Biased random-key genetic algorithms

Mauricio G. C. Resende
Mathematical Optimization & Planning
Amazon.com
Seattle, Washington

resendem AT amazon DOT com

Invited talk given at Instituto de Informática
Federal U. of Rio Grande do Sul
Porto Alegre, Brazil ✤ December 9, 2015

Work done when speaker was employed at
AT&T Labs Research.

Joint work with José F. Gonçalves
U. do Porto
Portugal

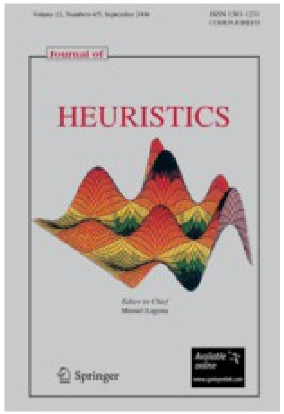
Thanks also to ...

Martin Ericsson, Panos Pardalos, Luciana Buriol, Celso Ribeiro, Mikkel Thorup, Diogo Andrade, Jorge Mendes, Thiago Noronha, Michael Hirsch, Tania Querido, Marcus Ritt, Paola Festa, Lee Breslau, Ilias Diakonikolas, Nick Duffield, Yu Gu, MohammadTaghi Hajiaghayi, David Johnson, Howard Karloff, Subhabrata Sen, Roger Reis, Cristian Martinez, Irene Loiseau, S. Rodriguez, Rodrigo Toso, Ricardo Silva, Luis Morán, José Luis González-Velarde, Carlos de Andrade, Flávio Miyazawa, João Lauro Facó, Alex Grasas, Helena Ramalhinho, Luciana Pessoa, Imma Caballé, Nuria Barba, Abraham Duarte, Rafael Martí, Miguel Costa, Marina Lucena, Efrain Ruiz, Maria Albareda-Sambola, Elena Fernández, Fernando Stefanello, Julliany Brandão, Max Zhang, Rakesh Sinha, Ken Reichmann, Robert Doverspike, and Renato Werneck ... who have worked with me on BRKGAs

Summary

- 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
- Applications
  - BRKGA for 2-dim and 3-dim packing
  - BRKGA for 3-dim bin packing
  - BRKGA for unequal area facility layout
- Concluding remarks

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://mauricio.resende.info/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

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

Decode by sorting vector of random keys

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

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

[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 $i = 1,\ldots,5$.

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).

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

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.

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

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

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

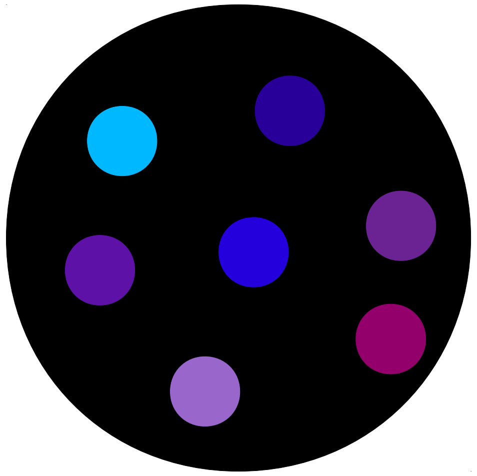
- Mating is done using parametrized uniform crossover (Spears & DeJong , 1990)
- For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

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.

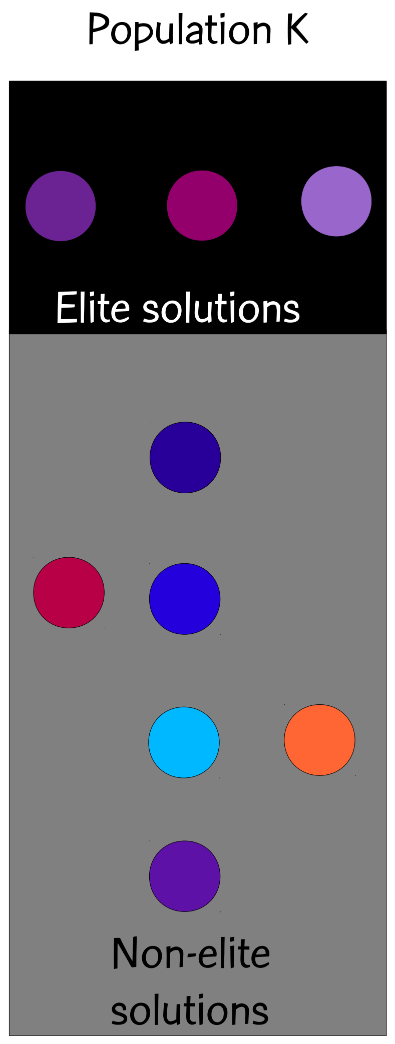
GAs and random keys

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).



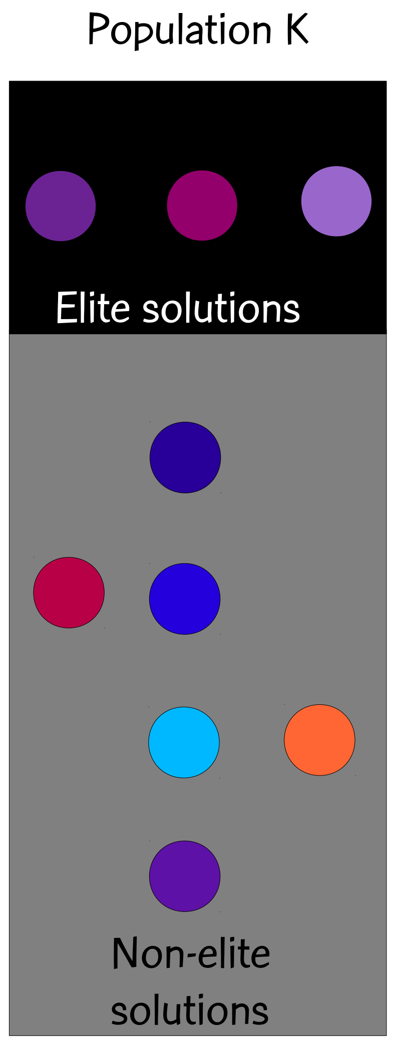
GAs and random keys

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



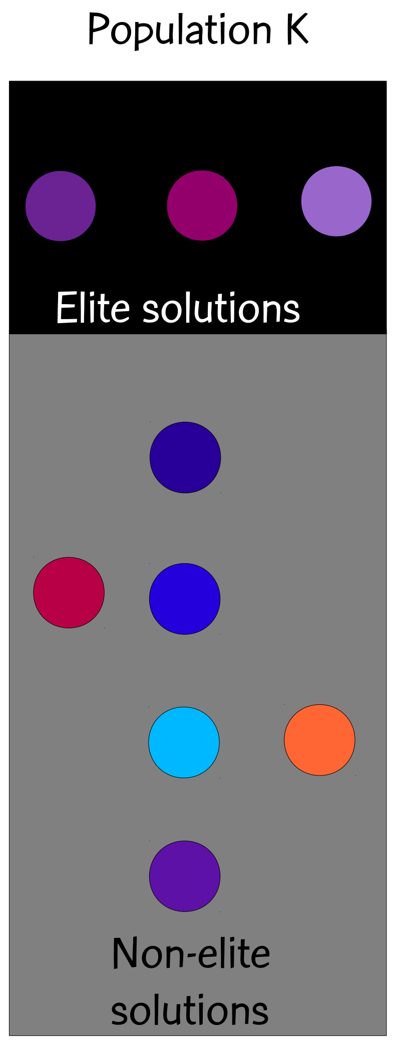
GAs and random keys

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



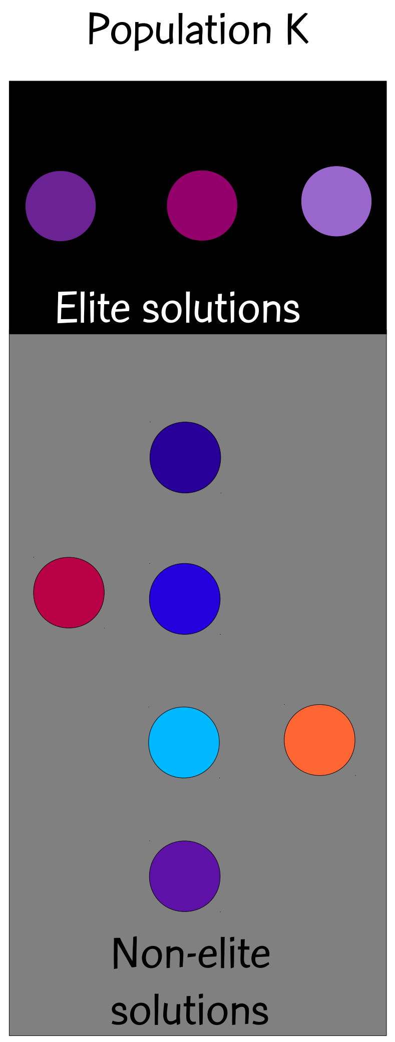
GAs and random keys

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



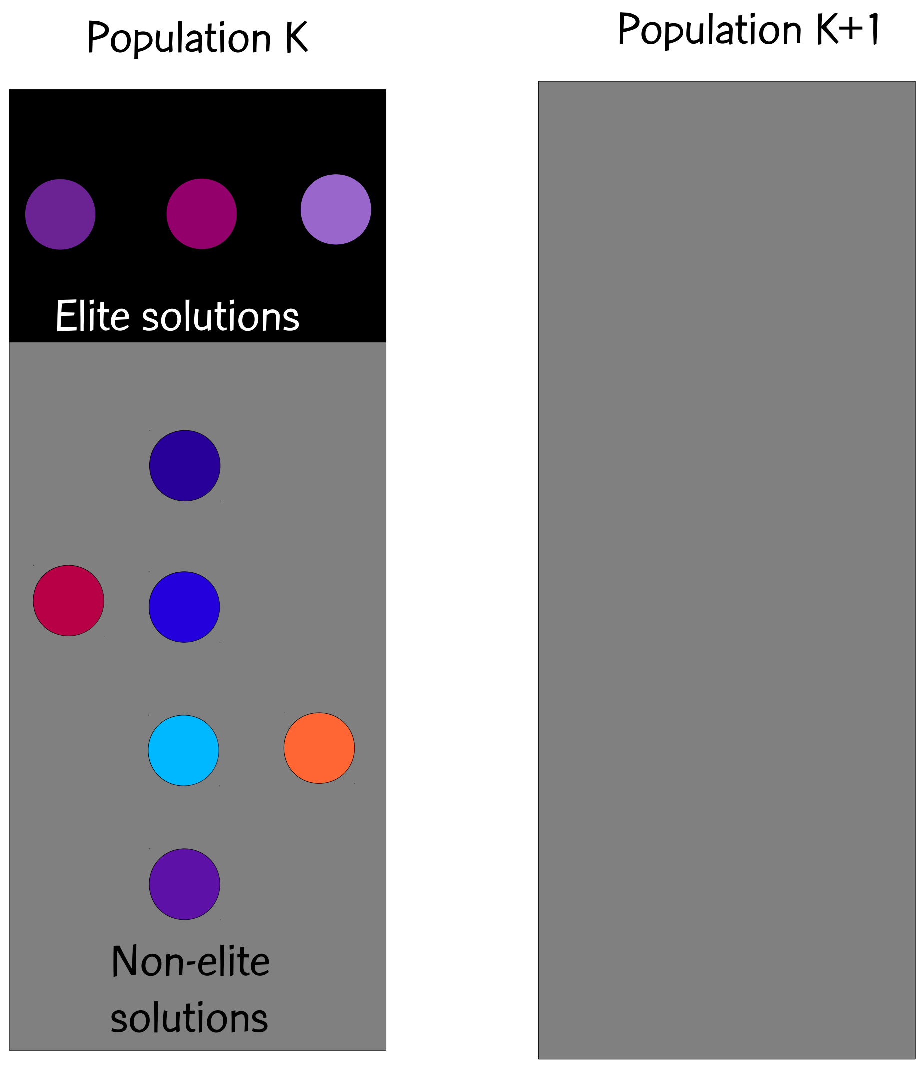
GAs and random keys

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

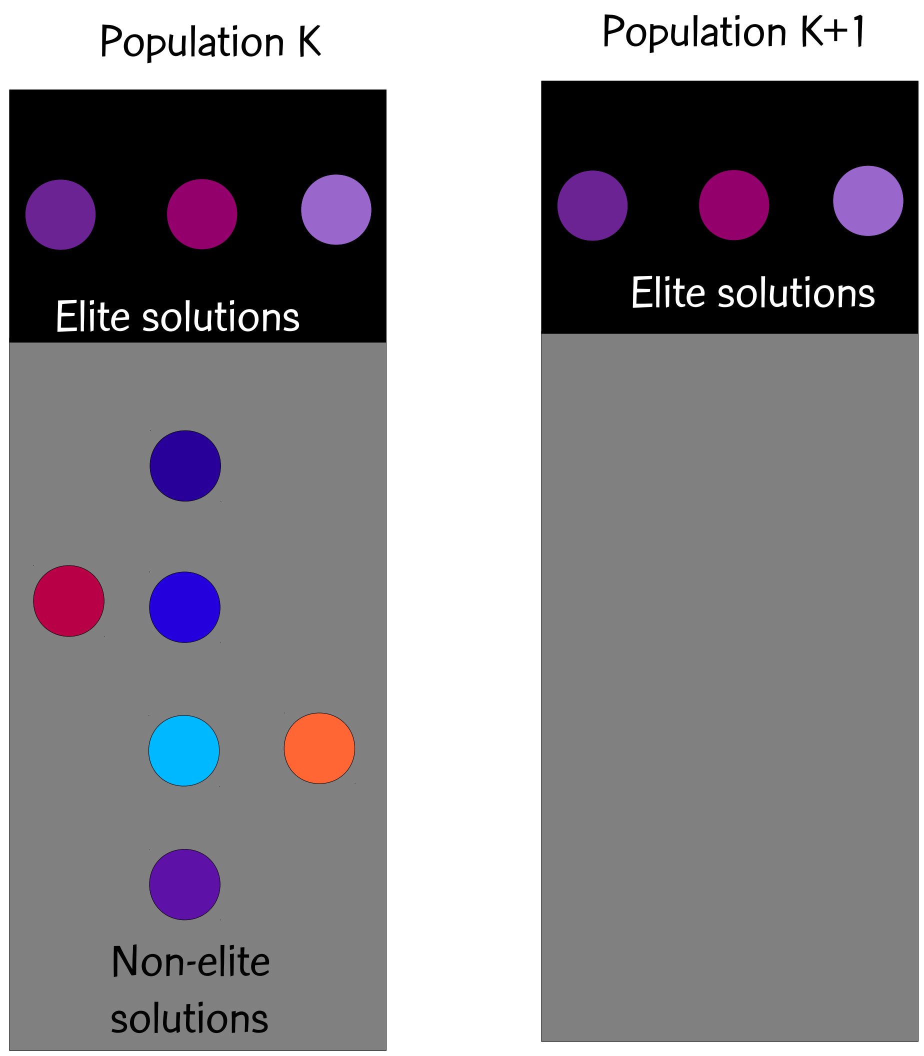
Evolutionary dynamics



GAs and random keys

Evolutionary dynamics

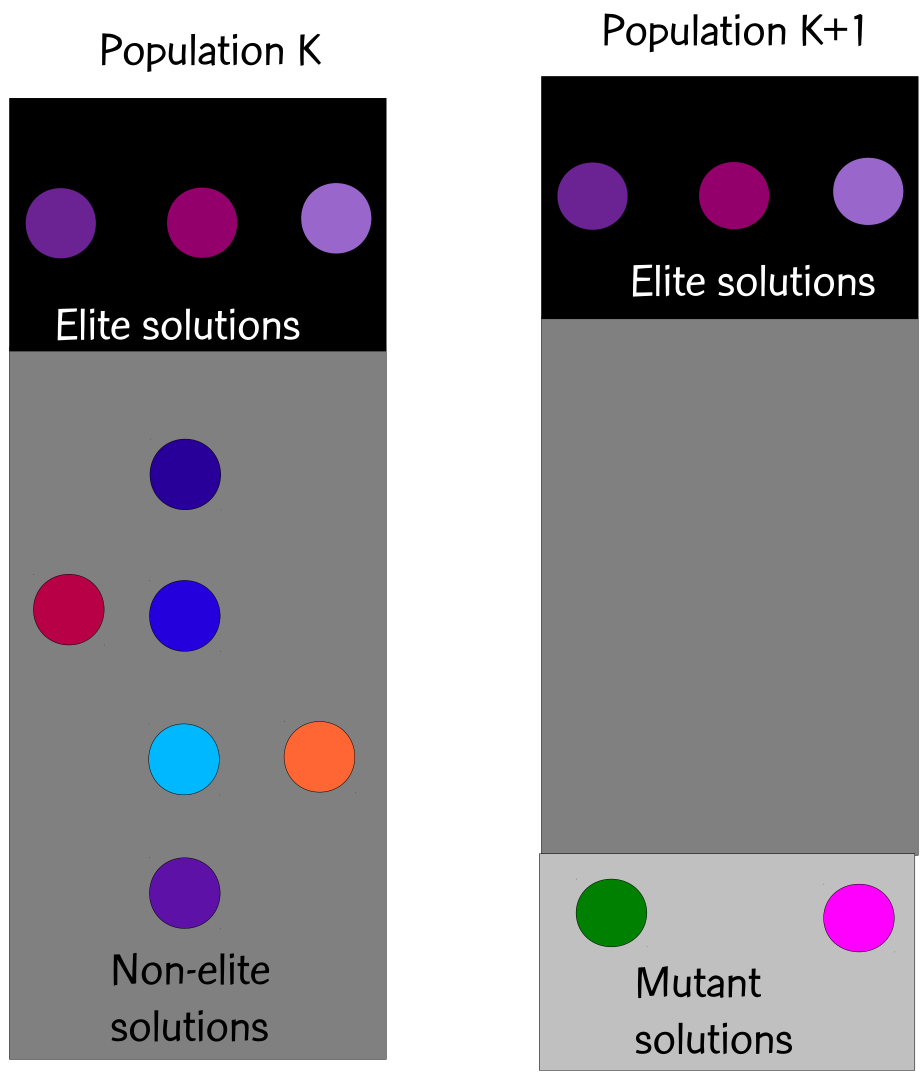
– Copy elite solutions from population K to population K+1



GAs and random keys

Evolutionary dynamics

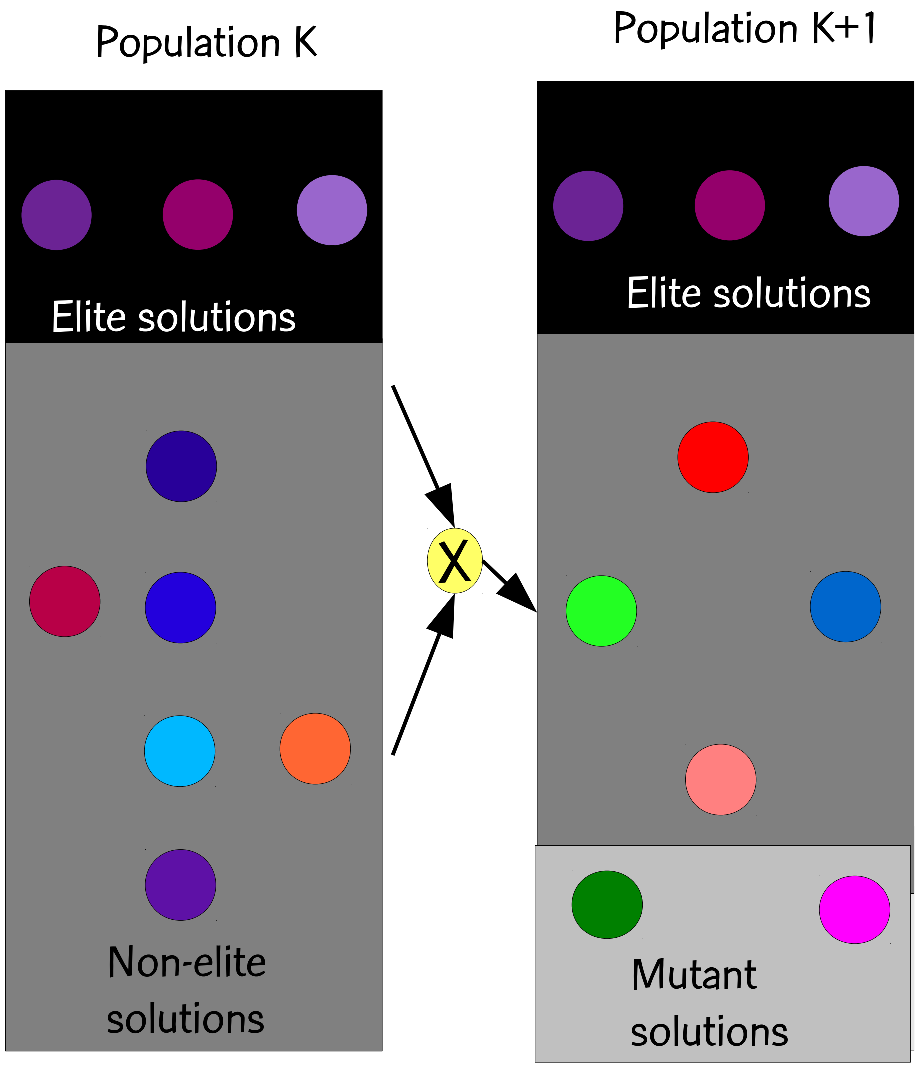
- Copy elite solutions from population K to population K+1
- Add R random solutions (mutants) to population K+1



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

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

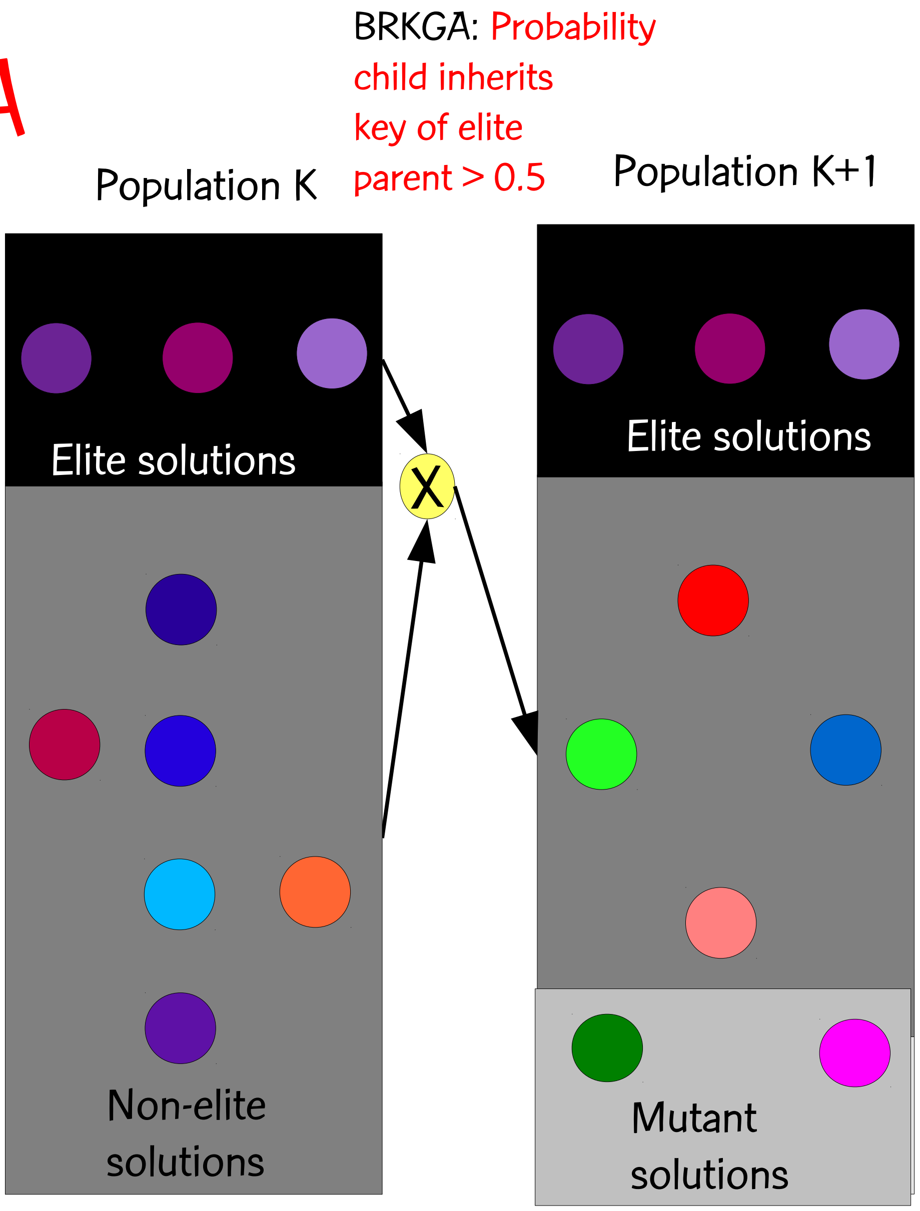
How RKGA & BRKGA differ

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

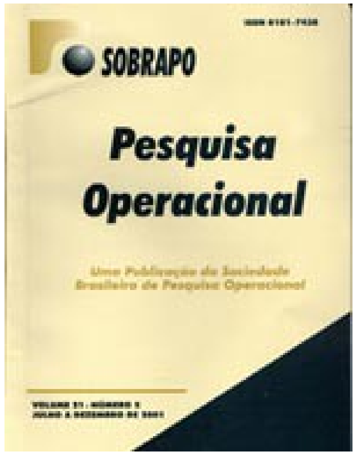
Biased random key GA

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


SOBRAPO

Pesquisa Operacional

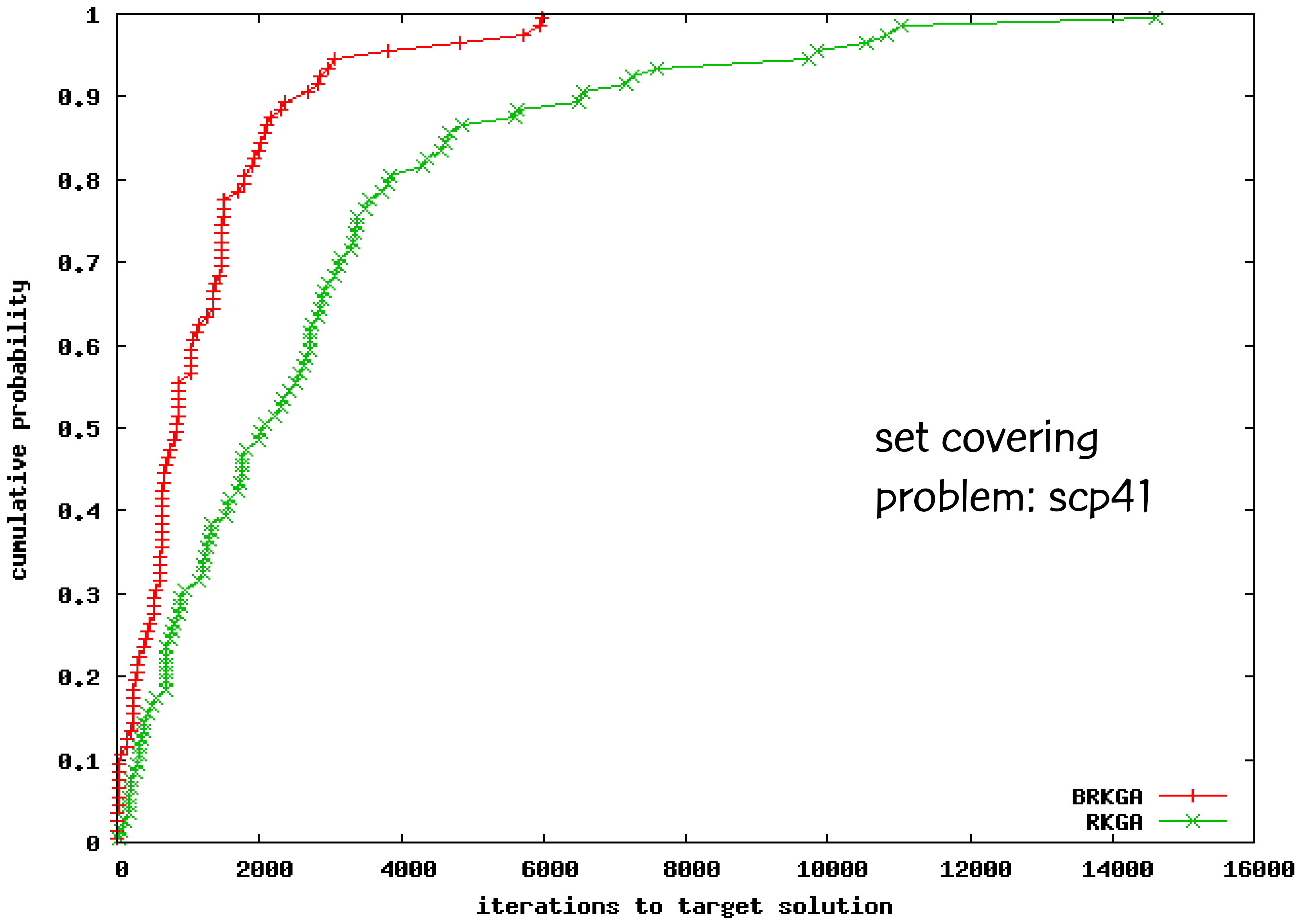
Uma Publicação da Sociedade Brasileira de Pesquisa Operacional

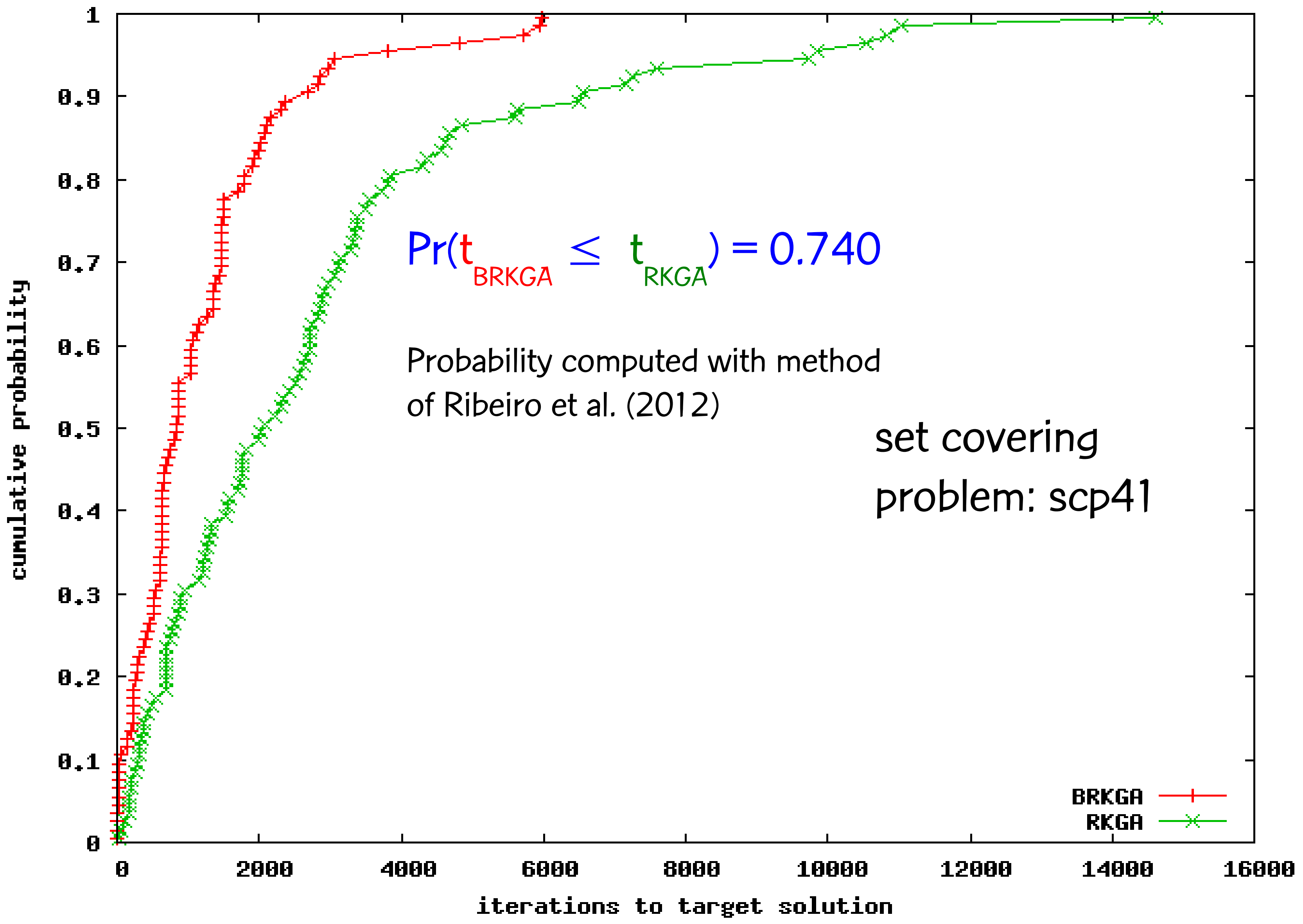
Gonçalves, R., and Toso,

"An experimental comparison of biased and unbiased random-key genetic algorithms",

Pesquisa Operacional, vol. 34, pp. 143-164, 2014.

amazon

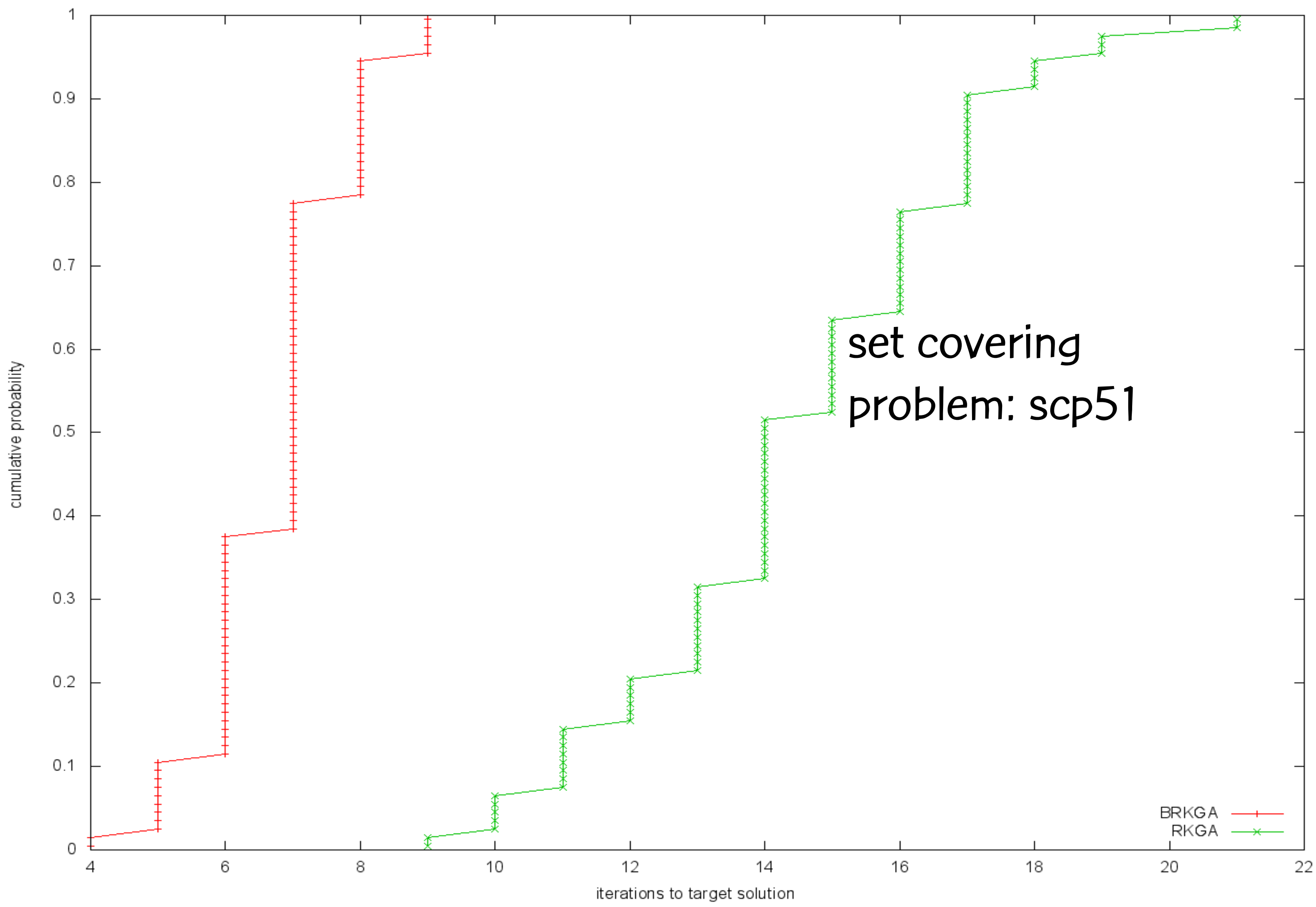

cumulative probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
12000
14000
16000
iterations to target solution
set covering problem: scp41
BRKGA
RKGA



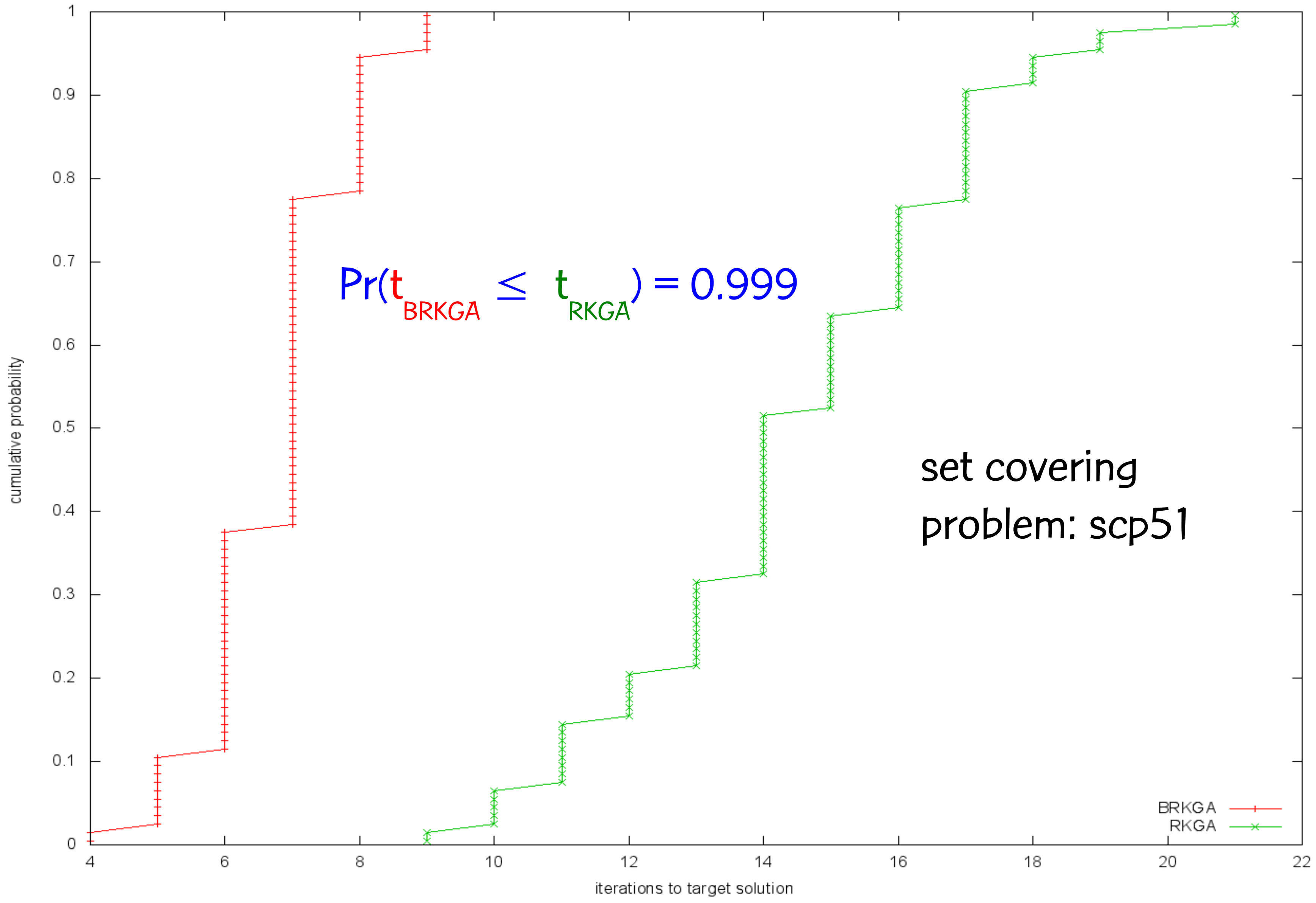
$Pr(t_{BRKGA} \leq t_{RKGA}) = 0.740$

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

set covering problem: scp41

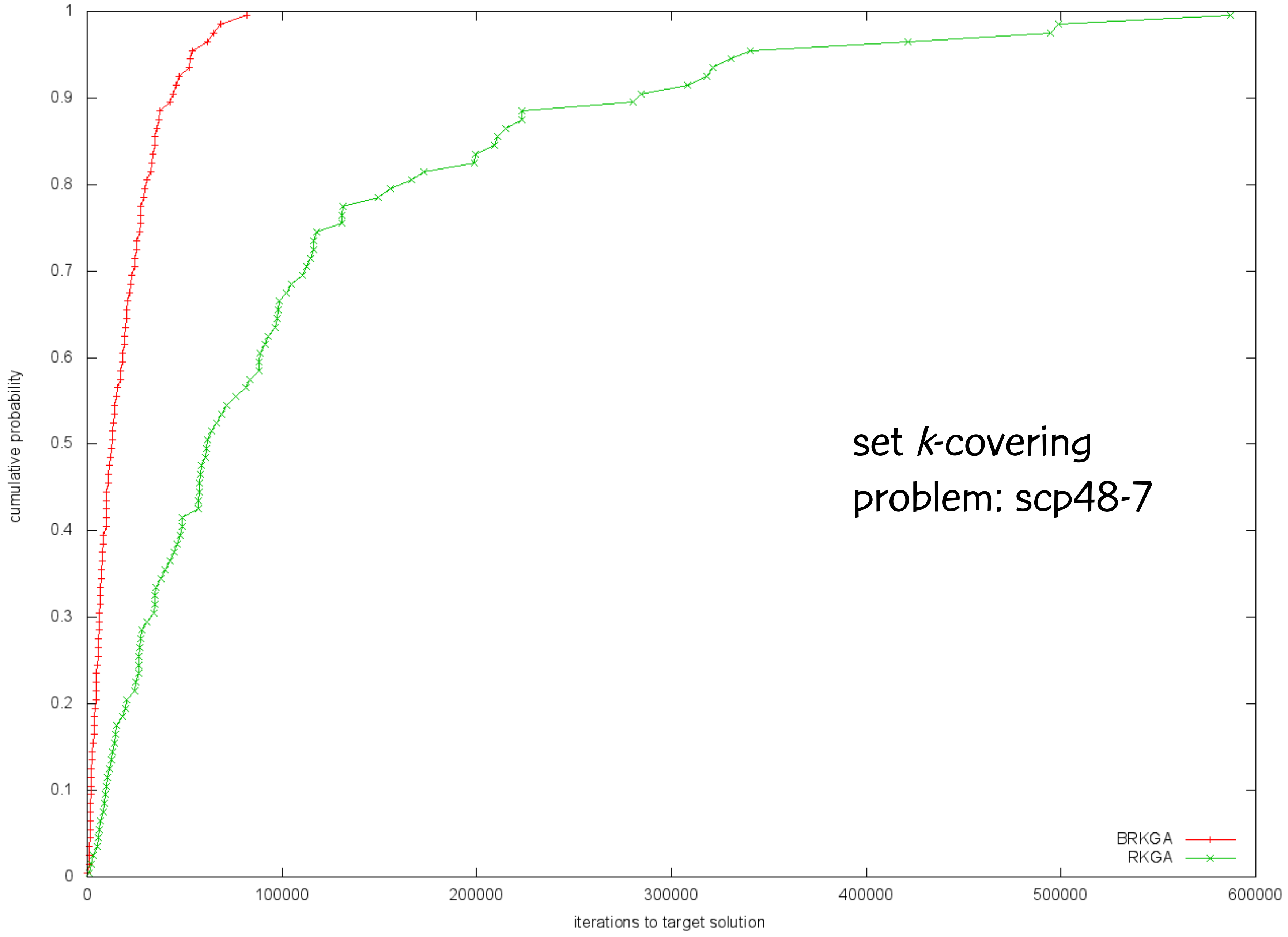


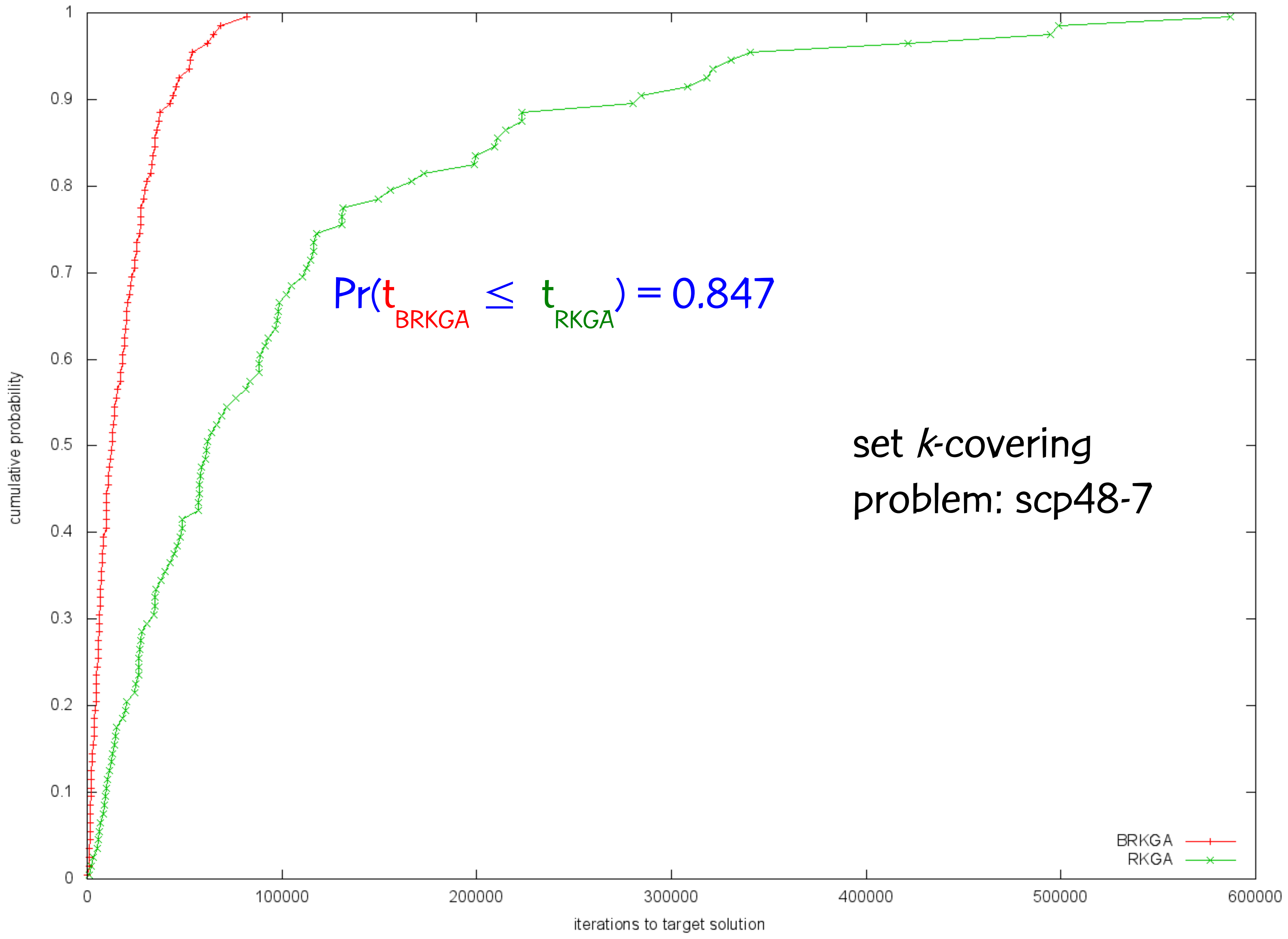
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
cumulative probability
4
6
8
10
12
14
16
18
20
22
iterations to target solution
BRKGA
RKGA
set covering problem: scp51



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

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
100000
200000
300000
400000
500000
600000
BRKGA
RKGA
set *k*-covering problem: scp48-7



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

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)

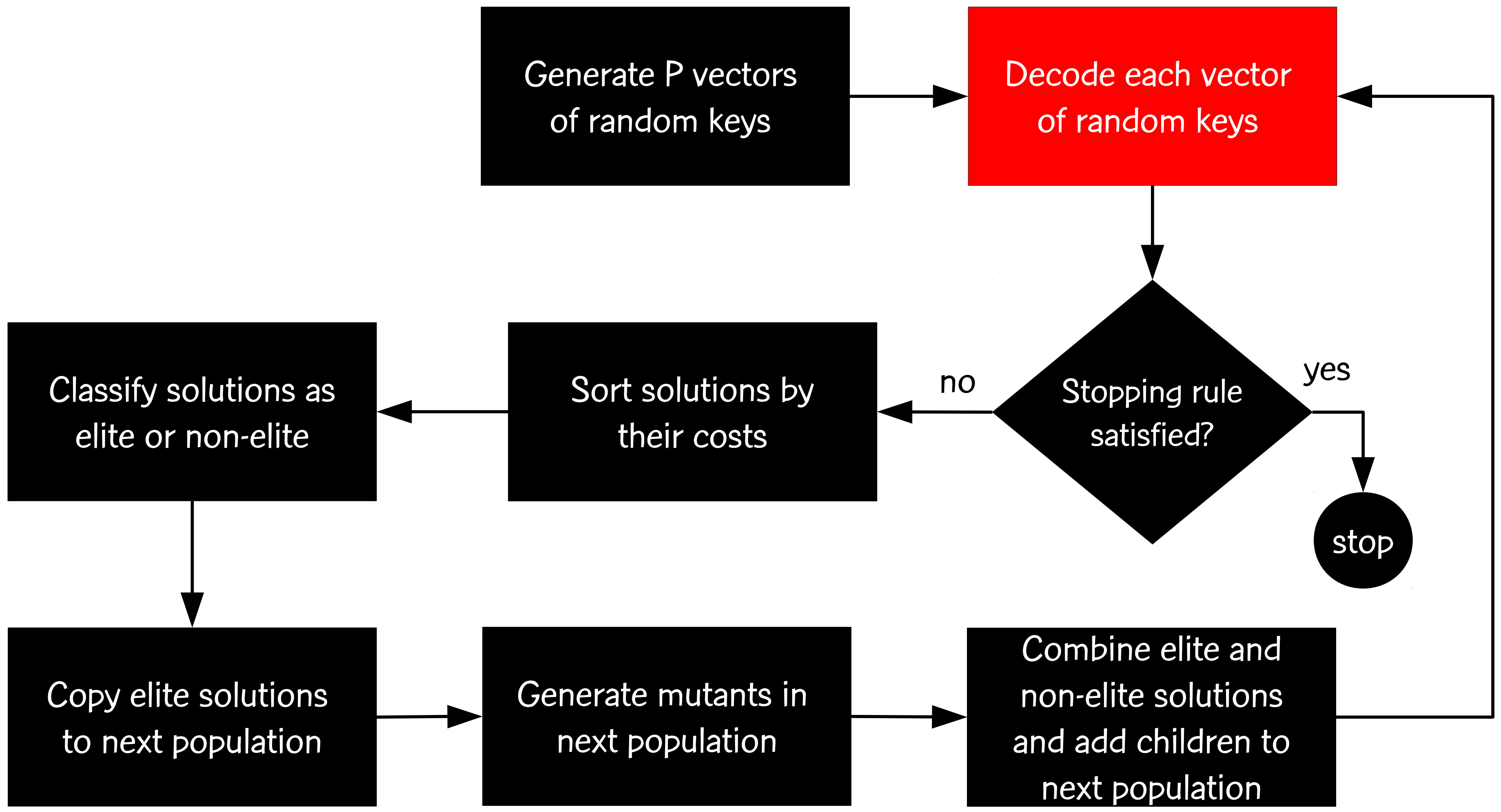
Observations

- 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.

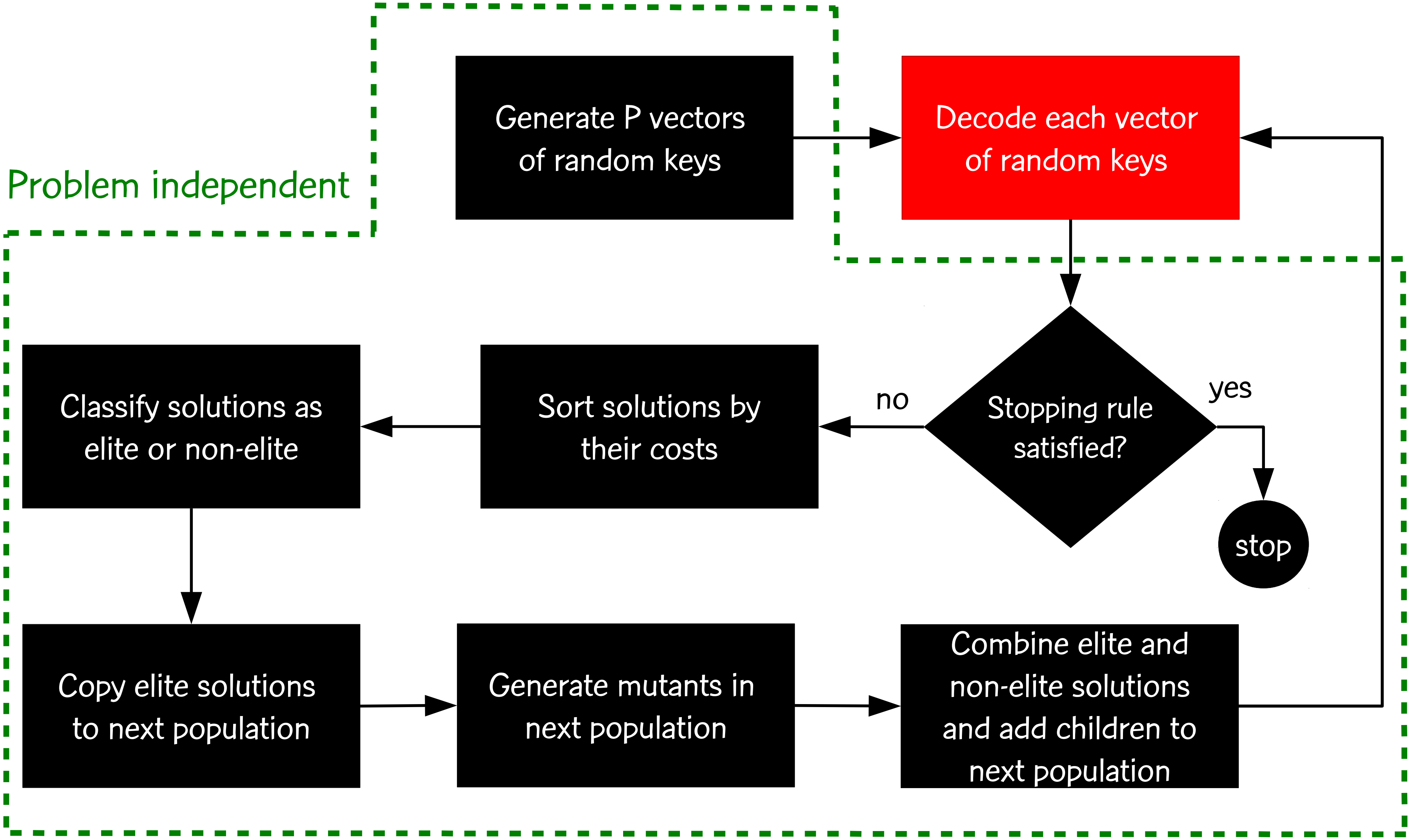
Observations

- 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)

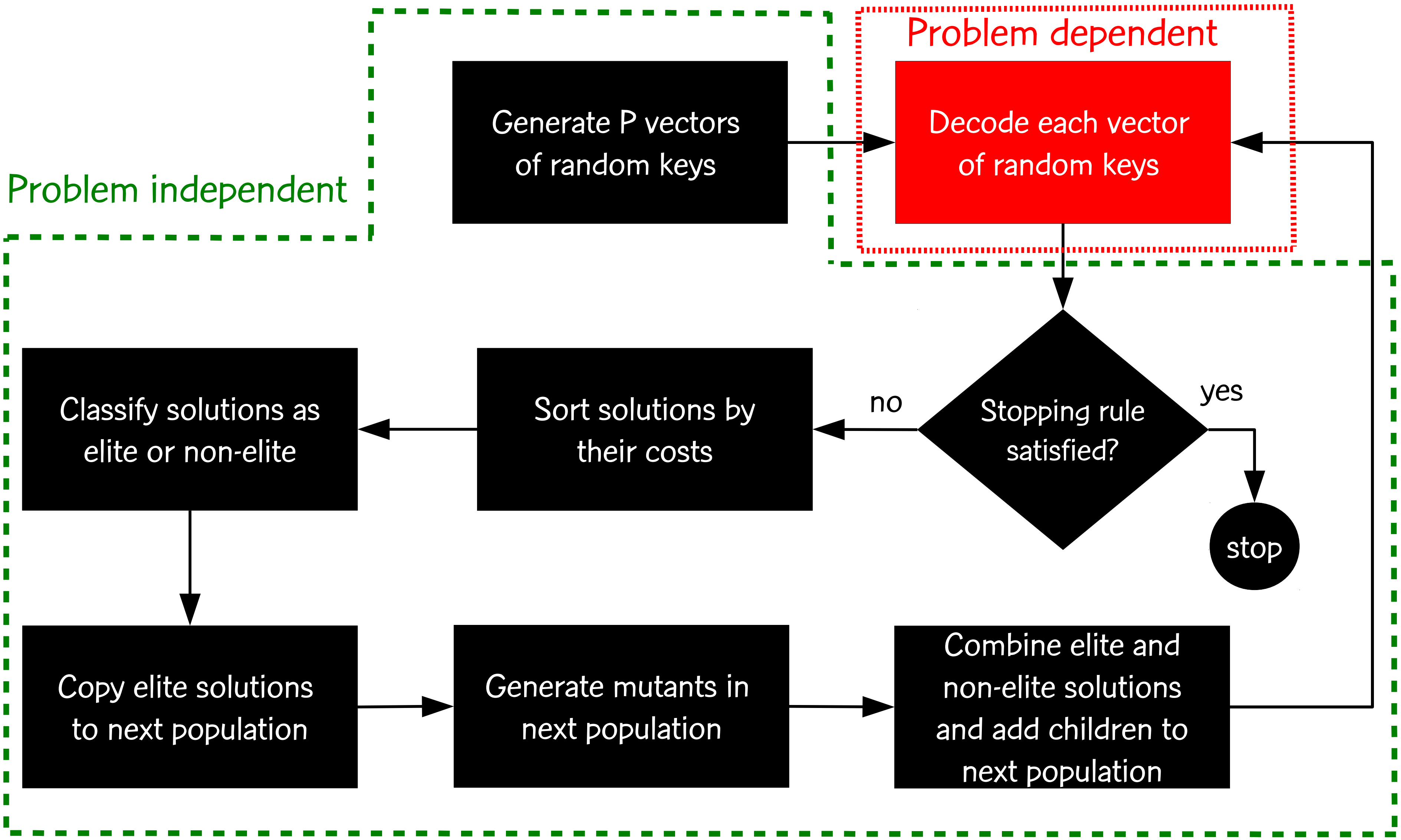
Framework for biased random-key genetic algorithms


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

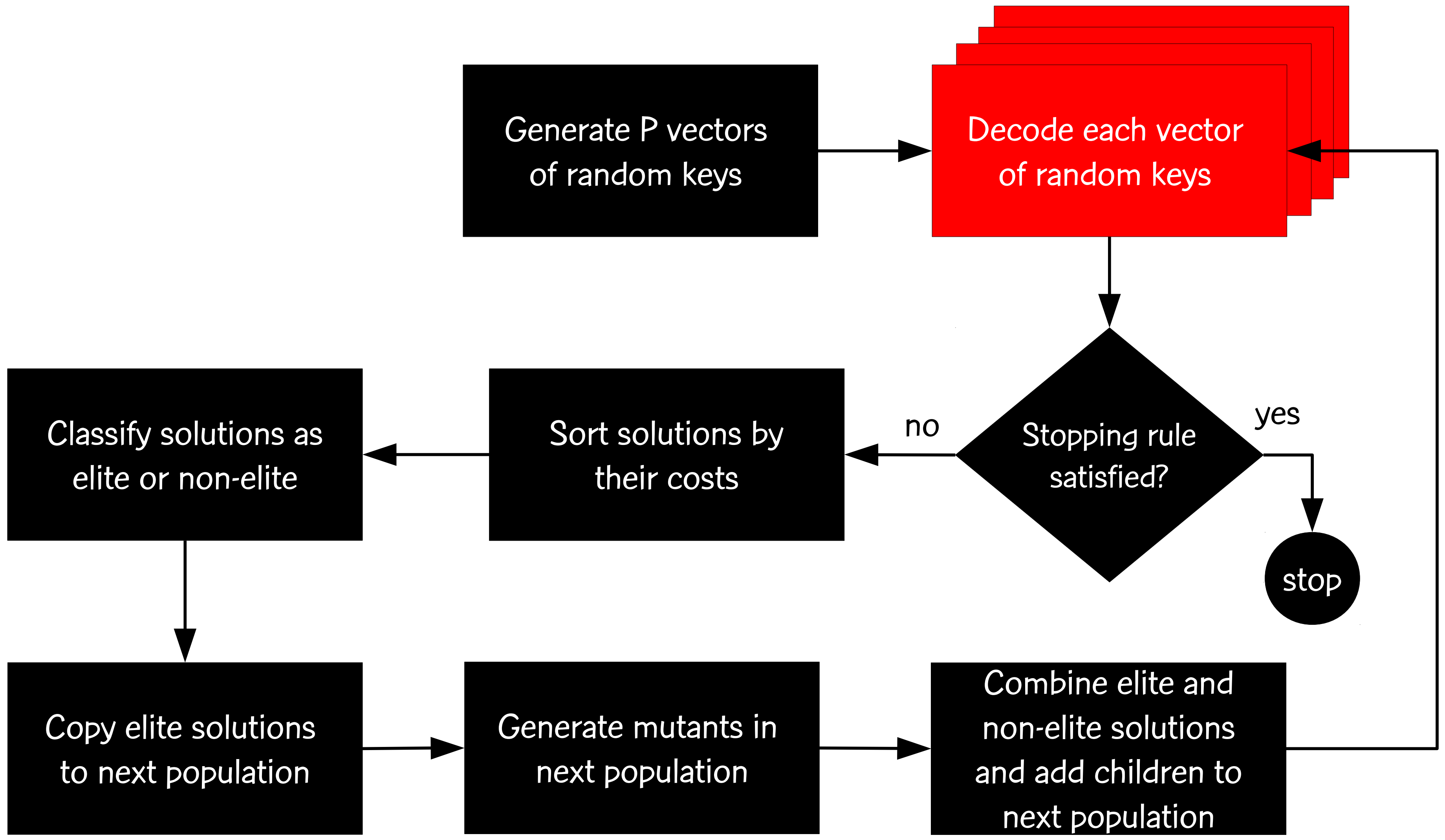
Framework for biased random-key genetic algorithms


Problem independent
Generate P vectors of random keys
Decode each vector of random keys
Stopping rule satisfied?
yes
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
Problem dependent
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

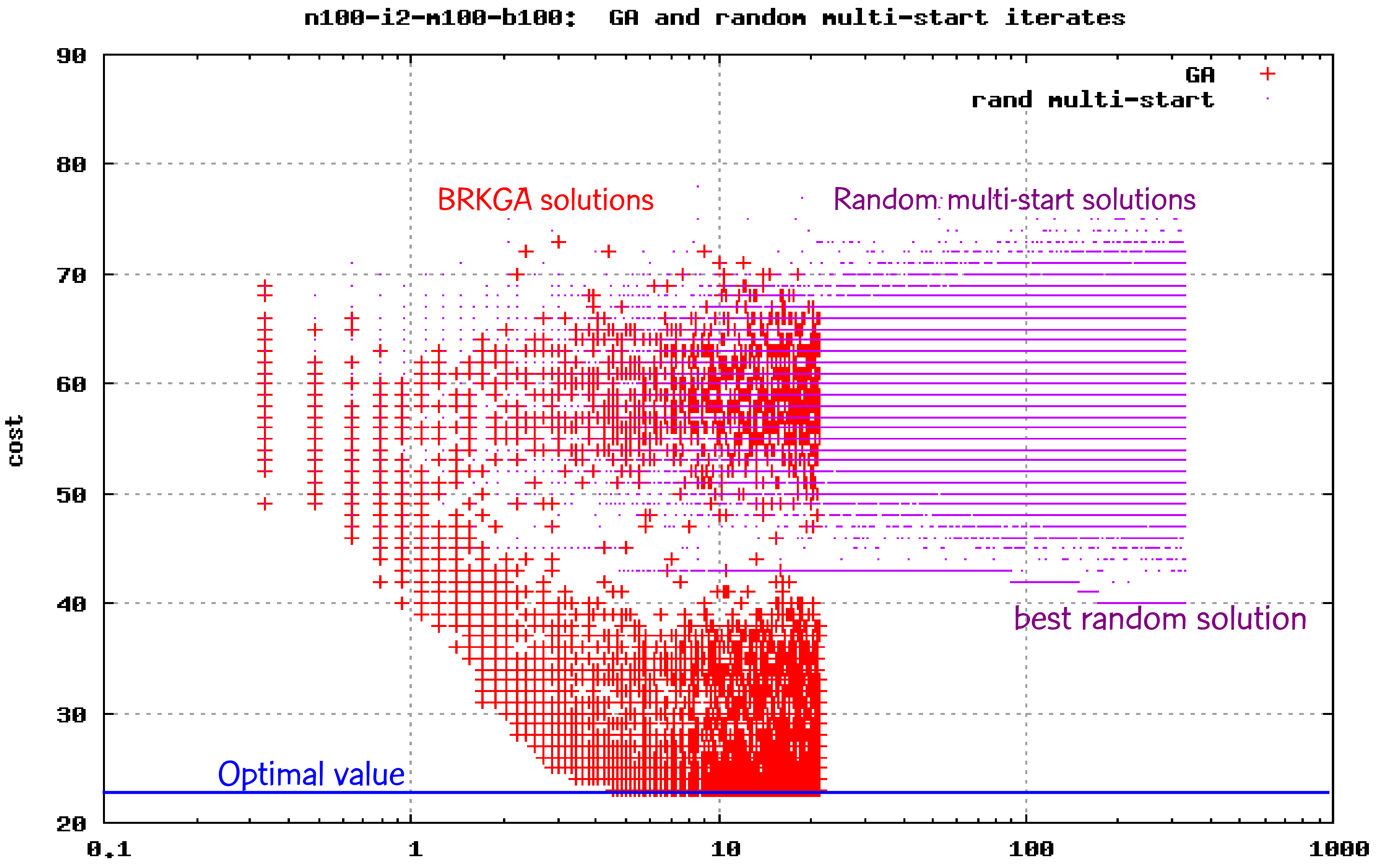
Decoding of random key vectors can be done in parallel


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

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

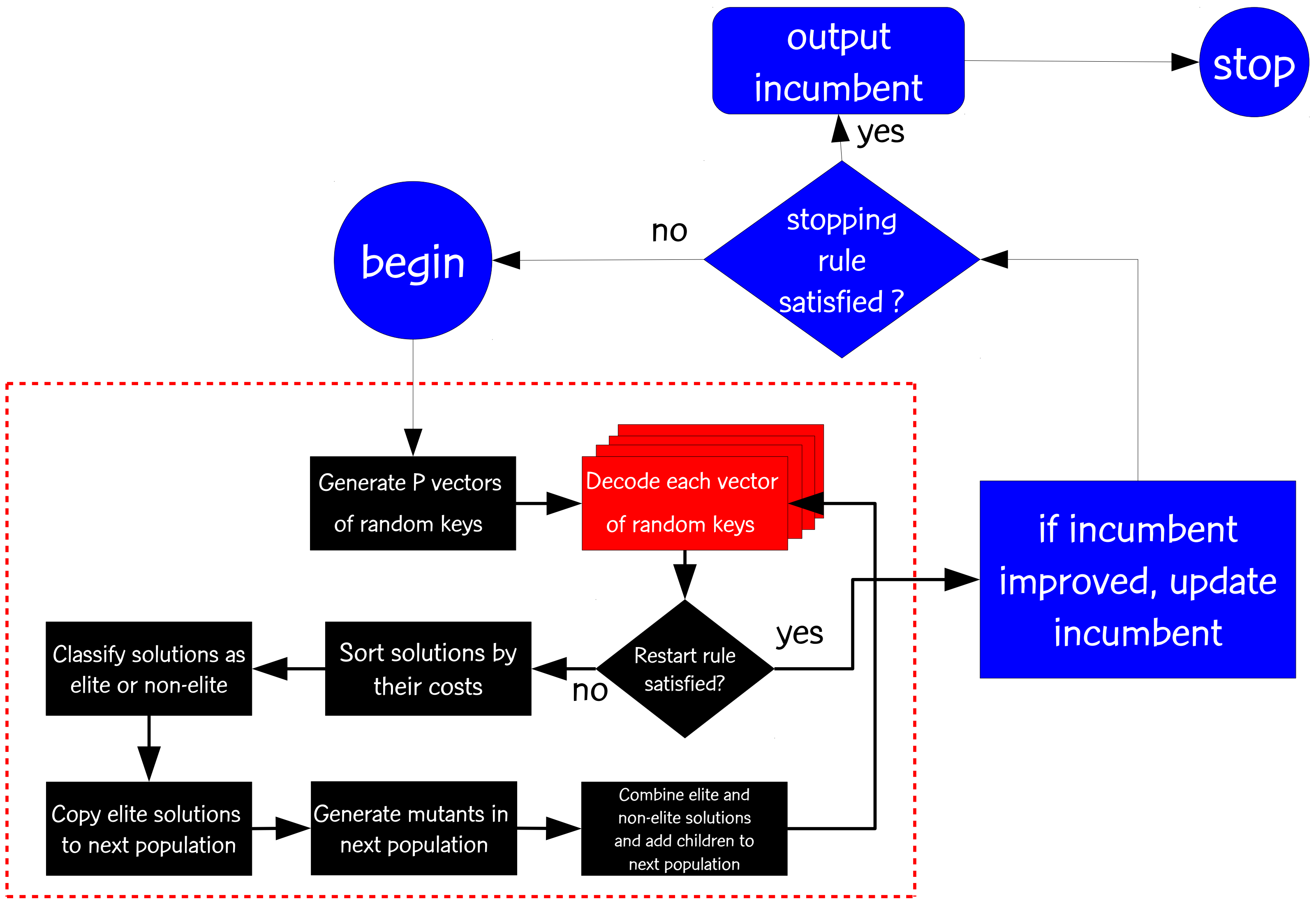
- Simulate a random multi-start decoding method with a BRKGA by setting size of elite partition to 1 and number of mutants to P–1
- Each iteration, best solution is maintained in elite set and P–1 random key vectors are generated as mutants ... no mating is done since population already has P individuals

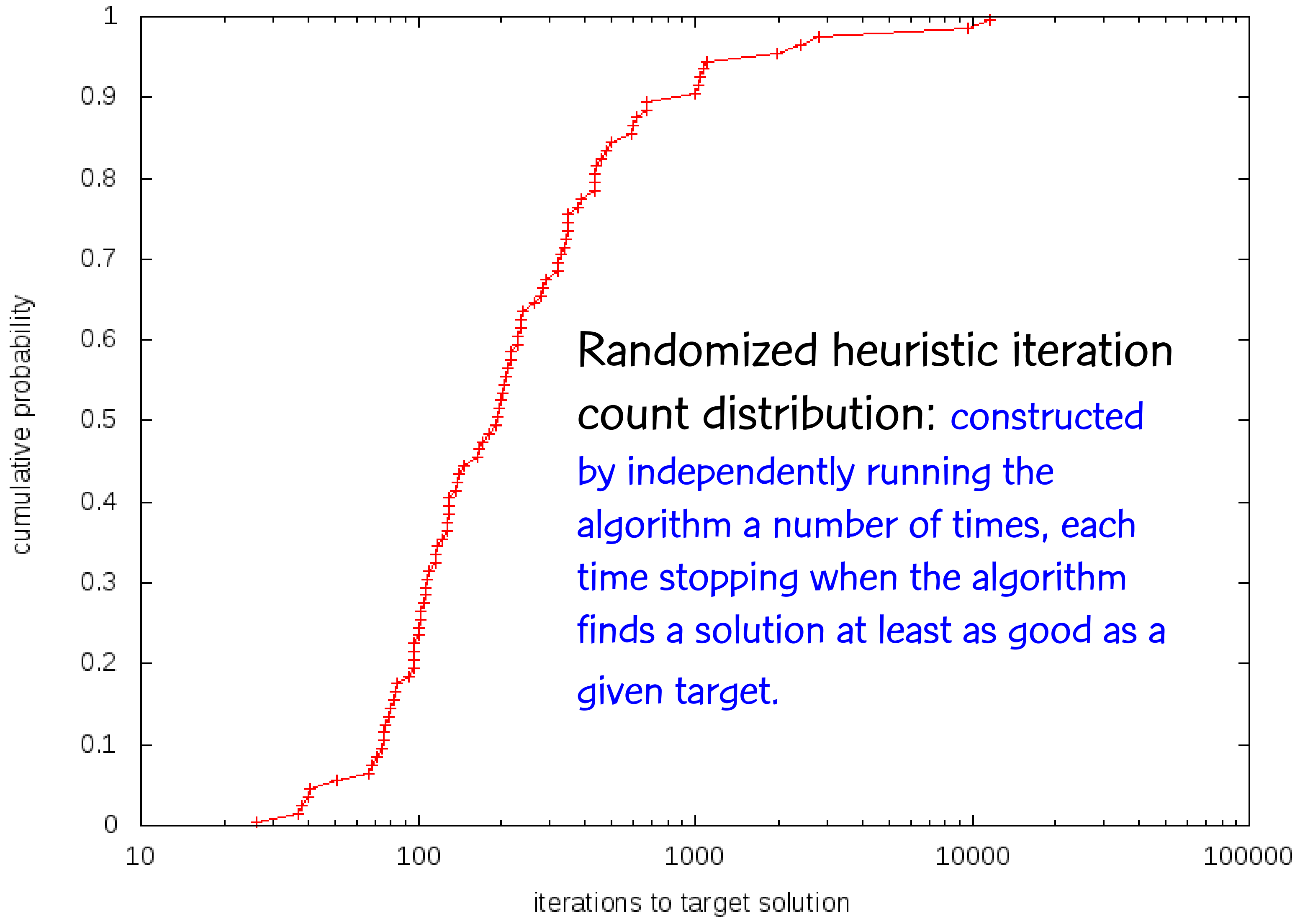
Network monitor location problem (opt = 23) solution



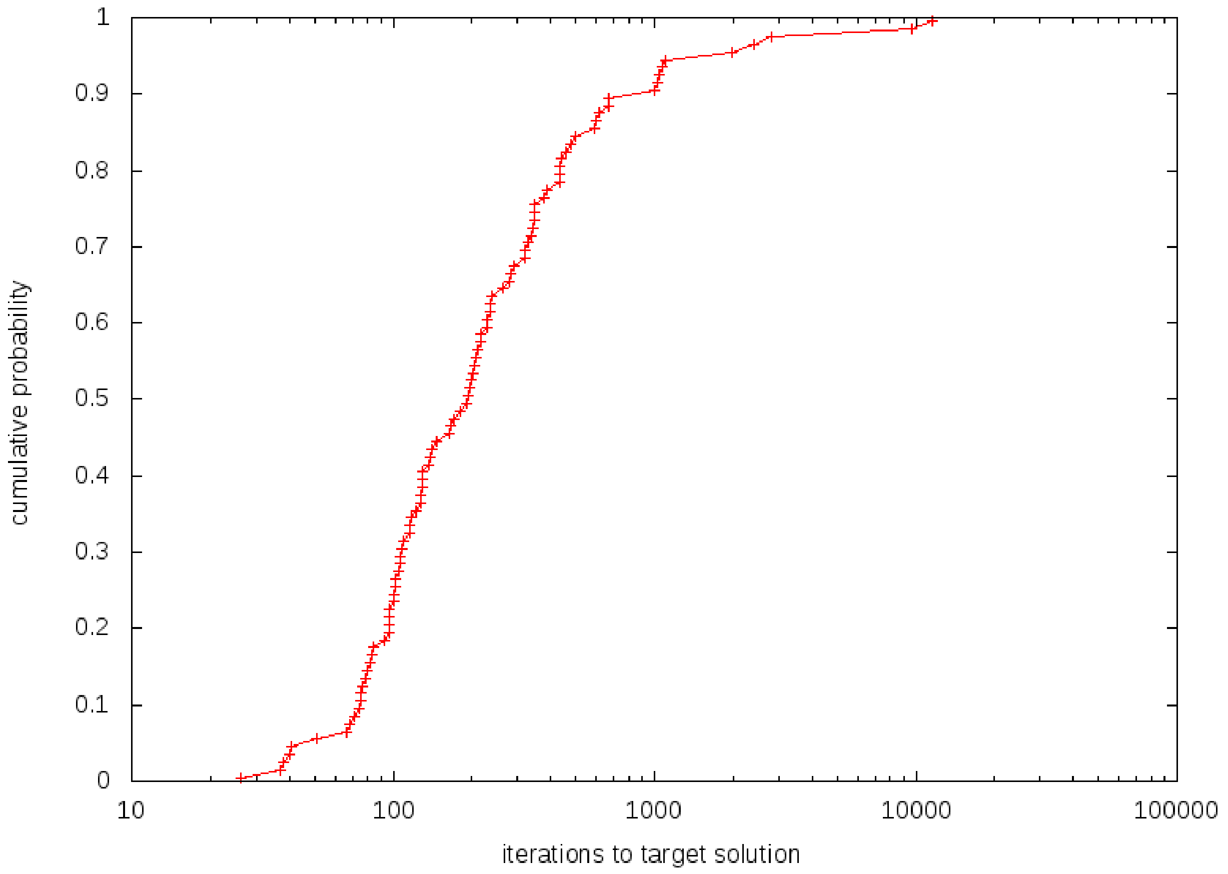
Time (ibm t41 secs)

BRKGA in multi-start strategy

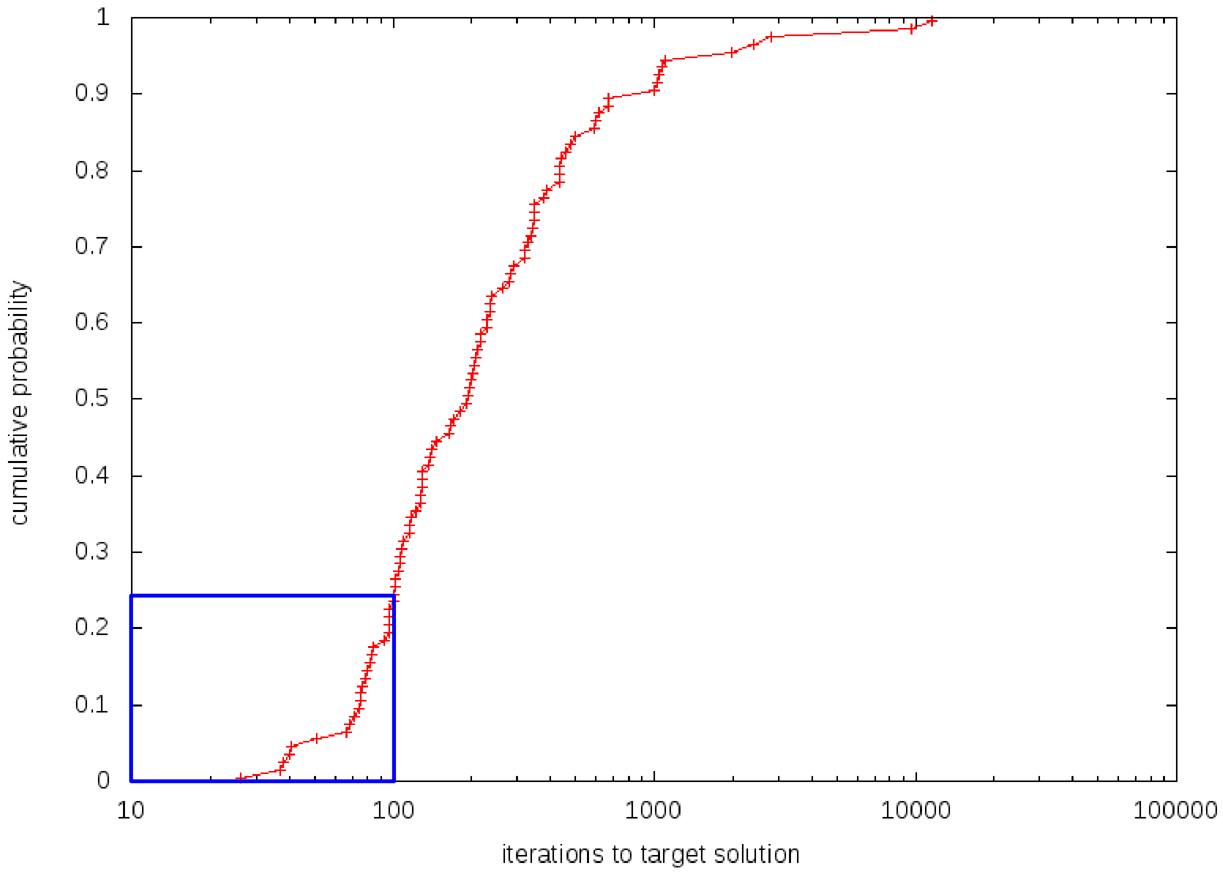

output incumbent
stop
yes
no
begin
stopping rule satisfied ?
Generate P vectors of random keys
Decode each vector of random keys
if incumbent improved, update incumbent
Restart rule satisfied?
yes
no
Classify solutions as elite or non-elite
Sort solutions by their costs
Copy elite solutions to next population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



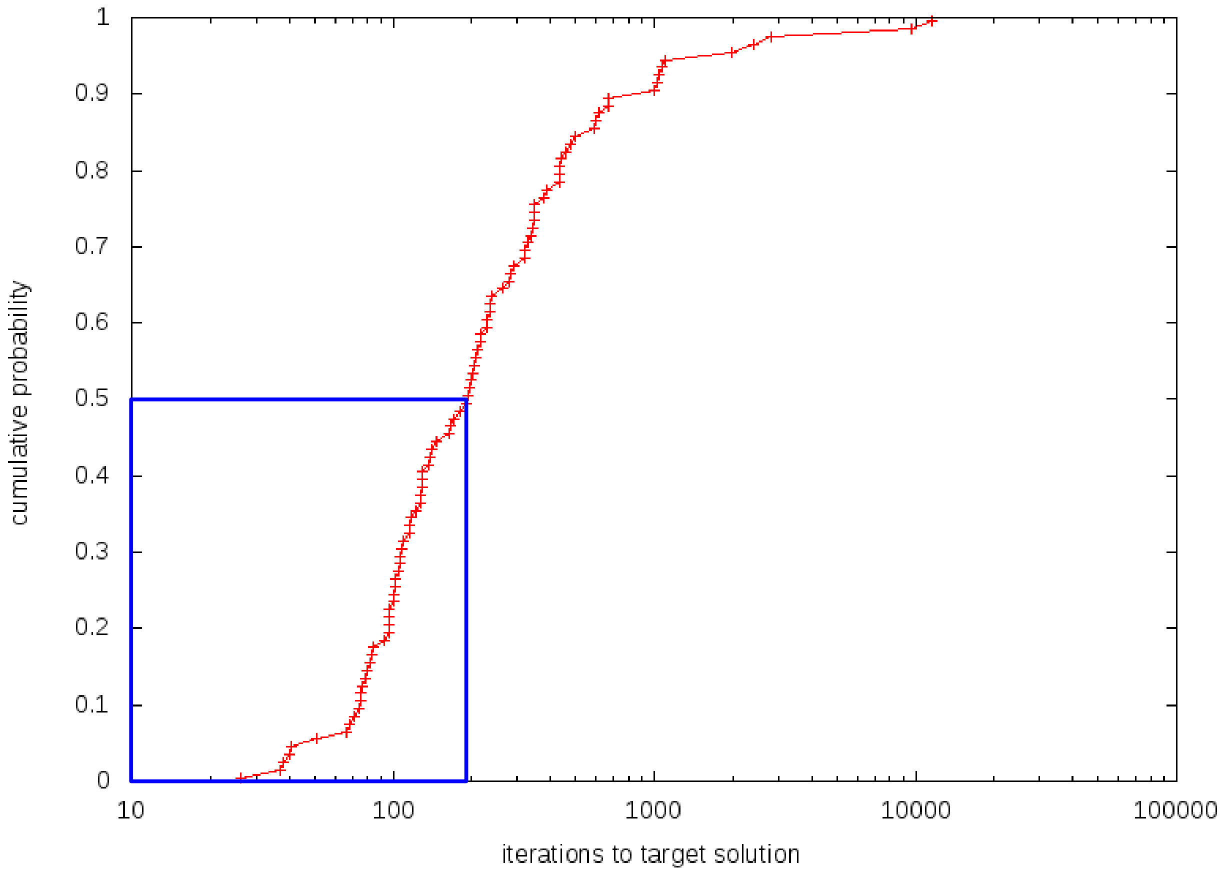
Randomized heuristic iteration count distribution: constructed by independently running the algorithm a number of times, each time stopping when the algorithm finds a solution at least as good as a given target.



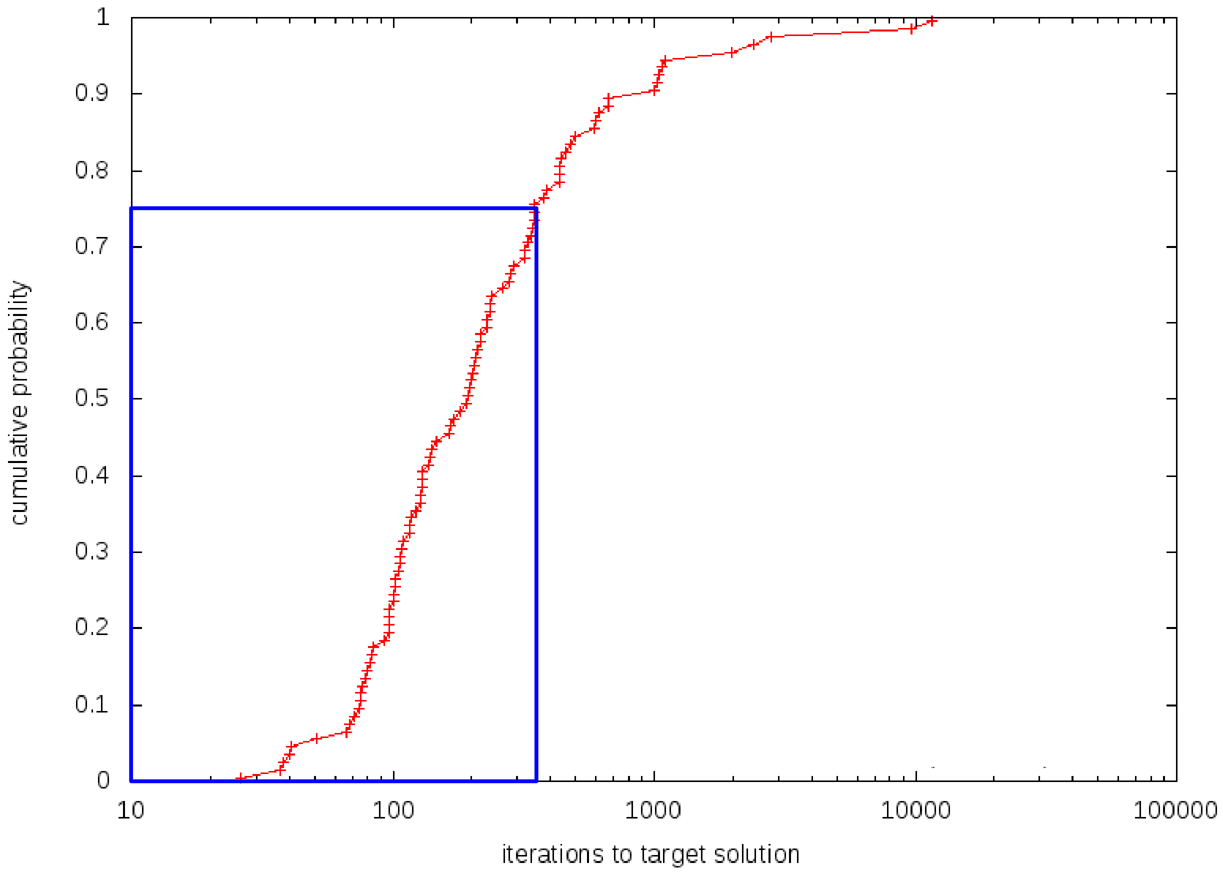
In most of the independent runs, the algorithm finds the target solution in relatively few iterations:



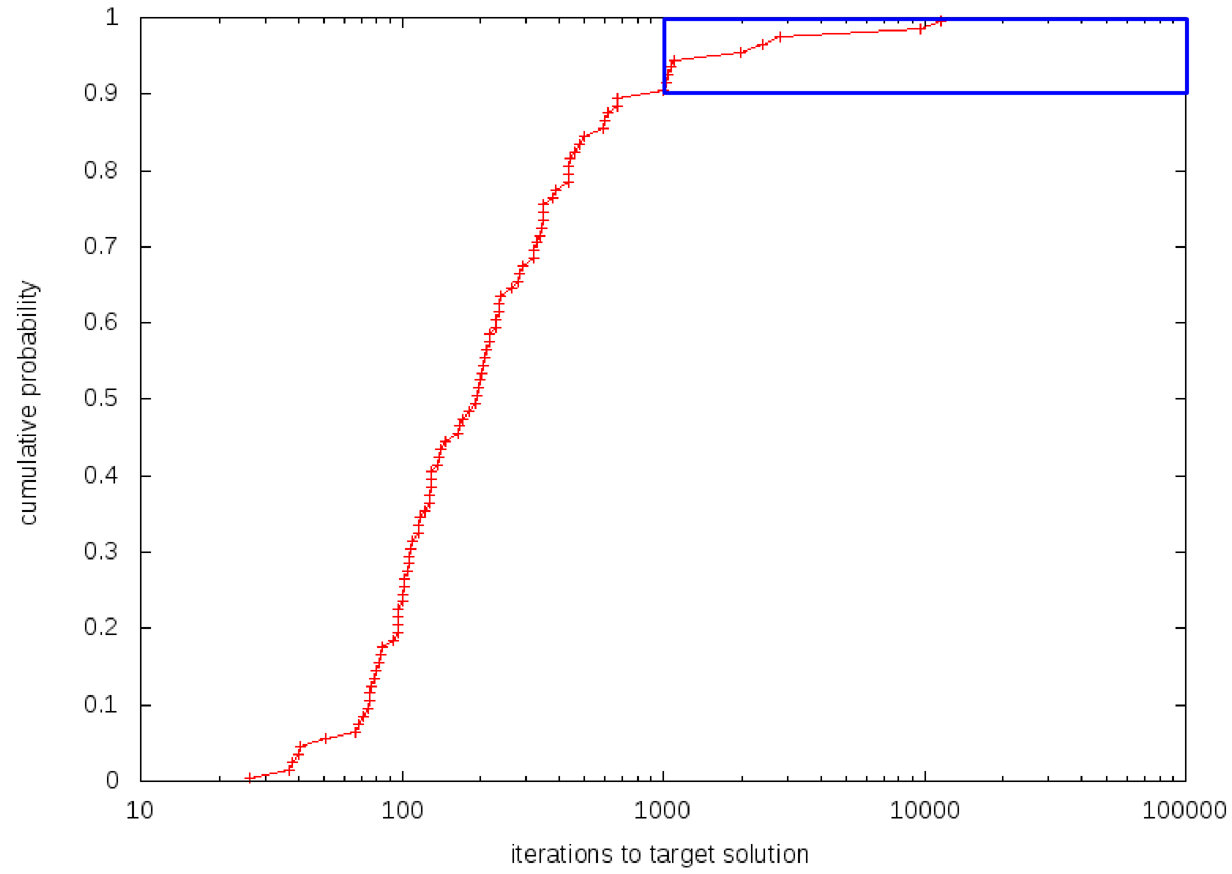
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 25% of the runs take fewer than 101 iterations



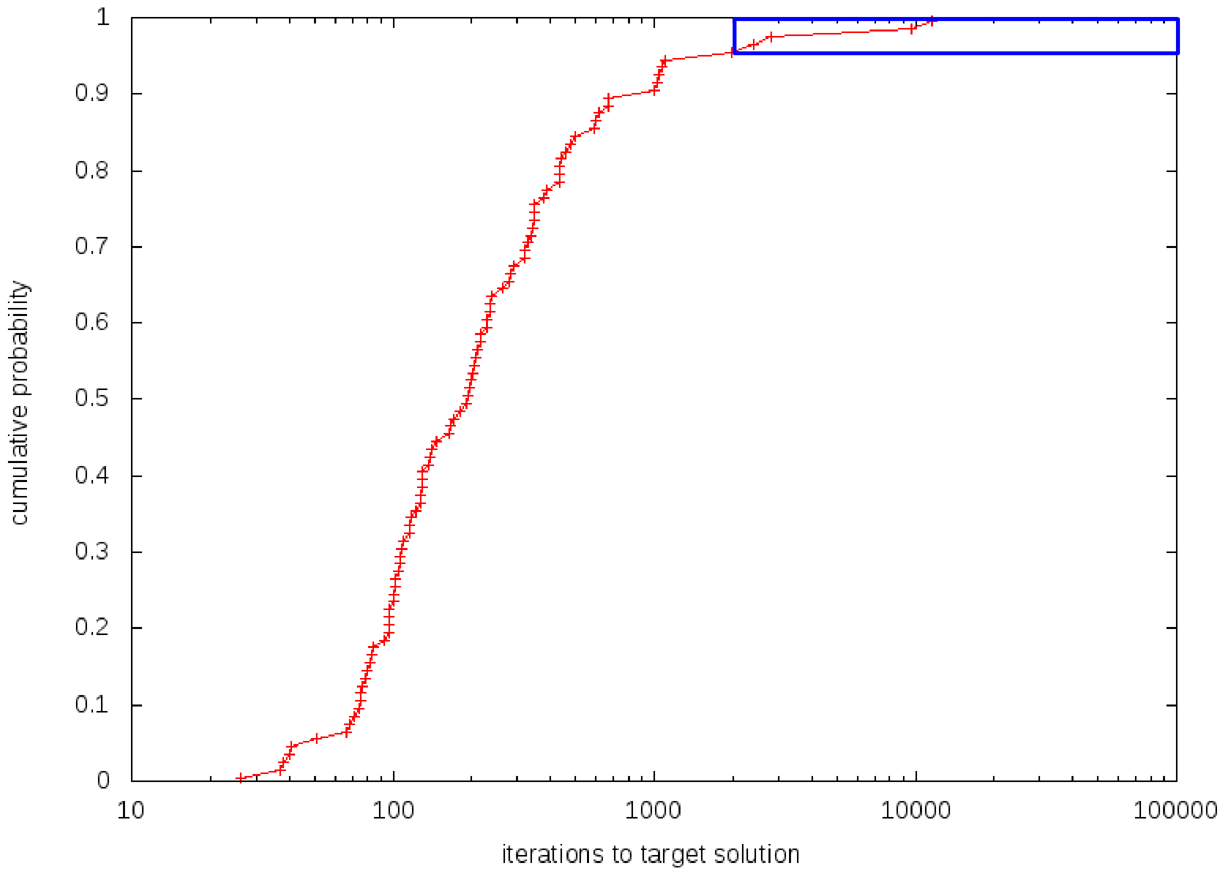
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 50% of the runs take fewer than 192 iterations



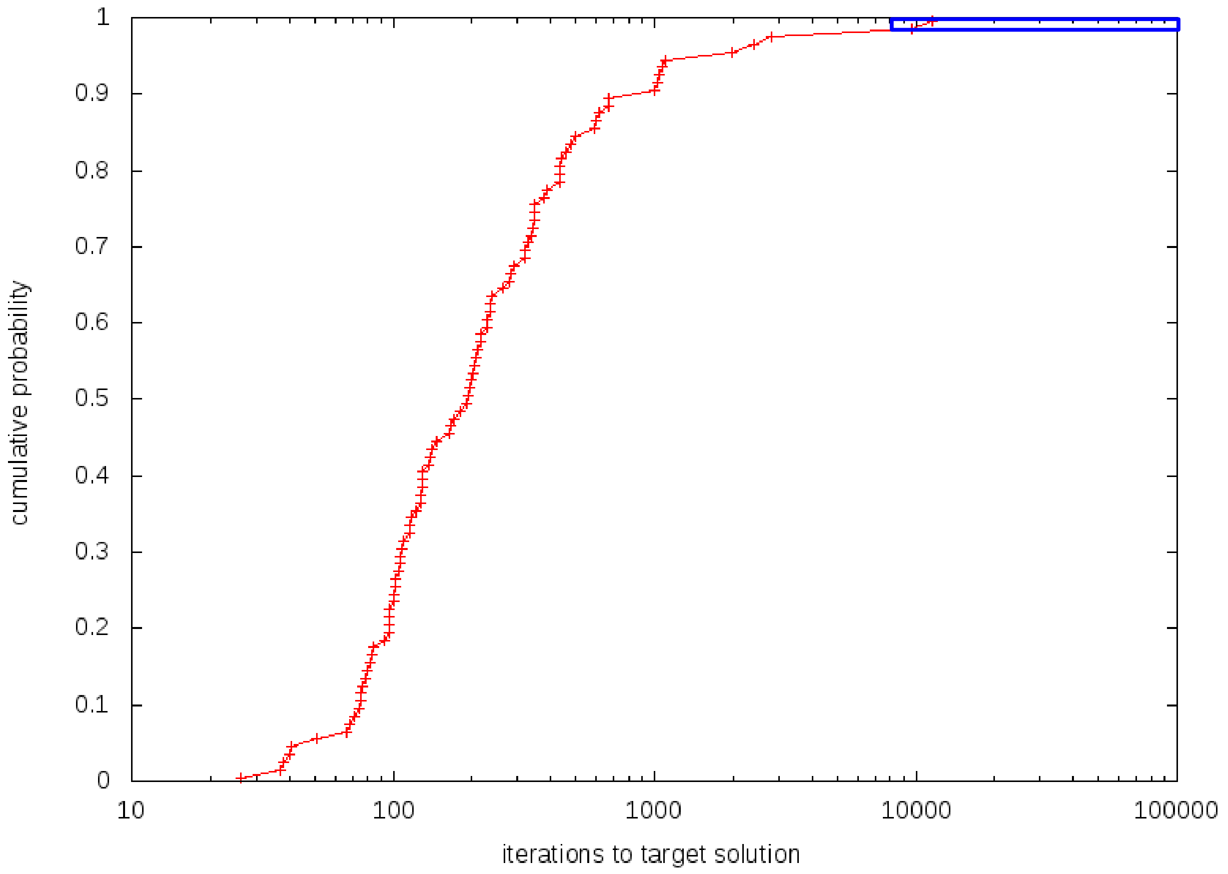
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 75% of the runs take fewer than 345 iterations



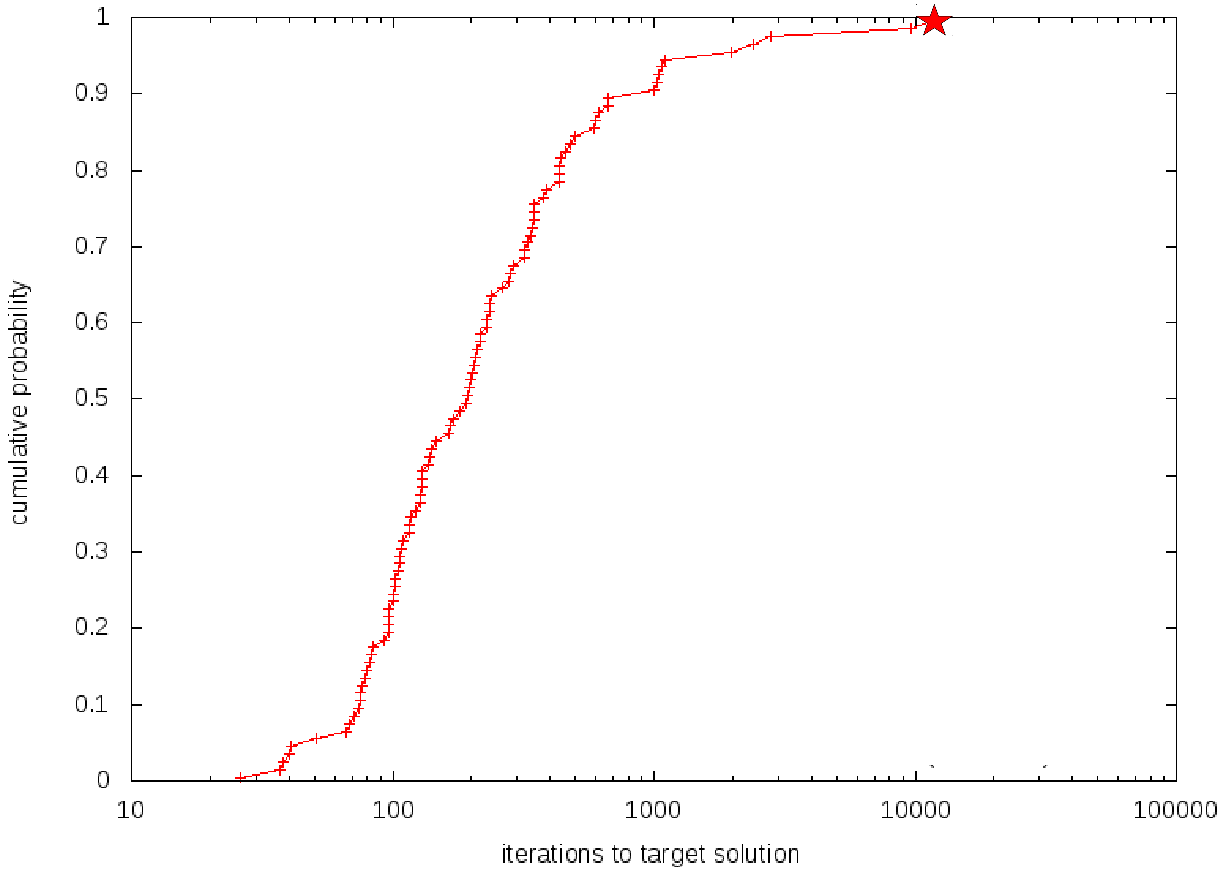
However, some runs take much longer: 10% of the runs take over 1000 iterations



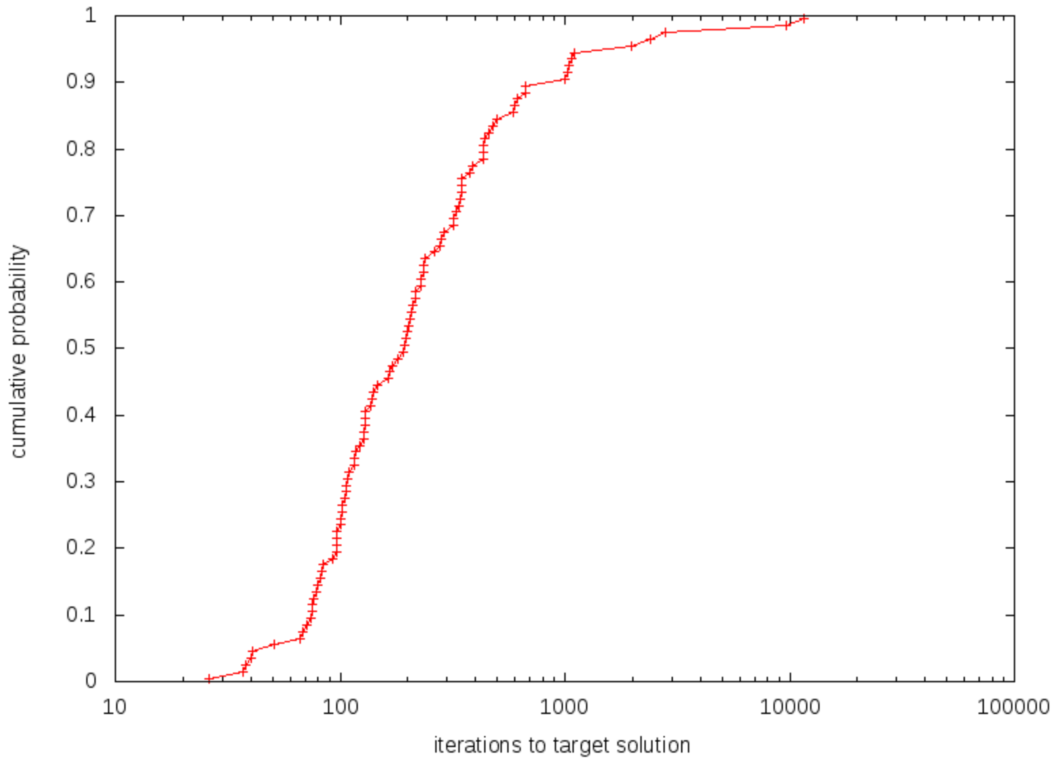
However, some runs take much longer: 5% of the runs take over 2000 iterations



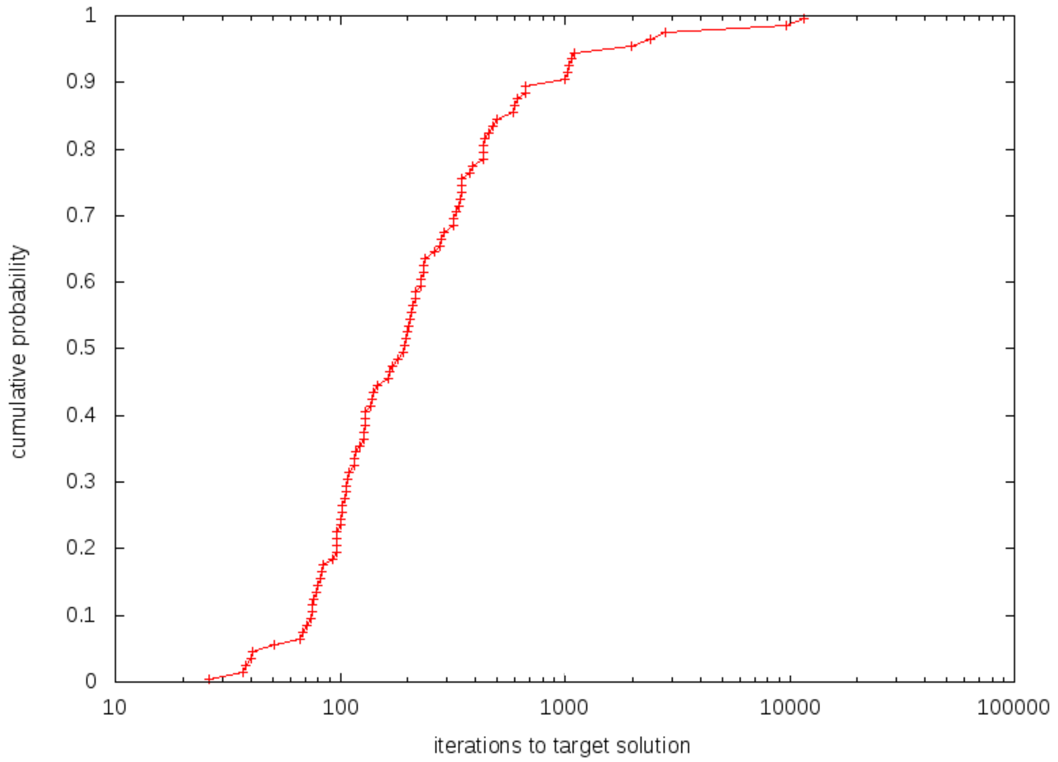
However, some runs take much longer: 2% of the runs take over 9715 iterations



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



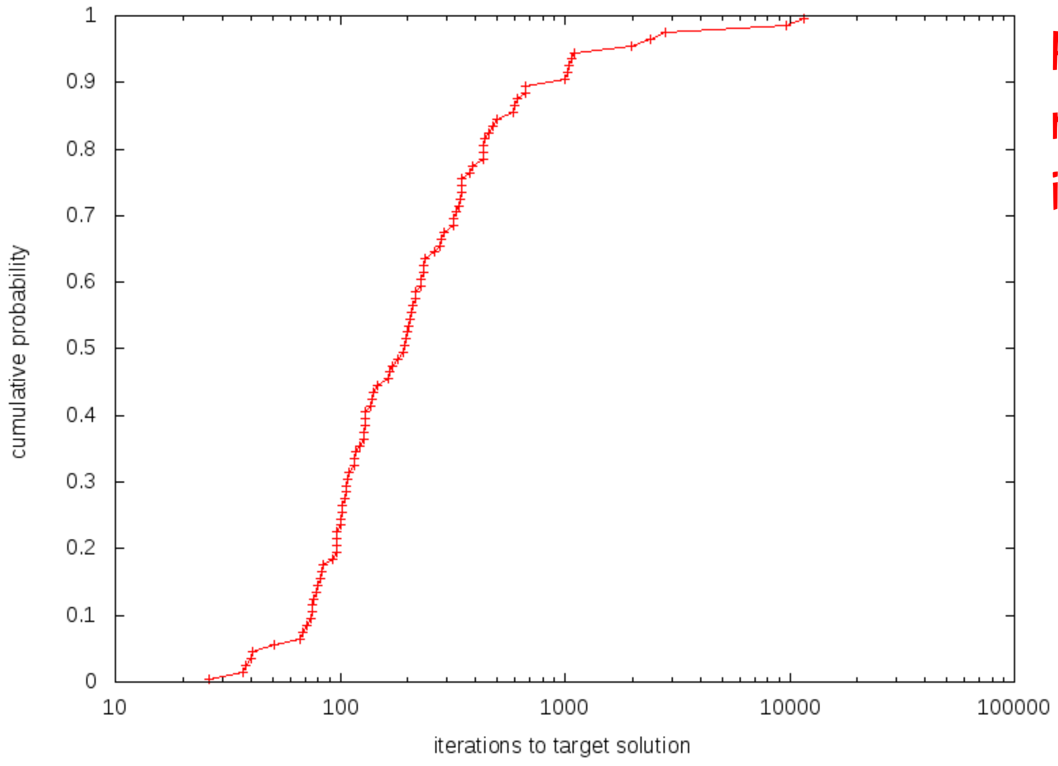
Probability that algorithm will take over 345 iterations: 25% = 1/4



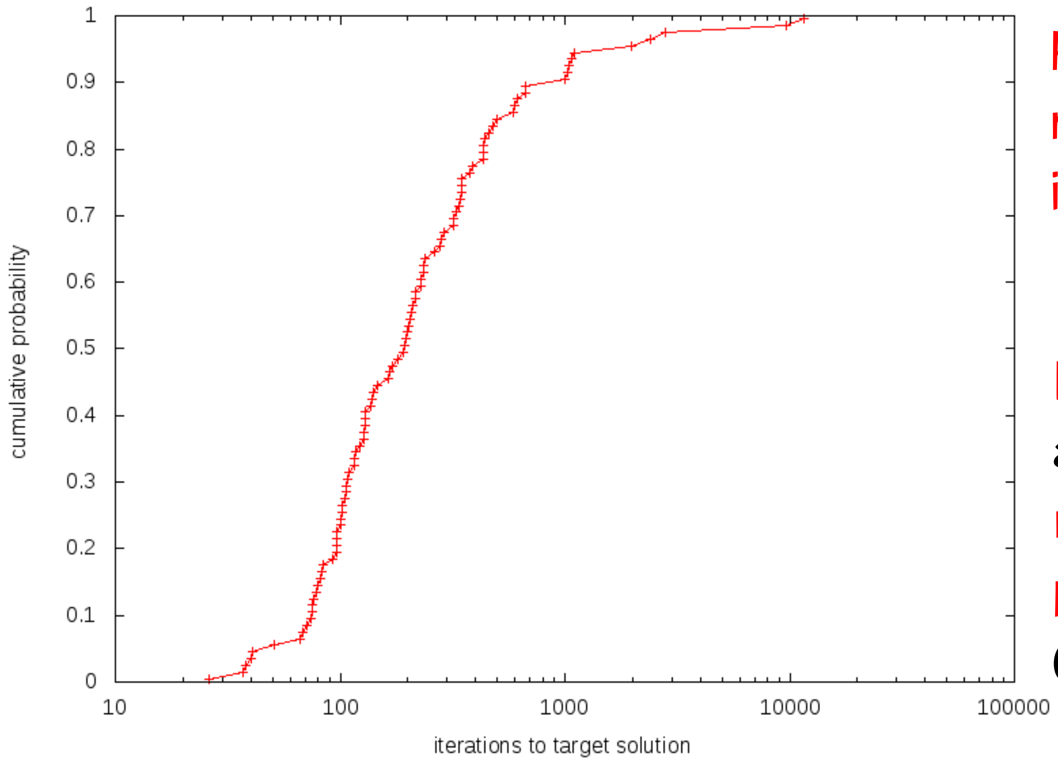
Probability that algorithm will take over 345 iterations: 25% = 1/4

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 = $1/4^2$

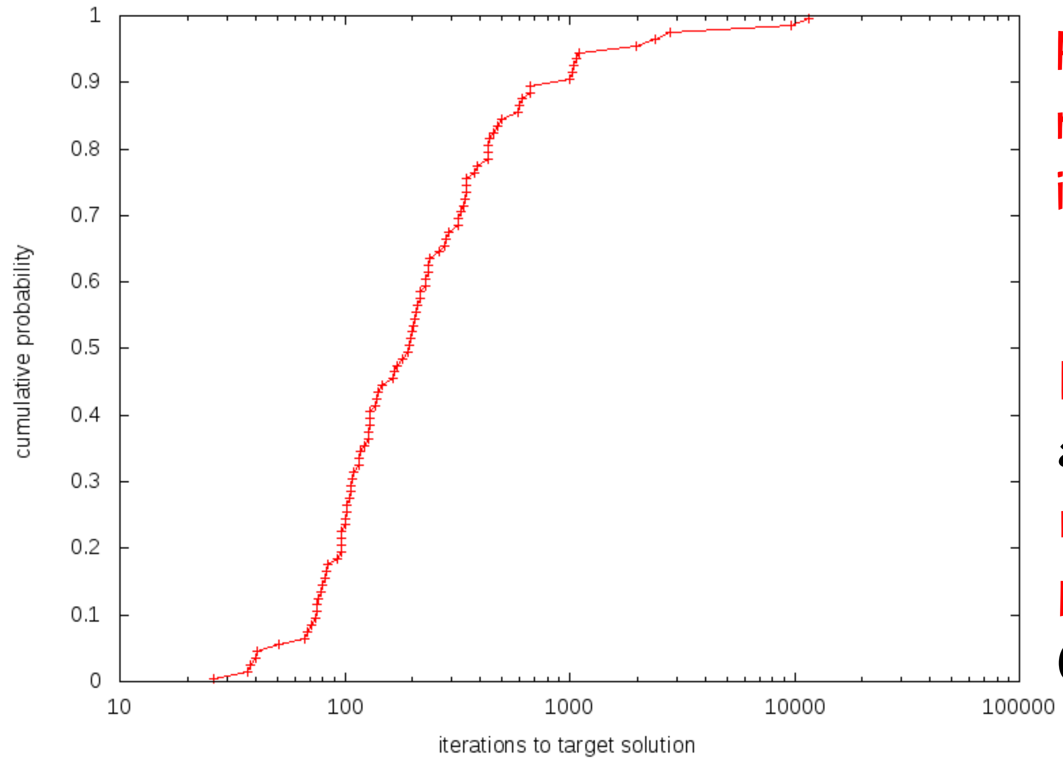


Probability that algorithm will still be running after K periods of 345 iterations: $1/4^K$



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 $S = \{\tau_1, \tau_2, \tau_3, \ldots\}$ 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.

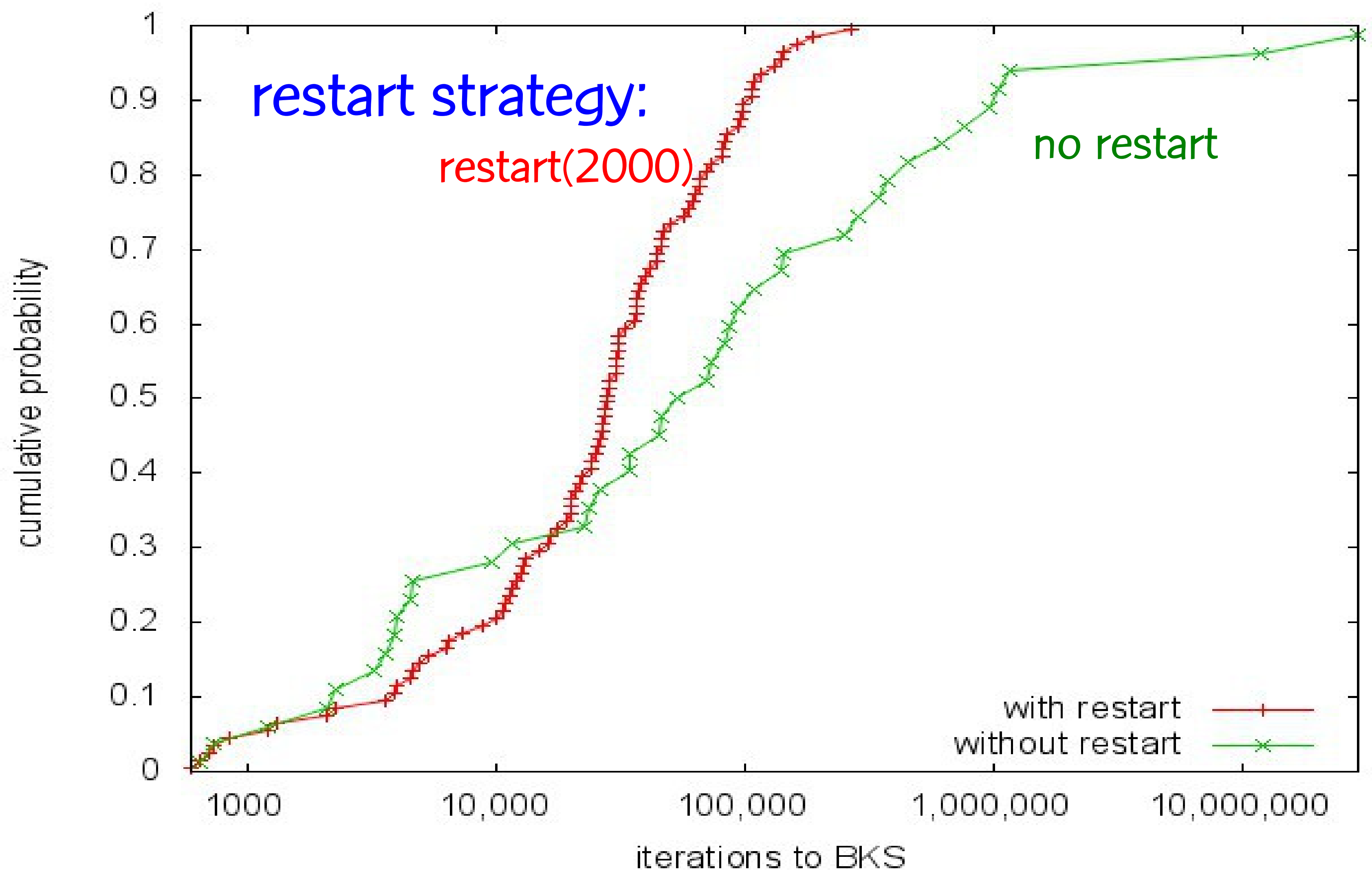
Restart strategy for BRKGA

- 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: Telecom application



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
- 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
- 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
- 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
- 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
- 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
- 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
- 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

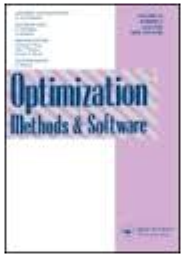
brkgaAPI: A C++ API for BRKGA

- 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

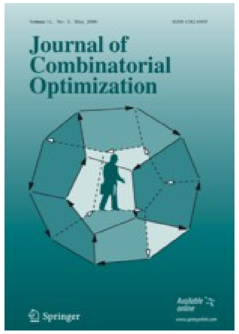


Paper: Rodrigo F. Toso and M.G.C.R.,
"A C++ Application Programming Interface for Biased Random-Key Genetic Algorithms,"
Optimization Methods & Software, vol. 30, pp. 81-93, 2015.

Software: http://mauricio.resende.info/src/brkgaAPI

An example BRKGA: Packing weighted rectangles

Reference



J.F. Gonçalves and R., "A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem," Journal of Combinatorial Optimization, vol. 22, pp. 180-201, 2011.

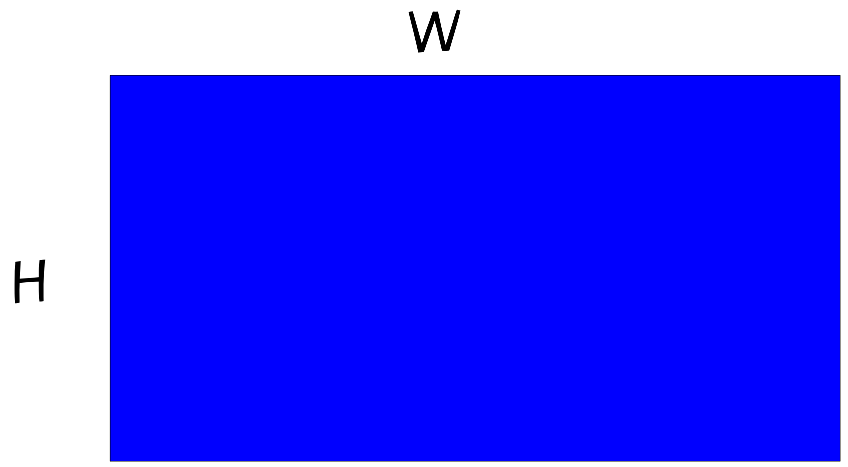
Tech report:
http://mauricio.resende.info/doc/pack2d.pdf

Constrained orthogonal packing

- Given a large planar stock rectangle (W, H) of width W and height H;

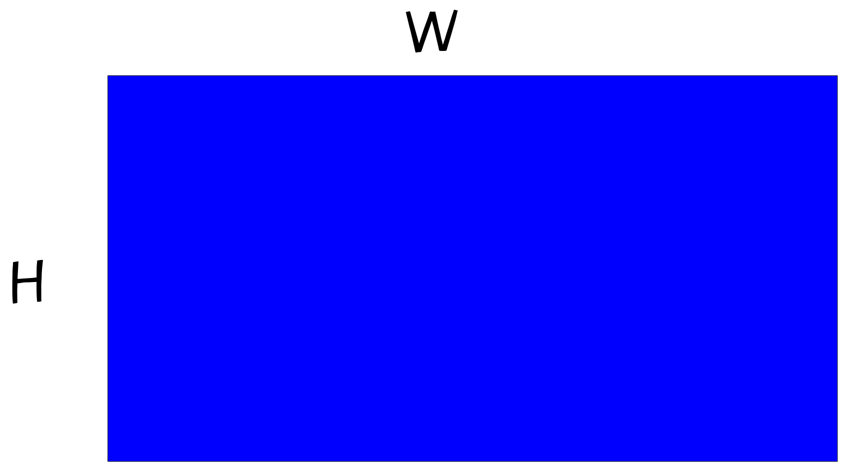
Constrained orthogonal packing

- Given a large planar stock rectangle (W, H) of width W and height H;



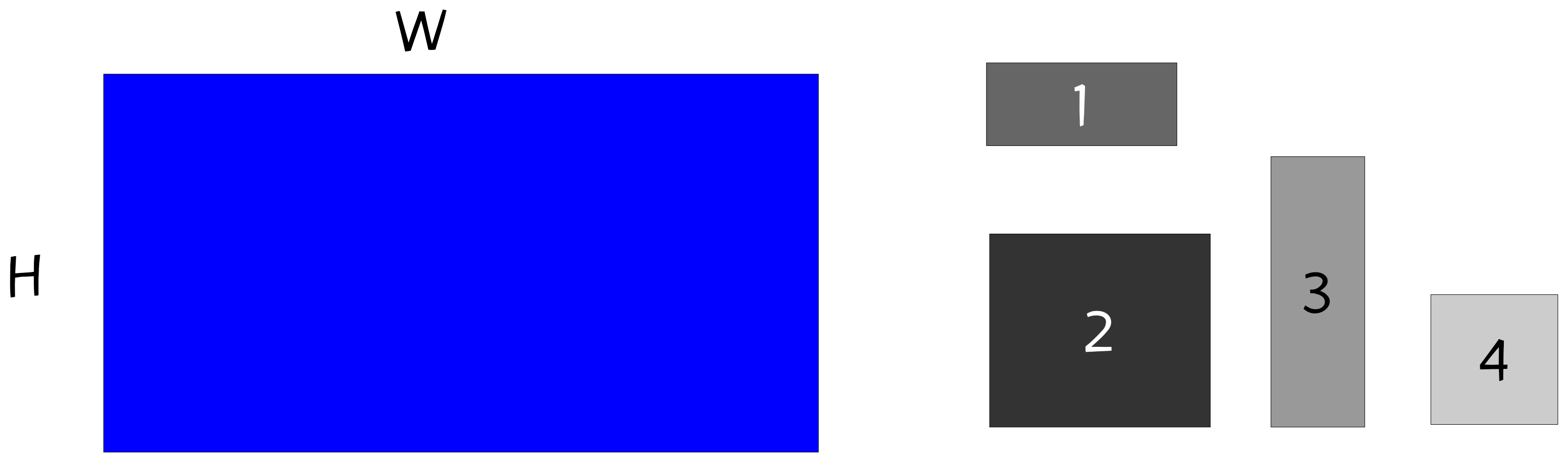
Constrained orthogonal packing

- Given a large planar stock rectangle (W, H) of width W and height H;
- Given N smaller rectangle types (w[i], h[i]), i = 1,...,N, each of width w[i], height h[i], and value v[i];



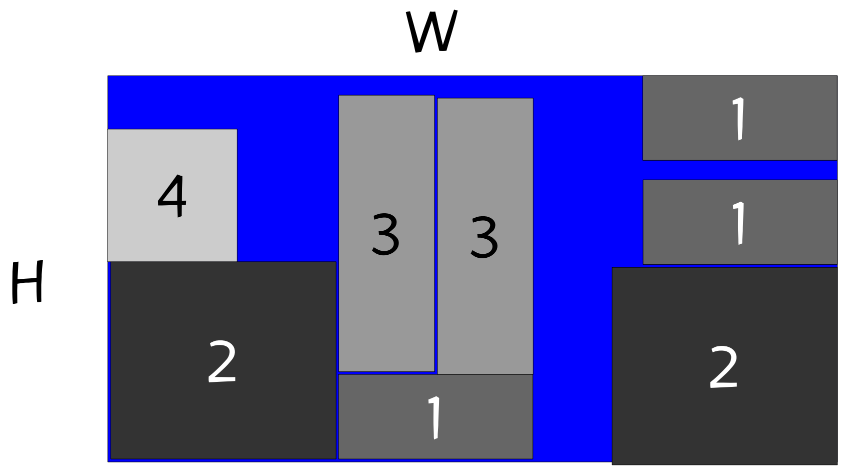
Constrained orthogonal packing

- Given a large planar stock rectangle (W, H) of width W and height H;
- Given N smaller rectangle types (w[i], h[i]), i = 1,...,N, each of width w[i], height h[i], and value v[i];



Constrained orthogonal packing

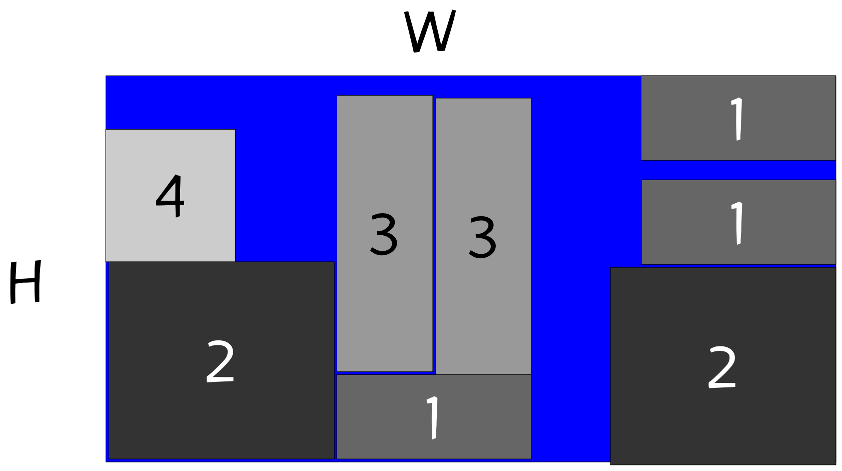
- r[i] rectangles of type i = 1, ..., N are to be packed in the large rectangle without overlap and such that their edges are parallel to the edges of the large rectangle;



Constrained orthogonal packing

- r[i] rectangles of type i = 1, ..., N are to be packed in the large rectangle without overlap and such that their edges are parallel to the edges of the large rectangle;
- For i = 1, ..., N, we require that:

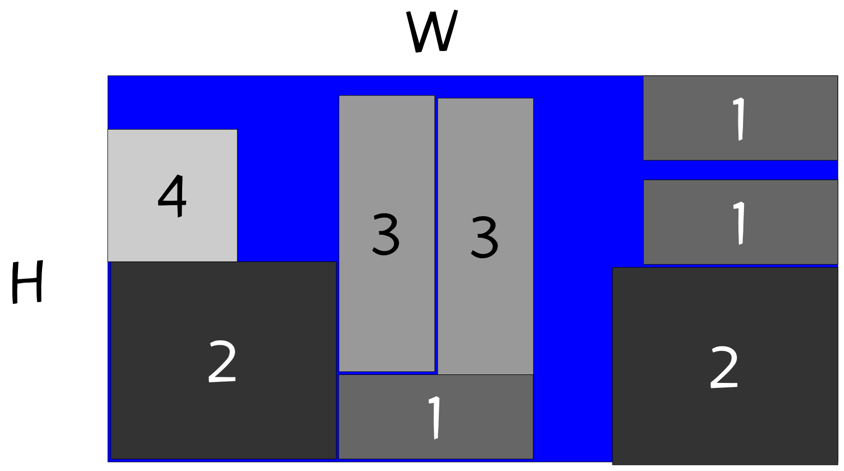
$$0 \le P[i] \le r[i] \le Q[i]$$



Constrained orthogonal packing

- r[i] rectangles of type i = 1, ..., N are to be packed in the large rectangle without overlap and such that their edges are parallel to the edges of the large rectangle;
- For i = 1, ..., N, we require that:

$$0 \leq P[i] \leq r[i] \leq Q[i]$$

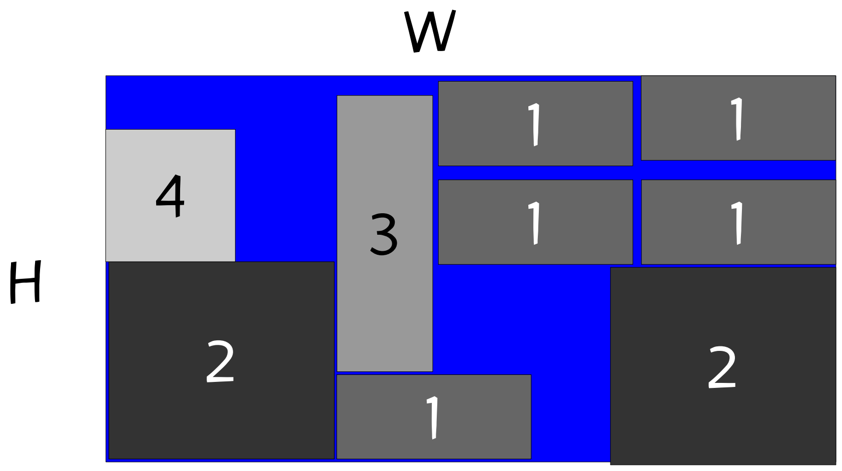


Suppose $5 \leq r[1] \leq 12$

Constrained orthogonal packing

- r[i] rectangles of type i = 1, ..., N are to be packed in the large rectangle without overlap and such that their edges are parallel to the edges of the large rectangle;
- For i = 1, ..., N, we require that:

$$0 \leq P[i] \leq r[i] \leq Q[i]$$

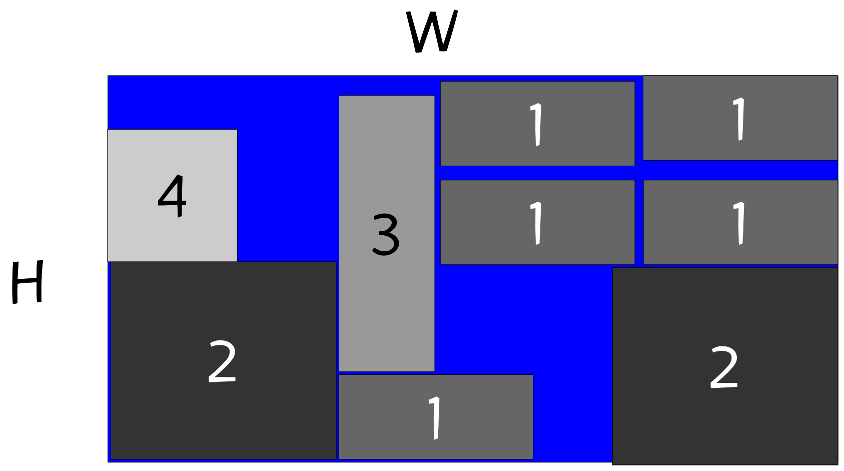


Suppose 5≤ r[1] ≤ 12

Objective

Among the many feasible packings, we want to find one that maximizes total value of packed rectangles:

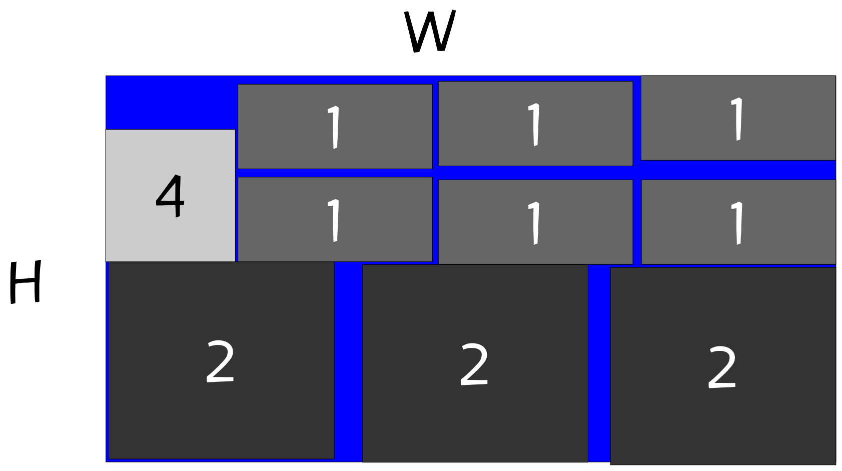
$$v[1]\,r[1] + v[2]\,r[2] + \cdots + v[N]\,r[N]$$



Objective

Among the many feasible packings, we want to find one that maximizes total value of packed rectangles:

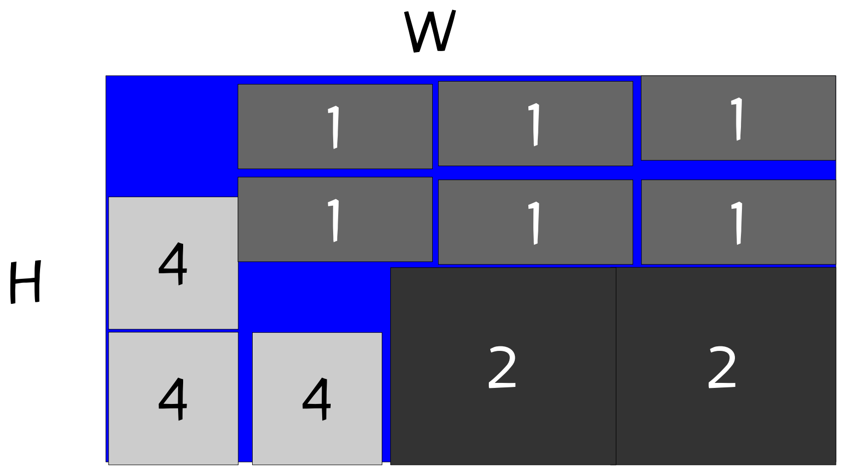
$$v[1]\, r[1] + v[2]\, r[2] + \cdots + v[N]\, r[N]$$



Objective

Among the many feasible packings, we want to find one that maximizes total value of packed rectangles:

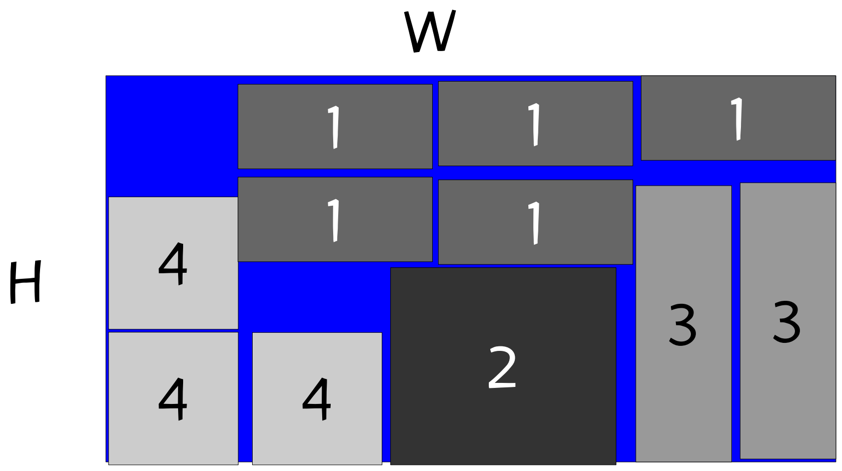
$$v[1]\ r[1] + v[2]\ r[2] + \cdots + v[N]\ r[N]$$



Objective

Among the many feasible packings, we want to find one that maximizes total value of packed rectangles:

$$v[1]\, r[1] + v[2]\, r[2] + \cdots + v[N]\, r[N]$$



Applications

Problem arises in several production processes, e.g.

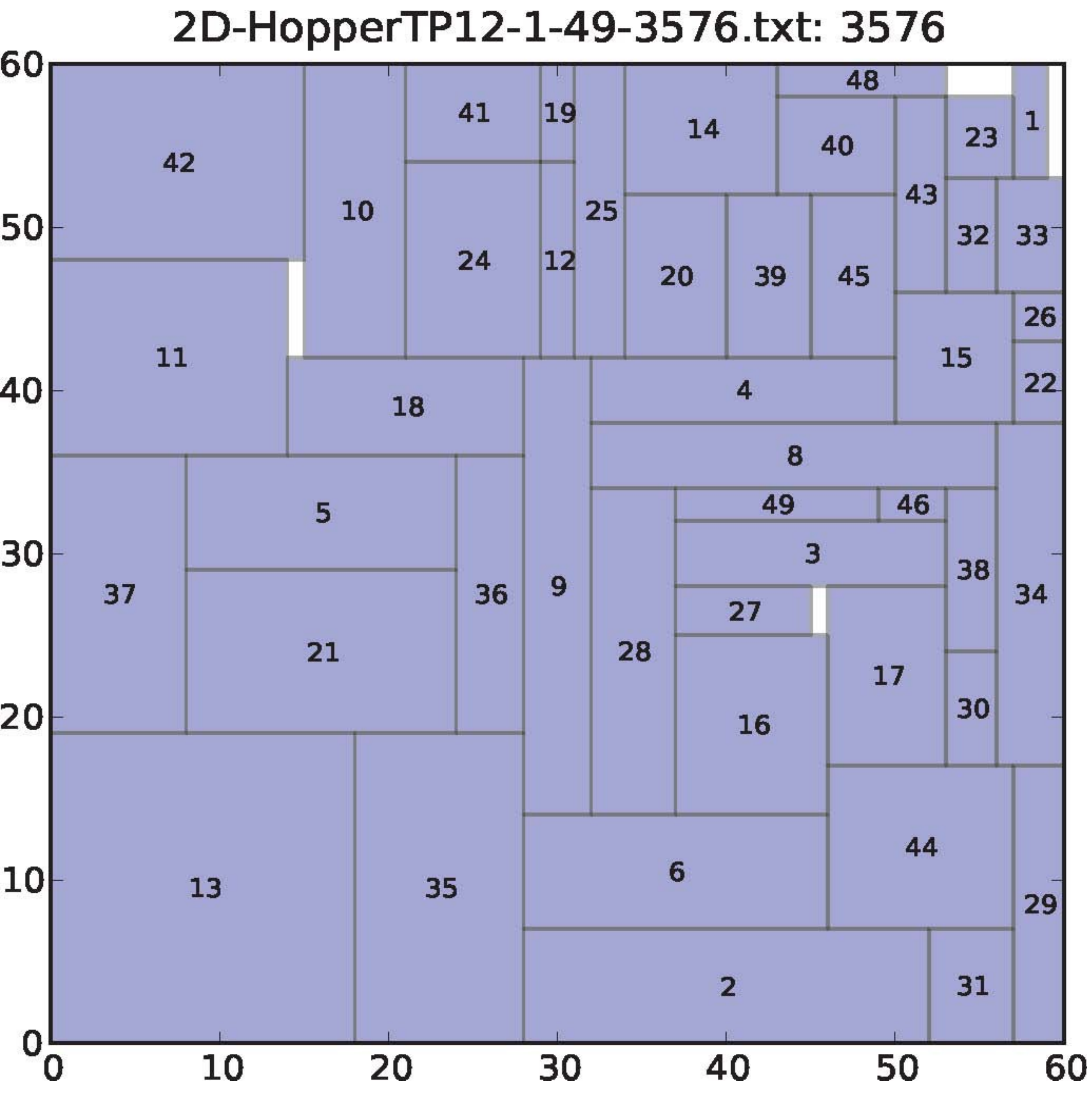
– Textile

– Glass

– Wood

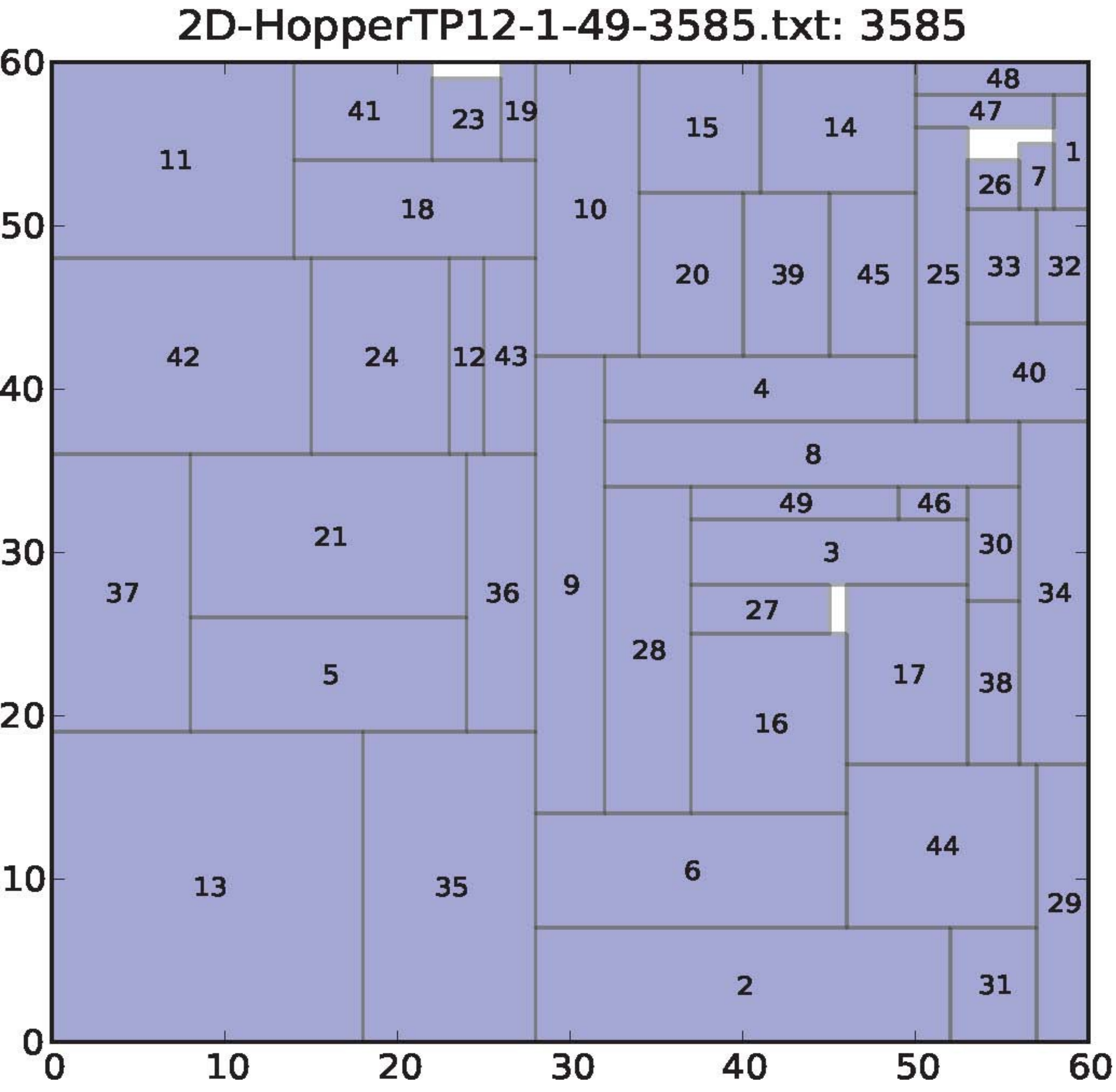
– Paper

where rectangular figures are cut from large rectangular sheets of materials.



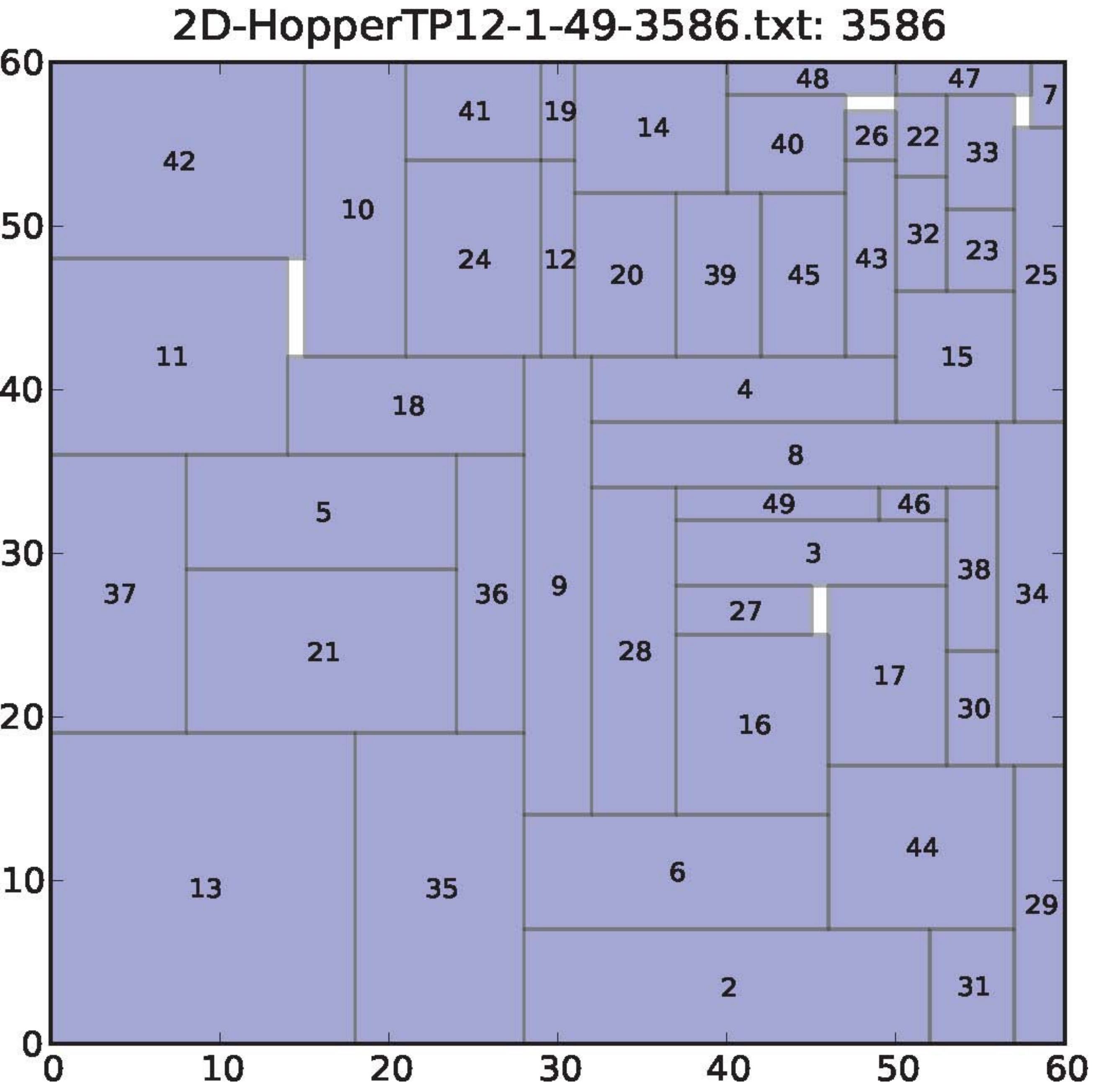
Hopper & Turton, 2001
Instance 4-1 60 x 60
Value: 3576

Previous best: 3580 by a Tabu Search heuristic (Alvarez-Valdes et al., 2007)



Hopper & Turton, 2001
Instance 4-2 60 x 60
Value: 3585

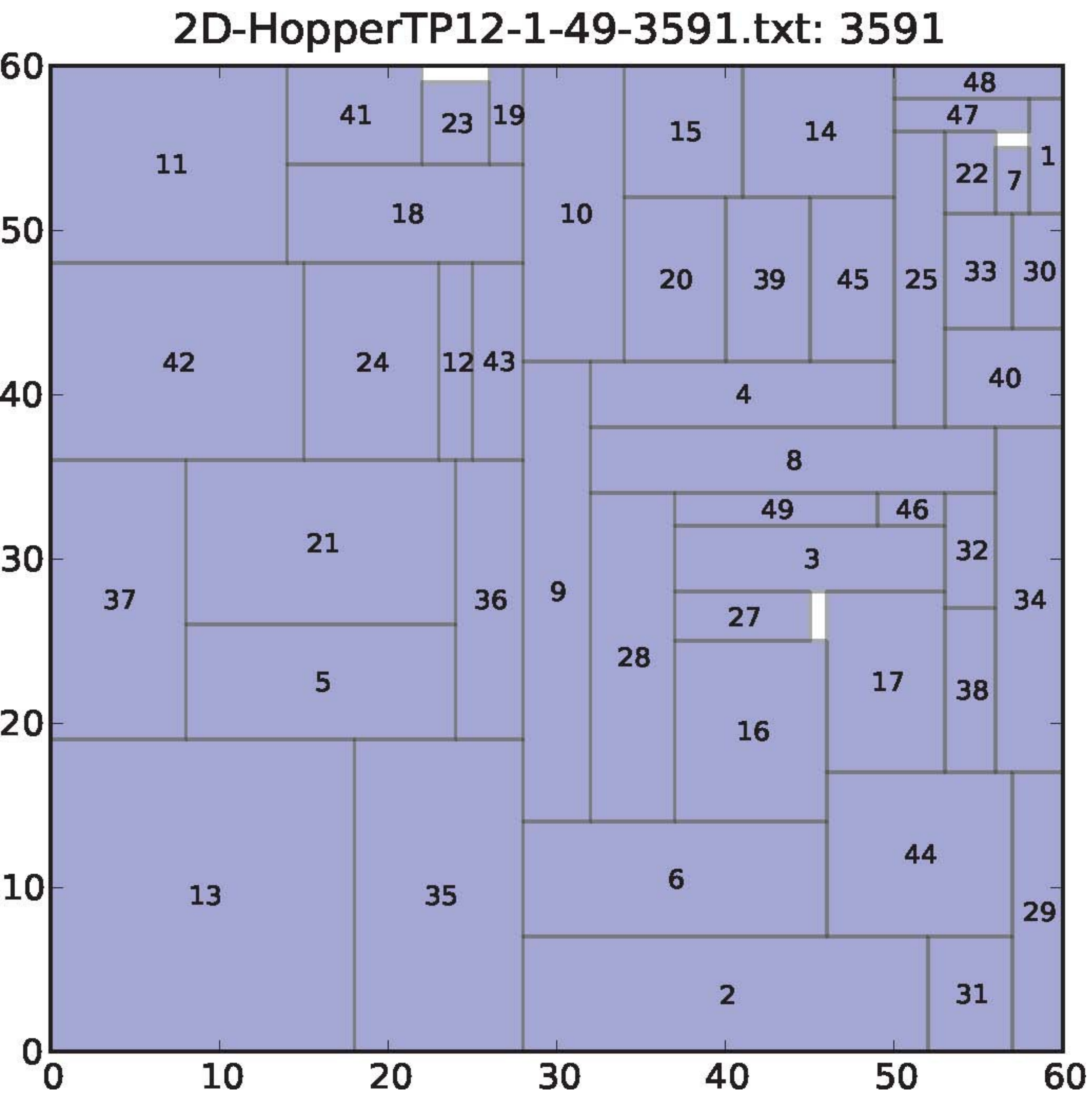
Previous best: 3580 by a Tabu Search heuristic (Alvarez-Valdes et al., 2007)



Hopper & Turton, 2001
Instance 4-2 60 x 60
Value: 3586

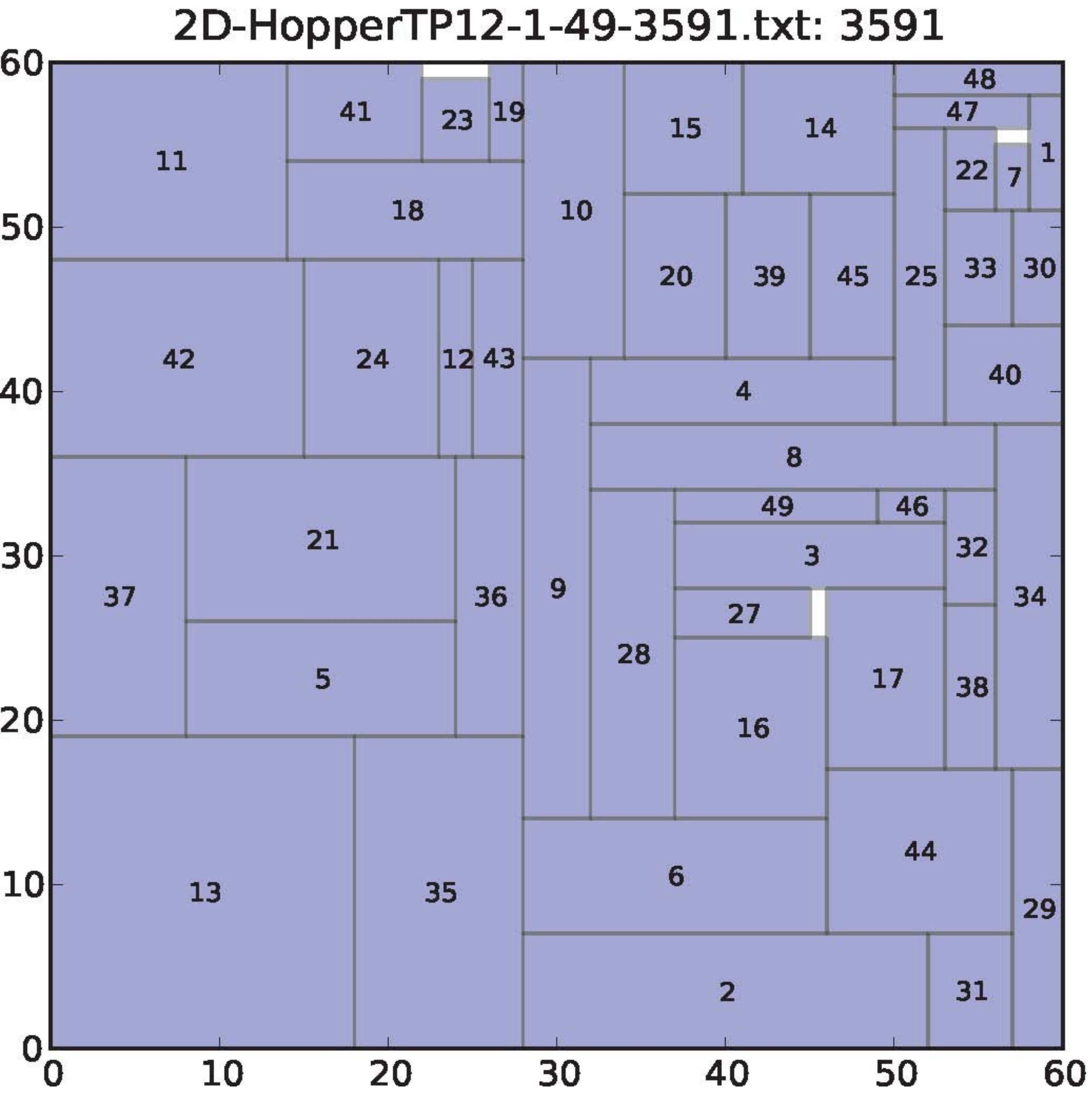
Previous best: 3580 by a Tabu Search heuristic (Alvarez-Valdes et al., 2007)

BRKGA



Hopper & Turton, 2001
Instance 4-2 60 x 60
Value: 3591

Previous best: 3580 by a Tabu Search heuristic (Alvarez-Valdes et al., 2007)



Hopper & Turton, 2001

Instance 4-2 60 x 60

Value: 3591

New best known solution!

Previous best: 3580 by a Tabu Search heuristic (Alvarez-Valdes et al., 2007)

BRKGA for constrained 2-dim orthogonal packing

Encoding

- Solutions are encoded as vectors X of
  $$2N' = 2 \{ Q[1] + Q[2] + \cdots + Q[N] \}$$
  random keys, where Q[i] is the maximum number of rectangles of type i (for i = 1, ..., N) that can be packed.
- $X = (X[1], ..., X[N'],\qquad X[N'+1], ..., X[2N'])$

Encoding

- Solutions are encoded as vectors X of

$$2N' = 2 \{ Q[1] + Q[2] + \cdots + Q[N] \}$$

  random keys, where Q[i] is the maximum number of rectangles of type i (for i = 1, ..., N) that can be packed.

- $X = (\underbrace{X[1], \ldots, X[N']}_{\text{Rectangle type packing sequence (RTPS)}}, \quad X[N'+1], \ldots, X[2N'])$

Encoding

- Solutions are encoded as vectors X of

$$2N' = 2 \{ Q[1] + Q[2] + \cdots + Q[N] \}$$

  random keys, where Q[i] is the maximum number of rectangles of type i (for i = 1, ..., N) that can be packed.

- $$X = (\underbrace{X[1], \ldots, X[N']}_{\text{Rectangle type packing sequence (RTPS)}}, \quad \underbrace{X[N'+1], \ldots, X[2N']}_{\text{Vector of placement procedures (VPP)}})$$

Decoding

- Simple heuristic to pack rectangles:
  - Make Q[i] copies of rectangle i, for i = 1, ..., N.
  - Order the N' = Q[1] + Q[2] + ⋯ + Q[N] rectangles in some way.
  - Process the rectangles in the above order. Place the rectangle in the stock rectangle according to one of the following heuristics: bottom-left (BL) or left-bottom (LB). If rectangle cannot be positioned, discard it and go on to the next rectangle in the order.

Decoding

- Simple heuristic to pack rectangles:
  - Make Q[i] copies of rectangle i, for i = 1, ..., N.
  - Order the N' = Q[1] + Q[2] + ··· + Q[N] rectangles in some way. **Sort first N' keys of X to obtain order.**
  - Process the rectangles in the above order. Place the rectangle in the stock rectangle according to one of the following heuristics: bottom-left (BL) or left-bottom (LB). If rectangle cannot be positioned, discard it and go on to the next rectangle in the order.

Decoding

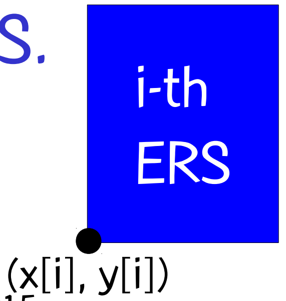
- Simple heuristic to pack rectangles:
  - Make Q[i] copies of rectangle i, for i = 1, ..., N.
  - Order the N' = Q[1] + Q[2] + ··· + Q[N] rectangles in some way. **Sort first N' keys of X to obtain order.**
  - Process the rectangles in the above order. Place the rectangle in the stock rectangle according to one of the following heuristics: bottom-left (BL) or left-bottom (LB). If rectangle cannot be positioned, discard it and go on to the next rectangle in the order. **Use the last N' keys of X to determine which heuristic to use. If k[N'+i] > 0.5 use LB, else use BL.**

Decoding

- A maximal empty rectangular space (ERS) is an empty rectangular space not contained in any other ERS.
- ERSs are generated and updated using the Difference Process of Lai and Chan (1997).
- When placing a rectangle, we limit ourselves only to maximal ERSs. We order all the maximal ERSs and place the rectangle in the first maximal ERS in which it fits.
- Let (x[i], y[i]) be the coordinates of the bottom left corner of the i-th ERS.

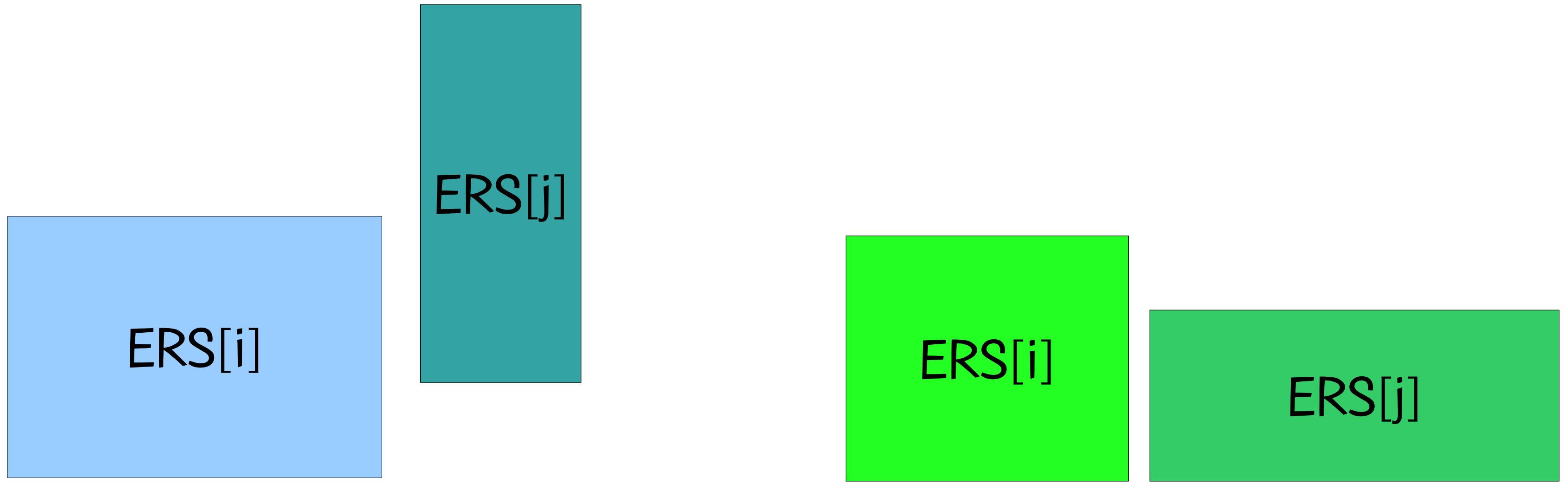
Decoding

- A maximal empty rectangular space (ERS) is an empty rectangular space not contained in any other ERS.
- ERSs are generated and updated using the Difference Process of Lai and Chan (1997).
- When placing a rectangle, we limit ourselves only to maximal ERSs. We order all the maximal ERSs and place the rectangle in the first maximal ERS in which it fits.
- Let (x[i], y[i]) be the coordinates of the bottom left corner of the i-th ERS.

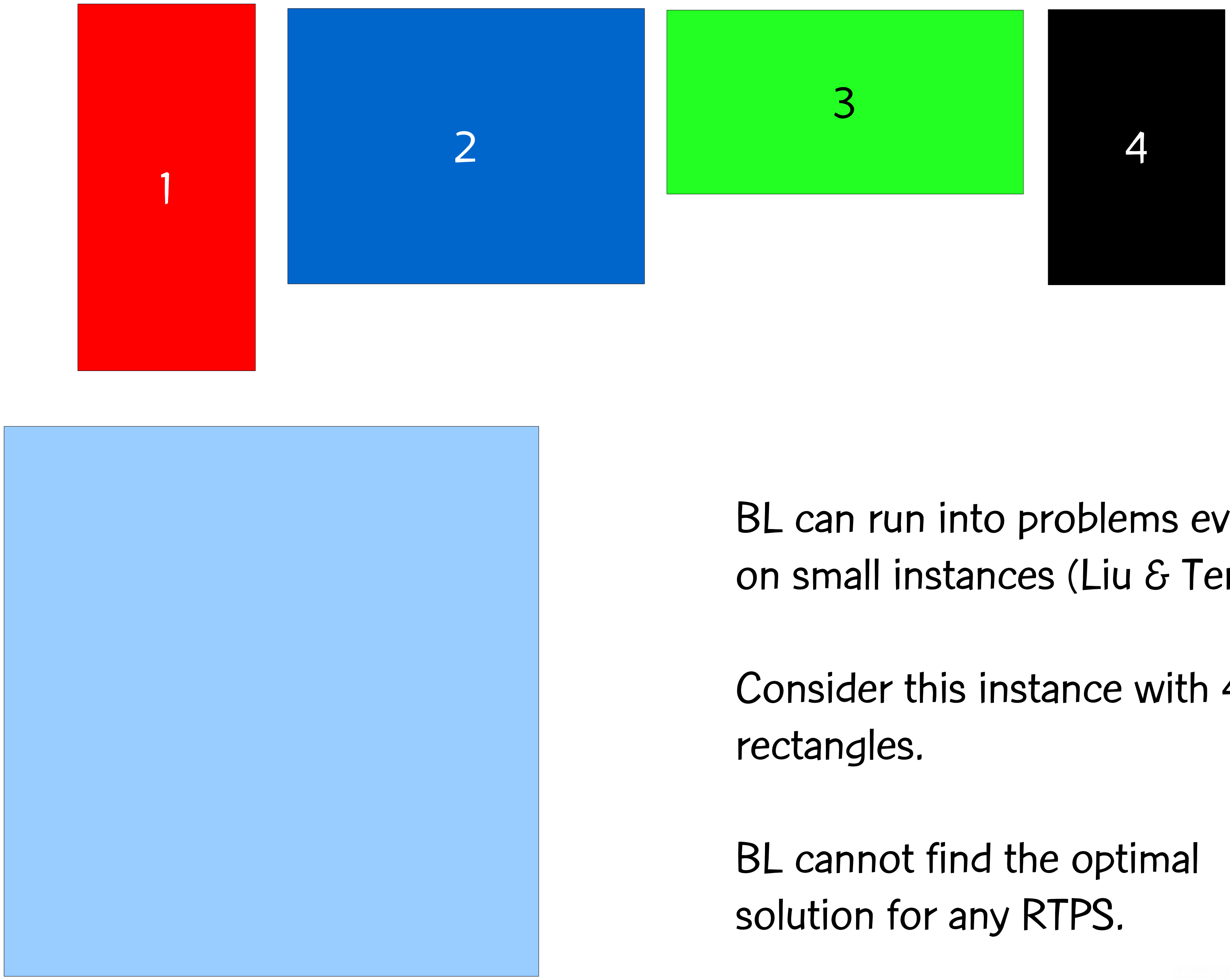


Decoding

- If BL is used, ERSs are ordered such that ERS[i] < ERS[j] if y[i] < y[j] or y[i] = y[j] and x[i] < x[j].



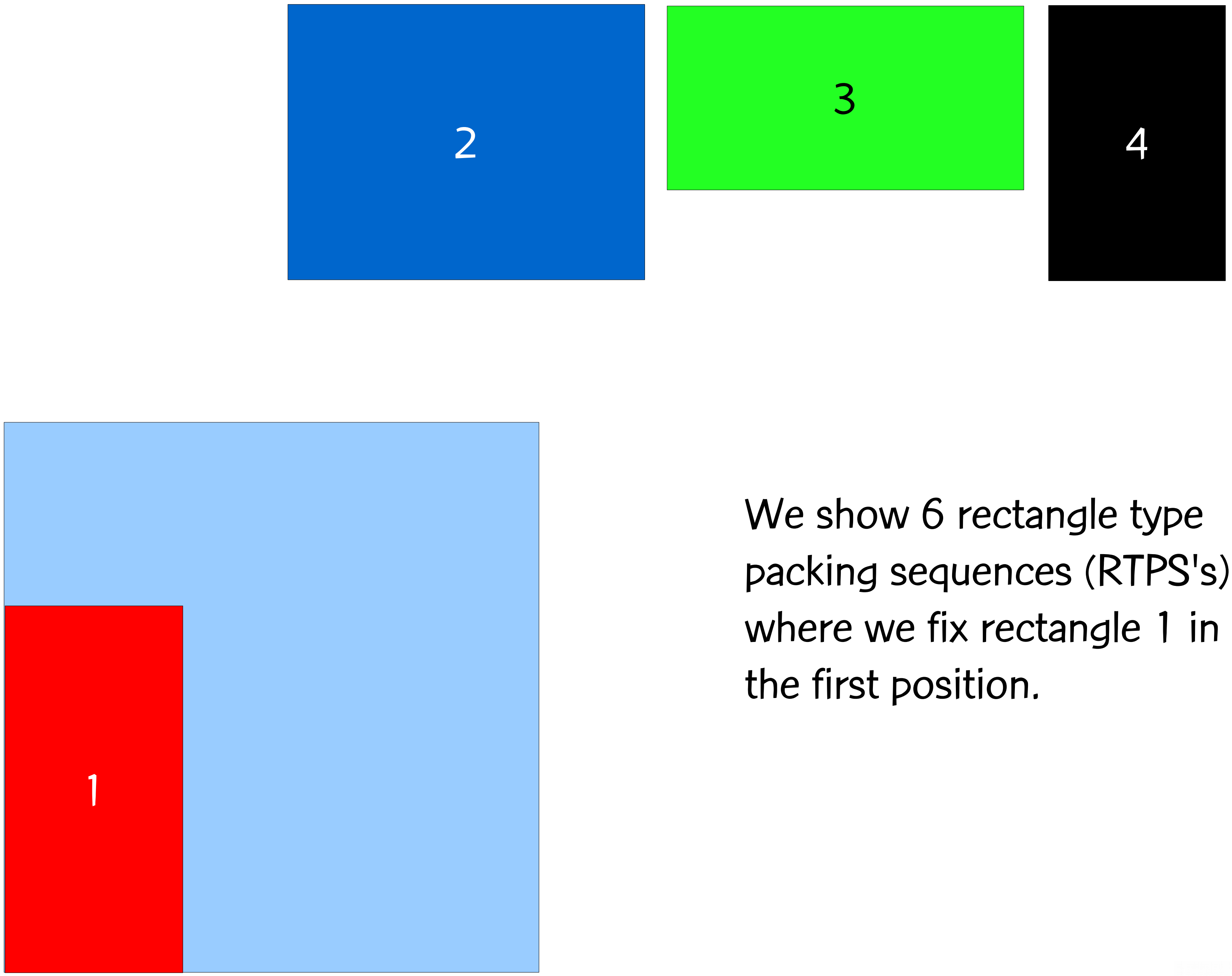
ERS[i] < ERS[j]



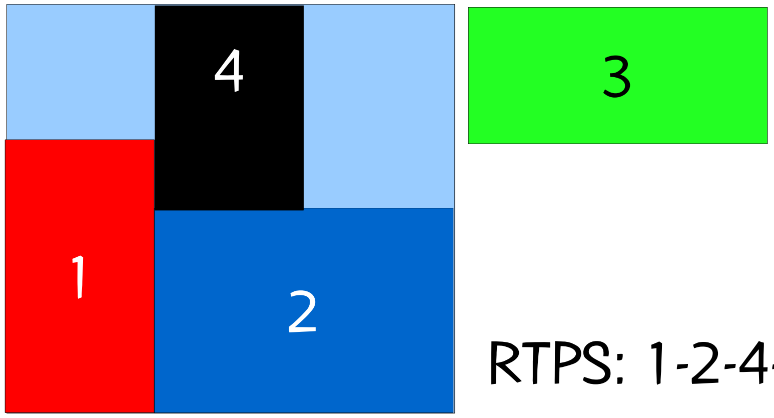
BL can run into problems even on small instances (Liu & Teng, 1999).

Consider this instance with 4 rectangles.

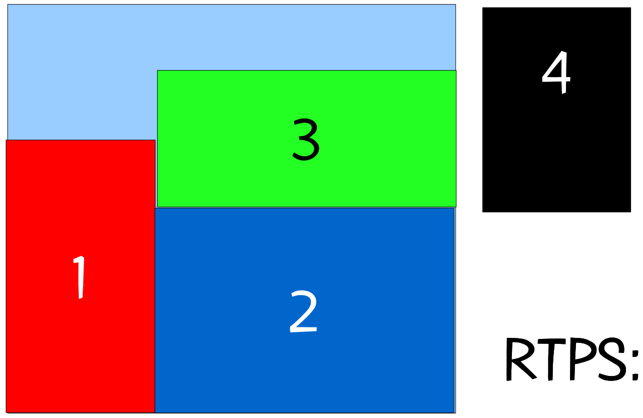
BL cannot find the optimal solution for any RTPS.



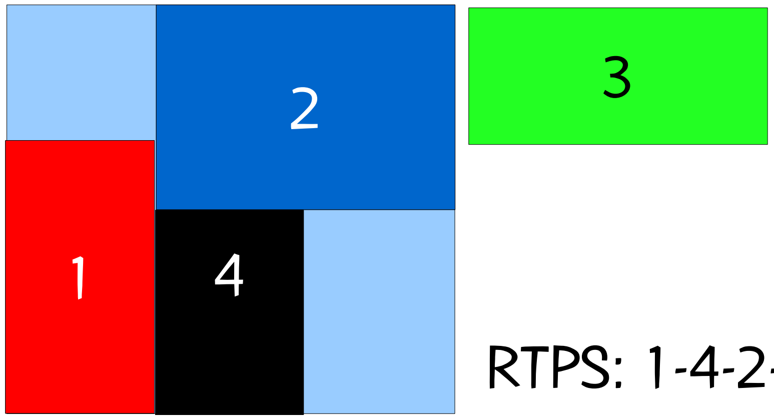
We show 6 rectangle type packing sequences (RTPS's) where we fix rectangle 1 in the first position.



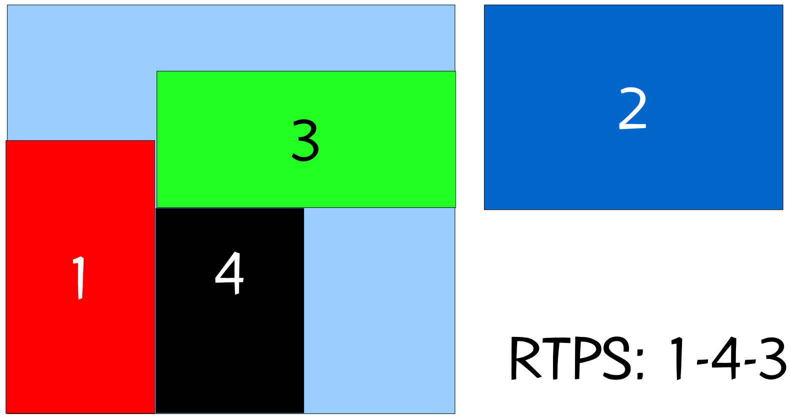
RTPS: 1-2-4-3



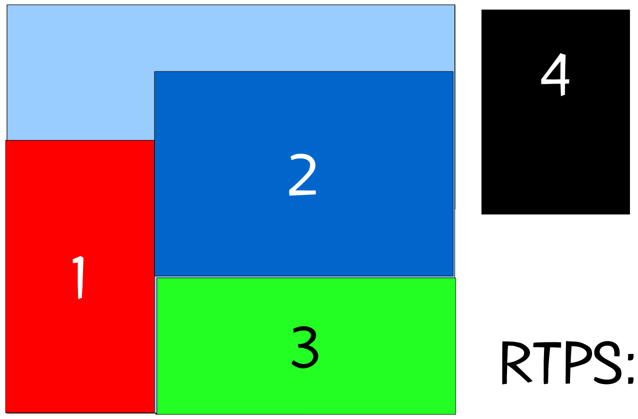
RTPS: 1-2-3-4



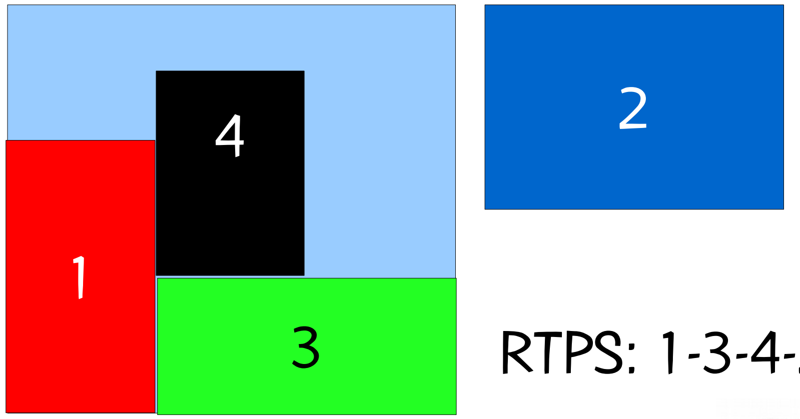
RTPS: 1-4-2-3



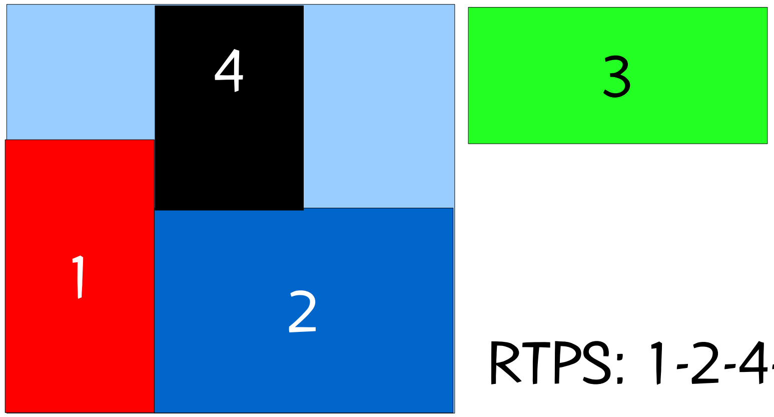
RTPS: 1-4-3-2



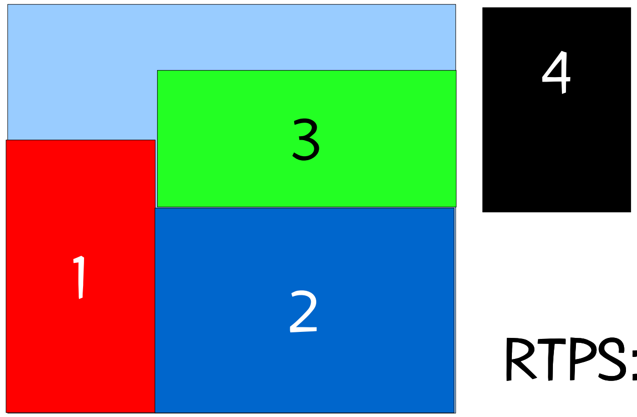
RTPS: 1-3-2-4



RTPS: 1-3-4-2

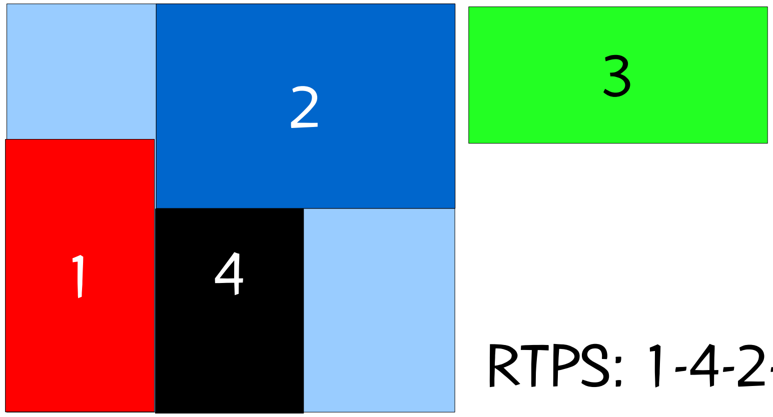


RTPS: 1-2-4-3

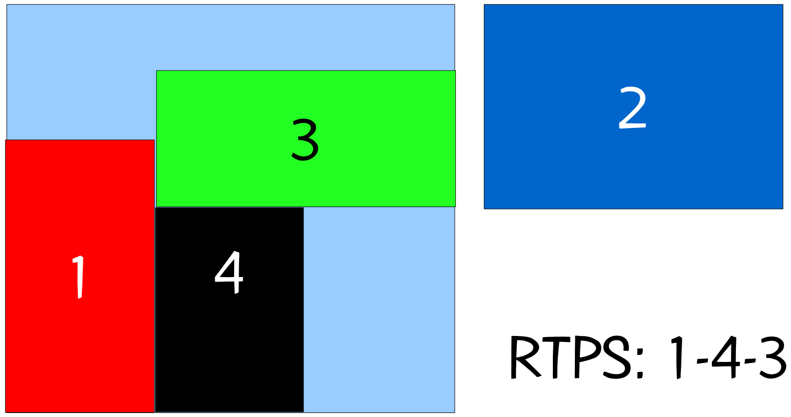


RTPS: 1-2-3-4

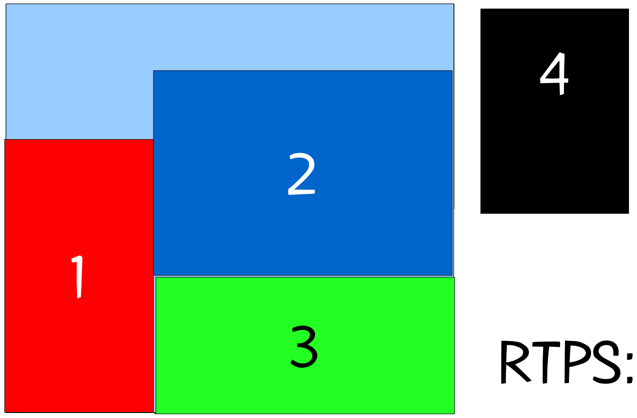
Similar infeasibilities are observed if 2, 3, or 4 is the first rectangle in the RTPS.



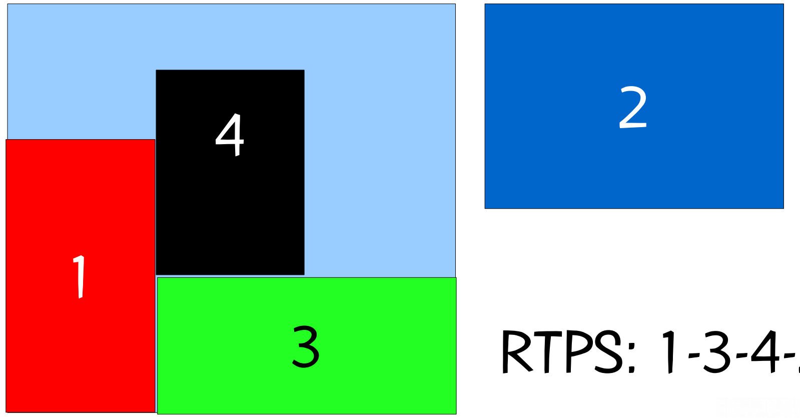
RTPS: 1-4-2-3



RTPS: 1-4-3-2



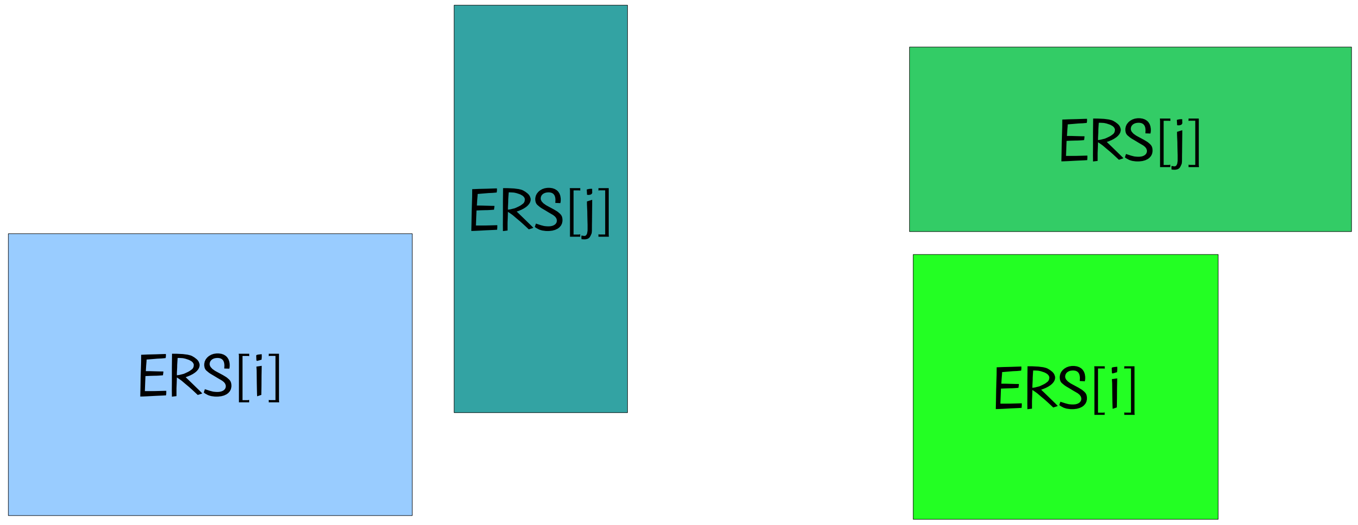
RTPS: 1-3-2-4



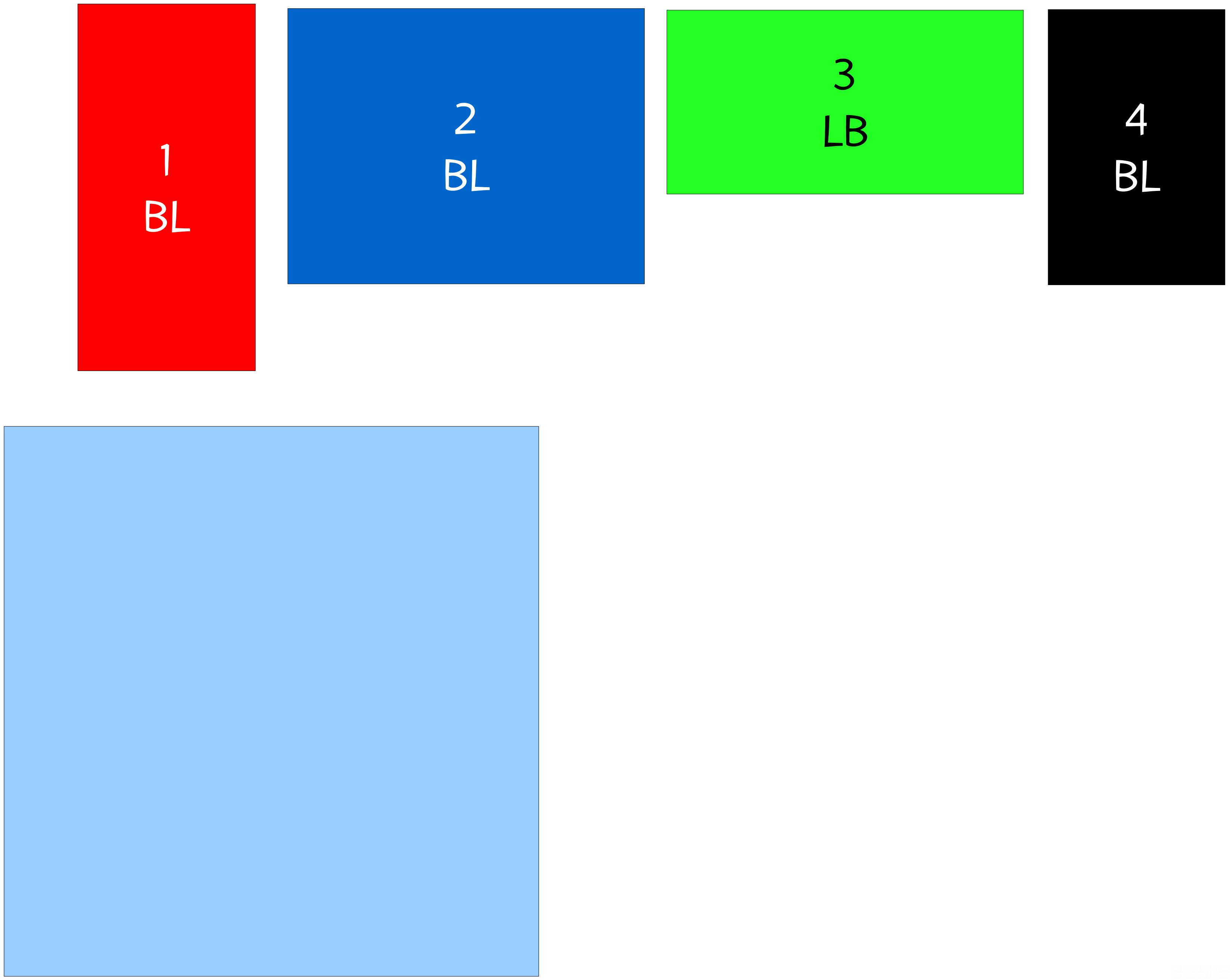
RTPS: 1-3-4-2

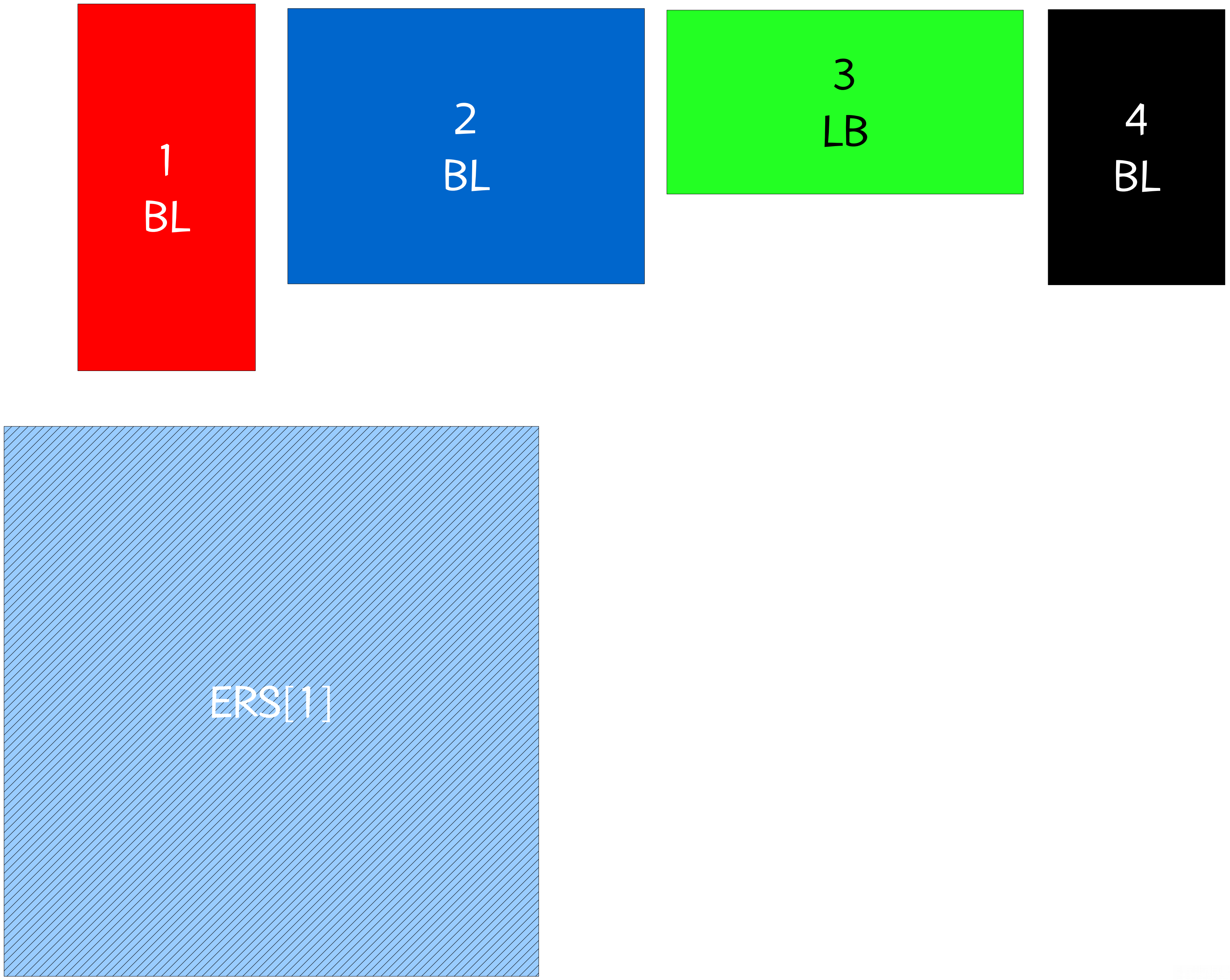
Decoding

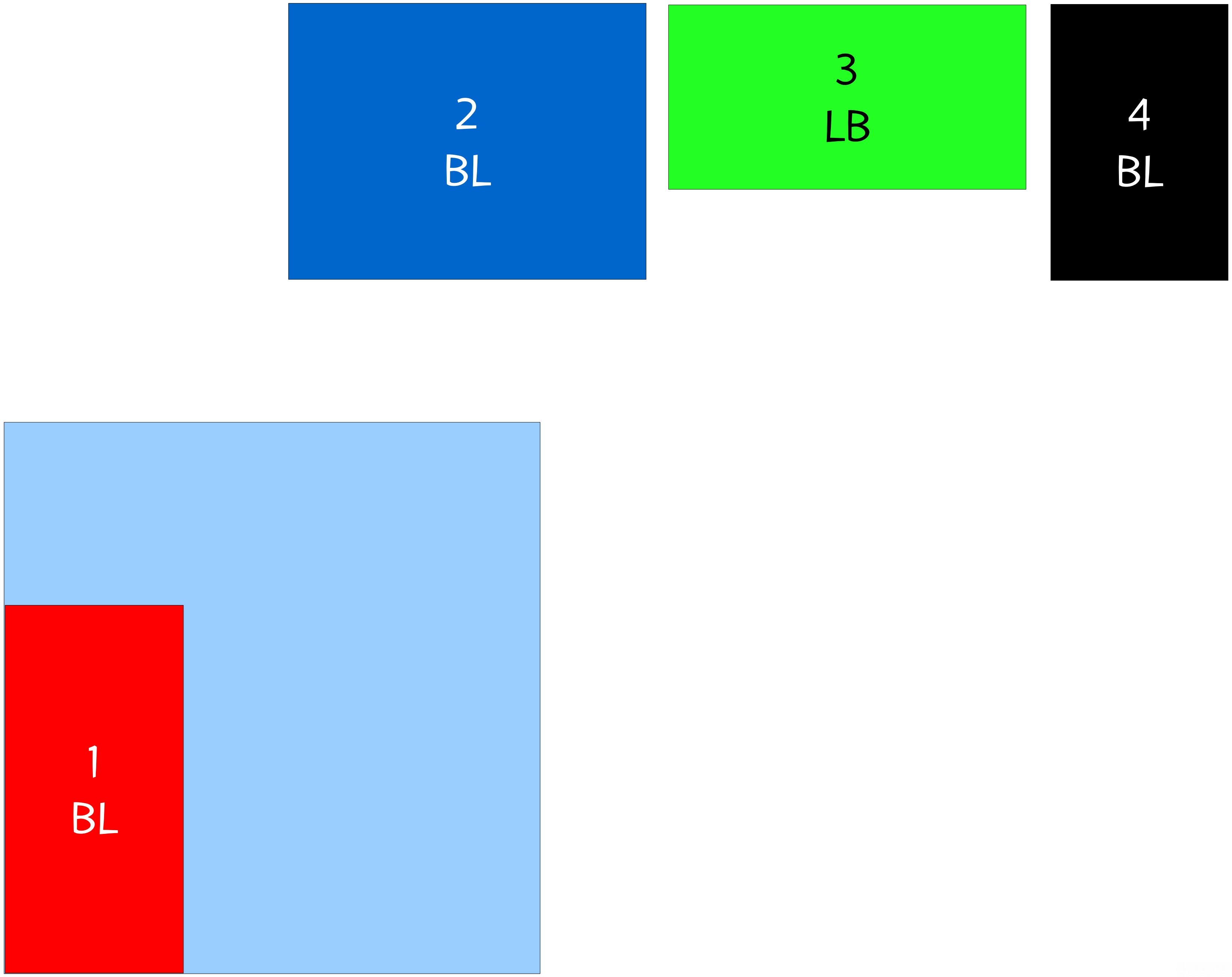
- If LB is used, ERSs are ordered such that ERS[i] < ERS[j] if x[i] < x[j] or x[i] = x[j] and y[i] < y[j].

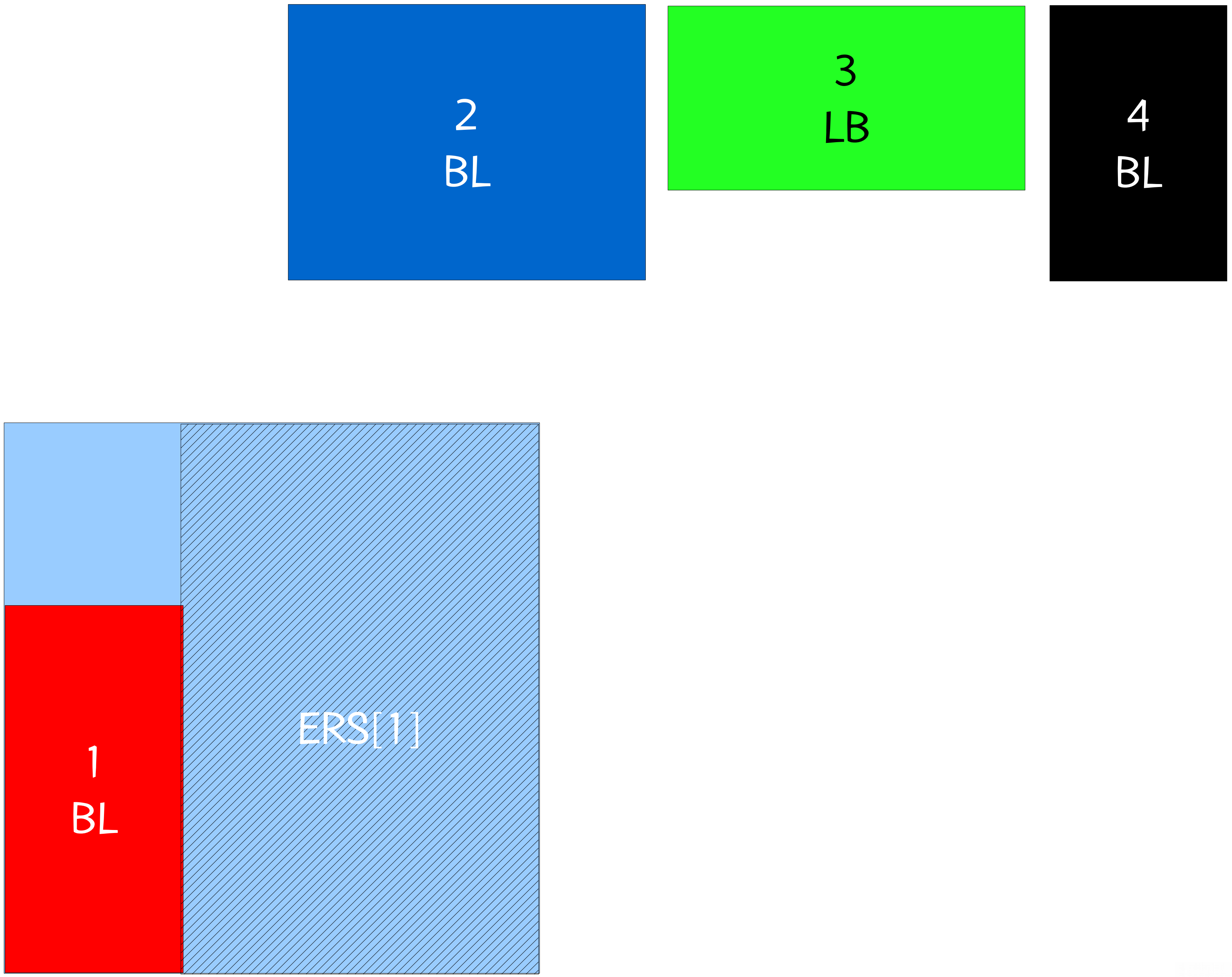


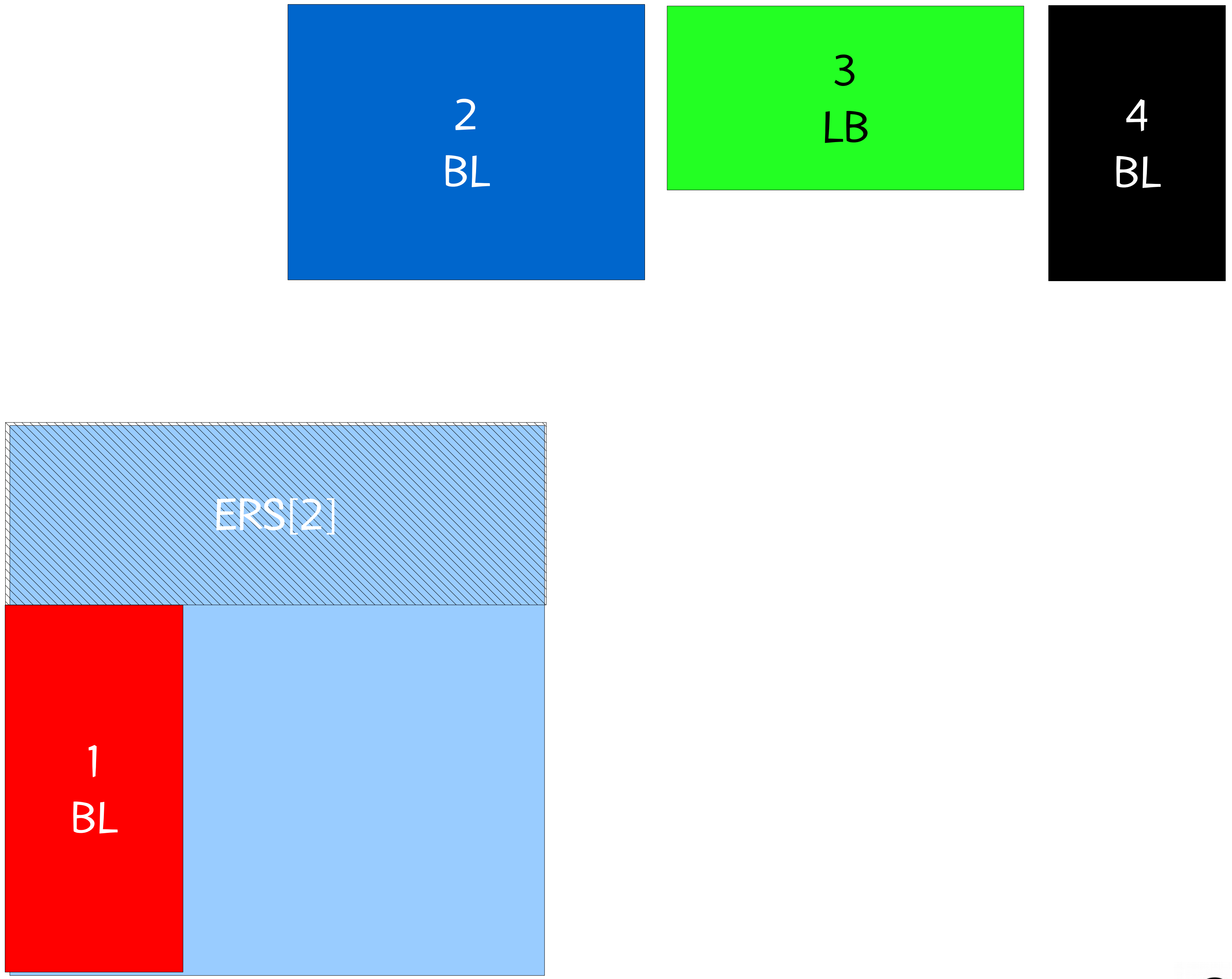
ERS[i] < ERS[j]

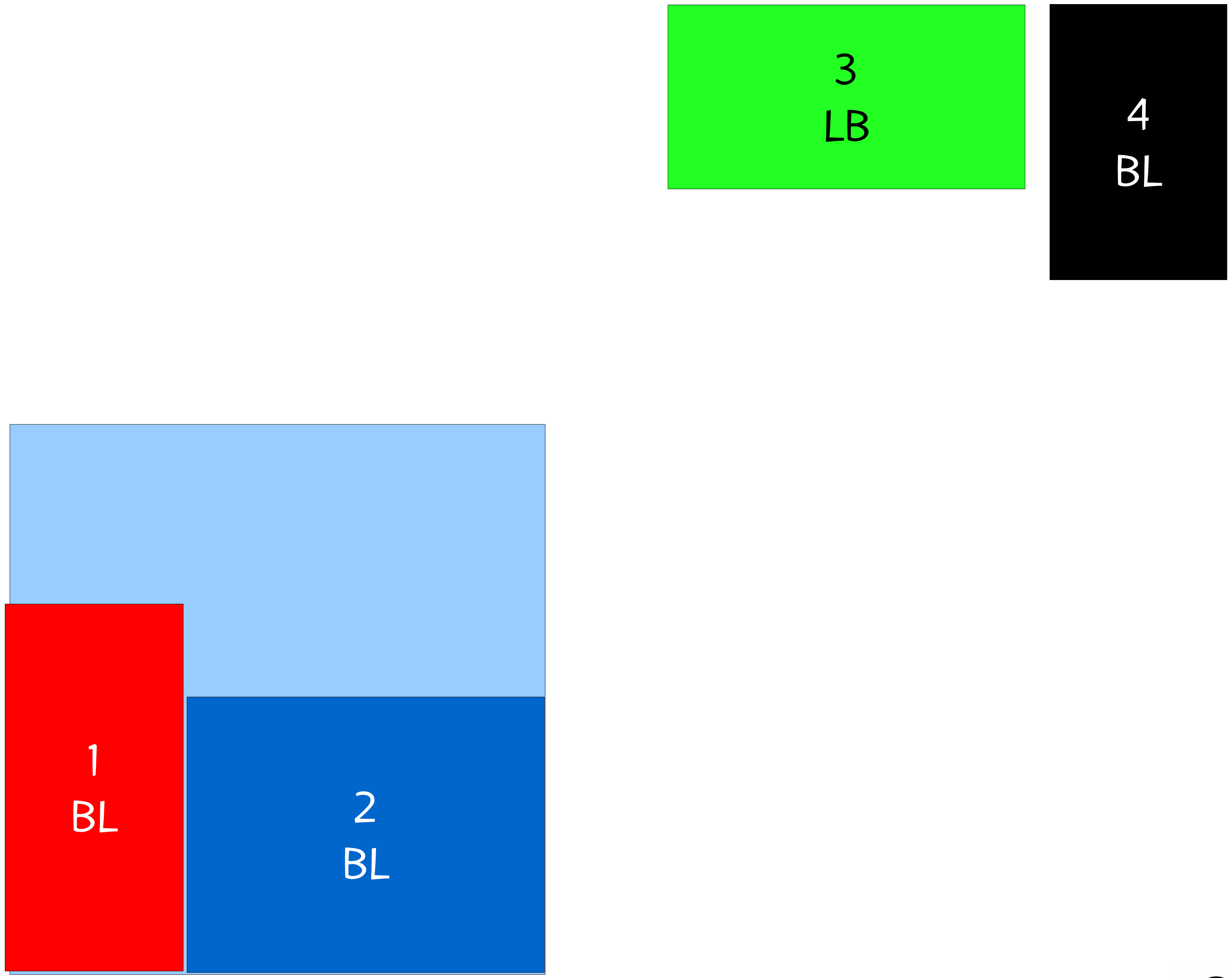

1
BL
2
BL
3
LB
4
BL

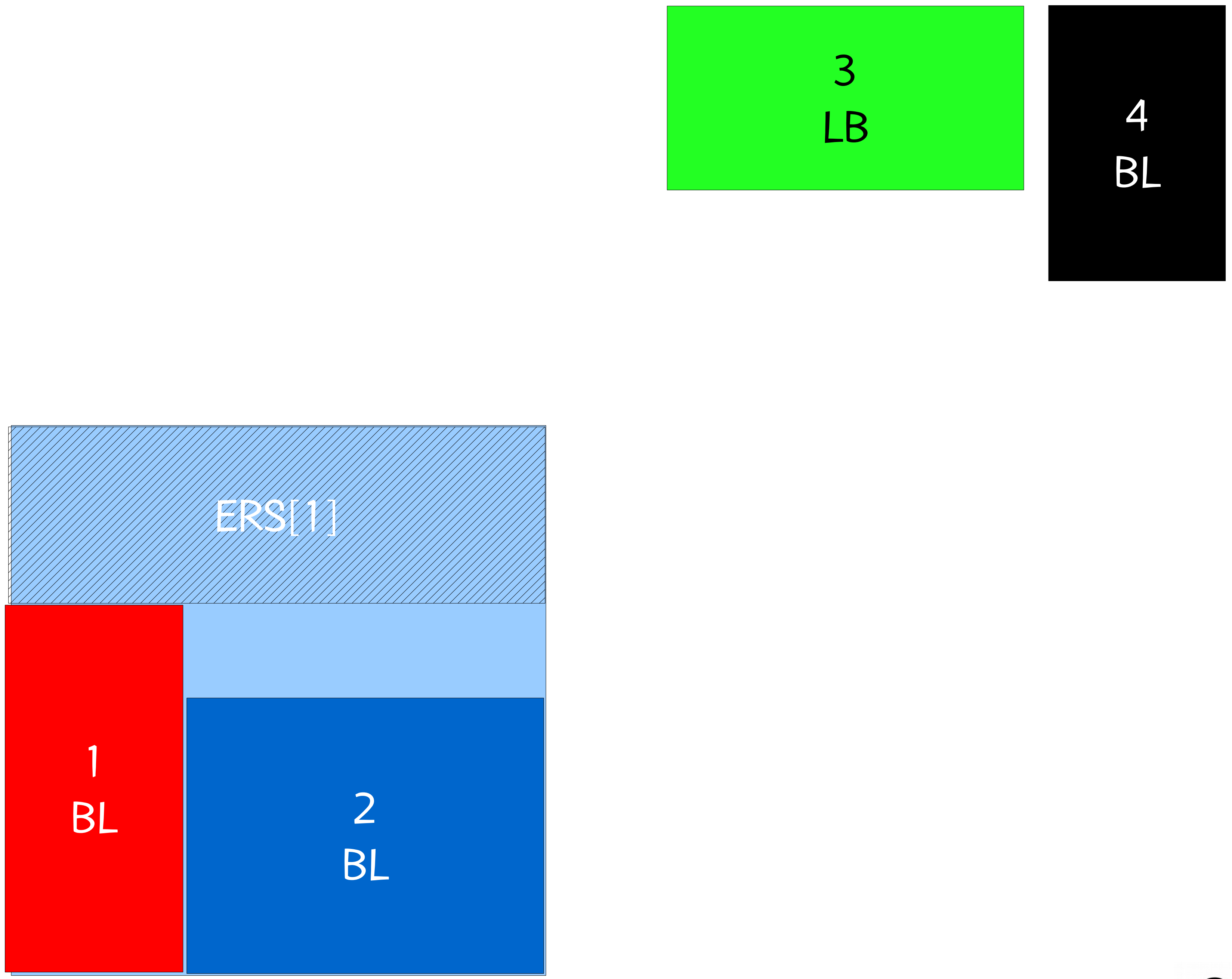

1
BL
2
BL
3
LB
4
BL
ERS[1]

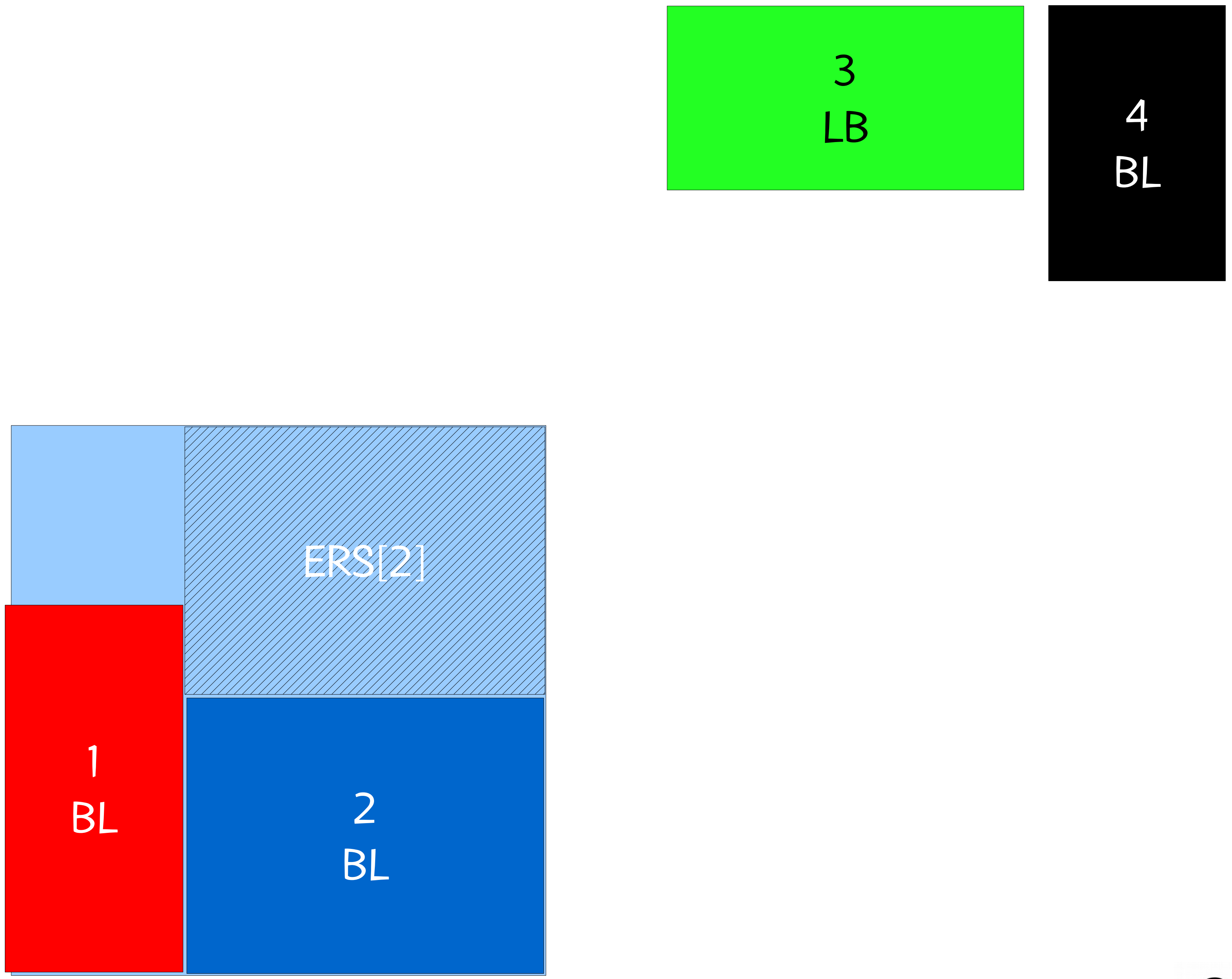

2
BL
3
LB
4
BL
1
BL

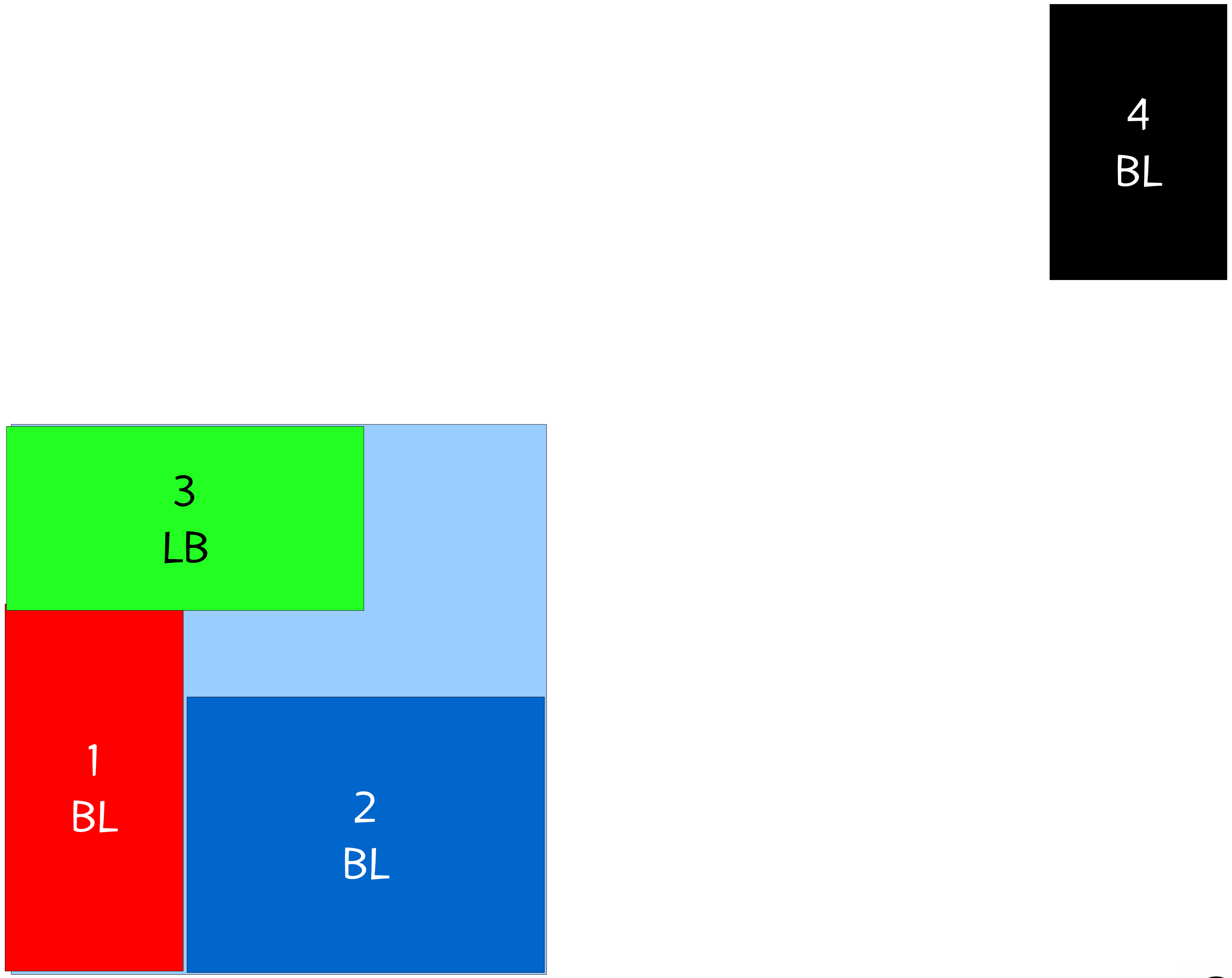

2
BL
3
LB
4
BL
ERS[1]
1
BL

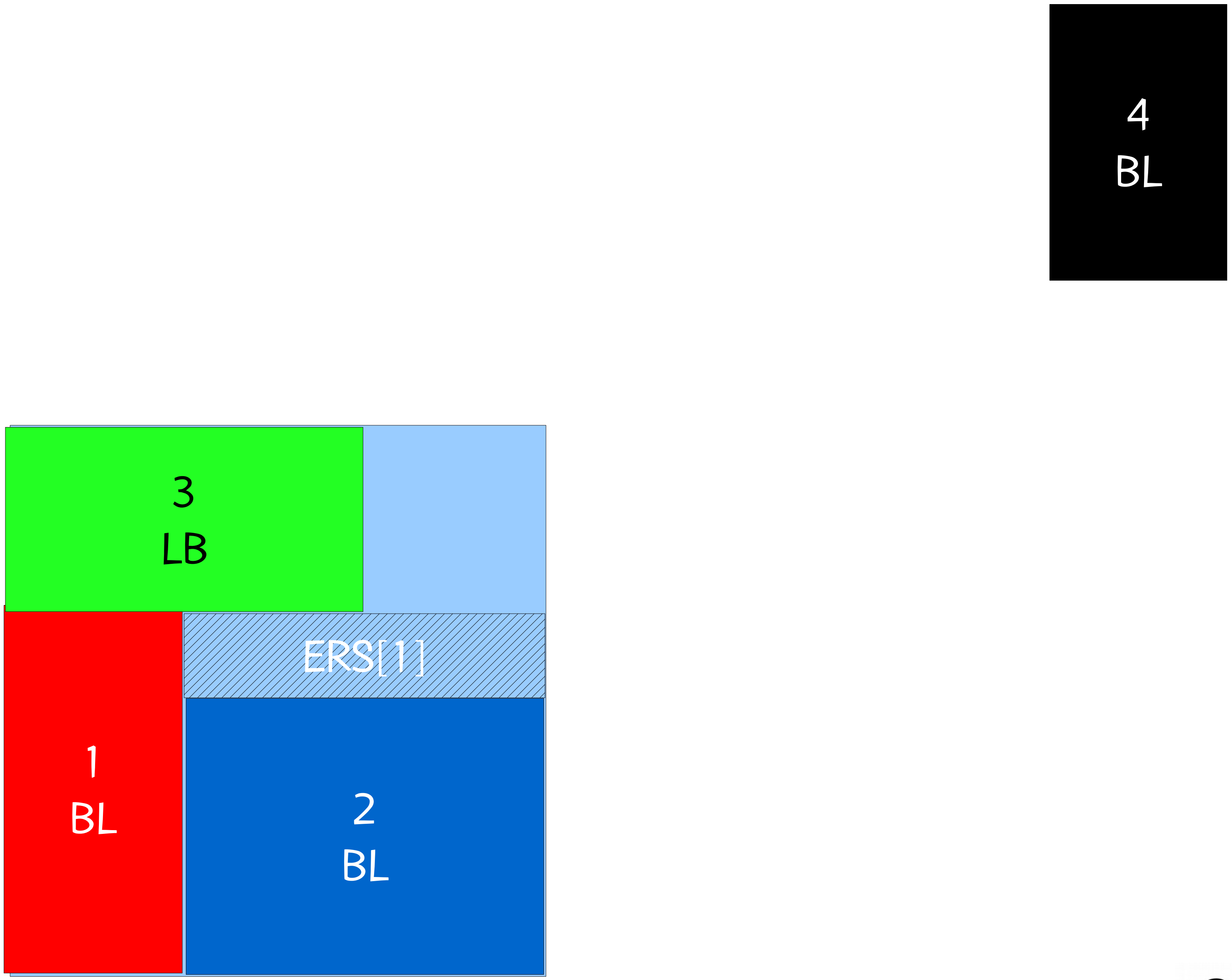

2
BL
3
LB
4
BL
ERS[2]
1
BL

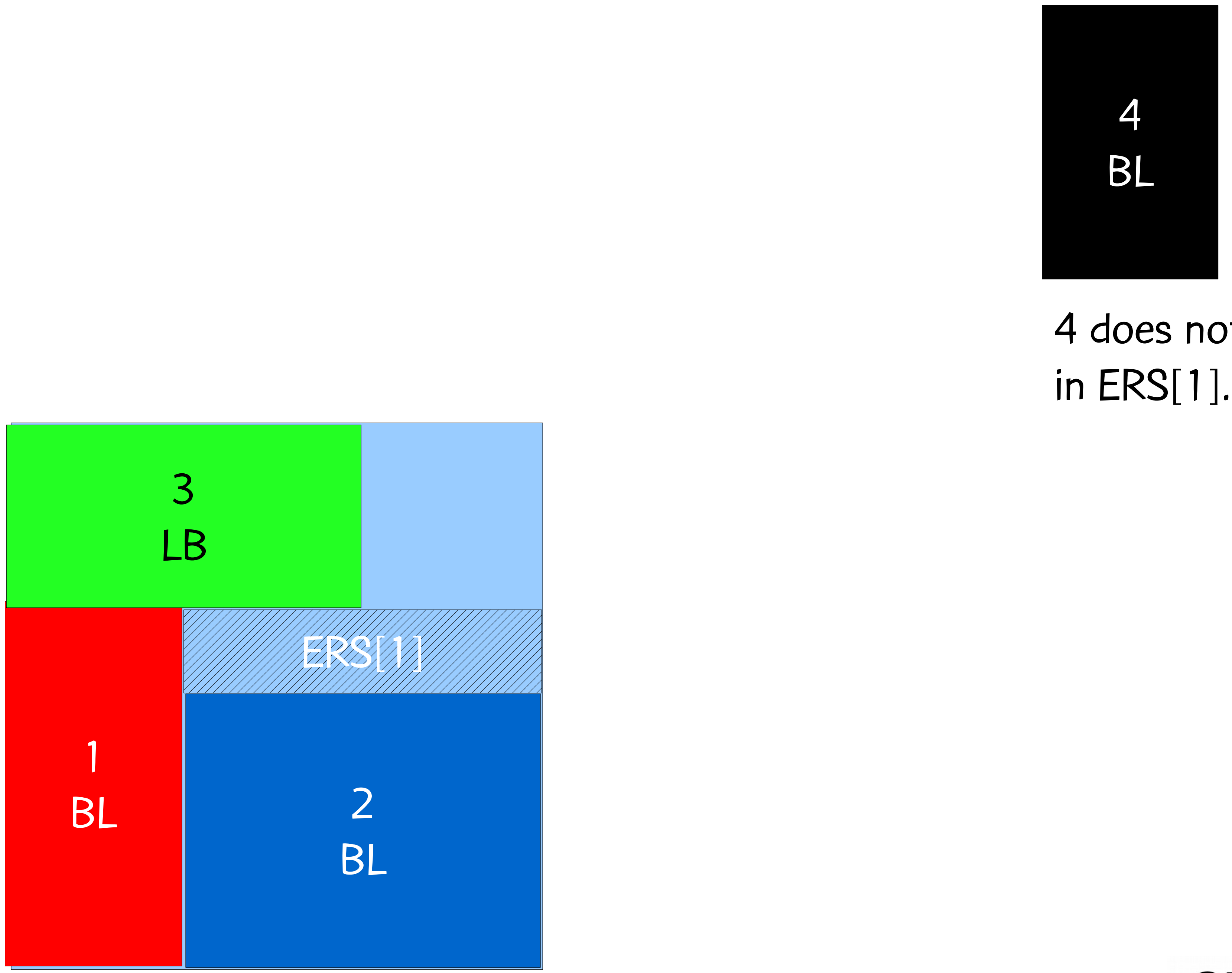

3
LB
4
BL
1
BL
2
BL


3
LB
4
BL
ERS[1]
1
BL
2
BL

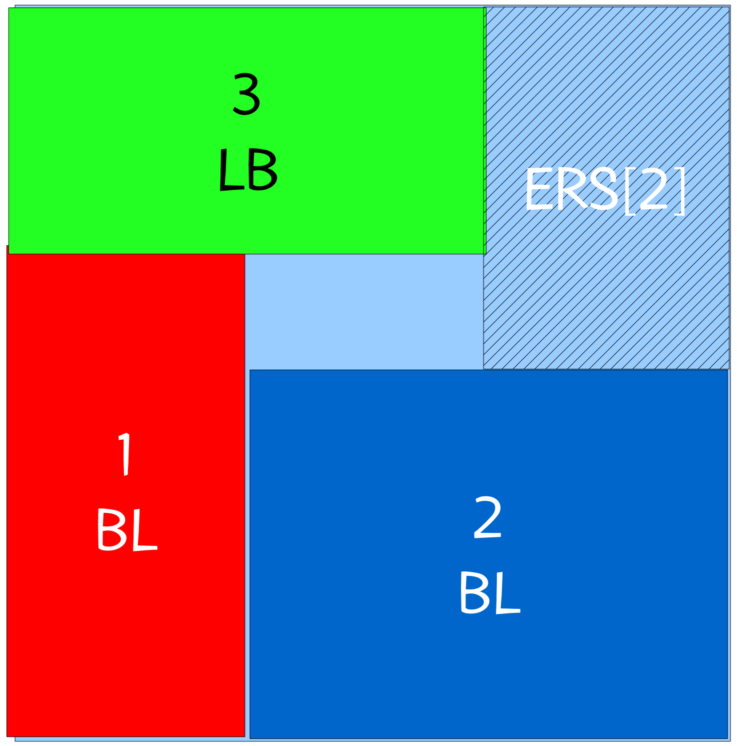

3
LB
4
BL
ERS[2]
1
BL
2
BL

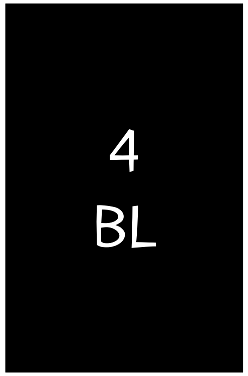

4
BL
3
LB
1
BL
2
BL


4
BL
3
LB
ERS[1]
1
BL
2
BL

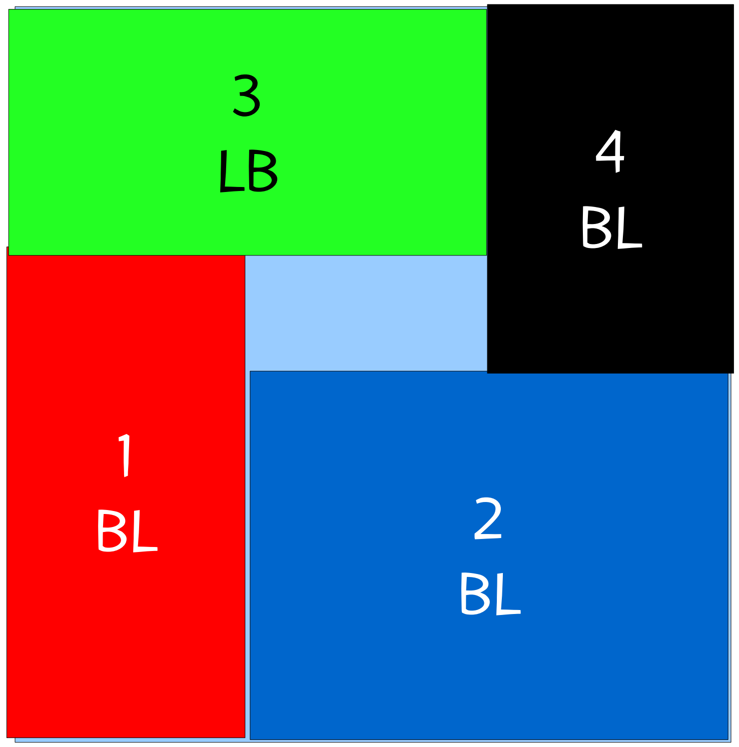


4 does not fit in ERS[1].





4 does fit
in ERS[2].



Optimal solution!

Experimental results

Design

- We compare solution values obtained by the parallel multi-population BRKGA with solutions obtained by the heuristics that produced the best computational results to date:

Design

- We compare solution values obtained by the parallel multi-population BRKGA with solutions obtained by the heuristics that produced the best computational results to date:
  - PH: population-based heuristic of Beasley (2004)

Design

- We compare solution values obtained by the parallel multi-population BRKGA with solutions obtained by the heuristics that produced the best computational results to date:
  - PH: population-based heuristic of Beasley (2004)
  - GA: genetic algorithm of Hadjiconsantinou & Iori (2007)

Design

- We compare solution values obtained by the parallel multi-population BRKGA with solutions obtained by the heuristics that produced the best computational results to date:
  - PH: population-based heuristic of Beasley (2004)
  - GA: genetic algorithm of Hadjiconsantinou & Iori (2007)
  - GRASP: greedy randomized adaptive search procedure of Alvarez-Valdes et al. (2005)

Design

- We compare solution values obtained by the parallel multi-population BRKGA with solutions obtained by the heuristics that produced the best computational results to date:
  - PH: population-based heuristic of Beasley (2004)
  - GA: genetic algorithm of Hadjiconsantinou & Iori (2007)
  - GRASP: greedy randomized adaptive search procedure of Alvarez-Valdes et al. (2005)
  - TABU: tabu search of Alvarez-Valdes et al. (2007)

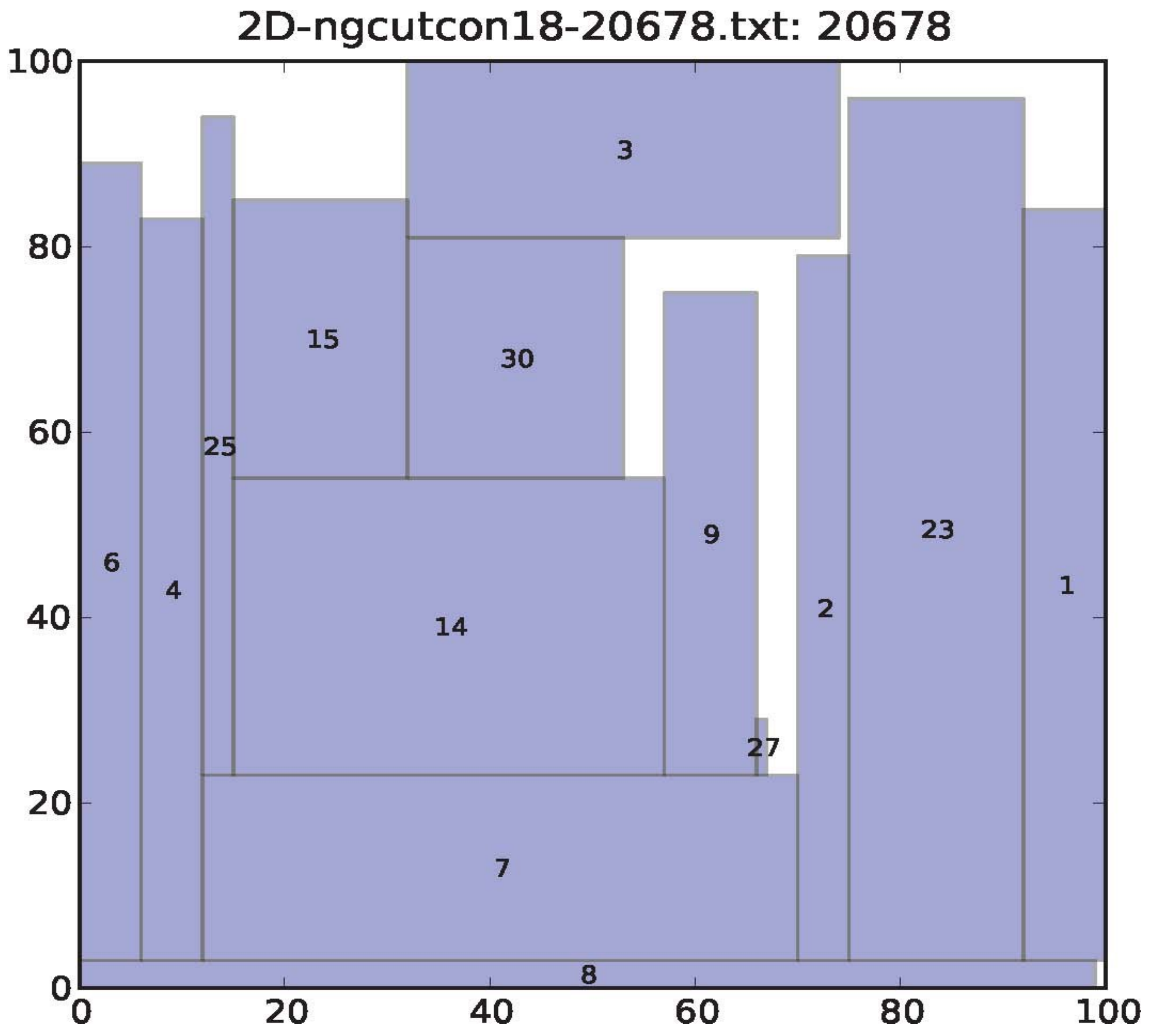
Number of best solutions / total instances

Problem	PH	GA	GRASP	TABU	BRKGA BL-LB-L-4NR
From literature (optimal)	13/21	**21/21**	18/21	**21/21**	**21/21**
Large random*	0/21	0/21	5/21	8/21	**20/21**
Zero-waste			5/31	17/31	**30/31**
Doubly constrained	11/21		12/21	17/21	**19/21**

* For large random: number of best average solutions / total instance classes

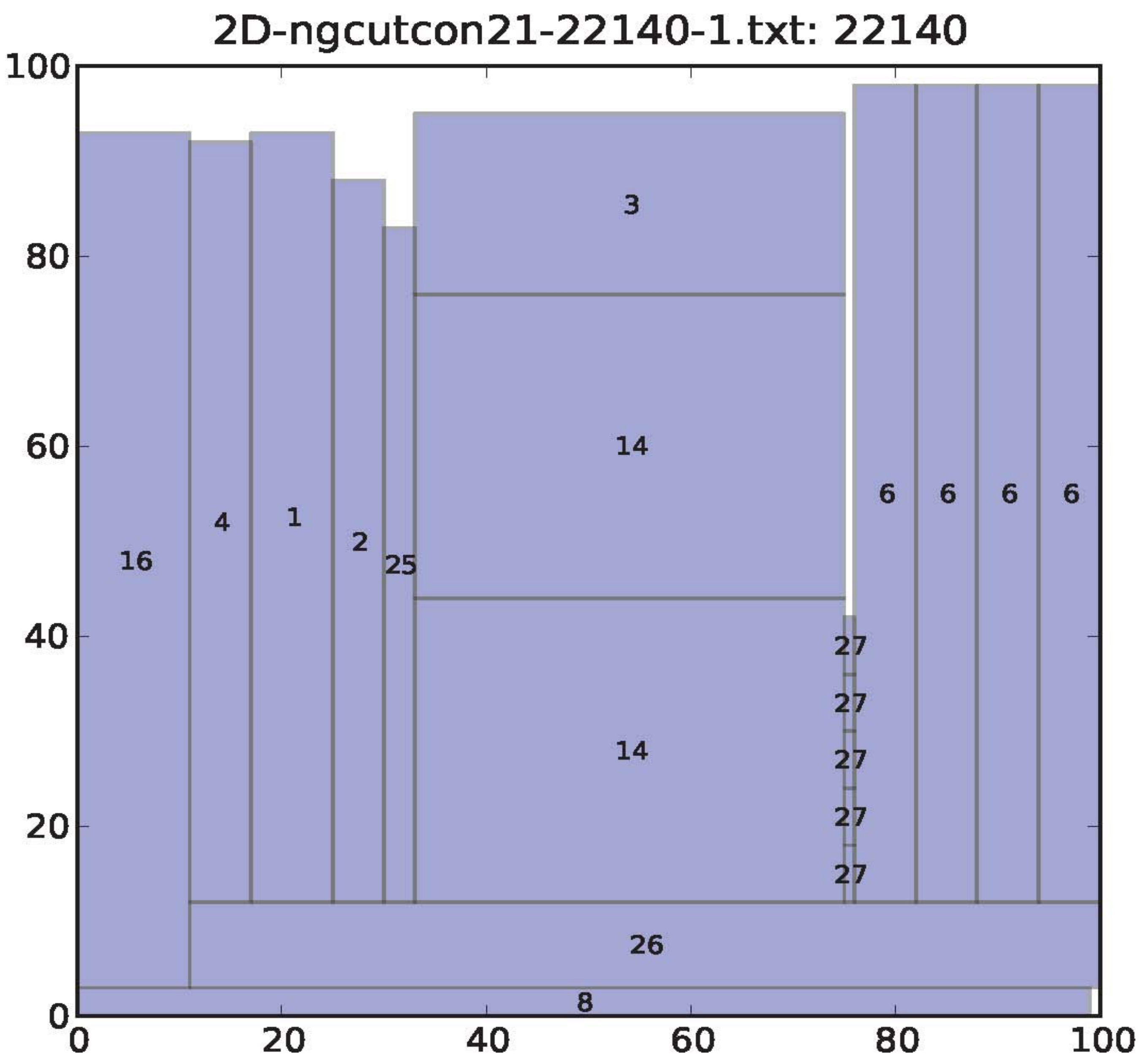
Minimum, average, and maximum solution times (secs) for BRKGA (BL-LB-L-4NR)

Problem	Min solution time (secs)	Avg solution time (secs)	Max solution time (secs)
From literature (optimal)	0.00	0.05	0.55
Large random	1.78	23.85	72.70
Zero-waste	0.01	82.21	808.03
Doubly constrained	0.00	1.16	16.87



New BKS for a 100 x100 doubly constrained instance of Fekete & Schepers (1997) of value **20678.** Previous best was **19657** by tabu search of Alvarez-Valdes et al., (2007).

30 types
30 rectangles



New BKS for a 100 x 100 doubly constrained instance Fekete & Schepers (1997) of value **22140**.

Previous BKS was **22011** by tabu search of Alvarez-Valdes et al. (2007).

29 types
97 rectangles

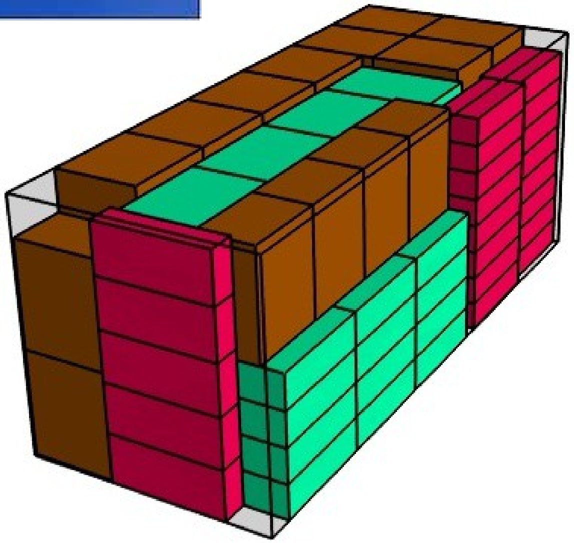
amazon

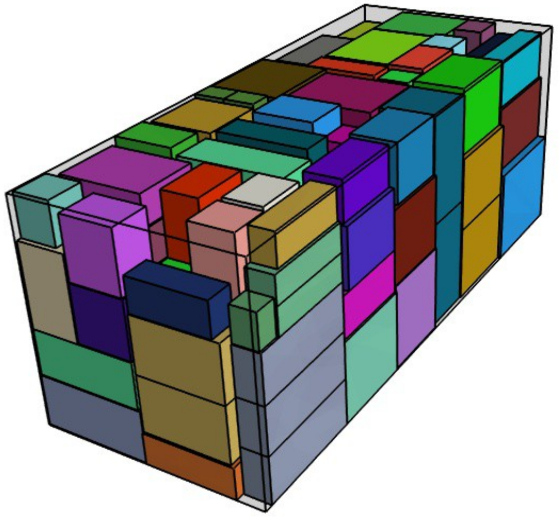
Some remarks

We have extended this to 3D packing:

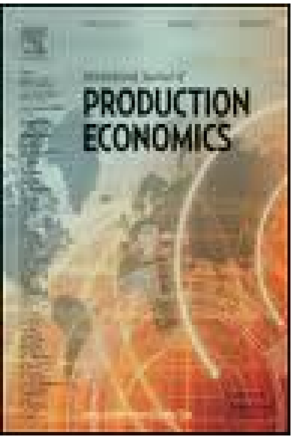
J.F. Gonçalves and M.G.C.R., "A parallel multi-population biased random-key genetic algorithm for a container loading problem," Computers & Operations Research, vol. 29, pp. 179-190, 2012.

Tech report: http://mauricio.resende.info/doc/brkga-pack3d.pdf





3D bin packing



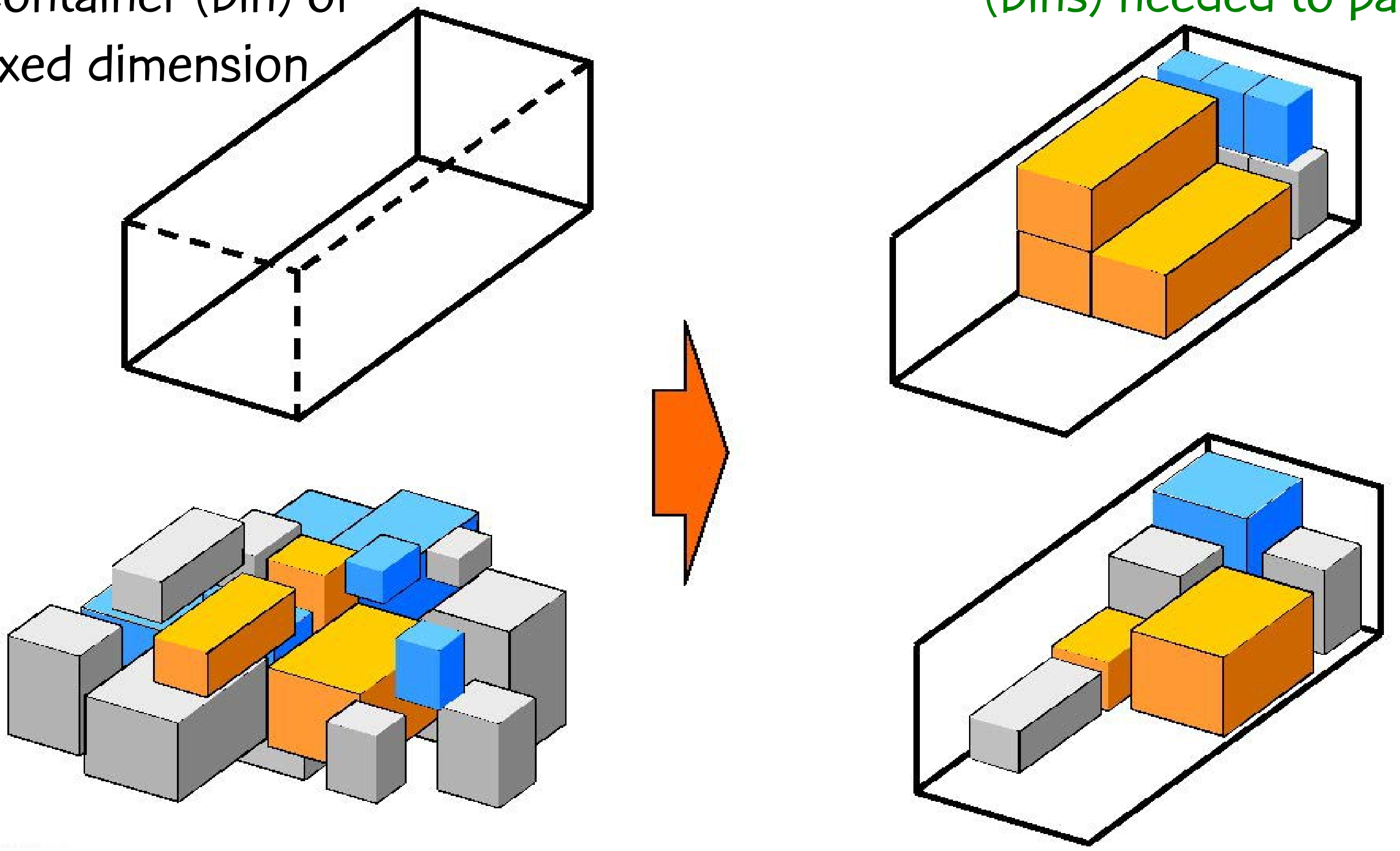
J.F. Gonçalves and R., "A biased random-key genetic algorithm for 2D and 3D bin packing problems," International J. of Production Economics, vol. 15, pp. 500–510, 2013.

http://mauricio.resende.info/doc/brkga-binpacking.pdf

3D bin packing problem

Minimize number of containers (bins) needed to pack all boxes

Container (bin) of fixed dimension

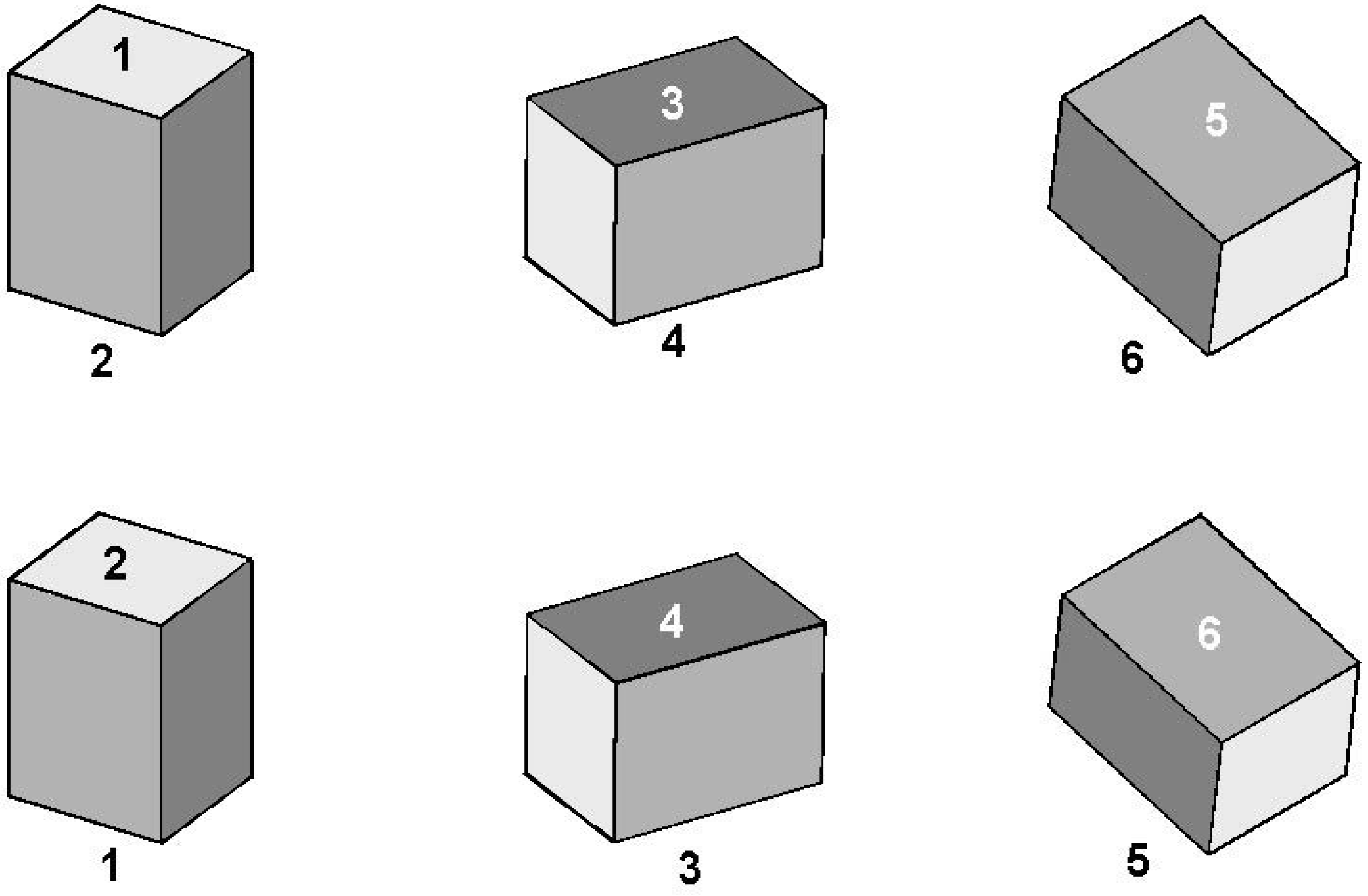


Boxes of different dimensions

3D bin packing constraints

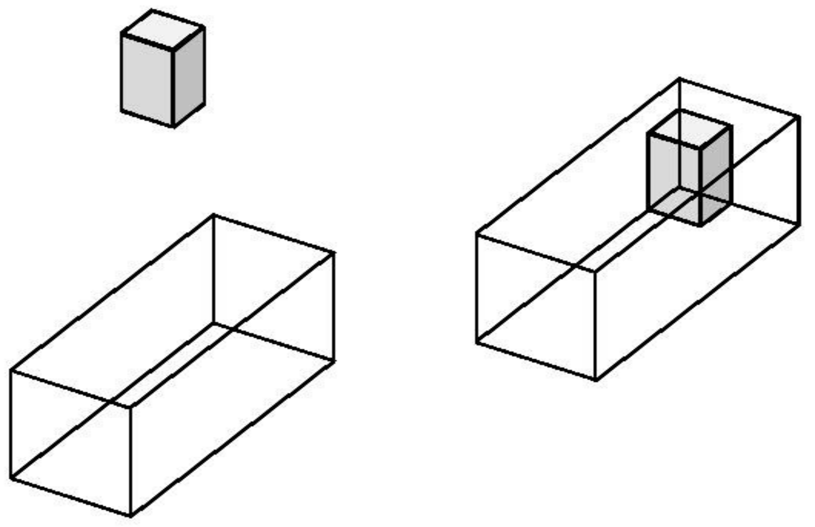
- Each box is placed completely within container
- Boxes do not overlap with each other
- Each box is placed parallel to the side walls of bin
- In some instances, only certain box orientations are allowed (there are at most six possible orientations)

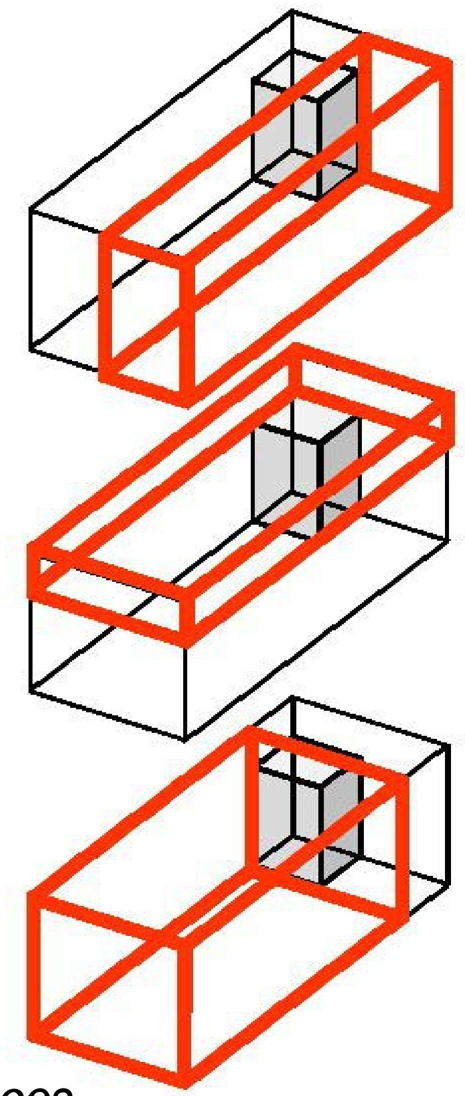
Six possible orientations for each box



Difference process - DP

(Lai & Chan, 1997)





When box is placed in container ...

use DP to keep track of maximal free spaces

Encoding

Solutions are encoded as vectors of 3n random keys, where n is the number of boxed to be packed.

$$X = (\underbrace{x_1, x_2, \dots, x_n}_{\text{Box packing sequence}}, \underbrace{x_{n+1}, x_{n+2}, \dots, x_{2n}}_{\text{Placement heuristic}}, \underbrace{x_{2n+1}, x_{2n+2}, \dots, x_{3n}}_{\text{Box orientation}})$$

Decoding

1) Sort first n keys of X to produce sequence boxes will be packed;
2) Use second n keys of X to determine which placement heuristic to use (back-bottom-left or back-left-bottom):
   - if $x_{n+i} < ½$ then use back-bottom-left to pack i-th box
   - if $x_{n+i} \geq ½$ then use back-left-bottom to pack i-th box
3) Use third n keys of X to determine which of six orientations to use when packing box:
   - $x_{2n+i} \in [0,1/6)$: orientation 1;
   - $x_{2n+i} \in [1/6,2/6)$: orientation 2; ...
   - $x_{2n+i} \in [5/6,1]$: orientation 6.

Decoding

For each box

- scan containers in order they were opened
- use placement heuristic to place box in first container in which box fits with its specified orientation
- if box does not fit in any open container, open new container and place box using placement heuristic with its specified orientation

Fitness function

Instead of using as fitness measure the number of bins (NB)

- use adjusted fitness: aNB
- aNB = NB + (LeastLoad / BinVolume), where
  - ✗ LeastLoad is load on least loaded bin
  - ✗ BinVolume is volume of bin: H x W x L

Experiment

- Parameters:
  - population size: $p = 30n$
  - size of elite partition: $p_e = .10p$
  - number of of mutans: $p_m = .15p$
  - crossover probability: 0.7
  - stopping criterion: 300 generations

Experiment

- Instances:
  - 320 instances of Martello et al. (2000)
  - generator is available at http://www.diku.dk/~pisinger/codes/html
  - 8 classes
  - 40 instances per class
  - 10 instances for each value of n ∈ {50, 100, 150, 200)

Experiment

- We compare BRKGA with:
  - TS3, the tabu search of Lodi et al. (2002)
  - GLS, the guided local search of Faroe et al. (2003)
  - TS2PACK, the tabu search of Crainic et al. (2009)
  - GRASP, the greedy randomized adaptive search procedure of Parreno et al. (2010)

Summary

Class	Bin size	BRKGA	GRASP	TS3	TS2PACK	GLS
1	100^3	**127.3**	**127.3**	127.9	128.2	128.3
2	100^3	**125.5**	125.8	126.8		
3	100^3	**126.5**	126.9	127.5		
4	100^3	294.0	294.0	294.0	**293.9**	294.2
5	100^3	**70.4**	70.5	71.4	71.0	70.8
6	10^3	**95.0**	95.4	96.1	95.8	96.0
7	40^3	**58.2**	59.4	60.0	59.0	59.0
8	100^3	**80.9**	82.0	82.6	81.9	81.9
Sum(rows 1, 4-8):		**725.8**	728.6	732.0	729.8	730.2
Sum(rows 1-8):		**977.8**	981.3	986.3		

The unequal area facility layout problem

Volume 169, Issue 2, 1 March 2006

ISSN 0377-2217

ELSEVIER

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Also available on SCIENCE DIRECT®

www.sciencedirect.com

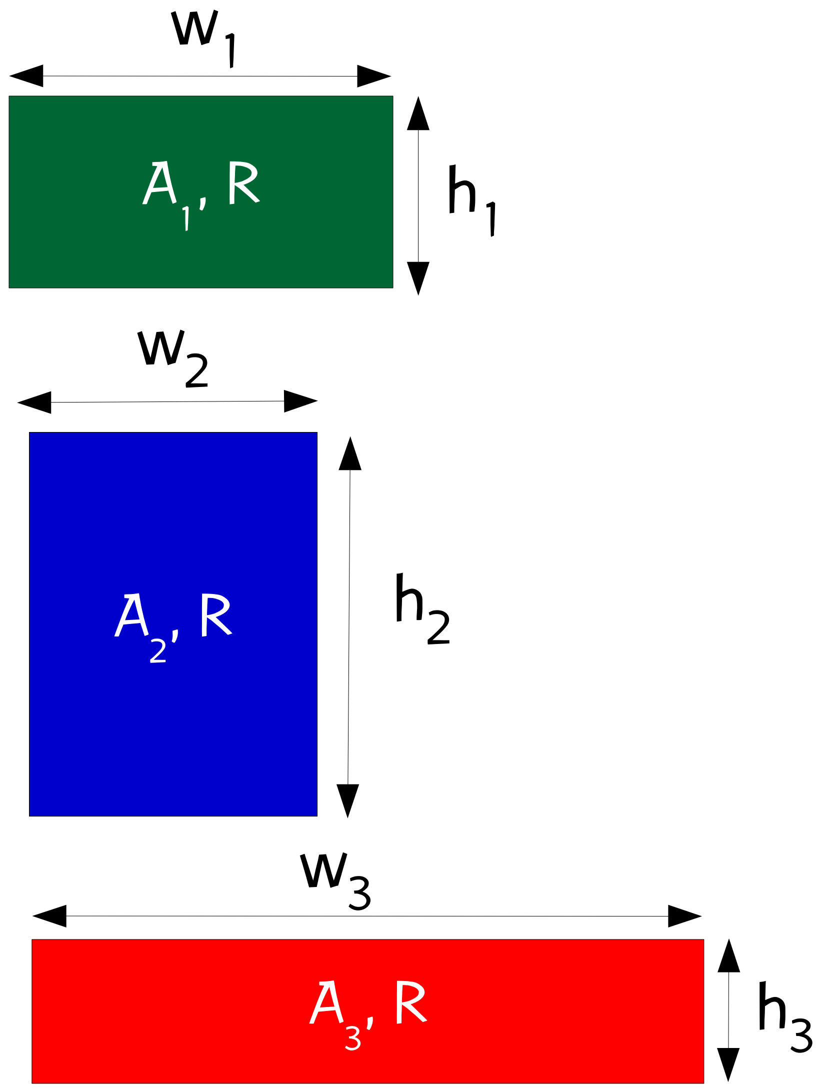
Including a Feature Cluster on
Scatter Search Methods for Optimization

Guest Editor:
R. Martí

www.elsevier.com/locate/ejor

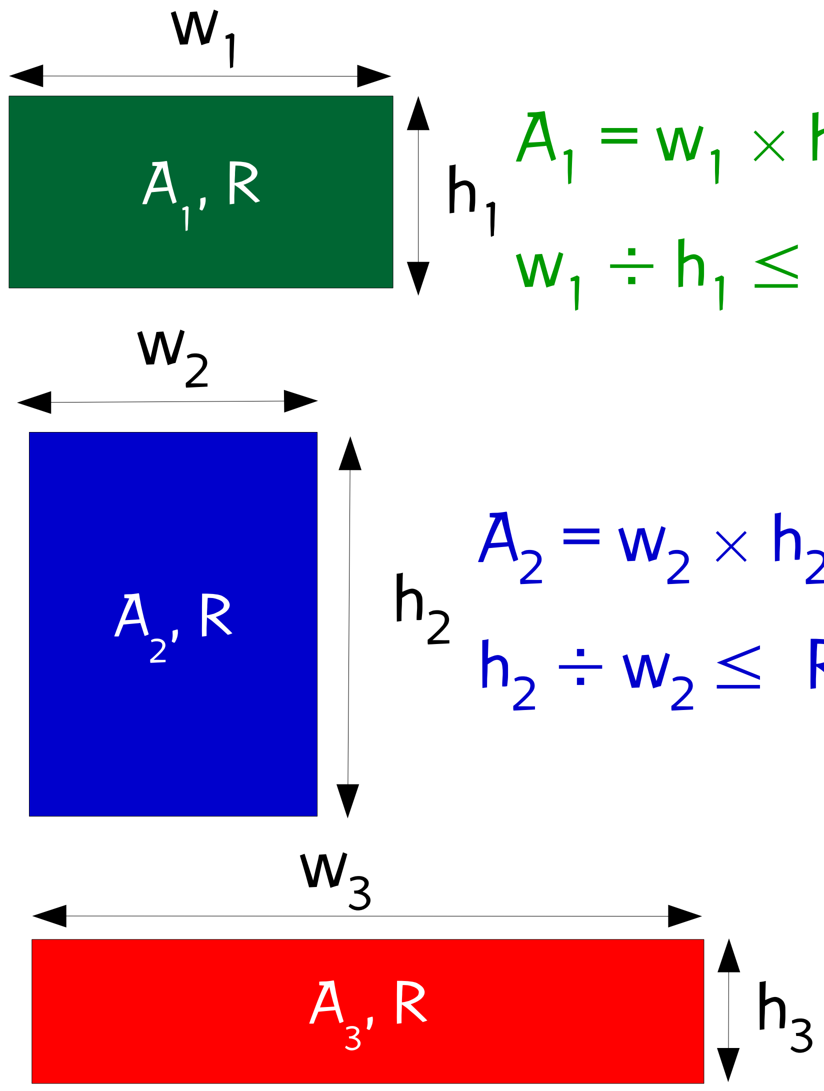
J.F. Gonçalves & R., "A biased random-key genetic algorithm for the unequal area facility layout problem," European J. of Operational Research, vol. 246, pp. 86-107, 2015

Unequal area facility layout



Given N rectangular facilities, $i = 1, 2, \ldots, N$, each having given area $A_i = w_i \times h_i$ all of maximum aspect ratio (between longest & shortest dimensions) R

Unequal area facility layout



$A_1 = w_1 \times h_1$

$w_1 \div h_1 \leq R$

$A_2 = w_2 \times h_2$

$h_2 \div w_2 \leq R$

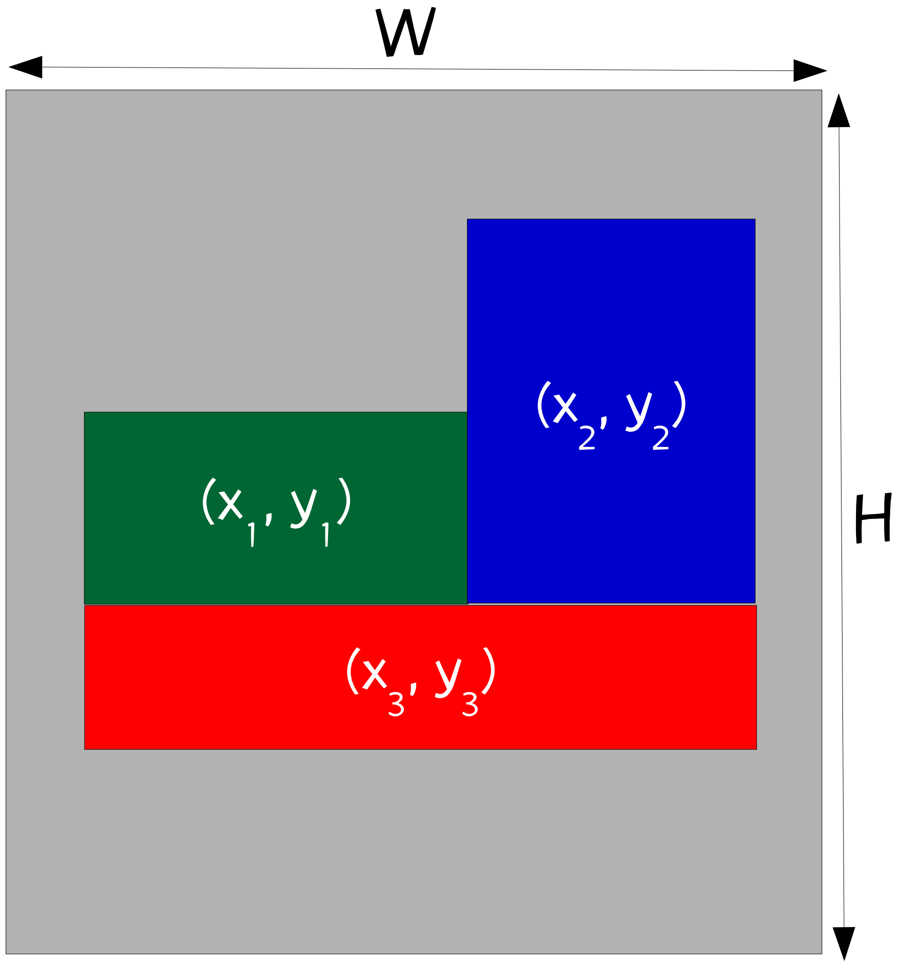
$A_3 = w_3 \times h_3$

$w_3 \div h_3 \leq R$

Given N rectangular facilities, $i = 1, 2, \ldots, N$, each having given area $A_i = w_i \times h_i$ all of maximum aspect ratio (between longest & shortest dimensions) R (Note that w_i and h_i are not given, only A_i and R are)

amazon

Unequal area facility layout



Layout the facilities, without overlap or rotation, on a rectangular floor of area $W \times H$ with centroids at coordinates $(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)$ and dimensions $w_1 \times h_1, w_2 \times h_2, \ldots, w_N \times h_N$.

Unequal area facility layout

We consider two types of problems

- In the constrained type, we are given the rectangular floor dimensions $W \times H$.
- In the unconstrained type, we assume the floor space can include all the facilities laid out horizontally or vertically at their maximum horizontal or vertical dimensions, i.e.

$$(W, H) = \left(\sum_{i=1}^{N} (A_i \times R)^{1/2}, \sum_{i=1}^{N} (A_i \times R)^{1/2} \right)$$

Unequal area facility layout

Of all feasible layouts, find one that minimizes

$$\sum_{i=1}^{N} \sum_{j=1}^{N} f_{i,j} \times c_{i,j} \times d_{i,j}$$

where

- $f_{i,j}$ is the flow between facilities i and j ($f_{i,i} = 0$)
- $c_{i,j}$ is the cost per unit distance between i and j
- $d_{i,j} = |x_i - x_j| + |y_i - y_j|$ is the rectilinear distance between (x_i, y_i) and (x_j, y_j)

Unequal area facility layout

Of all feasible layouts, find one that minimizes

$$\sum_{i=1}^{N} \sum_{j=1}^{N} f_{i,j} \times c_{i,j} \times d_{i,j}$$

quadratic assignment problem (QAP)

where

- $f_{i,j}$ is the flow between facilities i and j ($f_{i,i} = 0$)
- $c_{i,j}$ is the cost per unit distance between i and j
- $d_{i,j} = |x_i - x_j| + |y_i - y_j|$ is the rectilinear distance between (x_i,y_i) and (x_j,y_j)

Unequal area facility layout

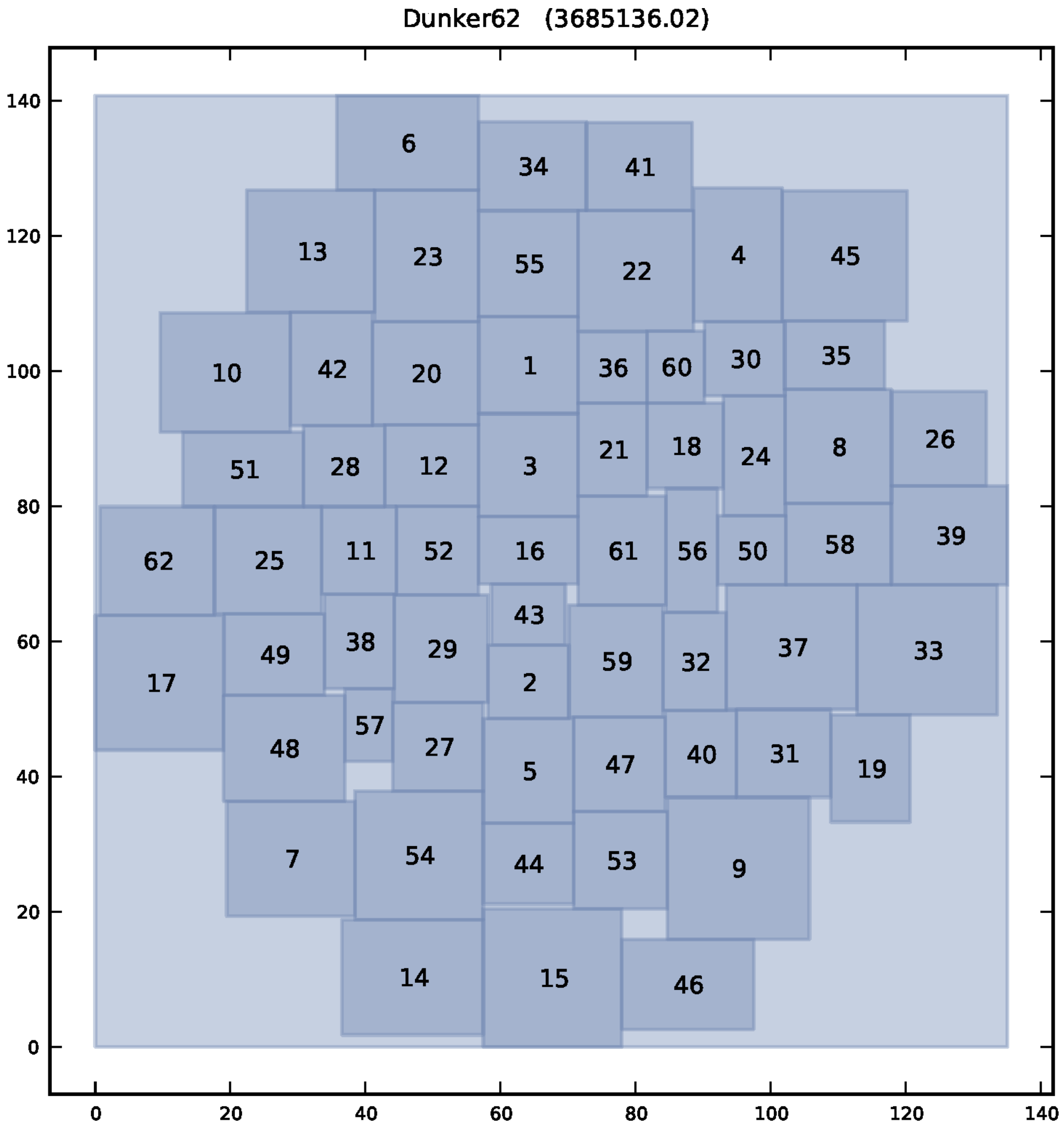
Of all feasible layouts, find one that minimizes

$$\sum_{i=1}^{N}\sum_{j=1}^{N} f_{i,j} \times c_{i,j} \times d_{i,j}$$

Besides rectilinear (R) distance metric, we also deal with Euclidean (E), and Squared Euclidean (SE) in paper.

where

- $f_{i,j}$ is the flow between facilities i and j ($f_{i,i} = 0$)
- $c_{i,j}$ is the cost per unit distance between i and j
- $d_{i,j} = |x_i - x_j| + |y_i - y_j|$ is the rectilinear distance between (x_i,y_i) and (x_j,y_j)



Dunker62

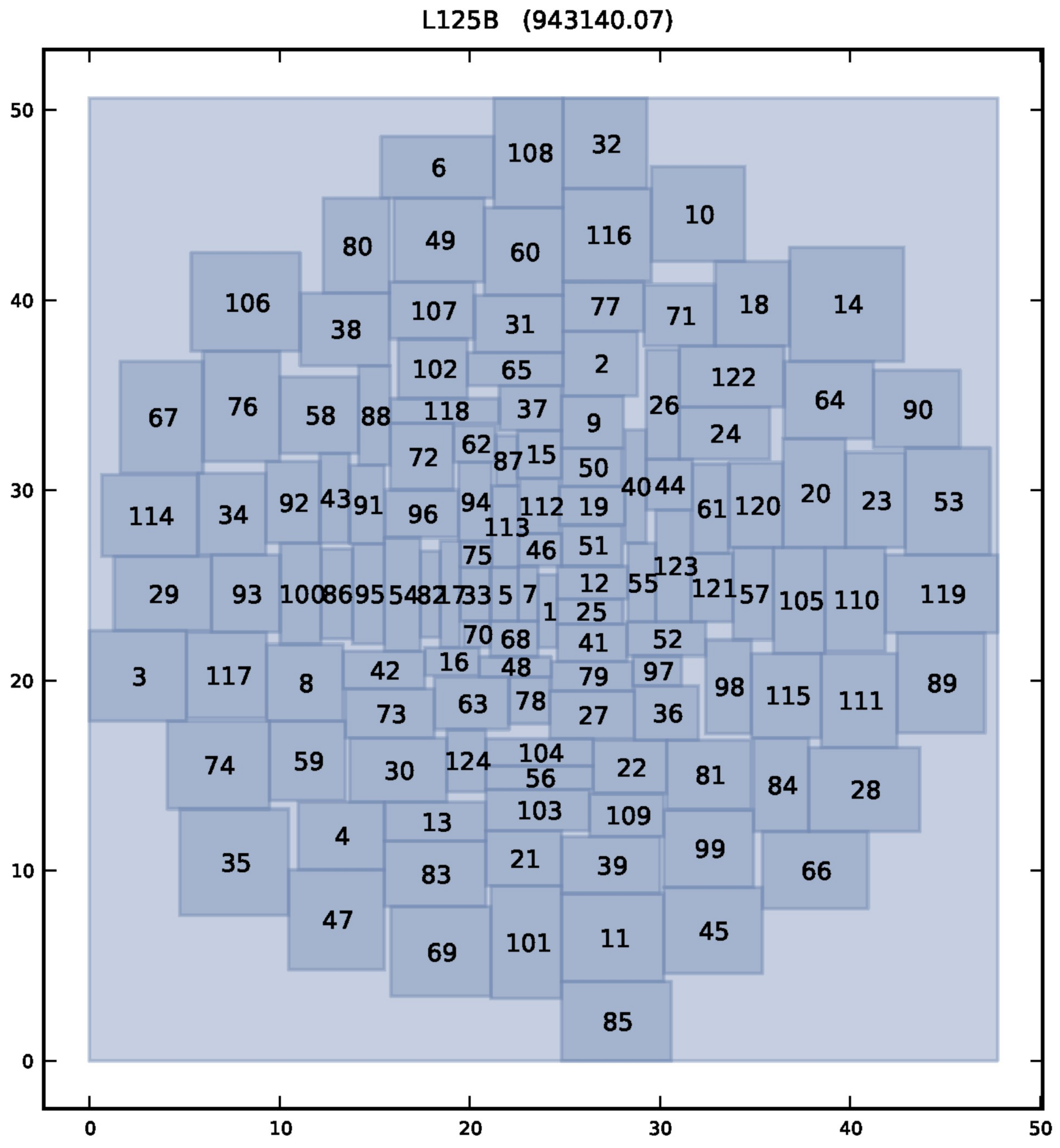
New best known solution: 3.68E6

Previous best known solution: 3.81E6

TS-BST (McKendall Jr. & Hajobyan, 2010)

BRKGA

amazon



L125B

New best known solution: 9.43E5

Previous best known solution: 1.01E6 TS-BST (McKendall Jr. & Hajobyan, 2010)

BRKGA

BRKGA for the unequal area facility layout problem

Encoding

Solutions are encoded with a vector of random keys of length 2N+2

$$X = (\ X_1, \ldots, X_N,\ X_{N+1}, \ldots, X_{2N},\ X_{2N+1}, X_{2N+2}\ )$$

Encoding

Solutions are encoded with a vector of random keys of length 2N+2

$$X = (\underbrace{X_1, \ldots, X_N}_{}, X_{N+1}, \ldots, X_{2N}, X_{2N+1}, X_{2N+2})$$

Facility placement sequence

Encoding

Solutions are encoded with a vector of random keys of length 2N+2

$$X = (\underbrace{X_1, \ldots, X_N,}_{\text{Facility placement sequence}} \; \underbrace{X_{N+1}, \ldots, X_{2N},}_{\text{Facility aspect ratios}} \; X_{2N+1}, X_{2N+2})$$

Encoding

Solutions are encoded with a vector of random keys of length 2N+2

$$X = (\underbrace{X_1, \ldots, X_N,}_{\text{Facility placement sequence}} \underbrace{X_{N+1}, \ldots, X_{2N},}_{\text{Facility aspect ratios}} \underbrace{X_{2N+1}, X_{2N+2}}_{\text{(x, y) coordinates of the first facility to be placed}})$$

Decoding

1. Use $X_1, \ldots, X_N$ to determine the sequence in which the facilities are placed on the floor space
2. Use $X_{N+1}, \ldots, X_{2N}$ to determine the aspect ratio of each facility
3. Use X_{2N+1}, X_{2N+2} to determine the (x, y) coordinates of the first facility to be placed on the floor space
4. Use results of (1)-(3) with placement heuristic to place all the facilities on the floor space
5. Evaluate fitness of solution

Decoder: Step 1

Use $X_1, \ldots, X_N$ to determine the sequence in which the facilities are placed on the floor space:

Simply sort the key values $X_1, \ldots, X_N$ to determine the indices of the permutation of the facilities.

Decoder: Step 2

Use $X_{N+1}, \ldots, X_{2N}$ to determine the aspect ratio of each facility:

Aspect ratio of facility i is

$$FAR_i = (1/R) + X_{N+i} \times (R - (1/R)),$$

where R is the given maximum facility aspect ratio.

Decoder: Step 2

Use $X_{N+1}, \ldots, X_{2N}$ to determine the aspect ratio of each facility:

Aspect ratio of facility i is

$$FAR_i = (1/R) + X_{N+i} \times (R - (1/R)),$$

where R is the given maximum facility aspect ratio.

$$\boxed{\begin{array}{l} w_i = (A_i \times FAR_i)^{1/2} \text{ and} \\ h_i = A_i / w_i \end{array}}$$

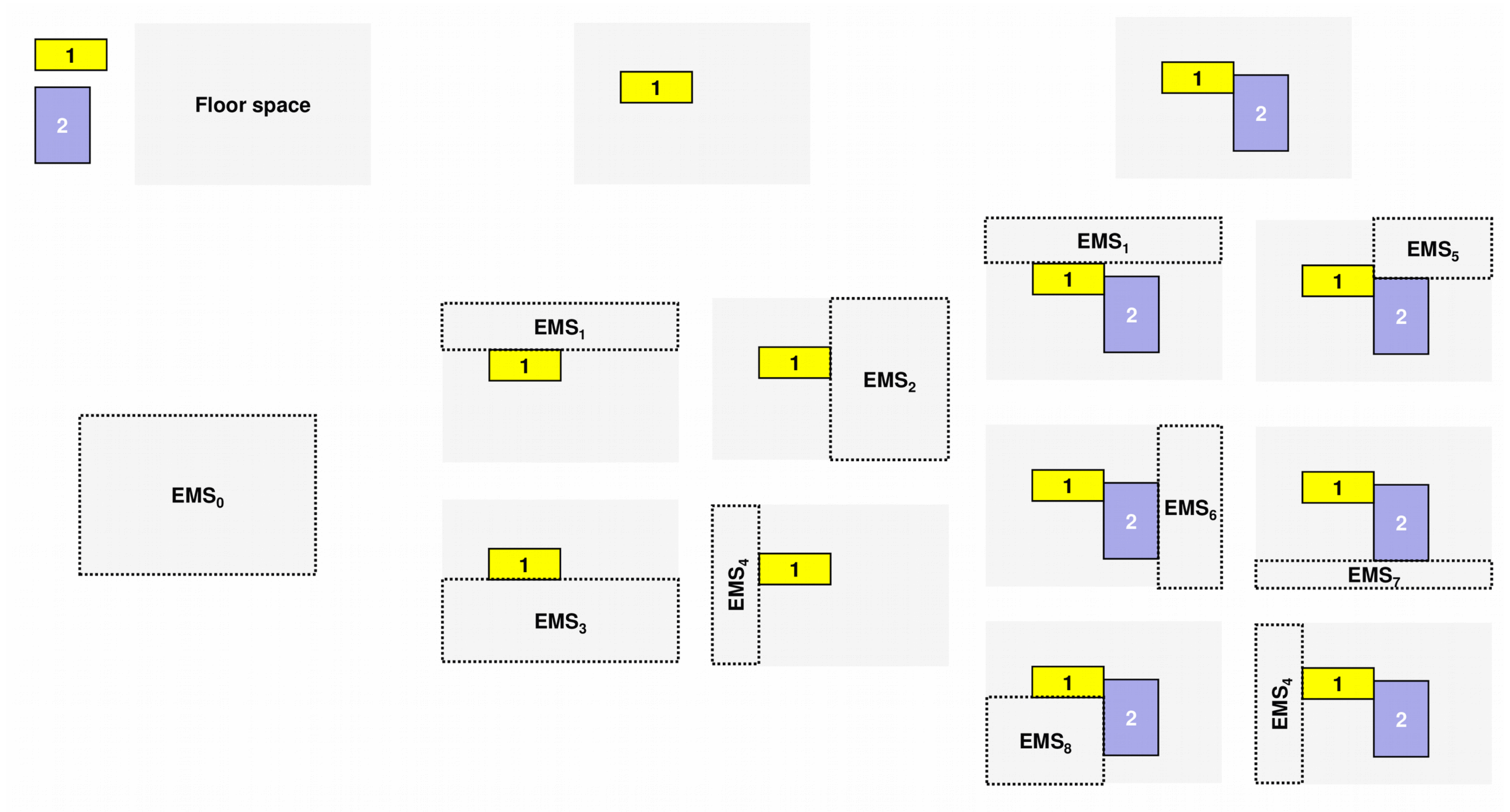
Decoder: Step 3

Use X_{2N+1}, X_{2N+2} to determine the (x, y) coordinates of the first facility to be placed on the floor space.

$$x = (w_i/2) + X_{2N+1} \times (W - w_i)$$

$$y = (h_i/2) + X_{2N+2} \times (H - h_i)$$

Decoder: Step 4 Makes use of empty maximal-spaces (EMS)

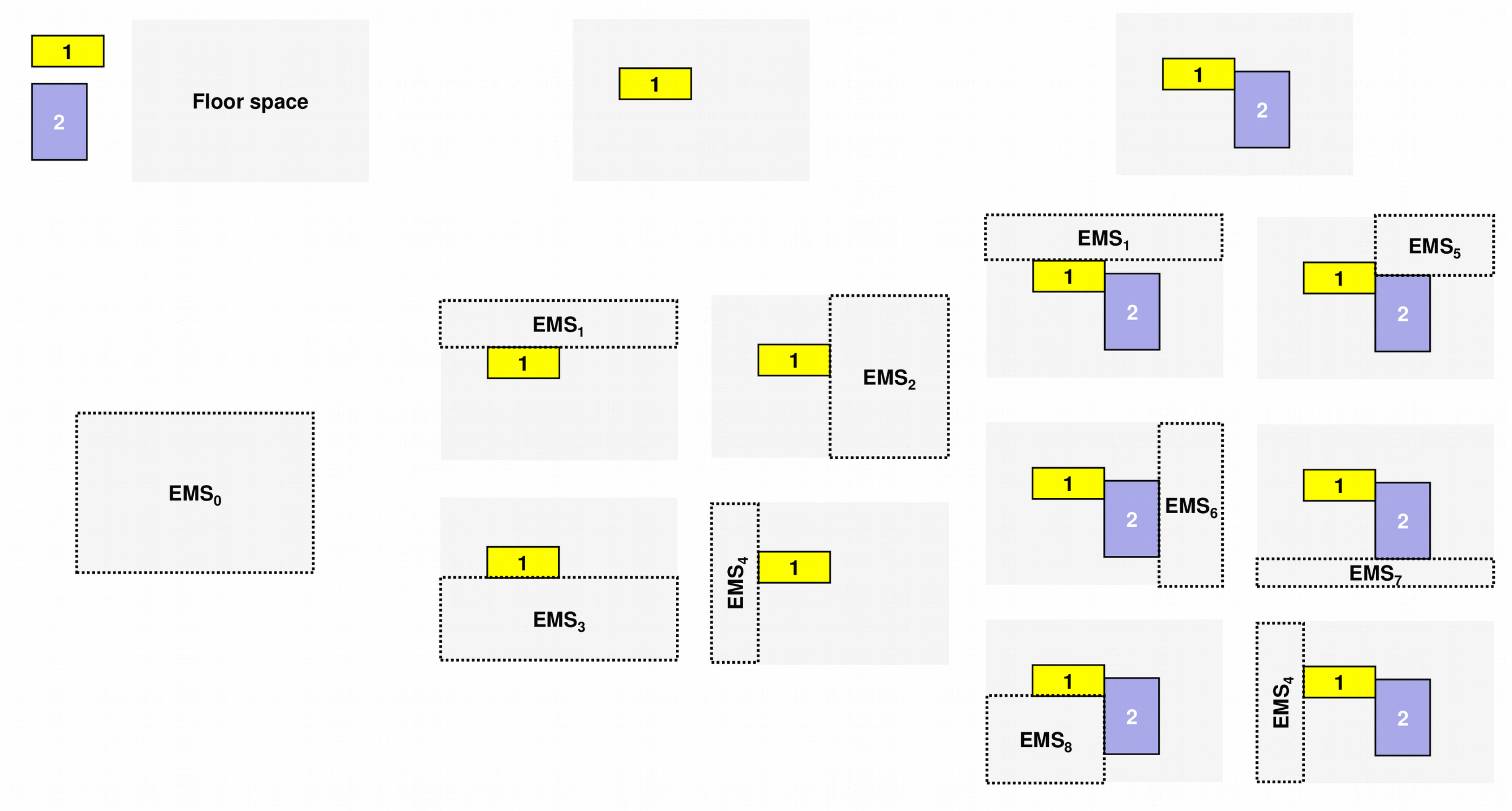


a) Facilities to be placed and the initial empty maximal-space (the floor space)

b) Empty maximal-spaces after placing facility 1.

c) Empty maximal-spaces after placing facility 2.

Decoder: Step 4 When placing a facility we only consider EMSs where the facility fits. This way we avoid overlapping.

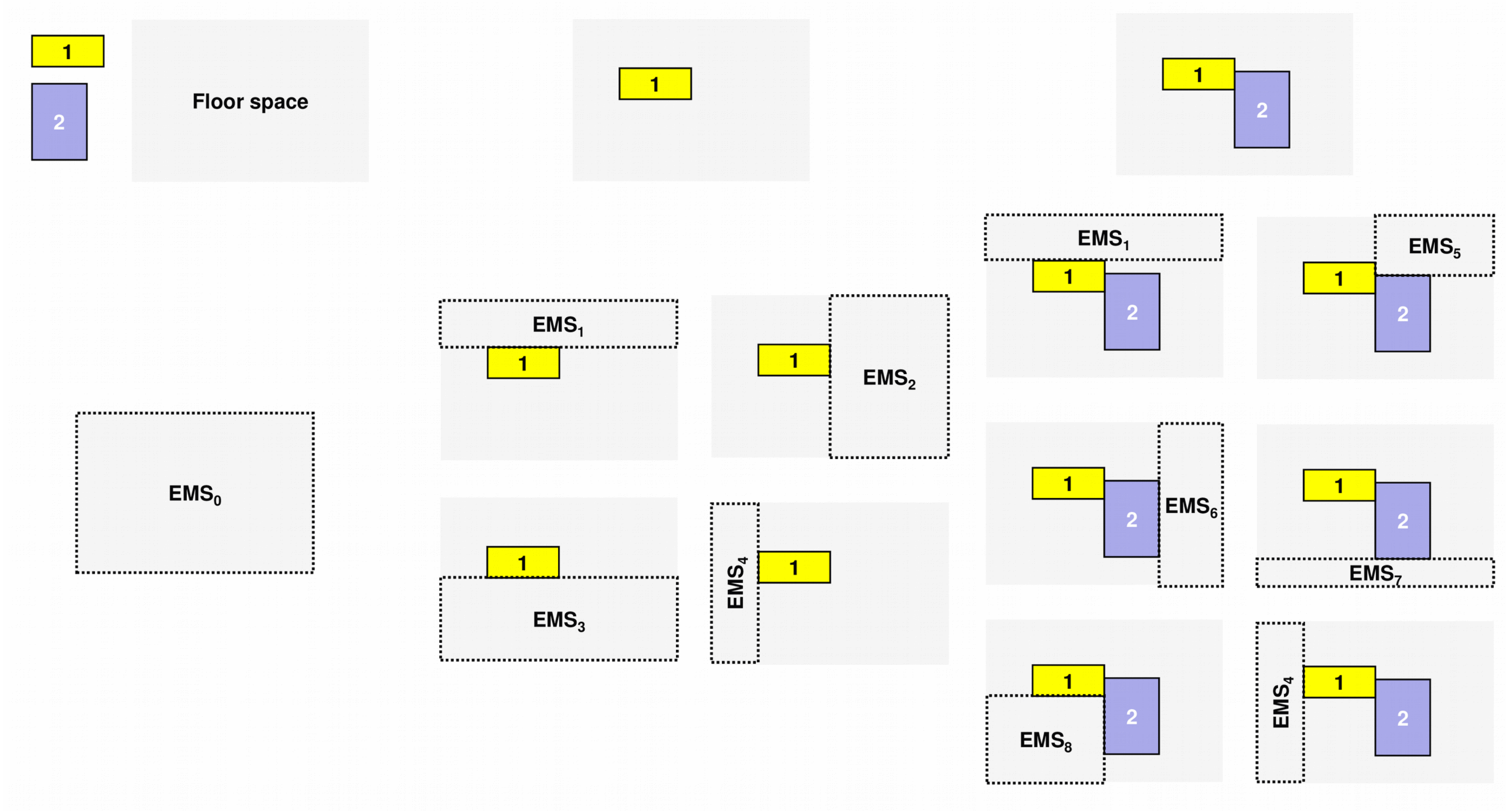


a) Facilities to be placed and the initial empty maximal-space (the floor space)

b) Empty maximal-spaces after placing facility 1.

c) Empty maximal-spaces after placing facility 2.

Decoder: Step 4 EMSs are generated and kept track of with the Difference Process (DP) of Lai and Chan (1997).



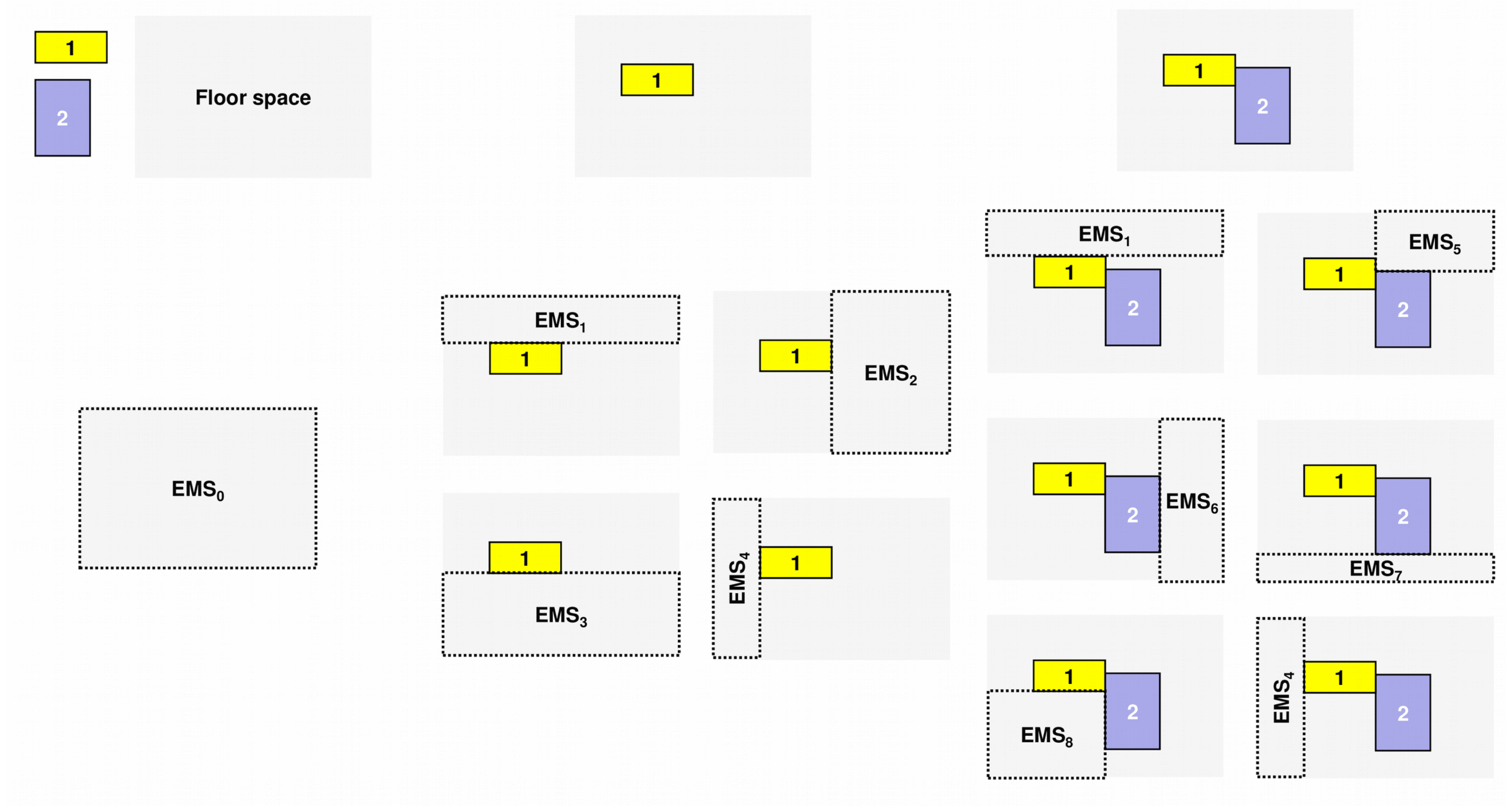
a) Facilities to be placed and the initial empty maximal-space (the floor space)

b) Empty maximal-spaces after placing facility 1.

c) Empty maximal-spaces after placing facility 2.

Decoder: Step 4

Recall that in the unconstrained case the floor space can include all facilities laid out horizontally or vertically.



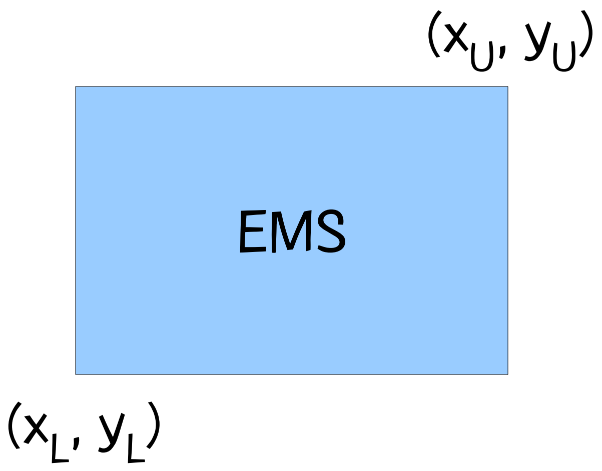
a) Facilities to be placed and the initial empty maximal-space (the floor space)

b) Empty maximal-spaces after placing facility 1.

c) Empty maximal-spaces after placing facility 2.

Decoder: Step 4

For each EMS in which the facility fits, we compute the incremental cost associated with placing the facility in that EMS and then place it in the least-cost EMS.



Compute positions that minimize cost of placing facility i in each available EMS $\{(x_L, y_L), (x_U, y_U)\}$ w.r.t. all already-placed facilities K:

$$\min \sum_{k \in K} c_{i,k} \times f_{i,k} \times d_{i,k}$$

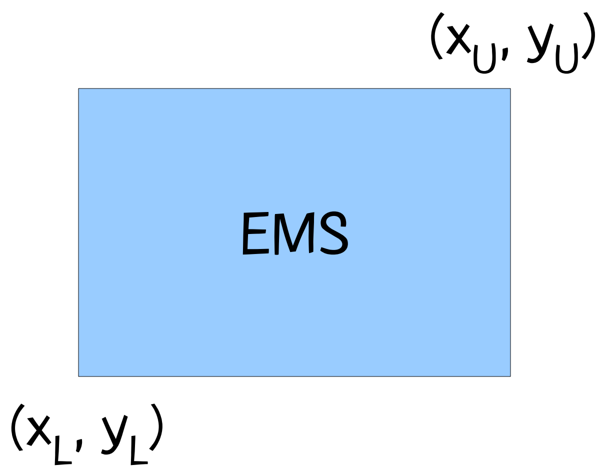
subject to:

$$x_L + w_i/2 \leq x_i \leq x_U - w_i/2$$

$$y_L + h_i/2 \leq y_i \leq y_U - h_i/2$$

Decoder: Step 4

For each EMS in which the facility fits, we compute the incremental cost associated with placing the facility in that EMS and then place it in the least-cost EMS.



Compute positions that minimize cost of placing facility i in each available EMS $\{(x_L, y_L), (x_U, y_U)\}$ w.r.t. all already-placed facilities K:

$$\min \sum_{k \in K} c_{i,k} \times f_{i,k} \times d_{i,k}$$

subject to:

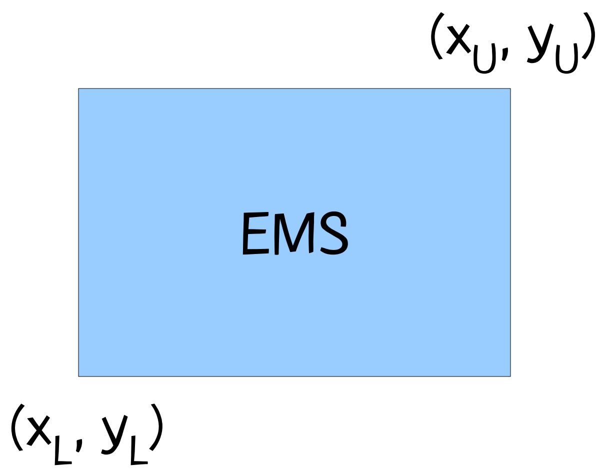
$$x_L + w_i/2 \le x_i \le x_U - w_i/2$$

$$y_L + h_i/2 \le y_i \le y_U - h_i/2$$

Instead of solving this directly with a NLP solver we propose a different approach.

Decoder: Step 4

For each EMS in which the facility fits, we compute the incremental cost associated with placing the facility in that EMS and then place it in the least-cost EMS.



Find the unconstrained optimum (UO) using a method described in Heragu (1997):

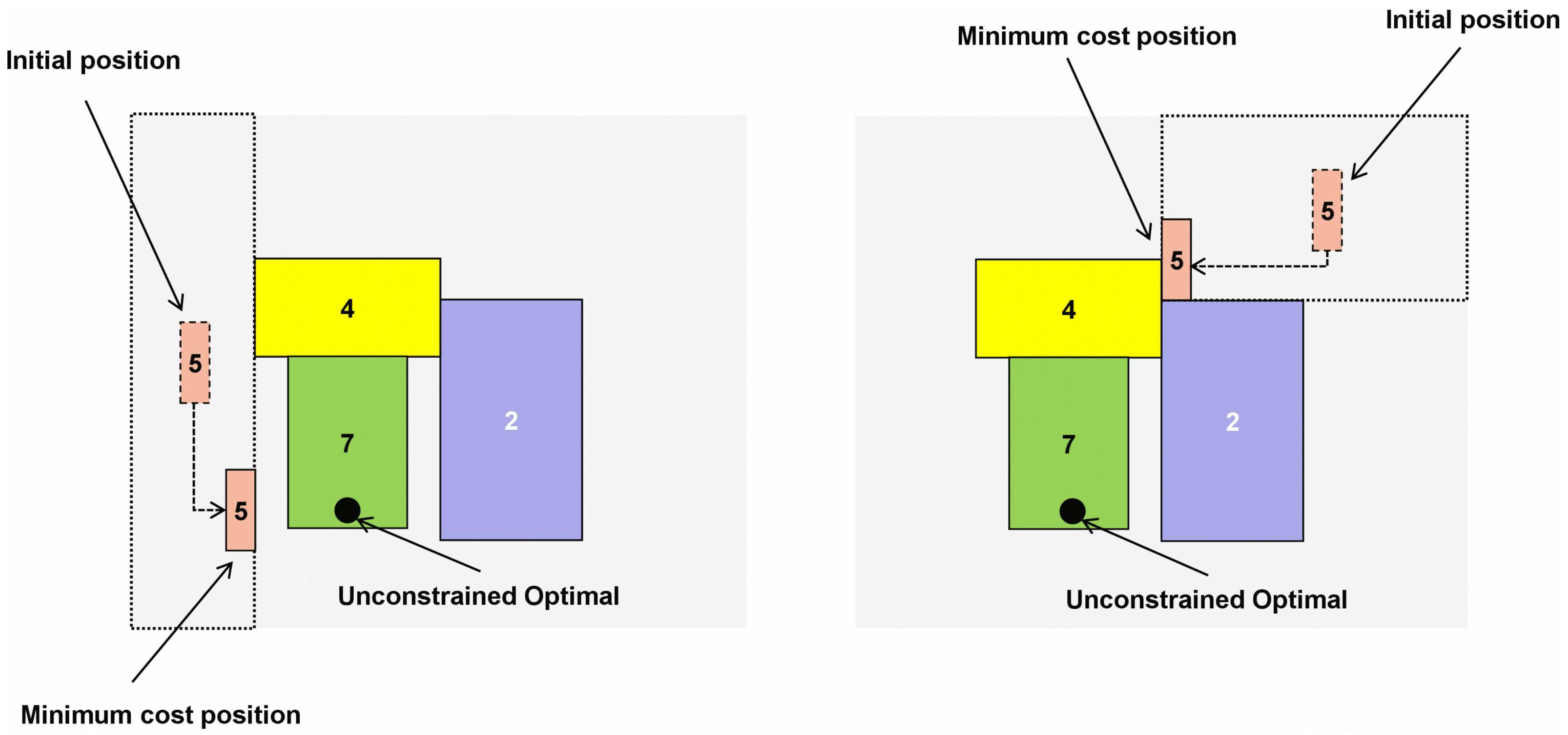
$$\min \sum_{k \in K} c_{i,k} \times f_{i,k} \times d_{i,k}$$

If there is no flow between facility i and the already laid-out facilities, then UO is assumed to be geometric center of all laid-out facilities.

Tentatively place facility i in the geometric center of each EMS in which it fits.

Decoder: Step 4

For each EMS in which the facility fits, we place the facility in the center of the EMS and move it as close as possible to the UO and compute the objective.



Experimental results – Unconstrained

We compare our BRKGA with eight algorithms:

1) Hierarchical approach with clusters (HA-C) of Tam and Li (1991)

2) GA with slicing tree structure (GA-STS) of Kado (1996)

3) Genetic programming algorithm (GP-STS) of Garces-Perez et al. (1996)

Experimental results – Unconstrained

We compare our BRKGA with eight algorithms:

4) GA with tree-structured genotype representation (GA-TSG) of Schnecke and Vornberger (1997)

5) Tabu search with slicing tree (TSaST) of Scholtz et al. (2009)

6) Commercial solver from Engineering Optimization Software (VIP-PLANOPT) based on algorithms of Mir and Imam (1996, 2001) and Imam and Mir (1998)

Experimental results – Unconstrained

We compare our BRKGA with eight algorithms:

7) Tabu search with boundary search technique (TS-BST) of McKendall Jr. and Hakobyan (2010)

8) The MIP solver from Gurobi Optimization (Gurobi) version 5.5.

Experimental results – Unconstrained

Benchmark instances:

- Seven L instances of Imam and Mir (1993, 1998), Mir and Imam (1996, 2001), and VIP-PLANOPT (2006, 2010) with 20 to 125 facilities
- Dunker62 instance of Dunker et al. (2003) with 62 facilities
- Eight TL instances of Tam and Li (1991) with 5 to 30 instances
- 100 random (RND) instances with known optimal with 10 to 100 facilities of Gonçalves & R. (2014)

Experimental results – Unconstrained

Computational setup:

– BRKGA coded in C++

– Experiments run on a computer with an Intel Xeon E5-2630 processor at 2.30 GHz and 16 GB of RAM running Linux O.S. (Fedora, release 18)

– BRKGA parameters

- Population size: $p = 100 \times N$
- Elite population: min ($0.25 \times p$, 50)
- Mutation population: $0.25 \times p$
- Inheritance probability: 0.70
- Stopping rule: 50 generations

Experimental results – Unconstrained

	VIP-PLANOPT		TSaST		TS-BST		BRKGA		
Dataset	Cost	Time	Cost	Time	Cost	Time	Cost	Time	%Impr
L20	1.13E3	0.3	-	-	1.15E3	10351.9	1.13E3	0.5	1.86
L28	6.45E3	1.5	-	-	-	-	6.01E3	1.0	6.72
L50	7.82E4	7.0	-	-	7.13E4	7626.5	6.94E4	6.3	2.65
L75	3.44E4	13.0	-	-	-	-	3.15E4	11.6	8.47
L100	5.38E5	14.0	-	-	4.97E5	11397.2	4.79E5	57.0	3.60
L125A	2.89E5	110.0	-	-	-	-	2.57E5	83.6	11.05
L125B	1.08E6	70.0	-	-	1.01E6	9250.3	9.43E5	118.7	6.51
Dunker62	3.94E6	4996.0	3.87E6	252.0	3.81E6	7304.1	3.69E6	9.1	3.35

Times are in seconds

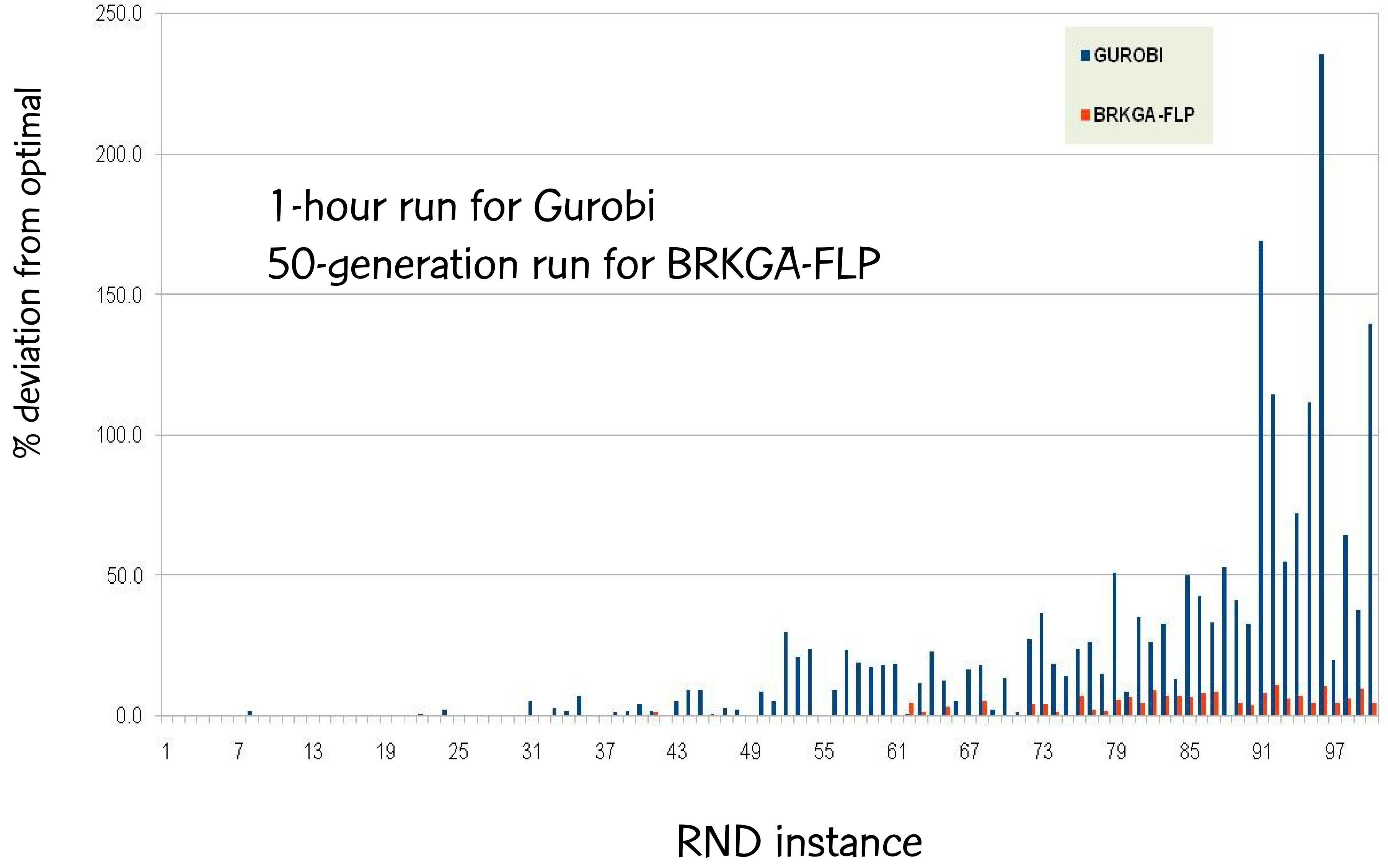
Experimental results – Unconstrained

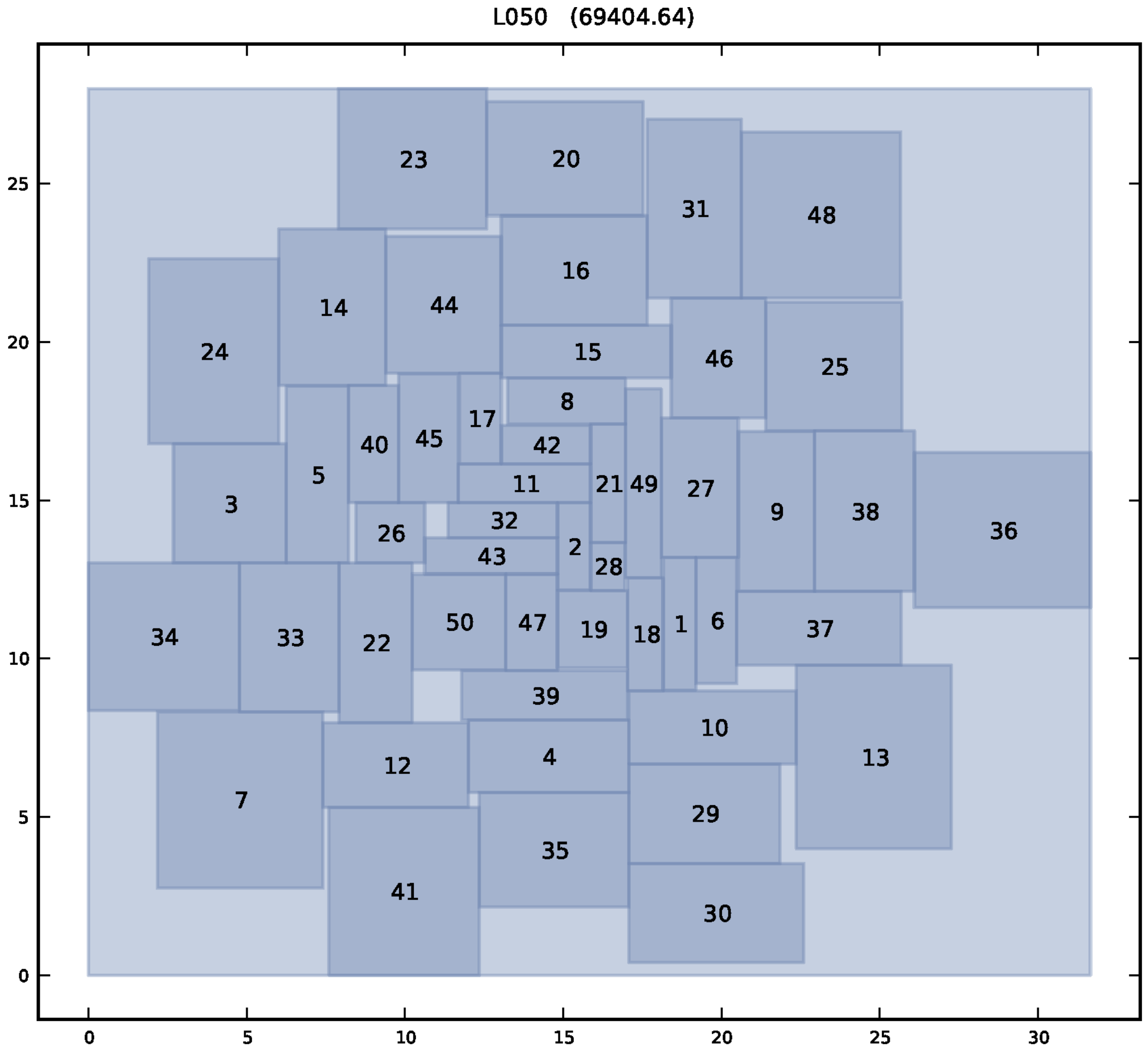
	HA-C	GA-STS	GP-STS	GA-TSG	TSaST		BRKGA		
Dataset	Cost	Cost	Cost	Cost	Cost	Time	Cost	Time	%Impr
TL05	247	228	226	214	213.5	2.3	210.1	0.035	1.60
TL06	514	361	384	327	348.8	3.0	345.0	0.049	(5.51)
TL07	559	596	568	629	562.9	2.5	549.7	0.060	1.67
TL08	839	878	878	833	810.4	4.7	799.1	0.080	1.40
TL12	3162	3283	3220	3164	3054.2	12.5	2920.5	0.162	4.38
TL15	5862	7384	7510	6813	6615.8	17.0	6395.4	0.251	(9.10)
TL20	-	16393	14033	13190	13198.4	50.0	9892.4	0.443	25.00
TL30	-	41095	39018	25358	33721.5	95.4	31454.2	1.132	6.72

Experimental results – Unconstrained (% deviation from optimum)

Each dataset consists of 10 instances, each with known optimum.

	Gurobi			BRKGA		
Dataset	**Time**	**Avg % Dev**	**Max % Dev**	**Time**	**Avg % Dev**	**Max % Dev**
RND10	3600	0.21	1.66	1.76	0.00	0.00
RND20	3600	0.01	0.12	6.13	0.00	0.00
RND30	3600	0.32	2.14	15.00	0.00	0.00
RND40	3600	2.37	7.10	28.67	0.00	0.00
RND50	3600	3.99	9.30	48.30	0.11	1.12
RND60	3600	16.65	29.73	72.86	0.02	0.15
RND70	3600	12.21	22.70	102.90	1.44	5.29
RND80	3600	22.31	50.97	143.37	3.31	7.10
RND90	3600	36.11	52.99	186.87	6.00	9.09
RND100	3600	101.78	235.31	235.84	7.36	10.97


250.0
200.0
150.0
100.0
50.0
0.0
% deviation from optimal
GUROBI
BRKGA-FLP
1-hour run for Gurobi
50-generation run for BRKGA-FLP
1
7
13
19
25
31
37
43
49
55
61
67
73
79
85
91
97
RND instance

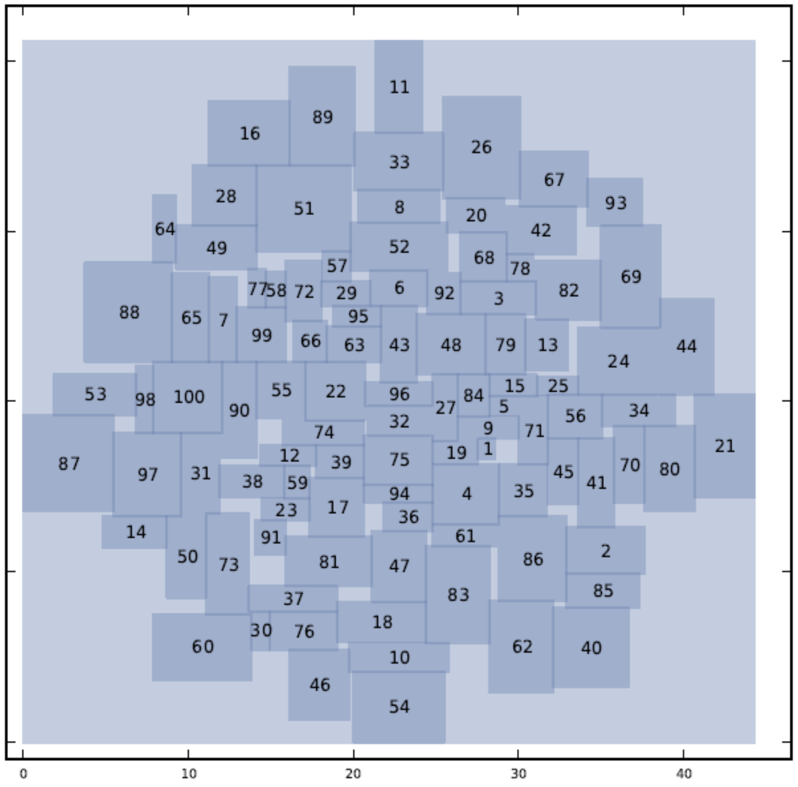


L050

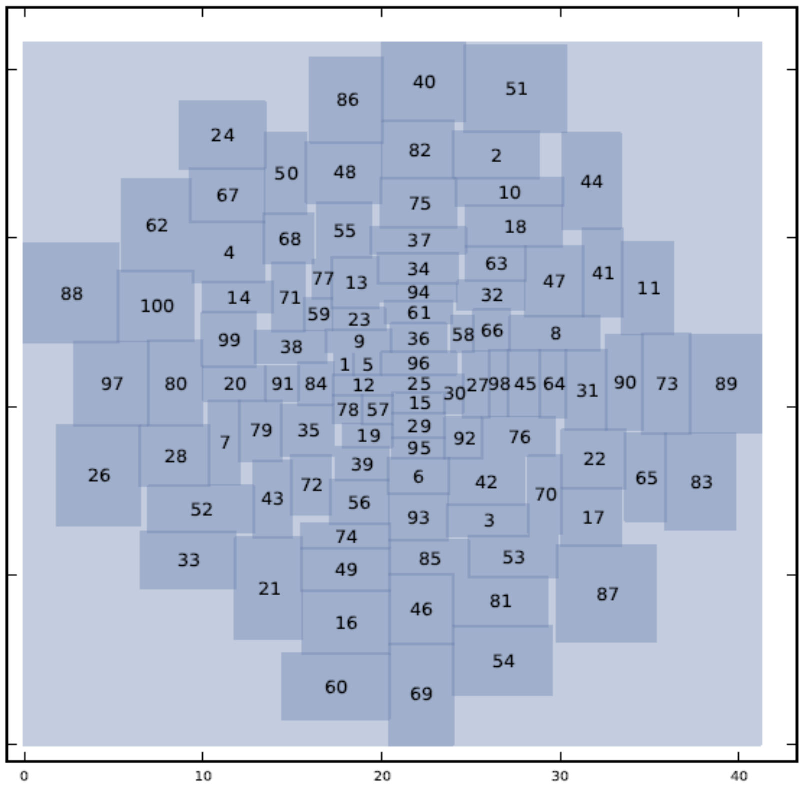
New best known Solution: 6.94E4

Previous best known Solution: 7.13E4 TS-BST (McKendall Jr. & Hajobyan, 2010)

1st generation: 530404.76

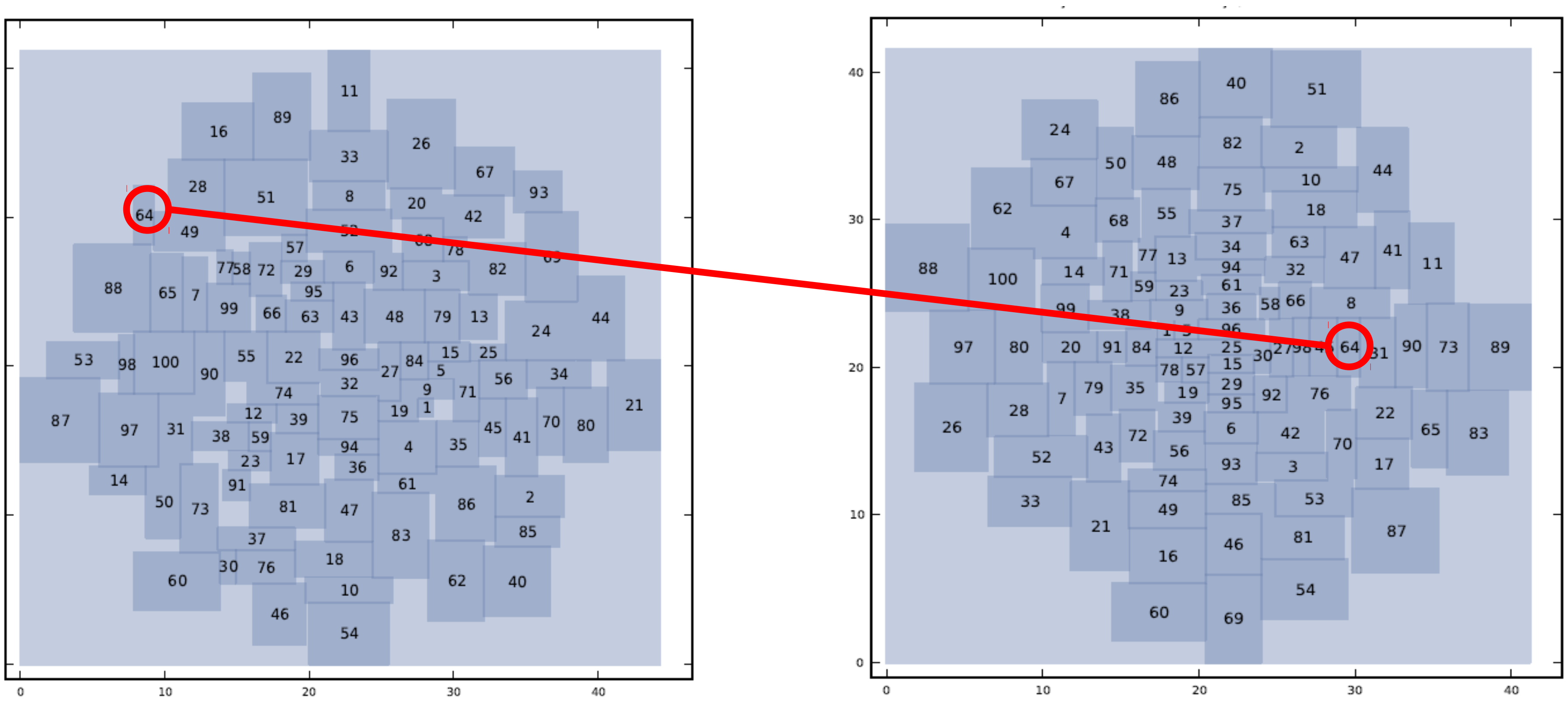


50th generation: 478910.09



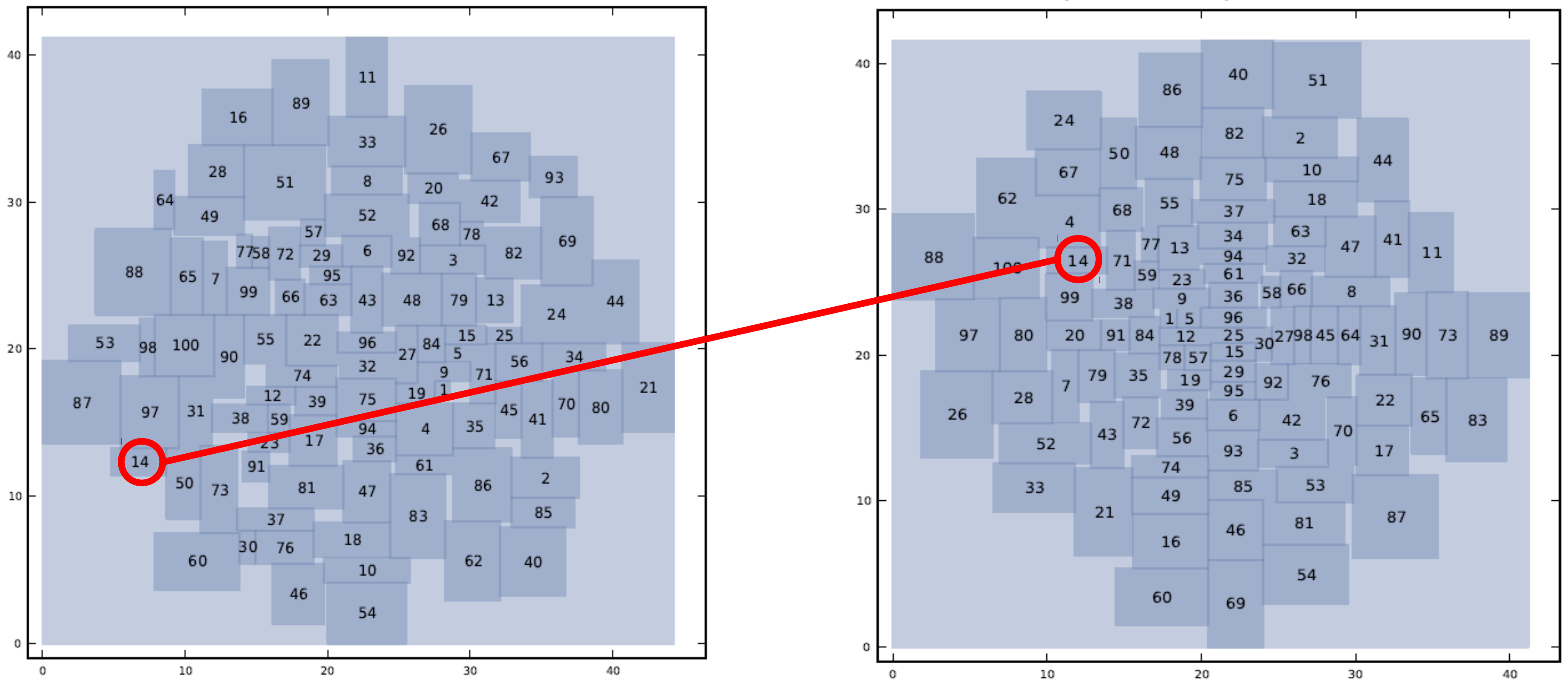
L100

1st generation: 530404.76 50th generation: 478910.09



L100

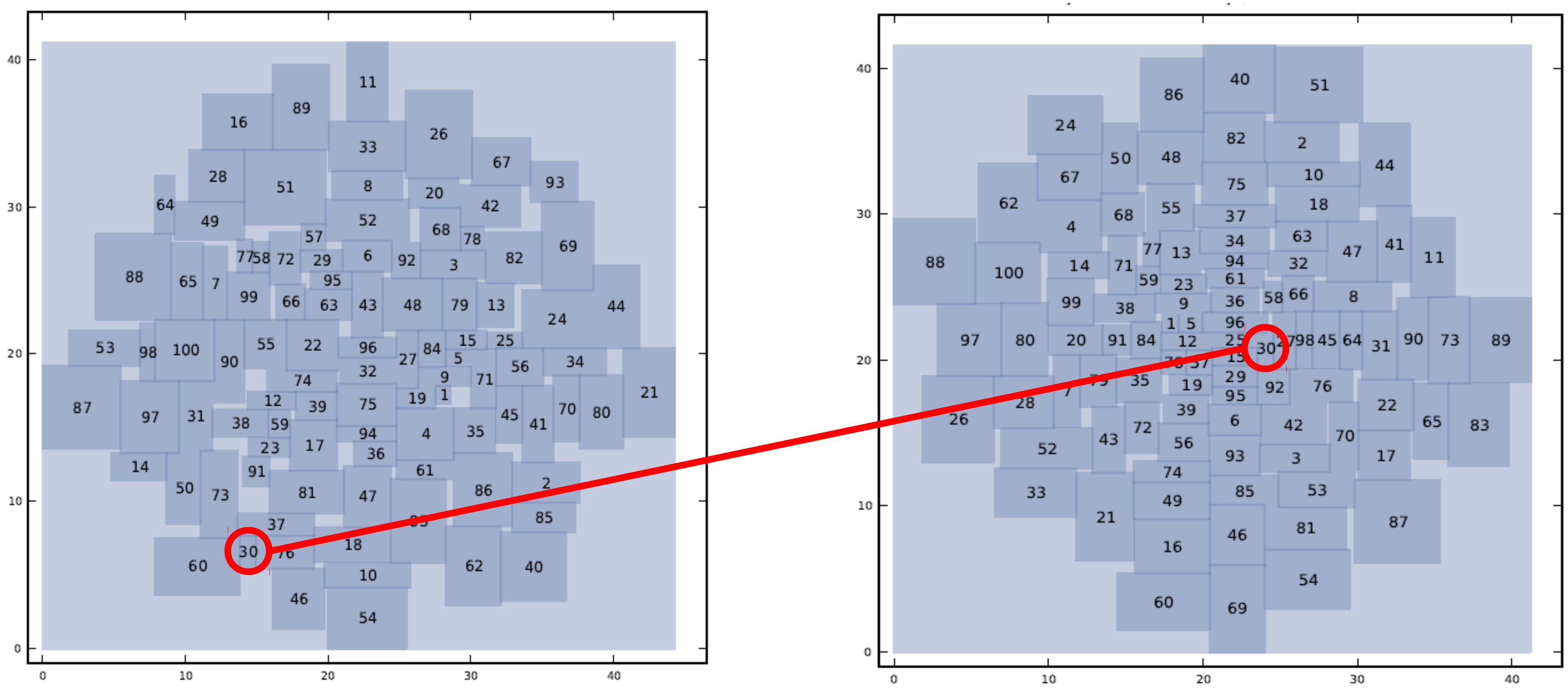
1st generation: 530404.76 50th generation: 478910.09



L100

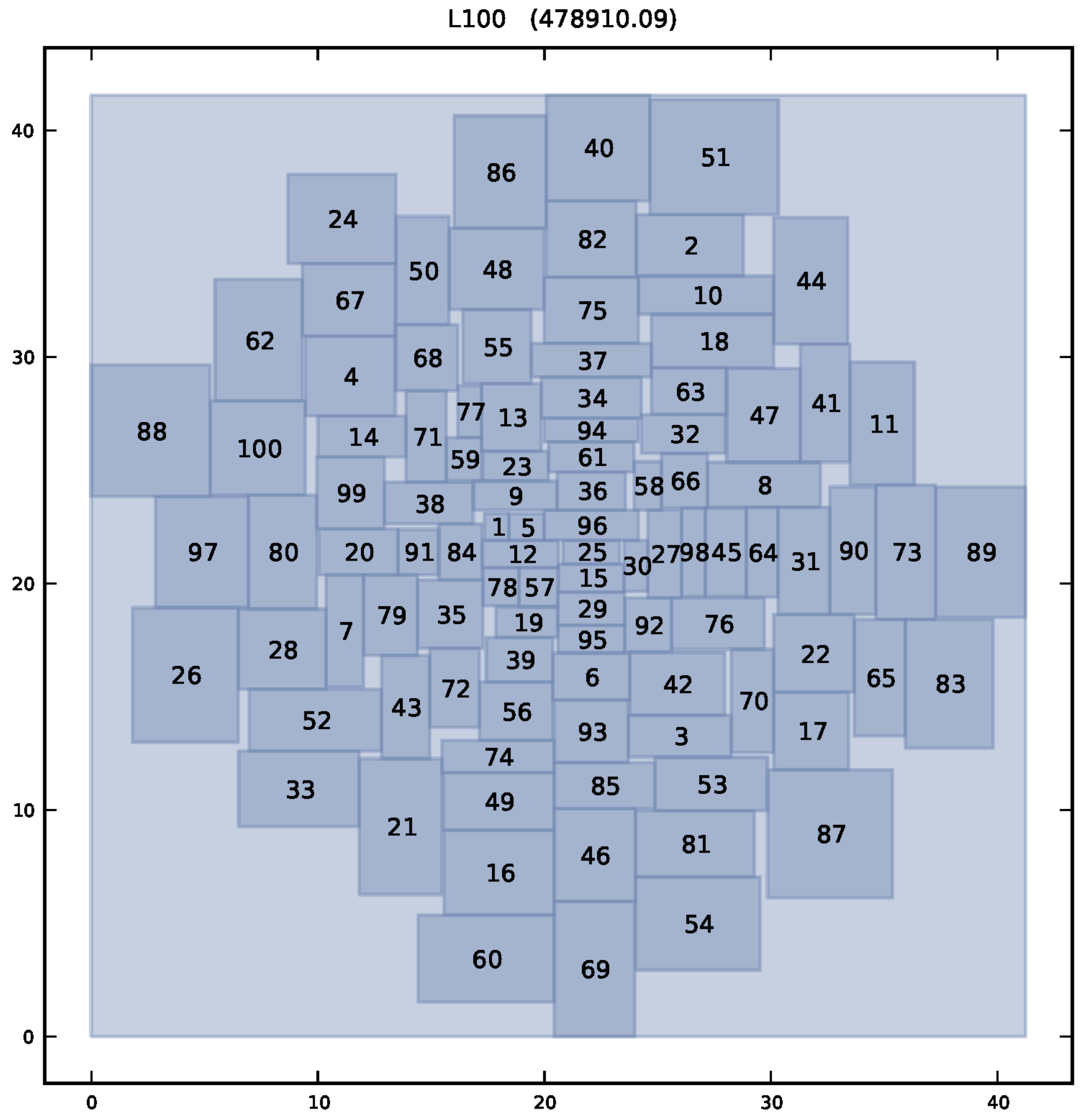
amazon

1st generation: 530404.76 — 50th generation: 478910.09



L100

L100

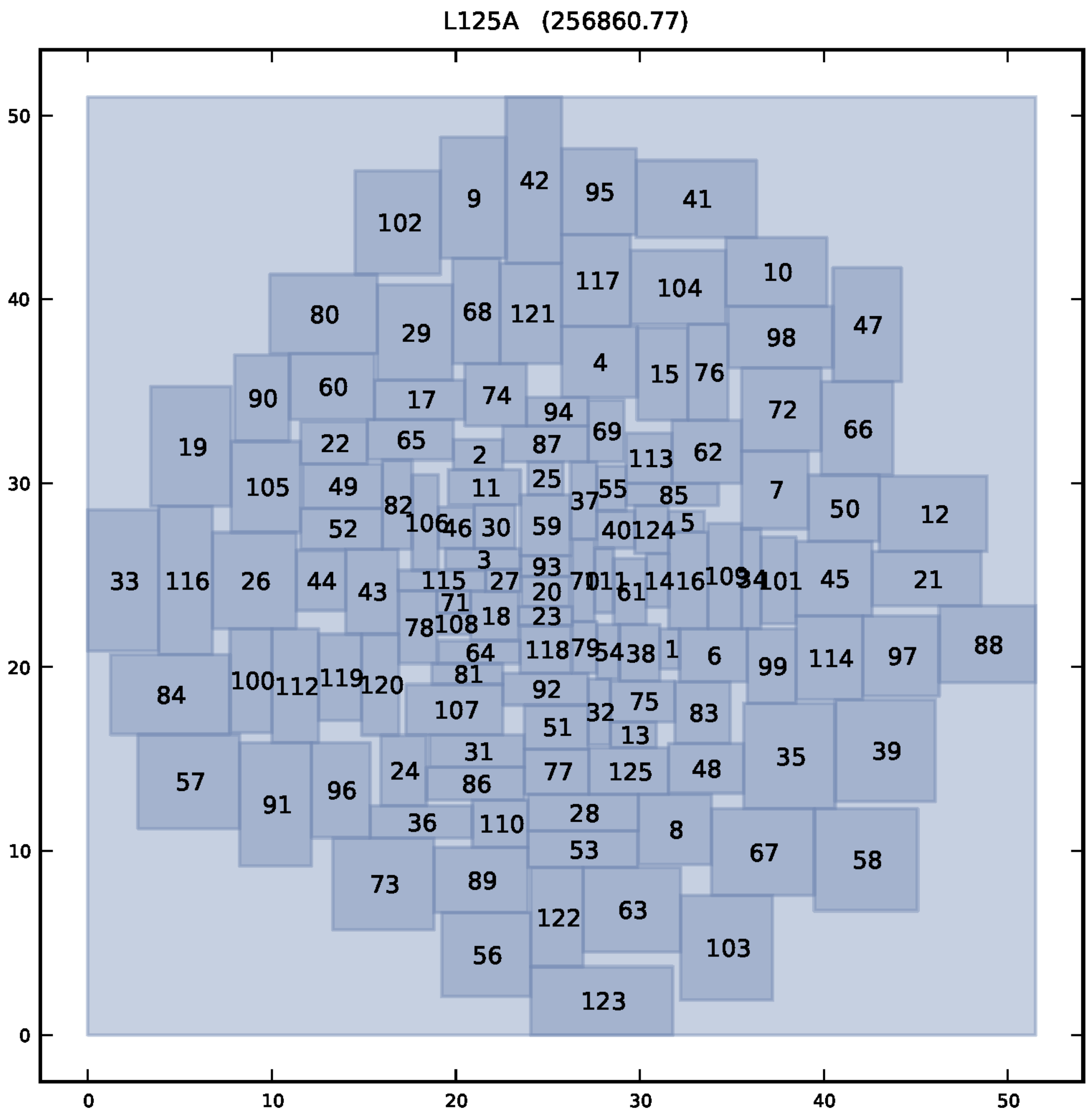


New best known Solution: 4.79E5

Previous best known Solution: 4.97E5
TS-BST (McKendall Jr. & Hajobyan, 2010)

BRKGA

amazon



L125A

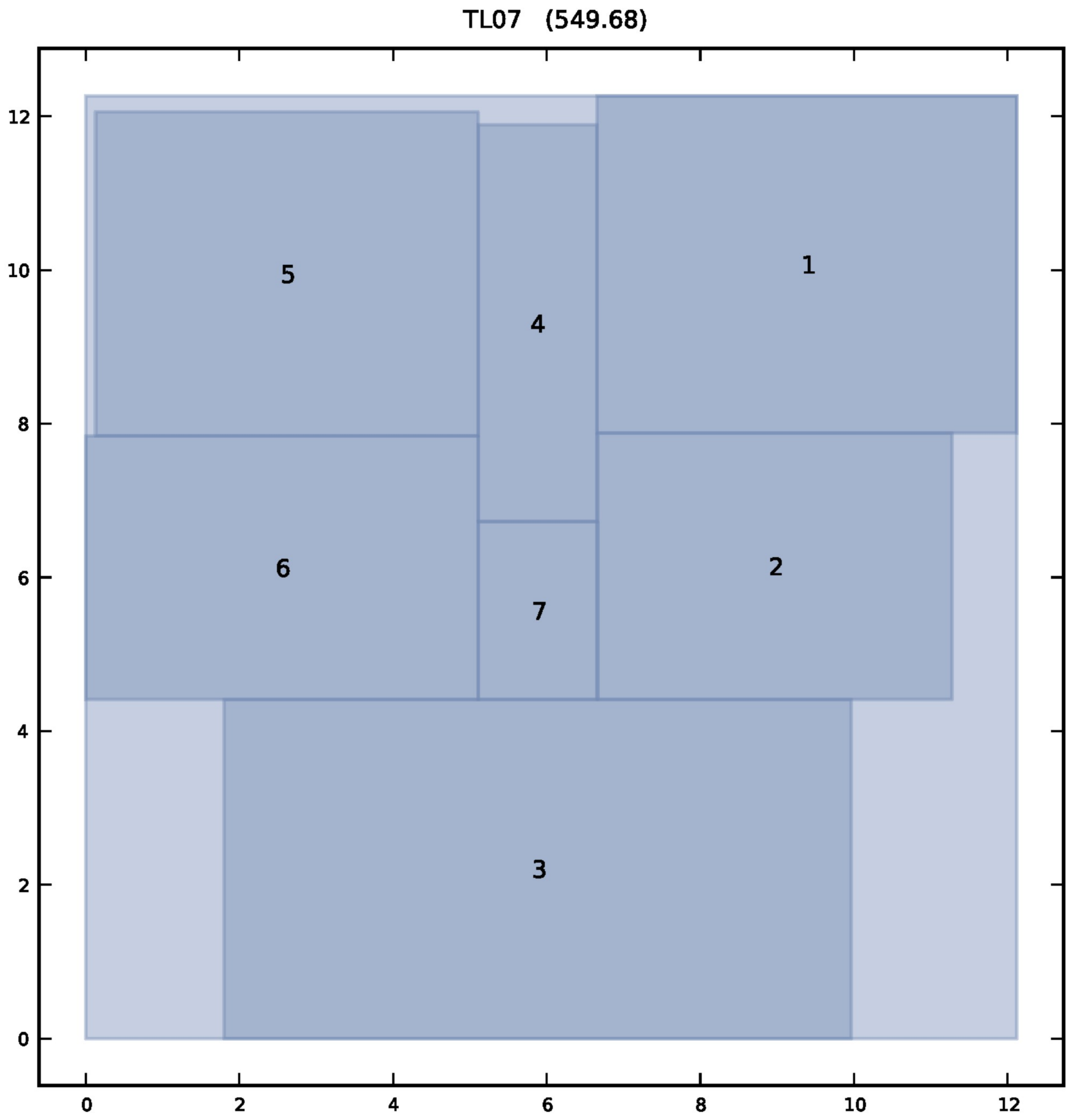
New best known

Solution: 2.57E5

Previous best known

Solution: 2.89E5

VIP-PLANOPT (2010)



TL07

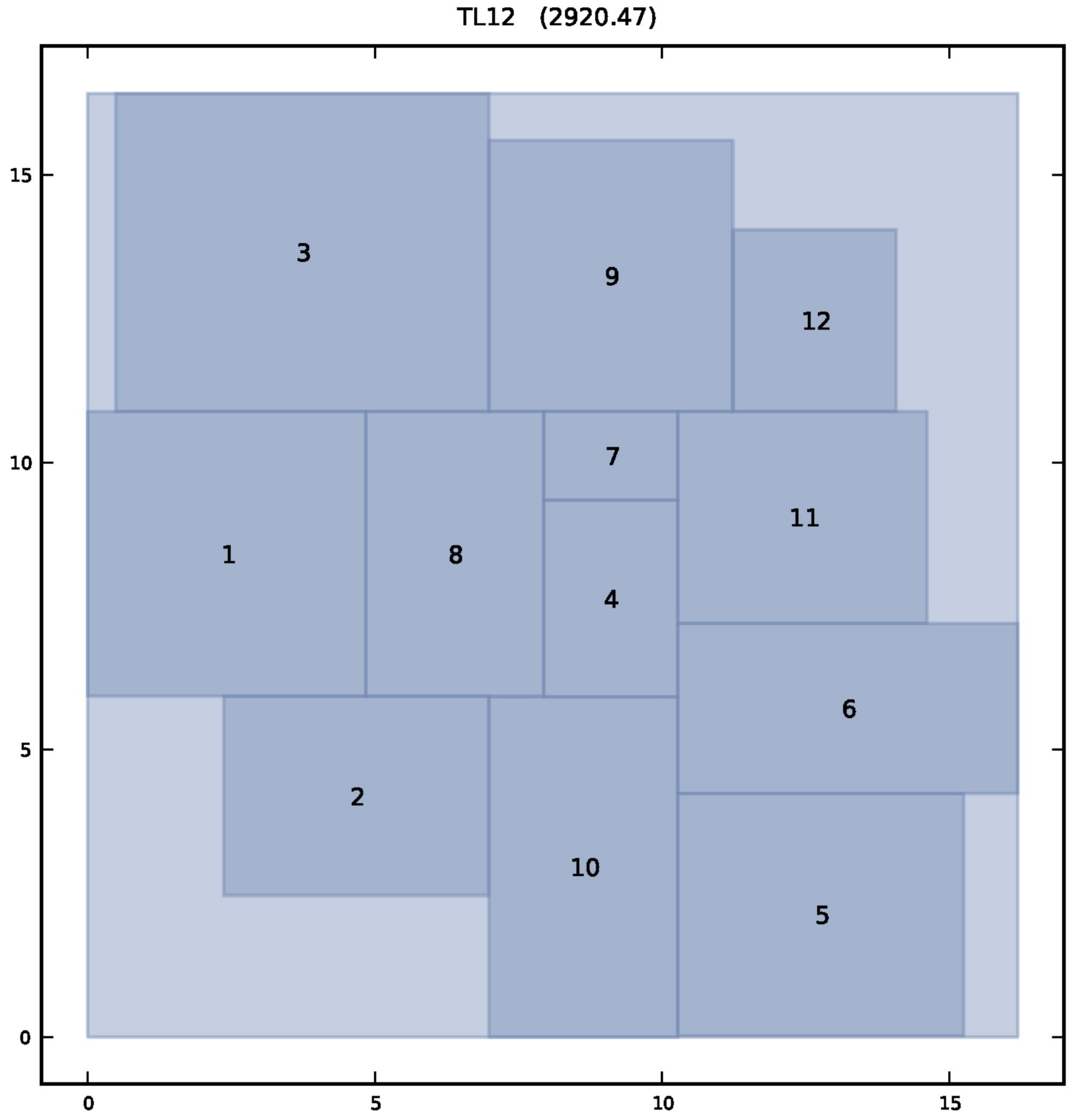
New best known
Solution: 549.7

Previous best known
Solution: 559.0
HA-C (Tam and Li, 1991)

BRKGA

UFRGS – Porto Alegre, Brazil ✤ Dec. 9, 2015

amazon



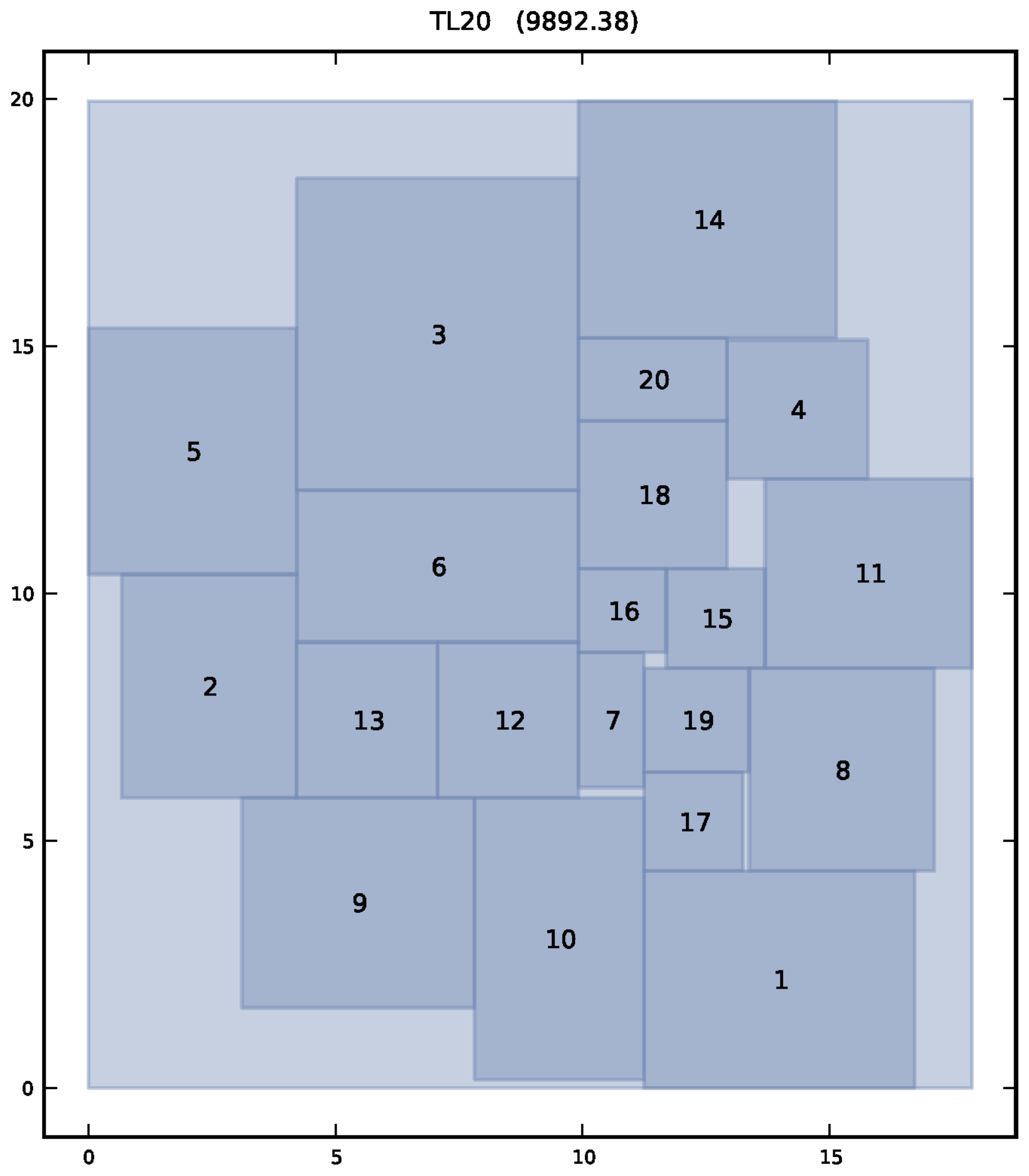
TL12

New best known
Solution: 2920.5

Previous best known
Solution: 3054.2
TSaST (Scholtz et al., 2009)

BRKGA

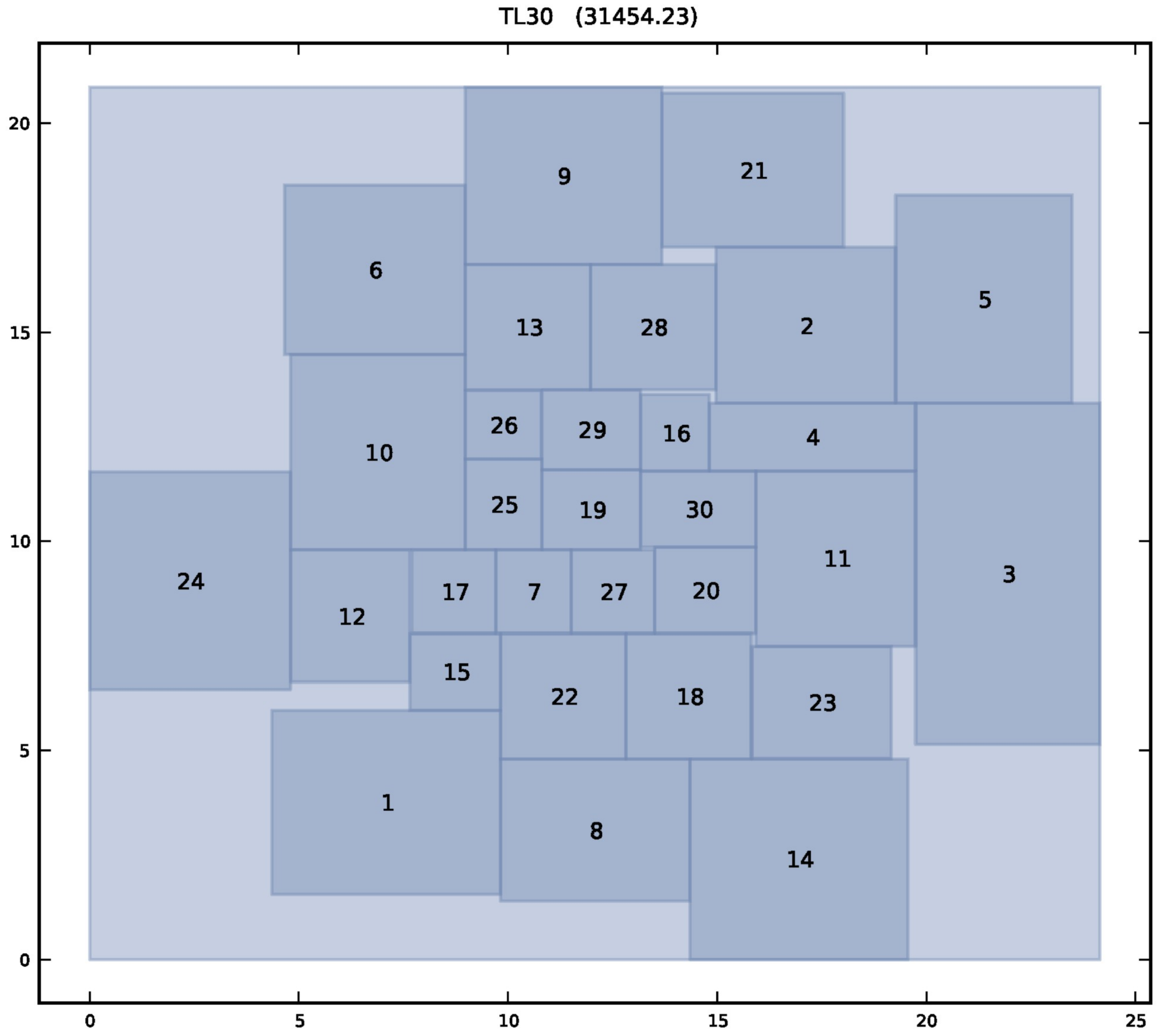
amazon



TL20

New best known Solution: 9892.4

Previous best known Solution: 13190.0 GA-TSG (Schnecke and Vornberger, 1997)



TL30

New best known
Solution: 31454.2

Previous best known
Solution: 33721.5
TSaST (Scholtz et al., 2009)

Other applications of BRKGA

Telecommunications

- Weight setting in OSPF routing (Ericsson et al., 2002; Buriol et al., 2005; Reis et al., 2011)
- Survivable network design (Andrade et al., 2006; Buriol et al., 2007; Ruiz et al., 2015; Andrade et al., 2015)
- Facility location (Breslau et al., 2011; Morán-Mirabal et al., 2013; Duarte et al., 2014; Stefanello et al., 2015)
- Routing & wavelength assignment (Noronha et al., 2011)
- Assignment of virtual machines to datacenters (Stefanello et al., 2015)
- Design of wireless backhaul network (Andrade et al., 2015)
- Cloud resource management (Heilig et al., 2015)

Other applications of BRKGA

Scheduling

- Job-shop scheduling (Gonçalves et al., 2005; Gonçalves & R., 2014)
- Project scheduling (Gonçalves et al., 2008; 2009; 2011)
- Survey of project scheduling (Gonçalves et al., 2014)
- Field technician scheduling (Damm et al., 2015)
- Scheduling divisible loads (Brandão et al., 2015)
- Scheduling Earth observations with agile satellite (Tangpattanakul et al., 2013)
- Multi-user Earth observation scheduling (Tangpattanakul et al., 2015)

Other applications of BRKGA

Manufacturing and facility layout

- Assembly line balancing (Gonçalves & Almeida, 2002,)
- Manufacturing cell formation (Gonçalves & R., 2004)
- Assembly line worker assignment and balancing (Moreira et al., 2012)
- Minimization of open stacks (Gonçalves et al., 2014)
- Minimization of tool switches (Chaves et al., 2014)
- Unequal area facility layout (Gonçalves & R., 2015)

Other applications of BRKGA

Algorithm engineering

- Automatic tuning of parameters (Festa et al., 2010; Morán-Mirabal et al., 2013)
- Benchmarking (Gonçalves et al., 2014)
- Extensions of BRKGA (Lucena et al., 2014)
- Application programming interface (Toso et al., 2015)

Other applications of BRKGA

Clustering, covering, and packing

– 2D/3D orthogonal packing (Gonçalves & R., 2011; 2012)

– 2D/3D bin packing (Gonçalves and R., 2013)

– Multi-objective 3D container loading (Zheng et al., 2014)

– Steiner triple covering (R. et al., 2014)

– Overlapping correlation clustering (Andrade et al., 2014)

– Winner determination in combinatorial auctions (Andrade et al., 2014)

Other applications of BRKGA

Routing

- Capacitated arc routing (Martinez et al., 2011)
- K-interconnected multi-depot multi-TSP (Andrade et al., 2013)
- Family TSP (Morán-Mirabal et al., 2014)
- Capacitated VRP for blood sample collection (Grasas et al., 2014)

Other applications of BRKGA

Graphs and Trees

– Stochastic Steiner tree (Hokama et al., 2014)

– Capacitated minimum spanning tree (Ruiz et al., 2015)

– Maximum cardinality quasi-clique (Pinto et al., 2015)

Other applications of BRKGA

Toll setting in road networks

– Road congestion minimization (Buriol et al., 2009; 2010; Stefanello et al., 2015)

Other applications of BRKGA

Continuous global optimization

- Bound-constrained GO (Silva et al., 2012)
- Nonlinearly-constrained GO (Silva et al., 2013)
- Python/C++ library for bound-constained GO (Silva et al., 2013)
- Finding multiple roots of system of nonlinear equations (Silva et al., 2014)

Thanks!

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

http://mauricio.resende.info