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

ARQUITETURA HÍBRIDA MULTIAGENTE AMAM APLICADA AO PROBLEMA *P-HUB* CENTRO

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

Aluno : Jardell Fillipe da Silva

Orientador : Prof. Dr. Sérgio Ricardo de Souza (CEFET-MG)

Co-Orientadora : Profa. Dra. Maria Amélia Lopes Silva (UFV)

Belo Horizonte - MG
Dezembro de 2020

Silva, Jardell Fillipe da
S586a Arquitetura híbrida multiagente AMAM aplicada ao problema p-Hub Centro / Jardell Fillipe da Silva. – 2020.
xi, 87 f.

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Sérgio Ricardo de Souza.
Coorientadora: Maria Amélia Lopes Silva.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Inteligência artificial – Teses. 2. Agentes inteligentes (Software) – Teses. 3. Programação heurística – Teses. 4. Sistemas inteligentes de controle – Teses. 5. Otimização combinatória – Teses. I. Souza, Sérgio Ricardo. II. Silva, Maria Amélia Lopes. III. Centro Federal de Educação Tecnológica de Minas Gerais.
IV. Título.

CDD 519.6

Elaboração da ficha catalográfica pela bibliotecária Jane Marangon Duarte, CRB 6º 1592 / Cefet/MG

Aos meus pais, Geraldo e Maria Zélia, pelo exemplo e apoio incondicional.

Agradecimentos

Agradecimento: “reconhecimento e declaração de se estar grato por algo dado ou outrem”. Sendo assim, todo acontecimento requer agradecimentos.

Agradeço a minha esposa, Juliana, pelo carinho, paciência e compreensão pelo tempo ausente. Para minha filha, Anne, e ao meu filho Raul, pela razão de batalhar todos os dias. Aos meus pais, pelo apoio incondicional.

Aos professores Sérgio Ricardo de Souza e Maria Amélia Lopes Silva, pelo orientação, dedicação e conhecimento transmitido. Aos professores Marcone Jamilson Freitas Souza, Flávio Vinícius Cruzeiro Martins, Elisangela Martins de Sá e todos professores do PPGMMC CEFET-MG pelo conhecimento e auxílio no decorrer das disciplinas.

A todos meus familiares e amigos, por sempre me apoiarem.

Agradeço a DEUS e a todos que, de certa forma, auxiliaram diretamente ou indiretamente nesta etapa. Obrigado a todos vocês.


PODE PARECER POUCO, MAS DE TIJOLINHO EM TIJOLINHO...
...PODEMOS CONSTRUIR OS MAIORES MUROS!
MUROS NÃO!
PODEMOS CONSTRUIR PONTES!
becidiustras@gmail.com

Resumo

Esta Dissertação estuda duas variantes do Problema *pHub* Centro (pHCP), denominadas Problema *pHub* Centro não Capacitado de Múltiplas Alocações (UMApHCP) e o Problema *pHub* Centro Capacitado de Múltiplas Alocações (CMApHCP). Inicialmente, a descrição destas variantes é apresentada, incluindo suas caracterizações, modelos matemáticos e uma revisão bibliográfica do estado da arte associado. Em seguida, introduz-se a fundamentação teórica necessária para o desenvolvimento dessa dissertação. Como se trata de problemas NP-Difíceis, esta dissertação utilizou-se, além de modelos de programação matemática, de técnicas heurísticas e metaheurísticas. A implementação destas últimas é realizada usando-se um estrutura híbrida multiagente denominada *Framework* `AMAM`. O *Framework* `AMAM` possui capacidade de hibridização de metaheurísticas através da abordagem multiagente. O espaço de busca de cada agente deste é o próprio espaço de busca do problema, possibilitando flexibilidade para tratar problemas distintos. A ação do agente no espaço de busca é autônoma e permite a execução simultânea de vários agentes, de forma cooperativa. Esta dissertação propõe três propostas de resolução, implementadas no *Framework* `AMAM`, para a solução dos problemas abordados, além da solução via programação matemática. A primeira proposta, para a resolução do UMApHCP, consiste em um método híbrido, que combina a fase de construção da metaheurística GRASP com um Algoritmo Genético. A segunda proposta, também para a resolução do UMApHCP, substitui, na primeira proposta, o algoritmo genético pela metaheurística ILS. A terceira proposta, para a solução do problema CMApHCP, utiliza a metaheurística VNS e a heurística de busca local VND, combinada com a fase de construção da metaheurística GRASP. Estas propostas foram avaliadas por meio de testes computacionais em conjuntos de instâncias disponíveis na literatura. Os resultados obtidos foram satisfatórios e comprovam a eficiência do *Framework* `AMAM`, além de mostrar que a utilização de sistemas multiagente é eficaz na resolução de problemas de otimização.

Palavras-chaves: Problemas de Localização de *Hubs*. *p-Hub* Centro. Metaheurísticas. Sistemas Multiagentes. *Framework* para Otimização. Otimização Combinatória.

Abstract

This dissertation addresses two variants of the pHub Center Problem (pHCP), called Uncapacitated Multiple Allocation pHub Center Problem (UMApHCP) and Capacitated Multiple Allocation pHub Center Problem (CMApHCP). First, the description of these variants is presented, including their characterizations, mathematical models, and a bibliographic review concerning the associated state-of-the-art. Following, the necessary theoretical foundation for the development of this work is revised. As these are NP-Hard problems, in addition to mathematical programming models, this work uses heuristic and metaheuristic techniques for solving them. The implementation of the heuristic and metaheuristics used a hybrid multi-agent structure called the AMAM Framework. The AMAM Framework can hybridize metaheuristics through the multi-agent approach. The search space for each agent in this framework is the problem search space, allowing flexibility to deal with different problems. The agents in the search space are autonomous, allowing the simultaneous and cooperative execution of several agents. This work proposes three resolution proposals, implemented in the AMAM Framework, for the solution of the addressed problems, in addition to the solution via mathematical programming. The first proposal, used for solving the UMApHCP problem, was a hybrid method, which combined the construction phase of the GRASP metaheuristic with a genetic algorithm. The second proposal replaces the genetic algorithm with the ILS metaheuristic in the first proposal. The third proposal, for solving the CMApHCP problem, also a hybrid method, combined the construction phase of the GRASP metaheuristic with the VNS metaheuristic and the VND local search heuristic. These proposals were evaluated using computational tests on instance sets available in the literature. The results obtained were satisfactory and proved the efficiency of the AMAM Framework, and, besides, showed that the use of multi-agent systems is effective in solving optimization problems.

Keywords: Hub Location Problem. p-Hub Center Problem. Metaheuristics. Multi-agent Systems. Optimization Frameworks. Combinatorial Optimization.

Sumário

Lista de Tabelas

Lista de Figuras

List of Algoritmos

Capítulo 1

Introdução

Sistemas com grande demanda de fluxo origem-destino entre todos os seus nós necessitam de uma boa programação logística para que todo o fluxo seja atendido da forma mais eficiente. Partindo de um sistema no qual todos os nós estão conectados e possuem demanda entre si, uma topologia ponto-a-ponto pode ser ineficiente e custosa (Farahani et al., 2013). Topologias eixo-raio são mais eficientes e geram economia tanto em razão financeira quanto em número de conexões (Kara e Tansel, 2001). Essas topologias caracterizam-se pela existência de centros de conexão (*hubs*), que podem ser utilizados para triagem, distribuição e transbordo de mercadorias. Em um sistema constituído de um grande número de nós, definir centros de distribuição e o fluxo de operação é uma tarefa onerosa e, desta necessidade, surge a proposição do Problema de Localização de *Hubs*. Farahani et al. (2013) apresentam este problema como uma nova e próspera área na teoria de localização. Alumur e Kara (2008), Farahani et al. (2013) e Hsieh e Kao (2019) apresentam o estado da arte, modelos, classificação, técnicas de solução e aplicações das variantes dos problemas de localização de *hubs*.

Uma variante do problema de localização de *hubs* consiste em localizar p *hubs* em uma topologia eixo-raio, de forma que p é predeterminado no problema. Usualmente, a forma de avaliar este problema se dá pela minimização do somatório do custo total em percorrer todo o sistema de nó a nó. Neste contexto, esta avaliação é feita pela minimização do custo total incorrido na localização de nós *hubs* e na alocação de nós que não são *hubs* para nós *hubs*, através do critério min-$\sum$, que consiste em minimizar a somatória de todo o percurso incorrido no grafo. Estes problemas visam, principalmente, o fator econômico, porém, podem gerar atendimentos discrepantes. Casos reais podem necessitar de atendimentos mais homogêneos, principalmente, em relação ao tempo de transporte. Com isso, pode-se utilizar, nesses sistemas, uma avaliação utilizando o critério min-max. Este critério consiste em minimizar o custo máximo em incorrer todas as demandas do grafo. Estas características podem ser encontradas em sistemas de urgência/emergência e sistemas de entrega de alimentos perecíveis, que têm, como característica intrínseca, a limitação do tempo de coleta/entrega. Nesses casos práticos, busca-se, além de minimizar o custo máximo de percurso, maior eficiência para problemas que exigem rapidez na coleta/entrega. Este problema é denominado como Problema *pHub* Centro (*pHub Center Problem*). Esta variante é uma das mais importantes do problema de localização de *hubs*.

Ernst et al. (2009) apresentam variantes do pHCP e comprovam sua complexidade NP-Difícil. Desta forma, a utilização de heurísticas e metaheurísticas se torna uma

boa estratégia na solução destes problemas. Metaheurísticas são procedimentos que permitem que a busca por soluções não fique presa em ótimos locais. Porém, estes métodos não garantem a otimalidade dos resultados, nem o quão distante estão deles. Boas revisões sobre metaheurísticas podem ser encontradas em Stützle (1998); Blum e Roli (2003); Blum et al. (2005); Gaspar-Cunha et al. (2012); Silva (2019).

A literatura recente mostra que, para a solução de problemas de otimização combinatória, a combinação de metaheurísticas (metaheurísticas híbridas) tem produzido os melhores resultados encontrados para a solução de problemas diversos (Blum et al., 2011). Metaheurísticas híbridas são métodos que combinam metaheurísticas, ou partes delas, com outras metaheurísticas ou até mesmo com outros métodos, como, por exemplo, métodos exatos. Esta técnica tem, como objetivo, beneficiar-se das melhores estratégias de cada método, utilizando-as em conjunto. Exemplos de metaheurísticas híbridas e sua história podem ser encontrados em Cotta et al. (2005) e Blum et al. (2011).

Atualmente, a demanda crescente por métodos mais flexíveis, capazes de solucionar diferentes classes de problemas sem grandes modificações, e por códigos reutilizáveis, que diminuam o tempo de desenvolvimento de soluções para problemas de otimização distintos, tem levado os pesquisadores ao desenvolvimento de *frameworks*.

Frameworks são ferramentas que estruturam e facilitam a implementação de códigos. Estas ferramentas agrupam códigos genéricos e reutilizáveis, sendo, desta forma, eficazes na implementação de métodos de solução. A utilização de *Frameworks* para a solução de problemas de otimização tem se expandido cada vez mais entre pesquisadores da área. *Frameworks* para otimização são ferramentas que facilitam a implementação de metaheurísticas e, até mesmo, em alguns casos, de metaheurísticas híbridas, e permitem a utilização de metaheurísticas pré-implementadas, reuso de códigos e apoio no processo de decisão. Diversos *frameworks* para otimização são encontrados na literatura recente. Silva et al. (2018) apresenta uma ampla revisão dos principais *frameworks* para otimização utilizando metaheurísticas disponíveis e suas aplicações.

Assim, esta dissertação propõe solucionar duas variantes do problema *pHub* Centro, utilizando-se de uma estrutura metaheurística multiagente (*framework*). Para isto, modelos matemáticos, para as variantes, são apresentados e inicialmente solucionados de forma exata. A seguir, métodos heurísticos e metaheurísticos são desenvolvidos e incorporados ao *framework*. Por fim, testes computacionais são apresentados, para avaliar a qualidade do desenvolvimento realizado.

1.1 Justificativa

A utilização de topologias eixo-raio gera economia em sistemas que contém demanda entre todos os pares de origem-destino. Além do campo estritamente financeiro, o Problema *pHub* Centro é uma variante importante da teoria de localização de *hubs*, tendo em vista sua ampla gama de aplicações em problemas em que o atendimento necessita de uma maior rapidez na coleta/entrega. Como exemplo, destaca-se a localização de sistemas urgência e emergência, transporte de alimentos perecíveis e transportes sujeitos a limite de tempo, dentre outros. Problemas de localização com critério de avaliação min-max são mais justos em relação à alocação de clientes

aos *hubs* e minimizam o custo máximo do sistema. Já a utilização de *frameworks* para metaheurísticas na resolução de problemas de otimização combinatória é uma alternativa de forte interesse, devido a sua facilidade de implementação e organização das estruturas dos códigos de programação.

1.2 Objetivos

1.2.1 Objetivos Gerais

A presente dissertação tem, como objetivo geral, solucionar duas variantes do Problema de Localização de *Hubs*: (i) Problema *pHub* Centro Não Capacitado de Múltiplas Alocações (*Uncapacitated Multiple Allocation pHub Center Problem - UMApHCP*); e (ii) Problema *pHub* Centro Capacitado de Múltiplas Alocações (*Capacitated Multiple Allocation pHub Center Problem - CMApHCP*), utilizando o *Framework* `AMAM` (Arquitetura Multi-Agente para Metaheurísticas) e suas ferramentas de aprendizagem e cooperação multi-agente. Desta forma, pretende-se alcançar resultados computacionais competitivos na resolução dessas variantes e demonstrar os benefícios do uso do *Framework* `AMAM` na solução de problemas distintos.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral desta dissertação, faz se necessário atingir os seguintes objetivos específicos:

(i) Conhecer a literatura concernente aos principais temas envolvidos neste projeto de dissertação, os quais são: Problema de Localização de *Hubs*, Problema *pHub* Centro, Metaheurísticas, Metaheurísticas Híbridas, Sistemas Multiagentes, Processamento Paralelo, Aprendizado de Máquina e *Frameworks* para Otimização usando Metaheurísticas;

(ii) Modelar o problema *pHub* Centro, mais especificamente, as variantes não capacitada e capacitada de múltiplas alocações;

(iii) Estudar o *Framework* `AMAM`, suas características e técnicas de solução;

(iv) Modelar computacionalmente e implementar os dois problemas a serem solucionados, no *Framework* `AMAM`;

(v) Implementar as metaheurísticas escolhidas, no *Framework* `AMAM`;

(vi) Efetuar testes computacionais;

(vii) Analisar estatisticamente e discutir os resultados obtidos.

1.3 Estrutura da Dissertação

Esta dissertação tem a seguinte estrutura: após a introdução a respeito do problema objeto de interesse e da proposta contida nesta dissertação, mostrada no Capítulo 1, apresenta-se, no Capítulo 2, a descrição do Problema de Localização de

Hubs (HLP) e do Problema *pHub* Centro (*p*HCP), bem como duas variantes destes problemas. Em seguida, o Capítulo 3 revisa a fundamentação teórica para o desenvolvimento dessa dissertação, em que se discute, nos termos aqui necessários, os conceitos de metaheurísticas, metaheurísticas híbridas e *frameworks* para otimização combinatória. No tocante a este último tópico, também são abordadas, nos termos próprios a esta dissertação, algumas de suas questões fundamentais, como Sistemas Multiagentes, Aprendizado de Máquinas e *Frameworks* para Otimização. Posteriormente, no Capítulo 4, nas Seções 4.1 e 4.2, é apresentado um Algoritmo Genético e um agente ILS_AMAM aplicado ao problema UMA*p*HCP, respectivamente. Logo em seguida, no Capítulo 5, experimentos aplicados e resultados encontrados com os métodos de solução para o problema UMA*p*HCP são explanados. No Capítulo 6, um agente VNS implementado no *Framework* `AMAM` é apresentado para solucionar o problema CMA*p*HCP. Os resultados obtidos com esse método são apresentados no Capítulo 7. O Capítulo 8 apresenta as considerações finais, lista as publicações derivadas desta dissertação e indica direções futuras de pesquisa.

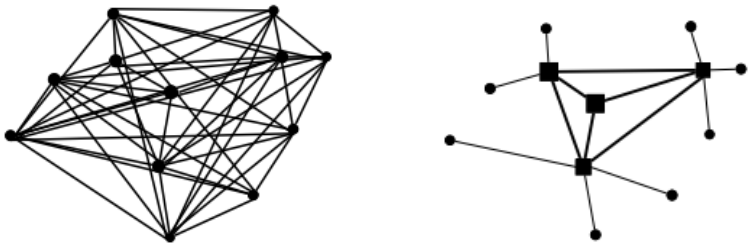
Capítulo 2

Descrição do Problema

Este capítulo descreve o Problema de Localização de *hubs* e as variantes deste, que são o foco deste trabalho. Inicialmente, na Seção 2.1, o Problema de Localização de *Hubs* é apresentado. Em seguida, na Seção 2.2, são descritos o Problema *pHub* Centro e as duas variantes de interesse (Seções 2.2.1 e 2.2.2). A Seção 2.3 apresenta as considerações do capítulo.

2.1 Problema de Localização de *Hubs*

Em um sistema que dispõe de fluxo entre todos os nós de origem-destino, uma topologia mais eficiente para essa rede se dá através de um sistema eixo-raio (*hub-and-spoke network*). Este tipo de rede é caracterizada pela existência de links intermediários, que fazem com que o número de links diretos entre os nós de origem-destino sejam otimizados. Por exemplo, uma rede de n nós com conexões diretas possui $n(n-1)$ links; já em um uma rede com 1 *hub*, o número de conexões diminui para apenas $2(n-1)$ links (Farahani et al., 2013).



(a) Ponto a Ponto (b) Eixo-Raio

Figura 2.1: Conexão entre pares de origem-destino (de Sá et al., 2010)

A Figura 2.1(a), mostrada em de Sá et al. (2010), apresenta um sistema de nós, com pares de demanda origem-destino e links diretos entre todos os nós. Já a Figura 2.1(b) apresenta o mesmo sistema com uma topologia eixo-raio, que possui links intermediários entre as demandas de origem-destino. Definir nós intermediários e a alocação dos clientes a eles é um problema denominado Problema de Localização de

Hub (*hub Location Problem - HLP*), no qual *hubs* são considerados pontos de roteamento, concentração, distribuição e/ou triagem de fluxo, para gerar economicidade do sistema. Entretanto, definir quais nós serão *hubs* e quais clientes alocar a estes nós não é uma tarefa trivial. Apesar de ser um conceito originado de sistemas de telecomunicação, *hubs* são frequentemente utilizados em sistemas logísticos, transportes, aviação civil, remessas expressas, serviços postais, dentre outros. Para que se justifique a existência do HLP, é necessário o movimento de pessoas, mercadorias e ou informações entre os pares de demanda origem-destino. O HLP é uma nova e próspera área de pesquisa em teoria da localização (Farahani et al., 2013). Boas revisões de problemas de localização de *hubs* podem ser encontradas em Alumur e Kara (2008); Farahani et al. (2013); Hsieh e Kao (2019), nas quais são apresentados o estado da arte, modelos, classificação, técnicas de solução e aplicações. O problema de localização de *hubs* possui diversas variantes, classificadas pelas seguintes propriedades:

- **Domínio da solução:** propriedade que define quais nós são candidatos a *hubs*:
  - (i) Rede - todos os nós da rede são candidatos a serem *hubs*:
  - (ii) Discreto - uma conjunto de nós específicos são candidatos a serem *hubs*:
  - (iii) Contínuo - os nós candidatos a serem *hubs* formam um contínuo.
- **Função objetivo:** propriedade que define a forma de avaliação do problema:
  - (i) Critério min-$\sum$ - o valor da função objetivo é o custo total incorrido pelo transporte de toda a rede:
  - (ii) Critério min-max - o valor da função objetivo é o custo máximo de transporte entre todas a demandas de origem-destino.
- **Origem do número de *hubs*:** propriedade que indica como o número de *hubs* é definido no problema:
  - (i) Exógena - o número de *hubs* a ser localizado é pré-determinado pelo problema:
  - (ii) Endógeno - o número de *hubs* a ser localizado não é pré-definido, mas sim extraído como parte da solução.
- **Número de *hubs*:** propriedade que demarca o número de *hubs*:
  - (i) *hub* único - um único *hub* é localizado:
  - (ii) Vários *hubs* - dois ou mais *hubs* são localizados.
- **Capacidade do *hub*:** propriedade que define limite de alocação aos *hubs*
  - (i) Não-capacitado - a capacidade dos nós *hubs* é ilimitada:
  - (ii) Capacitado - a capacidade dos nós *hubs* é limitada.
- **Custo de localização do *hub*:** propriedade que determina o custo de localização dos *hubs*:

(i) Sem custo - a localização dos *hubs* não é taxada:

(ii) Custo fixo - a localização dos *hubs* são taxadas igualmente:

(iii) Custo variável - cada *hub* possui uma taxa de instalação.

- **Alocação:** propriedade que define a forma de alocação dos nós não *hubs* aos nós *hubs*:

  (i) Simples alocação - os nós não *hubs* alocam-se a apenas um único *hub*;

  (ii) Múltipla alocação - os nós não *hubs* alocam-se a pelo menos 1 nó *hub*.

- **Custo de conexão com o *hub*:** propriedade que determina o custo de conexão dos nós não *hubs* a nós *hubs*:

  (i) Sem custo - não há custo entre a conexão;

  (ii) Custo fixo - o custo de conexão é igual para todos;

  (iii) Custo variável - o custo é distinto entre os nós e os *hubs*.

Os principais modelos de localização de hubs são:

- 1*Hub*;
- *pHub*;
- *pHub* Mediana;
- *pHub* com Custo Fixo de Link;
- *pHub* com Fluxo de Valor Mínimo;
- *pHub* com Capacidade Limitada;
- *pHub* Contínuo;
- *pHub* Multi-Objetivo;
- *pHub* Centro;
- *pHub* de Cobertura;
- *pHub* de Cobertura Máxima;
- *pHub* com Rede em Estrela.

Pesquisas com conceitos similares ao HLP foram desenvolvidas inicialmente por Hakimi (1964) e Toh e Higgins (1985), com estudos na área de otimização de nós e localização de hubs na aviação civil, respectivamente. Pesquisas relacionadas a localização de *hubs* foram introduzidos por O'kelly (1986) e O'Kelly (1986), apresentando formulações e modelos de solução para o HLP. O'kelly (1987) propõe uma formulação matemática quadrática e discute o problema de alocação única. A partir daí, muitas pesquisas sobre HLP foram publicadas, das quais algumas de destaque são apresentadas em Klincewicz (1991), Klincewicz (1992), Aykin (1994),Aykin (1995b), Aykin (1995a), Skorin-Kapov et al. (1996) e Ernst e Krishnamoorthy (1998). Estes trabalhos tratam do HLP de alocação única. Em Campbell (1994), a primeira formulação com características de alocações múltiplas de hubs é introduzida. Restrições de capacidade são incluídas nos problemas a partir de Ernst e Krishnamoorthy (1999).

Farahani et al. (2013) apresentam uma revisão detalhada em relação aos problemas de localização de *hubs*, de forma que modelos, características e pesquisas correlatas são exploradas. Para cada artigo analisado, os autores apresentam o método de resolução, números de eficiência de nós, assim como estudos de casos reais e a inferência feita em relação ao futuro da pesquisa. A presente revisão de literatura

se baseia em Farahani et al. (2013) e, adicionalmente, identifica projetos correlatos, para identificar as direções da pesquisa relacionadas à teoria de localização de *hubs*.

A partir da análise dos artigos apresentados em Farahani et al. (2013), é possível observar que modelos nos quais a forma de avaliação da solução é o critério min-max são poucos investigados. Em Farahani et al. (2013), apenas 11% das pesquisas que utilizam métodos exatos solucionam problemas deste tipo. Já pesquisas providas de métodos heurísticos ou metaheurísticos somam apenas 8% das pesquisas. Em geral, 9% de todas as pesquisas apresentadas em Farahani et al. (2013) abordam este tipo de problema. Já Zabihi e Gharakhani (2018) apresentam artigos correlatos sobre localização de *hubs* aplicados a sistemas marítimos. Para isso, realizam uma revisão da literatura a respeito de problemas de localização de *hubs*, na qual apresentam as direções das pesquisas correlatas nos anos de 2013 à 2015. Os resultados encontrados apontam:

- que o ano de 2015 corresponde a 50% das publicações entre esses anos;
- e apenas 13% das publicações discorre sobre problemas de localização de centro de *hubs*.

Pode-se, assim, observar um crescimento das pesquisas, nos últimos anos, que se concentram em solucionar problemas de localização de centro de *hubs*. Isso se deve, principalmente, a sua aplicação em problemas de localização de centro de urgência/emergência e sistemas restritos ao tempo de entrega/percurso. Assim sendo, esta dissertação foca em apresentar uma variante do problema de localização de *hubs*, denominada Problema *pHub* Centro (*pHub Center Problem* - *p*HCP), que será melhor descrita na seção seguinte.

2.2 Problema *pHub* Centro

Uma das mais importantes variações do problema de localização de *hubs* é o Problema *pHub* Centro (*pHub Center Problem* - *pHCP*). Este problema consiste em minimizar o custo máximo entre os pares origem-destino (critério min-max) em um sistema com demanda origem-destino entre todos os nós. O *p*HCP é uma variante que se destaca pela sua aplicação em problemas de localização de centros de urgência/emergência, transportes de produtos perecíveis, transportes em que motoristas estão restritos a um limite de tempo de serviço, por exemplo. O *p*HCP foi proposto em Campbell (1994), no qual são apresentadas as formulações de otimização linear inteira de quatro problemas de localização de *hubs* discreta:

(i) Problema da Mediana de *pHub* (*pHub Median Problem* - *pHMP*);

(ii) Problema de Localização de *hub* Não Capacitado (*Uncapacitated pHub Location Problem* - *UpHLP*);

(iii) Problema de Cobertura de *pHub* (*pHub Covering Problem* - *pHCP*);

(iv) Problema *pHub* Centro de Simples Alocação (*Single Allocation pHub Center Problem* - *SApHCP*).

Um novo modelo para o SApHCP foi proposto por Kara e Tansel (2000) e comparado com o modelo básico apresentado anteriormente em Campbell (1994). Kara e Tansel (2000) apresentam estudos sobre aspectos computacionais e sua complexidade (NP-Difícil) e demonstram que o modelo por eles proposto é melhor em tempo de execução e em requisitos básicos de armazenamento. Nota-se em Campbell et al. (2007) uma melhoria nas formulações, além de introduzir o problema de alocação do pHCP como um subproblema do pHCP. Resultados de complexidade e formulações para variantes do problema são apresentados. Determina-se que, em alguns casos especiais, alocar clientes no pHCP é um problema polinomialmente solúvel. Como exemplo, alocar clientes ao USA$_2$HCP, USApHCP sendo $\alpha = 0$, UMApHCP, CMApHCP quando $0 \leq x_{ijkm} \leq 1$. Porém, definir quais nós serão *hubs* e alocar clientes a eles é NP-Completo.

Em Ernst et al. (2009), novas formulações matemáticas são apresentadas e uma nova formulação de otimização inteira mista é proposta para o problema de alocação simples, além de duas novas formulações de otimização inteira e uma abordagem *Branch-and-Bound*, propostas para resolver o problema de múltiplas alocações. Testes computacionais comprovam a ineficiência das modelagens com 3 e 4 índices para ambos os problemas, além de demonstrarem sua complexidade computacional. Um algoritmo de resolução em duas fases para o problema de alocação única não capacitado é apresentado por Meyer et al. (2009). Na primeira fase, é construído um conjunto de combinações possíveis de *hubs* ótimos, usando *Branch-and-Bound* e um algoritmo de caminho mínimo. A segunda fase consiste em resolver o problema de alocação com uma formulação de tamanho reduzido. Adicionalmente, o algoritmo ACO (*Ant Colony Optimization* – Otimização de Colônia de Formigas) é implementado, para encontrar um bom limite superior. Os testes computacionais descritos comprovam a eficiência em instâncias com até 400 nós, atingindo seus resultados ótimos em tempo de execução razoável.

Problemas de *pHub* Centro com características estocásticas são abordados em Sim et al. (2009) e Yang et al. (2011). No primeiro, são tratados problemas de entrega de encomendas expressas com tempo de viagem variável. Resultados analíticos são discutidos e uma heurística baseada em busca local é construída para solucionar o problema de alocação única. Yang et al. (2011) apresentam um modelo de otimização linear inteira mista para resolução do pHCP, em que o tempo de viagem entre os pontos é considerado aleatório, além do mesmo possuir predições de acontecimento de viagens. Um método *Branch-and-Bound* foi construído para solucionar o problema. Yaman e Elloumi (2012) também propõem duas novas formulações para o problema de localização do *pHub* Centro, de modo a minimizar o comprimento do caminho de origem-destino mais longo, utilizando-se de um algoritmo baseado em Relaxação Lagrangeana e Busca Local para resolução do problema.

Brimberg et al. (2017b) propõem uma heurística para resolução do Problema de Alocação Simples não Capacitado (USApHCP) baseado no método de Busca em Vizinhança Variável Geral (*General Variable Neighbourhood Search - GVNS*). O algoritmo GVNS possui, como ferramenta de busca local, a estratégia de Descida em Vizinhança Variável (*Variable Neighbourhood Descent - VND*). Brimberg et al. (2017b) adotam duas estruturas de vizinhança para a busca no ambiente de soluções. Testes computacionais foram realizados para verificar a eficiência do método e os resultados provam sua eficiência em relação aos métodos anteriores. Deve-se ressal-

tar que o GVNS alcançou todas as soluções ótimas já conhecidas e, além disso, foi testado em instâncias de maiores dimensões não testadas anteriormente. Brimberg et al. (2017a) apresentam uma metaheurística de Busca em Vizinhança Variável Básica (*Basic Variable Neighbourhood Search - BVNS*) para resolução do Problema de Alocação Múltipla não Capacitado do *pHub* Centro (UMA*p*HCP). Testes computacionais são realizados com as instâncias de referência para o problema. Dois modelos matemáticos de 3 e 4 índices e uma heurística *Multi-Start* foi desenvolvida como base para os testes. Após toda a configuração de *hubs*, gerada pelo BVNS, é dada a alocação do *pHub* Centro através de uma adaptação do algoritmo de caminho mínimo de Floyd-Warshall.

Tabela 2.1: Artigos tratando do Problema p*Hub* Centro

Artigo	Alocação	Capacitado	NÂº Nós	Método de Solução
Ernst et al. (2009)	SA/MA	Não	200	Exato
Kara e Tansel (2000)	MA	Não	25	Exato
Yaman e Elloumi (2012)	SA	Não	70	Exato
Brimberg et al. (2017a)	MA	Não	40	Exato
Brimberg et al. (2017b)	SA	Não	200	Exato
Ernst et al. (2009)	SA/MA	Não	200	Heurístico
Calik et al. (2009)	SA	Não	81	Heurístico
Gavriliouk (2009)	SA	Não	1000	Heurístico
Meyer et al. (2009)	SA	Não	1000	Heurístico
Sim et al. (2009)	SA	Não	40	Heurístico
Bashiri et al. (2013)	SA	Sim	25	Heurístico
Brimberg et al. (2017a)	MA	Não	1000	Heurístico
Brimberg et al. (2017b)	SA	Não	1000	Heurístico

A Tabela 2.1 relaciona os artigos que tratam o problema p*Hub* Centro aqui descritos. Nela, estão contidos artigos que abrangem o problema p*Hub* Centro, e as características nele contido. Na Tabela 2.1, a coluna "Alocação", mostra a forma de alocação dos nós, na forma "SA" (simples alocação) ou "MA" (múltiplas alocações). A coluna "Capacitado" informa se o problema possui alguma restrição de capacidade. A coluna "NÂº Nós" informa o número máximo de nós contidos nas instâncias tratadas pelo artigo; por fim, a coluna "Método de Solução" mostra o tipo de método utilizado no desenrolar do estudo, indicando se é um método exato ou heurístico.

Em uma simples análise da Tabela 2.1, verifica-se que problemas p*Hub* Centro que possuem características de múltiplas alocações (MA) são menos tratados que os problemas de simples alocação (SA). Além disso, pode-se observar que problemas que possuem restrições de capacidade também são menos abordados. Podemos constatar, também, que, nas pesquisas com restrições de capacidade, as instâncias dos problemas possuem um menor número de nós. Dentro deste contexto, a presente dissertação direciona seus esforços para solucionar três variantes do Problema *p-hub* Centro. A primeira variante tratada é o Problema *pHub* Centro não Capacitado de Múltiplas Alocações (*Uncapacitated Multiple Allocation pHub Center Problem - UMApHCP*). Em seguida, são tratadas duas formulações do Problema *pHub* Centro Capacitado de Múltiplas Alocações (*Capacitated Multiple Allocation pHub Center*

Problem - CMApHCP), que se diferem pela forma na qual a capacidade do nó *hub* é atribuída. Estas formulações serão melhor descritas nas seções seguintes.

2.2.1 UMApHCP

O UMApHCP consiste em definir quais nós serão *hubs* e quais clientes serão alocados a eles, de forma que o custo máximo de transporte entre todos os pares origem-destino seja minimizado. Os nós não *hubs* não possuem restrições de alocação (múltipla alocação) e os nós *hubs* não possuem restrições de capacidade (não capacitado). O UMApHCP é descrito da seguinte forma: dado um grafo $E = (N, A)$, em que $N = \{1, 2, \ldots, i, \ldots, n\}$ representa o conjunto de nós e $A = \{(i, j) \mid i, j \in N\}$ representa o conjunto de arcos, defina o conjunto H de nós *hub*, de modo que $H \subset N$ e $|H| = p$. Assim, todo nó $i \in N$ é um candidato a entrar no conjunto H; todo nó $i \in N$ é alocado a pelo menos um *hub* pertencente ao conjunto H; todo nó $i \in N$ não *hub* pode se conectar a qualquer *hub* em H; os *hubs* em H não apresentam restrições de capacidade; o transporte direto entre nós não *hub* não é permitido. O custo de transporte d_{ij} é calculado por:

$$d_{ij} = \gamma c_{ih_i} + \alpha c_{h_i h_j} + \beta c_{h_j j} \tag{2.1}$$

de forma que γ, α e β são descontos/penalidades de transporte atribuídos aos *hubs*, conforme Ernst et al. (2009). Na Expressão (2.1), c_{ih_i} representa o custo de coleta entre o nó origem i e o *hub* h_i; $c_{h_i h_j}$ representa o custo de transferência entre o *hub* h_i e o *hub* h_j; e $c_{h_j j}$ representa o custo de distribuição entre o *hub* h_j e o nó destino j. Considerando-se que, para efetivar o arco (i, i), quando i é um nó não *hub*, temos que necessariamente passar por um nó *hub*, assume-se, então, que $c_{ii} \geq 0$. O objetivo do problema é minimizar o custo máximo entre os pares origem-destino do grafo. Assim, seja s uma variável de decisão contínua, que representa o valor da função objetivo; z_k uma variável de decisão binária, de modo que $z_k = 1$ se o nó k é definido como *hub* e $z_k = 0$, caso contrário; e y_{ijkl} uma variável de decisão binária de quatro índices, tal que $y_{ijkl} = 1$ se a demanda do arco (i, j) passa pelo arco formado pelos *hubs* (k, l), e $y_{ijkl} = 0$, caso contrário. O modelo matemático do UMApHCP, conforme proposto por Ernst et al. (2009), é dado por:

$$\begin{aligned}
\min \quad & s && (2.2)\\
\text{sujeito a:} \quad & \sum_{k \in N} z_k = p && (2.3)\\
& \sum_{k \in N} \sum_{l \in N} y_{ijkl} = 1, \quad i, j \in N && (2.4)\\
& \sum_{k \in N} y_{ijkl} \leq z_l, \quad i, j, l \in N && (2.5)\\
& \sum_{l \in N} y_{ijkl} \leq z_k, \quad i, j, k \in N && (2.6)\\
& s \geq \sum_{k \in N} \sum_{l \in N} (\gamma c_{ik} + \alpha c_{kl} + \beta c_{lj}) y_{ijkl}, \quad i, j \in N && (2.7)\\
& y_{ijkl}, z_k \in \{0, 1\}, \quad s \in \mathbb{R}_+, \quad i, j, k, l \in N && (2.8)
\end{aligned}$$

Neste modelo, a Expressão (2.3) garante que o número de *hubs* selecionados seja exatamente igual a p. A Expressão (2.4) assegura que a demanda (i, j) seja atendida apenas por um arco (k, l). As Expressões (2.5) e (2.6) garantem que os nós não *hubs* sejam atribuídos apenas a nós *hubs*. A Expressão (2.7) determina que s seja o limitante superior de custo, dentre todas as demandas do grafo.

2.2.2 CMApHCP

Para definir o modelo matemático para o problema CMApHCP, é naturalmente necessária a inclusão de restrições de capacidade no problema não-capacitado (UMAp-HCP). Os valores de capacidade dos nós *hubs* podem ser incluídos de diferentes formas (Campbell et al., 2007):

(i) atribuídos apenas ao nó de coleta;

(ii) atribuídos ao nó de coleta e ao nó de distribuição; ou,

(iii) atribuídos ao arco de coleta/entrega.

Na presente dissertação são tratados dois modelos para o CMApHCP. O primeiro modelo apresenta valores de capacidade para os nós de coleta e distribuição. No segundo modelo, os valores de capacidade são incluídos apenas no nó de coleta. Os dois modelos e suas restrições de capacidade são melhores descritos a seguir.

2.2.2.1 CMApHCP com Capacidade Dupla

Em sistema reais de demanda origem-destino, é mais efetivo considerar restrições de capacidade aos nós *hubs*. Desta forma, a capacidade, normalmente, está ligada tanto ao *hub* de coleta quanto ao de distribuição. Portanto, o primeiro modelo do CMApHCP considera a capacidade no *hub* de coleta e no *hub* de distribuição e, assim, o fluxo trafegado é descontado tanto da capacidade do *hub* de coleta, quanto na capacidade do *hub* de distribuição, ou seja:

$$\sum_{i \in N} \sum_{j \in N} w_{ij} \left[\sum_{l \in N} (y_{ijkl} + y_{ijlk}) - y_{ijkk} \right] \leq cap_k, \qquad k \in N \tag{2.9}$$

que inclui restrições de capacidade nos nós de coleta k e distribuição l $(y_{ijkl} + y_{ijlk})$, sendo w_{ij} a demanda entre os nós (i, j) e cap_k a capacidade do nó k. Além disso, para toda demanda distribuída pelo mesmo nó de coleta, o fluxo é descontado apenas da capacidade do nó responsável pela coleta do fluxo $(-y_{ijkk})$.

A caracterização do modelo matemático para o CMApHCP se dá a partir da descrição do UMApHCP, de forma que os nós *hubs* passam a possuir restrições de capacidade. Para construir o modelo matemático do CMApHCP com Capacidade Dupla, a Expressão (2.9) é incluída no modelo matemático do problema não capacitado (UMApHCP), de forma que o modelo completo para o problema é descrito por:

$$\min \quad s \tag{2.10}$$

$$\text{sujeito a:} \quad \sum_{k \in N} z_k = p \tag{2.11}$$

$$\sum_{k \in N} \sum_{l \in N} y_{ijkl} = 1, \quad i, j \in N \tag{2.12}$$

$$\sum_{k \in N} y_{ijkl} \leq z_l, \quad i, j, l \in N \tag{2.13}$$

$$\sum_{l \in N} y_{ijkl} \leq z_k, \quad i, j, k \in N \tag{2.14}$$

$$\sum_{i \in N} \sum_{j \in N} w_{ij} \left[\sum_{l \in N} (y_{ijkl} + y_{ijlk}) - y_{ijkk} \right] \leq cap_k, \quad k \in N \tag{2.15}$$

$$s \geq \sum_{k \in N} \sum_{l \in N} (\gamma c_{ik} + \alpha c_{kl} + \beta c_{lj}) y_{ijkl}, \quad i, j \in N \tag{2.16}$$

$$y_{ijkl}, z_k \in \{0, 1\}, \quad s \in \mathbb{R}_+, \quad i, j, k, l \in N \tag{2.17}$$

A Expressão (2.11) garante que o número de *hubs* selecionados seja exatamente igual a p *hubs*. A Expressão (2.12) garante que a demanda (i, j) seja atendida apenas por um arco (k, l). As Expressões (2.13) e (2.14) garantem que os nós não *hubs* sejam atribuídos apenas a nós *hubs* e a Expressão (2.16) determina que a variável s seja o limitante superior de custo, dentre todas as demandas do grafo. A Expressão (2.15) é inserida para garantir as características de capacidade ao problema.

2.2.2.2 CMApHCP com Capacidade Única

Em sistemas específicos, a capacidade pode estar atribuída apenas ao nó coleta do fluxo. Um exemplo típico dessa situação reside em sistemas postais, nos quais a capacidade pode estar majoritariamente atrelada à capacidade de triagem do nó de coleta. Esta capacidade pode representar as ações de triagem e transporte das correspondências ao nó de distribuição e, consequentemente, ao seus destinatários. Esta situação pode ser representada por:

$$\sum_{i \in N} \sum_{j \in N} \sum_{l \in N} w_{ij} y_{ijkl} \leq cap_k, \quad k \in N \tag{2.18}$$

em que a capacidade está atribuída somente ao nó de coleta k. Deve-se observar que a capacidade do nó k é subtraída pelo volume de tráfego w_{ij} coletado por ele.

Para construir o modelo completo do Problema CMApHCP com Capacidade Única, basta realizar a troca da Expressão (2.15) pela Expressão (2.18) no modelo apresentado pelas Expressões (2.10)-(2.17).

2.3 Considerações do Capítulo

O objetivo principal deste capítulo foi descrever o Problema de Localização de *Hubs*. A Seção 2.1 apresenta os Problemas de Localização de *Hubs*, suas classificações e variantes, além de apresentar pesquisas relevantes na área. Artigos correlatos

a este foram avaliados e descritos, neste trabalho, afim de melhor definir uma direção de pesquisa. A partir desta revisão de literatura, identificou-se uma lacuna na pesquisa de problemas de localização de *hub* com critério de avaliação min-max (*pHub* Centro), em específico problemas com características de alocações múltiplas (MA). Foi identificado também lacunas nos modelos relacionados à inclusão de restrições de capacidades dos *hubs*. Na Seção 2.2 foi realizado um levantamento mais detalhado das pesquisas que englobam estas características, com o objetivo principal de justificar estas afirmações. Finalmente, modelos matemáticos para as versões não-capacitada e capacitada do Problema *pHub* Centro de Múltiplas Alocações foram descritos nas Seções 2.2.1 e 2.2.2.

Capítulo 3

Fundamentação Teórica

Este capítulo realiza uma revisão de literatura a respeito de temas essenciais para o desenvolvimento desta dissertação. Inicialmente, na Seção 3.1, é apresentada uma revisão sobre metaheurísticas. Nessa seção, os conceitos relacionados às metaheurísticas e sua utilização como métodos de solução são introduzidos, e, em seguida, são relacionadas algumas das principais metaheurísticas disponíveis na literatura. O conceito de metaheurísticas híbridas é apresentado na seção seguinte (Seção 3.2). Na Seção 3.3 e 3.4 são introduzidos conceitos de sistemas multiagentes e aprendizagem de máquina, respectivamente. Particularmente, a Seção 3.4.1 direciona a discussão para uma forma específica de aprendizado de máquina, o aprendizado por reforço. A Seção 3.5 caracteriza *frameworks* para metaheurísticas, além de mostrar e comparar *frameworks* disponíveis na literatura . Em seguida, a Seção 3.6 apresenta as características e principais funcionalidades do *Framework* `AMAM`. Por fim, a Seção 3.7 registra as considerações do Capítulo.

3.1 Metaheurísticas

Em problemas de Otimização Combinatória, examinar todas as alternativas possíveis, quando o número de variáveis e possibilidades cresce, se torna, normalmente, não aplicável, exceto em problemas de pequenas dimensões (Gaspar-Cunha et al., 2012). Contudo, mesmo em problemas dessa complexidade, muitas vezes encontrar uma boa solução já pode ser satisfatório. Sendo assim, muitas pesquisas concentram seus esforços em resolver estes problemas com a utilização de heurísticas.

Uma heurística tem como objetivo alcançar boas soluções em tempo computacional reduzido, embora não garanta a otimalidade e não informe o quão distante a solução encontrada se encontra da solução ótima do problema. As heurísticas podem ser classificadas, segundo Blum e Roli (2003), como (i) heurísticas construtivas e (ii) heurísticas de refinamento. A primeira envolve métodos que constroem uma solução, elemento a elemento, sendo que a escolha de cada elemento a ser inserido considera uma função de avaliação definida de acordo com o problema. Já as heurísticas de refinamento consistem em melhorar uma solução corrente a partir da exploração sistemática, através de uma dada vizinhança do problema.

Metaheurísticas são métodos que combinam métodos heurísticos básicos em estruturas de nível superior destinadas a explorar de forma eficiente e eficaz um espaço de

busca (Blum e Roli, 2003). Metaheurísticas possuem diversas definições na literatura. Stützle (1998) define que:

> "*Metaheurísticas são tipicamente estratégias de alto nível que orientam uma heurística subjacente, mais específica do problema, para aumentar seu desempenho. O objetivo principal é evitar as desvantagens da heurística, permitindo que a busca local escape dos ótimos locais. Isso é conseguido permitindo a piora de movimentos ou gerando novas soluções iniciais para a busca local de uma forma mais "inteligente"do que apenas fornecer soluções iniciais aleatórias. Muitas das abordagens metaheurísticas dependem de decisões probabilísticas feitas durante a pesquisa. Mas, a principal diferença para a pesquisa aleatória pura é que em algoritmos metaheurísticos a aleatoriedade não é usada cegamente, mas de uma forma inteligente e tendenciosa.*"

Existem diferentes maneiras de se classificar e descrever algoritmos metaheurísticos (Blum et al., 2005). Blum et al. (2005) descrevem que, dependendo das características selecionadas para diferenciá-las, várias classificações são possíveis, sendo cada uma delas o resultado de um ponto de vista específico. Há classificações em metaheurísticas inspiradas na natureza × não inspiradas na natureza; em métodos baseados em memória × métodos sem memória; métodos que usam uma função objetivo dinâmica ou estática, dentre outras. Assim como Blum et al. (2005), essa dissertação classifica as metaheurísticas em duas vertentes: (i) metaheurísticas baseadas em busca local × (ii) metaheurísticas baseada em populações.

As metaheurísticas baseadas em busca local ou de trajetória incluem métodos que consistem em aplicar movimentos em uma solução corrente para encontrar novas soluções promissoras. Algumas das principais metaheurísticas baseadas em busca local encontradas na literatura são: SA (*Simulated Annealing*) (Kirkpatrick et al., 1983), TS (*Tabu Search*) (Glover e Laguna, 1998), GRASP (*Greedy Randomized Adaptative Search Procedure*) (Feo e Resende, 1995), VNS (*Variable Neigborhood Search*) (Mladenović e Hansen, 1997) e ILS (*Iterated Local Search*) (Lourenço et al., 2001). Por outro lado, as metaheurísticas baseadas em população consistem em criar populações de soluções e combiná-las para encontrar melhores soluções. As principais metaheurísticas baseadas em população são: GA (*Genetic Algorithms*) (Goldberg, 1989), ACO (*Ant Colony Optimization*) (Dorigo et al., 1996), PSO (*Particle Swarm Optimization*) (Eberhart e Kennedy, 1995).

A seguir, apresentamos um breve resumo a respeito das principais metaheurísticas disponíveis na literatura. Para uma melhor compreensão acerca dessas metaheurísticas e do próprio tema, sugere-se a leitura de Stützle (1998); Blum e Roli (2003); Blum et al. (2005); Gaspar-Cunha et al. (2012); Silva (2019), além da própria literatura de referência de cada metaheurística.

SA - Simulated Annealing - A metaheurística *Simulated Annealing (SA)* é uma técnica baseada em busca local, proposta inicialmente por Kirkpatrick et al. (1983), que simula o resfriamento de átomos aquecidos, comumente conhecida como recozimento de metais. Tal método consiste em, partindo de uma solução inicial qualquer, a cada iteração do *loop*, um único vizinho da solução corrente é gerado. Se este vizinho é melhor que a melhor solução corrente (solução de melhora), é aceito como a nova

solução corrente. Caso este vizinho seja uma solução de piora, esta solução poderá ser aceita com uma certa probabilidade $e^{-\Delta/T}$. Nesta expressão, T é um parâmetro, que, inicialmente, recebe um valor alto, de forma que o método tende a aceitar maior número de soluções de piora e, gradativamente, esse valor é diminuído, de modo que, quanto mais próximo de 0 (zero), um número menor de soluções de piora é aceito; o parâmetro Δ corresponde à diferença da função objetivo atual em relação à função objetivo corrente. *Simulated Annealing* é um método rápido, de baixo computacional e que independe de uma boa solução inicial.

TS - Tabu Search - A metaheurística Busca Tabu (*Tabu Search - TS*) é um método de busca local que explora a vizinhança de uma solução corrente a partir do melhor vizinho da mesma. Proposta por Glover e Laguna (1998), esta técnica desenvolve uma estrutura de memória, afim de evitar ciclos no processo de evolução do algoritmo. A estrutura de memória tem a função de evitar que o processo fique preso em ótimos locais. A estratégia de escolha sempre pela melhor solução vizinha pode fazer com que ciclos de soluções sejam encontrados; desta forma, uma lista com soluções (Lista Tabu) é criada, para armazenar soluções já visitadas. Assim, uma melhor solução vizinha só é aceita caso ela não esteja contida na Lista Tabu. A TS é um método eficiente, porém lento, e se torna uma boa alternativa para problemas que apresentem muitas soluções iguais.

GRASP - Greedy Randomized Adaptive Search Procedure - A Metaheurística GRASP *(Greedy Randomized Adaptive Search Procedure - GRASP)*, proposta por Feo e Resende (1995), é um método de busca local iterativa, composta por duas etapas: (i) fase de construção e (ii) fase de busca local. A primeira fase consiste em criar, elemento a elemento, uma solução inicial, enquanto que a segunda fase consiste em encontrar um ótimo local. Na fase de construção, uma solução é criada de forma iterativa: a cada iteração, uma lista de elementos candidatos a entrar na solução é criada e ordenada, conforme o benefício preestabelecido de cada elemento. A partir desta lista, um elemento é selecionado de um subconjunto que contém os melhores elementos candidatos. Este método de escolha permite que diferentes soluções iniciais sejam criadas. O GRASP é um método de simples implementação e eficiente em um vasto número de problemas testados.

VNS - Variable Neighborhood Search - A metaheurística VNS (*Variable Neighborhood Search - VNS*) também é um método baseado em busca local, que realiza uma busca em vizinhança variável. Proposta por Mladenović e Hansen (1997), tal busca consiste em explorar, gradativamente, vizinhanças mais distantes, através de trocas sistemáticas de vizinhança. O VNS não segue uma trajetória, como outras metaheurísticas baseadas em busca local, mas, sim, explora vizinhanças gradativamente mais "distantes" (Mladenović e Hansen, 1997). De forma que, dada uma solução corrente e uma determinada estrutura de vizinhança, um vizinho é gerado e uma busca local é realizada, se o ótimo local gerado for melhor que a melhor solução até então, o processo de busca continua na mesma estrutura de vizinhança. Caso contrário, o processo muda para outra estrutura de vizinhança. A busca local também pode fazer uso de mais de uma estrutura de vizinhança, assim como a versão original do VNS, que utiliza-se de uma busca local VND (*Variable Neighborhood Descent*).

ILS - Iterated Local Search - A metaheurística ILS (*Iterated Local Search - ILS*), introduzido por Lourenço et al. (2001), também é um método baseado em busca local. O método ILS consiste em realizar buscas locais em novas soluções geradas por meio de pertubações realizadas nas soluções correntes. O ILS é dividido em quatro etapas: geração da solução inicial, busca local, pertubação e critério de aceitação. A primeira etapa é responsável pela criação da solução inicial. A segunda aplica a busca local na solução corrente. Já a terceira etapa consiste na aplicação da pertubação na solução. E por fim, a última etapa avalia qual solução continua no processo de busca, de acordo com critérios de aceitação definidos. A efetividade desta metaheurística está diretamente ligada ao conjunto de ótimo locais encontrados e aos métodos de busca local, pertubação e critérios de aceitação implementados.

GA - Genetic Algorithms - A metaheurística Algoritmo Genético (*Genetic Algorithms - GA*) é uma técnica análoga aos processos naturais de evolução, tendo sido proposta por Goldberg (1989). O GA consiste em, dada uma população, os indivíduos mais aptos geneticamente possuem uma maior probabilidade de sobrevivência e de gerar indivíduos mais aptos e propensos a sobreviverem. Cada indivíduo (solução) em um GA possui uma codificação e um método de decodificação. Cada indivíduo, definido por um cromossomo, é composto por genes, que são atributos da solução. Dada a forma de codificação e decodificação de um indivíduo, devemos avaliá-los. Para tal, uma função de aptidão é definida. O método GA é composto por 4 componentes principais: seleção dos pais, cruzamento, mutação e definição da população sobrevivente. A seleção consiste em selecionar pais, em uma população, afim de submetê-los ao cruzamento. A seleção comumente está relaciona à função de aptidão dos indivíduos, indivíduos mais aptos possuem maiores chances. O cruzamento consiste em combinar características dos pais selecionados, afim de gerar indivíduos mais aptos. A mutação consiste em uma pequena alteração genética em dado indivíduo. Já a definição da solução sobrevivente consiste em definir quais são os indivíduos que irão para a próxima geração do algoritmo. Algoritmos Genéticos são métodos eficientes, porém, lentos computacionalmente, além de fazerem uso de um bom número de parâmetros.

ACO - Ant Colony Optimization - A metaheurística Otimização por Colônia de Formigas (*Ant Colony Optimization Metaheuristic - ACO*) é um método que baseia-se no comportamento de um conjunto de agentes (formigas). Proposto por Dorigo et al. (1996), o ACO consiste na simulação do comportamento de uma colônia de formigas, em que os agentes cooperam entre si a partir da deposição de feromônio em busca de alimento. Inicialmente, as formigas caminham em busca de alimento e vão depositando feromônio no decorrer do caminho. Essa deposição sofre evaporação, fazendo, assim, com que caminhos mais longos sejam evitados; ao fim, o caminho de todas as formigas tende a ser o caminho mais curto, pois a deposição de feromônio será maior neste caminho. O ACO é um método populacional, pois cada formiga é uma solução, sendo assim, uma colônia de formigas representa uma população de soluções.

PSO - Particle Swarm Optimization - O método de Otimização por Enxame de Partículas (*Particle Swarm Optimization - PSO)* é uma técnica populacional,

proposta por Eberhart e Kennedy (1995), que simula o comportamento de um bando de pássaros à procura de um alvo. O alcance deste alvo é dado pelo trabalho em conjunto e pelo compartilhamento de informações. A evolução do algoritmo é dado pela experiência do individuo (componente cognitivo) e da população (componente social). A velocidade de evolução depende da velocidade anterior, distância entre sua posição atual e sua melhor posição, e distância entre sua posição e a melhor posição atingida pelo grupo. Com isso, suas trajetórias são descritas e a direção é dada por aqueles agentes que possuem uma melhor posição de sobrevoo.

3.2 Metaheurísticas Híbridas

Metaheurísticas híbridas são métodos que consistem em combinar metaheurísticas com outras metaheurísticas, incorporar componentes de metaheurísticas em outra, ou até mesmo combinar metaheurísticas com outros métodos de Inteligência Computacional ou Pesquisa Operacional (Silva, 2019), como, por exemplo, métodos exatos. Blum e Roli (2008) definem Metaheurísticas Híbridas, no amplo sentido, como: "*...integração de um conceito relacionado à metaheurística com algumas outras técnicas (possivelmente outra metaheurística).*"

Uma forma tradicional de criar uma metaheurística híbrida é a inclusão de um método de trajetória local em um algoritmo populacional. Porém, criar métodos híbridos não consiste apenas em mesclar métodos heurísticos. Podemos também incorporar métodos exatos a metaheurísticas e vice e versa. Atualmente, devido ao fato do foco do desenvolvimento não estar atrelado ao algoritmo e, sim, ao problema, várias pesquisas incorporam métodos híbridos na sua resolução, apresentando muita das vezes os melhores resultados obtidos (Cotta et al., 2005), (Blum et al., 2011). Revisões amplas sobre metaheurísticas híbridas podem ser encontrada em (Blum e Roli, 2008; Blum et al., 2010, 2011; Silva et al., 2018)

3.3 Sistemas Multiagentes

A Inteligência Artificial Distribuída (IAD) é uma subárea da Inteligência Artificial (IA) que utiliza vários processos autônomos para resolver problemas extensos e complexos. Cada processo resolve o problema de forma completa ou em partes, comunicando e partilhando informações entre-si. A IAD possui três áreas de pesquisas (Ferber e Weiss, 1999): (i) Inteligência Artificial Paralela; (ii) Solução de Problemas Distribuídos; e (iii) Sistemas Multiagentes. A Inteligência Artificial Paralela refere-se, basicamente, a técnicas de computação baseadas em multiprocessadores ou *clusters*, com o objetivo de diminuir o tempo de processamento. Já a segunda, similar à primeira, considera como um problema pode ser resolvido de forma a compartilhar recursos e conhecimento entre entidades de computação; porém, está técnica é muita rígida, devido as suas estratégias incorporadas. Como resultado, está área possui pouco ou nenhuma flexibilidade. Sistemas Multiagentes, terceira área, são técnicas atribuídas a agentes autônomos em um ambiente multiagente, em que estes sabem se portar como entidades autônomas de computação, tornando, assim, o sistema mais flexível.

Um conjunto de agentes autônomos, que desejam resolver um certo problema, compartilhando conhecimento e comunicando entre si, é definido como um Sistema Multiagente (SMA) (Balaji e Srinivasan, 2010). SMA é uma subárea da IAD que apresentou um grande crescimento devido a sua flexibilidade e inteligência disponível, solucionando problemas distribuídos. Em um SMA, cada entidade de computação é chamada de Agente. Em Wooldridge (2009), um Agente engloba dois recursos importantes. São eles:

(i) Autonomia - todo agente, até certo ponto, é capaz de tomar decisões, afim de alcançar seus objetivos;

(ii) Interação - todo agente é capaz de interagir com outros agentes, não só partilhando informações, mas participando de atividades sociais, como, por exemplo, cooperação, coordenação e negociação.

Já Balaji e Srinivasan (2010) relata que SMA's foram amplamente adotados, em diversos domínios, devido aos seus benefícios, sendo que alguns destes são apresentados a seguir:

(i) maior velocidade e eficiência, graças ao processamento paralelo e sincronismo;

(ii) baixa degradação em caso de falha, sendo assim mais robusto e confiável;

(iii) escalabilidade e flexibilidade, agentes podem ser adicionados no sistema;

(iv) menor custo, sistema centralizados são mais caros computacionalmente.

(v) reusabilidade, agentes podem ser facilmente substituídos ou atualizados.

Apesar de um SMA possuir mais benefícios que um sistema simples, os SMA's possuem alguns pontos críticos a serem tratados, são eles:

(i) ambiente, de forma que a ação de um agente altera todo o ambiente;

(ii) percepção, cada agente possui visão parcial do ambiente;

(iii) abstração, em SMA o agente não experimenta todo o espaço, contudo é necessário saber a que informações cada agente possa ter acesso;

(iv) conflitos podem ocorrem devido a falta de visão global dos agentes;

(v) inferência, criação de um mecanismo de inferência efetivo, pois agentes podem ser heterogêneos e decisões podem surtir efeitos distintos entre agentes.

Esta seção fez uma breve revisão sobre Sistemas Multiagentes. Posto isso, uma importante característica incorporada pelos agentes, e destacada neste trabalho, é o aprendizado. O Aprendizado consiste na tomada de decisões pelo agente, seja a partir da sua base de conhecimento, seja através da interação com outros agentes e com o ambiente. Técnicas de aprendizado serão apresentadas na seção seguinte.

3.4 Aprendizado de Máquinas

Aprendizado de Máquina (*Machine Learning*) consiste em, dada uma determinada unidade computacional, esta possua recursos que a permitam aprender de forma autônoma. Também conhecido como Aprendizado Automático, o Aprendizado de Máquina tem por objetivo capacitar um determinado sistema computacional para a "tomada de decisão". Desta forma, computadores aprendem sem responderem a funções pré estabelecidas, ou serem explicitamente programados. Atualmente, a Aprendizagem de Máquina está no foco da comunidade científica, principalmente por sua capacidade de gerar modelos robustos e analisar problemas mais complexos.

O Aprendizado de Máquina é subdividido em três abordagens, em consonância com a natureza de seu *feedback* de aprendizagem (Ottoni et al., 2012). São elas:

(i) Aprendizagem supervisionada - modelo de aprendizagem em que o agente possui a figura de um professor, o qual lhe apresenta exemplos de entradas e saídas desejáveis;

(ii) Aprendizagem não supervisionada - modelo de aprendizagem em que nenhum padrão de resultado lhe é dado como referência, tendo o próprio agente que atingir um meio para chegar a um resultado;

(iii) Aprendizagem por reforço - modelo de aprendizagem em que o agente interage com um ambiente dinâmico afim de retornar melhores resultados. Cada agente recebe um sinal de *feedback* de sua ação na exploração do ambiente.

Em problemas de otimização, é essencial que o agente consiga tratar ambientes dinâmicos. Assim, se torna essencial um aprendizado automático capaz de sobrepor esses desafios. O Aprendizado por Reforço (*Reinforcement Learning - RL*) se mostra uma boa alternativa. O aprendizado por reforço é melhor apresentado na Seção 3.4.1. Formas de aprendizado por reforços muito utilizadas são: Autômatos de Aprendizagem e *Q-Learning*. Estas técnicas são utilizadas nesta dissertação e são apresentadas nas Seções 3.4.1.1 e 3.4.1.2.

3.4.1 Aprendizado por Reforço

No Aprendizado por Reforço, um agente aprende seu comportamento através de tentativas e erros, interagindo em um ambiente dinâmico. Iniciado nos primórdios da cibernética, psicologia, ciência da computação, estatística e neurociência, atualmente o Aprendizado por Reforço atrai esforços dos pesquisadores na área de Aprendizado de Máquina e Inteligência Artificial, devido principalmente a sua capacidade de modelar problemas sem que uma programação fortemente especifica seja desenvolvida. No aprendizado por reforço, apenas um modelo de recompensas é estabelecido.

Duas estratégias principais para resolver problemas de aprendizado por reforço são apresentadas em Kaelbling et al. (1996):

(i) Procura no espaço de comportamentos, a fim de encontrar aqueles que retornem melhores desempenhos.

(ii) Técnicas estatísticas e de programação dinâmica como ferramenta de estimar dados comportamentos.

Em um modelo tradicional de aprendizado por reforço, um agente conecta-se ao ambiente dinâmico por meio de percepções e ações, conforme a Figura 3.1 mostra.

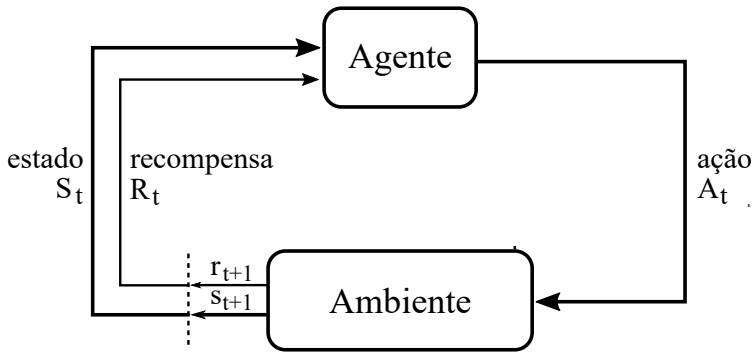


Figura 3.1: Modelo Aprendizado por Reforço. (Sutton e Barto, 1998).

Observa-se na Figura 3.1 que, a cada iteração, o agente recebe a entrada S_t, que indica o estado do Ambiente, e o agente escolhe uma ação A_t como saída. A ação altera o estado do ambiente e o valor dessa transição de estado é comunicado ao agente através de um sinal de reforço R_{t+1}. A próxima ação do agente é dada através do estado S_{t+1}. O comportamento do agente consiste em escolher ações que tendem a aumentar a soma dos valores de reforço a longo prazo. O agente pode aprender ao longo do tempo, por tentativas e erros sistemáticos, guiados por uma ampla variedade de algoritmos. O modelo de Aprendizado por Reforço é composto por: um ambiente, um conjunto discreto de ações e um conjunto de sinais de reforço.

A tarefa do agente é encontrar uma política de controle/decisão, definida pelo mapeamento de estados e ações, que maximize o somatório dos reforços encontrados. Em geral, o ambiente do agente é não determinístico, pois espera-se que a mesma ação, no mesmo estado, em tempos distintos, possuam valor de *feedback* diferentes. Existem várias diferenças entre o Aprendizado Supervisionado e o Aprendizado por Reforço, sendo a principal a não apresentação de pares de entrada e saída pelo modelo de Aprendizagem por Reforço. É importante que o agente colete informações sobre as possíveis transições e recompensas das ações afim de agir de forma otimizada. Uma forma de Aprendizado por Reforço se dá através de Autômatos de Aprendizagem que será melhor descrito na seção seguinte.

3.4.1.1 Autômatos de Aprendizagem

Autômatos de Aprendizagem (*Learning Automata*) é um tipo de Aprendizado por Reforço que permite a atuação em ambientes com pouca informação pré-definida e no qual informações adicionais são adquiridas em tempo real (Silva et al., 2019). Em autômatos de aprendizagem, o estado interno do agente é dado como uma distribuição de probabilidade, de acordo com as ações que seriam escolhidas. A probabilidade de tomar ações diferentes é ajustada de acordo com os sucessos e fracassos anteriores.

O funcionamento de um Autômato de Aprendizagem é dado pela sequência a seguir:

(i) a probabilidade inicial associada a cada ação são iguais;

(ii) uma ação é selecionada aleatoriamente;

(iii) observa-se, assim, a resposta do ambiente a esta ação;

(iv) atualiza-se as probabilidades com base na resposta do ambiente;

(v) a nova ação é escolhida considerando as probabilidades atualizadas;

(vi) o procedimento é repetido a partir do passo (iii).

Uma revisão detalhada sobre Autômatos de Aprendizado é dada em Narendra e Thathachar (1974).

3.4.1.2 *Q-Learning*

O algoritmo *Q-learning* foi proposto por Watkins (1989); Watkins e Dayan (1992) e implementa um método de aprendizado por reforço. O algoritmo *Q-learning* consiste em encontrar uma política ótima de tomada de decisão. Ele não requer um modelo do ambiente e pode tratar problemas transitórios e estocásticos sem modificações.

Para qualquer modelo de decisão de Markov finito (MDP - *Markov Decision Process*), o *Q-Learning* encontra uma política ótima, que maximiza o valor da soma das recompensas. Para o entendimento do *Q-learning*, seja a expressão:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{\alpha'} Q(s',a') - Q(s,a) \right] \tag{3.1}$$

Na Expressão (3.1), r corresponde ao valor de recompensa de mudar do estado atual s para o estado resultante s', sendo que a é a ação tomada. O parâmetro α representa a taxa de aprendizagem e γ é o fator de desconto. A taxa de aprendizado e o fator de desconto são parâmetros que dependem diretamente do problema tratado. A função Q estima a utilidade esperada da tomada de uma ação em um determinado estado s. Um episódio é definido aqui como uma sequência de estados que vão desde um estado inicial até o estado final.

Algoritmo 1 Algoritmo *Q-Learning*

Entrada: *QLearning*(r, α, ϵ, γ)

Inicialize $Q(s,a)$ arbitrariamente;

repita

  ▷ para cada episódio

  Inicialize s;

  repita

    ▷ para cada etapa do episódio

    Escolha a de s usando a política derivada de Q (e.g., ϵ-greedy);

    Tome uma atitude a;

    Observe o próximo estado s' e a recompensa r;

    $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{\alpha'} Q(s',a') - Q(s,a) \right]$;

    $s \leftarrow s'$;

  até *s terminar*;

até *Alcançar o número de episódios*;

O Algoritmo 1 apresenta o algoritmo *Q-learning*. O *Q-learning* é um algoritmo prático, popular e sem modelo, eficaz para aprender com reforço atrasado. Porém, sua conversão para uma boa política pode ser lenta.

3.5 *Frameworks* para Metaheurísticas

Frameworks são abstrações que são utilizadas para agrupar códigos que se repetem em um projeto de desenvolvimento de *software*. O preceito de um *framework* é a reutilização de códigos. Sendo assim, a utilização de *frameworks* torna o desenvolvimento de *softwares* menos oneroso e complexo, conseguindo assim aumentar a produtividade e qualidade do processo. A construção de *frameworks* se apodera de três princípios da Orientação a Objetos: (i) abstração; (ii) polimorfismo; e (iii) herança.

A utilização de *frameworks* na solução de problemas de otimização combinatória permite a resolução de diversos tipos de problemas, facilitando a inserção de novos agentes (métodos de resolução) e permitindo a hibridização desses métodos. Assim, os *frameworks* se tornam ferramentas interessantes para a resolução de problemas de otimização combinatória.

Uma revisão e análise comparativa dos principais *frameworks* para otimização encontrados na literatura é apresentada em Silva et al. (2018). Nesta revisão, os autores identificam características desejáveis e lacunas existentes no desenvolvimento de *Frameworks* de Otimização usando Metaheurísticas, destacando, dentre eles, estruturas que apresentam sistemas multiagentes no desenvolvimento de metaheurísticas híbridas para a solução de problemas de otimização.

Com base em Silva et al. (2018), buscando avaliar estes *frameworks*, a presente dissertação se concentrou na análise das seguintes características:

(i) Hibridização - propriedade do *framework* que permite a implementação de metaheurísticas híbridas;

(ii) Computação paralela/distribuída - capacidade de implementação de métodos paralelos ou distribuídos no *framework*. Mais especificamente, a execução de subtarefas em *Threads* ou execução em máquinas distintas;

(iii) Sistemas Multiagentes - utilização dos conceitos de sistemas multiagentes na definição do framework;

(iv) Aprendizado - capacidade do agente do *framework* de aprender durante o processo de busca da solução.

Tabela 3.1: Características desejáveis em um *Framework* (Silva et al., 2018)

			Computação			
Framework	**Referências**	**Hib.**	**Par.**	**Dist.**	**Age.**	**Apren. Agentes**
AgE	Turek et al. (2016)	Sim	Sim	Sim	Sim	Não
AMAM	Silva (2007) Fernandes (2009) Silva et al. (2014, 2015) Silva et al. (2019)	Sim	Sim	Não	Sim	*Learning Automata Q-Learnig.*
AMF	Meignan et al. (2008b,a) Meignan et al. (2009, 2010)	Sim	-	-	Sim	Aprendizado por Reforço Mimetismo.
CMA	Malek (2009, 2010)	Sim	Sim	Sim	Sim	Não
DAFO	Danoy et al. (2005, 2010)	Sim	-	-	Sim	Não
EasyLocal++	Gaspero e Schaerf (2001) Gaspero e Schaerf (2002) Gaspero e Schaerf (2003)	Sim	-	-	-	-
ECJ	White (2012)	Não	Sim	Sim	-	-
EvA2	Kronfeld et al. (2010)	Não	Sim	Sim	-	-
FOM	Parejo et al. (2003)	Não	Não	Não	-	-
HeuristicLab	Wagner e Affenzeller (2004) Wagner e Affenzeller (2005) Wagner et al. (2010, 2014)	Não	Sim	Sim	-	-
HotFrame	Fink et al. (1999) Fink e Voß (2002)	Não	*Não*	Não	-	-
HyFlex	Ochoa et al. (2012)	Sim	-	-	-	-
JABAT	Barbucha et al. (2010)	Sim	Sim	Sim	Sim	Não
JAMES	De Beukelaer et al. (2015) De Beukelaer et al. (2017)	Não	Sim	Não	-	-
JCLEC	Ventura et al. (2008) Cano et al. (2015)	Não	Não	Não	-	-
jMetal	Durillo et al. (2010) Durillo e Nebro (2011)	Sim	Sim	Sim	-	-
LBMAS	Lotfi e Acan (2015) Lotfi e Acan (2016)	Sim	-	-	Sim	*Learning Automata.*
MACS	Martin et al. (2016)	Sim	Sim	Sim	Sim	Aprendizado por Reforço
MAGMA	Milano e Roli (2004)	Sim	-	-	Sim	Não
MALLBA	Alba et al. (2002, 2006) Alba et al. (2007)	Não	Sim	Sim	-	-
MANGO	Kerçelli et al. (2008) Günay et al. (2009) Aydemir et al. (2012)	Sim	Sim	Sim	Sim	Não
OptFrame	Coelho et al. (2010, 2011)	Sim	Sim	Sim	-	-
Opt4j	Lukasiewycz et al. (2011)	Não	Sim	Sim	-	-
ParadisEO	Cahon et al. (2004) Liefooghe et al. (2011) Melab et al. (2013) Humeau et al. (2013)	Sim	Sim	Sim	-	-
SMA	Aydin (2010, 2012, 2013)	Sim	-	-	Sim	Não

A Tabela 3.1 destaca, das características desejáveis analisadas por Silva et al. (2018), apenas as de interesse para esta dissertação. A Tabela 3.1 mostra os *frameworks* para Otimização usando Metaheurísticas identificados em Silva et al. (2018), de forma que é verificada a presença ou ausência das características destacadas. Nesta tabela, a coluna coluna "Hib" indica se o *framework* permite a implementação de metaheurísticas híbridas. As colunas "Par" e "Dist" indicam se o *framework* utiliza-se de métodos de paralelismo e computação distribuída, respectivamente. Já a coluna "Age." indica se o *framework* utiliza os conceitos de sistema multiagentes na sua definição. Por fim, a coluna "Apren. Agente." indica se os agentes do *framework* implementam algum tipo de aprendizado de máquina e qual a forma utilizada. O aprendizado é avaliado apenas para os *frameworks* que implementam o conceito de sistemas multiagentes em sua estrutura.

Desta forma, após avaliar a Tabela 3.1, observa-se que, dos 25 *frameworks* analisa-

dos, 16 permitem a hibridização de metaheurísticas e 15 deles implementam alguma capacidade de computação paralela ou distribuída. Porém, somente 11 *frameworks* permitem a implementação de sistemas multiagentes e em apenas 4 destes o agente do sistema possui a capacidade de aprendizado: `AMAM`, `AMF`, `LBMAS` e `MACS`. É importante destacar também que, dos *frameworks* analisados, apenas dois possuem as 4 características desejáveis apresentadas anteriormente: `AMAM` e `MACS`.

O *framework* `AMAM` (Arquitetura MultiAgente para Metaheurística) foi proposto, inicialmente, em Silva (2007), e, em seguida, novas versões foram apresentadas em Fernandes (2009), Silva et al. (2015) e Silva (2019). O *Framework* `AMAM` é uma arquitetura multiagente para metaheurísticas, na qual cada agente implementa uma metaheurística/metaheurística híbrida que age de forma a buscar uma solução para o problema instanciado. Neste *framework*, o espaço de busca do problema tratado corresponde ao ambiente do sistema multiagente. Desta forma, com uma simples troca do ambiente, altera-se com facilidade o problema a ser solucionado. A facilidade de inclusão de novos agentes na estrutura multiagente do *framework* garante a sua escalabilidade, sendo que a adição desses agentes afeta ao mínimo o restante da estrutura. Cada agente possui capacidades adaptativas, desenvolvidas a partir dos conceitos de Aprendizagem por Reforço.

A arquitetura apresenta também uma estrutura de cooperação entre os agentes, dada, na versão atual, apresentada nesta dissertação na Seção 3.6.4, através do compartilhamento de soluções em um espaço denominado *Pool* de Soluções. O objetivo desta estrutura de cooperação é orientar os agentes no espaço de soluções em direções mais promissoras, buscando melhorar o resultado e reduzir o tempo de solução. As regras de acesso ao *Pool* de Soluções são apresentadas em Silva (2019).

O *framework* `MACS` (*Multi-agent Cooperative Search* – Busca Cooperativa Multiagente) foi proposto por Martin et al. (2016). `MACS` é um *framework* distribuído baseado em agentes. Esta estrutura é implementada na plataforma `JADE` (Bellifemine et al., 2007), importante *middleware* para desenvolvimento de sistemas multiagentes utilizando Java. `JADE` fornece ferramentas de infra-estrutura de comunicação e visualização. A proposta do `MACS` consiste em que cada agente execute diferentes combinações de metaheurísticas e heurísticas de busca local. Tal ferramenta, assim como a `AMAM`, possui uma estrutura de cooperação, responsável pela comunicação entre os agentes durante a busca da solução. Segundo Martin et al. (2016), o `MACS` possui dois tipos de agentes, que são eles:

(i) Agente lançador: compõe o problema a ser resolvido pelos agentes metaheurísticos, convertendo o problema para protocolo proposto de mensagens entre agentes;

(ii) Agente metaheurístico: determina as metaheurísticas/heurísticas de busca local disponíveis, para execução pelos agentes. Cada agente pode usar uma combinação de parâmetros distintos de configuração.

Os elementos *SolutionElements*, *Edge*, *Constraints* e *SolutionData* são elementos da estrutura básica do `MACS` responsáveis pela construção do protocolo de comunicação do *framework*. Martin et al. (2016) ainda define que o `MACS` tem, como foco, problemas de programação e roteamento. O protocolo de comunicação é chamado de *conversa* no `MACS`. Uma *conversa* é obtida a partir da identificação de bons padrões

de uma determinada solução de melhora, sendo que este padrão é compartilhado com todos os agentes. Cada Agente metaheurístico quebra a solução em *edges* e envia para o Agente lançador. Os *edges* obtidos farão parte de próximas soluções. Portanto, a hibridização de metaheurísticas ocorre de uma maneira diferente, uma vez que nesta proposta apenas partes da solução são compartilhadas, e não toda a solução.

O *Framework* MACS também incorpora conceitos de Aprendizagem de Máquinas. O modelo de aprendizagem utilizado é implementado através de uma ferramenta de memória de curto prazo. Esta memória armazena bons *edges* das soluções, que são utilizados no inicio de cada *conversa* entre os agentes. Desta forma, estes *edges* influenciam a construção de novas soluções. Estes *edges* são utilizados para reordenar a lista de *edges* a serem inseridos nas soluções.

Nesta seção foi apresentada uma análise entre *frameworks*, extraída de Silva (2019). Do comparativo apresentado por Silva (2019), foram destacadas as características desejáveis a um *framework* que são de interesse desta dissertação. O objetivo aqui é, considerando os principais *frameworks* da literatura, identificar o *framework* ideal para o uso na solução de problemas de otimização usando metaheurísticas e que atenda aos interesses propostos. A partir daí, foram destacados os *frameworks* MACS e AMAM. Os dois *frameworks* foram melhor analisados e a opção foi feita pela utilização do *Framework* AMAM. Esta escolha se dá devido ao *Framework* MACS ser fortemente ligado à resolução de um conjunto de problemas específico e ao seu compartilhamento de informações ocorrer apenas com partes das soluções. Adicionalmente, o aprendizado utilizado no *frameworks* MACS é baseado apenas em uma memória de curto prazo de objetos (*edges*). Por outro lado, o *Framework* AMAM não está ligado a um conjunto específico de problemas de otimização e o compartilhamento ocorre com a solução inteira. Já o modelo de aprendizagem consiste em algoritmos mais robustos da literatura e é utilizado no ajuste de parâmetros das metaheurísticas.

3.6 Framework AMAM

Esta seção apresenta o *Framework* AMAM (Arquitetura MultiAgente para Metaheurística), detalhando sua concepção e principais características. Inicialmente, a Seção 3.6.1 apresenta sua estrutura de funcionamento. Em seguida, nas Seções 3.6.2, 3.6.3 e 3.6.4, são descritas as dimensões nas quais o *Framework* AMAM é dividido. Por fim, a Seção 3.6.5 apresenta a versão atual e o modelo de licença utilizado.

O *Framework* AMAM foi proposto inicialmente em Silva (2007), e, em seguida, novas versões foram apresentadas em Fernandes (2009), Silva et al. (2015) e Silva et al. (2019). O *Framework* AMAM é uma arquitetura multiagente para metaheurísticas, na qual cada agente implementa uma metaheurística e age de forma a buscar uma solução para o problema instanciado. Neste *framework*, o espaço de busca do problema tratado corresponde ao ambiente do sistema multiagente. Assim, com uma simples troca do ambiente, altera-se com facilidade o problema a ser tratado. A capacidade de movimentação de cada agente pelo ambiente se dá pelas formas de manipulação da solução que cada problema possui. A facilidade de inclusão de novos agentes na estrutura multiagente do *framework* garante a sua escalabilidade, sendo que a adição desses agentes afeta minimamente o restante da estrutura. Estes agentes possuem ainda a capacidade de interagir e perceber o ambiente de forma cooperativa, além de

possuírem capacidades adaptativas.

A estrutura de cooperação entres os agentes da arquitetura AMAM é realizada, na versão atual, através do compartilhamento de soluções em um espaço denominado *Pool* de Soluções. O objetivo desta estrutura é orientar os agentes no espaço de soluções em direções mais promissoras, buscando melhorar o resultado e reduzir o tempo de solução. As regras de acesso ao *Pool* de Soluções são apresentadas em Silva et al. (2019).

3.6.1 Estrutura de Funcionamento

A Figura 3.2 apresenta a estrutura do *Framework* AMAM. Este *framework* é dividido em três dimensões, que são:

- Dimensão de Ambiente: dimensão que corresponde ao espaço de busca do problema tratado. Sendo assim, formado por todas as soluções do problema. Desta forma, todas as informações sobre o problema são definidas nesta dimensão;
- Dimensão de Agente: dimensão correspondente ao agente propriamente dito. Cada agente possui suas habilidades e sua forma de interação com o ambiente;
- Dimensão Social: dimensão responsável pela interação e cooperação entre os agentes.

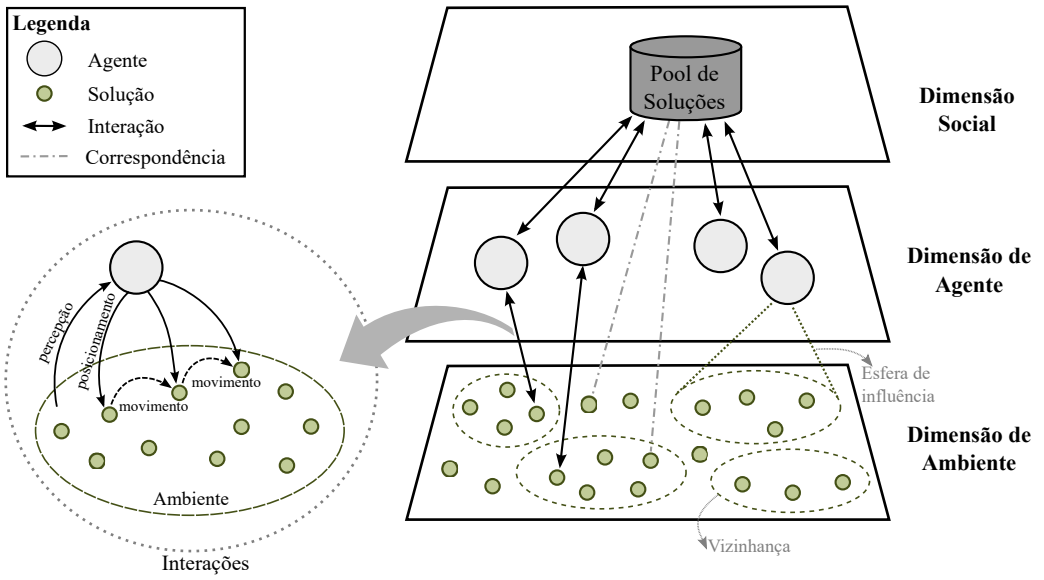


Figura 3.2: Estrutura do *Framework* AMAM (Silva et al., 2019).

Observa-se, na Figura 3.2, que o agente, através de sua capacidade de percepção, se posiciona no espaço de busca de soluções e, assim, pode se mover pelo mesmo em busca de melhores soluções. Para isso, todo agente encapsula uma heurística/metaheurística que definirá seu comportamento no processo de busca da solução. O *Framework* AMAM disponibiliza tanto a implementação completa de heurísticas e metaheurísticas, prontas para uso, quanto o arcabouço de heurísticas e metaheurísticas que facilitam o desenvolvimento de novos métodos.

O Framework AMAM apresenta quatro pacotes principais de codificação:

- Ambiente (*Environment*) - compreende todas as informações do problema em especifico, como dados de entrada das instâncias, representação da solução e formas de manipulação das mesmas;
- Métodos (*Methods*) - responsável pelas classes utilizadas nas definições dos métodos de resolução do problema de otimização, incluindo as heurísticas e metaheurísticas;
- Experimento (*Experiment*) - é o módulo responsável pela execução de um experimento no *Framework* AMAM. Nele estão contidas todas as classes que fazem com que uma execução do *framework* seja realizada conforme esperado. É nele também que todos os dados e resultados obtidos nas execuções são gravados;
- Sistema Multiagente (*MultiagentSystem*) - compreende todas as classe necessárias para o sistema multiagente do *framework*, como, por exemplo, as *Threads*, para execução independente de cada agente, *Cooperation* e *Learning*.

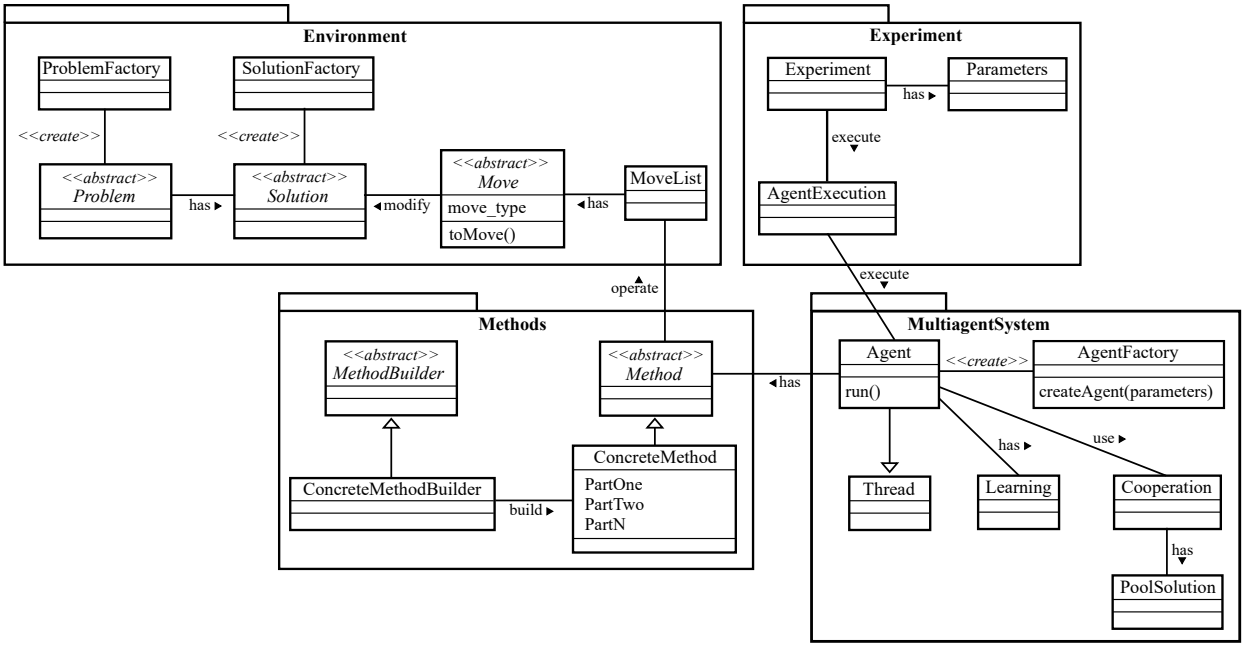


Figura 3.3: Diagrama de Classe *Framework* AMAM (Silva et al., 2019).

A Figura 3.3 apresenta o diagrama de classe do *Framework* AMAM, contendo os quatro principais pacotes de implementação da estrutura. Podemos observar nesta Figura que a dimensão de ambiente é instanciada pelo pacote *Environment*, que é responsável pelos dados do problema, a representação da solução, assim como pelas formas de movimentação no ambiente de busca. O pacote *Methods* implementa as heurísticas/metaheurísticas para a solução do problema. Observa-se ainda que cada método opera uma lista de movimentos contida no pacote *Enviroment*. Cada elemento *Agent*, definido no pacote *MultiagentSystem*, possui um método (*Methods*) atribuído a ele. O pacote *MultiagentSystem* é responsável pela instanciação dos agentes, além de possuir módulos de paralelismo, aprendizagem e cooperação entre agentes. Pode-se também observar o pacote *Experiment*, pacote responsável pela execução do experimento propriamente dito, contendo os parâmetros do problema e da execução.

3.6.2 Dimensão do Ambiente

O ambiente do *Framework* AMAM é o próprio espaço de busca do problema. Assim, a Dimensão do Ambiente consiste em: (i) definir a estrutura de dados que descreve o problema (Classe *Problem*); (ii) como uma solução é representada computacionalmente (Classe *Solution*), e (iii) como é dado o movimento do agente neste ambiente (Classe *Move*), ou seja:

- Classe *Problem* - consiste na definição do problema, com a leitura dos dados da instância e das informações específicas;
- Classe *Solution* - responsável pela representação da solução na sua forma computacional. Ela define funções que operam sobre a solução, tais como: avaliação da função objetivo e inserção de elementos na solução;
- Classe *Move* - responsável pela forma de movimentação do agente pelo ambiente.

As classes *Problem* e *Solution* são classes abstratas, responsáveis por estruturar computacionalmente o problema de otimização. A classe *Move* trata a movimentação do agente e compõe a lista de possíveis movimentos (MoveList).

3.6.3 Dimensão dos Agentes

A Dimensão dos Agentes define as características que cada agente possui. Cada agente do *Framework* AMAM possui características/habilidades que vão de percepção, posicionamento e movimento no espaço de busca, até aprendizado e compartilhamento de soluções. São elas:

- Percepção do Ambiente - característica que torna o agente capaz de enxergar parcialmente o ambiente ao qual faz parte;
- Posicionamento - capacidade de se posicionar no ambiente de busca da solução, seja através de uma nova solução ou pela escolha de uma solução existente;
- Movimento - condição de se movimentar pelo espaço de busca, de uma solução a outra, usando, por exemplo, estruturas de vizinhanças ou operadores genéticos (Algoritmos Genéticos).

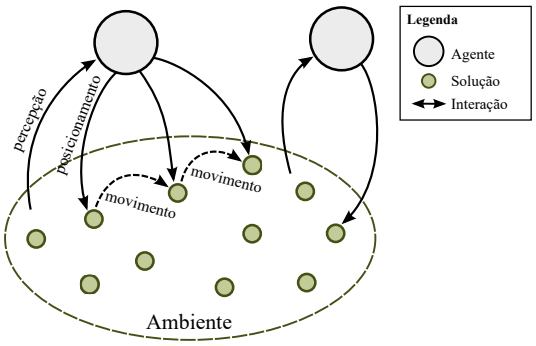


Figura 3.4: Estrutura do *Framework* AMAM.

A Figura 3.4 apresenta a interação do agente com o Ambiente, o espaço de busca do problema tratado. Observe que o agente tem a percepção do ambiente se posiciona e se movimenta pelo espaço busca.

Com avanço do Aprendizado de Máquinas e a necessidade do *Framework* `AMAM` de se adaptar às especificidades dos diversos problemas de otimização, torna-se essencial a adaptabilidade dos agentes. Neste sentido, os agentes do *Framework* `AMAM` possuem duas formas de aprendizagem implementadas:

(i) Autômatos de Aprendizagem - primeira forma de aprendizagem incorporada ao *framework* `AMAM`. Aprendizado semelhante ao apresentado em Narendra e Thathachar (1974) ou à roleta utilizada em Algoritmos Genéticos. Este algoritmo é aplicado na escolha da ordem de vizinhança na busca local do método;

(ii) *Q-Learning* - também utilizado na escolha da estrutura de vizinhança da busca local, aqui realizada pelo método de Descida em Vizinhança Variável (*Variable Neighborhood Descend - VND*) (Mladenović e Hansen, 1997). Baseado no método de Aprendizado por Reforço *Q-Learning*, utiliza uma tabela Q para determinar os valores dos pares estado-ação no decorrer do processo de busca. Esta tabela define a recompensa recebida por determinada ação.

3.6.4 Dimensão Social

A Dimensão Social é responsável pela interação e cooperação entre os agentes do sistema. Esta interação é obtida através de uma estrutura denominada *Pool* de Soluções que é responsável pela troca de informações entre os agentes. A estrutura de cooperação da `AMAM` tem como principal objetivo guiar os agentes por regiões mais promissoras no espaço de busca.

O *Pool* de Soluções possui um tamanho pré-definido e regras de acesso ao mesmo, tanto para leitura quanto escrita. As soluções são armazenadas no *Pool* ao final de cada iteração do agente e obedecendo as regras de diversidades do *Pool*. O funcionamento do Pool é descrito em detalhes em (Silva et al., 2019).

3.6.5 Versão `AMAM`

A versão mais recente do *Framework* `AMAM`, lançada em outubro de 2020, está disponível em https://github.com/mamelials/AMAM-MultiagenteArchitecture-for-Metaheuristics, sob a licença GNU LGPLv3.

3.7 Considerações do Capítulo

Este Capítulo descreve brevemente o estado da arte que fundamenta esta pesquisa. Inicialmente, é destacada a utilização de metaheurísticas na resolução de problemas de otimização combinatória, principalmente, em problemas da classe NP-Difícil, assim como são apresentadas as principais metaheurísticas encontradas atualmente na literatura. Em seguida, a hibridização de metaheurísticas é discutida.

Na Seção 3.3 são apresentados conceitos de sistemas multiagentes, com o objetivo de identificar benefícios e pontos críticos da utilização destes modelos. É discutida

também a utilização de sistemas multiagentes na solução de problemas de otimização combinatória.

Uma característica importante e desejável em um sistema multiagentes, o aprendizado automático, é apresentado na Seção 3.4. Uma breve descrição de técnicas de Aprendizado de Máquinas, especificamente de Aprendizado por Reforço, foi apresentada, permitindo, assim, identificar os benefícios da utilização destas técnicas.

Por fim, *Frameworks* para otimização usando metaheurísticas são apresentados, com intuito de identificar características desejáveis a este tipo de ferramenta. A partir dessa análise de *frameworks*, baseada em Silva (2019), optou-se pela utilização do *Framework* `AMAM`. Na Seção final, o *Framework* `AMAM` é descrito, com suas principais características e funcionalidades.

Capítulo 4

Proposta de solução do UMApHCP

Este capítulo introduz duas propostas de solução do Problema *pHub* Centro não Capacitado de Múltiplas Alocações (UMApHCP). A Seção 4.1 apresenta uma implementação de algoritmo genético para a solução deste problema. Em seguida, a Seção 4.2 mostra a implementação da metaheurística *Iterated Local Search* (ILS) para a solução do UMApHCP. Esta implementação de ILS é realizada usando o *Framework* `AMAM`.

4.1 Algoritmo Genético Aplicado ao UMApHCP

Esta seção se concentra em apresentar uma implementação de Algoritmo Genético (AG – *Genetic Algorithm*) para solucionar o Problema *pHub* Centro não Capacitado de Múltiplas Alocações (UMApHCP). O algoritmo genético implementado nesta proposta é um GA híbrido, que utiliza a metaheurística GRASP para a construção de soluções iniciais e um método exato de caminho mínimo para alocar os clientes aos *hubs*. É implementado aqui o GA no formato tradicional, composto de uma população inicial, uma função de avaliação, cruzamento e mutação. Este método utiliza-se de técnicas evolutivas e conceitos de aprendizado nas escolha dos operadores. As subseções seguintes detalham o GA implementado.

4.1.1 Codificação da Solução

A codificação de um indivíduo (solução) é dada por um vetor Z de p posições, que indica os nós que serão *hubs* no sistema. A Expressão (4.1) representa um indivíduo, em que os nós 1, 3 e 4 são determinados como *hubs* em um sistema em que $n = 5$ e $p = 3$, considerando $n = |N|$ como o número de nós do problema.

$$Z_1 = \left[\begin{array}{ccc} 1 & 3 & 4 \end{array}\right], \qquad Z_2 = \left[\begin{array}{ccc} 4 & 1 & 3 \end{array}\right] \tag{4.1}$$

Observe, na Expressão (4.1), que os vetores Z_1 e Z_2 de *hubs* representam o mesmo indivíduo. É importante destacar que a ordem em que os *hubs* são inseridos não interfere no valor da função de aptidão do indivíduo.

4.1.2 Decodificação da Solução

Para decodificar uma solução são necessários, além do vetor Z, duas matrizes auxiliares, que determinam a alocação dos clientes aos *hubs*. A primeira matriz determina a alocação do nó de origem a um *hub*, enquanto a segunda determina a qual *hub* o nó de destino está alocado. Cada elemento da matriz contém o índice do vetor Z de *hubs* em que o cliente está alocado.

Considere o exemplo mostrado pelas Expressões (4.2) e (4.3), que representam um indivíduo para o UMApHCP, com $n = 5$ e $p = 3$, de forma que a Expressão (4.2) mostra os nós que serão *hubs* e as matrizes M_1 e M_2 da Expressão (4.3) representam, respectivamente, o link do nó de origem e o link do nó de destino de uma demanda origem-destino. Desta forma, a matriz M_1 na posição (i, j) contém o índice do vetor Z, correspondente ao *hub* responsável pela coleta da demanda (i, j). Já a matriz M_2, na posição (i, j), contém o índice do vetor Z correspondente ao *hub* responsável pela distribuição da demanda (i, j).

$$Z = \begin{bmatrix} 1 & 3 & 4 \end{bmatrix} \tag{4.2}$$

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 3 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 3 & 2 & 2 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 & 3 \\ 1 & 2 & 2 & 3 & 2 \\ 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 3 & 3 \end{bmatrix} \tag{4.3}$$

Como exemplo, para atender à demanda $(2, 5)$, deve-se fazer o caminho $(2 \rightarrow 3 \rightarrow 4 \rightarrow 5)$, pois, na matriz M_1, a demanda $(2, 5)$ está ligada ao índice 2 do vetor Z e ligada ao índice 3 do vetor Z na matriz M_2. Assim, a demanda $(2, 5)$ está ligada ao arco $(3, 4)$ de *hubs*.

4.1.3 Adaptação do Algoritmo de Floyd-Warshall

A alocação de clientes aos *hubs* é dada como polinomialmente solúvel para o UMApHCP em Campbell et al. (2007). Desta forma, para definir essa alocação, é preciso encontrar o caminho mínimo entre todos pares de origem-destino do grafo. Para tal, é utilizada uma adaptação do algoritmo de Floyd-Warshall (Ahuja et al., 1993; Ernst e Krishnamoorthy, 1998; Brimberg et al., 2017a), que consiste em encontrar o caminho mínimo para todos os pares ordenados do grafo. A adaptação do algoritmo de Floyd-Warshall garante a redução da complexidade deste algoritmo de $O(n^3)$ para $O(n^2 p)$, reduzindo, portanto, o tempo de processamento do método. Para toda configuração de Z gerada, o algoritmo de Floyd-Warshall é executado, de modo que se determine as matrizes M_1 e M_2 de alocação e o custo entre todos os pares origem-destino, sendo esse custo armazenado em uma matriz $C_{n \times n}$, criada pelo método. A matriz C é iniciada com cada elemento tendo valor infinito positivo, caso a demanda (i, j) não possua conexão direta, e com a distância entre o nó origem e o nó destino multiplicado pelo fator de desconto, caso haja conexão direta entre a demanda (i, j).

O Algoritmo 2 apresenta a adaptação usada do algoritmo Floyd-Warshall. Ele tem, como entradas, as matrizes $C_{n \times n}$, que correspondem ao custo e ao caminho

Algoritmo 2 Adaptação de Floyd-Warshall

Entrada: $C_{n\times n}, n, p, Z_p$
Saída: $C_{n\times n}, M1_{n\times n}, M2_{n\times n}$
1 **início**
2 **para** $k = 1$ **até** p **faça**
3 **para** $i = 1$ **até** n **faça**
4 **para** $j = 1$ **até** N **faça**
5 **se** $C[i,j] > C[i,Z_k] + C[Z_k,j]$ **então**
6 $C[i,j] \leftarrow C[i,Z_k] + C[Z_k,j]$
  Atualiza Caminho($M1, M2$)
7 **fim**
8 **fim**
9 **fim**
10 **fim**
11 **fim**
retorna $C_{n\times n}$, $M1_{n\times n}, M2_{n\times n}$

inicial existentes entre cada demanda (i, j) do grafo; n, que informa o número de nós do problema; p, que corresponde ao numero de *hubs*; e o vetor Z_p, que corresponde aos nós *hubs*. Este algoritmo retorna as matrizes $C_{n\times n}$, $M1_{n\times n}$ e $M2_{n\times n}$, com o custo e o caminho mínimo de todas as demandas já calculados. Note que, na linha 2, os dois próximos *loops* são executados apenas p vezes. Com isto, podemos reduzir a complexidade do algoritmo para $O(N^2p)$. A linha 5 compara se caminhar de (i, j) passando por Z_k tem um menor custo. Caso isso ocorra, as matrizes C, $M1$ e $M2$ são atualizadas.

4.1.4 Construção da Solução Inicial

Para construir uma solução inicial para o UMApHCP para o GA, utilizou-se da fase de construção da metaheurística GRASP (Feo e Resende, 1995). Inicialmente, uma lista LRC (Lista Restrita de Candidatos) de nós candidatos a entrar na solução é criada, sendo C, a lista de candidatos, definida inicialmente como sendo formada por todos os nós do sistema. A lista LRC é determinada pela maior distância de cada nó em relação a todos os outros nós do sistema, de forma que aqueles que possuem a menor maior distância são inseridos no inicio da lista, conforme função guia $g(t)$ do GRASP a seguir:

$$g(t) = \max(d_{ti}), \quad \forall\ i \neq t \quad : \quad i, t \in C \tag{4.4}$$

Nesta Expressão, C corresponde aos nós candidatos a serem *hubs* na solução e d_{ti} o custo de sair do nó t e chegar ao nó i. O tamanho desta lista é definido pela variável per, que determina o quão guloso o método será. O valor de per varia entre 0 e 1, sendo que, quando $per = 0$, o método será totalmente guloso, e quando $per = 1$, será totalmente aleatório. Para construir uma população de boa qualidade e diversa, o valor de per varia durante a construção da população inicial, sendo dado por:

$$per = \frac{\log i}{\log tam\_populacao} \tag{4.5}$$

A Expressão (4.5) determina o valor da variável per durante o processo de construção da população inicial. A variável i representa o índice do indivíduo que está sendo inserido na população. O valor de per é determinado em base logarítmica para que não sejam inseridos muitas indivíduos pelo critério guloso na população inicial e, assim, a diversidade da mesma seja garantida.

Algoritmo 3 Constrói População Inicial

Entrada: tam_pop,
Saída: pop
1 $pop \leftarrow \emptyset$
$ins \leftarrow 1$
$per \leftarrow 0$
repita
2 $ind \leftarrow \emptyset$
$cont \leftarrow 1$
$C \leftarrow N$
repita
3 $g(t_{\min}) = \min\{g(t) \mid t \in C\}$
$g(t_{\max}) = \max\{g(t) \mid t \in C\}$
$RCL = \{t \in C \mid g(t) \leq g(t_{\min}) + per(g(t_{\max}) - g(t_{\min}))\}$
Aleatoriamente, selecione um elemento $t \in RCL$
$ind \leftarrow ind \cup \{t\}$
$cont \leftarrow cont + 1$
$C \leftarrow C \backslash \{t\}$
4 **até** $cont = p$;
5 $pop \leftarrow pop \cup ind$
$ins \leftarrow ins + 1$
$per \leftarrow$ Atualiza per (ins, tam_pop)
6 **até** $ins = tam\_pop$;
7 **retorna** pop

O Algoritmo 3 apresenta a construção da população inicial. O parâmetro de entrada é a dimensão da população (tam_pop). A saída é a população gerada (pop). Observe que o laço da linha 6 é responsável pela criação da população e o laço interno da linha 4 é responsável pela criação de um indivíduo. Observa-se também, no algoritmo, que o individuo é criado conforme um percentual de aleatoriedade (variável per), apresentado na linha 3, que é atualizada na linha 5, a partir da Expressão (4.5).

4.1.5 Função de aptidão

Para avaliação dos indivíduos no problema, utilizou-se de uma função de aptidão que representa a própria função objetivo do UMApHCP, ou seja:

$$apt = \max\{C_{ij}\} \quad \forall\, i, j = 1, \ldots, n \tag{4.6}$$

Na Expressão (4.6), atribui-se à função de aptidão apt o maior valor da matriz de custo C. De forma que esta seja minimizada (critério min-max).

4.1.6 Cruzamento

A seleção dos indivíduos que serão enviados para o cruzamento é dada por um torneio binário. O torneio binário consiste em: dados dois indivíduos selecionados aleatoriamente, selecione aquele que possuir melhor aptidão. Para cada cruzamento, o torneio binário é repetido por duas vezes, de maneira a selecionar dois pais para o mesmo cruzamento. Para o algoritmo genético proposto, foram desenvolvidos quatro operadores de cruzamento e dois métodos que combinam estes quatro cruzamentos. A seguir, são apresentados os operadores de cruzamento implementados.

(i) Cruzamento de 1 ponto de corte: o cruzamento de um ponto de corte é uma forma tradicional de realizar cruzamento em algoritmo genético. Dada a seleção de dois pais, este cruzamento consiste em gerar um ponto de corte aleatório.

O filho 1 recebe, do pai 1, os genes anteriores ao ponto de corte 1, e o filho 2 recebe, do pai 2, os genes anteriores ao ponto de corte 1. A partir do ponto de corte, os genes são alterados, de forma que o filho 1 receba os genes do pai 2 e o filho 2 receba os genes do pai 1. Para garantir a factibilidade do problema, inclui-se conceitos do operador de cruzamento `OX`. Quando o gene a ser inserido já estiver contido no cromossomo, insere-se os genes referentes à primeira parte de seu pai original;

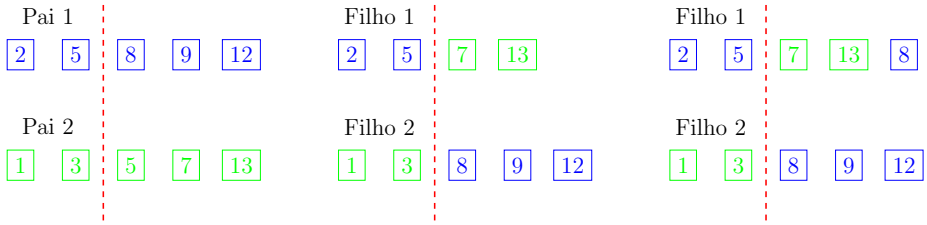


Figura 4.1: Cruzamento de 1 ponto de corte.

Na Figura 4.1, o filho 1 recebeu os genes de valor 2 e 5 referentes ao pai 1; já o filho 2 recebeu os genes de valores 1 e 3, referentes ao pai 2. Em seguida, o filho 1 recebeu os genes referente ao pai 2, de valor 7 e 13, pois o gene de valor 5 já estava contido no cromossomo. O filho 2 recebeu os genes de valor 8, 9 e 12, referentes ao pai 1. Na etapa final, para garantir a factibilidade, inseriu-se, no filho 1, o gene de valor 8, referente à segunda parte do cromossomo do pai 1.

(ii) Cruzamento de 2 pontos de corte: similar ao cruzamento de um ponto de corte, o cruzamento de dois pontos de cortes consiste em, dados os pais, dois pontos de corte aleatórios são gerados. O filho recebe os genes de um pai, entre o ponto de corte 1 e o ponto de corte 2. No passo dois, são inseridos os genes do outro pai, fora do intervalo entre os pontos de corte, a partir do primeiro gene do cromossomo. No passo dois, para garantir a factibilidade, caso algum gene já esteja inserido no cromossomo, genes entre os pontos de corte do pai original são inseridos no cromossomo, partindo do primeiro ponto de corte, até que o cromossomo esteja completo

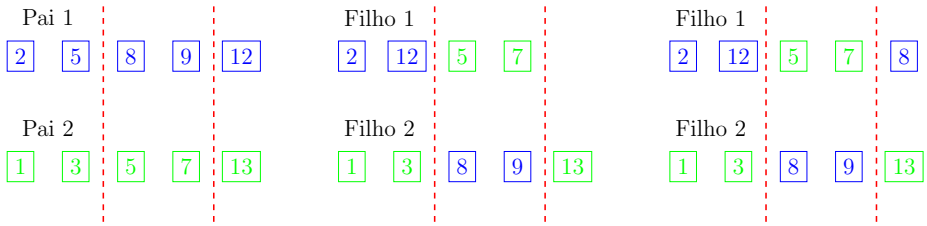


Figura 4.2: Cruzamento de 2 pontos de corte.

A Figura 4.2 mostra o cruzamento de dois pontos de corte. Neste caso, aleatoriamente, os pontos de cortes gerados foram 2 e 4. O filho 1 recebeu os genes de valor 5 e 7 do pai 2 e o filho 2 recebeu os genes de valor 8 e 9 do pai 1. Em seguida, o filho 2 recebeu os genes de valor 1, 3 e 13 referentes ao pai 2. Já o filho 1 receberia os genes 2, 5 e 12 do pai 1, porém o gene de valor 5 já estava

inserido no cromossomo, sendo assim inseriu-se o gene de valor 8 pertencente ao pai 1.

(iii) Cruzamento Aleatório: para o cruzamento aleatório, igualmente aos cruzamentos anteriores, 2 pais são escolhidos. A partir daí, enquanto o cromossomo do filho não estiver completo, aleatoriamente seleciona-se um pai e um gene a ser inserido na solução. Isso é repetido até que o cromossomo do filho esteja completo.

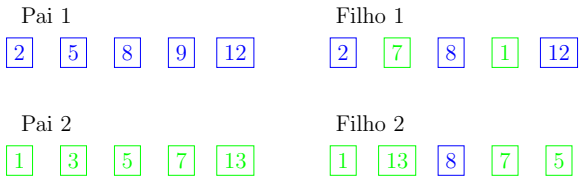


Figura 4.3: Cruzamento aleatório.

A Figura 4.3 representa o cruzamento aleatório. Neste caso, o filho recebe aleatoriamente um gene de um dos pais, até que o cromossomo do filho esteja completo. Note que o filho 1 recebe os genes de valor 2, 8 e 12 do pai 1 e os genes de valor 7 e 1 do pai 2, enquanto o filho 2 recebe os genes de valor 1, 13, 7 e 5 do pai 2 e apenas o gene de valor 8 do pai 1.

(iv) Cruzamento Fixo: o cruzamento fixo baseia-se em fixar os genes que se repetem em ambos os pais. Em seguida, é criada uma lista auxiliar, com os valores dos genes que não se repetem no pai 1 ou no pai 2. A partir daí, seleciona-se, aleatoriamente, pelo índice do vetor auxiliar, o gene que entrará no cromossomo do filho 1, ou seja, o gene pertencente ao vetor auxiliar do pai 1 ou do pai 2. Após, obrigatoriamente o filho 2 recebe o gene do vetor auxiliar, não selecionado anteriormente, o processo é repetido até o final dos índices do vetor auxiliar.

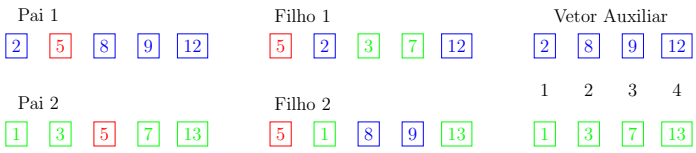


Figura 4.4: Cruzamento fixo.

A Figura 4.4 representa o cruzamento fixo. Note que o gene com valor 5 está repetido no pai 1 e no pai 2. Assim, o gene é herdado tanto no filho 1 quanto no filho 2. A partir daí, cria-se um vetor auxiliar para cada pai, com os genes que não se repetem. Os filhos recebem os valores dos genes de forma aleatória para cada índice do vetor auxiliar. Observa-se, então, no sorteio entre os valores de índice 1, que o filho 1 recebeu o gene de valor 8 referente ao pai 1 e o filho 2 o gene de valor 1, referente ao pai 2. Após, o filho 1 recebeu os genes de valor 3 e 7 do pai 2 e o gene de valor 12 do pai 1. Já o filho 2 recebeu os genes 8 e 9 do pai 1 e o gene 13 do pai 2.

4.1.7 Escolha do Cruzamento

São propostos dois métodos de escolha do cruzamento, a serem usados a cada geração: `Todos` e `Aprendizado`. Estes dois métodos utilizam os quatro cruzamentos apresentados anteriormente.

No método `Todos`, o operador de cruzamento a ser usado é escolhido aleatoriamente, em forma de roleta. Neste método, todos os cruzamentos têm a mesma chance de serem escolhidos, durante todo o processo de evolução. No método `Aprendizado`, os cruzamentos iniciam-se com chances iguais e, à medida que um certo cruzamento é escolhido, caso melhore os filhos, recebe um beneficio percentual (1%) pela sua escolha. Dessa maneira, o método tende a escolher cruzamentos com benefícios maiores e que, consequentemente, tende a melhorar seus descendentes.

Entretanto, este último método pode levar à utilização de apenas um dos cruzamentos ao longo das gerações. Como gerações diferentes podem apresentar comportamentos distintos e com o objetivo de explorar espaços de busca diversos, à medida que a população evolui, o benefício dos cruzamento é reiniciado, com base nas gerações do algoritmo.

4.1.8 Mutação

A mutação é dada pela alteração de *mut* posições do vetor de *hubs* Z. Está alteração consiste em retirar um nó *hub* do vetor de *hubs* e transformar um nó não *hub* em nó *hub*. A quantidade *mut* de *hubs* a serem alterados é dada por:

$$mut = \left\lceil \frac{p \times 100}{1000} \right\rceil \tag{4.7}$$

Esta expressão garante que pelo menos um bit do vetor de *hubs* seja alterado. Além disso, à medida que o número de *hubs* aumenta, o número de bits a serem alterados também cresce. Note que, como p é o número de *hubs* do problema, o número de mutações está atrelado à quantidade de *hubs* a se definir.

4.1.9 Seleção da População Sobrevivente

A geração seguinte do algoritmo genético é obtida de seguinte forma: se, em determinada seleção de pais, não ocorrer cruzamento, os próprios pais entram na geração seguinte; de outro modo, se, em uma dada seleção de pais, ocorrer o cruzamento, os filhos farão parte da próxima geração. Além disso, o algoritmo mantém elitismo simples em todas as suas gerações, ou seja, o melhor individuo é mantido na geração seguinte.

4.1.10 Algoritmo Genético proposto

O Algoritmo 4 apresenta o algoritmo genético proposto. O algoritmo recebe, como entrada, os parâmetros tam_pop, num_ger, tx_cruz e tx_mut, que representam o tamanho da população, o numero de gerações, a taxa de cruzamento e a taxa de mutação, respectivamente. A saída é dada pela variável $melhor\_ind$, que representa o melhor indivíduo encontrado pelo algoritmo. Observe que tanto o cruzamento

Algoritmo 4 Algoritmo Genético Proposto

```
   Entrada: tam_pop, num_ger, tx_cruz, tx_mut
   Saída: melhor_ind
1  pop ← Constrói População Inicial(tam_pop)
   Avalia População(pop)
   melhor_ind ← min(pop)
   ger ← 1
   repita
2      cont ← 1
       nova_pop ← [ ]
       repita
3          pais ← Seleciona dois pais (pop)
           se ale[0, 1] < tx_cruz então
4              filhos ← Cruzamento (pais)
               nova_pop ← nova_pop ∪ filhos
5          senão
6              nova_pop ← nova_pop ∪ pais
7          fim
8          se ale[0, 1] < tx_mut então
9              Mutação(nova_pop[cont])
10         fim
11         se ale[0, 1] < tx_mut então
12             Mutação(nova_pop[cont + 1])
13         fim
14         cont ← cont + 2
15     até cont = tam_pop;
16     Insere melhor indivíduo da geração anterior(nova_pop, melhor_ind)
       pop ← nova_pop
       Avalia População(pop)
       melhor_ind ← min(pop)
17     ger ← ger + 1
18 até ger = num_ger;
19 retorna melhor_ind
```

(linha 3) quanto a mutação (linhas 8 ou 11) ocorrem a partir de uma variável aleatória. Também pode-se observar o elitismo do algoritmo (linha 16), de forma que a melhor solução da geração anterior é inserida na nova população no lugar do pior indivíduo da nova população.

4.2 Agente ILS_AMAM Aplicado ao UMApHCP

Esse seção apresenta a utilização de agentes metaheurísticos no *Framework* `AMAM` para solucionar o UMApHCP. Cada agente implementa uma metaheurística híbrida formada pela metaheurística ILS combinada com o método GRASP, utilizado, aqui, para a construção de soluções iniciais. Adicionalmente, é utilizado um método de caminho mínimo para alocar clientes aos *hubs*. A seguir, são apresentados todos os detalhes desta implementação.

4.2.1 Representação da Solução

Representa-se uma solução do UMApHCP por um vetor binário Z, de dimensão n, e uma matriz inteira $M_{n\times n}$, sendo $n = |N|$ o número de nós. Cada elemento i do vetor binário representa o nó $i \in N$, que recebe o valor 1 se o nó i é um nó *hub*, e 0 se o nó i não é um hub. Desta forma, na representação de F mostrada na Expressão (4.8), tendo $n = 5$ nós e $p = 2$ *hubs*, os nós 2 e 4 são nós *hub* e os nós 1, 3 e 5 são nós não hub. A matriz M, por sua vez, representa o caminho de todo o grafo

com demanda origem-destino. Cada elemento (i, j) define o nó destino. Um exemplo desta representação é mostrado na Expressão (4.9). Observe que, para cumprir a demanda dada pelo arco $(1, 3)$, é necessário apenas passar pelo nó *hub* 2, pois, ao acessar o elemento de índice $(1, 3)$, encontra-se o *hub* 2; daí, ao acessar o elemento de índice $(2, 3)$ encontra, por fim, o nó destino 3; já para cumprir a demanda $(3, 1)$, deve-se passar pelos nós 4 e 2, ou seja, pelo arco $(4, 2)$. Ao acessar os índices $(3, 1)$ e $(4, 1)$, encontra-se os *hubs* 4 e 2, respectivamente. Ao acessar o elemento de índice $(2, 1)$ da matriz M, encontra-se, por fim, o nó destino 1.

$$Z = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{4.8}$$

$$M = \begin{bmatrix} 1 & 2 & 2 & 4 & 4 \\ 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 3 & 4 & 4 \\ 2 & 2 & 3 & 4 & 5 \\ 4 & 2 & 2 & 4 & 5 \end{bmatrix} \tag{4.9}$$

4.2.2 Estrutura de Vizinhança

A estrutura de vizinhança do UMApHCP é dada pela troca de um nó *hub* com um nó não *hub*, sendo denominada `swap_hub`. Como exemplo, a configuração $Z' = [10010]$ é uma solução vizinha de $Z = [01010]$, de modo que o nó 2 deixou de ser um nó *hub* e o nó 1 passa a ser o novo nó *hub*. A exploração do espaço de soluções é feita por $p - 1$ movimentos. Estes movimentos consistem em realizar i trocas `swap_hub` em uma solução corrente, de forma que i varia de 1 até $p - 1$. Após a geração de um vizinho, é necessário alocar clientes aos *hubs*, de forma que o sistema seja totalmente configurado. Sendo assim, é necessária a utilização de um método que determine o caminho entre todos os pares de demanda origem-destino e, assim, definir a configuração da matriz M. O método para obtenção do caminho mínimo, utilizado aqui para este fim, foi o algoritmo de caminho mínimo de Floyd-Warshall, apresentado na Seção 4.1.3.

4.2.3 Caminho Mínimo

O Algoritmo 5 apresenta a adaptação do Algoritmo 2, apresentado na Seção 4.1.3. Algumas diferenças dentre estes algoritmos devem ser enfatizadas, devido à diferença na representação da solução dos métodos utilizados.

O Algoritmo 5 apresenta a adaptação usada do algoritmo Floyd-Warshall. Ele tem, como entrada, as matrizes $C_{n\times n}$ e $M_{n\times n}$, que correspondem, respectivamente, ao custo e ao caminho inicial existentes entre cada demanda (i, j) do grafo; n, que informa o número de nós do problema; e o vetor Z_n, que corresponde aos nós *hubs*. Este algoritmo retorna as matrizes $C_{n\times n}$ e $M_{n\times n}$, com o custo e o caminho mínimo de todas as demandas já calculados. Note que, na linha 3, os dois próximos *loops* são executados se, e somente se, o nó k é *hub*. Com isto podemos reduzir a complexidade do algoritmo para $O(N^2p)$. A Linha 7 compara se caminhar de (i, j) passando por k tem-se um menor custo. Caso isso ocorra, as matrizes C e M são atualizadas.

Algoritmo 5 Adaptação de Floyd-Warshall

```
Entrada: C_{n×n}, M_{n×n}, n, Z_n
Saída: C_{n×n}, M_{n×n}
1  início
2      para k = 1 até n faça
3          se Z[k] = 1 então
4              para i = 1 até n faça
5                  se k ≠ i então
6                      para j = 1 até n faça
7                          se C[i,j] > C[i,k] + C[k,j] então
8                              C[i,j] ← C[i,k] + C[k,j]
                               M[i,j] ← M[k,j];
9                          fim
10                     fim
11                 fim
12             fim
13         fim
14     fim
15 fim
   retorna C_{n×n}, M_{n×n}
```

4.2.4 Função de Avaliação

A avaliação de uma solução *sol* é dada por:

$$f(sol) = \max\{C_{ij}\}, \quad i, j \in N \tag{4.10}$$

valor este que deve ser minimizado. Na Expressão (4.10), a condição max consiste em encontrar o maior custo calculado, dentre todas as demandas (i, j) do sistema, de forma que esta seja minimizada (critério min-max).

4.2.5 Construção da Solução Inicial

No algoritmo proposto, a implementação da construção da solução inicial se dá por uma construção parcialmente aleatória, semelhante à fase de construção inicial do algoritmo GRASP (Feo e Resende, 1995). Constrói-se uma lista restrita RCL_n de candidatos e, logo em seguida, atribui-se a cada nó i a maior distância entre ele e todos os nós j, definido pela função guia $g(t)$ do GRASP apresentada a seguir:

$$g(t) = \max(d_{ti}) \quad \forall\ i \neq t, i \in C \tag{4.11}$$

Nesta expressão, C corresponde à lista de nós candidatos a serem *hubs* na solução e d_{ti} o custo de sair do nó t e chegar ao nó i. Logo após, define-se, aleatoriamente, p *hubs*, condicionados à distância em relação ao melhor valor encontrado na RCL_n.

O Algoritmo 6 mostra como a solução inicial para o UMApHCP é gerada. Uma lista de candidatos é construída na linha 2, sendo que a linha 2 é responsável por garantir a aleatoriedade do método, de forma que sua gulosidade está condicionada a um percentual *per*. Neste trabalho, adotou-se o valor de $per = 0,75$.

4.2.6 Busca Local

Para resolução do UMApHCP foi implementada uma heurística de Primeira Melhora como técnica de busca local. A busca local é dada por uma exploração da vizinhança (V) da solução e, assim que um dado vizinho melhore a solução corrente,

Algoritmo 6 Construção Parcialmente Aleatória

Entrada: p, N, $dist_{n\times n}$, per
Saída: sol
1 $sol \leftarrow \emptyset$
$ins \leftarrow 1$
$C \leftarrow N$
repita
2 $g(t_{\min}) = \min\{g(t) \mid t \in C\}$
$g(t_{\max}) = \max\{g(t) \mid t \in C\}$
$RCL = \{t \in C \mid g(t) \leq g(t_{\min}) + per(g(t_{\max}) - g(t_{\min}))\}$
Aleatoriamente, selecione um elemento $t \in RCL$
$sol \leftarrow sol \cup \{t\}$
$ins \leftarrow ins + 1$
$C \leftarrow C \backslash \{t\}$
3 **até** $ins = p$;
4 **retorna** sol

esta solução é aceita, e passa a ser a solução corrente do método. Os vizinhos são analisados sequencialmente, na ordem em que eles surgem na busca. Esse procedimento é repetido enquanto houver melhoria na solução corrente. A busca local implementada é mostrada no Algoritmo 7.

Algoritmo 7 Busca Local Primeira melhora

Entrada: $sol\_corrente$
Saída: $melhor\_sol$
1 $melhor\_sol \leftarrow sol\_corrente$
$melhorou \leftarrow$ ***false***
repita
2 Selecione, de forma ordenada, $sol' \in V(sol)$;
se $f(sol') < f(melhor\_sol)$ **então**
3 $melhor\_sol \leftarrow sol'$
$melhorou \leftarrow$ ***true***
4 **senão**
5 $melhorou \leftarrow$ ***false***
6 **fim**
7 **até** $melhorou =$ ***false***;
8 **retorna** $melhor\_solução$

No procedimento do Algoritmo 7, a solução corrente é refinada para encontrar um ótimo local em relação a sua vizinhança. Note que a linha 2 gera um novo vizinho, e sua função é avaliada de forma a verificar se tal vizinho é melhor que a solução corrente.

4.2.7 Agente ILS

O método implementado, nesta proposta, como agente do *Framework*, é a metaheurística *Iterated Local Search* (ILS). Este é um método de busca iterativo, que se utiliza de perturbações progressivas na solução para tentar escapar da armadilha de ótimos locais, ou pesquisar diferentes ótimos locais. Iterativamente, este método constrói soluções, aplicando, em soluções correntes conhecidas, perturbações, tendo, como principal objetivo, a diversificação da busca. O ILS é uma metaheurística de fácil entendimento conceitual, porém, de uma grande eficiência em diversos problemas computacionais difíceis (Lourenço et al., 2001). Basicamente, o ILS é composto por quatro módulos:

(i) geração da solução inicial;

Algoritmo 8 Agente *Iterated Local Search*

```
Entrada: iter_max, nivel_max
Saída: sol*
1  sol^0 ← Gerar Solução Inicial
   sol* ← Busca Local(sol^0)
   iter ← 1
   nivel ← 1
   repita
2      sol' ← Perturba(sol*, nivel)
       sol'' ← Busca Local(sol')
       se f(sol'') < f(sol*) então
3          sol* ← sol''
           iter ← 1
           nivel ← 1
4      senão
5          iter ← iter + 1
           se nivel < nivel_max então
6              nivel ← nivel + 1
7          fim
8      fim
9  até iter = iter_max;
10 retorna sol*
```

(ii) busca local;

(iii) pertubação; e

(iv) critério de aceitação.

O processo do ILS consiste em criar uma solução inicial: aplica-se uma busca local, retornando um ótimo local. A partir daí, enquanto o critério de aceitação não é alcançado, aplica-se perturbações em níveis na solução corrente e, em seguida, novamente, é aplicada a busca local na solução perturbada. Ao fim do processo, retorna-se um ótimo local.

O Algoritmo 8 indica o procedimento da metaheurística ILS. O parâmetro $nivel_{\max}$ de perturbação do ILS é dado pelo parâmetro de entrada p, que determina o número de *hubs* a ser selecionados, sendo que o valor do mesmo corresponde a p-1 níveis de pertubações, para que não sejam geradas soluções totalmente aleatórias para o método, de forma que ele não se perca no espaço de busca. A perturbação em nível do ILS é executado na linha 2 e o critério de parada, na forma de número de execuções sem melhora, é inserido na linha 9.

Capítulo 5

Resultados UMApHCP

Este Capítulo apresenta os resultados obtidos com os experimentos computacionais aplicando os métodos de solução do UMApHCP descritos no Capítulo 4. Inicialmente, a Seção 5.1 apresenta a metodologia dos testes computacionais. A Seção 5.2 apresenta os resultados obtidos a partir da solução via programação matemática e usando o *solver* CPLEX. A Seção 5.3 mostra os resultados encontrados com o Algoritmo Genético aplicado ao problema. Por fim, a Seção 5.4 contém os resultados alcançados com o método ILS_AMAM.

5.1 Metodologia dos Testes para o UMApHCP

Para a execução dos experimentos computacionais, inicialmente aplicou-se, usando o *solver* CPLEX, o modelo de programação matemática para o problema UMApHCP. Em seguida, foram aplicados o Algoritmo Genético mostrado no Algoritmo 4 e a metaheurística ILS_AMAM mostrada no Algoritmo 8, este último instanciado no *Framework* `AMAM`.

Para realização dos testes foram utilizados instâncias disponíveis e consolidadas na literatura para o problema. Os resultados computacionais alcançados, utilizando metaheurísticas, foram comparados com os resultados apresentados em Brimberg et al. (2017a), uma vez que este artigo apresenta o mesmo modelo matemático para o problema não capacitado. O ambiente computacional de teste está indicado em cada uma das situações avaliadas.

5.2 Resultados para o UMApHCP via CPLEX

Para validar os modelos matemáticos do UMApHCP apresentados na Seção 2.2.1, foi desenvolvida uma implementação do modelo de programação matemática. Esta programação matemática foi desenvolvida em linguagem ILOG Concert Technology `C++`, usando-se o *solver* CPLEX versão acadêmica 12.9 e o editor de texto `Gedit` - version 3.36.1. Utilizou-se para este teste o conjunto de instâncias AP - (*Australian Post*), introduzido em Ernst e Krishnamoorthy (1996), que foi constituído a partir de dados reais do Serviço Postal Australiano. As instâncias deste conjunto possuem até 200 nós. Os testes computacionais foram realizados em um computador *Intel(R) Xeon(R) Silver* 4110 CPU 2.10GHz, 16 núcleos, memória RAM de 32GB, sistema

operacional *CentOS 6*, mantido pelo Laboratório de Computação de Alto Desempenho (LCAD) do CEFET-MG. O limite de tempo de execução de 7200 segundos foi definido para a execução de cada instância e os resultados computacionais obtidos são apresentados a seguir.

Os testes computacionais executados com o modelo UMApHCP utilizam os seguintes fatores de desconto/penalidade no cálculo do custo de transporte:

(i) $\gamma = 1,0$, $\alpha = 0,75$ e $\beta = 1,0$;

(ii) $\gamma = 3,0$, $\alpha = 0,75$ e $\beta = 2,0$.

Estes fatores foram extraídos do próprio conjunto de instâncias AP. O valores desses fatores dados no item (i) correspondem aos mesmos valores utilizados em Brimberg et al. (2017a). Desta forma, γ e β são fatores de penalidade no fluxo do nó de origem para o nó *hub* de coleta e do nó *hub* de distribuição para o nó de destino, respectivamente. Já α é fator de desconto do fluxo entre *hubs*.

Os resultados obtidos são apresentados nas Tabelas 5.1 e 5.2. Estas tabelas diferem-se pelos fatores de desconto tratados. As instâncias 1-1 referem-se aos valores de $\gamma = 1$ e $\beta = 1$. Neste caso, foram testadas os conjuntos tendo 10, 20, 25, 40 e 50 nós. Já as instâncias 3-2 referem-se aos valores de $\gamma = 3$ e $\beta = 2$. Neste caso, foram testadas os conjuntos tendo 10, 20, 25 e 40 nós.

As Tabelas 5.1 e 5.2 apresentam os resultados obtidos na execução dos testes para a solução do UMApHCP utilizando as instâncias AP, com os valores de γ e β iguais a 1-1 e 3-2, respectivamente. A coluna "Instância" mostra o nome da instância testada. Este nome possui a forma APn_h, em que "n" indica o número de nós e "h" o número de *hubs* considerado. A coluna "Solução" indica se foi encontrada uma solução e se esta solução é ótima ou não. A coluna "Valor" apresenta o valor da solução, caso seja encontrada. Já a coluna *gap*(%) mostra o *gap* relativo entre os melhores limitantes primal e dual. Por fim, a coluna "Tempo" apresenta o tempo computacional obtido.

Analisando estas tabelas, é possível observar que:

(i) no grupo de instâncias com γ e β utilizando os valores 1-1, apenas a instância `AP40_4` e as instâncias com 50 nós não apresentaram soluções ótimas;

(ii) no grupo de instâncias com γ e β utilizando os valores 3-2, apenas para as instâncias com 40 nós não foram encontradas soluções ótimas.

5.3 Resultados para o UMApHCP via Algoritmo Genético

Foram realizados testes computacionais utilizando o algoritmo genético apresentado pelo Algoritmo 4. Este algoritmo foi implementado em linguagem JAVA, utilizando IDE Eclipse (4.11.0). Utilizou-se, para os testes computacionais, os conjuntos de instâncias:

- CAB - (*Civil Aeronautics Board*): conjunto de instâncias apresentado em O'kelly (1987), extraído de uma pesquisa do Conselho de Aviação Civil dos EUA durante os anos de 1970. As instâncias deste conjunto possuem 25 nós;

Tabela 5.1: Resultados para UMA*p*HCP – Solução via CPLEX – Instâncias 1-1

Instância	**Solução**	**Valor**	***gap*(%)**	**Tempo(s)**
AP10_2	Ótima	39922,11	0,00	30,30
AP10_3	Ótima	32713,94	0,00	38,93
AP10_4	Ótima	31577,96	0,00	33,68
AP10_5	Ótima	30371,32	0,00	36,20
AP20_2	Ótima	45954,15	0,00	21,21
AP20_3	Ótima	40909,59	0,00	22,14
AP20_4	Ótima	38320,25	0,00	31,81
AP20_5	Ótima	37868,15	0,00	24,47
AP20_10	Ótima	37868,15	0,00	18,01
AP25_2	Ótima	51533,30	0,00	84,31
AP25_3	Ótima	45552,50	0,00	70,14
AP25_4	Ótima	45552,50	0,00	80,87
AP25_5	Ótima	45552,50	0,00	84,41
AP25_10	Ótima	45552,50	0,00	65,43
AP40_2	Ótima	61140,80	0,00	1893,87
AP40_3	Ótima	56309,88	0,00	2501,43
AP40_4	Factível	51279,14	0,00	7200,47
AP40_5	Ótima	49741,20	0,00	2694,46
AP40_10	Ótima	49741,20	0,00	1650,60
AP50_2	Factível	58449,92	0,23	7128,14
AP50_3	Factível	52896,09	0,19	7127,38
AP50_4	Factível	50707,87	0,44	7131,81
AP50_5	Factível	50707,87	0,52	7131,52
AP50_10	Factível	50707,87	0,52	7132,40

- URAND: conjunto de instâncias de maior porte, geradas aleatoriamente, introduzidas em Meyer et al. (2009), tendo conjuntos de 100, 200, 300 e 400 nós, e em Ilić et al. (2010), tendo conjuntos de 500 e 1000 nós. Os testes aqui apresentados se referem apenas aos conjuntos de 100, 200 e 300 nós.

Todos os testes computacionais realizados foram executados por 33 vezes em um computador *Intel Core I5* (8250U), 1,6 GHz (4 Cores 8 Threads), 8 GB de memória RAM e sistema operacional *Ubuntu 19.04*. Para a apresentação dos resultados computacionais obtidos, primeiramente, será mostrada a escolha do operador de cruzamento adotado; em seguida, os resultados computacionais completos são postos.

5.3.1 Escolha do operador de cruzamento

Inicialmente, foi realizado um conjunto de testes para avaliar qual método de seleção do cruzamento seria utilizado na execução do GA, ou seja, se o método `Todos` ou o método via `Aprendizado`. Estes métodos estão descritos na Seção 4.1.7. Para estes testes foram utilizadas instâncias do conjunto URAND, selecionadas aleatoriamente, sendo 2 instâncias com 100 nós (tm100.05 e tm100.10), 2 instância com 200 nós (tm200.02 e tm200.40) e 2 instâncias com 300nós (tm300.05 e tm300.15).

Tabela 5.2: Resultados para UMA*p*HCP – Solução via CPLEX – Instâncias 3-2

Instância	Solução	Valor	gap (%)	Tempo(s)
AP10_2	Ótima	99805.28	0.00	40.84
AP10_3	Ótima	70337.49	0.00	45.96
AP10_4	Ótima	68714.17	0.00	48.21
AP10_5	Ótima	55439.28	0.00	80.40
AP20_2	Ótima	110220.25	0.00	55.39
AP20_3	Ótima	92839.94	0.00	83.52
AP20_4	Ótima	80901.66	0.00	133.03
AP20_5	Ótima	74162.48	0.00	134.00
AP20_10	Ótima	47794.95	0.00	163.64
AP25_2	Ótima	117182.56	0.00	265.67
AP25_3	Ótima	102737.89	0.00	227.09
AP25_4	Ótima	88159.77	0.00	402.16
AP25_5	Ótima	78173.77	0.00	458.65
AP25_10	Ótima	53964.09	0.00	652.37
AP40_2	Factível	128083.20	0.16	7188.55
AP40_3	Factível	98279.19	0.27	7206.51
AP40_4	Factível	82726.64	0.60	7189.93
AP40_5	Factível	79435.96	0.64	7187.84
AP40_10	Factível	54412.07	0.74	7185.15

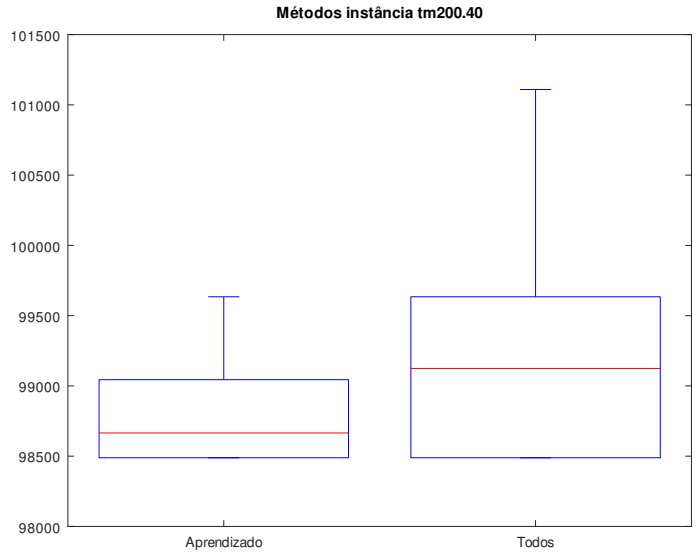


Figura 5.1: Resultados dos Métodos `aprendizado` e `todos` para a instância tm200.40.

Para este fim, foram definidos, os seguintes parâmetros:

(a) Tamanho da População=100;

(b) Taxa de Cruzamento=0, 85;

(c) Taxa de Mutação=0, 05; e

(d) Número de gerações=500.

Os valores destes parâmetros foram definidos empiricamente. Os resultados obtidos a partir do uso dos dois métodos foram comparados utilizando um teste de hipóteses paramétrico, com nível de confiança de 95%. O teste utilizado foi o Teste ANOVA (análise de variância), que consiste em testar se duas ou mais populações são iguais ou possuem diferenças significativas. Com base nesta análise, evidenciou-se que os dois métodos possuem diferenças significativas e, a partir da afirmação obtida com o Teste ANOVA, utilizou-se do gráfico da Figura 5.1 para constatar qual dos métodos é mais eficiente. Concluiu-se, então, que a utilização do método de `aprendizado` foi mais eficiente e este foi tomado, portanto, como método para a resolução, via algoritmo genético, do UMApHCP.

5.3.2 Apresentação dos resultados

Para a validação do desenvolvimento proposto, foram realizados testes computacionais para verificar o comportamento da população ao longo da evolução do algoritmo.

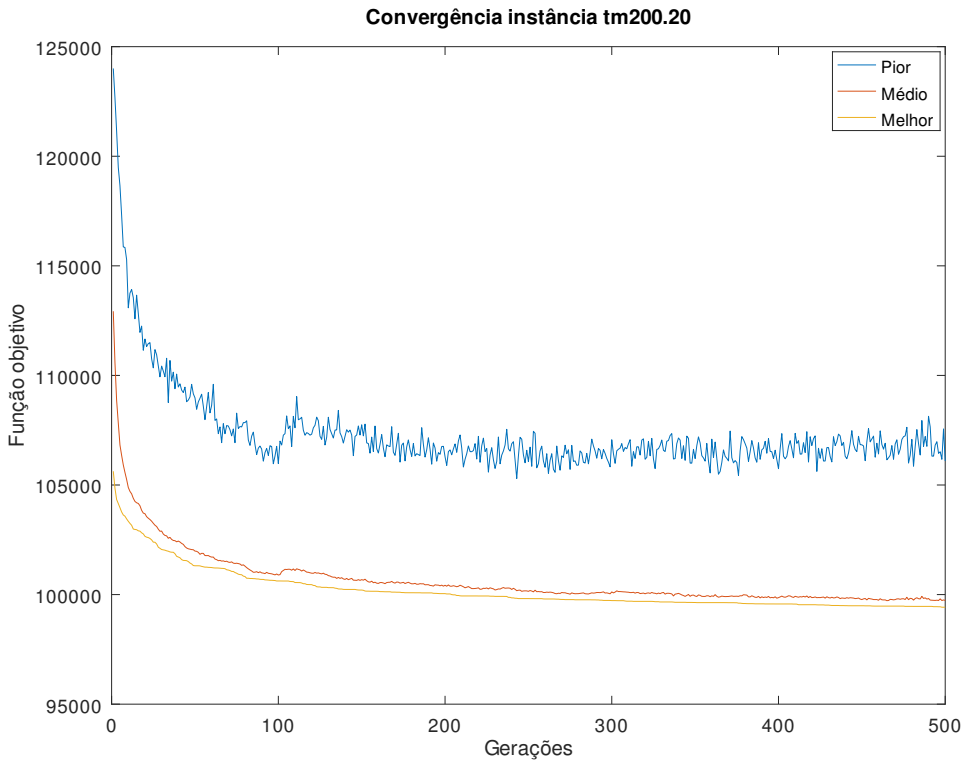


Figura 5.2: Comportamento da população.

A Figura 5.2 mostra a evolução da função objetivo ao longo das gerações do algoritmo GA, utilizando a instância tm200.20. Este gráfico apresenta o pior individuo, a média dos indivíduos e o melhor individuo da população durante a evolução do GA. Note que a média dos valores da população encontra-se sempre próxima à melhor solução em todas as gerações; o pior indivíduo de cada geração, porém, encontra-se afastado do melhor indivíduo da população, constatando-se, assim, que a população

do algoritmo tende a ser boa e diversa. Observa-se também o processo de elitismo do algoritmo, pois o valor da função objetivo do melhor individuo de cada geração é sempre igual ou maior que o valor da função objetivo do melhor individuo da geração anterior.

Para melhor analisar o desenvolvimento proposto, foram realizadas comparações com os resultados apresentados pelo algoritmo BVNS, proposto em Brimberg et al. (2017a). O valor de $gap(\%)$ entre a melhor solução e a mediana das soluções, em relação à melhor solução encontrada pelo BVNS, é calculado da seguinte forma:

$$gap(\%) = \left(\frac{AGpHCP\_Valor - BVNS\_Valor}{BVNS\_Valor} \right) 100 \tag{5.1}$$

Na Expressão (5.1), $AGpHCP\_Valor$ é o valor encontrado no algoritmo genético proposto. Seu valor é dado pela melhor solução encontrada pelo algoritmo proposto, quando o $gap(\%)$ é em relação à melhor solução; é dado pela mediana das soluções, quanto o cálculo do $gap(\%)$ é feito em relação à mediana. A variável $BVNS\_Valor$ é o valor da melhor solução encontrado em Brimberg et al. (2017a).

No conjunto de instâncias CAB, o algoritmo genético proposto alcançou, em todas as instâncias, o valor ótimo apresentado pelo BVNS; portanto, nesse caso, o valor de $gap(\%)$=$0,00\%$ tanto para a mediana quanto para a melhor solução. Este resultado valida o desenvolvimento proposto com relação às instâncias com número pequeno de nós. Os resultados para o conjunto de instâncias URAND são apresentados na Tabela 5.3.

Os resultados para o conjunto de instâncias URAND são apresentados na Tabela 5.3. Esta tabela apresenta a melhor solução e a mediana encontradas pela aplicação do algoritmo genético apresentado no Algoritmo 4, e as compara com o melhor valor de função objetivo encontrado na literatura, determinando-se, assim, os valores de $gap(\%)$. Esta Tabela mostra também a média dos valores de $gap(\%)$ encontrados. Observa-se que tanto os valores de $gap(\%)$ da melhor solução quanto os valores de $gap(\%)$ da mediana são menores que 1%. Esta é uma medida importante da eficiência e da precisão do algoritmo genético proposto em determinar as melhores soluções para esta classe de instâncias.

Tabela 5.4: $Gap(\%)$ em relação ao número de nós.

Nº de nós	100	200	300
gap(%) (Mediana)	0.19	0.83	1.29
gap(%) (Melhor)	0.00	0.12	0.42

Tabela 5.5: $Gap(\%)$ em relação ao número de *hubs*.

Nº de *hubs*	2	3	4	5	10	15	20	30	40
gap(%) (Mediana)	0.00	0.11	1.08	1.56	1.54	1.22	1.16	0.27	0.00
gap(%) (Melhor)	0.00	0.0	0.00	0.14	0.87	0.77	0.45	0.00	0.00

As Tabelas 5.4 e 5.5 mostram o $gap(\%)$ das instâncias em relação ao número de nós e ao número de *hubs*, respectivamente. Note que, à medida que o número de

Tabela 5.3: Resultados para UMApHCP – Solução via Algoritmo Genético – conjunto de instâncias URAND.

Instância	Melhor	Mediana	Melhor Lit	gap(%) Melhor	gap(%) Mediana
tm100.02	124922,34	124922,34	124922,34	0,00	0,00
tm100.03	114692	114692	114692,05	0,00	0,00
tm100.04	107478,75	107801,25	107478,64	0,00	0,30
tm100.05	105545,75	106056,25	105545,51	0,00	0,48
tm100.10	99144,25	99451,25	98558,78	0,59	0,91
tm100.15	97238,25	97238,25	97238,22	0,00	0,00
tm100.20	97238,25	97238,25	97238,22	0,00	0,00
tm100.30	97238,25	97238,25	97238,22	0,00	0,00
tm100.40	97238,25	97238,25	97238,22	0,00	0,00
tm200.02	130467	130467	130466,92	0,00	0,00
tm200.03	117735,45	117735,45	117735,45	0,00	0,00
tm200.04	111377,52	112509,5	111377,52	0,00	1,02
tm200.05	106820,5	109369,5	106820,05	0,00	2,39
tm200.10	103017	103803,75	102182,66	0,82	1,59
tm200.15	99680,25	100338,75	98558,78	1,14	1,81
tm200.20	98488,5	99634,25	98488,51	0,00	1,16
tm200.30	98488,5	98488,5	98488,51	0,00	0,00
tm200.40	98488,5	98488,5	98488,51	0,00	0,00
tm300.02	131341	131341	131341,06	0,00	0,00
tm300.03	120490,01	121409	120490,01	0,00	0,76
tm300.04	111880,71	114043	111880,71	0,00	1,93
tm300.05	108973,75	110491,75	108520,46	0,42	1,82
tm300.10	104161,25	105103,25	102920,26	1,21	2,12
tm300.15	101823,25	102511,25	100635,29	1,18	1,86
tm300.20	100151	101109,25	98827,2	1,34	2,31
tm300.30	98488,5	99299,75	98488,51	0,00	0,82
tm300.40	98488,5	98488,5	98488,51	0,00	0,00
Média	-	-	-	0,25	0,79

nós aumenta, o valor de $gap(\%)$ das melhores soluções também aumenta, conforme mostra a Tabela 5.4. Observe que, conforme a Tabela 5.5, não se verifica a mesma tendência ocorrida no caso do número de nós, pois, para soluções com $p = 40$, o algoritmo alcança as melhores soluções e o valor de $gap(\%)$ da mediana também é $gap(\%) = 0,00$.

5.4 Resultados para o UMApHCP via ILS_AMAM

Esta seção apresenta os resultados referentes à aplicação do método ILS_AMAM, apresentado na Seção 4.2 e formalizado no Algoritmo 8, para a solução de instâncias do UMApHCP. Foram realizados testes computacionais envolvendo os conjuntos de instâncias URAND, para o caso de 100 e 200 nós, bem como para os conjuntos de

instâncias AP, para o caso de 10, 20, 25, 40, 50, 100 e 200 nós. Os testes realizados envolveram a variação do número de agentes utilizados e a análise do impacto dessa variação na qualidade dos resultados.

5.4.1 Testes para instâncias URAND

Os primeiros testes foram realizados utilizando conjunto de instâncias URAND, que possuem maior número de nós, e foram geradas aleatoriamente, sendo introduzidas, no caso considerado, por Meyer et al. (2009). Os testes computacionais foram executados por 33 vezes em um computador *Intel Core I5* (8250U), 1,6 GHz (4 Cores 8 Threads), 8 GB de memória RAM e sistema operacional *Ubuntu 19.04*. O *Framework* `AMAM` é implementado em linguagem JAVA, utilizando IDE Eclipse (4.11.0). Os resultados computacionais obtidos são apresentados nas Tabelas 5.6 e 5.7 e na Figura 5.3.

A Tabela 5.6 apresenta o conjunto dos resultados obtidos em cada instância testada com número de agentes variando entre 1, 2, 4 e 8, mostrando o melhor, o pior e a média dos valores encontrados com o uso do *Framework* `AMAM` para estas configurações de agentes. Em todos os casos foi utilizado o mesmo agente ILS.

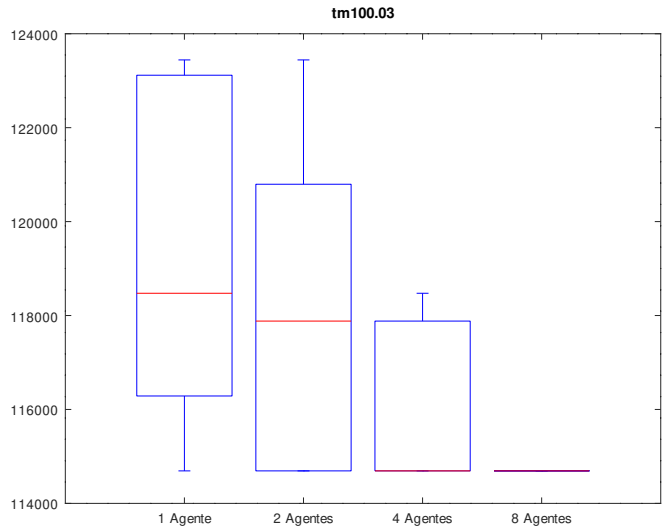


Figura 5.3: *Boxplot* dos resultados para a instância tm100.03, com 1, 2, 4 e 8 agentes.

A Figura 5.3 mostra o *boxplot* com os resultados obtidos na instância tm100.03, com a inserção de 1, 2, 4 e 8 agentes. Observa-se a melhoria na robustez do *framework* `AMAM`. Note que, à medida que insere-se novos agentes, tanto a mediana quanto a dispersão dos resultados ou se mantém ou diminuem.

A Tabela 5.7 compara os resultados obtidos pelo algoritmo BVNS apresentado em Brimberg et al. (2017a) e os melhores resultados encontrados com a utilização do *framework* `AMAM`, de modo a verificar sua eficiência. O *gap*% é calculado pela expressão:

$$gap(\%) = 100\frac{AMAM\_valor - BVNS\_valor}{BVNS\_valor} \tag{5.2}$$

Tabela 5.6: Resultados para conjunto de instâncias URAND, considerando a variação do número de agentes: 1, 2, 4 e 8 agentes.

Instância	Nº **Agentes**	**Melhor Valor**	**Pior Valor**	**Média**
tm100.02	1	124922.34	129660,00	127246,43
tm100.02	2	124922.34	126873,00	126266.31
tm100.02	4	124922.34	126477,00	125487.61
tm100.02	8	124922.34	124922.34	124922.34
tm100.03	1	114692.05	123422.75	119300.81
tm100.03	2	114692.05	123422.75	117943.06
tm100.03	4	114692.05	118474,00	115905.75
tm100.03	8	114692.05	114692,05	114692,05
tm100.04	1	107478.64	117016,00	111010.27
tm100.04	2	107478.64	114715.5	109616.45
tm100.04	4	107478.64	107478.64	107478.64
tm100.04	8	107478.64	107478.64	107478.64
tm100.05	1	105545.51	106056.25	105800.87
tm100.05	2	105545.51	106056.25	105800.87
tm100.05	4	105545.51	105545.51	105545.51
tm100.05	8	105545.51	105545.51	105545.51
tm200.02	1	130466.92	130466.92	130466.92
tm200.02	2	130466.92	130466.92	130466.92
tm200.02	4	130466.92	130466.92	130466.92
tm200.02	8	130466.92	130466.92	130466.92
tm200.03	1	117735.45	122081,00	120754.90
tm200.03	2	117735.45	122081,00	119301.70
tm200.03	4	117735.45	121456.5	118445.34
tm200.03	8	117735.45	120775,00	119255.22
tm200.04	1	111377.52	115251.25	112443.45
tm200.04	2	111377.52	112704,00	111584.86
tm200.04	4	111377.52	111377.52	111377.52
tm200.04	8	111377.52	111377.52	111377.52
tm200.05	1	106820.05	109052.75	107100.70
tm200.05	2	106820.05	109052.75	107023.43
tm200.05	4	106820.05	106820.05	106820.05
tm200.05	8	106820.05	106820.05	106820.05

Tabela 5.7: Comparação ILS_AMAM × BVNS

Instância	**ILS_AMAM**	**BVNS**	$gap(\%)$
tm100.02	124922.34	124922.34	0.00
tm100.03	114692.05	114692.05	0.00
tm100.04	107478.64	107478.64	0.00
tm100.05	105545.51	105545.51	0.00
tm200.02	130466.92	130466.92	0.00
tm200.03	117735.45	117735.45	0.00
tm200.04	111377.52	111377.52	0.00
tm200.05	106820.05	106820.05	0.00

Pelos resultados apresentados na Tabela 5.7, observa-se que o *framework* `AMAM` alcançou os melhores valores obtidos pelo BVNS em todas as instâncias apresentadas, sendo o $gap(\%)$=0.00 para todas elas. Com isso, conclui-se que o *framework* `AMAM` é eficiente em relação à busca pela solução do UMApHCP.

5.4.2 Testes para instâncias AP

A segunda parte dos testes computacionais para o método ILS_AMAM foi executada utilizando-se o conjunto de instâncias AP - (*Australian Post*), introduzido em Ernst e Krishnamoorthy (1996), que envolvem dados reais do Serviço Postal Australiano. As instâncias deste conjunto possuem até 200 nós. Os testes computacionais foram realizados em um computador *Intel(R) Xeon(R) Silver* 4110 CPU 2.10GHz, 16 núcleos, memória RAM de 32GB, sistema operacional *CentOS 6*, mantido pelo Laboratório de Computação de Alto Desempenho (LCAD) do CEFET-MG. Para cada número particular de agentes, separou-se o número de núcleos do processador igual ao número de agentes. Por exemplo, para o teste com 1 agente, separou-se 1 núcleo para processamento; para 2 agentes, separou-se 2 núcleos de processamento; e assim sucessivamente.

Assim como nos testes apresentados na Seção 5.2, usando o método exato via CPLEX, os testes computacionais executados com o modelo UMApHCP utilizam os seguintes fatores de desconto/penalidade no cálculo do custo de transporte: (i) $\gamma = 1,0$, $\alpha = 0,75$ e $\beta = 1,0$, ou seja, os mesmos fatores utilizados em Brimberg et al. (2017a); (ii) $\gamma = 3,0$, $\alpha = 0,75$ e $\beta = 2,0$, fatores extraídos do próprio conjunto de instâncias AP. Desta forma, γ e β são fatores de penalidade no fluxo do nó de origem para o nó *hub* de coleta e do nó *hub* de distribuição para o nó de destino, respectivamente. Já α é um fator de desconto do fluxo entre *hubs*. Os resultados computacionais obtidos são apresentados a seguir.

A Tabela 5.8 apresenta os resultados encontrados na execução do ILS_AMAM, utilizando as instâncias AP. A coluna "Instância" contém as instâncias AP no formato número de nós_*hubs*. No conjunto de colunas "Sem penalidade 1-1", são apresentados os resultados obtidos com a execução dos testes computacionais em instâncias sem penalidade de coleta e entrega $(1-1)$. As colunas são: "BVNS" melhores resultados apresentados em Brimberg et al. (2017a); "CPLEX" resultados obtidos com o *solver* CPLEX; e a coluna "ILS_AMAM" mostra os melhores resultados obtidos com o método ILS_AMAM. Em seguida, o conjunto de colunas "Com penalidade" apresenta os resultados encontrados com a execução dos testes com penalidade $(3-2)$. A coluna "CPLEX" apresenta os resultados obtidos com o *solver* CPLEX e a coluna "ILS_AMAM" apresenta os resultados dos referidos métodos.

A Tabela 5.9 apresenta os tempos computacionais encontrados na execução do método ILS_AMAM aplicado ao UMApHCP sem penalidade de coleta e entrega $(1-1)$. A coluna "Instância" apresenta a instância testada. Em seguida as colunas "1 Agente","2 Agentes","4 Agentes" e "8 Agentes" apresentam o tempo computacional médio obtido utilizando 1, 2, 4 e 8 agentes respectivamente.

A Tabela 5.10 apresenta o tempo computacional encontrado na execução do método ILS_AMAM aplicado ao UMApHCP com penalidade de coleta e entrega $(3-2)$. A coluna "Instância" apresenta a instância testada. As colunas "1 Agente","2 Agentes","4 Agentes" e "8 Agentes" apresentam o tempo computacional obtido utilizando

Tabela 5.8: Valor função Objetivo 1-1 e 3-2

	Sem penalidade 1-1			Com penalidade 3-2	
Instância	**BVNS**	**CPLEX**	**ILS_AMAM**	**CPLEX**	**ILS_AMAM**
AP10_2	39922,11	39922,11	39922,11	99805,28	99805,28
AP10_3	32713,94	32713,94	32713,94	70337,95	70336,95
AP10_4	31577,96	31577,96	31577,96	68714,17	68714,17
AP10_5	30371,32	30371,32	30371,32	55439,28	55439,28
AP20_2	45954,15	45954,15	45954,15	110220,25	110220,25
AP20_3	40909,59	40909,59	40909,59	92839,94	92839,94
AP20_4	38320,25	38320,25	38320,25	80901,66	80901,66
AP20_5	37868,15	37868,15	37868,15	74162,48	74162,48
AP20_10	37868,15	37868,15	37868,15	47794,95	47794,95
AP25_2	51533,30	51533,30	51533,30	117182,56	117182,56
AP25_3	45552,50	45552,50	45552,50	102737,89	102737,89
AP25_4	45552,50	45552,50	45552,50	88159,77	88159,77
AP25_5	45552,50	45552,50	45552,50	78173,77	78173,77
AP25_10	45552,50	45552,50	45552,50	53964,09	53964,09
AP40_2	61140,80	61140,80	61140,80	-	145245,30
AP40_3	56309,88	56309,88	56309,88	-	121326,15
AP40_4	51279,14	51279,14	51279,14	-	109959,32
AP40_5	49741,20	49741,20	49741,20	-	97860,40
AP40_10	49741,20	49741,20	49741,20	-	61827,73
AP50_2	61179,03	-	61179,03	-	149423,50
AP50_3	56729,94	-	56729,94	-	123595,23
AP50_4	52905,77	-	52905,77	-	109466,92
AP50_5	50707,87	-	50707,87	-	93574,70
AP50_10	50707,87	-	50707,87	-	68170,17
AP100_2	63197,10	-	63197,10	-	148384,41
AP100_3	57925,66	-	57925,66	-	123366,17
AP100_4	54704,51	-	54704,51	-	105724,08
AP100_5	53949,33	-	53949,33	-	99157,75
AP100_10	51860,03	-	51860,03	-	71533,02
AP100_20	51860,03	-	51860,03	-	58758,45
AP100_30	51860,03	-	51860,03	-	51860,03
AP100_40	51860,03	-	51860,03	-	51860,03
AP200_2	67083,28	-	67083,28	-	158283,44
AP200_3	62945,55	-	62945,55	-	133677,56
AP200_4	59420,93	-	59420,93	-	110099,97
AP200_5	57419,32	-	57419,32	-	104590,95
AP200_10	55958,75	-	55958,75	-	-
AP200_20	55958,75	-	55958,75	-	-
AP200_30	55958,75	-	55958,75	-	-
AP200_40	55958,75	-	55958,75	-	-

1, 2, 4 e 8 agentes respectivamente.

As Figuras 5.4(a) e 5.4(b) apresentam gráficos de caixa dos tempos computacionais

Tabela 5.9: Tempo computacional com diferentes números de agentes (1-1)

Instância	1 Agente	2 Agentes	4 Agentes	8 Agentes
AP10_2	0,00	0,00	0,00	0,00
AP10_3	0,01	0,01	0,01	0,00
AP10_4	0,03	0,02	0,04	0,02
AP10_5	0,02	0,01	0,03	0,02
AP20_2	0,01	0,01	0,02	0,01
AP20_3	0,03	0,02	0,02	0,02
AP20_4	0,03	0,02	0,01	0,01
AP20_5	0,04	0,01	0,01	0,01
AP20_10	0,02	0,02	0,02	0,01
AP25_2	0,02	0,01	0,02	0,01
AP25_3	0,06	0,03	0,03	0,02
AP25_4	0,03	0,02	0,02	0,01
AP25_5	0,04	0,03	0,02	0,02
AP25_10	0,04	0,04	0,03	0,03
AP40_2	3,53	3,54	3,54	2,94
AP40_3	0,29	0,11	0,08	0,07
AP40_4	0,26	0,21	0,12	0,10
AP40_5	0,15	0,14	0,12	0,10
AP40_10	0,22	0,21	0,13	0,14
AP50_2	7,67	8,00	7,58	6,49
AP50_3	1,75	1,78	1,81	1,53
AP50_4	4,27	4,29	4,29	3,70
AP50_5	5,70	5,70	5,80	5,24
AP50_10	0,51	0,53	0,43	0,34
AP100_2	20,36	13,96	9,66	3,65
AP100_3	4,16	2,08	1,93	1,60
AP100_4	6,22	2,71	2,47	1,85
AP100_5	18,15	7,84	3,24	1,51
AP100_10	9,56	9,54	9,80	9,01
AP100_20	13,07	12,97	13,66	12,23
AP100_30	18,54	19,76	19,80	17,47
AP100_40	24,43	24,98	26,22	23,44
AP200_2	10,51	10,25	9,49	8,87
AP200_3	27,85	24,88	26,35	24,36
AP200_4	59,76	51,82	48,47	51,33
AP200_5	114,77	152,46	83,23	52,82
AP200_10	42,99	56,58	58,54	51,88
AP200_20	54,94	62,28	102,49	87,55
AP200_30	72,84	76,85	139,06	123,52
AP200_40	98,28	98,94	149,73	168,74

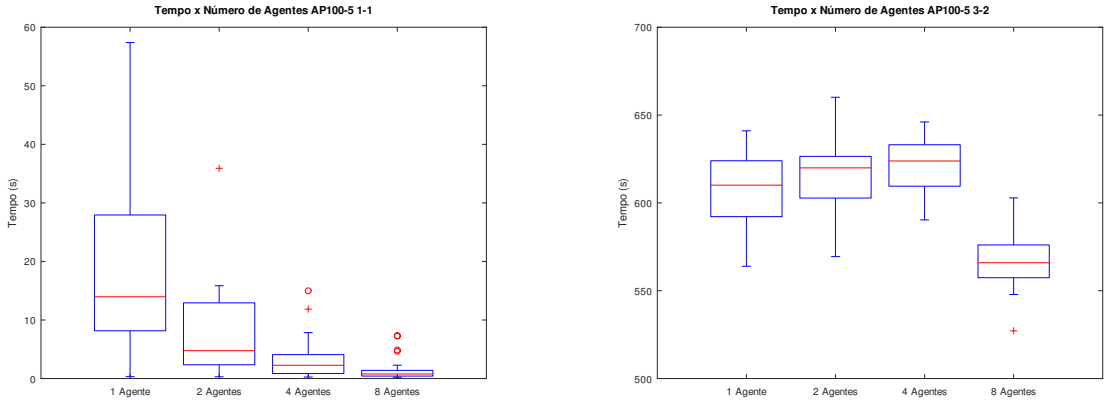
obtidos com a execução da instância AP100_5. Na Figura 5.4(a) os resultados foram obtidos com a execução das instância sem penalidade de coleta e entrega, e são apresentados os resultados por número de agentes. Observa a partir do gráfico que a

Tabela 5.10: Tempo computacional para diferentes números de agentes (3-2)

Instância	1 Agente	2 Agentes	4 Agentes	8 agentes
AP10_2	0,01	0,00	0,00	0,01
AP10_3	0,01	0,00	0,01	0,02
AP10_4	0,01	0,00	0,01	0,04
AP10_5	0,01	0,00	0,01	0,02
AP20_2	0,02	0,01	0,02	0,02
AP20_3	0,04	0,02	0,01	0,01
AP20_4	0,05	0,03	0,01	0,01
AP20_5	0,10	0,05	0,02	0,02
AP20_10	0,43	0,33	0,25	0,18
AP25_2	0,04	0,02	0,04	0,01
AP25_3	2,04	2,14	1,77	1,74
AP25_4	4,56	5,44	4,86	4,61
AP25_5	7,31	8,45	7,15	7,04
AP25_10	19,34	22,77	20,99	19,64
AP40_2	3,02	3,66	3,17	3,02
AP40_3	9,61	10,46	10,24	10,23
AP40_4	19,33	20,83	20,20	19,57
AP40_5	28,48	31,36	29,95	28,62
AP40_10	137,37	157,75	148,13	144,97
AP50_2	6,20	6,55	5,59	6,34
AP50_3	22,91	23,93	23,23	23,88
AP50_4	36,95	46,00	44,92	45,36
AP50_5	60,65	67,98	64,88	64,25
AP50_10	247,25	264,29	256,29	255,64
AP100_2	61,52	62,51	59,13	52,62
AP100_3	195,88	191,78	192,90	163,07
AP100_4	376,30	356,72	378,74	335,52
AP100_5	606,54	618,65	619,97	568,18
AP100_10	2683,69	2653,77	2443,23	2650,74
AP100_20	5815,20	5560,68	5872,02	4877,87
AP100_30	10236,45	9963,15	10750,83	9668,24
AP100_40	7965,56	7932,16	8459,93	8526,03
AP200_2	1016,23	395,06	301,98	492,23
AP200_3	1497,04	1049,06	883,73	1147,79
AP200_4	3151,31	2661,46	2291,25	2302,82
AP200_5	4583,32	3379,22	3213,88	3183,23

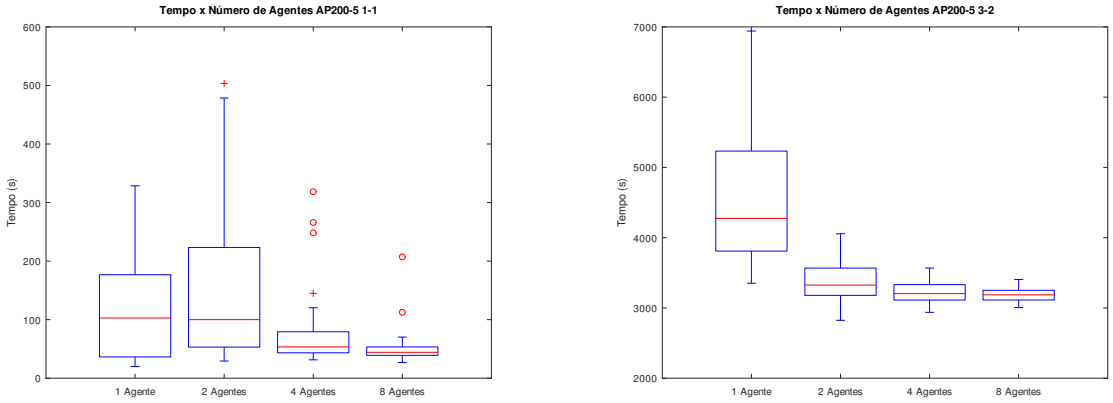
utilização de mais agentes torna o *Framework* `AMAM` mais rápido para a execução da referida instância. Já na Figura 5.4(b) os resultados são obtidos com as instâncias com penalidade de coleta e entrega $(3-2)$, nela podemos observar que a utilização de 8 agentes é mais eficiente que a utilização de 1, 2 e 4 agentes.

As Figuras 5.5(a) e 5.5(b) apresentam os tempos de execução computacional obtidos com a execução dos testes com a instância AP200_5 sem o uso de penalidades e com a aplicação de penalidades de coleta e entrega, respectivamente. A partir do



(a) 1-1 (b) 3-2

Figura 5.4: Tempo × Número de Agentes AP100_5



(a) 1-1 (b) 3-2

Figura 5.5: Tempo × Número de Agentes AP200_5

gráfico da Figura 5.5(a) nota-se que a utilização de 8 agentes torna o *Framwork* `AMAM` mais robusto em relação ao tempo de execução computacional. Da mesma forma, o gráfico da Figura 5.5(b) mostra que a utilização de maior número de agentes no *framework* torna-o mais rápido e robusto.

Capítulo 6

Agente VNS_AMAM Aplicado à solução do CMApHCP

Este capítulo descreve a solução do Problema *pHub* Centro Capacitado de Múltiplas Alocações (CMApHCP) usando o procedimento denominado VNS_AMAM. Este procedimento consiste em instanciar a metaheurística híbrida GRASP-VNS (*GRASP - Greedy Randomized Adaptive Search Procedure- Variable Neighborhood Search*) na forma de agente autônomo no *Framework* `AMAM`. Neste agente, a fase de construção da metaheurística GRASP é utilizada para construção de soluções iniciais. As seções a seguir apresentam todos os detalhes desta implementação.

6.1 Representação da Solução

Esta seção apresenta a representação de uma solução do Problema *pHub* Centro Capacitado de Múltiplas Alocações (CMApHCP) para a implementação a ser realizada no *Framework* `AMAM`.

Uma solução do CMApHCP é representada por um vetor Z, de dimensão p, e uma matriz M de valores inteiros, de dimensão $n \times n$, sendo $n = |N|$ o número de nós do grafo. Cada elemento do vetor Z representa o nó $n \in N$, que será determinado como nó *hub*. Desta forma, na representação de Z mostrada na Expressão (6.1), tem-se, para $n = 5$ nós e $p = 3$ *hubs*, os nós 1, 2 e 4 como nós *hub* e os nós 3 e 5 como nós não *hub*.

$$Z = \begin{bmatrix} 4 & 1 & 2 \end{bmatrix} \tag{6.1}$$

A matriz M, por sua vez, representa o caminho de todo o grafo com demanda origem-destino. Cada elemento (i, j) desta matriz recebe o arco que irá atender à demanda (i, j). Um exemplo desta representação é mostrado na Expressão (6.2).

$$M = \begin{bmatrix} 5 & 4 & 2 & 5 & 8 \\ 4 & 0 & 8 & 0 & 2 \\ 7 & 7 & 5 & 7 & 8 \\ 7 & 0 & 0 & 0 & 8 \\ 7 & 8 & 8 & 8 & 8 \end{bmatrix} \tag{6.2}$$

Observe que, para cumprir a demanda $(1, 3)$, o arco 2 é atribuído à demanda. Isto representaria o caminho $1 \rightarrow 4 \rightarrow 2 \rightarrow 3$. A definição dos arcos é feita a partir dos índices do vetor Z de *hubs*, que valem de 0 até $p-1$. Para que seja definido o numero de cada arco, utilizamos a formula $hub\_coleta \times p + hub\_distribuicao$, em que hub_coleta é o índice do vetor Z correspondente ao *hub* de coleta e $hub\_distribuicao$ é o índice do vetor Z correspondente ao *hub* de distribuição.

A Expressão (6.3) apresenta a numeração de arcos em um sistema com 3 *hubs*.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \end{array} \tag{6.3}$$

Observe que o arco composto pelo *hub* de coleta com índice 0 e *hub* de distribuição de índice 2 é nomeado como arco 2. Para se obter o caminho reverso, a partir do arco de transporte, deve-se definir os nós de coleta e distribuição e dividir o valor do arco pelo número p de *hubs*. Assim, determina-se o índice do *hub* de coleta; o resto da divisão do arco pelo número p de *hubs* determina o índice do *hub* de distribuição. Desta forma, definindo o *hub* de coleta e o *hub* de distribuição do arco 2, tem-se:

- **coleta:** $2 \div 3 = 0 \rightarrow Z_0 \rightarrow \text{nó} = 4$;
- **distribuição:** $2 \mod 3 = 2 \rightarrow Z_2 \rightarrow \text{nó} = 2$

6.2 Estrutura de Vizinhança

Para que o agente possa se movimentar no espaço de soluções, deve-se definir estruturas de vizinhança e movimentos associados ao problema. Assim, foram definidos dois módulos de estruturas de vizinhanças para o CMApHCP, um para tratar o vetor Z de *hubs* e outro para tratar a matriz de arcos de transporte M.

Para o primeiro módulo de estruturas de vizinhança, foram definidas duas estruturas de vizinhança. A primeira estrutura, denominada `swap_hub`, consiste em alterar um nó *hub* por um nó não *hub*. Por exemplo, uma solução vizinha do vetor Z, apresentado em 6.1, é dada por:

$$Z' = \begin{bmatrix} \mathbf{3} & 1 & 2 \end{bmatrix} \tag{6.4}$$

Observe que o nó 4 deixou de ser *hub* e o nó 3 é o novo *hub*. Os movimentos referentes a esta estrutura de vizinhança correspondem a alterar i *hubs* do vetor Z, em que i varia de 1 até $p-1$. A segunda estrutura de vizinhança para este módulo, denominada `shift_hub`, consiste em trocar os *hubs* de posição no vetor Z. Como exemplo, a Expressão (6.5) representa uma solução vizinha do vetor Z mostrado inicialmente na Expressão (6.1):

$$Z' = \begin{bmatrix} \mathbf{2} & 1 & \mathbf{4} \end{bmatrix} \tag{6.5}$$

Note que os *hubs* 2 e 4 foram alterados de posição. Foram definidos $p-1$ movimentos a partir desta estrutura de vizinhança, que consiste em realizar i trocas `shift_hub` na solução, para i variando de 1 até $p-1$.

Para o segundo módulo, foram definidas quatro estruturas de vizinhança. A primeira estrutura de vizinhança, nomeada `swap_arco`, consiste em alterar o arco de transporte de uma dada demanda. Para isso, deve-se alterar o arco por um novo arco. Por exemplo, para alterar o arco de entrega da demanda $(1,3)$, que atualmente é atendido pelo arco 2, devemos simplesmente definir um novo arco de uma solução vizinha a (6.2), na forma:

$$M = \begin{bmatrix} 5 & 4 & 2 & 5 & 8 \\ 4 & 0 & 8 & 0 & 2 \\ 7 & 7 & 5 & 7 & 8 \\ 7 & 0 & 0 & 0 & 8 \\ 7 & 8 & 8 & 8 & 8 \end{bmatrix} \xrightarrow{\texttt{swap\_arco}} M' = \begin{bmatrix} 5 & 4 & \mathbf{8} & 5 & 8 \\ 4 & 0 & 8 & 0 & 2 \\ 7 & 7 & 5 & 7 & 8 \\ 7 & 0 & 0 & 0 & 8 \\ 7 & 8 & 8 & 8 & 8 \end{bmatrix} \tag{6.6}$$

Note que a demanda $(1,3)$ deixou de ser atendida pelos *hubs* 4 e 2 (arco 2) e passou a ser atendida apenas pelo *hub* 2 (arco 8). As duas próximas estruturas de vizinhança são similares à estrutura de vizinhança `swap_arco`, porém os arcos são limitados à troca. O arco da demanda pode ser alterado apenas para os arcos em que apenas o *hub* de coleta e apenas o *hub* de distribuição pode ser alterado. A segunda estrutura, nomeada `swap_coleta`, consiste em alterar o *hub* de coleta de uma dada demanda. Para alterar o *hub* de coleta da demanda, devemos alterar o arco correspondente à demanda, de forma que o novo arco seja trocado apenas pelos arcos em que a linha da matriz de arcos corresponde à linha do *hub* de coleta do arco atual. Para definirmos a referida linha, efetuamos a operação $arco \div p$. Por exemplo, para trocar o arco de coleta da demanda $(1,3)$, atendida pelo arco 2, deve-se alterar o arco 2 pelos arcos contidos na linha 1 da matriz (6.3), que são eles os arcos 0 e 1.

Já a estrutura denominada `swap_distribuição` consiste em alterar o arco de transporte, de forma que apenas o *hub* de distribuição seja alterado. Para isso, devemos trocar o arco apenas pelos arcos contidos na coluna obtida pela operação $arco \mod p$. Por exemplo para alterar apenas o *hub* de distribuição da demanda $(1,3)$ atendida pelo arco 2, pode-se trocar o arco pelos arcos 5 e 8. Por fim, foi definida a estrutura de vizinhança `shift_arco`. Essa vizinhança consiste em trocar o arco de duas demandas do grafo. Como exemplo, a Expressão 6.7 representa uma solução vizinha à solução apresentada na Expressão 6.2, na forma:

$$M = \begin{bmatrix} 5 & 4 & 2 & 5 & 8 \\ 4 & 0 & 8 & 0 & 2 \\ 7 & 7 & 5 & 7 & 8 \\ 7 & 0 & 0 & 0 & 8 \\ 7 & 8 & 8 & 8 & 8 \end{bmatrix} \xrightarrow{\texttt{shift\_arco}} M = \begin{bmatrix} 5 & 4 & \mathbf{8} & 5 & 8 \\ 4 & 0 & 8 & 0 & 2 \\ 7 & 7 & 5 & 7 & 8 \\ 7 & 0 & 0 & 0 & 8 \\ 7 & 8 & \mathbf{2} & 8 & 8 \end{bmatrix} \tag{6.7}$$

Observe que os arcos das demandas $(1,3)$ e $(5,3)$ foram trocados. A partir das estruturas de vizinhança apresentadas, definiu-se os i movimentos a serem utilizados, i variando de 1 até $n/3$. Estes movimentos são:

(i) `i_swap_coleta` - movimento que consiste em realizar i alterações `swap_coleta`;

(ii) `i_swap_distribuição` - movimento que consiste em realizar i alterações usando `swap_distribuição`;

(iii) `i_swap_arco` - movimento que consiste em realizar i alterações `swap_arco`;

(iv) `i_shift_arco` - movimento que consiste em realizar i trocas `shift_arco`.

6.3 Função de Avaliação

A avaliação de uma solução *sol* é dada por:

$$f(sol) = \max\{C_{ij}\} + \\ + \max\left\{0, -\beta\left[\sum_{k=1}^{p}(cap_k - f\_hub_k) \ : \ (cap_k - f\_hub_k) < 0\right]\right\}, \ i,j \in N \qquad (6.8)$$

Este valor deve ser minimizado. A condição max $\{C_{ij}\}$ consiste em encontrar o maior custo, calculado dentre todas as demandas (i, j) da instância, somado ao maior fator entre zero (0) e a somatória da capacidade excedida dos *hubs*, multiplicada por um fator de penalidade β. Nesta expressão, f_hub_k representa o fluxo trafegado no *hub* k, segundo a forma de atribuição de capacidade, e o fator de penalização β é definido como o custo em percorrer todas as demandas (i, j).

6.4 Construção da Solução Inicial

No algoritmo proposto, a implementação da construção da solução inicial se dá por uma construção gulosa aleatória, semelhante à fase de construção inicial da metaheurística GRASP (Feo e Resende, 1995). São divididas em duas etapas: (i) constrói-se o vetor Z de *hubs*; (ii) os nós não *hubs* são alocados aos *hubs* de coleta e distribuição. Inicialmente, constrói-se uma lista restrita RCL_n de candidatos e, logo em seguida, atribui-se a cada nó t o fator custo beneficio obtido ao calcular a maior distância entre ele e todos os nós j divididos pela capacidade do nó t, definido pelo função guia $g(t)$ dada por:

$$g(t) = \max(d_{ti}) \div cap_t \quad \forall \ i \neq t, i \in C \qquad (6.9)$$

Nesta expressão, C corresponde aos nós candidatos a serem *hubs* na solução, d_{ti} é o custo de saída do nó t e chegada ao nó i e cap_t a capacidade do nó t. Logo após, define-se, aleatoriamente, p *hubs*, condicionados ao melhor valor de custo beneficio encontrado na RCL_n. A segunda etapa consiste em alocar os nós não *hubs* aos *hubs*. Para isso é utilizado o algoritmo de caminho mínimo de Floyd-Warshall, apresentado na Seção 4.1.3. O algoritmo para construção da solução inicial é semelhante ao Algoritmo 6, excetuando-se a parcela referente ao cálculo da função de custo benefício, ao criar a lista restrita de candidatos. O Algoritmo 9 formaliza essa proposta.

Algoritmo 9 Construção Gulosa Aleatória

Entrada: p, N, $dist_{n\times n}$, cap_n, α
Saída: sol
$sol \leftarrow \emptyset$
$ins \leftarrow 1$
$C \leftarrow N$
repita
  $g(t_{\min}) = \min\{g(t) \mid t \in C\}$
  $g(t_{\max}) = \max\{g(t) \mid t \in C\}$
  $RCL = \{t \in C \mid g(t) \leq g(t_{\min}) + per(g(t_{\max}) - g(t_{\min}))\}$
  Aleatoriamente, selecione um elemento $t \in RCL$
  $sol \leftarrow sol \cup \{t\}$
  $ins \leftarrow ins + 1$
  $C \leftarrow C \backslash \{t\}$
até $ins = p$;
$sol \leftarrow$ Adaptação de Floyd-Warshall$(C_{n\times n}, n, p, Z_p)$
retorna sol

Na linha 6.4 do Algoritmo 9 criada a Lista Restrita de candidatos (RCL), considerando o fator *per* de itens em relação ao número de nós do problema. Na linha 6.4, um item da lista é selecionado aleatoriamente. Esse procedimento é repetido por p vezes, conforme linha 6.4. Ao final da seleção dos *hubs*, o algoritmo de caminho mínimo é chamado, conforme mostra a linha 6.4.

6.5 Busca Local

A técnica de busca local implementada pelo VNS_AMAM consiste no método VND (*Variable Neighborhood Descend*). Este método consiste em encontrar ótimos locais para um número n de estruturas de vizinhança. Seguindo uma determinada ordem de vizinhança, partindo de uma solução corrente, é gerado um vizinho de uma estrutura de vizinhança e aplicado o método de busca local. Caso a solução gerada seja melhor que a atual, ele passara a ser a solução corrente e volta-se à primeira estrutura de vizinhança. Caso contrário, a busca local avança para a estrutura de vizinhança seguinte. Isso ocorre seguidamente até que as n estruturas de vizinhanças não consigam melhorar a solução corrente.

Para a resolução do CMApHCP foi implementada a heurística do melhor vizinho como técnica de busca local. A busca local é dada por uma exploração da vizinhança da solução e, ao final de cada etapa retorna o melhor vizinho daquela solução. Caso esta solução seja aceita, passa a ser a solução corrente do método. Esse procedimento é repetido enquanto houver melhoria na solução corrente. A busca local implementada é mostrada no Algoritmo 10.

Algoritmo 10 Busca Local VND

Entrada: $sol\_corrente$
Saída: $melhor\_sol$
$melhor\_sol \leftarrow sol\_corrente$
$k \leftarrow 1$
repita
  Encontre o melhor vizinho $s' \in V_k\ (s)$
  se $f(sol') < f(melhor\_sol)$ **então**
    $melhor\_sol \leftarrow sol'$
    $k \leftarrow 1$
  senão
    $k \leftarrow k + 1$
  fim
até $k <= |V|$;
retorna $melhor\_sol$

No procedimento mostrado pelo Algoritmo 10, a solução corrente é refinada para encontrar um ótimo local em relação a $|V|$ estruturas de vizinhança (linha 6.5). Note que a linha 6.5 gera um novo vizinho em relação à estrutura de vizinhança k, e sua função é avaliada de forma a verificar se tal vizinho é melhor que a solução corrente. Caso seja, a solução corrente passa a ser a solução s'. Caso contrário, k passa a valer $k+1$ (linha 6.5). A busca local deste algoritmo utiliza as estruturas de vizinhança referentes ao segundo módulo de estruturas apresentados na Seção 6.2. A ordem de utilização destas estruturas é: (i) `swap_arco`; (ii)`swap_coleta`; (iii)`swap_distribuição` e; (iv) `shift_arco`. Definida, a partir da complexidade das estruturas de vizinhança. Esse processo é utilizado sucessivamente até encontrar um ótimo local em relação às $|V|$ estruturas de vizinhança.

6.5.0.1 Aprendizado do Agente

O *Framawork* `AMAM` possui a possibilidade do agente aprender de forma autônoma. Para isso, o agente VNS_AMAM implementa um algoritmo de aprendizado por reforço (RL), i.e, o Algoritmo *Q-learning* apresentado na Seção 3.4.1. O aprendizado é utilizado para definir a sequência de estruturas de vizinhança a se utilizar durante a busca local VND. Desta forma, não seria necessário um conhecimento maior acerca da complexidade de cada estrutura de vizinhança.

6.6 Agente VNS_AMAM

O método implementado, nesta proposta, pelo agente do *Framework*, é a metaheurística híbrida constituída pela metaheurística VNS (*Variable Neighborhood Search*), tendo a construção da solução inicial fundada na fase inicial da metaheurística GRASP (*Greedy Randomized Adaptative Search Procedure*) conforme apresentado na Seção 6.4.

O VNS é um método de busca local (Mladenović e Hansen, 1997), que se utiliza de uma sequência de vizinhanças, $V_1, \ldots, V_k$, para tentar escapar de ótimos locais ou pesquisar diferentes ótimos locais. Mladenović e Hansen (1997) apresentam três fatores que o VNS explora: (i) o ótimo local de uma vizinhança não é necessariamente o ótimo local de outra estrutura de vizinhança; (ii) um ótimo global é um ótimo local para k estruturas de vizinhanças; (iii) em muitos problemas de otimização, ótimos locais de diferentes estruturas de vizinhanças estão próximos entre si. Basicamente, o VNS é composto por quatro módulos: (i) geração da solução inicial; (ii) busca local; (iii) estruturas de vizinhanças; e (iv) critério de aceitação. O processo do VNS consiste em criar uma solução inicial; aplica-se uma busca local, retornando um ótimo local e tomando esta solução como a solução corrente.

A partir daí, enquanto o critério de aceitação não é alcançado, gera-se um vizinho (s') a partir da estrutura de vizinhança V_k. Em seguida, novamente, é aplicada a busca local no vizinho s'. Caso a solução seja melhor que a solução corrente, a solução corrente passa a ser s' e a estrutura de vizinhança selecionada é $k = 1$. Se o vizinho não for melhor que a solução corrente, a vizinhança passa a valer $k + 1$. Este processo é repetido até $k = |V|$. Ao final do processo, retorna-se um ótimo local em relação a $k = |V|$ estruturas de vizinhanças.

Algoritmo 11 *Variable Neighborhood Search*

Entrada: $iter_{\max}$
Saída: sol^*
$sol^0 \leftarrow$ Gerar Solução Inicial
$sol^* \leftarrow$ Busca Local(sol^0)
$iter \leftarrow 1$
repita
  $k \leftarrow 1$
  repita
    $sol' \leftarrow$ Gerar um vizinho qualquer (sol^*, V_k)
    $sol'' \leftarrow$ Busca Local(sol')
    se $f(sol'') < f(sol^*)$ **então**
      $sol^* \leftarrow sol''$
      $k \leftarrow 1$
    senão
      $k \leftarrow nivel + 1$
    fim
  até $k <= |V|$;
  $iter \leftarrow iter + 1$
até $iter = iter_{\max}$;
retorna sol^*

O Algoritmo 11 indica o procedimento da metaheurística VNS. Os parâmetros do método são $nivel_{\max}$ e as estruturas de vizinhança V adotadas. Na linha 6.6 são gerados vizinhos, a partir da estrutura de vizinhança V_k. A estruturas de vizinhança responsável pela geração dos vizinhos são as estruturas `swap_hub` e `shift_hub`, apresentadas na Seção 6.2. Além das duas estruturas de vizinhança, uma terceira vizinhança corresponde a coletar uma solução no *pool* de soluções do *Framework*.

Capítulo 7

Resultados CMApHCP

Este Capítulo apresenta os resultados computacionais obtidos com os experimentos para a solução do CMApHCP. Inicialmente, a Seção 7.1 apresenta a metodologia utilizada para a realização dos testes computacionais. A Seção 7.2 apresenta os resultados obtidos com o uso do *solver* CPLEX. Na Seção 7.2.1 são apresentados os resultados obtidos com os experimentos de capacidade dupla. Em seguida, na Seção 7.2.2, os resultados com capacidade única são apresentados. Por fim, a Seção 5.4 contém os resultados alcançados com o Agente VNS_AMAM no *framework* `AMAAM`.

7.1 Metodologia para os testes para o CMApHCP

A metodologia adotada para a realização dos testes computacionais é centrada na comparação entre os resultados alcançados para a solução, usando o *solver* CPLEX, dos modelos de programação matemática de capacidade única e capacidade dupla para o problema CMApHCP e os resultados alcançados na solução destes mesmos modelos usando a metaheurística VNS, instanciada como um agente no *Framework* `AMAM`. Para realização destes testes foi utilizado um conjunto de instâncias da literatura para o problema.

7.2 Resultado do CMApHCP usando *solver* CPLEX

Para validar os modelos matemáticos do CMApHCP apresentados na Seção 2.2.2, foi desenvolvida uma modelagem de programação matemática. O modelo de programação matemática foi desenvolvido em linguagem ILOG Concert Technology `C++`, usando-se o *solver* CPLEX versão acadêmica 12.9 e o editor de texto `Gedit` - version 3.36.1. Os testes computacionais foram realizados em um computador *Intel(R) Xeon(R) Silver* 4110 CPU 2.10GHz, 16 núcleos, memória RAM de 32GB, sistema operacional *CentOS 6*, mantido pelo Laboratório de Computação de Alto Desempenho (LCAD) do CEFET-MG. Para realização dos experimentos computacionais do CMApHCP utilizou-se o conjunto de instâncias AP - (*Australian Post*), introduzido em Ernst e Krishnamoorthy (1996), que apresenta dados reais do Serviço Postal Australiano. As instâncias deste conjunto possuem 200 nós. O limite de tempo de execução de 7200 segundos foi definido para a execução de cada instância e os resultados computacionais obtidos com as duas variantes do problema capacitado são

Tabela 7.1: Resultados para o CMApHCP com Capacidade Dupla - Instâncias L

Instância	Solução	Valor	Gap(%)	Tempo(s)
AP10_2	Ótima	99805,28	0	41,03
AP10_3	Ótima	70337,49	0	35,26
AP10_4	Ótima	68714,17	0	47,83
AP10_5	Ótima	55439,28	0	54,17
AP20_2	Ótima	110220,25	0	194,71
AP20_3	Ótima	92839,94	0	485,74
AP20_4	Ótima	82439,73	0	573,87
AP20_5	Ótima	74162,48	0	366,64
AP20_10	Ótima	47794,95	0	158,83
AP25_2	Ótima	118497,02	0	3739,42
AP25_3	Ótima	102737,89	0	1296,31
AP25_4	Ótima	89747,25	0	2391,88
AP25_5	Ótima	82234,52	0	3116,21
AP25_10	Ótima	53964,09	0	1121,43
AP40_2	Factível	117685,42	0,54	7166,63
AP40_3	-	-	-	-
AP40_4	Factível	81679,74	0,71	7175,25
AP40_5	Factível	76660,71	0,69	7173,59
AP40_10	Factível	53731,60	0,80	7193,82
AP50_2	-	-	-	-
AP50_3	-	-	-	-
AP50_4	-	-	-	-
AP50_5	Factível	74019,30	0,67	7307,48
AP50_10	Factível	54078,20	0,84	7145,97

apresentados a seguir.

7.2.1 Resultados do CMApHCP com Capacidade Dupla

Os testes computacionais executados com o modelo CMApHCP de capacidade dupla usam os fatores de desconto/penalidade adotados no cálculo do custo de transporte: $\gamma = 3,0$; $\alpha = 0,75$; e $\beta = 2,0$. Desta forma, γ e β são os fatores de penalidade no fluxo do nó de origem para o nó *hub* de coleta e do nó *hub* de distribuição para o nó de destino, respectivamente, enquanto α é o fator de desconto do fluxo entre *hubs*. Os resultados obtidos são apresentados nas Tabelas 7.1 e 7.2.

As Tabelas 7.1 e 7.2 apresentam os resultados obtidos na execução dos testes da proposta CMApHCP com dupla capacidade, utilizando as instâncias AP com restrições de capacidade `L` e `T`, respectivamente. A coluna "Solução" indica se foi encontrada uma solução e se esta solução é ótima ou não. A coluna "Valor" apresenta o valor da solução, caso ela seja encontrada. A coluna *gap*(%) mostra o *gap* relativo entre os melhores limitantes primal e dual. Por fim, a coluna "Tempo" apresenta o tempo computacional obtido.

Tabela 7.2: Resultados dos testes computacionais para CMApHCP com Capacidade Dupla - Instâncias T

Instância	Solução	Valor	Gap(%)	Tempo(s)
AP10_2	Ótima	115233,10	0	4,86
AP10_3	Ótima	78517,15	0	3,31
AP10_4	Ótima	70337,49	0	3,69
AP10_5	Ótima	59730,78	0	5,09
AP20_2	Ótima	129086,95	0	1413,59
AP20_3	Ótima	99412,48	0	1165,57
AP20_4	Ótima	84492,95	0	1776,33
AP20_5	Ótima	75759,94	0	1839,89
AP20_10	Ótima	47794,95	0	503,91
AP25_2	-	-	-	-
AP25_3	Ótima	117182,56	0	5092,78
AP25_4	Ótima	104375,53	0	5609,71
AP25_5	Ótima	82672,45	0	5720,07
AP25_10	Ótima	54960,00	0	5603,72
AP40_2	-	-	-	-
AP40_3	-	-	-	-
AP40_4	Factível	81674,48	0,63	7172,06
AP40_5	Factível	72765,30	0,61	7165,80
AP40_10	Factível	51487,88	0,82	7174,88
AP50_2	-	-	-	-
AP50_3	-	-	-	-
AP50_4	-	-	-	-
AP50_5	-	-	-	-
AP50_10	Factível	54078,20	0,80	7154,36

Analisando-se as tabelas é possível observar que:

(i) nos resultados do grupo `L`, as instâncias `AP40_3T`, `AP50_2T`, `AP50_3T` e `AP50_4T` não alcançaram nenhuma solução factível, enquanto que as instâncias `AP40_2T`, `AP40_4T`, `AP40_5T`, `AP40_10T`, `AP50_5T` e `AP50_10T` encontraram ao menos uma solução factível e as demais instâncias apresentaram resultados ótimos;

(ii) nos resultados do grupo `T`, as instâncias `AP25_2T`, `AP40_2T`, `AP40_3T`, `AP50_2T`, `AP50_3T`, `AP50_4T` e `AP50_5T` não apresentaram nenhuma solução factível, enquanto que as instâncias `AP40_4T`,`AP40_5T`, `AP40_10T` e `AP50_10T` apresentam soluções factíveis e o restante das instâncias apresentaram resultados ótimos.

7.2.2 Resultados do CMApHCP com Capacidade Única

Testes computacionais também foram executados com o modelo de CMApHCP de capacidade única. Os valores de desconto/penalidade são os mesmos utilizados na Seção 7.2.1, sendo $\gamma = 3;0$; $\alpha = 0,75$; e $\beta = 2,0$.

Tabela 7.3: Resultados para o CMA*p*HCP com Capacidade Única - Instâncias L

Instância	**Solução**	**Valor**	***gap*(%)**	**Tempo(s)**
AP10_2	Ótima	99805,28	0	38,70
AP10_3	Ótima	70337,49	0	37,55
AP10_4	Ótima	68714,17	0	54,71
AP10_5	Ótima	55439,28	0	61,18
AP20_2	Ótima	110220,25	0	108,62
AP20_3	Ótima	92839,94	0	177,70
AP20_4	Ótima	80901,66	0	294,70
AP20_5	Ótima	74162,48	0	143,24
AP20_10	Ótima	47794,95	0	268,19
AP25_2	Ótima	117182,56	0	1460,14
AP25_3	Ótima	102737,89	0	546,28
AP25_4	Ótima	89747,25	0	1313,98
AP25_5	Ótima	78173,77	0	911,01
AP25_10	Ótima	53964,09	0	1297,88
AP40_2	Factível	117685,42	0,54	7179,39
AP40_3	Factível	94694,81	0,56	7176,13
AP40_4	Factível	82434,24	0,67	7173,71
AP40_5	Factível	72768,26	0,70	7173,26
AP40_10	Factível	54036,97	0,78	7169,83
AP50_2	-	-	-	-
AP50_3	Factível	95698,52	0,65	7082,42
AP50_4	Factível	82415,45	0,69	7123,18
AP50_5	Factível	74020,16	0,70	7126,53
AP50_10	Factível	54078,20	0,84	7118,15

As Tabelas 7.3 e 7.4 apresentam os resultados encontrados na execução dos testes da proposta CMA*p*HCP com capacidade única, utilizando as instâncias AP com restrições de capacidade `L` e `T`, respectivamente. A coluna "Solução" indica se foi encontrada uma solução e se esta é ótima ou não. A coluna "Valor" apresenta o valor da solução. Já a coluna GAP(%) mostra o *gap* relativo entre os melhores limitantes primal e dual. Por fim, a coluna "Tempo" apresenta o tempo computacional obtido.

A partir destas tabelas, é possível observar que:

(i) nos resultados do grupo `L`, não foi obtido soluções ótimas na instâncias com 40 e 50 nós, sendo que, apenas na instância `AP50_2T` não foi encontrado nenhuma solução factível;

(ii) nos resultados do grupo `T`, as instâncias `AP25_2T`, `AP40_2T` `AP40_3T`, `AP40_4T`, `AP50_2T`, `AP50_3T` e `AP50_4T` não apresentaram nenhuma solução factível, as instâncias `AP25_4T`, `AP25_5T`, `AP40_5T`, `AP40_10T`, `AP50_5T` e `AP50_10T` apresentaram soluções factíveis e o restante das instâncias apresentaram resultados ótimos.

Tabela 7.4: Resultados para o CMA*p*HCP com Capacidade Única - Instâncias T

Instância	**Solução**	**Valor**	***gap*(%)**	**Tempo(s)**
AP10_2	Ótima	106375,56	0	14,94
AP10_3	Ótima	78517,15	0	12,97
AP10_4	Ótima	68714,17	0	4,92
AP10_5	Ótima	55439,28	0	3,97
AP20_2	Ótima	129086,95	0	1786,39
AP20_3	Ótima	99412,48	0	1417,30
AP20_4	Ótima	84492,95	0	1847,00
AP20_5	Ótima	75759,94	0	1631,30
AP20_10	Ótima	47794,95	0	548,92
AP25_2	-	-	-	-
AP25_3	Ótima	117182,56	0	5563,85
AP25_4	Factível	71454,06	0,38	7198,70
AP25_5	Factível	61997,08	0,31	7206,90
AP25_10	Ótima	54757,94	0	3327,53
AP40_2	-	-	-	-
AP40_3	-	-	-	-
AP40_4	-	-	-	-
AP40_5	Factível	72765,30	0,55	7128,07
AP40_10	Factível	52816,03	0,81	7137,68
AP50_2	-	-	-	-
AP50_3	-	-	-	-
AP50_4	-	-	-	-
AP50_5	Factível	74019,30	0,72	7132,38
AP50_10	Factível	54078,20	0,80	7139,90

7.2.3 Considerações a respeito dos Modelos

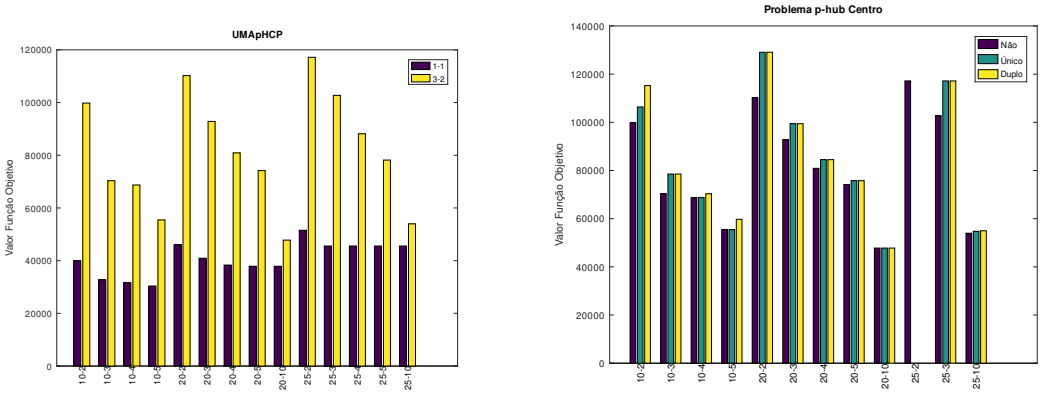
Algumas particularidades dos modelos analisados neste Capítulo devem ser destacadas:

(i) no modelo não capacitado, a inclusão de fatores de penalidade de coleta e de entrega torna-o mais difícil de se solucionar;

(ii) os modelos com restrições de capacidade são mais difíceis de serem solucionados, assim como o modelo de capacidade dupla possui maior dificuldade de solução que o de capacidade única;

(iii) seguindo o mesmo raciocínio, o conjunto de instâncias `T` é mais difícil de solucionar.

A Figura 7.1(a) apresenta o comportamento do valor das soluções obtidas com o modelo não capacitado utilizando fatores de desconto/penalidade 1-1 e 3-2. Deve-se observar que o modelo com fatores nulos (1-1) apresenta valores superiores aos que apresentam fatores de penalidade (3-2).



(a) Não Capacitado 1-1 × 3-2 (b) Não Capacitado × Única × Dupla

Figura 7.1: Considerações dos Modelos *p*HCP.

Já a Figura 7.1(b) compara os resultados obtidos nos 3 modelos. Observa-se que apresentam comportamentos similares em relação ao valor da função objetivo. Porém, o problema não capacitado sempre apresenta valor de função objetivo menor ou igual aos modelos capacitados. Observa-se também que, conforme se aumente o número de *hubs*, o valor da função objetivo melhora.

7.3 Resultados para o CMA*p*HCP usando o Agente VNS_AMAM

Para validar o método VNS_AMAM foram realizados testes computacionais. Para realização destes experimentos computacionais, utilizou-se o conjunto de instâncias AP - (*Australian Post*), já descrito anteriormente. As instâncias deste conjunto possuem 200 nós. Todos os testes computacionais realizados foram executados por 33 vezes em um computador *Intel Core I5* (8250U), 1,6 GHz (4 Cores 8 *Threads*), 8 GB de memória RAM e sistema operacional *Ubuntu 19.04*.

Os testes foram realizados com instâncias tendo até 25 nós, em que os resultados exatos obtiveram soluções ótimas, e a forma de inclusão de capacidade utilizada foi a capacidade dupla. Os fatores de desconto/penalidade adotados no cálculo do custo de transporte foram: $\gamma = 3,0$; $\alpha = 0,75$; e $\beta = 2,0$ e o conjunto de instâncias utilizado foi o conjunto de instâncias `L` (menos apertados) e o critério de parada foi o tempo computacional de 7200(s). Os resultados computacionais obtidos são apresentados na Tabela 7.5.

Tabela 7.5: Resultado com diferentes números de agentes

Instância	Concert CPLEX	VNS AMAM	*gap*% FO	Tempo (s) AMAM
AP10_2	99805,28	99805,28	0,00	35,30
AP10_3	70337,49	70337,49	0,00	33,70
AP10_4	68714,17	68714,17	0,00	40,12
AP10_5	55439,28	55439,28	0,00	53,81
AP20_2	110220,25	110220,25	0,00	200,00
AP20_3	92839,94	92839,94	0,00	501,98
AP20_4	82439,73	82439,73	0,00	520,31
AP20_5	74162,48	74162,48	0,00	303,76
AP20_10	47794,95	47794,95	0,00	123,54
AP25_2	118497,02	118497,02	0,00	3624,31
AP25_3	102737,89	102737,89	0,00	987,45
AP25_4	89747,25	-	-	7200,00
AP25_5	82234,52	-	-	7200,00
AP25_10	53964,09	53964,09	0,00	997,63

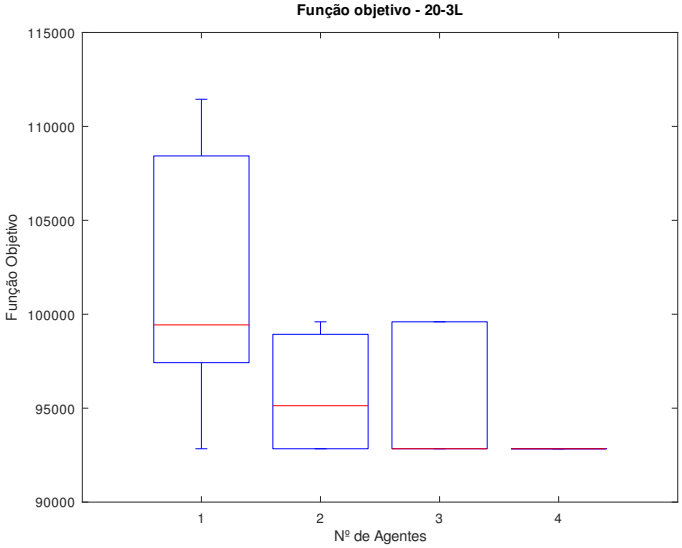


Figura 7.2: Boxplot dos resultados para a instância AP20_3L, com 1, 2, 4 e 8 agentes.

A Tabela 7.5 apresenta os resultados obtidos na execução do método VNS_AMAM aplicado ao CMA*p*HCP com penalidade de coleta e entrega (3-2). A coluna "Instância" apresenta a instância testada. Em seguida as colunas "Concert_CPLEX" e "VNS_AMAM" apresentam os resultados obtidos com os métodos exato e heurístico para o CMA*p*HCP, respectivamente. A Coluna "*gap* %" apresenta o *gap* da solução heurística em relação a solução exata. Já a coluna e "Tempo(s) AMAM" apresenta o tempo computacional obtido utilizando o método VNS_AMAM.

A partir dos resultados observa-se que o método VNS_AMAM obtém os resultados ótimos nas instâncias com menor numero de nós, porém o tempo computacional não é satisfatório. Além disso, observa-se que, nas instâncias `AP25_4` e `AP25_5`, não

foi encontrado solução factível com o tempo computacional estabelecido. Pode se observar também que as instâncias que não apresentaram soluções ótimas são aquelas que diferem seu resultado dos testes computacionais do o UMApHCP (Tabela 5.8). Com isso, destaca a necessidade de novas técnicas para a solução do CMApHCP.

A Figura 7.2 apresenta os resultados obtidos com 30 execuções da instância `AP20_3L`. Observa-se que, apesar do tempo computacional não apresentar bons resultados, escalabilidade torna os resultados mais robustos, Podemos observar que em todas as execuções com 8 agentes obteve-se a solução ótima.

Capítulo 8

Considerações Finais e Direções Futuras

8.1 Considerações Finais

A presente dissertação apresenta uma revisão da literatura em relação aos principais temas contido neste projeto. Apresentando lacunas e direções sobre o problema. Propostas de resolução para variantes do Problema *pHub* Centro. Modelos exatos e heurísticos foram desenvolvidos e alguns resultados computacionais obtidos, são apresentados. O resultados e a publicações derivadas deste projeto sustentam a direção de pesquisa do projeto. No fim desta dissertação, as publicações derivadas deste projeto e as propostas de continuidade são apresentadas.

Inicialmente, pelo levantamento bibliográfico, identificou-se que o Problema *pHub* Centro é um problema de interesse de pesquisadores da área e comprovar sua complexidade como NP-Difícil. Esta dissertação apresenta ainda modelos matemáticos para variantes do Problema *pHub* Centro: (i) variante que não apresenta restrições de capacidade, denominada UMA*p*HCP; (ii) duas variantes do CMA*p*HCP, que apresentam formas de capacidade diferentes, sendo elas capacidade dupla (coleta e distribuição) e capacidade única (somente coleta). Em seguida, afim de encontrar ferramentas para solucionar o problemas de otimização, observou-se que *frameworks* para otimização que utilizam metaheurísticas são excelentes ferramentas para implementação de metaheurísticas e metahaheurísticas hibridas. Isto se da pela facilidade em implementar as mesmas utilizando-se de códigos já estruturados. A partir da comparação de *frameworks* apresentada, optou-se pela utilização do *Framework* `AMAM`. Isto se deu pelo fato do mesmo tratar uma gama maior de problemas de otimização e apresentar algoritmos mais consolidados na literatura sobre aprendizado por reforço e compartilhamento de soluções.

Por fim, são apresentados as técnicas de solução para o problema. Primeiro, os modelos matemáticos foram solucionados de forma exata utilizando o *solver* CPLEX. Os experimentos computacionais utilizaram um grupo de instâncias disponíveis na literatura. Os resultados obtidos foram capazes de validar os modelos matemáticos e comprovar sua complexidade de resolução. Entretanto, resolver o problema de forma exata tem alto custo computacional. Do mesmo modo, observa-se que encontrar soluções ótimas, em tempo hábil, para instâncias acima de 25 nós, não é uma tarefa trivial. Observa-se também as diferenças entre os modelos apresentados, em relação às

soluções obtidas e ao tempo de execução. Os modelos com restrições de capacidades se mostram mais difíceis de serem solucionados.

Com isso modelos heurísticos foram desenvolvidos para solucionar as variantes não capacitada e capacitada do problema. Os resultados obtidos para o problema não capacitado foram satisfatórios e comprovam eficiência do *Framework* `AMAM`. Além de provar que a utilização de sistemas multiagente é eficaz na resolução do problema. Os testes utilizando sistemas multiagentes foram mais robustos e em tese mais rápidos. Os resultados obtidos com o problema capacitado comprovam também que a utilização de sistemas multiagentes são eficientes. Porém para solucionar o problema capacitado de forma efetiva é necessário que sejam estudadas as técnicas heurísticas desenvolvidas e caso necessário a criação de novas técnicas para solução.

8.2 Publicações Derivadas Desta Pesquisa

Nesta seção é listada as publicações originadas desta pesquisa.

1. Título: **Arquitetura Híbrida Multiagente Aplicada ao Problema pHub Centro Não Capacitado De Múltiplas Alocações.**
   - Autores: Jardell F. da Silva, Maria Amélia L. Silva, Sérgio R. de Souza e Marcone J. F. Souza
   - Evento: 2019 LI Simpósio Brasileiro de Pesquisa Operacional (SBPO 2019)
   - Local: Limeira, SP, Brasil
   - Período: 02 a 05 de setembro de 2019
   - Trabalho completo apresentado em evento
2. Título: **Algoritmo Genético Aplicado à Solução do Problema p-hub Centro Não Capacitado de Múltiplas Alocações.**
   - Autores: Jardell F. da Silva, Flávio V. C. Martins, Maria Amélia L. Silva e Sérgio R. de Souza.
   - Evento: 2019 XVI Encontro Nacional de Inteligência Artificial e Computacional (ENIAC 2019)
   - Local: Salvador, BA, Brasil
   - Período: 15 a 18 de outubro de 2019
   - Trabalho completo apresentado em evento
3. Título: **Uma abordagem exata do problema p-hub Centro.**
   - Autores: Jardell F. da Silva, Maria Amélia L. Silva, Elisangela M. de Sá, Sérgio R. de Souza e Marcone J. F. Souza.
   - Evento: 2020 LII Simpósio Brasileiro de Pesquisa Operacional (SBPO 2020)
   - Local: João Pessoa, PB, Brasil (online)
   - Período: 03 a 05 de novembro de 2020
   - Trabalho completo apresentado em evento

8.3 Direções Futuras

Esta dissertação apresentou o Problema *pHub* Centro e algumas de suas variantes e as solucionou de forma exata e heurística. Foram encontrados resultados satisfatórios, porém, ainda existem lacunas e avanços a serem conquistados. Alguns direções futuras são apresentadas a seguir:

- Realizar levantamento bibliográfico mais detalhado de todos os temas associados a dissertação: *Frameworks* para Otimização usando Metaheurísticas, Aprendizado por Reforço, Aprendizado por Reforço Multiagente e Problema *pHub* Centro Capacitado de Múltipla Alocações (CMA*p*HCP);

- Examinar e compreender o funcionamento do *Framework* `AMAM`, em especial suas técnicas de cooperação e aprendizado, afim de propor a incorporação de novos recursos e aplicá-los a solução do problema *pHub* Centro;

- Implementar e testar novas metaheurísticas e/ou metaheurísticas híbridas para solução do CMA*p*HCP, baseadas nas existentes na literatura.

- Criação de novas instâncias, para o CMA*p*HCP, com número maiores de nós, baseando-se nas instâncias disponíveis na literatura (*Australian Post*), abordando cenários em que existam diferentes demandas entre os pares origem-destino, diferentes capacidades e diferentes fatores de desconto/penalidade de coleta, distribuição e entrega. Além de valida-las com os métodos implementados.

- Implementação de métodos de aprendizado por reforço, tanto individual como social no *Framework* `AMAM`.

Um objetivo de grande interesse é o aprendizado por reforço. O *Framework* `AMAM` possui técnicas de RL individual e compartilhamento de informações entre os agentes. Porém, em um sistema multiagente, devido a sua complexidade, se faz necessário que o agente vá além do aprendizado individual e seja capaz de aprender de forma social (aprendizado multiagente). Considerando sua simplicidade e generalidade de configuração, o aprendizado por reforço também se torna interessante para o aprendizado multiagente (*Multi-Agent Reinforcement Learning* - MARL). Sistemas multiagentes podem se portar de forma totalmente cooperativa, totalmente competitiva e mais geral (nem cooperativa nem competitiva), e diversas formas de aprendizado por reforço estão disponíveis na literatura. A partir daí apresenta-se como trabalho futuro um amplo levantamento bibliográfico, a respeito de métodos de Aprendizado por Reforço Multiagente, que possam ser incorporados na estrutura do *Framework* `AMAM` e implementados na resolução de problemas de otimização combinatória.

Referências Bibliográficas

Ahuja, Ravindra K.; Magnanti, Thomas L. e Orlin, James B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Alba, E.; Almeida, F.; Blesa, M.; Cotta, C.; Díaz, M.; Dorta, I.; Gabarró, J.; León, C.; Luque, G.; Petit, J.; Rodríguez, C.; Rojas, A. e Xhafa, F. (2006). Efficient parallel LAN/WAN algorithms for optimization. The Mallba Project. *Parallel Computing*, v. 32, n. 5–6, p. 415–440.

Alba, E.; Luque, G.; Garcia-Nieto, J.; Ordonez, G. e Leguizamon, G. (2007). MALLBA: a software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications*, v. 1, n. 1, p. 74–85.

Alba, Enrique; Almeida, Francisco; Blesa, María J.; Cabeza, J.; Cotta, Carlos; Díaz, M.; Dorta, I.; Gabarró, Joachim; León, C.; andLuzMarina Moreno, J. Luna; Pablos, C.; Petit, Jordi; Rojas, A. e Xhafa, Fatos. (2002). MALLBA: A library of skeletons for combinatorial optimisation (research note). Monien, Burkhard e Feldmann, Rainer, editors, *Proceedings of the 8th International Euro-Par Conference on Parallel Processing (Euro-Par '02)*, volume 2400 of *Lecture Notes in Computer Science*, p. 927–932. Springer-Verlag, (2002).

Alumur, Sibel e Kara, Bahar Y. (2008). Network hub location problems: The state of the art. *European Journal of Operational Research*, v. 190, n. 1, p. 1 – 21. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2007.06.008. URL `http://www.sciencedirect.com/science/article/pii/S0377221707005577`.

Aydemir, F. B.; Günay, A.; Öztoprak, F.; Ş.I. Birbil, e Yolum, P. (2012). Multiagent cooperation for solving global optimization problems: an extendible framework with example cooperation strategies. *Journal of Global Optimization*, v. 57, n. 2, p. 499–519.

Aydin, M. E. July(2010). Collaboration of heterogenous metaheuristic agents. *2010 Fifth International Conference on Digital Information Management (ICDIM)*, p. 540–545, July(2010).

Aydin, M. E. (2012). Coordinating metaheuristic agents with swarm intelligence. *Journal of Intelligent Manufacturing*, v. 23, n. 4, p. 991–999.

Aydin, M. E. (2013). Agentification of individuals: A multi-agent approach to metaheuristics. *Journal of Computer Science & Systems Biology*, v. 6, n. 5.

Aykin, Turgut. (1994). Lagrangian relaxation based approaches to capacitated hub-and-spoke network design problem. *European Journal of Operational Research*, v. 79, n. 3, p. 501–523.

Aykin, Turgut. (1995)a. The hub location and routing problem. *European Journal of Operational Research*, v. 83, n. 1, p. 200–219.

Aykin, Turgut. (1995)b. Networking policies for hub-and-spoke systems with application to the air transportation system. *Transportation Science*, v. 29, n. 3, p. 201–221.

Balaji, P. G. e Srinivasan, D. (2010). *An Introduction to Multi-Agent Systems*, p. 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-14435-6. doi: 10.1007/978-3-642-14435-6_1. URL `https://doi.org/10.1007/978-3-642-14435-6_1`.

Barbucha, Dariusz; Czarnowski, Ireneusz; Jędrzejowicz, Piotr; Ratajczak-Ropel, Ewa e Wierzbowska, Izabela. (2010). JABAT middleware as a tool for solving optimization problems. Nguyen, Ngoc Thanh e Kowalczyk, Ryszard, editors, *Transactions on Computational Collective Intelligence II*, p. 181–195. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bashiri, Mahdi; Mirzaei, Masoud e Randall, Marcus. (2013). Modeling fuzzy capacitated p-hub center problem and a genetic algorithm solution. *Applied Mathematical Modelling*, v. 37, n. 5, p. 3513 – 3525. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2012.07.018. URL `http://www.sciencedirect.com/science/article/pii/S0307904X12004301`.

Bellifemine, Fabio; Poggi, Agostino e Rimassa, Giovanni. (2007). *Developing multi-agent systems with JADE*. John Wiley.

Blum, C.; Puchinger, J.; Raidl, G. R. e Roli, A. (2010). A brief survey on hybrid metaheuristics. Filipič, B. e Šilc, J., editors, *Proceedings of the 4th International Conference on Bio-inspired Optimization Methods and their Applications (BIOMA 2010)*, p. 3–18, Jozef Stefan Institute, Ljubljana, Slovenia.

Blum, C. e Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, v. 35, n. 3, p. 268–308.

Blum, Christian; Puchinger, Jakob; Raidl, Günther R. e Roli, Andrea. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, v. 11, n. 6, p. 4135–4151. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2011.02.032. URL `http://www.sciencedirect.com/science/article/pii/S1568494611000962`.

Blum, Christian e Roli, Andrea. (2008). *Hybrid Metaheuristics: An Introduction*, p. 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-78295-7. doi: 10.1007/978-3-540-78295-7_1. URL `https://doi.org/10.1007/978-3-540-78295-7_1`.

Blum, Christian; Roli, Andrea e Alba, Enrique. (2005). An introduction to metaheuristic techniques. *Parallel Metaheuristics: A New Class of Algorithms*, v. 47, p. 1.

Brimberg, Jack; Mladenović, Nenad; Todosijević, Raca e Urošević, Dragan. Feb(2017)a. A basic variable neighborhood search heuristic for the uncapacitated multiple allocation p-hub center problem. *Optimization Letters*, v. 11, n. 2, p. 313–327. ISSN 1862-4480. doi: 10.1007/s11590-015-0973-5. URL `https://doi.org/10.1007/s11590-015-0973-5`.

Brimberg, Jack; Mladenović, Nenad; Todosijević, Raca e Urošević, Dragan. Feb(2017)b. General variable neighborhood search for the uncapacitated single allocation p-hub center problem. *Optimization Letters*, v. 11, n. 2, p. 377–388. ISSN 1862-4480. doi: 10.1007/s11590-016-1004-x. URL `https://doi.org/10.1007/s11590-016-1004-x`.

Cahon, S.; Melab, N. e Talbi, E.-G. (2004). ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, v. 10, n. 3, p. 357–380.

Calik, Hatice; Alumur, Sibel A.; Kara, Bahar Y. e Karasan, Oya E. (2009). A tabu-search based heuristic for the hub covering problem over incomplete hub networks. *Computers & Operations Research*, v. 36, n. 12, p. 3088 – 3096. ISSN 0305-0548.

Campbell, Ann Melissa; Lowe, Timothy J e Zhang, Li. (2007). The p-hub center allocation problem. *European Journal of Operational Research*, v. 176, n. 2, p. 819–835.

Campbell, James F. (1994). Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, v. 72, n. 2, p. 387 – 405. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(94)90318-2. URL `http://www.sciencedirect.com/science/article/pii/0377221794903182`.

Cano, Alberto; Luna, José María; Zafra, Amelia e Ventura, Sebastián. January(2015). A classification module for genetic programming algorithms in jclec. *J. Machine Learning Research*, v. 16, n. 1, p. 491–494. ISSN 1532-4435.

Coelho, I. M.; Munhoz, P. L. A.; Haddad, M. N.; Coelho, V. N.; Silva, M. M.; Souza, M. J. F. e Ochi, L. S. MAY(2011). OptFrame: A computational framework for combinatorial optimization problems. *Proc. of the VII ALIO EURO Workshop on Applied Combinatorial Optimization*, p. 51–54. ALIO/EURO 2011, MAY(2011).

Coelho, I. M.; Ribas, S.; Perché, M. H. P.; Munhoz, P. L. A.; Souza, M. J. F. e Ochi, L. S. Sept(2010). OptFrame: a computational framework for combinatorial problems. *Proceedings of the XLII Simpósio Brasileiro de Pesquisa Operacional*, p. 1887–1898, Sept(2010).

Cotta, Carlos; Talbi, El-Ghazali e Alba, Enrique. (2005). Parallel hybrid metaheuristics. Alba, Enrique, editor, *Parallel Metaheuristics: a New Class of Algorithms*, p. 347–370. John Wiley & Sons.

Danoy, Grégoire; Bouvry, Pascal e Boissier, Olivier. (2005). Dafo, a multi-agent framework for decomposable functions optimization. Khosla, Rajiv; Howlett, Robert J. e Jain, Lakhmi C., editors, *Knowledge-Based Intelligent Information and Engineering Systems: 9th International Conference, KES 2005, Melbourne, Australia, September 14-16, 2005, Proceedings, Part IV*, p. 626–632. Springer Berlin Heidelberg, Berlin, Heidelberg.

Danoy, Grégoire; Bouvry, Pascal e Boissier, Olivier. (2010). A multi-agent organizational framework for coevolutionary optimization. Jensen, Kurt; Donatelli, Susanna e Koutny, Maciej, editors, *Transactions on Petri Nets and Other Models of Concurrency IV*, p. 199–224. Springer Berlin Heidelberg, Berlin, Heidelberg.

De Beukelaer, Herman; Davenport, Guy F.; De Meyer, Geert e Fack, Veerle. (2015). JAMES: A modern object-oriented java framework for discrete optimization using local search metaheuristics. *Proc. 4th International Symposium and 26Th National Conference on Operational Research: Hellenic Operational Research Society*, p. 134–138, (2015).

De Beukelaer, Herman; Davenport, Guy F.; De Meyer, Geert e Fack, Veerle. (2017). JAMES: An object-oriented java framework for discrete optimization using local search metaheuristics. *Software: Practice and Experience*, v. 47, n. 6, p. 921–938. doi: 10.1002/spe.2459.

de Sá, Elisangela Martins; de Camargo, Ricardo Saraiva e de Miranda Júnior, Gilberto. (2010). Redes eixo-raio aplicada ao transporte público. *Anais do XLII SBPO*, p. 2415–2426, Bento Gonçalves, RS.

Dorigo, Marco; Maniezzo, Vittorio; Colorni, Alberto e others,. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics*, v. 26, n. 1, p. 29–41.

Durillo, Juan J. e Nebro, Antonio J. (2011). jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, v. 42, n. 10, p. 760–771.

Durillo, Juan J; Nebro, Antonio J e Alba, Enrique. (2010). The jMetal framework for multi-objective optimization: Design and architecture. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, p. 1–8, (2010).

Eberhart, Russell e Kennedy, James. (1995). Particle swarm optimization. *Proceedings of the IEEE international conference on neural networks*, volume 4, p. 1942–1948. Citeseer, (1995).

Ernst, Andreas T.; Hamacher, Horst; Jiang, Houyuan; Krishnamoorthy, Mohan e Woeginger, Gerhard. (2009). Uncapacitated single and multiple allocation p-hub center problems. *Computers & Operations Research*, v. 36, n. 7, p. 2230 – 2241. ISSN 0305-0548.

Ernst, Andreas T. e Krishnamoorthy, Mohan. (1996). Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, v. 4, n. 3, p. 139 – 154.

Ernst, Andreas T e Krishnamoorthy, Mohan. (1998). Exact and heuristic algorithms for the uncapacitated multiple allocation p-hub median problem. *European Journal of Operational Research*, v. 104, n. 1, p. 100–112.

Ernst, Andreas T e Krishnamoorthy, Mohan. (1999). Solution algorithms for the capacitated single allocation hub location problem. *Annals of Operations Research*, v. 86, p. 141–159.

Farahani, Reza Zanjirani; Hekmatfar, Masoud; Arabani, Alireza Boloori e Nikbakhsh, Ehsan. (2013). Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, v. 64, n. 4, p. 1096 – 1109. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2013.01.012. URL `http://www.sciencedirect.com/science/article/pii/S0360835213000326`.

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 9, p. 849–859.

Ferber, Jacques e Weiss, Gerhard. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

Fernandes, F. C. sep(2009). Arquitetura multiagente metaheurísticos cooperativos para resolução de problemas de otimização combinatória. Dissertação de mestrado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brasil.

Fink, Andreas e Voß, Stefan. (2002). Hotframe: A heuristic optimization framework. Voß, Stefan e Woodruff, David L., editors, *Optimization Software Class Libraries*, p. 81–154. Springer US, Boston, MA. ISBN 978-0-306-48126-0. doi: 10.1007/0-306-48126-X_4. URL `http://dx.doi.org/10.1007/0-306-48126-X_4`.

Fink, Andreas; Voß, Stefan e Woodruff, David L. (1999). Building reusable software components for heuristic search. Kall, Peter e Lüthi, Hans-Jakob, editors, *Operations Research Proceedings 1998: Selected Papers of the International Conference on Operations Research Zurich, August 31 – September 3, 1998*, volume 1998 of *Operations Research Proceedings 1998*, p. 210–219. Springer Berlin Heidelberg.

Gaspar-Cunha, Antônio; Takahashi, Ricardo e Antunes, Carlos Henggeler. (2012). *Manual de Computação Evolutiva e Metaheurística*. Imprensa da Universidade de Coimbra e Editora da Universidade Federal de Minas Gerais, Coimbra, Portugal.

Gaspero, Luca Di e Schaerf, Andrea. (2001). A case-study for EasyLocal++: the course timetabling problem. Relatório Técnico Technical Report UDMI/13/2001/RR, Dipartimento di Matematica e Informatica - Universit'a di Udine, Italy.

Gaspero, Luca Di e Schaerf, Andrea. (2002). Writing local search algorithms using Easylocal++. Voß, Stefan e Woodruff, David L., editors, *Optimization Software Class Libraries*, p. 155–175. Springer US, Boston, MA.

Gaspero, Luca Di e Schaerf, Andrea. (2003). EASYLOCAL++: an object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience*, v. 33, n. 8, p. 733–765.

Gavriliouk, Elena O. (2009). Aggregation in hub location problems. *Computers & Operations Research*, v. 36, n. 12, p. 3136 – 3142. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2009.01.010. URL `http://www.sciencedirect.com/science/article/pii/S0305054809000239`. New developments on hub location.

Glover, F. W. e Laguna, M. (1998). *Tabu Search*. Springer-Verlag.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1th edição.

Günay, Akin; Öztoprak, Figen; Ş.Ilker Birbil, e Yolum, Pinar. (2009). Solving global optimization problems using MANGO. Håkansson, Anne; Nguyen, Ngoc Thanh; Hartung, Ronald L.; Howlett, Robert J. e Jain, Lakhmi C., editors, *Agent and Multi-Agent Systems: Technologies and Applications*, volume 5559 of *Lecture Notes in Computer Science*, p. 783–792. Springer Berlin Heidelberg.

Hakimi, S Louis. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, v. 12, n. 3, p. 450–459.

Hsieh, Sun-Yuan e Kao, Shih-Shun. (2019). A survey of hub location problems. *Journal of Interconnection Networks*, v. 19, n. 01, p. 1940005.

Humeau, J.; Liefooghe, A.; Talbi, E.-G. e Verel, S. (2013). ParadisEO-MO: from fitness landscape analysis to efficient local search algorithms. *Journal of Heuristics*, v. 19, n. 6, p. 881–915.

Ilić, Aleksandar; Urošević, Dragan; Brimberg, Jack e Mladenović, Nenad. (2010). A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, v. 206, n. 2, p. 289 – 300.

Kaelbling, Leslie Pack; Littman, Michael L e Moore, Andrew W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285.

Kara, Bahar Y. e Tansel, Barbaros C. (2000). On the single-assignment p-hub center problem. *European Journal of Operational Research*, v. 125, n. 3, p. 648 – 655. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(99)00274-X. URL `http://www.sciencedirect.com/science/article/pii/S037722179900274X`.

Kara, Bahar Y e Tansel, Barbaros Ç. (2001). The latest arrival hub location problem. *Management Science*, v. 47, n. 10, p. 1408–1420.

Kerçelli, L.; Sezer, A.; Öztoprak, F.; Yolum, P. e Ş.I. Birbil,. (2008). MANGO: A multiagent environment for global optimization. *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'08)*, p. 86–91, Estoril, Portugal.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680. ISSN 0036-8075. doi: 10.1126/science.220.4598.671. URL `https://science.sciencemag.org/content/220/4598/671`.

Klincewicz, John G. (1991). Heuristics for the p-hub location problem. *European Journal of Operational Research*, v. 53, n. 1, p. 25–37.

Klincewicz, John G. (1992). Avoiding local optima in thep-hub location problem using tabu search and grasp. *Annals of Operations research*, v. 40, n. 1, p. 283–302.

Kronfeld, Marcel; Planatscher, Hannes e Zell, Andreas. (2010). The EvA2 optimization framework. Blum, Christian e Battiti, Roberto, editors, *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*, p. 247–250. Springer Berlin Heidelberg, Berlin, Heidelberg.

Liefooghe, A.; Jourdan, L. e Talbi, E. (2011). A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. *European Journal of Operational Research*, v. 209, n. 2, p. 104–112.

Lotfi, Nasser e Acan, Adnan. (2015). Learning-based multi-agent system for solving combinatorial optimization problems: A new architecture. Onieva, Enrique; Santos, Igor; Osaba, Eneko; Quintián, Héctor e Corchado, Emilio, editors, *Hybrid Artificial Intelligent Systems: 10th International Conference, HAIS 2015, Bilbao, Spain, June 22-24, 2015, Proceedings*, p. 319–332. Springer International Publishing. ISBN 978-3-319-19644-2. doi: 10.1007/978-3-319-19644-2_27.

Lotfi, Nasser e Acan, Adnan. (2016). A tournament-based competitive-cooperative multiagent architecture for real parameter optimization. *Soft Computing*, v. 20, n. 11, p. 4597–4617.

Lourenço, H. R.; Martin, O. e Stützle, T. (2001). A beginner's introduction to Iterated Local Search. *Proceedings of the 4th Metaheuristics International Conf. (MIC 2001)*, p. 1–11, Porto, Portugal.

Lukasiewycz, Martin; Glaß, Michael; Reimann, Felix e Teich, Jürgen. (2011). Opt4J: A modular framework for meta-heuristic optimization. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, p. 1723–1730, New York, NY, USA. ISBN 978-1-4503-0557-0.

Malek, R. Oct(2010). An agent-based hyper-heuristic approach to combinatorial optimization problems. *Proceedings of the 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, volume 3, p. 428–434, Oct(2010).

Malek, Richard. (2009). Collaboration of metaheuristic algorithms through a multi-agent system. Mařík, Vladímir; Strasser, Thomas e Zoitl, Alois, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, p. 72–81. Springer.

Martin, Simon; Ouelhadj, Djamila; Beullens, Patrick; Ozcan, Ender; Juan, Angel A. e Burke, Edmund K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, v. 254, n. 1, p. 169–178.

Meignan, D.; Creput, J.-C. e Koukam, A. (2008)a. A coalition-based metaheuristic for the vehicle routing problem. *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008)*, p. 1176–1182, (2008)a.

Meignan, David; Créput, Jean-Charles e Koukam, Abderrafiâa. (2008)b. An organizational view of metaheuristics. Jennings, N.R.; Rogers, A.; Petcu, A. e Ramchurn, S. D., editors, *First International Workshop on Optimisation in Multi-Agent Systems, AAMAS'08*, p. 77–85, (2008)b.

Meignan, David; Créput, Jean-Charles e Koukam, Abderrafiaa. (2009). A cooperative and self-adaptive metaheuristic for the facility location problem. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO'09, p. 317–324, New York, NY, USA. ACM.

Meignan, David; Koukam, Abderrafiaa e Créput, Jean-Charles. (2010). Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, v. 16, n. 6, p. 859–879.

Melab, N.; Luong, T. Van; Boufaras, K. e Talbi, E. (2013). ParadisEO-MO-GPU: A framework for parallel GPU-based local search metaheuristics. *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, p. 1189–1196. ACM, (2013).

Meyer, T.; Ernst, A.T. e Krishnamoorthy, M. (2009). A 2-phase algorithm for solving the single allocation p-hub center problem. *Computers & Operations Research*, v. 36, n. 12, p. 3143 – 3151. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2008.07.011. URL `http://www.sciencedirect.com/science/article/pii/S0305054808001342`. New developments on hub location.

Milano, M. e Roli, A. (2004). MAGMA: A multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, v. 34, n. 2, p. 925–941.

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

Narendra, K. S. e Thathachar, M. A. L. July(1974). Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, v. SMC-4, n. 4, p. 323–334. ISSN 0018-9472. doi: 10.1109/TSMC.1974.5408453.

Ochoa, Gabriela; Hyde, Matthew; Curtois, Tim; Vazquez-Rodriguez, Jose A.; Walker, James; Gendreau, Michel; Kendall, Graham; McCollum, Barry; Parkes, Andrew J.; Petrovic, Sanja e Burke, Edmund K. (2012). HyFlex: A benchmark framework for cross-domain heuristic search. Hao, Jin-Kao e Middendorf, Martin, editors, *Evolutionary Computation in Combinatorial Optimization: 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, p. 136–147. Springer Berlin Heidelberg, Berlin, Heidelberg.

O'Kelly, Morton E. (1986). Activity levels at hub facilities in interacting networks. *Geographical Analysis*, v. 18, n. 4, p. 343–356.

O'kelly, Morton E. (1986). The location of interacting hub facilities. *Transportation science*, v. 20, n. 2, p. 92–106.

O'kelly, Morton E. (1987). A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, v. 32, n. 3, p. 393 – 404. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(87)80007-3.

Ottoni, André Luiz Carvalho; Lamperti, Rubisson Duarte; Nepomuceno, Erivelton Geraldo; Oliveira, MS e Oliveira, FF. (2012). Modelagem e simulação de um sistema de aprendizado de reforço para robôs. *VIII Encontro Mineiro de Engenharia de Produçao, ISSN*, v. 629, p. 1983.

Parejo, J. A.; Racero, J.; Guerrero, F.; Kwok, T. e Smith, K. A. (2003). FOM: A framework for metaheuristic optimization. Sloot, Peter M. A.; Abramson, David; Bogdanov, Alexander V.; Gorbachev, Yuriy E.; Dongarra, Jack J. e Zomaya, Albert Y., editors, *Computational Science — ICCS 2003: International Conference, Melbourne, Australia and St. Petersburg, Russia, June 2–4, 2003 Proceedings, Part IV*, p. 886–895. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silva, M. A. L. (2007). Modelagem de uma arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória. Dissertação de mestrado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brazil.

Silva, M. A. L. (2019). Amam: *Framework* multiagente para otimização usando metaheuísticas. Tese de doutorado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brazil.

Silva, M. A. L.; de Souza, S. R.; de Oliveira, S. M. e Souza, M. J. F. (2014). An agent-based metaheuristic approach applied to the vehicle routing problem with time-windows. *Proceedings of the 2014 Brazilian Conference on Intelligent Systems - Enc. Nac. de Inteligência Artificial e Computacional (BRACIS-ENIAC 2014)*, São Carlos, SP, Brazil.

Silva, M. A. L.; de Souza, S. R.; Souza, M. J. F. e de Oliveira, S. M. Nov(2015). A multi-agent metaheuristic optimization framework with cooperation. *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, p. 104–109, Natal, Brazil. doi: 10.1109/BRACIS.2015.64.

Silva, Maria Amélia Lopes; de Souza, Sérgio Ricardo; Souza, Marcone Jamilson Freitas e Bazzan, Ana Lúcia C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, v. 131, p. 148 – 171.

Silva, Maria Amélia Lopes; de Souza, Sérgio Ricardo; Souza, Marcone Jamilson Freitas e deFrança Filho, Moacir Felizardo. (2018). Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. *Applied Soft Computing*, v. 71, p. 433–459.

Sim, Thaddeus; Lowe, Timothy J. e Thomas, Barrett W. (2009). The stochastic p-hub center problem with service-level constraints. *Computers & Operations Research*, v. 36, n. 12, p. 3166 – 3177. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2008.11.020. URL `http://www.sciencedirect.com/science/article/pii/S0305054808002396`. New developments on hub location.

Skorin-Kapov, Darko; Skorin-Kapov, Jadranka e O'Kelly, Morton. (1996). Tight linear programming relaxations of uncapacitated p-hub median problems. *European journal of operational research*, v. 94, n. 3, p. 582–593.

Stützle, Thomas. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Phd thesis, Technischen Universität Darmstadt, (1998).

Sutton, Richard S. e Barto, Andrew G. (1998). *Reinforcement learning: An introduction*, volume 135. MIT press Cambridge.

Toh, Rex S e Higgins, Richard G. (1985). The impact of hub and spoke network centralization and route monopoly on domestic airline profitability. *Transportation Journal*, v. 24, n. 4, p. 16–27.

Turek, Wojciech; Stypka, Jan; Krzywicki, Daniel; Anielski, Piotr; Pietak, Kamil; Byrski, Aleksander e Kisiel-Dorohinicki, Marek. (2016). Highly scalable erlang framework for agent-based metaheuristic computing. *Journal of Computational Science*, v. 17, n. Part 1, p. 234 – 248. ISSN 1877-7503.

Ventura, Sebastián; Romero, Cristóbal; Zafra, Amelia; Delgado, José A. e Hervás, César. (2008). JCLEC: a java framework for evolutionary computation. *Soft Computing*, v. 12, n. 4, p. 381–392.

Wagner, S. e Affenzeller, M. (2004). HeuristicLab Grid: A flexible and extensible environment for parallel heuristic optimization. *Proceedings of the 15th International Conference on Systems Science*, volume 1, p. 289–296, (2004).

Wagner, S. e Affenzeller, M. (2005). HeuristicLab: A generic and extensible optimization environment. Ribeiro, Bernardete; Albrecht, Rudolf F.; Dobnikar, Andrej; W.Pearson, David e Steele, Nigel C., editors, *Proceedings of the International Conference in Adaptive and Natural Computing Algorithms, Coimbra, Portugal, 2005*, p. 538–541. Springer Vienna.

Wagner, S.; Beham, A.; Kronberger, G. K.; Kommenda, M.; Pitzer, E.; Kofler, M.; Vonolfen, S.; Winkler, S. M.; Dorfer, V. e Affenzeller, M. (2010). Heuristiclab 3.3: A unified approach to metaheuristic optimization. *Proceedings of the VII Congreso Español sobre Metaheursíticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2010)*, Valencia, Spain.

Wagner, S.; Kronberger, G.; Beham, A.; Kommenda, M.; Scheibenpflug, A.; Pitzer, E.; Vonolfen, S.; Kofler, M.; Winkler, S.; Dorfer, V. e Affenzeller, M. (2014). Architecture and design of the HeuristicLab optimization environment. Klempous, Ryszard; Nikodem, Jan; Jacak, Witold e Chaczko, Zenon, editors, *Advanced Methods and Applications in Computational Intelligence*, volume 6 of *Topics in Intelligent Engineering and Informatics*, p. 197–261. Springer.

Watkins, Christopher J. C. H. e Dayan, Peter. (1992). Q-learning. *Machine Learning*, v. 8, n. 3, p. 279–292.

Watkins, Christopher John Cornish Hellaby. *Learning from delayed rewards*. PhD thesis, University of Cambridge, Cambridge, England, (1989).

White, David R. (2012). Software review: the ECJ toolkit. *Genetic Programming and Evolvable Machines*, v. 13, n. 1, p. 65–67.

Wooldridge, Michael. (2009). *An introduction to multiagent systems*. John Wiley & Sons.

Yaman, Hande e Elloumi, Sourour. (2012). Star p-hub center problem and star p-hub median problem with bounded path lengths. *Computers & Operations Research*, v. 39, n. 11, p. 2725 – 2732. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2012.02.005. URL `http://www.sciencedirect.com/science/article/pii/S0305054812000317`.

Yang, Kai; Liu, Yankui e Zhang, Xin. (2011). Stochastic p-hub center problem with discrete time distributions. *Proceedings of the 8th International Symposium on Neural Networks (ISNN 2011)*, volume Part II, p. 182–191. Springer, (2011).

Zabihi, A e Gharakhani, M. (2018). A literature survey of hub location problems and methods with emphasis on the marine transportations. *Uncertain Supply Chain Management*, v. 6, n. 1, p. 91–116.