PROD. TEC. ITV. MI- N0015/2021
DOI:10.29223/PROD.TEC.ITV.MI.2021.15.Beneteli

PRODUÇÃO TÉCNICA ITV MI

PROCEDIMENTOS PARA INTEGRAÇÃO IRACE-MATLAB: CALIBRANDO PARÂMETROS DE HEURÍSTICAS

Relatório parcial do projeto OptiPlant

Autores ITV:
Tatianna Aparecida Pereira Beneteli
Thiago Antônio Melo Euzébio
Luciano Perdigão Cota

Autores parceiros:
Robson Aparecido Duarte

Ouro Preto
Minas Gerais, Brasil

Agosto/2021

Título: Procedimentos para integração Irace-Matlab: calibrando parâmetros de heurísticas	
PROD. TEC. ITV MI – N0015/2021	Revisão
Classificação: ( ) Confidencial ( ) Restrita ( ) Uso Interno (X) Pública	01

Informações Confidenciais - Informações estratégicas para o Instituto e sua Mantenedora. Seu manuseio é restrito a usuários previamente autorizados pelo Gestor da Informação.

Informações Restritas - Informação cujo conhecimento, manuseio e controle de acesso devem estar limitados a um grupo restrito de empregados que necessitam utilizá-la para exercer suas atividades profissionais.

Informações de Uso Interno - São informações destinadas à utilização interna por empregados e prestadores de serviço

Informações Públicas - Informações que podem ser distribuídas ao público externo, o que, usualmente, é feito através dos canais corporativos apropriados

Dados Internacionais de Catalogação na Publicação (CIP)

B461p

Beneteli, Tatianna Aparecida Pereira
Procedimentos para integração Irace-Matlab: calibrando parâmetros de heurísticas. Tatianna Aparecida Pereira Beneteli...[et al.] - Ouro Preto, MG: ITV, 2021.

19 p.: il.

1. Calibração de Parâmetros. 2. Heurísticas. 3. Matlab. 4. Irace. I. Euzébio, Thiago Antônio. II. Cota, Luciano Perdigão. III. Duarte, Robson Aparecido. IV. Título.

CDD.23. ed. 005.1

Bibliotecária responsável: Nisa Gonçalves – CRB 2 - 525

RESUMO EXECUTIVO

Técnicas heurísticas são estratégias inspiradas em processos intuitivos que procuram boas soluções para problemas reais em um tempo computacional aceitável para a tomada de decisão. Estas técnicas possuem parâmetros de entrada que necessitam de calibração. Existe uma ferramenta chamada **irace**, disponibilizada na linguagem R, específica para a etapa de calibração. Neste relatório, apresenta-se os procedimentos para o uso da ferramenta **irace** na calibração de parâmetros de heurísticas implementadas no Matlab®.

RESUMO

Este relatório técnico descreve as etapas que devem ser realizadas para utilizar o pacote **irace** na calibração de parâmetros de heurísticas implementadas no software Matlab®. São apresentadas as sintaxes dos arquivos de configuração dos parâmetros a serem calibrados, assim como seus valores iniciais, as combinações proibidas, as instâncias e as configurações de cada cenário. A utilização do **irace** para a calibração de parâmetros, como apresentado nesse relatório, pode conduzir a um melhor desempenho das técnicas heurísticas, já que elimina a etapa de escolha empírica dos parâmetros utilizados.

Palavras-chave: Calibração de parâmetros. Heurísticas. Matlab. Irace.

ABSTRACT

This technical report describes the steps for using the **irace** package to calibrate heuristic parameters implemented in Matlab® software. The syntaxes of the configuration files of the parameters to be calibrated are presented, and their initial values, prohibited combinations, instances, and configurations for each scenario. The use of **irace** for parameter calibration, as presented in this report, can lead to a better performance of heuristic techniques, as it eliminates the step of empirically choosing the parameters used.

Keywords: Parameter calibrations. Heuristics. Matlab. Irace.

LISTA DE SÍMBOLOS E ABREVIAÇÕES

ILS Iterated Local Search

SA Simulated Annealing

GA Genetic Algorithm

DE Differential Evolution

SUMÁRIO

1 INTRODUÇÃO

Este relatório técnico tem como objetivo apresentar os procedimentos para o uso da ferramenta **irace** na calibração de parâmetros em heurísticas implementadas em Matlab®. São abordadas as configurações básicas da ferramenta, incluindo aquelas que permitem a troca de informações entre as plataformas.

O **irace** (LÓPEZ-IBÁÑEZ et al., 2020) é um pacote desenvolvido em linguagem de programação R (R CORE TEAM, 2020) que implementa um procedimento *Iterated Racing* para configuração automática de parâmetros. Nesse procedimento, segundo López-Ibáñez et al. (2016), novas configurações são obtidas seguindo uma distribuição específica e as melhores dessas configurações são selecionadas. Em seguida, a distribuição das amostras é atualizada, de forma a direcioná-las para a região das melhores configurações. O procedimento é então repetido até que um critério de parada seja atendido. Na seleção por corrida, as configurações são avaliadas por meio de um custo associado a cada uma delas. Um teste estatístico é realizado e, se houver evidência estatística suficiente para identificar configurações candidatas que têm desempenho pior que alguma outra, as configurações com pior desempenho são removidas da corrida.

O **irace** pode ser utilizado na calibração dos parâmetros de diversas técnicas heurísticas. Essas técnicas são inspiradas em processos intuitivos que procuram boas soluções em um tempo computacional aceitável para a tomada de decisão (COTA, 2018). Porém, não há garantia que a solução encontrada seja ótima. Alguns exemplos dessas técnicas são: Iterated Local Search (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003), Simulated Annealing (SA) (KIRKPATRICK; GELATT; VECCHI, 1983), Genetic Algorithm (GA) (GOLDBERG, 1989) e Differential Evolution (DE) (STORN; PRICE, 1997). O uso do **irace** está presente em trabalhos como: Benavides e Ritt (2015) para ILS; Kabova et al. (2017) para SA; Silva, Plastino e Freitas (2018) para GA; e Botelho (2016) para DE. Dentre os parâmetros que podem ser calibrados com o auxílio do **irace** podemos citar o tamanho da população, a taxa e a técnica utilizada para etapa de cruzamento e mutação, dentre outros.

O Matlab® é um software que, dentre diversas outras aplicações, se mostra adequado também para a implementação de técnicas heurísticas. Além de permitir desenvolver o algoritmo por meio da programação em linha de código, heurísticas como GA e SA estão presentes na *toolbox* de otimização, bastando apenas realizar a calibração dos parâmetros. Nessa etapa é que se faz necessária a integração do Matlab® com o **irace**. Como as duas linguagens possuem diferenças significativas de sintaxe e o desenvolvedor em Matlab® não necessariamente é familiarizado com a linguagem R, este relatório busca auxiliar nas etapas necessárias para a integração entre softwares.

O relatório técnico está organizado em mais dois capítulos. O Capítulo 2 traz os procedimentos de uso do **irace**, abordando a criação do ambiente e a configuração dos

componentes. O Capítulo 3 apresenta as considerações finais.

2 PROCEDIMENTOS DE USO

2.1 CRIAÇÃO DE AMBIENTE DE DESENVOLVIMENTO

As etapas necessárias para a criação do ambiente de desenvolvimento são apresentadas a seguir.

1. Crie uma pasta `~/tuning/` para armazenar todos os arquivos que serão criados nas etapas seguintes.

2. Dentro da nova pasta, adicione o *script* do Matlab® com o algoritmo, que será calibrado pelo **irace**, e todas as demais funções que se fizerem necessárias para a sua execução.

3. Ainda dentro da nova pasta, crie um arquivo de texto `parameters.txt` e determine a faixa dos parâmetros a ser considerada pelo **irace** durante a calibração.

4. *Etapa opcional*: Caso tenha conhecimento de valores iniciais para os parâmetros, pode-se defini-los dentro de um novo arquivo de texto `configurations.txt`. A utilização desse arquivo deverá ser indicada mais adiante no arquivo `scenario.txt`.

5. *Etapa opcional*: No caso de existirem combinações de parâmetros que devem ser desconsideradas pelo **irace**, um novo arquivo de texto `forbidden.txt` deve ser criado para indicar essas combinações. A utilização desse arquivo deverá ser indicada mais adiante no arquivo `scenario.txt`.

6. Por padrão, as instâncias a serem utilizadas pelo algoritmo devem ser armazenadas em arquivos individuais na pasta `~/tuning/Instances`. Caso não se utilize nenhuma instância, uma *string* vazia deve ser utilizada no arquivo `scenario.txt` para definir a não utilização do diretório de instâncias.

7. *Etapa opcional*: Pode-se criar um arquivo de texto `instances-list.txt` dentro da pasta `~/tuning/` para indicar quais instâncias da pasta devem ser utilizadas pelo **irace** e se existem parâmetros específicos para cada uma das instâncias. A utilização desse arquivo deverá ser indicada mais adiante no arquivo `scenario.txt`. Em caso da não utilização da pasta `~/tuning/Instances`, as instâncias podem ser definidas diretamente no arquivo `instances-list.txt`.

8. Crie um arquivo `scenario.txt` para definir diversas opções como o número máximo e/ou o tempo máximo de execuções do algoritmo. O uso de arquivos opcionais (`configurations.txt`, `forbidden.txt` e `instances-list.txt`) deverá ser indicado nesse arquivo. A integração com o Matlab® é realizada definindo no `targetRunner` quais informações serão enviadas e quais serão recebidas.

9. A cada cenário criado é recomendada a execução de um teste para verificar eles foram configurados corretamente. Softwares como o RStudio®, RGui ou outro para a linguagem R podem ser utilizados e devem ter os pacotes `irace` e `matlabr` instalados. Com diretório de trabalho definido em `~/tuning/`, o seguinte código em R deve ser executado:

```
1 library("irace")
2 scenario <- readScenario(filename = "scenario.txt", scenario =
    defaultScenario())
3 checkIraceScenario(scenario = scenario)
```

Algoritmo 2.1 – Verificar se os cenários estão configurados corretamente.

10. Por fim, novamente com o diretório de trabalho configurado como `~/tuning/`, o seguinte código em R deve ser executado para iniciar a calibração utilizando o **irace**:

```
1 library("irace")
2 scenario <- readScenario(filename = "scenario.txt", scenario =
    defaultScenario())
3 irace.main(scenario = scenario)
```

Algoritmo 2.2 – Executar o **irace**.

2.2 CONFIGURAÇÃO DE COMPONENTES

Nesta seção são detalhados os componentes do ambiente de desenvolvimento.

2.2.1 Parâmetros (`parameters.txt`)

O arquivo `parameters.txt` traz as informações dos parâmetros a serem calibrados. Cada parâmetro deve ser definido em uma linha respeitando o seguinte formato:

`<name> <label> <type> <range> | <condition>`

Onde:

- `<name>` é um identificador para o parâmetro, usado apenas dentro do **irace**.
- `<label>` é a forma utilizada para atribuir valores aos parâmetros no algoritmo a ser executado. No Matlab®, a estrutura de atribuição é composta pelo nome da variável seguida do sinal de igualdade. Este item deve estar entre aspas duplas.
- `<type>` é o tipo do parâmetro indicado como uma única letra. Utiliza-se '`i`' para o tipo inteiro, '`r`' para números reais, '`o`' para ordinais e '`c`' para categóricos.

- `<range>` descreve os possíveis valores para cada parâmetro, devendo estar entre parênteses. Para parâmetros do tipo números reais ou inteiros, deve-se indicar apenas o limite inferior e o superior da faixa a ser testada, por exemplo, uma faixa entre 0 e 1 é indicada por `(0,1)`. Para parâmetros ordinais e categóricos, os valores devem estar separados por vírgulas, por exemplo, `(0.6, 0.7, 0.8, 0.9)`.

- `<condition>` é opcional e indica a condição para que o parâmetro seja enviado ao algoritmo a ser executado. Por padrão, todas as variáveis são enviadas. Quando utilizadas as condições, são enviadas apenas as variáveis cujo o resultado da condição seja verdadeiro. As expressões devem ser escritas na linguagem R e utilizando os identificadores definidos em `<name>`.

Um exemplo de configuração para os parâmetros é apresentado a seguir. Destaca-se que é necessário deixar uma linha vazia após o último parâmetro. Observe que a condição para `param1` foi escrita na linguagem R utilizando o indicador `tipo` e não `param4`, que é o nome do parâmetro utilizado no *script* do Matlab®.

```
1 param1  "param1="  i  (1, 10)              | tipo %in% c("A", "C")
2 param2  "param2="  r  (0.1, 0.9)
3 param3  "nivel="   o  ("baixo", "alto")
4 tipo    "param4="  c  ("A", "B", "C")
5 param5  "param5="  c  (1, 2, 3.4, 5)       | param2 > 0.25 && param2 <= 0.35
6
```

Algoritmo 2.3 – Exemplo para o arquivo `parameters.txt`.

2.2.2 Configurações iniciais (`configurations.txt`)

O arquivo `configurations.txt` é opcional e contém valores iniciais para os parâmetros. A primeira linha deve indicar o nome do parâmetro correspondente a cada coluna. Os nomes devem ser idênticos aos identificadores dos parâmetros `<name>` definidos no arquivo `parameters.txt`. As demais linhas correspondem a cada configuração e indicam os valores dos parâmetros. Caso um parâmetros não seja utilizado em determinada configuração, utiliza-se o termo `NA`. Deve-se deixar uma linha vazia após a última configuração. Um exemplo para o arquivo `configurations.txt` é apresentado a seguir.

```
1 param1  param2  param3  tipo    param5
2 1       0.3     "baixo" "C"     2
3 3       0.7     "alto"  "A"     NA
4
```

Algoritmo 2.4 – Exemplo para o arquivo `configurations.txt`.

2.2.3 Combinações proibidas (`forbidden.txt`)

O arquivo `forbidden.txt` é opcional e deve ser usado caso existam configurações que possam gerar falhas no algoritmo (por exemplo, consumo excessivo de processamento e/ou memória) ou quando é sabido que alguma configuração não irá gerar resultados satisfatórios. Cada restrição deve ser declarada em uma linha através de uma expressão lógica escrita na linguagem R. Os nomes dos parâmetros devem ser idênticos aos identificadores `<name>` definidos no arquivo `parameters.txt`. Se uma configuração faz a expressão lógica ser verdadeira, então essa configuração é descartada. Como exemplos de alguns operadores lógicos aceitos temos `==` `!=` `>=` `<=` `>` `<` `|` `&` `!` `%in%`. Outros operadores podem ser encontrados em:

`<https://stat.ethz.ch/R-manual/R-devel/library/base/html/Syntax.html>`

Um exemplo do arquivo `forbidden.txt` é apresentado a seguir.

```
1 param1 < 5 & param3 == "baixo"
2 param2 > 0.8 & tipo %in% c("A", "B")
3
```

Algoritmo 2.5 – Exemplo para o arquivo `forbidden.txt`.

2.2.4 Arquivos de instâncias (`~/tuning/Instances`)

As instâncias podem ser utilizadas para a calibração de duas maneiras distintas. A primeira forma é adicionar os arquivos com dados das instâncias na pasta `~/tuning/Instances` e acrescentar no `targetRunner` a função do Matlab® para carregar arquivos. A segunda maneira é criar um arquivo de texto vazio para cada instância que se deseja utilizar. Neste caso, o `targetRunner` é configurado para passar para ao Matlab® apenas o identificador da instância, isto é, um número inteiro entre 1 e o número total de arquivos criados. O *script* do Matlab® deve então ser configurado de forma a utilizar esse número identificador para selecionar quais dados devem ser carregados no *workspace*.

2.2.5 Lista de instâncias (`instances-list.txt`)

O arquivo `instances-list.txt` é opcional. Cada instância deve ser declarada em uma linha e o arquivo pode ser utilizado de duas formas distintas. Na primeira forma, cada linha deve conter o nome do arquivo armazenado na pasta `~/tuning/Instances`, seguido de um espaço em branco e da definição de parâmetros específicos para essa instância, caso se aplique. O formato da definição desses parâmetros específicos deve seguir a estrutura utilizada pelo Matlab®, isto é, o nome do parâmetro seguido do sinal de igualdade e do valor. Na segunda forma, a pasta `~/tuning/Instances` não é utilizada e as instâncias são definidas diretamente no arquivo `instances-list.txt`. Cada linha deve ser escrita de

forma que o Matlab® compreenda, seja chamando uma função ou atribuindo diretamente valores para as variáveis. Pode ser necessário adicionar caracteres para adequar a linguagem ao Matlab®, dependendo de como a instância é passada para o algoritmo teste no arquivo `scenario.txt`.

Um exemplo das diferentes formas de se utilizar o arquivo `instances-list.txt` é apresentado a seguir. Na Linha 1 observa-se um exemplo da primeira forma sem parâmetros específicos. Na Linha 2 a mesma forma é utilizada, mas com um parâmetro específico `k`. Já na Linha 3 um exemplo da segunda forma é apresentado. Note que esse é um exemplo ilustrativo e que o arquivo utilizado deve ser escrito utilizando apenas uma das formas.

```
1 instancia_1.mat
2 instancia_2.mat ';k=[4 5 6 7]    # Definindo parâmetros adicionais
3 [4 5 6 7]                        # Sem utilizar pasta de instâncias
4
```

Algoritmo 2.6 – Exemplo para o arquivo `instances-list.txt`.

2.2.6 Definições de cenário (`scenario.txt`)

No arquivo `scenario.txt` são definidas as configurações utilizadas pelo **irace**. Cada configuração deve ser definida em uma linha e a lista completa de possíveis configurações está disponível em López-Ibáñez et al. (2020). Alguns exemplos são apresentados a seguir.

- `maxExperiments = 1000`
  Define como 1000 o número máximo de vezes que o algoritmo avaliado pode ser executado.
- `maxTime = 100`
  Define como 100 segundos o tempo máximo de cada execução do algoritmo avaliado. Para utilizar essa opção, o `targetRunner` deve retornar o tempo gasto.
- `configurationsFile="configurations.txt"`
  Habilita o uso do arquivo de configurações iniciais.
- `forbiddenFile = "forbidden.txt"`
  Habilita o uso do arquivo de configurações proibidas.
- `trainInstancesFile = "instances-list.txt"`
  Habilita o uso do arquivo com a lista das instâncias.
- `trainInstancesDir = ""`
  Desabilita o uso da pasta de instâncias.

- `parallel = 5`
  Habilita cinco execuções paralelas do `targetRunner`.

Também no arquivo `scenario.txt` é definido o `targetRunner` como uma função na linguagem R que faz a integração com o Matlab®. É requerido que o pacote `matlabr` esteja instalado. O *script* principal do Matlab® deve estar na mesma pasta do arquivo `scenario.txt`.

A seguir, apresenta-se um exemplo do arquivo `scenario.txt` escrito em linguagem R. Neste exemplo, o método heurístico que será calibrado é implementado no Matlab® em um *script* chamado de `MAIN`. Este método recebe como parâmetros a instância (`instance`), uma semente aleatória (`seed`) e cinco parâmetros que serão calibrados (`param1`, `param2`, `nivel`, `param4` e `param5`).

```
maxExperiments = 1000
trainInstancesFile = "instances-list.txt"

targetRunner <- function(experiment, scenario)
{
  main.script <- "MAIN(instance,seed,param1,param2,nivel,param4,param5)"

  require(matlabr)
  debugLevel       <- scenario$debugLevel
  configuration.id <- experiment$id.configuration
  instance.id      <- experiment$id.instance
  seed             <- experiment$seed
  configuration    <- experiment$configuration
  instance         <- experiment$instance
  switches         <- experiment$switches

  args <- irace::buildCommandLine(configuration, switches)
  args <- strsplit(args, split=" ")[[1]]

  matlabr::run_matlab_code(c(args, paste0("seed = ", seed, ";"), paste0("
      instance = '", instance, "';"), paste0("run('", main.script, "')")))

  cost <- as.numeric(readLines("Custo.txt"))
  time <- as.numeric(readLines("Tempo.txt"))

  return(list(cost = cost, time = time))
}

```

Algoritmo 2.7 – Exemplo para o arquivo `scenario.txt`.

A Linha 1 do código apresentado define o número máximo de execuções do algoritmo. A Linha 2 indica o uso do arquivo com a lista de instâncias. O `targetRunner` é definido na Linha 4 como uma função que recebe uma lista que contém informações sobre as configuração e as instância para execução de um experimento (argumento `experiment`) e uma lista de cenários (argumento `scenario`). Na Linha 6, a função `MAIN`, criada no Matlab® para execução da técnica heurística, é indicada com seus parâmetros que utilizam as mesmas *labels* definidas no arquivo `parameters.txt`. A Linha 8 carrega o pacote `matlabr`. A Linha 9 atribui à variável `debugLevel` um valor associado ao nível de informação a ser apresentado na saída de texto do **irace**. Nas Linhas 10 a 15 são atribuídas às variáveis, respectivamente, uma *string* alfanumérica que identifica uma configuração, uma *string* alfanumérica que identifica uma instância, a semente utilizada, a configuração (com uma coluna por nome de parâmetro), a instância para a avaliação e um vetor de *labels*. As atribuições ocorrem para que esses parâmetros possam ser enviados ao Matlab®, se necessário. A Linha 17 utiliza um comando do pacote **irace** que recebe os valores para cada parâmetro na configuração atual e as respectivas *labels* e retorna uma *string* (variável `args`) que pode ser utilizada como uma linha de comando para ser utilizada na heurística a ser calibrada. Na Linha 18, a *string* é subdividida nos locais em que há um espaço vazio. O comando da Linha 20 envia para o Matlab®, além da *string* subdividida (`args`), a semente, o caminho da instância utilizada e o comando `run`, utilizado pelo Matlab® para executar uma função. No caso, a função a ser executada é o *script* indicado em `main.script`. O *script* deve armazenar em arquivos de texto após uma execução o custo e, opcionalmente, o tempo gasto. No exemplo, o custo é armazenado em `Custo.txt` e o tempo em `Tempo.txt`, na mesma pasta do arquivo `scenario.txt`. Os arquivos são lidos e armazenados nas variáveis `cost` (Linha 22) e `time` (Linha 23).

Possíveis erros na integração podem ser identificados ao se analisar o *script* temporário que é criado pelo pacote **matlabr**. O caminho desse arquivo geralmente é exibido no console do software R ao iniciar uma execução (não paralela) do **irace** e pode ser semelhante a: `C:\Users\USUARIO\AppData\Local\Temp\Rtmpi6sjD7\file37c47bed6cce.m`. Por ser um arquivo temporário, os nomes da pasta e do próprio arquivo variam constantemente.

O trecho `paste0("instance = '", instance, "';")` presente na Linha 20 do Algoritmo 2.7 é escrito para enviar o diretório do arquivo de instância e é utilizado nos casos em que a lista de instâncias é escrita no formato da Linha 1 do Algoritmo 2.6. Caso a lista de instâncias seja escrita no formato da Linha 2 do Algoritmo 2.6, o trecho citado deve ser substituído por `paste0("instance = '", instance)`. Para o caso em que usa-se o formato da Linha 3 do Algoritmo 2.6, a substituição é por `paste0("instance = ", instance)`.

Em caso de execuções paralelas, é necessário que os arquivos que armazenam os resultados e tempos tenham nomes diferentes para cada execução. Uma sugestão é definir parte do nome dos arquivos como um `paste0` acrescentado nos argumentos da função `c` que

está dentro da função `run_matlab_code` da Linha 20 do Algoritmo 2.7. O trecho acrescentado deve ser `paste0("arquivo_id = '",configuration.id,instance.id,seed,"'")` e `arquivo_id` deve ser utilizado no algoritmo do Matlab® para dar nome aos arquivos que armazenarão os dados. Além disso, é recomendado que os arquivos sejam armazenados em uma pasta específica, `~/tuning/Resultados`, por exemplo. Nesse caso, a leitura dos arquivos é feita substituindo as Linhas 22 e 23 do Algoritmo 2.7, respectivamente, por:

```
cost <- as.numeric(readLines(paste0("Resultados/Result",
          configuration.id,instance.id,seed,".txt")))
time <- as.numeric(readLines(paste0("Resultados/Tempo",
          configuration.id,instance.id,seed,".txt")))
```

Ainda sobre execuções paralelas, caso seja necessário interromper a execução da calibração e esteja sendo utilizado o software RGui, deve-se utilizar a opção "Parar todas as computações", do menu "Misc". Além disso, para evitar erros com relação à pasta da *toolbox*, recomenda-se fazer a seguinte alteração no Matlab®: na guia "HOME", clique na opção "PREFERENCE" da seção "ENVIRONMENT". Na janela que se abre, escolha a opção "General" na guia lateral e desmarque a opção "Enable toolbox path cache".

2.2.7 Exemplo de heurística (`MAIN.m`)

O algoritmo a ser testado pode ser escrito como um *script* ou função do Matlab®, como apresentado no Algoritmo 2.8. Opcionalmente, ele pode utilizar internamente a semente para a geração de seus valores aleatórios e retornar o tempo gasto pela técnica heurística.

```
1  function Result = MAIN(instance,seed,param1,param2,nivel,param4,param5)
2  %   MAIN Summary of this function goes here
3  %   Detailed explanation goes here
4
5      rng(seed,'twister');                % Utilizando a semente do irace
6      tic;                                % Iniciando a contagem de tempo
7      load(instance);                     % Carregando o arquivo de instância
8
9      % Início da estratégia
10     ...
11     ...
12     % Fim da estratégia
13
14     writematrix(Result,['Custo.txt'])   % Criando o arquivo para o custo
15     tempo = toc;                        % Encerrando a contagem de tempo
16     writematrix(tempo,['Tempo.txt'])    % Criando o arquivo para o tempo
17 end
```

Algoritmo 2.8 – Exemplo para o arquivo `MAIN.m`.

No caso de execuções paralelas, como citado anteriormente, a variável `arquivo_id` deve ser utilizada para nomear os arquivos. Dessa forma as Linhas 14 e 16 do Algoritmo 2.8 são substituídas, respectivamente, por:

```
writematrix(Result,['Resultados/Result' arquivo_id '.txt'])
```

```
writematrix(Result,['Resultados/Tempo' arquivo_id '.txt'])
```

3 CONSIDERAÇÕES FINAIS

Este relatório tem com objetivo descrever e exemplificar a integração entre o **irace** e o Matlab® para a calibração de parâmetros de heurísticas. Nesta abordagem, considera-se que as heurísticas estão implementadas no Matlab® e a ferramenta **irace** na linguagem R. O relatório contém informações básicas a respeito da integração das ferramentas. Para itens não abordados neste documento, recomenda-se utilizar o Guia de Usuário do **irace**, disponível em:
<https://cran.r-project.org/web/packages/irace/vignettes/irace-package.pdf>.

REFERÊNCIAS

BENAVIDES, A. J.; RITT, M. Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops. In: **Twenty-Fifth International Conference on Automated Planning and Scheduling**. Jerusalem, Israel: [s.n.], 2015. p. 34–41. Citado na página 6.

BOTELHO, M. d. M. **Análise experimental de operadores de recombinação para o algorítimo de evolução diferencial**. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, UFMG, 2016. Citado na página 6.

COTA, L. P. **Abordagens exatas e heurísticas para o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência**. Tese (Doutorado) — Programa de Pós-Graduação em Engenharia Elétrica, UFMG, 2018. Citado na página 6.

GOLDBERG, D. **Genetic Algorithms in Search, Optimization, and Machine Learning**. [S.l.]: Addison-Wesley Publishing Company, 1989. (Artificial Intelligence). Citado na página 6.

KABOVA, E. A.; COLE, J. C.; KORB, O.; LÓPEZ-IBÁÑEZ, M.; WILLIAMS, A. C.; SHANKLAND, K. Improved performance of crystal structure solution from powder diffraction data through parameter tuning of a simulated annealing algorithm. **Journal of Applied Crystallography**, International Union of Crystallography, v. 50, n. 5, p. 1411–1420, 2017. Citado na página 6.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. Citado na página 6.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; STÜTZLE, T.; BIRATTARI, M. The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, v. 3, p. 43 – 58, 2016. Citado na página 6.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; STÜTZLE, T.; BIRATTARI, M. Irace: Iterated racing for automatic algorithm configuration. 2020. R package version 3.4.1. Disponível em: <https://CRAN.R-project.org/package=irace>. Citado 2 vezes nas páginas 6 e 12.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: **Handbook of metaheuristics**. [S.l.]: Springer, 2003. p. 320–353. Citado na página 6.

R CORE TEAM. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2020. Disponível em: <https://www.R-project.org/>. Citado na página 6.

SILVA, P. N. d.; PLASTINO, A.; FREITAS, A. A. A novel genetic algorithm for feature selection in hierarchical feature spaces. In: SIAM. **Proceedings of the 2018 SIAM International Conference on Data Mining**. San Diego, USA, 2018. p. 738–746. Citado na página 6.

STORN, R.; PRICE, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. **Journal of global optimization**, Springer, v. 11, n. 4, p. 341–359, 1997. Citado na página 6.