Biased random-key genetic algorithms: An advanced tutorial

Mauricio G. C. Resende - Amazon.com, Seattle

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

Celso C. Ribeiro - U. Federal Fluminense, Rio de Janeiro

GECCO'16, July 20–24, 2016, Denver, Colorado, USA.
ACM 978-1-4503-4206-3/16/07.
DOI: http://dx.doi.org/10.1145/2908961.2926996

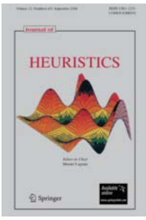
Summary

- Applications
  - 2-dim and 3-dim packing
  - 3-dim bin packing
  - Unequal area facility layout
  - Routing in IP networks
  - Redundant content distribution in IP networks
  - Scheduling divisible loads
- Concluding remarks

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

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 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.

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

- 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.

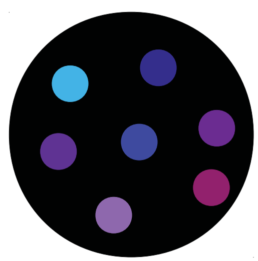
S = (0.25, 0.19, 0.67, 0.05, 0.89)
s(1) s(2) s(3) s(4) s(5)

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

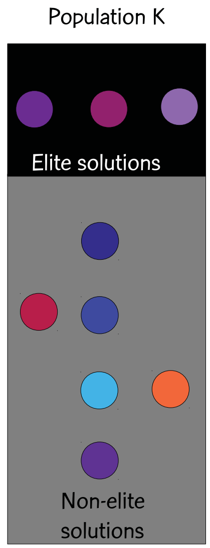
GAs and random keys

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 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.



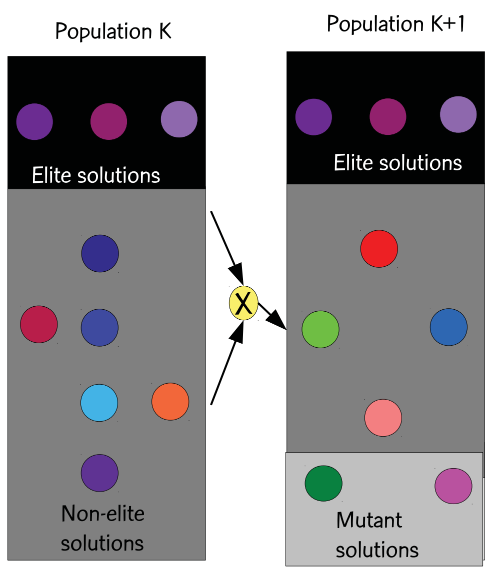
Biased random key genetic algorithm

- 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.

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.



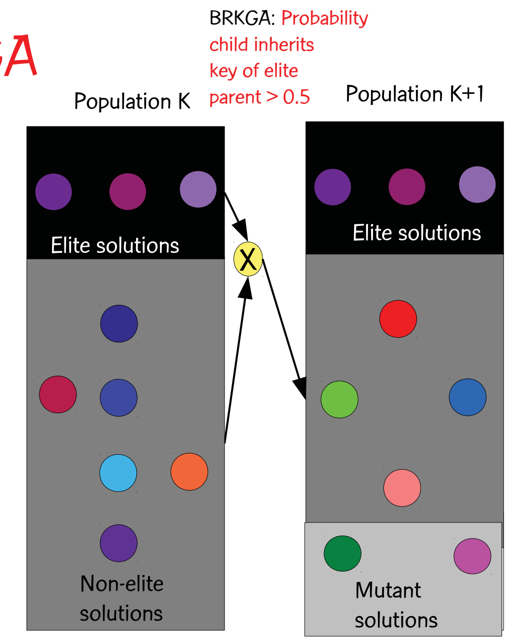
How RKGA & BRKGA differ

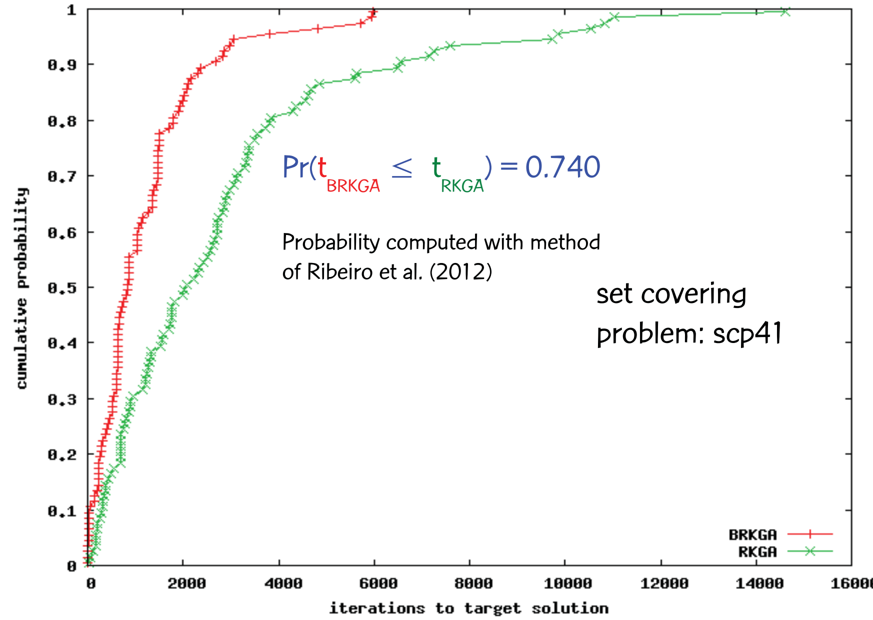
RKGA	BRKGA
both parents chosen at random from entire population	both parents chosen at random but one parent chosen from population of elite solutions
either parent can be parent A in parametrized uniform crossover	best fit parent is parent A in parametrized uniform crossover

Biased random key GA

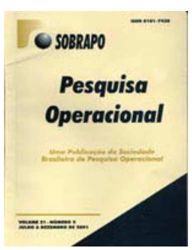
Evolutionary dynamics

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





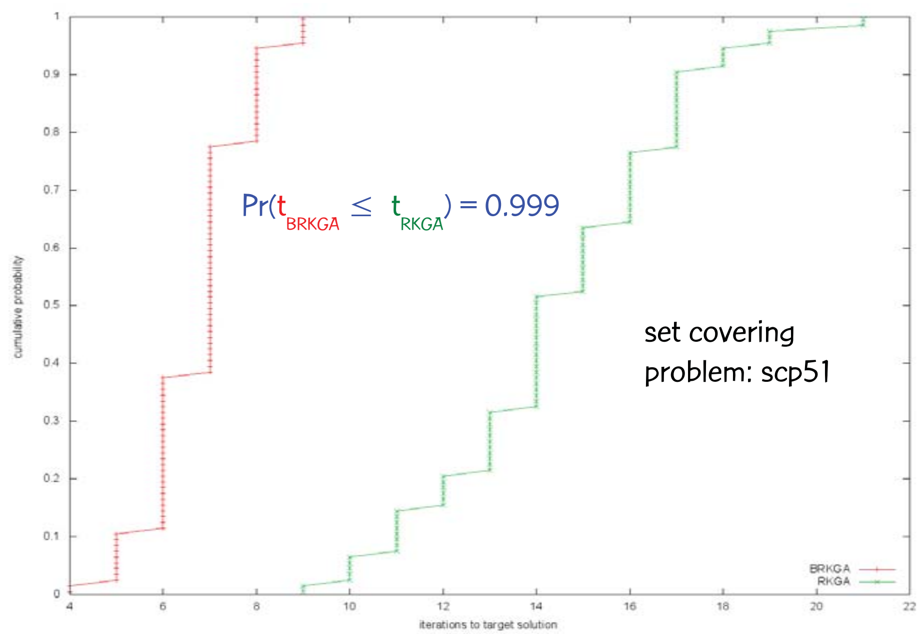
Paper comparing BRKGA and Bean's Method

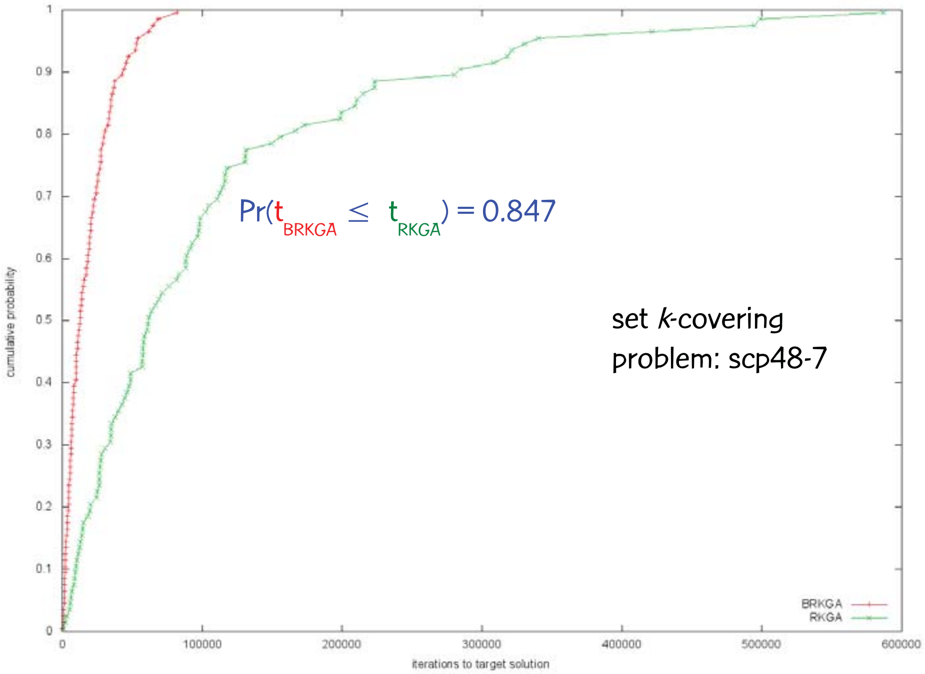


Gonçalves, R., and Toso,

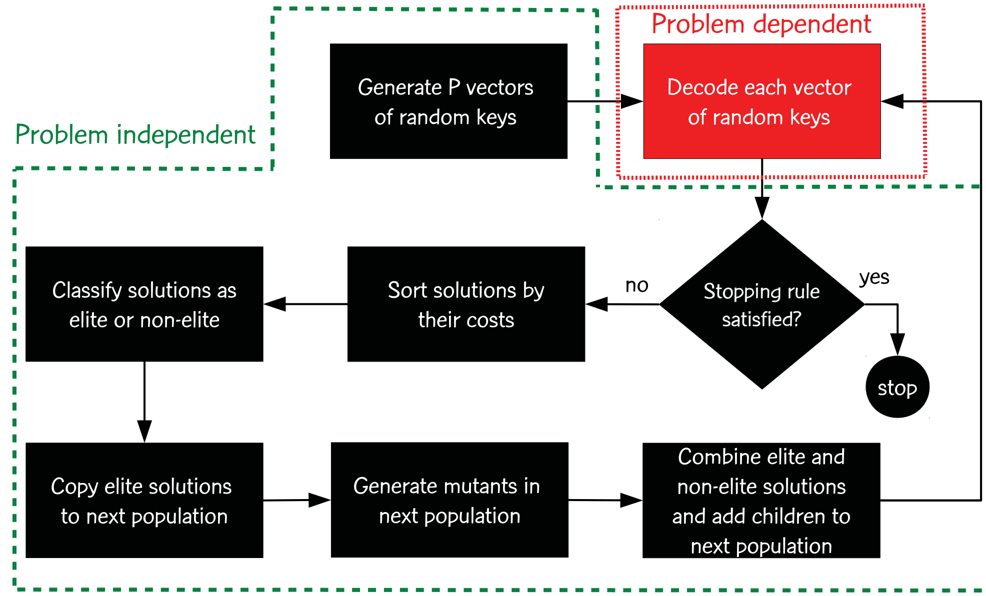
"An experimental comparison of biased and unbiased random-key genetic algorithms",

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





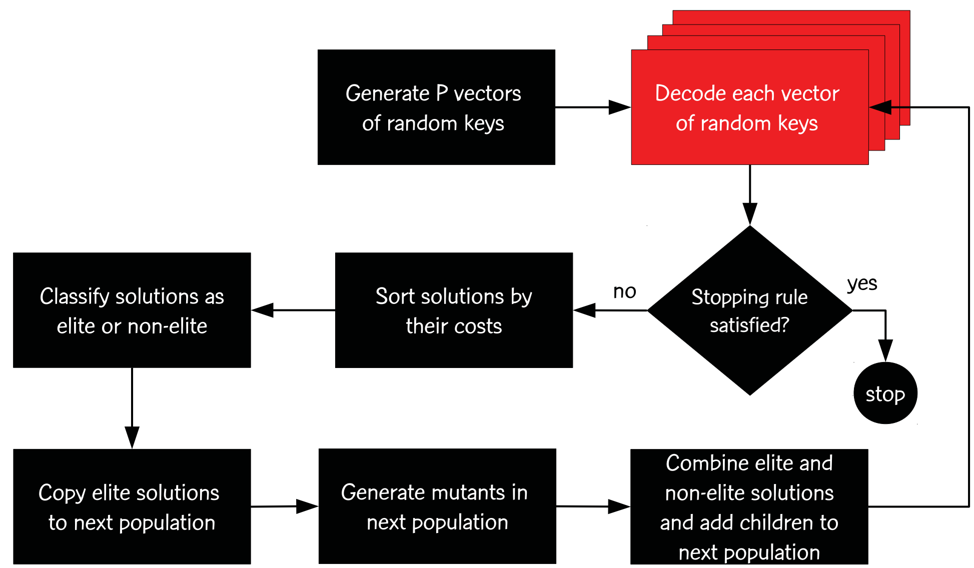
Framework for biased random-key genetic algorithms



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)

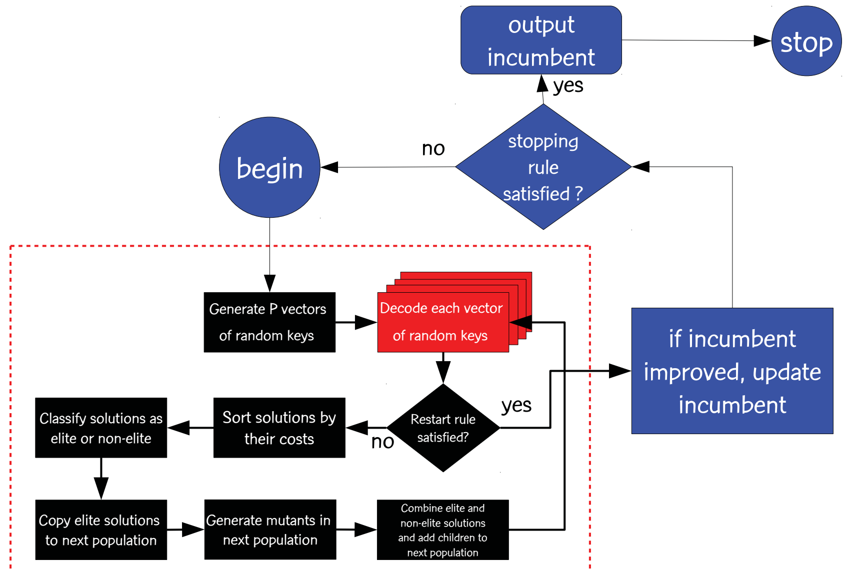
Decoding of random key vectors can be done in parallel



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

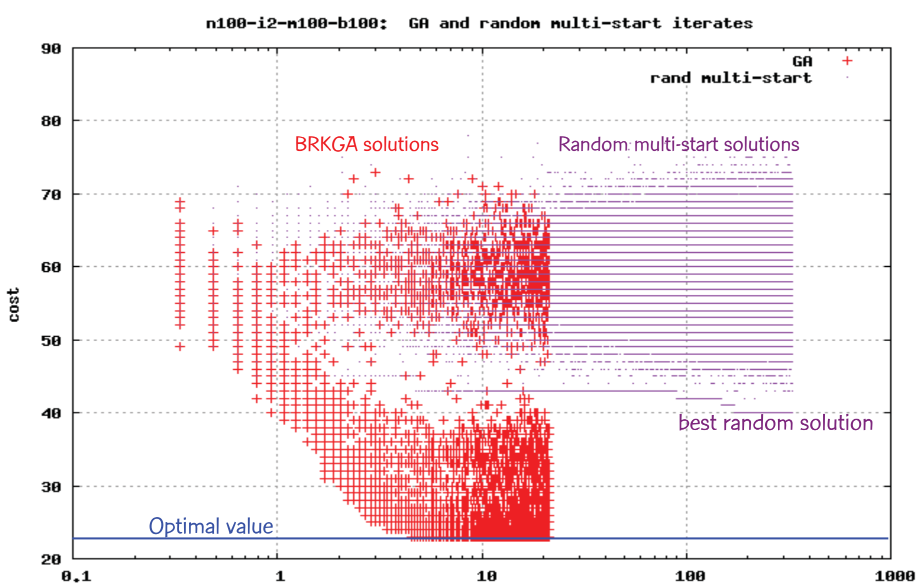
- Simulate a random multi-start decoding method with a BRKGA by setting size of elite partition to 1 and number of mutants to P–1
- Each iteration, best solution is maintained in elite set and P–1 random key vectors are generated as mutants … no mating is done since population already has P individuals

BRKGA in multi-start strategy

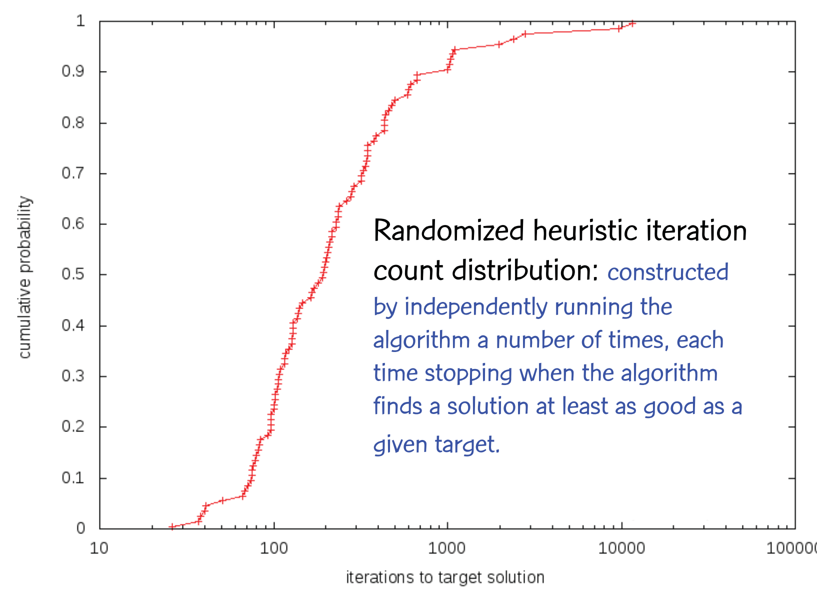


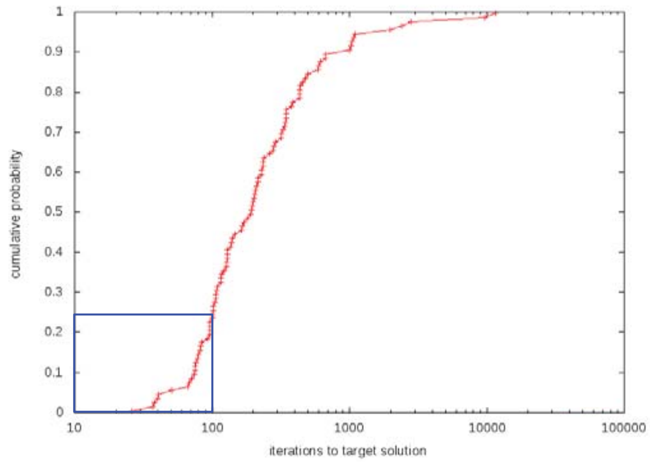
Network monitor location problem (opt = 23)

solution

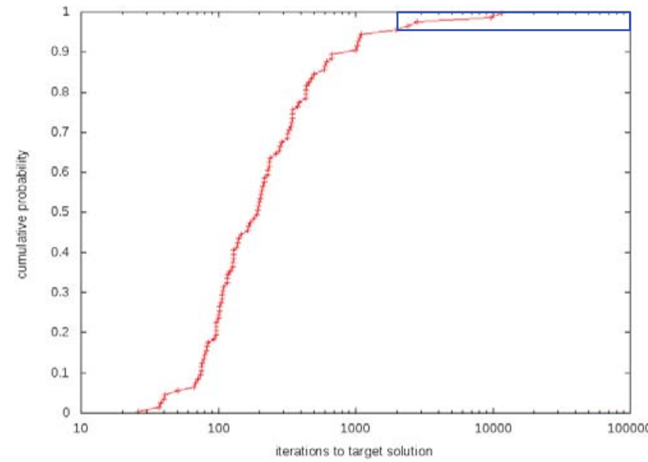


Time (ibm t41 secs)

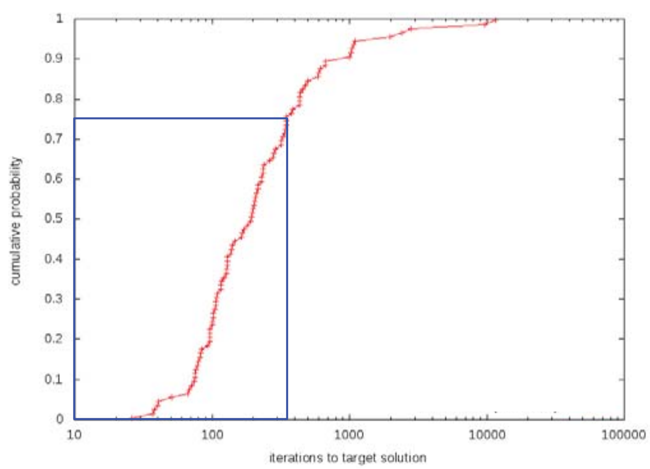




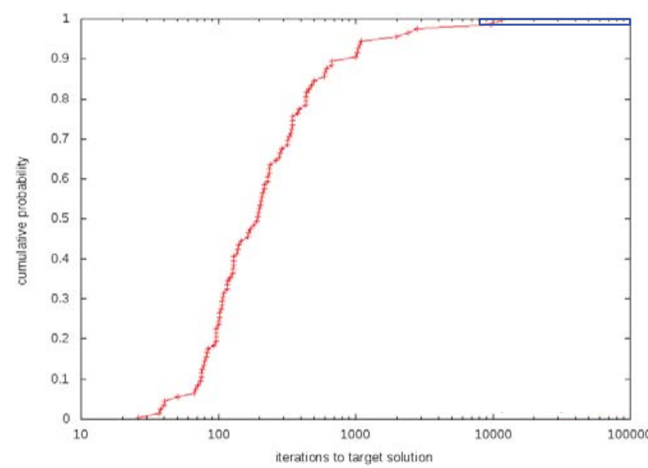
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 25% of the runs take fewer than 101 iterations



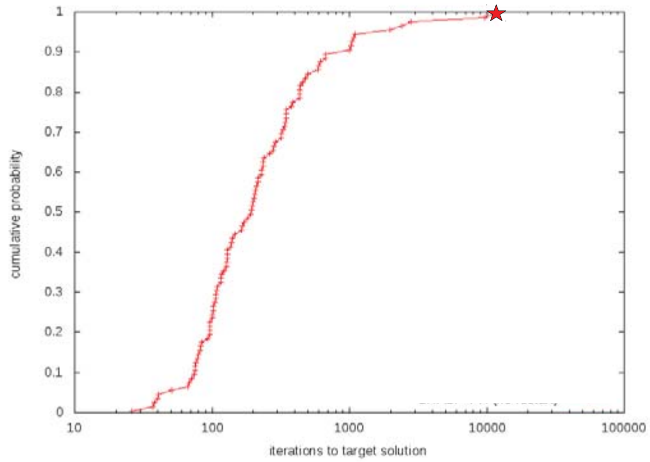
However, some runs take much longer: 5% of the runs take over 2000 iterations



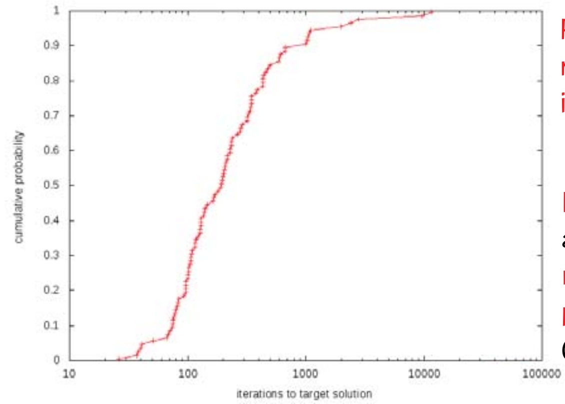
In most of the independent runs, the algorithm finds the target solution in relatively few iterations: 75% of the runs take fewer than 345 iterations



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



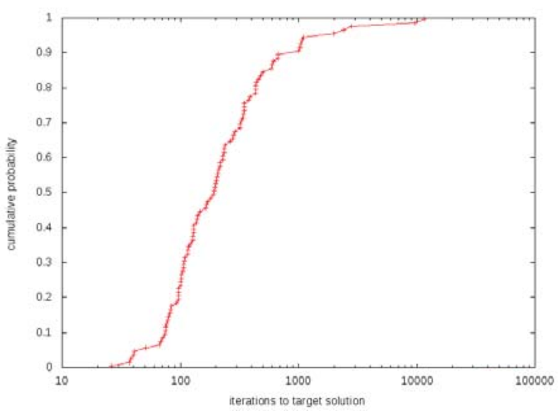
However, some runs take much longer: the longest run took 11607 iterations



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.



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 = $1/4^2$

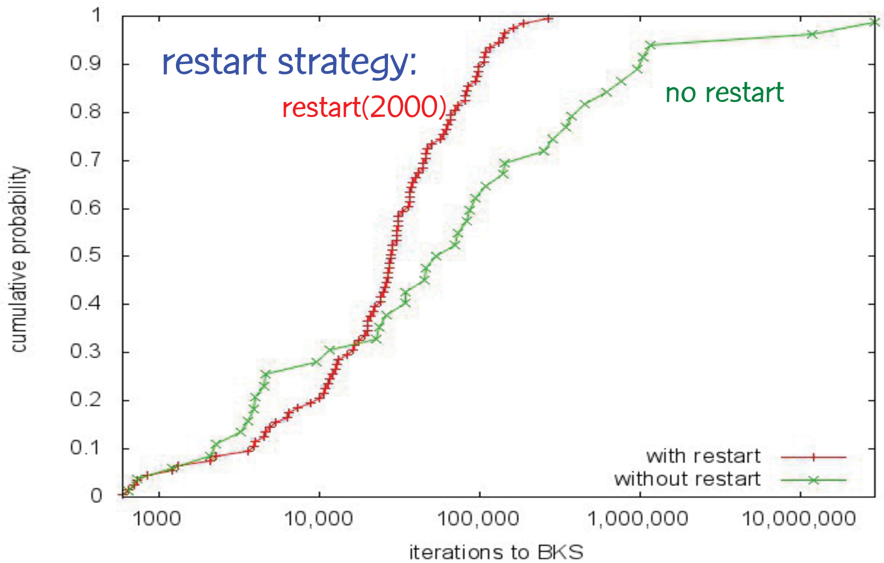
Restart strategies

- 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

Example of restart strategy for BRKGA: Telecom application



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)

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: 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

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

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.

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

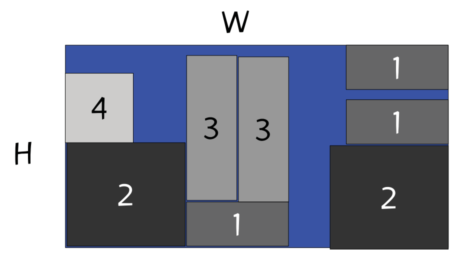
Constrained orthogonal packing

- Given a large planar stock rectangle (W, H) of width W and height H;
- 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;
- For i = 1, ..., N, we require that:

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

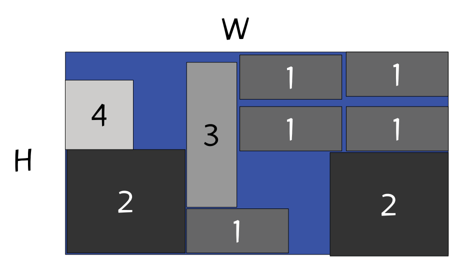


Suppose $5 \leq r[1] \leq 12$

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:

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

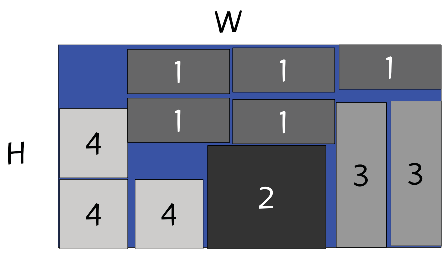


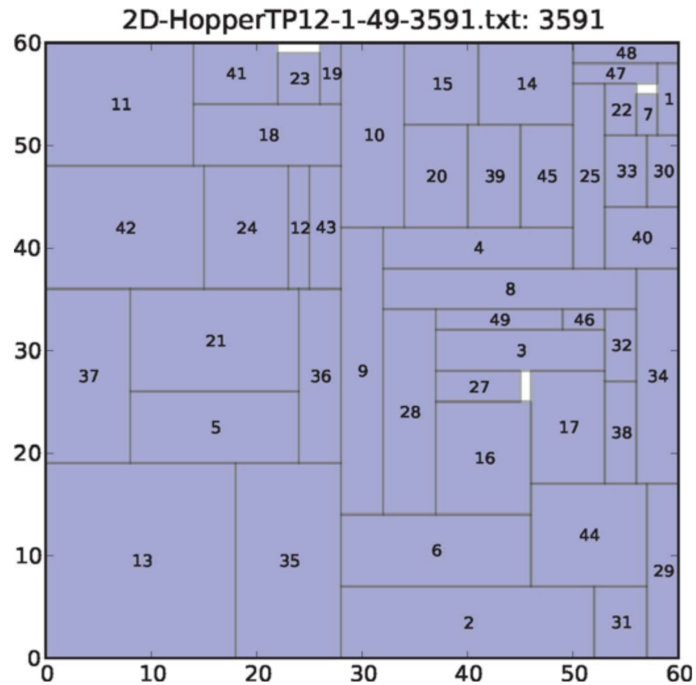
Suppose $5 \leq r[1] \leq 12$

Objective

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

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





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)

Applications

Problem arises in several production processes, e.g.

- Textile
- Glass
- Wood
- Paper

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

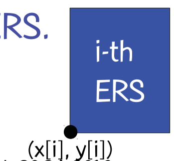
BRKGA for constrained 2-dim orthogonal packing

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

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.

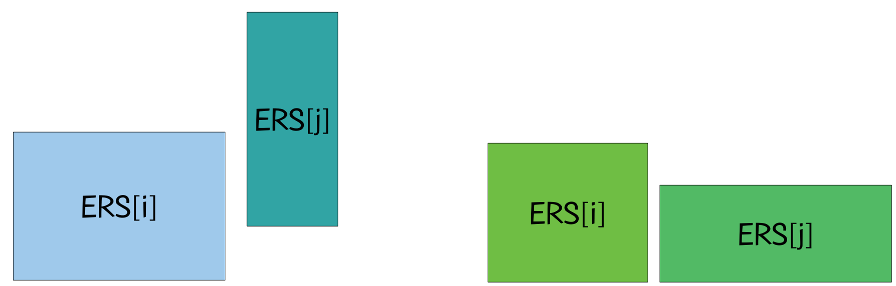


Decoding

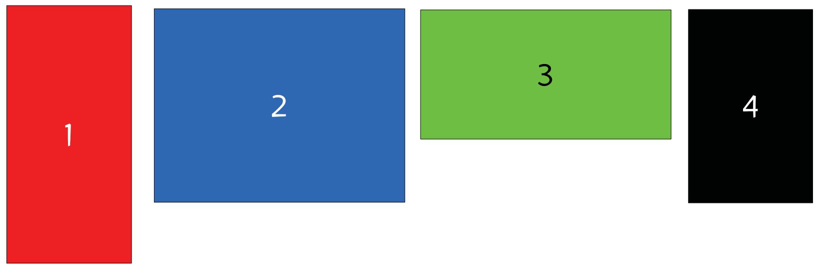
- Simple heuristic to pack rectangles:
  - Make Q[i] copies of rectangle i, for i = 1, ..., N.
  - Order the $N' = Q[1] + Q[2] + \cdots + Q[N]$ rectangles in some way. **Sort first N' keys of X to obtain order.**
  - Process the rectangles in the above order. Place the rectangle in the stock rectangle 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.**

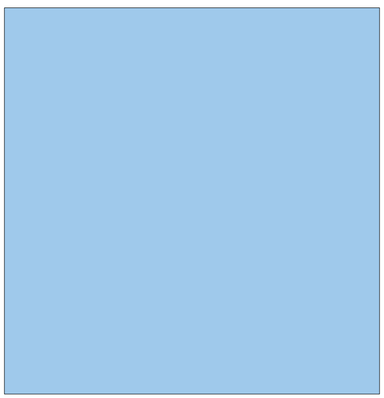
Decoding

- 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]





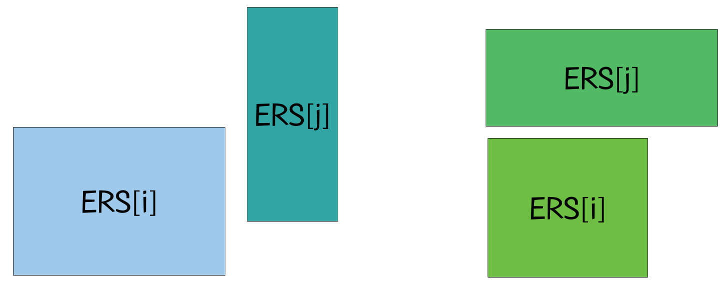
BL can run into problems even on small instances (Liu & Teng, 1999).

Consider this instance with 4 rectangles.

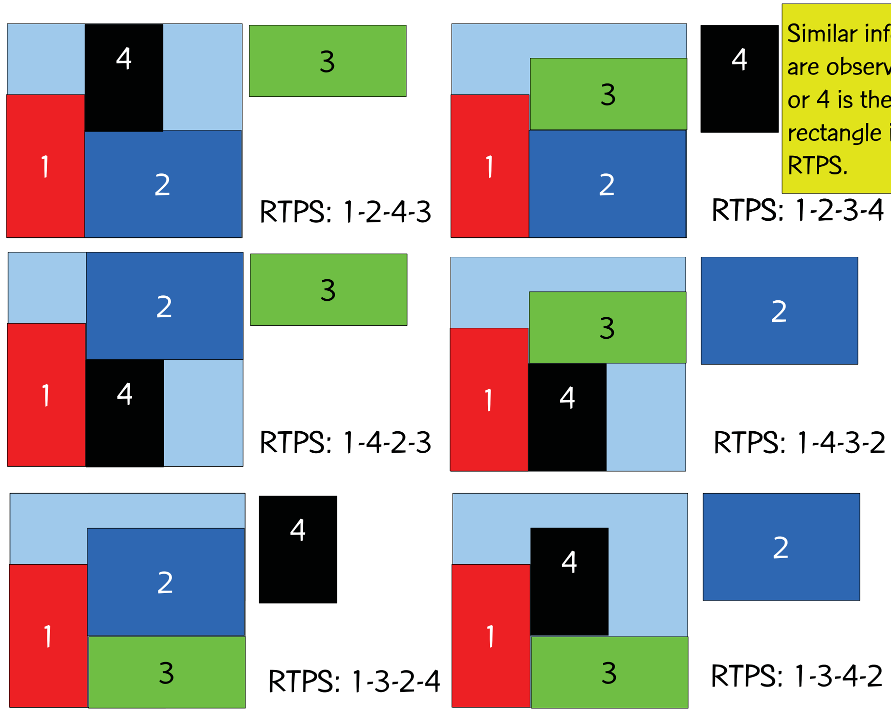
BL cannot find the optimal solution for any RTPS.

Decoding

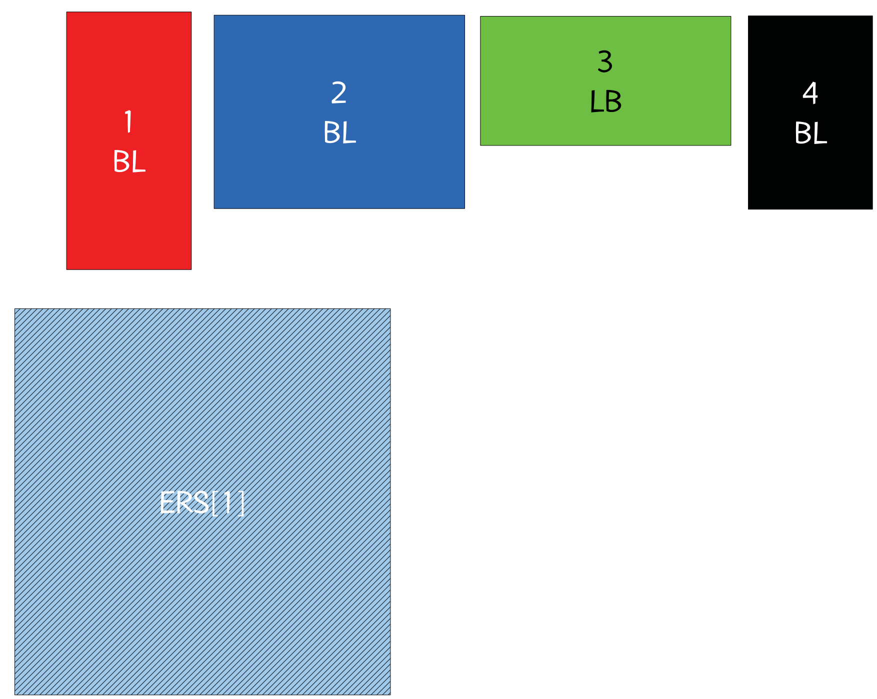
- If LB is used, ERSs are ordered such that ERS[i] < ERS[j] if x[i] < x[j] or x[i] = x[j] and y[i] < y[j].

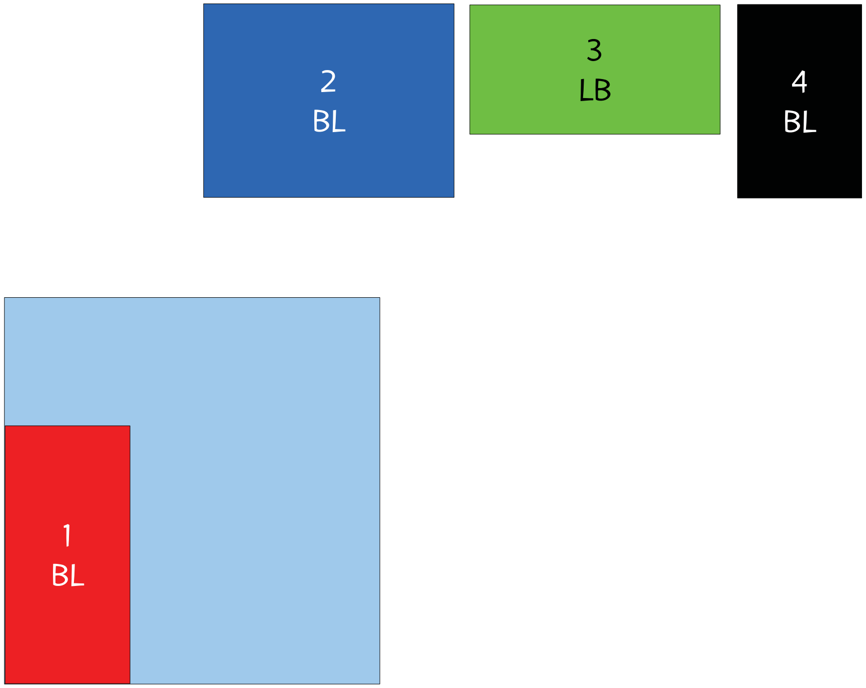


ERS[i] < ERS[j]

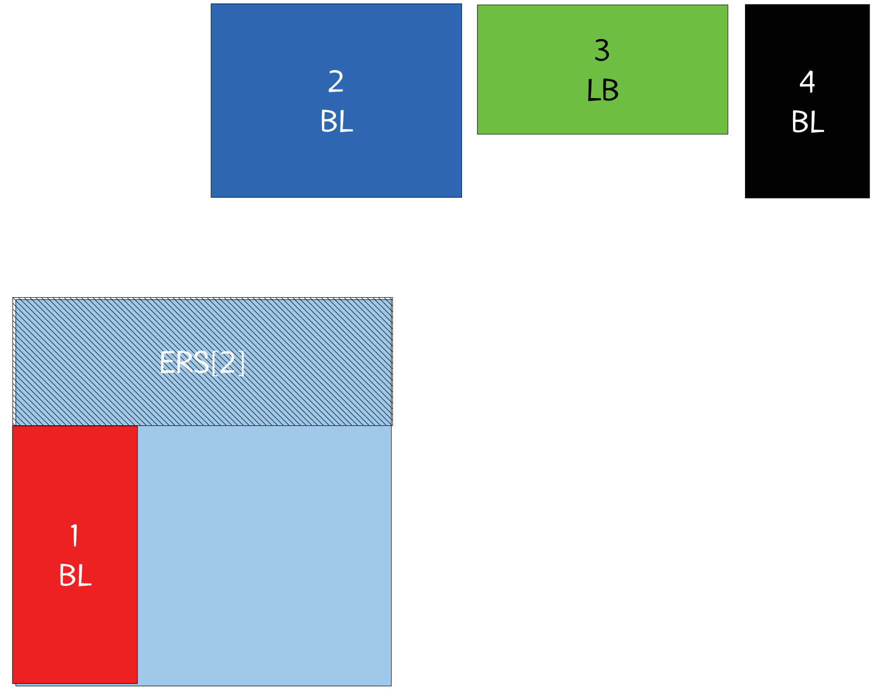


Similar infeasibilities are observed if 2, 3, or 4 is the first rectangle in the RTPS.

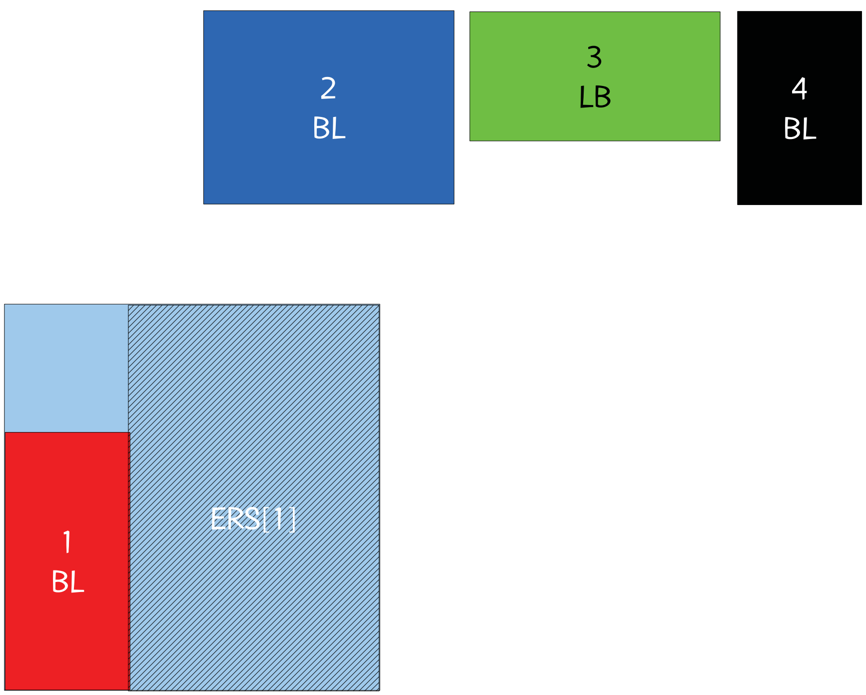




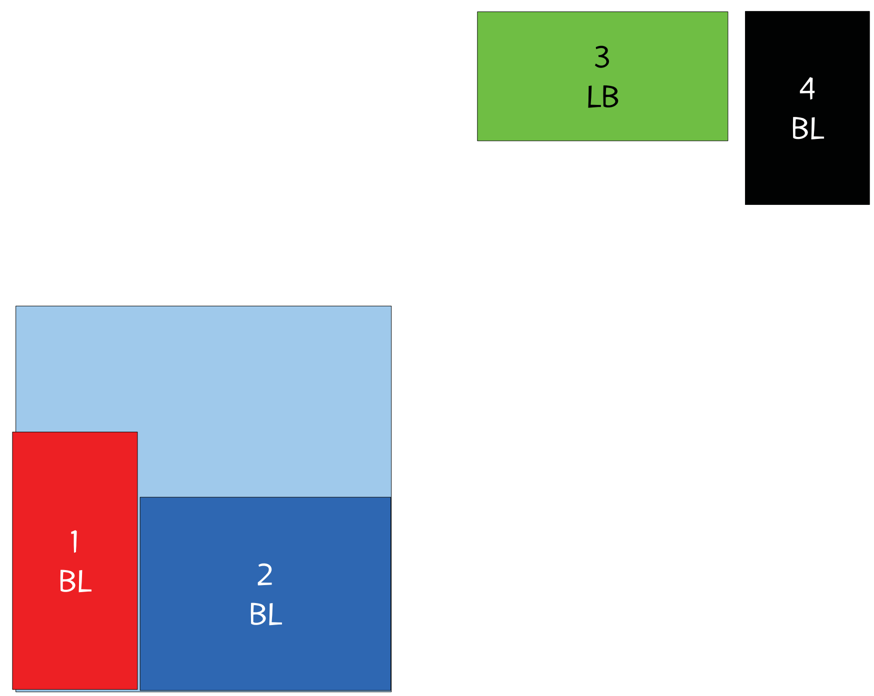
2
BL
3
LB
4
BL
1
BL



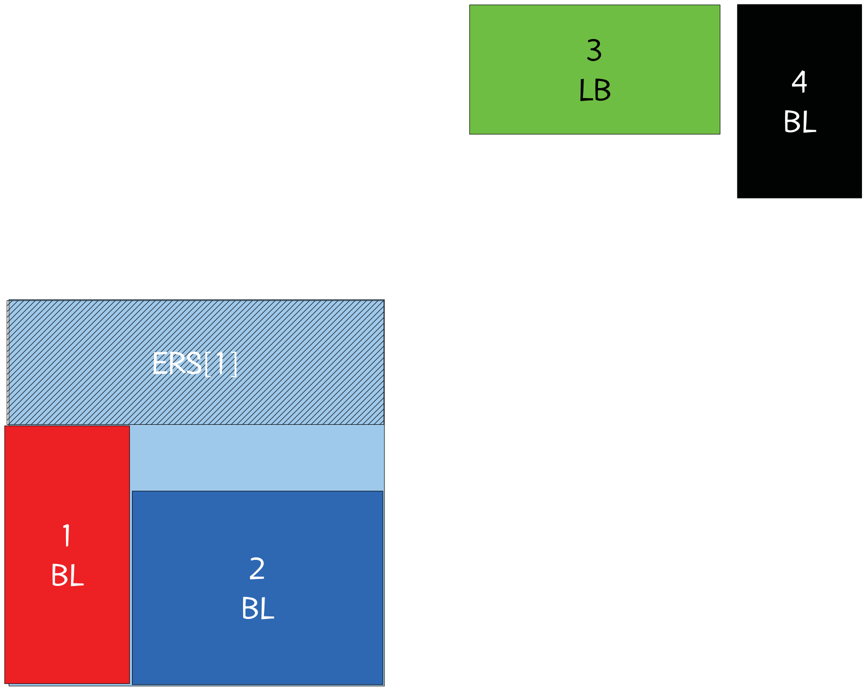
2
BL
3
LB
4
BL
ERS[2]
1
BL

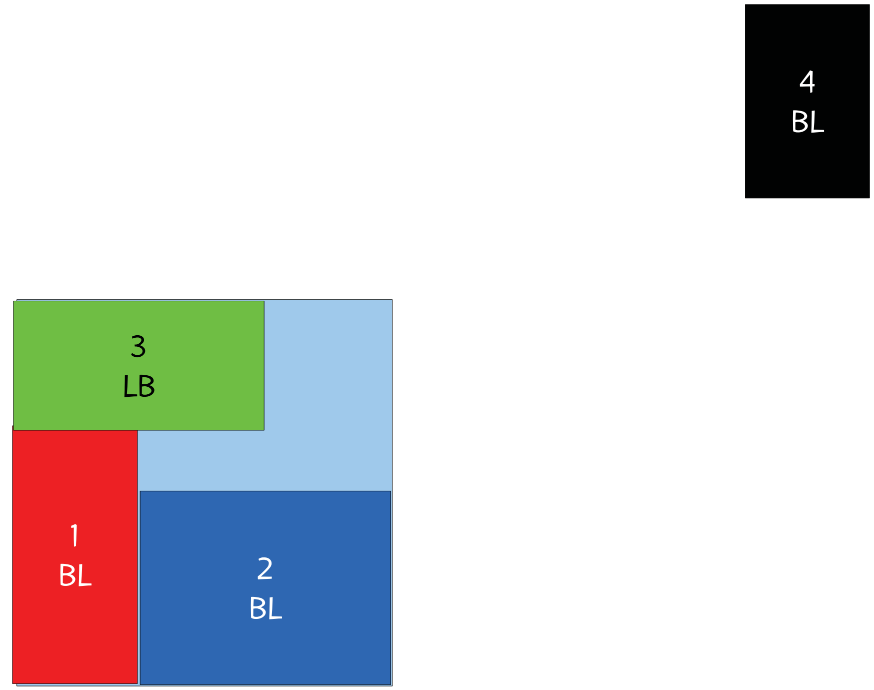


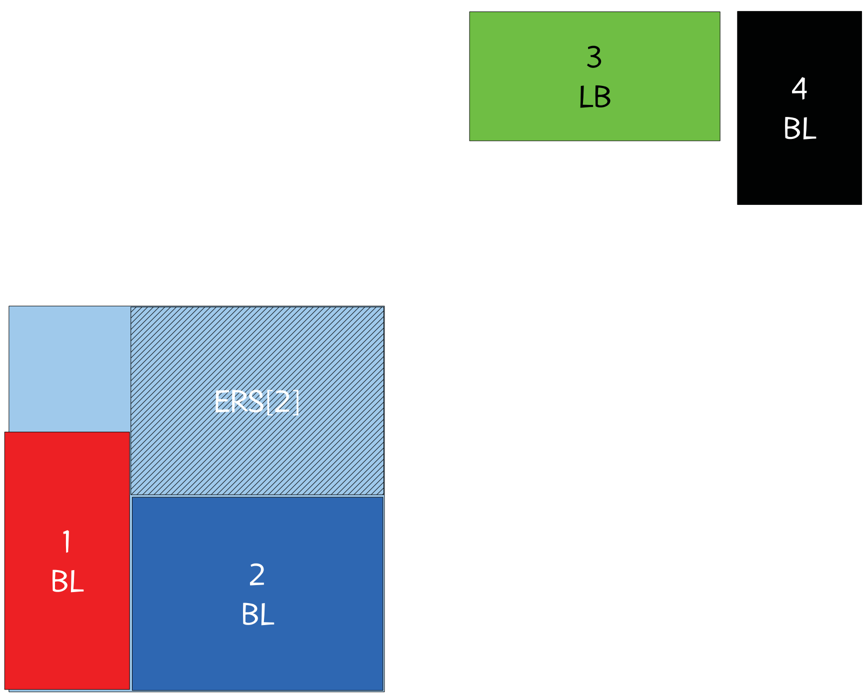
2
BL
3
LB
4
BL
ERS[1]
1
BL

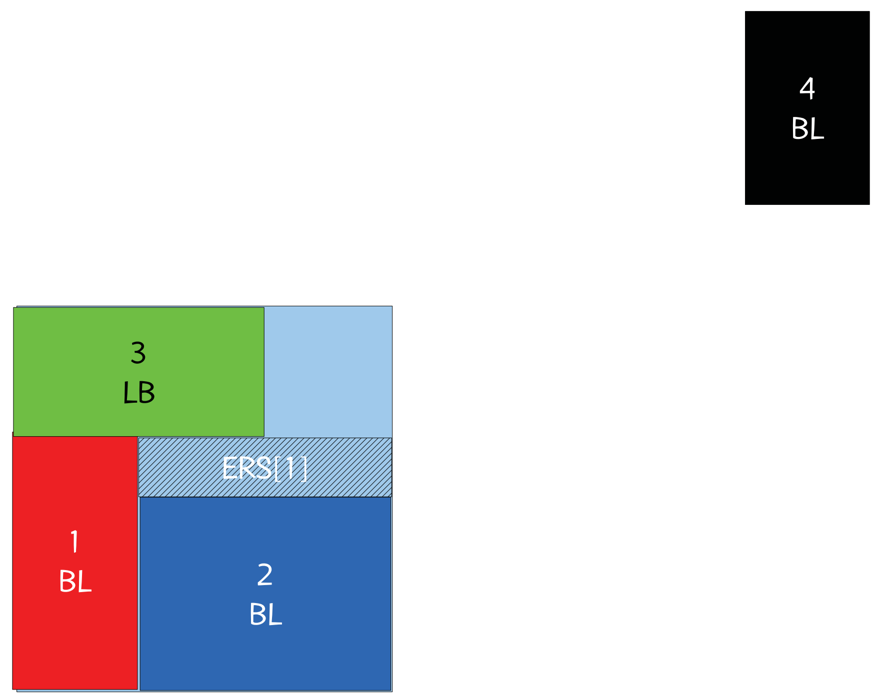


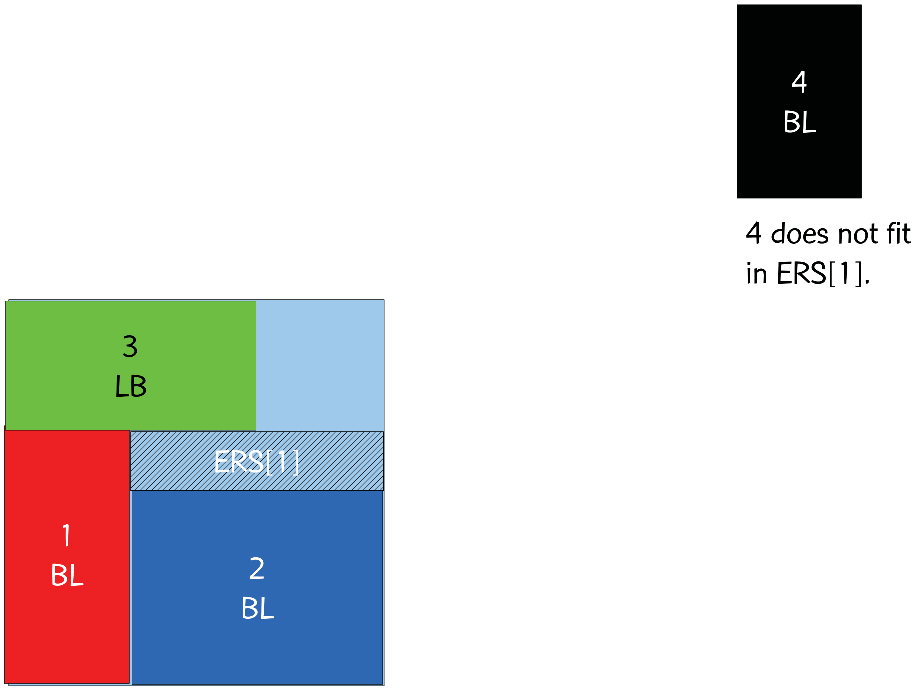
3
LB
4
BL
1
BL
2
BL

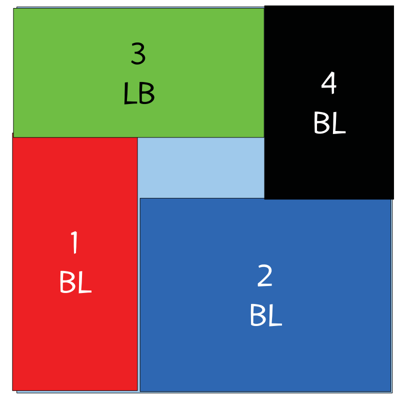

3
LB
4
BL
ERS[1]
1
BL
2
BL


4
BL
3
LB
1
BL
2
BL

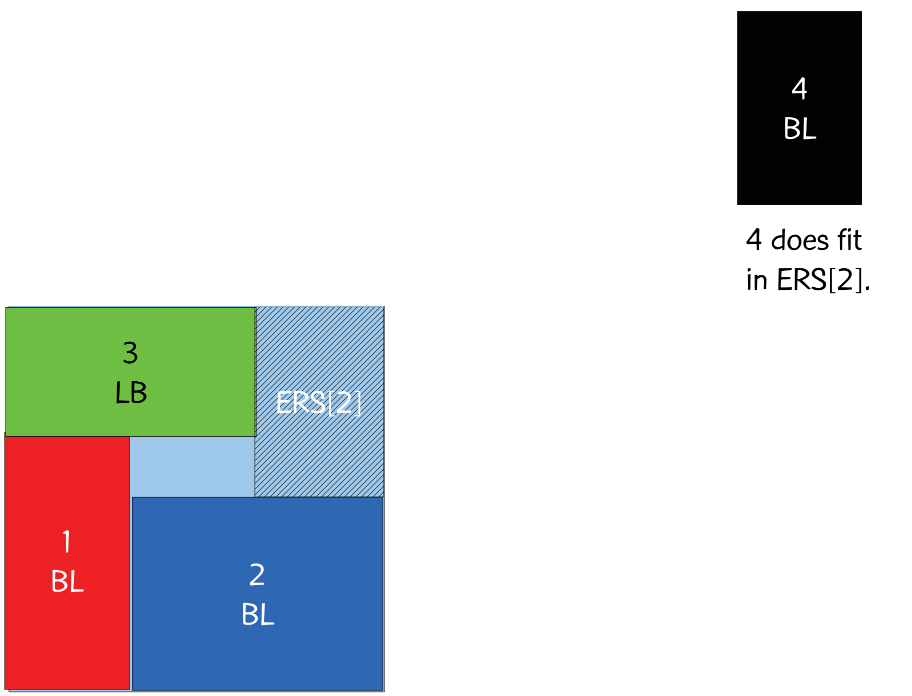

3
LB
4
BL
ERS[2]
1
BL
2
BL


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





Optimal solution!



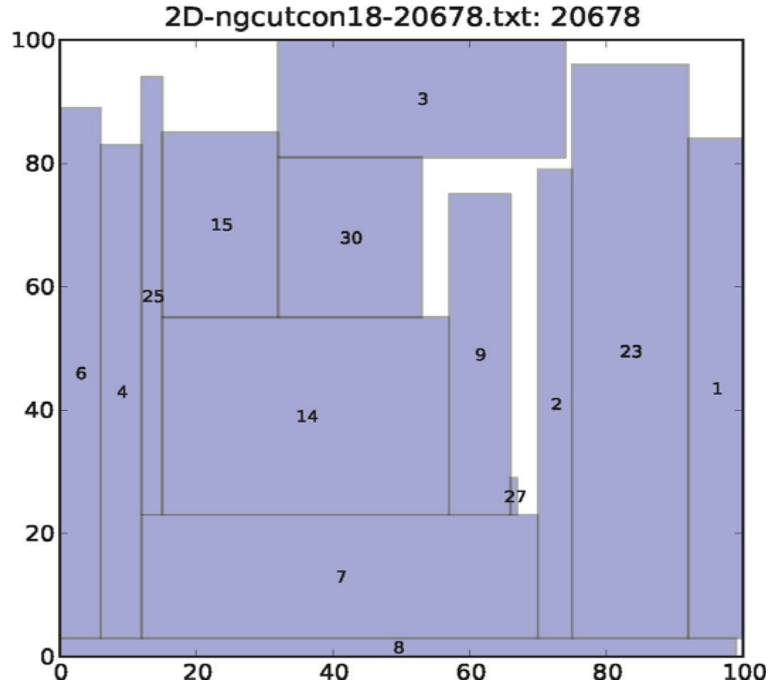
Design

- 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

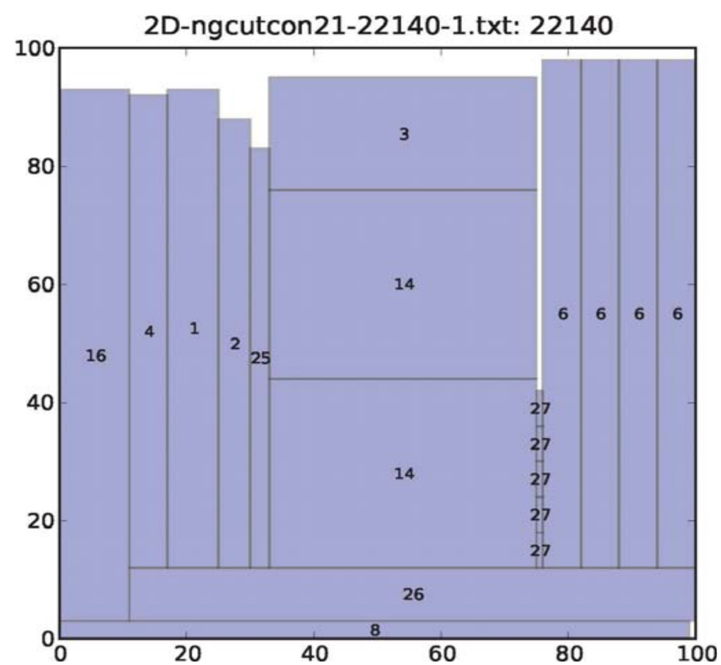


New BKS for a 100 x100 doubly constrained instance of Fekete & Schepers (1997) of value **20678.** Previous best was **19657** by tabu search of Alvarez-Valdes et al., (2007).

30 types
30 rectangles

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

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

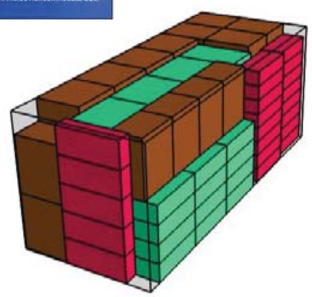
29 types
97 rectangles

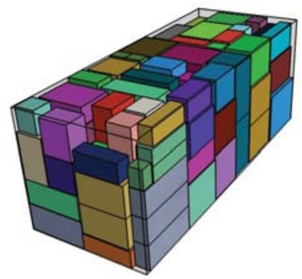
Some remarks

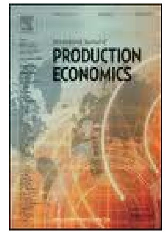
We have extended this to 3D packing:

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







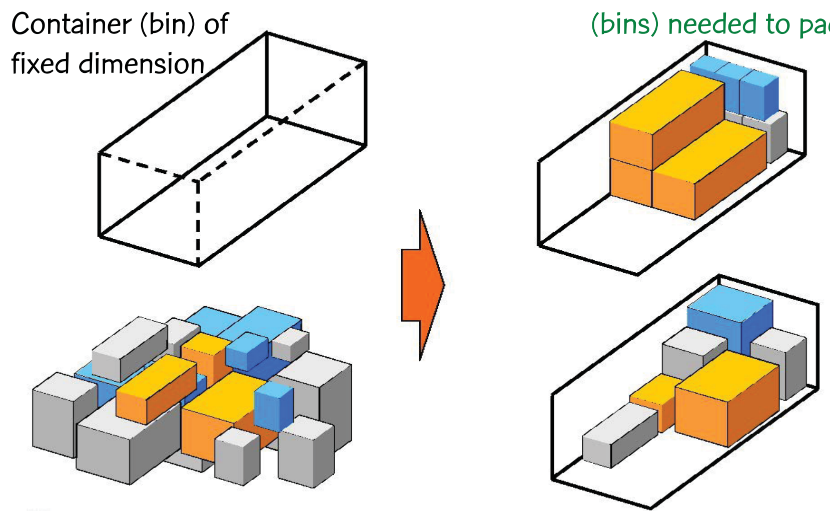
J.F. Gonçalves and R., "A biased random-key genetic algorithm for 2D and 3D bin packing problems," International J. of Production Economics, vol. 15, pp. 500–510, 2013.

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

3D bin packing

3D bin packing problem

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



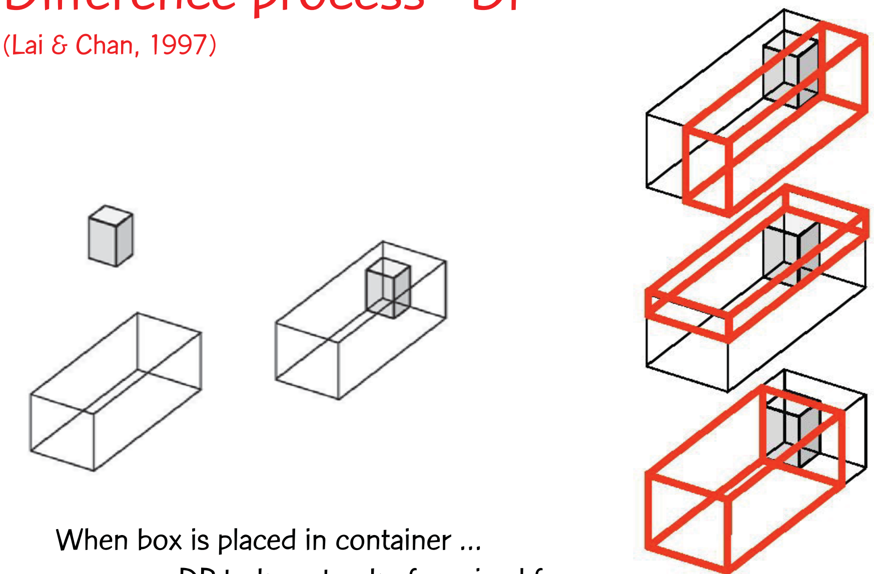
Boxes of different dimensions

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)

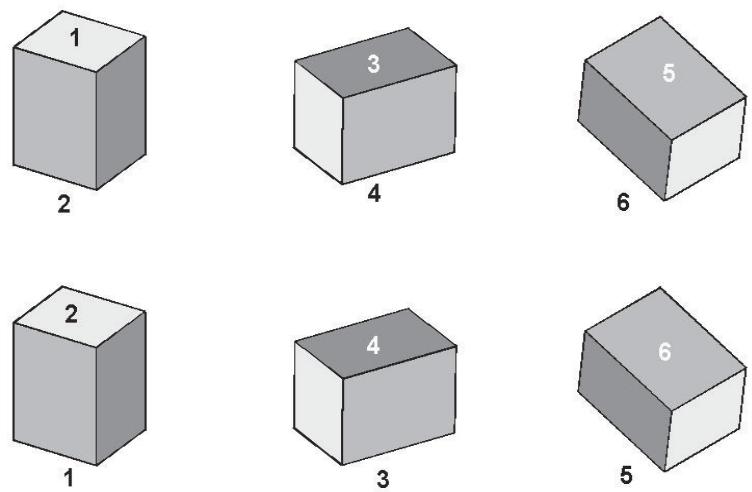
Difference process - DP

(Lai & Chan, 1997)



When box is placed in container ...
use DP to keep track of maximal free spaces

Six possible orientations for each box



Encoding

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

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

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.

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)

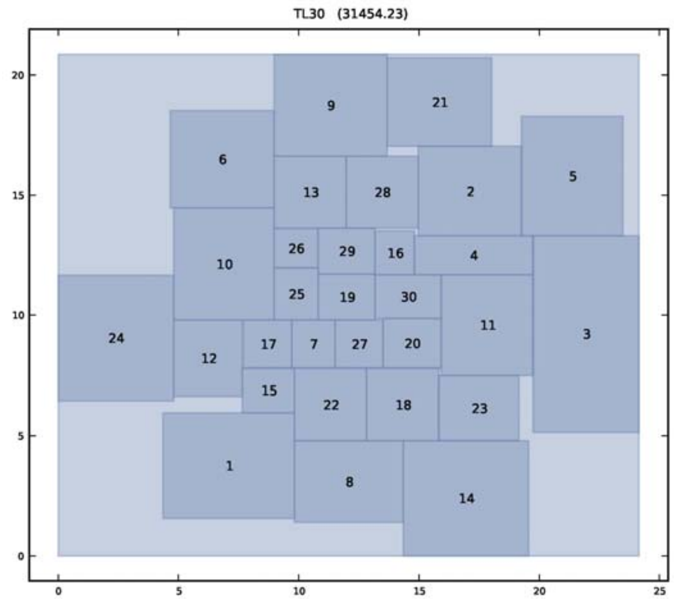
Decoding

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

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		

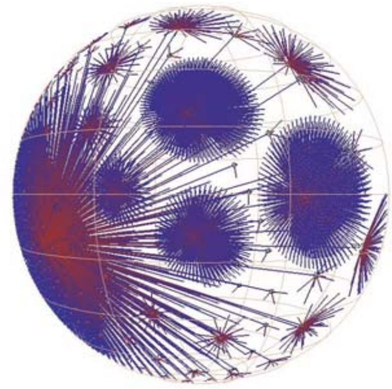


TL30

New best known
Solution: 31454.2

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

The Internet



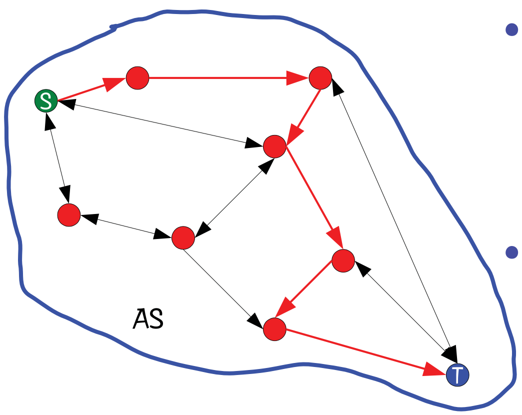
- The Internet is composed of many (inter-connected) autonomous systems (AS).
- An AS is a network controlled by a single entity, e.g. ISP, university, corporation, country, …

OSPF routing in IP networks

Routing

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

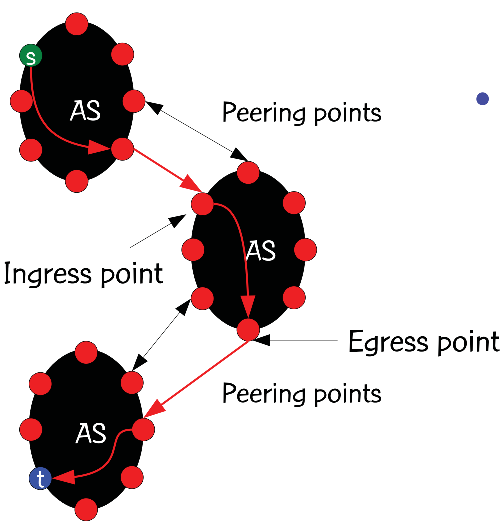
IGP Routing



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

IGP Routing

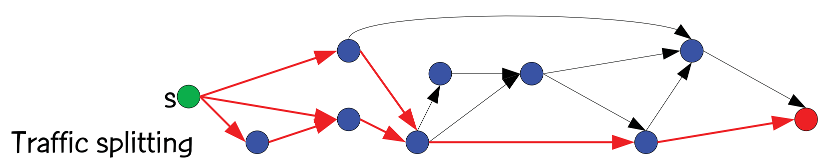
BGP Routing



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

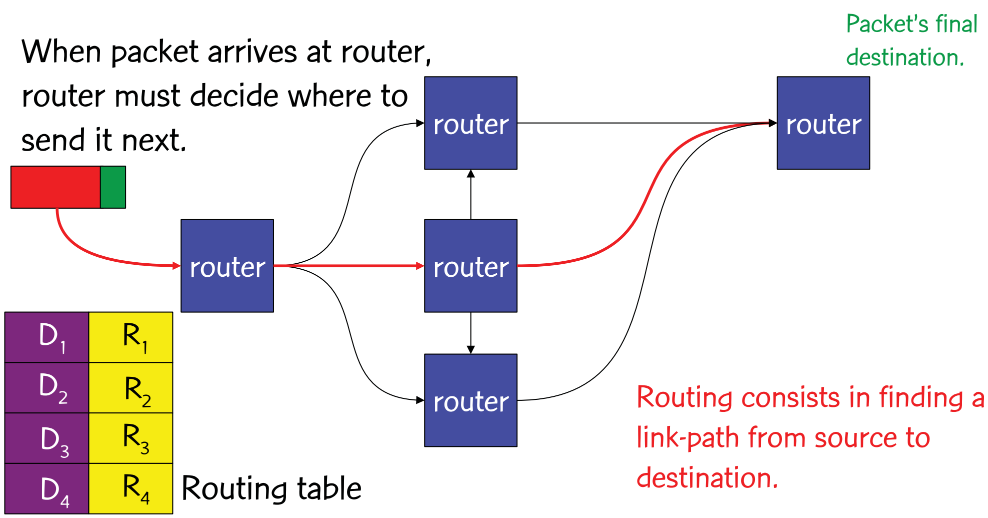
OSPF routing

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



Packet routing

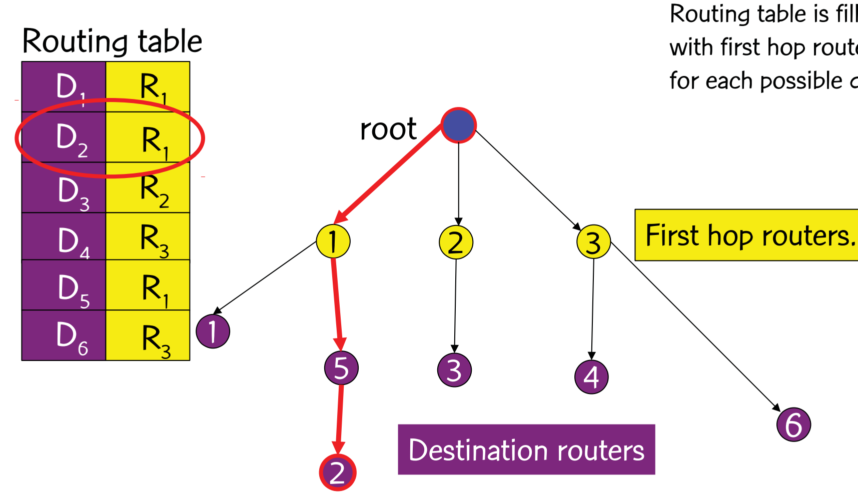
When packet arrives at router, router must decide where to send it next.



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

OSPF routing

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



OSPF routing

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

OSPF weight setting

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

Minimization of congestion

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

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

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

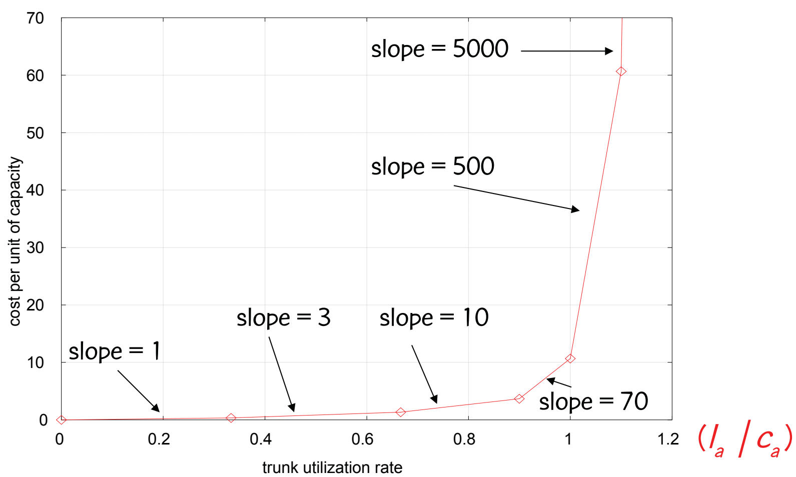
$\Phi_a(l_a)$ is piecewise linear and convex,

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

OSPF weight setting problem

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

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



BRKGA for OSPF routing in IP networks

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

Tech report version:

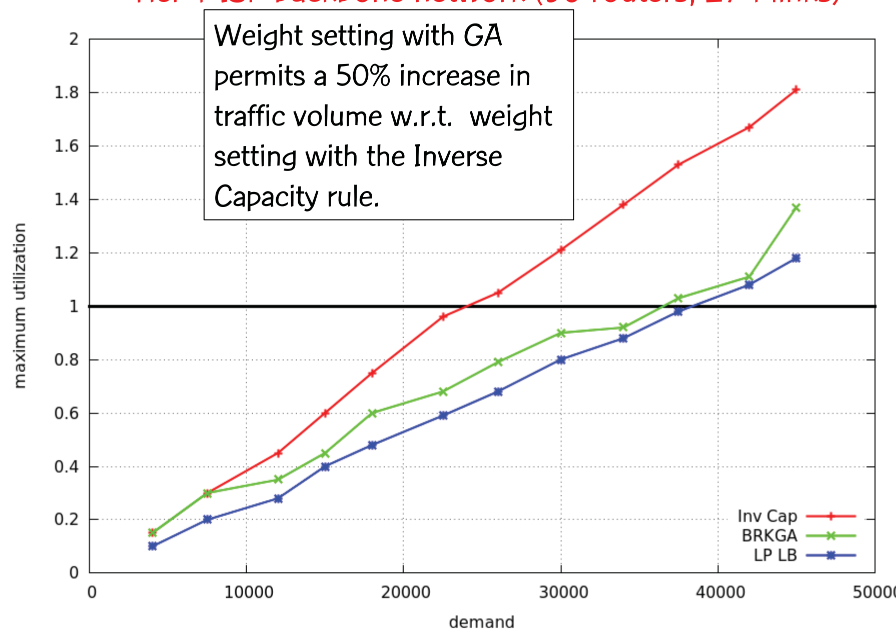
http://www2.research.att.com/~mgcr/doc/gaospf.pdf

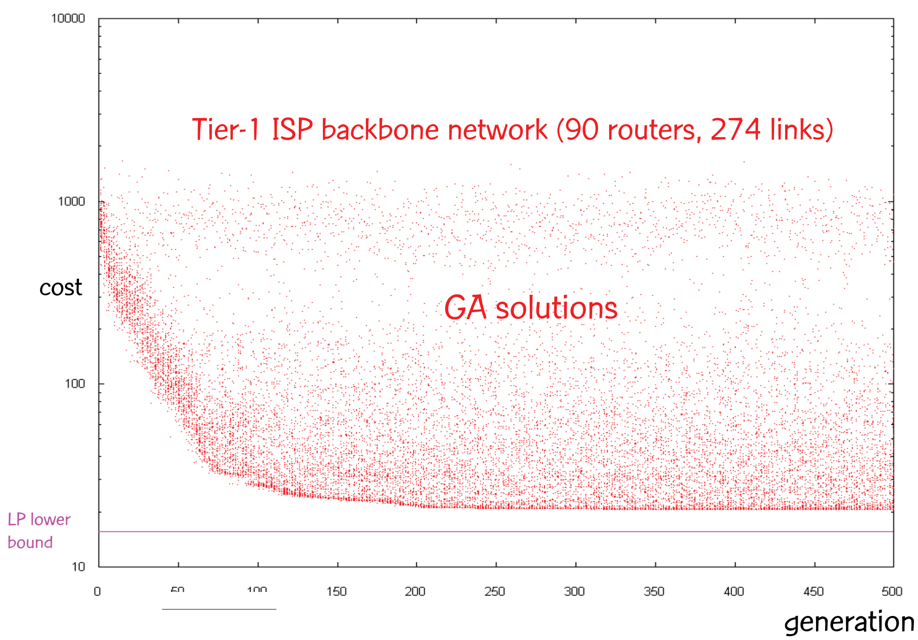
BRKGA for OSPF routing in IP networks

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

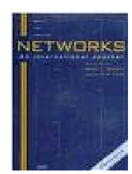
- Encoding:
  - A vector X of N random keys, where N is the number of links. The i-th random key corresponds to the i-th link weight.
- Decoding:
  - For i = 1, ..., N: set $w(i) = \text{ceil}\ (X(i) \times w_{max})$
  - Compute shortest paths and route traffic according to OSPF.
  - Compute load on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

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





Improved BRKGA for OSPF routing in IP networks



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

Tech report version:

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

Improved BRKGA for OSPF routing in IP networks

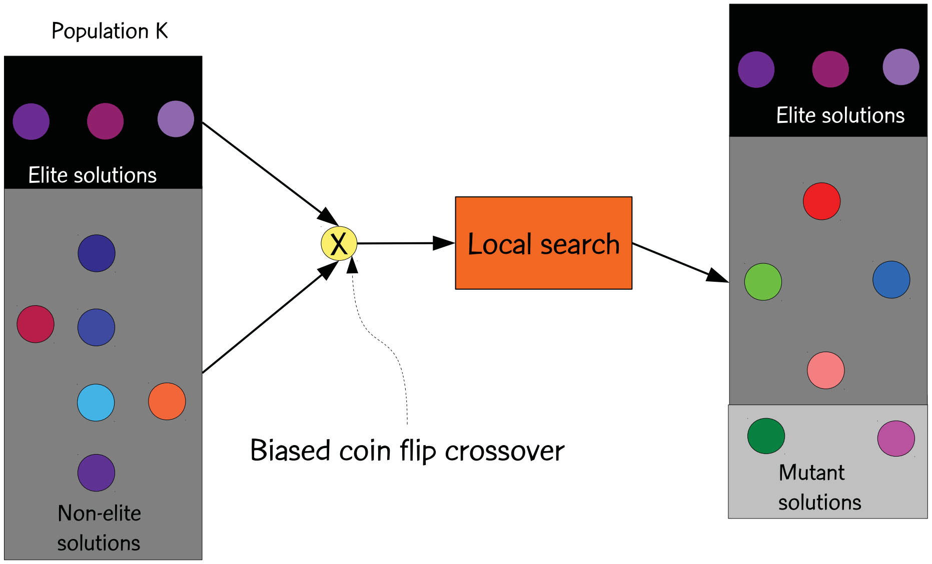
Buriol, R., Ribeiro, and Thorup (Networks, 2005)

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

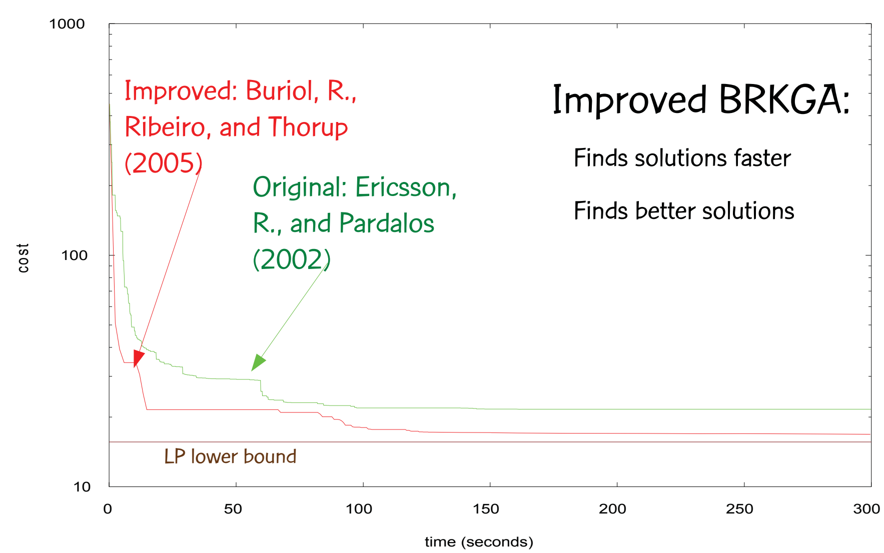
Fast local search

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

Decoder has a local search phase



Effect of decoder with fast local search

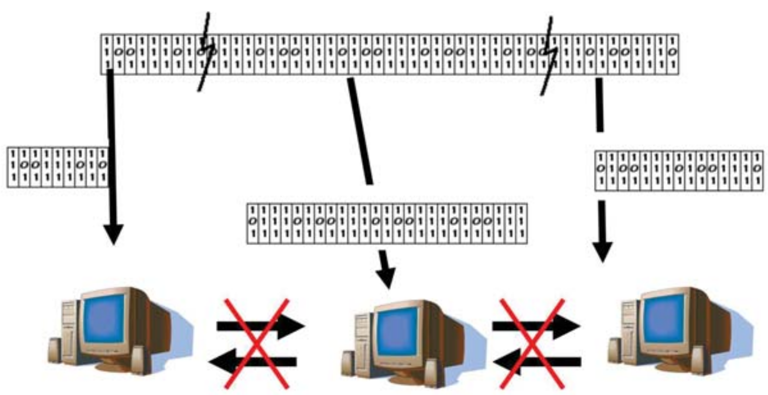


Summary

1. Divisible load scheduling
   - Divisible load model
   - System model and problem formulation
   - Related work
2. BRKGA: Biased random keys genetic algorithm
3. Computational experiments
   - Test environment
   - Instances
   - Numerical results
4. Concluding remarks and extension to multi-round scheduling
   - Concluding remarks
   - Extension to multi-round scheduling

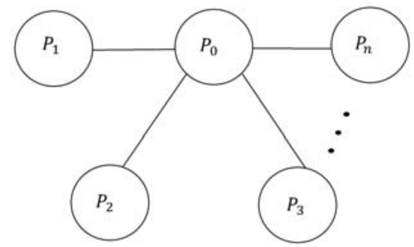
Divisible load model

- Load may be split continuously into arbitrarily many small chunks
- No precedence constraints



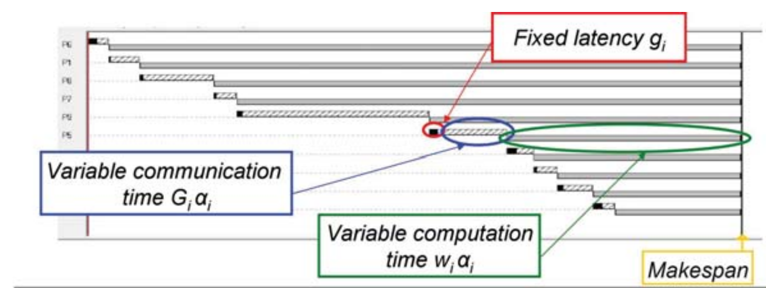
System model and problem formulation

- Interconnection topology: star network
  - ▷ Dedicated grid
- Model: one master - n workers
  - ▷ Master owns the total load W
- No communication/computation overlap in any processor
- No communication overlap through the master



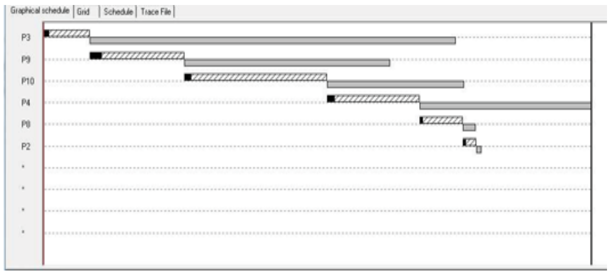
System model and problem formulation

- Single-round scheduling
  - ▷ Each processor receives portion α_i of total load
  - ▷ Master takes $g_i + G_i\alpha_i$ time units to send the data to processor P_i
  - ▷ Processor P_i takes $w_i\alpha_i$ time units to process data



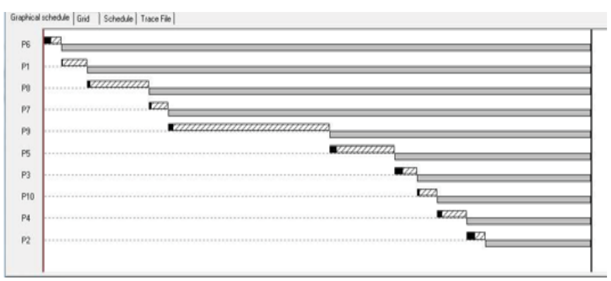
Single-round scheduling

- Non-optimal scheduling:

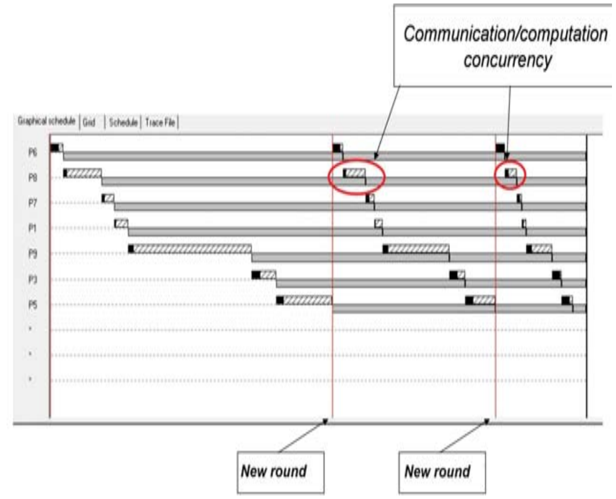


Single-round scheduling

- Optimal scheduling:



Multi-round scheduling



- This work: single-round problem

Single-round scheduling

- Problem consists of determining . . .
  - ▷ the processors to be used,
  - ▷ their activation order,
  - ▷ and their loads,
- . . . so as to minimize the makespan

BRKGA: Decoder for DLS-SR

- BRKGA for DLS-SR evolves a population of chromosomes that consists of vectors of real numbers (keys).
- Each solution represented by keys in the range [0, 1), one key for each processor.
- Each solution is decoded by a heuristic that receives the vector of keys and builds a feasible solution for DLS-SR.
- Solution quality depends on the order in which the processors are routed.
- The decoding consists of two steps: first, the processors are sorted. in a non-decreasing order of their random keys; next, the resulting order is used as the input for the decoder heuristic.

BRKGA: Decoder for DLS-SR

- Decoder: AlgRap algorithm of Abib and Ribeiro (2009).
- Given a permutation of the processors in P, the decoder computes in $O(|P|)$ time the set of active processors and the amount of load that has to be sent to each of them to minimize the makespan.
- In addition to the number of processors and all their data, this algorithm takes as input a vector π describing the activation order, such that $\pi(i) = j$ indicates that processor j is the i-th to be activated, for $i, j = 1, \ldots, n$.
- For instance, if $n = 3$ and $\pi =< 2, 3, 1 >$, then processor 2 is the first to be activated, processor 3 is the second, and processor 1 is the third.

BRKGA: Decoder for DLS-SR

- Given some activation order, the algorithm starts by sending all the load exclusively to the first processor.
- Number ℓ of processors is iteratively increased from 1 to n, until the makespan deteriorates (lines 10–12).
- Optimal number of processors is set as $\ell^* = \ell - 1$ (lines 18–23).
- Compute the load α_{ℓ^*} sent to the last processor (line 24).
- Loads α_i, for $i = 1, \ldots, \ell^* - 1$, are recursively computed from ℓ^* (lines 25–27).
- Decoder implements these computations in time $O(n)$.

Test environment

- BRKGA-DLS implemented in C++ and compiled with GNU C++ version 4.6.3.
- Experiments performed on a Quad-Core AMD Opteron(tm) Processor 2350, with 16 GB of RAM memory.
- Comparisons with CPLEX, HeuRet, and multistart procedure MS-DLS.
- Version 12.6 of CPLEX was used and the maximum CPU time was set to 24 hours.
- Ten runs of each heuristic for each instance, with different seeds for the random number generator.

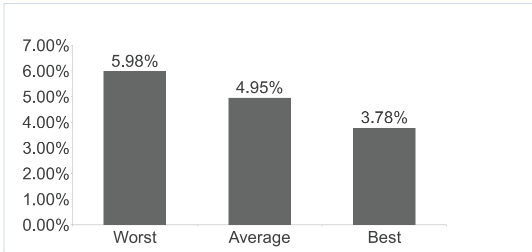
Instances

- Instances used in the three first experiments: same proposed in Abib and Ribeiro (2009).
- 120 grid configurations with $n = 10, 20, 40, 80, 160$ worker processors and eight combinations of the parameters g_i, G_i and w_i, $i = 1, \ldots, n$, each of them ranging either in the interval [1, 100] (*low*) or in the interval [1000, 100000] (*high*).

Numerical results

- The first experiment: evaluates if BRKGA-DLS efficiently identifies the relationships between keys and good solutions and converges faster to near-optimal solutions.
- We compare its performance with that of the multi-start procedure.
- Each iteration of MS-DLS procedure applies the same decoding heuristic of BRKGA-DLS, but using randomly generated values for the keys.
- BRKGA-DLS was run for 1000 generations and MS-DLS for $1000 \times |V|$ iterations, where $|V| = 5 \times |P|$ is the population size of BRKGA-DLS (same number of solutions are evaluated and compared).

Average percent relative reduction over the 720 instances of the best, average and worse solution values found by BRKGA-DLS with respect to those obtained by MS-DLS



- Average solution values found by BRKGA-DLS were 4.95% better than those provided by MS-DLS.
- BRKGA-DLS identifies the relationships between keys and good solutions, making the evolutionary process converge to better solutions faster than MS-DLS.

Summary of the numerical results obtained with BRKGA-DLS, HeuRet, and MS-DLS for 720 test instances

- In the second experiment, we compare BRKGA-DLS with HeuRet, and MS-DLS. HeuRet is a deterministic algorithm, while the others are randomized.

	MS-DLS	HeuRet	BRKGA-DLS
Optimal values (over 497 instances)	177	320	413
Best values (over 720 instances)	189	313	645
Best values (over 7200 runs)	2166	-	6191
Score value	803	112	1

- “score” represents the sum over all instances of the number of methods that found strictly better solutions than the specific heuristic being considered: best heuristics have lower score values.
- BRKGA-DLS outperformed the previously existing HeuRet heuristic and MS-DLS with respect to all measures considered.

New instances

- 20 new, larger, and more realistic instances with $|P| = 320$ and $W = 10,000$.
- The values of G_i and g_i have been randomly generated in the ranges $[1, 100]$ and $[100, 100.000]$, respectively.
- Differently from Abib and Ribeiro (2009), the values of w_i have been randomly generated in the interval $[200, 500]$.
- These values are more realistic, since the processing rate of a real computer is always larger than its communication rate.
- BRKGA-DLS stops after $|P|$ generations without improvement in the best solution found.

Extension: Multi-round scheduling

- Extension of this approach to the harder case of multi-round (or multi-installment) scheduling.
- Load is distributed to the active processors in several consecutive bursts, reducing the waste in each processor and making better use of the resources to reduce the overall makespan.
- Concurrency between communication in burst $k + 1$ and computation in burst k.
- Multi-round scheduling consists of determining ...
  - not only the processors to be used, their activation order, and their loads,
  - but also the number of rounds...
- ...so as to minimize the makespan.
- On average, BRKGA improved the makespan obtained by closed forms of Shokripour et al. (2012) by 12%.

BRKGA vs. HeuRet on 320-processor instances

- Makespan obtained by BRKGA-DLS is always smaller than that given by HeuRet.
- Coefficient of variation of BRKGA-DLS is very small, indicating its robustness.
- Percent relative reduction of BRKGA-DLS with respect to HeuRet amounted to 3.19% for instance dls.320.10 and to 2.38% on average.
- Although the running times of BRKGA-DLS are larger than those of HeuRet, their average values never exceeded the time taken by HeuRet by more than 30 seconds.
- Larger running times are not a major issue in practice (parallel processing).

Thanks!

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

http://mauricio.resende.info