Packing with biased random-key genetic algorithms

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

resendem AT amazon DOT com

Lecture given at IC - UNICAMP
Campinas (SP), Brazil  ✦  March 3, 2015

Work done when speaker was employed at
AT&T Labs Research.
Joint work with José F. Gonçalves
U. do Porto
Portugal
Summary

• Metaheuristics and basic concepts of genetic algorithms
• Random-key genetic algorithm of Bean (1994)
• Biased random-key genetic algorithms (BRKGA)
  – Encoding / Decoding
  – Initial population
  – Evolutionary mechanisms
  – Problem independent / problem dependent components
  – Multi-start strategy
  – Specifying a BRKGA
  – Application programming interface (API) for BRKGA
• BRKGA for 2-dim and 3-dim packing
• BRKGA for 3-dim bin packing
• Concluding remarks
Metaheuristics are heuristics to devise heuristics.
Metaheuristics

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

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

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

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

Holland (1975)
Genetic algorithms

Individual: solution (chromosome = string of genes)
Population: set of fixed number of individuals
Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.
Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

A series of generations are produced by the algorithm. The most fit individual of the last generation is the solution.
Genetic algorithms evolve population applying Darwin's principle of survival of the fittest.

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

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

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

Parents drawn from generation K

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

Combine parents

Mutation

Child in generation \( K+1 \)

Parents drawn from generation K
Crossover and mutation

Combining parents

Mutation

Elements: a, b, c
Crossover and mutation

Crossover: Combines parents ... passing along to offspring characteristics of each parent ... Intensification of search
Crossover and mutation

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


Tech report version:

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

\[
\begin{bmatrix}
1, & 2, & 3, & 4, & 5 \\
\end{bmatrix}
\]

\[
X = \begin{bmatrix}
0.099, & 0.216, & 0.802, & 0.368, & 0.658
\end{bmatrix}
\]
Encoding with random keys: Sequencing

Encoding

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

Decode by sorting vector of random keys

\[
\begin{bmatrix}
1, & 2, & 4, & 5, & 3
\end{bmatrix}
\]
\[
X = \begin{bmatrix}
0.099, & 0.216, & 0.368, & 0.658, & 0.802
\end{bmatrix}
\]
Encoding with random keys: Sequencing

Therefore, the vector of random keys:

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

encodes the sequence: 1 – 2 – 4 – 5 – 3
Encoding with random keys: Subset selection (select 3 of 5 elements)

Encoding

\[
\begin{bmatrix}
1, & 2, & 3, & 4, & 5
\end{bmatrix}
\]

\[
X = [0.099, 0.216, 0.802, 0.368, 0.658]
\]
Encoding with random keys: Subset selection (select 3 of 5 elements)

Encoding

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

Decode by sorting vector of random keys

\[
\begin{bmatrix}
1, & 2, & 4, & 5, & 3
\end{bmatrix}
\]
\[X = [0.099, 0.216, 0.368, 0.658, 0.802]\]
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 $\in [0,10]$ to a subset of 3 of 5 elements

**Encoding**

$$
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5
\end{array}
$$

$$
X = \begin{bmatrix}
0.099 & 0.216 & 0.802 & 0.368 & 0.658 \\
0.4634 & 0.5611 & 0.2752 & 0.4874 & 0.0348
\end{bmatrix}
$$
Encoding with random keys: Assigning integer weights $\in [0, 10]$ to a subset of 3 of 5 elements

**Encoding**

\[
\begin{array}{c|cccc}
1 & 2 & 3 & 4 & 5 \\
\hline
1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[
X = \begin{bmatrix}
0.099 & 0.216 & 0.802 & 0.368 & 0.658 \\
0.4634 & 0.5611 & 0.2752 & 0.4874 & 0.0348
\end{bmatrix}
\]

Decode by sorting the first 5 keys and assign as the weight the value $W_i = \text{floor} \left[ 10 X_{5+i} \right] + 1$ to the 3 elements with smallest keys $X_i$, for $i = 1, ..., 5$. 
Encoding with random keys: Assigning integer weights $\in [0,10]$ to a subset of 3 of 5 elements

Therefore, the vector of random keys:

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

encodes the weight vector $W = (5, 6, -, 5, -)$
Genetic algorithms and random keys
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.
GAs and random keys

• Introduced by Bean (1994) for sequencing problems.

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

\[
S = (0.25, 0.19, 0.67, 0.05, 0.89)
\]

\[
s(1) \ s(2) \ s(3) \ s(4) \ s(5)
\]
GAs and random keys

- Introduced by Bean (1994) for sequencing problems.

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

- Sorting random keys results in a sequencing order.

\[
S = (0.25, 0.19, 0.67, 0.05, 0.89) \\
\text{s(1) s(2) s(3) s(4) s(5)}
\]

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

Sequence: 4 – 2 – 1 – 3 – 5
GAs and random keys

- Mating is done using parametrized uniform crossover \((\text{Spears \& DeJong, 1990})\)

\[
a = (0.25, 0.19, 0.67, 0.05, 0.89) \\
b = (0.63, 0.90, 0.76, 0.93, 0.08)
\]
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 \((\text{Spears} \ & \text{DeJong}, \ 1990)\)

• For each gene, flip a biased coin to choose which parent passes the allele (key, or value of gene) to the child.

\[
\begin{align*}
\text{a} &= (0.25, 0.19, 0.67, 0.05, 0.89) \\
\text{b} &= (0.63, 0.90, 0.76, 0.93, 0.08) \\
\text{c} &= (0.25, 0.90)
\end{align*}
\]
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.

\[
\begin{align*}
a &= (0.25, 0.19, 0.67, 0.05, 0.89) \\
b &= (0.63, 0.90, 0.76, 0.93, 0.08) \\
c &= (0.25, 0.90, 0.76, 0.05)
\end{align*}
\]
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

Population K

Elite solutions

Non-elite solutions

Population K+1
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

Packing with a BRKGA
### How RKGA & BRKGA differ

<table>
<thead>
<tr>
<th>RKGA</th>
<th>BRKGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>both parents chosen at random from entire population</td>
<td>both parents chosen at random but one parent chosen from population of elite solutions</td>
</tr>
<tr>
<td>either parent can be parent A in parametrized uniform crossover</td>
<td></td>
</tr>
</tbody>
</table>
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

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.
set covering problem: scp41
Pr(\(t_{\text{BRKGA}} \leq t_{\text{RKGA}}\)) = 0.740

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

set covering problem: scp41
set covering problem: scp51
\[ \Pr(t_{\text{BRKGA}} \leq t_{\text{RKGA}}) = 0.999 \]

set covering problem: scp51
set covering problem: scp1

iterations to target solution

cumulative probability

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000

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

set covering problem: scp1
set \( k \)-covering problem: scp41-2
set $k$-covering problem: scp41-2

$\Pr(t_{\text{BRKGA}} \leq t_{\text{RKGA}}) = 0.999$
set $k$-covering problem: scp45-11
Packing with a BRKGA

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

set k-covering problem: scp45-11
set \( k \)-covering problem: scp48-7
Pr(t_{BRKGA} \leq t_{RKGA}) = 0.847

set k-covering problem: scp48-7
Observations

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

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

• **Elitist strategy:** best solutions are passed without change from one generation to the next (incumbent is kept)
Observations

- Random method: keys are randomly generated so solutions are always vectors of random keys.
- Elitist strategy: best solutions are passed without change from one generation to the next (incumbent is kept).
- Child inherits more characteristics of elite parent: one parent is always selected (with replacement) from the small elite set and probability that child inherits key of elite parent $> 0.5$ Not so in the RKGA of Bean.
Observations

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

- **Elitist strategy:** best solutions are passed without change from one generation to the next (incumbent is kept).

- **Child inherits more characteristics of elite parent:** one parent is always selected (with replacement) from the small elite set and probability that child inherits key of elite parent $> 0.5$. Not so in the RKGA of Bean.

- **No mutation in crossover:** mutants are used instead (they play same role as mutation in GAs ... help escape local optima).
Framework for biased random-key genetic algorithms

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

Stop
Framework for biased random-key genetic algorithms

1. Generate $P$ vectors of random keys
2. Decode each vector of random keys
3. Sort solutions by their costs
4. Stopping rule satisfied?
   - no: Classify solutions as elite or non-elite
     - Copy elite solutions to next population
   - yes: Generate mutants in next population
5. Combine elite and non-elite solutions and add children to next population
Framework for biased random-key genetic algorithms

1. Generate P vectors of random keys
2. Decode each vector of random keys
3. Sort solutions by their costs
4. Classify solutions as elite or non-elite
5. Copy elite solutions to next population
6. Generate mutants in next population
7. Combine elite and non-elite solutions and add children to next population
8. Stopping rule satisfied?
   - no: go back to 3
   - yes: stop

Problem independent

Problem dependent
Decoding of random key vectors can be done in parallel

1. Generate P vectors of random keys
2. Decode each vector of random keys
3. Sort solutions by their costs
4. Classify solutions as elite or non-elite
5. Copy elite solutions to next population
6. Generate mutants in next population
7. Combine elite and non-elite solutions and add children to next population
8. Stopping rule satisfied?
   - Yes: stop
   - No: go back to step 2
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)

![Graph](image)

- **BRKGA solutions**
- **Random multi-start solutions**
- **Best random solution**
- **Optimal value**

Time (ibm t41 secs)
BRKGA in multi-start strategy

begin

Generate P vectors of random keys
Decode each vector of random keys

Classify solutions as elite or non-elite
Sort solutions by their costs
Restart rule satisfied?

no

yes

if incumbent improved, update incumbent

output incumbent

stopping rule satisfied?

no

yes

begin

if incumbent improved, update incumbent

stop
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 \times \text{probability of taking over 690 iterations given it took over 345} = \frac{1}{4} \times \frac{1}{4} = \frac{1}{16}
Probability that algorithm will still be running after $K$ periods of $345$ iterations: $\frac{1}{4^K}$
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): $1/4^5 \approx 0.0977\%$
Probability that algorithm will still be running after $K$ periods of 345 iterations: $\frac{1}{4^K}$

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

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 \( \tau_1, \tau_1 + \tau_2, \tau_1 + \tau_2 + \tau_3, \ldots \) when the algorithm is restarted from scratch.
- Luby et al. (1993) prove that the optimal restart strategy uses \( \tau_1 = \tau_2 = \tau_3 = \ldots = \tau^* \), where \( \tau^* \) is a constant.
Restart strategy for BRKGA

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

- **We conjecture that** number of iterations between improvement of the incumbent *(best so far)* solution varies less w.r.t. heuristic/ instance/ target than run times.

- **We propose the following restart strategy:** Keep track of the last generation when the incumbent improved and restart BRKGA if \( K \) generations have gone by without improvement.

- **We call this strategy** restart(\( K \))
Example of restart strategy for BRKGA: Load balancing

**restart strategy:**
- restart(2000)
- no restart

Cumulative probability

Iterations to BKS

With restart

Without restart

Amazon

UNICAMP – Campinas, Brazil  Mar. 3, 2015
Packing with a BRKGA
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,”

Software: http://mauricio.resende.info/src/brkgaAPI
An example BRKGA: Packing weighted rectangles
Reference


Tech report:
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 \((w[i], h[i])\), \(i = 1,\ldots,N\), 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 \((w[i], h[i])\), \(i = 1, \ldots, N\), each of width \(w[i]\), height \(h[i]\), and value \(v[i]\);
Constrained orthogonal packing

- $r[i]$ rectangles of type $i = 1, \ldots, N$ 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:
  \[
  0 \leq P[i] \leq r[i] \leq Q[i]
  \]
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 \( i = 1, \ldots, N \) 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 \( i = 1, \ldots, N \), 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:

Objective

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

Objective

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

Objective

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

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)
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.
- $X = ( X[1], ..., X[N'], X[N'+1], ..., X[2N'] )$
Encoding

• Solutions are encoded as vectors $X$ of
  random keys, where $Q[i]$ is the maximum number of rectangles of type $i$ (for $i = 1, \ldots, N$) that can be packed.

  $X = (X[1], \ldots, X[N'], X[N'+1], \ldots, X[2N'])$

  Rectangle type packing sequence (RTPS)
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.

• $X = (X[1], ..., X[N'], X[N'+1], ..., X[2N'])$

  - Rectangle type packing sequence (RTPS)
  - Vector of placement procedures (VPP)
Decoding

• **Simple heuristic to pack rectangles:**
  - Make $Q[i]$ copies of rectangle $i$, for $i = 1, ..., N$.
  - 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 $i = 1, \ldots, N$.
  
  – 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.
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

- 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** 
  \( ER[i] < ER[j] \) if \( y[i] < y[j] \) or \( y[i] = y[j] \) and \( x[i] < x[j] \).
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.
Similar infeasibilities are observed if 2, 3, or 4 is the first rectangle in the 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\[j\]

ERS\[i\] < ERS\[j\]
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
Packing with a BRKGA
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 Hadjicostantinou & 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:
  - 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 Hadjicostantisnou & 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

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

* For large random: number of best average solutions / total instance classes
Minimum, average, and maximum solution times (secs) for BRKGA (BL-LB-L-4NR)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Min solution time (secs)</th>
<th>Avg solution time (secs)</th>
<th>Max solution time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>From literature (optimal)</td>
<td>0.00</td>
<td>0.05</td>
<td>0.55</td>
</tr>
<tr>
<td>Large random</td>
<td>1.78</td>
<td>23.85</td>
<td>72.70</td>
</tr>
<tr>
<td>Zero-waste</td>
<td>0.01</td>
<td>82.21</td>
<td>808.03</td>
</tr>
<tr>
<td>Doubly constrained</td>
<td>0.00</td>
<td>1.16</td>
<td>16.87</td>
</tr>
</tbody>
</table>
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
Some remarks

We have extended this to 3D packing:


3D bin packing

3D bin packing problem

Container (bin) of fixed dimension

Boxes of different dimensions

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

Packing with a BRKGA
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

1. 
2. 
3. 
4. 
5. 
6. 

1. 
2. 
3. 
4. 
5. 
6.
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.

$$X = (x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{2n}, x_{2n+1}, x_{2n+2}, \ldots, x_{3n})$$

- **Box packing sequence**
- **Placement heuristic**
- **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 $x_{n+i} < \frac{1}{2}$ then use back-bottom-left to pack $i$-th box
   • if $x_{n+i} \geq \frac{1}{2}$ 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: \(aNB\)

– \(aNB = NB + \left( \frac{\text{LeastLoad}}{\text{BinVolume}} \right)\), where

  ✓ LeastLoad is load on least loaded bin
  ✓ BinVolume is volume of bin: \(H \times W \times L\)
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](http://www.diku.dk/~pisinger/codes/html)
  - 8 classes
  - 40 instances per class
  - 10 instances for each value of $n \in \{50, 100, 150, 200\}$
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

<table>
<thead>
<tr>
<th>Class</th>
<th>Bin size</th>
<th>BRKGA</th>
<th>GRASP</th>
<th>TS3</th>
<th>TS2PACK</th>
<th>GLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100³</td>
<td>127.3</td>
<td>127.3</td>
<td>127.9</td>
<td>128.2</td>
<td>128.3</td>
</tr>
<tr>
<td>2</td>
<td>100³</td>
<td>125.5</td>
<td>125.8</td>
<td>126.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>100³</td>
<td>126.5</td>
<td>126.9</td>
<td>127.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>100³</td>
<td>294.0</td>
<td>294.0</td>
<td>294.0</td>
<td>293.9</td>
<td>294.2</td>
</tr>
<tr>
<td>5</td>
<td>100³</td>
<td>70.4</td>
<td>70.5</td>
<td>71.4</td>
<td>71.0</td>
<td>70.8</td>
</tr>
<tr>
<td>6</td>
<td>10³</td>
<td>95.0</td>
<td>95.4</td>
<td>96.1</td>
<td>95.8</td>
<td>96.0</td>
</tr>
<tr>
<td>7</td>
<td>40³</td>
<td>58.2</td>
<td>59.4</td>
<td>60.0</td>
<td>59.0</td>
<td>59.0</td>
</tr>
<tr>
<td>8</td>
<td>100³</td>
<td>80.9</td>
<td>82.0</td>
<td>82.6</td>
<td>81.9</td>
<td>81.9</td>
</tr>
<tr>
<td>Sum(rows 1, 4-8):</td>
<td>725.8</td>
<td>728.6</td>
<td>732.0</td>
<td>729.8</td>
<td>730.2</td>
<td></td>
</tr>
<tr>
<td>Sum(rows 1-8):</td>
<td>977.8</td>
<td>981.3</td>
<td>986.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concluding remarks

• Reviewed BRKGA framework
• Applied framework to
  – 2D/3D packing to maximize value packed
  – 2D/3D bin packing to minimize number of bins
• All decoders were simple heuristics
• BRKGA “learned” how to “operate” the heuristics
• In all cases, several new best known solutions were produced
Other applications of BRKGA

Telecommunications

- Survey (R., 2012)

- 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)
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)
Other applications of BRKGA

Manufacturing and facility layout

– Manufacturing cell formation (Gonçalves & R., 2004)
– Minimization of open stacks (Gonçalves et al., 2014)
– Unequal area facility layout (Gonçalves & R., 2014)
– Minimization of tool switches (Chaves et al., 2014)
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 (R. et al., 2012)
- Steiner triple covering (Lucena 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

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-constrained 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