CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Mestrado em Modelagem Matemática e Computacional

PROBLEMA GENERALIZADO DE ATRIBUIÇÃO: CONTRIBUIÇÕES AO ESTUDO DE ALGORITMOS MONO E MULTIOBJETIVOS

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno: Robert Fabrício Subtil

Orientador: Prof. Dr Marcone Jamilson Freitas Souza

Co-Orientador: Prof. Dr Sérgio Ricardo de Souza

Belo Horizonte - MG
Junho de 2012

Agradecimento

Agradeço primeiramente a Deus pelo dom da vida.

Ao Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) e ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, pela oportunidade de realizar o curso.

Ao Professor Marcone Jamilson Freitas Souza pela brilhante orientação e pelos ensinamentos tão valiosos. Marcone, muito obrigado por me ensinar a enxergar a área de otimização com novas perspectivas!

Ao meu co-orientador, Professor Sérgio Ricardo de Souza, pelos conselhos e amizade. Sérgio, muito obrigado por nossas conversas, realmente fizeram a diferença em diversos momentos.

Ao Professor Eduardo Gontijo Carrano pelos ensinamentos e trabalhos realizados em conjunto, foram oportunidades ímpares de amadurecimento na minha formação em pesquisa.

Aos professores do Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, pelos conhecimentos transmitidos.

A todos os amigos do CEFET, em especial à Gleice Mônica, Mayra e Dayanne, pelo companheirismo e trocas de experiências.

Aos amigos Diego Mello e Júlio Alves, pelas conversas e discussões em torno de otimização, mercado e pesquisa. Muito obrigado, vocês contribuíram diretamente neste trabalho.

Aos meus Pais: Valdir Subtil e Maria da Penha Subtil. Hoje, mais do que nunca, vejo o quanto são importantes para a minha formação.

Aos meus irmãos e cunhados pelo incentivo.

À minha esposa Andreza, pelo amor, incentivo, dedicação, doação e compreensão durante esse período que foi tão difícil, porém, valioso. Andreza, você é a grande responsável por tudo isso! Te amo!

Agradeço ainda, sem citar nomes, a todos aqueles que de alguma forma contribuíram para a realização deste trabalho.

Resumo

Este trabalho trata o Problema Generalizado de Atribuição (PGA), o qual é um dos mais representativos problemas de Otimização Combinatória. O PGA consiste em encontrar o menor custo para a atribuição de n tarefas a m agentes, sendo que cada tarefa deve ser atribuída a apenas um agente, sujeito à capacidade dos mesmos. Para resolvê-lo foram desenvolvidos cinco algoritmos heurísticos híbridos que combinam os procedimentos *Simulated Annealing*, Descida em Vizinhança Variável, Busca Tabu com Relaxação Adaptativa, Reconexão por Caminhos e *Ejection Chain*. Foram realizados experimentos computacionais que demonstraram a efetividade dos algoritmos propostos em relação a algumas abordagens presentes na literatura. Além disso, foram realizados testes estatísticos para verificar diferenças na qualidade das soluções apresentadas pelos algoritmos propostos. Neste trabalho também é tratada uma variação multiobjetivo do PGA, na qual é adicionada uma função de equilíbrio que tenta balancear a atribuição das tarefas entre os agentes. Para tal variação do problema, foi proposto um algoritmo evolucionário multiobjetivo baseado no NSGA-II. Foram realizados testes que comprovaram que a proposta é capaz de encontrar boas soluções aproximadas de Pareto.

PALAVRAS-CHAVE: Problema Generalizado de Atribuição, *Simulated Annealing*, Descida em Vizinhança Variável, Busca Tabu, Relaxação Adaptativa, Reconexão por Caminhos, *Ejection Chain*, Otimização Multiobjetivo, NSGA-II.

Abstract

This work deals the Generalized Assignment Problem (GAP), which is one of the most representative combinatorial optimisation problems. The GAP consists in finding the lowest cost to the assignment of n tasks to m agents, and each task should be assigned to only one agent, subject to their capabilities. To solve this problem was developed five hybrids algorithms which combines the procedures Simulated Annealing, Variable Neighborhood Descent, Tabu Search with Adaptive Relaxation, Path Relinking and Ejection Chain. We performed computational experiments which demonstrated the effectiveness of proposed algorithms compared with some approaches in the literature. Furthermore, statistical tests were performed to verify differences in the quality of solutions presented by the proposed algorithms. This work also treated a multiobjective variation of the GAP, which is added an equilibrium function trying to balance the allocation of tasks among agents. To this variation problem, we propose a multiobjective evolutionary algorithm based on NSGA-II. Tests were conducted which demonstrated that the proposal is able to find good approximate Pareto solutions.

KEY-WORDS: Generalized Assignment Problem, Simulated Annealing, Variable Neighborhood Descent, Tabu Search, Adaptive Relaxation, Path Relinking, Ejection Chain, Multiobjective Optimization, NSGA-II.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O Problema Generalizado de Atribuição (PGA) é um problema clássico de Otimização Combinatória. De uma forma geral, o problema pode ser estabelecido como: um conjunto de m agentes deve ser atribuído a um conjunto de n tarefas com um custo de atribuição associado. É necessário executar todas as tarefas, atribuindo-se a cada uma delas apenas um agente e respeitando-se a capacidade de cada agente, de tal forma que o custo total da atribuição seja minimizado.

Este problema tem grande importância prática, estando presente, por exemplo, no simples planejamento diário das tarefas de uma equipe, em que as tarefas devem ser distribuídas de modo a aproveitar melhor o tempo. No entanto, tal problema também aparece em contextos mais complexos e de difícil resolução, como: atribuição de tarefas em redes de computadores (Balachandran, 1976), localização de facilidades (Ross e Soland, 1977), carregamento de contêineres (Hung e Fisk, 1979), escalonamento de tarefas em máquinas paralelas (Lenstra et al., 1990), roteamento de veículos (Fisher e Jaikumar, 2006), comparação de estoque/documentos (Dawande e Kalagnanam, 1998), escalonamento de recursos (Mazzola et al., 1989), escalonamento de multiprocessadores (Turek et al., 1992), dentre inúmeros outros. Além disto, existem muitos problemas de decisão que, caso não sejam diretamente um problema generalizado de atribuição, contêm um problema de atribuição como um subproblema.

O PGA faz parte da classe de problemas NP-Difíceis (Sahni e Gonzalez, 1976; Chu e Beasley, 1997), o que significa que ainda não existem algoritmos de complexidade polinomial para resolvê-lo na otimalidade. Essa característica torna o PGA interessante do ponto de vista de pesquisa, pois encontrar algoritmos que resultem em soluções eficientes (boa qualidade e com o mínimo de tempo computacional) para tal problema constitui um grande desafio.

Além da formulação clássica mono-objetivo, o presente trabalho apresenta uma variação do PGA ao introduzir uma segunda função objetivo ao problema, denominada Função de Equilíbrio. Essa função é representada pela máxima diferença entre o consumo de recursos dos agentes, de modo que sua minimização provê a distribuição homogênea das tarefas. Tal função objetivo pode ser útil em casos práticos, como:

(i) No gerenciamento de projetos, onde algumas tarefas devem ser atribuídas a diferentes integrantes de uma equipe. Se um grupo de pessoas receber uma

quantidade excessiva de tarefas (e consequentemente outros integrantes permanecerem ociosos), a eficiência e a qualidade do trabalho realizado por tal grupo estarão comprometidas, devido à sobrecarga de tarefas;

(*ii*) Em redes de computadores, onde é importante manter o equilíbrio da carga afim de melhorar o desempenho global da rede. A alocação adequada de tarefas em tais redes reduz a chance de ocorrer sobrecargas e, consequentemente, aumenta suas taxas de transferência de dados. O balanceamento de cargas em redes de computadores é comumente empregado em sites populares, servidores NNTP (*Network News Transfer Protocol*), servidores DNS (*Domain Name System*), dentre outros;

(*iii*) Em computação paralela síncrona, na qual o balanceamento das tarefas reduz o tempo ocioso de processadores e, consequentemente, diminui o tempo total para execução de alguma tarefa paralelizável.

O interesse deste trabalho está voltado ao PGA na sua formulação clássica mono-objetivo, bem como à sua versão multiobjetivo composta pela Função de Equilíbrio citada anteriormente.

Para resolver o problema mono-objetivo, foram propostos cinco algoritmos híbridos combinando as técnicas *Simulated Annealing* (SA), Descida em Vizinhança Variável (*Variable Neighborhood Descent* - VND), Busca Tabu com Relaxação Adaptativa (BTRA), Reconexão por Caminhos (RC) e *Ejection chain* (EC).

Para o problema multiobjetivo, foi utilizado o método NSGA-II como base para o algoritmo propsoto a esse problema.

1.1 Objetivos

1.1.1 Geral

O objetivo geral deste trabalho é estudar o PGA, propor algoritmos heurísticos para resolução desse problema, bem como analisar tais algoritmos por meio de experimentos computacionais e comparações com outras abordagens presentes na literatura, no caso do problema mono-objetivo.

Além disso, propõe-se tratar o caso multiobjetivo com a introdução de uma Função de Equilíbrio, e desenvolver algoritmos heurísticos para gerar soluções candidatas à Fronteira de Pareto.

1.1.2 Específico

Para que o objetivo geral fosse alcançado, foi necessário que os seguintes objetivos específicos fossem atingidos:

- Estudo da literatura associada aos principais temas envolvidos no trabalho, quais sejam: Problema Generalizado de Atribuição; Algoritmos aplicados à soluções de tal problema; Otimização Multiobjetivo; Métodos para resolução de Problemas Multiobjetivos;

- Estudo e implementação de métodos heurísticos mono e multiobjetivos aplicando-os às formulações clássica e multiobjetivo do PGA;
- Comparação, quando possível, dos resultados obtidos pelos métodos propostos com resultados conhecidos na literatura.

1.2 Justificativa

O estudo do Problema Generalizado de Atribuição é motivado pela possibilidade de se modelar diversos problemas práticos por meio de sua estrutura. Sua importância advém não apenas de sua aplicabilidade na forma direta, mas também do fato de aparecer como subestrutura de vários outros problemas.

Do ponto de vista teórico, o estudo do PGA é motivado por sua complexidade computacional, tendo em vista que pertence à classe de problemas da classe NP-difícil e, portanto, se opõe à utilização de métodos polinomiais para resolvê-lo na otimalidade. Existem, porém, algumas abordagens as quais fazem uso de métodos exatos ((Narciso e Lorena, 1999), (Guignard e Rosenwein, 1989)); no entanto, estes métodos são aplicados às instâncias com tamanhos limitados. Por esse motivo, o estudo e a utilização de heurísticas para resolver problemas dessa natureza vem crescendo cada vez mais.

Além disso, não foram encontrados na literatura trabalhos com a versão multiobjetivo abordada, de modo que se torna justificado o estudo em torno dessa variação do problema.

Assim, o interesse deste trabalho foi realizar um estudo de métodos heurísticos para resolver o PGA nas duas versões (mono e multiobjetivo), de modo a fazer comparações de resultados, quando possível, com outros métodos presentes na literatura e, com isso, contribuiu com a pesquisa de métodos aplicados à problemas de natureza combinatória.

1.3 Organização do Trabalho

O restante do trabalho está organizado como segue.

No Capítulo 2 é apresentado o problema estudado. Para tanto, a formulação matemática do PGA é descrita e discutida. Ainda nesse capítulo, é apresentada a versão multiobjetivo proposta, bem como a sua formulação matemática.

Uma revisão bibliográfica em torno das abordagens presentes na literatura para tratar o PGA é feita no Capítulo 3. Além disso, tal capítulo apresenta os métodos heurísticos utilzados neste trabalho para abordar o problema mono-objetivo. Conceitos sobre Otimização Multiobjetivo também são apresentados nesse capítulo, bem como é feita a revisão sobre o método NSGA-II, o qual foi utilizado como base para abordar a variação multiobjetivo do PGA.

No Capítulo 4 são apresentadas as metodologias propostas neste trabalho para resolver o problema mono-objetivo e multiobjetivo.

No Capítulo 5 são descritos os problemas-teste da literatura utilizados para verificar a eficiência dos métodos, bem como os testes realizados e os resultados obtidos

para o PGA mono-objetivo e multiobjetivo. Por fim, no capítulo 6 são apresentadas as considerações finais a respeito do trabalho e as propostas para trabalhos futuros.

Capítulo 2

Problema Generalizado de Atribuição

O Problema Generalizado de Atribuição (PGA) é um problema clássico de Otimização Combinatória, que consiste em atribuir n tarefas a m agentes ao menor custo possível, de modo que cada tarefa seja atribuída a apenas um único agente e cada agente, por sua vez, não exceda sua capacidade máxima(Balachandran, 1976).

Neste trabalho também é proposta uma variação à formulação clássica do PGA, na qual é adicionada uma função objetivo denominada Função de Equilíbrio que, quando minimizada, resulta na distribuição homogênea das tarefas entre os agentes. Assim, o novo problema, denominado moPGA, é composto por duas funções objetivo, custo e equilíbrio, caracterizando-o como um problema de otimização multiobjetivo.

A formulação matemática do PGA mono-objetivo é apresentada na seção 2.1, enquanto a variação moPGA é descrita na seção 2.2.

2.1 O PGA Mono-objetivo

Sejam um conjunto $\mathcal{A}$ de agentes, um conjunto $\mathcal{T}$ de tarefas, c_{ij} o custo de atribuir a tarefa $j \in \mathcal{T}$ ao agente $i \in \mathcal{A}$, a_{ij} a quantidade de recursos necessários ao agente $i \in \mathcal{A}$ para desempenhar a tarefa $j \in \mathcal{T}$ e b_i a capacidade do agente $i \in \mathcal{A}$. Considerando x_{ij} a variável binária de decisão, que assume o valor "1" se a tarefa $j \in \mathcal{T}$ é atribuída ao agente $i \in \mathcal{A}$ ou "0", caso contrário, então o PGA pode ser modelado matematicamente como (Yagiura et al., 2006):

$$f_C(x) = \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{T}} c_{ij} \cdot x_{ij} \tag{2.1}$$

$$\text{sujeito a : } \begin{cases} \sum_{j \in \mathcal{T}} a_{ij} \cdot x_{ij} \leq b_i & , \forall i \in \mathcal{A} \\ \sum_{i \in \mathcal{A}} x_{ij} = 1 & , \forall j \in \mathcal{T} \\ x_{ij} \in \{0, 1\} & , \forall i \in \mathcal{A},\ \forall j \in \mathcal{T} \end{cases} \tag{2.2}$$

Nesta formulação, a Eq. (2.1) é a função de custo que deve ser minimizada. No segundo conjunto de equações (2.2), a primeira restrição assegura que a capacidade

dos agentes não é violada, a segunda garante que cada tarefa é atribuída a apenas um agente e a terceira restrição define que cada variável x_{ij} é binária.

2.2 O PGA Multiobjetivo

A versão multiobjetivo considerada do PGA (moPGA) consiste no problema generalizado de atribuição de tarefas no qual se deseja encontrar o menor custo das atribuições e, ao mesmo tempo, distribuir as tarefas de maneira homogênea entre os agentes, respeitando as mesmas restrições de indivisibilidade das tarefas e capacidade dos agentes presentes na formulação clássica do PGA. Para tanto, um segundo objetivo, denominado Função de Equilíbrio, é introduzido, com vistas à minimização da diferença no consumo de recursos entre os agentes mais requisitado e menos requisitado. A função de equilíbrio é descrita pela equação (2.3).

$$f_E(x) = \max_{i \in \mathcal{A}} \left(\sum_{j \in \mathcal{T}} a_{i,j} \cdot x_{i,j} \right) - \min_{i \in \mathcal{A}} \left(\sum_{j \in \mathcal{T}} a_{i,j} \cdot x_{i,j} \right) \tag{2.3}$$

Finalmente, o Problema Generalizado de Atribuição Multiobjetivo (moPGA) pode ser formulado como segue:

$$\mathcal{X}^* = \arg\min_{x} \begin{bmatrix} f_C(x) \\ f_E(x) \end{bmatrix} \tag{2.4}$$

$$\text{sujeito a} : \begin{cases} (G1) : \sum_{j \in \mathcal{T}} a_{i,j} \cdot x_{i,j} - b_i \leq 0 & , \ \forall i \in \mathcal{A} \\ (G2) : \sum_{i \in \mathcal{A}} x_{i,j} - 1 = 0 & , \ \forall j \in \mathcal{T} \\ (G3) : x_{i,j} \in \{0, 1\} & , \ \forall i \in \mathcal{A}, \ \forall j \in \mathcal{T} \end{cases} \tag{2.5}$$

Nesta formulação, a equação (2.4) define as funções de custo e equilíbrio, respectivamente. O equilíbrio é modelado como a diferença entre o agente mais ocupado e o agente mais ocioso. Tal diferença deve ser minimizada. Assim como na formulação clássica mono-objetivo, em (2.5), as restrições $g_1(x)$ asseguram que a máxima capacidade de cada agente não seja violada, as restrições $g_2(x)$ garantem que cada tarefa é atribuída a apenas um agente e as restrições $g_3(x)$ definem que cada variável $x_{i,j}$ seja binária. Uma característica importante desta formulação é que a mesma não pode ser resolvida por um método de programação linear inteira, uma vez que $f_E(\cdot)$ é não-linear.

Capítulo 3

Revisão de Literatura

Este capítulo é dedicado à revisão bibliográfica em torno do tema proposto neste trabalho. Inicialmente, na seção 3.1 são apresentadas algumas abordagens presentes na literatura aplicadas ao PGA na sua formulação clássica mono-objetivo. Na seção 3.2 são feitas revisões dos métodos heurísticos aplicados neste trabalho para o problema mono-objetivo. Na seção 3.3 os conceitos de Otimização Multiobjetivo são revisados. Por fim, a seção 3.4 apresenta o método escolhido como base para resolver o problema de otimização multiobjetivo abordado.

3.1 Trabalhos relacionados

Esta seção apresenta algumas das técnicas já propostas para o Problema Generalizado de Atribuição.

No trabalho de Martello e Toth (1990) é proposto um método heurístico de duas fases. Na primeira fase, uma solução inicial é construída usando uma função gulosa, que guia o processo construtivo. Na segunda fase tal solução inicial é melhorada usando um simples procedimento de troca.

Cattrysse et al. (1994) reformulam o PGA como um problema de particionamento de conjuntos. Uma heurística baseada em tal reformulação é proposta nesse trabalho. Cada coluna representa uma atribuição factível de tarefas a agentes. Para cada agente, colunas são geradas resolvendo o problema da mochila no qual os coeficientes são obtidos a partir do vetor de variáveis duais da relaxação da programação linear. Uma vez que esse problema é degenerado, um procedimento de subida dual é aplicado para obter as variáveis duais. Então, um procedimento de otimização por subgradiente é utilizado para melhorar o limitante inferior.

Osman (1995) propõe um procedimento heurístico híbrido que combina os métodos *Simulated Annealing* e Busca Tabu. Tal procedimento utiliza um mecanismo λ-*generation* o qual descreve como uma solução pode ser alterada para gerar outra solução vizinha, além de utilizar a estratégia de ocilação da Busca Tabu no regime de resfriamento do *Simulated Annealing* afim de induzir um comportamento oscilatório nos valores da temperatura. Além disso, uma Busca Tabu também é proposta para resolver o PGA.

Um algoritmo genético é proposto por Chu e Beasley (1997). A representação da solução é dada por um vetor de inteiros no qual cada elemento identifica uma

atribuição. Avaliações de aptidão e não-aptidão são utilizadas para lidar com soluções factíveis e infactíveis. Além de utilizar operadores de mutação e cruzamento, um operador heurístico de duas-fases também é utilizado: na primeira fase este operador tenta recuperar a viabilidade através da redução da pontuação da inaptidão do indivíduo. Na segunda fase tenta melhorar o custo dessa solução sem violar a restrição de capacidade.

Yagiura et al. (1999b) propuseram um algoritmo de Busca em Profundidade Variável para o PGA. Eles incorporaram movimentos adaptativos de troca e mudança onde alguns desses movimentos são proibidos a fim de evitar ciclagem. O método também permite que a busca visite soluções infactíveis modificando a função objetivo para penalizar as capacidades violadas dos agentes. Em Yagiura et al. (1998), um processo de *branching search* para construir a vizinhança é incorporado a fim de melhorar o desempenho do algoritmo proposto em Yagiura et al. (1999b).

Em Yagiura et al. (1999a) é proposto um algoritmo de Busca Tabu em que *ejection chains* são utilizados para criar movimentos mais complexos e poderosos. O algoritmo mantém um balanço entre visitas a regiões factíveis e infactíveis usando um mecanismo automático de ajuste do parâmetro de penalidade na função objetivo.

Em Diaz e Fernandez (2001) também é proposto um algoritmo de Busca Tabu para resolver o PGA. Os autores apresentam uma formulação relaxada que permite a busca ultrapassar os limties de factibilidade utilizando uma penalidade na função objetivo, assim como em Yagiura et al. (1999a). Um esquema de oscilação estratégica é utilizado para permitir a altenância entre soluções factíveis e infactíveis afim de proporcionar à busca um nível maior de flexibilidade. Estratégias de intensificação e diversificação também são implementadas por meio de memória baseada em frequência.

Yagiura et al. (2006) apresentam uma abordagem através do método de Reconexão por Caminhos associado à Busca Tabu que fornece um mecanismo para a geração de novas soluções pela combinação de duas ou mais soluções de referência. Este algoritmo também apresenta uma abordagem de *ejection chains*, onde Relaxação Lagrangiana fornece informações ajustadas de custo para orientar a busca na vizinhança por soluções promissoras. Além disso, é incorporado um mecanismo automático de ajuste de parâmetros da busca com o intuito de manter um equilíbrio nas visitas a regiões factíveis e infactíveis.

Mais recentemente, Woodcock e Wilson (2010) propuseram a combinação do método exato *Branch & Bound* com a estratégia heurística de Busca Tabu. O algoritmo descrito no trabalho utiliza um software comercial para resolver os subproblemas gerados a partir da estratégia heurística. A abordagem de Busca Tabu faz uso do conceito *Referent Domain Optimization* descrito em Glover e Laguna (1997). Além disso, a relaxação linear do PGA, que é parte da abordagem *Branch & Bound*, é capaz de sugerir quais variáveis podem assumir valores binários.

3.2 Métodos Heurísticos

Conforme dito anteriormente, o PGA é de natureza combinatória e pertence à classe de problemas NP-difíceis, ou seja, não é conhecido ainda nenhum algoritmo de complexidade polinomial que o resolva. Desta maneira, a aplicação de métodos

heurísticos tem sido muito comum dentre os trabalhos encontrados na literatura. Seguindo essa linha de pesquisa, o presente trabalho se concentra na aplicação e combinação de métodos heurísticos para encontrar soluções eficientes para o PGA. Desta maneira, as próximas seções apresentam os métodos utilizados neste trabalho.

3.2.1 *Simulated Annealing*

O *Simulated Annealing* (SA), proposto por Kirkpatrick et al. (1983), é um método de busca local probabilístico, que se fundamenta em uma analogia com a termodinâmica ao simular o resfriamento de um conjunto de átomos aquecidos, operação conhecida como recozimento.

Para se obter metais mais estáveis, estruturalmente fortes e de menor energia, eles são aquecidos a uma temperatura suficientemente alta e depois são lentamente resfriados. Durante o recozimento, o material passa por vários estados possíveis.

Devido à analogia com o processo de recozimento dos metais, as soluções do espaço de busca correspondem aos possíveis estados de um metal. A energia de cada estado é representada pelo valor da função objetivo e, por fim, o valor de um ótimo local (possivelmente global) corresponde à energia mínima.

O método parte de uma solução inicial qualquer e cada iteração consta basicamente de um *loop*. Em cada iteração deste *loop* interno é gerado aleatoriamente um único vizinho s' da solução corrente s. Seja $\Delta = f(s') - f(s)$ e considere que o problema seja de minimização. Se $\Delta < 0$ a solução vizinha passa a ser a nova solução corrente. Caso contrário ($\Delta \geq 0$), o vizinho pode ser aceito com uma probabilidade $e^{\frac{-\Delta}{kT}}$, sendo k a constante de Boltzmann e T a temperatura corrente.

A temperatura T assume inicialmente um valor suficientemente alto T_0. A cada iteração i do método, a temperatura é diminuída, ou seja, $T_i \leftarrow \alpha \cdot T_{i-1}$, com $0 < \alpha < 1$. Os valores de T_0 e α são parâmetros do método.

É possível notar que quanto maior a temperatura, maior a probabilidade de se aceitar uma solução de piora. Desta maneira, a cada iteração do algoritmo a chance de uma solução de piora ser aceita é reduzida. As soluções de piora são aceitas para que se escape das armadilhas dos ótimos locais.

O término do método se dá quando a temperatura atinge um valor próximo de zero e somente vizinhos de melhora são aceitos, evidenciando o encontro de um ótimo local.

O Algoritmo 1 descreve o SA básico para o problema de minimização de uma função f. A estrutura de vizinhança N, a razão de resfriamento α, o número de iterações para cada temperatura ($SAmax$), a temperatura inicial T_0, um número real próximo de zero (ϵ) e uma solução inicial s são as entradas do método.

Mais detalhes sobre o método podem ser encontrados em (Kirkpatrick et al., 1983), (Ribeiro, 1996), e (Souza, 2009).

3.2.2 Descida em Vizinhança Variável (VND)

Proposto por Mladenovic e Hansen (Mladenović e Hansen, 1997), o VND é um procedimento de busca local que explora o espaço de soluções por meio da troca sistemática de estruturas de vizinhança.

Algoritmo 1: *Simulated Annealing.*

procedimento $SA(f(.), N(.), \alpha, SAmax, T_0, \epsilon, s)$
$s^* \leftarrow s$; /* Melhor solução obtida */
$IterT \leftarrow 0$; /* Número de iterações na temperatura T */
$T \leftarrow T_0$; /* Temperatura corrente */
enquanto $(T > \epsilon)$ faça
  enquanto $(IterT < SAmax)$ faça
    $IterT \leftarrow IterT + 1$;
    Aleatoriamente, escolha $s' \in N(s)$;
    $\Delta \leftarrow f(s') - f(s)$;
    se $(\Delta < 0)$ então
      $s \leftarrow s'$;
      se $(f(s') < f(s^*))$ então $s^* \leftarrow s'$;
    senão
      Aleatoriamente, escolha $x \in [0, 1]$;
      se $(x < e^{\frac{-\Delta}{kT}})$ então $s \leftarrow s'$;
    fim-se
  fim-enquanto;
  $T \leftarrow \alpha \cdot T$;
  $IterT \leftarrow 0$
fim-enquanto;
Retorne s^*;
fim SA;

O procedimento parte de uma solução e de um número pré-definido de estruturas de vizinhança em uma dada ordem de exploração. A busca inicia na primeira vizinhança $N^{(1)}$. Para cada vizinhança $N^{(k)}$ é procurada a melhor solução vizinha da solução corrente. Se o vizinho encontrado for melhor que a solução corrente, então esse vizinho passa a ser a nova solução corrente e o procedimento continua a exploração do espaço de soluções usando a primeira estrutura de vizinhança; caso contrário, o procedimento passa para a próxima vizinhança. O procedimento é interrompido quando é atingida a última estrutura de vizinhança sem melhora na solução corrente.

O pseudocódigo do VND é dado pelo Algoritmo 2. Nele é considerado o problema de minimização de uma função f, dados um conjunto de r estruturas diferentes $N = \{N^{(1)}, N^{(2)}, \cdots, N^{(r)}\}$ e uma solução inicial s.

Detalhes sobre o funcionamento do VND podem ser encontrados em (Mladenović e Hansen, 1997) e (Hansen e Mladenovirć, 2003).

3.2.3 Busca Tabu

A Busca Tabu (BT) é uma metaheurística baseada em busca local proposta em Glover (1986).

Dadas uma estrutura de vizinhança e uma solução inicial, a BT consiste basicamente em, iterativamente, explorar a vizinhança atual e mudar da solução corrente para o melhor vizinho encontrado, mesmo que o melhor vizinho não represente uma

Algoritmo 2: Descida em Vizinhança Variável.

procedimento $VND(f(.), N(.), r, s)$
Seja r o número de estruturas diferentes de vizinhança;
$k \leftarrow 1$; `/* Tipo de estrutura de vizinhança corrente */`
enquanto $(k \leq r)$ faça
  Encontre o melhor vizinho $s' \in N^{(k)}(s)$;
  se $(f(s') < f(s))$ então
    $s \leftarrow s'$;
    $k \leftarrow 1$
  senão
    $k \leftarrow k + 1$
  fim-se
fim-enquanto;
Retorne s;
fim VND;

solução de melhora. Isto é feito até que um critério de parada seja satisfeito.

A estratégia de aceitar soluções de piora é utilizada para escapar das armadilhas dos ótimos locais. Porém, esta tática pode fazer com que o algoritmo cicle, isto é, o algoritmo pode voltar a uma solução já gerada anteriormente. Por exemplo, quando a solução corrente é um ótimo local, seu melhor vizinho não será de melhora e será aceito. Na iteração seguinte, o ótimo local será novamente gerado por ser o melhor vizinho da solução corrente.

Para evitar ciclagem, o ideal seria armazenar todas as soluções já geradas pelo método em uma lista, denominada lista tabu. No entanto, isto é computacionalmente inviável. Uma alternativa é armazenar apenas as últimas $|LT|$ soluções geradas. Mas uma lista deste tamanho evitaria ciclos de até $|LT|$ iterações, além de poder ser inviável armazenar $|LT|$ soluções e testar se uma solução está ou não na lista tabu.

Na prática, o mais comum é criar uma lista tabu de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos $|LT|$ movimentos realizados, sendo $|LT|$ um parâmetro do método, e funciona como uma fila FIFO (*First-In-First-Out*).

Apesar de reduzir o risco de ciclagem, uma lista tabu de movimentos pode ser muito restritiva (impede o retorno a uma solução já gerada anteriormente e também que novas soluções sejam geradas). Para tentar corrigir este problema, usa-se um critério de aspiração, que retira sob certas condições, o *status* tabu de um movimento. Um exemplo de critério de aspiração é aceitar um movimento, mesmo que tabu, se ele melhorar o valor da função objetivo global (critério de aspiração por objetivo). Outra possibilidade é realizar o movimento tabu mais antigo se todos os possíveis movimentos forem tabus (aspiração por *default*).

Geralmente, o método é interrompido após um número máximo de iterações $BTmax$ sem melhora na melhor solução encontrada.

O Algoritmo 3 descreve a BT básica aplicada ao problema de minimização de uma função f. A estrutura de vizinhança N, a função de aspiração A, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração, o tamanho da lista

tabu $|LT|$, o número máximo de iterações sem melhora no valor da melhor solução e uma solução inicial são as entradas do método.

Algoritmo 3: Busca Tabu.

procedimento *Busca_Tabu*($f(.)$, $N(.)$, $A(.)$, $|V|$, $|LT|$, $BTmax$, s)
$s^* \leftarrow s$; /* Melhor solução obtida */
$Iter \leftarrow 0$; /* Contador do número de iterações sem melhora de s^* */
$LT \leftarrow \emptyset$; /* Lista tabu */
Inicialize a função de aspiração A;
enquanto ($Iter \leq BTmax$) faça
  $Iter \leftarrow Iter + 1$;
  Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subset N(s)$ tal que o movimento m não seja tabu ($m \notin LT$) ou s' atenda a condição de aspiração ($f(s') < A(f(s))$);
  Atualize a LT;
  $s \leftarrow s'$;
  se ($f(s) < f(s^*)$) então
    $s^* \leftarrow s$;
    $Iter \leftarrow 0$;
  fim-se
  Atualize A;
fim-enquanto;
Retorne s^*;
fim *Busca_Tabu*;

Mais informações a respeito da BT são encontradas em (Glover, 1986), (Glover, 1996), (Glover e Laguna, 1997), (Gendreau, 2003) e (Souza, 2009).

3.2.4 Reconexão por Caminhos

A Reconexão por Caminhos (RC) é uma estratégia que faz um balanço entre intensificação e diversificação. Foi proposta em Glover (1996) para ser utilizada em conjunto com a Busca Tabu (Vide Seção 3.2.3). Considerando um par de soluções, sendo uma delas a solução base e a outra solução guia, o objetivo deste método é partir da solução base e caminhar para a solução guia por meio da inserção gradativa de atributos da solução guia na solução base. As soluções intermediárias sofrem, então, uma busca local em que se fixam todos os atributos da solução guia que já foram incrementados a ela. O método é interrompido quando a solução base possuir todos os atributos da solução guia.

Geralmente a RC é utilizada como um método de pós-otimização ou então para intensificar a busca em torno de ótimos locais. No primeiro caso, armazena-se num conjunto as melhores soluções distintas encontradas por algum procedimento heurístico e posteriormente aplica-se a RC em pares de soluções deste conjunto, cujo nome é conhecido na literatura como Conjunto Elite (CE). No segundo caso, sempre que é encontrado um ótimo local em um determinado procedimento, aplica-se a RC no par de soluções formado pelo ótimo local e a melhor solução até então encontrada pelo algoritmo.

O CE possui tamanho fixo e para fazer parte dele a solução candidata deve obedecer a algum dos seguintes critérios:

- Ser melhor que a melhor solução do CE.
- Ser melhor que a pior solução do CE e ainda se diferenciar de todas as outras soluções em um certo percentual dos atributos, definido por *difElite*.

O segundo critério é adotado para que o CE não seja formado por soluções muito parecidas. Estando o CE completo, para que uma nova solução seja inserida, a de pior avaliação é removida.

O pseudocódigo dessa técnica é dado pelo Algoritmo 4. Mais informações sobre a mesma podem ser encontradas em Glover (1996), Glover e Laguna (2003) e Souza (2009).

Algoritmo 4: ReconexaoPorCaminhos(s, sol_{base}, sol_{guia}, $f()$, $N()$)

```
início
    Inicialize ListaAtributos com todos os atributos da solução guia;
    enquanto (sol_base ≠ sol_guia) faça
        melhorAtributo ← ∅ ;
        melhorValor ← ∞ ;
        para j = 1 até total de atributos de ListaAtributos faça
            Insira em sol_base o atributo j de sol_guia;
            se (f(sol_base) < melhorValor) então
                melhorAtributo ← j ;
                melhorValor ← f(sol_base) ;
            fim
            Desfaça a inserção do atributo j em sol_base;
        fim
        Insira em sol_base o atributo melhorAtributo de sol_guia;
        Retire de ListaAtributos o atributo melhorAtributo;
        s' ← Aplique uma busca local em sol_base preservando todos os
        atributos já inseridos;
        se (f(s') < f(s)) então
            s ← s' ;
        fim
    fim
    retorne s;
fim
```

3.2.5 Relaxação Adaptativa

A Relaxação Adaptativa consiste numa técnica que possibilita alcançar uma combinação entre intensificação e diversificação, de modo que orienta movimentos em relação a um nível crítico, o qual pode ser identificado por um estágio de construção ou um intervalo escolhido de valores para uma função. Este nível normalmente representa um ponto onde o método seria interrompido. Ao alcançar tal fronteira, as regras para selecionar os movimentos são modificadas, de modo que a região

definida pelo nível crítico seja atravessada e, desta maneira, o método não seja finalizado. Tal procedimento continua até uma determinada profundidade da fronteira de oscilação e volta novamente. O limite da oscilação é novamente alcançado e atravessado, porém, na direção oposta, e assim por diante. A Figura 3.1 exemplifica esse procedimento.

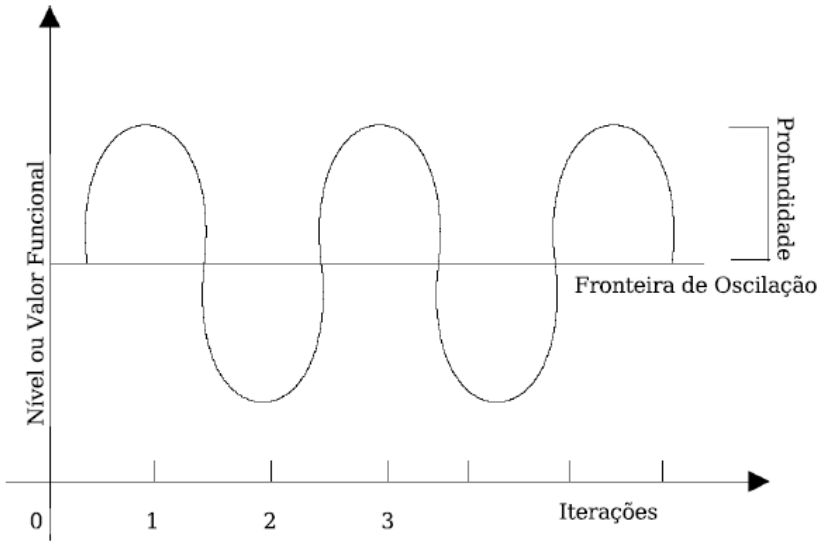

Figura 3.1: Oscilação Estratégica.

Essa técnica, também conhecida como Oscilação Estratégica, está diretamente ligada às origens da Busca Tabu (Glover e Laguna, 1997). Seu nome foi dado pelo fato de o processo abordar e atravessar o nível crítico repetidamente a partir de diferentes direções, criando um comportamento oscilatório.

Mecanismos tabu, tais como aqueles estabelecidos pelas funções da memória de curto prazo são utilizados para evitar a possibilidade de percorrer uma trajetória já visitada.

Um mecanismo de relaxação adaptativa bastante utilizado é o de Schaerf (1996), no qual os pesos, para cada fonte de inviabilidade da função de avaliação, são dinamicamente ajustados, conforme proposta de Gendreau et al. (1994). O peso ω_i, para cada fonte de inviabilidade i, é multiplicado por um fator α_i que varia de acordo com o seguinte esquema:

1. No início da busca $\alpha_i \leftarrow 1$;
2. A cada k movimentos:
   - se todas as k soluções visitadas são factíveis em relação à inviabilidade i, então $\alpha_i \leftarrow \alpha_i/\gamma$;
   - se todas as k soluções visitadas são infactíveis em relação à inviabilidade i, então $\alpha_i \leftarrow \alpha_i \times \gamma$;
   - se algumas soluções são factíveis e algumas outras são infactíveis, então α_i permanece inalterado.

O parâmetro γ pode ser definido randomicamente, a cada vez, no intervalo [1,8;2,2] como em Schaerf (1996), ou pode ser fixado no valor 2, como proposto em Gendreau et al. (1994).

Com o objetivo de não perder o referencial de avaliação das soluções, cada valor de α_i é limitado por duas constantes $\alpha_{i,\min}$ e $\alpha_{i,\max}$. Desta maneira, se α_i assumir um valor superior a $\alpha_{i,\max}$, é atrbuído a ele o valor de $\alpha_{i,\max}$. O inverso também é considerado, ou seja, se α_i assumir um valor inferior a $\alpha_{i,\min}$, é atrbuído a ele o valor de $\alpha_{i,\min}$. Este mecanismo de atualização do valor de α_i é importante, haja vista que após uma longa sequência de soluções factíveis ou infactíveis, ocorreria a perda do referencial de avaliação das soluções, em decorrência de valores muito altos ou muito baixos, respectivamente, para tal parâmetro.

3.2.6 *Ejection Chain*

Métodos baseados em *ejection chain* constituem uma combinação implícita entre intensificação e diversificação (Cavique et al., 1999). Nestes procedimentos, avalia-se a vizinhança de uma solução por transformações sucessivas de uma estrutura que, normalmente, não representa uma solução completa ou factível, mas é utilizada como referência para a avaliação de soluções teste em cada passo da *ejection chain*. A transformação de uma estrutura em outra consiste em substituir alguns atributos da estrutura corrente por novos atributos, forçando outros a deixarem a mesma.

A EC (Rego e Roucairol, 1996) foi proposta para tratar problemas de roteamento de veículos. Neste caso, inicialmente seleciona-se um subconjunto de m rotas $R = \{r_1, r_2, ..., r_m\}$ de forma arbitrária. Em seguida, transfere-se um cliente da rota r_1 para a rota r_2, um cliente de r_2 para r_3 e assim sucessivamente até que um cliente seja transferido da rota r_m para a primeira rota r_1. Nesse movimento os clientes são escolhidos de forma aleatória.

3.3 Otimização Multiobjetivo

Uma grande quantidade de problemas de decisão encontrados no mundo real é de natureza multiobjetivo, nos quais a tomada de decisão envolve dois ou mais objetivos e, normalmente, estes são conflitantes entre si. Em muitas aplicações são realizadas simplificações no problema original a fim de combinar todos os objetivos em uma única função, ou escolhendo apenas os objetivos classificados como prioritários ou importantes.

Esse tipo de problema possui uma característica particular, pois não apresenta uma solução única e sim um conjunto de soluções válidas, em que cada uma delas pode ser considerada uma solução para o problema em questão, ou seja, existe uma investigação que tem como objetivo encontrar um vetor de variáveis de decisão, que satisfaça as restrições e otimize um vetor em que os elementos representam as funções objetivo. Estas funções formam uma descrição matemática dos critérios de desempenho, que normalmente estão em conflito. As soluções encontradas são denominadas ótimas de Pareto e não existem outras soluções no espaço de busca melhores do que elas quando todos os objetivos são simultaneamente considerados.

Nos últimos anos vêm crescendo o número de pesquisas na área da otimização

multiobjetivo. A motivação para este aumento vem do cotidiano, pois a maioria dos problemas a serem resolvidos possuem mais de um objetivo. Um exemplo de aplicação multiobjetivo é a compra de um imóvel, no qual deseja-se comprar um imóvel com o menor custo, com a maior área útil e com localização privilegiada. Quais imóveis atenderiam a estes três objetivos? Observa-se que estes objetivos são conflitantes, já que para menor custo, menor será a área útil do imóvel e pior será a sua localização.

3.3.1 Formulação do Problema Multiobjetivo

Um problema de otimização de multiobjetivo consiste basicamente na obtenção de um conjunto de soluções que satisfaça um conjunto de restrições e minimize ou maximize uma função constituída por vários objetivos. Matematicamente, um problema de otimização multiobjetivo pode ser posto na seguinte forma:

$$\mathcal{X}^* = \arg\min_x \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \tag{3.1}$$

$$\text{sujeito a}: \; x \in \mathcal{F}_x \subset X \tag{3.2}$$

Nesta formulação, $x \in X$ é o vetor decisão, X é o espaço de parâmetros de otimização, $\mathcal{F}_x$ é o conjunto factível (soluções que satisfazem as restrições do problema), $[f_1(x)...f_m(x)]'$ é o vetor de funções objetivo do problema e $\mathcal{X}^*$ é o conjunto de pontos eficientes (soluções ótimas de Pareto).

Este conjunto, o qual constitui a solução do problema de otimização multiobjetivo, consiste de todos os vetores decisão em que o vetor objetivo correspondente não pode ser melhorado em nenhuma dimensão sem piorar em outra.

Dados dois vetores de decisão a e b, diz-se que a domina b se, e somente se:

$$\begin{array}{lllllll} \forall i & \in & \{1, \cdots, m\} & | & f_i(a) & \leq & f_i(b) \quad \wedge \\ \exists j & \in & \{1, \cdots, m\} & | & f_j(a) & < & f_j(b) \end{array} \tag{3.3}$$

Todos os vetores decisão que não são dominados por qualquer outro vetor decisão de um determinado conjunto, são chamados de não-dominados em relação a este conjunto. Consequentemente, uma solução ótima de Pareto é um vetor x que não é dominado por nenhum outro vetor do conjunto factível F_x, e o conjunto de todas as soluções ótimas de Pareto é o conjunto ótimo de Pareto X^*. A imagem do conjunto ótimo de Pareto no espaço da função objetivo Y^* é comumente referenciada como *Pareto-front*. Um exemplo de *Pareto-front* discreto e como a análise de dominância é feita nele, é mostrado na Figura 3.2. Este exemplo é extraído de Carrano et al. (2006).

No exemplo apresentado pela Figura 3.2, considere um problema de otimização com variáveis discretas que podem assumir um número infinito de valores. Cada uma das instâncias do problema de otimização biobjetivo tem seus dois objetivos, $f_1(.)$ e $f_2(.)$ avaliados. Cada avaliação corresponde a um ponto $f_1 \times f_2$ no plano, como mostrado na figura. Os pontos marcados como círculos (ponto I até IV) são as soluções eficientes, e os pontos marcados como quadrados são os pontos dominados. Observe, por exemplo, que o ponto I domina o ponto VI, com seus ambos objetivos

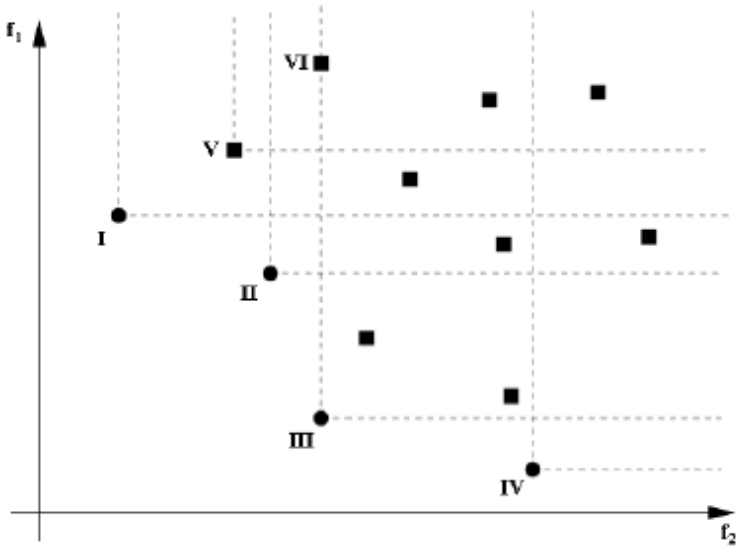

Figura 3.2: Exemplo da análise de dominância

f_1 e f_2 sendo menores do que no ponto VI. O ponto III também domina VI, tendo somente o objetivo f_2 menor, e o objetivo f_1 é igual ao do ponto VI. Note que embora o ponto V também domine o VI, este não é uma solução eficiente, afinal é dominado pelo ponto I. Observe também que mesmo o ponto IV sendo uma solução eficiente, ele não domina o ponto VI, embora seu objetivo f_2 seja menor, o objetivo f_1 é muito maior do que no ponto VI. As soluções de I até IV são eficientes porque elas não são dominadas por qualquer outra solução, enquanto as outras soluções são dominadas, uma vez que são piores ou iguais em todos os objetivos ou em pelo menos um, das soluções do conjunto pareto (I até IV).

3.4 NSGA-II

Com o intuito de abordar problemas de otimização multiobjetivo, Srinivas e Deb (1995) propuseram o algoritmo NSGA (*Non-Sorting Genetic Algorithm*) que possui duas finalidades principais: a) gerar as soluções não dominadas e b) manter diversidade destas soluções. A cada geração os n_{pop} indivíduos são agrupados em subpopulações que se distribuem em fronteiras não dominadas. Técnicas de nicho buscam desenvolver estas subpopulações espalhando soluções sobre cada fronteira não dominada.

Nesta época, os modelos de Algoritmos Evolucionários Multiobjetivos (MOEAs - *Multiobjective Evolutionary Algorithms*) que utilizavam mecanismos de classificação baseados em não dominância e partilha, estavam recebendo críticas principalmente por causa do custo computacional de ordenamento por não dominância, igual a $\theta(MN^3)$, sendo M o número de objetivos e N o tamanho da população. Além disso, utilizavam uma abordagem não-elitista, o que poderia levar à perda de boas soluções. Adicionalmente, possuíam a necessidade de se especificar um parâmetro de partilha, utilizado para manter a diversidade da população, sendo um parâmetro

de difícil determinação.

Neste contexto, o NSGA-II foi proposto em Deb et al. (2002) com o intuito de amenizar esses pontos negativos. Para resolver o custo relacionado à classificação da população em fronteiras de pontos eficientes (ordenamento por não dominância), o NSGA-II associa a cada solução candidata um contador para o número de soluções que a dominam, e outra contendo índices das soluções dominadas pela solução candidata. As soluções não dominadas pertencem à primeira fronteira e são separadas do grupo. O processo continua, e as fronteiras são separadas uma a uma. Em Deb et al. (2002) é demonstrado que a complexidade computacional deste método é dada por $\theta(MN^2)$. Este método de ordenação por não dominância é conhecido como *Fast Non Dominated Sort* e seu pseudocódigo, de acordo com Deb et al. (2002), é apresentado no Algoritmo 5.

Algoritmo 5: Procedimento *FastNonDominatedSorting*(S)

para $cada(s \in S)$ **faça**
  $S_s \leftarrow \emptyset$;
  $n_s \leftarrow 0$;
  para $cada(q \in S)$ **faça**
    se $(s \prec q)$ **então**
      $S_s \leftarrow S_s \bigcup \{q\}$; `// q é adicionada ao conjunto de soluções dominadas por s`
    fim
    se $(q \prec s)$ **então**
      $n_s \leftarrow n_s + 1$;
    fim
  fim
  se $(n_s = 0)$ **então**
    $s_{rank} \leftarrow 1$; `// insere` s `no` *front* `1`
    $F_1 \leftarrow F_1 \bigcup \{s\}$;
  fim
fim
enquanto $(F_i \neq \emptyset)$ **faça**
  $Q \leftarrow \emptyset$;
  para $cada(s \in F_i)$ **faça**
    para $cada(q \in S_s)$ **faça**
      $n_q \leftarrow n_q - 1$;
      se $(n_q = 0)$ **então**
        $q_{rank} \leftarrow i + 1$; `// q pertence ao próximo` *front*
        $Q \leftarrow Q \bigcup \{q\}$; `// insere em outro nível de não-dominância`
        $i \leftarrow i + 1$;
        $F_i \leftarrow Q$
      fim
    fim
  fim
fim

Para preservar a diversidade, o NSGA-II substituiu a função de partilha por uma abordagem de *crowded comparison*, sendo que esta nova aproximação não requer qualquer definição de parâmetro feita pelo usuário para manter a diversidade entre os membros da população. A função *Crowding Distance* (distância de agrupamento) é definida como sendo o perímetro do cubóide criado a partir de um ponto, tendo como vértices os vizinhos deste ponto. Só há sentido em calcular seu valor entre as soluções de uma mesma fronteira. As soluções extremas das fronteiras, que possuem apenas um vizinho, recebem um valor infinito e sempre terão preferência no método elitista.

Na Figura 3.3, apresentada em Deb et al. (2002), é exibida uma representação gráfica do que vem a ser a *crowding distance* de uma solução i em uma fronteira. O cálculo da função *crowding distance* necessita do ordenamento em ordem crescente dos objetivos. Em seguida, cada solução extrema da fronteira recebe um valor arbitrariamente grande. Todas as soluções intermediárias receberão um valor igual à diferença absoluta normalizada em função dos valores das duas soluções adjacentes pertencentes à mesma fronteira. O cálculo é repetido para cada objetivo. O valor global de cada solução corresponde à soma dos valores em relação a cada objetivo, calculado da forma supracitada. Antes de calcular a função *crowding distance*, cada objetivo é normalizado. O algoritmo para cálculo da *crowding distance* é apresentado pelo Algoritmo 6.

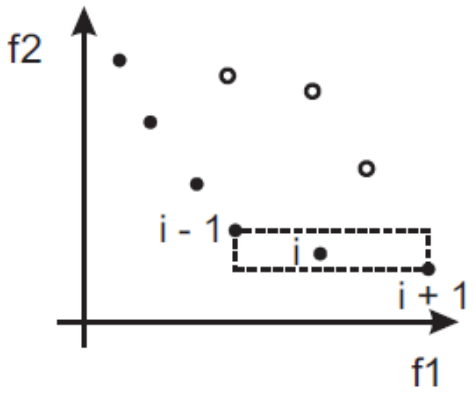

Figura 3.3: Cubóide gerado a partir de uma solução.

Algoritmo 6: Procedimento *CrowdingDistance*()

$l \leftarrow |I|$;
para $cada(i \in I)$ **faça**
  $I[i]_{cd} \leftarrow 0$;
fim
para $cada(objetivom)$ **faça**
  $I \leftarrow ordena(I, m)$; `// ordena I em relação ao objetivo m em ordem crescente`
  $I[1]_{cd} \leftarrow I[l]_{cd} \leftarrow \infty$; `// atribui infinito às soluções extremas de I`
  para $(i \leftarrow 2\ ate(i-1))$ **faça**
    $I[i]_{cd} \leftarrow I[i]_{cd} + ((I[i+1].m - I[i-1].m)/(f_m^{max} - f_m^{min}))$
  fim
fim

O elistimo é introduzido comparando as melhores soluções da nova geração com a anterior. Os vencedores são mantidos nas primeiras fronteiras.

O Algoritmo 7 descreve o esquema geral do NSGA-II

Algoritmo 7: Procedimento NSGAII(N,N_A)

$t \leftarrow 0$;
$P_t \leftarrow$ **nova_população**(N);
$Q_t \leftarrow \emptyset$;
$A \leftarrow$ **non_dominated**(P_t);
enquanto *não criterio_de_parada* **faça**
  $R_t \leftarrow P_t \cup Q_t$;
  $\mathcal{F} \leftarrow$ **fast_non_dominated_sorting**(R_t);
  $P_{t+1} \leftarrow \emptyset$;
  $i \leftarrow 1$;
  enquanto $|P_{t+1}| + |\mathcal{F}_i| \leq N$ **faça**
    $\mathcal{C}_i \leftarrow$ **crowding_distance**$(\mathcal{F}_i)$;
    $P_{t+1} \leftarrow P_t \cup \mathcal{F}_i$;
    $i \leftarrow i + 1$;
  fim
  $\mathcal{F}_i \leftarrow$ **sort**$(\mathcal{F}_i, \mathcal{C}_i,$ 'descending');
  $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$;
  $Q_{t+1} \leftarrow$ **selection**(P_{t+1}, N);
  $Q_{t+1} \leftarrow$ **crossover**(Q_{t+1});
  $Q_{t+1} \leftarrow$ **mutation**(Q_{t+1});
  $t \leftarrow t + 1$;
  $A \leftarrow$ **non_dominated**$(A \cup Q_t)$;
fim

Neste Algoritmo:

- $P_t \leftarrow$ **nova_população**(N) gera uma população aleatória com N indivíduos seguindo o esquema de codificação adotado para as soluções;
- $A \leftarrow$ **non_dominated**(P_t) Retorna os indivíduos que se encontram na pri-

meira fronteira da população P;

- $\mathcal{F} \leftarrow$ **fast_non_dominated_sorting**(R_t) aplica o procedimento *fast non-dominated sorting* para encontrar a fronteira de cada solução na população R_t
- $\mathcal{C}_i \leftarrow$ **crowding_distance_assignment**$(\mathcal{F}_i)$ aplica o procedimento *crowding distance assignment* para estimar como as soluções da fronteira i estão espalhadas no espaço objetivo. Soluções que estão em áreas menos populosas recebem um valor maior na avaliação de *crowding* e, por outro lado, soluções em áreas mais populosas recebem um valor menor;
- $\mathcal{F}_i \leftarrow$ **sort**$(\mathcal{F}_i, \mathcal{C}_i,$ 'descending') ordena as soluções da fronteira i em ordem decrescente de C_i;
- $Q_{t+1} \leftarrow$ **selection**(P_{t+1}, N) usa o torneio estocástico binário para a seleção a partir da população P_{t+1}.

Capítulo 4

Metodologia

O Problema Generalizado de Atribuição é comumente resolvido na literatura por métodos heurísticos devido à sua complexidade computacional, principalmente quando o enfoque da pesquisa está em encontrar soluções para instâncias de médio e grande porte. Neste trabalho, portanto, foram propostas abordagens heurísticas que são combinações de métodos para a resolução do PGA.

Para a formulação mono-objetivo do problema foram desenvolvidos cinco algoritmos heurísticos híbridos combinando os métodos *Simulated Annealing* (SA), Descida em Vizinhança Variável (*Variable Neighborhood Descent* - VND), Busca Tabu com Relaxação Adaptativa (BTRA), Reconexão por Caminhos (RC) e *Ejection chain* (EC).

A seção 4.1 apresenta como tais métodos foram aplicados ao PGA e como eles foram combinados nos algoritmos implementados.

Para a formulação multiobjetivo do problema, este trabalho propôs uma versão discreta melhorada do algoritmo NSGA-II, na qual foram propostos novos operadores de cruzamento e mutação. A seção 4.2 descreve a proposta.

4.1 Abordagens Mono-objetivo

Os métodos heurísticos utilizados neste trabalho foram combinados em cinco algoritmos que buscam soluções para o PGA. Antes de apresentar como tais métodos foram aplicados, a seção 4.1.1 descreve a representação da solução do problema; a seção 4.1.2 estabelece as estruturas de vizinhança e a seção 4.1.3 apresenta a função de avaliação que foi utilizada pelos algoritmos. Para solução inicial foram desenvolvidos três métodos: Aleatório, Construção Parcialmente Gulosa e Construção Híbrida. Tais métodos são descritos na seção 4.1.4. O método *Simulated Annealing* aplicado ao PGA é apresentado na seção 4.1.5, bem como as aplicações dos métodos VND, Busca Tabu, Reconexão por Caminhos e *Ejection Chain* são apresentadas nas seções 4.1.6, 4.1.8, 4.1.7 e 4.1.9, respectivamente.

Por fim, os algoritmos que combinam tais métodos são apresentados na seção 4.1.10.

4.1.1 Representação da solução

Cada solução foi codificada como um vetor de inteiros $1 \times |\mathcal{T}|$, sendo $|\mathcal{T}|$ o número de tarefas. Cada posição desse vetor pode assumir valores inteiros no intervalo $\{1, |\mathcal{A}|\}$, em que $|\mathcal{A}|$ representa o número de agentes. Um exemplo de codificação para uma instância PGA $A4 \times 10$ (4 agentes $\times$ 10 tarefas) é apresentado na Figura 4.1.

3	4	4	2	1	4	3	2	2	4

Figura 4.1: Exemplo de representação

Na solução candidata representada pela Figura 4.1, a tarefa 5 é atribuída ao agente $A1$, as tarefas 4, 8 e 9 são atribuídas ao agente $A2$, as tarefas 1 e 7 são atribuídas ao agente $A3$ e as tarefas 2, 3, 6 e 10 são atribuídas ao agente $A4$.

Observe que, pela representação adotada, a segunda restrição do conjunto de restrições 2.2 é automaticamente satisfeita, já que cada célula do vetor solução s recebe um único agente.

4.1.2 Estruturas de vizinhança

Foram utilizados dois tipos de movimentos para definir a vizinhança $N(s)$ de uma dada solução s. A Figura 4.2 apresenta uma ilustração dessas estruturas.

O primeiro movimento consiste em substituir um agente alocado a uma tarefa por outro agente. O conjunto de todas as soluções s' geradas a partir de s por meio de movimentos de substituição define a vizinhança $N^{(S)}(s)$.

O segundo movimento é caracterizado pela troca de agentes entre duas tarefas. O conjunto de todas as soluções s' geradas a partir de s com base em movimentos de troca define a vizinhança $N^{(T)}(s)$.

3	4	4	2	1	4	3	2	2	4	Solução s
3	4	4	2	1	4	3	2	2	1	(a) Vizinhança $N^{(S)}(s)$
4	4	4	2	1	4	3	2	2	3	(b) Vizinhança $N^{(T)}(s)$

Figura 4.2: Estruturas de vizinhança

No exemplo representado pela Figura 4.2 com uma instância PGA A4 $\times$ 10 (4 agentes $\times$ 10 tarefas), o movimento (a) resulta na solução s' tendo o agente $A4$ substituído pelo agente $A1$ na tarefa 10. O movimento (b) resulta na solução s' com as trocas dos agentes $A3$ e $A4$ anteriormente alocados às tarefas 1 e 10, respectivamente.

A vizinhança $N(s)$ de uma dada solução s é definida pela união dessas duas vizinhanças, isto é, $N(s) = N^{(S)}(s) \cup N^{(T)}(s)$.

4.1.3 Função de avaliação

Uma solução s é avaliada com base na função $f_C^m(s)$, dada pela Eq. (4.1). Essa função, que deve ser minimizada, considera os custos das atribuições dos agentes às

tarefas bem como penaliza o uso de recursos indisponíveis de cada agente.

$$f_C^m(s) = f_C(s) + r_{inv} \tag{4.1}$$

Na Eq. 4.1, $r_{inv} = \sum_{i \in \mathcal{A}} \rho_i \times \max \left\{ 0, \sum_{j \in \mathcal{T}} (a_{ij} \cdot x_{ij}) - b_i \right\}$ e $f_C(s)$ é calculada conforme a Eq. (2.1) e

A segunda parcela dessa função de avaliação penaliza as soluções em que a capacidade de cada agente i foi ultrapassada. Essa quantidade de violação referente ao primeiro conjunto de restrições (2.2) é multiplicada por um fator ρ_i, dado por $\rho_i = \sum_{j \in \mathcal{T}} a_{ij}$.

4.1.4 Geração da solução inicial

Para solução inicial foram desenvolvidos três métodos. As seções 4.1.4.1, 4.1.4.2 e 4.1.4.3 descrevem os mesmos.

4.1.4.1 Solução inicial Aleatória

Neste método, os agentes são atribuídos de maneira aleatória à cada tarefa. A restrição de indivisibilidade das tarefas é mantida devido à estrutura de representação da solução, porém, a restrição de capacidade dos agentes não é considerada nesse método. Desta maneira, é possível que o mesmo gere soluções iniciais infactíveis.

O pseudocódigo do procedimento aleatório é apresentado no Algoritmo 8.

Algoritmo 8: Procedimento gera_solucao_inicial_aleatoria()

$s \leftarrow \emptyset$;
para $j = 1, 2, \cdots, |\mathcal{T}|$ **faça**
  $s[j] \leftarrow$ agente $i \in \mathcal{A}$ escolhido aleatoriamente;
fim
Retorna s;

4.1.4.2 Solução inicial Parcialmente Gulosa

O método apresentado na seção 4.1.4.1, por ser totalmente aleatório, não considera os custos de atribuição das tarefas; logo, é possível que gere soluções iniciais com custos altos. Desta maneira, foi proposto um método que considera tais custos na geração das soluções iniciais.

A ideia consiste em testar todos os agentes para cada tarefa e selecionar os dois melhores em função do custo de atribuição. De posse desses dois agentes candidatos, o método seleciona aleatoriamente o agente que será atribuído à tarefa.

O pseudocódigo do procedimento é dado pelo Algoritmo 9

Algoritmo 9: Procedimento gera_solucao_inicial_parcialmente_gulosa()

$s \leftarrow \emptyset$;
para $j = 1, 2, \cdots, |\mathcal{T}|$ **faça**
  $[agenteCandidato_1, agenteCandidato_2] \leftarrow Agentes\_Menores\_Custos(j)$;
  $s[j] \leftarrow agenteCandidato_1$ ou $agenteCandidato_2$ escolhido aleatoriamente;
fim
Retorna s;

4.1.4.3 Solução inicial Híbrida

Neste método a solução inicial é obtida pela aplicação dos procedimentos *Simulated Annealing* e Descida em Vizinhança Variável (VND) a uma solução construída aleatoriamente. O pseudocódigo do procedimento heurístico híbrido é apresentado no Algoritmo 10.

Algoritmo 10: Procedimento gera_solucao_inicial_hibrida()

$s \leftarrow \emptyset$;
para $j = 1, 2, \cdots, |\mathcal{T}|$ **faça**
  $s[j] \leftarrow$ agente $i \in \mathcal{A}$ escolhido aleatoriamente;
fim
$T_0 \leftarrow TemperaturaInicial()$;
$s \leftarrow SA(f_C^m(s), s, T_0)$;
Retorna s;

O procedimento VND é acionado internamente pelo SA sempre que esse encontra uma solução de melhora. Esse caráter híbrido combinando os procedimentos SA e VND será apresentado com maiores detahes na seção 4.1.5.

4.1.5 *Simulated Annealing* aplicado ao PGA

Neste trabalho, o algoritmo SA básico foi combinado com o algoritmo VND, onde esse segundo é acionado internamente pelo SA para realizar uma busca local nas soluções que melhoram a solução global. Portanto, conforme pode ser visto no Algoritmo 11, são mantidas todas as características do método SA e inserida a chamada ao método VND.

Conforme observado mais adiante, na seção 4.1.10, o SA foi utilizado especificamente na geração de soluções iniciais, de forma a facilitar o método de refinamento na busca por soluções competitivas.

Os vizinhos aleatórios foram gerados com o movimento de substituição $N^{(S)}(s)$ apresentado na seção 4.1.2.

A temperatura inicial T_0 foi obtida de forma auto-adaptativa por meio de simulação, utilizando o mesmo princípio do método *Simulated Annealing*. O procedimento é iniciado partindo-se de uma solução s e de um valor baixo para a temperatura. Em seguida um número de vizinhos de s é gerado e verifica-se quantos deles são aceitos em SAmax iterações dessa temperatura. Se o número de vizinhos aceitos for alto, como por exemplo 90% (dado como parâmetro), a temperatura

Algoritmo 11: Procedimento $SA(f(.), \alpha, SAmax, T_0, s)$

```
s* ← s ;                                   // Melhor solução obtida até então
IterT ← 0 ;                                // Número de iterações na temperatura T
T ← T_0 ;                                  // Temperatura corrente
enquanto T > 0 faça
    enquanto IterT < SAmax faça
        IterT ← IterT + 1;
        Gere um novo vizinho qualquer s' ∈ N(s) ;
        Δ = f(s') − f(s);
        se Δ < 0 então
            s ← s';
            se f(s') < f(s*) então
                s* ← s';
                VND(s');                   // Busca Local na solução de melhora
                se f(s') < f(s*) então
                    s* ← s';
                fim
            fim
        senão
            Tome x ∈ [0, 1];
            se x < e^{−Δ/T} então
                s ← s';
            fim
        fim
    fim
    T ← α × T;
    IterT ← 0;
fim
s ← s*;
Retorne s;
```

corrente é retornada como temperatura inicial. Caso contrário, deve-se aumentar a temperatura segundo um certa taxa, por exemplo 10% e repetir o processo. O método termina quando o número de vizinhos cobertos pela temperatura atingir o limite definido. O pseudocódigo é dado pelo Algoritmo 12. Mais detalhes sobre tal procedimento podem ser vistos em (Souza, 2009).

4.1.6 Descida em Vizinhança Variável (VND) aplicada ao PGA

Neste trabalho foi mantida a forma clássica do método VND, definindo-se a seguinte ordem de vizinhanças: 1) $N^{(S)}(s)$ e 2) $N^{(T)}(s)$. Esta ordem foi utilizada tendo em vista que a primeira vizinhança é menos complexa que a segunda, e está coerente com a abstração de que o método VND explora vizinhanças gradativamente mais "distantes". Para cada vizinhança, aplica-se o Método de Descida (MD), em que somente movimentos de melhora são aceitos. Sempre que há melhora da solução

Algoritmo 12: Procedimento $TemperaturaInicial(f(.), N(.), \beta, \gamma, SAmax, T_0, s)$

```
T ← T_0 ;                                          // Temperatura corrente
Continua ← TRUE;
enquanto Continua faça
    Aceitos ← 0 ;      // Número de vizinhos aceitos na Temperatura T
    para (IterT = 1, 2, ..., SAmax) faça
        Gere um novo vizinho qualquer s' ∈ N(s) ;
        Δ = f(s') − f(s);
        se Δ < 0 então
            Aceitos ← Aceitos + 1;
        senão
            Tome x ∈ [0, 1];
            se x < e^{−Δ/T} então
                Aceitos ← Aceitos + 1;
            fim
        fim
    fim
    se Aceitos ≥ γ × SAmax então
        Continua ← FALSE;
    senão
        T ← β × T;
    fim
fim
Retorne T;
```

corrente, retorna-se à primeira vizinhança. O método termina quando é aplicado o MD nas duas estruturas de vizinhança e nenhuma melhora é encontrada na solução corrente. O pseudocódigo é dado no Algoritmo 13.

4.1.7 Reconexão por Caminhos aplicado ao PGA

O método Reconexão por Caminhos foi aplicado na forma bidirecional, isto é, dadas duas soluções s_1 e s_2, caminha-se tanto de s_1 para s_2, quanto de s_2 para s_1.

A Reconexão por Caminhos aplicada ao PGA não conta com a busca local durante a inserção dos atributos. Tal busca é acionada apenas no final quando a melhor solução é retornada das duas trajetórias. O método escolhido para a busca local foi o VND.

4.1.8 Busca Tabu aplicada ao PGA

Para a Busca Tabu foi implementada uma matriz tabu $n \times m$ que armazena os movimentos proibidos, sendo n o número de agentes e m a quantidade de tarefas. O tempo de duração tabu foi sorteado dentro de uma faixa $[T_{\min}, T_{\max}]$, em que $T_{\min} = 0,2 \times n - 7$ e $T_{\max} = 0,2 \times n + 7$.

Durante a execução do método, o Conjunto Elite (CE) de soluções de boa qualidade pode ser formado. Tal conjunto possui tamanho fixo e para fazer parte dele

Algoritmo 13: Descida em Vizinhança Variável.

procedimento $VND(f(.), N(.), r, s)$
$r \leftarrow 2$; `/* o número de estruturas diferentes de vizinhança */`
$k \leftarrow 1$; `/* Tipo de estrutura de vizinhança corrente */`
enquanto $(k \leq r)$ faça
  caso *k:*
    1: Método_Descida(s,$N^{(S)}(s)$);
    2: Método_Descida(s,$N^{(T)}(s)$)
  fim
  se $(f(s') < f(s))$ então
    $s \leftarrow s'$;
    $k \leftarrow 1$
  senão
    $k \leftarrow k + 1$
  fim-se
fim-enquanto;
Retorne s;
fim VND;

a solução candidata deve obedecer a um dos dois critérios já apresentados na seção 3.2.4. Devido ao fato de a Busca Tabu ter sido aplicada de formas diferentes pelos algoritmos descritos na seção 4.1.10, em alguns casos o CE não é aplicado. Quando utilizado, o conjunto viabiliza a estratégia de intensificação feita pela Reconexão por Caminhos.

Como critério de aspiração, foi adotada a aspiração por objetivo global. Dessa forma, uma solução tabu s é aceita caso seu valor seja menor que a melhor solução global.

O pseudocódigo da Busca Tabu aplicada ao PGA é apresentado no Algoritmo 14. Os parâmetros do método são o número de iterações sem melhora ($BTmax$), o tamanho do conjunto elite ($|CE|$), o número de iterações sem atualização do CE ($nMaxIterSemElite$) e os limites inferior e superior da duração tabu, $T_{\min}$ e $T_{\max}$,

respectivamente.

Algoritmo 14: BuscaTabu(s, $f_C^m()$, $N()$, CE, $BTmax$,$nMaxIterSemElite$,$T_{\min}, T_{\max}$)

início
- $s^* \leftarrow s$; `// Melhor solução`
- $T \leftarrow \emptyset$; `// Lista Tabu`
- $IterCorrente \leftarrow 0$; `// Iteração corrente`
- $MelhorIter \leftarrow 0$; `// Iteração da melhor solução`
- $nIterSemElite \leftarrow 0$; `// Iterações sem atualização do CE`
- **enquanto** ($IterCorrente - MelhorIter \leq BTmax$) **faça**
  - Seja $s' \leftarrow s \oplus m$ o melhor elemento de CN, tal que $CN \in \{N^{(S)}(s), N^{(T)}(s)\}$ e tal que o movimento m não seja tabu ($m \notin T$) ou, se tabu, s' atenda a condição de aspiração, isto é $f_C^m(s') < f_C^m(s^\star)$;
  - Atualize a matriz de duração tabu T;
  - $s \leftarrow s'$;
  - **se** *s deve pertencer ao conjunto de soluções elite* **então**
    - Inserir s no ConjuntoElite;
    - $nIterSemElite \leftarrow 0$;
  - **senão**
    - $nIterSemElite \leftarrow nIterSemElite + 1$;
    - **se** *$f_C^m(s')$ é a melhor solução encontrada enquanto $nIterSemElite < nMaxIterSemElite$* **então**
      - Atualize a melhor solução encontrada nesse intervalo;
    - **fim**
    - **se** $nIterSemElite = nMaxIterSemElite$ **então**
      - Selecione aleatoriamente uma solução do Conjunto Elite;
      - Aplique o método de Reconexão por Caminhos entre essa solução e a melhor encontrada enquanto $nIterSemElite < nMaxIterSemElite$;
      - Atualize s após a saída do método Reconexão por Caminhos;
      - $nIterSemElite \leftarrow 0$;
    - **fim**
  - **fim**
  - **se** ($f_C^m(s) < f_C^m(s^\star)$ **então**
    - $s^\star \leftarrow s$;
    - $MelhorIter \leftarrow IterCorrente$;
  - **fim**
  - $IterCorrente \leftarrow IterCorrente + 1$;
- **fim**
- **retorne** s^*;

fim

4.1.8.1 Relaxação Adaptativa

Em alguns casos, descritos na seção 4.1.10, é aplicada a Relaxação Adaptativa. Isso ocorre após certo número de iterações sem melhora na solução global durante a

fase de Busca Tabu (no caso, dez iterações, valor esse determinado por experimentos empíricos). Esse mecanismo atualiza o peso dos custos dinâmicos com o objetivo de redirecionar a busca para outras regiões mais promissoras do espaço de soluções, conforme descrito na seção 3.2.6. O mecanismo adotado foi o mesmo de (Schaerf, 1996), em que os pesos atribuídos para cada fonte de inviabilidade são ajustados dinamicamente, como proposto em (Gendreau et al., 1994).

O pseudocódigo da Busca Tabu com Relaxação Adaptativa aplicada ao PGA é visto no Algoritmo 15.

4.1.9 *Ejection Chain* aplicado ao PGA

A aplicação da técnica EC ao PGA parte da seleção de um subconjunto de k tarefas $T = \{t_1, t_2, ..., t_m\}$ de tamanho aleatório. Feito isso, transfere-se o agente da tarefa t_1 para a tarefa t_2, o agente de t_2 para t_3 e assim sucessivamente até que o agente seja transferido da tarefa t_m para a primeira tarefa t_1.

Para a EC não foi implementada uma memória tabu específica, de modo que o movimento de substituição do agente de t_2 pelo agente de t_1 é inserido na matriz tabu mantida pela Busca Tabu.

4.1.10 Algoritmos propostos ao PGA

Nessa seção, os algoritmos propostos para estudo serão apresentados. Foram desenvolvidos cinco algoritmos combinando os métodos descritos na seções anteriores.

Vale ressaltar que em todos os algoritmos foram aplicadas a representação da solução, as estruturas de vizinhança, bem como a função de avaliação conforme apresentadas nas seções 4.1.1, 4.1.2 e 4.1.3, respectivamente. Portanto, as seções que descrevem os algoritmos se limitam apenas a apresentar a composição dos métodos utilizados em cada um deles.

4.1.10.1 BTRC

No algoritmo BTRC, a solução inicial é dada pela Construção Híbrida, apresentada na seção 4.1.4.3. Como visto, esse procedimento combina os métodos SA e VND. A temperatura inicial T_0 para o SA foi obtida de forma auto-adaptativa por meio de simulação, conforme descrito na seção 4.1.5. Os parâmetros utilizados para a simulação foram: taxa de aumento da temperatura $\beta = 10\%$, taxa mínima de aceitação de soluções vizinhas $\gamma = 5\%$ e temperatura de partida $T_0 = 10$.

Construída a solução inicial, ela é refinada pelo procedimento BT descrito na seção 4.1.8. Conforme dito, durante sua execução, a BT armazena um conjunto de soluções elite e aplica uma estratégia de intensificação com o procedimento RC, descrito na seção 4.1.7.

Por fim, o algoritmo BTRC utiliza o método RC também como estratégia de pós-otimização, fazendo reconexões nos caminhos entre todas as soluções presentes no Conjunto Elite.

Algoritmo 15: BuscaTabuRA(s, $f_C^m()$, $N()$, CE, $BTmax$,$nMaxIterSemElite$,$T_{\min}, T_{\max}$).

```
início
    s* ← s ;                                              // Melhor solução
    T ← ∅;                                                    // Lista Tabu
    IterCorrente ← 0 ;                                  // Iteração corrente
    MelhorIter ← 0 ;                          // Iteração da melhor solução
    enquanto (IterCorrente − MelhorIter ≤ BTmax) faça
        Seja V ⊆ CN, tal que CN ∈ {N^(S)(s), N^(T)(s)};
        melhorCustoDinamico ← ∞;
        para todo Movimento m ∈ V faça
            se (f_C^m(s ⊕ m) < f_C^m(s*)) então
                melhorMovimento ← m;
                interromper;
            senão
                se (m ∉ T) então
                    se (f_C^m(s ⊕ m) < f_C^m(s)) então
                        melhorMovimento ← m;
                        interromper;
                    senão
                        se (CustoPorPesosDinamicos(s ⊕ m) <
                        melhorCustoDinamico) então
                            melhorMovimento ← m;
                            melhorCustoDinamico ←
                            CustoPorPesosDinamicos(s ⊕ m);
                        fim
                    fim
                fim
            fim
        fim
        s ← s ⊕ melhorMovimento;
        se (f_C^m(s) < f_C^m(s*) então
            s* ← s;
            MelhorIter ← IterCorrente;
        fim
        Atualizar a matriz de duração tabu T;
        se iteracaoAtualizacao() então
            AtualizarPesosDinamicos();
        fim
        IterCorrente ← IterCorrente + 1;
    fim
    retorne s*;
fim
```

4.1.10.2 BTRA

O algoritmo BTRA se difere do BTRC no que diz respeito ao mecanismo de intensificação e uso de mecanismos de pós-otimização.

A solução incial é dada pela Construção Híbrida, seguindo os mesmos parâmetros do algoritmo BTRC.

O refinamento da solução inicial é feito pelo método de Busca Tabu com Relaxação Adaptativa, como descrito na seção 4.1.8.1. Com isso o BTRA tenta alcançar uma combinação entre intensificação e diversificação.

Sobre mecanismos de pós-otimização, o BTRA é insento de implementações e, portanto, não possui essa etapa.

4.1.10.3 randBTRA

Com o objetivo de avaliar o impacto da solução inicial no resultado final, o algoritmo randBTRA tem como proposta alterar o BTRA aplicando uma solução inicial Aleatória, conforme a descrita na seção 4.1.4.1.

Construída a solução inicial, o randBTRA utiliza o método Busca Tabu com Relaxação Adaptativa no refinamento dessa solução.

A etapa de pós-otimização também é excluída desse algoritmo.

4.1.10.4 greedyBTRA

O algoritmo greedyBTRA tem o mesmo objetivo do randBTRA, isto é, avaliar o impacto da solução inicial no resultado final. No entanto, a diferença entre eles se resume no procedimento escolhido para geração da solução inicial. O greedyBTRA faz uso da construção Parcialmente Gulosa, descrita na seção 4.1.4.2.

A fase de refinamento é dada pelo método de Busca Tabu com Relaxação Adaptativa, tal como os algoritmos BTRA e randBTRA.

A etapa de pós-otimização também é excluída desse algoritmo.

4.1.10.5 BTRA-EC

No algoritmo BTRA-EC, a solução inicial é gerada a partir da Construção Híbrida. Os parâmetros utilizados são os mesmos aplicados nos algoritmos BTRC e BTRA.

A Busca Tabu com Relaxação Adaptativa também é utilizada na fase de refinamento da solução inicial, como nos algoritmos BTRA, randBTRA e greedyBTRA.

O BTRA-EC se diferencia ao aplicar mecanismos de perturbação e busca local na fase de refinamento da solução inicial. O algoritmo adota os procedimentos *Ejection Chain* e VND, periodicamente. A perturbação é feita pelo EC na solução corrente da Busca Tabu a cada $BTmax/3$ iterações sem melhora (intervalo definido a partir de experimentações). Após o Ejection Chain, a solução perturbada é submetida ao método VND para que seja feita uma busca local.

O BTRA-EC altera a trajetória da Busca Tabu toda vez que aplica tal procedimento, haja vista que o resultado dessa perturbação e busca local se torna a solução corrente do método de refinamento.

4.2 Abordagens Multiobjetivo

O problema multiobjetivo abordado possui duas caracteristicas principais:

- É um problema combinatório, com funções objetivo e restrições discretas;
- Uma das funções objetivo, a função de equilíbrio, é não-linear.

Tais aspectos impedem o uso da maior parte dos métodos determinísticos de otimização, sugerindo o emprego de outros métodos como o Algoritmo Genético (AG).

Os algoritmos genéticos são particularmente adequados para o problema em questão, de modo que sua formulação é facilmente adaptável para lidar com problemas multiobjetivo ((Coello, 2000; Fonseca e Fleming, 1995)).

O algoritmo proposto neste trabalho para abordar o Poblema Generalizado de Atribuição Multiobjetivo (moPGA) é baseado no NSGA-II, proposto em (Deb et al., 2002) e revisado na seção 3.4. Para tanto, foram aplicadas melhorias nos operadores de cruzamento e mutação. A proposta é apresentada nas seções seguintes.

4.2.1 Seleção e Elitismo

Neste trabalho foram utilizados os mesmos procedimentos de seleção e elitismo do algoritmo original NSGA-II.

O elitismo é dado usando *fast non-dominated sorting*, o qual divide a solução em fronteiras baseadas na análise de dominância.

A densidade das soluções em torno de cada ponto (*crowding*) foi estimada usando *crowding distance assignment*.

Os valores de densidade e fronteiras atribuídos a cada solução foram usados na seleção do algoritmo. As soluções são selecionadas usando um torneio estocástico binário. Inicialmente as soluções são comparadas com relação aos valores de fronteira, e a solução com melhor valor ganhará. Se as fronteiras forem iguais, então as soluções são comparadas com relação à densidade, sendo que aquela que possuir o maior valor de densidade é selecionada.

As descrições detalhadas sobre o torneio estocástico binário, *fast non-dominated sorting* e *crowding distance assignment* podem ser encontradas na referência original (Deb et al., 2002).

4.2.2 Representação da solução

A solução é representada da mesma maneira como proposto na abordagem mono-objetivo, por meio de um vetor de inteiros. Detalhes podem ser vistos na seção 4.1.1.

4.2.3 Operador de cruzamento

A proposta é utilizar o Cruzamento Uniforme como operador de cruzamento. Um exemplo desse tipo de operador aplicado ao PGA é ilustrado pela Figura 4.3.

Na Figura 4.3, p_1 e p_2 são as soluções pais, o_1 e o_2 são as soluções filhos e $ref.$ é a referência. Inicialmente, p_1 é copiado para o_1 e p_2 para o_2. Feito isso, os valores de

p_1	3	4	4	2	1	4	3	2	2	4
p_2	1	3	4	1	1	2	2	3	2	3
ref.	0	1	0	0	1	1	1	0	0	1
o_1	3	3	4	2	1	2	2	2	2	3
o_2	1	4	4	1	1	4	3	3	2	4

Figura 4.3: Exemplo do Cruzamento Uniforme.

o_1 e o_2 são trocados nas posições indicadas com o valor 1 na referência $ref.$ (colunas cinzas).

4.2.4 Operador de mutação

O tradicional *Swap Mutation* é utilizado como base para o operador de mutação empregado neste trabalho, o qual foi denominado GAPMutation. No entanto, alguns conhecimentos em torno do problema foram inseridos no operador, com o objetivo de melhorar sua eficiência em relação ao problema abordado.

Dada a solução s que sofrerá a mutação, o operador executa os seguintes passos:

1. Escolher, aleatoriamente, uma posição i do vetor s (agente s_i que iria realizar a tarefa i);
2. Gerar um valor aleatório r_D seguindo distribuição uniforme
3. Se $r_D \leq 0.2$, então:
   - Substituir s_i por um novo agente aleatório;
4. Se $r_D > 0.2 e r_D > 0.2 \leq 0.5$, então:
   - Substituir s_i pelo agente menos requisitado;
5. Se $r_D > 0.5$, então:
   - Substituir s_i pelo agente que despende a menor quantidade de recursos para executar a tarefa i;

Os valores para os intervalos foram definidos empiricamente com base em experimentos realizados para calibração do algoritmo.

4.2.5 Funções de Avaliação da Aptidão

As funções de avaliação foram baseadas nas funções de custo (2.1) e equilíbrio (2.3) apresentadas nas seções 2.1 e 2.2, respectivamente.

Para que a primeira restrição apresentada pela Equação 2.5 seja considerada, a função de custo foi adaptada com uma multiplicação por um fator exponencial, cujo objetivo é penalizar soluções infactíveis nesta restrição que trata da limitação de recursos disponíveis por agente. O objetivo remodelado é apresentado na Equação 4.2

$$f_C^m(x) = f_C(x) \cdot e^{r_{ind}} \tag{4.2}$$

onde $r_{ind} = \sum_{i \in \mathcal{A}} \max \left\{ 0, \sum_{j \in \mathcal{T}} (a_{ij} \cdot x_{ij}) - b_i \right\}$.

Este tipo de função de penalidade garante que as soluções que satisfaçam todas as restrições não sejam penalizadas. Por outro lado, os valores das funções de custo das soluções que não comprirem com a primeira restrição aumentará rapidamente com a requisição de recursos indisponíveis.

Capítulo 5

Resultados

Neste capítulo são apresentados e analisados os resultados obtidos com os algoritmos mono e multiobjetivo. São apresentados, também, os principais parâmetros dos mesmos, bem como a comparação dos resultados obtidos com outros da literatura.

Para testar os algoritmos desenvolvidos foram consideradas 24 instâncias-teste, cujas características principais estão descritas na seção 5.1.

A seção 5.2 apresenta os resultados e análises para o problema clássico mono-objetivo. Para o problema multiobjetivo os resultados são apresentados na seção 5.3.

5.1 Instâncias-teste

Nos experimentos computacionais realizados foram consideradas instâncias do problema disponíveis na OR-Library (Beasley). Tais instâncias estão divididas em cinco tipos: A, B, C, D e E, sendo que os tipos A e B são considerados fáceis e os tipos C, D e E de complexidade média (Yagiura et al., 2006).

Cada tipo contém 6 instâncias, geradas a partir da combinação de valores de m e n, sendo $m \in \{5, 10, 20\}$ e $n \in \{100, 200\}$. A quantidade a_{ij} de recursos do agente i demandada para realizar a tarefa j, o custo c_{ij} de atribuição do agente i à tarefa j e a quantidade bi de recursos disponíveis ao agente i foram valorados conforme descrito a seguir (Chu e Beasley, 1997):

- Tipo A: a_{ij} são valores inteiros aleatórios no intervalo $[5, 25]$, c_{ij} são valores inteiros aleatórios no intervalo $[10, 50]$ e
$$b_i = 0.6 \left\{ 0.6(n/m)15 + 0.4 \max_{i \in I} \sum_{j \in J, i_j^{min} = i} a_{ij} \right\};$$
- Tipo B: a_{ij} são valores inteiros aleatórios no intervalo $[5, 25]$, c_{ij} são valores inteiros aleatórios no intervalo $[10, 50]$ e
$$b_i = 0.7 \left\{ 0.6(n/m)15 + 0.4 \max_{i \in I} \sum_{j \in J, i_j^{min} = i} a_{ij} \right\};$$

- Tipo C: a_{ij} são valores inteiros aleatórios no intervalo $[5, 25]$, c_{ij} são valores inteiros aleatórios no intervalo $[10, 50]$ e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$;
- Tipo D: a_{ij} são valores inteiros aleatórios no intervalo $[1, 100]$, $c_{ij} = 111 - a_{ij} + e_1$, onde e_1 são valores aleatórios no intervalo $[-10, 10]$, e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$;
- Tipo E: $a_{ij} = 1 - 10lne_2$, onde e_2 são números aleatórios no intervalo $[0, 1]$, $c_{ij} = 1000/a_{ij} - 10e_3$, onde e_3 são números aleatórios no intervalo $[0, 1]$ e $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

5.2 Resultados Mono-objetivo

Nessa seção, os resultados apresentados pelos algoritmos de otimização mono-objetivo são confrontados e analisados.

Inicialmente, a seção 5.2.1 descreve os principais parâmetros adotados para os algoritmos.

Na seção 5.2.2 são comparados os desempenhos dos algoritmos BTRC e BTRA (apresentados nas seções 4.1.10.1 e 4.1.10.2, respectivamente), com outros métodos presentes na literatura. Também nessa seção, os métodos BTRC e BTRA são submetidos à uma análise de distribuição de probabilidade empírica de se alcançar um dado valor alvo (isto é, encontrar uma solução com um custo no mínimo tão bom quanto o valor alvo) em função do tempo.

Na seção 5.2.3 são feitos testes comparativos entre os algoritmos BTRC, BTRA, randBTRA, greedyBTRA e BTRA-EC (apresentados nas seções,4.1.10.1, 4.1.10.2, 4.1.10.3, 4.1.10.4 e 4.1.10.5, respectivamente). Foram feitas comparações considerando as melhores soluções encontradas por esses algoritmos, bem como a média das melhores soluções e, além disso, foram aplicadas análises estatísticas utilizando Teste-Z para verificar a significância da diferença entre as médias das melhores soluções apresentadas por esses algoritmos.

5.2.1 Determinação de parâmetros dos algoritmos

Os parâmetros adotados para cada método foram definidos empiricamente baseados em experimentos realizados.

Para a Busca Tabu foram definidos os seguintes parâmetros: número de iterações sem melhora - $BTmax = 2000$; número máximo de iterações sem atualização no Conjunto Elite - $nMaxIterSemElite = 0,20 \times BTmax$; tamanho do Conjunto Elite - $CE = 5$.

Para o *Simulated Annealing* seguem os parâmetros definidos: taxa de resfriamento - $\alpha = 0,998$; número de iterações feitas em cada temperatura - $SAmax = (k \times m \times n)$, sendo a constante $k = 3$. Os parâmetros para determinar a temperatura inicial foram os seguintes: taxa de aumento da temperatura - $\beta = 10\%$; taxa mínima de aceitação de soluções vizinhas - $\gamma = 5\%$; temperatura de partida - $T_0 = 10$.

Na aplicação da Relaxação Adaptativa foram definidos os seguintes parâmetros: número de iterações sem melhora na Busca Tabu que aciona a RA - $\mu = 10$; $\sigma_{min} = 0,0001$, $\sigma_{max} = 5$ e $\gamma \in [1,8; 2,2]$.

5.2.2 Comparações entre os algoritmos BTRC e BTRA

Nessas comparações, os métodos foram aplicados às instâncias dos tipo C, D e E, descritas na seção 5.1.

A Tabela 5.1 apresenta os resultados da comparação entre os algoritmos BTRA e BTRC, propostos neste trabalho, e os algoritmos PREC (Yagiura et al., 2006), CB (Chu e Beasley, 1997), Bvdsj (Yagiura et al., 1998), RA (Amini e Racer, 1995), LKGG (Laguna et al., 1995), TSBB (Woodcock e Wilson, 2010) e DF (Diaz e Fernandez, 2001). Nesta tabela, os melhores resultados, isto é, aqueles com os menores valores para a função de custo dada pela Eq. (4.1) estão destacados em negrito.

Tabela 5.1: Comparações dos algoritmos BTRC e BTRA com outros da literatura

Tipo	Agentes	Tarefas	BTRC	BTRA	TSBB	PREC	BVDS-j	RA	LKGG	CB	DF
C	5	100	*1937*	**1931**	**1931**	**1931**	**1931**	*1938*	**1931**	**1931**	**1931**
	10	100	*1403*	1403	**1402**	**1402**	*1403*	*1405*	*1403*	*1403*	**1402**
	20	100	*1250*	1248	**1243**	**1243**	1244	*1250*	1245	1244	**1243**
	5	200	*3460*	3459	**3456**	**3456**	3457	*3469*	3457	3458	3457
	10	200	*2820*	2818	**2806**	2807	2808	*2835*	2812	2814	2807
	20	200	2399	2400	**2391**	**2391**	*2400*	*2419*	2396	2397	**2391**
D	5	100	*6403*	6394	6356	**6353**	6362	ND	6386	6373	6357
	10	100	*6430*	6425	6366	6356	6370	*6532*	6406	6379	**6355**
	20	100	6317	6318	6254	**6211**	6245	*6428*	6297	6269	6220
	5	200	*12816*	12802	12745	**12744**	12755	ND	12788	12796	12747
	10	200	*12596*	12570	12456	**12438**	12473	*12799*	12537	*12601*	12457
	20	200	*12468*	12449	12332	**12269**	12318	*12665*	12436	*12452*	12351
E	5	100	*12748*	12696	**12681**	**12681**	12682	*12917*	12687	ND	12681
	10	100	*11772*	11622	11584	**11577**	11599	*12047*	*11641*	ND	11581
	20	100	*8646*	8569	8479	**8444**	8484	*9004*	8522	ND	8460
	5	200	*25181*	24975	24933	24933	24933	*25649*	*25147*	ND	**24931**
	10	200	*23609*	23366	**23307**	23310	23348	*24717*	*23567*	ND	23318
	20	200	*23050*	22602	22393	**22379**	22437	24117	*22659*	ND	22422

ND = não disponível

A Tabela 5.1 mostra que os algoritmos BTRA e BTRC encontraram resultados iguais e até melhores em relação a outras abordagens da literatura. Em destaque, os resultados dessas abordagens estão em itálico, facilitando a observação. Note que para todas as instâncias, os algoritmos desenvolvidos neste trabalho foram superiores ao RA, com destaque nas instâncias dos grupos D e E, as quais são mais complexas (Yagiura et al., 2006). Nessas instâncias, o algoritmos propostos apresentaram uma melhora significativa em relação ao algoritmo RA. Além disso, resultados iguais ou melhores foram encontrados quando confrontados com o BVDSj, LKGG e CB.

O algoritmo BTRA encontrou o melhor resultado conhecido da literatura para a instância C5x100 e encontrou valores próximos aos melhores conhecidos para as demais instâncias do tipo C.

Ao fazer a comparação entre os algoritmos desenvolvidos, nota-se que o BTRA foi superior ao BTRC, exceto para as instâncias C20x200 e D20x100, nas quais o BTRA não conseguiu alcançar resultados melhores.

Como experimento adicional, foi aplicado um teste de probabilidade empírica, seguindo a metodologia de Aiex et al. (2002), para verificar a capacidade dos algorit-

mos BTRC e BTRA de alcançarem um valor alvo. Para execução dos experimentos, utilizou-se as instâncias C5x100 e E20x200, tendo como alvo os valores de custo 1945 e 23386, respectivamente. Cada algoritmo foi executado 100 vezes para cada instância, sendo que em cada rodada ele era interrompido após alcançar o alvo. Não foram permitidos tempos de execução repetidos; assim, os tempos repetidos foram descartados e uma nova execução foi feita. Após determinados os tempos para as 100 execuções, estes foram ordenados de forma crescente e, para cada tempo t_i, foi associada uma probabilidade $\rho_i = (i - 0,05)/100$. Os resultados dos gráficos $t_i \times \rho_i$ são apresentados na Figura 5.1.

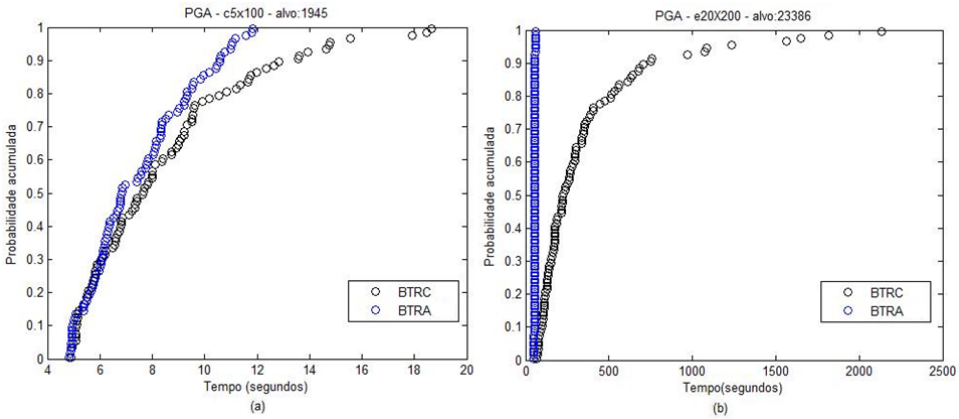

Figura 5.1: Teste de Probabilidade Empírica.

É possível observar na Figura 5.1 que o algoritmo BTRA demandou menos tempo que o BTRC para alcançar os valores alvo. Para a instância C20x200, ele necessita de aproximadamente 12 segundos para encontrar o respectivo valor com quase 100% de probabilidade, enquanto que o algoritmo BTRC necessita de cerca de 19 segundos. A vantagem do algoritmo BTRA é ainda maior para a instância E20x200, haja vista que o BTRA necessita de um tempo significativamente menor do que o algoritmo BTRC para encontrar o valor alvo, como pode ser observado na Figura 5.1.

Diante das comparações feitas nessa seção, é possível notar o bom desempenho do BTRA frente a algumas abordagens da literatura e, além disso, se destacou diante o BTRC. Outros testes e comparações serão apresentados na seção 5.2.3, os quais darão subsídios para afirmar se tal superioridade é verdadeira.

5.2.3 Comparações entre os algoritmos BTRC, BTRA, randBTRA, greedyBTRA e BTRA-EC

Nessa seção, os resultados dos algoritmos propostos neste trabalho são confrotados e analisados. As instâncias utilizadas foram do tipo A, B, C, D e E, descritas na seção 5.1. Nesses testes, foram realizadas 50 (cinquenta) execuções de cada algoritmo em cada instância. Foram propostas três avaliações, a saber: Comparação entre as melhores soluções encontradas pelos algoritmos (seção 5.2.3.1); Comparação entre as médias das melhores soluções encontradas pelos algoritmos (seção 5.2.3.2); Testes Estatísticos (seção 5.2.3.3).

5.2.3.1 Comparação entre as melhores soluções encontradas pelos algoritmos

Nessa avaliação serão apresentadas as melhores soluções encontradas por cada algoritmo em cada instância. Antes, porém, observou-se que o comportamento dos resultados se difere, em partes, entre as classes de problemas (pequeno e médio). Assim, para o melhor entendimento, decidiu-se apresentar os resultados separando os tipos de instâncias A e B dos tipos C, D, E.

A Tabela 5.2 apresenta as melhores soluções encontradas para as instâncias de menor complexidade (tipo A e B). Entende-se por melhores soluções aquelas com os menores valores para a função de custo descrita pela Eq. (4.1). Os resultados estão organizados por instância X algoritmo. As células em negrito destacam as soluções vencedoras na comparação por instância e, consequentemente, os algoritmos que encontraram tais soluções.

Tabela 5.2: Comparação entre as melhores soluções encontradas por cada algoritmo nas instâncias de menor complexidade

tipo	agentes	tarefas	BTRC	BTRA	randBTRA	greedyBTRA	BTRA-EC
A	5	100	**1698**	**1698**	**1698**	**1698**	**1698**
	10	100	**1360**	**1360**	**1360**	**1360**	**1360**
	20	100	**1158**	**1158**	1159	1159	**1158**
	5	200	**3235**	**3235**	**3235**	**3235**	**3235**
	10	200	**2623**	**2623**	2624	**2623**	**2623**
	20	200	**2339**	**2339**	2342	2342	**2339**
B	5	100	**1843**	**1843**	1846	**1843**	**1843**
	10	100	**1407**	**1407**	**1407**	**1407**	**1407**
	20	100	**1166**	**1166**	1170	1169	**1166**
	5	200	**3556**	3560	3559	3557	3562
	10	200	**2836**	2842	2843	2841	2840
	20	200	2341	2341	2352	2348	**2340**

Note que nessa comparação os algoritmos BTRC, BTRA e BTRA-EC apresentaram a maior quantidade de soluções vencedoras. Para as instâncias do tipo A, esses algoritmos retornaram as soluções ótimas conhecidas (encontradas em Chu e Beasley (1997)). Como visto, os algoritmos randBTRA e greedyBTRA não apresentaram bons resultados, mostrando que a mudança na estratégia de solução inicial nesses dois algoritmos impactou no resultado final para as instâncias do tipo A e B.

A comparação para as instâncias de complexidade média (tipo C, D e E) é apresentada na Tabela 5.3, cuja organização segue a mesma dada na Tabela 5.2 já apresentada.

Observe que para as instâncias de complexidade média os algoritmos BTRA, randBTRA e greedyBTRA apresentaram as maiores quantidades de soluções vencedoras. Os algoritmos BTRA-EC e BTRC foram capazes de apresentar tais soluções em poucos casos. Diferentemente do que foi visto nas instâncias de menor complexidade, os algoritmos randBTRA e greedyBTRA se equipararam ao BTRA. Logo, ao que tudo indica, para as instâncias de complexidade média, a solução inicial não impacta significativamente a qualidade da solução final, considerando os métodos implementados.

Tabela 5.3: Comparação entre as melhores soluções encontradas por cada algoritmo nas instâncias de complexidade média

tipo	agentes	tarefas	BTRC	BTRA	randBTRA	greedyBTRA	BTRA-EC
C	5	100	1937	**1931**	1933	**1931**	**1931**
	10	100	**1403**	**1403**	**1403**	1406	1404
	20	100	1250	1248	1246	1247	**1245**
	5	200	3460	3459	3459	3459	**3458**
	10	200	2820	2818	2815	**2814**	2823
	20	200	**2399**	2400	2409	2413	2402
D	5	100	6403	6394	6390	**6389**	6398
	10	100	6430	6425	**6423**	6424	6435
	20	100	6317	6318	**6301**	6307	6322
	5	200	12816	12802	**12799**	12802	12810
	10	200	12596	12570	12566	**12546**	12579
	20	200	12468	12449	12457	**12446**	12484
E	5	100	12748	**12696**	12700	12704	12703
	10	100	11772	**11622**	11631	11671	11645
	20	100	8646	8569	**8548**	8563	8579
	5	200	25181	**24975**	24982	24979	24998
	10	200	23609	**23366**	23400	23374	23410
	20	200	23050	22602	**22558**	22575	22613

5.2.3.2 Comparação entre as médias das melhores soluções encontradas pelos algoritmos

Outra proposta foi investigar as médias das melhores soluções encontradas por cada algoritmo. Como dito, os algoritmos foram executados 50 (cinquenta) vezes para cada instância; assim, foram analisadas as médias das soluções retornadas nesse conjunto de execuções. Conforme feito nas comparações com as melhores soluções encontradas (seção 5.2.3.1), as análises foram separadas por complexidade das instâncias.

A Tabela 5.4 apresenta a comparação entre as médias das melhores soluções encontradas pelos algoritmos aplicados às instâncias do tipo A e B. As células em negrito destacam as melhores médias encontradas. A tabela segue a mesma organização das demais apresentadas nas comparações anteriores.

Tabela 5.4: Comparação entre as médias das melhores soluções encontradas por cada algoritmo nas instâncias de menor complexidade

tipo	agentes	tarefas	BTRC	BTRA	randBTRA	greedyBTRA	BTRA-EC
A	5	100	**1698**	**1698**	1698,44	**1698**	**1698**
	10	100	**1360**	**1360**	1360,66	1360,68	**1360**
	20	100	**1158**	**1158**	1161,74	1162,1	**1158**
	5	200	**3235**	**3235**	3235,78	3235,6	**3235**
	10	200	**2623**	**2623**	2625,88	2625,94	**2623**
	20	200	**2339**	**2339**	2345,72	2345,8	**2339**
B	5	100	1856,4	1853,68	**1852,38**	1852,56	1854,2
	10	100	**1407,76**	1408,04	1410,68	1410,14	1407,96
	20	100	1167,66	**1167,44**	1174,46	1174,04	1167,54
	5	200	**3563,82**	3565,66	3564,62	3564,52	3566,52
	10	200	**2847,34**	2850,4	2851,36	2850,22	2849,26
	20	200	**2345,52**	2345,6	2357,94	2356,42	2346,78

Pela Tabela 5.4 é possível notar o mesmo comportamento visto ao comparar as melhores soluções, isto é, os algoritmos BTRC, BTRA e BTRA-EC apresentaram as melhores médias. Destaque ao BTRC, que detém praticamente todas as melhores médias nas instâncias do tipo B. Observe, também, que os algoritmos BTRC, BTRA e BTRA-EC apresentaram como média as soluções ótimas para as instâncias do tipo A (encontradas em Chu e Beasley (1997)), ou seja, em todas as execuções desses algoritmos o valor ótimo foi encontrado. As médias das melhores soluções encontradas pelos algoritmos randBTRA e greedyBTRA não conseguiram se destacar nesse teste. Mais uma vez é mostrado pelos resultados desses dois algoritmos que a solução inicial teve impacto no resultado para as instâncias de menor complexidade.

O mesmo teste foi realizado para as instâncias de complexidade média (tipo C, D e E). A Tabela 5.5 apresenta as comparações.

Tabela 5.5: Comparação entre as médias das melhores soluções encontradas por cada algoritmo nas instâncias de complexidade média

tipo	agentes	tarefas	BTRC	BTRA	randBTRA	greedyBTRA	BTRA-EC
C	5	100	1939,18	1939,04	**1938,88**	1939	1939,52
	10	100	**1410,5**	1412,4	1412,32	1412,68	1412,3
	20	100	1253,76	**1253,42**	1259,58	1259,94	1254,8
	5	200	3467,56	3468,16	**3467,36**	3468,06	3468,06
	10	200	**2825,38**	2827,98	2828,1	2826,5	2828,7
	20	200	**2410,46**	2413,5	2420,72	2419,66	2412,96
D	5	100	6429,955	**6414,34**	6416,72	6415,02	6415,3
	10	100	6464,92	**6458,82**	6459,24	6460,68	6467,42
	20	100	6343,04	**6340,8**	6342,3	6341,08	6350,62
	5	200	12842,48	12828,48	12830,7	**12825,96**	12833,94
	10	200	12612,49	12601,54	12601,32	**12600,7**	12608,08
	20	200	12508,28	**12493,32**	12494,38	12499,58	12515,98
E	5	100	12832,98	**12753,44**	12786,22	12758,34	12764,4
	10	100	11859,65	**11709,44**	11755,58	11765,48	11766,4
	20	100	8789,46	**8640,28**	8644,3	8651,48	8676,78
	5	200	25290,16	**25038,36**	25039,8	25097,44	25055,94
	10	200	24000,17	**23443,82**	23480,2	23493,56	23496,26
	20	200	23348,22	**22662,52**	22705,86	22727,96	22744,3

Na Tabela 5.5 é possível notar que o algoritmo BTRA detém a maior parte das médias vencedoras nessa comparação, seguido dos algoritmos BTRC, randBTRA e greedyBTRA com quantidades bastante inferiores. O algoritmo BTRA-EC não apresentou nenhuma média vencedora.

Observe que ao comparar as melhores soluções encontradas pelos algoritmos nas instâncias de complexidade média (seção 5.2.3.1), os algoritmos randBTRA e greedyBTRA se apresentaram mais competitivos em relação ao algoritmo BTRC. No entanto, na comparação das médias (tabela 5.5) esses algoritmos se equipararam. O algoritmo BTRA manteve-se superior nas duas comparações, ao contrário do BTRA-EC que se manteve ruim em ambas comparações. Isso pode indicar que a combinação BTRA com *Ejection chain*, implementada neste trabalho, não resolve bem as instâncias de complexidade média. O mesmo pode ser observado para a combinação implementada BT + RC, visto que o algoritmo BTRC se comportou bem somente nas instâncias de menor complexidade.

5.2.3.3 Testes Estatísticos

Para complementar a análise dos resultados, o procedimento estatístico Teste-Z foi aplicado nas soluções encontradas pelos algoritmos.

Esse procedimento permite verificar a significância da diferença entre as médias de duas amostras. Portanto, tal técnica foi utilizada a fim de identificar (por instância) se os diferentes algoritmos apresentaram soluções que os diferiram um do outro com certo grau de significância. Caso o teste aponte que existem diferenças entre os mesmos, também será identificado qual deles apresentou o melhor desempenho. Detalhes sobre o Teste-Z podem ser vistos em Ross (2004) e Montgomery e Runger (2003).

Neste trabalho, uma amostra compreende o conjunto das melhores soluções retornadas por um algoritmo a uma dada instância. Como foram realizadas 50 (cinquenta) execuções de cada algoritmo a cada instância, o tamanho das amostras analisadas foi de 50 observações.

Foram realizados testes em pares de algoritmos para cada instância, haja visto que o Teste-Z é aplicado entre as médias de duas amostras. Para a realização dos mesmos foram fornecidos os seguintes dados:

- μ_1: média dos resultados encontrados pelo $algoritmo_1$;
- μ_2: média dos resultados encontrados pelo $algoritmo_2$;
- s_1^2: variância dos resultados encontrados pelo algoritmo1;
- s_2^2: variância dos resultados encontrados pelo algoritmo2;
- n_1: quantidade de soluções dadas pelo algoritmo-1;
- n_2: quantidade de soluções dadas pelo algoritmo-2;

A quantidade Z_{calc} foi calculada conforme apresentado na Eq.(5.1). A mesma aproxima-se de uma distribuição normal padrão, com média 0 e variância 1.

$$Z_{calc} = \frac{(\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \; N(0,1) \tag{5.1}$$

Em cima desta quantidade foram realizados testes de hipóteses ((Ross, 2004) e (Montgomery e Runger, 2003)) para determinar se (i) as médias das amostras 1 e 2 são iguais (isto posto como hipótese nula $H_0 : \mu_1 = \mu_2$), ou se (ii) tais médias são diferentes ($H_1 : \mu_1 < \mu_2$ ou $\mu_1 > \mu_2$, como hipótese alternativa).

O grau de significância dos testes foi de $\alpha = 0,05$, ou seja, existe uma chance de 5% de rejeitar a hipótese H_0 sendo ela verdadeira. Em experimentos científicos, esse valor é considerado confiável para confirmar resultados avaliados sob a luz da estatística. Por esse motivo, foi considerado esse nível de significância suficiente para as análises.

Inicialmente, foram avaliados os algoritmos BTRC e BTRA. Os testes foram aplicados às instâncias do tipo B, C, D e E. Instâncias do tipo A ficaram de fora dos testes devido ao fato de a variância dos resultados apresentados pelos algoritmos ser

igual a zero, haja vista que os mesmos retornaram as soluções ótimas em todas as execuções para tais instâncias (seção 5.2.3.2).

Foram realizados 24 testes considerando as instâncias citadas. A Tabela 5.6 apresenta os principais resultados desses testes.

Tabela 5.6: Resultados do Teste-Z aplicado aos algoritmos BTRC e BTRA

			BTRC			BTRA			Teste-Z		
tipo	agentes	tarefas	μ_1	s_1^2	n_1	μ_2	s_2^2	n_1	Z_{calc}	p-valor	$Z_{critico}$
B	5	100	1856,40	31,02	50	1853,68	17,94	50	2,7488	0,0030	1,6449
	10	100	1407,76	1,86	50	1408,04	1,88	50	-1,0244	0,1528	1,6449
	20	100	1167,66	1,94	50	1167,44	3,39	50	0,6733	0,2504	1,6449
	5	200	3563,82	13,58	50	3565,66	6,64	50	-2,8937	0,0019	1,6449
	10	200	2847,34	34,84	50	2850,40	17,67	50	-2,9858	0,0014	1,6449
	20	200	2345,52	6,50	50	2345,60	5,47	50	-0,1635	0,4351	1,6449
C	5	100	1939,18	17,82	50	1939,04	13,47	50	0,1770	0,4298	1,6449
	10	100	1410,50	15,03	50	1412,40	9,51	50	-2,7120	0,0033	1,6449
	20	100	1253,76	12,39	50	1253,42	13,60	50	0,4716	0,3186	1,6449
	5	200	3467,56	17,76	50	3468,16	10,01	50	-0,8050	0,2104	1,6449
	10	200	2825,38	25,46	50	2827,98	16,35	50	-2,8432	0,0022	1,6449
	20	200	2410,46	25,93	50	2413,50	25,40	50	-3,0005	0,0013	1,6449
D	5	100	6429,95	221,72	50	6414,34	108,31	50	5,8171	0,0000	1,6449
	10	100	6464,92	324,93	50	6458,82	159,46	50	1,9598	0,0250	1,6449
	20	100	6343,04	325,39	50	6340,80	142,41	50	0,7323	0,2320	1,6449
	5	200	12842,48	273,76	50	12828,48	118,30	50	4,9996	0,0000	1,6449
	10	200	12612,49	661,08	50	12601,54	200,13	50	2,2878	0,0111	1,6449
	20	200	12508,28	556,65	50	12493,32	280,30	50	3,6565	0,0001	1,6449
E	5	100	12832,98	2760,96	50	12753,44	1405,93	50	8,7129	0,0000	1,6449
	10	100	11859,65	3455,83	50	11709,44	1861,97	50	14,1706	0,0000	1,6449
	20	100	8789,46	6861,80	50	8640,28	1087,19	50	11,8315	0,0000	1,6449
	5	200	25290,16	13974,71	50	25038,36	1845,79	50	14,1557	0,0000	1,6449
	10	200	24000,17	21551,67	50	23443,82	1061,21	50	25,6564	0,0000	1,6449
	20	200	23348,22	43388,69	50	22662,52	1454,34	50	23,1168	0,0000	1,6449

A partir da Tabela 5.6 é possível fazer algumas análises. Note que para a instância B5x100, o teste retornou um p-valor igual a 0,00299, portanto, é possível rejeitar a hipótese nula H_0 e aceitar a hipótese alternativa H_1 com um nível de confiança acima de 95%, ou seja, as médias das amostras são diferentes. Além disso, ao observar a quantidade Z_{calc} calculada notará que foi maior que o $Z_{critico}$ (2,7489 > 1,64). Isso confirma que, em média, as soluções encontradas por BTRC são maiores que as soluções dadas por BTRA, nessa instância.

Outra situação é dada para a instância B10X100. Observe que a hipótese nula não é rejeitada a $\alpha = 0{,}05$, pois o p-valor é maior que α (0,15 > 0,05), além disso, Z_{calc} é maior que -$Z_{critico}$ e menor que $Z_{critico}$ (-1,64 < -1,024 < 1,64). Nesse caso, não é possível afirmar que há diferença entre a médias das soluções encontradas por BTRC e BTRA.

Por fim, a terceira condição possível no teste é vista nos resultados apresentados para a instância B5X200. A hipótese H_0 é rejeitada a $\alpha = 0{,}05$, pois o p-valor é menor que α (0,0014 < 0,05) e Z_{calc} é menor que -$Z_{critico}$ (-2,985 < -1,64). Nesse caso, pode-se dizer que a média das soluções encontradas por BTRC é menor do que a média das soluções de BTRA.

Ao sumarizar as análises, foi possível notar que dados os 24 testes, 12 deles mostraram que o BTRA apresentou soluções melhores, o BTRC foi responsável por apresentar soluções melhores para cinco instâncias e, para sete delas, não foi possível definir diferença entre os algoritmos. A Figura 5.2 apresenta um gráfico com essa

sumarização, bem como mostra outros dois gráficos com resultados de comparações entre esses dois algoritmos ao considerar as melhores soluções encontradas por cada um e as médias das melhores soluções.

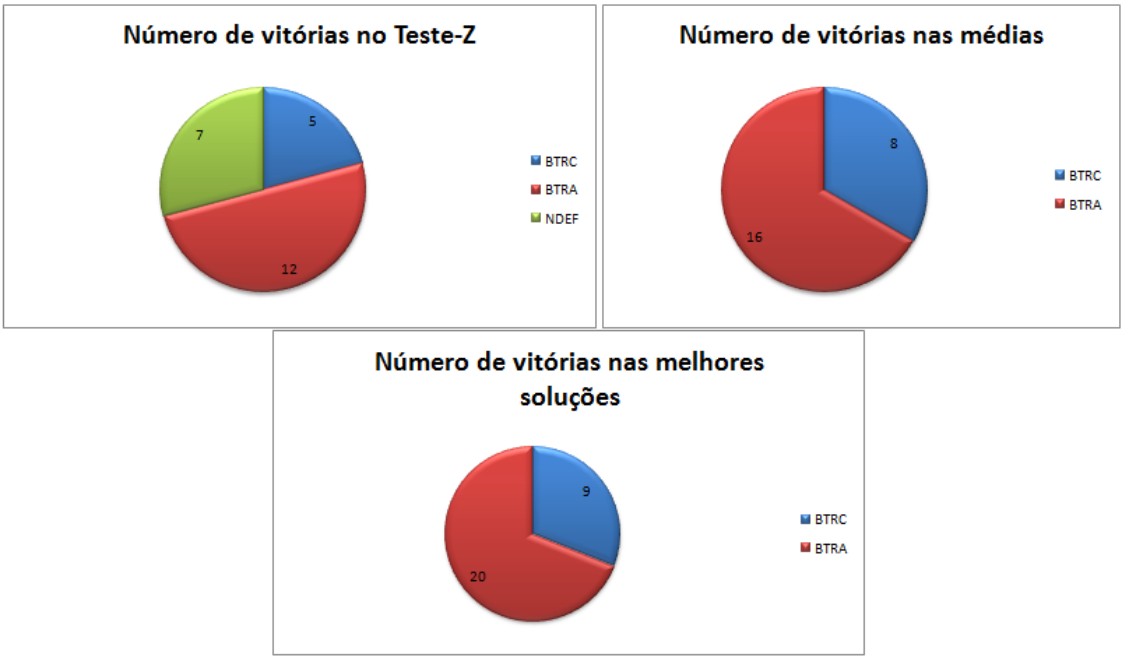

Figura 5.2: Comparações entre os algoritmos BTRC e BTRA

Note pelos gráficos apresentados na Figura 5.2 que o algoritmo BTRA foi superior ao BTRC em todas as comparações. Fica claro que a combinação Busca Tabu + Relaxação Adaptativa implementada neste trabalho apresentou melhores soluções em relação à combinação da Busca Tabu com Reconexão por Caminhos.

Dado que o BTRA foi superior ao BTRC e que os outros algoritmos são variações do BTRA, os próximos testes foram feitos entre esse algoritmo e os demais implementados. A metodologia e as análises seguem os mesmos critérios aplicados nos testes entre BTRC e BTRA.

O próxima comparação foi realizada com o algoritmo randBTRA. A Tabela 5.7 apresenta os resultados do Teste-Z aplicado nas soluções dos dois algoritmos.

A Figura 5.3 apresenta a análise sumarizada dos resultados do Teste-Z, além da comparação entre os dois algoritmos no que se refere às melhores soluções encontradas por cada um e as médias das melhores soluções.

Observe na Figura 5.3 que pelo Teste-Z o número de vitórias do algoritmo BTRA é muito superior ao número apresentado por randBTRA; porém, para a maioria das instâncias não foi possível definir qual algoritmo foi melhor. Ao considerar apenas as médias das soluções dadas por algoritmo, sem levar em consideração a variância amostral, o BTRA também se apresentou melhor. Entretanto, ao comparar os dois algoritmos em relação às melhores soluções encontradas, os algoritmos praticamente empataram. Portanto, a definição do melhor algoritmo depende do critério. Ao considerar as médias, o algoritmo BTRA se destaca. Porém, não é possível definir o algoritmo vencedor na comparação ao considerar as melhores soluções.

Tabela 5.7: Resultados do Teste-Z aplicado aos algoritmos BTRA e randBTRA

			BTRA			randBTRA			Teste-Z		
tipo	agentes	tarefas	μ_1	s_1^2	n_1	μ_2	s_2^2	n_1	Z_{calc}	p-valor	$Z_{critico}$
B	5	100	1853,68	17,94	50	1852,38	13,34	50	1,6436	0,0501	1,6449
	10	100	1408,04	1,88	50	1410,68	4,22	50	-7,5596	0,0000	1,6449
	20	100	1167,44	3,39	50	1174,46	6,01	50	-16,1880	0,0000	1,6449
	5	200	3565,66	6,64	50	3564,62	8,81	50	1,8710	0,0307	1,6449
	10	200	2850,40	17,67	50	2851,36	16,24	50	-1,1657	0,1219	1,6449
	20	200	2345,60	5,47	50	2357,94	7,65	50	-24,0909	0,0000	1,6449
C	5	100	1939,04	13,47	50	1938,88	14,68	50	0,2133	0,4156	1,6449
	10	100	1412,40	9,51	50	1412,32	7,90	50	0,1356	0,4461	1,6449
	20	100	1253,42	13,60	50	1259,58	7,84	50	-9,4079	0,0000	1,6449
	5	200	3468,16	10,01	50	3467,36	10,97	50	1,2349	0,1084	1,6449
	10	200	2827,98	16,35	50	2828,10	16,62	50	-0,1478	0,4413	1,6449
	20	200	2413,50	25,40	50	2420,72	25,68	50	-7,1438	0,0000	1,6449
D	5	100	6414,34	108,31	50	6416,72	105,96	50	-1,1497	0,1251	1,6449
	10	100	6458,82	159,46	50	6459,24	157,33	50	-0,1669	0,4337	1,6449
	20	100	6340,80	142,41	50	6342,30	167,11	50	-0,6029	0,2733	1,6449
	5	200	12828,48	118,30	50	12830,70	104,83	50	-1,0509	0,1466	1,6449
	10	200	12601,54	200,13	50	12601,32	266,51	50	0,0720	0,4713	1,6449
	20	200	12493,32	280,30	50	12494,38	300,65	50	-0,3110	0,3779	1,6449
E	5	100	12753,44	1405,93	50	12786,22	8224,87	50	-2,3619	0,0091	1,6449
	10	100	11709,44	1861,97	50	11755,58	4635,51	50	-4,0475	0,0000	1,6449
	20	100	8640,28	1087,19	50	8644,30	1115,64	50	-0,6056	0,2724	1,6449
	5	200	25038,36	1845,79	50	25039,80	2024,45	50	-0,1637	0,4350	1,6449
	10	200	23443,82	1061,21	50	23480,20	5121,80	50	-3,2715	0,0005	1,6449
	20	200	22662,52	1454,34	50	22705,86	3900,41	50	-4,1880	0,0000	1,6449

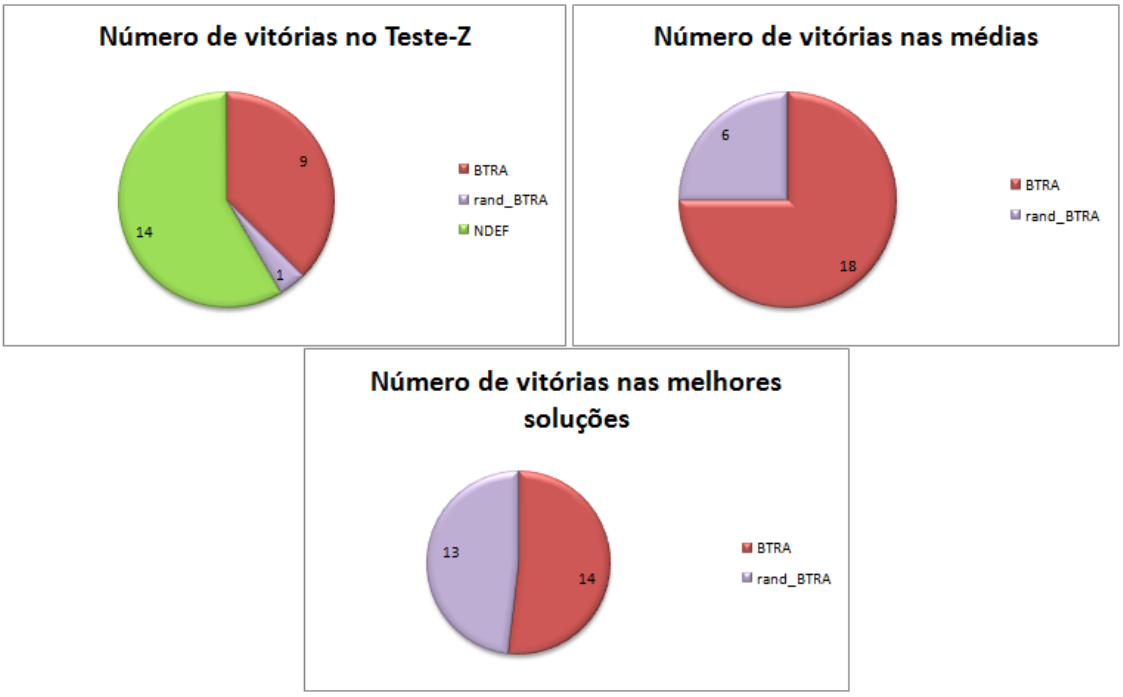

Figura 5.3: Comparações entre os algoritmos BTRA e randBTRA

O teste seguinte foi realizado entre o algoritmo BTRA e greedyBTRA. A Tabela 5.8 apresenta os resultados do Teste-Z aplicado nas soluções desses algoritmos.

A análise sumarizada dos resultados do Teste-Z, bem como a comparação entre

Tabela 5.8: Resultados do Teste-Z aplicado aos algoritmos BTRA e greedyBTRA

			BTRA			greedyBTRA			Teste-Z		
tipo	agentes	tarefas	μ_1	s_1^2	n_1	μ_2	s_2^2	n_1	Z_{calc}	p-valor	$Z_{critico}$
B	5	100	1853,68	17,94	50	1852,56	18,46	50	1,3128	0,0946	1,6449
	10	100	1408,04	1,88	50	1410,14	3,67	50	-6,3033	0,0000	1,6449
	20	100	1167,44	3,39	50	1174,04	9,35	50	-13,0753	0,0000	1,6449
	5	200	3565,66	6,64	50	3564,52	7,68	50	2,1302	0,0166	1,6449
	10	200	2850,40	17,67	50	2850,22	21,36	50	0,2037	0,4193	1,6449
	20	200	2345,60	5,47	50	2356,42	10,62	50	-19,0765	0,0000	1,6449
C	5	100	1939,04	13,47	50	1939,00	9,80	50	0,0586	0,4766	1,6449
	10	100	1412,40	9,51	50	1412,68	15,77	50	-0,3938	0,3469	1,6449
	20	100	1253,42	13,60	50	1259,94	10,83	50	-9,3279	0,0000	1,6449
	5	200	3468,16	10,01	50	3468,06	8,79	50	0,1631	0,4352	1,6449
	10	200	2827,98	16,35	50	2826,50	20,26	50	1,7298	0,0418	1,6449
	20	200	2413,50	25,40	50	2419,66	23,09	50	-6,2556	0,0000	1,6449
D	5	100	6414,34	108,31	50	6415,02	65,08	50	-0,3652	0,3575	1,6449
	10	100	6458,82	159,46	50	6460,68	181,49	50	-0,7123	0,2381	1,6449
	20	100	6340,80	142,41	50	6341,08	195,26	50	-0,1077	0,4571	1,6449
	5	200	12828,48	118,30	50	12825,96	73,59	50	1,2864	0,0992	1,6449
	10	200	12601,54	200,13	50	12600,70	235,44	50	0,2846	0,3880	1,6449
	20	200	12493,32	280,30	50	12499,58	303,96	50	-1,8313	0,0335	1,6449
E	5	100	12753,44	1405,93	50	12758,34	1282,31	50	-0,6683	0,2520	1,6449
	10	100	11709,44	1861,97	50	11765,48	3186,30	50	-5,5771	0,0000	1,6449
	20	100	8640,28	1087,19	50	8651,48	1036,34	50	-1,7186	0,0428	1,6449
	5	200	25038,36	1845,79	50	25097,44	4256,17	50	-5,3480	0,0000	1,6449
	10	200	23443,82	1061,21	50	23493,56	4268,66	50	-4,8176	0,0000	1,6449
	20	200	22662,52	1454,34	50	22727,96	3566,86	50	-6,5302	0,0000	1,6449

os dois algoritmos no que se refere às melhores soluções e as médias das melhores soluções podem ser vistas na Figura 5.4

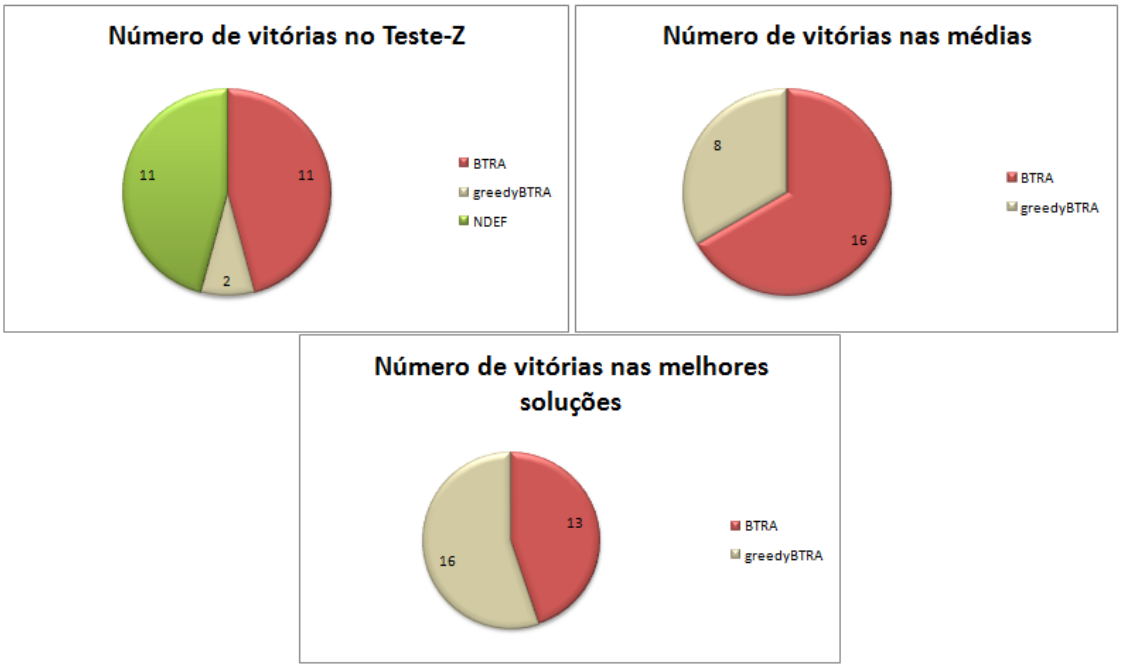

Figura 5.4: Comparações entre os algoritmos BTRA e greedyBTRA

Pelo Teste-Z é possível notar a superioridade do algoritmo BTRA, como pode ser visto na Figura 5.4. Esse resultado confirma a comparação feita pelas médias,

quando a variância amostral não é considerada . Porém, quando o objetivo é avaliar as melhores soluções, note que o algoritmo greedyBTRA apresentou uma leve vantagem. Desta maneira, ao considerar as médias, o algoritmo BTRA pode ser dado como superior, porém, quando a comparação é feita pelas melhores solução, não é possível destacar um algoritmo, apesar da vantagem apresentada pelo greedyBTRA.

Por fim, o último teste foi realizado entre os algoritmos BTRA e BTRA-EC. A Tabela 5.9 apresenta os resultados do Teste-Z aplicado nas soluções dos algoritmos.

Tabela 5.9: Resultados do Teste-Z aplicado aos algoritmos BTRA e BTRA-EC

			BTRA			BTRA-EC			Teste-Z		
tipo	agentes	tarefas	μ_1	s_1^2	n_1	μ_2	s_2^2	n_1	Z_{calc}	p-valor	$Z_{critico}$
B	5	100	1853,68	17,94	50	1854,20	15,59	50	-0,6350	0,2627	1,6449
	10	100	1408,04	1,88	50	1407,96	2,20	50	0,2801	0,3897	1,6449
	20	100	1167,44	3,39	50	1167,54	2,25	50	-0,2975	0,3830	1,6449
	5	200	3565,66	6,64	50	3566,52	6,50	50	-1,6778	0,0467	1,6449
	10	200	2850,40	17,67	50	2849,26	17,05	50	1,3679	0,0857	1,6449
	20	200	2345,60	5,47	50	2346,78	8,34	50	-2,2455	0,0124	1,6449
C	5	100	1939,04	13,47	50	1939,52	9,28	50	-0,7117	0,2383	1,6449
	10	100	1412,40	9,51	50	1412,30	17,48	50	0,1361	0,4459	1,6449
	20	100	1253,42	13,60	50	1254,80	20,86	50	-1,6625	0,0482	1,6449
	5	200	3468,16	10,01	50	3468,06	9,08	50	0,1618	0,4357	1,6449
	10	200	2827,98	16,35	50	2828,70	18,87	50	-0,8579	0,1955	1,6449
	20	200	2413,50	25,40	50	2412,96	36,41	50	0,4857	0,3136	1,6449
D	5	100	6414,34	108,31	50	6415,30	123,07	50	-0,4463	0,3277	1,6449
	10	100	6458,82	159,46	50	6467,42	208,13	50	-3,1718	0,0008	1,6449
	20	100	6340,80	142,41	50	6350,62	201,46	50	-3,7445	0,0001	1,6449
	5	200	12828,48	118,30	50	12833,94	112,67	50	-2,5404	0,0055	1,6449
	10	200	12601,54	200,13	50	12608,08	265,63	50	-2,1428	0,0161	1,6449
	20	200	12493,32	280,30	50	12515,98	442,31	50	-5,9606	0,0000	1,6449
E	5	100	12753,44	1405,93	50	12764,40	1241,35	50	-1,5062	0,0660	1,6449
	10	100	11709,44	1861,97	50	11766,40	1969,55	50	-6,5068	0,0000	1,6449
	20	100	8640,28	1087,19	50	8676,78	1945,24	50	-4,6869	0,0000	1,6449
	5	200	25038,36	1845,79	50	25055,94	1688,22	50	-2,0911	0,0183	1,6449
	10	200	23443,82	1061,21	50	23496,26	2794,97	50	-5,9713	0,0000	1,6449
	20	200	22662,52	1454,34	50	22744,30	3367,68	50	-8,3276	0,0000	1,6449

A Figura 5.5 apresenta a análise sumarizada dos resultados do Teste-Z, além da comparação entre os dois algoritmos no que se refere às melhores soluções encontradas por cada um e as médias das melhores soluções, sem considerar a variância amostral.

Neste caso, conforme apresentado nos gráficos da Figura 5.5, é possível notar a superioridade do algoritmo BTRA. No Teste-Z esse algoritmo apresentou as melhores médias para a maioria das instâncias. Nos outros testes o BTRA também se destacou em relação ao BTRA-EC. Portanto, para as instâncias testadas a inserção do módulo *Ejection Chain*, conforme implementado, não melhorou o desempenho da Busca Tabu com Relaxação Adaptativa.

5.3 Resultados Multiobjetivo

O algoritmo proposto na seção 4.2, o qual será referenciado como enNSGAII, foi aplicado para resolver 6 instâncias da OR-Library (Beasley):

- $A5 \times 100$: 5 agentes e 100 tarefas;

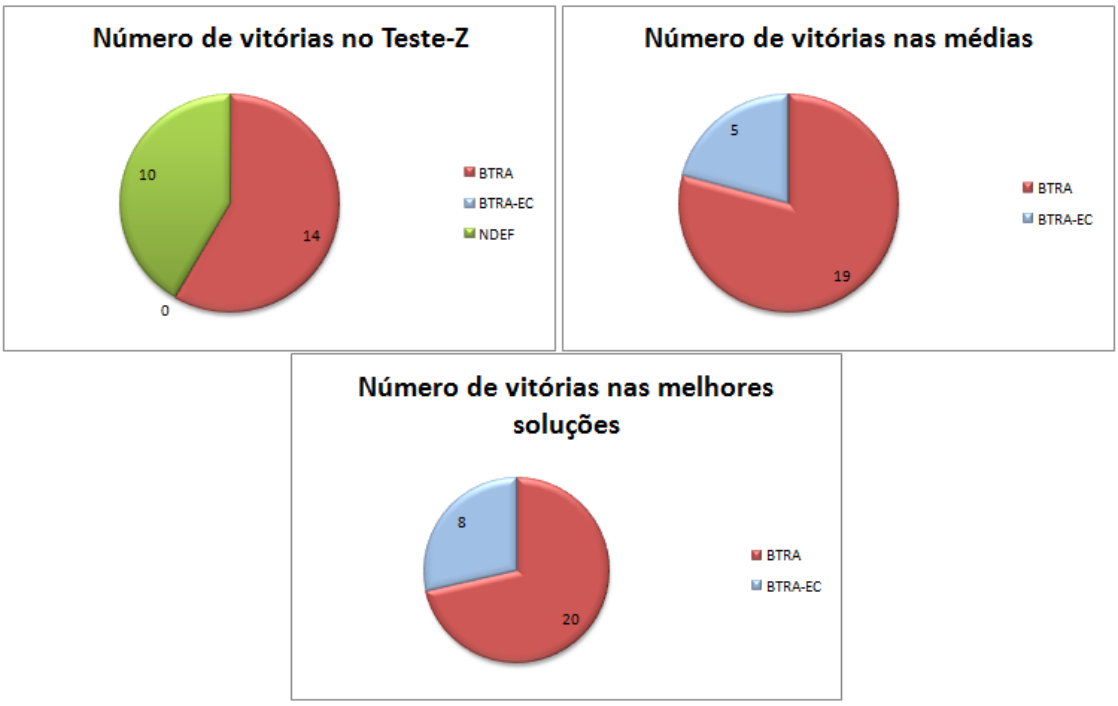

Figura 5.5: Comparações entre os algoritmos BTRA e BTRA-EC

- $A5 \times 200$: 5 agentes e 200 tarefas;
- $A10 \times 100$: 10 agentes e 100 tarefas;
- $A10 \times 200$: 10 agentes e 200 tarefas;
- $A20 \times 100$: 20 agentes e 100 tarefas;
- $A20 \times 200$: 20 agentes e 200 tarefas.

O ótimo global para estas instâncias, na versão mono-objetivo do PGA, são conhecidos e podem ser encontrados na OR-Library. Nos experimentos computacionais realizados, estas soluções também foram avaliadas com relação à Função de Equilíbrio, com o intuito de servirem de *benchmark*.

Os resultados obtidos pelo NSGA-II básico (baNSGAII) também são apresentados. Este algoritmo tem a mesma estrutura e parâmetros do enNSGAII, as diferenças são apenas em relação aos operadores de cruzamento e mutação, conforme apresentados na seção 4.2.

A abordagem proposta não foi comparada com outras abordagens de otimização multiobjetivo, pois até então não foram encontrados algoritmos na literatura que tratam o PGA usando esse conjunto de funções (custo da solução e homogeneidade nas atribuições das tarefas).

Os dois algoritmos foram parametrizados como segue:

- Tamanho da população: 50 indivíduos;
- Critério de parada: número máximo de gerações;

- Número máximo de gerações: 300 gerações;
- Probabilidade de cruzamento: 0,90;
- Probabilidade de mutação: 0,03.

Os resultados observados são discutidos nas próximas seções.

5.3.1 Instância $A5 \times 100$

O ótimo global (x^*) para tal instância mono-objetivo tem os seguintes valores de funções:

- $f_C(x^*) = 1698$ e $f_E(x^*) = 53$.

O enNSGAII obteve as seguintes soluções de custos mínimos (x^*_{GA}):

- $f_C(x^*_{GA}) = 1698$ e $f_E(x^*_{GA}) = 53$.

Esse resultado mostra que o algoritmo proposto foi capaz de encontrar a solução ótima para essa instância. Além disso, o algoritmo poderia encontrar outras 8 soluções eficientes, como pode ser visto na Figura 5.6. Observe que as soluções obtidas pelo enNSGAII dominam as soluções encontradas pelo baNSGAII.

A partir do conjunto de pontos eficientes alcançados pelo enNSGAII é possível verificar as vantagens da abordagem multiobjetivo quando comparada com apenas um objetivo. Por exemplo, uma solução designada como escolhida ('o' azul) melhora a Função de Equilíbrio de x^* em 60,4% enquanto aumenta a Função de Custo em apenas 0,1%.

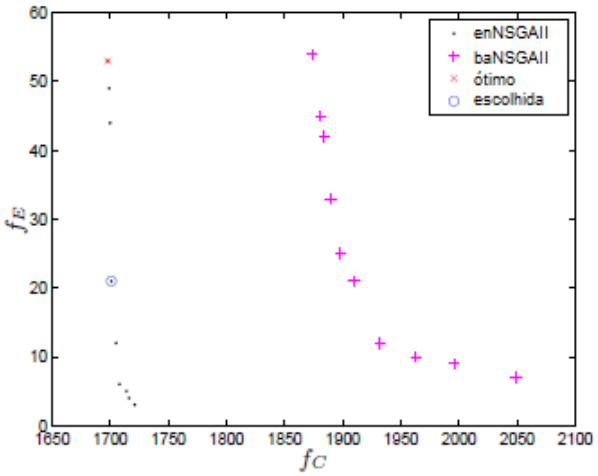

Figura 5.6: Instância $A5 \times 100$ – Conjunto de soluções eficientes

5.3.2 Instância $A5 \times 200$

O ótimo conhecido (x^*) para tal instância é:

- $f_C(x^*) = 3235$ e $f_E(x^*) = 135$.

O algoritmo proposto encontrou uma solução de custo mínimo (x^*_{GA}) com os seguintes valores de função:

- $f_C(x^*_{GA}) = 3249$ e $f_E(x^*_{GA}) = 57$.

Neste caso, o enNSGAII encontrou uma solução de custo mínimo próxima do ótimo global da versão mono-objetivo do problema. O enNSGAII também encontrou 14 soluções eficientes, como visto na Figura 5.7. Mais uma vez, o enNSGAII claramente dominou o baNSGAII.

A solução escolhida neste caso melhorou $f_E(x^*)$ em $93,3\%$ com apenas $1,7\%$ de perda em relação à $f_C(x^*)$.

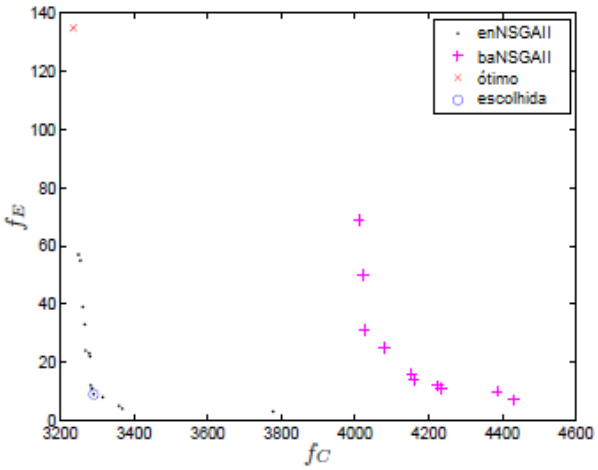

Figura 5.7: Instância $A5 \times 200$ – Conjunto de soluções eficientes

5.3.3 Instância $A10 \times 100$

Ótimo global (x^*):

- $f_C(x^*) = 1360$ e $f_E(x^*) = 94$.

Solução de custo mínimo (x^*_{GA}) alcançada pelo enNSGAII:

- $f_C(x^*_{GA}) = 1365$ e $f_E(x^*_{GA}) = 76$.

Assim como para a instância $A5 \times 200$, a solução de custo mínimo encontrada pelo enNSGAII é um pouco maior em relação ao valor ótimo, porém melhora em relação à Função de Equilíbrio. O conjunto de soluções eficientes para essa instância é apresentado na Figura 5.8. É possível observar que o enNSGAII encontrou soluções melhores em relação ao baNSGAII.

A solução escolhida para este caso melhora o ótimo global em $71,3\%$ em relação à f_E, com uma perda de $1,3\%$ em relação à f_C.

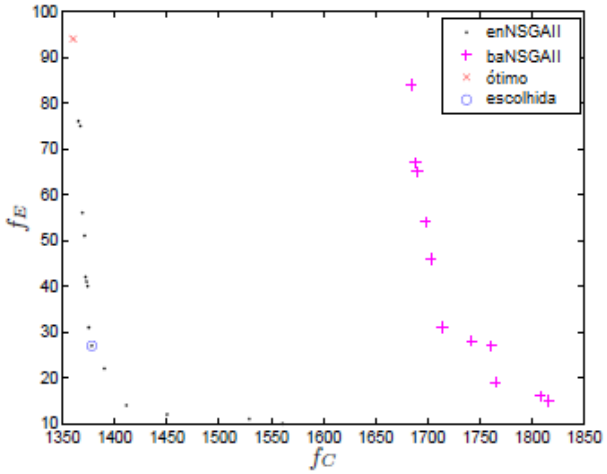

Figura 5.8: Instância $A10 \times 100$ – Conjunto de soluções eficientes

5.3.4 Instância $A10 \times 200$

Ótimo global (x^*):

- $f_C(x^*) = 2623$ e $f_E(x^*) = 127$.

Solução de custo mínimo (x^*_{GA}) alcançada pelo enNSGAII:

- $f_C(x^*_{GA}) = 2677$ e $f_E(x^*_{GA}) = 69$.

O conjunto de soluções eficientes é dado na Figura 5.9. Os resultados que foram observados seguem o mesmo comportamento das instâncias anteriores com relação a solução de custo mínimo e domínio do conjunto de soluções eficientes apresentados pelo baNSGAII.

A solução que foi escolhida neste caso apresenta $73,2\%$ de ganho em $f_E(x^*)$ e apenas 2.7% de perda em relação à $f_C(x^*)$.

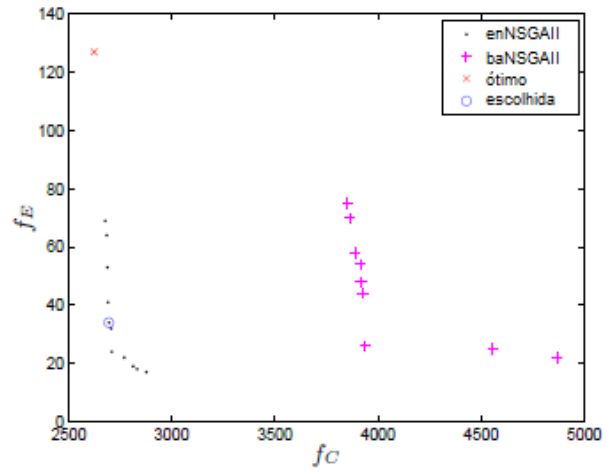

Figura 5.9: Instância $A10 \times 200$ – Conjunto de soluções eficientes

5.3.5 Instância $A20 \times 100$

Ótimo global (x^*):

- $f_C(x^*) = 1158$ e $f_E(x^*) = 94$.

Solução de custo mínimo (x^*_{GA}) alcançada pelo enNSGAII:

- $f_C(x^*_{GA}) = 1176$ and $f_E(x^*_{GA}) = 69$.

O conjunto de soluções eficientes é dado na Figura 5.10. Os resultados que foram observados seguem o mesmo comportamento das instâncias anteriores com relação à solução de custo mínimo e domínio do conjunto de soluções eficientes apresentados pelo baNSGAII.

A solução escolhida neste caso apresenta $69,1\%$ de ganho em $f_E(x^*)$ e $3,6\%$ de perda em relação à $f_C(x^*)$.

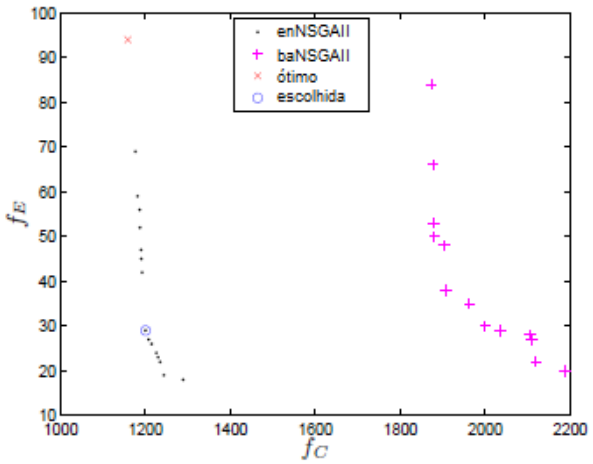

Figura 5.10: Instância $A20 \times 100$ – Conjunto de soluções eficientes

5.3.6 Instância $A20 \times 200$

Ótimo global (x^*):

- $f_C(x^*) = 2339$ e $f_E(x^*) = 109$.

Solução de custo mínimo (x^*_{GA}) alcançada pelo enNSGAII:

- $f_C(x^*_{GA}) = 2457$ e $f_E(x^*_{GA}) = 97$.

O conjunto de soluções eficientes é dado na Figura 5.11. Os resultados que foram observados seguem o mesmo comportamento das instâncias anteriores com relação a solução de custo mínimo e domínio do conjunto de soluções eficientes apresentados pelo baNSGAII.

A solução escolhida neste caso apresenta $52,8\%$ de ganho em $f_E(x^*)$ e $6,2\%$ de perda em relação à $f_C(x^*)$.

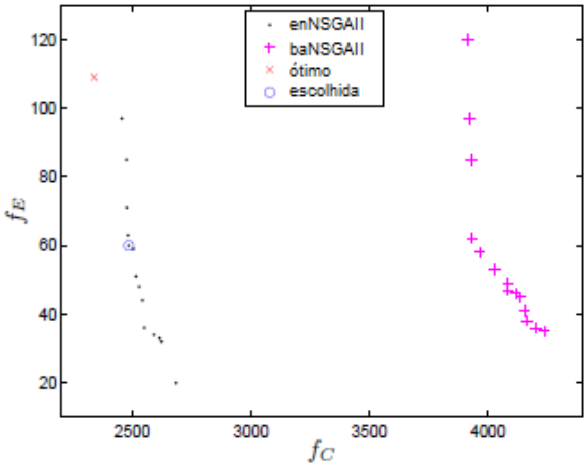

Figura 5.11: Instância $A20 \times 200$ – Conjunto de soluções eficientes

Uma vez que não foram encontrados trabalhos publicados nos quais o problema em questão é resolvido, não foi possível realizar uma comparação efetiva de resultados com outras abordagens já publicadas. Entretanto, observando os resultados apresentados, pode-se ver que a metodologia proposta nesta dissertação é capaz de encontrar um conjunto de soluções não dominadas ao comparar com a implementação do algoritmo NSGA-II básico. Além disso, os resultados apresentam um bom comportamento do enNSGAII na análise de *trade-off* ao ser comparado com as soluções ótimas já conhecidas para o problema mono-objetivo. Desta forma, nesta dissertação apresenta-se uma metodologia eficiente para o tratamento da versão multiobjetivo do PGA.

Capítulo 6

Considerações Finais

6.1 Conclusões

Este trabalho discutiu e propôs métodos de solução para duas versões do Problema Generalizado de Atribuição (PGA). Na primeira, o PGA é abordado na sua forma clássica mono-objetivo, onde se deseja encontrar o menor custo para a atribuição de n tarefas a m agentes, sendo que cada tarefa deve ser atribuída a apenas um agente, sujeito à capacidade dos mesmos. Na segunda versão é inserido um segundo objetivo, denominado Função de Equilíbrio, com vistas à minimização da diferença na quantidade de tarefas atribuídas entre os agentes mais requisitado e menos requisitado. Assim, com o PGA multiobjetivo se deseja encontrar o menor custo das atribuições e, ao mesmo tempo, distribuir as tarefas de maneira homogênea entre os agentes.

Para resolver o PGA mono-objetivo, foram propostos cinco algoritmos heurísticos híbridos (BTRC, BTRA, randBTRA, greedyBTRA e BTRA-EC) combinando os métodos *Simulated Annealing*, Descida em Vizinhança Variável (*Variable Neighborhood Descent*, Busca Tabu com Relaxação Adaptativa, Reconexão por Caminhos (RC) e *Ejection chain*. Foram realizados testes computacionais e comparações dos algoritmos BTRC e BTRA com outras abordagens propostas na literatura. Nestes testes, foi possível verificar que os algoritmos propostos encontraram soluções iguais e até melhores em relação a algumas das soluções presentes na literatura. Um teste de probabilidade empírica, seguindo a metodologia de Aiex et al. (2002), também foi aplicado entre os algoritmos BTRC e BTRA para verificar a capacidade dos mesmos de alcançarem um valor alvo. Nestes testes, o algoritmo BTRA se destacou em relação ao BTRC. Por fim, foram aplicados testes estatísticos com o objetivo de verificar a existência de diferenças significativas entre as médias das melhores soluções encontradas pelos algoritmos propostos. Para tanto, foi utilizado o Teste-Z nos pares de algoritmos: BTRC e BTRA; BTRA e randBTRA; BTRA e greedyBTRA; BTRA e BTRA-EC. Os resultados mostraram que o algoritmo BTRA foi superior aos demais, ou seja, as melhores soluções apresentadas por esse algoritmo possuem médias significativamente menores em relação aos demais, quando aplicado à maioria das instâncias testadas. Ao confrontar os pares citados em relação às melhores soluções encontradas, o BTRA foi superior aos algoritmos BTRC e BTRA-EC, e empatou com os algoritmos randBTRA e greedyBTRA.

Para a formulação multiobjetivo do problema, este trabalho propôs uma versão

discreta melhorada do algoritmo NSGA-II, na qual foram desenvolvidos novos operadores de cruzamento e mutação. Foram realizados testes com o algoritmo utilizando as instâncias do tipo A, presentes na OR-Library (Beasley). O algoritmo desenvolvido (enNSGA-II) foi comparado, primeiramente, com o NSGA-II(Deb et al., 2002). Nesta comparação, foi possível observar que em todos os casos as soluções encontradas pelo enNSGA-II dominam as soluções dadas pelo NSGAII. Além disso, o emprego da abordagem multiobjetivo proporciona uma análise adicional, a qual é o *trade-off* entre o custo de atribuição e a distribuição homogênea das tarefas. Tal análise foi feita comparando os resultados apresentados pelo enNSGA-II com os valores ótimos conhecidos na literatura para a versão mono-objetivo do PGA. Nesta análise foi possível observar que o enNSGA-II foi capaz de encontrar soluções que estão perto do ótimo global dos problemas e ainda proporcionar uma melhoria significativa na distribuição homogênea das tarefas, com acréscimo pequeno na função de custo. A existência de tais soluções favoráveis na análise de *trade-off*, sugere que a formulação proposta para o PGA Multiobjetivo pode ser muito útil em contextos práticos.

6.2 Trabalhos Futuros

Em relação ao PGA mono-objetivo, seguem as sugestões de trabalhos futuros:

- Implementar buscal local durante a inserção de atributos na Reconexão por Caminhos;
- Implementar outras formas de aplicação do mecanismo *Ejection Chain* ao PGA;
- Estudar outros algoritmos de refinamento para aplicação ao PGA;
- Implementar outros métodos de geração da solução inicial.

Para o PGA multiobjetivo, as sugestões são:

- Implementar outros métodos de geração da população inicial;
- Testar outros operadores genéticos;
- Aplicar outros métodos evolutivos multiobjetivos, como o SPEA2;
- Aplicar metaheurísticas baseadas em busca local, como, por exemplo, o *Multiobjective Variable Neighborhood Search* (MOVNS).

6.3 Publicações

Nesta seção estão relacionados os produtos originados a partir desta pesquisa e que foram publicados em anais de eventos nacionais e internacionais.

1. Subtil, R. F.; Carrano, E. G.; Souza, M. J. F. e Takahashi, R. H. C., *Using an enhanced integer NSGA-II for solving the Multiobjective Generalized Assignment Problem*, **IEEE World Congress on Computational Intelligence**, 2010, Barcelona.

2. Subtil, R. F.; Souza, M. J. F. e de Souza, S. R., Uma abordagem híbrida aplicada ao problema generalizado de atribuição. **XIII EMC - Encontro de Modelagem Matemática e Computacional**, 2010, Nova Friburgo (RJ).

3. Subtil, R. F.; Carrano, E. G.; Souza, M. J. F.; Takahashi, R. H. C. e deSouza, S. R., *Improved genetic operators for the multiobjective generalized assignment problem*, **VIII ENIA - Encontro Nacional de Inteligência Artificial**, 2011, Natal.

4. Subtil, R. F.; Souza, M. J. F. e de Souza, S. R., Um novo algoritmo híbrido para resolução do problema generalizado de atribuição, **XIV Simpósio de Pesquisa Operacional & Logística da Marinha**, 2011, Rio de Janeiro.

Referências Bibliográficas

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2002). Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373.

Amini, M. M. e Racer, M. (1995). A hybrid heuristic for the generalized assignment problem. *European Journal of Operational Research*, v. 87, p. 343–348.

Balachandran, V. (1976). An integer generalized transportation model for optimal job assignment in computer networks. *Operations Research*, v. 24, p. 742–759.

Beasley, J. E. Or-library - `http://www.brunel.ac.uk/depts/ma/research/jeb/orlib/gapinfo.html`,). Acessado em 01 de Fevereiro de 2010.

Carrano, E. G.; Soares, L. A. E.; Takahashi, R. H. C.; Saldanha, R. R. e Neto, O. M. (2006). Electric distribution network multiobjective design using a problem-specific genetic algorithm. *IEEE Transactions on Power Delivery*, v. 21, p. 995–1005.

Cattrysse, D. G.; Salomom, M. e Wassenhove, L. N. Van. (1994). A set partitioning heuristic for the generalized assignment problem. *European Journal of Operational Research*, v. 72, p. 167–174.

Cavique, L.; Rego, C. e Themido, I. (1999). Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, v. 50, p. 608 – 616.

Chu, P. C. e Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. *Computers and Operations Research*, v. 24, p. 17–23.

Coello, C. A. C. (2000). An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, v. 32, n. 2, p. 109–143.

Dawande, M. e Kalagnanam, J. (1998). The multiple knapsack problem with color constraints. Relatório técnico, IBM T. J. Watson Research.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA II. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 2, p. 182–197.

Diaz, J. A. e Fernandez, E. (2001). A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, v. 132, p. 22–38.

Fisher, M. L. e Jaikumar, R. (2006). A generalized assignment heuristic for vehicle routing. *Networks*, v. 11, p. 109–124.

Fonseca, C. M. e Fleming, P.J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, v. 3, n. 1, p. 1–16.

Gendreau, M. (2003). An introduction to tabu search. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, 37-54 2. Kluwer Academic Publishers.

Gendreau, M.; Hertz, A. e Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *INFORMSJC*, v. 40, p. 1276 – 1290.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, v. 13, n. 5, p. 533–549. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/0305-0548(86)90048-1.

Glover, F. (1996). *Tabu Search and adaptive memory programming - advances, applications and challenges*, p. 1–75. Kluwer Academic Publishers, interfaces in computer science and operations research edição.

Glover, F. e Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Glover, F. e Laguna, M. (2003). Scatter search and path relinking: Advances and applications. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, Capítulo 1, p. 1–35. Kluwer Academic Publishers.

Guignard, M. e Rosenwein, M. (1989). An improved dual-based algorithm for the generalized assignment problem. *Operations Research*, v. 37, p. 658–663.

Hansen, P. e Mladenovirć, N. (2003). Variable neighborhood search. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, 145-184 6. Kluwer Academic Publishers.

Hung, M. S. e Fisk, J. C. (1979). A heuristic routine for solving large loading problems. *Naval Research Logistic Quarterly*, v. 26, p. 643–650.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, v. 220, p. 671–680.

Laguna, M.; Kelly, J. P.; Gonzalez-Velarde, J. I. e Glover, F. (1995). Tabu search for the multilevel generalized assignment problem. *European Journal of Operations Research*, v. 42, p. 677–687.

Lenstra, J. K.; Shmoys, D. B. e Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming: Series A and B*, v. 46, p. 259–271.

Martello, S. e Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. *Wiley,Chichester*.

Mazzola, J. B.; Neebe, A. W. e Dunn, C. V. R. (1989). Production planning of a flexible manufacturing system in a material requirements planning environment. *International Journal of Flexible Manufacturing Systems*, v. 1, p. 115–142.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, n. 11, p. 1097–1100.

Montgomery, D. C. e Runger, G. C. (2003). *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Inc, New York, USA.

Narciso, M. G. e Lorena, L. A. N. (1999). Lagrangean / surrogate relaxation for generalized assignment problems. *European Journal of Operational Research*, v. 114, p. 165–177.

Osman, I. H. (1995). Heuristics for the generalized assignment problem: Simulated Annealing and Tabu Search approaches. *OR Spektrum*, v. 17, p. 211–225.

Rego, C. e Roucairol, C. (1996). Meta-heuristics theory and applications. *Kluwer Academic Publisher, Boston*, p. 661 – 675.

Ribeiro, C. C. (1996). Metaheuristics and applications. In *Advanced School on Artificial Intelligence*, Estoril, Portugal.

Ross, G. T. e Soland, R. M. (1977). Modeling facility location problems as generalized assignment problems. *Management Science*, v. 24, p. 345–357.

Ross, S. M. (2004). *Introduction to Probability and Statistics for Engineers and Scientists*. Elselvier, California, USA.

Sahni, S. e Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM*, v. 23, p. 555–565.

Schaerf, A. (1996). Tabu search techniques for large high-school timetabling problems. *AAAI-96 Proceedings*, p. 363–368, Amsterdam.

Souza, M. J. F. (2009). Inteligência computacional para otimização. Notas de aula. Disponível em `http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm`. Acesso em: 10 jan. 2009.

Srinivas, N. e Deb, K. (1995). Multiobjective function optmization using nondominated sorting genetic algorithms. *Evolutionary Computation*, v. 2, p. 221–248.

Turek, J.; Wolf, J. L. e Yu, P. S. (1992). Approximate algorithms scheduling parallelizable tasks. *Proceedings of ACM Symposium on Parallel Algorithms and Architectures*, p. 323–332, San Diego, USA.

Woodcock, A. J. e Wilson, J. M. (2010). A hybrid tabu search/branch & bound approach to solving the generalized assignment problem. *European Journal of Operational Research*, v. 207, p. 566–578.

Yagiura, M.; Glover, F. e Ibaraki, T. (2006). A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, v. 169, p. 548–569.

Yagiura, M.; Ibaraki, T. e Glover, F. (1999)a. An ejection chain approach for the generalised assignment problem. *Technical Report 99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University.*

Yagiura, M.; Yamaguchi, T. e Ibaraki, T. (1998). A variable depth search algorithm with branching search for the generalised assignment problem. *Optimisation Methods & Software*, v. 10, p. 419–441.

Yagiura, M.; Yamaguchi, T. e Ibaraki, T. (1999)b. A variable depth search algorithm for the generalised assignment problem. *Metha-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers*, p. 459–471.