

Configuração de parâmetros de metaheurísticas com o pacote irace

17 de outubro de 2019

Introdução

Muitos algoritmos propostos para resolver problemas de otimização envolvem um grande número de parâmetros que precisam ser cuidadosamente configurados para alcançar seu melhor desempenho, por exemplo

- ▶ algoritmos evolutivos (taxa de mutação, crossover, tamanho da população, etc.)
- ▶ CPLEX (estratégias de branching, pré-processamento, etc.)

Introdução

Por muitos anos, o ajuste de parâmetros dos algoritmos de otimização foram realizados de maneira manual, ou seja,

- ▶ o desenvolvedor escolhe primeiro algumas configurações de parâmetro
- ▶ executa experimentos para testá-los
- ▶ examina os resultados e decide se deseja testar outras configurações ou interromper o processo

Introdução

Embora a calibração manual de parâmetros seja melhor do que nenhuma calibração, existem algumas desvantagens, como:

- ▶ esforço humano despendido
- ▶ guiada pela experiência e intuição pessoais e, portanto, tendenciosos e não reproduzíveis
- ▶ poucas alternativas de configurações de parâmetros exploradas
- ▶ geralmente as mesmas instâncias usadas durante a fase de ajuste de parâmetros e de avaliação do algoritmo final

Introdução

Por causa dessas desvantagens, a calibração manual de parâmetros tem sido substituída por métodos automáticos de configuração de parâmetros

A configuração automática de algoritmos pode ser descrita como o problema de encontrar boas configurações de parâmetros para resolver instâncias ainda não vistas, aprendendo sobre um conjunto de instâncias de treinamento do problema

Introdução

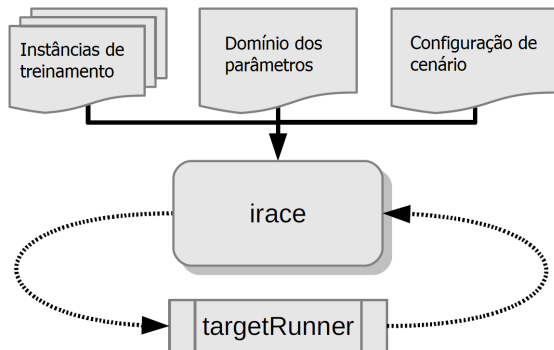
Existem duas fases claramente delimitadas

- ▶ 1 - escolha de uma configuração, dado um conjunto de instâncias de treinamento representativas de um problema específico
- ▶ 2 - teste, onde a configuração escolhida é usada para resolver instâncias não vistas do mesmo problema.

O objetivo é encontrar, durante a fase 1, uma configuração de parâmetros que minimize os custos sobre o conjunto de instâncias a ser visto durante a fase 2.

irace

Pacote proposto para configurar automaticamente os algoritmos de otimização encontrando as configurações mais adequadas, dado um conjunto de instâncias



<http://iridia.ulb.ac.be/lperez/matheuristics2016-irace/slides/matheuristics2016-tutorial.pdf>

Publicado em

- López-Ibáñez, M. et al. **The irace package: Iterated Racing for Automatic Algorithm Configuration**. Operations Research Perspectives, v. 3, p. 43 - 58, 2016.

Operations Research Perspectives 3 (2016) 43–58



Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp



The irace package: Iterated racing for automatic algorithm configuration



Manuel López-Ibáñez^{a,*}, Jérémie Dubois-Lacoste^b, Leslie Pérez Cáceres^b, Mauro Birattari^b, Thomas Stützle^b

^aAlliance Manchester Business School, University of Manchester, UK

^bIRIDIA, Université Libre de Bruxelles (ULB), CP 194/6, Av. F. Roosevelt 50 - B-1050 Brussels, Belgium

ARTICLE INFO

Article history:

Available online 21 September 2016

Keywords:

Automatic algorithm configuration
Racing
Parameter tuning

ABSTRACT

Modern optimization algorithms typically require the setting of a large number of parameters to optimize their performance. The immediate goal of automatic algorithm configuration is to find, automatically, the best parameter settings of an optimizer. Ultimately, automatic algorithm configuration has the potential to lead to new design paradigms for optimization software. The irace package is a software package that implements a number of automatic configuration procedures. In particular, it offers *iterated* racing procedures, which have been used successfully to automatically configure various state-of-the-art algorithms. The iterated racing procedures implemented in irace include the iterated F-race algorithm and several extensions and improvements over it. In this paper, we describe the rationale underlying the iterated racing procedures and introduce a number of recent extensions. Among these, we introduce a restart mechanism to avoid premature convergence, the use of truncated sampling distributions to handle correctly parameter bounds, and an elitist racing procedure for ensuring that the best configurations returned are also those evaluated in the highest number of training instances. We experimentally evaluate the most recent version of irace and demonstrate with a number of example applications the use and potential of irace, in particular, and automatic algorithm configuration, in general.

© 2016 The Authors. Published by Elsevier Ltd.

This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

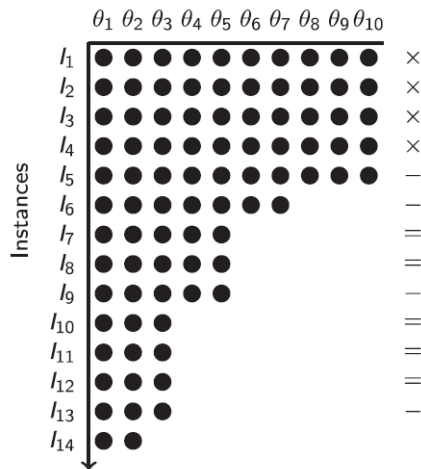
irace

- ▶ É um software livre
- ▶ Implementado como um pacote do R
- ▶ Documentação completa e atualizada
 - ▶ <http://iridia.ulb.ac.be/irace/>
 - ▶ <http://iridia.ulb.ac.be/irace/README.html>
 - ▶ <https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf>
 - ▶ <http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2011-004.pdf>

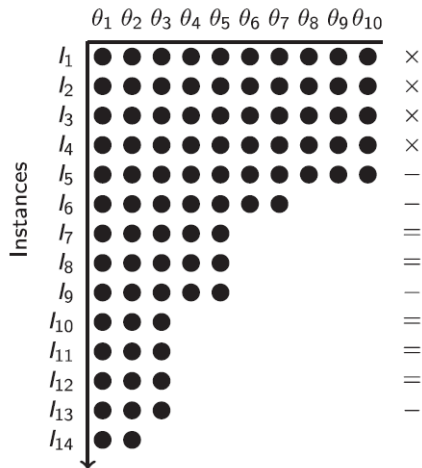
irace

O pacote irace é composto por métodos de “corridas iteradas” (*iterated racing*), que

- ▶ inicia com um conjunto finito de configurações candidatas θ_i
- ▶ em cada etapa da corrida, as configurações candidatas são avaliadas em uma única instância I_j

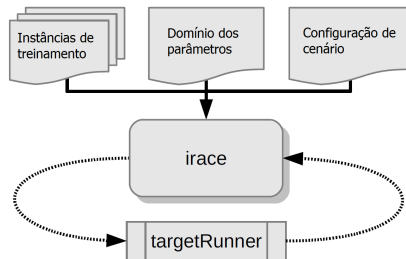


- ▶ após algumas etapas, as configurações candidatas que apresentam desempenho estatisticamente pior que pelo menos uma outra são descartadas
- ▶ a corrida continua com as configurações restantes



Precisamos definir:

- ▶ Conjunto de instâncias
- ▶ Os parâmetros e seus domínio
- ▶ Cenário
- ▶ TargetRunner

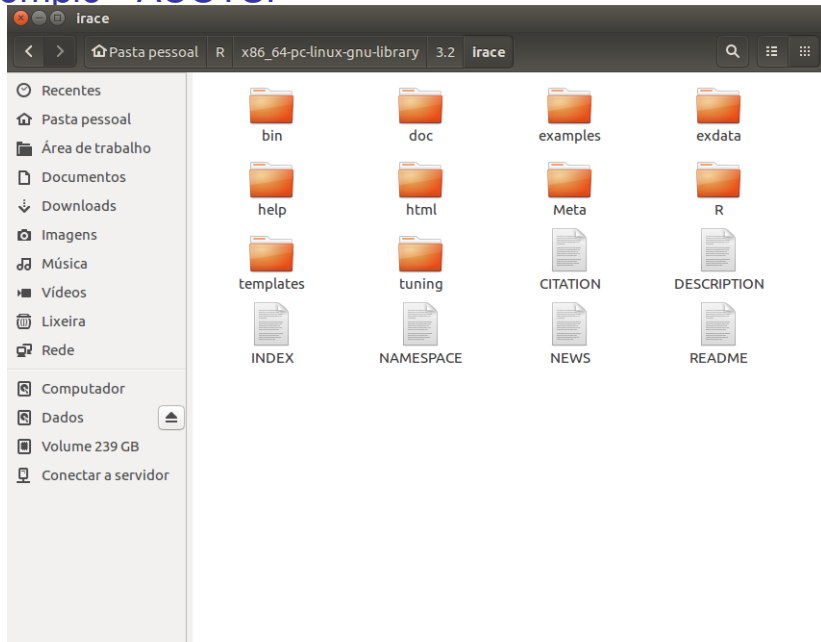


Exemplo - ACOTSP

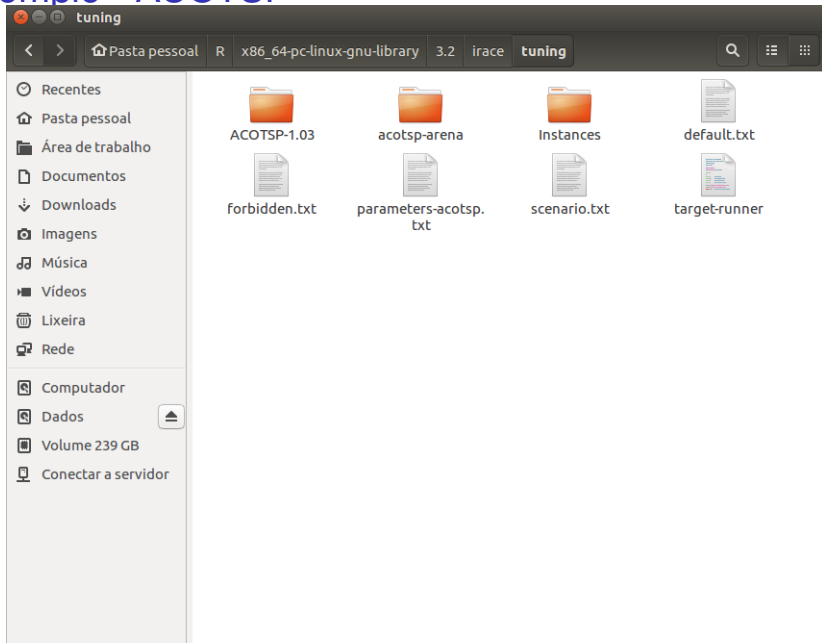
ACOTSP é um pacote de software que implementa vários algoritmos aplicados ao TSP (*Traveling Salesman Problem*) simétrico. Os algoritmos implementados são Ant System (as), Elitist Ant System (eas), MAX-MIN Ant System (mmas), Rank-based version of Ant System (ras) e Ant Colony System (acs).

<http://iridia.ulb.ac.be/irace/>

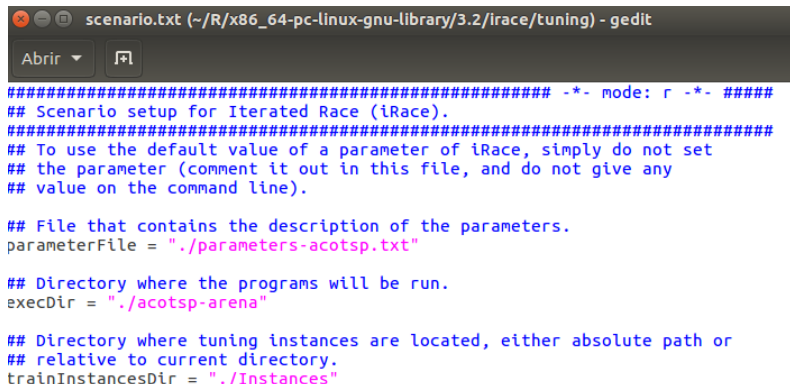
Exemplo - ACOTSP



Exemplo - ACOTSP



Exemplo - ACOTSP - Scenario.txt



```
##### -*- mode: r -*- #####
## Scenario setup for Iterated Race (iRace).
#####
## To use the default value of a parameter of iRace, simply do not set
## the parameter (comment it out in this file, and do not give any
## value on the command line).

## File that contains the description of the parameters.
parameterFile = "./parameters-acotsp.txt"

## Directory where the programs will be run.
execDir = "./acotsp-arena"

## Directory where tuning instances are located, either absolute path or
## relative to current directory.
trainInstancesDir = "./Instances"
```


Exemplo - ACOTSP - Scenario.txt

```
## The maximum number of runs (invocations of targetRunner) that will performed. It
## determines the (maximum) budget of experiments for the tuning.
maxExperiments = 5000

## File that contains a set of initial configurations. If empty or NULL,
## all initial configurations are randomly generated.
# configurationsFile = ""

## Indicates the number of decimal places to be considered for the
## real parameters.
digits = 2

## A value of 0 silences all debug messages. Higher values provide
## more verbose debug messages.
# debugLevel = 0

targetRunner = "../tuning/target-runner"

## END of scenario file
#####
```

Parâmetros

- ▶ Cada parâmetro deve ser associado à um tipo que define o seu domínio e o modo como o irace vai lidar com eles (“r”, “i”, “c” e “o”)
- ▶ “r” e “i”: um intervalo fechado do tipo ‘(<limite inferior>, <limite superior>)’ deve ser especificado
- ▶ “c” (categóricos): possuem um conjunto finito de valores ‘(<valor 1>, ..., <valor n>)
- ▶ “o” (ordinais): são definidos por um conjunto ordenado de possíveis valores no mesmo formato que os parâmetros categóricos (ex: “low”, “midium”, “high”)

Exemplo - ACOTSP - parameters-acotsp.txt

Descrição do espaço de parâmetros é dada como <nome> <label> <tipo> <intervalo> [| <condição>].

```
parameters-acotsp.txt (~/R/x86_64-pc-linux-gnu-library/3.2/irace/tuning) - gedit
Abrir ▾  Salvar

### Parameter file for the ACOTSP software
# name      switch      type      values      [conditions (using R syntax)]
algorithm   "--"      c      (as,mmas,eas,ras,acs)
localsearch "--localsearch "  c      (0, 1, 2, 3)
alpha       "--alpha "    r      (0.00, 5.00)
beta        "--beta "   r      (0.00, 10.00)
rho         "--rho "    r      (0.01, 1.00)
ants        "--ants "    i      (5, 100)
q0          "--q0 "     r      (0.0, 1.0)
rasrank     "--rasranks " i      (1, 100)
elitists    "--elitists " i      (1, 750)
nnls        "--nnls "   i      (5, 50)
dlb         "--dlb "    c      (0, 1)

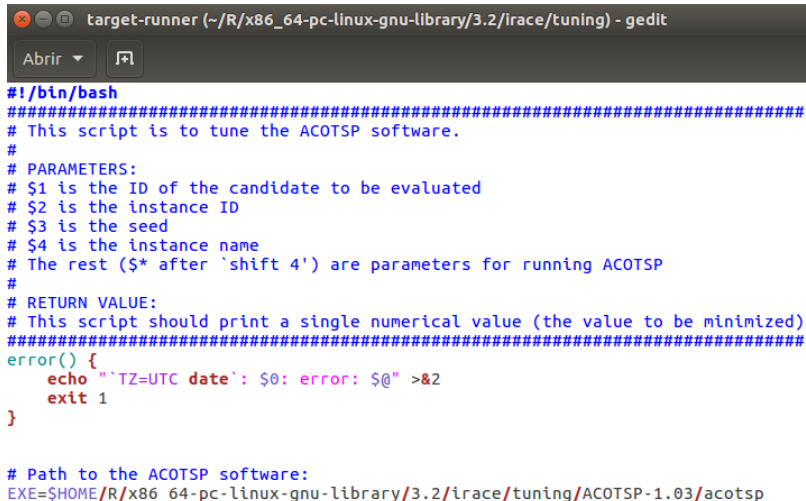
| algorithm == "acs"
| algorithm == "ras"
| algorithm == "eas"
| localsearch %in% c(1, 2, 3)
| localsearch %in% c(1,2,3)
```

Exemplo: - -eas - -localsearch 0 - -alpha 2.92 - -beta 3.06 - -rho 0.6 -
-ants 80

Exemplo - ACOTSP - Target Runner

Saída do irace:

```
target-runner 1 113 734718556 /home/user/instances/tsp/2000-533.tsp  
- -eas - -localsearch 0 - -alpha 2.92 - -beta 3.06 - -rho 0.6 - -ants 80
```



```
#!/bin/bash  
#####  
# This script is to tune the ACOTSP software.  
#  
# PARAMETERS:  
# $1 is the ID of the candidate to be evaluated  
# $2 is the instance ID  
# $3 is the seed  
# $4 is the instance name  
# The rest ($* after `shift 4`) are parameters for running ACOTSP  
#  
# RETURN VALUE:  
# This script should print a single numerical value (the value to be minimized)  
#####  
error() {  
    echo "`TZ=UTC date`: $0: error: @$" >&2  
    exit 1  
}  
  
# Path to the ACOTSP software:  
EXE=$HOME/R/x86_64-pc-linux-gnu-library/3.2/irace/tuning/ACOTSP-1.03/acotsp
```

Exemplo - ACOTSP - Target Runner

```
# Fixed parameters that should be always passed to ACOTSP.
# The time to be used is always 10 seconds, and we want only one run:
FIXED_PARAMS="--tries 1 --time 10 --quiet "

CONFIG_ID="$1"
INSTANCE_ID="$2"
SEED="$3"
INSTANCE="$4"
# All other parameters are the candidate parameters to be passed to program
shift 4 || error "Not enough parameters"
CONFIG_PARAMS=$*

STDOUT=c${CONFIG_ID}-${INSTANCE_ID}-${SEED}.stdout
STDERR=c${CONFIG_ID}-${INSTANCE_ID}-${SEED}.stderr

if [ ! -x "${EXE}" ]; then
    error "${EXE}: not found or not executable (pwd: $(pwd))"
fi
```

Exemplo - ACOTSP - Target Runner

```
# Now we can call ACOTSP by building a command line with all parameters for it
$EXE ${FIXED_PARAMS} -i $INSTANCE --seed $SEED ${CONFIG_PARAMS} 1> $STDOUT 2> $STDERR

# The output of the candidate $CONFIG_ID should be written in the file
# c${CONFIG_ID}.stdout (see target runner for ACOTSP).
# Does this file exist?
if [ ! -s "${STDOUT}" ]; then
    # In this case, the file does not exist. Let's exit with a value
    # different from 0. In this case irace will stop with an error.
    error "${STDOUT}: No such file or directory"
fi
```

Gera a entrada do seu algoritmo...

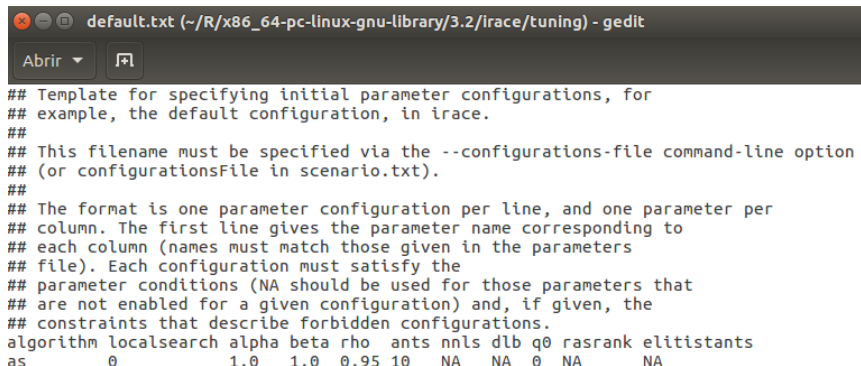
Exemplo - ACOTSP - Target Runner

```
# Ok, the file exist. It contains the whole output written by ACOTSP.
# This script should return a single numerical value, the best objective
# value found by this run of ACOTSP. The following line is to extract
# this value from the file containing ACOTSP output.
COST=$(cat ${STDOUT} | grep -o -E 'Best [-+0-9.e]+' | cut -d ' ' -f2)
if ! [[ "$COST" =~ ^[-+0-9.e]+$ ]] ; then
    error "${STDOUT}: Output is not a number"
fi

# Print it!
echo "$COST"

# We are done with our duty. Clean files and exit with 0 (no error).
rm -f "${STDOUT}" "${STDERR}"
rm -f best.* stat.* cmp.*
exit 0
```

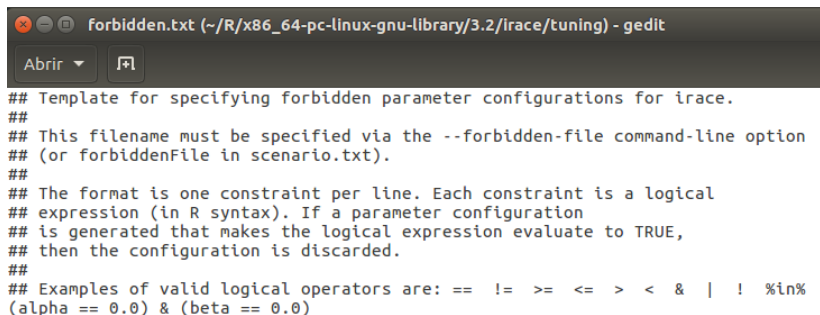
Exemplo - ACOTSP - Default



A screenshot of a gedit window titled "default.txt (~/R/x86_64-pc-linux-gnu-library/3.2/irace/tuning) - gedit". The window contains a template for specifying initial parameter configurations. The text is as follows:

```
## Template for specifying initial parameter configurations, for
## example, the default configuration, in irace.
##
## This filename must be specified via the --configurations-file command-line option
## (or configurationsFile in scenario.txt).
##
## The format is one parameter configuration per line, and one parameter per
## column. The first line gives the parameter name corresponding to
## each column (names must match those given in the parameters
## file). Each configuration must satisfy the
## parameter conditions (NA should be used for those parameters that
## are not enabled for a given configuration) and, if given, the
## constraints that describe forbidden configurations.
algorithm localsearch alpha beta rho ants nnls dlb q0 rasrank elitstants
as          0          1.0  1.0  0.95 10  NA  NA  0  NA      NA
```


Exemplo - ACOTSP - Forbidden

A screenshot of a gedit text editor window. The title bar shows the file name 'forbidden.txt' and the path '~ / R / x86_64 - pc - linux - gnu - library / 3.2 / irace / tuning'. Below the title bar are two buttons: 'Abrir' with a dropdown arrow and a file icon. The main text area contains several lines of comments in a monospaced font, explaining the purpose and usage of the file.

```
## Template for specifying forbidden parameter configurations for irace.  
##  
## This filename must be specified via the --forbidden-file command-line option  
## (or forbiddenFile in scenario.txt).  
##  
## The format is one constraint per line. Each constraint is a logical  
## expression (in R syntax). If a parameter configuration  
## is generated that makes the logical expression evaluate to TRUE,  
## then the configuration is discarded.  
##  
## Examples of valid logical operators are: == != >= <= > < & | ! %in%  
(alpha == 0.0) & (beta == 0.0)
```

Exemplo - ACOTSP

```
larissa@Ap71: ~/R/x86_64-pc-linux-gnu-library/3.2/irace/tuning
larissa@Ap71:~/R/x86_64-pc-linux-gnu-library/3.2/irace/tuning$ irace
-----
# irace: An implementation in R of (Elitist) Iterated Racing
# Version: 3.3.2238:2239
# Copyright (C) 2010-2019
# Manuel Lopez-Ibanez <manuel.lopez-ibanez@manchester.ac.uk>
# Jeremie Dubois-Lacoste
# Leslie Perez Caceres <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions. See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# irace builds upon previous code from the race package:
#   race: Racing methods for the selection of the best
#   Copyright (C) 2003 Mauro Birattari
#-----
# installed at: /home/larissa/R/x86_64-pc-linux-gnu-library/3.2/irace
# called with:
Warning: A default scenario file './scenario.txt' has been found and will be read
# 2019-10-09 08:18:50 -03: Initialization
# Elitist race
# Elitist new instances: 1
# Elitist limit: 2
# nbIterations: 5
# minNbSurvival: 5
# nbParameters: 11
# seed: 9046424
# confidence level: 0.95
# budget: 5000
# mu: 5
# deterministic: FALSE

# 2019-10-09 08:18:50 -03: Iteration 1 of 5
# experimentsUsedSoFar: 0
# remainingBudget: 5000
# currentBudget: 1000
# nbConfigurations: 166
# Markers:
#   X No test is performed.
#   - The test is performed and some configurations are discarded.
#   = The test is performed but no configuration is discarded.
#   ! The test is performed and configurations could be discarded but elite configurations are preserved.
#   . All alive configurations are elite and nothing is discarded

-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Instance| Alive| Best| Mean best| Exp so far| W time| rho|KenW| Qvar|
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Parâmetros do irace

- ▶ $B = \text{maxExperiments}$: máximo de experimentos que serão realizados (sugestão: [1 000, 100 000])
- ▶ μ = valor usado para calcular o número mínimo de iterações
- ▶ $N^{\text{Min}} = \text{minNbSurvival}$: mínimo de configuração “vivas” necessárias para continuar a iteração
- ▶ T^{first} : quantidade de instâncias antes de avaliar estatisticamente uma configuração (default = 5)
- ▶ T^{each} : novos testes estatísticos realizados a cada T^{each} instâncias (default = 1)

Exemplo - ACOTSP - Saída

Warning: A default scenario file './scenario.txt' has been found and will be read

2019-10-09 08:18:50 -03: Initialization

Elitist race

Elitist new instances: 1

Elitist limit: 2

nbIterations: 5

minNbSurvival: 5

nbParameters: 11

seed: 90446424

confidence level: 0.95

budget: 5000

mu: 5

deterministic: FALSE

2019-10-09 08:18:50 -03: Iteration 1 of 5

experimentsUsedSoFar: 0

remainingBudget: 5000

currentBudget: 1000

nbConfigurations: 166

Markers:

x No test is performed.

- The test is performed and some configurations are discarded.

= The test is performed but no configuration is discarded.

! The test is performed and configurations could be discarded but elite configurations are not.

. All alive configurations are elite and nothing is discarded

Exemplo - ACOTSP - Saída

- ▶ Estima a quantidade de iterações (*racess*):
$$N^{iter} = \lfloor 2 + \log_2 N^{param} \rfloor$$
- ▶ Cada iteração é limitada por um *budget*:
$$B_j = (B - B_{used}) / (N^{iter} - j + 1), \text{ em que } j = 1, \dots, N^{iter}$$
- ▶ A quantidade de configurações amostradas em casa iteração é dada por: $|\Theta_j| = N_j = \lfloor B_j / \mu + \min(5, j) \rfloor$

Exemplo - ACOTSP - Saída

	Instance	Alive	Best	Mean best	Exp so far	W time	rho	KenW	Qvar
x	1	166	20	32871413.00	166	00:28:23	NA	NA	NA
x	2	166	3	32837657.00	332	00:28:24	+0.97	0.99	0.0019
x	3	166	3	32725404.33	498	00:28:25	+0.97	0.98	0.0021
x	4	166	96	32783960.00	664	00:28:25	+0.97	0.98	0.0020
-	5	12	96	32778879.80	830	00:28:21	+0.06	0.25	0.8424
=	6	12	96	32790201.50	842	00:02:02	+0.05	0.21	0.8752
=	7	12	96	32817562.00	854	00:02:02	+0.11	0.23	0.8233

Best-so-far configuration: 96 mean value: 32817562.00

Description of the best-so-far configuration:

.ID.	algorithm	localsearch	alpha	beta	rho	ants	q0	rasrank	elitistants	nnls	dlb	.PARENT.	
96	96	acs	2	1.86	5.95	0.13	29	0.6	NA	NA	12	0	NA

2019-10-09 10:44:56 -03: Elite configurations (first number is the configuration ID; listed from best to worst according to the sum of ranks):

	algorithm	localsearch	alpha	beta	rho	ants	q0	rasrank	elitistants	nnls	dlb
96	acs	2	1.86	5.95	0.13	29	0.60	NA	NA	12	0
20	acs	3	4.97	8.62	0.47	75	0.16	NA	NA	16	0
50	as	3	4.02	4.32	0.66	27	NA	NA	NA	8	1
57	mmas	3	1.92	5.29	0.61	64	NA	NA	NA	8	1
140	ead	3	3.20	2.99	0.20	22	NA	NA	194	17	1

As corridas terminam quando $B_j < N_j^{surv}$ ou $N_j^{surv} \leq N^{min}$

Exemplo - ACOTSP - Saída

2019-10-09 10:44:56 -03: Iteration 2 of 5

experimentsUsedSoFar: 854

remainingBudget: 4146

currentBudget: 1036

nbConfigurations: 133

Markers:

x No test is performed.

- The test is performed and some configurations are discarded.

= The test is performed but no configuration is discarded.

! The test is performed and configurations could be discarded but elite configurations are preserved.

. All alive configurations are elite and nothing is discarded

	Instance	Alive	Best	Mean best	Exp so far	W time	rho	KenW	Qvar
[x]	8	133	244	32519741.00	133	00:22:42	NA	NA	NA
[x]	3	133	244	32407264.00	261	00:21:50	+0.96	0.98	0.0020
[x]	1	133	244	32533812.67	389	00:21:52	+0.97	0.98	0.0022
[x]	7	133	244	32611998.00	517	00:21:51	+0.97	0.98	0.0027
[-]	6	14	244	32624181.40	645	00:21:51	+0.72	0.78	0.2915
[-]	4	8	244	32658089.17	654	00:01:32	+0.71	0.76	0.2388
[!]	2	8	244	32644909.00	657	00:00:30	+0.73	0.77	0.2276
[-]	5	3	244	32649517.38	660	00:00:30	+0.23	0.33	0.5210

Best-so-far configuration: 244 mean value: 32649517.38

Description of the best-so-far configuration:

.ID.	algorithm	localsearch	alpha	beta	rho	ants	q0	rasrank	elitists	nnls	dlb	.PARENT.	
244	244	acs	3	1.67	6.08	0.35	36	0.65	NA	NA	13	1	96

2019-10-09 12:37:38 -03: Elite configurations (first number is the configuration ID; listed from best to worst according to the sum of ranks):

Exemplo - ACOTSP - Saída

```
# 2019-10-09 22:31:22 -03: Stopped because there is not enough budget left to race more than
the minimum (5)
# You may either increase the budget or set 'minNbSurvival' to a lower value
# Iteration: 9
# nbIterations: 9
# experimentsUsedSoFar: 4997
# timeUsed: 0
# remainingBudget: 3
# currentBudget: 3
# number of elites: 5
# nbConfigurations: 4
# Best configurations (first number is the configuration ID; listed from best to worst according to
|the sum of ranks):
  algorithm localsearch alpha beta  rho ants  q0 rasrank elitists nls dlb
513      acs          3  1.87 3.57 0.46  43 0.16      NA      NA  11  1
733      acs          3  2.07 6.93 0.34  41 0.72      NA      NA  14  1
661      acs          3  2.10 7.31 0.34  42 0.66      NA      NA  16  1
728      acs          3  2.08 7.64 0.34  41 0.70      NA      NA  15  1
655      acs          3  2.14 7.29 0.31  41 0.65      NA      NA  15  1
# Best configurations as commandlines (first number is the configuration ID; same order as above):
513 --acs --localsearch 3 --alpha 1.87 --beta 3.57 --rho 0.46 --ants 43 --q0 0.16 --nls 11 --dlb 1
733 --acs --localsearch 3 --alpha 2.07 --beta 6.93 --rho 0.34 --ants 41 --q0 0.72 --nls 14 --dlb 1
661 --acs --localsearch 3 --alpha 2.1 --beta 7.31 --rho 0.34 --ants 42 --q0 0.66 --nls 16 --dlb 1
728 --acs --localsearch 3 --alpha 2.08 --beta 7.64 --rho 0.34 --ants 41 --q0 0.7 --nls 15 --dlb 1
655 --acs --localsearch 3 --alpha 2.14 --beta 7.29 --rho 0.31 --ants 41 --q0 0.65 --nls 15 --dlb 1
```


Dúvidas?

Larissa Oliveira: Itido@icmc.usp.br

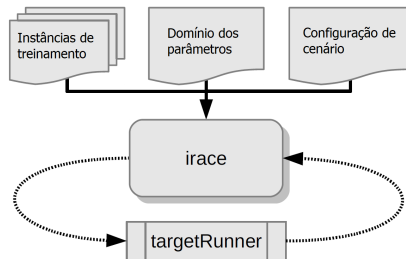
Configuração de parâmetros de metaheurísticas com o pacote irace

17 de outubro de 2019

irace

Precisamos definir:

- ▶ Conjunto de instâncias
- ▶ Os parâmetros e seus domínio
- ▶ Cenário
- ▶ TargetRunner



Atividades do segundo dia

- ▶ Instalar o R
- ▶ Instalar o irace
- ▶ Calibrar o ACOTSP
- ▶ Calibrar o BRKGATSP

<https://github.com/larissatebaldi/iraceUnifesp>

Instalação (GNU/Linux)

- ▶ **Passo 1:** Instalar R
 - ▶ `$ sudo apt-get install r-base`
- ▶ **Passo 2:** Instalar o pacote irace
 - ▶ `$ R`
 - ▶ `R> install.packages("irace")`
 - ▶ `R> library(irace)`
 - ▶ `R> system.file(package="irace")` (esse é o IRACE_HOME)
 - ▶ `R> CTRL+d`

Para o GNU/Linux e OS X: Adicionar ao final do `.bash_profile`, `.bashrc` (gedit `.bashrc`) ou `.profile`:

```
export IRACE_HOME= *local onde irace está instalado*  
export PATH=$IRACE_HOME/bin/:$PATH
```

Instalação (Windows)

- ▶ **Passo 1:** Instalar RStudio
- ▶ **Passo 2:** Instalar o pacote irace (dentro do RStudio)
 - ▶ `R> install.packages("irace")`
 - ▶ `R> library(irace)`
 - ▶ `R> system.file(package="irace")` (esse é o IRACE_HOME)
 - ▶ `R> CTRL+d`

Para o Windows: Adicionar às variáveis de ambiente (PATH), por exemplo:

`C:\Program Files\R\R-3.2.2\bin;`

`C:\Users\Larissa\Documents\R\win-library\3.2\irace\bin`

Executando o irace

- ▶ **Passo 1:** Criar a pasta “tuning” dentro da pasta do irace
- ▶ **Passo 2:** Criar as pastas “Instances” e “Arena” dentro da pasta “tuning”
- ▶ **Passo 3:** Adicionar os arquivos*:
 - ▶ scenario.txt
 - ▶ instances-list.txt
 - ▶ parameters.txt
 - ▶ target-runner
 - ▶ seu executável

*Entre outros. Há templates disponíveis na pasta /irace/templates/

irace (GNU/Linux)

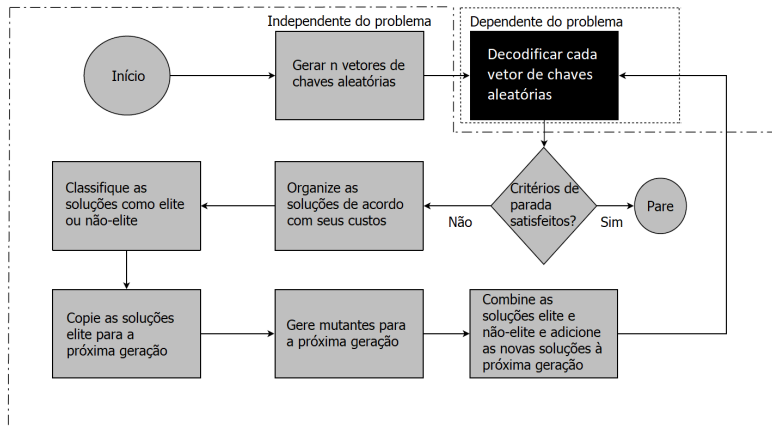
Biased random-key genetic algorithm

A metaheurística evolutiva BRKGA foi proposta por Gonçalves e Resende (2011), baseada na RKGA de Bean (1994), em que:

- ▶ cada solução é representada por um vetor de n chaves aleatórias
- ▶ **cada chave é um número real aleatório entre $[0,1)$**
- ▶ e um decodificador transforma cada vetor em uma solução do problema e retornar um valor representativo

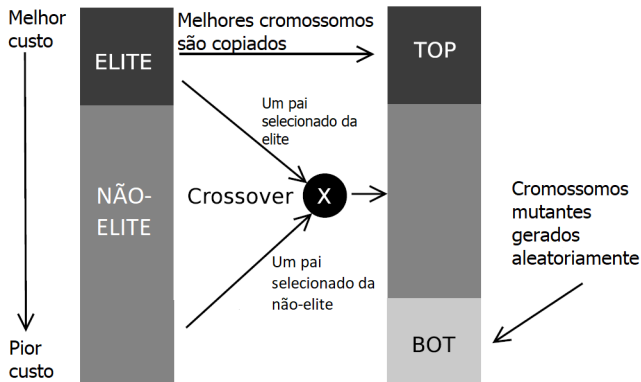
BRKGA - Framework

Há uma clara divisão entre as partes: depende e independente do problema



Fonte: Adaptado de Gonçalves e Resende (2011).

BRKGA - Dinâmica evolutiva



Fonte: Adaptado de Gonçalves e Resende (2011).

BRKGA - Crossover entre dois cromossomos

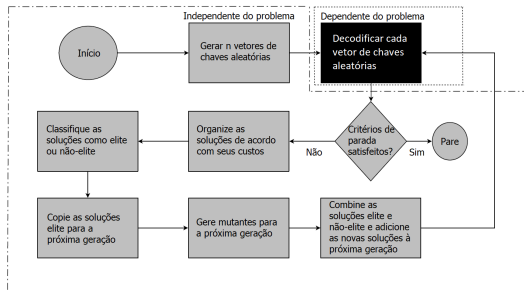


Fonte: Adaptado de Gonçalves e Resende (2011).

BRKGA

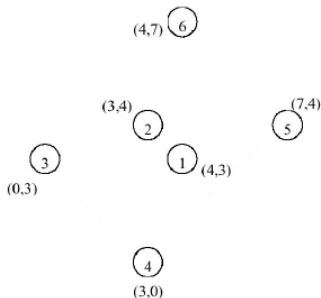
Para adaptar o BRKGA a qualquer problema de otimização é necessário alterar apenas a parte do algoritmo que é dependente do problema:

- cromossomo
- decodificador
- fitness



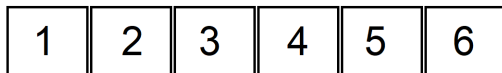
Toy problem 1

Considere um exemplo do problema do caixeiro viajante (TSP) com 6 cidades:



Cromossomo

Definimos que o vetor de chaves aleatórias do BRKGA representará a sequência de cidades:



$$n = 6$$

Decodificador

Decodificador seria, por exemplo, considerar

0.5	0.3	0.1	0.8	0.4	0.6
1	2	3	4	5	6

Decodificador

0.5	0.3	0.1	0.8	0.4	0.6
1	2	3	4	5	6

Ordenando os valores do cromossomo...

0.1	0.3	0.4	0.5	0.6	0.8
3	2	5	1	6	4

Cálculo da fitness

Cálculo da distância da sequência de cidades 3-2-5-1-6-4

$$\sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij} \quad (1)$$

ou seja,

$$d_{32} + d_{25} + d_{51} + d_{16} + d_{64} + d_{43} \quad (2)$$

Parâmetros

Por se tratar de uma metaheurística evolutiva, o BRKGA possui vários parâmetros que influenciam sua eficiência

Parâmetros	Valores recomendados
tamanho da população (p)	$p = an$, a constate $1 \leq a \in \mathbb{R}$ e n o tamanho do cromossomo
tamanho da pop. elite (p_e)	$0,10p \leq p_e \leq 0,25p$
tamanho da pop. mutantes (p_m)	$0,10p \leq p_m \leq 0,30p$
prob. de herança da chave elite (ρ_a)	$0,5 < \rho_a \leq 0,8$

Tabela: Parâmetros e valores recomendados por Gonçalves e Resende (2011).

Cr terios de parada

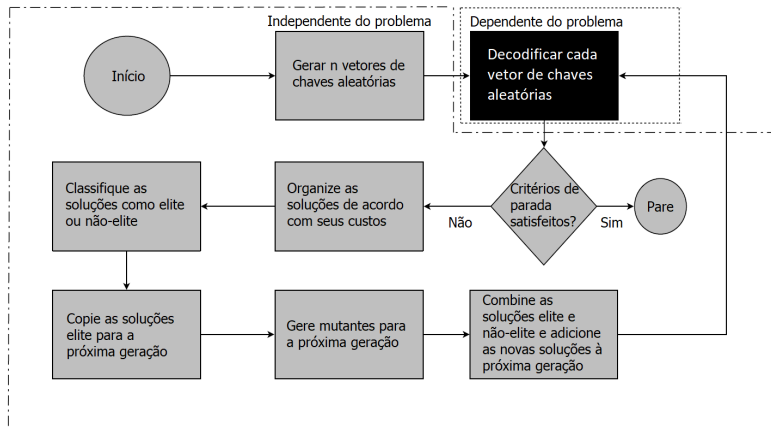
Exemplos:

- ▶ n mero fixo de gera  es
- ▶ n mero de gera  es sem evolu  o na solu  o
- ▶ tempo
- ▶ solu  o encontrada ser t o boa quanto um valor dado

brkgaAPI: A C++ application programming interface for biased random-key genetic algorithms

O brkgaAPI (Toso e Resende (2012)) manipula automaticamente a parte independente do problema de otimização, incluindo gerenciamento de população e dinâmica evolutiva, deixando ao usuário a tarefa de implementar um procedimento dependente, ou seja, converter um vetor de chaves aleatórias em uma solução para o problema.

<http://mauricio.resende.info/src/brkgaAPI/>



Fonte: Adaptado de Gonçalves e Resende (2011).

brkgaAPI

6 arquivos que compõem do brkgaAPI:

- ▶ BRKGA.h
 - ▶ initialize
 - ▶ evolution
 - ▶ exchangeElite
 - ▶ getBestChromosome, getBestFitness, getPopulation
 - ▶ tratamento de erros "Chromosome size equals zero."
- ▶ MTRand.h
 - ▶ gerador de números aleatórios

brkgaAPI

- ▶ Population.h
 - ▶ sortFitness, setFitness, getChromosome, getFitness...
 - ▶ tratamento de erros "Population size p cannot be zero."
- ▶ **samplecode.cpp**
 - ▶ main
- ▶ **SampleDecoder.cpp**
 - ▶ decode
- ▶ SampleDecoder.h

Na prática...

Calibrar o BRKGATSP

Entrada:

```
./main <intance> <configID> <instanceID> <seed>  
<multipPop> <pe> <pm> <rhoe> <MaxGeracao>
```

Exemplo:

```
./main C:\Users\Larissa\Documents\R\win-  
library\3.2\irace\tuning\BRKGA\instancia100.txt 1 1 0 2 0.3 0.1  
0.7 1000
```


Referências bibliográficas I

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, v. 6, n. 2, p. 154–160, 1994.

GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, v. 17, n. 5, p. 487 – 525, 2011.

TOSO, R. F.; RESENDE, M. G. *A C++ application programming interface for biased random-key genetic algorithms*. [S.l.], 2012.

Dúvidas?

Obrigada!

Larissa T. Oliveira: ltido@icmc.usp.br