UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uso da Metaheurística Biased Random Key Genetic Algorithm e Configuração Automática de Parâmetros para o Problema da Próxima Versão de Software

André de Souza Andrade

Orientador

Dr. Márcio de Oliveira Barros

Co-orientadora

Dra. Adriana Césario de Faria Alvim

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2018

Uso da Metaheurística Biased Random Key Genetic Algorithm e Configuração Automática de Parâmetros para o Problema da Próxima Versão de Software

ANDRÉ DE SOUZA ANDRADE

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Dr. Márcio de Oliveira Barros — UNIRIO

Dra. Adriana Cesário de Faria Alvim — UNIRIO

Dra. Ana Cristina Bicharra Garcia — UNIRIO

Dra. Isabel Cristina Mello Rosseti — UFF

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2018.

Catalogação informatizada pelo(a) autor(a)

A553

Andrade, André de Souza
Uso da Metaheurística Biased Random Key Genetic Algorithm e Configuração Automática de Parâmetros para o Problema da Próxima Versão de Software / André de Souza Andrade. -- Rio de Janeiro, 2018.
101 f.

Orientador: Márcio de Oliveira Barros.
Coorientadora: Adriana Cesário de Faria Alvim.
Dissertação (Mestrado) - Universidade Federal do Estado do Rio de Janeiro, Programa de Pós-Graduação em Informática, 2018.

1. Engenharia de Software Baseada em Busca. 2. Next Release Problem. 3. Biased Random Key Genetic Algorithm. 4. Configuração Automática de Parâmetros. 5. K-Fold Cross Validation. I. Barros, Márcio de Oliveira, orient. II. Alvim, Adriana Cesário de Faria, coorient. III. Título.

À minha amada mãe Sônia e meu pai José, que me deram a vida. À dra. Maria Thereza Telles Uchôa, psicóloga, que ma devolveu.

Agradecimentos

Agradeço ao Darma (ensinamentos) de Buda, um ser humano como nós, que não satisfeito com o conforto e regalias de sua casa, dispôs-se a sacrificar a própria vida para descobrir as reais causas do sofrimento e as formas de cessá-lo, e ensiná-las a todos.

Agradeço aos professores e professoras do passado, que me proporcionaram nutrir sentimentos bons a tudo que tenha relação com o ensino e o aprendizado, como minha admiração pelo papel que desempenhavam, meu respeito e carinho pelas pessoas que eram, minha reverência ao espaço escolar e o valor que sempre dei ao conhecimento.

Faltam-me palavras para agradecer a atitude paciente e compassiva dos meus orientadores Márcio Barros e Adriana Alvim. Cada crítica, cada sondagem desconcertante, cada elogio, cada voto de confiança foram senão expressões de guias amorosos espraiando luz pela senda da vida e do saber acadêmico e realizando a sua preciosa obra nessa existência.

Agradeço às minhas filhas Alice e Júlia, que perceberam o quão indisponível fiquei em nossos momentos de convívio e que de certa forma compreenderam, muito pela idade que têm, mas em especial por serem pessoas maravilhosas. Elas sabem que o papai está pavimentando caminhos nessa estrada da vida, para que possam seguir em segurança. Agradeço também aos meus irmãos Adriana, Anderson e Aline, os maiores presentes de meus pais, que mesmo fisicamente distantes desejaram o melhor nessa minha jornada.

Agradeço aos professores do PPGI-UNIRIO pelo empenho em fazerem desse programa de pós graduação um espaço vibrante e especial de acolhimento e motivação para o estudo e a pesquisa. Agradeço também aos amigos do mestrado que se uniram no grupo de WhatsApp "Chama o Le Boudec"com o propósito de apoio mútuo em uma das disciplinas no primeiro período, mas que se manteve junto ao longo de todo o curso.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Andrade, André de Souza. **Uso da Metaheurística Biased Random Key Genetic Algorithm e Configuração Automática de Parâmetros para o Problema da Próxima Versão de Software**. UNIRIO, 2018. 101 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

O planejamento de versões (*releases*) de software a fim de maximizar a satisfação do cliente, considerando as restrições de recursos para a implementação dos novos requisitos, é determinante no sucesso ou fracasso do software – para a sua aceitação, decidir em qual momento qual *feature* será entregue é tão relevante quanto levantar e identificar corretamente os requisitos. Tal problema de otimização combinatória da área de Engenharia de Requisitos é conhecido como o Problema da Próxima Versão do Software (*Next Release Problem - NRP*) e possui as versões mono e multi-objetivo.

Este trabalho utilizou a metaheurística *Biased Random Key Genetic Algorithm* para resolver o problema em sua versão mono-objetivo, com o suporte da ferramenta IRACE para a configuração automática de parâmetros e da técnica *K-Fold Cross Validation* para o melhor aproveitamento das instâncias disponíveis quando usadas como instâncias de treinamento. A contribuição deste trabalho está na nova abordagem para o problema, tanto pela metaheurística empregada como pelo método usado para a escolha dos parâmetros da metaheurística, substituindo a experiência humana pelo método científico.

Ao executar o algoritmo proposto, consistindo de um método construtivo associado à metaheurística, sobre dois conjuntos de instâncias amplamente utilizados na literatura, foi observado uma distância média para os ótimos das instâncias de 1,6%. A melhor heurística conhecida para o NRP, o BMA (*Backbone-Based Multilevel Algorithm*), obteve uma distância média de 4,1%, sobre o que se conclui que este trabalho proporcionou uma melhoria de 63% em relação ao melhor resultado encontrado na literatura.

Palavras-chave: Engenharia de Software Baseada em Busca; Next Release Problem; Biased Random Key Genetic Algorithm; Configuração Automática de Parâmetros; K-Fold Cross Validation.

ABSTRACT

Software release planning in order to maximize customer satisfaction, regarding resource constraints for the development of new features, is determinant if a software will succeed or not – for its acceptance, the decision about which release a requirement or feature will be delivered in is as relevant as the right elicitation and identification of the requirements. Such a combinatorial optimization problem of Requirements Engineering area is known as Next Release Problem, with its mono and multi objective versions.

This work used the Biased Random Key Genetic Algorithm metaheuristic to solve the problem's mono objective version with IRACE and K-Fold Cross Validation as an automatic parameter tuning tool and as a technique to yield better use of provided instances when used as training instances, respectively. The work contribution lays on a new approach to the problem, both by the metaheuristic and by the method for choosing the metaheuristic parameters, replacing the human expertise by the scientific method.

After run the proposed algorithm, a constructive method combined with the BRKGA metaheuristic, using two commonly used large scale instance sets, it was observed a average distance from the optimal solutions of 1,6%. The best known heuristic for the NRP, namely BMA (*Backbone-Based Multilevel Algorithm*), got 4,1% for the average distance from the optimal solutions. So, it implies that this work yielded an improvement of 63% if compared with the best results in previous works.

Keywords: Search-based Software Engineering; Next Release Problem; Biased Random Key Genetic Algorithm; Automatic Parameter Tuning; K-Fold Cross Validation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

BMA	Backbone-based Multilevel Algorithm
BRKGA	Biased Random Key Genetic Algorithm
GA	Genetic Algorithm
NRP	Next Release Problem
RP	Release Planning
SBSE	Search-Based Software Engineering

1. Introdução

1.1 Motivação

Na perspectiva da Engenharia de Requisitos, existem dois tipos de desenvolvimento de software: desenvolvimento de software tradicional sob medida (em inglês *bespoke development*) e desenvolvimento de software orientado ao mercado (em inglês *market-driven development*) [1]. O primeiro tipo possui como principal objetivo a ser cumprido a conformidade com a especificação de requisitos, usada como um contrato entre o desenvolvedor do software e o cliente, tendo como medida de sucesso a aceitação do produto, cujos requisitos foram elicitados, analisados e validados e os usuários são conhecidos e identificados. Já o segundo tipo visa lançar o software o mais rápido possível no mercado, ampliando ou assegurando a sua participação nesse mercado cujos consumidores são usuários desconhecidos ou vagamente conhecidos, dos quais não se sabe ou tem-se uma vaga noção de suas necessidades traduzidas em requisitos. Nesse tipo, especial atenção é dada ao ciclo de vida do software, com suas versões lançadas de forma iterativa e incremental e cujas melhorias são baseadas nas experiências de uso das versões anteriores.

Um cenário típico de desenvolvimento de software orientado ao mercado e atualmente tão em voga é o dos aplicativos para celulares (ou Apps), onde não existe um cliente em particular, porém muitos, uns mais, outros menos relevantes, cujos requisitos podem variar em complexidade, quantidade e definição, a ponto de existir conflitos de interesse. Nesse cenário, o planejamento de versões possui uma importância especial e requer um tratamento computacional adequado à medida que a quantidade de clientes e de requisitos crescem em escala global. Aliado ao gerenciamento desses requisitos, o planejamento de versões de um software desempenha um papel crucial no sucesso do software. Além disso, Carlshamre et al. [2] indicam que há uma tendência crescente na realização de projetos com desenvolvimento de software orientado ao mercado comparativamente a projetos de desenvolvimento de software tradicional sob medida.

O planejamento de versões de software a fim de maximizar a satisfação do cliente, considerando as restrições de recursos para a implementação dos novos requisitos, é determinante para o sucesso do software – decidir em que momento cada requisito será implementado é tão relevante quanto levantar e identificar corretamente os requisitos. Tal planejamento foi modelado como um problema de otimização combinatória da área de Engenharia de Requisitos, cujo nome é *Next Release Problem* (NRP) ou o Problema da Próxima Versão, um problema NP-Difícil [3] cuja solução exata para instâncias muito grandes consome um tempo de processamento inaceitável.

1.2 Definição do Problema

Em Engenharia de Software, a atividade de seleção e priorização de requisitos é descrita como o problema de descobrir o melhor subconjunto de requisitos para a próxima versão de um software, dado um conjunto de requisitos que não podem ser implementados por completo nesta versão [4]. Por melhor subconjunto, entende-se o conjunto de requisitos que, se implementado, gerará o maior lucro possível para os desenvolvedores de software, desde que respeitada a restrição de orçamento para a sua implementação. Tal restrição refere-se ao custo de cada requisito implementado, cuja soma não deve ultrapassar um orçamento previamente definido.

Nesse contexto, tem-se como entrada do problema os clientes e suas respectivas listas de requisitos desejados, onde cada cliente está associado a uma expectativa de lucro caso a sua lista de requisitos seja implementada na próxima versão do software. O lucro esperado de cada cliente é independente da sua lista de requisitos e dos demais clientes. Cada requisito possui um custo de implementação e pode ter dependências em relação a outros requisitos, de forma que, para ser implementado, seus precedentes também devem ser implementados. Diferentes clientes podem desejar um ou mais requisitos em comum, sendo possível implementar requisitos que agradam a ambos assumindo apenas um custo de implementação do referido requisito.

Com base nas premissas acima, é possível definir o NRP como o problema que tem como objetivo determinar um subconjunto de clientes cujos requisitos serão implementados na próxima versão do software de forma a obter o máximo lucro, respeitando a restrição de orçamento e considerando que a cada cliente está associado um subconjunto de requisitos que precisa ser totalmente implementado para que o cliente fique satisfeito. Portanto, o NRP consiste em determinar um conjunto de requisitos que forneça a máxima satisfação dos clientes envolvidos, desde que um orçamento seja respeitado.

Uma definição formal para o NRP pode ser enunciada conforme segue [5]. Seja R o conjunto de m requisitos candidatos à próxima versão do software, onde cada requisito $j \in R$ tem um custo não negativo c_j de implementação. Seja S o conjunto de n clientes interessados na próxima versão do software e seja R_i o conjunto de requisitos de interesse do cliente i, onde $R_i \subseteq R, \forall i \in S$. Se todos os requisitos desejados pelo cliente i forem implementados na próxima versão, o cliente gerará um lucro ω_i para a empresa de desenvolvimento de software.

As dependências entre requisitos são modeladas como um grafo acíclico direcionado $G = (V, E)$, onde o conjunto de vértices V é igual ao conjunto de requisitos R e o conjunto de arestas E expressa a dependência entre dois requisitos de tal forma que se $(r', r) \in E$, então o requisito r depende do requisito r', ou seja, se r for escolhido para compor a próxima versão, r' também deverá ser escolhido de modo a satisfazer a dependência. Seja ancestrais(r) o conjunto dos requisitos que devem ser implementados para garantir a implementação de r. Estes requisitos são representados por todos os vértices para os quais existe um caminho em G que chega em r, incluindo o próprio vértice r. A definição recursiva de ancestrais(r) é:

(1) Se $u = r$, então $u \in$ ancestrais(r);

(2) Se $(u, v) \in E$ e $v \in$ ancestrais(r), então $u \in$ ancestrais(r);

(3) Se $u \in$ ancestrais(r), então u foi introduzido pelas regras 1 ou 2.

Por analogia, os requisitos que devem ser implementados para garantir a implementação de R_i são representados por ancestrais$(R_i) = \bigcup_{r \in R_i}$ ancestrais(r). Para qualquer subconjunto de requisitos $R' \subseteq R$, seu custo de implementação é custo$(R') = \sum_{j \in R'} c_j$. Portanto, um cliente $i \in S$ é atendido quando todos os requisitos em ancestrais(R_i) são implementados, o que gera um custo igual a custo(ancestrais$(R_i)) = \sum_{j \in \text{ancestrais}(R_i)} c_j$ e um lucro igual a ω_i. Dado que o orçamento para a próxima versão do software é limitado por uma constante positiva b, a solução para o NRP consiste em atender um subconjunto de clientes $S' \subseteq S$ tal que o lucro seja maximizado, respeitando a restrição de orçamento. Desta forma, uma solução S' pode ser representada como um vetor $X = (x_1, x_2, x_3, \ldots, x_n)$ de n posições, onde $x_i = 1$ indica que o cliente i é atendido e $x_i = 0$ indica o contrário. A solução S' deve maximizar o lucro ω expresso por $\sum_{i=1}^{n} \omega_i \times x_i$, sujeito à restrição de orçamento dada por custo$(\bigcup_{i=1}^{n} \{\text{ancestrais}(R_i) : x_i = 1\}) \leq b$.

1.2.1 Um Exemplo do Problema

Seja uma instância do NRP definida por três clientes e oito requisitos. Seja o vetor $\omega = (30, 25, 50)$, referente aos lucros esperados dos clientes, caso sua lista de requisitos seja implementada. Seja o vetor $c = (6, 10, 16, 4, 1, 7, 6, 1)$, referente ao custo de cada requisito. Seja a restrição de orçamento b igual a 36. Seja o grafo de dependência entre requisitos e a relação entre requisitos e clientes conforme representados pela Figura 1.1. Com isto, é possível definir os requisitos que devem ser implementados para satisfazer cada cliente e seus custos: $R_1 = \{r_1, r_3, r_4\}$, com custo 26, $R_2 = \{r_1, r_2, r_4, r_5, r_6, r_7, r_8\}$, com custo 35, e $R_3 = \{r_2, r_6, r_8\}$, com custo 18.

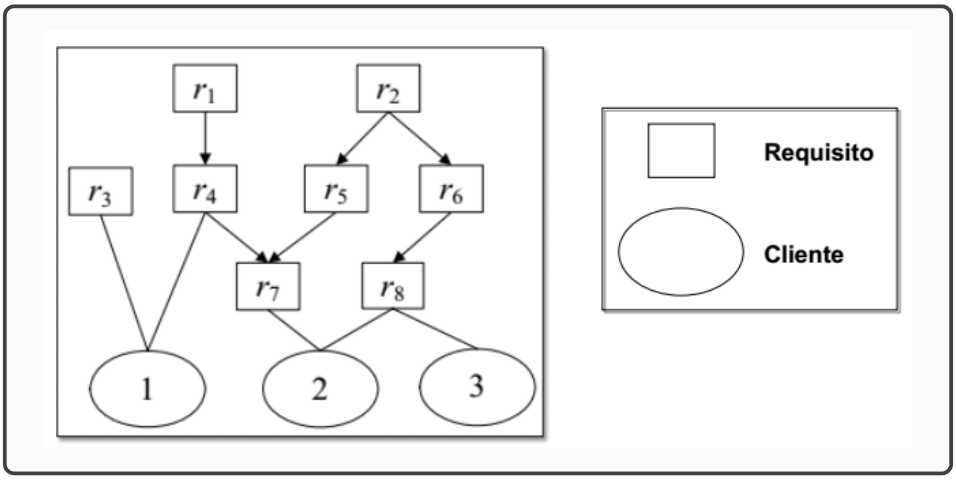

Figura 1.1: Dependência entre os requisitos e pertinência entre requisitos e clientes.

O espaço de soluções do NRP é o conjunto de todos os subconjuntos de clientes, ou seja, 2^n, onde n é o número de clientes. Portanto, o espaço de soluções é uma função exponencial de n, o que significa que quando forem muitos os clientes, o número de soluções possíveis será muito grande. Isso posto, é fácil concluir que o problema não pode ser tratado por uma análise completa de todas as soluções, como em uma típica busca exaustiva, se n for muito grande. Apenas para ilustrar, no caso de apenas 50 clientes seriam 1,126 x 10^{15} soluções, ou seja, da ordem de grandeza de quatrilhões.

Para o exemplo da Figura 1.1, têm-se oito soluções candidatas. No entanto, devido ao orçamento, nem todas as soluções são viáveis: soluções que atendam aos clientes 1 e 2 ou aos clientes 1 e 3 não são viáveis, pois a primeira tem um custo 51 e a segunda 44, ambas excedendo o orçamento de 36. As soluções viáveis são em número de cinco: cliente 1, com lucro 30 e custo 26; cliente 2, com lucro 25 e custo 35; cliente 3, com lucro 50 e custo 18; clientes 2 e 3, com lucro 75 e custo 35; nenhum cliente, com lucro 0 e custo 0. Por ter apresentado o maior lucro (o valor de 75), a melhor solução é representada pelo subconjunto cujos elementos são os clientes 2 e 3.

1.3 Objetivos

A literatura apresenta diversas propostas para a resolução do problema NRP, incluindo buscas locais, metaheurísticas populacionais (algoritmos genéticos e otimização de colônia de formigas), busca local iterada e técnicas de programação linear [6] [7] [8] [9] [10] [11] [12] [13].

Técnicas ou métodos heurísticos, ou simplesmente heurísticas, são procedimentos algorítmicos adaptados aos problemas de otimização combinatória, capazes de apresentar soluções de boa qualidade em tempo compatível com a necessidade de cada problema. As heurísticas podem ser divididas em três categorias que diferem basicamente na forma como exploram o espaço de soluções dos problemas: métodos construtivos, busca local ou em vizinhança e metaheurísticas, que são orientações gerais de como explorar o espaço de soluções e flexíveis a ponto de serem adaptadas ao problema de otimização que se quer tratar. É possível afirmar que boa parte das heurísticas são baseadas em metaheurísticas.

Como objetivo principal, o presente trabalho propõe desenvolver e implementar uma heurística baseada na metaheurística Algoritmos Genéticos de Chaves Aleatórias Viciadas (BRKGA, do inglês *Biased Random Key Genetic Algorithm*) [14], que vem obtendo bons resultados quando utilizada em outros problemas de otimização, para encontrar boas soluções para o NRP. Além disso, não foram encontrados trabalhos na literatura que utilizem esta metaheurística para o NRP.

Os resultados obtidos pela heurística proposta serão comparados com os melhores resultados encontrados pela heurística estado da arte para o NRP [15] e com os valores ótimos encontrados com a utilização de métodos exatos no trabalho de Veerapen et al. [9]. Tais comparações serão realizadas quanto à qualidade da solução, não sendo comparados os tempos de processamento, pois os trabalhos da literatura utilizam em seus experimentos configurações de ambiente de processamento difíceis de serem reproduzidas, dado que os experimentos foram realizados em anos anteriores e, muitas vezes, sem uma especificação clara do ambiente computacional em que foram executados. No entanto, os tempos de processamento obtidos neste trabalho não devem ultrapassar a ordem de magnitude dos tempos de processamento obtidos pela heurística estado da arte para o NRP.

Como objetivo secundário, com o propósito de demonstrar a validade em usar uma ferramenta para configuração automática de parâmetros, este trabalho propõe a execução de dois experimentos similares: um utilizando configuração automática de parâmetros e outro uma escolha aleatória dos parâmetros. A intenção é observar se há diferenças significativas nos resultados finais.

Portanto, os objetivos da pesquisa são:

1. Propor e implementar uma heurística baseada na metaheurística *Biased Random Key Genetic Algorithm* (BRKGA) para o problema NRP;
2. Propor e implementar uma estratégia baseada na técnica *K-Fold Cross Validation* para a utilização das instâncias disponíveis de forma a, juntamente com uma ferramenta para a configuração automática de parâmetros, realizar a escolha dos valores dos parâmetros da metaheurística a serem usados no estudo experimental;
3. Projetar, executar e analisar os resultados de um estudo experimental utilizando a heurística do objetivo 1, com instâncias utilizadas recorrentemente em outros trabalhos da literatura. Comparar os resultados obtidos com os resultados encontrados na literatura, considerando a eficácia (qualidade das soluções) no estudo comparativo.
4. Comparar os resultados obtidos no objetivo 2 com os resultados obtidos com a utilização de valores dos parâmetros obtidos segundo um padrão aleatório.

1.4 Justificativa

Problemas de otimização combinatória podem ser resolvidos por métodos exatos para algumas de suas instâncias. No entanto, para instâncias grandes, a utilização de tais métodos pode levar a tempos de processamento inaceitáveis. Para tratar essa questão e chegar a soluções de boa qualidade, próximas do ótimo, faz-se uso de técnicas heurísticas fortemente baseadas em algoritmos construtivos, na aleatoriedade na execução desses algoritmos, obtida através da geração e uso de números pseudo-randômicos, e na abordagem utilizada para as aproximações sucessivas em direção à melhor solução. Dentre essas abordagens, pode-se destacar as metaheurísticas [18].

Uma metaheurística é um *framework* algorítmico de alto nível de abstração e independente do problema a ser abordado, que fornece um conjunto de orientações e estratégias para o desenvolvimento de algoritmos heurísticos de otimização. Contudo, o termo também é usado para se referir à implementação de um algoritmo heurístico de otimização para um problema específico, de acordo com as orientações expressas por tal *framework* [16]. Dada essa característica de generalização atribuída a uma metaheurística, o que a torna apta, pelo menos em teoria, a ser aplicada a qualquer problema de otimização combinatória, há a necessidade de parâmetros de configuração que devem ser ajustados conforme a heurística implementada, o problema e as instâncias desse problema, de forma a se obter soluções de qualidade em tempos de execução tão pequenos quanto possível.

Os valores atribuídos aos parâmetros de uma metaheurística, determinantes para o bom desempenho da metaheurística tanto em termos de tempo de execução quanto da qualidade das soluções geradas, são definidos de forma intuitiva em muitos casos. A observação dos resultados de algumas poucas execuções da metaheurística ou ainda, o conhecimento e experiência do pesquisador em relação ao problema sobre o qual se está aplicando a metaheurística são alguns dos expedientes utilizados para a definição dos valores dos parâmetros. Tal fato revela a ausência de um método bem definido que oriente a definição desses valores, resultando em forte dependência em relação a quem configura (pesquisador), além de ser um procedimento dispendioso em termos de tempo e suscetível a erros por ser manual e não automático.

Utilizar um método bem definido ou uma ferramenta baseada em um método bem definido, que garanta reprodutibilidade no procedimento de configuração de parâmetros de uma metaheurística, pode assegurar combinações de parâmetros que geram boas soluções para a maioria das instâncias, economia de tempo e independência de expertise sobre o problema ao qual se está aplicando a metaheurística.

Uma classificação interessante sobre os tipos de trabalhos com experimentação utilizando heurísticas sugere a divisão em quatro diferentes categorias, embora seja possível trabalhos que permeiem duas ou mais categorias: (i) trabalhos que resultam em artigos de aplicação, onde o objetivo é descrever o impacto do algoritmo na resolução de um problema inédito ou já conhecido, importando mais a qualidade das soluções que o desempenho no tempo de execução; (ii) trabalhos que pretendem mostrar quão melhores e mais rápidos são seus algoritmos em relação aos de trabalhos já publicados (artigos do tipo "*horserace*"); (iii) trabalhos de análise experimental, cuja motivação é entender os pontos fortes e fracos dos algoritmos; e, por fim, (iv) trabalhos experimentais de caso médio, em que se busca compreender o comportamento de caso médio quando o algoritmo é executado em uma distribuição de instâncias onde a análise probabilística é difícil [17].

O NRP é frequentemente objeto de estudo com a aplicação de novos algoritmos, seja através da proposta de novos métodos construtivos e/ou a aplicação de metaheurísticas conhecidas porém nunca utilizadas no problema. A intenção desses estudos é demonstrar o ganho em eficácia (qualidade das soluções) e eficiência (tempo de execução) em relação a trabalhos anteriores, um típico caso que se enquadra na segunda categoria descrita no parágrafo anterior.

No caso do presente trabalho de pesquisa, as 39 instâncias utilizadas já tiveram seus ótimos calculados por Veerapen et al. [9]. Tal fato poderia colocar em dúvida a relevância do presente trabalho, uma vez que um de seus principais objetivos é propor uma nova

heurística para um problema já resolvido para as instâncias utilizadas. No entanto, por se tratar de um problema em que muitas outras novas instâncias de diferentes características e tamanhos podem surgir ao longo do tempo, não é garantido que os métodos exatos que calcularam os ótimos para as 39 instâncias realizem o mesmo feito para as novas instâncias. A escolha de um problema e de um conjunto de instâncias já resolvidas, ou seja, com os ótimos calculados, é justificada pelo motivo do presente trabalho propor e implementar uma estratégia para a configuração dos parâmetros da metaheurística, conforme descrito no segundo objetivo, estratégia esta que poderia ser utilizada com qualquer outro conjunto de instâncias, algumas das quais possivelmente inviáveis de serem abordadas pelos métodos exatos em um tempo computacional aceitável.

1.5 Estrutura do Trabalho

O Capítulo 1 apresenta a definição do problema, a motivação e justificativa para o estudo. Por fim, expõe os objetivos deste trabalho e suas contribuições.

O Capítulo 2 apresenta a fundamentação teórica sem a qual não seria possível o trabalho. Discorre sobre a metaheurística empregada para resolver o NRP e os conceitos, técnicas e ferramentas utilizados na configuração automática de parâmetros. O capítulo também menciona outros trabalhos de pesquisa relacionados.

O Capítulo 3 apresenta a proposta de solução, uma combinação de três componentes: a heurística para o NRP, consistindo da metaheurística BRKGA em conjunto com o algoritmo construtivo, nomeada como BRKGA_NRP, a estratégia baseada na técnica *K-Fold Cross Validation* para seleção de instâncias e a ferramenta escolhida para a configuração automática de parâmetros.

O Capítulo 4 apresenta os experimentos computacionais realizados no trabalho: o estudo de convergência, que auxiliou no entendimento do comportamento da heurística quando em execução e justificou a escolha de dois importantes parâmetros da metaheurística, a execução da estratégia baseada na técnica *K-Fold Cross Validation* em associação à ferramenta de configuração automática de parâmetros bem como a execução desta mesma técnica só que associada a um padrão aleatório, ambos para a obtenção dos valores a serem utilizados na terceira e última etapa dos experimentos, a execução da heurística proposta. Ainda no capítulo, é realizada a análise comparativa dos resultados obtidos.

O Capítulo 5 apresenta as conclusões do trabalho assim como as sugestões de pesquisa para trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

2.1 A Metaheurística *Biased Random Key Genetic Algorithm*

Muitas metaheurísticas têm sido propostas nos últimos trinta anos [18]. Entre elas, encontram-se Procedimentos de Busca Gulosos, Aleatórios e Adaptativos (GRASP, do inglês *Greedy Randomized Adaptive Search Procedure*) [19], Arrefecimento Simulado (em inglês *Simulated Annealing*) [20], Busca Tabu (em inglês *Tabu Search*) [21], Busca em Vizinhança Variável (em inglês *Variable Neighborhood Search*) [22], Busca Dispersa (em inglês *Scatter Search*) [23], Busca Local Iterada (em inglês *Iterated Local Search*) [24], Otimização por Colônia de Formigas (ACO, do inglês *Ant Colony Optimization*) [25], Otimização por Enxames de Partículas (PSO, do inglês *Particle Swarm Optimization*) [26] e Algoritmos Genéticos (GA, do inglês *Genetic Algorithm*) [27]. Muito embora haja notável esforço no emprego de métodos exatos para encontrar soluções ótimas para problemas de otimização combinatória, a razão para tanta variedade de metaheurísticas é que para muitos problemas, encontrar soluções de boa qualidade rapidamente, mesmo não sendo ótimas, gera mais benefícios que encontrar a solução ótima na contrapartida de tempo e outros recursos de processamento.

Algoritmos Genéticos [28] utilizam o conceito de sobrevivência do melhor ou mais apto para encontrar soluções ótimas ou próximas do ótimo em problemas de otimização combinatória. Em uma dada população, cada indivíduo é constituído de um único cromossomo, que por sua vez representa de forma codificada uma solução. Cada cromossomo consiste de uma sequência de genes, cujos valores são conhecidos como alelos. Uma função matemática decodifica os valores dos genes de um dado cromossomo em um valor de solução, que se busca otimizar. Como em um processo biológico evolutivo, Algoritmos Genéticos promovem a evolução de uma população ao longo das gerações. A cada geração, uma nova população é criada pela combinação ou cruzamento entre indivíduos da população anterior com a intenção de gerar descendentes. Além dos descendentes, a

nova população também pode ser constituída por mutantes, que são indivíduos gerados de forma aleatória, cujo objetivo é evitar mínimos (ou máximos, dependendo do tipo de otimização) locais. O conceito de sobrevivência do mais apto ou de seleção natural é levado a efeito na escolha dos indivíduos para o cruzamento: aqueles cujo cromossomo está associado a soluções melhores são priorizados em relação àqueles com soluções piores.

Algoritmos Genéticos com Chaves Aleatórias (RKGA, do inglês *Random Key Genetic Algorithm*) foram inicialmente introduzidos por Bean [29] para resolver problemas de otimização combinatória envolvendo sequenciamento. O RKGA promove a evolução de uma população de indivíduos ao longo de um número pré-definido de iterações ou gerações. Cada cromossomo é representado por uma sequência de chaves aleatórias (números reais aleatoriamente gerados entre 0 e 1). A população inicial é constituída de p vetores, cada qual representando um indivíduo. Após o *fitness* de cada indivíduo ser calculado por um algoritmo determinístico que decodifica o cromossomo em uma solução candidata, a população é particionada em dois grupos de indivíduos: um grupo menor, constituído pelos indivíduos que apresentaram os melhores *fitness* e designado como estrato elite, e um grupo maior, constituído pelos demais indivíduos. Para a próxima geração, o RKGA usa uma estratégia elitista, pois utiliza todos os cromossomos do estrato elite, o que garante que as melhores soluções obtidas em gerações anteriores sejam levadas para as futuras gerações, resultando invariavelmente em uma heurística com uma tendência a resultados cada vez melhores a cada iteração.

Ainda na composição da próxima geração, o RKGA introduz uma quantidade de descendentes de indivíduos da geração atual e uma quantidade de mutantes. Em ambos os casos, tratam-se de quantidades fixas pré-definidas que se mantêm ao longo das gerações, que somadas ao também constante número de indivíduos do estrato elite, resultam no valor p, o tamanho da população. No primeiro caso, a heurística promove o cruzamento entre um número fixo de indivíduos da geração atual, considerando que um descendente é gerado para cada dois indivíduos. Todos os indivíduos possuem a mesma probabilidade de serem escolhidos para o cruzamento, não importando se são do estrato elite ou não. Um sorteio gene a gene de cada cromossomo, seguindo uma distribuição binomial com probabilidades diferentes de sucesso e fracasso (no caso do "Cara ou Coroa", as probabilidades para sucesso e fracasso são as mesmas e iguais a 0,5), determina o alelo a ser herdado pelo descendente. Já no caso da mutação, os mutantes nada mais são que indivíduos gerados aleatoriamente, com cada um de seus alelos com valores sorteados entre 0 e 1, da mesma forma que a população inicial, na primeira geração, foi gerada.

A metaheurística Algoritmos Genéticos com Chaves Aleatórias Viciadas (BRKGA, do inglês *Biased Random Key Genetic Algorithm*) [14] difere de RKGA no modo como

dois indivíduos são escolhidos para realizar o cruzamento e gerar descendentes. Na metaheurística BRKGA, um indivíduo é escolhido do estrato elite com a probabilidade $1/p_e$ e o outro indivíduo é escolhido dentre os demais da população com probabilidade $1/(p - p_e)$, onde p_e é o número de indivíduos do estrato elite. Ou seja, em um cruzamento, um dos indivíduos sempre estará entre aqueles que possuem os melhores *fitness*. A Figura 2.1 ilustra como ocorre a evolução da população na metaheurística BRKGA.

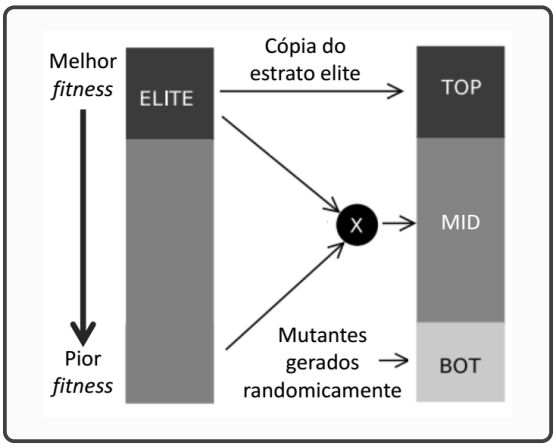

Figura 2.1: Transição da geração atual para a próxima geração.

Primeiramente, a população da geração atual é ordenada de acordo com o *fitness* de cada indivíduo. Para a futura geração, os indivíduos com os melhores *fitness* (estrato elite) são copiados para a partição TOP. A partição MID é formada pelos descendentes resultantes do cruzamento entre um indivíduo do estrato elite e outro que não pertence a esse estrato. A partição BOT corresponde aos mutantes, novos indivíduos criados para a futura geração, cujos alelos possuem valores gerados randomicamente. Quando a futura geração torna-se a geração corrente, os *fitness* são calculados para os descendentes e para os mutantes, a ordenação é realizada e um novo estrato elite é observado, com a situação possível de que alguns indivíduos antes desse estrato percam espaço para indivíduos que estavam em MID e BOT. Por fim, toda a mecânica da evolução ocorre novamente.

A Figura 2.2 mostra como ocorre o cruzamento na metaheurística BRKGA. No exemplo, os indivíduos possuem um cromossomo com quatro genes. O Cromossomo 1 refere-se ao indivíduo do estrato elite e o Cromossomo 2 ao indivíduo da parte da população fora desse estrato. A probabilidade de que um gene do cromossomo do indivíduo do estrato elite seja herdado por seu descendente na próxima geração, representado por ρ_e, possui o valor de 0,7. Logo, a probabilidade de um gene do Cromossomo 1 prevalecer no gene do descendente é 0,7 e a do Cromossomo 2 é 0,3. O sorteio de um número real entre 0 e 1 determina qual gene de um dos dois cromossomos prevalecerá. É realizado um sorteio para cada gene. Se o número sorteado for menor ou igual a 0,7, ganha o Cromossomo 1 (estrato elite) e se for maior que 0,7 e menor ou igual a 1, ganha o Cromossomo

2. Na Figura 2.2, pode ser observado que o Cromossomo 1 levou o primeiro, terceiro e quarto gene para seu descendente, pois apenas o sorteio para o segundo gene foi um valor superior a 0,7 (0,89).

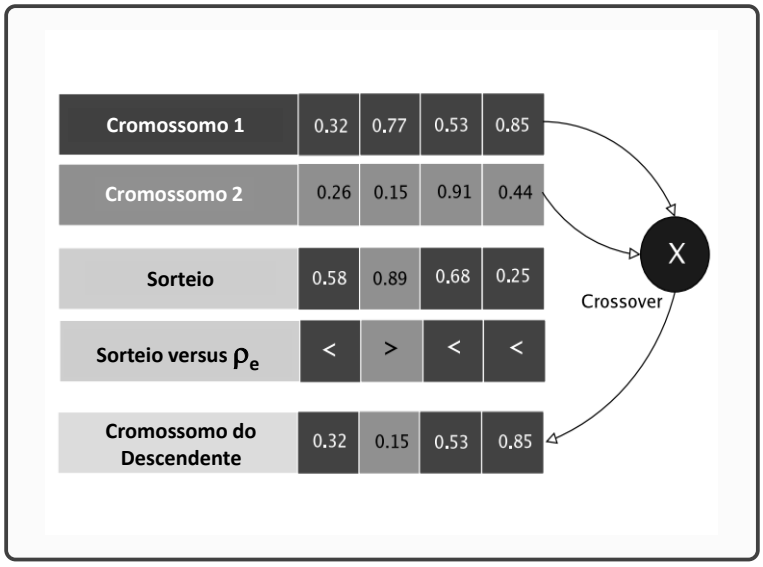

Figura 2.2: Cruzamento entre dois indivíduos na metaheurística BRKGA.

Um dos propósitos de uma metaheurística é ser flexível o bastante para abordar qualquer problema de otimização combinatória. Na metaheurística BRKGA, essa flexibilidade é possível graças à organização estrutural do *framework* [14], que apresenta uma parte independente e outra dependente do problema. A parte independente trata do *modus operandi* da metaheurística BRKGA, ou seja, os detalhes de como a população evolui de uma geração para a outra, conforme ilustra a Figura 2.3. A parte dependente do problema, também conhecida como decodificador, consiste de uma heurística construtiva específica para o problema e concebida pelo trabalho de pesquisa que adotou a metaheurística BRKGA como proposta de solução. Em geral, usando estruturas de dados eficientes, essas heurísticas buscam soluções eficientes e eficazes, isto é, soluções de baixo custo computacional e boa qualidade. Nesta parte, a informação representada pelo cromossomo e armazenada nas chaves aleatórias de seus genes é decodificada para ser usada pelos algoritmos do método construtivo, bem como o *fitness* é calculado ao final de seu processamento.

A metaheurística BRKGA possui alguns parâmetros relativos à parte independente do problema que precisam ser configurados. Estes parâmetros são o tamanho da população, o percentual da população definido como estrato elite, o percentual da população correspondendo aos mutantes introduzidos a cada nova geração, a probabilidade do gene do descendente herdar o alelo do cromossomo do indivíduo do estrato elite e o critério de parada. Tal critério pode ser um número fixo de gerações, um limite de tempo ou

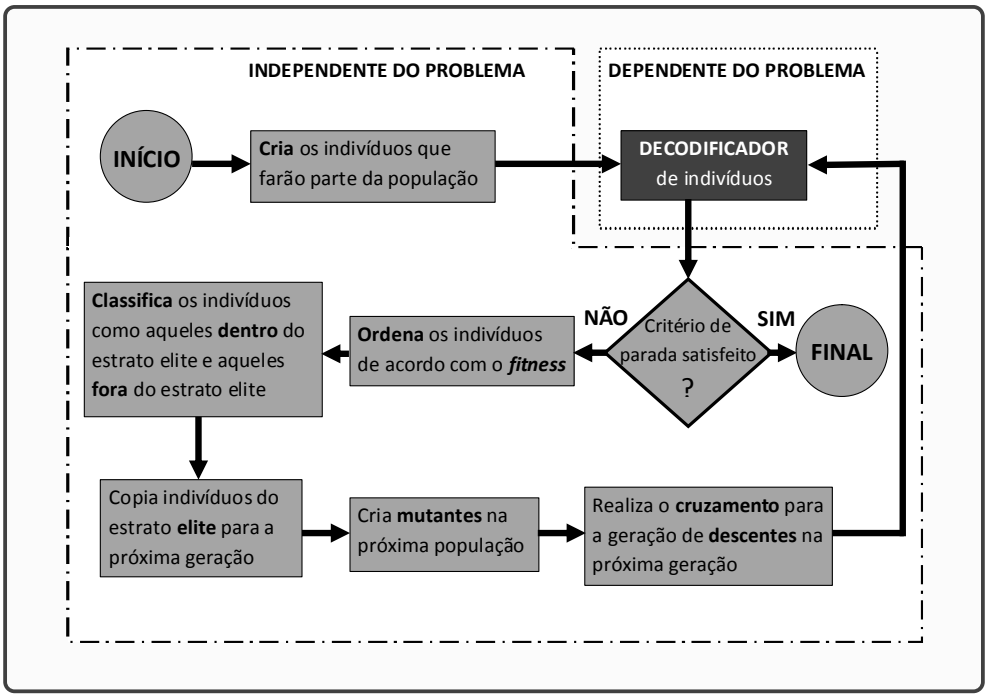

Figura 2.3: *Framework* da metaheurística BRKGA.

quando o *fitness* alcança um valor pré definido, em geral, um percentual em relação ao ótimo, caso seja conhecido, ou algum limite superior ou inferior (em inglês *upper bound* e *lower bound*, respectivamente). A Tabela 2.1 possui valores recomendados pelo autor da metaheurística para os parâmetros, baseado em sua experiência no uso e aplicação da metaheurística BRKGA [14]. Vale ressaltar que n (tamanho do cromossomo, que é a quantidade de genes) é um parâmetro definido na parte dependente do problema, uma vez que tal parâmetro codifica ou traduz a representação do problema.

Tabela 2.1: Valores recomendados para os parâmetros da metaheurística BRKGA.

Parâmetro	Descrição	Valor recomendado
n	Tamanho do cromossomo	Sem recomendação, pois se trata de um parâmetro dependente do problema
p	Tamanho da população	$p = an$, onde $1 \leq a \in \mathbb{R}$ é uma constante
p_e	Percentual do estrato elite	$0{,}10p \leq p_e \leq 0{,}25p$
p_m	Percentual de mutantes	$0{,}10p \leq p_m \leq 0{,}30p$
ρ_e	Probabilidade do descendente herdar o "gene elite"	$0{,}50 < \rho_e \leq 0{,}80$

Além dos parâmetros já descritos, existem três outros que precisam ser configurados caso se queira usar processamento paralelo: número de populações independentes, periodicidade (em número de gerações) para a troca dos indivíduos com melhores *fitness* entre as populações independentes e número de indivíduos com melhores *fitness* que serão tro-

cados entre as populações independentes. O primeiro parâmetro refere-se às populações que evoluirão independentemente, cada qual em seu processo. O segundo e o terceiro parâmetros dizem respeito a introduzir diversidade na qualidades das soluções (*fitness*) geradas com o objetivo de alcançar soluções ainda melhores mais rapidamente (em um número menor de gerações).

2.2 Configuração Automática de Parâmetros de Metaheurísticas

Muitos algoritmos concebidos para resolver problemas de otimização combinatória envolvem um grande número de escolhas de *design* e parâmetros específicos que precisam ser cuidadosamente configurados para que tais algoritmos alcancem o melhor possível em eficácia (qualidade das soluções) e eficiência (tempo de processamento). Isto se aplica tanto em métodos exatos, como *Branch-and-Bound* e técnicas que utilizam programação linear, como em métodos heurísticos, como busca local e metaheurísticas. Obter o melhor em eficácia e eficiência desses algoritmos pode envolver a configuração de dezenas de centenas de parâmetros. Mesmo quando o algoritmo possui uma ou mais configurações padrão de parâmetros, certamente estas configurações foram definidas quando na resolução de outros problemas de otimização combinatória ou até mesmo na resolução do mesmo problema que se está pesquisando, porém em contextos de aplicação diferentes, como por exemplo, instâncias diferentes das que se está usando [30] [31].

Por muito tempo, a configuração de parâmetros de algoritmos de otimização tem sido realizada de forma intuitiva. Em geral, o desenvolvedor do algoritmo propõe um conjunto pequeno de configurações, ou seja, de combinações de valores para cada um dos parâmetros e executa experimentos para avaliar o resultado obtido por cada uma das configurações do conjunto. Com base nesta avaliação, o desenvolvedor decide se testa novas combinações de valores, se modifica o algoritmo ou se dá por encerrado o processo de escolha de uma ou mais configurações padrão. O modo manual como é executado esse processo traz desvantagens como: a demanda por um tempo considerável do desenvolvedor ou pesquisador em uma tarefa que poderia ser automatizada; a necessidade de intuição e experiência pessoal do desenvolvedor ou pesquisador, que tanto pode levar a bons resultados, como pode levar a resultados tendenciosos, ou seja, bons resultados apenas para casos sobre os quais o desenvolvedor tenha domínio de conhecimento; por não ser uma tarefa automatizada, poucas configurações são exploradas e; frequentemente as mesmas instâncias que são usadas durante a etapa de escolha das combinações de parâmetros são também usadas na avaliação no algoritmo final, o que pode levar a uma avaliação equivocada ou tendenciosa das soluções geradas [32].

A configuração automática de parâmetros pode ser descrita como o problema de encontrar combinações de valores para os parâmetros que gerem bons resultados em instâncias diferentes daquelas utilizadas na etapa em que tais combinações foram determinadas [33]. Em uma perspectiva de aprendizagem de máquina, os bons resultados obtidos na etapa de treinamento devem ser confirmados na etapa de teste ou validação em instâncias diferentes, não importando o quanto ou como diferentes são, se em tamanho, complexidade ou outro aspecto que caracterize instâncias de um dado problema de otimização combinatória. Em outras palavras, o problema de encontrar boas configurações de parâmetros para a resolução de instâncias ainda não conhecidas através da aplicação em um conjunto menor de instâncias, já conhecidas. Portanto, trata-se de uma tarefa de predição. Tal abordagem é dita *offline*, pois uma vez determinada a configuração de parâmetros na etapa de treinamento, ela será utilizada sem ajustes na etapa de validação e em ambientes de produção. Em abordagens *online*, ajustes nos valores iniciais dos parâmetros são feitos ao longo da execução do algoritmo com base na avaliação de resultados intermediários, como a taxa de convergência. No entanto, abordagens *online* também possuem parâmetros que precisam ser definidos de forma *offline*, como quais parâmetros são corrigidos em tempo de execução [34].

O motivo para que os parâmetros necessitem ser configurados é o fato de não existir uma única combinação de valores ótima para toda e qualquer aplicação do algoritmo, de forma que a configuração ótima desses parâmetros depende do problema a ser abordado. De fato, configurar tais parâmetros não é uma tarefa trivial e por si só constitui-se em um problema de otimização combinatória [33]. Por esse motivo, para boa parte das metaheurísticas existentes, senão todas, a definição de valores para cada um dos parâmetros, e claro, as possíveis combinações desses valores que definem uma dada configuração, ocorre, em geral, de três formas: a) simplesmente replicando os valores de trabalhos anteriores que usaram a mesma metaheurística em problemas iguais ou de características similares; b) baseada na experiência do pesquisador e; c) através de ferramentas de configuração automática de parâmetros.

No primeiro caso, há chances de algum êxito no procedimento. No entanto, embora possa existir a semelhança entre dois problemas em suas definições e características, as instâncias de cada um podem possuir particularidades que degradem a qualidade da solução gerada (valores de *fitness*) – onde se observou um valor de *fitness* muito bom para uma dada combinação de valores dos parâmetros, agora se observa um valor ruim para a mesma combinação, justamente porque o conjunto de instâncias mudou.

No segundo caso, o pesquisador deve conhecer bem o problema, como detalhes de suas instâncias, e a própria metaheurística, de forma a observar se existe alguma relação

entre as características das instâncias e os parâmetros da metaheurística e usar este conhecimento para realizar a configuração dos parâmetros, o que traz uma forte dependência de um fator humano: a experiência do pesquisador.

Já no terceiro caso, uma ferramenta de configuração automática de parâmetros tem em si um ou mais métodos que orientam a escolha da melhor combinação de valores para os parâmetros da metaheurística. Resumidamente, essa escolha é realizada com base na comparação de valores de *fitness* resultantes de sucessivas execuções da metaheurística, que recebeu como entrada os valores dos parâmetros passados pela ferramenta de configuração automática. A ferramenta realiza as próximas escolhas dos valores dos parâmetros, dando preferência àqueles situados na mesma região do domínio daqueles que proporcionaram os melhores valores de *fitness* retornados pela metaheurística.

2.2.1 A Ferramenta de Software IRACE

Com o objetivo de propor abordagens alternativas à configuração manual de parâmetros, cujas desvantagens foram expostas anteriormente, muitos trabalhos de pesquisa têm sido realizados com o propósito de resolver o problema de automatizar a configuração de parâmetros. Métodos inspirados em Projeto ou Planejamento de Experimentos (DOE, do inglês *Design Of Experiments*) foram propostos, como CALIBRA [35], métodos baseados em metodologia de superfície de resposta [36] e técnicas que aplicam análise de variância (ANOVA, do inglês *Analysis of Variance*) [37]. Também foram propostos otimizadores numéricos do tipo caixa-preta, como CMA-ES [38], BOBYQA [39] e MADS [40], e algoritmos baseados em metaheurísticas clássicas, como REVAC [41], EVOCA [42] e ParamILS [43], onde os dois primeiros utilizam Algoritmos Genéticos e o último a Busca Local Iterada.

Para avaliar se uma dada combinação de parâmetros gera bons resultados em uma metaheurística aplicada a um certo problema, o algoritmo necessariamente tem de ser executado, pois a avaliação é baseada no *fitness* gerado ao final da execução. Dependendo do tempo que o algoritmo da metaheurística consome para concluir sua execução e a quantidade de vezes que a execução será realizada, pode haver uma alta demanda por recursos computacionais e tempo. Além disso, certas combinações de valores para os parâmetros resultarão em mais recursos computacionais e tempo e outras combinações menos. Por esse motivo, alguns métodos foram propostos com o objetivo de estimar o tempo consumido e o *fitness* gerado quando uma certa configuração é aplicada e só de fato executar o algoritmo da metaheurística caso seja uma configuração promissora, a despeito do tempo consumido, e atualizar o modelo de predição com os valores de tempo

e *fitness* obtidos após essa execução. Como exemplo desses métodos, tem-se SMAC [44] e SPOT [45].

Por fim, alguns métodos selecionam uma configuração dentre um número de configurações candidatas utilizando testes estatísticos sequenciais, em um procedimento conhecido como competição de corrida ("*racing*"em inglês) [46]. As configurações candidatas iniciais podem ser criadas aleatoriamente ou baseadas em algum conhecimento específico sobre o problema. No caso de competição iterada de corrida ("*iterated racing*"em inglês), um modelo estatístico de amostragem é refinado a cada iteração, de acordo com os resultados dos testes estatísticos anteriores [47].

A competição iterada de corrida é um método que considera várias iterações (ou corridas) no objetivo de realizar a configuração automática de parâmetros, selecionando as melhores configurações através de testes estatísticos sequenciais ao longo de cada iteração e levando adiante para as iterações subsequentes as melhores configurações. Consiste em três etapas: amostragem de novas configurações de acordo com uma distribuição de probabilidade em particular, seleção através de testes estatísticos sequenciais das melhores configurações dentre as novas da amostra e as anteriores vindas de iterações passadas e atualização da distribuição de probabilidade com a intenção de criar uma tendência para que boas configurações sejam sorteadas na amostragem. O processo todo é repetido até que um critério de parada seja alcançado [48].

Desenvolvida na linguagem R [49] pelo laboratório de pesquisas em Inteligência Artificial (IRIDIA, do francês *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*) do departamento de Computação e Engenharia de Decisão (CoDE, do inglês *Computer and Decision Engineering*) da Universidade Livre de Bruxelas (em francês *Université Libre de Bruxelles*), a ferramenta de software IRACE [50] implementa a competição iterada de corrida em suas definições e características conforme resumidas nas três etapas descritas anteriormente.

Uma série de trabalhos anteriores culminou com a ferramenta IRACE e as extensões adicionadas em suas versões lançadas posteriormente. Desde o início, tais trabalhos basearam-se em abordagens no formato "competição de corrida". O primeiro trabalho propôs o F-Race [47], uma abordagem para a configuração automática de parâmetros baseada no algoritmo de competição de corrida e no teste estatístico de Friedman [51] (*"Friedman's non-parametric two-way analysis of variance by ranks"* em inglês). Em seguida, Birattari et al. propuseram uma melhoria sobre F-Race chamada I/F-Race (do inglês *Iterated F-Race*) [48], com a atualização da distribuição de probabilidade da amostra através de repetidas aplicações do método F-Race. Finalmente, a ferramenta IRACE

foi lançado com a incorporação nos trabalhos anteriores das seguintes funcionalidades: (i) teste estatístico t de Student; (ii) distribuição de probabilidade normal truncada para os parâmetros numéricos (reais e inteiros) do algoritmo a ser configurado; (iii) suporte a processamento paralelo; (iv) um procedimento elitista para garantir que as melhores configurações encontradas tenham sido de fato avaliadas no maior número possível de instâncias; e (v) uma estratégia de reinício para evitar convergência prematura. Uma taxonomia para a ferramenta IRACE é mostrada na Figura 2.4.

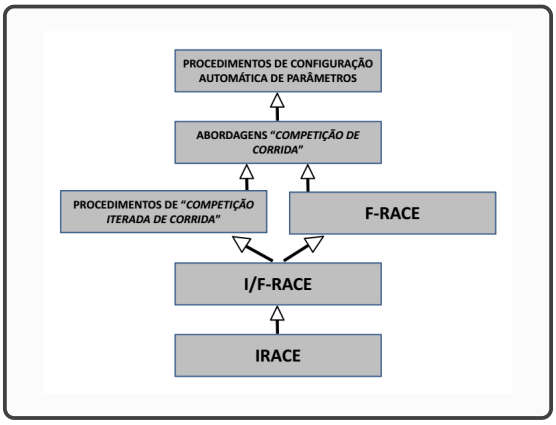

Figura 2.4: IRACE: Taxonomia de métodos, técnicas e produtos relacionados.

A ferramenta IRACE considera quatro categorias de parâmetros de configuração de uma dada metaheurística: números inteiros, números reais, valores ordinais e valores categóricos. A cada parâmetro é atribuída uma distribuição de probabilidade que é independente das distribuições de probabilidade dos demais parâmetros. Para os parâmetros representados por números inteiros, números reais ou valores ordinais, a distribuição de probabilidade é uma distribuição normal truncada, e para os representados por valores categóricos, uma distribuição discreta inicialmente uniforme. Em cada iteração, a atualização das distribuições de probabilidade de cada parâmetro consiste em modificar a média e o desvio padrão para a distribuição normal e os valores de probabilidade para a distribuição discreta, que a partir da segunda iteração deixa de ser uniforme. Tais atualizações aumentam a probabilidade de que valores dos parâmetros pertencentes às melhores configurações encontradas até o momento sejam sorteados.

Após novas configurações serem sorteadas, as melhores configurações são escolhidas com base nos *fitness* obtidos pela execução da metaheurística para cada uma das configurações em uma dada instância do problema. No exemplo da Figura 2.5, tem-se 10 configurações θ_i. Em cada passo da iteração, as configurações candidatas são aplicadas em uma única instância (I_j). Quando um certo número de passos é alcançado, são descartadas as configurações candidatas que tiveram um desempenho pior que ao menos uma das demais configurações. Após a eliminação das configurações, o processo conti-

nua com as configurações remanescentes até um próximo teste estatístico com o objetivo de descartar mais configurações. O processo termina quando resta um número mínimo pré-definido de configurações ou é alcançado um número máximo pré-definido de experimentos. Por experimento, compreende-se a execução da metaheurística (ou o algoritmo que se está configurando) sobre uma dada instância, utilizando como entrada uma dada configuração. Como uma dada configuração é aplicada em várias instâncias ao longo de todo o processo, cada configuração candidata tem um conjunto de resultados (valores de *fitness*). As configurações candidatas com pior desempenho são descartadas com base nos resultados de testes estatísticos sequenciais, que avaliam se há uma diferença estatisticamente significativa entre os conjuntos de resultados de duas configurações, pois as configurações são comparadas de duas em duas.

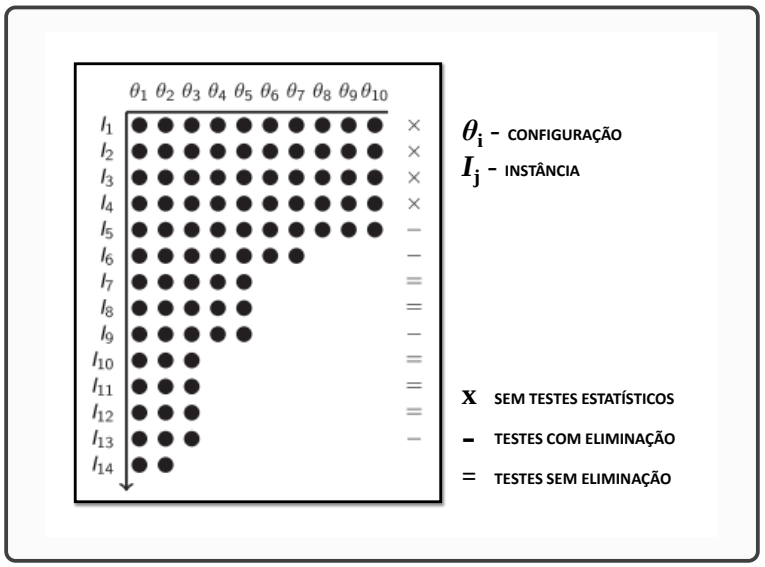

Figura 2.5: IRACE: Etapas de uma iteração.

Para que se possa enviar para a implementação executável da metaheurística uma configuração com valores sorteados segundo a distribuição de probabilidade de cada um dos parâmetros, bem como receber o (*fitness*) recuperado da execução da implementação da metaheurística com a configuração enviada, de forma a usá-lo nos testes estatísticos que determinarão se uma dada configuração será mantida ou descartada, a ferramenta IRACE precisa realizar a comunicação com a implementação da metaheurística. Essa comunicação é intermediada por uma rotina ou programa executável escrito em qualquer linguagem que possa ser chamado pela linguagem R. No entanto, a arquitetura de integração recomendada para o uso com a ferramenta IRACE sugere que essa rotina seja um arquivo batch, no caso do sistema operacional Windows, ou bash, no caso do sistema operacional Linux, conforme mostra a Figura 2.6. A rotina deve ser capaz de: a) receber alguns parâmetros de entrada da ferramenta IRACE, como a configuração gerada, o nome e localização da instância sobre a qual a metaheurística será executada e um número a

ser usado pela implementação da metaheurística como semente para a geração de números pseudo-randômicos; b) executar a implementação da metaheurística encaminhando os valores enviados pela ferramenta IRACE; c) receber o *fitness*, o tempo de execução ou ambos, enviados pela implementação da metaheurística; e d) encaminhar esses valores para a ferramenta IRACE.

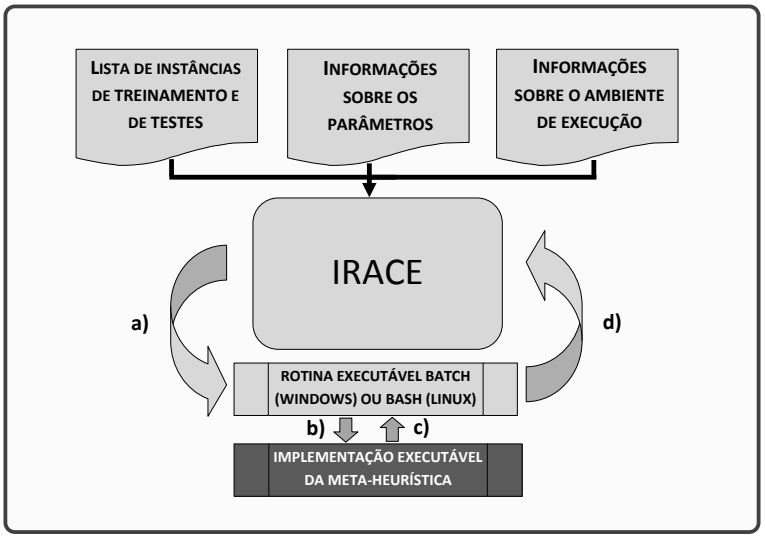

Figura 2.6: IRACE: Arquitetura de integração.

Como repositórios de informações auxiliares ao processamento da ferramenta IRACE, tem-se um arquivo com a lista de instâncias de treinamento e de testes, um arquivo com informações gerais do ambiente de execução, como diretórios, localização de arquivos e programas e valores de variáveis internas à ferramenta IRACE necessárias à sua execução, e um arquivo com informações sobre os parâmetros da metaheurística que serão configurados, como a categoria (inteiro, real, ordinal ou categórico), intervalo de valores (inteiro ou real) ou valores nominais (ordinal ou categórico), "alias", ou nome como cada parâmetro é conhecido pela implementação da metaheurística e, por fim, relações de dependência entre os parâmetros, onde se um parâmetro depende de outro parâmetro, os valores assumidos por este determinarão os valores que aquele poderá assumir.

Embora seja possível à ferramenta IRACE realizar diretamente a chamada ao programa executável, que é a implementação do algoritmo ou metaheurística para o qual a configuração automática de parâmetros será feita, usar um programa intermediário, tal como um script batch (ou bash), permite que alguma lógica adicional que não tenha relação alguma com a implementação da metaheurística, mas sim com ajustes ou transformações nos valores sorteados dos parâmetros, seja incluída. Outro motivo para o uso de um programa intermediário é que a ferramenta IRACE necessariamente passa quatro valores que não são os parâmetros de configuração e que precisam ser tratados ou ignorados. Tais valores são um identificador interno para a configuração representada pelos parâme-

tros, um identificador interno para a instância, uma semente gerada randomicamente e o caminho completo no sistema de arquivos acompanhado do nome da instância. Logo, manter um programa intermediário para abordar essas questões permite ao programa executável da metaheurística menos acoplamento em relação à ferramenta IRACE, com a obrigatoriedade de receber em sua interface de entrada apenas os valores dos parâmetros e informações sobre o nome e a localização da instância.

2.3 A Técnica *K-Fold Cross Validation*

Na avaliação de modelos estatísticos criados para descrever um conjunto de dados associado a um fenômeno qualquer, uma das principais preocupações é ter um modelo que represente satisfatoriamente os dados utilizados na sua formulação assim como dados desconhecidos associados ao mesmo fenômeno, ainda não vistos e portanto não utilizados tanto na calibração quanto na validação do modelo. Modelos fidedignos ao conjunto de dados usado na sua formulação, porém com pouca aderência a novos dados são indesejados. Esse problema é conhecido em Estatística como balanceamento entre variância e tendenciosidade (em inglês "*bias-variance trade-off*") e em Aprendizagem de Máquina como sobreajuste (em inglês "*overfitting*") [52]. O modelo que obtém a melhor generalização para os dados é aquele que consegue o melhor equilíbrio entre ser fidedigno ao conjunto de dados usado para a sua formulação, também conhecido como conjunto de treinamento, e o número de parâmetros estimados variando livremente. Tal modelo ideal deve ter baixa variância (alta precisão) e baixa tendenciosidade (alta acurácia ou fidelidade). Na prática, modelos relativamente flexíveis, com muitos parâmetros estimados variando livremente, ajustam-se muito bem ao conjunto de treinamento e tendem a possuir alta variância e baixa tendenciosidade, enquanto que modelos com menos parâmetros tendem a ter baixa variância e alta tendenciosidade.

Para mitigar o problema de balanceamento entre variância e tendenciosidade, visando obter um modelo estatístico que descreva um conjunto de dados da forma mais fidedigna possível, algumas técnicas de validação do modelo (ou validação de regressão, em um termo mais adequado à área de Estatística) foram propostas. Dentre estas técnicas, a técnica *K-Fold Cross Validation* [52] se destaca pela simplicidade de implementação e custo computacional. *K-Fold Cross Validation* consiste em dividir o conjunto P de todas as instâncias em K partições de aproximadamente mesmo tamanho, P_1, P_2, ..., P_K. Tanto a etapa de treinamento quanto a de teste ocorrerão K vezes. Em cada momento $t \in \{1, 2, ..., K\}$, as instâncias de treinamento utilizadas pertencerão ao subconjunto $P \setminus P_t$ e as de teste ao subconjunto P_t. Se K não for um divisor de $|P|$, será criada uma nova partição

com as instâncias restantes. O treinamento é feito em K rodadas, onde a cada rodada uma partição é considerada como partição de instâncias de teste e as demais como partições de instâncias de treinamento [53]. O exemplo da Figura 2.7 mostra de forma simplificada os momentos t da técnica com K = 4 (*4-Fold Cross Validation*). No momento $t = 1$, o conjunto de treinamento corresponde à união das partições P_2, P_3 e P_4, com a partição P_1 representando o conjunto de teste. No momento $t = 2$, o conjunto de treinamento é representado pelas partições P_1, P_3 e P_4 e o conjunto de teste representado pela partição P_2. No momento $t = 3$, o conjunto de treinamento corresponde às partições P_1, P_2 e P_4, com a partição P_3 como o conjunto de teste. Finalmente, no momento $t = 4$, o conjunto de treinamento corresponde à união das partições P_1, P_2 e P_3, com a partição P_4 representando o conjunto de teste.

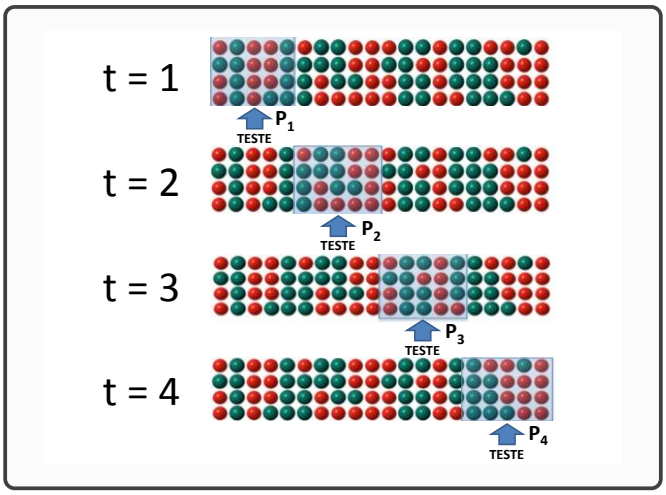

Figura 2.7: *K-Fold Cross Validation*: Exemplo com K = 4.

2.4 Trabalhos Relacionados

O NRP possui duas versões quando considerado o número de objetivos que se deseja otimizar a partir da formulação do problema: mono-objetivo e bi-objetivo. No primeiro caso, o objetivo é maximizar o lucro associado ao conjunto de clientes contemplados na próxima versão do software, respeitando a restrição do orçamento, que impõe limite ao conjunto total de requisitos que precisam ser implementados para que clientes sejam satisfeitos e o lucro seja realizado. No segundo caso, o orçamento deixa de ser uma restrição para tornar-se também um objetivo a ser otimizado, juntamente com o lucro. Como em uma situação clássica de lucros e despesas em uma operação de caixa, busca-se maximizar o lucro e minimizar o orçamento ou custo total dos requisitos implementados.

2.4.1 Formulação Mono-objetivo do NRP

O primeiro trabalho a citar o NRP é de autoria de Bagnall et al. [5]. Nele, o problema é enunciado em sua versão mono-objetivo e demonstrado ser NP-Difícil. São criadas quinze instâncias sintéticas, todas com relações de dependência entre os requisitos, sendo a menor com 100 clientes e 140 requisitos e as maiores respectivamente com 750 clientes e 3250 requisitos e 1000 clientes e 1500 requisitos. Tais instâncias foram utilizadas por três abordagens propostas para resolver o problema: métodos exatos, algoritmos gulosos e técnicas de busca local. Utilizando métodos exatos, um modelo de programação inteira e um algoritmo *Branch-And-Bound* foram aplicados a um modelo de programação linear da formulação do problema. Na abordagem de algoritmos gulosos, que foram desenvolvidos com o objetivo de gerar limites inferiores para a função objetivo em tempos de execução pequenos, a metaheurística GRASP é utilizada para melhorar os algoritmos gulosos que obtiveram os melhores resultados. Em relação às técnicas de busca local, as heurísticas Subida da Encosta (HC, do inglês *Hill Climbing*) e Arrefecimento Simulado (SA, do inglês *Simulated Annealing*) foram utilizadas com uma estrutura simples de vizinhança, com o objetivo de gerar soluções próximas do ótimo em tempos de execução pequenos. Como conclusão, os autores afirmam que os métodos exatos são suficientes para instâncias com poucos clientes e requisitos, porém para as instâncias maiores, o SA obteve os melhores resultados.

Com a disseminação e uso de algoritmos populacionais, surgiram trabalhos utilizando Algoritmos Genéticos (GA, do inglês *Genetic Algorithm*) e Otimização de Colônia de Formigas (ACO, do inglês *Ant Colonization Optimization*). De Souza et al. [7] propõem um algoritmo baseado na metaheurística ACO e comparam com um SA e um GA, utilizando apenas instâncias sintéticas. São realizadas três comparações entre os três algoritmos, onde cada comparação corresponde respectivamente a ACO, SA e GA com 1000 avaliações, ACO com 1000 e SA e GA com 10000 avaliações e, por fim, ACO com limitação de tempo e SA e GA com 1000 avaliações. Nas três comparações, ACO apresenta melhores resultados em quase todas as instâncias. No entanto, ACO apresenta, na média, tempos de execução até 60 vezes maiores que GA e até 90 vezes maiores que SA.

Também utilizando ACO, Jiang et al. [6] propõem a heurística HACO (Hibrid ACO), uma melhoria à metaheurística tradicional com a incorporação da informação sobre a vizinhança para orientar as escolhas de cada formiga. Além disso, as soluções geradas por esse ACO híbrido são processadas pela heurística HC com o objetivo de melhorar a qualidade. Para atestar a eficácia e eficiência do algoritmo proposto, foram utilizados para fins de comparação implementações de SA, GRASP e do próprio ACO sem a me-

lhoria proposta. Os resultados indicam que HACO supera todas as heurísticas usadas na comparação tanto em qualidade das soluções com em tempo de execução.

Utilizando uma formulação diferente, onde além das premissas para clientes e requisitos enunciadas na formulação clássica do NRP, cada requisito passa a ter uma importância relativa para cada cliente, que representa a relevância ou prioridade do requisito na perspectiva de cada cliente, Del Sagrado et al. [8] propõem resolver o problema considerando também vários tipos de interações entre os requisitos e não apenas as relações de precedência. Com a importância variando de acordo com cada cliente, cada requisito tem um valor agregado (ou "satisfação global", no termo usado pelos autores) para o projeto de software, que é o somatório, considerando todos os clientes, do produto entre o lucro associado ao cliente e a importância do requisito para este cliente. Como consequência da mudança na formulação, o objetivo que se quer maximizar passa a ser o somatório desses valores agregados de cada um dos requisitos e não mais o lucro total. A restrição de orçamento continua sendo a mesma da formulação clássica do NRP. No entanto, embora cada requisito inicie o projeto ou planejamento da versão de software com um custo de implementação, este pode variar se o requisito estiver em um tipo de interação com outros requisitos que aumente ou diminua seu custo caso um desses outros requisitos seja escolhido para a próxima versão. Outra categoria de interação entre requisitos funciona de forma similar, levando o valor agregado de um requisito, e não o custo, a aumentar ou diminuir. Os autores implementaram três heurísticas para comparação: ACO, GRASP e GA. Os resultados observados mostram que GRASP superou ACO, que superou GA tanto em tempo de execução quanto na qualidade das soluções, embora os autores afirmem que ACO pode gerar resultados melhores através de ajustes em seus parâmetros.

Com a preocupação em resolver instâncias realistas do NRP de grandes dimensões, justificada pelo crescimento em escala da quantidade de requisitos a serem gerenciados ser listado como um dos nove destaques no futuro da engenharia de requisitos, Xuan et al. [15] sustentam que abordagens para resolver diretamente o NRP, seja através de métodos exatos seja por heurísticas, não são escaláveis, ou seja, tais métodos até possuem um bom desempenho em instâncias de tamanhos pequenos, mas à medida que as instâncias tornam-se maiores, com mais clientes, mais requisitos e mais relações de dependência entre os requisitos, enfim mais próximas da realidade, tais métodos tendem a apresentar um desempenho degradado na qualidade das soluções geradas e no tempo de execução. Como alternativa, os autores propõem quebrar o problema maior, representado pela instância, em problemas menores, procedimento chamado de redução de instância, e de forma iterativa, chamada de multinível pelo autores, para depois refinar a solução para o problema original. Isto é feito pelo algoritmo proposto, o *Backbone-based Multilevel*

Algorithm (BMA). Baseados em trabalhos anteriores, os autores utilizam uma estrutura chamada *backbone*, que pode ser vista como a parte comum dentre as melhores soluções, que possui os clientes e seus requisitos. A heurística adotada para encontrar os ótimos locais nas instâncias reduzidas é SA. O BMA é comparado com uma implementação de GA e uma implementação de uma variante de SA, conhecida como Multistart Strategy-based SA, ou simplesmente MSSA. O BMA superou a implementação para o GA e para o MSSA, e hoje constitui a heurística estado-da-arte para o NRP, com valores de *fitness* próximos dos ótimos conhecidos. A propósito, Veerapen et al. calcularam os ótimos para as 39 instâncias do NRP utilizadas no presente trabalho através de métodos exatos [9].

2.4.2 Formulação Bi-objetivo do NRP

Pioneiro na formulação bi-objetivo para o NRP, o trabalho de Zhang et al. [54] é motivado pela constatação de que múltiplos objetivos, alguns dos quais conflitantes, tratados em uma única função objetivo através do uso de pesos para cada objetivo, levava a resultados equivocados e pouca informação de suporte à decisão a respeito de *trade-offs* entre os objetivos. Considerando o lucro gerado pela satisfação dos clientes e o custo de implementação dos requisitos como objetivos conflitantes a serem atingidos, os autores propõem a aplicação de quatro algoritmos para a geração da Curva ou Fronteira de Pareto: uma busca randômica, um GA simples com os dois objetivos reduzidos a uma única função através do uso de pesos, um GA adaptado para considerar os dois objetivos separadamente, chamado pelos autores de Pareto GA e o NSGA-II (do inglês "*Non-dominated Sorting Genetic Algorithm*"). Os resultados mostram que o NSGA-II superou o Pareto GA, o GA simples e a busca randômica. No caso de problemas multi-objetivo, superar uma heurística significa dizer que os resultados gerados na Fronteira de Pareto pelo algoritmo que obteve os melhores resultados dominaram os resultados daqueles algoritmos superados. Outra contribuição do trabalho é o fato de fornecer informações que respondem a perguntas relacionadas a *trade-offs* entre os objetivos como "Qual o aumento do custo total da versão de software caso o lucro seja aumentado em três vezes?".

Utilizando três metaheurísticas multi-objetivo, NSGA-II, MOCell (do inglês *Multi-Objective Cellular Genetic Algorithm*) e PAES (do inglês *Pareto Archived Evolution Strategy*), sobre seis instâncias sintéticas geradas através de sorteio de clientes e requisitos sem relações de dependência e sobre uma instância com dados reais fornecida pela Motorola, Durillo et al. [55] propõem um estudo para avaliar a qualidade das soluções usando dois indicadores: Hypervolume, um indicador que leva em conta a convergência em direção à Fronteira de Pareto ótima e a diversidade das soluções geradas, e Spread, um indicador que mede a distribuição das soluções em uma Fronteira de Pareto. Em relação ao indica-

dor Hypervolume, NSGA-II foi o algoritmo que apresentou os melhores resultados e foi o mais rápido na obtenção de um conjunto preciso de soluções, MOCell equiparou-se em eficácia ao NSGA-II quando o número de soluções aumentou e PAES apresentou o conjunto de soluções menos preciso. Em relação ao indicador Spread, MOCell apresentou as Fronteiras de Pareto com a melhor das soluções em todas as instâncias e PAES apresentou os piores resultados. Em síntese, NSGA-II obteve o maior número de soluções ótimas, MOCell obteve uma melhor distribuição das soluções e PAES obteve os menores tempos de execução, embora com resultados inferiores aos das outras duas metaheurísticas.

2.5 Considerações Finais

Neste capítulo, foi apresentada a fundamentação teórica para o desenvolvimento do presente trabalho de pesquisa. Como o trabalho tem o objetivo de resolver um problema de otimização combinatória através de métodos heurísticos, o primeiro tópico abordado foi descrever os conceitos e definições da metaheurística adotada,. Devido a necessidade de atribuir valores aos parâmetros da metaheurística que resultem em soluções eficazes e eficientes, decidiu-se que esses parâmetros deveriam ser automaticamente configurados, em um procedimento independente de conhecimento prévio tanto da metaheurística BRKGA quanto do problema abordado, o NRP. Portanto, no segundo tópico, foram apresentados o tema de configuração automática de parâmetros e uma ferramenta de software que implementa seus conceitos, técnicas e métodos, a ferramenta IRACE. Tal ferramenta foi escolhida para realizar a configuração dos parâmetros porque tem sido usada ultimamente em trabalhos de aplicação da metaheurística BRKGA, alguns deles de autoria do próprio autor da metaheurística. Adicionalmente, é apresentado o tópico sobre *K-Fold Cross Validation*, uma técnica usada para validação de modelos preditivos cujo objetivo é saber se as considerações sobre uma amostra feitas na etapa de treinamento, com dados conhecidos, são válidas para uma amostra de dados desconhecidos na etapa de teste, além de ser recomendada quando a quantidade de dados disponíveis para ambas as etapas é supostamente pequena a ponto de levar à condição indesejada de *overfitting*. Ainda neste capítulo, também foram apresentados alguns trabalhos relacionados com a intenção de situar o presente trabalho no panorama geral das pesquisas da área bem como dar respaldo aos objetivos e justificativas descritos nas Seções 1.3 e 1.4.

No próximo capítulo, as definições gerais apresentadas nos três tópicos foram adaptadas e implementadas para compor a solução proposta, que pode ser considerada uma customização da fundamentação teórica para o presente trabalho.

3. Proposta de Solução

Este capítulo apresenta em detalhes os três componentes da solução proposta: (i) o método construtivo proposto, implementado sobre um *framework* computacional para a metaheurística BRKGA fornecido por [56]; (ii) a implementação da técnica *K-Fold Cross Validation*; e (iii) a preparação (*setup*) da ferramenta de software IRACE para ser executada em associação com o *framework* computacional. Embora sejam apresentados separadamente, há comunicação entre os componentes quando executados. De acordo com qual estudo experimental é realizado, há três possibilidades de comunicação ou integração: (i) entre a ferramenta IRACE e o *framework* computacional; (ii) entre a implementação do *K-Fold Cross Validation* e o *framework* computacional; e (iii) entre a implementação do *K-Fold Cross Validation* e a ferramenta IRACE. Nas duas primeiras possibilidades, a comunicação é intermediada por um script batch do sistema operacional Windows; na terceira possibilidade, a comunicação é intermediada por meio de chamadas à função e acesso a estruturas de dados da linguagem R, já que ambos os componentes foram implementados nessa linguagem. Vale salientar que o *framework* computacional para a metaheurística BRKGA não foi desenvolvido por este trabalho de pesquisa.

3.1 O *Framework* Computacional para a Metaheurística BRKGA

De forma a implementar o *framework* conceitual da metaheurística BRKGA, um *framework* computacional desenvolvido na linguagem de programação C++ foi proposto pelos mesmos autores da metaheurística [56]. O *framework* contempla a parte independente do problema representada pela Figura 2.3, incluindo o gerenciamento da população e a dinâmica de evolução, deixando para o pesquisador que está aplicando a metaheurística BRKGA a um problema de otimização combinatória o trabalho de implementar a parte dependente do problema, que é converter um vetor de chaves aleatórias em uma solução, processo conhecido como decodificação.

O *framework* possui apenas duas classes e duas interfaces. As classes são: *Population*, que armazena e dispõe as informações sobre a população da geração corrente, tais como tamanho do cromossomo e da população, melhor *fitness* e *fitness* do i-ésimo cromossomo, além de operações como atribuir o *fitness* a um cromossomo e ordenar a população de acordo com o *fitness*; e *BRKGA*, que determina a dinâmica de evolução e o gerenciamento da população através de métodos para inicialização e evolução da população. As interfaces são: *Decoder*, que representa a parte dependente do problema e onde o pesquisador deve implementar o método que calcula o *fitness* usando os alelos (valores das chaves aleatórias) de cada cromossomo; e *RNG* (do inglês *Random Number Generator*), onde o pesquisador deve implementar um gerador de números pseudo-randômicos de sua escolha para ser usado no sorteio das chaves aleatórias e no sorteio dos cromossomos que farão o cruzamento para a geração de descendentes. No caso do presente trabalho, foi utilizado o gerador Mersenne Twister [57].

A classe *BRKGA*, responsável pelo *modus operandi* da metaheurística, realiza chamadas aos métodos de *Population*, *Decoder* e *RNG* para poder executar cada passo da parte independente do problema. A classe *Population* contém as estruturas de dados que representam a população e seus cromossomos e é instanciada pela classe *BRKGA* para reter informações da população da geração atual e da subsequente, portanto duas instâncias. Esse número aumenta de acordo com o número de populações independentes, de forma que se tenha duas instâncias por população independente. As implementações das interfaces *Decoder* e *RNG* (*NRP_Decoder* e *MTRand*) são instanciadas e passadas como parâmetros para a classe *BRKGA*. A Figura 3.1 exibe o diagrama de classes do *framework*.

Na prática, o *framework* inicia seu processamento com uma população contendo exatamente p cromossomos, cada cromossomo tendo n chaves aleatórias, e evolui essa população ao longo de um certo número de gerações até um critério de parada ser alcançado. Em uma dada geração k, o processo evolutivo tem os seguintes passos:

(1) Cada cromossomo é decodificado de forma a se obter seu *fitness*;

(2) A população é particionada em dois grupos: um conjunto de cromossomos do estrato elite contendo p_e cromossomos com os melhores *fitness* e outro conjunto com os demais cromossomos;

(3) A população da geração $k + 1$ terá p_e cromossomos do estrato elite da geração k, p_m cromossomos gerados randomicamente e p - p_e - p_m descendentes, que são cromossomos gerados pelo cruzamento entre um cromossomo do estrato elite e outro dentre os demais cromossomos, ambos escolhidos aleatoriamente.

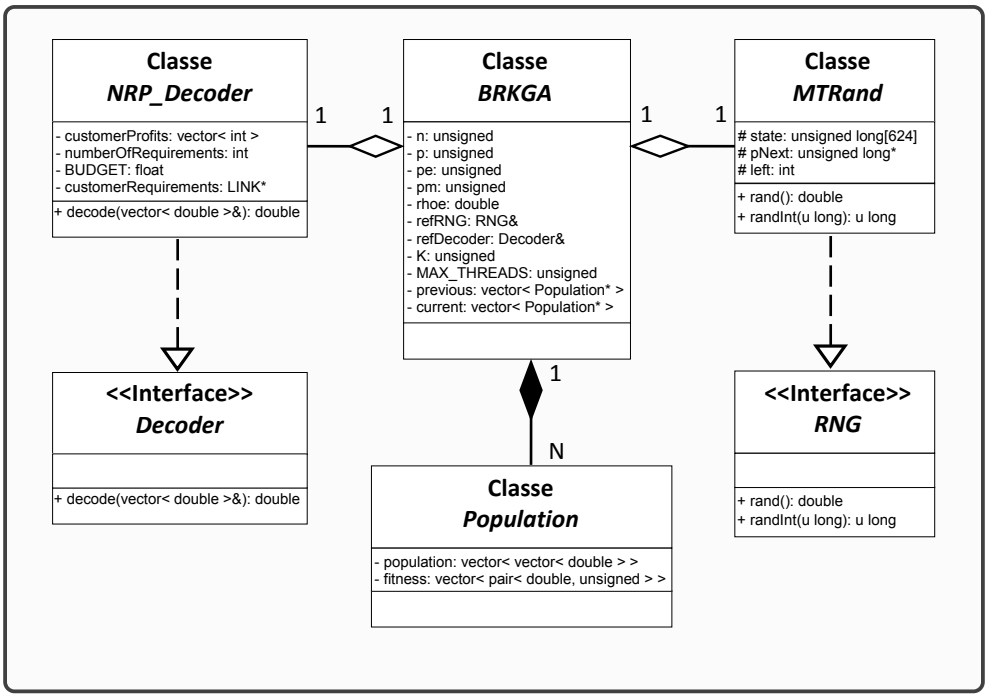

Figura 3.1: Diagrama de classes do *framework* do BRKGA.

3.1.1 Pré-processamento da Relação de Dependência entre Requisitos

Um dos motivos que aumentam a ordem de complexidade do algoritmo de um método construtivo para o problema do NRP é a dependência entre requisitos. Como visto na definição do NRP (Seção 1.2), essa dependência ocasiona uma relação de precedência entre os requisitos, de forma que para incluir um cliente como parte da solução não basta verificar se seus requisitos cabem no orçamento, mas sim se os custos de implementação de seus requisitos precedentes, dos precedentes dos precedentes e assim por diante, também cabem. As dependências são representadas por grafos unidirecionais não circulares, portanto não apresentam ciclos.

As instâncias do problema informam as relações de dependência entre os requisitos no formato de um vetor de duas dimensões (um vetor de vetores ou um vetor de listas), onde a primeira dimensão representa um requisito dependente e a segunda dimensão os requisitos precedentes deste. O número total de requisitos da instância determina a primeira dimensão do vetor, de tamanho fixo. A segunda dimensão do vetor varia para cada índice da primeira dimensão, segundo a quantidade de requisitos precedentes. Portanto, caso um requisito não tenha precedentes, seu índice de primeira dimensão não terá informações na segunda dimensão, uma vez que o respectivo vetor de requisitos precedentes está vazio. Outra observação importante é em relação aos níveis de dependência: o vetor de requisitos precedentes armazena apenas dependências de primeiro nível. Dependências de primeiro nível são aquelas em que entre o requisito dependente e o requisito precedente

não existe nenhum outro requisito. Já nas dependências de segundo nível, entre os requisitos dependente e precedente existe um requisito que precede o primeiro e depende do segundo, ou seja, são relações em que um determinado requisito depende de um requisito, que depende de outro. O mesmo raciocínio segue para os demais níveis.

Para os requisitos solicitados pelos clientes, um vetor de duas dimensões também é utilizado, em que a primeira dimensão representa os clientes e a segunda dimensão representa o conjunto de requisitos desejados. O número total de clientes da instância determina a primeira dimensão do vetor, de tamanho fixo. Assim como na estrutura de dados para a relação de dependência entre os requisitos, a segunda dimensão do vetor varia para cada índice da primeira dimensão, segundo a quantidade de requisitos desejados. No entanto, pelo fato de cada cliente sempre desejar algum requisito, sempre haverá índices de segunda dimensão para todo e qualquer índice de primeira dimensão. Um dado importante nessa representação é que não existe, dentre os requisitos desejados, requisitos duplicados nem requisitos que mantenham relação de dependência entre si em qualquer nível. Portanto, se em algum momento o cliente solicitou requisitos nessas duas condições, houve um refinamento antes das instâncias serem disponibilizadas por seus criadores. A Figura 3.2 ilustra a estrutura de dados de duas dimensões que representa os clientes e seus requisitos desejados.

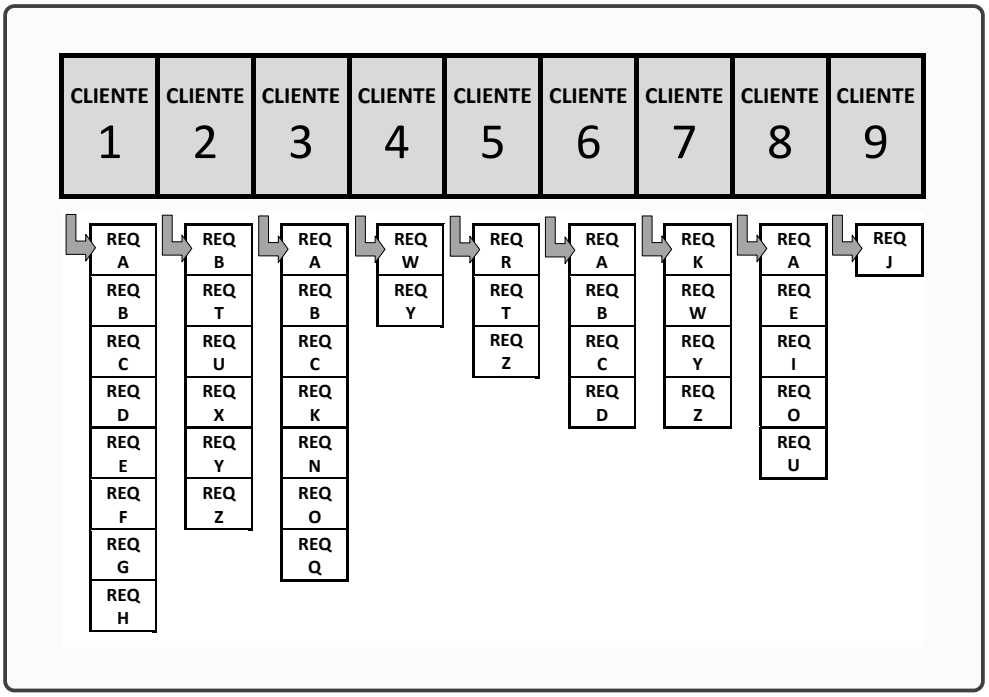

Figura 3.2: Estrutura de dados de duas dimensões para clientes e seus requisitos.

Percorrer tais estruturas de dados para o cálculo do custo de implementação dos requisitos, toda vez que se pretende incluir um cliente na solução, procedimento que é realizado para cada gene (um gene representa um cliente) de cada cromossomo da população em

cada uma das gerações, é uma ação que exige uma análise criteriosa a fim de agilizar o acesso ao custo de cada requisito. Com o objetivo de obter uma complexidade linear da ordem de $O(n)$ no melhor caso, quando todos os clientes desejam um único requisito, e $O(nm)$ no pior caso, quando todos os clientes desejam todos os requisitos, onde n e m representam respectivamente o número total de clientes e o número total de requisitos, foi criada uma única estrutura de dados consolidando as informações das duas estruturas de dados descritas acima, tanto a que representa as dependências entre os requisitos quanto a que representa os clientes e seus requisitos. Assim que os dados em arquivo das instâncias são armazenados em memória, um processamento é realizado sobre essas duas estruturas de forma a percorrer o grafo de dependências de cada requisito e associar para cada cliente não apenas seus requisitos desejados mas também todos os requisitos necessários para que os seus requisitos desejados possam ser implementados.

A nova estrutura de dados, novamente um vetor de duas dimensões, armazena as informações dos clientes (lucro associado), referenciadas pelo índice da primeira dimensão, e as informações tanto dos requisitos (custo de implementação) necessários para a implementação dos requisitos desejados como dos próprios requisitos desejados, referenciadas pelo índice da segunda dimensão. Dessa forma, o acesso aos dados dos requisitos de cada cliente é mais rápido nas avaliações de *fitness* de cada cromossomo ao longo das gerações.

3.1.2 O Método Construtivo

Basicamente, todas as abordagens para resolver problemas de otimização combinatória podem ser caracterizadas como métodos de busca. Nesse contexto, pode-se definir mecanismos genéricos aplicáveis a uma vasta gama de algoritmos de busca. Tais algoritmos são categorizados como algoritmos de busca sistemática e algoritmos de busca heurística [58]. Os algoritmos ou métodos da primeira categoria têm como objetivo encontrar a solução ótima. Muitas vezes, quando o método exato não consegue chegar até o fim devido ao tamanho da instância e/ou problemas de memória, alguns métodos exatos também são capazes de fornecer um limite inferior ou superior, dependendo se é um problema de minimização ou maximização. Tais limites são muito importantes na avaliação de heurísticas, pois fornecem uma medida para determinar a qualidade da solução gerada pelo método heurístico. Já os algoritmos ou métodos de busca heurística têm como objetivos encontrar soluções de qualidade em tempo razoável. Os métodos de busca heurística são classificados como métodos construtivos, métodos de busca local e as metaheurísticas. Em geral, métodos construtivos constroem, de forma determinística, uma única solução para o problema. É possível usar um mecanismo aleatório para introduzir diversidade no método construtivo e assim gerar mais de uma solução. Em buscas locais e heurísticas

baseadas em metaheurísticas, a execução é baseada em gerar e avaliar soluções candidatas de forma iterativa, eventualmente também com o uso de componentes aleatórios. No presente trabalho, os métodos utilizados foram a metaheurística e o método construtivo.

Embora o *framework* computacional do BRKGA forneça as funcionalidades da metaheurística relacionadas ao seu *modus operandi*, principalmente a dinâmica de evolução da população, é necessária a sua adequação ao problema de otimização combinatória que se quer resolver. Tal adequação é realizada na parte do *framework* dependente do problema. Na prática, a adequação consiste em tomar um cromossomo com suas chaves aleatórias e contextualizar estas informações como algo útil ou adequado ao problema. De modo geral, esta tradução ou decodificação do cromossomo consiste em utilizar as chaves aleatórias para encontrar um agrupamento, uma ordenação ou uma atribuição de um conjunto discreto e finito de valores segundo as condições definidas pelo problema de otimização. No caso do NRP, o objetivo é encontrar um agrupamento de clientes, não importando a ordem, cujos requisitos irão compor a próxima versão de software, de forma que o lucro obtido com tal agrupamento seja o maior possível.

No presente trabalho, o decodificador é representado pelo método construtivo descrito no Algoritmo 1. É neste ponto da heurística onde a pesquisa pode expressar inovação e criatividade na proposição de novas formas de abordar o problema. Com a intenção de diminuir o tempo de processamento na busca pela solução, novas ideias algorítmicas são propostas para tornar mais simples e direta a avaliação das soluções candidatas. Como se está lidando com a natureza combinatória do problema, além de bons algoritmos, deve-se priorizar a utilização de estruturas de dados que favoreçam acesso rápido às informações, como aquelas estruturas que utilizam índices e que possuam o menor custo possível de ordenação.

A fim de tornar simples a leitura e compreensão do algoritmo proposto, foram utilizadas sete funções para representar de forma resumida uma operação sobre algum dado, nomeadas como: *Ordena*, *ObtemTamanho*, *ObtemRequisitos*, *VerificaVisitado*, *ObtemCusto*, *MarcaVisitado* e *ObtemLucro*. Quando implementadas em código, boa parte dessas funções podem ser substituídas por funções nativas da linguagem de programação utilizada e operações de leitura e escrita em estruturas de dados com acesso direto através de índice, como listas, vetores e matrizes. A propósito, uma estrutura de dados desse tipo foi utilizada, qual seja *listaRequisitosCliente*. As operações de atribuição de valor são representadas pelo símbolo "$\leftarrow$". As operações aritméticas e de comparação, como igual, maior, menor e suas combinações, são expressas sem alteração, e valores booleanos são expressos como "verdadeiro"e "falso".

Algoritmo 1: Método construtivo para o NRP.

```
   Entrada: cromossomo
   Saída: lucro
1  início
2      Ordena (cromossomo)
3      numClientes ← ObtemTamanho (cromossomo)
4      custoTotal ← 0
5      para i ← 1 .. numClientes faça
6          listaRequisitosCliente ← ObtemRequisitos (i, cromossomo)
7          numRequisitos ← ObtemTamanho (listaRequisitosCliente)
8          custoRequisitosCliente ← 0
9          para j ← 1 .. numRequisitos faça
10             requisito ← listaRequisitosCliente[ j ]
11             se VerificaVisitado (requisito) = falso então
12                 custo ← ObtemCusto (requisito)
13                 custoRequisitosCliente ← custoRequisitosCliente + custo
14                 se custoTotal + custoRequisitosCliente ≥ BUDGET então
15                     Interrompe o para .. faça
16                 fim
17             fim
18         fim
19         se custoTotal + custoRequisitosCliente ≤ BUDGET então
20             MarcaVisitado (listaRequisitosCliente)
21             custoTotal ← custoTotal + custoRequisitosCliente
22             lucroCliente ← lucroCliente + ObtemLucro (i)
23         fim
24     fim
25 fim
```

Com a intenção de deixar claro o custo computacional do método construtivo, são descritos os custos em termos de complexidade algorítmica de cada uma das sete funções. Considerando n o número de clientes, tem-se: *Ordena* com $O(n \log n)$, pois usa um algoritmo de ordenação; *ObtemTamanho* com $O(1)$, pois consulta uma meta-informação da estrutura de dados (no caso, o tamanho); *ObtemRequisitos* com $O(1)$, pois acessa a informação diretamente pelo índice do cliente; *VerificaVisitado* com $O(1)$, pois consulta a informação diretamente pelo índice do requisito; *ObtemCusto* com $O(1)$, pois acessa a informação diretamente pelo índice do requisito; *MarcaVisitado* com $O(1)$, pois registra a informação diretamente pelo índice do requisito, e; *ObtemLucro* com $O(1)$, pois acessa a informação diretamente pelo índice do cliente. Portanto, com exceção da função *Ordena*, as funções possuem complexidade de ordem constante. Por fim, dois laços de repetição aninhados foram utilizados ("**para** .. **faça**"). Neste trecho do método construtivo, a complexidade do algoritmo é $O(nm)$, onde n e m representam respectivamente o número de clientes e o número de requisitos.

O algoritmo proposto ordena os clientes baseado nas chaves aleatórias atribuídas a cada um deles pela metaheurística, com clientes e respectivas chaves aleatórias representados pela estrutura de dados *cromossomo* (linha 2). Respeitando essa ordem (linhas 3 e 5), o algoritmo obtém os requisitos desejados pelo cliente, assim como os seus requisitos precedentes em todos os níveis de dependência (vide Seção 3.1.1), através da função *ObtemRequisitos* (linha 6) e calcula o custo para implementar os requisitos de cada cliente (linhas 12 e 13) através da função *ObtemCusto*, desconsiderando neste cálculo aqueles requisitos associados a clientes antecessores na ordenação e já contemplados (linha 11) e que, portanto, já fazem parte da solução candidata.

A cada requisito, o método construtivo verifica se seu custo cabe no orçamento (linha 14). Caso afirmativo, segue para o próximo requisito do cliente em avaliação ou, caso não haja mais requisitos para o cliente, segue para o próximo cliente, atualizando o repositório de requisitos visitados, o custo total dos requisitos até agora implementados e o lucro da solução, valor este que se busca maximizar (linhas 20, 21 e 22). Caso não caiba no orçamento, segue para o próximo cliente (linha 15), pois há a possibilidade de que ainda haja clientes na ordem informada pela metaheurística cujo conjunto de requisitos, alguns dos quais já implementados por estarem associados a clientes já incluídos na solução, caiba no orçamento. Esse procedimento de avaliação segue até o último cliente na ordem informada.

3.2 Implementação da Técnica *K-Fold Cross Validation*

A técnica *K-Fold Cross Validation* foi utilizada em dois experimentos realizados para encontrar a melhor configuração de parâmetros da metaheurística BRKGA quando aplicada ao NRP: um experimento de configuração automática de parâmetros com o uso da ferramenta IRACE e outro, também para determinar a melhor configuração de parâmetros, sem o uso da ferramenta IRACE. A melhor configuração obtida no experimento com a ferramenta IRACE foi utilizada no experimento final, contra o qual foram comparados os melhores resultados da literatura. Como inicialmente a técnica seria usada apenas no experimento com a ferramenta IRACE, ferramenta desenvolvida e executada no ambiente da linguagem R, optou-se por adotar essa linguagem para implementar o *K-Fold Cross Validation*. Mais tarde, foi identificada a necessidade de realizar o experimento sem o uso da ferramenta IRACE, o que não representou um problema do ponto de vista de integração entre a implementação da técnica (linguagem R) e o arquivo executável do *framework* computacional, uma vez que a linguagem R fornece funções para executar programas externamente ao ambiente R.

K-Fold Cross Validation consiste em particionar o conjunto P de todas as instâncias em K subconjuntos de mesmo tamanho, quando K é um divisor do número total de instâncias, ou em K-1 subconjuntos de mesmo tamanho e um subconjunto com as instâncias restantes (cuja quantidade é igual ao resto da divisão do número total de instância pelo valor de K), quando K não é um divisor do número total de instâncias. As partições são representadas por P_1, P_2, ..., P_K. Em cada momento $t \in \{1, 2, ..., K\}$, as instâncias de treinamento utilizadas pertencerão ao subconjunto $P \setminus P_t$ e as de teste ao subconjunto P_t (vide Figura 2.7 e Figura 3.3 para exemplos). No presente trabalho, os valores de K foram determinados a partir da escolha do número de instâncias por partição, que por sua vez foi baseado no número total de instâncias utilizadas. Com o objetivo de reduzir a variância dos valores gerados na configuração automática de parâmetros e se certificar que a ferramenta de configuração convergiria para valores similares em diferentes execuções de *K-Fold Cross Validation*, com K variando, foram escolhidos vários valores para o número de instâncias por partição e, a partir de cada um desses valores, K foi calculado. O critério para a escolha desses valores foi que uma dada partição não poderia ter uma única instância, com a justificativa de que só haveria uma única instância para validação, e nem poderia ter aproximadamente metade do número total de instâncias, com a justificativa de que o número de instâncias de treinamento seria aproximadamente o mesmo das instâncias de validação. O Algoritmo 2 apresenta o cálculo do valor de K a partir do número de instâncias por partição e foi utilizado na implementação do *K-Fold Cross Validation* na linguaguem R. O código-fonte dos scripts na linguagem R pode ser visto no apêndice A.

Com base no critério para a escolha do número de instâncias por partição explicado no parágrafo anterior, os valores escolhidos foram 3, 5, 7, 9, 11, 13 e 15. A razão para iniciar com o número 3 é por representar um número pequeno de instâncias bem como ser um divisor de 39, o número total de instâncias. Já a razão para ser uma sequência de números ímpares é pelo simples fato de que faria pouca diferença ter quantidades tão próximas nas partições, como 3 e 4 ou 4 e 5 ou 5 e 6 e assim por diante em uma sequência de números inteiros consecutivos. Por fim, a razão para a sequência terminar com o número 15 é que o número de instâncias de treinamento seria 60% maior que o de instâncias de teste (24 instâncias contra 15, pois são duas partições com 15 instâncias e uma com 9), um percentual favorável em se tratando de evitar a situação indesejada de mesmo número de instâncias de treinamento e teste. Se a sequência terminasse em 17 e 19, esses percentuais seriam respectivamente 30% (22 instâncias contra 17, pois são duas partições com 17 instâncias e uma com 5) e 5% (20 instâncias contra 19, pois são duas partições com 19 instâncias e uma com 1). É possível ainda perceber que a partir de 21, quando haveria apenas duas partições, uma de duas possibilidades de *2-Fold Cross Validation* sempre levaria a mais instâncias de teste que de treinamento, o que também seria indesejado.

Algoritmo 2: Cálculo do valor de K e atribuição das instâncias às partições.

Entrada: Sequência de valores representando o nº de instâncias por partição;
Conjunto de todas as instâncias.
Saída: Partições com as instâncias sorteadas.

```
1  início
2      para cada nº de instâncias por partição da sequência faça
3          Divida o nº total de instâncias pelo nº de instâncias por partição
4          se resto da divisão = 0 então
5              nº de partições = quociente da divisão
6          senão
7              nº de partições = 1 + quociente da divisão
8          fim
9          Crie tantos conjuntos vazios quanto for o nº de partições
10         para cada um dos conjuntos vazios faça
11             se resto da divisão = 0 então
12                 nº de instâncias para sorteio = nº de instâncias por partição
13             senão
14                 se o conjunto vazio corrente for o último da sequência então
15                     nº de instâncias para sorteio = resto da divisão
16                 senão
17                     nº de instâncias para sorteio = nº de instâncias por partição
18                 fim
19             fim
20             Sorteie as instâncias dentre aquelas que ainda não foram sorteadas
                segundo o nº de instâncias para sorteio e uma distribuição uniforme
21             Atribua as instâncias sorteadas ao conjunto vazio corrente
22         fim
23     fim
24 fim
```

A Tabela 3.1 exibe os valores de K calculados a partir do número de instâncias por partição. Tomando como exemplo cinco instâncias por partição, tem-se o valor de K igual a 8: sete partições com cinco instâncias e uma partição com quatro instâncias. Em relação à coluna "Instâncias de treinamento/teste", ainda com o mesmo exemplo, em sete casos ou execuções do *K-Fold Cross Validation* (execuções t $\in$ {1, 2, ..., 7}) serão 34 instâncias de treinamento e cinco instâncias de teste, e em um caso (t = 8) serão 35 instâncias de treinamento e quatro instâncias de teste. A soma dos valores entre parênteses deve ser igual ao valor de K. Os valores entre parênteses representam a quantidade de casos (ou situações ou execuções ou iterações do *K-Fold Cross Validation*) em que serão utilizadas o respectivo número de instâncias de treinamento e de teste. A Figura 3.3 ilustra o exemplo.

Tabela 3.1: K-Fold Cross Validation: Valores de K utilizados.

Instâncias por partição	Total de instâncias	K	Instâncias de treinamento/teste
3	39	13	36/3 (13 casos)
5	39	8	34/5 (7 casos) e 35/4 (1 caso)
7	39	6	32/7 (5 casos) e 35/2 (1 caso)
9	39	5	30/9 (4 casos) e 36/3 (1 caso)
11	39	4	28/11 (3 casos) e 33/6 (1 caso)
13	39	3	26/13 (3 casos)
15	39	3	24/15 (2 casos) e 30/9 (1 caso)

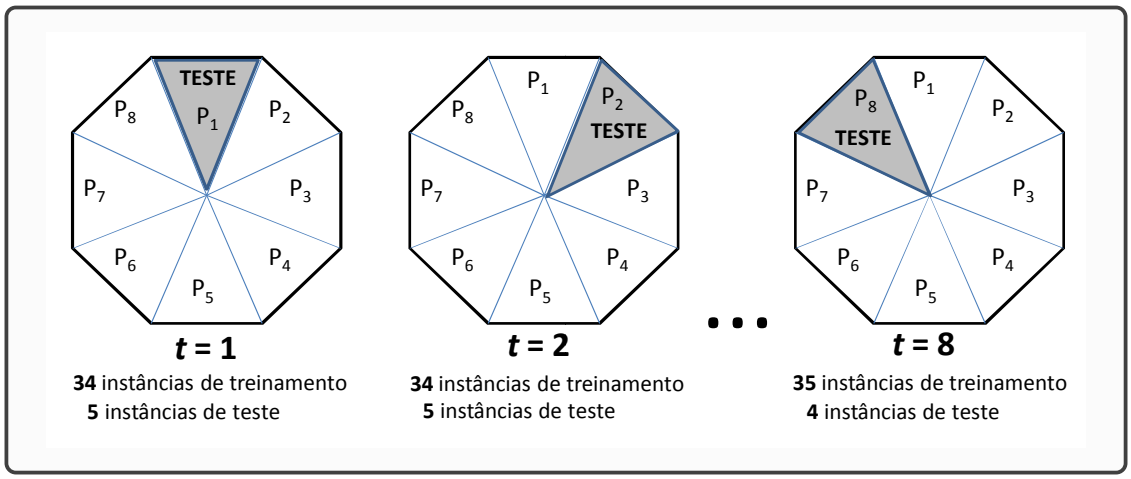

Figura 3.3: K-Fold Cross Validation: Exemplo com K=8 e 39 instâncias.

3.3 Preparação da Ferramenta IRACE

Dada a flexibilidade da ferramenta de software IRACE em suportar diferentes cenários de configuração automática de parâmetros, tais como configuração para problemas mono-objetivo e de problemas multi-objetivo, que faz uso tanto de métrica unária de qualidade (como *hypervolume*) quanto da soma ponderada das variáveis que se quer otimizar, dependência entre os parâmetros do algoritmo ou metaheurística, quando certos valores de um parâmetro determinam quais valores são permitidos para um outro parâmetro (e em muitos casos, até se este parâmetro será utilizado), suporte a quatro tipos de parâmetros (números reais e inteiros, valores ordinais e cardinais), independência da linguagem de implementação do algoritmo ou metaheurística, condição de parada do procedimento de configuração automática baseada ou no número de execuções ou no tempo total de execução da metaheurística, possibilidade de realizar a otimização ou do valor retornado pela função objetivo (*fitness*) ou do tempo de execução do algoritmo ou metaheurística, escolha dentre quatro tipos de testes estatísticos, é de se esperar que um esforço de configuração de ambiente, que inclui customizações de arquivos de configuração e de ajustes em variáveis internas da ferramenta IRACE, venha na contrapartida dessa flexibilidade.

Cada um dos recursos da ferramenta IRACE citados acima que caracteriza sua flexibilidade só é possível graças à atribuição de valores a variáveis internas através de arquivos de configuração ou através de linha de comando do ambiente R e graças ao uso de uma rotina executável que realiza a intermediação entre a ferramenta IRACE e o programa executável do algoritmo ou metaheurística cujos parâmetros serão configurados. Para apoiar a execução da ferramenta IRACE no presente trabalho, foram utilizados quatro arquivos do tipo texto:

- um arquivo para definir quais parâmetros da metaheurística serão configurados, com informações como o tipo e o domínio de valores de cada parâmetro;
- um arquivo para realizar a atribuição de valores às variáveis internas, em especial o número máximo de execuções do programa executável da metaheurística necessárias para realizar todo o processos de configuração automática de parâmetros, o tipo de teste utilizado nos testes estatísticos sequenciais e o número de casas decimais considerado no sorteio dos parâmetros do tipo real;
- um arquivo com os nomes das instâncias de treinamento, na ordem em que as instâncias foram sorteadas quando na criação das partições;
- um arquivo com os nomes das instâncias de teste, na ordem em que as instâncias foram sorteadas quando na criação das partições.

Por fim, foi criado um script batch do sistema operacional Windows para promover a integração entre a ferramenta IRACE, implementado em R, e o programa executável da metaheurística, implementado em C++. Logo, ao todo foram necessários cinco artefatos para apoiar a execução da ferramenta IRACE.

Conforme a Tabela 3.1, a soma de todos os valores de K utilizados na solução proposta resulta em 42 casos. A configuração automática de parâmetros ocorre para cada caso de modo independente dos demais e o que caracteriza cada caso, portanto cada execução independente da configuração automática, distinguindo-o dos demais é o arranjo das 39 instâncias em instâncias de treinamento e de teste. Esse arranjo é realizado pela técnica *K-Fold Cross Validation* e representado nos arquivos com os nomes das instâncias de treinamento e de teste. Portanto, desses cinco artefatos utilizados em cada uma das 42 execuções independentes da ferramenta IRACE, apenas o arquivo com os nomes das instâncias de treinamento e o arquivo com os nomes das instâncias de teste possuem seu conteúdo modificado de execução para execução.

3.4 Considerações Finais

Neste capítulo, foi apresentada a proposta de solução para o presente trabalho de pesquisa. A descrição da proposta evidencia a integração entre os três componentes da solução, a saber: (i) a implementação executável da heurística proposta nomeada BRKGA_NRP, baseada na metaheurística BRKGA e desenvolvida utilizando o *framework* computacional fornecido pelos autores da metaheurística; (ii) a ferramenta IRACE e todos os seus artefatos customizáveis, necessários para a sua comunicação com a heurística proposta; e (iii) a implementação do *K-Fold Cross Validation* na linguagem R. Cada um dos componentes foi descrito em detalhes, em especial aqueles que envolveram lógica de programação, de forma que os algoritmos foram explicados em seus trechos mais importantes.

No próximo capítulo, a solução proposta é aplicada sobre as instâncias escolhidas do NRP em experimentos computacionais. Os resultados da execução da solução proposta são avaliados e comparados com a heurística estado-da-arte para o NRP.

4. Avaliação Experimental

Neste capítulo, são apresentados os resultados dos experimentos computacionais realizados com a heurística proposta BRKGA_NRP baseada na metaheurística BRKGA aplicada ao problema NRP. Antes da execução dos experimentos que comparam a abordagem proposta com o estado da arte, foi realizado um estudo de convergência com o objetivo de escolher os melhores valores para os parâmetros da metaheurística tamanho da população e número de gerações, que consistiu em analisar a convergência da heurística proposta para diferentes valores desses parâmetros.

Uma vez definido o melhor valor para cada um dos dois parâmetros, um experimento foi realizado utilizando a ferramenta de configuração automática de parâmetros IRACE (Seção 4.4.1). Tal experimento consistiu em assumir como fixos os valores definidos no estudo de convergência e realizar a configuração para outros três parâmetros da metaheurística, a saber: percentual de cromossomos do estrato elite, percentual de mutantes introduzidos a cada nova geração e a probabilidade de que um gene do cromossomo do indivíduo do estrato elite seja herdado por seu descendente na próxima geração quando no cruzamento.

Definidos os valores para os cinco parâmetros de configuração, os experimentos computacionais (Seção 4.4) que compararam BRKGA_NRP com o estado da arte foram realizados e seus resultados comparados com os melhores resultados da heurística estado da arte. Adicionalmente, foi realizado um experimento para justificar o uso da ferramenta IRACE (Seção 4.4.2), que consistiu em utilizar um padrão aleatório, seguindo uma distribuição de probabilidade uniforme, para definir os valores dos parâmetros da metaheurística.

4.1 Questões de Pesquisa

QP1. A heurística proposta no presente trabalho supera a heurística que representa o estado-da-arte para o NRP mono-objetivo em termos de qualidade das soluções?

Até o presente momento, a heurística BMA [15] foi a técnica heurística que produziu os melhores resultados em termos de qualidade de soluções para as instâncias utilizadas neste trabalho. Por utilizar uma ferramenta para configuração automática de parâmetros associada com uma técnica para evitar *overfitting* e fazer uso eficaz do pequeno número de instâncias disponíveis para a etapa de treinamento (*K-Fold Cross Validation*), espera-se que a heurística proposta supere o BMA.

QP2. A combinação de valores dos parâmetros da metaheurística obtida pela configuração automática de parâmetros com o uso da ferramenta IRACE proporciona melhores resultados quando comparados com os resultados obtidos com o uso de um padrão aleatório?

A resposta a esta questão justificará ou não o uso da ferramenta IRACE para a configuração automática de parâmetros. A ferramenta IRACE utiliza uma técnica baseada em testes estatísticos sequenciais e atualização das distribuições de probabilidade dos parâmetros a serem configurados ao longo da execução da configuração. Entretanto, não é garantido que ela gere uma combinação de valores que, aplicada nos experimentos computacionais, produzirá resultados melhores quando comparados com os resultados obtidos sem o uso da ferramenta IRACE. Para confrontar com os resultados obtidos com o uso da ferramenta IRACE, uma combinação de valores obtida através de um procedimento aleatório, rigorosamente com a mesma quantidade de execuções e o mesmo número de valores sorteados pela ferramenta IRACE, será gerada e aplicada na heurística proposta segundo os mesmos passos dos experimentos computacionais. Espera-se que os resultados obtidos com a combinação de valores obtida com o uso da ferramenta IRACE superem aqueles obtidos com a combinação de valores obtida através do padrão aleatório.

QP3. Quão distantes são os resultados da heurística proposta no presente trabalho e a solução ótima de cada instância?

Em uma publicação do ano de 2015, Veerapen et al. [9] propuseram uma abordagem utilizando programação linear inteira (portanto um método exato) para resolver todas as instâncias do NRP utilizadas neste trabalho. Esta questão de pesquisa atesta a qualidade das soluções geradas pela heurística proposta em relação aos ótimos de cada instância reportados no trabalho de Veerapen et al.

4.2 Instâncias do Problema

O presente trabalho utiliza 39 instâncias. Destas, 15 instâncias pertencem ao conjunto de instâncias denominadas como clássicas e as demais 24 instâncias pertencem ao conjunto de instâncias denominadas como realistas. Todas as instâncias são caracterizadas pelo número total de requisitos, o custo de cada requisito, o número de requisitos por cliente, o número total de clientes e o lucro associado a cada cliente. As instâncias clássicas possuem relações de dependência entre os requisitos, enquanto que as instâncias realistas não possuem. As relações de dependência entre os requisitos são caracterizadas pelo número de níveis de dependência, o número de requisitos por nível (cuja soma, considerando todos os níveis, resulta no número total de requisitos) e o número máximo de requisitos dependentes que um único requisito de um dado nível pode ter. É assumido que os requisitos que não dependem de nenhum outro requisito estão no nível 1 e aqueles que não possuem dependentes estão no último nível. A seguir, é apresentada uma descrição detalhada de cada conjunto de instâncias.

As instâncias clássicas foram criadas por Xuan et al. [15], baseando-se no trabalho de Bagnall [5]. Tais instâncias são divididas em cinco grupos, onde cada grupo possui três instâncias. O que diferencia cada instância dentro do grupo é o orçamento para implementar os requisitos: a primeira instância do grupo possui um orçamento correspondente a 30% do custo de implementar todos os requisitos, a segunda, 50% e a terceira, o orçamento de 70%. A Tabela 4.1 descreve os cinco grupos do conjunto de instâncias clássicas. O nome do grupo é exibido na coluna “Grupo”. O nome da instância propriamente dita deriva do nome do grupo, bastando adicionar o percentual correspondente ao orçamento. Por exemplo, para o grupo “nrp-1”, tem-se as instâncias “nrp-1-30”, “nrp-1-50” e “nrp-1-70”. As colunas “Reqs./Nível” (requisitos por nível), “Custo Req.” (custo dos requisitos) e “Depend.” (número máximo de requisitos dependentes que um único requisito de um dado nível pode ter) exibem informações segregadas por nível. As informações de cada nível estão separadas pelo caractere “/” (“barra”). Os valores situados na extrema esquerda correspondem ao nível 1, seguindo até a extrema direita, que correspondem ao último nível, que, a depender da instância, pode ser o nível 3 ou o nível 5. As colunas “Reqs./Nível”, “Reqs./Cli.” (número de requisitos por cliente) e “Lucro” (lucro associado a cada cliente) representam as informações usando a notação de intervalo, que significa que qualquer valor inteiro deste intervalo é válido. Por último, a coluna “Custo Tot.” representa a soma dos custos de todos os requisitos, valor a partir do qual será calculado o orçamento de cada instância ao aplicar os três percentuais, quais sejam, 30%, 50% e 70%.

Tabela 4.1: Instâncias clássicas.

Grupo	Reqs./Nível	Custo Req.	Depend.	Reqs./Cli.	Clientes	Lucro	Custo Tot.
nrp-1	20/40/80	[1,5]/[2,8]/[5,10]	8/2/0	[1,5]	100	[10,50]	857
nrp-2	20/40/80/ 160/320	[1,5]/[2,7]/[3,9]/ [4,10]/[5,15]	8/6/4/ 2/0	[1,5]	500	[10,50]	5.048
nrp-3	250/500/750	[1,5]/[2,8]/[5,10]	8/2/0	[1,5]	500	[10,50]	8.870
nrp-4	250/500/750/ 1000/750	[1,5]/[2,7]/[3,9]/ [4,10]/[5,15]	8/6/4/ 2/0	[1,5]	750	[10,50]	22.161
nrp-5	500/500/500	[1,3]/2/[3,5]	4/4/0	[1,5]	1000	[10,50]	3.992

Como um exemplo, tome-se o grupo “nrp-2”. Suas três instâncias “nrp-2-30”, “nrp-2-50” e “nrp-2-70” terão um orçamento respectivamente de 1514,4, 2524,0 e 3533,6. Os custos dos requisitos do primeiro nível variam de 1 a 5 unidades, do segundo nível, de 2 a 7 unidades, do terceiro, de 3 a 9 unidades, do quarto, de 4 a 10 unidades e do quinto e último nível, de 5 a 15 unidades. Um requisito do primeiro nível tem no máximo oito dependentes, do segundo nível, no máximo seis, do terceiro nível, quatro dependentes e do quarto nível, dois dependentes. Existem 500 clientes em cada uma das instâncias do grupo, cada um demandando de um a cinco requisitos e com um lucro associado de 10 a 50 unidades. As instâncias realistas foram criadas por Xuan et al. [15] a partir de repositórios de erros de três projetos de software livre. Um registro de erro e um usuário no repositório equivalem, respectivamente, a um requisito e um cliente de uma instância do NRP. Os requisitos por cliente e o custo do requisito em uma instância do NRP foram inspirados, respectivamente, nos comentários de usuário no registro de erros, em que cada comentário equivale a uma solicitação de requisito, e na gravidade do erro.

O conjunto de instâncias realistas é dividido em 12 grupos, onde cada grupo possui duas instâncias. De forma similar às instâncias clássicas, o que diferencia cada instância dentro do grupo é o orçamento para implementar os requisitos. Dessa vez, apenas os percentuais de 30% e de 50% do custo de implementar todos os requisitos (coluna “Custo Tot.”) são considerados. A Tabela 4.2 mostra detalhes de cada grupo de instâncias. Como não existem requisitos dependentes, as colunas “Reqs./Nível” e “Depend.” estão ausentes. As instâncias de um mesmo grupo compartilham as mesmas características, diferindo apenas no orçamento disponível para a implementação dos seus requisitos.

Tabela 4.2: Instâncias realistas.

Grupo	Reqs.	Custo Req.	Reqs./Cli.	Clientes	Lucro	Custo Tot.
nrp-e1	3.502	[1,7]	[4,20]	536	[10,50]	13.150
nrp-e2	4.254	[1,7]	[5,30]	491	[10,50]	15.928
nrp-e3	2.844	[1,7]	[4,15]	456	[10,50]	10.399
nrp-e4	3.186	[1,7]	[5,20]	399	[10,50]	11.699
nrp-g1	2.690	[1,7]	[4,20]	445	[10,50]	13.277
nrp-g2	2.650	[1,7]	[5,30]	315	[10,50]	12.626
nrp-g3	2.512	[1,7]	[4,15]	423	[10,50]	12.258
nrp-g4	2.246	[1,7]	[5,20]	294	[10,50]	10.700
nrp-m1	4.060	[1,7]	[4,20]	768	[10,50]	15.741
nrp-m2	4.368	[1,7]	[5,30]	617	[10,50]	16.997
nrp-m3	3.566	[1,7]	[4,15]	765	[10,50]	13.800
nrp-m4	3.643	[1,7]	[5,20]	568	[10,50]	14.194

4.3 Definição dos Parâmetros Tamanho da População e Número de Gerações

Nesta Seção, é descrito como os valores para os parâmetros da metaheurística tamanho da população e número de gerações foram obtidos. Para tanto, foi realizado um estudo de convergência. Tal estudo deixou evidente a influência desses dois parâmetros da metaheurística na qualidade das soluções geradas e no tempo de execução do algoritmo. Em resumo, quanto maiores os valores para esses parâmetros, melhor a qualidade das soluções e maior o tempo de execução. A questão central do estudo de convergência foi decidir quais valores adotar de forma a ter soluções de boa qualidade sem comprometer demasiadamente o tempo de execução.

O objetivo do estudo de convergência foi entender o comportamento assintótico da heurística proposta, de forma a determinar valores para os parâmetros tamanho da população e número de gerações. A decisão de fixar tais parâmetros em vez de submetê-los à configuração automática veio da constatação de que quanto maiores os valores desses parâmetros, melhor a qualidade das soluções geradas (*fitness*), porém na contrapartida de maiores tempos de execução. Antes de fixá-los, foram escolhidas algumas opções de valores para ambos os parâmetros, na suposição de que tais valores levariam a resultados próximos aos obtidos pela heurística estado da arte do NRP, o que foi confirmado.

Uma versão modificada do *framework* computacional para a metaheurística BRKGA foi desenvolvida para gravação em arquivo do melhor *fitness* de cada uma das 300 gerações para populações de cinco tamanhos diferentes: 100, 150, 200, 250 e 300 indivíduos. Dessa forma, foi possível acompanhar a evolução da heurística da primeira à última geração. Cada instância foi executada 30 vezes de forma independente, ou seja, uma semente

aleatória diferente foi gerada aleatoriamente para o gerador de números pseudo-aleatórios para cada uma das execuções. Também foi desenvolvido um programa na linguagem R para a leitura dos dados gerados, processamento e construção de gráficos para auxiliar na escolha dos parâmetros. Ao todo, foram gerados 39 gráficos, cada um correspondendo a uma instância. Vale salientar que cada valor de *fitness* em um gráfico é a média do resultado das 30 execuções.

As Figuras 4.1 a 4.5 mostram os gráficos gerados para todas as instâncias. Cada gráfico possui cinco curvas correspondentes aos cinco tamanhos de população empregados no estudo e devidamente representados em uma legenda. O eixo das abcissas representa a geração no processo evolutivo da heurística, cujos valores variam de 1 (primeira geração) a 300 (última geração). O eixo das ordenadas representa o *fitness*. De modo a facilitar a leitura do gráfico, nenhum valor é exibido ao longo do segmento de reta que representa o eixo das ordenadas. Em vez disso, uma sequência de sete valores, exibida ao lado direito da legenda da população, mostra os valores de *fitness* para as gerações primeira, 25ª, 50ª, 75ª, 100ª, 200ª e, por fim, 300ª. Em todos os gráficos, é possível perceber um crescimento acentuado da curva até um ponto em que começa a apresentar um crescimento lento e gradual até a melhor solução obtida na última geração. Como esperado, quanto maior o tamanho da população maiores são os *fitness* para a mesma geração. Tal fato é representado em cada gráfico com as curvas relativas a tamanhos de população maiores localizadas sempre acima das curvas relativas a tamanhos de população menores, com exceção da instância "nrp-1-70", onde as curvas se igualam.

Antes de definir o tamanho da população, foi definido o número de gerações para cada instância. Com a inspeção visual dos gráficos, foram definidos os valores de 125, 100 e 75 para o número de gerações, de acordo com a instância. Para as instâncias com orçamento de 30% do custo para implementar todos os requisitos (instâncias cujo nome termina em "-30"), foi definido o valor de 125 gerações. Para as instâncias com orçamento de 50% (instâncias cujo nome termina em "-50"), o valor definido foi de 100 gerações. Para as instâncias com orçamento de 70% (instâncias cujo nome termina em "-70"), o valor foi de 75 gerações. A justificativa para esta decisão é que as curvas do gráfico de cada instância adquirem um comportamento assintótico a partir desses valores, com um crescimento lento e gradual até o melhor *fitness* alcançado na última geração. Nas gerações anteriores a esses três valores, dependendo da instância, cada curva apresenta um crescimento rápido, demonstrando uma diferença relevante a cada geração no desenrolar do processo evolutivo. Portanto, adotar um valor menor que os escolhidos resultaria na má utilização da influência do parâmetro número de gerações na obtenção de soluções de boa qualidade, com o possível desperdício de um potencial para *fitness* melhores.

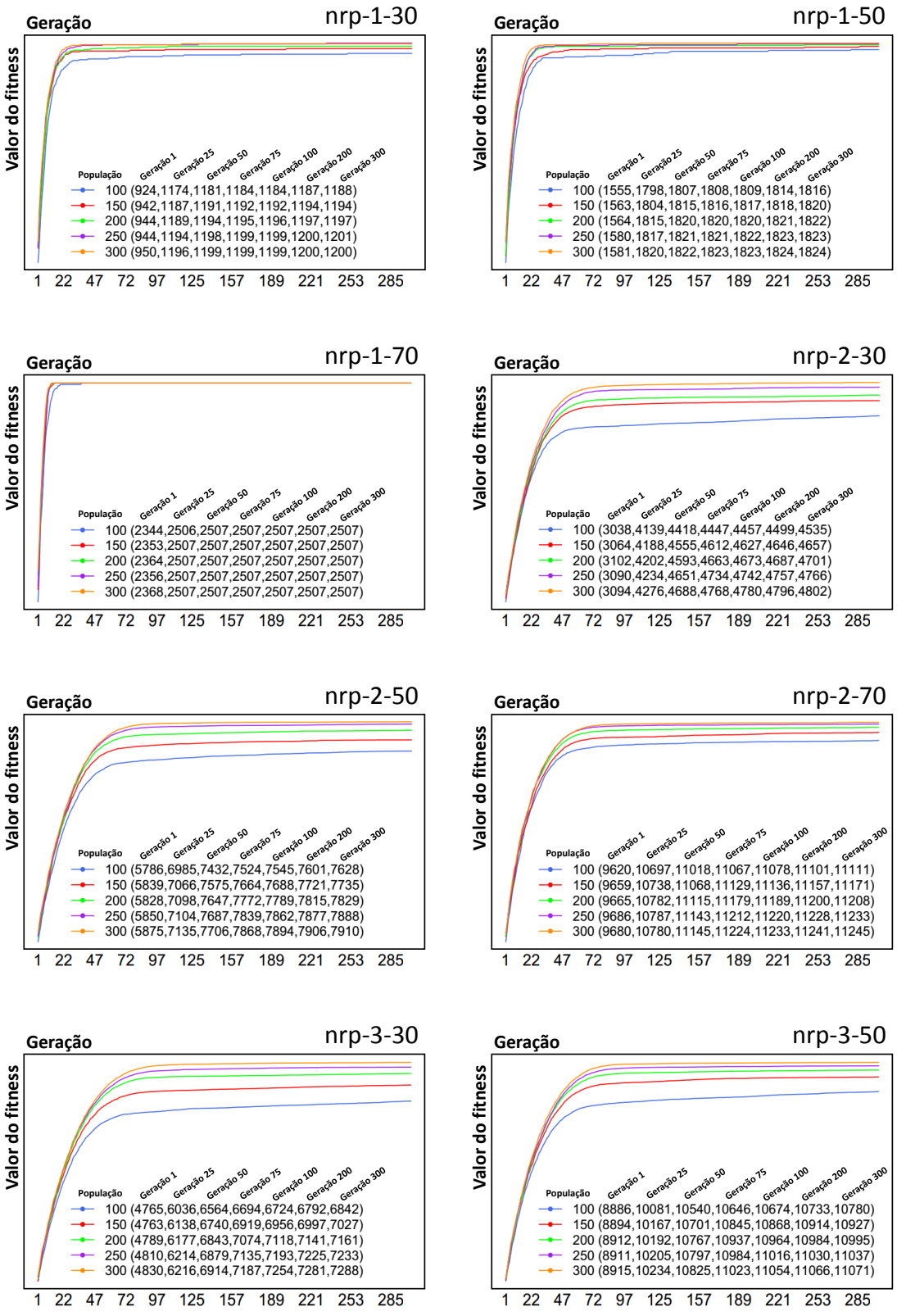

Figura 4.1: Estudo de convergência: instâncias “nrp-1-30” a “nrp-3-50”.

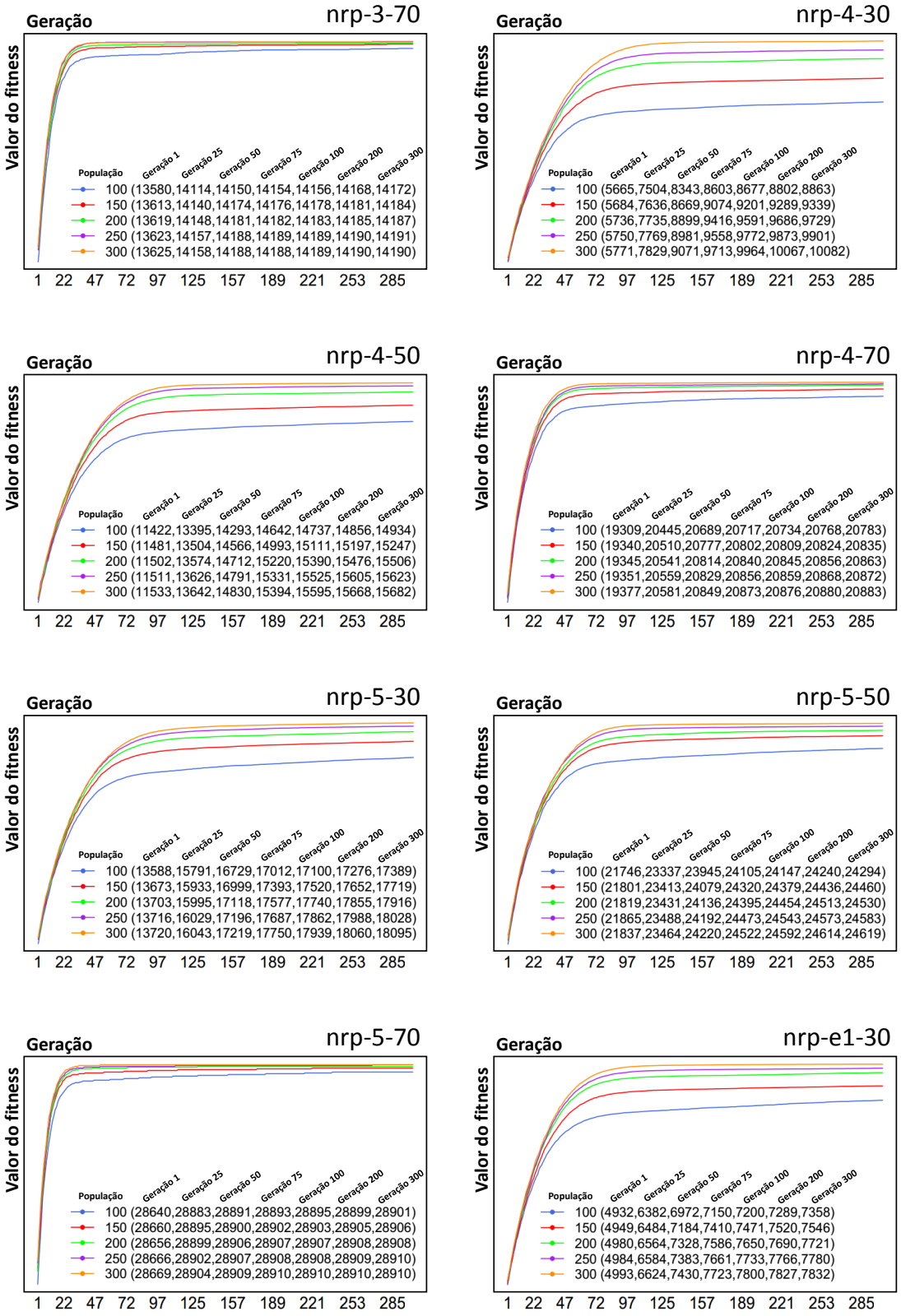

Figura 4.2: Estudo de convergência: instâncias “nrp-3-70” a “nrp-e1-30”.

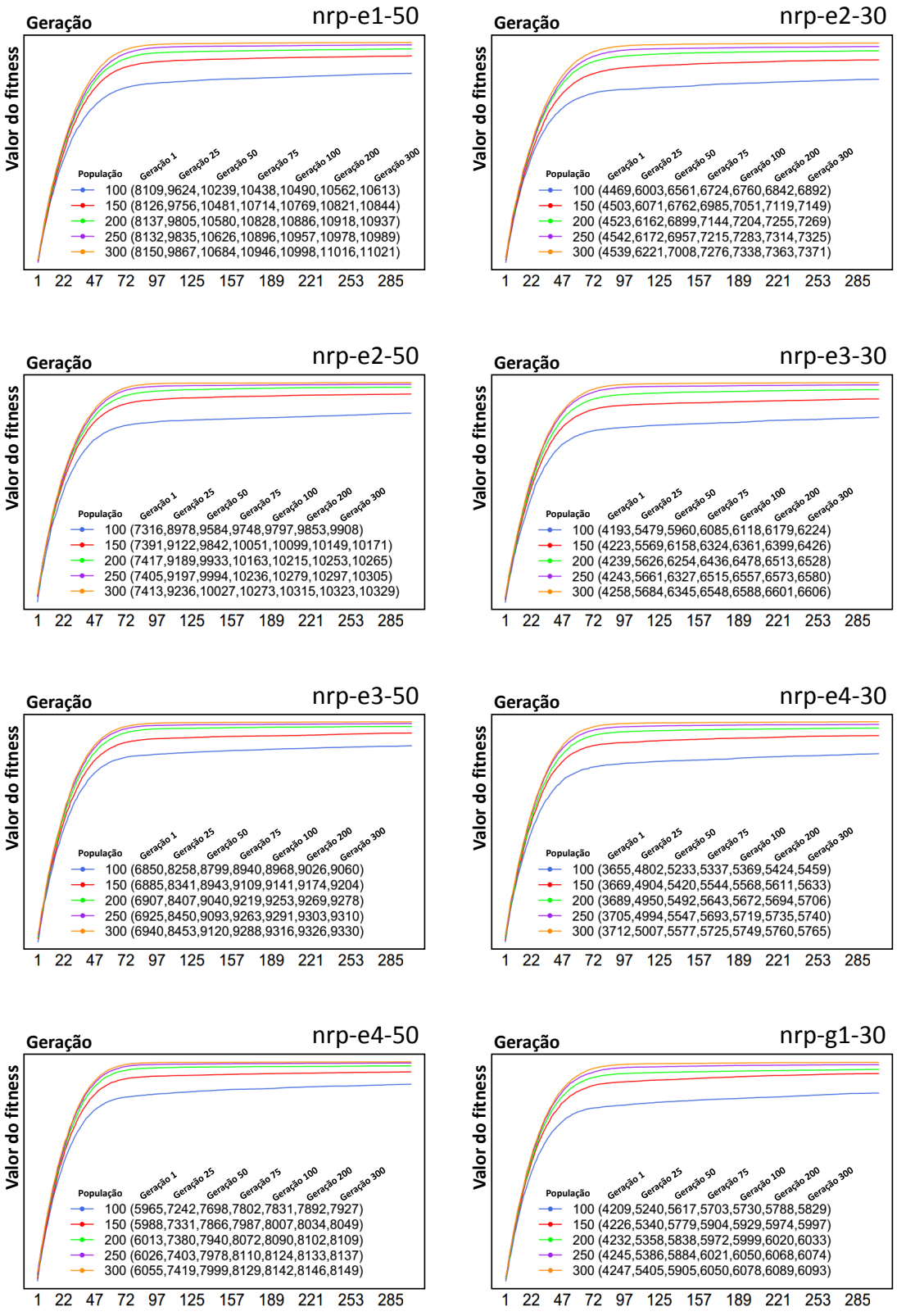

Figura 4.3: Estudo de convergência: instâncias “nrp-e1-50” a “nrp-g1-30”.

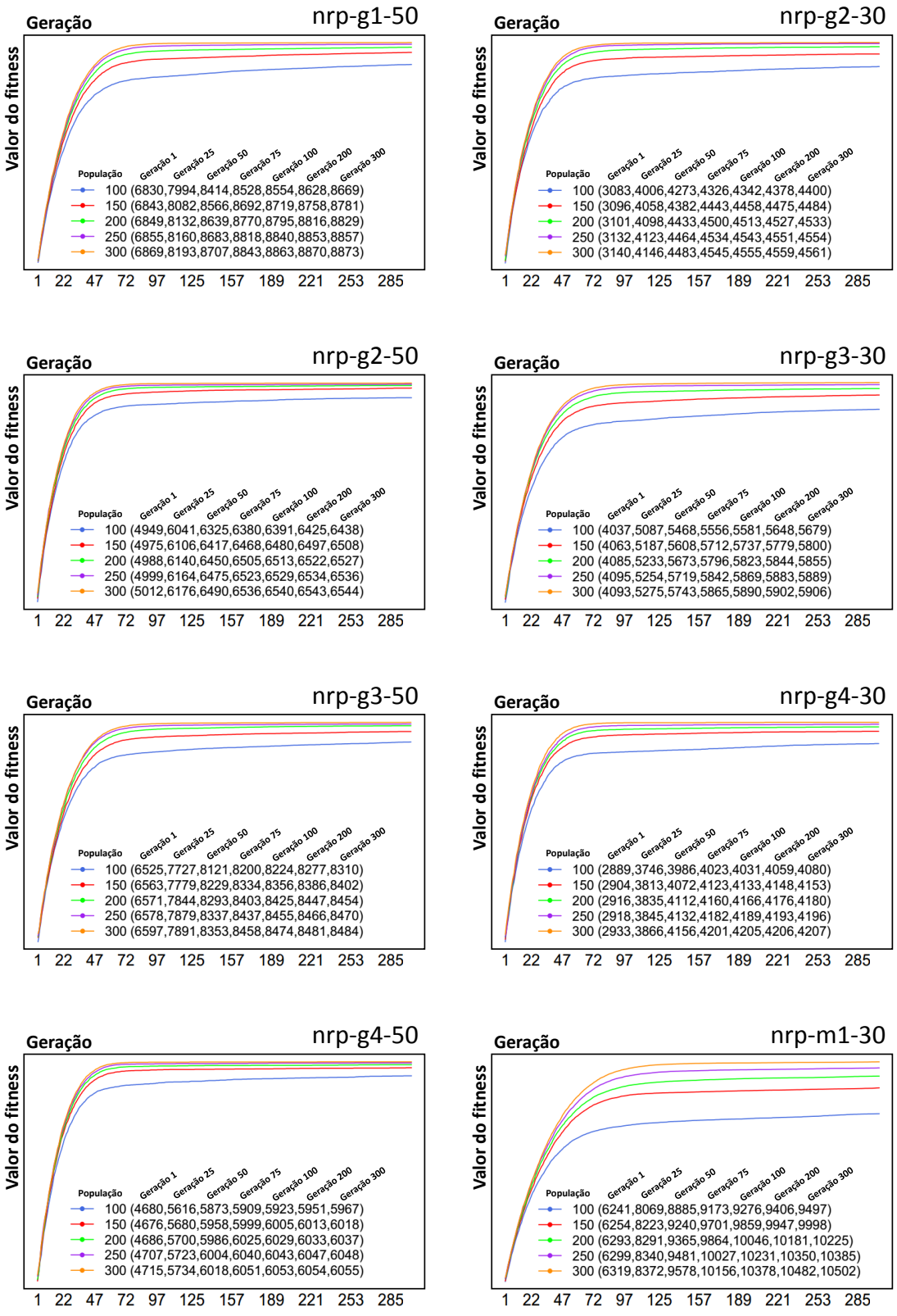

Figura 4.4: Estudo de convergência: instâncias “nrp-g1-50” a “nrp-m1-30”.

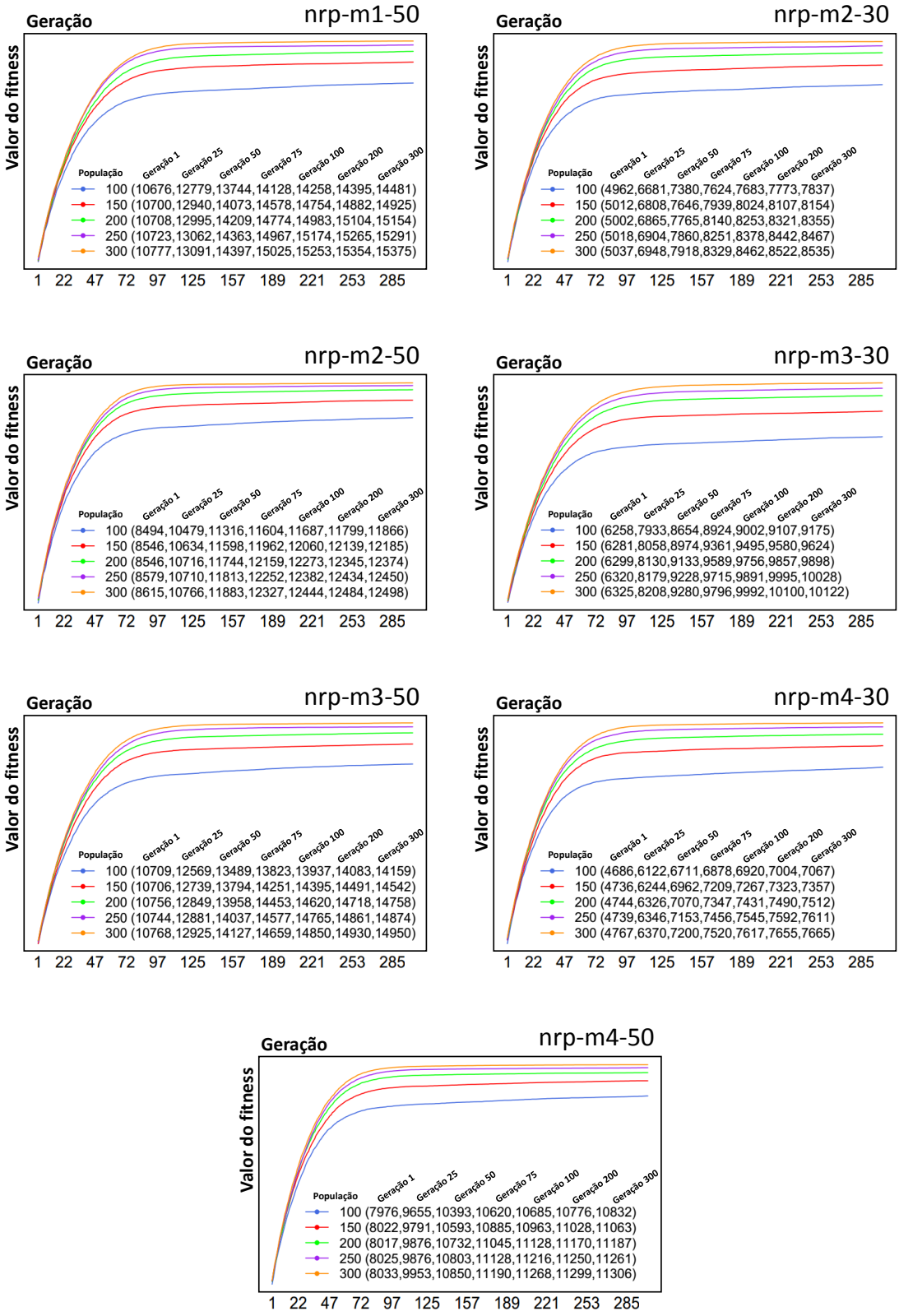

Figura 4.5: Estudo de convergência: instâncias “nrp-m1-50” a “nrp-m4-50”.

O tamanho da população foi definido a partir dos valores do número de gerações já escolhidos. A Tabela 4.3 foi elaborada com os valores utilizados na construção dos gráficos. Considerando que cada gráfico possui cinco curvas e que cada curva corresponde a um diferente tamanho da população, a coluna "Fitness" exibe o *fitness* obtido na geração 125 para as instâncias cujo nome termina em "-30", na geração 100 para as instâncias cujo nome termina em "-50" e na geração 75 para as instâncias cujo nome termina em "-70". A coluna "Ganho" informa o quanto o *fitness* aumentou em termos percentuais em relação ao *fitness* obtido com o tamanho da população imediatamente à esquerda. No gráfico, essa coluna representa a distância vertical entre as duas curvas correspondentes aos dois tamanhos da população (a razão multiplicada por 100 entre a distância vertical e o *fitness* da população imediatamente à esquerda). Vale salientar que os valores de *fitness* de ambos os tamanhos da população foram obtidos na mesma geração, uma vez que a comparação entre esses valores é realizada para a mesma instância.

Para todas as instâncias de uma forma geral, a partir da população de tamanho 200, esses ganhos começam a ficar cada vez menores, sugerindo que as curvas estão se aproximando entre si mais lentamente. Por exemplo, a média dos ganhos percentuais do tamanho da população 150 em relação ao tamanho da população 100 é de 2,58%, do tamanho da população 200 em relação ao tamanho da população 150 é de 1,31%, do tamanho da população 250 em relação ao tamanho da população 200 é de 0,74% e do tamanho da população 300 em relação ao tamanho da população 250 é de 0,47%. Como pouco ganho há de se obter em populações de tamanho maior que 200 e como o aumento do tamanho da população faz com que o tempo de execução aumente, o tamanho da população foi estabelecido como 200.

Estabelecidos os valores para os parâmetros tamanho da população e número de gerações, a próxima tarefa será determinar através de configuração automática os valores para os parâmetros percentual de cromossomos do estrato elite, percentual de mutantes introduzidos a cada nova geração e a probabilidade de que um gene do cromossomo do indivíduo do estrato elite seja herdado por seu descendente na próxima geração quando no cruzamento. Cabe aqui uma observação: para o estudo de convergência, estes três parâmetros tiveram valores fixos baseados no trabalho de Toso et al. [56], autores do *framework* computacional, que utilizaram em seus experimentos os valores 15% (0,15), 10% (0,10) e 0,70 respectivamente para os três parâmetros. Estes valores têm respaldo no trabalho que apresentou a metaheurística BRKGA para a comunidade acadêmica [14], posto que foram utilizados pelos autores para avaliar a eficácia e eficiência da metaheurística em alguns problemas de otimização combinatória.

Tabela 4.3: Estudo de convergência: ganho ao longo das gerações.

	TAMANHO DA POPULAÇÃO								
	100	150		200		250		300	
	Fitness	Fitness	Ganho	Fitness	Ganho	Fitness	Ganho	Fitness	Ganho
nrp-1-30	1.186	1.193	0,59%	1.197	0,34%	1.199	0,17%	1.200	0,08%
nrp-1-50	1.809	1.817	0,44%	1.820	0,17%	1.822	0,11%	1.823	0,05%
nrp-1-70	2.507	2.507	0,00%	2.507	0,00%	2.507	0,00%	2.507	0,00%
nrp-2-30	4.471	4.634	3,65%	4.679	0,97%	4.746	1,43%	4.786	0,84%
nrp-2-50	7.545	7.688	1,90%	7.789	1,31%	7.862	0,94%	7.894	0,41%
nrp-2-70	11.067	11.129	0,56%	11.179	0,45%	11.212	0,30%	11.224	0,11%
nrp-3-30	6.755	6.968	3,15%	7.127	2,28%	7.206	1,11%	7.267	0,85%
nrp-3-50	10.674	10.868	1,82%	10.964	0,88%	11.016	0,47%	11.054	0,34%
nrp-3-70	14.154	14.176	0,16%	14.182	0,04%	14.189	0,05%	14.188	-0,01%
nrp-4-30	8.718	9.241	6,00%	9.650	4,43%	9.835	1,92%	10.037	2,05%
nrp-4-50	14.737	15.111	2,54%	15.390	1,85%	15.525	0,88%	15.595	0,45%
nrp-4-70	20.717	20.802	0,41%	20.840	0,18%	20.856	0,08%	20.873	0,08%
nrp-5-30	17.158	17.570	2,40%	17.793	1,27%	17.925	0,74%	18.001	0,42%
nrp-5-50	24.147	24.379	0,96%	24.454	0,31%	24.543	0,36%	24.592	0,20%
nrp-5-70	28.893	28.902	0,03%	28.907	0,02%	28.908	0,00%	28.910	0,01%
nrp-e1-30	7.225	7.491	3,68%	7.670	2,39%	7.750	1,04%	7.816	0,85%
nrp-e1-50	10.490	10.769	2,66%	10.886	1,09%	10.957	0,65%	10.998	0,37%
nrp-e2-30	6.786	7.076	4,27%	7.228	2,15%	7.297	0,95%	7.351	0,74%
nrp-e2-50	9.797	10.099	3,08%	10.215	1,15%	10.279	0,63%	10.315	0,35%
nrp-e3-30	6.138	6.373	3,83%	6.492	1,87%	6.564	1,11%	6.594	0,46%
nrp-e3-50	8.968	9.141	1,93%	9.253	1,23%	9.291	0,41%	9.316	0,27%
nrp-e4-30	5.386	5.584	3,68%	5.681	1,74%	5.726	0,79%	5.754	0,49%
nrp-e4-50	7.831	8.007	2,25%	8.090	1,04%	8.124	0,42%	8.142	0,22%
nrp-g1-30	5.750	5.942	3,34%	6.007	1,09%	6.055	0,80%	6.083	0,46%
nrp-g1-50	8.554	8.719	1,93%	8.795	0,87%	8.840	0,51%	8.863	0,26%
nrp-g2-30	4.353	4.463	2,53%	4.518	1,23%	4.547	0,64%	4.557	0,22%
nrp-g2-50	6.391	6.480	1,39%	6.513	0,51%	6.529	0,25%	6.540	0,17%
nrp-g3-30	5.601	5.749	2,64%	5.830	1,41%	5.877	0,81%	5.895	0,31%
nrp-g3-50	8.224	8.356	1,61%	8.425	0,83%	8.455	0,36%	8.474	0,22%
nrp-g4-30	4.038	4.137	2,45%	4.169	0,77%	4.190	0,50%	4.205	0,36%
nrp-g4-50	5.923	6.005	1,38%	6.029	0,40%	6.043	0,23%	6.053	0,17%
nrp-m1-30	9.329	9.899	6,11%	10.112	2,15%	10.304	1,90%	10.449	1,41%
nrp-m1-50	14.258	14.754	3,48%	14.983	1,55%	15.174	1,27%	15.253	0,52%
nrp-m2-30	7.713	8.056	4,45%	8.286	2,86%	8.417	1,58%	8.499	0,97%
nrp-m2-50	11.687	12.060	3,19%	12.273	1,77%	12.382	0,89%	12.444	0,50%
nrp-m3-30	9.044	9.534	5,42%	9.816	2,96%	9.951	1,38%	10.058	1,08%
nrp-m3-50	13.937	14.395	3,29%	14.620	1,56%	14.765	0,99%	14.850	0,58%
nrp-m4-30	6.946	7.284	4,87%	7.460	2,42%	7.570	1,47%	7.637	0,89%
nrp-m4-50	10.685	10.963	2,60%	11.128	1,51%	11.216	0,79%	11.268	0,46%

4.4 Experimentos Computacionais

Após a definição dos valores para os parâmetro tamanho da população e número de gerações, foi realizado um experimento para a definição dos valores dos demais parâmetros da metaheurística: percentual de cromossomos do estrato elite, percentual de mutantes introduzidos a cada nova geração e a probabilidade de que um gene do cromossomo do indivíduo do estrato elite seja herdado por seu descendente na próxima geração quando no cruzamento. Esse experimento foi realizado com o suporte da técnica *K-Fold Cross Validation* associada à ferramenta de configuração automática de parâmetros IRACE.

Semelhante ao experimento acima descrito, um outro experimento foi realizado com a intenção de validar o uso da ferramenta IRACE. Utilizando a mesma implementação da técnica *K-Fold Cross Validation*, porém sem a ferramenta IRACE, várias combinações de valores para os três parâmetros da metaheurística foram sorteadas seguindo uma distribuição de probabilidade uniforme e aplicadas na heurística proposta respeitando o mesmo número de execuções do segundo experimento.

Definidos os valores para os cinco parâmetros da metaheurística, a heurística proposta foi executada 30 vezes para cada uma das 39 instâncias com o objetivo de medir o *fitness*. Para garantir reprodutibilidade, a cada uma das 30 execuções foi associada uma semente aleatória diferente, totalizando 30 sementes utilizadas por todas as instâncias. Por exemplo, a semente associada à quinta execução de uma dada instância é a mesma da quinta execução das demais instâncias. A execução dos experimentos foi realizada tanto com os valores dos parâmetros obtidos com a ferramenta IRACE (Seção 4.4.1) como com os valores obtidos com o sorteio segundo um padrão aleatório (Seção 4.4.2). Para cada uma das 39 instâncias, foram obtidos a mediana, a média e o desvio padrão dos valores de *fitness* gerados pelas 30 execuções.

Os experimentos foram realizados em vários ambientes computacionais, porém similares em configuração. Foram utilizadas máquinas Intel Core i7-2600 3,40 GHz com Windows 7 Professional 64 bits 2009 SP1, Intel Core i7-3770 3,40 GHz com Windows 8.1 Professional 64 bits 2013 x64 based e Intel Core i7 L640 2,13 GHz com Windows 7 Professional 64 bits 2009 SP1, todas com 4,00 GB de memória RAM.

4.4.1 Experimento de BRKGA_NRP com o Uso da Ferramenta IRACE

Neste experimento, foi utilizada a ferramenta para configuração automática de parâmetros IRACE em associação com a técnica *K-Fold Cross Validation*. Conforme descrito da Seção 2.2.1, há a necessidade de informar à ferramenta IRACE o conjunto de valores

para os parâmetros que serão configurados (vide Figura 2.6). Para números reais e inteiros, esse conjunto pode ser informado como um intervalo de valores. Com os intervalos, a ferramenta IRACE cria as distribuições de probabilidade para cada parâmetro, a partir das quais os sorteios serão realizados. Para este experimento, os limites inferior e superior dos intervalos foram escolhidos com base em alguns trabalhos que têm entre seus participantes um dos autores da metaheurística BRKGA. A seguir, os valores utilizados nesses trabalhos para os parâmetros percentual de cromossomos do estrato elite (p_e), percentual de mutantes introduzidos a cada nova geração (p_m) e a probabilidade de que um gene do cromossomo do indivíduo do estrato elite seja herdado por seu descendente na próxima geração quando no cruzamento (ρ_e): em [59], Stefanello et al. utilizaram $p_e \in$ [0,10, 0,30], $p_m \in$ [0,05, 0,30] e $\rho_e \in$ [0,5, 0,8]; em [60], Andrade et al. adotaram $p_e \in$ [0,15, 0,30], $p_m \in$ [0,10, 0,20] e $\rho_e \in$ [0,5, 0,8]; em [61], de Andrade et al. utilizaram $p_e \in$ [0,10, 0,30], $p_m \in$ [0,05, 0,20] e $\rho_e \in$ [0,5, 0,8]. Para considerar tais valores, foram escolhidos os intervalos $p_e \in$ [0,10, 0,30], $p_m \in$ [0,05, 0,30] e $\rho_e \in$ [0,5, 0,8].

Para que a ferramenta IRACE possa definir uma ordem de melhores e piores configurações através de testes estatísticos e ao final de sua execução chegar à configuração que apresentou os melhores resultados, a heurística proposta deve entregar para a ferramenta IRACE um valor comparável e independente das características das diferentes instâncias utilizadas (passos "c)" e "d)" na Figura 2.6). Usar simplesmente o *fitness* é uma escolha inadequada, pois dada as diferenças de orçamento, de dependências entre os requisitos, de requisitos solicitados por cada um dos clientes e de lucro total (soma dos lucros de todos os clientes da instância) que existem entre as 39 instâncias, tal valor só poderia ser utilizado pela ferramenta IRACE quando na comparação com execuções da heurística sobre a mesma instância. Para resolver essa questão, a métrica proposta no presente trabalho é a distância relativa entre o lucro na solução encontrada ($\mathrm{lucro}_{\mathrm{SOL}}$) e a soma dos lucros de todos os clientes ($\mathrm{lucro}_{\mathrm{TOTAL}}$), soma esta que se encontra bem acima do ótimo, dada as restrições de orçamento para a implementação dos requisitos. A fórmula $\mathrm{fitness}_{\mathrm{IRACE}}$ (Equação 4.1) representa a métrica utilizada pela ferramenta IRACE. Dessa forma, seja qual for a ordem de grandeza do lucro encontrado para uma instância em particular, o valor entregue à ferramenta IRACE é relativo e portanto, comparável. Como a ferramenta IRACE trabalha com minimização da função objetivo, a fórmula está adequada.

$$\mathrm{fitness}_{\mathrm{IRACE}} = \frac{\mathrm{lucro}_{\mathrm{TOTAL}} - \mathrm{lucro}_{\mathrm{SOL}}}{\mathrm{lucro}_{\mathrm{TOTAL}}} \tag{4.1}$$

Conforme a Tabela 3.1 na Seção 3.2 do Capítulo 3, onde a proposta de solução utilizando *K-Fold Cross Validation* é apresentada, foram utilizados sete valores para K, de acordo com o número de instâncias por partição: 13, 8, 6, 5, 4, 3 e 3, calculados respectivamente a partir de 3, 5, 7, 9, 11, 13 e 15 instâncias por partição. Como ilustra a Figura 3.3, o valor de K determina o valor de t, que é o momento de execução de *K-Fold Cross Validation*, com t $\in$ {1, 2, ..., K}, ou seja, se são K partições, serão K execuções no total, onde cada execução utiliza K-1 partições para a etapa de treinamento e 1 partição para a etapa de teste. Como a solução proposta faz uso de *K-Fold Cross Validation* associado à ferramenta IRACE, com o primeiro acionando o segundo para a realização das etapas de treinamento e teste, ambos compartilham a mesma execução. Portanto, o número de execuções | t | de *K-Fold Cross Validation* é exatamente o mesmo número de execuções independentes da ferramenta IRACE. Por independente, entende-se uma execução que não dependa de qualquer informação proveniente de alguma execução anterior e nem tenha que gerar informações para serem utilizadas em uma execução subsequente. O código-fonte na linguagem R da implementação de *K-Fold Cross Validation* associado à ferramenta IRACE pode ser visto no apêndice A.1.

De acordo com a soma dos valores de K descritos na Tabela 3.1, foram realizadas 42 execuções independentes da ferramenta IRACE, cada qual com aproximadamente 1500 execuções da heurística proposta. Cada execução da ferramenta IRACE gerou uma combinação de valores para os três parâmetros da metaheurística. A Tabela 4.4 exibe as melhores configurações obtidas em cada execução. A primeira coluna contém nomes ("alias") para a t-ésima execução. Por exemplo, a execução "N-05_K-04" corresponde à quarta execução (t = 4) de *8-Fold Cross Validation* (K = 8, calculado a partir de cinco instâncias por partição). Nessa execução, a quarta partição (que possui cinco instâncias) seria utilizada na etapa de teste e as demais partições (com 34 instâncias ao todo) foram utilizadas na etapa de treinamento. O número que acompanha a letra "N" refere-se ao número de instâncias por partição e o número que se segue após a letra "K" é a t-ésima execução. É possível observar que nas execuções cujos nomes iniciam com "N-05", os números que acompanham a letra "K" variam de 1 a 8.

Na Seção 2.3, a técnica *K-Fold Cross Validation* foi apresentada como consistindo de etapas de treinamento e teste. Entretanto, o presente trabalho não realiza a etapa de teste. Logo, a combinação de valores dos parâmetros que gerou o melhor resultado (melhor configuração) para a t-ésima execução da ferramenta IRACE não é aplicada nas instâncias de teste. Tal combinação obtida na etapa de treinamento é utilizada juntamente com as melhores combinações das outras 41 execuções independentes da ferramenta IRACE, também obtidas na etapa de treinamento, para orientar a escolha dos valores individuais

Tabela 4.4: Valores dos parâmetros obtidos nos experimentos com a ferramenta IRACE.

	Configur. sorteadas	p_e	p_m	ρ_e
N-03_K-01	88	0,18	0,05	0,55
N-03_K-02	85	0,22	0,05	0,56
N-03_K-03	75	0,21	0,06	0,59
N-03_K-04	106	0,18	0,05	0,52
N-03_K-05	97	0,21	0,06	0,54
N-03_K-06	111	0,24	0,06	0,55
N-03_K-07	104	0,22	0,06	0,58
N-03_K-08	77	0,25	0,05	0,57
N-03_K-09	94	0,25	0,05	0,57
N-03_K-10	90	0,24	0,05	0,57
N-03_K-11	83	0,24	0,06	0,55
N-03_K-12	120	0,23	0,05	0,54
N-03_K-13	88	0,23	0,05	0,54
N-05_K-01	97	0,18	0,06	0,51
N-05_K-02	97	0,24	0,06	0,55
N-05_K-03	85	0,19	0,07	0,55
N-05_K-04	116	0,24	0,05	0,55
N-05_K-05	95	0,21	0,05	0,54
N-05_K-06	109	0,26	0,05	0,57
N-05_K-07	84	0,17	0,06	0,54
N-05_K-08	104	0,24	0,06	0,56
N-07_K-01	93	0,21	0,05	0,51
N-07_K-02	80	0,24	0,06	0,57
N-07_K-03	104	0,23	0,06	0,51
N-07_K-04	80	0,25	0,06	0,57
N-07_K-05	101	0,20	0,07	0,55
N-07_K-06	78	0,25	0,06	0,59
N-09_K-01	113	0,22	0,06	0,57
N-09_K-02	86	0,20	0,07	0,55
N-09_K-03	94	0,23	0,06	0,59
N-09_K-04	103	0,20	0,05	0,53
N-09_K-05	90	0,19	0,06	0,52
N-11_K-01	122	0,19	0,06	0,52
N-11_K-02	103	0,21	0,05	0,57
N-11_K-03	106	0,23	0,06	0,56
N-11_K-04	83	0,18	0,06	0,53
N-13_K-01	94	0,22	0,06	0,57
N-13_K-02	95	0,21	0,05	0,54
N-13_K-03	89	0,25	0,05	0,58
N-15_K-01	97	0,17	0,06	0,54
N-15_K-02	80	0,24	0,06	0,58
N-15_K-03	82	0,20	0,06	0,52

de cada parâmetro e então usar essa nova combinação na execução dos experimentos para a comparação com os resultados da heurística estado-da-arte para o NRP.

São duas as justificativas para não realizar a etapa de teste: (i) os valores de cada parâmetro das 42 melhores combinações representam as regiões mais promissoras do espaço de busca avaliado pela ferramenta IRACE e escolher um valor para um dos parâmetros nessa região (usando a média, moda ou mediana dos valores dos parâmetros das melhores combinações) resultará em bons valores de *fitness* e; (ii) quanto maior o valor de K, menor será o número de instâncias testadas pela melhor combinação de valores obtida na execução independente da ferramenta IRACE. Neste segundo caso, uma combinação gerada com base em um maior valor de K terá menos instâncias para competir com uma combinação gerada para um valor menor de K se a melhor combinação entre elas for selecionada a partir da média de qualidade dos resultados nas instâncias de teste ou em uma comparação entre as instâncias. O menor número de instâncias nas combinações com maior valor de K resulta em uma amostra menos fidedigna e talvez com maior variância que aquelas com mais resultados. Tome-se como exemplos K = 3 e K = 13. No primeiro caso, cada partição possui 13 instâncias e t $\in$ {1, 2, 3}, que resulta em três melhores combinações. Como a etapa de teste seria realizada em 13 instâncias, cada melhor combinação terá 13 resultados. No segundo caso, cada partição possui três instâncias e t $\in$ {1, 2, ..., 13}, que resulta em 13 melhores combinações. Como a etapa de teste seria realizada com três instâncias, cada uma das 13 melhores combinações terá apenas três resultados.

Ainda na Tabela 4.4, a segunda coluna ("Configur. sorteadas") informa o número de combinações diferentes de parâmetros que foram sorteadas pela ferramenta IRACE dada uma provisão de 1500 execuções. É curioso observar que de 1500 combinações possíveis de parâmetros, já que foi esse o número de chamadas realizadas pela ferramenta IRACE ao programa executável da heurística proposta, apenas uma média de pouco mais de 6% desse número foi de configurações sorteadas. A razão para isto acontecer está no fato da ferramenta IRACE usar a mesma configuração em várias instâncias diferentes, de forma a garantir um tamanho de amostra com os resultados da execução da heurística suficientemente expressivo para os testes estatísticos sequenciais (vide Seção 2.2.1). A terceira, quarta e quinta colunas referem-se aos três parâmetros da metaheurítica, a saber, p_e, p_m e ρ_e.

Com o objetivo de orientar a escolha de um único valor para cada parâmetro a partir das 42 combinações geradas pelas execuções independentes, foram criados três gráficos de dispersão mostrados nas Figuras 4.6, 4.7 e 4.8.

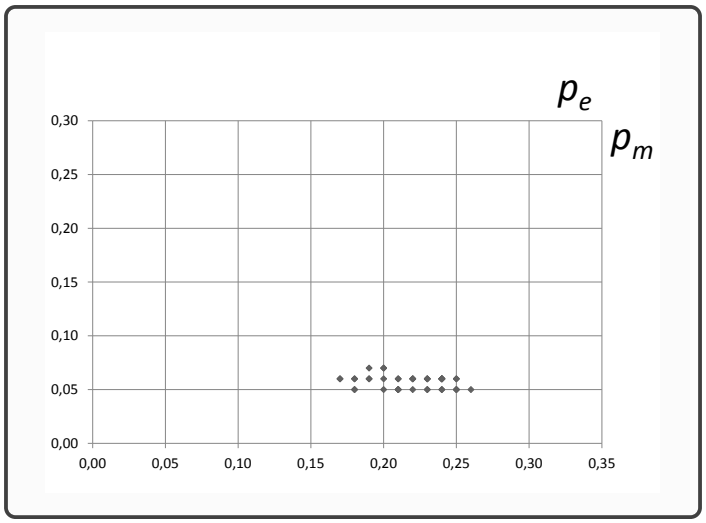

Figura 4.6: Experimento com o uso da ferramenta IRACE: Gráf. de dispersão p_e x p_m.

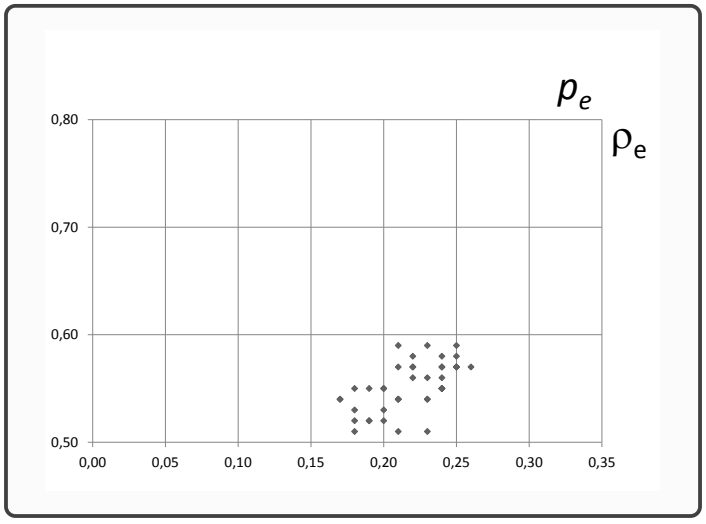

Figura 4.7: Experimento com o uso da ferramenta IRACE: Gráf. de dispersão p_e x ρ_e.

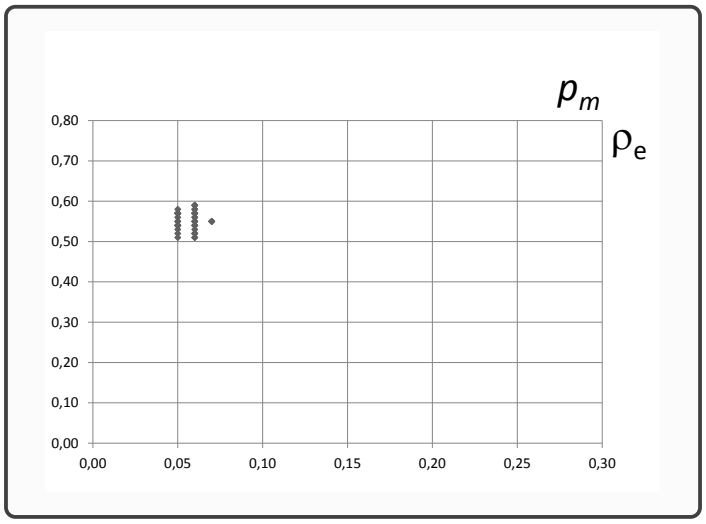

Figura 4.8: Experimento com o uso da ferramenta IRACE: Gráf. de dispersão p_m x ρ_e.

O valor 0,21 foi escolhido para p_e, visto que ocorre com maior frequência no gráfico da Figura 4.7 e está próximo à média 0,22. Os três valores que apresentaram maior frequência foram, na ordem decrescente da frequência, 0,21, 0,23 e 0,22, respectivamente com seis, cinco e quatro ocorrências. Para p_m, a média não foi considerada devido à alta frequência do valor 0,06 nos gráficos das Figuras 4.6 e 4.8. Tal valor apresentou 23 ocorrências contra 16 ocorrências do valor 0,05 e apenas três ocorrências do valor 0,07. Por fim, com o auxílio do gráfico da Figura 4.8, o valor para ρ_e foi selecionado. Tomando o valor 0,06 já escolhido para p_m e observando os valores correspondentes para ρ_e, que variaram de 0,50 a 0,59, foi adotado o valor central 0,55 para ρ_e.

4.4.2 Experimento de BRKGA_NRP sem o Uso da Ferramenta IRACE

Uma questão que surge é se há necessidade de usar uma ferramenta de configuração automática de parâmetros, com técnicas e métodos sofisticados, para a obtenção de uma combinação de valores de parâmetros para que a metaheurística gere valores de *fitness* de alta qualidade. Tal questão foi enunciada como a segunda questão de pesquisa (QP2) do presente trabalho (vide Seção 4.1) e para auxiliar em sua resposta realizada na Seção 4.5.2, este experimento foi realizado.

O experimento consistiu em trocar por um mecanismo simples de sorteio segundo uma distribuição uniforme de probabilidade, o papel da ferramenta IRACE de sortear e selecionar os valores dos três parâmetros através de testes estatísticos sequenciais e atualização sistemática das suas distribuições de probabilidade. Assim como no experimento com o uso da ferramenta IRACE, foram realizadas 42 execuções independentes, seguindo a mesma organização do *K-Fold Cross Validation*. O número de sorteios realizados foi rigorosamente igual ao número de combinações geradas pela ferramenta IRACE, assim como a quantidade de execuções da heurística proposta foi a mesma.

Na ferramenta IRACE, uma mesma combinação é executada uma única vez em várias instâncias diferentes, ou melhor, em vários pares diferentes de instância e semente aleatória. Essa forma de operar, executando uma combinação de valores de parâmetros em vários pares de instância e semente, foi também realizada pelo experimento sem o uso da ferramenta IRACE. Portanto, para cada combinação da ferramenta IRACE usada em um certo número de pares diferentes de instância e semente, houve uma respectiva combinação sorteada aleatoriamente e também usada nesses mesmos pares. O código-fonte na linguagem R da implementação de *K-Fold Cross Validation* sem o uso da ferramenta IRACE, seguindo um padrão aleatório de sorteio dos valores dos parâmetros, pode ser visto no apêndice A.2.

Seguindo as mesmas explicações da seção anterior (Seção 4.4.1), a Tabela 4.5 exibe as melhores configurações obtidas em cada uma das 42 execuções independentes, desta vez sem a ferramenta IRACE, cada qual também com aproximadamente 1500 execuções da heurística proposta.

Tabela 4.5: Valores dos parâmetros obtidos nos experimentos sem a ferramenta IRACE.

	Configur. sorteadas	p_e	p_m	ρ_e
N-03_K-01	88	0,24	0,11	0,71
N-03_K-02	85	0,18	0,27	0,61
N-03_K-03	75	0,18	0,09	0,60
N-03_K-04	106	0,28	0,19	0,52
N-03_K-05	97	0,28	0,10	0,53
N-03_K-06	111	0,26	0,27	0,78
N-03_K-07	104	0,14	0,14	0,63
N-03_K-08	77	0,18	0,06	0,51
N-03_K-09	94	0,22	0,11	0,51
N-03_K-10	90	0,15	0,19	0,65
N-03_K-11	83	0,15	0,10	0,72
N-03_K-12	120	0,19	0,09	0,53
N-03_K-13	88	0,29	0,24	0,59
N-05_K-01	97	0,23	0,08	0,53
N-05_K-02	97	0,19	0,06	0,64
N-05_K-03	85	0,27	0,26	0,60
N-05_K-04	116	0,18	0,21	0,58
N-05_K-05	95	0,18	0,29	0,79
N-05_K-06	109	0,25	0,19	0,79
N-05_K-07	84	0,15	0,06	0,54
N-05_K-08	104	0,14	0,21	0,72
N-07_K-01	93	0,28	0,07	0,61
N-07_K-02	80	0,18	0,25	0,74
N-07_K-03	104	0,29	0,09	0,69
N-07_K-04	80	0,15	0,12	0,65
N-07_K-05	101	0,26	0,08	0,54
N-07_K-06	78	0,28	0,06	0,68
N-09_K-01	113	0,25	0,17	0,78
N-09_K-02	86	0,28	0,10	0,59
N-09_K-03	94	0,21	0,23	0,62
N-09_K-04	103	0,22	0,05	0,58
N-09_K-05	90	0,27	0,10	0,60
N-11_K-01	122	0,17	0,13	0,53
N-11_K-02	103	0,22	0,07	0,70
N-11_K-03	106	0,29	0,08	0,51
N-11_K-04	83	0,22	0,15	0,54
N-13_K-01	94	0,21	0,22	0,59
N-13_K-02	95	0,30	0,14	0,67
N-13_K-03	89	0,20	0,14	0,53
N-15_K-01	97	0,22	0,24	0,59
N-15_K-02	80	0,26	0,07	0,60
N-15_K-03	82	0,26	0,14	0,68

De forma similar ao experimento com o uso da ferramenta IRACE, também foram criados três gráficos de dispersão. Eles são apresentados nas Figuras 4.9, 4.10 e 4.11 e orientam a escolha de um único valor para cada parâmetro a partir das 42 combinações geradas. Como era de se esperar, já que as combinações de valores dos parâmetros seguiram uma distribuição de probabilidade uniforme no sorteio, os pontos nos três gráficos estão mais distribuídos pelo domínio de valores de cada um dos parâmetros do que nos gráficos gerados com o uso da ferramenta IRACE (Figuras 4.6, 4.7 e 4.8). A escolha dos valores dos três parâmetros no experimento sem o uso da ferramenta IRACE seguiu critérios similares àqueles utilizados no experimento com o uso da ferramenta IRACE.

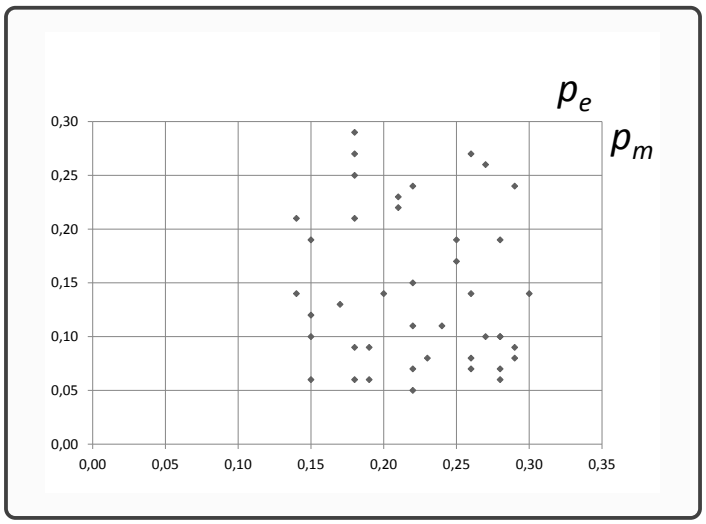

Figura 4.9: Experimento sem o uso da ferramenta IRACE: Gráf. de dispersão p_e x p_m.

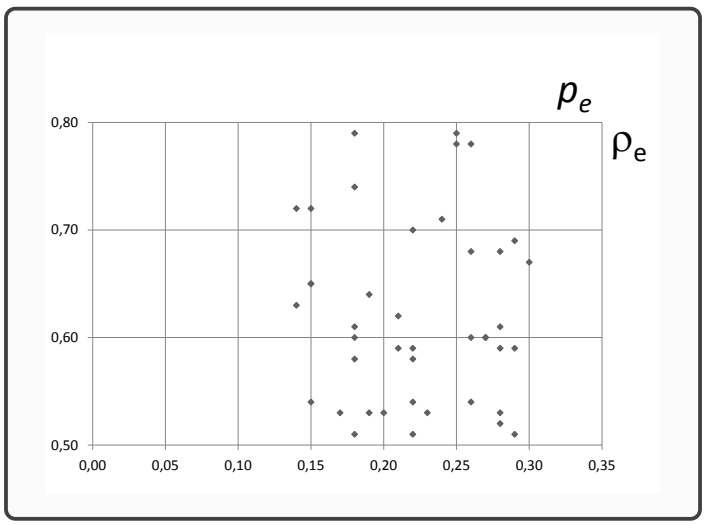

Figura 4.10: Experimento sem o uso da ferramenta IRACE: Gráf. de dispersão p_e x ρ_e.

O valor 0,22 foi escolhido para p_e, que é a média dos valores observados nas 42 execuções e também o valor com a segunda maior frequência (cinco ocorrências). O valor com a maior frequência (0,18), com seis ocorrências, não foi escolhido pois está

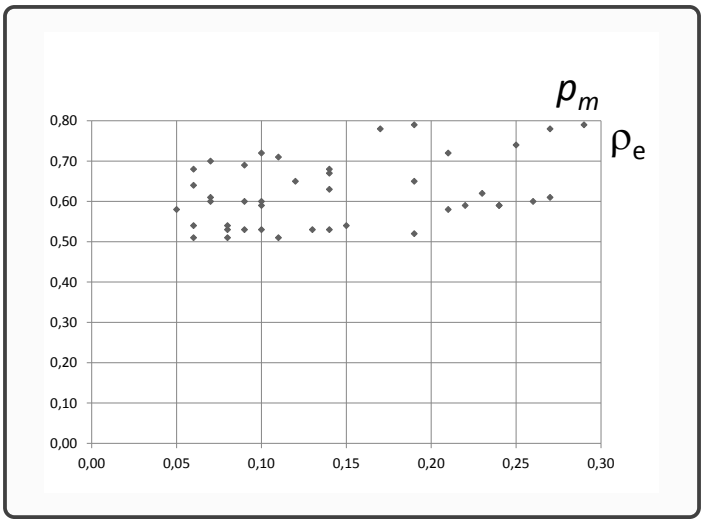

Figura 4.11: Experimento sem o uso da ferramenta IRACE: Gráf. de dispersão p_m x ρ_e.

mais distante da média. Para p_m, foi escolhido 0,14, que é a média e o valor de maior frequência, que juntamente com 0,06 possuem quatro ocorrências. Para ρ_e, foi escolhido 0,60, pois fixado o valor 0,22 para p_e no gráfico da Figura 4.10, observa-se ρ_e variando de 0,50 a 0,70, intervalo cujo valor central é 0,60. Vale salientar que apenas oito valores de ρ_e, dentre 42 valores, cuja média é 0,62, estão fora desse intervalo.

4.5 Análise e Discussão

Nesta Seção, são discutidos os resultados dos experimentos realizados com e sem a ferramenta IRACE. São comparados os resultados entre a heurística proposta BRKGA_NRP utilizando os valores dos cinco parâmetros configurados pelo IRACE e BMA. Também são comparados os resultados entre a heurística proposta BRKGA_NRP e BRKGA_NRP-R, a mesma heurística proposta só que utilizando os valores dos cinco parâmetros configurados pelo padrão randômico (ou aleatório). Por fim, os resultados de BRKGA_NRP, BRKGA_NRP-R e BMA são comparados com os ótimos das instâncias.

4.5.1 Comparação entre as Heurísticas BRKGA_NRP e BMA

Para a comparação com os resultados de BMA, a heurística estado da arte para o NRP [15], foi utilizado o teste de inferência estatística não paramétrico de Wilcoxon-Mann-Whitney [62] com nível de confiança de 95% (ou nível de significância igual a 5%, ou seja, $\alpha = 0{,}05$), que compara as medianas de duas amostras independentes e não exige normalidade e homocedasticidade das amostras comparadas.

A Tabela 4.6 exibe informações para a comparação entre a heurística BRKGA_NRP e os resultados obtidos pela heurística BMA. Na coluna "MELHOR FITNESS", constam o melhor *fitness* obtido nas 30 execuções da heurística BRKGA_NRP, o melhor *fitness* obtido nas execuções da heurística BMA e a diferença percentual entre estes dois valores, representada pela coluna "%" e segundo a fórmula $100 \times \frac{\text{BRKGA\_NRP}-\text{BMA}}{\text{BMA}}$. Na coluna "FITNESS MÉDIO", constam a média do *fitness* obtido nas 30 execuções da heurística BRKGA_NRP, a média do *fitness* obtido nas execuções da heurística BMA e a diferença percentual entre estes dois valores (coluna "%"). Na coluna "PV", consta o *p-value* para saber se os resultados das heurísticas BRKGA_NRP e BMA são significativamente diferentes. Como se tem apenas o melhor *fitness* e o *fitness* médio da heurística BMA, ambos obtidos em [15], apenas estes dois valores foram utilizados para o cálculo da mediana da heurística BMA nos testes estatísticos de Wilcoxon-Mann-Whitney contra os resultados das 30 execuções da heurística BRKGA_NRP.

De acordo com as duas colunas "%", o BRKGA_NRP superou o BMA em todas as instâncias, tanto no melhor *fitness* quanto no *fitness* médio, com exceção da instância "nrp-1-70", onde resultados iguais entre BRKGA_NRP e BMA foram obtidos. No entanto, o *p-value* foi maior que 0,05 em sete instâncias, não sendo possível afirmar que os resultados para estas instâncias são significativamente diferentes com $\alpha = 0,05$. Portanto, a resposta para a primeira questão de pesquisa (QP1) é que a heurística BRKGA_NRP supera BMA em 32 das 39 instâncias. Nas outras sete instâncias, houve um empate técnico.

Nas colunas "%" de "FITNESS MÉDIO" e "PV", percebe-se que instâncias com valores da distância percentual menores que 1,00 ("%" < 1,00) possuem os valores dos respectivos *p-value* maiores que 0,05 ("PV" > 0,05), com a instância "nrp-g2-50" como exceção, pois seu *p-value* é igual a 0,0214. De modo inverso, as instâncias com valores da distância percentual maiores que 1,00 ("%" > 1,00) possuem os valores dos respectivos *p-value* menores que 0,05 ("PV" < 0,05), com as instâncias "nrp1-50" e "nrp2-30" como exceções, com *p-values* 0,5255 e 0,0645, respectivamente. Isso indica uma coerência do teste estatístico escolhido para avaliar a diferença entre os resultados obtidos pela heurística proposta BRKGA_NRP e BMA: quando esses resultados estão muito próximos (valores pequenos de "%", entre 0 e 1%), a diferença não possui significância estatística (valores de "PV" maiores que o nível de significância α escolhido para o teste); quando os resultados se distanciam entre si (valores maiores de "%", a partir de 1%), o teste indica que os resultados são significativamente diferentes (valores de "PV" menores que α).

Tabela 4.6: Comparação entre os resultados obtidos por BRKGA_NRP e BMA.

	MELHOR FITNESS			FITNESS MÉDIO			
	BRKGA_NRP	BMA	%	BRKGA_NRP	BMA	%	PV
nrp-1-30	1.204	1.201	0,25	1.198,5	1.188,3	0,86	0,6099
nrp-1-50	1.836	1.824	0,66	1.821,2	1.796,0	1,40	0,5255
nrp-1-70	2.507	2.507	0,00	2.507,0	2.507,0	0,00	-
nrp-2-30	4.842	4.726	2,45	4.763,7	4.605,0	3,45	0,0645
nrp-2-50	7.979	7.566	5,46	7.885,3	7.414,0	6,36	0,0216
nrp-2-70	11.246	10.987	2,36	11.207,7	10.924,7	2,59	0,0216
nrp-3-30	7.341	7.123	3,06	7.250,0	7.086,0	2,31	0,0265
nrp-3-50	11.103	10.897	1,89	11.045,1	10.787,2	2,39	0,0216
nrp-3-70	14.196	14.180	0,11	14.187,6	14.159,2	0,20	0,0509
nrp-4-30	10.108	9.818	2,95	9.960,9	9.710,0	2,58	0,0471
nrp-4-50	15.658	15.025	4,21	15.540,9	14.815,5	4,90	0,0216
nrp-4-70	20.899	20.853	0,22	20.864,6	20.819,7	0,22	0,0730
nrp-5-30	18.123	17.200	5,37	17.966,5	17.026,9	5,52	0,0040
nrp-5-50	24.604	24.240	1,50	24.565,0	24.087,5	1,98	0,0215
nrp-5-70	28.912	28.909	0,01	28.908,8	28.894,2	0,05	0,1527
nrp-e1-30	7.841	7.572	3,55	7.789,7	7.528,2	3,47	0,0216
nrp-e1-50	11.021	10.664	3,35	10.969,3	10.589,2	3,59	0,0216
nrp-e2-30	7.385	7.169	3,01	7.330,4	7.109,9	3,10	0,0216
nrp-e2-50	10.335	10.098	2,35	10.292,9	9.999,8	2,93	0,0216
nrp-e3-30	6.611	6.461	2,32	6.576,4	6.413,0	2,55	0,0216
nrp-e3-50	9.334	9.175	1,73	9.305,0	9.100,1	2,25	0,0216
nrp-e4-30	5.782	5.692	1,58	5.738,2	5.636,2	1,81	0,0265
nrp-e4-50	8.165	8.043	1,52	8.133,9	7.968,0	2,08	0,0216
nrp-g1-30	6.114	5.938	2,96	6.073,6	5.911,3	2,75	0,0216
nrp-g1-50	8.880	8.714	1,90	8.853,1	8.660,0	2,23	0,0216
nrp-g2-30	4.570	4.526	0,97	4.552,4	4.486,2	1,47	0,0238
nrp-g2-50	6.548	6.502	0,71	6.534,5	6.470,2	0,99	0,0214
nrp-g3-30	5.912	5.802	1,90	5.886,8	5.736,5	2,62	0,0216
nrp-g3-50	8.488	8.402	1,02	8.462,4	8.326,8	1,63	0,0216
nrp-g4-30	4.212	4.190	0,53	4.195,9	4.159,0	0,89	0,0729
nrp-g4-50	6.059	6.030	0,48	6.050,7	5.986,5	1,07	0,0213
nrp-m1-30	10.464	10.008	4,56	10.353,1	9.889,6	4,69	0,0040
nrp-m1-50	15.312	14.588	4,96	15.174,0	14.437,7	5,10	0,0216
nrp-m2-30	8.565	8.272	3,54	8.466,8	8.147,5	3,92	0,0216
nrp-m2-50	12.448	11.975	3,95	12.382,2	11.883,5	4,20	0,0216
nrp-m3-30	10.104	9.559	5,70	10.012,3	9.499,7	5,40	0,0040
nrp-m3-50	14.881	14.138	5,26	14.792,0	14.036,6	5,38	0,0216
nrp-m4-30	7.674	7.408	3,59	7.605,3	7.319,3	3,91	0,0216
nrp-m4-50	11.288	10.893	3,63	11.232,0	10.790,7	4,09	0,0216

4.5.2 Comparação entre os Experimentos com e sem o Uso da Ferramenta IRACE

Para justificar o uso da ferramenta IRACE para a configuração automática de parâmetros da metaheurística, foi realizado um estudo experimental sem a presença da ferramenta IRACE (Seção 4.4.2). A seguir, serão discutidos os resultados de forma a responder à questão de pesquisa QP2 (vide Seção 4.1).

Para a comparação entre os resultados da heurística proposta executada com os valores dos parâmetros obtidos com o uso da ferramenta IRACE e os resultados da execução com os valores dos parâmetros obtidos sem o uso da ferramenta IRACE, foram aplicados o teste não paramétrico de Wilcoxon-Mann-Whitney, com nível de confiança igual a 95% (α = 0,05), e uma medida de tamanho de efeito não paramétrica, com o propósito de identificar a quantidade de vezes que o experimento com a ferramenta IRACE superou o experimento sem a ferramenta IRACE. Para tal, a medida baseia-se na comparação aos pares de cada um dos 30 resultados dos experimentos.

A medida de tamanho de efeito escolhida foi a $\hat{A}_{12}$ de Vargha e Delaney [62], que pode ser definida como a probabilidade do tratamento A gerar resultados melhores que o tratamento B, de forma que $\hat{A}_{12}$ indica a frequência em que um tratamento foi melhor que o outro. Por exemplo, $\hat{A}_{12}$ = 50% quando ambos os tratamentos são equivalentes, ou seja, ambos ganharam em metade das comparações, empataram em todas as comparações ou empataram em algumas comparações e nas restantes cada um ganhou a metade das vezes. Por outro lado, $\hat{A}_{12}$ = 70% indica que o tratamento A gera resultados melhores em 70% das comparações com B; já se $\hat{A}_{12}$ = 30%, é o tratamento B que gera resultados melhores em 70% das comparações com A (ou o tratamento A gera resultados melhores em 30% das comparações com B).

A Tabela 4.7 mostra as médias e desvio padrão dos resultados com o uso da ferramenta IRACE (colunas "BRKGA_NRP" e "DP") e dos resultados sem o uso da ferramenta IRACE (colunas "BRKGA_NRP-R" e "DP-R"), bem como a distância percentual entre as médias (coluna "%"), o *p-value* da comparação entre as medianas (coluna "PV") e a medida de tamanho de efeito $\hat{A}_{12}$ (coluna "ES"). É possível perceber, pelos valores apresentados nas colunas "%" e "PV", que os resultados com o uso da ferramenta IRACE superaram os resultados sem o uso desta ferramenta em 32 instâncias, mesmo que com uma margem pequena (0,93% na média, consideradas as 32 instâncias).

Nas duas instâncias onde os resultados sem o uso da ferramenta IRACE superaram os resultados com o uso da ferramenta ("nrp-1-50" e "nrp-2-30"), o *p-value* apresentou um valor bem acima de 0,05, ou seja, as medianas dos dois experimentos para essas duas

Tabela 4.7: Comparação entre os experimentos com e sem o uso da ferramenta IRACE.

	BRKGA_NRP	DP	BRKGA_NRP-R	DP-R	%	PV	ES
nrp-1-30	1.198,53	5,96	1.197,83	6,83	0,06%	0,6021	53,94%
nrp-1-50	1.821,20	6,48	1.821,50	7,22	-0,02%	0,9223	49,22%
nrp-1-70	2.507,00	0,00	2.507,00	0,00	0,00%	-	50,00%
nrp-2-30	4.763,73	50,03	4.769,17	60,61	-0,11%	0,6627	46,67%
nrp-2-50	7.885,33	37,92	7.824,50	41,53	0,78%	< 0,001	86,56%
nrp-2-70	11.207,73	27,55	11.155,20	26,96	0,47%	< 0,001	91,50%
nrp-3-30	7.250,00	50,32	7.198,00	45,09	0,72%	< 0,001	78,89%
nrp-3-50	11.045,10	29,49	10.964,67	36,20	0,73%	< 0,001	95,17%
nrp-3-70	14.187,63	7,05	14.187,80	4,74	0,00%	0,5983	54,00%
nrp-4-30	9.960,90	99,84	9.735,40	100,82	2,32%	< 0,001	93,83%
nrp-4-50	15.540,93	71,55	15.287,80	61,05	1,66%	< 0,001	98,83%
nrp-4-70	20.864,57	17,60	20.847,83	18,41	0,08%	< 0,001	77,22%
nrp-5-30	17.966,47	81,46	17.819,43	64,66	0,83%	< 0,001	92,72%
nrp-5-50	24.565,00	22,56	24.449,60	36,08	0,47%	< 0,001	99,56%
nrp-5-70	28.908,83	2,83	28.909,03	2,37	0,00%	0,9880	50,17%
nrp-e1-30	7.789,73	29,59	7.713,63	40,99	0,99%	< 0,001	92,72%
nrp-e1-50	10.969,33	29,30	10.867,57	35,71	0,94%	< 0,001	98,33%
nrp-e2-30	7.330,37	27,49	7.275,27	31,01	0,76%	< 0,001	91,72%
nrp-e2-50	10.292,90	29,92	10.191,63	41,23	0,99%	< 0,001	97,83%
nrp-e3-30	6.576,40	27,20	6.539,00	24,57	0,57%	< 0,001	83,22%
nrp-e3-50	9.304,97	18,47	9.252,87	22,88	0,56%	< 0,001	96,78%
nrp-e4-30	5.738,23	21,31	5.714,00	28,49	0,42%	< 0,001	75,22%
nrp-e4-50	8.133,90	14,57	8.095,33	13,23	0,48%	< 0,001	97,67%
nrp-g1-30	6.073,60	23,06	6.041,97	21,55	0,52%	< 0,001	82,78%
nrp-g1-50	8.853,07	15,59	8.802,43	18,34	0,58%	< 0,001	98,22%
nrp-g2-30	4.552,37	10,47	4.543,77	11,48	0,19%	0,0074	70,17%
nrp-g2-50	6.534,53	7,79	6.519,67	11,29	0,23%	< 0,001	85,83%
nrp-g3-30	5.886,77	15,54	5.858,40	23,42	0,48%	< 0,001	84,61%
nrp-g3-50	8.462,43	13,98	8.430,00	15,15	0,38%	< 0,001	94,00%
nrp-g4-30	4.195,93	8,61	4.192,43	7,52	0,08%	0,0833	63,06%
nrp-g4-50	6.050,67	5,63	6.042,23	9,05	0,14%	< 0,001	78,28%
nrp-m1-30	10.353,13	65,15	10.180,63	70,39	1,69%	< 0,001	97,22%
nrp-m1-50	15.173,97	60,95	14.865,20	56,58	2,08%	< 0,001	100,00%
nrp-m2-30	8.466,83	55,32	8.329,13	53,72	1,65%	< 0,001	95,67%
nrp-m2-50	12.382,23	35,44	12.180,67	49,11	1,65%	< 0,001	100,00%
nrp-m3-30	10.012,27	61,25	9.830,43	45,61	1,85%	< 0,001	99,22%
nrp-m3-50	14.792,03	44,77	14.502,20	62,58	2,00%	< 0,001	100,00%
nrp-m4-30	7.605,33	47,46	7.511,20	40,75	1,25%	< 0,001	91,78%
nrp-m4-50	11.232,03	38,14	11.102,87	49,11	1,16%	< 0,001	96,72%

instâncias não são significativamente diferentes. Portanto, pode ser considerado um caso de empate. Na instância "nrp-1-70", tanto os valores de *fitness* das 30 execuções com os parâmetros gerados com o uso da ferramenta IRACE quanto os valores de *fitness* das 30 execuções com os parâmetros gerados sem a ferramenta IRACE foram rigorosamente iguais, caracterizando um empate. Tal fato é confirmado pela medida $\hat{A}_{12}$, que possui o valor de 50%. Por fim, para as demais quatro instâncias ("nrp-1-30", "nrp-3-70", "nrp-5-70" e "nrp-g4-30"), embora os resultados com o uso da ferramenta IRACE tenham sido melhores, o *p-value* apresentou valores acima de 0,05, não indicando diferença significativa na comparação com os resultados sem a ferramenta.

Em relação à medida de tamanho de efeito $\hat{A}_{12}$, os sete casos de empate apresentaram valores próximos de 50%. Para os 32 casos em que os resultados com o uso da ferramenta IRACE superaram os resultados sem o uso da ferramenta, $\hat{A}_{12}$ apresentou um valor médio de 91,32%, o que fornece a resposta para a questão de pesquisa QP2, qual seja, de que os resultados dos experimentos com a heurística proposta utilizando os parâmetros obtidos com o uso da ferramenta IRACE proporcionam melhores resultados em relação àqueles utilizando os parâmetros obtidos sem o uso da ferramenta IRACE, justificando assim o uso da ferramenta de configuração automática de parâmetros IRACE.

4.5.3 Comparação com os Ótimos Calculados por Veerapen et al.

Em 2015, Veerapen et al. calcularam os ótimos para as 39 instâncias aqui utilizadas através de uma abordagem de programação linear inteira [9]. A terceira questão da pesquisa (QP3) é respondida através do exame da Tabela 4.8, que mostra os valores da distância percentual para o ótimo a partir das médias dos resultados obtidos com as 30 execuções da heurística proposta com os parâmetros obtidos com o auxílio da ferramenta IRACE (coluna "BRKGA_NRP") e sem o auxílio da ferramenta IRACE (coluna "BRKGA_NRP-R"), e a média das execuções do BMA segundo [15] (coluna "BMA"). Se consideradas todas as instâncias, as colunas "BRKGA_NRP", "BRKGA_NRP-R" e "BMA" apresentaram respectivamente 1,55%, 2,28% e 4,14% de distância média para o ótimo. Baseando-se na distância média para o ótimo, BRKGA_NRP obteve uma melhora de 63% em relação a BMA e é melhor 32% em relação a BRKGA_NRP-R. A distância entre os resultados de BRKGA_NRP e os ótimos foi inferior a 1% em 19 das 39 instâncias.

Tabela 4.8: Comparação com os ótimos.

	ÓTIMO	BRKGA_NRP	BRKGA_NRP-R	BMA
nrp-1-30	1204	0,45%	0,51%	1,30%
nrp-1-50	1840	1,02%	1,01%	2,39%
nrp-1-70	2507	0,00%	0,00%	0,00%
nrp-2-30	4970	4,15%	4,04%	7,34%
nrp-2-50	8065	2,23%	2,98%	8,07%
nrp-2-70	11316	0,96%	1,42%	3,46%
nrp-3-30	7488	3,18%	3,87%	5,37%
nrp-3-50	11159	1,02%	1,74%	3,33%
nrp-3-70	14196	0,06%	0,06%	0,26%
nrp-4-30	10690	6,82%	8,93%	9,17%
nrp-4-50	15985	2,78%	4,36%	7,32%
nrp-4-70	20913	0,23%	0,31%	0,45%
nrp-5-30	18510	2,94%	3,73%	8,01%
nrp-5-50	24701	0,55%	1,02%	2,48%
nrp-5-70	28912	0,01%	0,01%	0,06%
nrp-e1-30	7919	1,63%	2,59%	4,93%
nrp-e1-50	11070	0,91%	1,83%	4,34%
nrp-e2-30	7446	1,55%	2,29%	4,51%
nrp-e2-50	10381	0,85%	1,82%	3,67%
nrp-e3-30	6666	1,34%	1,91%	3,80%
nrp-e3-50	9362	0,61%	1,17%	2,80%
nrp-e4-30	5891	2,59%	3,00%	4,33%
nrp-e4-50	8174	0,49%	0,96%	2,52%
nrp-g1-30	6130	0,92%	1,44%	3,57%
nrp-g1-50	8897	0,49%	1,06%	2,66%
nrp-g2-30	4580	0,60%	0,79%	2,05%
nrp-g2-50	6553	0,28%	0,51%	1,26%
nrp-g3-30	5932	0,76%	1,24%	3,30%
nrp-g3-50	8501	0,45%	0,84%	2,05%
nrp-g4-30	4218	0,52%	0,61%	1,40%
nrp-g4-50	6063	0,20%	0,34%	1,26%
nrp-m1-30	10770	3,87%	5,47%	8,17%
nrp-m1-50	15540	2,36%	4,34%	7,09%
nrp-m2-30	8707	2,76%	4,34%	6,43%
nrp-m2-50	12585	1,61%	3,21%	5,57%
nrp-m3-30	10391	3,64%	5,39%	8,58%
nrp-m3-50	15096	2,01%	3,93%	7,02%
nrp-m4-30	7777	2,21%	3,42%	5,89%
nrp-m4-50	11369	1,20%	2,34%	5,09%

4.5.4 Ameaças à Validade

Wohlin et al. [63] descrevem as fases do processo de experimentação em Engenharia de Software. A segunda fase, nomeada como fase de planejamento, envolve, entre outros tópicos, a avaliação da validade dos experimentos realizados em um trabalho de pesquisa. Essa avaliação fundamenta-se em identificar, analisar e tratar as potenciais ameaças ou riscos que invalidem os resultados e suas conclusões obtidos com os experimentos. Uma vez realizadas a identificação e a análise das ameaças, um tratamento adequado deve ser dado a cada uma das ameaças, seja eliminando, mitigando ou até mesmo assumindo uma dada ameaça.

Os autores sugerem a classificação das ameaças à validade em quatro categorias: validade de conclusão, validade interna, validade de construção e validade externa. Cada categoria possui uma lista de ameaças, onde cada item da lista possui uma descrição genérica o bastante para contemplar as diversas áreas de experimentação em Engenharia de Software. Ao todo, são 33 itens divididos entre as quatro categorias e cada item deve ser verificado em relação ao trabalho de pesquisa com o objetivo de identificar alguma ameaça à validade. Barros e Dias-Neto [64] traduzem alguns dos 33 itens da lista de ameaças à validade citados em Wohlin et al. [63] para a realidade dos trabalhos na área de Engenharia de Software Baseada em Buscas, mantendo as quatro categorias. A nova lista proposta totaliza 14 itens, os quais foram examinados pelo presente trabalho de pesquisa para identificar as ameaças à validade.

Ameaças à validade de conclusão afetam a capacidade para extrair conclusões corretas sobre o relacionamento entre o tratamento e o resultado de um experimento. Por corretas, entende-se as conclusões que tenham respaldo em evidências estatisticamente significativas. Em relação a essas ameaças, o presente trabalho responde satisfatoriamente aos quatro itens desta categoria. Para considerar a variação randômica associada às metaheurísticas, visto que elas possuem componentes aleatórios, nos resultados foram realizadas 30 execuções da heurística proposta para cada instância, tanto no estudo de convergência (Seção 4.3) como nos experimentos computacionais (Seção 4.4). No estudo de convergência e nos dois experimentos com e sem o uso da ferramenta IRACE, o valor médio do *fitness* e a sua mediana foram utilizados para fins de comparação, inclusive contra o ótimo. O melhor valor do *fitness* dentre as 30 execuções foi utilizado apenas para a comparação com o melhor valor do *fitness* do BMA. Como medida de dispersão, foi utilizado o desvio padrão. Portanto, houve um cuidado em apresentar estatísticas descritivas de forma clara. Já a importância da escolha de uma comparação adequada foi contemplada pela adoção dos resultados do algoritmo estado-da-arte do NRP, o BMA, para avaliar a eficácia da

heurística proposta. No caso do uso da ferramenta IRACE, a comparação foi realizada contra os resultados obtidos sem o uso da ferramenta. Por fim, o teste estatístico escolhido, que não exige normalidade nem homocedasticidade, está aderente às características dos dados avaliados.

Ameaças à validade interna consistem nos fatores fora do controle do pesquisador que podem afetar as variáveis independentes e seus resultados. Logo, o plano do experimento deve assegurar que as relações entre tratamento e resultado são fruto exclusivamente do relacionamento de causa e efeito entre variáveis independentes e dependentes e não de outro fator não controlado. Para mitigar essas ameaças, o presente trabalho busca deixar claro: (i) quais parâmetros da metaheurística foram utilizados e como seus valores foram obtidos (Seção 4.3 e Seção 4.4); (ii) como a heurística proposta e a técnica *K-Fold Cross Validation* foram implementadas, disponibilizando o código-fonte e explicando a lógica através de pseudo-código (Algoritmos 1 e 2 e Apêndices A.1 e A.2); (iii) como as 39 instâncias foram obtidas bem como as características de cada uma delas (Seção 4.2); e (iv) a utilização de 24 instâncias realistas obtidas de Xuan et al. [15] (Seção 4.2) como forma de contemplar aspectos do NRP mais próximos da realidade.

Ameaças à validade de construção dizem respeito ao relacionamento entre a observação dos resultados do experimento e a teoria que suporta o experimento, de modo que a teoria deve ser uma generalização dos resultados. Em outras palavras, o tratamento deve refletir o constructo da causa e o resultado deve refletir o constructo do efeito. Uma dessas ameaças é a ausência de métricas de eficácia adequadas para o problema em questão. Tal ameaça foi resolvida com a escolha do lucro como métrica. Um outra ameaça é a ausência de discussão sobre o modelo do problema a ser otimizado, considerando as limitações decorrentes da simplificação originada do modelo na tentativa deste representar a realidade e as consequências dessas limitações. No caso deste trabalho, o modelo utilizado trata-se do NRP em sua formulação mono-objetivo, que foi proposto em 2001 [5] e desde então avaliado pelos mais diversos trabalhos, sendo portanto muito discutido e conhecido.

Por último, ameaças à validade externa referem-se à generalização dos resultados observados no experimento para uma população maior. No caso de Engenharia de Software Baseada em Busca, tais ameaças estão relacionadas à preocupação se as instâncias escolhidas, conhecidas e caracterizadas, representam adequadamente as demais instâncias do problema, que são desconhecidas, a ponto dos resultados observados nas instâncias conhecidas serem satisfatoriamente generalizados para as instâncias desconhecidas. Essa preocupação refere-se tanto à quantidade de instâncias utilizadas no experimento como às propriedades que definem cada uma dessas instâncias. No presente trabalho, é notória a pequena quantidade de instâncias utilizadas. No entanto, conforme enunciado no item 3

da Seção 1.3, o objetivo do trabalho é comparar os resultados obtidos com os resultados encontrados na literatura, com instâncias utilizadas recorrentemente nos outros trabalhos da literatura. A dificuldade de encontrar mais instâncias para o NRP implicou no número limitado de instâncias utilizadas nos experimentos relatados neste capítulo.

4.6 Considerações Finais

Neste capítulo, foram apresentadas as questões de pesquisa que orientaram os experimentos computacionais planejados e executados com o objetivo de avaliar a heurística $\mathsf{BRKGA}_{\mathsf{NRP}}$. As instâncias do problema foram descritas em detalhes, o ambiente computacional onde os experimentos foram realizados foi citado e o procedimento de definição dos parâmetros da metaheurística utilizada foi descrito em duas partes: a definição dos valores para os parâmetros tamanho da população e número de gerações (estudo de convergência) e a definição dos valores para os parâmetros percentual de cromossomos do estrato elite, percentual de mutantes e probabilidade do descendente herdar o gene do pai que pertence ao estrato elite.

Por fim, foram realizadas três análises comparativas: (i) entre os resultados da heurística proposta (com o uso da ferramenta IRACE) e da heurística estado da arte para o NRP (BMA); (ii) entre os resultados obtidos com o uso da ferramenta IRACE e sem o uso da ferramenta; e (iii) entre os resultados da heurística proposta (com e sem o uso da ferramenta IRACE) e os valores ótimos para as instâncias. Os resultados demonstraram que a heurística proposta superou BMA e que usar a ferramenta de configuração automática de parâmetros IRACE é mais favorável à obtenção de melhores valores de *fitness* do que usar parâmetros obtidos através de seleção aleatória. Os resultados também demonstraram que os valores de *fitness* da heurística proposta são em média 1,6% piores que os ótimos.

No próximo capítulo, são apresentadas as conclusões deste trabalho, bem como a possibilidade de trabalhos futuros a partir do que foi iniciado na presente pesquisa.

5. Conclusão

O presente trabalho de pesquisa apresentou a aplicação da heurística BRKGA_NRP, baseada na metaheurística BRKGA, ao problema de otimização combinatória NRP. A metaheurística BRKGA, do tipo populacional, possui parâmetros de configuração que dizem respeito a informações da população e das gerações ao longo do processo evolutivo e que influenciam na eficácia (qualidade das soluções) e na eficiência (tempo de execução) das heurísticas que usam BRKGA como guia. Este trabalho propôs um procedimento sistemático e automatizado para a escolha dos valores desses parâmetros por meio do uso da ferramenta IRACE, uma ferramenta de configuração automática de parâmetros que vem adquirindo alguma notoriedade em função de sua crescente adoção. O uso da ferramenta foi validado por um estudo comparativo em relação a valores dos parâmetros obtidos segundo um padrão aleatório. Após a escolha dos parâmetros, a heurística proposta foi executada e seus resultados foram comparados com a heurística estado da arte para o NRP, o BMA, e com os ótimos calculados através de métodos exatos publicados em um trabalho anterior [9]. Por estes resultados, a heurística proposta BRKGA_NRP mostra-se uma solução atraente para abordar o NRP.

5.1 Contribuições

Os objetivos do presente trabalho, enunciados na Seção 1.3, foram elaborados com a intenção de gerar contribuições para a área de Engenharia de Software Baseada em Buscas, sem, no entanto, desconsiderar seu potencial para gerar contribuições em áreas de maior abrangência, como Pesquisa Operacional e Otimização Combinatória, uma vez que utiliza uma metaheurística aplicada a um problema de otimização. Portanto, as contribuições científicas e tecnológicas do trabalho de pesquisa estão vinculadas ao cumprimento destes objetivos.

Para cumprir o primeiro objetivo, o presente trabalho propôs e implementou um método construtivo para o problema NRP (BRKGA_NRP como uma nova técnica de busca heurística para resolver o NRP), ao mesmo tempo que proporcionou uma ampliação do portfólio de utilização da metaheurística BRKGA com a sua aplicação em um problema clássico da literatura da Engenharia de Software.

Outra contribuição relevante foi a elaboração do estudo de convergência para a definição dos valores do tamanho da população e do número de gerações da metaheurística BRKGA, importantes para a eficácia e eficiência das heurísticas baseadas em BRKGA. Tal estudo, detalhado em cada um de seus passos, pode ser utilizado por outros trabalhos de pesquisa em que se pretenda utilizar BRKGA em algum problema de otimização combinatória. Tal contribuição veio em cumprimento ao segundo objetivo da pesquisa, ao decidir quais parâmetros da metaheurística seriam submetidos à configuração automática e quais não seriam, que foi o caso do tamanho da população e do número de gerações.

Ao cumprir o segundo e terceiro objetivos, o presente trabalho forneceu evidências de que utilizar a heurística proposta associada à técnica *K-Fold Cross Validation* e à ferramenta IRACE é uma alternativa atraente tomando como base os resultados obtidos pela heurística estado da arte para o NRP.

Por fim, o cumprimento do quarto objetivo resultou na evidência experimental da utilidade da ferramenta IRACE para realizar a configuração automática de parâmetros, ao comparar seus resultados com os resultados de uma configuração sugerida por um padrão aleatório.

5.2 Lições Aprendidas

Quando na escolha da métrica a ser utilizada pela ferramenta IRACE (Seção 4.4.1), inicialmente o valor escolhido foi a distância relativa entre o lucro na solução encontrada ($\mathrm{lucro}_{\mathrm{SOL}}$) e o lucro ótimo ($\mathrm{lucro}_{\mathrm{OTIMO}}$), valor este expresso pela Equação 5.1 e que assegurava de forma precisa um critério de comparação inter instâncias. Dito de outra forma, quanto mais próximo de zero o valor da métrica, melhor o *fitness* calculado utilizando aquele conjunto de parâmetros da metaheurística, independente de qual instância estivesse sendo utilizada.

$$\mathrm{fitness}_{\mathrm{IRACE}} = \frac{\mathrm{lucro}_{\mathrm{OTIMO}} - \mathrm{lucro}_{\mathrm{SOL}}}{\mathrm{lucro}_{\mathrm{OTIMO}}} \tag{5.1}$$

No entanto, como o presente trabalho propõe um método que tem por objetivo encontrar bons valores de *fitness* também para instâncias que não as 39 instâncias aqui utilizadas, muitas das quais provavelmente sem o ótimo conhecido, não faria sentido considerar o ótimo. Como alternativa, a métrica representada pela Equação 4.1 (Seção 4.4.1) foi proposta na ausência de alguma outra métrica que representasse de forma fidedigna a métrica da Equação 5.1. O que se percebe é que, dada as características de cada instância, a soma dos lucros de todos os clientes ($\mathtt{lucro}_{\mathtt{TOTAL}}$) pode estar muito acima do ótimo ($\mathtt{lucro}_{\mathtt{OTIMO}}$) em algumas instâncias e mais próxima do ótimo em outras instâncias. O fato é que como a distância entre $\mathtt{lucro}_{\mathtt{TOTAL}}$ e $\mathtt{lucro}_{\mathtt{OTIMO}}$ é específica de cada instância, a métrica escolhida no presente trabalho pode gerar distorções nos resultados. É o caso, por exemplo, quando o *fitness* calculado para uma dada instância está bem próximo do ótimo, mas muito longe da soma dos lucros de todos os clientes (pois as características dessa instância proporcionam essa condição) e o *fitness* calculado de uma outra instância está um pouco mais distante do ótimo, porém para essa segunda instância, a soma dos lucros de todos os clientes é próxima do ótimo. Com a métrica adotada, a execução da heurística (e seu conjunto de parâmetros) que resultou no segundo *fitness* seria equivocadamente considerada mais eficaz que aquela do primeiro *fitness*.

Outra questão enfrentada foi a ausência de critérios para definir o número de execuções da heurística definido na ferramenta IRACE para cada uma das 42 execuções independentes do *K-Fold Cross Validation*. O valor limite escolhido foi 1500, porém acredita-se ser possível que um número menor possa ser utilizado sem prejuízo na qualidade das soluções geradas.

5.3 Trabalhos Futuros

Conforme citado na Seção 2.5, a ferramenta de configuração automática de parâmetros utilizada no presente trabalho foi escolhida pelo fato de ter sido empregada em trabalhos recentes de aplicação da metaheurística BRKGA, alguns deles de autoria dos próprios autores da metaheurística. No entanto, conforme citado na Seção 2.2.1, existem outras opções de ferramentas de configuração automática de parâmetros, algumas de propósito geral e com flexibilidade para uso com diversas metaheurísticas e diversos métodos exatos, como a própria ferramenta IRACE, e outras específicas para certas heurísticas. Em um trabalho futuro, a técnica *K-Fold Cross Validation* pode ser associada a outras ferramentas de configuração automática para que um estudo comparativo seja realizado com o objetivo de saber qual ferramenta obtém os melhores resultados, tanto em termos de qualidade das soluções quanto no tempo dispendido para realizar a configuração.

Ao visitar todos os genes do cromossomo de forma sequencial, o método construtivo da heurística proposta verifica a possibilidade de inclusão de todos os clientes. Mesmo após ter alcançado o cliente que proporcionou o primeiro estouro de orçamento, e que obviamente não pôde ser incluído na solução, o método construtivo segue avaliando cliente a cliente na intenção de encontrar mais clientes cujos requisitos ainda caibam no orçamento. Nessa abordagem, a inclusão de um novo cliente na solução acontece quando a soma dos custos de seus requisitos (e requisitos antecessores) é um valor pequeno o bastante para caber no orçamento, quando esse novo cliente possui requisitos que já foram incluídos no cômputo do custo da solução (ou seja, foram solicitados por outros clientes já incluídos na solução) ou por causa de uma combinação das duas situações anteriores. O uso dessa abordagem permitiu que o custo da solução associada a cada cromossomo apresentasse um valor menor que o orçamento em apenas uma única unidade de custo aproximadamente, independente do tamanho da instância, o que indica um grande aproveitamento do orçamento.

Embora haja um grande aproveitamento do orçamento na abordagem proposta (avaliação cliente a cliente até o final da sequência), é possível que o *fitness* da solução obtida com essa abordagem ofereça pouco ou nenhum ganho em relação ao *fitness* da solução obtida utilizando como critério de parada o primeiro estouro do orçamento. Investigar essa possibilidade e decidir qual a melhor abordagem, incluindo, entre as opções, outras condições de critério de parada, entre o momento em que o orçamento é excedido pela primeira vez e o momento em que o último cliente da sequência é avaliado, é uma análise interessante que será deixada para um trabalho futuro. Dentre essas opções, é possível citar como exemplos de critérios de parada os próximos estouros do orçamento (segundo, terceiro, quarto, etc.) ou a adoção de um percentual máximo de clientes que devem ser avaliados dentre os clientes restantes após um desses estouros do orçamento.

Algumas operações do *framework* computacional da metaheurística BRKGA possuem independência em relação às demais operações, de modo que podem ser executadas em paralelo sem prejudicar a execução do *framework* como um todo. Tais operações são a criação da população inicial de cromossomos e o processo evolutivo do algoritmo genético, que inclui a introdução de mutantes na população, o cruzamento para a geração de descendentes e o cálculo do *fitness* de cada cromossomo. Como há suporte para processamento paralelo tanto na ferramenta IRACE (pacotes R “parallel” e “Rmpi”) quanto no *framework* computacional da metaheurística BRKGA (OpenMP) e os processadores atuais mais simples possuem dois núcleos (*cores*) independentes, uma proposta para um trabalho futuro seria usar esse recurso para gerar soluções de melhor qualidade em um tempo menor de execução.

Outro recurso fornecido pelo *framework* computacional da metaheurística BRKGA, e não explorado pelo presente trabalho, é a utilização de múltiplas populações que evoluem de forma independente. Após um certo número de gerações, os cromossomos com os melhores *fitness* são trocados entre as populações em substituição aos cromossomos com os piores *fitness*, com o cuidado de preservar o tamanho fixo da população. Ao final da execução do algoritmo, a melhor solução escolhida é a melhor solução dentre todas as populações. Utilizar tal recurso constitui-se em mais uma proposta para trabalhos futuros.

Referências Bibliográficas

[1] och Dag, J. N. (2002). Elicitation and management of user requirements in market-driven software development. *Department of Communication Systems Lund Institute of Technology, Licentiate Thesis.*

[2] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & och Dag, J. N. (2001). An industrial survey of requirements interdependencies in software product release planning. In Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on (pp. 84-91). IEEE.

[3] Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and software Technology, 43*(14), 833-839.

[4] Baker, P., Harman, M., Steinhofel, K., & Skaliotis, A. (2006, September). Search based approaches to component selection and prioritization for the next release problem. *In Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on* (pp. 176-185). IEEE.

[5] Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. Information and software technology, 43(14), 883-890.

[6] Jiang, H., Zhang, J., Xuan, J., Ren, Z., & Hu, Y. (2010, June). A hybrid ACO algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on* (pp. 166-171). IEEE.

[7] de Souza, J. T., Maia, C. L. B., do Nascimento Ferreira, T., Do Carmo, R. A. F., & Brasil, M. M. A. (2011, September). An ant colony optimization approach to the software release planning with dependent requirements. In *International Symposium on Search Based Software Engineering* (pp. 142-157). Springer, Berlin, Heidelberg.

[8] Del Sagrado, J., Del Águila, I. M., & Orellana, F. J. (2010, September). Ant colony optimization for the next release problem: A comparative study. In *Search Based*

Software Engineering (SSBSE), 2010 Second International Symposium on (pp. 67-76). IEEE.

[9] Veerapen, N., Ochoa, G., Harman, M., & Burke, E. K. (2015). An integer linear programming approach to the single and bi-objective next release problem. *Information and Software Technology,* 65, 1-13.

[10] Li, L. (2016, September). Exact analysis for next release problem. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International* (pp. 438-443). IEEE.

[11] Harman, M., Krinke, J., Ren, J., & Yoo, S. (2009, July). Search based data sensitivity analysis applied to requirement engineering. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 1681-1688). ACM.

[12] Omidvar, M. N., Li, X., Yang, Z., & Yao, X. (2010, July). Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (pp. 1-8). IEEE.

[13] Saliu, M. O., & Ruhe, G. (2007, September). Bi-objective release planning for evolving software systems. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (pp. 105-114). ACM.

[14] Gonçalves, J. F., & Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*(5), 487-525.

[15] Xuan, J., Jiang, H., Ren, Z., & Luo, Z. (2012). Solving the large scale next release problem with a backbone-based multilevel algorithm. *IEEE Transactions on Software Engineering, 38*(5), 1195-1212.

[16] Sörensen, K. (2015). Metaheuristics-the metaphor exposed. *International Transactions in Operational Research,* 22(1), 3-18.

[17] Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges,* 59, 215-250.

[18] Glover, F. W., & Kochenberger, G. A. (Eds.). (2006). *Handbook of metaheuristics* (Vol. 57). Springer Science & Business Media.

[19] Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization, 6*(2), 109-133.

[20] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science, 220*(4598), 671-680.

[21] Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093-2229). Springer, Boston, MA.

[22] Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research, 24*(11), 1097-1100.

[23] Glover, F., Laguna, M., & Martí, R. (2003). Scatter search and path relinking: Advances and applications. In *Handbook of metaheuristics* (pp. 1-35). Springer, Boston, MA.

[24] Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics* (pp. 320-353). Springer, Boston, MA.

[25] Dorigo, M., & Birattari, M. (2011). Ant colony optimization. In *Encyclopedia of machine learning* (pp. 36-39). Springer, Boston, MA.

[26] Kennedy, J. (2011). Particle swarm optimization. In *Encyclopedia of machine learning* (pp. 760-766). Springer, Boston, MA.

[27] Reeves, C. (2003). Genetic algorithms. In *Handbook of metaheuristics* (pp. 55-82). Springer, Boston, MA.

[28] Holland, J. H., & Goldberg, D. (1989). Genetic algorithms in search, optimization and machine learning. *Massachusetts: Addison-Wesley*.

[29] Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing, 6*(2), 154-160.

[30] Cáceres, L. P., López-Ibáñez, M., Hoos, H., & Stützle, T. (2017, June). An experimental study of adaptive capping in irace. In *International Conference on Learning and Intelligent Optimization* (pp. 235-250). Springer, Cham.

[31] Hutter, F., López-Ibánez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2014, February). AClib: A benchmark library for algorithm configuration. In *International Conference on Learning and Intelligent Optimization* (pp. 36-40). Springer, Cham.

[32] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43-58.

[33] Birattari, M., & Kacprzyk, J. (2009). *Tuning metaheuristics: a machine learning perspective* (Vol. 197). Berlin: Springer.

[34] Karafotias, G., Hoogendoorn, M., & Eiben, Á. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation, 19*(2), 167-187.

[35] Adenso-Diaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research, 54*(1), 99-114.

[36] Coy, S. P., Golden, B. L., Runger, G. C., & Wasil, E. A. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics, 7*(1), 77-97.

[37] Ridge, E., & Kudenko, D. (2007). Tuning the performance of the MMAS heuristic. In *Engineering stochastic local search algorithms. designing, implementing and analyzing effective heuristics* (pp. 46-60). Springer, Berlin, Heidelberg.

[38] Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation, 9*(2), 159-195.

[39] Powell, M. J. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge,* 26-46.

[40] Audet, C., & Orban, D. (2006). Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization, 17*(3), 642-664.

[41] Nannen, V., & Eiben, A. E. (2007, September). Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on* (pp. 103-110). IEEE.

[42] Riff, M. C., & Montero, E. (2013, June). A new algorithm for reducing metaheuristic design effort. In *Evolutionary Computation (CEC), 2013 IEEE Congress on* (pp. 3283-3290). IEEE.

[43] Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research, 36,* 267-306.

[44] Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011, January). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization* (pp. 507-523). Springer, Berlin, Heidelberg.

[45] Bartz-Beielstein, T., Lasarczyk, C. W., & Preuß, M. (2005, September). Sequential parameter optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on* (Vol. 1, pp. 773-780). IEEE.

[46] Maron, O., & Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review, 11*(1-5), 193-225.

[47] Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002, July). A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (pp. 11-18). Morgan Kaufmann Publishers Inc..

[48] Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311-336). Springer, Berlin, Heidelberg.

[49] Team, R. C. (2000). R language definition. *Vienna, Austria: R foundation for statistical computing.*

[50] López-Ibáñez, M., Cáceres, L. P., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2016). The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004.*

[51] Conover, W. J., & Conover, W. J. (1980). Practical nonparametric statistics.

[52] Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics surveys, 4,* 40-79.

[53] Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, No. 2, pp. 1137-1145).

[54] Zhang, Y., Harman, M., & Mansouri, S. A. (2007, July). The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (pp. 1129-1137). ACM.

[55] Durillo, J. J., Zhang, Y., Alba, E., & Nebro, A. J. (2009, May). A study of the multi-objective next release problem. In *Search Based Software Engineering, 2009 1st International Symposium on* (pp. 49-58). IEEE.

[56] Toso, R. F., & Resende, M. G. (2015). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software, 30*(1), 81-93.

[57] Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS), 8*(1), 3-30.

[58] Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. (2006). *Metaheuristics for hard optimization: methods and case studies*. Springer Science & Business Media.

[59] Stefanello, F., Aggarwal, V., Buriol, L. S., Gonçalves, J. F., & Resende, M. G. (2015, July). A biased random-key genetic algorithm for placement of virtual machines across geo-separated data centers. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (pp. 919-926). ACM.

[60] Andrade, C. E., Resende, M. G., Zhang, W., Sinha, R. K., Reichmann, K. C., Doverspike, R. D., & Miyazawa, F. K. (2015). A biased random-key genetic algorithm for wireless backhaul network design. *Applied Soft Computing, 33,* 150-169.

[61] de Andrade, C. E., Toso, R. F., Resende, M. G., & Miyazawa, F. K. (2015). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary computation, 23*(2), 279-307.

[62] Arcuri, A., & Briand, L. (2011, May). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Software Engineering (ICSE), 2011 33rd International Conference on* (pp. 1-10). IEEE.

[63] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

[64] de Oliveira Barros, M., & Dias-Neto, A. C. (2011). 0006/2011-Threats to validity in search-based software engineering empirical studies. *RelaTe-DIA, 5*(1).

A. Código-fonte Implementado na Linguagem R

A.1 *K-Fold Cross Validation* e Configuração Automática de Parâmetros com IRACE

```
library(irace)
numInstanciasPorParticao <- c(3,5,7,9,11,13,15)
instancias <- c(
"nrp1-30", "nrp2-30", "nrp3-30", "nrp4-30", "nrp5-30", #1..5
"nrp1-50", "nrp2-50", "nrp3-50", "nrp4-50", "nrp5-50", #6..10
"nrp1-70", "nrp2-70", "nrp3-70", "nrp4-70", "nrp5-70", #11..15
"nrp-e1-30", "nrp-e2-30", "nrp-e3-30", "nrp-e4-30", #16..19
"nrp-g1-30", "nrp-g2-30", "nrp-g3-30", "nrp-g4-30", #20..23
"nrp-m1-30", "nrp-m2-30", "nrp-m3-30", "nrp-m4-30", #24..27
"nrp-e1-50", "nrp-e2-50", "nrp-e3-50", "nrp-e4-50", #28..31
"nrp-g1-50", "nrp-g2-50", "nrp-g3-50", "nrp-g4-50", #32..35
"nrp-m1-50", "nrp-m2-50", "nrp-m3-50", "nrp-m4-50"  #36..39
)

# ETAPA DE TREINAMENTO
for (numInstancias_ in numInstanciasPorParticao) {
   remainder = 39 %% numInstancias_
   if (remainder == 0)
      numParticoes = (39 %/% numInstancias_)
   else
      numParticoes = (39 %/% numInstancias_) + 1
   particoes <- vector(mode="list", length=numParticoes)
   set.seed(0)
   instanciasPorSortear <- seq(1, 39, by=1)
   for (particao_ in 1:numParticoes) {
      if (remainder == 0) {
         numIntanciasSorteio = numInstancias_
      } else {
         if (particao_ == numParticoes)
            numIntanciasSorteio = remainder
         else
            numIntanciasSorteio = numInstancias_
      } # if (remainder == 0)...else
      if (length(instanciasPorSortear)==1)
         particoes[[particao_]] <- instanciasPorSortear
      else
         particoes[[particao_]] <- sample(instanciasPorSortear,
            ↪ numIntanciasSorteio)
```

```
      instanciasPorSortear <- instanciasPorSortear [!instanciasPorSortear %in%
         ↪ particoes[[particao_]]]
   } # for (particao_ in 1:numParticoes)
   for (particao_ in 1:numParticoes) {
      testText <- vector(mode="character", length=length(particoes[[particao
         ↪ _]]))
      testFile <- file("testing.txt")
      for (instancia_ in 1:length(particoes[[particao_]]))
         testText[instancia_] <- paste(instancias[particoes[[particao_]][
            ↪ instancia_]], ".txt", sep="")
      writeLines(testText, testFile)
      close(testFile)
      trainText <- vector(mode="character", length=(39 - length(particoes[[
         ↪ particao_]])))
      trainFile <- file("training.txt")
      posicao_  <- 1
      for (particaoTreinamento_ in 1:numParticoes) {
         if (particaoTreinamento_ != particao_) {
            for (instancia_ in 1:length(particoes[[particaoTreinamento_]])) {
               trainText[posicao_] <- paste(instancias[particoes[[
                  ↪ particaoTreinamento_]][instancia_]], ".txt", sep="")
               posicao_ <- posicao_ + 1

            } # for (instancia_ in 1:length(particoes[[particaoTreinamento_]]))
         } # if (particaoTreinamento_ != particao_)
      } # for (particaoTreinamento_ in 1:numParticoes)
      writeLines(trainText, trainFile)
      close(trainFile)
      if (numInstancias_ < 10)
         nesimaInstancia <- paste("0", numInstancias_, sep="")
      if (particao_ < 10)
         nesimaParticao  <- paste("0", particao_, sep="")
      subDir <- paste("./exec-dir/", "N-", nesimaInstancia, "_", "K-",
         ↪ nesimaParticao, sep="")
      dir.create(file.path(subDir), showWarnings = FALSE)
      file.copy("./testing.txt",       subDir)
      file.copy("./training.txt",      subDir)
      file.copy("./parameters.txt",    subDir)
      file.copy("./scenario.txt",      subDir)
      file.copy("./target-runner.bat", subDir)
      file.copy("./NRP-BRKGA.exe",     subDir)
      setwd(subDir)
      parameters <- readParameters("parameters.txt")
      scenario   <- readScenario("scenario.txt")
      irace(scenario = scenario, parameters = parameters)
      setwd("../../")
   } # for (particao_ in 1:numParticoes)
} # for (numInstancias_ in numInstanciasPorParticao)

# ETAPA DE TESTE
diretorios <- list.dirs("./exec-dir/", recursive=FALSE)
for (diretorio_ in diretorios) {
   setwd(diretorio_)
   testing.main(logFile = "./irace.Rdata")
   testingFiles <- row.names(file.info(list.files(path="./",pattern="t.*\\.
      ↪ stdout$", full.names=TRUE)))
```

```
   testingInfoText <- vector(mode="character", length=length(testingFiles))
   testingInfoFile <- file("../K-FOLD_TESTING.txt","a")
   dir.create(file.path("./TESTING"), showWarnings = FALSE)
   kFoldInfo <- paste(substr(diretorio_, 13, 21), substr(diretorio_, 15, 16),
       ↪ substr(diretorio_, 20, 21), sep="; ")
   contador <- 0
   for (testingFile_ in testingFiles) {
      testingFile = file(testingFile_, "r")
      firstLine = readLines(testingFile, n=1)
      close(testingFile)
      contador <- contador + 1
      testingInfoText[contador] <- paste(kFoldInfo, firstLine, sep="; ")
      file.copy(testingFile_, "./TESTING")
      file.remove(testingFile_)
   } # for (testingFile_ in testingFiles)
   writeLines(testingInfoText, testingInfoFile)
   close(testingInfoFile)
   load("irace.Rdata")
   percentualElite    <- iraceResults$allConfigurations$percentualElite_pe[
       ↪ iraceResults$state$eliteConfigurations[[1]][1]]
   percentualMutantes <- iraceResults$allConfigurations$percentualMutantes_pm[
       ↪ iraceResults$state$eliteConfigurations[[1]][1]]
   probabHerancaElite <- iraceResults$allConfigurations$probabHerancaElite_rhoe
       ↪ [iraceResults$state$eliteConfigurations[[1]][1]]
   idConfig           <- iraceResults$allConfigurations$.ID.[iraceResults$state
       ↪ $eliteConfigurations[[1]][1]]
   bestConfigInfo <- paste(percentualElite, percentualMutantes,
       ↪ probabHerancaElite, idConfig, sep="; ")
   bestConfigFile <- file("../K-FOLD_BESTCONFIG.txt", "a")
   bestConfigText <- paste(kFoldInfo, bestConfigInfo, sep="; ")
   writeLines(bestConfigText, bestConfigFile)
   close(bestConfigFile)
   setwd("../../")
} # for (diretorio_ in diretorios)
```

A.2 *K-Fold Cross Validation* e Configuração Automática de Parâmetros sem IRACE

```
library(irace)
diretorios <- list.dirs("./exec-dir/", recursive=FALSE)
for (diretorio_ in diretorios) {

# ETAPA DE TREINAMENTO
   kFoldInfo <- paste(substr(diretorio_, 13, 21), substr(diretorio_, 15, 16),
       ↪ substr(diretorio_, 20, 21), sep="; ")
   setwd(diretorio_)
   load("irace.Rdata")
   dir.create(file.path("./RANDOM"), showWarnings = FALSE)
   file.copy("./target-runner.bat", "./RANDOM")
   file.copy("./NRP-BRKGA.exe",     "./RANDOM")
   setwd("./RANDOM")
   candidate_df <- data.frame(candidate=character(), percElite_pe=double(),
       ↪ percMutantes_pm=double(), probHerancaElite_rhoe=double(),
       ↪ stringsAsFactors=FALSE)
   randomLog_df <- data.frame(fitness=double(), candidate=character(), instance
       ↪ _id=character(), seed=double(), percElite_pe=double(), percMutantes_
       ↪ pm=double(), probHerancaElite_rhoe=double(), stringsAsFactors=FALSE)
   numConfigur <- length(iraceResults$experiments[1,])
   numInstSeed <- length(iraceResults$experiments[,1])
   for (candidate in 1:numConfigur) {
      percElite_pe          <- round(runif(1, 0.10, 0.30),2)
      percMutantes_pm       <- round(runif(1, 0.05, 0.30),2)
      probHerancaElite_rhoe <- round(runif(1, 0.50, 0.80),2)
      candidate_line <- list(candidate=candidate, percElite_pe=percElite_pe,
          ↪ percMutantes_pm=percMutantes_pm, probHerancaElite_rhoe=
          ↪ probHerancaElite_rhoe)
      candidate_df   <- rbind(candidate_df, candidate_line, stringsAsFactors=
          ↪ FALSE)
   } # for (candidate in 1:numConfigur)
   for (candidate in 1:numConfigur) {
      for (instSeed_ in 1:numInstSeed) {
         if (!is.na(iraceResults$experiments[instSeed_,candidate])) {
            instance_id <- iraceResults$state$.irace$instancesList[instSeed_,1]
            instance <- iraceResults$scenario$instances[instance_id]
            seed <- iraceResults$state$.irace$instancesList[instSeed_,2]
            percElite_pe <- candidate_df$percElite_pe[candidate]
            percMutantes_pm <- candidate_df$percMutantes_pm[candidate]
            probHerancaElite_rhoe <- candidate_df$probHerancaElite_rhoe[
                ↪ candidate]
            candidate_parameters <- paste("-e", percElite_pe, "-m",
                ↪ percMutantes_pm, "-r",probHerancaElite_rhoe,sep=" ")
            targerRunnerParameters <- paste(candidate, instance_id, seed,
                ↪ instance, candidate_parameters, sep=" ")
            fitness <- system (paste("target-runner.bat",
                ↪ targerRunnerParameters, sep=" "), intern = TRUE)
            fitness <- substr(fitness,1,8)
            filename <- paste("c", candidate, "-",instance_id,".stdout",sep="")
            filestream <- file(filename, "r")
            firstLine <- readLines(filename, n=1)
            close(filestream)
            filestreamRandomLog <- file("../../K-FOLD_RANDOM_LOG.txt", "a")
```

```
            infoTextRandomLog <- paste(kFoldInfo, firstLine, candidate,instance
                ↪ _id, percElite_pe, percMutantes_pm, probHerancaElite_rhoe,
                ↪ sep="; ")
            writeLines(infoTextRandomLog, filestreamRandomLog)
            close(filestreamRandomLog)
            randomLog_line <- list(fitness=fitness,candidate=candidate,instance
                ↪ _id=instance_id, seed=seed, percElite_pe=percElite_pe,
                ↪ percMutantes_pm=percMutantes_pm, probHerancaElite_rhoe=
                ↪ probHerancaElite_rhoe)
            randomLog_df <- rbind(randomLog_df, randomLog_line,
                ↪ stringsAsFactors=FALSE)
         } # if (!is.na(iraceResults$experiments[instSeed_,candidate]))
      } # for (instSeed_ in 1:numInstSeed)
   } # for (candidate in 1:numConfigur)
   dir.create(file.path("./TESTING"), showWarnings = FALSE)
   theBest_df <- randomLog_df[order(randomLog_df$fitness, decreasing=FALSE),]
   theBest_df <- theBest_df[1,]

   candidate             <- theBest_df$candidate
   seed                  <- theBest_df$seed
   percElite_pe          <- theBest_df$percElite_pe
   percMutantes_pm       <- theBest_df$percMutantes_pm
   probHerancaElite_rhoe <- theBest_df$probHerancaElite_rhoe
   filestreamBestConfig <- file("../../K-FOLD_RANDOM_BESTCONFIG.txt","a")
   infoTextBestConfig <- paste(percElite_pe, percMutantes_pm, probHerancaElite_
       ↪ rhoe, candidate, sep="; ")
   infoTextBestConfig <- paste(kFoldInfo, infoTextBestConfig, sep="; ")
   writeLines(infoTextBestConfig, filestreamBestConfig)
   close(filestreamBestConfig)
# ETAPA DE TESTE
   for (testInstance_ in 1:length(iraceResults$scenario$testInstances)) {
      instance_id <- names(iraceResults$scenario$testInstances[testInstance_])
      instance <- iraceResults$scenario$testInstances[[testInstance_]]
      candidate_parameters <- paste("-e", percElite_pe, "-m", percMutantes_pm,
          ↪ "-r", probHerancaElite_rhoe, sep=" ")
      targerRunnerParameters <- paste(candidate, instance_id, seed, instance,
          ↪ candidate_parameters, sep=" ")
      fitness <- system (paste("target-runner.bat", targerRunnerParameters, sep
          ↪ =" "), intern=TRUE)
      fitness <- substr(fitness,1,8)
      filename <- paste("c", candidate, "-", instance_id, ".stdout",sep="")
      filestream <- file(filename, "r")
      firstLine  <- readLines(filestream, n=1)
      close(filestream)
      file.copy(filename, "./TESTING")
      file.remove(filename)
      filestreamRandomTest <- file("../../K-FOLD_RANDOM_TESTING.txt","a")
      infoTextRandomTest <- paste(kFoldInfo, firstLine, candidate, instance_id,
          ↪  percElite_pe, percMutantes_pm, probHerancaElite_rhoe, sep="; ")
      writeLines(infoTextRandomTest, filestreamRandomTest)
      close(filestreamRandomTest)
   } # for (testInstance_ in 1:length(iraceResults$scenario$testInstances))
   setwd("../../../")

} # for (diretorio_ in diretorios)
```