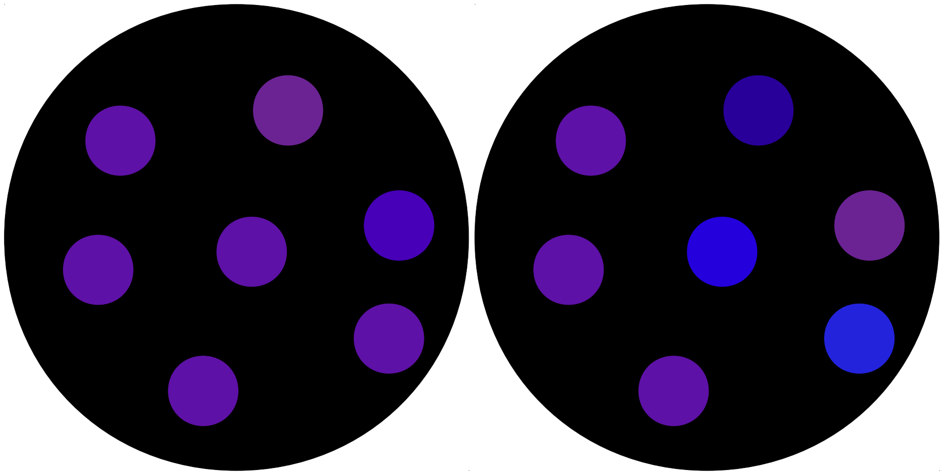


Mutation: Randomly changes chromosome of offspring …
Driver of evolutionary process …
Diversification of search

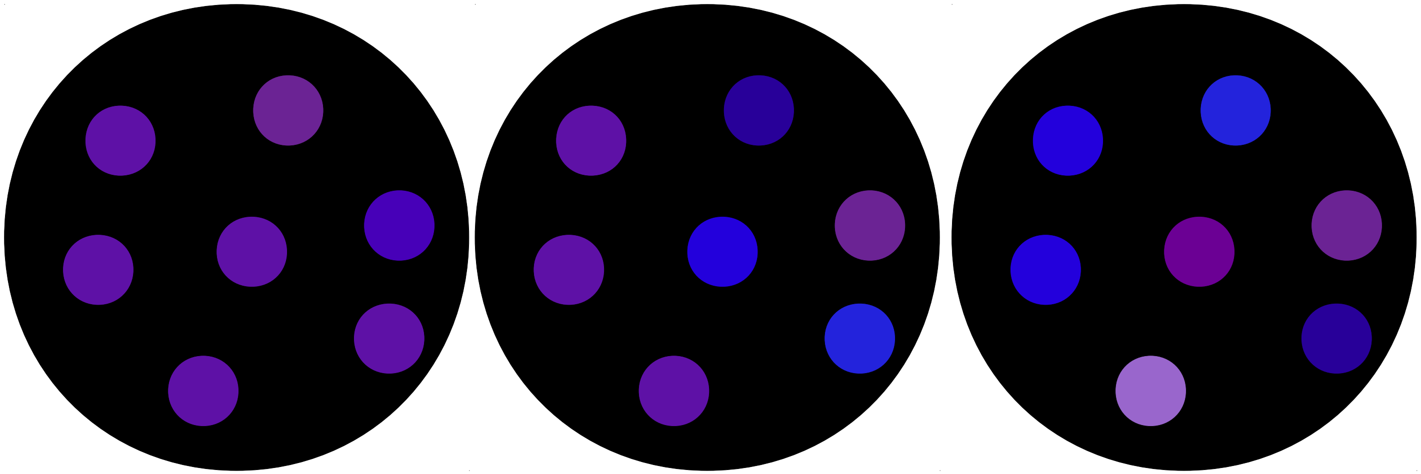
Evolution of solutions



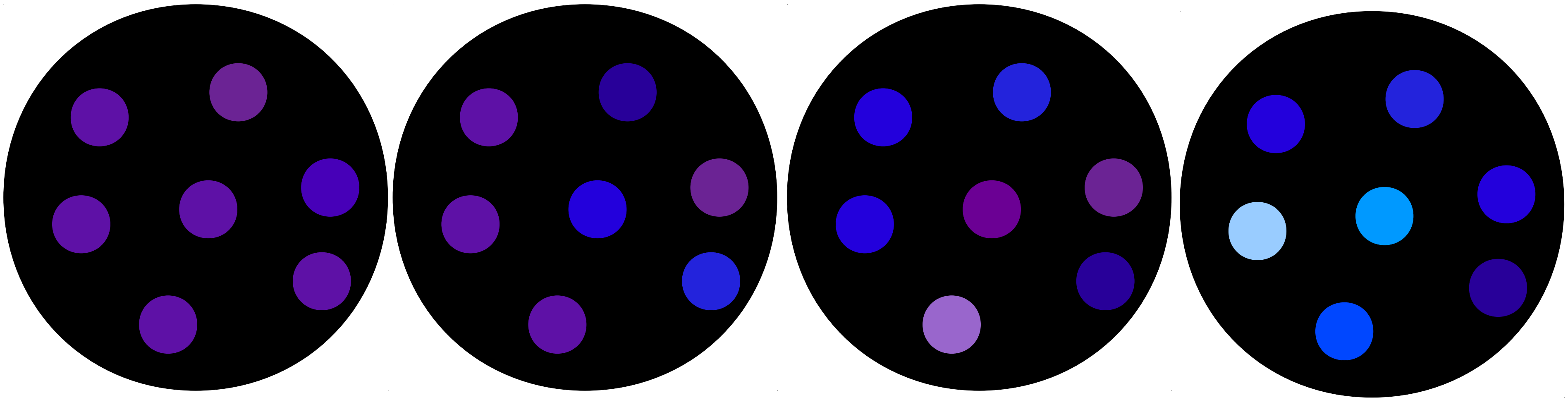
Evolution of solutions



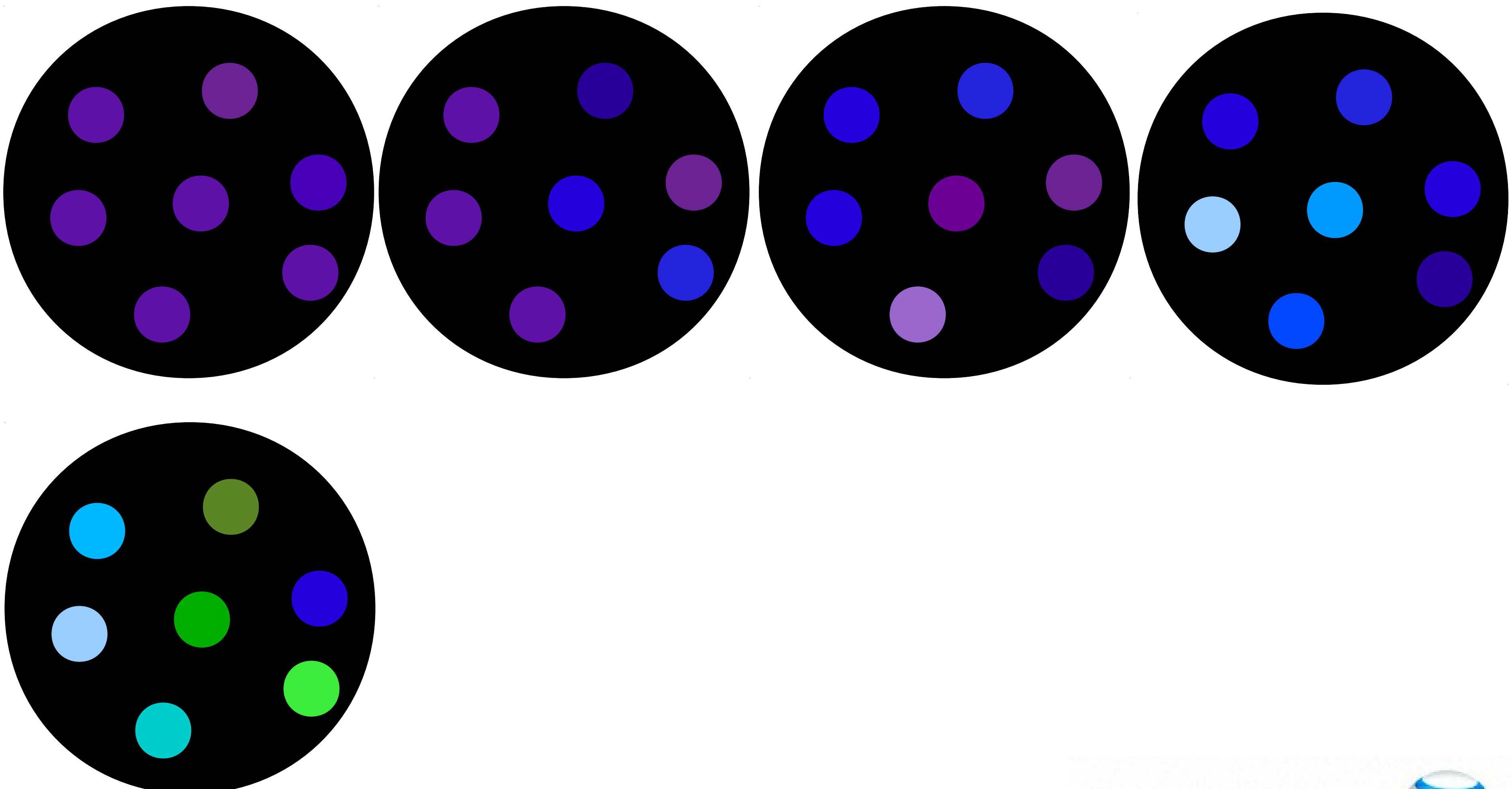
Evolution of solutions



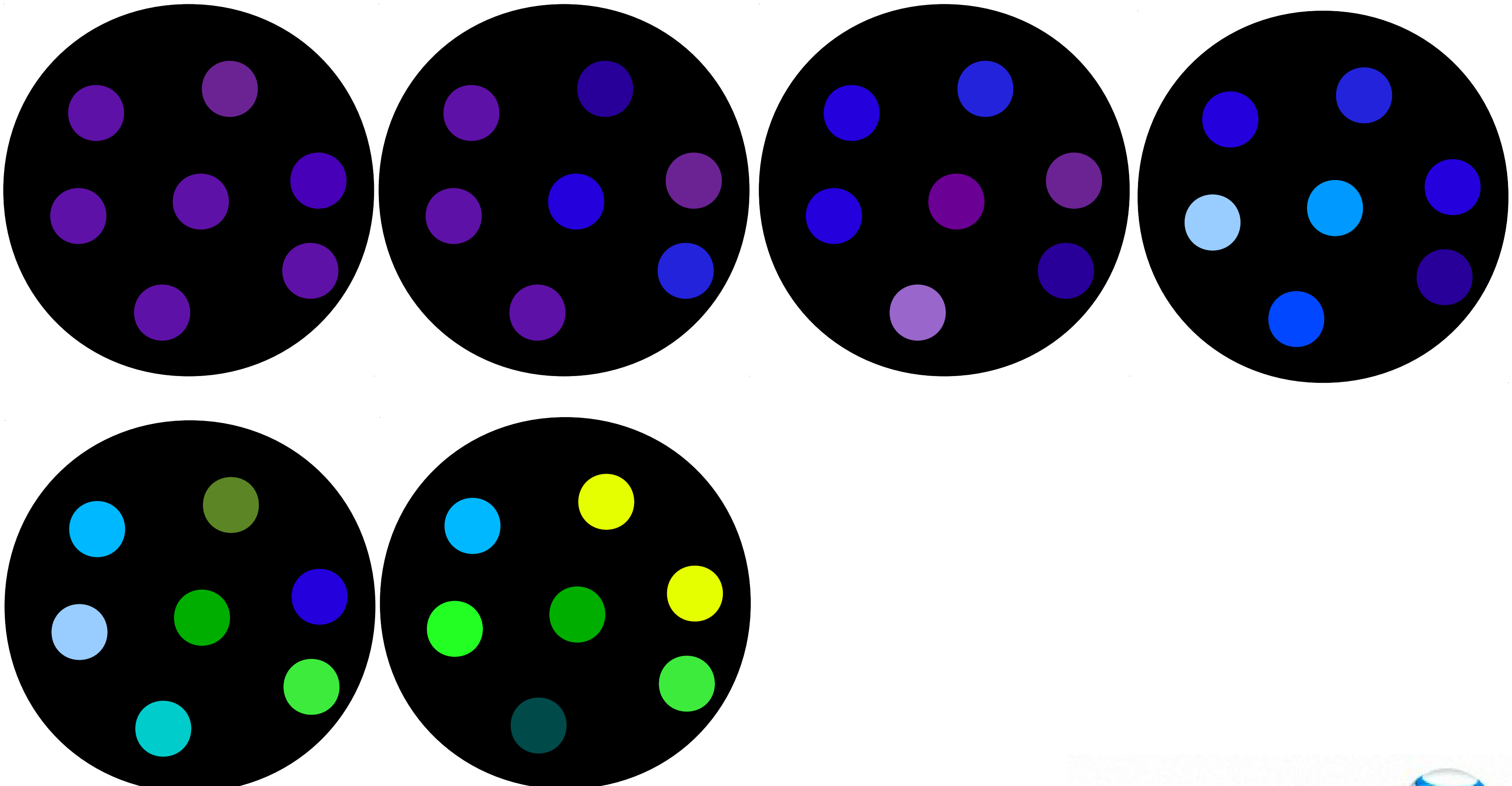
Evolution of solutions



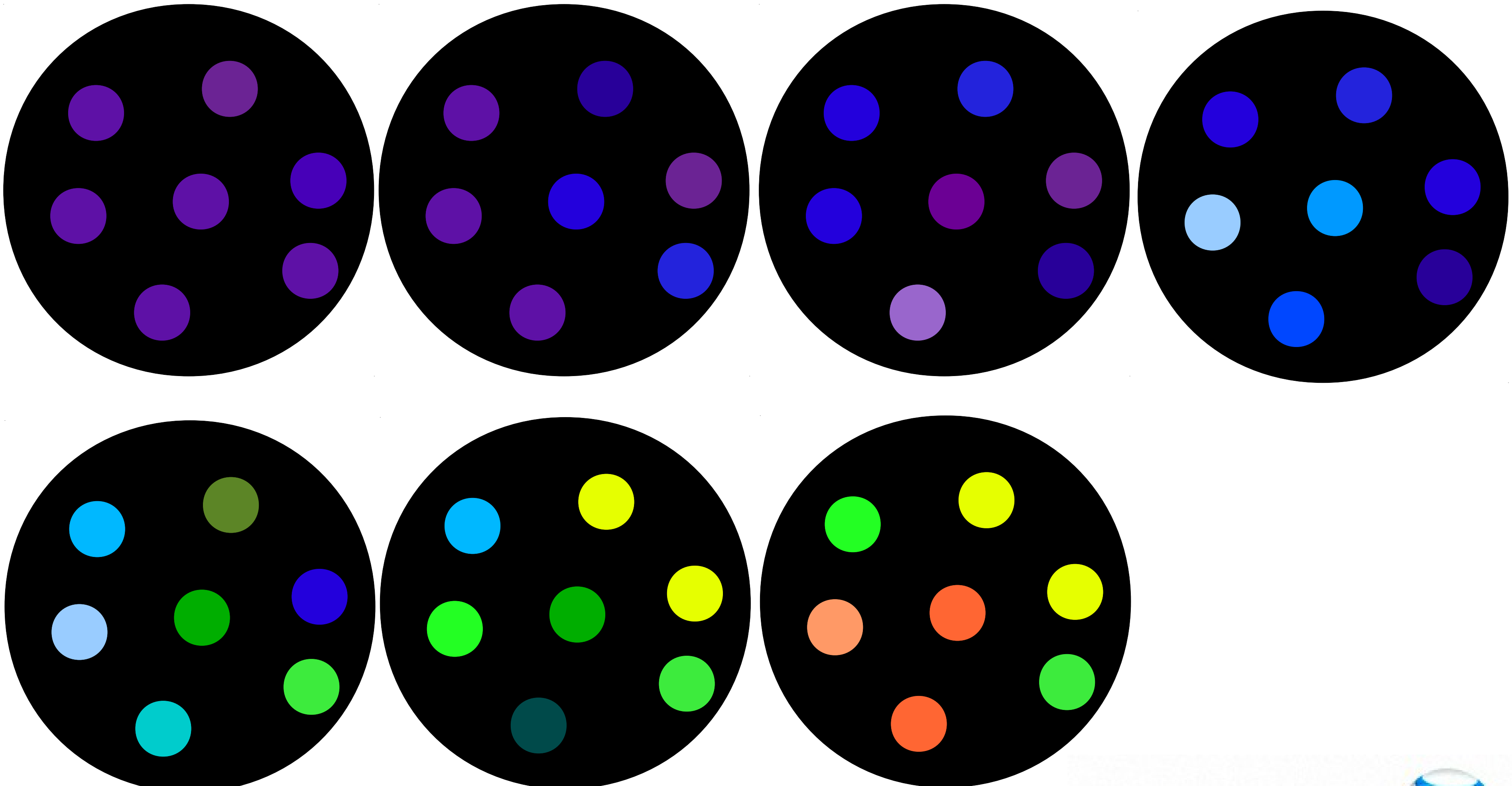
Evolution of solutions



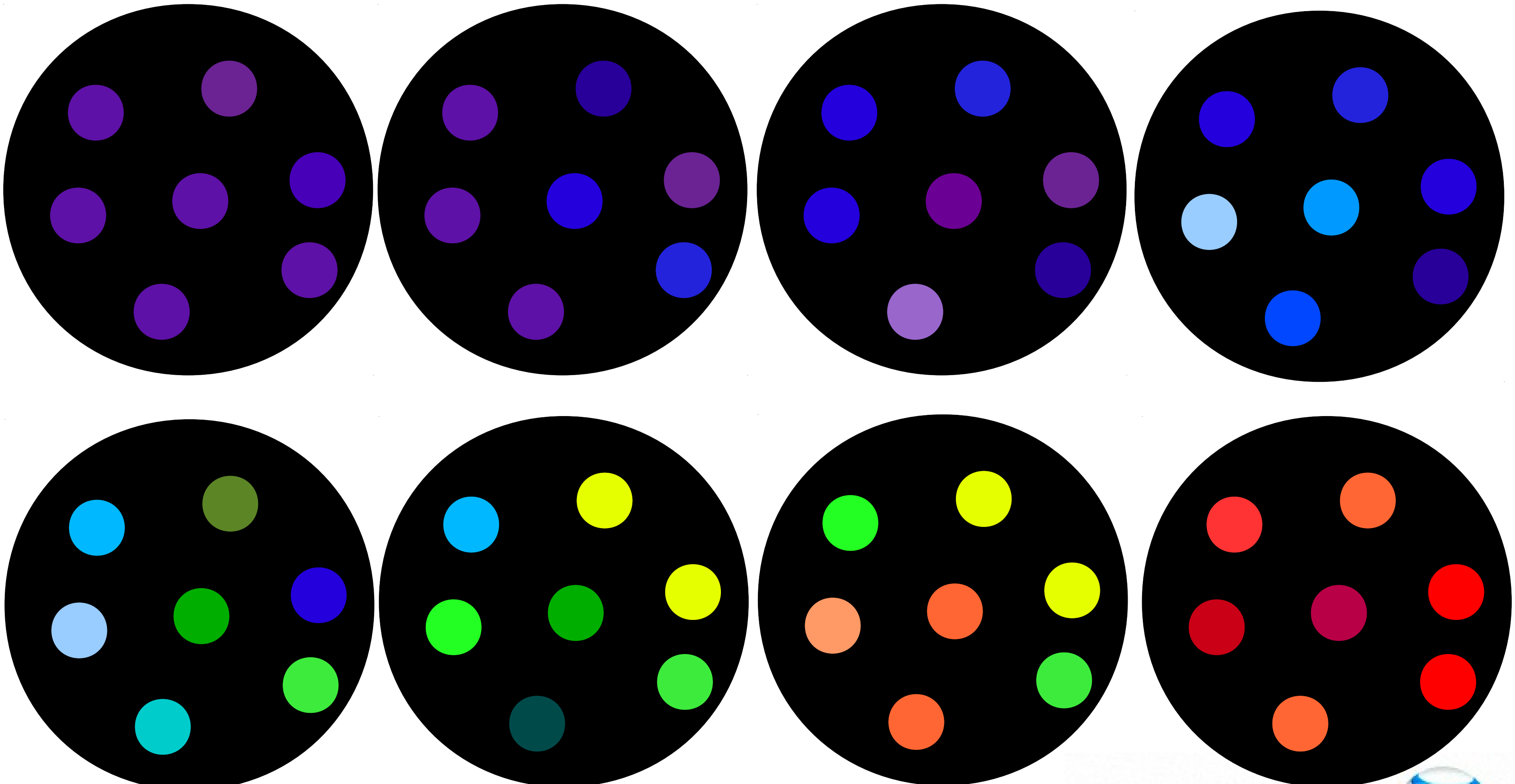
Evolution of solutions



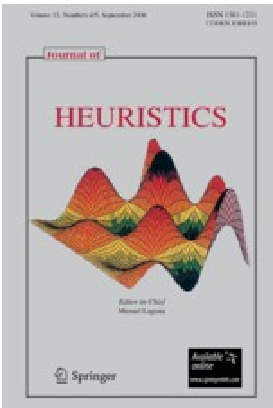
Evolution of solutions



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 inputs a vector of random keys and outputs a feasible solution of the 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

[1, 2, 3, 4, 5]

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

Decode by sorting vector of random keys

[1, 2, 4, 5, 3]

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

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

[1, 2, 3, 4, 5]

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

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

Encoding

[1, 2, 3, 4, 5]

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

Decode by sorting vector of random keys

[1, 2, 4, 5, 3]

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

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 ∈ [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 $i = 1,\ldots,5$.

Rethink Possible

att.com/rethinkpossible

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 = (0.25, 0.19, 0.67, 0.05, 0.89)
s(1) s(2) s(3) s(4) s(5)

GAs and random keys

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

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

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

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)

Rethink Possible

att.com/rethinkpossible

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)

Rethink Possible

att.com/rethinkpossible

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

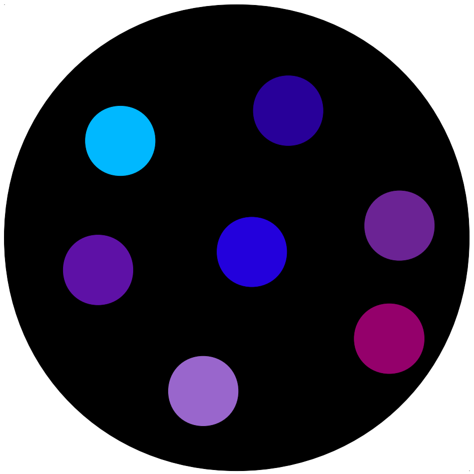
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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.

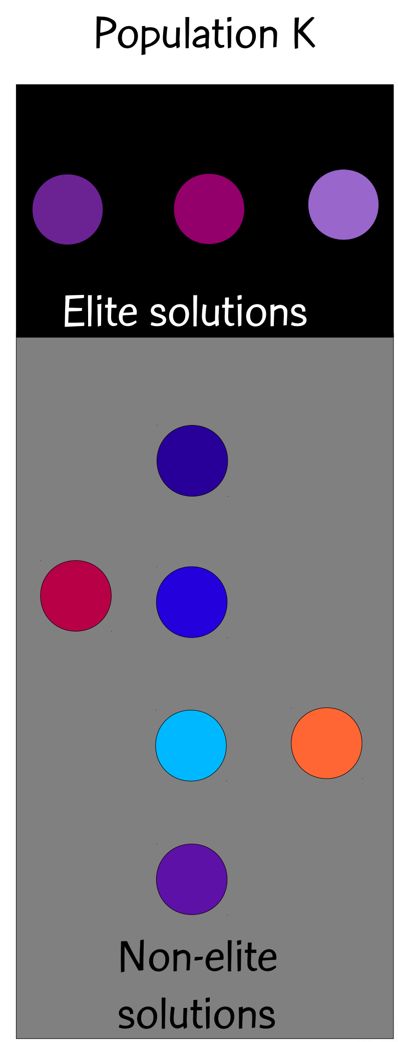
GAs and random keys

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).



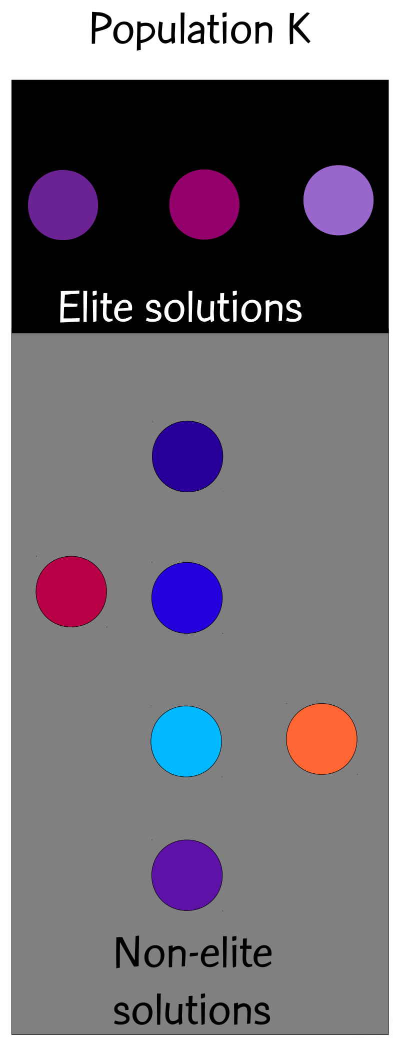
GAs and random keys

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



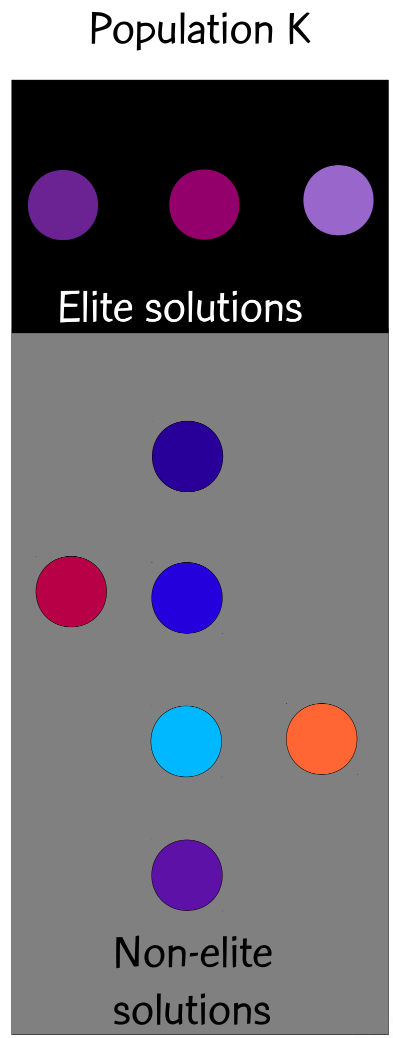
GAs and random keys

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



GAs and random keys

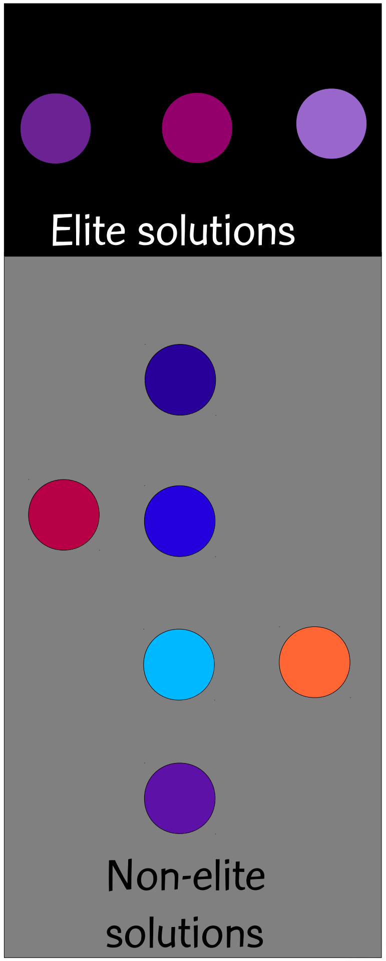
At the K-th generation, compute the cost of each solution and partition the solutions into two sets: elite solutions and non-elite solutions.



GAs and random keys

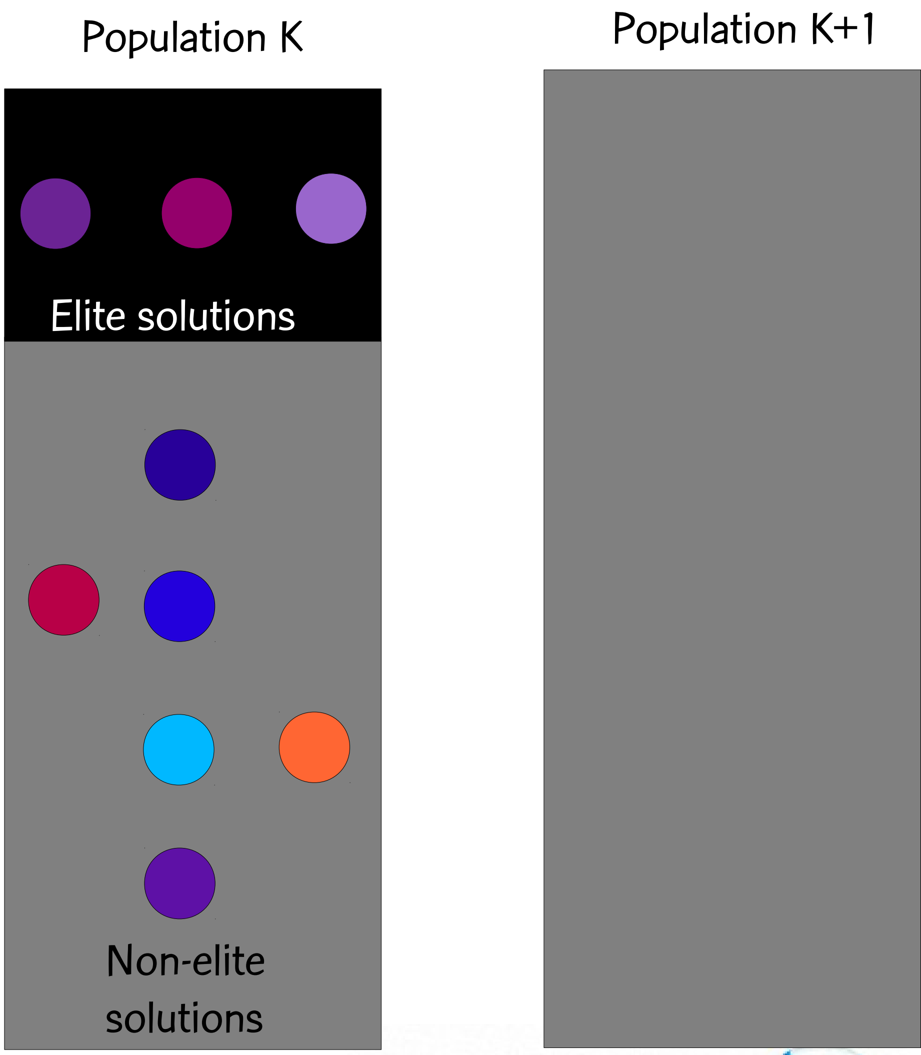
At the K-th generation, compute the cost of each solution and partition the solutions into two sets: elite solutions and non-elite solutions. Elite set should be smaller of the two sets and contain best solutions.

Population K



GAs and random keys

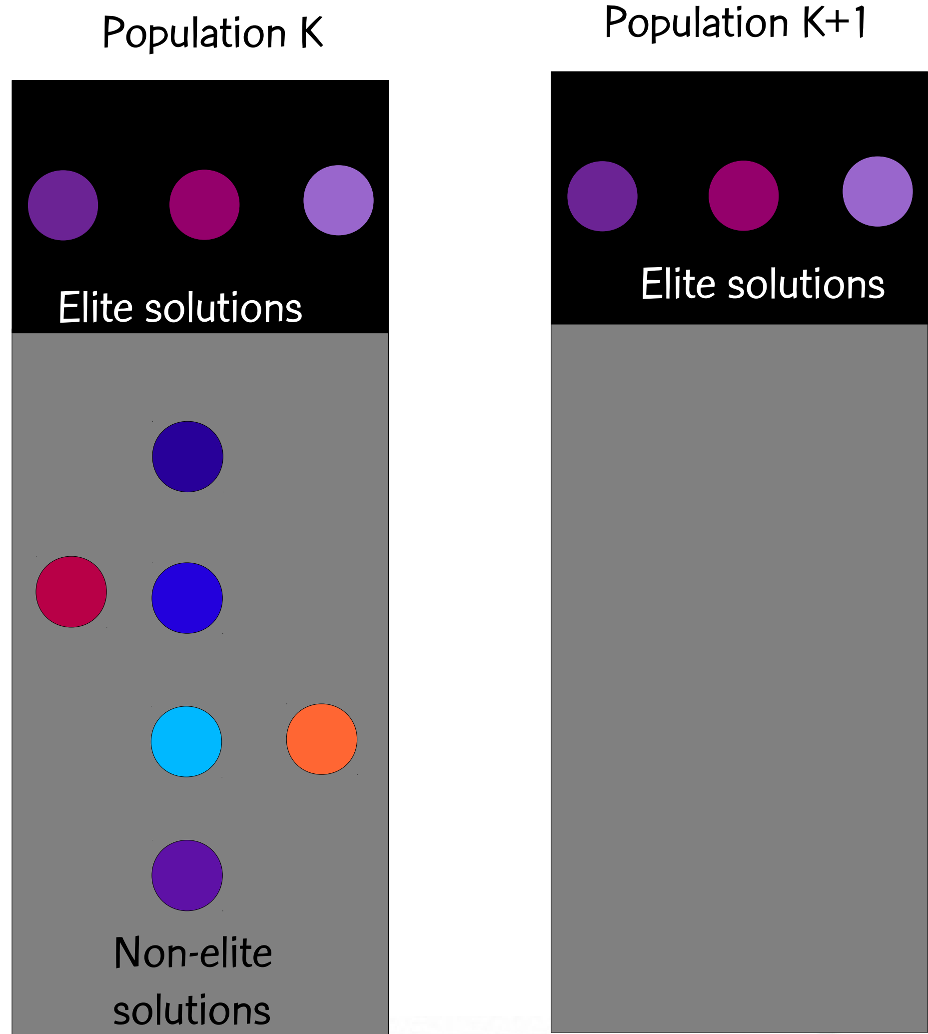
Evolutionary dynamics



GAs and random keys

Evolutionary dynamics

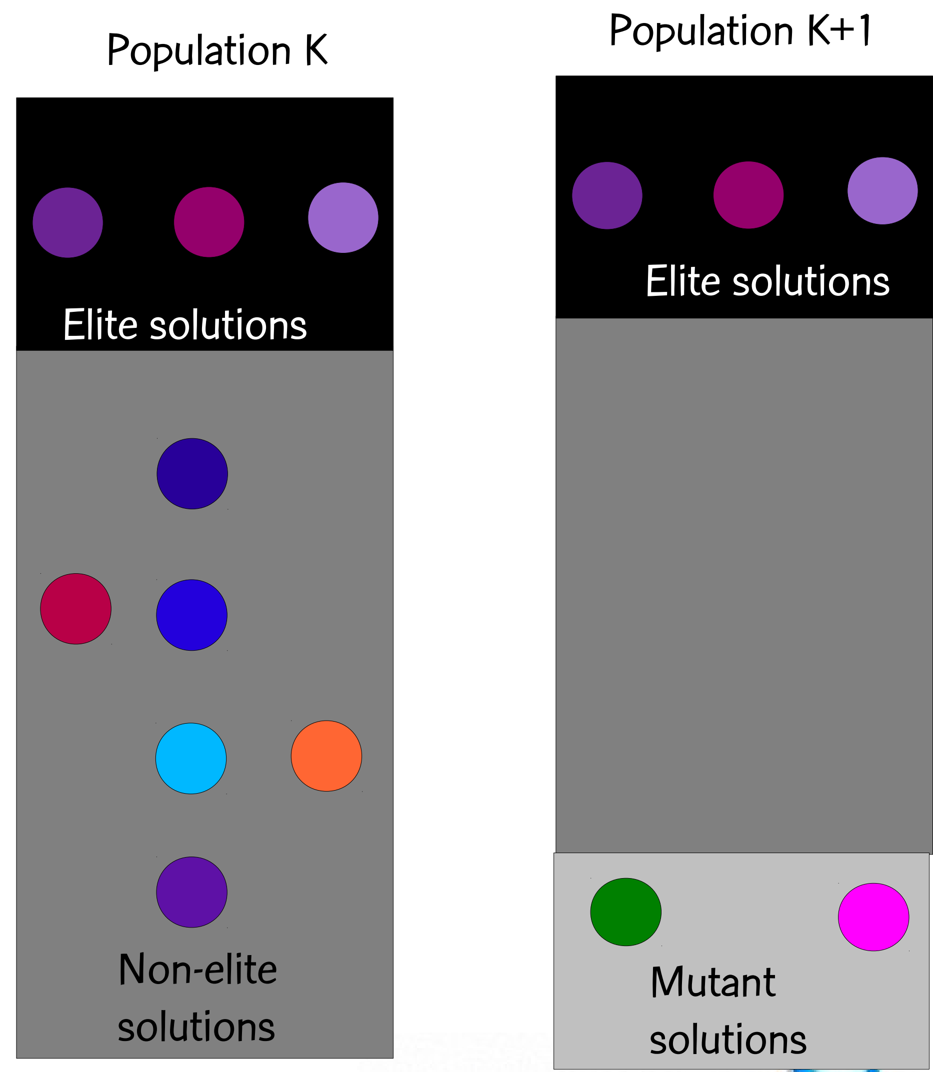
– Copy elite solutions from population K to population K+1



GAs and random keys

Evolutionary dynamics

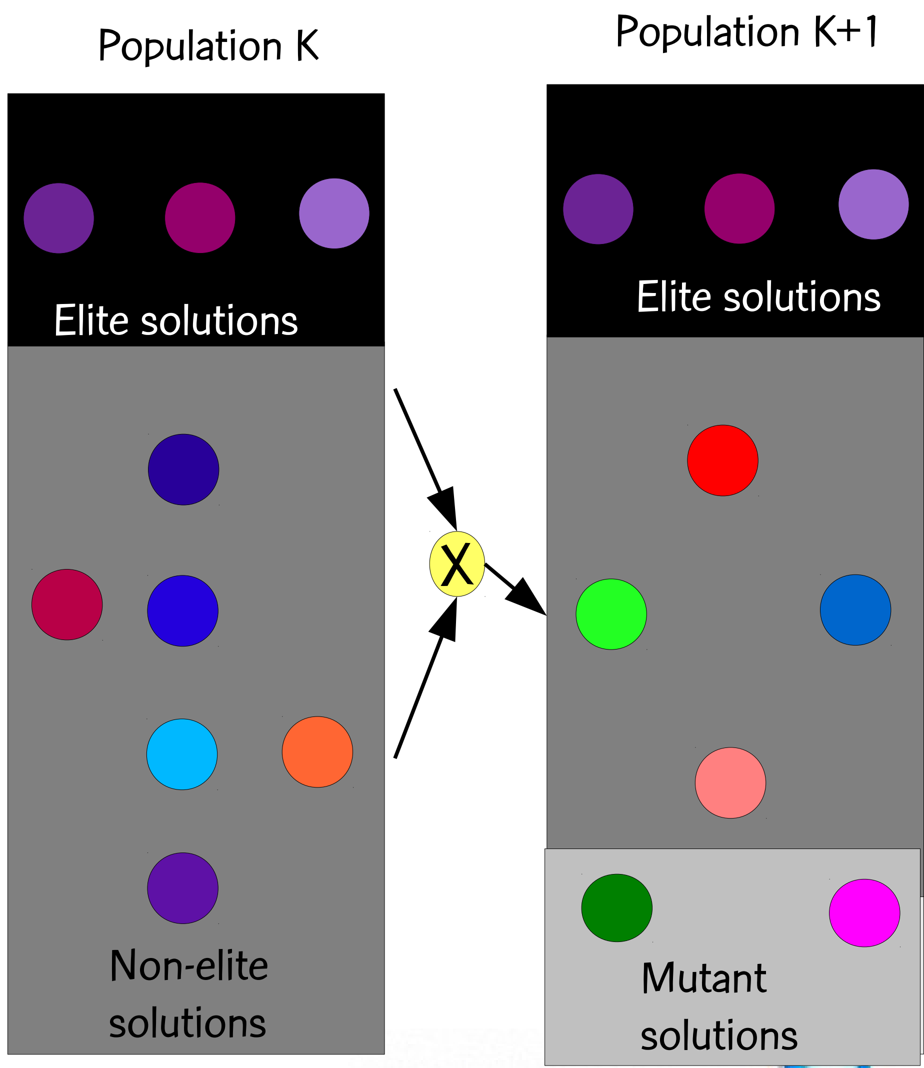
- Copy elite solutions from population K to population K+1
- Add R random solutions (mutants) to population K+1



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

both parents chosen at random from entire population

BRKGA

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

both parents chosen at random from entire population

either parent can be parent A in parametrized uniform crossover

BRKGA

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

How RKGA & BRKGA differ

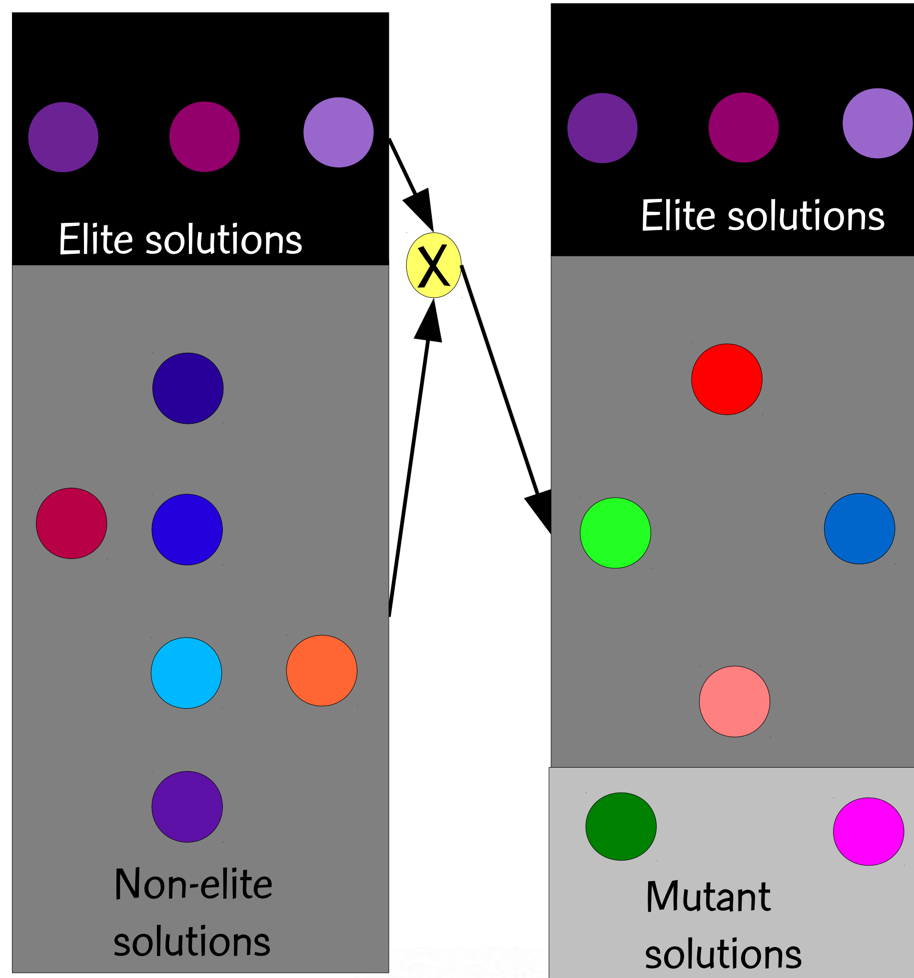
RKGA	BRKGA
both parents chosen at random from entire population	both parents chosen at random but one parent chosen from population of elite solutions
either parent can be parent A in parametrized uniform crossover	best fit parent is parent A in parametrized uniform crossover

Biased random key GA

BRKGA: Probability child inherits key of elite parent > 0.5

Population K

Population K+1

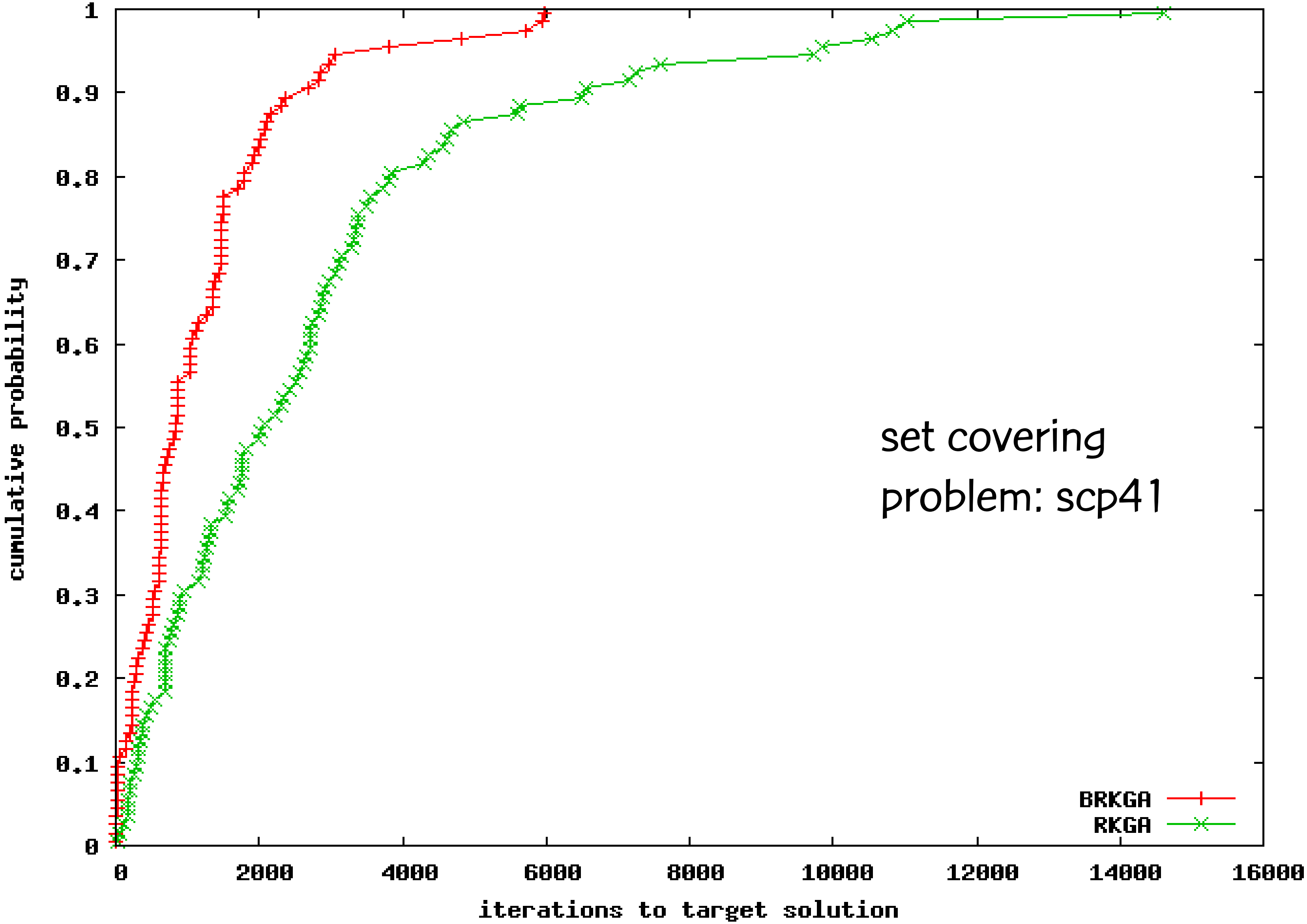


Evolutionary dynamics

- 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.

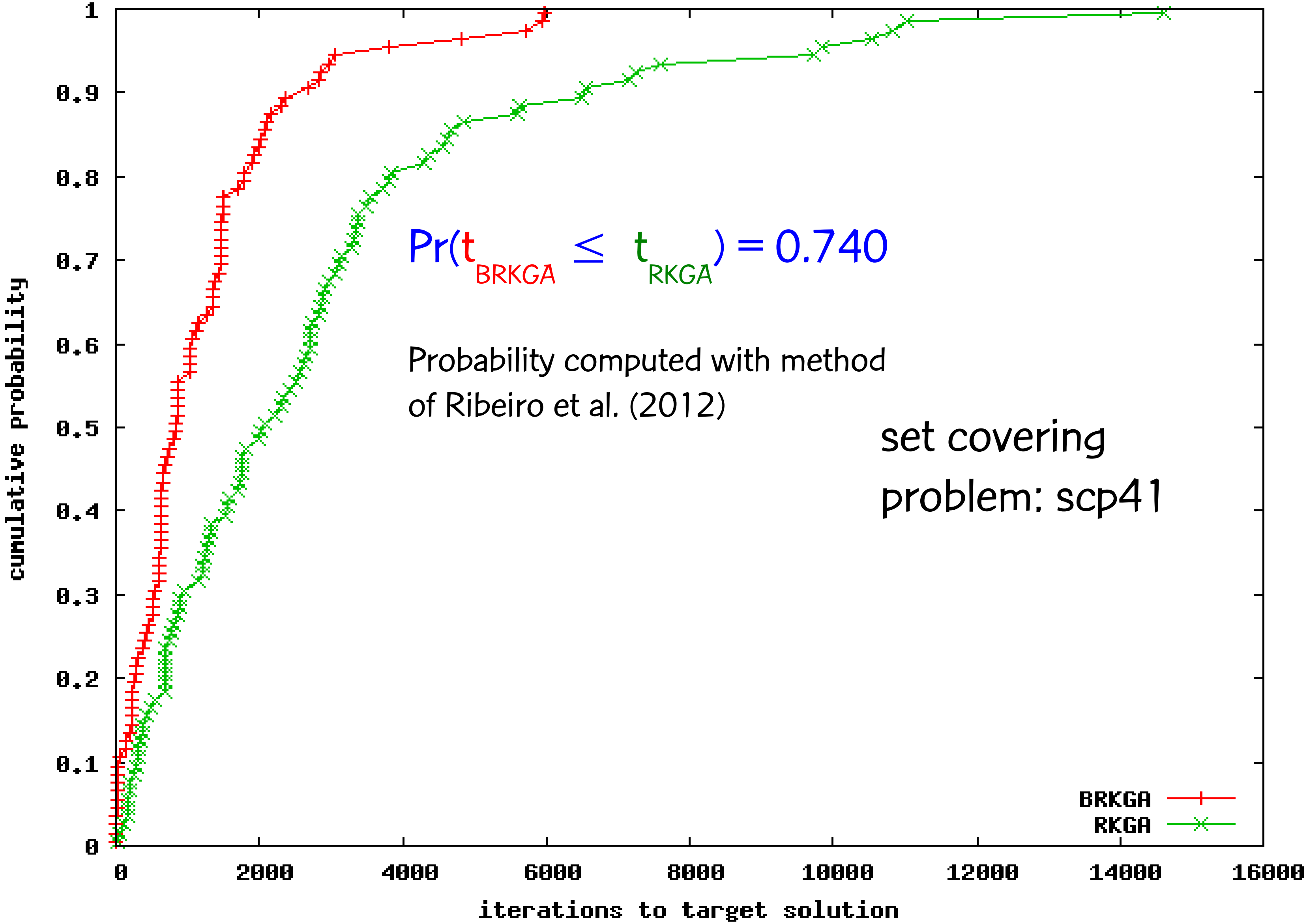
Paper comparing BRKGA and Bean's Method

Gonçalves, R., and Toso, "Biased and unbiased random-key genetic algorithms: An experimental analysis", Proceedings of the 10th Metaheuristics International Conference (MIC 2013), Singapore, August 2013.



Rethink Possible

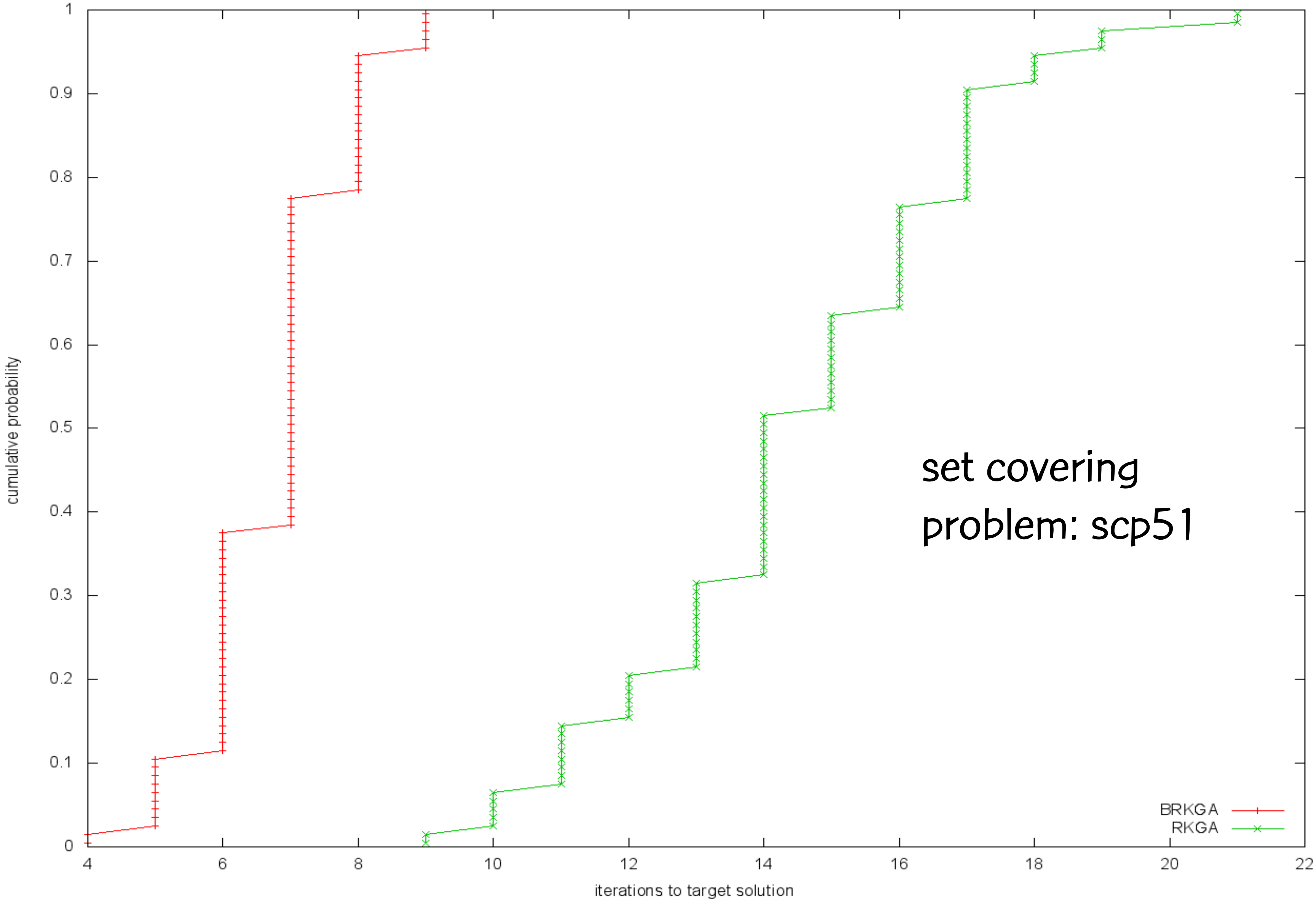
att.com/rethinkpossible



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

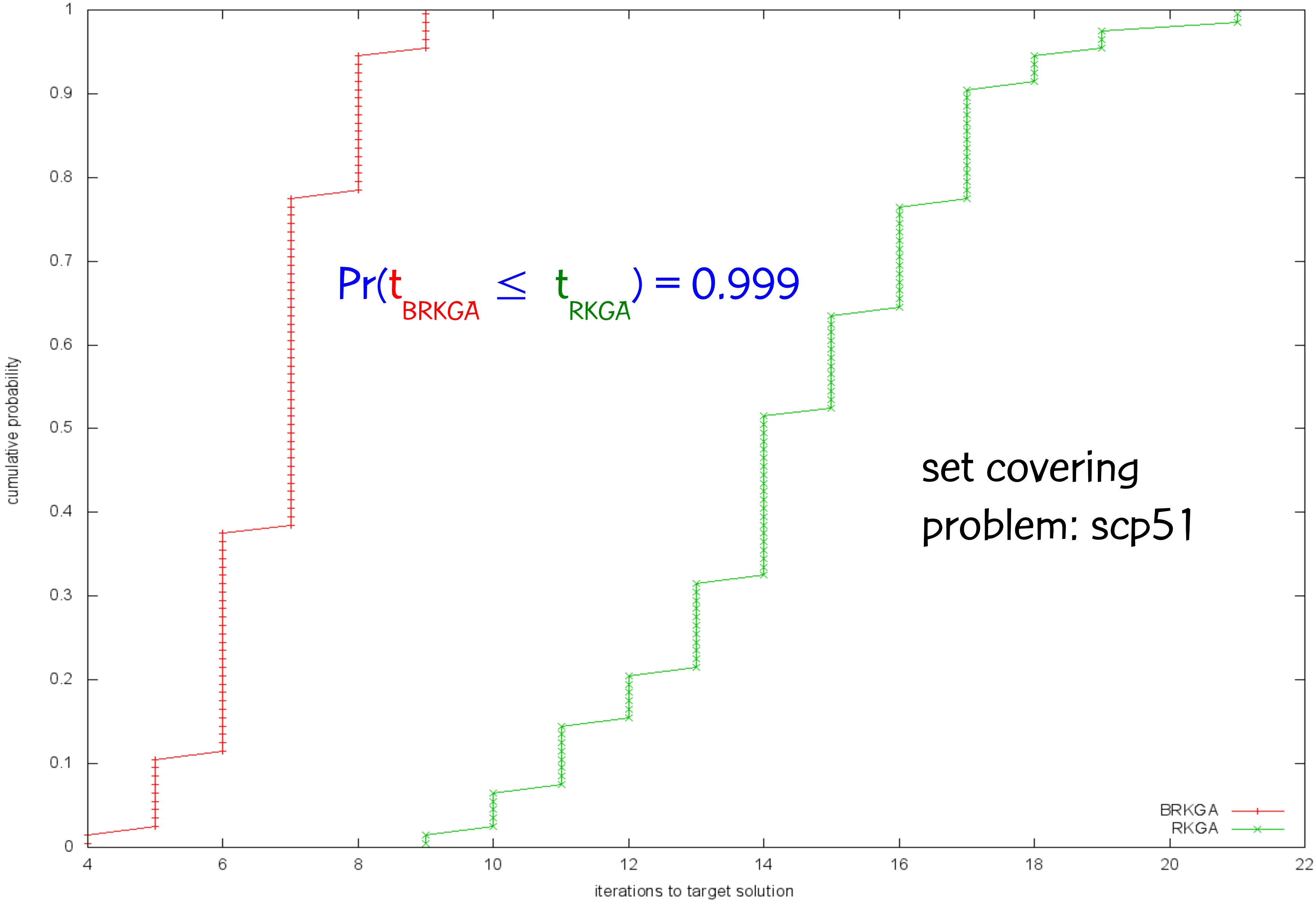
Probability computed with method of Ribeiro et al. (2012)

set covering problem: scp41

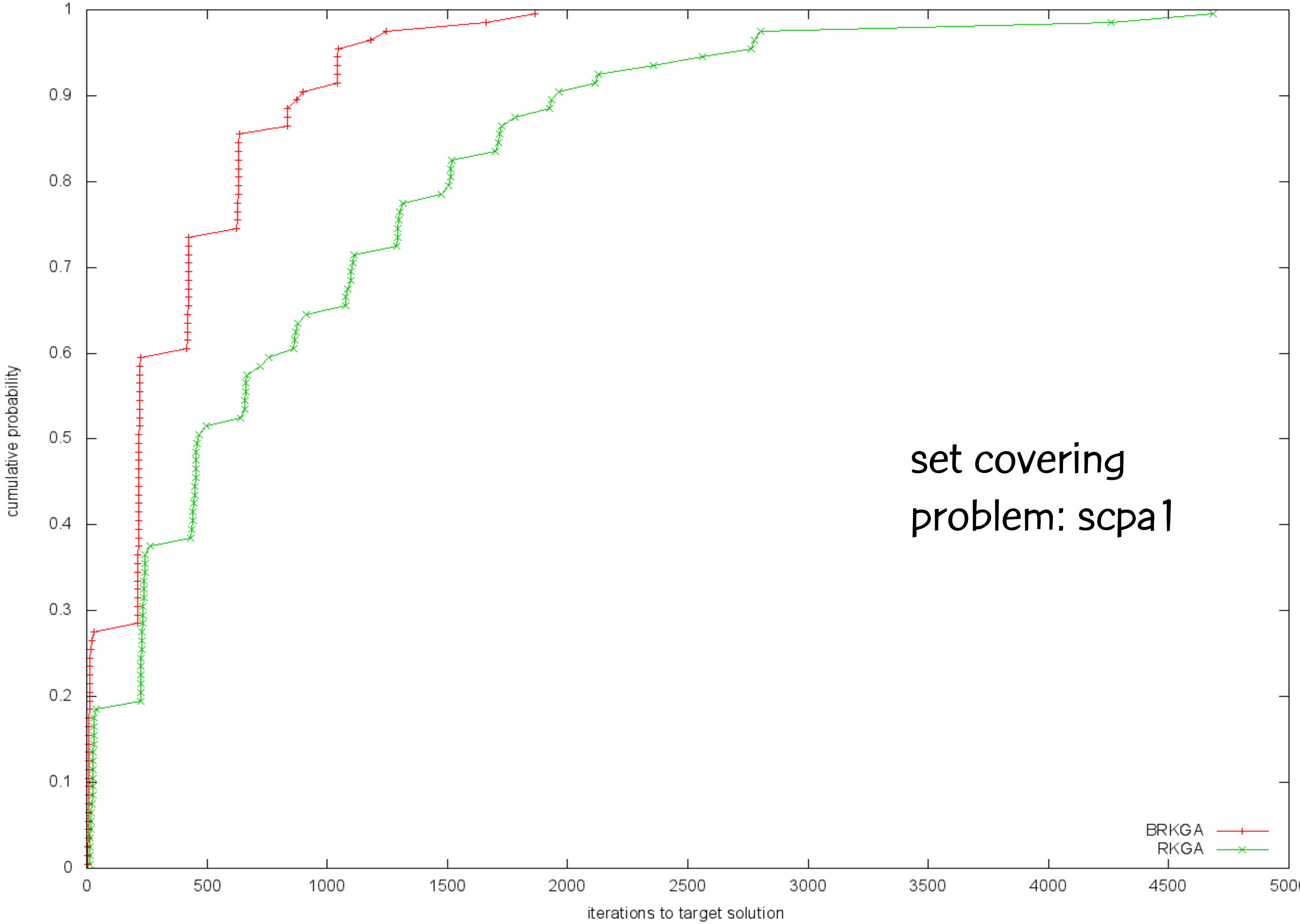


Rethink Possible

att.com/rethinkpossible

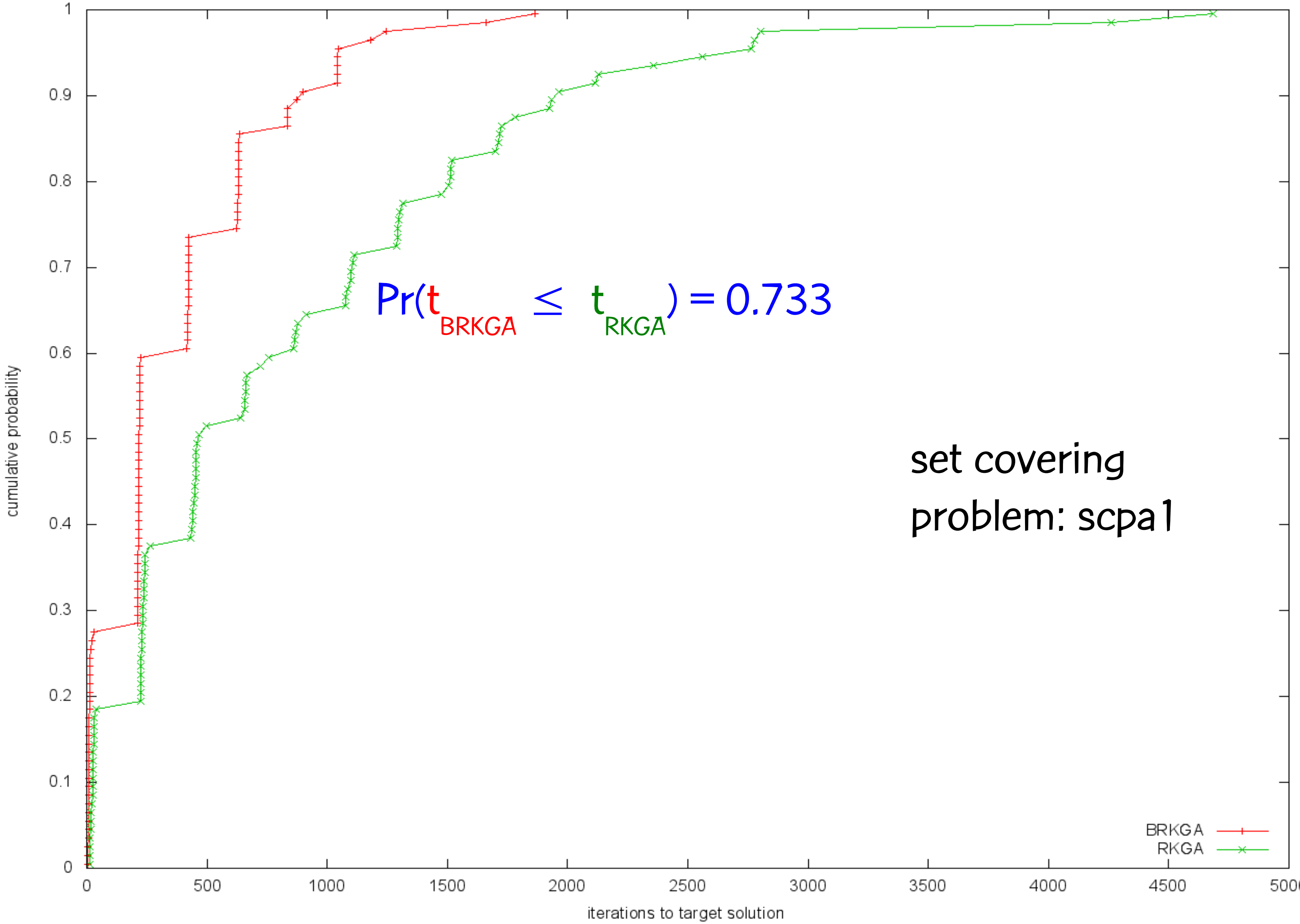


$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.999$
set covering
problem: scp51
cumulative probability
iterations to target solution
BRKGA
RKGA
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
4
6
8
10
12
14
16
18
20
22



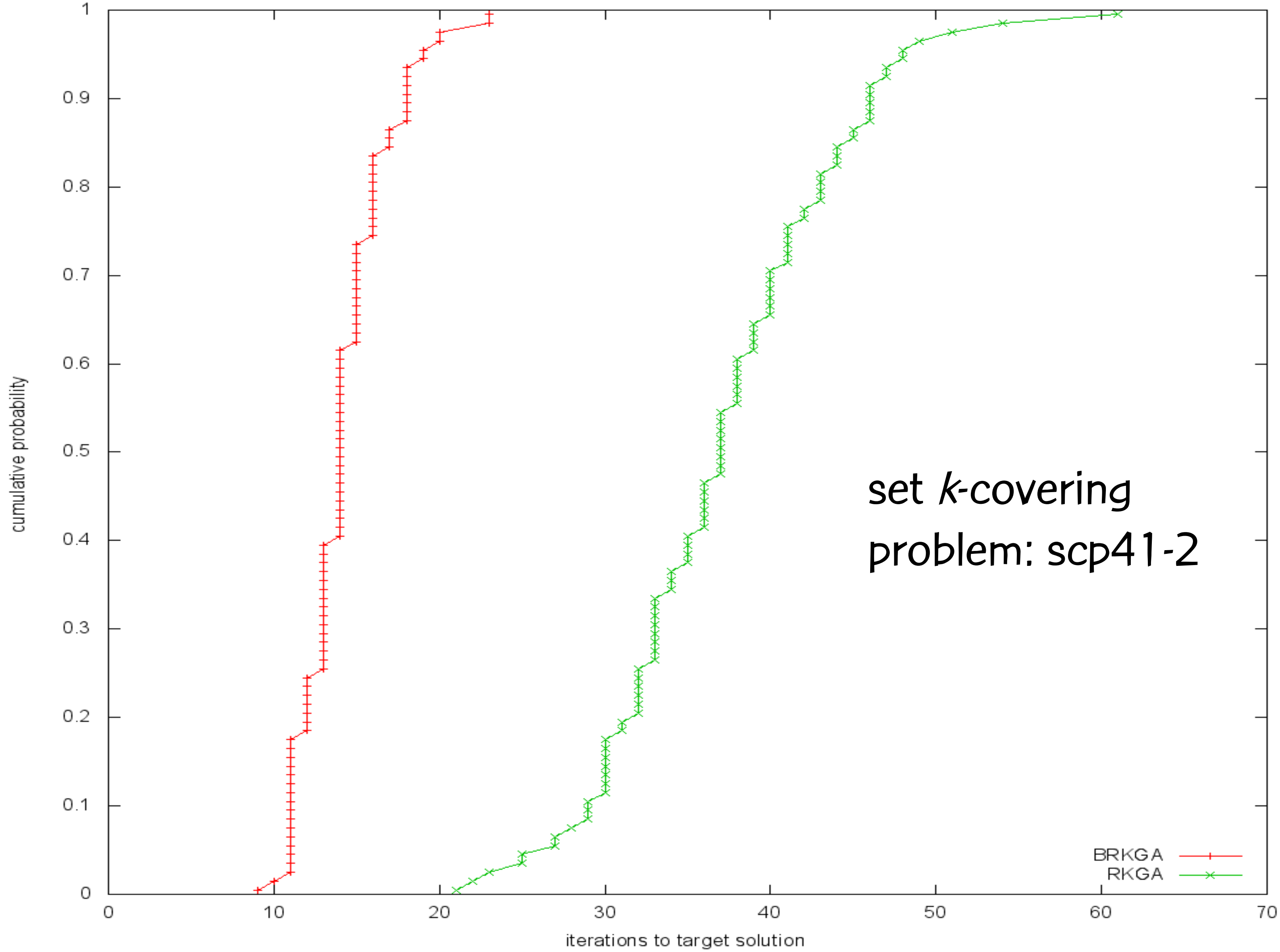
cumulative probability
iterations to target solution
BRKGA
RKGA
set covering problem: scpa1

BRKGA



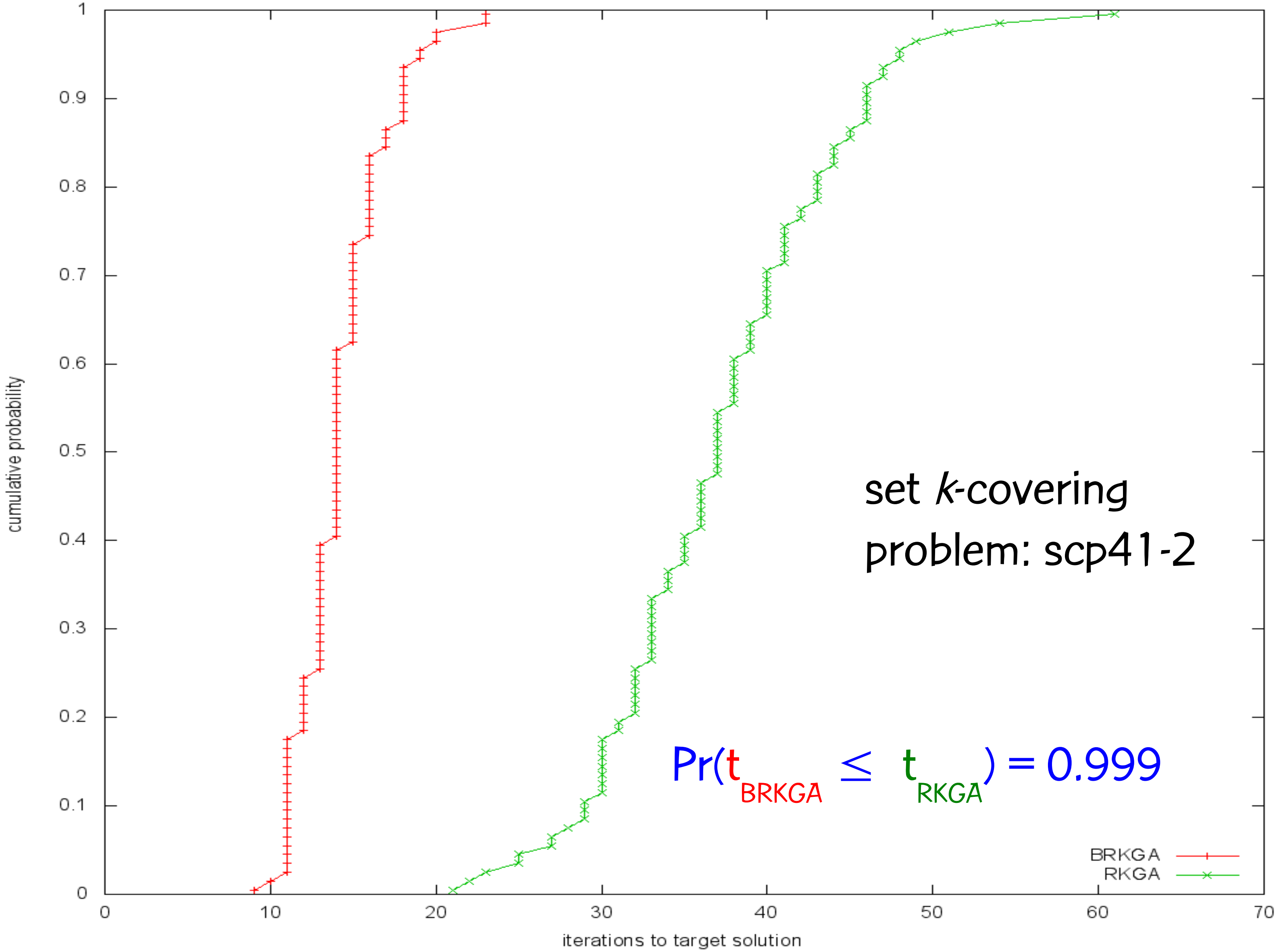
Rethink Possible

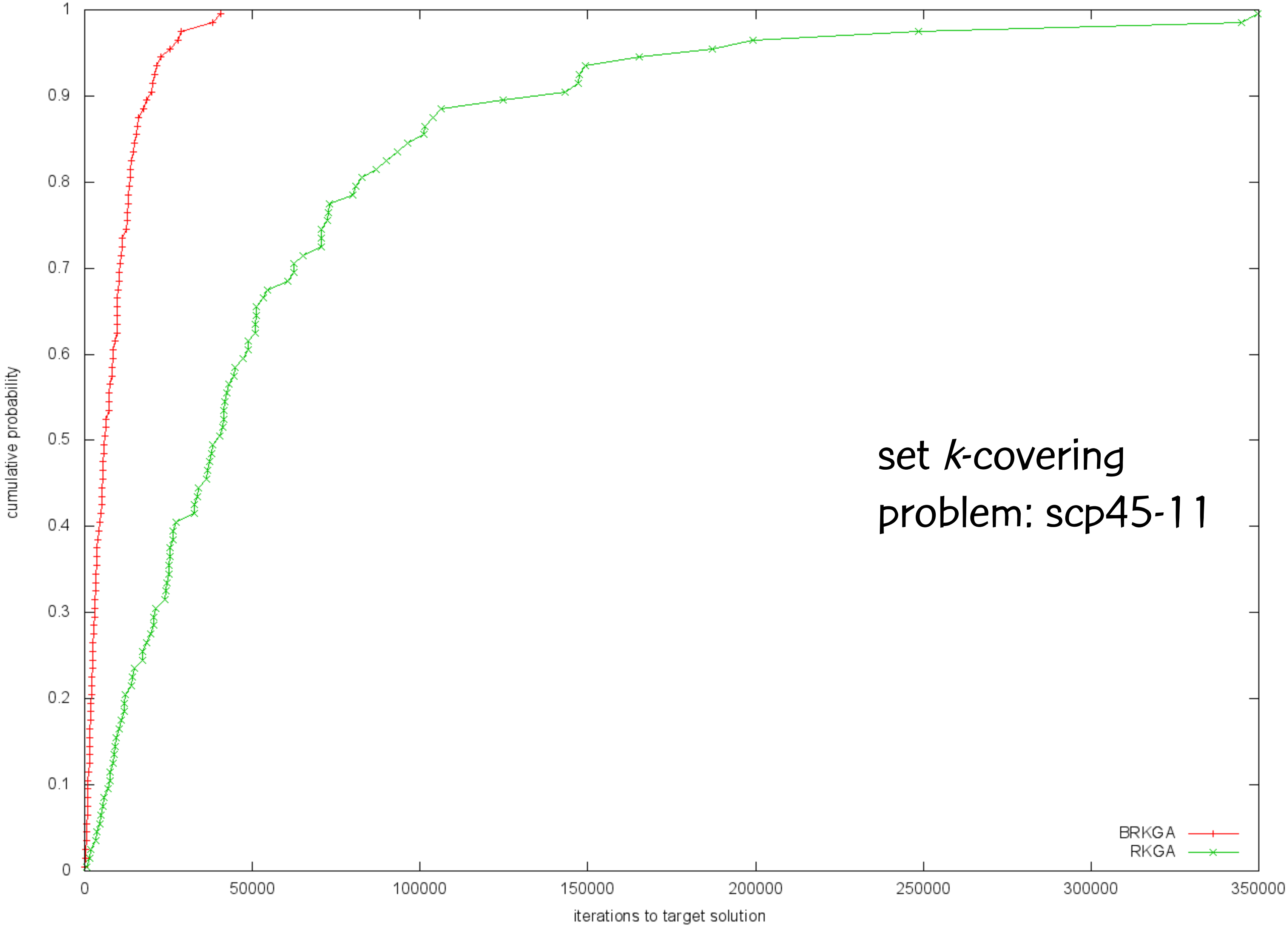
att.com/rethinkpossible



Rethink Possible

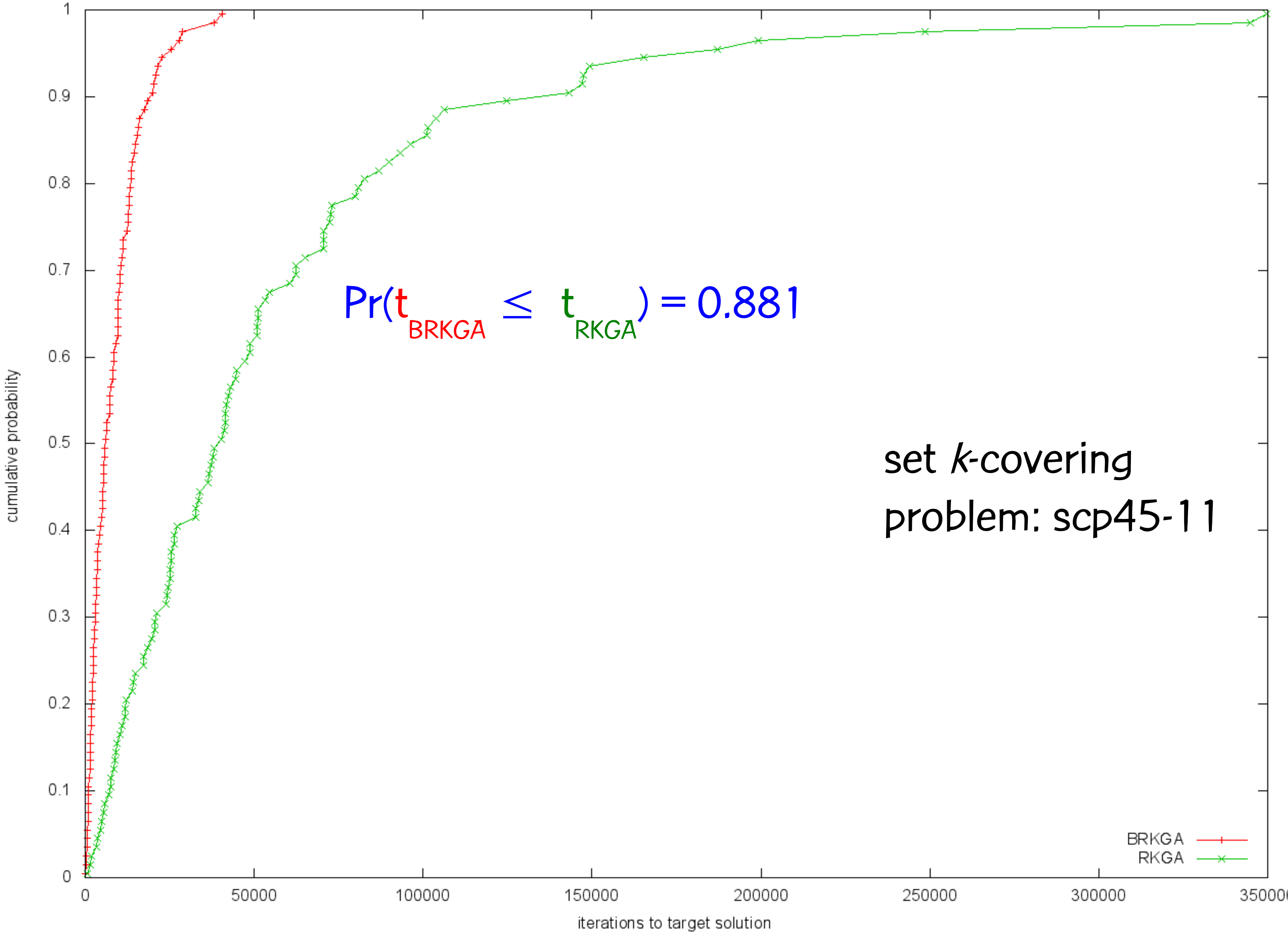
att.com/rethinkpossible


set k-covering problem: scp41-2
Pr(t_BRKGA ≤ t_RKGA) = 0.999
cumulative probability
iterations to target solution
BRKGA
RKGA
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



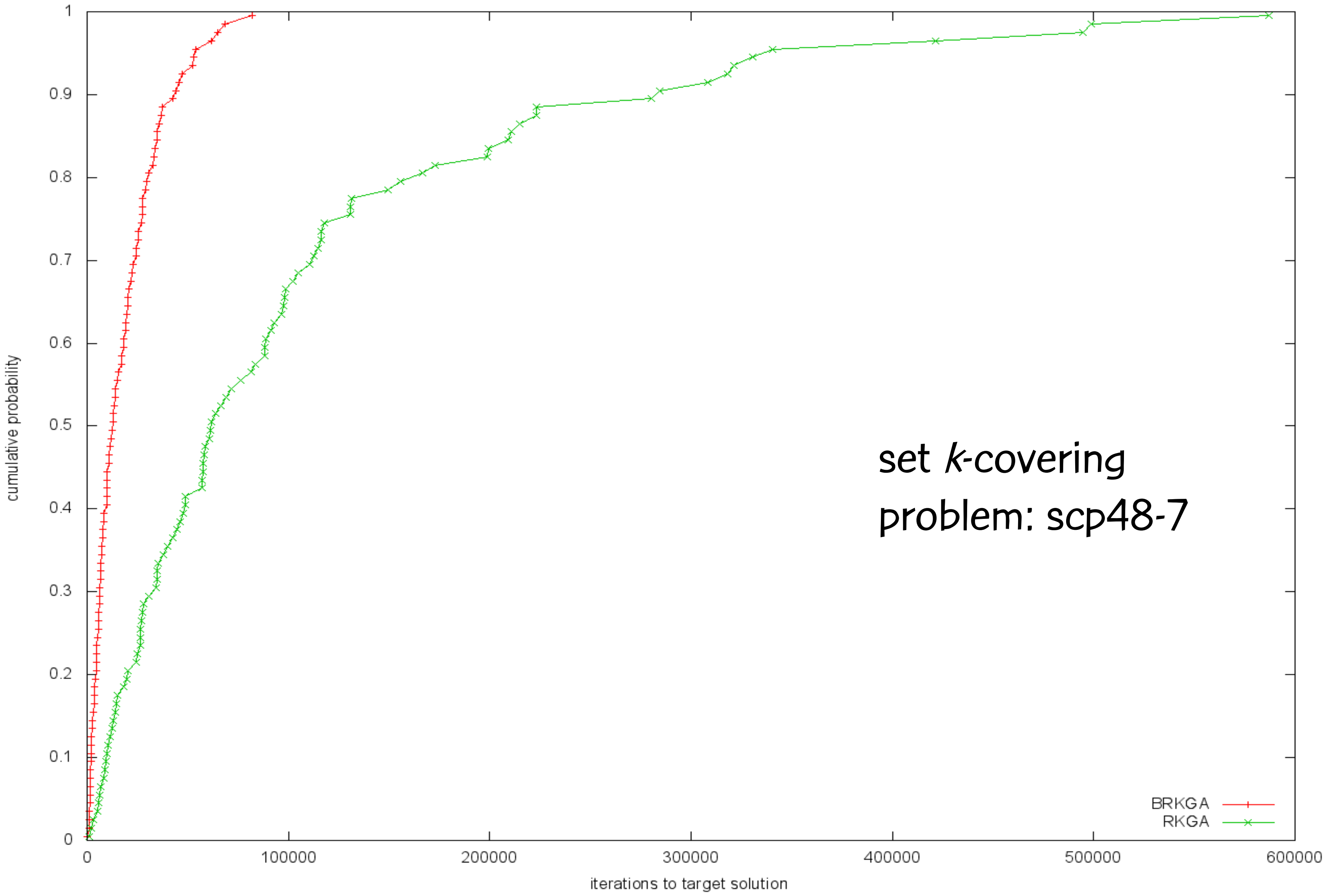
Rethink Possible

att.com/rethinkpossible

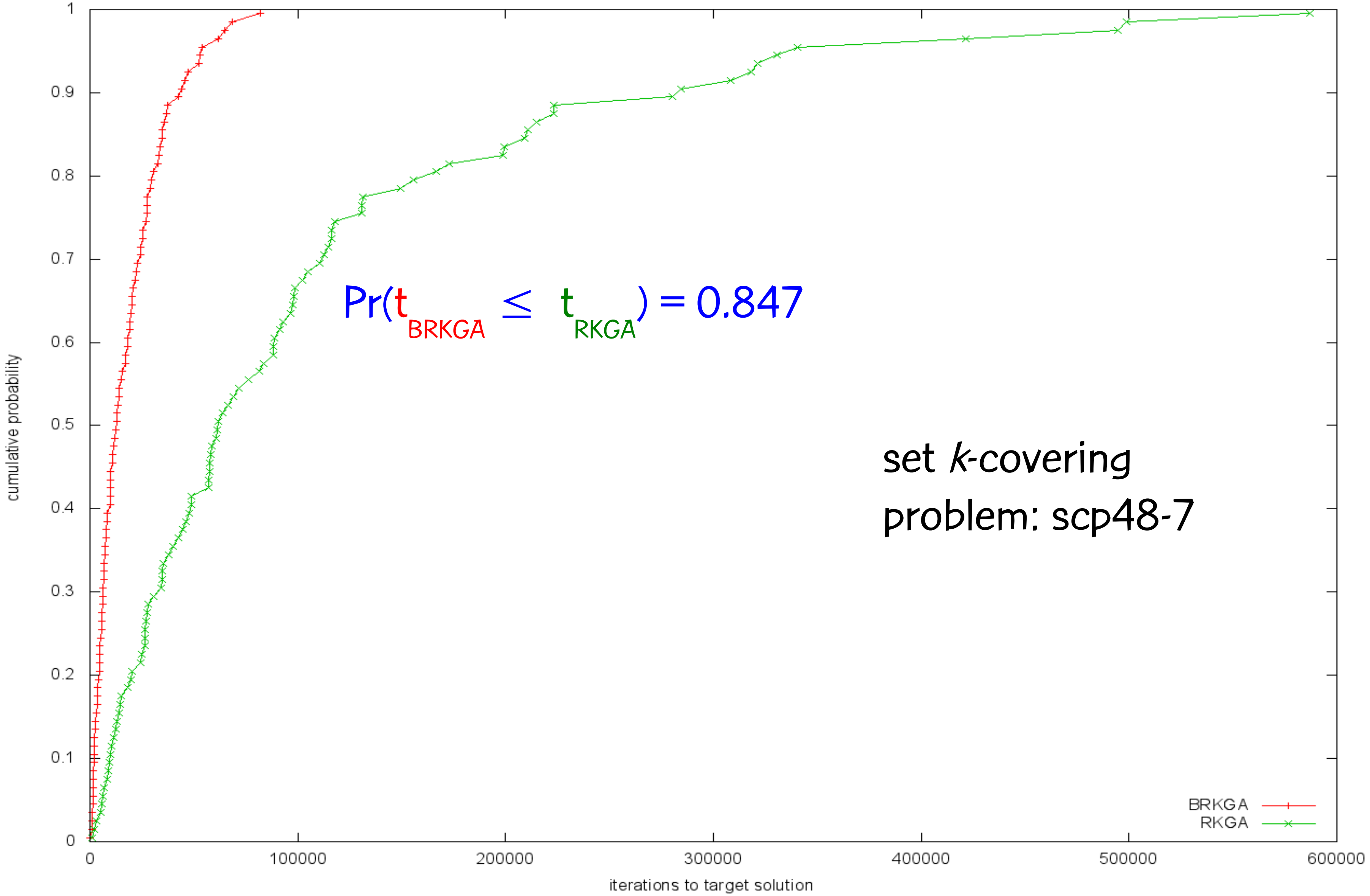


$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.881$

set *k*-covering problem: scp45-11



set *k*-covering problem: scp48-7



set *k*-covering problem: scp48-7

Rethink Possible

att.com/rethinkpossible

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)

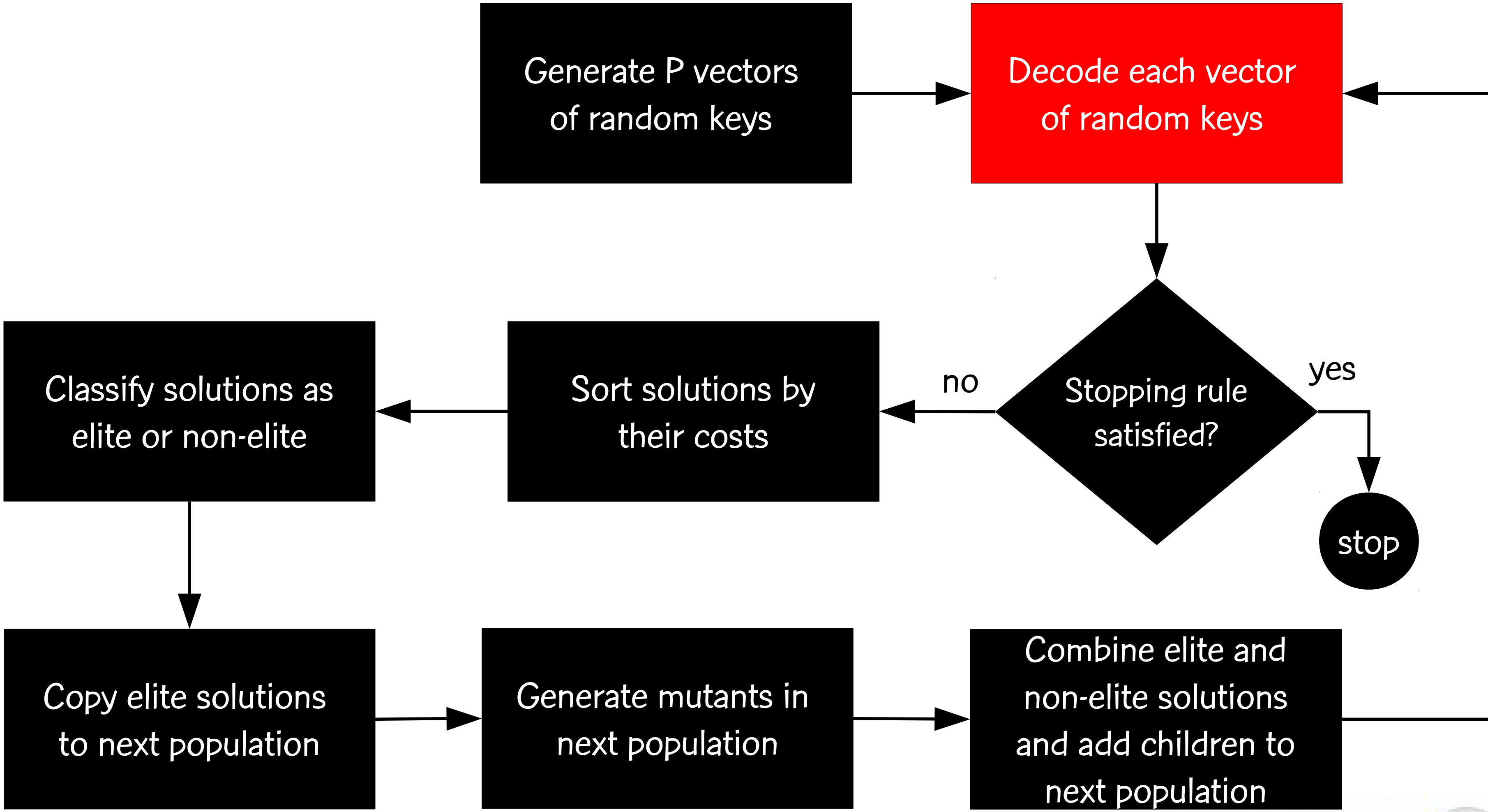
Observations

- 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.

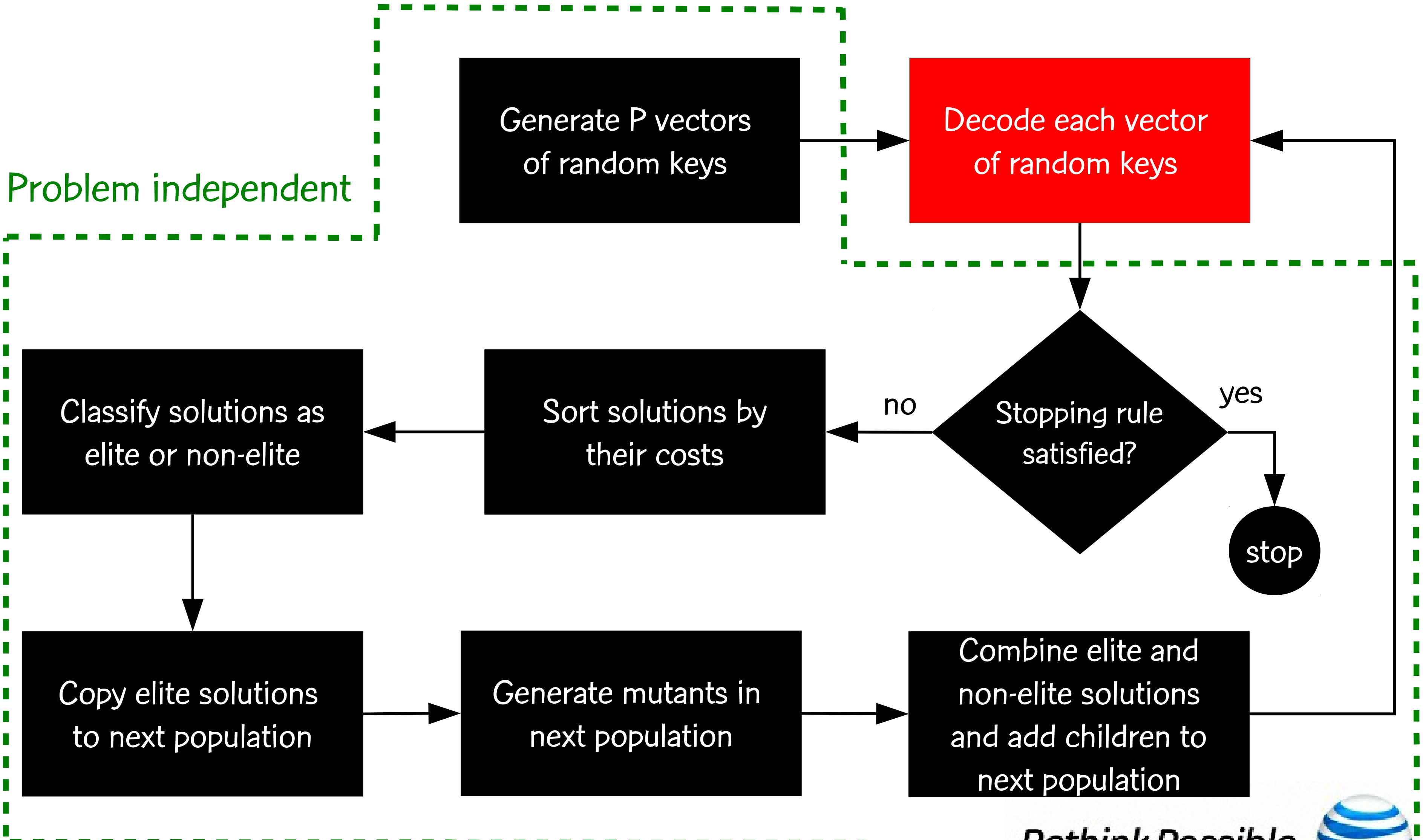
Observations

- 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)

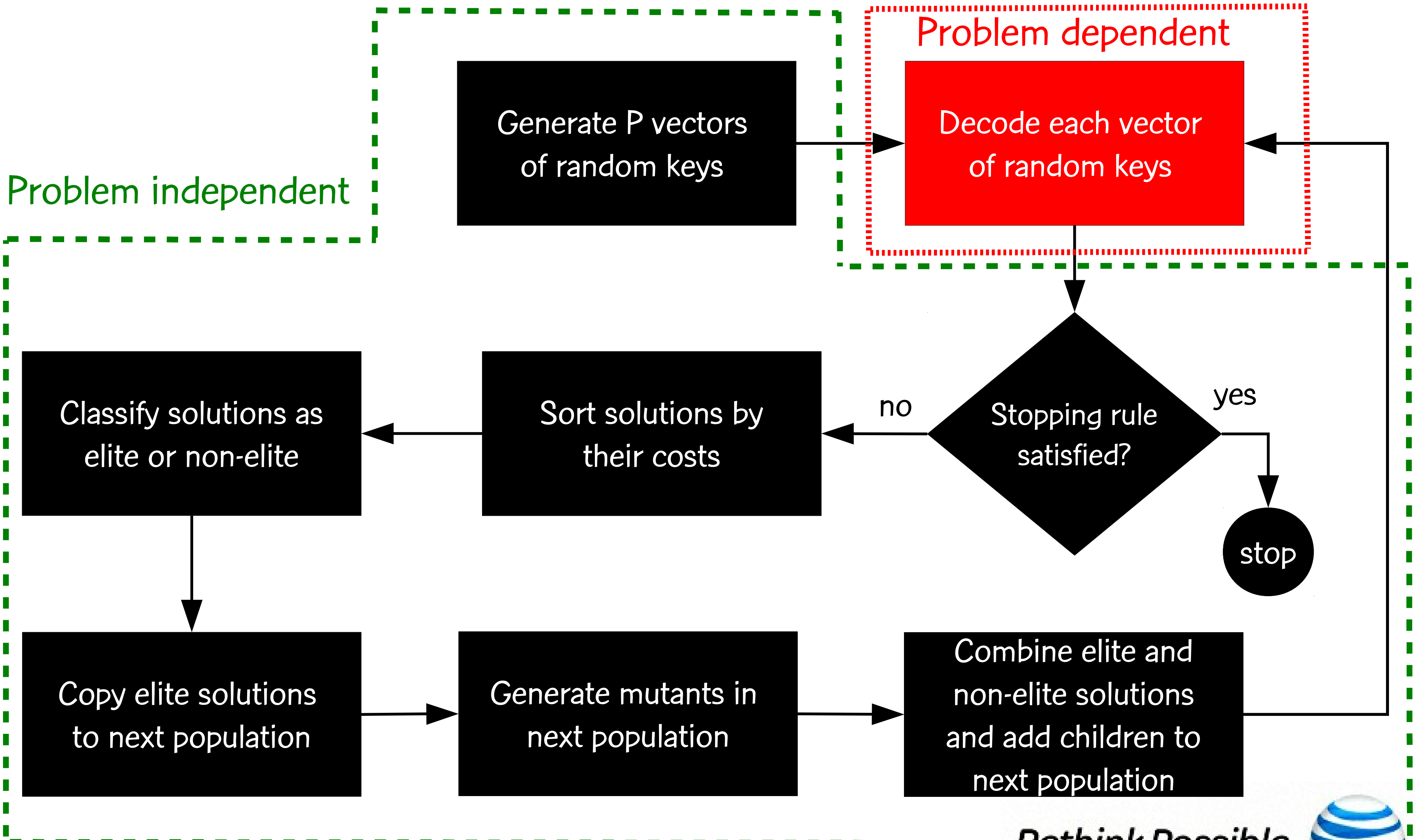
Framework for biased random-key genetic algorithms



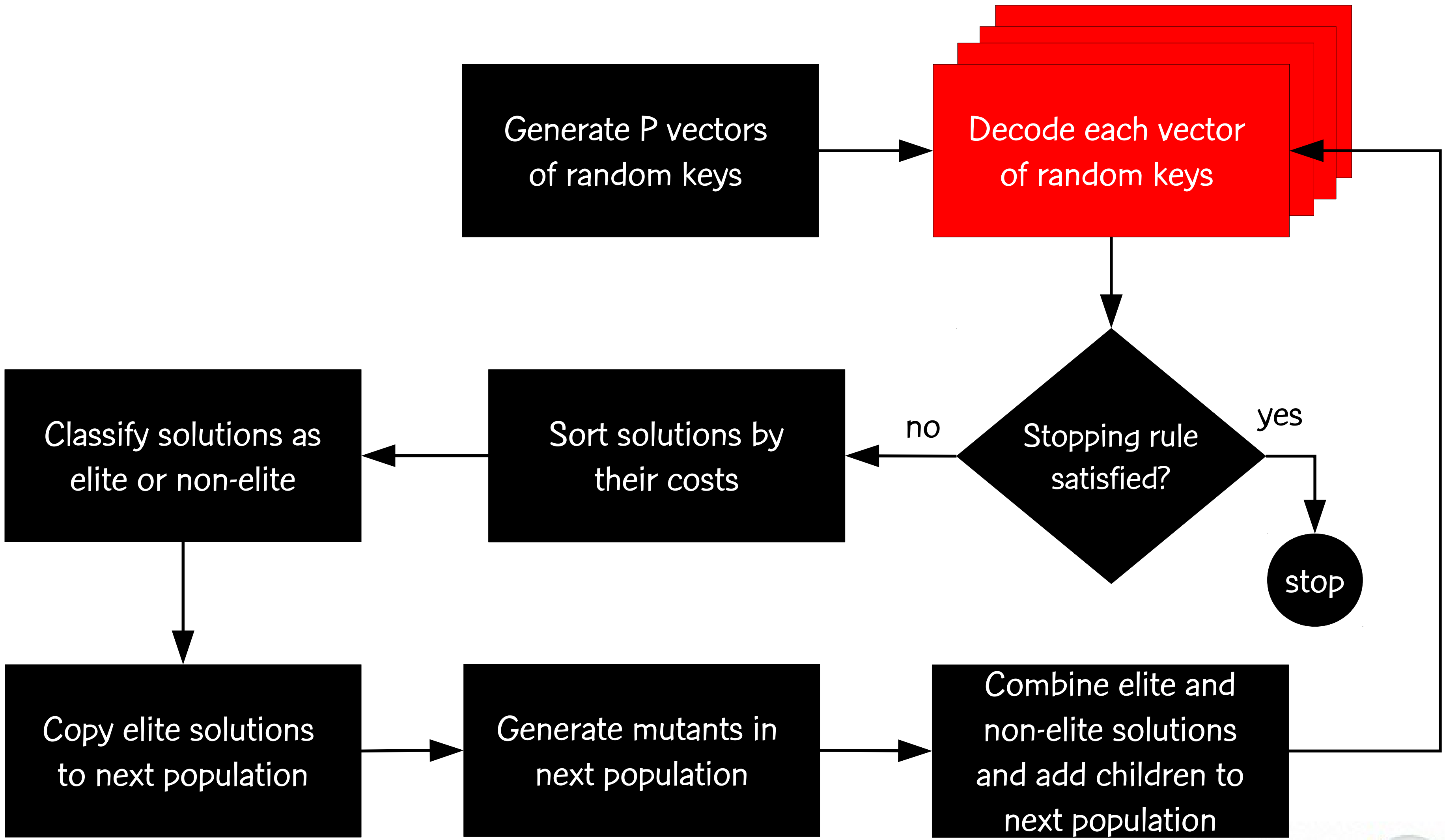
Framework for biased random-key genetic algorithms



Framework for biased random-key genetic algorithms



Decoding of random key vectors can be done in parallel

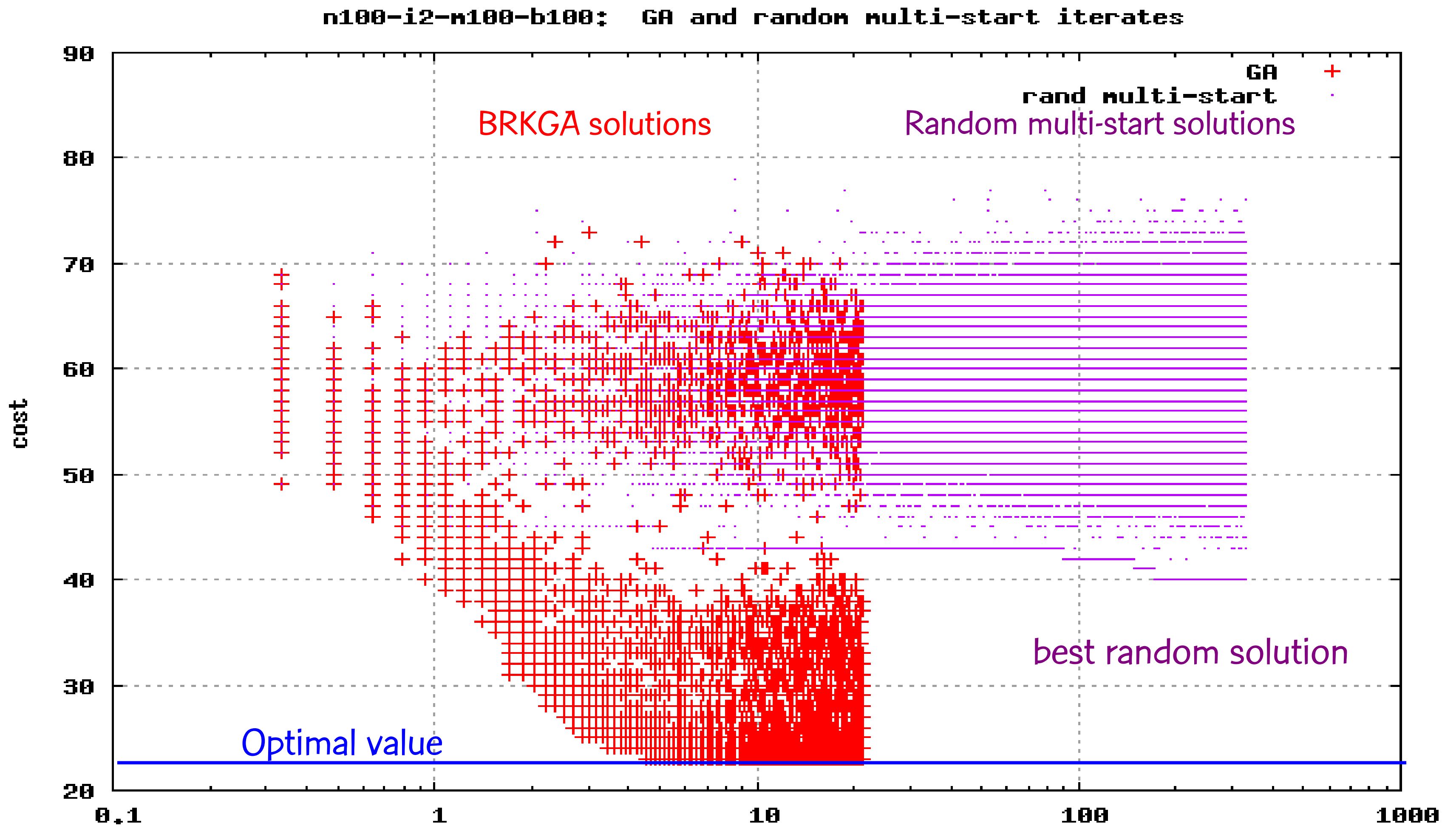


Is a BRKGA any different from applying the decoder to random keys?

- 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

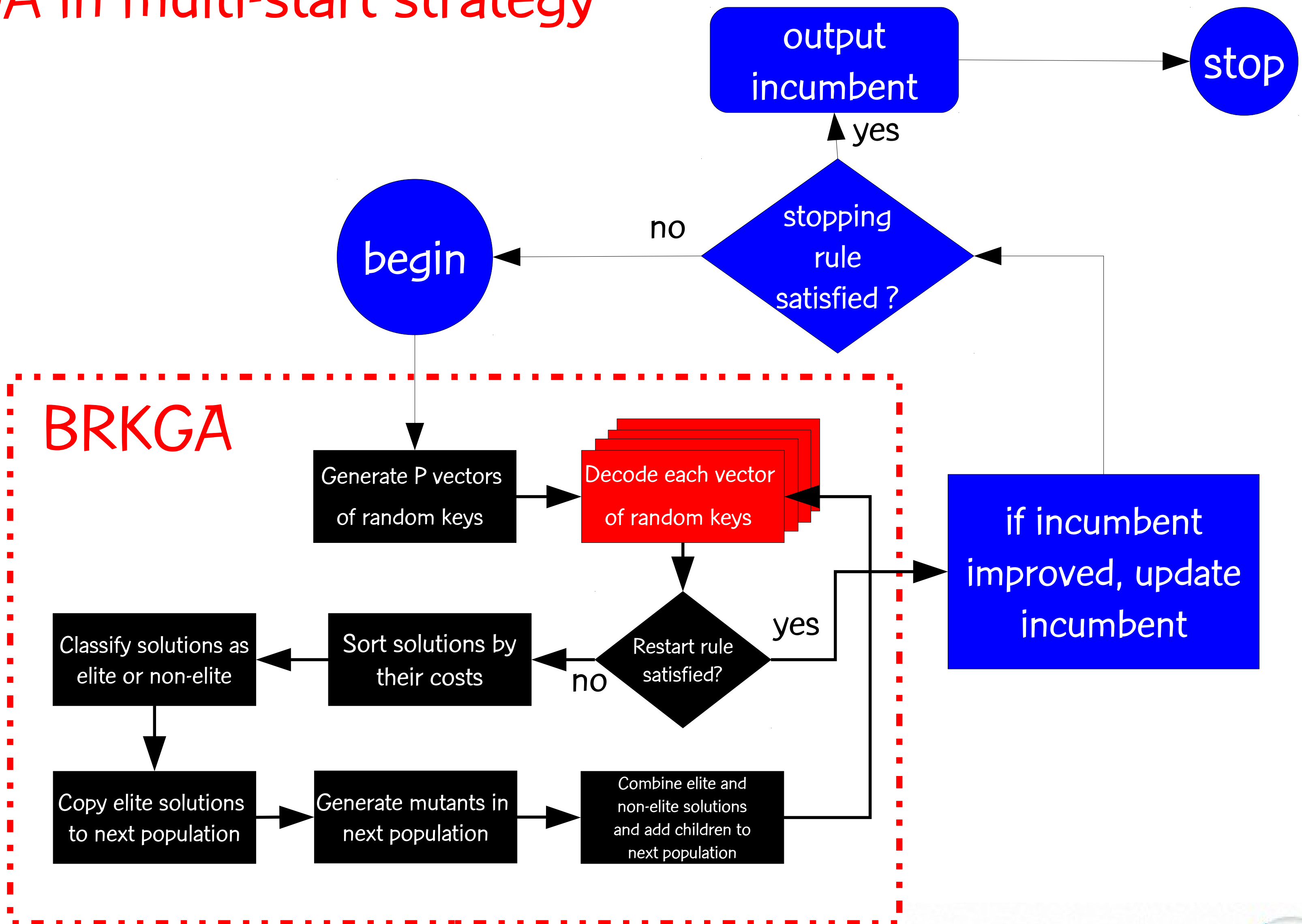


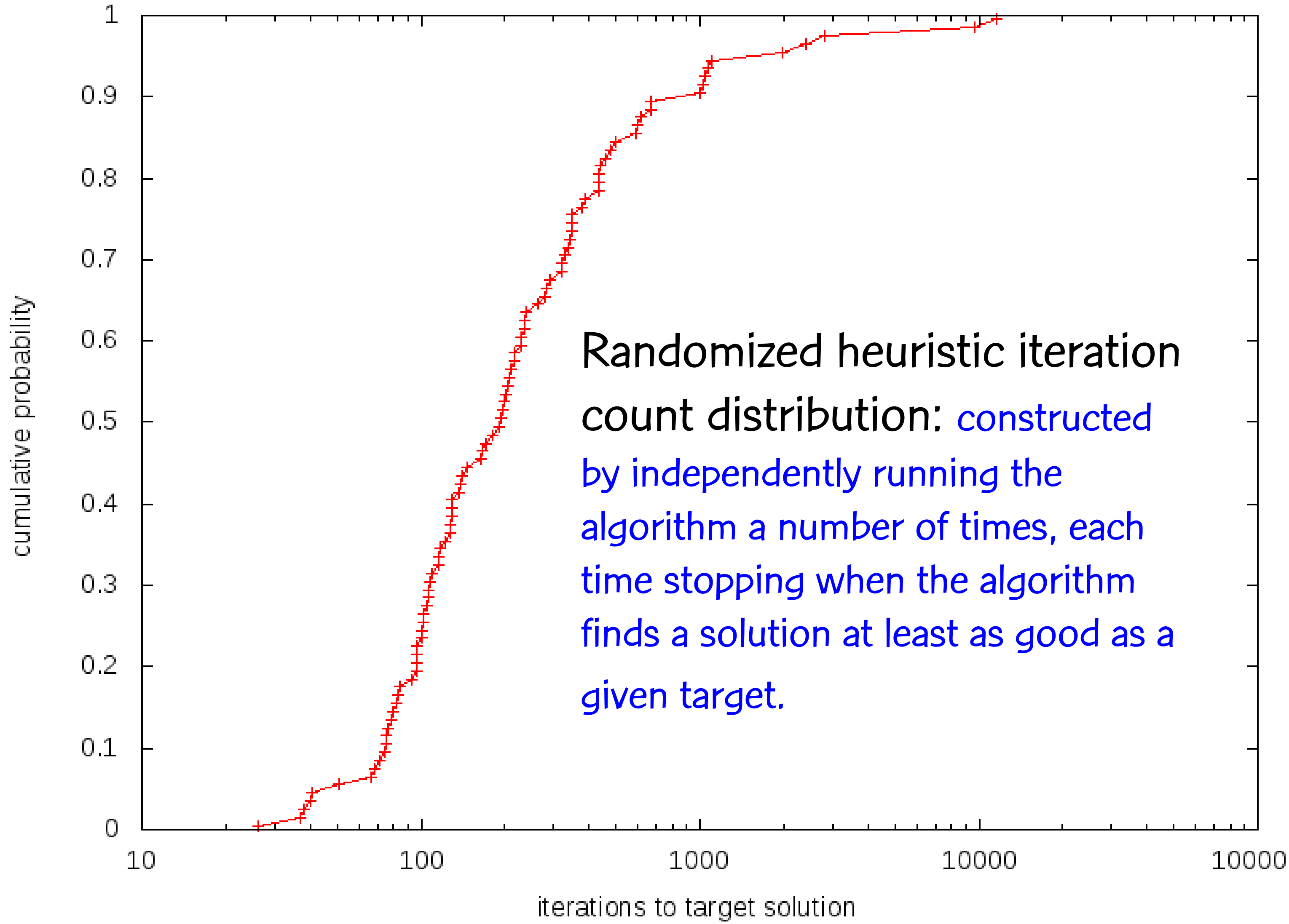
Time (ibm t41 secs)

Rethink Possible

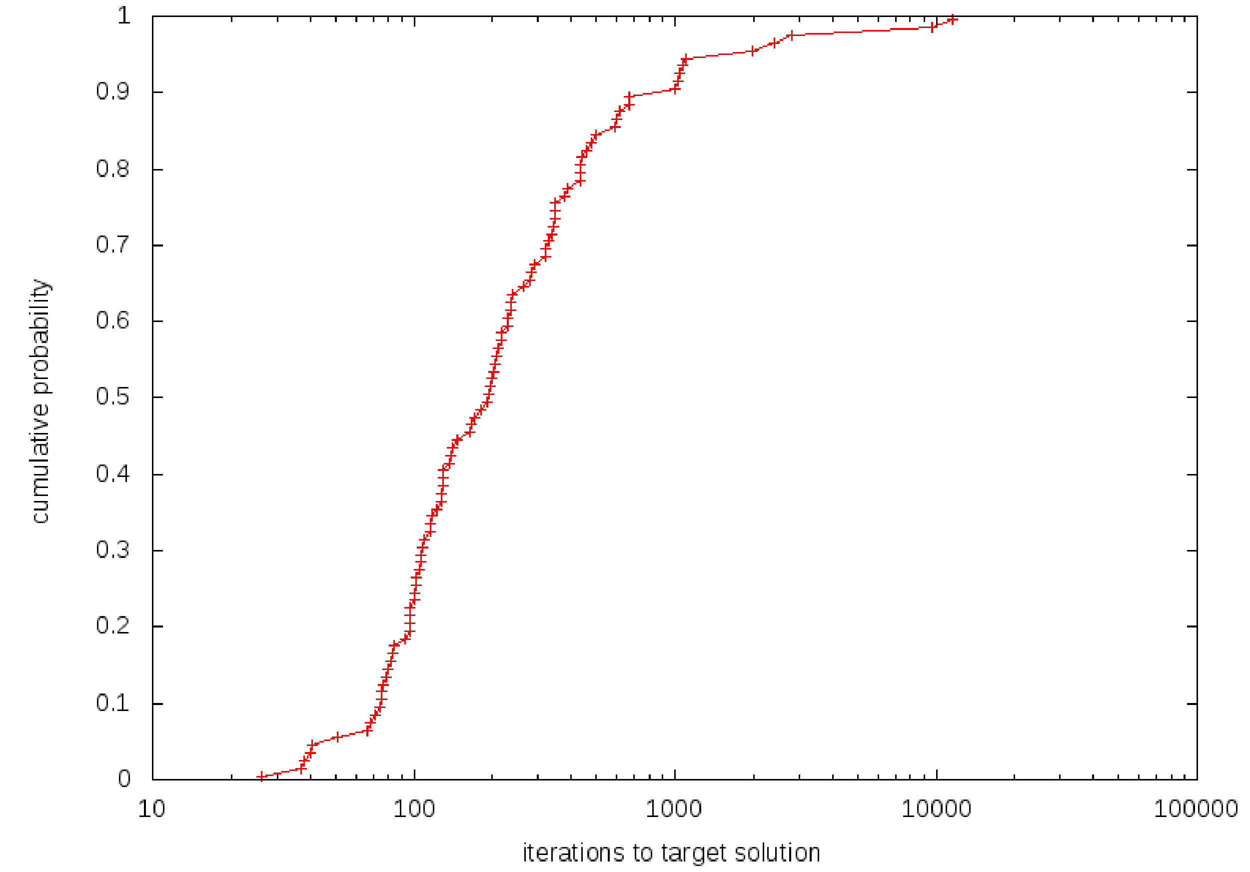
att.com/rethinkpossible

BRKGA in multi-start strategy

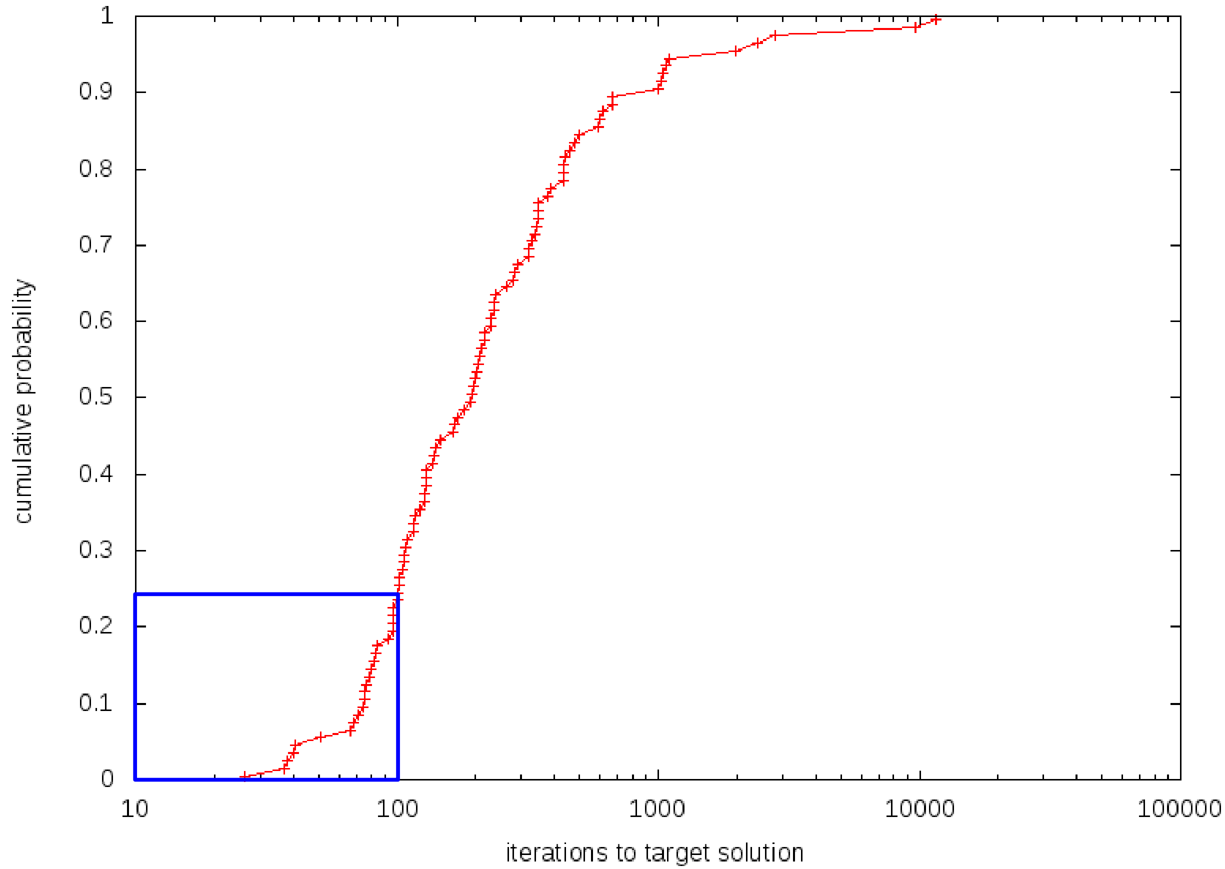




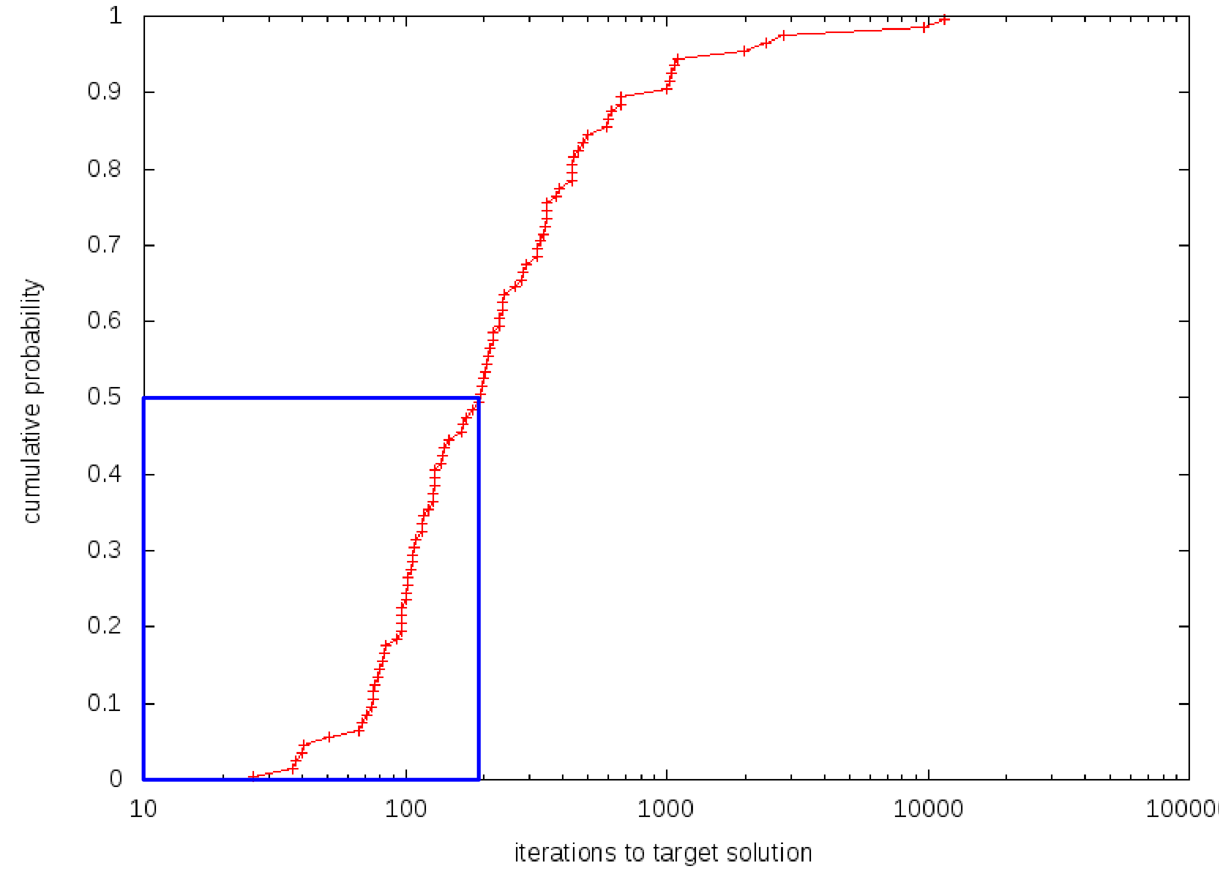
Randomized heuristic iteration count distribution: constructed by independently running the algorithm a number of times, each time stopping when the algorithm finds a solution at least as good as a given target.



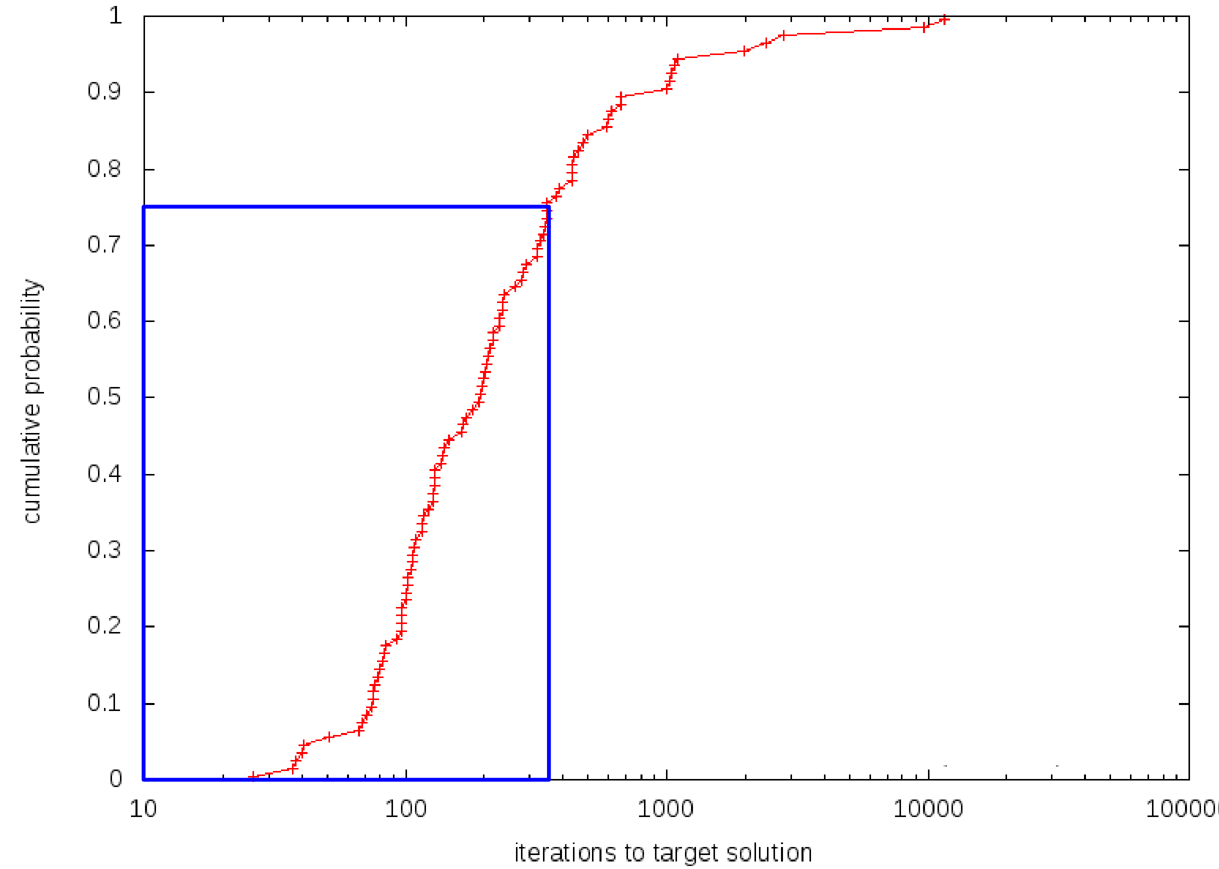
In most of the independent runs, the algorithm finds the target solution in relatively few iterations:



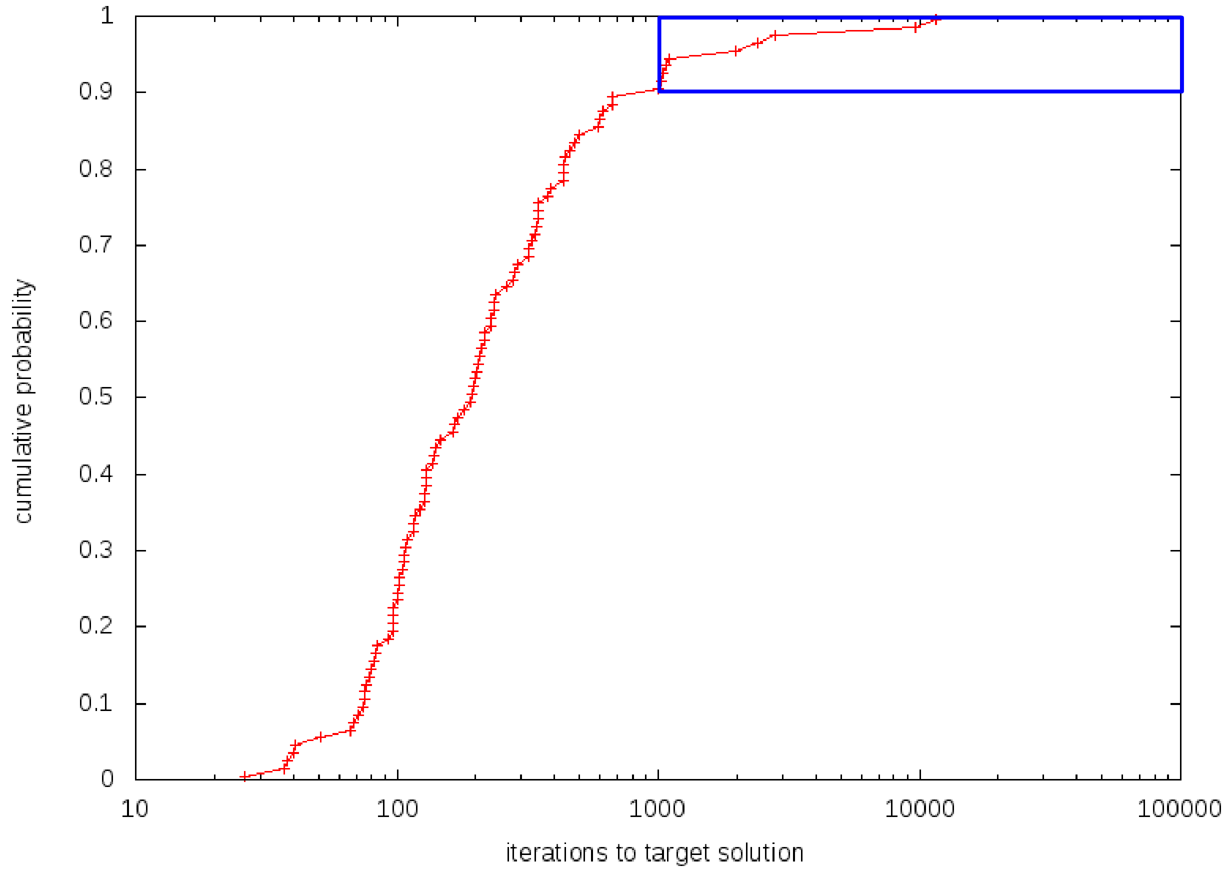
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 25% of the runs take fewer than 101 iterations



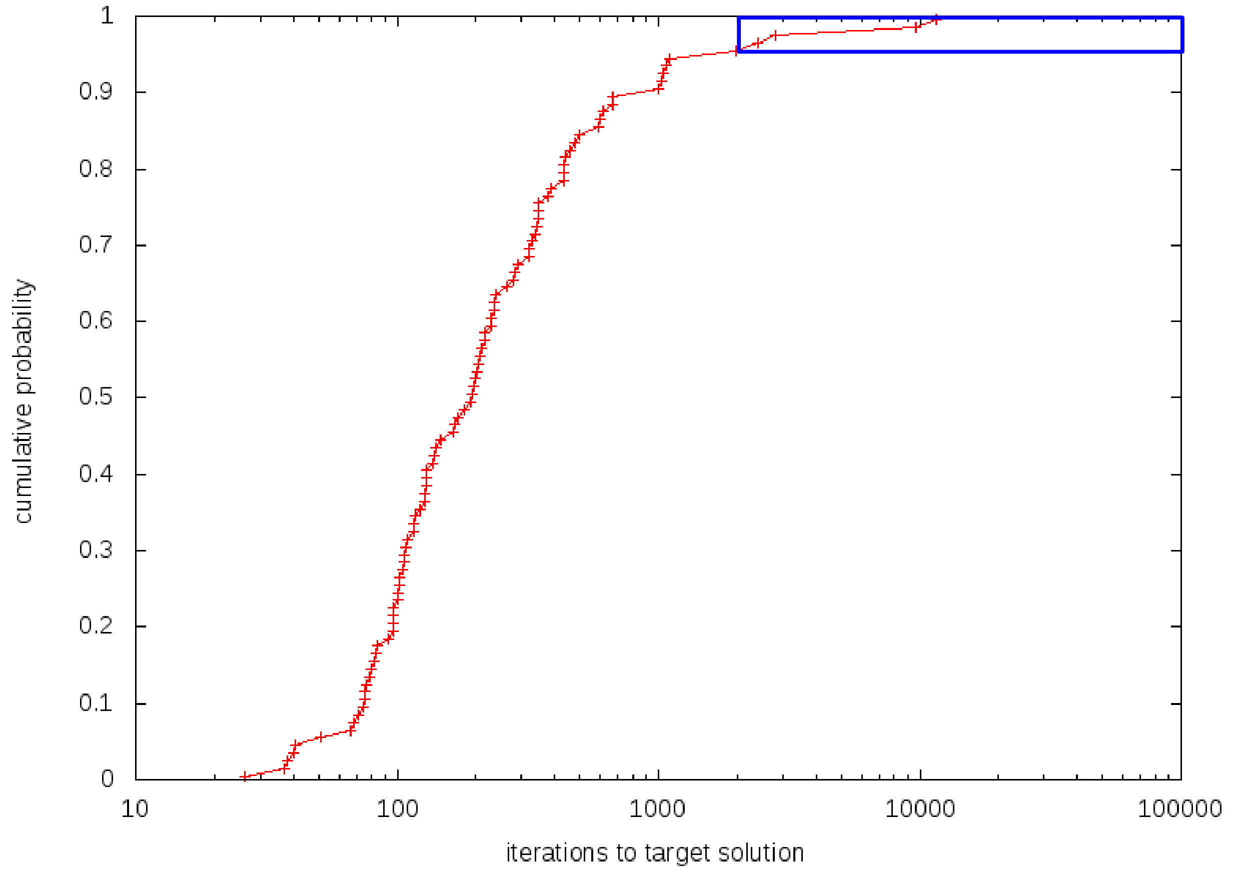
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 50% of the runs take fewer than 192 iterations



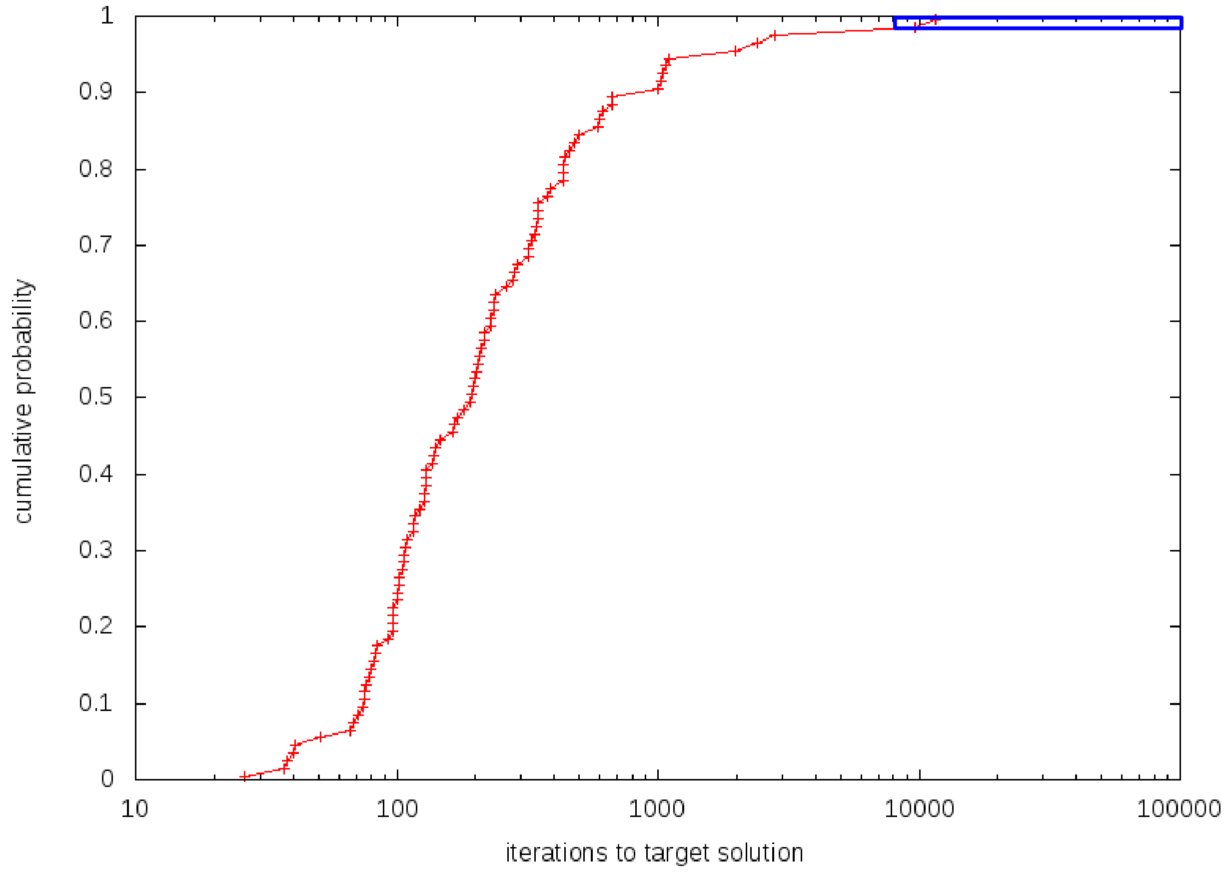
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 75% of the runs take fewer than 345 iterations



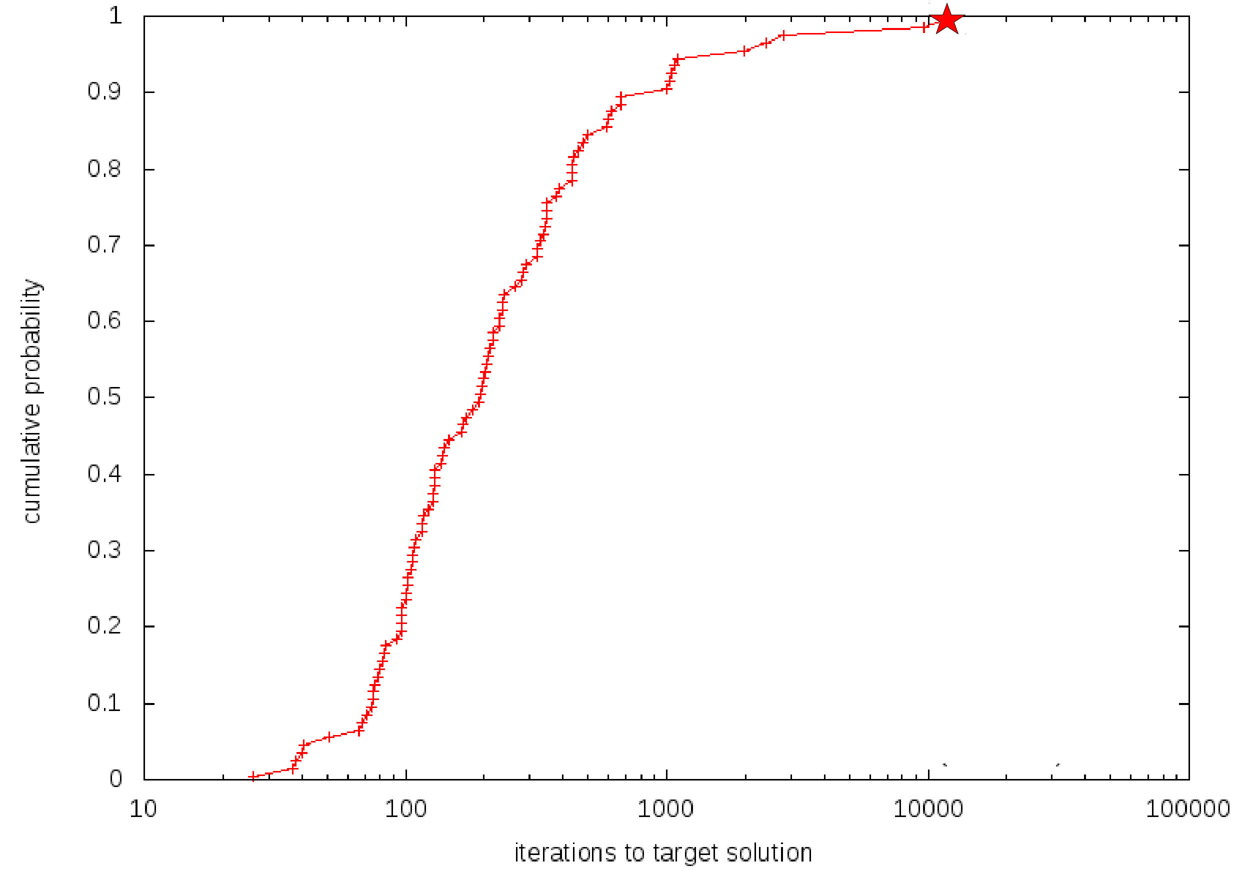
However, some runs take much longer: 10% of the runs take over 1000 iterations



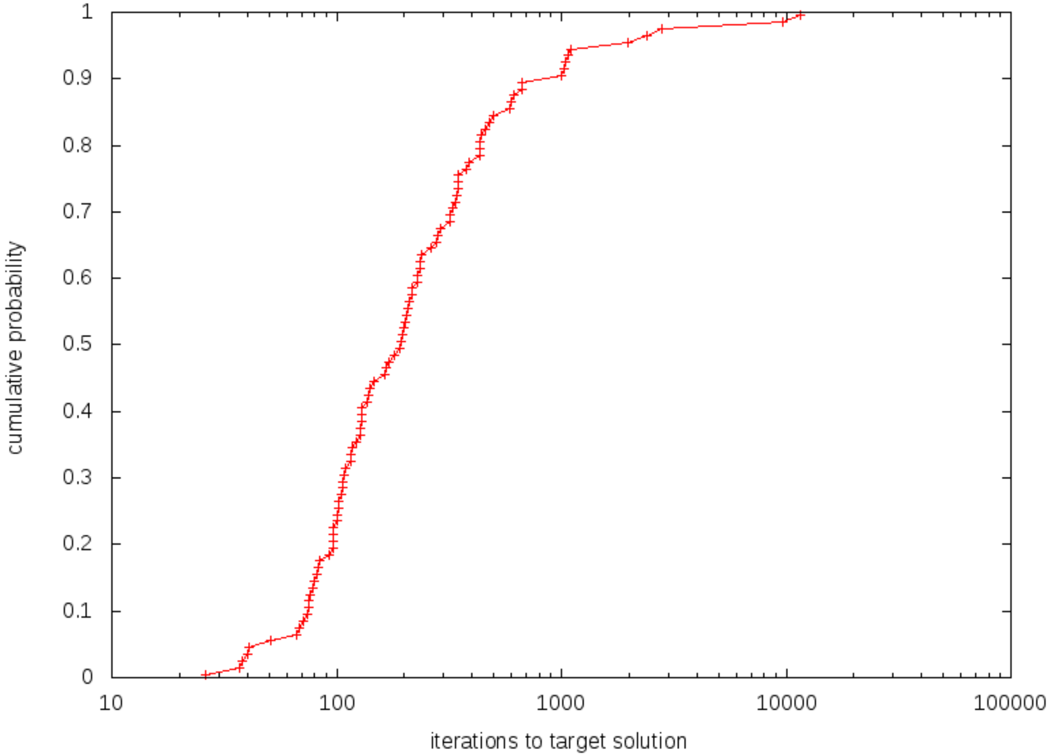
However, some runs take much longer: 5% of the runs take over 2000 iterations



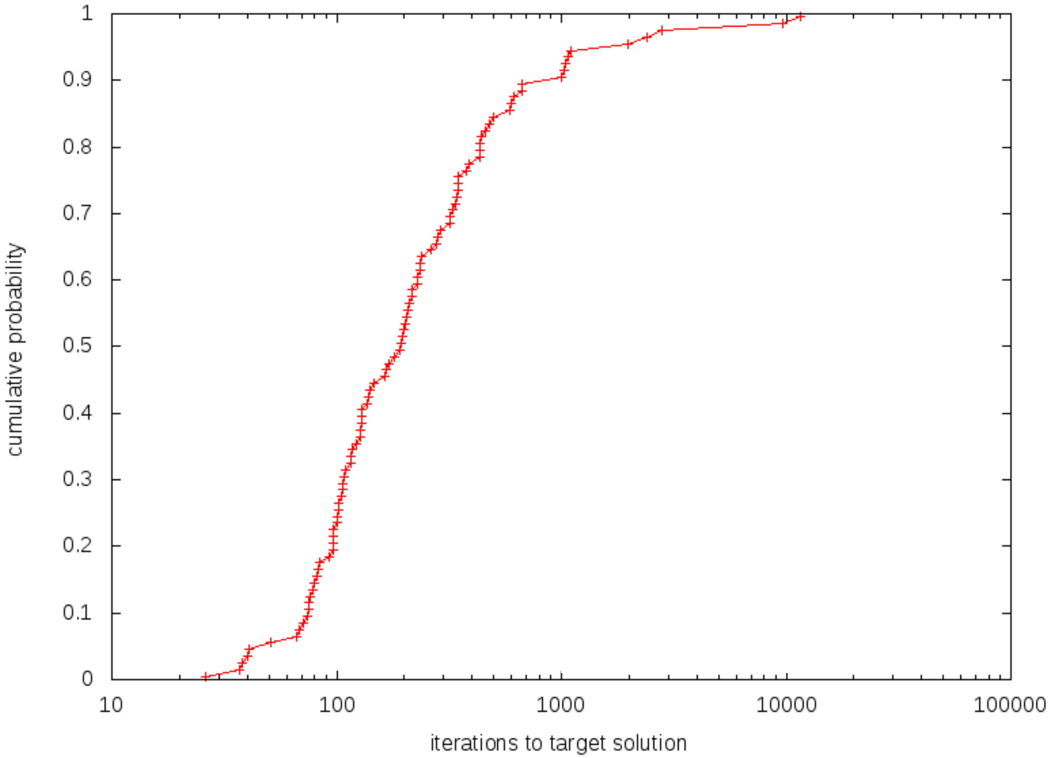
However, some runs take much longer: 2% of the runs take over 9715 iterations



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



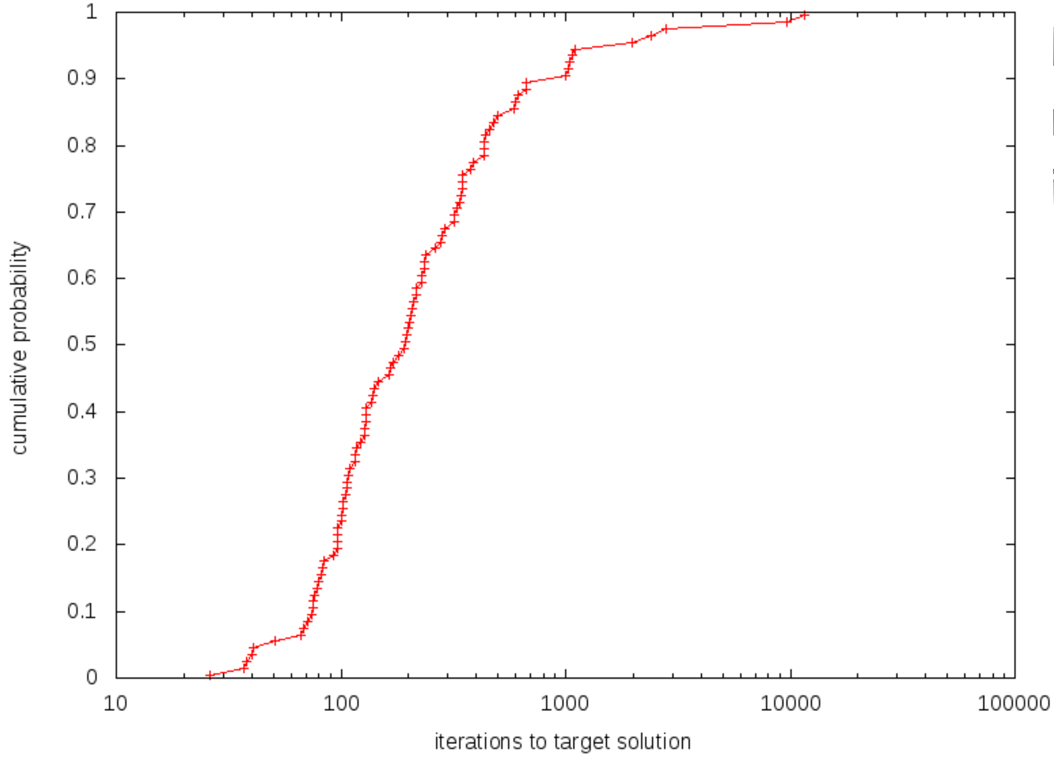
Probability that algorithm will take over 345 iterations: 25% = 1/4



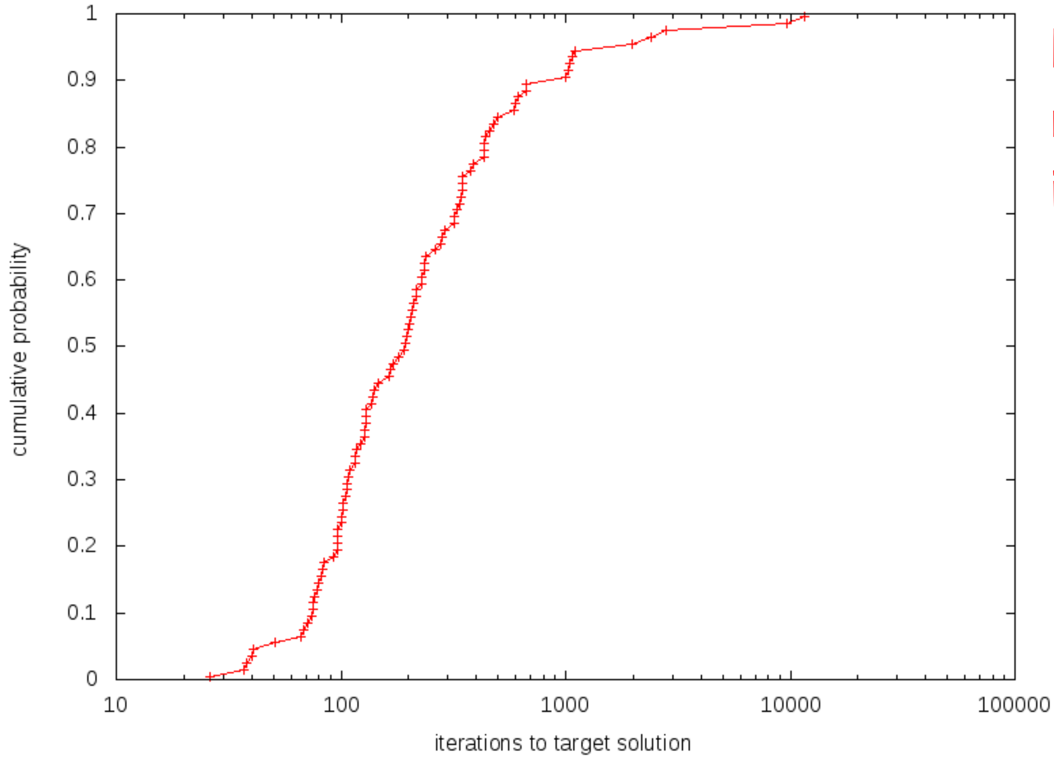
Probability that algorithm will take over 345 iterations: 25% = 1/4

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 = $1/4^2$

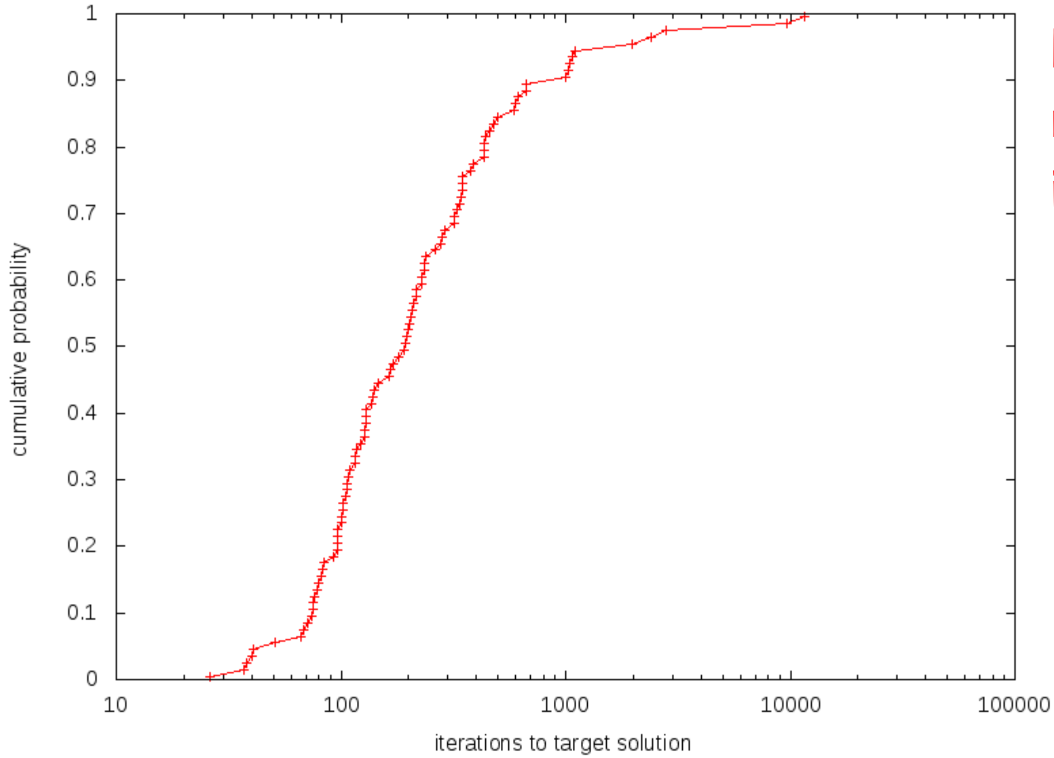


Probability that algorithm will still be running after K periods of 345 iterations: $1/4^K$



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: $1/4^K$

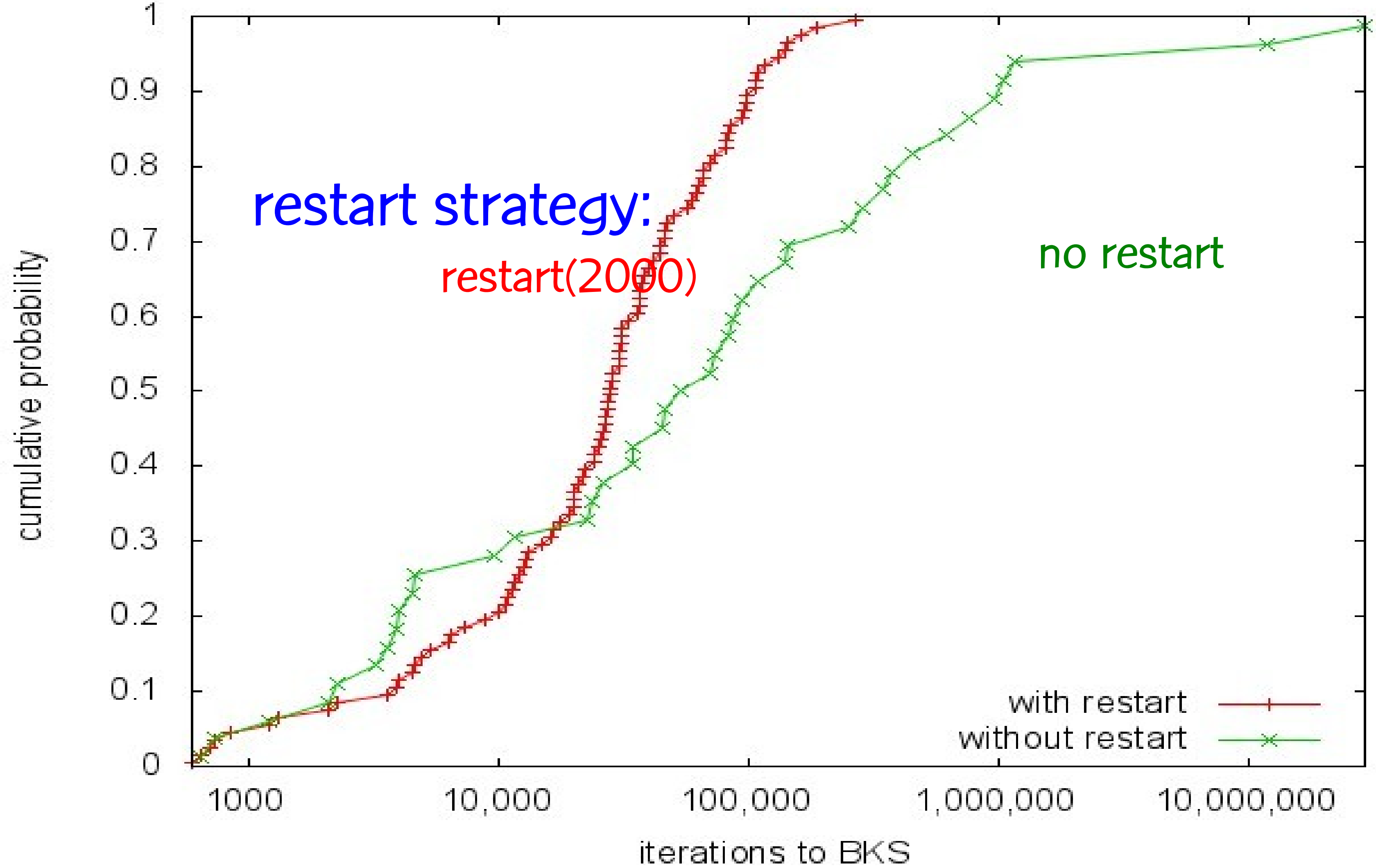
For example, probability that algorithm with restart will still be running after 1725 iterations (5 periods of 345 iterations): $1/4^5 \cong$ 0.0977%

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

Rethink Possible

att.com/rethinkpossible

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

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

Concluding remarks

- Reviewed BRKGA framework
- Showed BRKGA outperforms RKGA of Bean (1994)
- Reviewed restart mechanisms for BRKGA heuristics
- Showed how to specify a BRKGA heuristic
- Presented an C++ API for BRKGA

Thanks!

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

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