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

AMAM: *FRAMEWORK* MULTIAGENTE PARA OTIMIZAÇÃO USANDO METAHEURÍSTICAS

Tese de Doutorado apresentado ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos necessários para obtenção do título de Doutora em Modelagem Matemática e Computacional.

Aluno : Maria Amélia Lopes Silva

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

Co-Orientador : Prof. Dr. Marcone Jamilson Freitas Souza (UFOP)

Belo Horizonte - MG
Junho de 2019

Silva, Maria Amélia Lopes
S237a AMAM: framework multiagente para otimização usando metaheurísticas / Maria Amélia Lopes Silva. – 2019.
135 f.

Tese de doutorado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Sérgio Ricardo de Souza.
Coorientador: Marcone Jamilson Freitas Souza.
Tese (doutorado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Metaheurística – Teses. 2. Sistemas de multiagentes – Teses. 3. Agentes inteligentes (Software) – Teses. 4. Computadores híbridos – Teses. 5. Otimização combinatória – Teses. I. Souza, Sérgio Ricardo de. II. Souza, Marcone Jamilson Freitas. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 004.8

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

AMAM: FRAMEWORK MULTIAGENTE PARA OTIMIZAÇÃO USANDO METAHEURÍSTICAS.

Tese de Doutorado apresentada por **Maria Amélia Lopes Silva**, em 24 de junho de 2019, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Sérgio Ricardo de Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Marcone Jamilson Freitas Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Profª. Drª. Ana Lúcia Cetertich Bazzan
Universidade Federal do Rio Grande do Sul

Profª. Drª. Elizabeth Ferreira Gouvêa Goldbarg
Universidade Federal do Rio Grande do Norte

Prof. Dr. Alexsio Mendes Lacerda
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Henrique Elias Borges
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida à impressão,

Prof. Dr. Thiago de Souza Rodrigues
Coordenador do Programa de Pós-Graduação Stricto Sensu em Modelagem Matemática e Computacional

Ao meu “menino”, Murilo, a sua simples existência já me motiva a ser melhor.

Agradecimentos

"Aqueles que passam por nós, não vão sós, não nos deixam sós.

Deixam um pouco de si, levam um pouco de nós."

Antoine de Saint-Exupéry

Ao terminar este trabalho de tese percebo que os agradecimentos não podem ficar limitados às contribuições recebidas ao longo destes seis anos. Mesmo que estes seis anos tenham sido mais longos do que o esperado. Percebo a importância de cada um que passou pela minha vida, tanto para a formação da pessoa, quanto para a formação da pesquisadora que sou hoje. Entretanto, alguns nomes precisam ser citados.

Primeiramente, como em tudo que faço na vida, agradeço a Deus, por me permitir ser tão positiva e por me fazer enxergar as infinitas oportunidades que a vida tem a oferecer.

Agradeço especialmente ao Professor Sérgio Ricardo de Souza e ao Professor Marcone Jamilson Freitas Souza. O significado do termo "orientar" diz muito do que vocês me proporcionaram, que ultrapassa esta tese. "Guiar, encaminhar, dirigir, direcionar, conduzir, e ainda, aconselhar, estimular, ensinar, instruir e influenciar. Sou duplamente premiada por poder contar com vocês nesta caminhada. A dedicação de vocês ao ofício de ensinar me inspira a ser uma profissional melhor. Agradeço as longas conversas, a paciência, a amizade e o incentivo constante. Não tenham dúvidas de que marcaram minha história.

Agradeço a toda minha família. Aos meus pais, que são e sempre serão meus maiores exemplos de vida. Aos meus irmãos, é através de nossa amizade que vencemos tudo. Aos meus, Leonardo e Murilo, vocês fazem tudo ter mais significado.

Aos velhos e novos amigos, agradeço o convívio e por direta ou indiretamente contribuírem na minha busca pelo equilíbrio.

Agradeço aos membros da banca examinadora, Profa. Dra. Ana Lucia Cetertich Bazzan, Prof. Dr. Anísio Mendes Lacerda, Profa. Dra. Elizabeth Ferreira Gouvêa Goldbarg e Prof. Dr. Henrique Elias Borges pelo interesse e disponibilidade. Especialmente, agradeço ao apoio dado pela professora Ana Bazzan, que, compartilhando seu conhecimento, muito contribuiu para o desenvolvimento desta tese.

Agradeço à Universidade Federal de Viçosa pelo apoio.

A todos vocês, meus sinceros agradecimentos.


"ESSENCIAL À VIDA DE TODOS OS SERES VIVOS", COMEÇA COM A LETRA "A"...
COMO VOCÊ CONSEGUIU ERRAR ESSA?!
É ÓBVIO QUE É "ÁGUA"!
O QUE VOCÊ COLOCOU NA PROVA?!!
AMOR!
beckilustras@gmail.com

Resumo

Esta tese apresenta um *framework* multiagente para otimização usando metaheurísticas, denominado Arquitetura Multiagente para Metaheurísticas (AMAM). O *framework* AMAM é uma estrutura genérica e flexível, que tem, como principal característica, a facilidade de hibridização de metaheurísticas, a partir da utilização de conceitos relacionados a sistemas multiagentes. Nesta proposta, cada agente atua independentemente no espaço de busca de um problema de otimização combinatória. Os agentes compartilham informações e colaboram entre si através do ambiente. Esta tese tem, como principal contribuição, a consolidação do *framework* AMAM como uma ferramenta capaz de resolver diferentes problemas de otimização e que permita a fácil hibridização de metaheurísticas. Para tal, propõe a revisão da estrutura do *framework* AMAM, com a incorporação de novos recursos que permitam dinamizar e aperfeiçoar o processo de solução. A estrutura do *framework* foi dividida em dimensões, ao se considerar suas diferentes perspectivas. A remoção de estruturas de coordenação explícita e de elementos que intermediavam a comunicação permitiram aumentar a autonomia do agente. A cooperação entre os agentes foi aprimorada, buscando maior diversidade nas soluções disponíveis na estrutura cooperativa, através da definição de novos critérios de inserção de novas soluções. É proposta também a incorporação de capacidades auto-adaptativas nos agentes. O objetivo é permitir que o agente modifique suas ações com base nas experiências obtidas na interação com os outros agentes e com o ambiente, usando conceitos de Aprendizagem de Máquina. Neste sentido, são apresentadas duas propostas de agentes adaptativos baseados no algoritmo *Q-Learning* e em Autômatos de Aprendizagem. Para melhor introdução e validação do *framework* AMAM, esta tese utiliza instanciações do *framework* para dois problemas clássicos de otimização combinatória: Problema de Roteamento de Veículos com Janelas de Tempo (sigla em inglês, VRPTW) e o Problema de Sequenciamento de Máquina Paralela Não Relacionada com Tempos de Configuração Dependentes de Sequência (sigla em inglês, UPMSP-ST). Os experimentos demonstraram a efetiva redução nos custos das soluções com o uso de agentes cooperativos e a escalabilidade da proposta. Os experimentos também confirmaram que a capacidade de aprender atribuída ao agente influencia diretamente a qualidade das soluções, tanto do ponto de vista individual quanto do ponto de vista do trabalho em equipe. Sendo assim, a adaptabilidade dos agentes é confirmada, demonstrando que as técnicas de aprendizado utilizadas conseguem superar a necessidade de conhecimento das características específicas do problema a ser tratado. Os resultados obtidos possibilitaram concluir que o *framework* aqui apresentado é um passo à frente em relação aos demais *frameworks* da literatura quanto à adaptação aos aspectos particulares dos problemas tratados.

Palavras-chaves: Metaheurísticas. Sistemas Multiagentes. Cooperação. Hibridização de Metaheurísticas. Otimização Combinatória.

Abstract

This thesis presents a multi-agent metaheuristic optimization framework, called Multi-agent Architecture for Metaheuristics (AMAM). AMAM is a generic and flexible framework, which has, as its main strength, the ease of hybridization of metaheuristics, from the use of concepts related to multi-agent systems. In this proposal, each agent acts independently in the search space of a combinatorial optimization problem. Agents share information and collaborate through the environment. This thesis has, as its main contribution, the consolidation of the AMAM framework as a tool capable of solving different optimization problems and allowing the easy hybridization of metaheuristics. To this end, it proposes the revision of the AMAM framework structure, with the incorporation of new resources that allow streamlining as well as to improve the solution process. The structure of the framework was divided into dimensions, considering its different perspectives. The removal of explicit coordination structures and of elements that intermediated the communication allowed to increase the autonomy of the agent. The cooperation between the agents was improved, seeking greater diversity in the solutions available in the cooperative structure, through the definition of new criteria for insertion of new solutions. This thesis also proposes the incorporation of self-adaptive capabilities in the agents. The goal is to allow the agent to modify their actions based on the experiences gained in interacting with other agents and the environment using Machine Learning concepts. In this sense, two proposals of adaptive agents based on the Q-Learning algorithm and Learning Automata are presented. For a better introduction and validation of the AMAM framework, this work uses framework instantiations for two classical combinatorial optimization problems: Vehicle Routing Problem with Time Windows (VRPTW) and Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSP-ST). The main objective of the experiments was to evaluate the performance of the contributions proposed here for the framework. The experiments demonstrated the effective reduction in solution costs with the use of cooperative agents and the scalability of the proposal. The experiments also confirmed that the learning ability attributed to the agent directly influences the quality of the solutions, both from the individual point of view and from the teamwork. Therefore, the adaptability of the agents is confirmed, demonstrating that the used learning techniques can overcome the need for knowledge of the specific characteristics of the problem being treated. The results obtained allow us to conclude that the framework presented here is a step forward concerning the other frameworks of the literature regarding the adaptation to the particular aspects of the problems.

Keywords: Metaheuristics. Multi-agent Systems. Cooperation. Hybridization of Metaheuristics. Combinatorial Optimization.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Apresentação geral

Metaheurísticas têm se consolidado como uma das principais metodologias para a resolução de Problemas de Otimização de diversas classes. Esta afirmação é justificada, em parte, pela versatilidade e grande adaptabilidade das metaheurísticas para resolver problemas, e, por outro lado, principalmente pela possibilidade de se obter, em tempo computacional limitado, soluções de boa qualidade para problemas de otimização complexos e de grandes dimensões. Problemas com essas características, em muitos e importantes casos, não possuem solução ótima conhecida em um tempo computacional limitado, se apenas técnicas de programação matemática forem aplicadas. Essa afirmação é particularmente verdadeira para a classe de problemas NP-difíceis, que envolve problemas de grande relevância teórica e prática, como é o caso dos Problemas de Roteamento de Veículos e Sequenciamento em Máquinas Paralelas.

Blum e Roli (2003), Talbi (2009), Gendreau e Potvin (2010) e Blum et al. (2011), dentre outros autores, apresentam formulações gerais e conceituais sobre metaheurísticas, bem como revisões importantes sobre seu uso na resolução de problemas de otimização.

Nos últimos anos, a combinação de duas ou mais metaheurísticas para resolver problemas de otimização vem crescendo (Cotta et al., 2005; Blum et al., 2011). Esta combinação de metaheurísticas é conhecida como hibridização de metaheurísticas. O principal objetivo da hibridização de metaheurísticas é aplicar em conjunto as melhores características de cada metaheurística para resolver um problema, permitindo, além de obter melhor qualidade de solução em um tempo menor, aumentar a capacidade de lidar com problemas mais complexos. Cotta et al. (2005) e Blum et al. (2011) afirmam que as hibridizações são responsáveis por muitos dos melhores resultados encontrados na literatura para várias classes de problemas de otimização, o que justifica o crescente interesse por esta abordagem.

Diversos autores buscam mecanismos de classificação para metaheurísticas híbridas (Talbi, 2002; Cotta et al., 2005; Raidl, 2006; Blum e Roli, 2008; Blum et al., 2010, 2011). No entanto, há uma dificuldade em empregar uma hierarquia simples nessa classificação, devido principalmente ao fato de que esses algoritmos possuem características sobrepostas. Além disso, muitas são as formas de hibridização de metaheurísticas que podem ser encontradas na literatura. A demanda por *softwares* cada vez mais adaptativos e inteligentes tem conduzido à incorporação de novas estratégias que diversificam essas formas de hibridização. Entre essas estratégias, destacam-se, aqui, as metaheurísticas cooperativas e metaheurísticas paralelas.

A cooperação entre metaheurísticas é utilizada como estratégia para a troca de infor-

mações entre os algoritmos envolvidos na busca da solução do problema. Essa troca de informações tem, como objetivo principal, orientar a busca pelas regiões mais promissoras do espaço de solução. Por sua vez, o paralelismo permite a execução simultânea de métodos e a consequente redução no tempo de busca. A combinação entre cooperação e paralelismo está sendo intensamente desenvolvida e está se tornando cada dia mais importante no contexto da otimização.

O aumento no uso de metaheurísticas, seja por especialistas ou não especialistas em métodos para resolver problemas de otimização, tem orientado pesquisadores no desenvolvimento de *frameworks*. Esses especialistas estão buscando ferramentas que possam facilitar a solução desses problemas, além de oferecer recursos que aprimoram o desempenho do processo de solução. Ao mesmo tempo, no caso dos pesquisadores, a busca por novas técnicas para resolver essa classe de problemas é um desafio fundamental, mas também direcionado ao desenvolvimento dessas ferramentas de *software*.

Os *frameworks* oferecem flexibilidade na incorporação de novos métodos de solução, sem exigir esforços para refazer a aplicação e, portanto, permitindo a determinação de melhores soluções com baixo custo de desenvolvimento. *Frameworks* fornecem uma estrutura de recursos genéricos para a solução de problemas de um domínio específico, tornando o desenvolvimento de novas aplicações neste domínio muito mais simples. Um *Framework* para Otimização usando Metaheurísticas (em inglês, *Metaheuristic Optimization Frameworks* - MOF), como mencionado em Parejo et al. (2012), é uma ferramenta de *software* que fornece implementações de um conjunto de metaheurísticas, através de códigos reutilizáveis, facilitando o desenvolvimento de aplicativos para solução de problemas de otimização.

Vários *framework*s para metaheurísticas podem ser encontrados na literatura, a maioria deles com características e propostas similares. Uma revisão bibliográfica e um estudo comparativo com os principais *frameworks* disponíveis podem ser encontrados em Parejo et al. (2012) e Silva et al. (2018). Os trabalhos correlatos a estes, encontrados na literatura recente, são detalhados na Seção 2.2.

Dentro deste contexto, no entanto, a hibridização de metaheurísticas não tem sido o foco de *frameworks* para otimização. Uma alternativa importante apresentada em muitos trabalhos para preencher essa lacuna é a utilização do conceito de agentes para o desenvolvimento de aplicações para solução de problemas de otimização. O conceito de agentes e sua generalização, na forma de sistemas multiagentes, constituem estratégias de grande importância nesse sentido, pois, como aponta Aydemir et al. (2012), o uso de sistemas multiagentes no contexto da Otimização permite a exploração simultânea de diferentes regiões do espaço de busca, possibilitando maior diversidade e melhores soluções. A abordagem baseada em agentes é usada como um elo entre as diferentes metaheurísticas envolvidas na solução de um problema. Neste caso, cada agente é responsável por realizar sua tarefa e, ao mesmo tempo, trocar informações com outros agentes em relação ao estágio de sua busca. Um conjunto específico de estudos que usam essa abordagem também é apresentada na Seção 2.2. Os trabalhos em questão também podem ser identificados como *frameworks*, pois definem estruturas genéricas capazes de lidar com diferentes problemas de otimização.

Nesse sentido, a partir de uma análise dos diversos *frameworks* encontrados na literatura recente, várias questões surgem:

(i) os *frameworks* atuais facilitam o desenvolvimento de aplicações de metaheurísticas híbridas?

(ii) eles permitem o desenvolvimento de aplicações com cooperação (troca de informações) entre métodos que são executados independente e simultaneamente?

(iii) qual é a melhor maneira de coordenar as metaheurísticas quando o objetivo é explorar o potencial de hibridização?

(iv) ao mesmo tempo, como poderia ser desenvolvida uma estrutura mais robusta, capaz de lidar com diferentes problemas com mudanças mínimas?

Este trabalho busca responder estas questões através do desenvolvimento de um *framework* multiagente para otimização usando metaheurísticas que preencha as lacunas existentes no desenvolvimento destas ferramentas.

Para tal, o presente trabalho apresenta o *framework* multiagente para otimização usando metaheurísticas denominado **Arquitetura MultiAgente para Metaheurística** (AMAM), proposto inicialmente em Silva (2007). Esta estrutura permite a fácil hibridização de metaheurísticas para resolver problemas de otimização combinatória, usando uma estrutura multiagente. Dessa forma, trata-se de uma estrutura genérica e flexível, na qual cada metaheurística é definida como um agente autônomo que interage com seu ambiente cooperativamente. O objetivo principal deste trabalho é realizar uma revisão do *framework* AMAM, assim como propor novos recursos que permitam dinamizar e aperfeiçoar o processo de solução.

Como outros *frameworks* disponíveis na literatura, o *framework* AMAM apresenta características comuns, como:

(i) metaheurísticas pré-implementadas para teste e reuso;

(ii) apoio à avaliação e comparação de diferentes métodos e

(iii) facilidade no desenvolvimento de uma metaheurística em particular e sua adequação ao problema tratado.

Além dessas características, o *framework* apresentado aqui tem a força da hibridização de metaheurísticas através da abordagem cooperativa paralela gerenciada por Sistemas MultiAgentes (MAS). MAS são usados aqui como ligação entre diferentes metaheurísticas para resolver problemas de otimização. Cada agente é responsável por realizar sua própria tarefa e, ao mesmo tempo, por usar as soluções fornecidas por outros agentes para melhorar suas próprias soluções. Nesta abordagem, os agentes interagem e trabalham juntos para atingir um objetivo pré-definido. A interação entre os vários agentes ocorre por meio de um conjunto de soluções. A comunicação é regida pelas regras de acesso a este conjunto de soluções, tanto para escrita quanto para leitura, visando garantir diversidade no compartilhamento de informações da busca. Vários trabalhos anteriores (Fernandes et al., 2009; Silva et al., 2014, 2015, 2018, 2019) apresentaram diferentes etapas do desenvolvimento deste *framework*, sendo boa parte deles (Silva et al., 2014, 2015, 2018, 2019) frutos do presente trabalho de tese.

Considerando que grande parte dos bons resultados encontrados para problemas de otimização conhecidos são obtidos com adaptações dos algoritmos de solução direcionadas às características específicas do problema, um dos principais questionamentos realizados em relação ao uso de *frameworks* neste contexto está na generalização dos métodos para que atuem em qualquer problema. Sendo assim, os *frameworks* geralmente perdem na qualidade da soluções geradas quando comparados a uma estrutura de solução específica, embora, por outro lado, ofereçam uma gama de vantagens na sua utilização.

Desta forma, esta tese propõe a incorporação de capacidades auto-adaptativas aos agentes do *framework*, uma questão discutida inicialmente em Silva et al. (2015). O objetivo é

inserir no agente habilidades que permitam que ele se adapte a características específicas do problema. Neste cenário, o agente modifica suas ações com base nas experiências adquiridas na interação com os demais agentes e com o meio ambiente. Esta experiência é obtida usando dois conceitos de aprendizado de máquina ou, mais especificamente, Aprendizado por Reforço (*Reinforcement Learning* - RL), segundo Sutton e Barto (1998). São eles (i) o algoritmo *Q-Learning* (Watkins e Dayan, 1992) e (ii) Autômatos de Aprendizagem (*Learning Automata* - LA) Narendra e Thathachar (1974). A estratégia de aprendizado é definida aqui para cada agente individualmente.

A principal motivação deste trabalho é apresentar o *framework* AMAM como uma ferramenta de *software* consolidada para resolver problemas de otimização combinatória usando metaheurísticas, incluindo características importantes, como a autonomia dos agentes, sem qualquer tipo de coordenação explícita entre eles. Ao mesmo tempo, este *framework* introduz, segundo o conhecimento da autora, o primeiro uso de aprendizado por reforço para *frameworks* especializados na resolução de problemas de otimização combinatória. A capacidade adaptativa embutida permite que os agentes se ajustem a problemas específicos, proporcionando o melhor desempenho deste no *framework*.

Para melhor introdução e validação do *framework* AMAM, este trabalho instancia o *framework* para dois problemas de otimização clássicos e bem conhecidos. O primeiro é o Problema de Roteamento de Veículos com Janela de Tempo (em inglês, *Vehicle Routing Problem with Time Window* – VRPTW) (Toth e Vigo, 2002) e o segundo é o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempo de Configuração dependente de Sequência (em inglês, *Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times* – UPMSP-ST) (Allahverdi et al., 2008; Allahverdi, 2015). Ambos, VRPTW e UPMSP-ST, são problemas NP-difíceis e, consequentemente, adequados para serem usados aqui para demonstrar as potencialidades do *framework* AMAM.

Estas instanciações são utilizadas nos experimentos computacionais para avaliar o desempenho do *framework* e dos recursos adicionados a ele. Os experimentos foram conduzidos com o objetivo de analisar o desempenho dos agentes do *framework*, tanto individualmente, quanto em equipe; tanto em relação aos agentes adaptativos propostos aqui, quanto em relação à estrutura cooperativa apresentada. O objetivo principal destes experimentos é avaliar se a forma de aprendizagem, embutida no agente, tem influência direta no desempenho do *framework* em relação à qualidade dos resultados obtidos, tanto do ponto de vista individual quanto do ponto de vista do trabalho em equipe. Além disso, propõe-se avaliar se o comportamento cooperativo influencia diretamente a qualidade das soluções.

1.2 Objetivos

1.2.1 Objetivo Geral

A presente proposta tem, como objetivo geral, o Projeto e a Implementação de um *Framework* Multiagente para Otimização Combinatória usando Metaheurísticas que atenda aos seguintes requisitos:

(i) O *framework* deve ser capaz de solucionar Problemas de Otimização Combinatória;

(ii) O *framework* deve incorporar conceitos de Sistemas Multiagentes, com enfoque especial no desenvolvimento de agentes autônomos e no comportamento cooperativo entre os agentes;

(iii) O *framework* deve possuir agentes adaptativos, com estruturas que permitam aquisição de experiências com base nas próprias ações do agente e que se adapte às características específicas dos diferentes problemas a serem solucionados.

1.2.2 Objetivos Específicos

Para que este objetivo geral seja alcançado, é necessário atingir os seguintes objetivos específicos:

(i) Conhecer a literatura concernente aos principais temas envolvidos nesta tese, os quais são: Hibridização de Metaheurísticas, Metaheurísticas Cooperativas e Paralelas, *Frameworks* para Otimização usando Metaheurística, Sistemas Multiagentes, e Aprendizado de Máquina;

(ii) Projetar o *Framework* proposto, a partir de uma linguagem de modelagem Orientada a Objetos e do uso de Padrões de Projeto de *Software*, que permitirão que o *framework* seja facilmente estendido;

(iii) Projetar o agente adaptativo, definindo estruturas de aprendizado que permitam a tomada de decisão com base nas experiências do próprio agente;

(iv) Implementar o *Framework*, com o desenvolvimento da estrutura de Otimização, da estrutura Multiagente e dos agentes adaptativos, atendendo aos requisitos já estabelecidos;

(v) Testar e validar o *Framework* desenvolvido, solucionando Problemas de Otimização Combinatória amplamente conhecidos, buscando demonstrar a flexibilidade da proposta;

(vi) Analisar estatisticamente os resultados obtidos.

1.3 Contribuições

As principais contribuições desta tese são:

(i) Revisão da estrutura multiagente do *framework* AMAM, através do aumento da autonomia dos agentes, sem qualquer tipo de coordenação explícita entre eles.

(ii) Atribuição de capacidades adaptativas ao agente do *framework*, usando os conceitos de Aprendizado de Máquina, permitindo ao agente melhor se adaptar aos parâmetros específicos do problema a ser tratado pelo *framework*;

(iii) Aprimoramento da cooperação entre os agentes, através da inclusão de critérios de inserção de novas soluções na estrutura cooperativa, buscando maior diversidade de soluções;

(iv) Demonstração de como o aprimoramento das habilidades individuais dos agentes influencia diretamente o desempenho cooperativo destes.

1.4 Estrutura do Trabalho

O restante desta tese está organizada como segue. O Capítulo 2 apresenta o referencial teórico base para o desenvolvimento desta pesquisa, mostrando uma visão geral da hibridização de metaheurísticas, um conjunto de trabalhos correlatos a este e um comparativo detalhado entre estes trabalhos. Adicionalmente, o Capítulo 2 apresenta importantes conceitos de aprendizado de máquina.

O Capítulo 3 aborda os problemas de otimização utilizados como estudos de caso nesta tese. Nele são apresentados o Problema de Roteamento de Veículos com Janelas de Tempo (VRPTW) e o Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempos de Configuração Dependentes da Sequência (UPMSP-ST).

O Capítulo 4 descreve o *framework* AMAM, seus principais componentes e detalha as contribuições propostas neste trabalho. No Capítulo 5 são descritos todos os detalhes de implementação, instâncias teste, parâmetros utilizados nos experimentos computacionais realizados, assim como os resultados obtidos e a discussão destes.

Finalmente, o Capítulo 6 apresenta as conclusões finais e discute direções futuras de pesquisa.

Capítulo 2

Fundamentação Teórica

Neste Capítulo é realizada uma ampla revisão dos conceitos base para o desenvolvimento desta tese e dos principais trabalhos relacionados a este, disponíveis na literatura. Este capítulo está organizado como segue. Na Seção 2.1 é apresentada uma visão geral sobre hibridização de metaheurísticas. Na Seção 2.2 são discutidos os principais trabalhos correlatos encontrados na literatura recente. A Seção 2.3 apresenta uma análise comparativa dos trabalhos correlatos apresentados na seção anterior. A Seção 2.4 descreve conceitos relevantes de aprendizado de máquina, enquanto a Seção 2.5 encerra com as considerações sobre o capítulo.

2.1 Uma visão geral sobre a Hibridização de Metaheurísticas

Atualmente, a hibridização de metaheurísticas está presente em grande parte dos trabalhos científicos voltados à resolução de problemas de otimização. A principal razão para o uso crescente dessa técnica são os bons resultados obtidos. De acordo com Cotta et al. (2005), Blum et al. (2011) e Boussaid et al. (2013), as hibridizações são responsáveis por muitos dos melhores resultados na literatura para várias classes de problemas de otimização.

Em grande parte das revisões bibliográficas relacionadas a este tópico, o termo *Metaheurística Híbrida* é definido como a combinação de metaheurísticas com outras metaheurísticas e/ou com outros métodos, geralmente de diversas áreas de Inteligência Computacional e Pesquisa Operacional. Raidl (2006) adiciona a esta definição a possibilidade de combinar não apenas estruturas completas de metaheurísticas, mas também componentes de metaheurísticas. Essa combinação visa proporcionar uma sinergia entre metaheurísticas, a partir da junção das características de cada algoritmo, buscando, assim, obter melhor desempenho quando comparado ao uso de cada técnica em sua forma pura ou isolada.

Revisões relacionadas a metaheurísticas híbridas podem ser encontradas em Talbi (2002), Cotta et al. (2005), Raidl (2006), Blum e Roli (2008), Blum et al. (2010), Blum et al. (2011) e Silva et al. (2018). Da mesma forma, várias publicações destacam a combinação de metaheurísticas com outros métodos, como métodos exatos, como em Puchinger e Raidl (2005) e Jourdan et al. (2009), ou a combinação de métodos aproximados específicos, como algoritmos evolutivos, de acordo com El-Mihoub et al. (2006) e Rodriguez et al. (2012).

Autores como Talbi (2002), El-Abd e Kamel (2005), Cotta et al. (2005), Raidl (2006) e Jourdan et al. (2009) propuseram taxonomias diferentes para metaheurísticas híbridas. Es-

ses autores definem classificações que permitem a comparação de metaheurísticas híbridas qualitativamente, bem como a identificação de semelhanças e componentes-chave.

A taxonomia introduzida por Talbi (2002) é utilizada como base para várias das propostas apresentadas posteriormente relacionadas a novas taxonomias, como Cotta et al. (2005); El-Abd e Kamel (2005); Raidl (2006), e envolve questões relacionadas tanto ao projeto quanto à estrutura do algoritmo, à implementação, ao *hardware*, à linguagem de programação e ao ambiente de execução. Do ponto de vista de projeto, duas questões são consideradas, uma subordinada à outra:

(i) qual o nível de hibridização realizado? e

(ii) qual é a ordem em que metaheurísticas são executadas?

A classificação de Raidl (2006) propõe adicionar mais duas questões àquelas levantadas anteriormente:

(iii) o que é hibridizado? e

(iv) qual estratégia de controle é usada?

Em relação ao nível de hibridização, dois tipos são considerados nesta taxonomia:

(i) Hibridização de Baixo Nível, na qual uma parte de uma metaheurística é substituída por outra metaheurística, ou, em vista da extensão deste conceito introduzido por Raidl (2006), a possibilidade de combinação de componentes de uma metaheurística; e

(ii) Hibridização de Alto Nível, na qual as metaheurísticas envolvidas são auto-suficientes. A hibridização de baixo nível se assemelha ao projeto de um novo algoritmo, uma vez que leva à aplicação de um único elemento resultante da combinação.

Em relação ao tipo de algoritmo hibridizado, três categorias são identificadas:

(i) metaheurísticas com metaheurísticas;

(ii) metaheurísticas com algoritmos específicos/simulação; e

(iii) metaheurísticas com outras técnicas de Pesquisa Operacional e Inteligência Artificial, como métodos de programação matemática, redes neurais e lógica *fuzzy*, por exemplo.

A estratégia de controle pode ser coercitiva ou cooperativa, sendo a primeira relacionada a situações em que um algoritmo é subordinado a outro, e a segunda relacionada a combinações nas quais os algoritmos trocam informações, mas um não é parte do outro. As metaheurísticas baseadas na busca cooperativa são atualmente um campo importante na Otimização, recebendo grande destaque nos últimos anos. O uso de conceitos multiagentes na hibridização de metaheurísticas aplica a estratégia de controle cooperativo.

A ordem de execução é classificada de três maneiras:

(i) bloco, em que cada metaheurística é executada uma vez, em sequência, e a informação é passada em apenas uma direção;

(ii) intercalado, no qual as metaheurísticas são executadas uma após a outra, mas, ao contrário da classificação anterior, os algoritmos devem interagir de forma mais elaborada; e

(iii) paralelo, no qual as metaheurísticas são executadas simultaneamente e trocam informações periodicamente.

Essa classificação também apresenta características que distinguem as propostas relacionadas à ordem de execução paralela, levando em consideração questões como *hardware*, arquitetura, memória e outros elementos que a influenciam. Metaheurísticas paralelas também recebem grande destaque na literatura recente. A ordem de execução paralela tem sido usada para reduzir o tempo computacional gasto para encontrar a solução, principalmente em problemas mais complexos. Em muitos trabalhos recentes, as propostas que se ajustam à hibridização de alto nível fazem uso da estratégia de controle cooperativo por meio da ordem de execução paralela, que permite que algoritmos auto-suficientes sejam executados simultaneamente e troquem informações sobre a busca.

No entanto, essas classificações têm dificuldades em categorizar as metaheurísticas híbridas desenvolvidas empregando uma hierarquia simples. As dificuldades se devem ao fato de que essas implementações não pertencem a uma única classe de metaheurísticas e residem, em particular, na maneira como as características dos algoritmos podem se sobrepor. Da mesma forma, muitos autores tratam as subclasses de metaheurísticas híbridas, como metaheurísticas cooperativas, metaheurísticas paralelas e hiperheurísticas, como classes independentes, propondo definições e classificações específicas (Crainic e Toulouse, 2003; Alba, 2005; Cotta et al., 2005; El-Abd e Kamel, 2005; Crainic e Toulouse, 2010). Essas subáreas destacam-se das demais porque estão se desenvolvendo intensivamente e se tornando cada dia mais importantes no contexto da otimização. Essa importância é discutida e enfatizada neste texto a seguir, ainda nesta mesma seção.

Na sequência, a Subseção 2.1.1 analisa questões relacionadas à cooperação entre metaheurísticas. A Subseção 2.1.2 apresenta uma breve análise sobre metaheurísticas paralelas. Além disso, a Subseção 2.1.3 discute as hiperheurísticas como uma proposta de hibridização de metaheurísticas.

2.1.1 Metaheurísticas Cooperativas

Vários autores na literatura enfatizam a cooperação entre metaheurísticas. Revisões e classificações relacionadas ao tópico podem ser encontradas em Blum e Roli (2003) e El-Abd e Kamel (2005). Cada um desses autores apresenta seu próprio conceito para a ideia de busca cooperativa e o contexto no qual ela é usada. Para ajudar na integridade desta tese, alguns desses conceitos são apresentados abaixo.

Blum e Roli (2003) definem busca cooperativa como um processo de resolução de problemas de otimização, realizado por vários algoritmos (ou instâncias do mesmo algoritmo), que compartilham informações sobe o espaço de busca, como, por exemplo, o estágio de busca, soluções e sub-problemas. El-Abd e Kamel (2005) também descrevem os algoritmos de busca cooperativa como um tema particular e apresentam definições e taxonomias específicas. Segundo esses autores:

> *a busca cooperativa é uma categoria de algoritmos paralelos, na qual vários algoritmos de busca são executados em paralelo para resolver o problema de otimização em questão.*

Crainic e Toulouse (2003) consideram *cooperação* como uma estratégia para solução de problemas e projeto de algoritmos. Estes autores afirmam que existem duas características comuns a todas as diferentes formas de cooperação:

> *(i) conjunto de programas altamente autônomos (APs), cada um implementando um método de solução particular; e (ii) um esquema de cooperação combinando esses programas em uma única estratégia de solução.*

A partir dos conceitos acima, podemos definir *cooperação*, no contexto de Otimização, como o uso de diferentes agentes trabalhando em conjunto, trocando informações para alcançar um objetivo comum. Frequentemente, esse objetivo é encontrar uma solução para um determinado problema. No caso aqui considerado, os agentes são entendidos como elementos que implementam, cada um deles, um método de busca para a solução do problema e, além disso, atuam de forma autônoma.

De acordo com as taxonomias introduzidas por Talbi (2002), a busca cooperativa pode ser realizada de forma sequencial ou de forma paralela. Alguns exemplos de trabalhos que mostram como realizar busca cooperativa são Alba et al. (2006); Villaverde et al. (2012); Amaya et al. (2010); Jin et al. (2014).

No contexto das metaheurísticas cooperativas, é importante destacar a Co-evolução Cooperativa. Co-evolução Cooperativa é um paradigma para resolver problemas de otimização complexos de alta dimensão, nos quais o problema de interesse é particionado em subproblemas de menor dimensão, cada um sendo resolvido por um algoritmo diferente em subpopulações específicas (Potter e De Jong, 2000). Assim, não se trata simplesmente de dividir a população total em subpopulações, mas de dividir o problema em subproblemas e resolvê-los de acordo com condições particulares. A questão imediata é a coordenação entre os vários subproblemas. A cooperação entre subpopulações ocorre na avaliação cooperativa de cada indivíduo. Essa interação entre subpopulações acontece dentro de um domínio compartilhado. Nesse caso, o objetivo global de todo o sistema é essencialmente o objetivo local de cada subcomponente. De acordo com Potter e De Jong (2000) e de Gong et al. (2015), ao desenvolver algoritmos co-evolucionários, quatro questões precisam ser abordadas:

(i) a decomposição do problema;

(ii) a evolução dos subcomponentes interdependentes;

(iii) a avaliação; e

(iv) a manutenção da diversidade.

Um trabalho recente a esse respeito é Gong et al. (2015) e uma boa discussão é encontrada em Byrski et al. (2015).

2.1.2 Metaheurísticas Paralelas

Crainic e Toulouse (2010) definem computação paralela/distribuída como:

> *vários processos que trabalham simultaneamente em vários processadores, resolvendo uma dada instância de um problema.*

Ainda de acordo com estes autores, o paralelismo

> *surge a partir da decomposição da carga computacional total e da distribuição das tarefas resultantes aos processadores disponíveis. A decomposição pode dizer respeito ao algoritmo, aos dados da instância do problema ou à estrutura do problema.*

Assim, quanto mais apta a decomposição de um algoritmo, mais fácil será o processo de paralelização de suas funções ou dados. Vários artigos dedicados a metaheurísticas paralelas estão disponíveis na literatura, destacando-se os encontrados em Trienekens e Bruin (1992), Crainic e Gendreau (2002), Alba (2005) e El-Abd e Kamel (2005).

Metaheurísticas, no entanto, como apontado por Crainic e Toulouse (2010), pertencem à categoria de algoritmos que são difíceis de serem paralelizados. Essa dificuldade se deve principalmente à forte dependência entre cada iteração das metaheurísticas. Na maioria dos métodos baseados em trajetória, por exemplo, há uma forte dependência entre as sucessivas iterações. Uma situação semelhante de dependência ocorre com os métodos evolutivos (isto é, as metaheurísticas baseadas em população), em relação à passagem de uma geração para outra.

Metaheurísticas híbridas, por outro lado, especialmente aquelas baseadas na busca cooperativa, facilitam o processo de paralelização. Algoritmos diferentes (ou instâncias de um mesmo algoritmo), rodando de forma independente, certamente exploram diferentes regiões do espaço de busca e se tornam uma fonte de paralelismo fácil de ser implementada.

Em metaheurísticas paralelas, vários métodos podem ser usados para executar simultaneamente. Cabe ao projetista determinar como controlar o processo de solução e como as informações serão trocadas entre as metaheurísticas. As técnicas de busca cooperativa são frequentemente implementadas em paralelo. No entanto, deve ficar claro que o uso de técnicas de paralelização de metaheurísticas não implica necessariamente na existência de cooperação entre metaheurísticas. A decomposição de um algoritmo em processos separados, sem qualquer cooperação, se enquadra na categoria de metaheurísticas paralelas (assim como algoritmos distribuídos), mas não pode ser vista como metaheurísticas cooperativas, já que não há troca de informações durante o processo de busca. Portanto, as hibridizações sem cooperação também podem ser implementadas em paralelo, proporcionando apenas a redução do tempo de execução.

Exemplos de trabalhos mostrando o uso de metaheurísticas paralelas são Toutouh e Alba (2012), Alirezaei et al. (2012) e González-Álvarez e Vega-Rodríguez (2013).

2.1.3 Hiperheurísticas e Hibridização de Metaheurísticas

No contexto de Otimização, o termo *hiperheurística* é usado para descrever *heurísticas para escolher heurísticas*, como dito por Cowling et al. (2001), ou *uma metodologia automatizada para selecionar ou gerar heurísticas para resolver problemas computacionais difíceis*, como encontrado em Chakhlevitch e Cowling (2008). De acordo com Burke et al. (2003),

> *uma hiperheurística é uma abordagem de alto nível que, dada uma instância de problema particular e um número de heurísticas de baixo nível, pode selecionar e aplicar uma heurística de baixo nível apropriada a cada ponto de decisão.*

Desse modo, a hiperheurística opera sobre um conjunto de heurísticas, selecionando, durante a busca, uma heurística de baixo nível, mais apropriada para encontrar a solução dentre as disponíveis. A hiperheurística lida, portanto, com dois espaços de busca:

(i) o espaço de busca da hiperheurística; e

(ii) o espaço de busca do problema.

A Figura 2.1 mostra a relação entre os espaços de busca da hiperheurística, como apontado por Burke et al. (2007). O espaço de busca da hiperheurística envolve o espaço das heurísticas, ou seja, as heurísticas disponíveis para resolver o problema. O espaço de busca do problema é formado pelas soluções viáveis do problema tratado.

Como resultado, à luz do conceito adotado para hibridização de metaheurísticas, a hiperheurística pode ser vista como uma importante categoria de hibridismo, envolvendo associações, combinações e cooperação (em vários níveis hierárquicos) entre metaheurísticas para resolver problemas de otimização.

Como as metaheurísticas, as hiperheurísticas não são específicas a um problema. Assim, as decisões referentes à seleção das heurísticas utilizadas são realizadas com base em medidas independentes do problema, como, por exemplo, a qualidade da solução (Özcan et al., 2008).

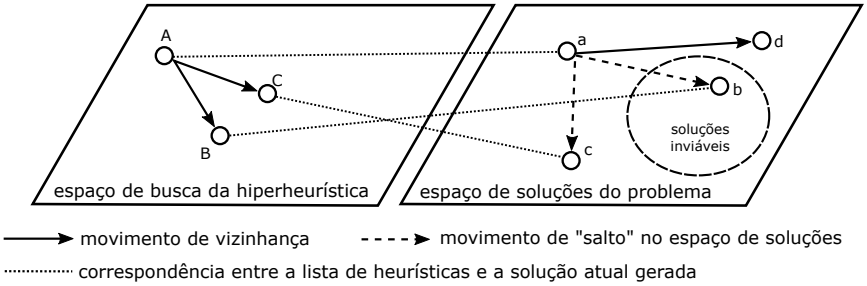


Figura 2.1: A relação entre o espaço de busca da hiperheurística e o espaço de soluções do problema (Burke et al., 2007).

Hiperheurísticas são boas alternativas nos casos em que é necessário um ajuste fino dos parâmetros envolvidos, permitindo a implementação e avaliação de diferentes heurísticas para um problema. Alguns exemplos usando hiperheurísticas são mostrados em Cowling et al. (2001); Burke et al. (2007); Malek (2010); Özcan e Kheiri (2012).

2.2 Trabalhos Correlatos

Uma revisão detalhada dos principais *frameworks* disponíveis na literatura é realizada, nesta seção, com propósito de identificar as lacunas existentes no desenvolvimento destas ferramentas e distinguir as características desejáveis a um *framework* para otimização usando metaheurísticas. Esta investigação do estado da arte permitiu listar os principais trabalhos correlatos a este. O enfoque dado a esta análise é direcionado à hibridização de metaheurísticas, em especial àquelas que utilizam os conceitos de metaheurísticas cooperativas e metaheurísticas paralelas.

Esta seção está organizada da seguinte forma. Os *frameworks* apresentados aqui são divididos em duas categorias: *Frameworks* para Otimização usando Metaheurísticas e *Frameworks* Multiagentes para Otimização usando Metaheurísticas. Estas ferramentas são descritas nas seções 2.2.1 e 2.2.2, respectivamente. Em seguida, na Seção 2.3, é feita uma análise comparativa das ferramentas listadas na presente seção.

2.2.1 *Frameworks* para Otimização usando Metaheurísticas

Na busca por melhores soluções usando metaheurísticas, estruturas genéricas, chamadas *frameworks*, têm sido utilizadas. Essas estruturas têm a flexibilidade como característica fundamental e, assim, permitem a adição de novos métodos, sem exigir qualquer esforço para refazer toda a implementação. Além disso, facilitam o desenvolvimento de metaheurísticas e, ao mesmo tempo, diversos trabalhos incorporam o conceito de hibridização. *Frameworks* voltados para otimização com metaheurísticas são o assunto central da análise nesta seção.

Examinando o significado da palavra *framework* no Dicionário Oxford de Ciência da Computação (Butterfield e Ngondi, 2016):

> *"Um modelo para o desenvolvimento de aplicativos ou componentes de software.*

Na literatura, vários autores apresentam definições para o termo *framework*, que envolvem principalmente o conceito de orientação a objetos:

> *Um framework é um conjunto de classes que incorpora um projeto abstrato para soluções para uma família de problemas relacionados.* (Johnson e Foote, 1988)

> *Um framework é um conjunto de classes cooperantes que compõem um projeto reutilizável para uma classe específica de software.* (Gamma et al., 1995)

Sendo assim, um *framework* pode ser entendido como um esqueleto que fornece funcionalidade genérica para resolver problemas em um domínio específico pela junção de vários códigos comuns. Um *framework* pré-define a estrutura geral de como um aplicativo deve ser construído a partir dele, definindo classes e objetos, a colaboração entre essas classes e as *threads* de controle. Uma aplicação específica pode ser desenvolvida com a customização do *framework*, baseada na definição de subclasses de classes abstratas já definidas. Assim, cabe ao usuário do *framework* desenvolver os elementos específicos de sua aplicação.

Um *Framework* para Otimização usando Metaheurísticas (*Metaheuristic Optimization Framework* - MOF), como mencionado em Parejo et al. (2012), é uma ferramenta de *software* que fornece implementações dos principais métodos metaheurísticos por meio de códigos reutilizáveis que facilitam o desenvolvimento de aplicativos para problemas de otimização.

O uso de *frameworks* permite, para pesquisadores da área de Otimização, inúmeras vantagens, como metaheurísticas pré-implementadas para teste e reuso; apoio à avaliação e comparação de diferentes métodos; facilidade de desenvolvimento de uma metaheurística particular e sua adequação ao problema. Adicionalmente, a hibridização de metaheurística também pode ser facilitada com o uso de *frameworks*. As propostas de *frameworks* que possuem esse recurso se diferem na abordagem usada na organização das metaheurísticas e na maneira como elas interagem umas com as outras.

Numerosos *frameworks* para resolução de problemas usando metaheurísticas estão disponíveis na literatura, a maior parte deles com características e propostas similares. As subseções a seguir apresentam os quinze *Frameworks* para Otimização usando Metaheurística mais referenciados na literatura, organizados em ordem alfabética com relação à sua sigla. Além disso, as principais características desses *frameworks* são avaliadas na Seção 2.3.

2.2.1.1 `EasyLocal++`

Em Di Gaspero e Schaerf (2001, 2002, 2003), o *framework* `EasyLocal++` é apresentado. Ele foi desenvolvido em linguagem `C++`, para projeto e análise de algoritmos de busca local visando solucionar problemas de Otimização Combinatória. O objetivo do *framework* é capturar os principais recursos técnicos da busca local e suas combinações, através de um projeto orientado a objetos.

Como é especializado apenas em métodos de busca local, o *framework* `EasyLocal++` não oferece a diversidade de métodos que outras estruturas da literatura possuem. A hibridização suportada pelo `EasyLocal++` só é possível a partir de modelos de algoritmos híbridos pré-definidos e não é simples adaptar para novas estruturas. Di Gaspero e Schaerf (2003) é a última publicação relacionada a este *framework*. No entanto, o sítio de internet associado e o próprio *framework* foram atualizados nos últimos anos, sendo a última atualização lançada em fevereiro de 2019. A versão mais recente (versão 3.0) do *framework* `EasyLocal++` está disponível em `https://bitbucket.org/satt/easylocal-3`.

2.2.1.2 `ECJ`

`ECJ` (*Evolutionary Computing system written in Java*, em português, Sistema de Computação Evolutiva escrito em Java), é o *framework* proposto em White (2012), que possibilita pesquisas em Computação Evolutiva. Nesse sentido, o `ECJ` é uma ferramenta genérica que visa disponibilizar a implementação de todas as formas de computação evolutiva.

Desenvolvido em Java, estratégias como padrões de projeto e herança, usados na definição do *framework*, permitem que ele suporte muitos métodos alternativos para funções comuns, como inicialização de população e operadores de mutação e seleção, sem esforço adicional do usuário.

Este *framework* oferece, como recursos adicionais, uma interface gráfica e suporte para paralelismo e computação distribuída. A interface gráfica do `ECJ` permite carregar e executar algoritmos com base em arquivos de parâmetros, arquivos de *checkpoint*, edição de parâmetros e gráficos de estatísticas. Em relação às funções de execução paralela e distribuída, o *framework* `ECJ` possui suporte *multi-thread* e a execução é baseada em arquitetura *master-slave* e modelos de ilhas. Por outro lado, essa estrutura não permite a hibridização de metaheurísticas.

Luke (2017) é a publicação mais recente sobre o `ECJ`. É apresentada uma proposta para expandir o `ECJ`, a fim de atualizar o padrão Java do `ECJ`; realizar mudanças na arquitetura do *framework*, incorporar novas metaheurísticas, permitir a hibridização de metaheurísticas, o tratamento de metaheurísticas multiobjetivas e a análise de resultados, bem como aumentar a usabilidade do *framework*. A última versão disponível do `ECJ` é o número 26, novembro de 2018, e pode ser encontrada no sítio do projeto `http://cs.gmu.edu/~eclab/projects/ecj/`.

2.2.1.3 *Evolutionary Algorithms Framework* (`EvA2`)

Kronfeld et al. (2010) apresentam um *framework* chamado `EvA2` (***Ev**olutionary **A**lgorithms framework*, em português, *Framework* de Algoritmos Evolutivos). `EvA2` é um *framework* de otimização metaheurística, direcionado a Algoritmos Evolutivos, derivado da chamada *JavaEvA optimization toolbox*. A arquitetura do *framework* `EvA2` é implementada em Java e tem como principal característica uma estrutura cliente-servidor, que permite que sua busca seja distribuída pelos nós de uma rede.

EvA2 fornece a implementação de várias estratégias de otimização, preferencialmente baseadas na população. Embora sua ênfase esteja nos métodos evolutivos, o *framework* EvA2 implementa métodos clássicos e de trajetória, como método Simplex Nelder-Mead e *Simulated Annealing*.

Além da estrutura cliente-servidor, outro recurso oferecido pelo EvA2 é a interface gráfica. Essa interface permite que os usuários que não conhecem a teoria dos algoritmos evolutivos usem esses métodos para resolver um problema específico. Essa estrutura não oferece nenhuma alternativa para a hibridização de metaheurísticas.

A versão mais recente do EvA2, disponível em http://www.ra.cs.uni-tuebingen.de/software/EvA2/, é 2.2.0, de dezembro de 2015.

2.2.1.4 FOM: *Framework for Metaheuristic Optimization*

Em Parejo et al. (2003), uma estrutura orientada a objetos para otimização usando metaheurísticas, chamada FOM (*Framework for Metaheuristic Optimization*), é apresentada. O principal objetivo deste *framework* é propor uma estrutura geral que separe, por meio de interfaces pré-definidas, o problema a ser resolvido dos algoritmos de solução envolvidos.

FOM oferece, como diferencial, a possibilidade de ajustar parâmetros através de um elemento chamado Observadores. Esses elementos seguem a evolução dos parâmetros das metaheurísticas, em cada iteração, e utilizam essas informações, através de elementos chamados Visualizadores, para gerar gráficos de evolução ou para armazenar os valores obtidos.

Dois problemas clássicos de Pesquisa Operacional (problema SAT e problema do caixeiro viajante – TSP) são usados para o desenvolvimento de aplicativos em Parejo et al. (2003). A partir dos resultados obtidos nestas aplicações, os autores destacam que o uso do *framework* FOM torna as implementações menos eficientes, mesmo que a reutilização de código reduza o tempo de desenvolvimento e induza uma estrutura correta e robusta na implementação dos métodos.

O FOM também oferece interface gráfica de usuário, que permite a instanciação de metaheurísticas, com as características desejadas para a solução do problema selecionado pelo usuário. No que diz respeito à hibridização de metaheurísticas, não oferece qualquer possibilidade deste tipo de combinação.

Embora o último artigo relacionado à estrutura seja datado de 2003, seu desenvolvimento continuou, sendo a última atualização do código realizada em 2013. A versão mais recente do FOM (versão 0.5, de julho de 2013) está disponível em http://www.isa.us.es/fom sob a licença GNU LGPL.

2.2.1.5 HeuristicLab

Iniciado em 2002, o ambiente de *software* HeuristicLab foi desenvolvido no Laboratório de Algoritmos Heurísticos e Evolutivos (*Heuristic and Evolutionary Algorithm Laboratory* - HEAL) e foi apresentado em Wagner e Affenzeller (2004, 2005); Wagner et al. (2010, 2014). É um sistema de *software* extensível e flexível para heurísticas e algoritmos evolutivos. Entre outros recursos, o sistema facilita o ajuste do algoritmo a um problema específico, permite a implementação de vários algoritmos e execução paralela e/ou distribuída desses algoritmos.

Destaca-se pelo número e variedade de técnicas implementadas, bem como pela variedade de problemas abordados, incluindo também problemas multiobjetivos. No entanto, em relação à hibridização de metaheurísticas, o *framework* HeuristicLab não oferece qualquer possibilidade de implementação.

Esse *framework* foi estendido em Wagner e Affenzeller (2004) para uma versão inspirada em computação *grid*, chamada `HeuristicLab Grid`. Nesta versão, cada computador participante recebe uma instalação de cliente (implementada em `C#`), que pesquisa as tarefas em um servidor de banco de dados central. As tarefas são processadas localmente e os resultados obtidos são armazenados no banco de dados central. Além disso, `HeuristicLab Grid` inclui serviços da Web que permitem a construção de diferentes tipos de interface.

Uma revisão detalhada do desenvolvimento do *framework* `HeuristicLab` é apresentada em Wagner et al. (2014); Elyasaf e Sipper (2014), descrevendo as características de projeto de todas as versões disponíveis desde 2002. Segundo os autores, o `HeuristicLab` tem sido amplamente utilizado pelo grupo de pesquisa do Laboratório HEAL. Os autores também afirmam que o `HeuristicLab` é um projeto ainda em andamento. A última versão para download (versão 3.3.15) está disponível em `http://dev.heuristiclab.com/` e foi publicada em janeiro de 2018.

2.2.1.6 *Heuristic OpTimization FRAMEwork* (`HotFrame`)

Em Fink et al. (1999); Fink e Voß (2002), o *Heuristic OpTimization FRAMEwork* (`HotFrame`), ou, em português, *Framework* de Otimização Heurística, é proposto. É uma estrutura orientada a objetos, desenvolvida na linguagem `C++`. Fink e Voß (2002) descrevem detalhadamente sua arquitetura, mostrando componentes, implementação e aplicações. Essa descrição expõe como o *framework* abrange conceitos de metaheurísticas, como espaço de solução, estruturas de vizinhança e critérios tabus, levando a componentes genéricos e parametrizados. Desta forma, o *framework* `Hotframe` pode ser estendido em várias direções e permite a adição de novos problemas a serem resolvidos.

Em relação à usabilidade deste *framework*, os autores afirmam que o uso direto requer conhecimentos básicos de *frameworks* e metaheurísticas. Adicionalmente, foi gerada uma interface gráfica, a partir de um *software* gerador de interfaces, permitindo a configuração de experimentos e facilitando sua aplicação por usuários não especialistas.

O `Hotframe` é muito semelhante aos outros *frameworks* básicos, não oferecendo recursos adicionais que permitam potenciar a busca da solução. Com relação à hibridização de metaheurísticas, não oferece nenhuma alternativa para esse tipo de combinação. Embora publicações relacionadas a esse *framework* estejam disponíveis na literatura, nenhuma publicação sobre ele é conhecida após 2003. As informações referentes ao *download* do *framework* `Hotframe` estão disponíveis em: `https://www.bwl.uni-hamburg.de/en/iwi/forschung/projekte.html`. A última atualização foi realizada em julho de 2003.

2.2.1.7 `HyFlex`

`HyFlex` (*Hyper-heuristics Flexible framework*, em português, *Framework* Flexível para Hiperheurísticas), apresentado em Ochoa et al. (2012), é uma estrutura para implementação, teste e comparação de algoritmos de busca heurística iterativa de propósito geral e, em particular, de hiperheurísticas. Desenvolvido em Java, o `Hyflex` é projetado como uma biblioteca de classes modular e flexível. Segundo os autores, a modularidade é definida usando conceitos de decomposição de algoritmos de busca em duas partes principais: (i) uma parte de propósito geral: o algoritmo ou hiperheurística; e (ii) a parte específica do problema, fornecida pelo *framework* `HyFlex`.

Entre as características adicionais avaliadas neste estudo, o *framework* `Hyflex` destaca-se por oferecer a possibilidade de hibridização de metaheurísticas. No entanto, esta hibridização limita-se à maneira como as hiperheurísticas são definidas, não permitindo ao usuário

definir outras formas de combinação. Os outros recursos avançados, como paralelismo e interface gráfica, não são descritos pela documentação associada ao *framework*.

Várias publicações (Burke et al., 2010, 2011; Di Gaspero e Urli, 2011; Özcan e Kheiri, 2012) usam o *framework* `Hyflex` em suas propostas. A documentação da versão mais recente da estrutura do Hyflex (versão v1.1) pode ser encontrada em `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, lançada em fevereiro de 2012. Nenhuma nova publicação apresentando novas versões deste *framework* foi encontrada.

2.2.1.8 *Java Metaheuristics Search Framework* (JAMES)

De Beukelaer et al. (2015, 2017) apresenta o *framework* `JAMES` (*JAva MEtaheuristics Search*, em português, Busca Metaheurística em Java), em sua versão v1.1. O `JAMES` é uma estrutura para otimização discreta, focada no uso de algoritmos de busca local. Sua arquitetura define uma estrutura que separa fortemente a especificação do problema dos algoritmos de solução. A busca interage com o problema para obter, avaliar e validar soluções, sejam elas aleatórias ou construídas. A busca também inclui elementos chamados *ouvintes* (*listeners*), responsáveis por informar a ocorrência de certos eventos, por exemplo, quando uma nova melhor solução foi encontrada.

Como o *framework* é especializado em algoritmos de busca local, a estrutura de busca definida no `JAMES` é baseada no conceito de buscas em vizinhanças. Eles se movem pelo espaço de busca usando uma ou mais vizinhanças, ao longo de uma certa trajetória em direção a uma solução ótima.

Como recurso adicional, que potencializa a busca da solução, o *framework* `JAMES` apresenta apenas a possibilidade de utilizar *threads* na execução dos algoritmos, permitindo a execução paralela. Não oferece a possibilidade de combinar os métodos implementados e, portanto, a hibridização de metaheurísticas não é suportada.

O código e a documentação, bem como as implementações usadas para avaliá-lo, estão disponíveis em `http://www.jamesframework.org/`. A versão 1.1 é a última lançada, tendo sido apresentada em 16 de agosto de 2016.

2.2.1.9 *Java Class Library for Evolutionary Computation* (JCLEC)

Em Ventura et al. (2008); Ramírez et al. (2015, 2019) é apresentado um *framework* implementado em Java para o desenvolvimento de aplicações de computação evolucionárias. Ventura et al. (2008) propõem um *framework* chamado *Java Class Library for Evolutionary Computation* (JCLEC), em português, Biblioteca de Classes Java para Computação Evolutiva. Em Ramírez et al. (2015, 2019) uma extensão do *framework* é apresentada, fornecendo elementos e técnicas de busca adequados para otimização de múltiplos objetivos. Ramírez et al. (2019) mostra que o *framework* continua a se desenvolver e que um novo módulo, chamado `JCLEC-MO`, é definido.

O *framework* `JCLEC` oferece uma variedade de códigos e operadores genéticos, que podem ser combinados para resolver problemas de otimização. O `JCLEC-MO` permite aos seus usuários aplicar ou adaptar um grande número de algoritmos multiobjetivos com menor esforço de codificação. O módulo `JCLEC+` atua como ponte entre o `JCLEC` e o `JCLEC-MO`. Segundo os autores,

> *como um conjunto, o JCLEC-MO é construído sobre o* framework *JCLEC, como uma maneira de integrar e reutilizar elementos básicos. No entanto, o*

> *JCLEC-MO ainda mantém sua própria identidade, propondo uma nova arquitetura para atender aos requisitos específicos para otimização multiobjetivo.*

Dentre as características de destaque, os experimentos são criados usando arquivos de configuração `xml`, o que facilita a integração com outros sistemas. Oferece também suporte à analise dos resultados, a partir de bibliotecas externas e linguagens como `R`. A arquitetura do *framework* `JCLEC` inclui três módulos principais: (i) `JCLEC Core`: especifica os tipos de dados que definem a funcionalidade do *framework*; (ii) `Experiments Runner`: é um ambiente de execução dos algoritmos; e (iii) `GenLab`: é a interface gráfica do usuário.

É o *framework* com publicação mais recente da literatura, segundo o conhecimento da autora desta tese, sendo amplamente conhecido. Embora apresente uma interface amigável, oferecendo recursos que permitem a rápida criação de protótipos de aplicativos, essa estrutura é focada apenas em métodos de Computação Evolutiva e não oferece recursos adicionais que melhorem a busca da solução, como a hibridização.

Uma avaliação desta estrutura é realizada em Ventura et al. (2008), em que uma solução do problema da mochila 0/1 é mostrada. No entanto, os resultados não são apresentados. A versão mais recente do `JCLEC` é 4.0.0, lançada em julho de 2014 e está disponível em `http://jclec.sourceforge.net/` e o *framework* `JCLEC-MO`, versão 1.0, de 2019, está disponível em `https://www.uco.es/kdis/research/software/jclec-mo/`.

2.2.1.10 `jMetal` (*Metaheuristic Algorithms in Java*)

O *framework* `jMetal`, introduzido em Durillo et al. (2010) e Durillo e Nebro (2011), é uma estrutura orientada a objetos baseada na linguagem Java e especializada em problemas de otimização multiobjetivo. Desta forma, inclui um número significativo de métodos clássicos e modernos para resolver problemas de otimização multiobjetivo. Embora seja direcionado para problemas multiobjetivos, o *framework* `jMetal` também fornece implementações de métodos para problemas de otimização mono-objetivo.

Os principais elementos e operações do `jMetal`, que determinam a estrutura do *framework*, são definidos da seguinte forma: um algoritmo resolve um problema usando um (possivelmente mais de um) *SolutionSet* e um conjunto de objetos *Operator*. As classes *SolutionSet* e *Solution* permitem a implementação das metaheurísticas de base populacional, representando, respectivamente, população e indivíduos. A superclasse *Algorithm* é responsável por representar os otimizadores do *framework*, permitindo a definição de metaheurísticas, assim como os parâmetros e operadores envolvidos.

`JMetal` oferece recursos adicionais, como indicadores de qualidade de convergência de Pareto, testes estatísticos e interface gráfica. Os indicadores de qualidade que avaliam convergência e uniformidade dos gráficos de Pareto implementados são Distância Geracional Inversa (*Inverse Generational Distance* - IGD), Hipervolume (HV), Epsilon, *Spread* ou D e *Spread* Generalizado. A interface gráfica do `jMetal` permite a configuração e a implementação de estudos experimentais, bem como a determinação de quais testes estatísticos serão aplicados nos resultados finais. Os seguintes testes estatísticos estão disponíveis: Teste de Wilcoxon (gera automaticamente *script R*) e *Boxplots* (gera uma representação dos dados em gráficos de *boxplot*). O `JMetal` também permite a execução paralela e distribuída dos métodos implementados. Em relação à hibridização de metaheurísticas, não há possibilidade de realizar este formato de implementação.

Alguns recursos do `jMetal` são revisados em Nebro et al. (2015). O objetivo principal desta revisão é simplificar a estrutura e facilitar o uso do *framework*. Esse *framework*

ainda está em desenvolvimento, e sua versão mais recente, lançada em setembro de 2018, chamada `jMetal` 5.6, está disponível em `http://jmetal.github.io/jMetal/`.

2.2.1.11 `MALLBA`

Iniciado em 2000, o projeto `MALLBA`, apresentado em Alba et al. (2002), é uma biblioteca de esqueletos para Otimização (incluindo métodos exatos, heurísticos e híbridos), que permite execução paralela a partir de implementações sequenciais. Seus três ambientes-alvo são computadores sequenciais, LANs e WANs de estações de trabalho. Cada método de otimização é encapsulado em um esqueleto, que é uma estrutura que permite a criação de instâncias dos métodos disponíveis. Os esqueletos são implementados em classes de linguagem de programação `C++`.

O *framework* `MALLBA` oferece recursos relacionados ao paralelismo e à computação distribuída, não exigindo que o usuário tenha profundo conhecimento desse assunto. Além disso, a biblioteca `MALLBA` também oferece ferramentas que permitem a construção de esqueletos híbridos. No entanto, essas implementações são estruturas fechadas, não oferecendo ao usuário a possibilidade de definir novas formas de hibridização. `MALLBA LIBRARY` v2.0, a última versão deste projeto, foi apresentada em Alba et al. (2006, 2007). Está disponível para *download* em `http://neo.lcc.uma.es/mallba/easy-mallba/`, lançado em junho de 2006.

2.2.1.12 `OptFrame`

O *framework* `OptFrame` é introduzido em Coelho et al. (2010, 2011) e visa fornecer uma interface para elementos comuns de metaheurísticas baseados em metaheurísticas populacionais e de trajetória. Implementações em linguagem de programação `C++` de versões simples destes métodos estão disponíveis no `OptFrame`, permitindo ao usuário desenvolver versões mais eficientes, de acordo com o problema específico.

Um conceito importante na estrutura do `OptFrame` é o *Avaliador*. Ele encapsula as funções de avaliação da solução e permite a avaliação de funções mono ou multiobjetivo. O *framework* `OptFrame` oferece suporte ao paralelismo, seja com computadores com memória compartilhada ou memória distribuída. Possui um gerador paralelo de melhor vizinhança, que permite agilizar o processo, dividindo o espaço de busca entre os nós de processamento.

Na literatura, vários trabalhos mostram diferentes aplicações do `Optframe` para resolver problemas de otimização, dentre eles Coelho et al. (2009); Souza et al. (2010); Coelho et al. (2016a,b,c) e Coelho et al. (2017). O `Optframe` está em constante desenvolvimento e sua versão mais recente (versão 3.0), de novembro de 2018, está disponível em `https://github.com/OptFrame/optframe` sob a licença `GNU LGPLv3`.

2.2.1.13 `Opt4j`

Um *framework* usando metaheurísticas para otimização de tarefas complexas, chamada `Opt4j`, é apresentada em Lukasiewycz et al. (2011). O `Opt4j` é um *framework* para computação evolucionária desenvolvido em linguagem Java. A otimização é realizada a partir da decomposição de tarefas em sub-tarefas, que devem ser projetadas e desenvolvidas separadamente. Como essas sub-tarefas são normalmente correlacionadas, o *framework* usa conceitos de algoritmos evolutivos para separá-las em uma distinção estrita entre genótipo (representação genética) e fenótipo (representação de uma solução). Um decodificador é

usado para traduzir um genótipo em um fenótipo. Dessa forma, apenas fenótipos e decodificadores dependem do problema e precisam ser especificados.

Como o *framework* é focado em métodos de computação evolucionária, sua arquitetura básica tem, como elementos principais, a população e os indivíduos. Os operadores manipulam o genótipo do indivíduo para realizar a busca. Sendo assim, o `Opj4` não possui a flexibilidade para implementação de métodos que não pertencem a essa categoria e, como consequência, há dificuldades na implementação da hibridização.

O `Opt4j` permite a execução paralela e distribuída dos algoritmos implementados e fornece uma interface gráfica de usuário para facilitar as configurações e os testes. Todos os módulos e métodos disponíveis são listados automaticamente e nenhum código é necessário para visualizá-los. A configuração usada pode ser salva ou carregada de arquivos `XML`.

A versão atual desse *framework*, chamada `Opt4J` 3.1.4, foi lançada em novembro de 2015 e está disponível em `http://opt4j.sourceforge.net/`.

2.2.1.14 *Parallel and Distributed Evolving Objects* (`ParadisEO`)

O *software* `ParadisEO` (*Parallel and Distributed Evolving Objects*) é uma estrutura orientada a objeto do tipo caixa branca para projeto reutilizável de metaheurísticas paralelas e distribuídas, apresentada em Cahon et al. (2004), Liefooghe et al. (2011), Melab et al. (2013) e Humeau et al. (2013). O *framework* `ParadisEO` é atualmente composto por cinco módulos conectados:

- `ParadisEO-EO`: fornece um grande número de classes para o desenvolvimento de metaheurísticas baseadas em população;
- `ParadisEO-MO`: contém ferramentas para metaheurísticas de trajetória ou baseadas em um único ponto;
- `ParadisEO-MOEO`: dedica-se ao projeto e implementação de técnicas evolutivas para otimização multiobjetivo;
- `ParadisEO-PEO`: fornece classes para a implementação de metaheurísticas paralelas e distribuídas;
- `ParadiseEO-MO-GPU`: apresentado em Melab et al. (2013), o principal objetivo é explorar o poder da moderna unidade de processamento gráfico (GPU), com foco em metaheurísticas de trajetória.

O *framework* `ParadisEO` se destaca pela cobertura de ambas as técnicas de solução e tipos de problemas e pelos recursos adicionais existentes. Fatores como o módulo GPU diferenciam-no de outros *frameworks* da literatura. Entretanto, em relação à possibilidade de hibridização, apesar de ser relatada pelos autores como parte do módulo `ParadisEO-PEO`, restringe-se a modelos pré-definidos, limitando, portanto, a definição de novas estruturas.

Melab et al. (2013) é a última publicação encontrada na literatura sobre a estrutura do *framework* `ParadisEO`. A versão mais recente da estrutura `ParadisEO` (versão 2.0.1) pode ser encontrada em `http://paradiseo.gforge.inria.fr/`, lançada em novembro de 2012.

2.2.2 *Frameworks* Multiagentes para Otimização usando Metaheurísticas

A demanda por *softwares* cada vez mais adaptativos e inteligentes levou à incorporação de novas tecnologias no tratamento de diferentes problemas. Neste sentido, o conceito de agentes tem sido amplamente utilizado no desenvolvimento de diversas aplicações para resolução de problemas, como é o caso da área de otimização. A abordagem baseada em agentes distingue-se de outras por seu poder em modelar problemas de natureza distribuída e expressar, através das entidades do sistema, a complexidade das relações envolvidas. Esta abordagem baseia-se no comportamento social de seres humanos ou outras entidades biológicas, para o projeto e desenvolvimento desses sistemas, com ênfase nas ações e interações sociais.

Há pouca concordância na literatura em relação ao termo *agente*, pois, para diferentes domínios, existem diferentes níveis de importância e relevância para cada uma das propriedades relacionadas aos agentes. Russell e Norvig (1995) definem *agente* como:

> *Uma entidade que pode perceber seu ambiente através de sensores (ou sistemas de percepção) e agir sobre ela através de seus mecanismos de ação (ou sistemas de atuação, atuadores). Para isso, há uma representação parcial desse ambiente e pode, em um universo multiagente, se comunicar com outros agentes. O comportamento geral do agente ou comunidade de agentes é consequência de suas percepções e conhecimentos, bem como das iterações realizadas.*

Por outro lado, Wooldridge (2009) apresenta a seguinte definição:

> *Um agente é um sistema de computador que está localizado em um ambiente e é capaz de ação autônoma nesse ambiente para atingir seus objetivos de projeto.*

No contexto de métodos aplicados para otimização, Milano e Roli (2004) apresenta a seguinte definição:

> *Um agente pode ser entendido como um sistema capaz de construir uma solução, mover-se em um espaço de busca, comunicar-se com outros agentes, ser ativo e possivelmente adaptável.*

Um Sistema Multiagente (MAS), como afirma Wooldridge (2009), é caracterizado por um conjunto de agentes autônomos em um ambiente. Ainda de acordo com esse mesmo autor, um Sistema Multiagente é composto de agentes, cada um com uma função específica e um conjunto de metas a serem atingidas. A soma das habilidades de cada um dos agentes, neste sistema, permite a solução do problema que está sendo tratado. A interação constante entre os próprios agentes e entre eles e o ambiente define o comportamento geral do sistema.

Agerbeck e Hansen (2008) identifica duas formas nas quais os Sistemas Multiagentes são frequentemente aplicados à solução de problemas de otimização:

(i) através da divisão das restrições e variáveis do problema entre os agentes; e

(ii) como uma ligação entre diferentes metaheurísticas para resolver problemas de otimização.

No entanto, essas duas formas não cobrem todas as diferentes possibilidades associadas à aplicação de Sistemas Multiagentes para resolver problemas de otimização. Por

exemplo, uma formulação importante que não é abordada por Agerbeck e Hansen (2008) é relacionada à hiperheurística, discutida anteriormente na Seção 2.1.3. Outra formulação importante não abordada por esta proposta é a arquitetura MAGMA, definida por Milano e Roli (2004) e tratada neste trabalho na Seção 2.2.2.8. Em ambos os casos citados, uma metaheurística não é definida como um agente único, mas vem da ação de diferentes agentes distribuídos em níveis subordinados.

A partir dessas questões, propomos uma classificação para Sistemas Multiagentes aplicados à otimização, a fim de incluir vários trabalhos relacionados. Três categorias de trabalhos são identificadas neste caso:

(i) Sistemas Multiagentes com decomposição do espaço de busca: através da divisão das restrições e variáveis do problema entre os agentes, cada um deles é responsável por otimizar sua perspectiva local e, no final, as soluções parciais são combinadas para uma solução global. Essa abordagem é usada em trabalhos como Liu e Sycara (1994), Jin e Liu (2002) e Zheng et al. (2012);

(ii) Sistemas Multiagentes com decomposição de metaheurísticas: os elementos que compõem as metaheurísticas são definidos como agentes. Esses elementos podem ser heurísticas de baixo nível (por exemplo, heurísticas construtivas e de melhora), estratégias de busca (por exemplo, perturbação da solução, estratégias de diversificação), soluções (por exemplo, indivíduos de uma população), população ou parte de populações e/ou partículas (por exemplos, formigas e pássaros). Nessa abordagem, metaheurísticas, híbridas ou não, podem ser vistas como o resultado da interação entre diferentes tipos de agentes. Aydin (2013) e Milano e Roli (2004) são exemplos de trabalhos que usam esses conceitos para resolver problemas de otimização;

(iii) Sistemas Multiagentes com Agentes Metaheurísticos: considera cada metaheurística como um agente autônomo e propõe explorar as vantagens do uso de Sistemas Multiagentes para refinar e combinar as soluções dessas metaheurísticas. Nesta abordagem, cada agente é responsável por realizar sua própria tarefa e, ao mesmo tempo, usar as soluções fornecidas por outros agentes para melhorar seu resultado. Esses agentes interagem e trabalham juntos para atingir um objetivo predefinido. Alguns exemplos de estudos que usam essa abordagem são Talukdar et al. (2003), Fernandes et al. (2009) e Silva et al. (2007, 2014, 2015).

De acordo com Aydin (2012) e Milano e Roli (2004), o uso de sistemas multiagentes na solução de problemas de otimização permite que diferentes regiões do espaço de busca sejam tratadas simultaneamente, proporcionando maior diversidade e melhores soluções. Esses trabalhos também afirmam que esse uso permite maior simplicidade na implementação, reduz o tempo computacional e a complexidade envolvida.

As propostas disponíveis na literatura que utilizam o conceito de sistemas multiagentes para metaheurísticas são diversas quanto à abordagem utilizada na organização e na forma de cooperação entre as metaheurísticas. Os trabalhos apresentados na Subseções 2.2.2.1 a 2.2.2.10 exemplificam e demonstram como esta organização pode ser diversa. A Seção 2.3 consolida uma comparação entre esses trabalhos.

2.2.2.1 *Agent Evolution* (`AgE`)

O *framework* `AgE` (*Agent Evolution*), desenvolvido como um projeto de código aberto, é uma plataforma de propósito geral para executar diferentes metaheurísticas paralelas, ori-

entadas para componentes baseados em agentes. *Evolutionary Multi-Agent System* (EMAS) (em português, Sistema Multiagente Evolucionário) é um algoritmo para a construção de sistemas multiagentes, que pode ser executado em AgE (e outras plataformas), foi inicialmente apresentado em Cetnarowicz et al. (1996) e posteriormente usado em diversas aplicações, incluindo a solução de problema de otimização combinatória. Segundo os autores,

> *EMAS é uma metaheurística híbrida que combina sistemas multiagentes com algoritmos evolutivos* (Krzywicki et al., 2015; Turek et al., 2016).

Consiste na formulação de sistemas multiagentes como processos evolutivos, nos quais a população de agentes pode evoluir dinamicamente. Dessa forma, além das formas típicas de interação em um MAS, os agentes podem se reproduzir (gerar novos agentes), morrer (ser eliminados do sistema) ou migrar (em um caso de múltiplas populações).

O desenvolvimento desta proposta ocorreu particularmente em trabalhos realizados por pesquisadores do Grupo de Sistemas Inteligentes de Informação do Departamento de Ciência da Computação da AGH Universidade de Ciência e Tecnologia, em Kraków, na Polônia, sob o projeto Paraphrase AGH (http://www.paraphrase.agh.edu.pl/). Este projeto tem como foco principal o desenvolvimento de ferramentas que empregam o conceito de EMAS, além de um *framework* (AgE) que suporta competição e distribuição. Revisões referentes a esta proposta e suas implicações estão em Byrski et al. (2015) e Byrski e Kisiel-Dorohinicki (2017).

Em seu ciclo de desenvolvimento, o AgE foi apresentado em várias implementações. Implementações dedicadas do *framework* AgE são construídas usando Java (http://age.agh.edu.pl e https://gitlab.com/age-agh/age3), Python (https://github.com/maciek123/pyage), Scala (https://github.com/eleaar/scala-mas), bem como Erlang (https://github.com/ParaPhraseAGH/erlang-emas), associadas a diferentes fases do projeto, ainda em andamento. O ciclo atual mostra o uso de linguagens funcionais, como Erlang (http://www.erlang.org/) e Scala (http://www.scala-lang.org/). Os autores defendem a necessidade de mudar paradigmas de desenvolvimento de plataformas de computação, a fim de utilizar a disponibilidade de novas possibilidades de hardware em termos de computadores *multi* e *many-core*. De acordo com Turek et al. (2016),

> *O desenvolvimento eficiente de* software *para arquiteturas de muitos núcleos exigirá linguagens funcionais de alto nível com variáveis imutáveis, sem compartilhar o estado e passagem de mensagens simutaneamente.*

Em Turek et al. (2016) a implementação dedicada de Erlang AgE é apresentada, enquanto Krzywicki et al. (2015) mostra uma comparação entre as implementações AgE com Erlang e Scala.

A descrição na sequência diz respeito à implementação dedicada da Erlang AgE. A estrutura AgE é dividida em 4 elementos: aplicação MAS, ambiente de execução, corretor de migração e coleta de estatísticas. A aplicação multiagente é direcionada para as interações, definindo: tipos de agentes, estados e ações possíveis, mapeamento de estado/ação e funções para atualização de subpopulações. O ambiente de execução implementa a lógica computacional dessas funções e vincula os resultados às simulações. O módulo do agente de migração é responsável pela migração de agentes entre ambientes. O último módulo é responsável por coletar as estatísticas e métricas.

Um agente, na implementação do EMAS, é definido por um vetor de valores reais representando uma solução do problema a ser solucionado. Cada agente recebe de seus pais

uma quantidade inicial de energia. A energia é aqui um recurso não renovável e permitirá a seleção de agentes. Agentes que perdem toda a energia são removidos do sistema. Embora o número de agentes possa variar, a energia do sistema permanece constante. A população de soluções é dividida em subpopulações, buscando preservar a diversidade. Cada subpopulação é chamada de ilha. A interação entre os agentes nesse contexto é realizada por meio de arenas de encontro. Cada agente entra em uma arena de acordo com sua quantidade de energia. Desse modo, a dinâmica do sistema é definida da seguinte maneira: os agentes entram nas arenas de acordo com a energia e o comportamento de cada um; em sequência, a operação de reunião é aplicada em cada arena. De acordo com Krzywicki et al. (2015), o padrão usado é *muito flexível, pois pode ser implementado de forma centralizada e síncrona ou descentralizada e assíncrona*.

Quatro diferentes formas de interação entre os agentes são implementadas para definir o melhor modelo para alcançar um paralelismo eficiente no sistema multiagente:

(i) Sequencial: nesta versão, todas as populações estão envolvidas em um loop recursivo dentro de um único processo `Erlang`. As funções são aplicadas repetidamente para atualizar a população de agentes e, a cada passo, os agentes podem migrar entre populações com baixa probabilidade. O processo `Erlang` passa por uma coleção de ilhas.

(ii) Híbrido: nesta versão, cada população de agentes está contida em um processo `Erlang` separado e é executada em um loop separado. A migração do agente ocorre por meio da passagem de mensagens. Um único processo separado atua como um intermediário de migração. A probabilidade de migração é muito baixa.

(iii) Esqueletos: nesta versão, a implementação baseada em `SKEL` é o resultado da introdução dos esqueletos da biblioteca `SKEL` na versão sequencial.

(iv) Concorrente: nesta versão, cada agente é executado em um processo `Erlang` autônomo e interage apenas pelas arenas mediadoras. Neste modelo estão presentes as arenas responsáveis por cada tipo de interação: luta, reprodução, morte e migração. Cada população de agentes tem um conjunto separado de arenas.

De acordo com os endereços de URL acima, a última atualização para o Java `AgE3` foi lançada em junho de 2017; para o `Python AgE`, a última atualização foi lançada em maio de 2016; para a `Erlang AgE`, a última atualização foi lançada em abril de 2015; e para o `Scala AgE`, a última atualização foi lançada em agosto de 2017. Uma nova etapa no desenvolvimento deste projeto apresenta a proposta do paradigma baseado no agente Memético, que, de acordo com Żurek et al. (2017), *combina técnicas de computação evolucionária e busca local*.

2.2.2.2 *Agent Metaheuristic Framework* (`AMF`)

Meignan et al. (2008a,b, 2009, 2010) propõem o *Agent Metaheuristic Framework* (`AMF`) (em português, *Framework* de Agentes Metaheurísticas). O objetivo principal é usar a abordagem orientada a agente para apoiar o projeto e a hibridização de metaheurísticas. O `AMF` é baseado em um modelo organizacional que descreve uma metaheurística em termos de papéis. O modelo organizacional usado como referência para a proposta de `AMF` é baseado no metamodelo `RIO`. O metamodelo `RIO` envolve três conceitos básicos:

(i) Função (*Role*): abstração de um comportamento ou status em uma organização;

(ii) Interação (*Interaction*): *links* entre dois papéis, de modo que uma ação no primeiro papel produza uma reação no segundo;

(iii) Organização (*Organization*): conjunto de papéis e suas interações.

Os papéis representam os principais elementos que compõem as metaheurísticas, como intensificação, diversificação, memória e adaptação ou auto-adaptação. Segundo os autores, *para obter uma metaheurística a partir do modelo organizacional do* `AMF`*, é necessário refinar os diferentes papéis e determinar a estrutura multiagente do sistema de otimização.*

Para ilustrar o uso do modelo `AMF`, uma metaheurística denominada *Metaheuristic Coalition-Based* (`CBM`) e sua aplicação ao Problema de Roteamento de Veículos são apresentadas. A `CBM` é uma metaheurística baseada na metáfora de uma coalizão. Segundo os autores, *a coalizão é composta de vários agentes que têm a capacidade de tratar individualmente o problema de otimização, mas cooperam para coordenar e melhorar a busca.*

O sistema `CBM` é composto por vários agentes idênticos. O agente `CBM` manipula uma única solução e usa vários operadores para se mover no espaço de busca. Estes operadores estão relacionados com as tarefas de intensificação e diversificação.

Uma estratégia adaptativa é utilizada pelo agente `CBM`, através de um mecanismo de aprendizagem, para selecionar o operador mais adequado, baseado no contexto. A cooperação entre esses agentes ocorre de duas maneiras: (i) um agente compartilha sua solução mais conhecida; (ii) um agente compartilha suas regras de decisão interna, para permitir o mimetismo de comportamento. Consequentemente, a hibridização das metaheurísticas vem dessa cooperação entre os agentes.

A última publicação encontrada na literatura sobre o *framework* `AMF` é Meignan et al. (2010). As versões deste *framework*, apresentadas na literatura, não estão disponíveis para *download*.

2.2.2.3 *Asynchronous Teams* (`A-Teams`)

Em Talukdar e de Souza (1990) e Talukdar et al. (1998, 2003), o modelo conceitual de uma arquitetura multiagente para a solução de problemas, chamada *Asynchronous Teams* (`A-Teams`), é introduzido. Nessa arquitetura, um conjunto de agentes autônomos e memórias interconectadas formam uma rede de fluxo de dados cíclica, na qual as soluções encontradas circulam continuamente através dela.

Dois tipos de agentes modificam as soluções armazenadas nas memórias:

(i) Construtores: adicionam novas soluções às memórias;

(ii) Destrutores: eliminam soluções das memórias;

Nesta arquitetura, agentes independentes trabalham em paralelo e cooperam, modificando, cada um, as soluções dos outros. Cada agente implementa um algoritmo de solução de problemas e cada memória é dedicada a um problema.

A arquitetura `A-Teams` tem sido usada como base para vários trabalhos, como em Lukin et al. (1997), Rabak e Sichman (2003), Landa-Silva e Burke (2007), Carle et al. (2012), Barbucha et al. (2006), Barbucha et al. (2010) e Barbucha (2014), entre outros. As versões executáveis dessa arquitetura, em sua versão original, foram implementadas apenas para estudos de caso específicos, como mostrado em Talukdar et al. (2003), que apresenta várias aplicações para Problemas de Programação de Produção. Além disso, essa estrutura foi usada como base para estabelecer novas metaheurísticas baseadas em agentes, como o

caso dos artigos citados acima. Barbucha et al. (2010) apresentou uma revisão crítica da aplicação da arquitetura A-Teams ao longo dos anos.

Barbucha et al. (2006) e Barbucha et al. (2010) introduziram uma ferramenta para a construção de arquiteturas A-Teams para resolver diferentes problemas de otimização, denominada JABAT (*JADE-based A-Team environment*). Essa estrutura produz soluções para problemas de otimização combinatória, usando um conjunto de agentes de otimização, cada um implementando um algoritmo de solução. O JABAT é baseado em JADE (Bellifemine et al., 2007), um dos mais importantes *middlewares* para a construção de sistemas multiagentes usando Java, fornecendo ferramentas fundamentais de infra-estrutura de comunicação e visualização.

A estrutura do JABAT mostra um bom conjunto de recursos adicionais para melhorar o desempenho do processo de busca, como a execução paralela e distribuída; extensão acessível via *web*; e uso de aprendizado por reforço. Na execução distribuída, o processo de otimização pode ser executado em muitos computadores. O usuário pode adicionar ou excluir computadores envolvidos no processo de otimização. Nesses casos, o JABAT se adapta às mudanças, comandando os agentes de otimização. Além disso, a hibridização de metaheurísticas é possível no JABAT, a partir da interação entre os agentes definidos pelo modelo conceitual de A-Teams. No entanto, essa interação é regulada pelo *middleware* JADE.

Uma extensão do *framework* JABAT, chamada eJABAT, é proposta em Barbucha et al. (2009). A extensão eJABAT foi projetada para ser totalmente acessível via internet, permitindo, através de uma interface simples, o acesso à solução de problemas e a adição de novos computadores ao sistema. A última extensão deste *framework*, denominada *Cooperative JADE-based A-Team* (Cooperative JABAT), é proposta em Barbucha (2010), introduzindo um mecanismo de Aprendizado de Reforço. Nesse caso, a recompensa é atribuída aos agentes de otimização sempre que eles encontrarem uma solução melhor do que a encontrada anteriormente. Uma recompensa negativa também pode ser concedida se o agente não conseguir melhorar uma solução. O peso de cada agente nesse contexto é levado em conta quando as soluções são solicitadas na memória comum. Dessa forma, as soluções solicitadas não são enviadas imediatamente após a solicitação de cada agente, mas somente após todos os agentes fazerem suas solicitações e levando em consideração os pesos.

É importante observar que nenhuma das versões do JABAT descritas aqui está disponível para download.

2.2.2.4 *Collaboration of Metaheuristic Algorithms* (CMA)

Malek (2009) também usou a abordagem baseada em agentes em um *framework*, chamado *Collaboration of Metaheuristic Algorithms* (CMA) (em português, Colaboração de Algoritmos Metaheurísticos) para a execução de várias metaheurísticas, simultaneamente, a fim de resolver problemas de otimização combinatória. Cada metaheurística é implementada como uma caixa preta, com entradas e saídas, em que a entrada (soluções) é convertida em uma representação específica do algoritmo implementado.

Esta proposta usa um *framework* para sistemas multiagentes, chamado A-Globe (Sislak et al., 2005) como base para seu desenvolvimento. A-Globe oferece recursos multiagentes como esqueletos de agentes, mensagens, protocolos de *chat*, etc. De acordo com o conceito apresentado pelo autor, *o sistema pode ser imediatamente decomposto em blocos tais como metaheurísticas particulares, pool de soluções (global) e outros, cada bloco é manipulado pelo seu agente*. Este conceito colaborativo usado nesta estrutura é semelhante

à arquitetura MAGMA, apresentada em Subseção 2.2.2.8. O resultado da busca para uma metaheurística particular também é compartilhado pelas populações globais de soluções candidatas armazenadas em um *pool* de soluções.

Quatro tipos de agentes compõem o sistema deste *framework*:

(i) Agente Problema: responsável por (a) receber uma solicitação para iniciar a otimização, enviada por um usuário ou agente; (b) ler os dados do problema a partir de um arquivo de configuração; (c) inicializar os outros agentes; (d) medir o tempo total de otimização; e (e) comunicar aos agentes o final da busca;

(ii) Agente *Pool* de Soluções: responsável por gerenciar a população de soluções candidatas e fornecer soluções para outros agentes;

(iii) Agente Algoritmo: responsável por implementar e executar uma metaheurística específica;

(iv) Agente Consultor: responsável por fornecer parâmetros de configuração para metaheurísticas e receber relatórios após sua execução. Cada classe de Agente Algoritmo possui um Agente de Consultor associado a ele.

Em seu artigo mais recente, Malek (2010) usa os conceitos desse *framework* na definição de uma estrutura para hiperheurísticas. Nesta proposta, um recurso responsável por definir os parâmetros é adicionado por meio de um objeto chamado Política. O objetivo é fornecer novos parâmetros dinâmicos, a partir do *feedback* do próprio algoritmo. Assim, os relatórios enviados ao Agente Consultor são avaliados e usados para determinar os novos parâmetros de entrada do algoritmo. A possibilidade de execução paralela e distribuída dos algoritmos também está incluída nesta proposta.

A última publicação encontrada sobre o CMA é Malek (2010). A disponibilização de uma versão para *download* também não foi encontrada.

2.2.2.5 *Distributed Agent Framework for Optimization* (DAFO)

Danoy et al. (2005, 2010) propõem um sistema multiagente, direcionado para otimização evolucionária, chamado DAFO (*Distributed Agent Framework for Optimization*). Segundo os autores, *um elemento chave deste framework reside em seu modelo de organização multiagente, MAS4EVO (Multi-Agent System for EVolutionary Optimization, ou, em português, Sistema MultiAgente para Otimização Evolucinária), que é uma extensão do modelo Moise+* (Hubner et al., 2007). O principal objetivo do modelo MAS4EVO é superar as limitações em estruturas baseadas em agentes, definindo uma estrutura e interações baseadas em algoritmos genéticos coevolucionários (AGCs). Diferentemente dos algoritmos genéticos (AGs), nos quais uma população de indivíduos semelhantes evolui para representar uma solução global, nas subpopulações de indivíduos da AGC coevolvem, representando partes específicas da solução global que evoluem independentemente.

O ambiente multiagente DAFO é definido e representa o problema de otimização a ser resolvido. Três tipos de agentes compõem a estrutura:

(i) Agentes de Resolução de Problemas (*Problem Solving Agents* - PSA): responsáveis por buscar a solução para o problema usando uma metaheurística;

(ii) Agente Fábrica (*Fabric Agents* - FA): responsável por instanciar e configurar o aplicativo executado;

(iii) Agentes de Observação (*Observation Agents* - OA): responsáveis por observar os Agentes de Resolução de Problemas e fornecer resultados aos usuários.

A interação entre os agentes é definida por protocolos de comunicação (proposições `FIPA ACL` e `FIPA-SL`). Esses protocolos estruturam e expressam a sequência de mensagens de acordo com a estratégia de troca de soluções usada (melhor, aleatória, etc.) entre os AGs implementados pelos agentes. O modelo de organização `MAS4EVO` especifica os padrões globais de cooperação no comportamento do agente, definindo uma organização multiagente baseada em estratégias coevolucionárias. Essa forma de organização permite que o agente altere e reorganize o funcionamento geral do sistema.

Essa interação entre os agentes permite a hibridização de metaheurísticas. No entanto, os outros recursos adicionais de interesse nesta análise que permitem melhor desempenho de busca, como execução paralela e distribuída, não são cobertos pelo *framework* `DAFO`.

A versão mais recente desta estrutura mostrada na literatura é descrita em Danoy et al. (2010) e não está disponível para *download*.

2.2.2.6 *Learning-Based Multi-agent System* (`LBMAS`)

Em Lotfi e Acan (2015, 2016), é apresentado o `LBMAS` (*Learning-Based Multi-Agent System*, em português, Sistema MultiAgente Baseado em Aprendizagem) para resolver problemas de otimização combinatória. Esse sistema permite a colaboração entre agentes metaheurística em uma população de solução comum e um arquivo de dois estágios, também comum aos agentes. Desta forma, diferentes regiões do espaço de busca são exploradas usando o agente mais efetivo no momento.

A arquitetura `LBMAS` inclui sete Agentes Metaheurísticos e quatro Agentes do Sistema (Agente Problema, Agente *Pool* de Soluções, Agente Gerente e Agente de Arquivos), descritos abaixo:

- Agentes metaheurísticos: cada agente metaheurístico atua com sua própria estratégia de busca e fornece as soluções encontradas. As sete metaheurísticas implementadas pelos agentes metaheurísticos são Algoritmos Genéticos (AGs), Evolução Diferencial (DE), *Simulated Annealing* (SA), Otimização de Colônia de Formigas (ACO), *Great Deluge Algorithm* (GDA), Busca Tabu (TS) e a *Cross Entropy Method* (CE);
- Agente Problema: responsável por inicializar os parâmetros de entrada do problema;
- Agente *Pool* de Soluções: responsável por manipular todas as transações com a população de soluções comum;
- Agente de Arquivo: responsável pelas operações de recuperação e manipulação do arquivo comum de dois estágios;
- Agente Gerente: responsável por repassar as soluções para os agentes *Pool* de Soluções e Arquivo, que, por sua vez, atualizam as populações de soluções compartilhadas.

Em cada iteração da busca, a metaheurística a ser usada na próxima iteração é escolhida. Esta seleção é realizada utilizando os conceitos da roleta russa, em que os valores de avaliação das metaheurísticas são obtidos com base nos níveis de melhoria da função objetivo de cada uma das metaheurísticas. Todas as metaheurísticas têm inicialmente a mesma probabilidade de serem escolhidas e essa probabilidade muda (cresce ou diminui), de acordo com o desempenho individual dos agentes.

No LBMAS, os agentes cooperam compartilhando suas experiências individuais através de um arquivo de memória externa de dois estágios: o primeiro estágio armazena soluções promissoras com base em seu valor de avaliação; e o segundo estágio mantém as soluções de acordo com sua distribuição espacial, com base em uma medida de dissimilaridade definida. No entanto, embora os agentes compartilhem suas experiências, em cada iteração apenas uma metaheurística é executada e, portanto, a hibridização pode ocorrer apenas sequencialmente, com a informação sendo passada de uma iteração para a seguinte.

A última publicação sobre o *framework* LBMAS é Lotfi e Acan (2016) e uma versão para *download* não estava disponível.

2.2.2.7 *Multi-agent Cooperative Search* (MACS)

Martin et al. (2016) introduz um *framework* distribuído baseado em agentes chamado *Multi-agent Cooperative Search* (MACS), ou, em português, Busca Cooperativa Multiagente. Essa estrutura também é implementada usando a plataforma JADE (Bellifemine et al., 2007). Nesta proposta, cada agente executa uma combinação diferente de metaheurísticas/heurísticas de busca local e se adapta continuamente ao longo do processo de busca usando um protocolo de cooperação. Este protocolo de cooperação permite a comunicação entre os agentes envolvidos na busca da solução.

O MACS possui dois tipos de agentes:

(i) Agente lançador: responsável por preparar o problema a ser resolvido pelos demais agentes, convertendo o problema para protocolo proposto de mensagens entre agentes;

(ii) Agente metaheurístico: executa uma combinação de metaheurísticas/heurísticas de busca local disponíveis, de modo que cada agente pode usar um conjunto diferente de parâmetros de configuração.

O Agente lançador define a combinação de metaheurísticas/heurísticas de busca local e os parâmetros de configuração usados pelos agentes metaheurísticos.

Os principais elementos da estrutura básica do MACS estabelecem a base para o protocolo de comunicação do *framework* e, de acordo com os autores, *definem um conjunto de primitivas representacionais gerais que são usadas para modelar um número de problemas de programação e roteamento*. Esses elementos são:

- *SolutionElements*: objetos abstratos que representam elementos específicos do problema, como um cliente no Problema de Roteamento de Veículos;
- *Edge*: contém dois objetos *SolutionElements*. Esses pares são utilizados na permutação realizada no protocolo de cooperação para identificar bons padrões;
- *Constraints*: interface entre o *framework* e as restrições específicas do problema;
- *NodeList*: lista de objetos *SolutionElements* ou de *Edges*;
- *SolutionData*: lista de objetos *NodeList*.

Uma iteração do protocolo de comunicação é aqui chamada de conversa. Sendo assim, a comunicação entre os agentes se dá da seguinte forma: identificam-se bons padrões que compõem soluções de melhora, de acordo com a frequência em que ocorrem na conversa, e são, então, compartilhados entre os agentes. Para isso, cada Agente metaheurístico

quebra a solução atual em objetos *edge* e os envia para o Agente lançador, que reúne todos os objetos de borda e pontua-os de acordo com sua frequência. Esses elementos são compartilhados com os outros agentes e farão parte das próximas soluções. Portanto, a hibridização de metaheurísticas pode ocorrer na estrutura do `MACS` de uma maneira diferente do que em outras estruturas multiagentes, uma vez que nesta proposta apenas partes da solução são compartilhadas, e não a solução como um todo.

O conceito de aprendizagem também é usado na construção do *framework* `MACS`. O mecanismo de aprendizado implementado permite que cada agente mantenha uma memória de curto prazo de bons *edges*, que são usadas no início de cada conversa para influenciar como as novas soluções são construídas. Os *edges* identificados pelo aprendizado são usados para reordenar as listas de elementos a serem inseridos nas soluções.

A estrutura do `MACS` está disponível como um projeto de código aberto em `http://simonpmartin.github.io/macs/`, e sua última atualização foi lançada em abril de 2016.

2.2.2.8 `MAGMA`

A arquitetura `MAGMA`, apresentada por Milano e Roli (2004), usa a perspectiva multiagente na definição de uma estrutura na qual uma metaheurística pode ser vista como o resultado da interação entre os vários agentes da própria estrutura. Os agentes dessa arquitetura trabalham em níveis subordinados, cada nível correspondendo a uma ação da metaheurística:

(i) Nível 0: os Agentes Construtores de Solução implementam heurística construtiva;

(ii) Nível 1: os Agentes de Melhora de Solução implementam heurística de busca local;

(iii) Nível 2: os Agentes de Estratégia implementam as estratégias usadas para escapar do ótimo local;

(iv) Nível 3: os Agentes de Coordenação coordenam o processo de busca e os agentes dos níveis inferiores.

A comunicação entre dois níveis, feita através da interação entre os agentes, é dividida em dois tipos:

(i) horizontal: envolve a interação entre agentes de mesmo nível;

(ii) vertical: envolve a interação entre agentes de diferentes níveis.

Um exemplo de uma aplicação da arquitetura `MAGMA`, na qual há cooperação entre o algoritmo memético e a metaheurística GRASP (*Greedy Randomized Adaptive Search Procedures*), é apresentado em Milano e Roli (2004). O objetivo deste exemplo é mostrar como a busca cooperativa é alcançada pela adição de diferentes perspectivas e pela troca de informações. A metaheurística GRASP é definida em três níveis: o primeiro gera soluções iniciais usando o método Guloso Randômico (*Greedy Random*) de construção; o segundo executa um algoritmo de busca local; o terceiro armazena a melhor solução encontrada. A metaheurística Algoritmo Memético também é definida em três níveis: no nível 0, vários agentes geram soluções para a população inicial; no nível 1, os agentes melhoram as soluções iniciais usando a busca local; e no nível 2, os operadores de recombinação e mutação são aplicados, gerando novas soluções. A cooperação se dá através da troca de informações de estados, soluções e outras características do espaço de busca.

É importante notar que, na proposta da arquitetura MAGMA, uma metaheurística não é um único agente operando independentemente, mas surge da ação cooperativa de diferentes agentes atuando em vários níveis de abstração da arquitetura. A arquitetura MAGMA não foi disponibilizada na versão executável, permanecendo, até o momento, na versão conceitual mostrada em Milano e Roli (2004).

2.2.2.9 *MultiAgent eNvironment for Global Optimization* (MANGO)

Um ambiente para otimização que utiliza o conceito de sistemas multiagentes, chamado MANGO (*MultiAgent eNvironment for Global Optimization*), é apresentado por Kerçelli et al. (2008) e Aydemir et al. (2012). O sistema MANGO é um projeto em desenvolvimento e fornece um ambiente no qual cada agente executa um algoritmo de busca.

A cooperação entre agentes é um dos principais elementos do *framework*. Os agentes se comunicam e cooperam entre si por meio de uma linguagem de comunicação. O sistema fornece protocolos de cooperação e modelos organizacionais para realizar a comunicação, bem como a possibilidade de registro de novos agentes.

Nas versões iniciais deste *framework*, apresentadas em Kerçelli et al. (2008) e Günay et al. (2009), a plataforma JADE (uma estrutura para desenvolvimento e implementação de aplicações multiagentes, introduzida em Bellifemine et al. (2001)) foi usada para a implementação do sistema multiagente MANGO. Em sua versão analisada, como mostrado em Aydemir et al. (2012), um ambiente multiagente específico, também baseado no JADE, foi desenvolvido.

No ambiente MANGO, os agentes são definidos a partir da API MANGO, que usa a linguagem Java em sua implementação. Esses agentes podem ser executados de maneira distribuída, em uma plataforma de rede distribuída ou em um único computador. A cooperação é realizada com base no conceito de Arquitetura Orientada a Serviços (SOA). A comunicação é implementada em dois níveis: (i) um nível, feito sob o *Java Messaging Service* (JMS), é responsável pelas estruturas e protocolos de redes; (ii) o outro nível, construído sobre o primeiro, é responsável pela troca de mensagens entre os agentes.

A troca de mensagens entre os agentes do *framework* pode ocorrer de duas maneiras: interrupções e caixas de correio (*mailboxes*). Na troca de mensagens com base em interrupções, quando uma mensagem é enviada a um agente, isso interrompe imediatamente o processo de busca. Na troca de mensagens baseadas em caixas de correio, a mensagem é armazenada e o agente verificará sua caixa de correio quando for mais apropriado. Além disso, o agente de diretório (DA) é definido, que é o principal responsável pelo gerenciamento dos recursos de comunicação.

A comunicação entre os agentes, conforme descrito acima, permite a hibridização de metaheurísticas na estrutura da MANGO. A versão mais recente do ambiente MANGO foi lançada em outubro de 2013 e está disponível em `http://algopt.sabanciuniv.edu/mango/`.

2.2.2.10 *Swarm of Metaheuristic Agents* (SMA)

Algoritmos evolutivos também são amplamente utilizados na definição de aplicações envolvendo o conceito de agentes. Este é um senso comum de uso do conceito de agentes, associando-os diretamente com suas próprias estruturas de algoritmos evolutivos, como genes, memes, partículas ou formigas, ou outras estruturas bioinspiradas.

Nessa linha, três grupos de trabalho podem ser identificados:

(i) o primeiro considera que os elementos do algoritmo evolutivo (por exemplo, indivíduos da população ou soluções) são definidos como agentes;

(ii) a segunda abordagem considera que os elementos do algoritmo evolutivo (indivíduos da população), não-agentes, atuam de forma cooperativa na busca pela solução do problema;

(iii) o terceiro grupo de propostas usa algoritmos evolutivos para gerenciar um grupo de agentes metaheurísticos, de forma que cada agente implemente um método para encontrar soluções. Este último grupo de trabalho relaciona-se diretamente com formulações hiperheurísticas apresentadas na seção 2.1.3.

Aydin (2010, 2012, 2013) apresentam uma proposta de *framework* envolvendo agentes que utilizam inteligência de enxames para coordenação de agentes de busca. Nesta proposta, cada metaheurística é definida como um agente, mantém sua própria inteligência e a emprega na solução do problema. O algoritmo *Particle Swarm Optimization* (PSO) é usado para coordenar os agentes e dar suporte à diversificação da busca.

Para exemplificar, como mostrado no experimento mais recente descrito em Aydin (2013), uma população de agentes é definida, de modo que cada agente possui habilidades de busca, com base na metaheurística *Simulated Annealing* (SA), bem como capacidades de comunicação. Esses agentes são organizados por meio de um enxame que coopera para resolver o problema. Em cada geração do enxame de agentes *Simulated Annealing*, os agentes executam seus algoritmos de busca. Um algoritmo de cooperação é aplicado, após cada geração, trocando informações relacionadas às soluções obtidas.

Essa forma de organização multiagente restringe as possibilidades de interação entre agentes para o formato do algoritmo populacional utilizado. Portanto, a hibridização de metaheurísticas também é limitada a essas interações. O código do *framework* não está disponível para *download*, e novas versões da proposta não foram encontradas na literatura.

2.3 Comparativo dos Trabalhos Correlatos

Nesta seção, uma análise comparativa dos *frameworks* de otimização usando metaheurísticas, descritos na Seção 2.2, é realizada.

A metodologia adotada na elaboração deste comparativo consistiu em cinco etapas:

(i) coleta de dados;

(ii) definição de indicadores de análise;

(iii) classificação dos *frameworks* de acordo com esses indicadores;

(iv) seleção dos *frameworks* a serem analisados no artigo, a partir da classificação feita;

(v) análise comparativa.

Na etapa de coleta de dados, a busca por artigos foi realizada por meio da ferramenta *Google Scholar*. O *Google Scholar* foi usado por ser um mecanismo de busca eletrônico gratuito que permite alcançar diretamente todos os principais bancos de dados acadêmicos de periódicos e eventos científicos. Além disso, as informações de citações fornecidas permitiram determinar o impacto dos artigos na comunidade científica. Nesta etapa, uma extensa pesquisa foi realizada usando palavras-chave específicas:

- *metaheuristics optimization framework*;
- *multi-agent optimization framework*;
- *MAGMA framework*;
- *cooperation search*.

Essas quatro palavras-chave da pesquisa geraram novos termos de pesquisa, geradas pelos resultados da pesquisa inicial. As novas palavras utilizadas na busca estavam diretamente relacionadas aos *frameworks* mais consolidados, sendo eles: `ParaDisEO`, `MALLBA`, `EasyLocal++` e `JMetal`. Adicionalmente, todos os artigos citados em Parejo et al. (2012) foram analisados, seguindo uma busca do tipo bola de neve, a fim de encontrar trabalhos potencialmente relacionados. O conjunto de palavras de pesquisa adotadas e o procedimento de pesquisa em bola de neve possibilitaram uma revisão não tendenciosa, que agregou de estruturas consolidadas a trabalhos recentemente publicados, como o `MACS`, ou trabalhos pouco conhecidos, como `JABAT` e `OptFrame`. Esta etapa se encerrou com a identificação dos *frameworks* candidatos a serem analisados nesta tese.

A definição de indicadores que permitam (i) a diferenciação entre as características dos *frameworks*; (ii) uma classificação dos *frameworks*; e (iii) a seleção dos *frameworks* que são de real interesse foi o segundo passo desta análise. Os indicadores gerais adotados para definir os *frameworks* a serem analisados são:

(i) *Framework* para solução de problemas de otimização combinatória: os *frameworks* selecionados são exclusivos para resolver problemas de otimização combinatória mono e/ou multiobjetivo;

(ii) *Framework* de propósito geral: os *frameworks* devem apresentar uma estrutura de escopo geral, permitindo a solução de diferentes problemas de otimização combinatória, bem como a facilidade de inclusão de novos métodos. Assim, *frameworks* direcionados a classes específicas de problemas ou a uma metaheurística específica foram excluídos deste estudo;

(iii) *Framework* que suporta técnicas de otimização de trajetória e/ou população: *Frameworks* que implementam metaheurísticas baseadas em trajetória e/ou populacionais;

(iv) *Frameworks* que utilizam o conceito de agentes e que atendem aos três primeiros indicadores, ou seja, resolvem problemas de otimização combinatória, são de uso geral e suportam técnicas de otimização baseadas em trajetória e/ou de base populacional;

(v) *Frameworks* que tenham pelo menos uma publicação em revistas científicas e/ou conferências. Não serão considerados aqueles que foram publicados apenas na forma de relatórios técnicos, pois não possuem formas de avaliação do trabalho por pares.

O terceiro passo é classificar os *frameworks* de acordo com os indicadores. Esta etapa exige, para sua eficácia e eficiência, o conhecimento dos *frameworks* e a apresentação de informações mínimas sobre eles. O quarto passo é a seleção dos *frameworks* a serem avaliados, descartando aqueles que não estão aderentes aos objetivos de comparação propostos. Com base nesses critérios, os seguintes trabalhos são avaliados, já apresentados na Seção 2.2: `AgE` (Krzywicki et al., 2015; Turek et al., 2016; Byrski e Kisiel-Dorohinicki, 2017),

AMAM (Fernandes et al., 2009; Silva et al., 2014, 2015), AMF (Meignan et al., 2008a,b, 2009, 2010), *Collaboration of Metaheuristic Algorithms* (Malek, 2009, 2010), DAFO (Danoy et al., 2005, 2010), EasyLocal++ (Di Gaspero e Schaerf, 2001, 2002, 2003), ECJ (White, 2012), EvA2 (Kronfeld et al., 2010), FOM (Parejo et al., 2003), HeuristicLab (Wagner e Affenzeller, 2004, 2005; Wagner et al., 2010, 2014), HotFrame (Fink et al., 1999; Fink e Voß, 2002), Hyflex (Ochoa et al., 2012), JABAT (Barbucha et al., 2006, 2010), JAMES (De Beukelaer et al., 2015, 2017), JCLEC (Ventura et al., 2008; Cano et al., 2015), JMetal (Durillo et al., 2010; Durillo e Nebro, 2011), LBMAS (Lotfi e Acan, 2015, 2016), MACS (Martin et al., 2016), MAGMA (Milano e Roli, 2004), MALLBA (Alba et al., 2002, 2006, 2007), MANGO (Kerçelli et al., 2008; Günay et al., 2009; Aydemir et al., 2012), OptFrame (Coelho et al., 2011, 2016a), Opt4j (Lukasiewycz et al., 2011), ParadisEO (Cahon et al., 2004; Liefooghe et al., 2011; Melab et al., 2013; Humeau et al., 2013) e *Swarms of Metaheuristic Agents* (Aydin, 2010, 2012, 2013).

Finalmente, o último passo consiste em realizar a análise comparativa.

É importante deixar claro que esta análise não possui como objetivo o estabelecimento de um *ranking* entre os *frameworks* analisados, como realizado, por exemplo, em Parejo et al. (2012). Os *rankings* frequentemente carregam um alto grau de subjetividade na definição de métricas quantitativas de análise, o que pode obscurecer, e não esclarecer, questões vitais tanto no conhecimento do atual estado da arte do tema, quanto em apontar caminhos futuros para o desenvolvimento da pesquisa. A análise comparativa apresentada aqui é, portanto, estritamente qualitativa, o que não a isenta de erros, é claro, mas fornece elementos para que o leitor diferencie os *frameworks* analisados e esteja claro sobre as limitações e avanços atuais na construção de *frameworks* para resolver problemas de otimização.

A Tabela 2.1 apresenta, em um resumo, a lista dos *frameworks* analisados, bem como informações relevantes sobre as publicações, versões e sítios de internet onde estes podem ser encontrados. Nesta tabela, o símbolo "–" indica que o *framework* associado não possui um sítio na Internet para sua publicação.

O objetivo desta seção é identificar quais das características desejáveis a um *framework* para otimização estão disponíveis nas propostas relatadas nas seções anteriores. Em particular, procura-se discernir quais possibilitam, aos seus usuários, o desenvolvimento de metaheurísticas híbridas, enfocando abordagens cooperativas e paralelas.

Tabela 2.1: Resumo dos *Frameworks* Analisados

	Framework	**Tipo**	**Principais Referências**	**Última versão/ publicação e data**	**Site**
1	**AgE**	*Framework* Multi-agente para Otimização Evolucionária	Turek et al. (2016)	v.23, Outubro de 2015	`https://github.com/ParaPhraseAGH/labs-emas`
2	**AMAM**	*Framework* Multi-agente para Otimização	Fernandes et al. (2009), Silva et al. (2014, 2015)	Silva et al. (2015), Novembro de 2015	–
3	**AMF**	*Framework* Multi-agente para Otimização Evolucionária	Meignan et al. (2008a,b), Meignan et al. (2009, 2010)	Meignan et al. (2010), 2010	`http://www.lalea.fr/?page_id=45`
4	**CMA**	*Framework* Multi-agente para Otimização	Malek (2009, 2010)	Malek (2010), 2010	–
5	**DAFO**	*Framework* Multi-agente para Otimização Evolucionária	Danoy et al. (2005, 2010)	Danoy et al. (2010), 2010	–
6	**EasyLocal++**	*Framework* para Busca Local	Di Gaspero e Schaerf (2001), Di Gaspero e Schaerf (2002), Di Gaspero e Schaerf (2003)	v3.0, Fevereiro de 2018	`https://bitbucket.org/satt/easylocal-3`
7	**ECJ**	*Framework* para Otimização Evolucionária	White (2012)	v.25, Novembro de 2017	`http://cs.gmu.edu/~eclab/projects/ecj/`
8	**EvA2**	*Framework* para Otimização Evolucionária	Kronfeld et al. (2010)	v2.2.0, Dezembro de 2015	`http://www.ra.cs.uni-tuebingen.de/software/EvA2/`
9	**FOM**	*Framework* para Otimização	Parejo et al. (2003)	v0.5, Julho de 2013	`http://www.isa.us.es/fom`
10	**HeuristicLab**	*Framework* para Otimização	Wagner e Affenzeller (2004), Wagner e Affenzeller (2005), Wagner et al. (2010, 2014)	v3.3.15, Janeiro de 2018	`http://dev.heuristiclab.com/`
11	**HotFrame**	*Framework* para Otimização	Fink et al. (1999), Fink e Voß (2002)	Fink e Voß (2002), Julho de 2003	–
12	**HyFlex**	*Framework* de Hiperheurísticas para Otimização	Ochoa et al. (2012)	v1.1, Setembro de 2011	`http://www.asap.cs.nott.ac.uk/external/chesc2011/`
13	**JABAT**	*Framework* Multiagente para Otimização	Barbucha et al. (2006, 2010)	Barbucha et al. (2010), 2010	–
14	**JAMES**	*Framework* para Busca Local	De Beukelaer et al. (2015), De Beukelaer et al. (2017)	v1.1, Agosto de 2016	`http://www.jamesframework.org/`
15	**JCLEC**	*Framework* para Otimização Evolucionária	Ventura et al. (2008), Cano et al. (2015)	v4.0.0, Julho de 2014	`http://jclec.sourceforge.net/`
16	**jMetal**	*Framework* para Problemas Multiobjetivo	Durillo et al. (2010), Durillo e Nebro (2011)	v5.4, Dezembro de 2017	`http://jmetal.github.io/jMetal/`
17	**LBMAS**	*Framework* Multiagente para Otimização	Lotfi e Acan (2015), Lotfi e Acan (2016)	Lotfi e Acan (2016), Junho de 2015	–
18	**MACS**	*Framework* Multiagente para Otimização	Martin et al. (2016)	v1.0, Abril de 2016	`http://simonpmartin.github.io/macs/`
19	**MAGMA**	*Framework* Multiagente para Otimização	Milano e Roli (2004)	Milano e Roli (2004), Agosto de 2002	–
20	**MALLBA**	Biblioteca de Esqueletos para Otimização	Alba et al. (2002, 2006, 2007)	v2.0, Junho de 2006	`http://neo.lcc.uma.es/mallba/easy-mallba/`
21	**MANGO**	*Framework* Multiagente para Otimização	Kerçelli et al. (2008), Günay et al. (2009), Aydemir et al. (2012)	v1.0, Outubro de 2013	`http://algopt.sabanciuniv.edu/mango/`
22	**OptFrame**	*Framework* para Otimização	Coelho et al. (2011, 2016a)	v3.0, Dezembro de 2017	`https://github.com/OptFrame/optframe`
23	**Opt4j**	*Framework* para Otimização Evolucionária	Lukasiewycz et al. (2011)	v3.1.4, Novembro 2015	`http://opt4j.sourceforge.net`
24	**ParadisEO**	*Framework* para Otimização	Cahon et al. (2004), Liefooghe et al. (2011), Melab et al. (2013), Humeau et al. (2013)	v2.0.1, Novembro de 2012	`http://paradiseo.gforge.inria.fr/`
25	**SMA**	*Framework* Multiagente para Otimização	Aydin (2010), Aydin (2012), Aydin (2013)	Aydin (2013), Setembro de 2013	–

Uma discussão profunda sobre a análise comparativa é apresentada em seguida, associada às Tabelas 2.2, 2.3, 2.4, 2.5 e 2.7. Cada uma dessas tabelas é detalhada a seguir, a fim de facilitar o entendimento da informação nelas contidas.

Para a avaliação a ser realizada, foram analisados os seguintes aspectos:

(i) características básicas e avançadas de uma estrutura de otimização metaheurística;

(ii) características relacionadas ao suporte do processo de otimização; e

(iii) sistemas multiagentes baseados em hibridização.

Estes aspectos foram desenvolvidos de acordo com Talukdar et al. (2003), Milano e Roli (2004), Silva (2007), Parejo et al. (2012) e Aydin (2013).

No caso de uma Arquitetura Multiagente para Otimização Combinatória, são identificados alguns aspectos desejáveis, que também são utilizados como referência na avaliação realizada na presente comparação:

(i) os agentes devem ser autônomos;

(ii) o ambiente deve ser considerado, e modelado, como uma abstração primária, que fornece as condições de existência dos agentes, no sentido de sua autonomia (Russell e Norvig, 1995; Santos, 2003);

(iii) a arquitetura deve ser flexível, permitindo desenvolver aplicações para diferentes problemas de otimização combinatória;

(iv) a interação dos agentes (comunicação) deve ser assíncrona, para garantir sua autonomia;

(v) a arquitetura deve ser baseada em mecanismos individuais e coletivos de ação dos agentes, contemplando mecanismos de cooperação, *ad hoc* ou não, e de competição e seleção dos agentes mais aptos;

(vi) a aplicação da arquitetura deve ser realizada em um sistema de computador distribuído e, eventualmente, heterogêneo. Além disso, esta aplicação, se estiver usando um único processador, deve ter cada agente definido em uma *thread* separada, de modo a garantir sua independência e assincronia;

(vii) a adição/remoção de novos agentes deve ser possível em tempo de execução;

(viii) deve ser desejável que novas heurísticas e metaheurísticas possam surgir da cooperação entre agentes, sem interferência ou pré-programação do operador do *designer*/*software*;

(ix) os agentes (ou o sistema) devem ter uma memória dinâmica e persistente dos ambientes, para permitir o início de suas atividades a partir de *boas soluções*;

(x) a arquitetura deve ser capaz de lidar com agentes ou grupos de agentes, bem como métodos pontuais ou populacionais;

(xi) a arquitetura deve ser projetada em um alto nível de abstração, portanto, sendo robusta, escalável e, além disso, permitir que suas aplicações específicas sejam facilmente desenvolvidas.

Com base nos *benchmarks* apresentados acima, um conjunto de características foi identificado e utilizado como diretriz para a análise comparativa dos *frameworks* descritos na Seção 2.2. Esse conjunto de características é mostrado na Seção 2.3.1 a seguir.

2.3.1 Características Analisadas

A comparação entre os *frameworks* selecionados foi realizada considerando as características colocadas em evidência em Parejo et al. (2012) e Silva (2007), bem como as características usadas por Talbi (2002) e Raidl (2006) para classificar metaheurísticas híbridas. Baseado nestas características, vinte e dois aspectos, agrupados em quatro diferentes categorias, foram analisados em cada framework selecionado:

(a) Características gerais para *Frameworks*;

(b) Características avançadas para *Frameworks*;

(c) Características Multi-Agente;

(d) Recursos de suporte ao processo de otimização.

As Seções 2.3.2, 2.3.3, 2.3.4 e 2.3.5 apresentam as características analisadas em cada uma das quatro categorias, bem como os resultados obtidos da avaliação dos *frameworks* descritos nas Seções 2.2.1 e 2.2.2. Além disso, é apresentada uma profunda discussão sobre a análise comparativa, focando no seu significado para o melhor entendimento de como empreender o desenvolvimento de um *framework* usando o conceito de multi-agente. Nas tabelas apresentadas nas seções seguintes, para melhor compreensão, `CMA` significa *Collaboration of Metaheuristic Algorithms*, e `SMA` significa *Swarm of Metaheuristic Agents*. Os *frameworks* são apresentados nessas tabelas em ordem alfabética.

2.3.2 Características Gerais para *Frameworks*

As características gerais de um *Framework* de Otimização usando Metaheurísticas estão listadas abaixo. Elas definem as questões gerais dos *frameworks*, como a classe de problemas que eles tratam, a classe de metaheurísticas que suportam e quais metaheurísticas são implementadas previamente. São definidas como:

(i) Classe de Problema: se a estrutura suporta apenas problemas com uma única função objetivo - SOOP (*Single Object Optimization Problem*) - ou se suporta problemas com múltiplas funções objetivo concorrentes - MOOP (*Multi-Objective Optimization Problem*);

(ii) Metaheurísticas implementadas: lista das metaheurísticas que já foram implementados no *framework*;

(iii) Heurística de base populacional: se o *framework* implementa heurísticas baseadas em populações;

(iv) Heurística baseada em uma única solução: se o *framework* implementa heurísticas baseadas em soluções únicas.

Tabela 2.2: Características Gerais de *Frameworks* para Otimização usando Metaheurísticas

	Framework	Características			
		Classe de Problemas	**Métodos implementados**	**Pop.**	**Traj.**
1	**AgE**	Mono-Objetivo	Genetic Algorithms	Sim	Não
2	**AMAM**	Mono-Objetivo	Iterated Local Search, Variable Neighborhood Search, GRASP, Genetic Algorithms	Sim	Sim
3	**AMF**	Mono-Objetivo	Evolutionary Algorithm, VNS	Sim	Sim
4	**CMA**	Mono-Objetivo	Genetic Algorithm, Tabu Search	Sim	Sim
5	**DAFO**	Mono-Objetivo	Panmictic GAs (genGA and ssGA); CGAs (Cooperative Coevolutionary Genetic Algorithm - CCGA, Loosely Coupled Genetic Algorithm - LCGA and hybrid LCGA - hLCGA); LCGA Hybridized with the Next Ascent Hill Climbing (NAHC)	Sim	Não
6	**EasyLocal++**	Mono-Objetivo	Exclusive to Local Search Algorithm	Não	Sim
7	**ECJ**	Mono-Objetivo, Multi-Objetivo	Genetic Algorithms, Evolutionary Strategies, Particle Swarm Optimization and Differential Evolution, NSGA-II e SPEA-II	Sim	Não
8	**EvA2**	Mono-Objetivo, Multi-Objetivo	Evolution Strategies (ES), Differential Evolution (DE), Genetic Algorithms (GA), Genetic Programming, Memetic Algorithms, Order-based GA, (IPOP-)CMA-ES, CHC Adaptive Search, Scatter Search, Nelder-Mead-Simplex, Multi-objective EA, NSGA II, PESA II, SPEA II, Cluster-based niching EA, Island-model EA, Simulated Annealing, Multi-start Hill Climbing, PBIL, Particle Swarm Optimization (PSO), Tribes, Clustering-based Niching PSO, Population-based Incremental Learning (PBIL), Bayesian Optimization Algorithm	Sim	Sim
9	**FOM**	Mono-Objetivo	Steepest Descent/Hill Climbing, Iterative Steepest Descent, Tabu Search, Simulated Annealing, GRASP, Variable Neighborhood Search, Evolutionary Algorithms (Genetic Algorithms, ES), Ant Systems (Classical Ant System, Ant Colony Optimization, Min-Max Ant System)	Sim	Sim
10	**HeuristicLab**	Mono-Objetivo, Multi-Objetivo	Population-based Algorithms, Trajectory based Algorithms, Multi-Objective Algorithm (NSGA-II), Optimal Algorithms, Extensible Algorithm	Sim	Sim
11	**HotFrame**	Mono-Objetivo	Local Search, Simulated Annealing, Tabu Search, Evolutionary algorithms, Candidate Lists, Neighborhood Depth and Pilot Method	Sim	Sim
12	**HyFlex**	Mono-Objetivo	Hyper-Heuristics (Iterated local search hyper-heuristic, AdapHH, VNS-TW, MLPHUNTER), Mutational or perturbation heuristics, Ruin-recreate (destruction-construction) heuristics, Hill-climbing or local search heuristics, Crossover heuristics.	Sim	Sim
13	**JABAT**	Mono-Objetivo	Local search algorithm, crossing algorithm, precedence tree algorithm, tabu search algorithm, MCS-based algorithm, path relinking algorithm, gene expression programming algorithm, tabu search algorithm, Lin-Kernighan heuristics, 2-opt and 3-opt heuristics, random local search, hill-climbing local search, simulated annealing	Não	Sim
14	**JAMES**	Mono-Objetivo	Hill climbing, tabu search, variable neighborhood search and parallel tempering	No	Sim
15	**JCLEC**	Mono-Objetivo, Multi-Objetivo	Classic algorithms: simple generational, steady state and CHC, Multi-objective algorithms: NSGA-II and SPEA2, Memetic algorithms: generational and steady state, Scatter search algorithm, Niching algorithms: clearing, sequential and fitness sharing	Sim	Não
16	**jMetal**	Mono-Objetivo, Multi-Objetivo	Evolutionary Techniques	Sim	Não
17	**LBMAS**	Mono-Objetivo	Genetic Algorithms (GAs), Differential Evolution (DE), Simulated Annealing (SA), Ant Colony Optimization (ACO), Great Deluge Algorithm (GDA), Tabu Search (TS), and the Cross Entropy (CE) method	Sim	Sim
18	**MACS**	Mono-Objetivo	RandNEH and RandCWS	Não	Sim
19	**MAGMA**	Mono-Objetivo	–	–	–
20	**MALLBA**	Mono-Objetivo	Exact Techniques, Hill Climbing, Metropolis, Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA) and hybrid techniques (GA+TS, GS+SA,Bnb+SA)	Sim	Sim
21	**MANGO**	Mono-Objetivo	BFGS quasi-Newton Algorithm, Trust-region Algorithm, Random Search	Não	Sim
22	**OptFrame**	Mono-Objetivo, Multi-Objetivo	First Improvement, Best Improvement, Hill Climbing, Iterated Local Search, Simulated Annealing, Tabu Search, Variable Neighborhood Search, the basic versions of Genetic and Memetic Algorithms	Sim	Sim
23	**Opt4j**	Mono-Objetivo, Multi-Objetivo	SPEA2 and NSGA2, differential evolution, particle swarm optimization, and simulated annealing	Sim	Sim
24	**ParadisEO**	Mono-Objetivo, Multi-Objetivo	Modules: EO - Evolutionary Algorithms; MO - Trajectory Methods; MOEO: Evolutionary Methods	Sim	Sim
25	**SMA**	Mono-Objetivo	Bee colonies Optimization (BCO), Particle swarm optimisation (PSO), evolutionary simulated annealing (ESA)	Sim	Não

A Tabela 2.2 apresenta os resultados referentes à análise das características gerais dos *Frameworks* de Otimização usando Metaheurísticas descritos na Seção 2.2. Conclui-se que, dentre os vinte e cinco quadros avaliados, apenas oito (`ECJ`, `EvA2`, `HeuristicLab`, `JCLEC`, `jMetal`, `OptFrame`, `Opt4j`, `ParadisEO`) implementam a solução de problemas tanto mono-objetivo quanto problemas multiobjetivos. Os outros *frameworks* são direcionados apenas para problemas mono-objetivo.

Com relação aos métodos implementados, podemos dizer, também a partir da Tabela 2.2, que onze *frameworks* estão focados em técnicas específicas de otimização. Os *frameworks* `EasyLocal++`, `JABAT`, `JAMES`, `MACS` e `MANGO` implementam técnicas baseadas em trajetória e os *frameworks* `AgE`, `DAFO`, `ECJ`, `JCLEC`, `jMetal` e `SMA` implementam técnicas baseadas em população. Neste contexto, estes *frameworks* podem ser vistos como especializados apenas para estas técnicas, não sendo possível estendê-los além desses métodos especificados. Os *frameworks* `EvA2` e `Opt4j` também são especializados em técnicas baseadas em população, embora algumas técnicas de trajetória também sejam implementadas. O *framework* `Hyflex` é especializado no desenvolvimento de hiperheurísticas e também não oferece flexibilidade na extensão de métodos diferentes da proposta inicial para a qual foi desenvolvido.

Os *frameworks* que implementam maior número e uma variedade maior de métodos são `HeuristicLab` e `ParadisEO`. Estes *frameworks*, nesse sentido, podem ser considerados os mais completos dentre os avaliados nesta análise. O *framework* `ParadisEO` é o que apresenta o maior número e a maior diversidade de métodos implementados, incluindo algoritmos evolutivos e métodos de trajetória, e, além disso, apresenta a possibilidade de lidar com problemas mono e multiobjetivos. Por outro lado, o *framework* `MAGMA` permanece apenas como um modelo conceitual, não sendo conhecido, ao menos pela autora desta Tese, no momento de sua conclusão, versões executáveis desse *framework*.

2.3.3 Características Avançadas para *Frameworks*

As características avançadas dos *Frameworks* para Otimização usando Metaheurísticas estão listadas em sequência. Essas características definem as principais questões relativas à implementação de *frameworks* e, portanto, sua capacidade de adaptação a novos desafios, especialmente mudanças nos paradigmas tecnológicos que podem surgir ao longo do tempo. Essas características avançadas são definidas como:

(i) Linguagem de implementação: qual é a linguagem usada na implementação do *framework*;

(ii) Portabilidade: se o *framework* pode ser compilado/executado em diferentes plataformas;

(iii) Modelo de Licenciamento: tipo de licença de *software* utilizada;

(iv) Hibridização: capacidade do *framework* em permitir a hibridização de metaheurísticas;

(v) Suporte para computação paralela e distribuída: em uma aplicação paralela, as subtarefas são executadas simultaneamente em um ou mais processadores. Para este propósito, recursos de programação paralelos e distribuídos devem estar disponíveis. Nesta análise, avaliamos a disponibilidade de *threads* e recursos para computação distribuída nos *frameworks*:

(v.i) *Threads*: se a estrutura fornece recursos para desenvolvimento de aplicativos usando threads. *Threads* são as subtarefas de um programa paralelo, que permitem que várias execuções ocorram no mesmo ambiente de aplicativo em um computador;

(v.ii) Computação Distribuída: se a estrutura fornece recursos de programação distribuída para o desenvolvimento de aplicativos. A programação distribuída consiste em executar aplicativos concorrentes em diferentes máquinas;

(vi) Hiperheurística: capacidade do *framework* em permitir o desenvolvimento de hiperheurísticas;

(vii) Projeto em alto nível de abstração: os recursos do projeto utilizados devem permitir o desenvolvimento de componentes flexíveis e reutilizáveis.

Tabela 2.3: Características Avançadas de *Frameworks* para Otimização usando Metaheurísticas

	Framework	**Características**							
		Linguagem	**Port.**	**Licença**	**Hib.**	**Suporte a computação paralela e distribuída**		**Hip.**	**Projeto em alto nível de abstração**
						Threads	**Computação distribuída**		
1	**AgE**	Erlang	Não	Apache License 2.0	Sim	Sim	Sim		Sim
2	**AMAM**	Java	Sim	–	Sim	Sim	Não	Não	Sim
3	**AMF**	Java	Sim	–	Sim	–	–	Não	Sim
4	**CMA**	Plataforma A-Globe	–	–	Sim	Sim	Sim	Não	–
5	**DAFO**	–	–	–	Sim	–	–	Não	–
6	**EasyLocal++**	C++	Sim	–	Sim, Predefinido	–	–	Não	Sim
7	**ECJ**	Java	Sim	–	Não	Sim	Sim	Não	Sim
8	**EvA2**	Java	Sim	LGPL v3	Não	Sim	Sim	Não	Sim
9	**FOM**	Java	Sim	GNU LGPL	Não	Não	Não	Não	Sim
10	**HeuristicLab**	.NET	Não	GPLv3	Não	Sim, Versão HeuristicLab Grid	Sim, Versão HeuristicLab Grid	Não	Sim
11	**HotFrame**	C++	Sim	–	Não	Não	Não	Não	Sim
12	**HyFlex**	Java	Sim	–	Sim, Limitado	–	–	Sim	Sim
13	**JABAT**	Java	Sim	–	Sim	Sim	Sim	Não	Sim
14	**JAMES**	Java	Sim	Apache License 2.0	Não	Sim	Não	Não	Sim
15	**JCLEC**	Java	Sim	GNU GPLv3	Não	Não	Não	Não	Sim
16	**jMetal**	Java	Sim	GNU LGPL	Sim	Sim	Sim	Não	Sim
17	**LBMAS**	–	–	–	Sim	–	–	Não	–
18	**MACS**	Java (Plataforma `JADE`)	Sim	GPLv3	Sim	Sim	Sim	Não	Sim
19	**MAGMA**	Modelo Conceitual	–	–	Sim, Limitado	–	–	Não	–
20	**MALLBA**	C++	Sim	–	Não	Sim	Sim	Não	Sim
21	**MANGO**	Java	Sim	–	Sim	Sim	Sim	Não	Sim
22	**OptFrame**	C++	Sim	GNU LGPLv3	Sim	Sim	Sim	Não	Sim
23	**Opt4j**	Java	Sim	LGPL v2	Não	Sim	Sim	Não	Sim
24	**ParadisEO**	C++	Sim	GNU LGPL	Sim, Limitado	Sim, Módulo Específico (SMP e PEO)	Sim, Módulo Específico (SMP e PEO)	Não	Sim
25	**SMA**	Pop C++	Sim	–	Sim, Limitado	–	–	Sim	Não

A Tabela 2.3 apresenta os resultados referentes à análise das características avançadas para os *Frameworks* de Otimização usando Metaheurística descritos na seção 2.2. Nesta tabela, a coluna "Ling." mostra a linguagem utilizada para a implementação do *framework*; a coluna "Port." informa a respeito das características de portabilidade; a coluna "Licença" diz o modelo de licenciamento adotado; a coluna "Hib" informa a respeito das possibilidades de uso de hibridização no *framework*; a coluna "Hip." diz se o *framework* suporta ou não hiperherísticas.

Avaliando a Tabela 2.3, fica claro que a linguagem Java é usada no desenvolvimento da maioria dos *frameworks* analisados. Isso se deve principalmente à portabilidade da linguagem, ou seja, utilizando Java é possível desenvolver aplicativos para diversos dispositivos, como celulares, televisores, entre outros. Da mesma forma, é possível desenvolver um aplicativo em qualquer sistema operacional que possa ser executado por outro sistema operacional. Além disso, a Tabela 2.3 mostra que onze *frameworks* analisados estão disponíveis sob a licença de *software* livre. Os *frameworks* `HeuristicLab`, `JCLEC` e `MACS` estão sob a licença de *software* `GNU` *General Public License* (`GPL`), que define que qualquer trabalho derivado do original deve ser disponibilizado sob os mesmos termos da licença original. As estruturas `EvA2`, `FOM`, `jMetal`, `OptFrame`, `Opt4j` e `ParadisEO` estão sob a `GNU` *Lesser General Public License* (`LGPL`), que é semelhante à licença `GPL`, mas adiciona algumas permissões de licença, como permissão para participar do *software* que não está sob as licenças `GPL` ou `LGPL`, incluindo a possibilidade de associação com *software* proprietário. As estruturas `AgE` e `JAMES` estão disponíveis sob a licença `Apache`, que permite o uso e a distribuição do código-fonte e é compatível com a licença `GPL`.

A hibridização é avaliada, nesta análise, por duas maneiras: (i) a facilidade e flexibilidade que a estrutura fornece para desenvolver métodos híbridos, mostrados na Tabela 2.3; e (ii) se é possível hibridizar metaheurísticas a partir da interação entre os agentes existentes no *framework*, principalmente por meio da cooperação, apresentada na Tabela 2.4.

A Tabela 2.3 destaca três situações diferentes em relação à hibridização. A primeira situação refere-se aos *frameworks* marcados *Sim* na respectiva coluna da Tabela 2.3. Este conjunto de *frameworks* (`AgE`, `AMAM`, `AMF`, `CMA`, `DAFO`, `JABAT`, `jMetal`, `LBMAS`, `MACS`, `MANGO` e `Optframe`) são as propostas que efetivamente permitem a hibridização de métodos heurísticos e metaheurísticos em um contexto mais amplo, isto é, nestes casos a técnica de hibridização pode ser utilizada sem restrições. É importante ressaltar que, com exceção dos *frameworks* `JMetal` e `Optframe`, todos esses *frameworks* (`AgE`, `AMAM`, `AMF`, `CMA`, `DAFO`, `JABAT`, `LBMAS`, `MACS` e `MANGO`) utilizam o conceito de agentes em sua estruturação, conforme a Tabela 2.4. A segunda situação refere-se aos *frameworks* marcados com *Hibridização pré-definida* na coluna de hibridização da Tabela 2.3. Esta é a situação na qual a hibridização é realizada a partir do uso de estruturas já pré-definidas pelo próprio *framework*. O *framework* `EasyLocal++` apresenta apenas algoritmos híbridos pré-definidos. Sendo assim, não há facilidades para o desenvolvimento de novos algoritmos híbridos. O módulo `ParadisEO-PEO` do *framework* `ParadisEO` apresenta, da mesma forma, modelos pré-definidos de hibridização metaheurística. A terceira situação refere-se às estruturas marcadas como *hibridização limitada* na coluna de hibridização da Tabela 2.3. Esta é a situação em que a hibridização é limitada pelas estruturas do próprio *framework*. A hibridização no *framework* `Hyflex` é limitada à forma como as hiperheurísticas são definidas pelo próprio *framework*. Na estrutura `SMA`, a hibridização é limitada às formas de interação dos algoritmos populacionais. Na estrutura `MAGMA`, a hibridização é limitada às formas de interação entre os níveis da arquitetura.

Quanto à possibilidade de utilizar computação paralela e distribuída, três *frameworks* (FOM, HotFrame, JCLEC) não oferecem esses recursos. O *framework* MAGMA, apresentado apenas na forma conceitual, não fornece elementos suficientes para que se possa declarar quaisquer conclusões sobre o uso desses recursos. As estruturas AMF, DAFO, EasyLocal++, HyFlex, LBMAS e SMA não relatam a disponibilidade desses recursos. Todas as outras estruturas analisadas oferecem recursos de computação paralela e distribuída. De todos esses, apenas os *frameworks* AMAM e JAMES oferecem a possibilidade de inserção de *threads*; no entanto, não fornecem recursos de programação distribuída. No caso dos *frameworks* HeuristicLab e ParadisEO, essa possibilidade está associada ao uso de módulos específicos para esse fim. Deve ser notado que o *framework* ParadisEO possui um módulo específico para computação paralela usando arquitetura GPU (*Graphics Processing Units*), implementando apenas metaheurísticas de busca local (trajetória) (Melab et al., 2013). O *framework* AgE se destaca por oferecer recursos que facilitam o desenvolvimento distribuído, especialmente em arquiteturas *multi-core*.

Com relação à possibilidade de implementação de hiperheurísticas, podemos considerar que, além da Hyflex, ou seja, o *framework* específico construído para essa formulação, o único *framework* que possui essa capacidade é o SMA, que utiliza algoritmos baseados em inteligência coletiva para coordenar metaheurísticas e possibilitar a implementação de hiperheurísticas. Os trabalhos restantes não implementam diretamente hiperheurísticas. Os *frameworks* baseados em agentes, como AMAM, AMF, CMA, DAFO, JABAT, LBMAS, MACS, MAGMA e MANGO, também podem permitir a implementação de hiperheurísticas, de forma indireta, inserindo estruturas de controle e coordenação e com base nos próprios agentes. No entanto, é importante salientar que as hiperheurísticas podem comprometer a autonomia dos agentes envolvidos no *framework*, por sua inerente ação de acordo com seu próprio formato de escolha, controle e coordenação do desempenho das heurísticas e metaheurísticas. Isso é especialmente verdadeiro nos casos em que a autonomia dos agentes no espaço de solução do problema é um requisito para o funcionamento do *framework*. Nesses casos, a existência de estruturas de coordenação e controle, externas ao próprio agente, retira do agente sua capacidade de decisão quanto a sua relação com esse espaço de solução e com os demais agentes envolvidos.

Em relação às técnicas de projeto e desenvolvimento utilizadas, a maioria utiliza boas práticas de programação, o que permite aumentar a reutilização e facilitar o uso das ferramentas oferecidas. O *framework* MAGMA não permite nenhuma avaliação em relação a esta questão, pois possui apenas proposta conceitual. O *framework* CMA é desenvolvido com base na arquitetura *A-Globe* (Sislak et al., 2005), não sendo possível, portanto, inferir qualquer análise em relação às características de seu próprio projeto. Vale ressaltar que a maioria dos *frameworks* analisados é implementada em linguagens não proprietárias e baseada no paradigma orientado a objetos.

2.3.4 Características Multiagente

Nesta seção, concentramos nossa atenção apenas nos onze *frameworks* que foram construídos usando conceitos multiagentes. O objetivo é analisar e classificar essas propostas. Para isso, seis características são avaliadas, destacando como cada um destes *frameworks* incorpora propriedades inerentes a sistemas multiagentes. As características multiagentes avaliadas são definidas como:

(i) Agentes: se o *framework* usa o conceito de agente em sua definição;

(ii) Ambiente: como o ambiente do sistema multiagente utilizado é descrito;

(iii) Autonomia dos agentes: os agentes são autônomos, ou seja, agem sem interferência de outras entidades dos sistemas;

(iv) Cooperação: quando há compartilhamento de informações entre os agentes que efetivamente influenciam suas ações;

(iv.i) Tipo de comunicação:

(iv.i.i) Síncrona: se a cooperação é realizada a partir da comunicação síncrona, ou seja, se os processos, em algum instante de tempo, são interrompidos e se envolvem em alguma forma de comunicação e troca de informações;

(iv.i.ii) Assíncrona: se a cooperação é realizada a partir de comunicação assíncrona, ou seja, se cada processo é responsável por sua própria pesquisa e por se comunicar com outros processos. A pesquisa global termina apenas se todos os processos de pesquisa individuais terminarem;

(iv.ii) Descrição: descreve como a cooperação ocorre no *framework*;

(v) Aprendizagem: se algum tipo de aprendizado é implementado nos agentes ou como consequência de suas ações. Esta característica permite ao agente assimilar o conhecimento sobre o ambiente e/ou sobre outros agentes e usá-lo para melhorar sua capacidade de ação;

(vi) Hibridização: a capacidade de hibridizar metaheurísticas, obtidas a partir da interação entre os agentes existentes no *framework*, principalmente através da cooperação. Observe que a capacidade de interação entre os vários agentes empregados na solução do problema de otimização facilita o surgimento de métodos híbridos.

As Tabelas 2.4, 2.5 e 2.6 apresentam os resultados da avaliação realizada nesses onze *frameworks* com relação às características mostradas acima, referentes ao conceito de multiagentes. Eles são discutidos na sequência.

A Tabela 2.4 avalia se o *framework* usa ou não os conceitos de agentes autônomos. Onze propostas fazem uso deste conceito: `AgE`, `AMAM`, `AMF`, `CMA`, `DAFO`, `JABAT`, `LBMAS`, `MACS`, `MAGMA`, `MANGO` e `SMA`. O uso desse conceito permite a hibridização de metaheurísticas, seja na forma de uma combinação de métodos ou, em particular, usando o conceito de cooperação, que é inerente aos Sistemas Multiagentes.

Os recursos multiagentes foram avaliados apenas em relação aos *frameworks* que implementam o conceito de agentes em sua definição. O primeiro recurso avaliado é o ambiente da estrutura multiagente definido em cada proposta. O ambiente em um sistema multiagente é aquele em que o agente está localizado e envolve entidades passivas do sistema, ou seja, aquelas sem a capacidade de agir. O agente pode ter uma visão global ou local de seu ambiente, dependendo de como ele foi configurado.

A maioria das estruturas multiagente avaliadas (`AMF`, `CMA`, `JABAT`, `LBMAS`, `MAC`, `MANGO` e `SMA`) não deixa claro como é definido o ambiente no qual os agentes atuam. Apenas os *frameworks* `AgE`, `AMAM`, `DAFO` e `MAGMA` descrevem seus ambientes multiagentes. Na arquitetura `MAGMA`, o ambiente é definido pela tripla $(\mathcal{S}, \mathcal{N}, \mathcal{F})$, em que $\mathcal{S}$ é o conjunto de soluções, $\mathcal{N}$ é a função que define a estrutura de vizinhança e $\mathcal{F}$ é a função objetivo. No *framework* `AMAM`, o ambiente é o elemento que permite a flexibilidade da arquitetura, uma vez que corresponde ao espaço de busca do problema tratado. Assim, o ambiente deve ser

Tabela 2.4: Características Multiagente dos *Frameworks* para Otimização usando Metaheurísticas

Framework		Características				
		Agentes	**Ambiente**	**Autonomia dos agentes**	**Aprendizado**	**Hibridização (da interação entre os agentes)**
1	**AgE**	Sim	Sim. O ambiente do sistema multiagente é o espaço de busca do problema tratado.	Sim	Não	Sim
2	**AMAM**	Sim	Sim. O ambiente do sistema multiagente é o espaço de busca do problema tratado.	Sim	Sim. *Learning Automata*.	Sim
3	**AMF**	Sim	–	Sim	Sim. Aprendizado por Reforço e Mimetismo.	Sim
4	**CMA**	Sim	–	Sim	Não	Sim
5	**DAFO**	Sim	Sim. Problema de otimização que é fornecido e especificado pelo usuário.	Sim	Não	Sim
6	**JABAT**	Sim	–	Sim	Não	Sim
7	**LBMAS**	Sim	–	Sim, Limitado	Sim. *Learning automata*.	Sim
8	**MACS**	Sim	–	Sim	Sim. Aprendizado por Reforço.	Sim
9	**MAGMA**	Sim	Sim, Fitness Landscapes.	Não	Não	Sim
10	**MANGO**	Sim	–	Sim	Não	Sim
11	**SMA**	Sim	–	Não	Não	Sim

modelado para cada problema específico e a mudança de ambiente instanciado na aplicação corresponde à troca do problema a ser resolvido. Na *framework* `DAFO`, o ambiente também é definido pelo problema de otimização, que é fornecido e especificado pelo usuário. A Tabela 2.6 também evidencia essa relação entre o ambiente em que os agentes atuam e o espaço de soluções do problema a ser resolvido.

Em um sistema multiagente, a autonomia garante ao agente liberdade em suas escolhas e na execução de suas tarefas. A liberdade de ação envolve principalmente a comunicação entre agentes no ambiente multiagente, permitindo que eles escolham com quem desejam interagir. Nesse sentido, parte da autonomia do agente está diretamente relacionada à forma como a interação é definida. Os *frameworks* `LBMAS`, `MAGMA` e `SMA` definem estruturas que limitam, em parte, a autonomia dos agentes e a cooperação entre eles. No *framework* `LBMAS`, a autonomia dos agentes é comprometida pela presença de agentes intermediários (agente *Manager*, agente *Archive*, agente Solution Pool), responsáveis por gerenciar a troca de informações entre os agentes.

Na arquitetura `MAGMA`, este comprometimento da autonomia é devido ao fato de os agentes serem alocados para níveis subordinados, dependendo uns dos outros. O agente, na arquitetura `MAGMA`, é autônomo na realização de sua atividade específica. No entanto, depende de outros agentes e é coordenado pelos agentes de nível superior (nível 3). A cooperação, na arquitetura `MAGMA`, também é limitada por essa estrutura de níveis. A estrutura `SMA` usa inteligência de enxame para coordenar os agentes de pesquisa. Assim, os agentes são autônomos em seu processo de busca de soluções. No entanto, um algoritmo de cooperação rege a cooperação entre eles. A cooperação, na proposta `SMA`, limita-se à troca de informações entre os agentes ao final de cada geração do algoritmo evolutivo utilizado.

Os *frameworks* `AgE`, `AMAM`, `AMF`, `CMA`, `DAFO`, `JABAT`, `MACS` e `MANGO`, também baseados em sistemas multiagentes, possuem agentes autônomos e processos cooperativos que não interferem nessa autonomia. Essas características são essenciais para futuros desenvolvimentos

Tabela 2.5: Características de Cooperação Multiagente dos *Frameworks* de Otimização usando Metaheurística

Framework		Características de Cooperação		
		Tipo de Comunicação		Descrição da Cooperação
		Síncrono	Assíncrono	
1	**AgE**	Não	Sim	Os processos Erlang se comunicam usando mensagens assíncronas, o que elimina a possibilidade de criar gargalos de sincronização.
2	**AMAM**	Não	Sim	A cooperação ocorre através do compartilhamento de soluções disponíveis no Solutions *Pool*.
3	**AMF**	Não	Sim	A cooperação entre esses agentes ocorre de duas maneiras: (i) um agente compartilha sua solução mais conhecida (essa solução pode ser explorada por outros agentes com operadores de crossover); (ii) um agente compartilha suas regras de decisão interna para permitir o mímico de comportamento (favorece os comportamentos de pesquisa que geralmente encontram novas soluções melhores).
4	**CMA**	Não	Sim	A troca de informações entre agentes é definida pela plataforma multiagente A-Globe, que oferece recursos multi-agentes, como agentes esqueletos, mensagens, protocolos de bate-papo etc. A cooperação ocorre a partir do compartilhamento de populações globais de soluções candidatas armazenadas em um *pool* de soluções.
5	**DAFO**	Sim	Não	A interação entre agentes é definida por protocolos de comunicação (proposições FIPA ACL e FIPA-SL). O modelo de organização MAS4EVO especifica os padrões globais de cooperação no comportamento de agentes, definindo uma organização multiagente baseada em estratégias coevolucionárias.
6	**JABAT**	Não	Sim	A cooperação ocorre através de memórias interconectadas, nas quais as soluções estão constantemente circulando.
7	**LBMAS**	Não	Sim	Os agentes cooperam compartilhando suas experiências individuais através de um arquivo de memória externa de dois estágios.
8	**MACS**	Sim	Não	A cooperação ocorre através da conversação, que é um protocolo de cooperação baseado nas soluções parciais de retenção consideradas possíveis constituintes de boas soluções futuras. Existe um agente que assume o papel de um iniciador e os outros são respondedores.
9	**MAGMA**	Não	Sim	O compartilhamento de informações pode acontecer de duas maneiras: (i) horizontalmente, envolve a interação entre os agentes de mesmo nível; E (ii) verticalmente, envolve a interação entre diferentes agentes de nível. A comunicação entre agentes pode ser implementada através de qualquer tipo de mecanismo e protocolo.
10	**MANGO**	Não	Sim	O ambiente multi-agente utilizado no desenvolvimento do Mango (JADE) fornece protocolos de cooperação e modelos organizacionais para realizar a comunicação. A troca de mensagens entre os agentes do *framework* pode acontecer de duas maneiras: interrupções e caixas de correio. Interrupções: quando uma mensagem é enviada para um agente, isso interrompe imediatamente o processo de pesquisa. Caixas de correio: a mensagem é armazenada e o agente verifica sua caixa de correio quando é mais apropriado.
11	**SMA**	Sim	Não	Um algoritmo de cooperação é aplicado, após cada geração dos algoritmos evolutivos, trocando informações relacionadas às soluções obtidas.

Tabela 2.6: Características Multiagente de *Frameworks* de Otimização usando Metaheurística: Relação com espaço de busca

Framework		Características		
		Decomposição do Espaço de Busca	Decomposição Metaheurística	Agentes Metaheurísticos
1	**AgE**	X	X	
2	**AMAM**			X
3	**AMF**		X	
4	**CMA**			X
5	**DAFO**	X	X	X
6	**JABAT**		X	X
7	**LBMAS**			X
8	**MACS**			X
9	**MAGMA**		X	
10	**MANGO**			X
11	**SMA**			X

e expansões dessas estruturas.

A cooperação usada pelos *frameworks* baseados em agentes é discutida na Tabela 2.5. Duas questões são avaliadas em relação à cooperação: tipos e funcionamento. Os tipos de cooperação utilizados pelos *frameworks* são avaliados de acordo com o tipo de comunicação adotado, a saber: Síncrono e Assíncrono. Conforme observado na Tabela 2.5, a maioria das estruturas baseadas em agente usa comunicação assíncrona, que é aquela em que cada agente é responsável por sua própria pesquisa e por estabelecer comunicação com outros processos, independentemente de um momento específico ou de outros agentes.

A Tabela 2.4 nos permite identificar uma lacuna importante: o uso de aprendizado por agentes na busca de soluções. De acordo com Narendra e Thathachar (1974),

> *aprendizagem é definida como qualquer mudança relativamente permanente no comportamento resultante da experiência passada, e um sistema de aprendizagem é caracterizado por sua capacidade de melhorar seu comportamento com o tempo, em certo sentido tendendo a um objetivo final.*

Esta característica permite ao agente assimilar conhecimento sobre o ambiente e/ou sobre outros agentes e usá-lo para melhorar sua capacidade de ação. Os *frameworks* `AMAM`, `AMF`, `LBMAS` e `MACS` implementam diferentes tipos de aprendizado de máquina. Os *frameworks* `AMAM` e `LBMAS` usam conceitos de *Learning Automata* (Narendra e Thathachar, 1974) na tomada de decisão do agente. No modelo de aprendizagem baseado em *Learning Automata*, as ações do agente são selecionadas de acordo com uma distribuição de probabilidade específica, que é atualizada com base nas respostas do ambiente. Na estrutura `AMAM`, o aprendizado é usado para definir a ordem de aplicação das estruturas de vizinhança aplicadas nas buscas locais. Na estrutura `LBMAS`, a cada iteração, o aprendizado é usado na escolha da próxima metaheurística a ser usada. A estrutura `AMF` usa dois mecanismos de aprendizado: Aprendizado por Reforço (Kaelbling et al., 1996) e Aprendizado de Mimetismo (Yamaguchi et al., 1997). O Aprendizado por Reforço, utilizado na aprendizagem individual do agente, permite selecionar o operador mais adequado de acordo com suas experiências. O Aprendizado por Mimetismo permite a cooperação entre os agentes, compartilhando a matriz de pesos do agente considerado mais eficiente. O conceito de aprendizagem é utilizado, no *framework* `MACS`, a partir da memória de curto prazo de bons arcos, definida para cada agente. Essa memória é utilizada para modificar o desempenho de cada metaheurística, permitindo-lhes, portanto, encontrar melhores soluções. Nestes quatro *frameworks*, o aprendizado influencia diretamente a tomada de decisão do agente, buscando melhorar a qualidade das soluções obtidas com base na experiência adquirida. É importante notar que o *framework* `CMA` possui uma estrutura de aprendizado, utilizada, neste caso, para realizar o autoajuste dos parâmetros implementados nos métodos.

2.3.5 Características de Suporte ao Processo de Otimização

As características de suporte do processo de otimização presentes nos *Frameworks* de Otimização usando Metaheurísticas descritas na Seção 2.2 são apresentadas a seguir. Esses recursos representam tanto a usabilidade para o usuário quanto a possibilidade de ajuste automático de parâmetros no processo de otimização. É importante ressaltar que a presença ou ausência dessas características evidencia o grau de maturidade adquirido pelas propostas de *framework* em seu processo de desenvolvimento. Esses recursos de suporte são definidos como:

(i) Análise Estatística: permite a análise estatística dos resultados;

Tabela 2.7: Suporte ao Processo de Otimização

	Framework	Características		
		Análise Estatística	**Auto-Configuração (*Self-tuning*)**	**GUI (Interface de Usuário)**
1	**AgE**	Sim	Não	Não
2	**AMAM**	Não	Não	Não
3	**AMF**	Não	Não	Não
4	**CMA**	Não	Sim	Não
5	**DAFO**	Não	Não	Não
6	**EasyLocal++**	Não	Não	Não
7	**ECJ**	Não	Não	Sim
8	**EvA2**	Não	Não	Sim
9	**FOM**	Sim	Sim, Limitado	Sim
10	**HeuristicLab**	Sim	Não	Sim
11	**HotFrame**	Não	Não	Não
12	**HyFlex**	Não	Não	Não
13	**JABAT**	Não	Não	Não
14	**JAMES**	Não	Não	Não
15	**JCLEC**	Não	Não	Sim
16	**jMetal**	Sim	Não	Não
17	**LBMAS**	Não	Não	Não
18	**MACS**	Não	Não	Não
19	**MAGMA**	Não	Não	Não
20	**MALLBA**	Não	Não	Não
21	**MANGO**	Não	Não	Não
22	**OptFrame**	Não	Não	Não
23	**Opt4j**	Não	Não	Sim
24	**ParadisEO**	Não	Não	Não
25	**SMA**	Não	Não	Não

(ii) Auto-ajuste (*Self-tuning*): se possui a capacidade de ajuste automático dos parâmetros metaheurísticos implementados;

(iii) `GUI`: a interface do usuário.

A Tabela 2.7 mostra os resultados referentes à avaliação dos *frameworks* listadas na Tabela 2.1 sobre as questões relacionadas aos recursos de suporte ao processo de otimização. A partir dessa tabela, fica claro que poucos desses recursos são cobertos pelos *frameworks* avaliados. Apenas as estruturas `AgE`, `FOM`, `HeuristicLab`, `JCLEC`, `jMetal` e `ParadisEO` possuem recursos de análise estatística. O autoajuste dos parâmetros está presente em apenas dois *frameworks* (`CMA` e `FOM`). A proposta `CMA`, pelo Agente *Consultor*, permite configurar automaticamente o valor dos parâmetros dos relatórios recebidos sobre os algoritmos executados. O *framework* `FOM`, apesar de afirmar a existência de ajuste automático de parâmetros, não especifica como os dados observados na execução das metaheurísticas são utilizados no autoajuste dos parâmetros para as próximas execuções ou como são utilizados para melhorar os próximos resultados. O *framework* apenas especifica a possibilidade de monitorar os parâmetros e gerar gráficos que os demonstrem. Quanto à disponibilidade de interface gráfica, os *frameworks* `ECJ`, `EvA2`, `FOM`, `HeuristicLab`, `Hotframe`, `JCLEC`, `jMetal` e `Opt4j` afirmam ter tal facilidade.

2.4 Aprendizado Automático

O Aprendizado Automático ou Aprendizagem de Máquina (em inglês, *Machine Learning*) é um ramo da Ciência da Computação que estuda formas de programar sistemas de computadores para realizarem tarefas sem usar instruções específicas. Desta forma, através do aprendizado de máquina, o objetivo é atribuir ao computador a habilidade de aprender automaticamente.

Nos últimos anos, a Aprendizagem de Máquina tem atraído um interesse crescente da comunidade científica, devido principalmente a sua capacidade de produzir modelos capazes de analisar dados maiores e mais complexos e gerar resultados precisos.

Em abordagens tradicionais de aprendizagem automática, os sistemas são treinados através de pares de entrada e saída. Neste caso, se faz necessário um especialista de domínio capaz de fornecer valores de referência, que serão usados pelos algoritmos para identificar padrões que os permitam aprender. Entretanto, há situações em que não se conhece *a priori* o ambiente em que o agente está inserido, assim como a função que modela a política ótima (comportamento que o agente segue para atingir seus objetivos).

Neste sentido, se torna essencial um aprendizado automático capaz de atuar em um ambiente dinâmico e desconhecido. O Aprendizado por Reforço (em inglês, *Reinforcement Learning - RL*) é uma das formas de aprendizagem de máquina que permite a exploração de um ambiente dinâmico e é uma das técnicas usadas nesta proposta, na definição dos agentes adaptativos (veja Seção 2.4.1).

Autômatos de Aprendizagem (em inglês, *Learning Automata*), conforme Narendra e Thathachar (1974), se enquadra como Aprendizado por Reforço, permitindo a atuação em ambientes com pouca informação *a priori* e que informações adicionais são adquiridas em tempo real. Os Autômatos de Aprendizagem são também usados na definição das habilidades auto adaptativas dos agentes no *framework* AMAM. A seguir, estes conceitos são melhor apresentados.

2.4.1 Aprendizado por Reforço

O Aprendizado por Reforço (RL) consiste em aprender o que fazer em um ambiente dinâmico a partir de interações baseadas em tentativa e erro. Essas interações são reforçadas de acordo com os efeitos que causam no ambiente. No modelo definido pela aprendizagem por reforço, não há pares de entrada/saída e, portanto, o agente precisa reunir experiências para melhorar seu desempenho.

De acordo com Narendra e Thathachar (1974),

> *aprendizagem é definida como qualquer mudança relativamente permanente no comportamento resultante da experiência passada, e um sistema de aprendizagem é caracterizado por sua capacidade de melhorar seu comportamento com o tempo, em certo sentido tendendo a um objetivo final.*

Na aprendizagem por reforço, o comportamento é melhorado a partir de recompensas obtidas durante as interações do agente com o ambiente. A aprendizagem ocorre através da percepção:

(i) do estado do indivíduo no ambiente;

(ii) das ações realizadas nesse ambiente;

(iii) das mudanças de estado resultantes dessas ações; e

(iv) da recompensa que o ambiente retorna em resposta à ação executada.

Através da aprendizagem, o agente usa o valor de reforço na tomada de decisão subsequente.

Um sistema de Aprendizado por Reforço (RL) inclui três aspectos básicos:

(i) percepção;

(ii) ação; e

(iii) meta.

Nesse sistema, como mostrado na Figura 2.2, o agente percebe (parcialmente) o estado do ambiente e, com base no conhecimento obtido por meio dessa percepção, seleciona uma ação a ser executada. A ação executada afeta o ambiente, alterando o estado no qual o agente está. Todo agente dentro de um sistema de RL possui um estado meta que deve ser alcançado. O objetivo principal é levar o agente a selecionar uma sequência de ações até o estado meta, o que maximiza o reforço acumulado ao longo do tempo. Assim, é gerada uma política de controle/decisão, caracterizada pelo mapeamento de estados e ações, representando o comportamento que o sistema de RL segue até atingir o objetivo.

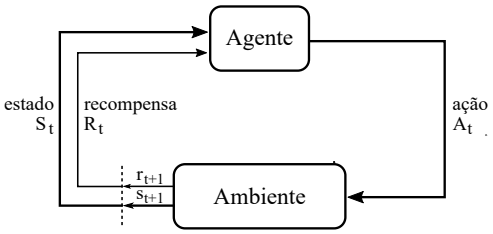


Figura 2.2: Interação agente-ambiente no aprendizado por reforço (Sutton e Barto, 1998).

Os principais elementos que compõem a formulação do Problema de Aprendizagem por Reforço são:

(i) Conjunto de estados: conjunto de todos os estados possíveis que descrevem o ambiente;

(ii) Conjunto de Ações: conjunto de todas as ações disponíveis;

(iii) Ambiente: o ambiente no problema de RL é dinâmico e deve ser pelo menos parcialmente observável;

(iv) Política de Controle/Decisão: define o comportamento do agente para atingir a meta. Uma política de controle mapeia o estado s em ações a e é expressa pela função $\Pi(s, a)$. Esta função define a probabilidade de que uma ação a seja escolhida em um estado s. Essas probabilidades mudam à medida que o agente acumula experiências como consequência das interações com o ambiente. Assim, a convergência para a política ótima Π^* expressa o processo de aprendizagem no problema da RL;

(v) Reforço/Recompensa: mostra o feedback do ambiente em relação ao comportamento do agente. O objetivo é maximizar esse feedback recebido do ambiente. Para atingir esse objetivo, o agente deve considerar o comportamento futuro na tomada de decisão no momento presente. Existem vários modelos que definem como o agente deve acumular as recompensas recebidas. O mais usado é o modelo com desconto de horizonte finito (em inglês, *finite-horizon discounted model*) (Kaelbling et al., 1996). Neste modelo, o sistema RL procura maximizar a recompensa esperada em função da sequência de valores recebidos até um instante de tempo T, na forma:

$$R_T = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \tag{2.1}$$

Considerando as recompensas recebidas a longo prazo, um fator de desconto γ é aplicado à expressão de reforço. Como consequência:

$$R_T = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + r_T = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

em que $0 \leq \gamma \leq 1$. Assim, se $\gamma = 0$, reforços imediatos são maximizados; se $\gamma = 1$, a mesma importância é dada aos ganhos imediatos e futuros;

(vi) Função Reforço (Função de Recompensa): as funções de reforço nem sempre são simples de definir e variam de acordo com o problema abordado;

(vii) Valor Função: valor obtido com o mapeamento do estado ou do par ação-estado, das recompensas atuais e futuras:

(a) Função Valor-Estado: função que considera apenas o estado e é denotada por $V^{\Pi}(s)$. A função valor-estado depende da política Π e é definida por:

$$V^{\Pi}(s) = E_{\Pi}\{R_t \mid s_t = s\} = E_{\Pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{2.3}$$

(b) Função Valor-Ação: função que considera o par estado-ação e é denotada por $Q^{\Pi}(s, a)$. Como na função valor-estado, a função valor-ação depende da política Π e é definida por:

$$Q^{\Pi}(s, a) = E_{\Pi}\{R_t \mid s_t = s, a_t = a\} = E_{\Pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \tag{2.4}$$

As funções valor-estado e valor-ação podem ser modeladas pelo Processo de Decisão de Markov (em inglês, *Markov Decision Process – MDP*) (Bellman, 1957; Bertsekas, 1987; Puterman, 1994). Uma boa revisão bibliográfica da Aprendizagem por Reforço é conduzida por Kaelbling et al. (1996).

2.4.2 Algoritmo *Q-Learning*

O algoritmo *Q-Learning*, introduzido por Watkins e Dayan (1992), destaca-se por ser amplamente utilizado, por ser livre de modelos e por dispensar o conhecimento de uma política. Desta forma, o agente, através do algoritmo *Q-learning*, atualiza seu valor de função, enquanto segue qualquer política. Este algoritmo permite encontrar uma política ótima de seleção de ações para qualquer Processo de Decisão de Markov finito (MDP). O objetivo é, a cada passo de um episódio, maximizar o valor da função $Q(s, a)$, definida como:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{\alpha'} Q(s', a') - Q(s, a)\right] \tag{2.5}$$

como mostrado no pseudocódigo apresentado no Algoritmo 1, em que α é a taxa de aprendizado; γ é o fator de desconto; s é o estado atual; a é a ação tomada; e s' é o estado resultante. 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 a 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*

procedure QLEARNING(r, α, ϵ, γ)
  Initialize $Q(s, a)$ arbitrarily;
  repeat ▷ for each episode
    Initialize s;
    repeat ▷ for each step of episode
      Choose a from s using policy derived from Q (e.g., ϵ-greedy);
      Take action a;
      Observe the next state s' and the reward r;
      $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{\alpha'} Q(s', a') - Q(s,a) \right]$;
      $s \leftarrow s'$;
    until s is terminal
  until Reaches the number of episodes
end procedure

2.4.3 Autômatos de Aprendizagem

O termo Autômatos de Aprendizagem (em inglês, *Learning Automata*) é apresentado inicialmente em Narendra e Thathachar (1974). Assim como no aprendizado por reforço, os autômatos de aprendizagem atuam em ambientes nos quais poucas informações são conhecidas *a priori*. Neste conceito, um vetor de probabilidades é associado às ações disponíveis e será usado para selecionar a próxima ação a ser tomada.

Sendo assim, um autômato de aprendizagem, como descrito por Narendra e Thathachar (1974), funciona da seguinte forma: inicialmente, a ação ótima não é conhecida e as probabilidades associadas a cada ação são iguais; uma ação é selecionada aleatoriamente; em seguida, é observada a resposta do ambiente a esta ação; as probabilidades são alteradas com base na resposta do ambiente; a nova ação deve ser selecionada considerando as probabilidades atualizadas; e este procedimento é repetido.

A interação do autômato de aprendizagem com o ambiente é apresentada na Figura 2.3. A entrada do ambiente é formada pelas ações do autômato e a resposta do ambiente será a entrada do autômato e influenciará na atualização do vetor de probabilidades.

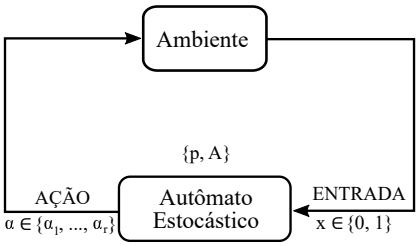


Figura 2.3: Interação agente-ambiente em Autômatos de Aprendizagem (Narendra e Thathachar, 1974).

Um autômato de aprendizagem é uma sêxtupla :

$$\{x, \phi, \alpha, p, A, G\} \tag{2.6}$$

em que:

(i) x: conjunto de entradas;

(ii) ϕ: conjunto de estados internos, de dimensão s;

(iii) α: conjunto de saídas ou conjunto de ações, de dimensão r;

(iv) p: vetor de probabilidades dos estados, que coordena a escolha do próximo estado a cada estágio;

(v) A: algoritmo que define o esquema de atualização, ou esquema de reforço, para atualizar o vetor de probabilidades com base no conjunto de entrada;

(vi) $G : \phi \rightarrow \alpha$: função de saída.

Deve-se salientar que, em Narendra e Thathachar (1974), os estados e ações são considerados sinônimos; desta forma, $s = r$. Neste contexto, somente um ambiente com característica de respostas aleatórias é de interesse nos problemas considerados.

Segundo Narendra e Thathachar (1974):

> *um autômato de aprendizagem é um autômato estocástico que opera em um ambiente aleatório e atualiza suas probabilidade de ação de acordo com as entradas recebidas do ambiente para melhorar seu desempenho.*

A apresentação detalhada de Autômatos de aprendizagem é realizada em Narendra e Thathachar (1974).

2.5 Considerações Parciais

O objetivo principal deste Capítulo foi apresentar conceitos essenciais para o desenvolvimento desta tese. A Seção 2.3 buscou distinguir as características desejáveis de um *framework* para a otimização usando metaheurísticas e identificar as lacunas existentes, especialmente aquelas relacionadas à hibridização de metaheurísticas, buscando embasar o *framework* proposto neste trabalho. Portanto, um esforço para investigar o estado da arte relacionado a essas ferramentas foi realizado. Inicialmente, foi apresentada uma visão geral da hibridização de metaheurísticas, com foco especial em metaheurísticas cooperativas e metaheurísticas paralelas. A combinação de cooperação e paralelismo, amplamente utilizada no contexto de otimização, também foi analisada. Foram apresentadas estruturas de propósito geral para otimização, denominadas *Frameworks*, que visam auxiliar os pesquisadores no desenvolvimento de metaheurísticas e minimizar o esforço computacional despendido. Adicionalmente, foi descrito um conjunto específico de trabalhos que utilizam os conceitos de Sistemas Multiagentes na organização das relações entre os métodos implementados, também vistos como *frameworks*.

Visando alcançar o objetivo proposto, foi realizada uma comparação entre os *frameworks* apresentados. Para tanto, os *frameworks* selecionados foram analisados utilizando cinco critérios pré-definidos. A comparação foi feita a partir de um conjunto de vinte e duas características, as quais foram definidas com base nos trabalhos de Talbi (2002), Raidl (2006) e Silva (2007) e Parejo et al. (2012). Essas características foram divididas em quatro categorias: (i) geral; (ii) avançado; (ii) multiagente; (iv) e suporte ao processo de otimização.

Entre os vários trabalhos listados neste capítulo, observa-se que a maioria dos *frameworks* possui características e propostas semelhantes.

A análise comparativa apresentada aqui avaliou todos os *frameworks* considerados em Parejo et al. (2012) e os que surgiram após a publicação deste artigo. Esta análise mostrou que a hibridização ainda não é uma característica forte dos *frameworks* existentes, como afirmado por Parejo et al. (2012). Assim, é de suma importância concluir que os *frameworks* permanecem sem apresentar, como uma propriedade, a facilidade e a flexibilidade para o desenvolvimento de metaheurísticas híbridas. Por outro lado, a hibridização de metaheurísticas mostrou-se uma metodologia muito importante para resolver problemas de otimização, uma vez que, como destacado em Blum et al. (2011), os melhores resultados obtidos na solução de problemas de otimização combinatória resultam de hibridizações.

No entanto, o desenvolvimento de *frameworks* não avançou nessa direção. O desenvolvimento de estruturas gerais que permitem ao usuário usar uma técnica de otimização de cada vez é visivelmente mais simples do que oferecer a possibilidade de combinar essas técnicas, possibilitando o surgimento de técnicas híbridas. Por outro lado, considerando os bons resultados obtidos com metaheurísticas híbridas, podemos dizer que há necessidade de ferramentas que ofereçam a possibilidade de combinar métodos para resolver problemas de otimização. Nesse sentido, os *frameworks* que utilizam as abordagens multiagentes são, como mostra a análise apresentada, uma boa opção no processo de hibridização, permitindo o desenvolvimento de estruturas genéricas, que possibilitam a interação entre metaheurísticas, independentemente de quais delas são utilizadas, e até mesmo independente do problema a ser tratado. Essas abordagens propõem formas de interação que facilitam a hibridização, permitindo, com considerável facilidade, o uso de técnicas cooperativas e, na grande maioria, recursos computacionais paralelos.

A comparação confirmou essa afirmação, mostrando que dentre as propostas que efetivamente possibilitam métodos de hibridização, destacam-se *frameworks* que utilizam o conceito de sistemas multiagentes em sua composição. A cooperação e a autonomia de ação dos agentes, presentes em propostas que utilizam abordagens multiagentes, flexibilizam a implementação de métodos híbridos. Além disso, o conceito de sistemas multiagentes permite que diferentes regiões do espaço de busca sejam operadas simultaneamente pelos agentes.

Neste contexto, é importante ressaltar a clara diferenças entre o grupo de *frameworks* de otimização baseados na abordagem multiagente (Seção 2.2.2) e o grupo de *frameworks* de otimização que não usam este conceito (Seção 2.2.1). Os *frameworks* que não utilizam o conceito multiagente apresentam um bom número de recursos avançados e recursos de suporte ao processo de otimização; por outro lado, os *frameworks* baseados na abordagem multiagente possuem características que beneficiam a hibridização das metaheurísticas.

O impacto de novas tecnologias de suporte a computação paralela e distribuída aponta para novas possibilidades de pesquisa que ainda precisam ser exploradas pelos desenvolvedores de *frameworks*. As estruturas `ParadisEO` e `AgE` se destacam das outras discutidas aqui porque apresentam implementações que facilitam o desenvolvimento distribuído e permitem o uso de estruturas de múltiplos e vários núcleos. Além disso, é necessário explorar novas linguagens de programação que facilitem o desenvolvimento usando esses novos recursos. De acordo com Krzywicki et al. (2015); Turek et al. (2016), linguagens funcionais, como `Erlang` e `Scala`, são destacadas a esse respeito.

Hiperheurística é outro conceito muito pouco explorado pelos *frameworks* analisados. Além da estrutura `Hyflex`, especializada em hiperheurísticas, apenas a estrutura `SMA`, que usa algoritmos baseados em inteligência coletiva para coordenar metaheurísticas, imple-

menta hiperheurísticas. No entanto, as hiperheurísticas podem restringir a autonomia de decisão de sistemas multiagentes. Assim, não parece aconselhável usar essas estruturas de coordenação em conjunto com sistemas multiagentes.

Entre os recursos comuns em sistemas multiagentes, o aprendizado não é implementado nem mesmo por estruturas que usam essa abordagem em seu projeto. Esse recurso pode ajudar no processo de busca, permitindo que o agente se adapte e possivelmente melhore sua capacidade de ação, sem, no entanto, restringir sua autonomia, mas sim, adicionar informações ao agente para ajudá-lo na tomada de decisão ao resolver um problema de otimização. Outra importante questão em aberto é o uso de sistemas multiagentes para resolver problemas de otimização multiobjetivo.

Além disso, faltam ferramentas para o suporte ao processo de otimização, como análise estatística, auto-ajuste de parâmetros e interfaces gráficas, características já avaliadas no comparativo apresentado.

A Seção 2.4 apresenta os conceitos usados no desenvolvimento dos agentes do *framework* AMAM. O objetivo é utilizar os conceitos de aprendizado de máquina no desenvolvimento de habilidades que permitam que o agente se adapte a características específicas do problema. Para tal, são detalhadas duas técnicas de Aprendizado por Reforço: algoritmo *Q-Learning* e Autômatos de Aprendizagem. Estas técnicas são utilizadas no desenvolvimento dos agentes adaptativos.

Capítulo 3

Estudos de Caso

Este Capítulo aborda dois problemas de grande importância na Otimização Combinatória e que são utilizados, nesta tese, na instanciação do framework AMAM. Os problemas considerados aqui para fins de estudo de caso são o Problema de Roteamento de Veículos com Janelas de Tempo (*Vehicle Routing Problem with Time-Windows* – VRPTW) e o Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempos de Configuração Dependentes da Sequência (*Unrelated Parallel Machine Scheduling Problem with sequence-dependent Setup-Times* – UPMSP-ST).

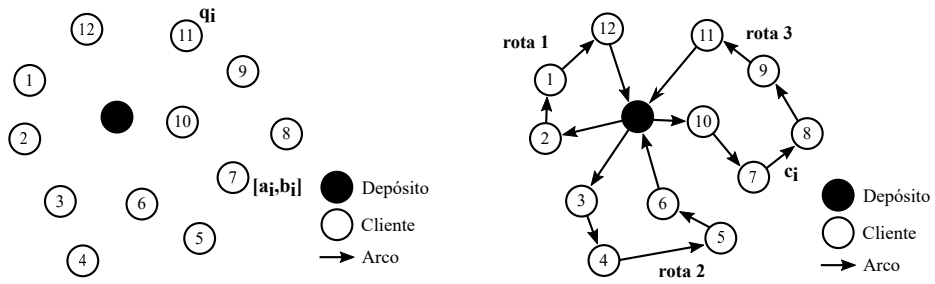
O VRPTW é um dos problemas de otimização combinatória mais estudados. Trata-se de uma generalização do clássico Problema do Caixeiro Viajante (TSP) (Applegate et al., 2007), no qual há uma frota de veículos, um conjunto de clientes, dispersos geograficamente, a serem atendidos, com suas respectivas demandas e seus horizontes de tempo para o atendimento. A descrição completa deste problema está apresentada na Seção 3.1, sob os termos de interesse deste trabalho. Para mais detalhes sobre o VRPTW, excelentes revisões de pesquisa envolvendo este problema podem ser encontradas, por exemplo, em Toth e Vigo (2002) e Toth e Vigo (2014).

O UPMSP-ST também tem forte relevância econômica em vários tipos de indústrias. Assim como o VRPTW, o UPMSP-ST pertence à classe de problemas NP-difíceis, o que justifica o uso de métodos metaheurísticos na solução desses problemas. Uma descrição detalhada do UPMSP-ST pode ser encontrada em Rabadi et al. (2006) e Vallada e Ruiz (2011), e boas referências relacionadas a este problema são Allahverdi et al. (2008) e Allahverdi (2015). A Seção 3.2 apresenta este problema, nos termos que interessam a este trabalho.

3.1 Caso 1: VRPTW

3.1.1 Definições Básicas

Neste problema, um conjunto $K = \{k \ : \ k = 1, 2, \ldots, |K|\}$ de veículos está localizado em um único depósito e deve servir a um conjunto $C = \{i \ : \ i = 1, 2, \ldots, N\}$ de clientes espalhados geograficamente. No caso aqui considerado, a frota de veículos é homogênea, ou seja, todos os veículos são iguais e possuem a mesma capacidade Q. Cada cliente i tem uma dada demanda q_i e deve ser atendido dentro de uma janela de tempo especificada $[a_i, b_i]$ (veja a Figura 3.1(a)). Uma solução para o VRPTW é um conjunto de rotas (veja a Figura 3.1(b)), em que cada rota é representada por uma lista ordenada de clientes, que determina a sequência na qual eles devem ser atendidos por um veículo. Os arcos mostram



(a) Grafo representando os elementos do VRPTW.

(b) Representação da solução do VRPTW.

Figura 3.1: Uma solução do VRPTW.

a conexão entre os clientes e têm um valor associado c_{ij}, que representa o custo de viagem entre o cliente i e o cliente j. A solução x mostrada na Figura 3.1(b) pode ser descrita como:

$$x = [0, 2, 1, 12, 0, 3, 4, 5, 6, 0, 10, 7, 8, 9, 11, 0]$$

na qual o índice 0 indica o depósito e $rota_1 = [0, 2, 1, 12, 0]$, $rota_2 = [0, 3, 4, 5, 6, 0]$ e $rota_3 = [0, 10, 7, 8, 9, 11, 0]$ são as três rotas desta solução. Assim, a solução x também pode ser descrita como $x = [rota_1, rota_2, rota_3]$.

O objetivo do VRPTW é determinar um conjunto de rotas para minimizar o custo total envolvido com essa operação. Cada rota está associada a um único veículo. As rotas devem começar e terminar no depósito. Neste caso, o custo de uma solução x é calculado de acordo com:

$$f(x) = \omega K(x) + \sum_{(i,j)\in E} c_{ij} \tag{3.1}$$

na qual:

- c_{ij} : custo entre os clientes (i, j), que pode ser relacionado à distância entre clientes;
- E : conjunto de arcos pertencentes à solução x;
- $K(x)$: número de veículos na solução x;
- ω : um fator de penalidade de alto valor, não negativo e arbitrário.

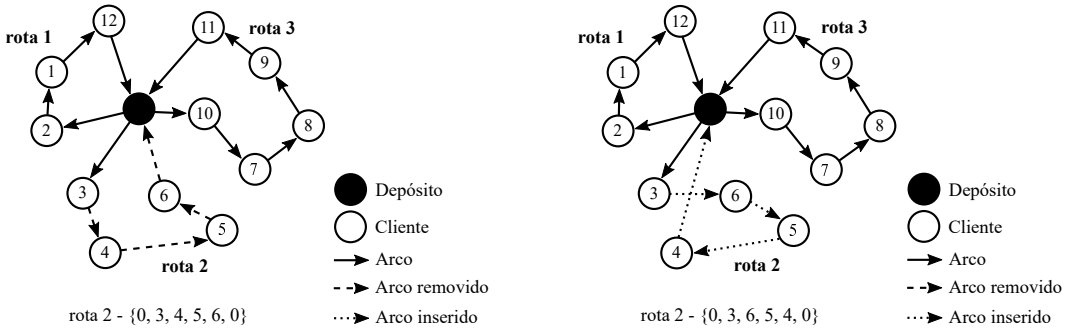
Nesta função, a prioridade é minimizar o número de veículos (ou rotas, consequentemente). Em caso de empate no número de veículos, a distância total percorrida deve ser minimizada.

3.1.2 Vizinhanças do VRPTW

Uma vizinhança é uma função $\mathcal{N}(x)$ que descreve um subconjunto de soluções associado à solução x pertencente ao espaço de soluções do problema. Cada solução desse subconjunto é chamada de vizinho. A função $\mathcal{N}(x)$ é definida como um operador que recebe uma solução x^1 e a transforma em outra solução x^2, pertencente à vizinhança de x^1 (Milano e Roli, 2004).

Para explorar o espaço de soluções, oito diferentes funções de vizinhança são usadas na instanciação do framework AMAM para resolver o VRPTW. O conhecimento dessas funções de vizinhança torna-se necessário por serem elementos chaves no modelo de aprendizado que foi descrito na Seção 4.4.1. Estas estruturas são apresentadas a seguir:

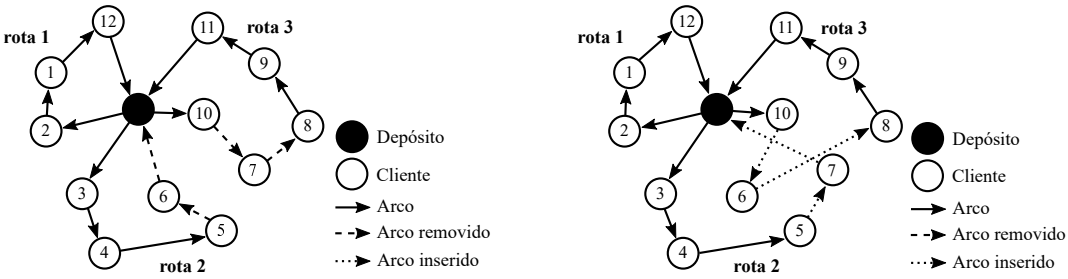
(i) *Intra-Route Swap* (Troca Intra-Rota): função de vizinhança que realiza o movimento de troca de um cliente com outro cliente da mesma rota. A Figura 3.2 ilustra essa função de vizinhança. Neste exemplo, os clientes 4 e 6 da rota 2 têm suas posições na rota trocadas;



(a) Troca dos clientes 4 e 6. (b) Solucão resultante.

Figura 3.2: Aplicação da função de vizinhança *Intra-Route Swap* em uma rota da solução.

(ii) *Inter-Route Swap* (Troca Inter-Rota): função de vizinhança que realiza o movimento de troca de um cliente de uma rota com um cliente de outra rota. A Figura 3.3 mostra a função de vizinhança *Inter-Route Swap*. Nesta figura, o cliente 6 é removido da rota 2 e inserido na rota 3 no lugar do cliente 7, que é consequentemente transferido para a rota 2 no lugar anteriormente ocupado pelo cliente 6;

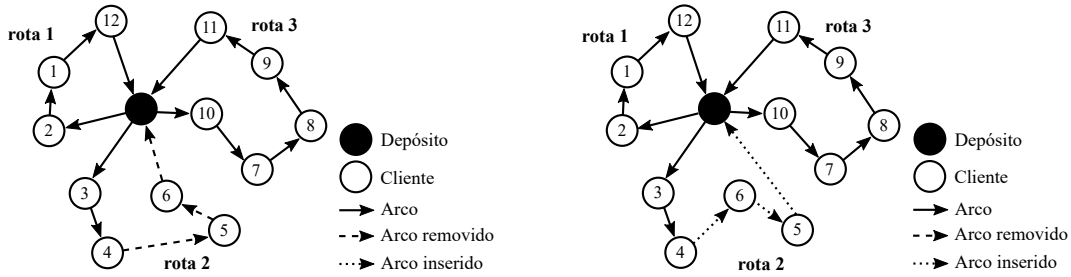


(a) Troca do cliente 6 da rota 2 com o cliente 7 da rota 3. (b) Solução resultante.

Figura 3.3: Uma aplicação da função de vizinhança *Inter-Route Swap* na solução.

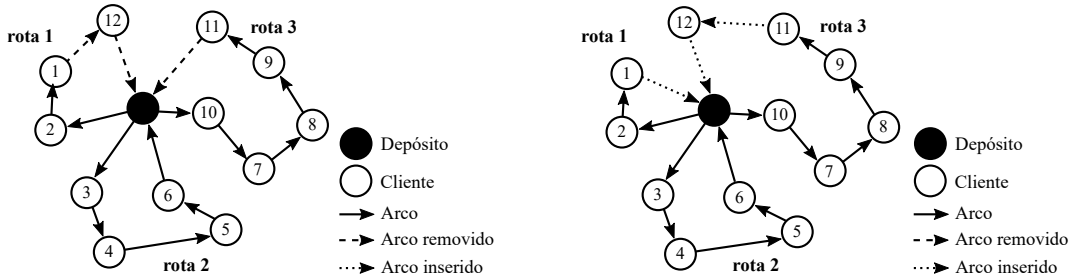
(iii) *Intra-Route Shift* (Realocação Intra-Rota): função de vizinhança que realiza o movimento de realocação de um cliente para outra posição na mesma rota. A Figura 3.4 mostra a aplicação da função *Intra-Route Shift*, na qual o cliente 6 da rota 2 é removido da sua posição e inserido entre os clientes 4 e 5 da mesma rota;

(iv) Inter-Route Shift (Realocação Inter-Rota): função de vizinhança que realiza a realocação de um cliente de uma rota para outra. A função vizinhança *Inter-Route Shift* é mostrada na Figura 3.5, na qual o cliente 12 é retirado da rota 1 e depois inserido na rota 3;



(a) Realocando o cliente 6 na mesma rota. (b) Solução Resultante.

Figura 3.4: Uma aplicação da função de vizinhança *Intra-Route Shift* na rota 2 da solução.



(a) Cliente 12 da rota 1 é realocado na rota 3. (b) Solução resultante.

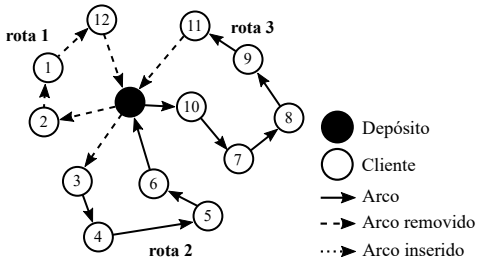
Figura 3.5: Uma aplicação da função de vizinhança *Inter-Route Shift* na solução.

(v) *Two Intra-Route Swap* (Duas Trocas Intra-Rota): função de vizinhança que consiste na troca de clientes na mesma rota, assim como na função de vizinhança *Intra-Route Swap*. No entanto, na função *Two Intra-Route Swap*, dois clientes consecutivos são trocados por dois outros clientes consecutivos da mesma rota;

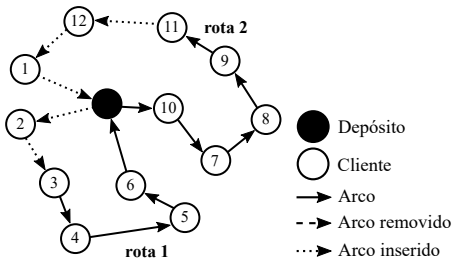
(vi) *Two Intra-Route Shift* (Duas Realocações Intra-Rota): função de vizinhança que consiste na realocação de clientes na mesma rota, assim como na função de vizinhança *Intra-Route Shift*. No entanto, na função *Two Intra-Route Shift*, dois clientes consecutivos são removidos de suas posições e reinseridos em outra posição da mesma rota;

(vii) *Eliminates Smaller Route* (Elimina a Menor Rota): função de vizinhança que procura eliminar a menor rota da solução. A menor rota é definida como a rota que possui o menor número de clientes. Para isso, os clientes da menor rota da solução são removidos e reinseridos em outras rotas da solução. A rota e a posição de inserção de cada cliente removido são aquelas que resultam no melhor valor da função objetivo, respeitando-se todas as restrições. Uma nova solução é gerada quando todos os clientes da menor rota são reinseridos em outras rotas. A Figura 3.6 mostra um exemplo da função de vizinhança *Eliminates Smaller Route*. Neste exemplo, a menor rota (rota 1) é excluída e seus clientes são inseridos em outras rotas (rotas de 2 e 3);

(viii) *Eliminates Random Route* (Elimina Rota Aleatória): A função de vizinhança *Eliminates Random Route* opera de forma semelhante à função *Eliminates Smaller Route*,



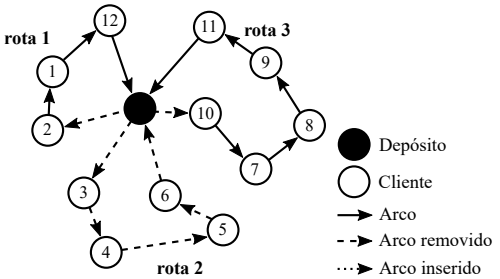
(a) Eliminando a rota 1 e inserindo os clientes em outras rotas.



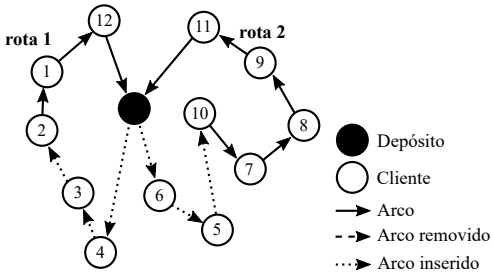
(b) Solução resultante.

Figura 3.6: Aplicação da função de vizinhança *Eliminates Smaller Route* na solução.

mas a rota a ser excluída é escolhida aleatoriamente.



(a) Eliminando a rota 2 e inserindo os clientes em outras rotas.



(b) Solução Resultante.

Figura 3.7: Aplicação da função de vizinhança *Eliminates Random Route* na solução.

3.2 Caso 2: UPMSP-ST

3.2.1 Definições Básicas

Neste problema, um conjunto $N = \{n : n = 1, 2, \ldots, |N|\}$ de tarefas deve ser alocado a um conjunto $M = \{m : m = 1, \ldots, |M|\}$ de máquinas, sendo que cada tarefa $j \in N$ deve ser alocada a uma única máquina $i \in M$. Na versão do UPMSP-ST considerada aqui, é definido o tempo de processamento p_{ij}, que representa o tempo requerido para processar a tarefa $j \in N$ na máquina $i \in M$; e o tempo de configuração S_{ijk}, que representa o tempo requerido para configurar a tarefa $k \in N$ após a tarefa $j \in N$ na máquina $i \in M$. As Tabelas 3.1 e 3.2 apresentam um exemplo, com dados referentes a uma instância de teste para calibração de experimentos com 8 tarefas e 2 máquinas, proposta por Vallada e Ruiz (2011). A Tabela 3.1 exibe os tempos de processamento de cada uma das 8 tarefas nas máquinas m_1 e m_2. A Tabela 3.2 mostra os tempos de configuração para as máquinas m_1 e m_2. Como o problema é dependente da sequência, uma matriz é usada para indicar o tempo de configuração da tarefa i após a execução da tarefa j.

Uma solução para o UPMSP-ST é uma lista de máquinas, em que cada máquina é representada por uma lista ordenada de tarefas, que define a sequência em que serão

Tabela 3.1: Tempos de processamento (*processing time*) de cada uma das 8 tarefas nas máquinas m_1 e m_2.

	1	2	3	4	5	6	7	8
m_1	67	3	29	85	11	36	25	12
m_2	43	40	29	26	46	49	7	5

Tabela 3.2: Tempos de configuração (*setup time*) para as máquinas m_1 e m_2.

(a) Máquina m_1.

m_1	1	2	3	4	5	6	7	8
1	0	5	7	8	8	9	8	5
2	7	0	4	6	6	7	6	8
3	5	2	0	6	6	3	6	7
4	6	9	9	0	6	2	2	8
5	4	8	6	5	0	6	8	2
6	4	2	2	2	2	0	3	8
7	7	1	7	2	9	4	0	9
8	3	8	4	3	8	7	2	0

(b) Máquina m_2.

m_2	1	2	3	4	5	6	7	8
1	0	2	3	1	2	4	9	5
2	4	0	5	3	1	8	9	7
3	1	3	0	5	5	2	8	7
4	4	2	5	0	3	6	4	5
5	7	6	5	6	0	6	5	2
6	3	6	3	9	4	0	1	6
7	4	1	6	7	2	7	0	9
8	4	9	6	3	6	8	8	0

realizadas. A Figura 3.8 ilustra uma solução do UPMSP-ST, na qual cada tarefa atribuída a sua respectiva máquina tem seu tempo de processamento representado na linha de tempo e os intervalos entre as tarefas representam o tempo de configuração. Os valores utilizados neste exemplo são baseados na instância das Tabelas 3.1 e 3.2.

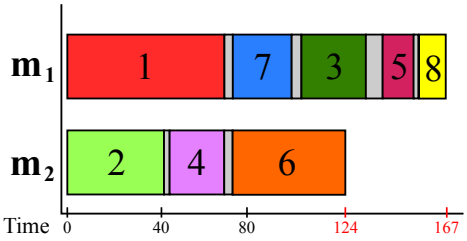


Figura 3.8: Exemplo de solução para o UPMSP-ST.

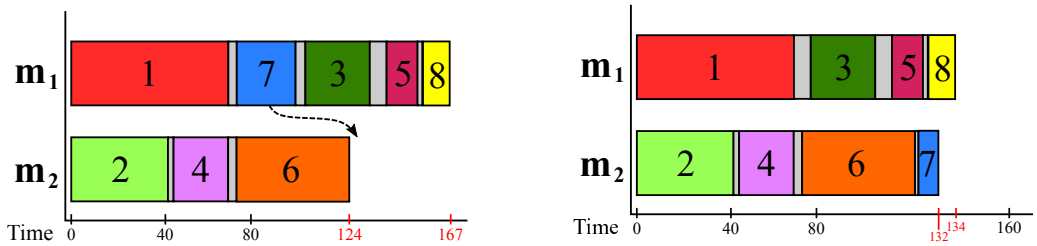
O objetivo do UPMSP-ST é alocar todas as n tarefas às m máquinas, com a finalidade de minimizar o tempo máximo de conclusão do sequenciamento, conhecido como *makespan*. Na Figura 3.8, o *makespan* é definido pelo tempo de conclusão da máquina M_1 e é dado pelo valor 167.

3.2.2 Vizinhanças do UPMSP-ST

Quatro funções de vizinhança foram utilizadas para explorar o espaço de soluções do UPMSP-ST na instanciação do framework. Assim como no VRPTW, essas funções de vizinhança compõem o conjunto de estados definido no aprendizado do agente. Estas estruturas são apresentadas a seguir:

(i) *Insertion in Different Machines* - IDM (Inserção em Diferentes Máquinas): função de vizinhança que realiza a realocação de uma tarefa de uma máquina para outra.

A Figura 3.9 mostra a aplicação desta função de vizinhança, na qual a tarefa 7 é removida da máquina m_1 e inserida na máquina m_2. Como é possível observar, o *makespan* da solução inicial, mostrado na Figura 3.9(a), é obtido pela máquina m_1, no valor de 167; após a aplicação da função de vizinhança, o novo *makespan* (solução da Figura 3.9(b)) é obtido também pela máquina m_1, no valor de 134.

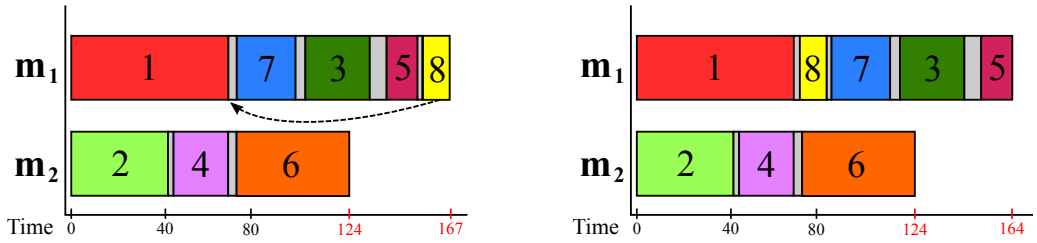


(a) Solução Inicial.

(b) Solução após a aplicação da função de vizinhança MIDM.

Figura 3.9: Exemplo: Vizinhança *Insertion in Different Machines*.

(ii) *Insertion in the Same Machines* - ISM (Inserção na mesma Máquina): função de vizinhança que realiza o movimento de realocação de uma tarefa para outra posição da mesma máquina. A Figura 3.10 mostra a aplicação desta função de vizinhança, na qual a tarefa 8 é removida da última posição da máquina m_1 e é inserida na segunda posição desta mesma máquina. Esta modificação reduz o *makespan*. De fato, antes do movimento, o *makespan* era 167 (Figura 3.10(a)) e, após o movimento, o *makespan* reduziu-se para 164 (Figura 3.10(b)).

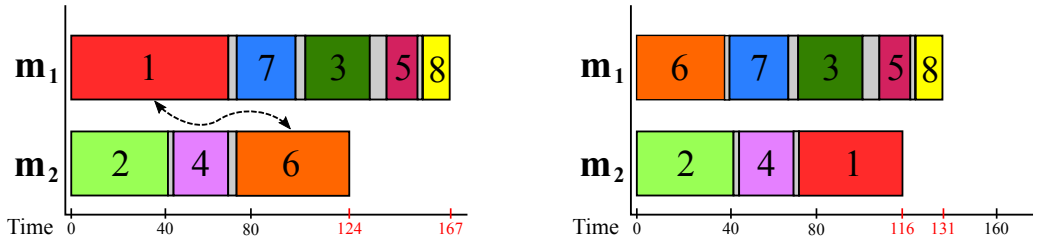


(a) Solução Inicial.

(b) Solução após a aplicação da função de vizinhança MISM.

Figura 3.10: Exemplo: vizinhança *Insertion in the Same Machines*.

(iii) *Swap between different machines* - SDM (Troca entre Diferentes Máquinas): função de vizinhança que realiza o movimento de troca de uma tarefa de uma máquina com uma tarefa de outra máquina. Essa função de vizinhança é mostrada na Figura 3.11, na qual a tarefa 1 da máquina m_1 é trocada com a tarefa 6 da máquina m_2. Neste caso, a aplicação do movimento leva a uma redução significativa do *makespan* da solução, já que, na solução inicial, mostrada na Figura 3.11(a), o valor do *makespan* é 167 e, na solução resultante (Figura 3.11(b)), o valor obtido é 131.

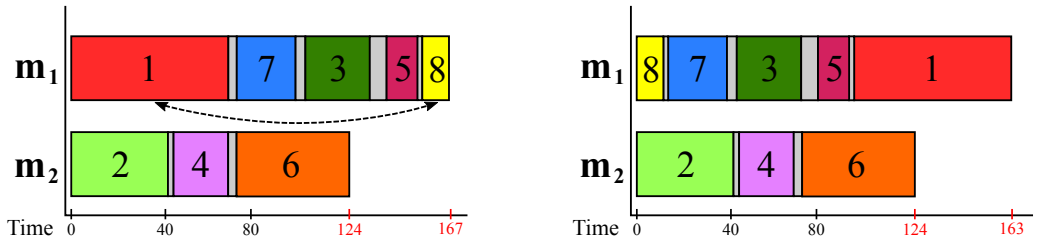


(a) Solução Inicial.

(b) Solução após a aplicação da função de vizinhança SDM.

Figura 3.11: Exemplo: Vizinhança *Swap between different machines*.

(iv) *Swap between Same Machine* - SSM (Troca na mesma máquina): função de vizinhança que realiza o movimento de troca de uma tarefa com outra tarefa da mesma máquina. Essa função de vizinhança é mostrada na Figura 3.12, na qual a tarefa 1 da máquina m_1 é trocada pela tarefa 8 desta mesma máquina. A solução inicial é mostrada na Figura 3.12(a), com o *makespan* 167. A solução obtida aplicando esse movimento é mostrada na Figura 3.12(b), com o *makespan* resultante avaliado em 163.



(a) Solução Inicial.

(b) Solução após a aplicação da função de vizinhança SSM.

Figura 3.12: Exemplo: Vizinhança *Swap between Same Machines*.

Capítulo 4

Framework AMAM

Este capítulo descreve detalhadamente o *framework* AMAM, apresentando o seu arcabouço, funcionamento e recursos avançados. O capítulo está organizado como segue. A Seção 4.1 faz uma apresentação geral do *framework* AMAM. A Seção 4.2 descreve a arquitetura e o funcionamento do *framework*. Sua estrutura é dividida em três dimensões, sendo elas: dimensão de ambiente, dimensão de Agente e dimensão Social. Estas três dimensões são apresentadas, respectivamente nas Seções 4.3, 4.4 e 4.5. A Seção 4.6 mostra como é realizado um experimento no *framework* através do pacote *Experiment*.

4.1 Apresentação geral do AMAM

Esta seção tem como objetivo detalhar o *framework* AMAM – *Arquitetura MultiAgente para Metaheurística*. O *framework* AMAM é uma arquitetura para metaheurísticas, baseada nos conceitos de sistemas multiagente, em que cada agente encapsula uma heurística/metaheurística e tem a função de buscar a solução para um dado problema de Otimização Combinatória. O ambiente do *framework*, no qual o agente existe e age, corresponde ao espaço de soluções do problema de otimização a ser resolvido. A capacidade de movimentação do agente através do espaço de soluções do problema é definida pelas formas de manipulação da solução, como, por exemplo, estruturas de vizinhança e operadores genéticos.

O *framework* é escalável, permitindo a fácil adição de novos agentes, com impacto mínimo no restante da arquitetura. Esses agentes interagem com o meio ambiente e com outros agentes cooperativamente, trocando e compartilhando informações sobre sua condição e sobre o meio ambiente. O agente possui também capacidades adaptativas que permitem que se ajuste melhor às mudanças do ambiente.

O modelo conceitual do *framework* AMAM foi proposto inicialmente em Silva (2007). Nesta proposta, sua estrutura, apresentada na Figura 4.1, era definida com seis elementos principais:

(i) Ambiente: Silva (2007) descreve o Ambiente como um dos principais elementos do *framework* AMAM. O ambiente corresponde ao espaço de busca do problema a ser solucionado. É nele que são definidas as características específicas do problema, como, por exemplo, as formas de manipular uma solução. Silva (2007) define duas operações que permitem a modificação do espaço de busca, sendo elas movimento (modificação da solução que leva a outra solução) e incremento (adição, elemento a elemento, para construção da solução);

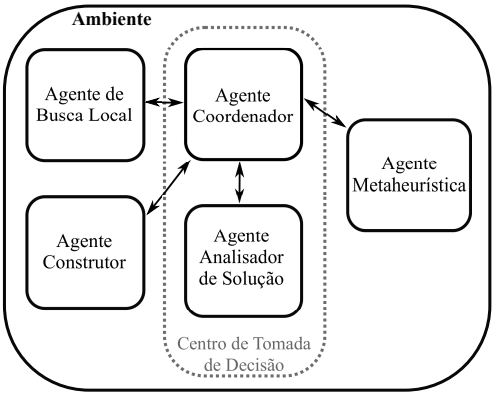


Figura 4.1: Modelo Conceitual do *Framework* AMAM proposto em Silva (2007).

(ii) Agente Construtor: é responsável pela construção de soluções iniciais. Para tal, faz uso das Heurísticas Construtivas;

(iii) Agente Busca Local: é responsável por refinar as soluções já encontradas. Para tal, faz uso das Heurísticas de Busca Local;

(iv) Agente Metaheurística: é responsável por explorar o espaço de busca e implementar estratégias que permitam que a busca escape de ótimo locais. Para tal, faz uso de algoritmos baseados em Metaheurísticas;

(v) Agente Coordenador: é responsável por coordenar as atividades e a comunicação entre os agentes no sistema multiagente;

(vi) Agente Analisador de Soluções: responsável pela análise e tomada de decisão com relação às soluções obtidas pelos Agentes Metaheurísticas.

Uma estrutura interna genérica é definida para os agentes de busca em Silva (2007). Dentro desta estrutura, a interface com o ambiente é realizada através de sensores e atuadores. Os sensores possibilitam ao agente perceber o ambiente, enquanto os atuadores proporcionam ao agente a capacidade de agir e modificar seu ambiente.

A comunicação e a cooperação entre os agentes, nesta versão inicial do *framework*, é gerida pelo Agente Coordenador. O Agente Coordenador centraliza as trocas de mensagens entre os agentes. Considerando que este agente conhece todos os agentes do sistema, ele deve, ao receber uma mensagem, enviar uma solicitação de serviço para que o agente responsável possa atendê-la. Uma outra forma de troca de informações, através do compartilhamento por meio do ambiente, também é realizada nesta versão. Neste caso, o sistema de comunicação quadro-negro, apresentado em Corkill (1991), é usado como referência. A instanciação do *framework* utilizada para teste em Silva (2007) usou o sistema de quadro-negro com um repositório com a mesma formulação da Lista Tabu. Desta forma, as soluções não recomendadas à exploração são armazenadas para serem evitadas pelos agentes de busca.

A tomada de decisão é, nesta versão inicial, controlada pelo Agente Analisador de Soluções. Neste caso, a análise das soluções, realizada por este agente, pode variar de acordo com a estratégia implementada pelo Agente Metaheurística, podendo ser, por exemplo, elitista ou probabilística. O objetivo aqui é, através desta estrutura de apoio à tomada de

decisão, permitir a utilização de diferentes formas de escolha da "melhor" solução, baseada em técnicas mais complexas, sem que grande esforço de implementação seja feito.

O modelo conceitual de Silva (2007) é utilizado no desenvolvimento de uma versão executável do *framework* AMAM, apresentada em de Oliveira (2008). Nesta versão, denominada AMAM 0.8, a estrutura geral proposta no modelo conceitual não é modificada. de Oliveira (2008) utiliza uma abordagem contemporânea para a definição da forma de comunicação, denominada genericamente de abordagem baseada na teoria da cognição situada (Clancey, 1997). Na cognição situada, a comunicação é realizada através de troca de estímulos. Desta forma, na versão AMAM 0.8, a interação entre agentes e ambiente é feita por meio do envio de estímulos. Neste caso, possíveis soluções são estímulos para os agentes.

Uma nova versão do *framework*, denominada AMAM 1.0, é apresentada em Fernandes (2009) e Fernandes et al. (2009). A principal contribuição apresentada nesta versão é a de uma estratégia de memória adaptativa compartilhada. De acordo com Fernandes (2009), o principal objetivo desta memória adaptativa é,

> *através do agente coordenador, armazenar, gerenciar e distribuir as informações de forma inteligente e eficiente, visando a interação, para solucionar problemas que estão além da capacidade ou conhecimento de cada agente metaheurística individualmente.*

Este modelo de cooperação se dá por meio de trocas assíncronas de informação. O mecanismo principal de cooperação é o *Pool de Soluções*. Este repositório de soluções armazena as informações que serão organizadas e distribuídas entre os agentes. O agente coordenador é, na versão AMAM 1.0, responsável por gerenciar o fluxo de informações no processo de cooperação. Desta forma, a comunicação entre os agentes é sempre intermediada pelo agente coordenador (veja a Figura 4.2). O *Pool* de Soluções é definido como um conjunto de elite de tamanho fixo, descartando sempre a pior solução, quando uma nova melhor solução é encontrada. O critério de seleção para as soluções do *Pool*, a serem enviadas aos agentes, se baseia no método de seleção da roleta.

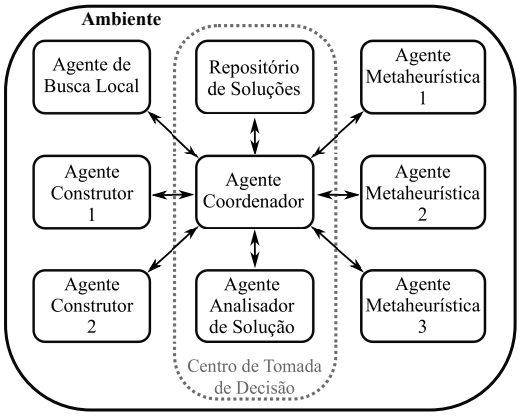


Figura 4.2: Estrutura geral da versão AMAM 1.0 (Fernandes, 2009).

Esta tese propõe uma revisão do *framework* AMAM, buscando retirar os elementos em excesso na estrutura existente, dinamizar e aperfeiçoar o processo de solução, já que, na área de Otimização Combinatória, o tempo gasto na solução do problema é um fator extremamente considerado. Da mesma forma, em se tratando de *framework*, a necessidade de

adaptação dos métodos aos problemas específicos leva à busca de estratégias que permitam o auto-ajuste do comportamento do método a cada problema apresentado.

A primeira alteração da estrutura geral, em relação à proposta inicial, é a remoção do Agente Coordenador e do Agente Analisador de Soluções, como apresentado em Silva et al. (2014). Ambos são responsáveis por intermediar a comunicação e tomar decisões dentro do *framework*; no entanto, eles limitam a autonomia de ação dos agentes no espaço de busca. Ao remover o Agente Coordenador, o principal objetivo é garantir esta autonomia: um agente não deve interferir diretamente nas decisões de outros agentes da estrutura. A cooperação é, então, realizada através do *Pool* de Soluções, no ambiente do sistema multiagente, de forma assíncrona. A cooperação, como definida nesta nova versão do *framework*, será descrita em detalhes na Seção 4.5.

O segundo ponto de mudança é marcado pela remoção de mais um elemento que intermediava a comunicação, qual seja, o *Pool* de Estímulos Ambientais. Nesta versão atual, as informações compartilhadas são enviadas diretamente ao *Pool* de Soluções (versão utilizada em Silva et al. (2015)). Todos os agentes têm acesso às soluções disponibilizadas no *Pool*. O *Pool* possui tamanho fixo e, sendo assim, é dotado de capacidade de controle do número de soluções e da diversidade destas. Os detalhes de implementação do *Pool* de Soluções e do compartilhamento de informações entre agentes também são apresentados na Seção 4.5.

Os agentes que implementam heurísticas e metaheurísticas (agente construtor, agente busca local e agente metaheurística), apresentados no modelo conceitual, são, na versão atual, identificados apenas como agentes. Todo agente do *framework* AMAM atua em uma *thread* independente e implementa um método de solução. Este método pode ser qualquer heurística ou metaheurística disponível no *framework* ou adicionada pelo usuário. Os agentes e sua capacidade de ação são apresentados na Seção 4.4.

Nesta proposta, a principal contribuição está na atribuição de capacidades auto-adaptativas aos agentes. A capacidade adaptativa permite ao agente modificar suas ações com base na experiência adquirida na interação com o ambiente e com os demais agentes. Duas propostas baseadas nos conceitos de aprendizagem de máquina são apresentadas na Seção 4.4.1.

Por fim, propomos a divisão da estrutura geral do *framework* em dimensões (Seção 4.2). Esta visão em dimensões permite a análise considerando as diferentes perspectivas: Problema de Otimização, Ambiente Multiagente, os Agentes e suas habilidades de busca e o Processo de Cooperação; e, por conseguinte, melhor entendimento da estrutura.

A arquitetura e o funcionamento da nova estrutura do *framework* AMAM é apresentada em detalhes nas seções seguintes.

4.2 Arquitetura e funcionamento

A Figura 4.3 apresenta a estrutura do *framework* AMAM. Esta estrutura é dividida em três dimensões, que são descritas de forma detalhada nas seções seguintes. São elas:

(i) Dimensão de Ambiente: dimensão que inclui o ambiente do sistema multiagente. O ambiente do sistema proposto corresponde ao espaço de busca do problema tratado, ou seja, ao conjunto de todas soluções viáveis. Sendo assim, todas as informações específicas relativas ao problema estão definidas nesta dimensão;

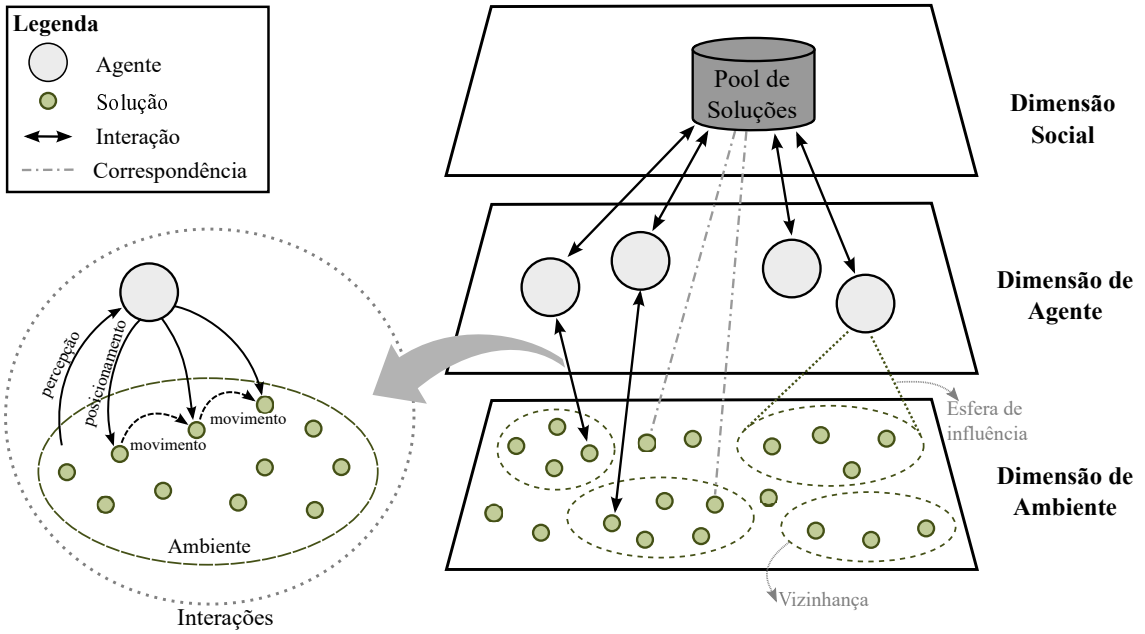


Figura 4.3: Dimensões da Arquitetura AMAM.

(ii) Dimensão de Agente: dimensão que envolve os agentes, suas habilidade e as formas de interação destes com o ambiente;

(iii) Dimensão Social: dimensão composta pelas formas de interação e cooperação entre agentes.

Todas as dimensões que definem o *framework* AMAM estão interligadas, e esta ligação é definida principalmente pelas formas de interação do agente com o ambiente e com os demais agentes dos sistema multiagente. O sistema multiagente proposto opera da seguinte forma: os agentes agem, de forma independente, no ambiente, percorrendo o espaço de busca do problema de otimização instanciado para solucioná-lo. Cada agente opera sobre o ambiente utilizando as habilidades de percepção e ação disponíveis para ele. Estas habilidades determinam a esfera de influência do agente sobre o ambiente; através da cooperação, os agentes disponibilizam o *status* de sua busca e compartilham informações para os demais agentes. O *pool* de soluções é utilizado para disponibilizar estas informações que estarão acessíveis a todos os agentes.

O diagrama UML com as principais classes do *framework* AMAM é apresentado na Figura 4.4. Os pacotes que compõem o *framework* são: *Environment*, *Methods*, *MultiagentSystem* e *Experiment*. Os pacotes *Environment* e *Method* envolvem as principais classes relacionadas à solução do problema de otimização. Todas as informações que são específicas ao problema estão no *Environment* do sistema. Estas informações incluem tanto os dados das instâncias teste, quanto a representação das soluções e as possíveis formas de manipulação destas. Estas formas de manipulação da solução são aqui denominadas *Moves*. O pacote *Methods* destaca as classes utilizadas na definição dos métodos de solução para o problema de otimização, incluindo heurísticas e metaheurísticas.

O pacote *MultiagentSystem* define as principais classes relacionadas ao ambiente multiagente do *framework*. A classe *Agent* tem sempre um *Method* associado a ela e é implementada em uma *Thread* independente. Outros pontos de destaque do sistema multiagente

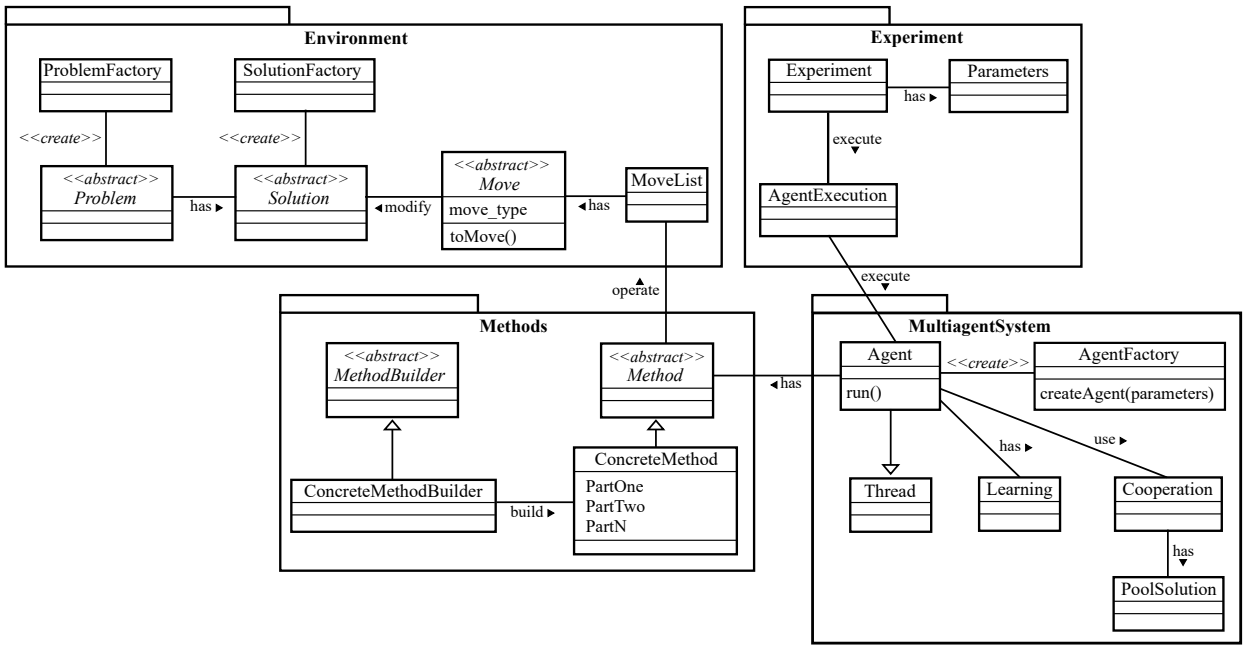


Figura 4.4: Estrutura do *Framework* AMAM.

é a *Cooperation* e as habilidades que podem ser atribuídas ao agente, como é o caso do *Learning*.

O pacote *Experiment* permite conduzir mais facilmente os experimentos a serem realizados, possibilitando gerenciar os parâmetros necessários e a execução dos agentes no sistema multiagente. Cada um destes elementos do *framework* serão apresentados em detalhes nas seções seguintes.

4.3 Dimensão de Ambiente

O ambiente do sistema multiagente do *framework* AMAM corresponde ao espaço de busca do problema de Otimização Combinatória a ser solucionado, ou seja, ao conjunto de todas as soluções viáveis deste problema. Isto quer dizer que, para a solução de diferentes problemas, é necessário apenas que o usuário defina um novo ambiente para cada problema, implementando os elementos que são específicos a ele. A Figura 4.5 apresenta o diagrama UML com as principais classes do ambiente do sistema multiagente. Um problema de otimização é descrito no *framework* por:

- Class *Problem*: define a estrutura de dados para a descrição do problema;
- Class *Solution*: define a representação computacional da solução do problema;
- Class *Move*: define as formas de manipulação da solução disponíveis aos agentes. Esta classe que permite ao agente se mover pelo ambiente, de uma solução à outra, por este motivo o nome dado.

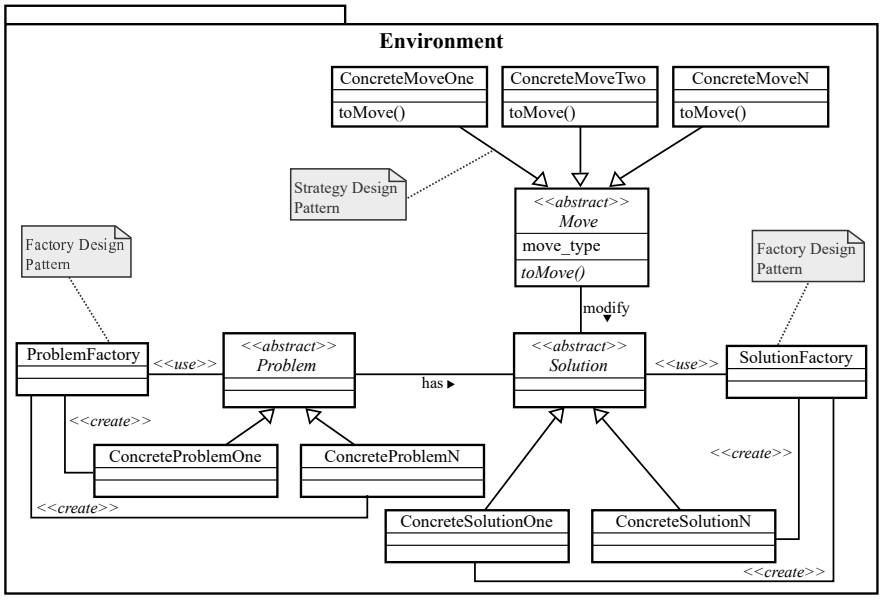


Figura 4.5: Estrutura do Ambiente do Sistema Multiagente AMAM.

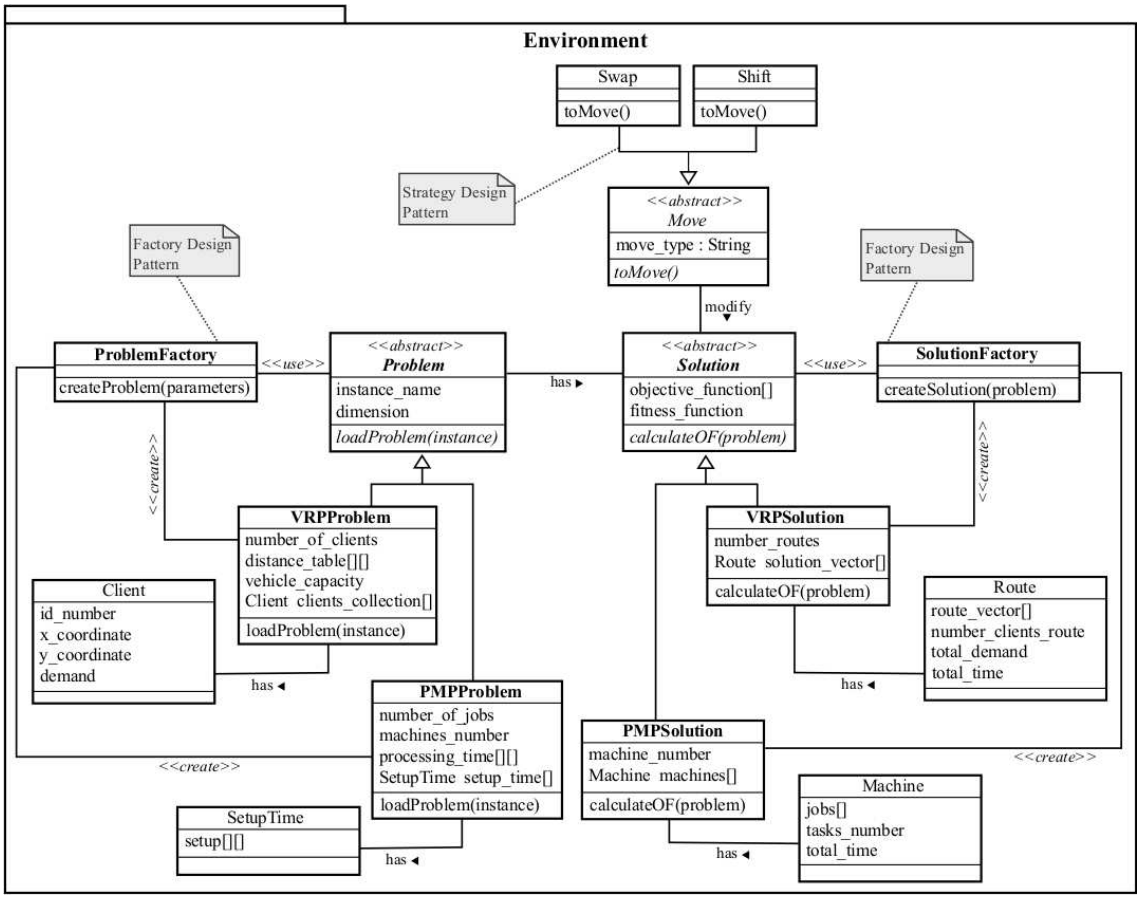


Figura 4.6: Exemplo do Ambiente do Sistema Multiagente aplicado ao Problema de Roteamento de Veículos (VRP) e ao Problema de Sequenciamento de Máquinas Paralelas (PMP).

Algoritmo 2 Função *createProblem* da classe *ProblemFactory*

```
1: function Problem createProblem(parameters)
2:     Problem p ← null;
3:     switch parameters.getProblemDescription() do
4:         case "vrp"
5:             p ← new VRPProblem(parameters);
6:         case "pmp"
7:             p ← new PMPProblem(parameters);
8:         case "mkp"
9:             p ← new MKPProblem(parameters);
10:        otherwise
11:            print("Type of problem not available!");
12:    return p
13: end function
```

O ambiente no qual os agentes irão atuar é definido em tempo de execução, a partir da escolha do problema pelo usuário. As classes *Problem* e *Solution* são abstratas e devem ser utilizadas como referência para a definição de representações específicas. Nestes dois casos, o padrão de projeto `Factory` permite que os elementos do problema a ser solucionado sejam definidos em tempo de execução. De acordo com este padrão de projeto, a lógica de criação de objetos é extraída para uma classe especializada denominada *Factory*. A classe *Factory* encapsula a criação dos objetos, deixando que as subclasses decidam quais objetos serão criados. O diagrama de classe apresentado na Figura 4.5 mostra como o padrão `Factory` é aplicado à construção de diferentes problemas e soluções.

A Figura 4.6 mostra o diagrama de classe com um exemplo do Ambiente do Sistema Multiagente aplicado ao Problema de Roteamento de Veículos (VRP) e ao Problema de Sequenciamento de Máquinas Paralelas (PMP). A classe `ProblemFactory` cria os objetos do tipo `Problem` através da função *Problem createProblem(Parameters parameters)*. O Algoritmo 2 apresenta esta função e ilustra como o padrão de projeto `Factory` é utilizado na criação de Problemas específicos. Neste caso, um dos parâmetros de entrada (*problem description*) define qual tipo de problema deve ser instanciado. Um novo problema pode ser facilmente tratado na *ProblemFactory* inserindo-se um novo *case* à função. No exemplo, as descrições dos problemas "vrp", "pmp", "mkp", indicam, respectivamente, Problema de Roteamento de Veículos, Problema de Sequenciamento em Máquinas Paralelas e Problema da Mochila Multidimensional. O mesmo é feito para a criação de soluções, como mostra o Algoritmo 3. A função *createSolution(Problem p)* instancia a solução em função do problema.

As formas de ação dos agentes sobre as soluções são definidas no ambiente através dos denominados, aqui, como *Move*. Os *Moves* incluem as formas de movimentação do agente pelo ambiente, sendo estas qualquer tipo de modificação de uma solução, como, por exemplo, funções de vizinhança e operadores genéticos. Para que os agentes sejam genéricos e possam atuar em qualquer ambiente (problema), o padrão de projeto `Strategy` permite ao agente operar os *Moves* sem a necessidade de conhecer informações específicas do problema. A Figura 4.5 mostra como o padrão de projeto `Strategy` é aplicado à definição dos *Moves*.

Algoritmo 3 Função *createSolution* da classe *SolutionFactory*

```
1: function SOLUTION CREATESOLUTION(p)
2:     Solution s ← null;
3:     switch p.getProblemDescription() do
4:         case "vrp"
5:             s ← new VRPSolution(p);
6:         case "pmp"
7:             s ← newPMPSolution(p);
8:         case "mkp"
9:             s ← newMKPSolution(p);
10:        otherwise
11:            print("Type of solution not available!");
12:    return s
13: end function
```

No exemplo da Figura 4.6, a classe *Move* é, através do padrão de projeto `Strategy`, estendida para dois tipos de movimentos (`Swap` e `Shift`). O método *toMove*() é reescrito nestas subclasses, definindo, assim, o comportamento específico de cada movimento.

4.4 Dimensão de Agente

A Dimensão de Agente inclui os agentes, suas habilidades e sua relação com o Ambiente do sistema multiagente. Esta relação define a capacidade do agente de perceber e alterar o ambiente, e determina sua esfera de influência, já que cada agente possui uma visão parcial do ambiente. Como mostrado na Figura 4.7, as capacidades de percepção e ação do agente são definidas neste ambiente como:

- Percepção do ambiente: habilidade do agente de acessar as informações sobre o ambiente (problema) que são requeridas a ele;
- Posicionamento: habilidade do agente em definir sua posição no ambiente, ou pela construção de uma nova solução ou pela escolha de uma solução já disponível;
- Movimento: habilidade do agente de se mover, de uma solução a outra, no ambiente. O movimento aqui compreende todas as formas de modificação da solução (como, por exemplo, estruturas de vizinhança e operadores) que permitem ao agente se mover de solução em solução.

Utilizando suas habilidades de percepção e ação, o agente percorre o ambiente e tem a função de buscar a solução para o problema de otimização. Para tal, todo agente encapsula uma heurística/metaheurística que definirá seu comportamento no processo de busca da solução. A principal classe para a definição das heurísticas e metaheurísticas no *framework* é a classe abstrata *Method*. A classe *Method* deve ser utilizada como referência para a definição de representações específicas. Todo agente do *framework* possui um método associado a ele (veja o diagrama de classe da Figura 4.4).

O movimento do agente pelo ambiente é possível, considerando que o *Method* implementado por ele opera os *Moves*, definidos na Dimensão do Ambiente, sem a necessidade

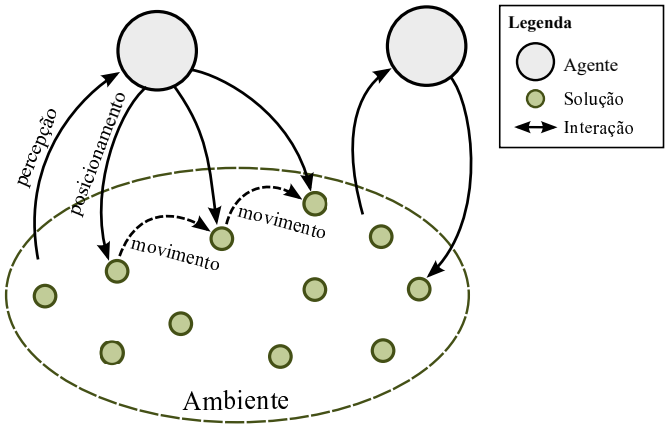


Figura 4.7: Interação entre Agente e Ambiente.

de conhecer informações específicas do problema. O padrão de projeto `Strategy` permite isso. A classe *Move* define o método *toMove()* e as classes derivadas dela sobrepõem o método original com o comportamento específico (veja o exemplo da Figura 4.5). Desta forma, a escolha do movimento a ser aplicado se dá em tempo de execução. É importante ressaltar que os movimentos, implementados para cada tipo de problema, definem a esfera de influência do agente sobre o ambiente.

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 inspirados nestas estruturas. A implementação destes métodos é feita utilizando os padrões de projeto `Strategy` e `Builder`.

O padrão `Strategy` é utilizado para encapsular famílias de algoritmos. Sendo assim, cada variação de uma heurística, metaheurística ou parte destes métodos é encapsulada em uma classe e se torna intercambiável, ou seja, o método pode variar independente do problema a ser solucionado ou do agente que o usa.

As heurísticas e metaheurísticas são aqui definidas como objetos complexos, compostos por diversos componentes independentes (veja a Figura 4.8). Assim, elas podem ser construídas de forma a criar diferentes representações, ou seja, diferentes versões do método. Para tal, é utilizado o padrão de projeto `Builder`. O padrão `builder` permite separar os passos de construção de um objeto em pequenos métodos, cada método deste implementado por uma classe do tipo interface. Uma classe comum, denominada *MethodBuilder*, reúne todos os passos e define como a heurística ou metaheurística deve ser construída, em tempo de execução.

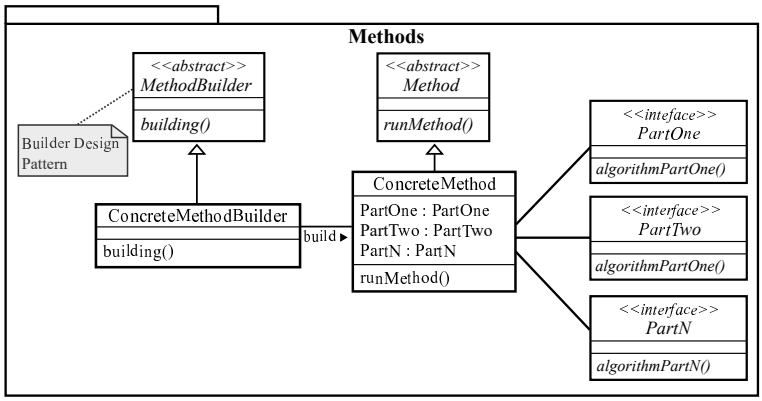


Figura 4.8: Padrão de Projeto *Builder* usado na definição de Heurísticas e Metaheurísticas.

Um exemplo do padrão de projeto `Builder`, apresentado na Figura 4.9, é aplicado à codificação de uma Heurística Construtiva Clássica no *framework*. Neste caso, a heurística é dividida em 6 partes, implementadas, cada uma, em uma interface. As heurísticas construtivas são fortemente ligadas ao problema, pois manipulam diretamente as soluções. A utilização deste padrão assegura que a classe *ConstructiveHeuristic* implemente uma construção de soluções genérica. O Algoritmo 4 apresenta a implementação da heurística construtiva do *framework* AMAM e permite observar o quão genérico ele é. A classe *ConstructiveHeuristicMethodBuilder* permite que a Heurística Construtiva seja construída em tempo de execução, possibilitando ao usuário, a partir da definição de cada um dos elementos, definir seu formato e as especificidades relativas ao problema. A Heurística Construtiva poderá, por exemplo, ser gulosa ou aleatória; ou possuir opções de condição de parada, possibilitando, por exemplo, criar soluções parciais ou completas.

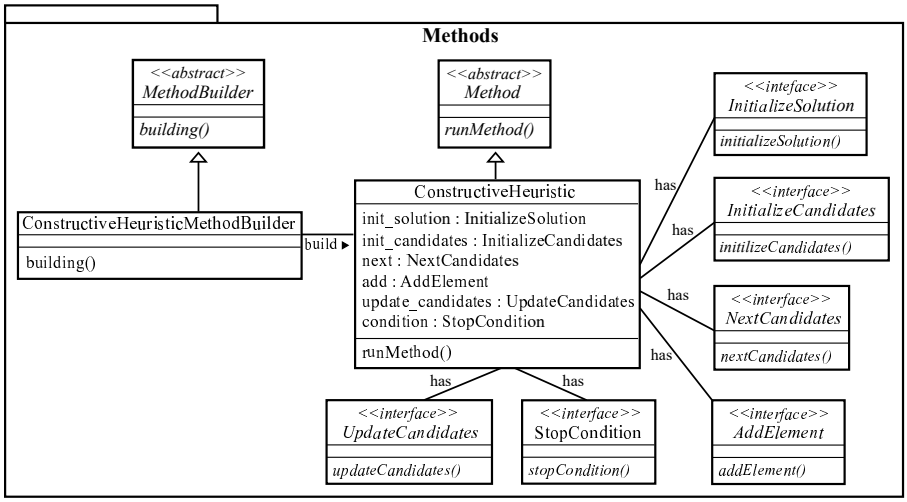


Figura 4.9: Padrão de Projeto *Builder* aplicado à codificação de uma Heurística Construtiva Clássica.

Um segundo exemplo da utilização do padrão de projeto `Builder`, apresentado na Figura 4.10, é aplicado à codificação da metaheurística *Iterated Local Search* (ILS). A metaheurística é aqui composta por 5 elementos: Construção, Busca Local, Perturbação, Condição de Parada e Critério de Aceitação. Assim como no exemplo anterior, o padrão

Algoritmo 4 Função *runMethod* da classe *ConstructiveHeuristic*

```
1: function SOLUTION RUNMETHOD()
2:     s ← init_solution.initializeSolution();
3:     init_candidates.initializeCandidates();
4:     while (condition.stopCondition()) do
5:         next.nextCandidate(s, this.function_type);
6:         s ← add.addElement(s);
7:         update_candidates.updateCandidates();
8:     end while
9:     return s
10: end function
```

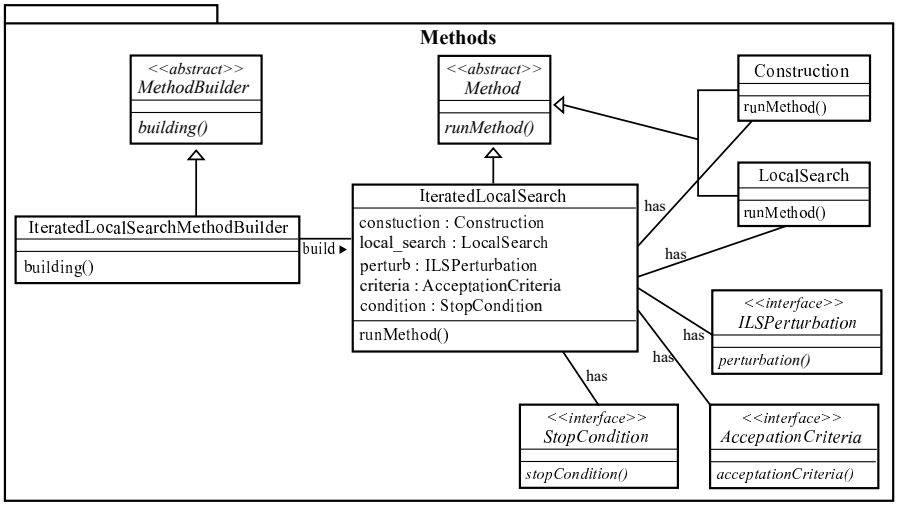


Figura 4.10: Padrão de Projeto `Builder` aplicado a codificação da metaheurística ILS.

de projeto `Builder` permite que a metaheurística seja implementada de forma totalmente genérica (veja o Algoritmo 5). Possibilita, também, a customização dos seus elementos. Desta forma, cada elemento da metaheurística ILS poderá ser definido em tempo de execução. Ela poderá implementar, por exemplo, uma busca local de descida completa ou de primeiro de melhora; ter, como condição de parada, o número de iterações ou o tempo de execução.

A classe *AgentFactory* é responsável por criar todos os agentes de um experimento. Ela irá associar a cada agente um método, a forma de cooperação a ser utilizada e outros recursos, como, por exemplo, o aprendizado. Cada agente do *framework* AMAM é definido em uma *thread* independente (veja Figura 4.4). Sendo assim, os agentes podem executar suas tarefas simultaneamente, com o objetivo de reduzir o tempo de busca da solução e cooperar efetivamente um com o outro. Embora cada agente trabalhe de forma independente, estes podem trocar informações entre si e compartilhar os mesmos recursos do sistema multiagente no qual estão situados. A cooperação entre os agentes é discutida em detalhes na Seção 4.5.

O agente do *framework* possui também capacidades auto-adaptativas, que permitem ao agente modificar suas ações baseado nas experiências adquiridas na interação com outros agentes e com o ambiente. Na versão atual do *framework*, esta experiência é obtida usando os conceitos de Aprendizado por Reforço apresentados em Narendra e Thathachar (1974)

Algoritmo 5 Função *runMethod* da classe *IteratedLocalSearch*

```
1: function RUNMETHOD()
2:     so ← construction.runMethod();
3:     s ← local_search.runMethod(so);
4:     while (condition.stopCondition()) do
5:         s′ ← perturb.perturbation(s);
6:         s″ ← local_search.runMethod(s′);
7:         s ← criteria.acceptationCriteria(s, s″);
8:     end while
9:     return s
10: end function
```

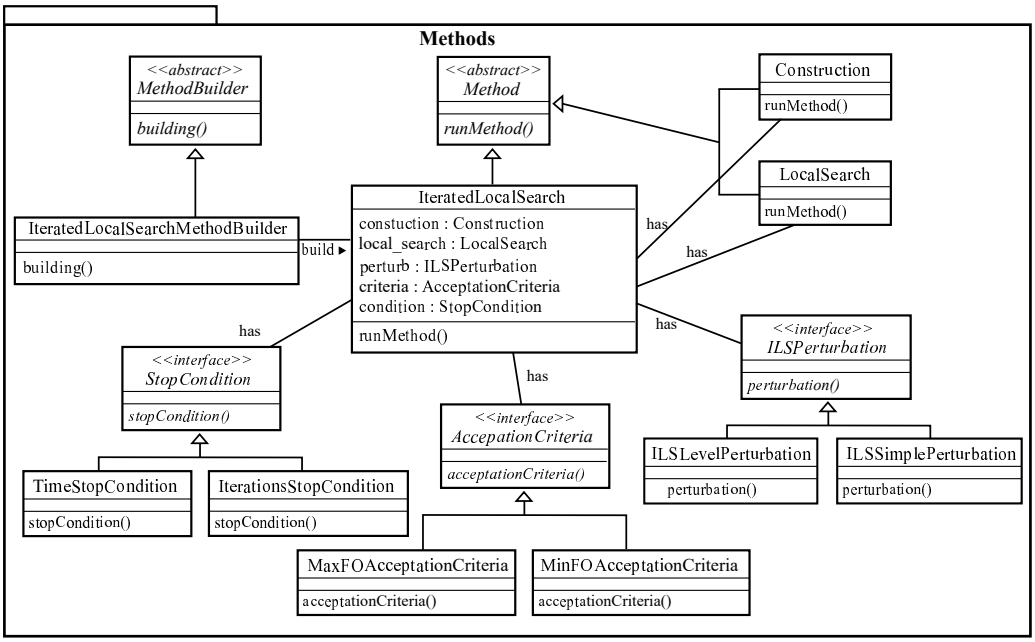


Figura 4.11: Padrão de Projeto `Builder` aplicado a codificação da metaheurística: detalhamento.

e Sutton e Barto (1998). As estratégias de aprendizagem são definidas para cada agente individualmente. A forma como estas novas habilidades estão implementadas no *framework* é apresentada na seção seguinte.

4.4.1 Capacidades Adaptativas do Agente

O principal objetivo da incorporação de habilidades adaptativas no agente é capacitá-lo para se adaptar às características específicas do problema de otimização. Estas habilidades permitem que as metaheurísticas, implementadas de forma genérica para atender às funções de um *framework*, possam se ajustar melhor à variação de problemas e aos parâmetros destes.

Nos modelos implementados no *framework* AMAM, estas habilidades são definidas de duas formas, apresentadas em Silva et al. (2015) e Silva et al. (2019), e descritas nas Seções seguintes. Inicialmente, nas Seções de 2.4 a 2.4.3, são apresentados os conceitos usados

Algoritmo 6 Algoritmo *Variable Neighborhood Descent* (VND)

```
1: procedure VND(x, k_max)   ▷ k_max é o número de estruturas de vizinhança diferentes
2:     k ← 1;
3:     while k ≤ k_max do
4:         x ← bestNeighbor(x, N(k));
5:         if f(x') < f(x) then
6:             x ← x';
7:             k ← 1;
8:         else
9:             k ← k + 1;
10:        end if
11:    end while
12: end procedure
```

como base para estas propostas.

4.4.1.1 Agente Adaptativo Baseado em Autômatos de Aprendizagem

O primeiro formato de habilidades adaptativas atribuídas ao agente do *framework* é baseado nos Autômatos de Aprendizagem (*Learning Automata - LA*). Nesta proposta, a ordem das vizinhanças na busca local é escolhida aplicando-se um conceito muito semelhante ao definido em Narendra e Thathachar (1974). Este conceito se assemelha também ao operador de seleção *roleta* dos Algoritmos Genéticos.

A capacidade adaptativa é atribuída ao agente por meio de uma Busca Local Adaptativa, denominada aqui como ALS-LA (*Adaptive Local Search - Learning Automata*), e é baseada na heurística Descida em Vizinhança Variável (*Variable Neighborhood Descend - VND*) (Mladenović e Hansen, 1997) e nos conceitos de Autômatos de Aprendizagem (LA).

VND é uma heurística de refinamento que explora o espaço de soluções pela troca sistemática de vizinhanças. O Algoritmo 6 apresenta a estrutura padrão do VND. Como é possível observar, para cada vizinhança $\mathcal{N}(k)$ selecionada pelo VND, uma busca local é executada na solução atual para encontrar seu melhor vizinho. Se a solução encontrada for melhor que a solução atual, a primeira função de vizinhança é usada novamente; caso contrário, a próxima função de vizinhança é usada, até que não haja mais vizinhanças disponíveis. O VND retorna um ótimo local em relação às vizinhanças exploradas.

O método VND emprega uma ordenação de vizinhança determinística, sendo este esquema de ordenação das vizinhanças um parâmetro a ser determinado. Em geral, esse esquema é baseado na ordem de crescimento da complexidade dessas estruturas (ou seja, a ordem de aplicação é pré-definida). Entretanto, esta ordem nem sempre produz a melhor solução, pois a melhor ordem pode ser altamente dependente da instância (Subramanian et al., 2010).

Portanto, na Busca Local Adaptativa ALS-LA proposta aqui, para cada estrutura de vizinhança m_1 disponível, uma probabilidade de escolha é atribuída. Inicialmente, todas as vizinhanças têm o mesmo valor no vetor de probabilidades (roleta). A probabilidade de escolha da vizinhança m_1 é atualizada por um fator de reforço w se o movimento gerado pela vizinhança m_1 melhorar a solução atual. A Figura 4.12 apresenta um exemplo do vetor de probabilidades colocado no formato de roleta. Naturalmente, quanto maior o percentual de probabilidade, maior a chance da vizinhança ser escolhida.

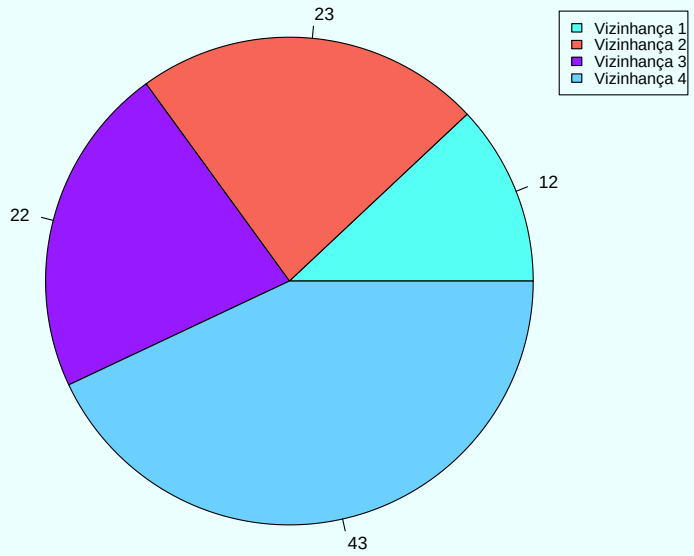


Figura 4.12: Exemplo do vetor de probabilidades colocado no formato de roleta.

O Processo de Decisão de Markov (MDP) para esta proposta é definido da seguinte forma:

- Conjunto de estados S: estados são as funções de vizinhança, disponíveis para o problema a ser tratado pelo *framework*. Estar em um estado, no contexto do agente adaptativo, implica em aplicar uma busca local à solução, utilizando a função de vizinhança definida no estado. No caso dos problemas de teste usados aqui, temos:

  (a) Para o estudo de caso do VRPTW: o conjunto de estados é formado pelas 8 funções de vizinhança listadas na Seção 3.1.2:

  $$S_{vrptw} = \{\textit{Intra-Route Swap}, \textit{Inter-Route Swap}, \textit{Intra-Route Shift}, \textit{Inter-Route Shift}, \textit{Two Intra-Route Swap}, \textit{Two Intra-Route Shift}, \textit{Eliminates Smaller Route}, \textit{Eliminates Random Route}\} \quad (4.1)$$

  (b) Para o estudo de caso do UPMSP-ST: o conjunto de estados é formado pelas 4 funções de vizinhança listadas na Seção 3.2.2:

  $$S_{upmsp-st} = \{\textit{Insertion in the Different Machines}, \textit{Insertion in the Same Machines}, \textit{Swap between Different Machines}, \textit{Swap between Same Machines}\} \quad (4.2)$$

- Conjunto de Ações $A(s)$: assim como em Narendra e Thathachar (1974), estados e ações são considerados sinônimos;

- Recompensa R: valor obtido empiricamente. O valor utilizado nos experimentos da Seção 4.6 é de 5%. Este valor está relacionado ao número de vizinhanças do

Algoritmo 7 Busca Local Adaptativa baseada em Autômatos de Aprendizagem – ALS-LA

```
1: function Solution adaptiveLocalSearchLearningAutomata(x_0, w)
2:     no_improvement ← 0;
3:     states_visited_count ← 0;
4:     x* ← x_0;
5:     x ← x_0;
6:     repeat
7:         k ← chooseAnState(no_improvement);
8:         x ← bestNeighbor(k, x);
9:         if (x is better than x*) then
10:            x* ← x;
11:            improved ← true;
12:            no_improvement ← 0;
13:            updateRoulette(k, w);
14:        else
15:            no_improvement + +;
16:            if (the state has been visited) then
17:                states_visited_count + +;
18:            end if
19:            if ((no_improvement > max_iterations_without_improvement) and (states_visited_count = q_size)) then
20:                improved ← false;
21:            end if
22:        end if
23:    until (improved = false)
24:    return x;
25: end function
```

problema teste utilizado, sendo que, se o número de vizinhanças for maior, esse valor deve diminuir.

Algoritmo 8 Função *chooseAnActionLA*

```
1: function chooseAnActionLA(no_improvement)
2:     if (no_improvement ≠ 0) then
3:         k ← rouletteWheel();
4:     else
5:         k ← maximumRoulette();
6:     end if
7:     return k;
8: end function
```

A Busca Local Adaptativa (ALS-LA) é apresentada no Algoritmo 7. Nesse algoritmo, a escolha da vizinhança k a ser aplicada a cada iteração é realizada pela função *chooseAnActionLA*(*no_improvement*) (linha 7 do Algoritmo 7). Esta função é detalhada no Algoritmo 8. Ela define qual será a forma de escolha da vizinhança com base no parâmetro `no_improvement`, que indica se houve melhora na solução na iteração anterior ou não. Se a solução encontrada na iteração anterior for melhor que a melhor solução já encontrada, a

Algoritmo 9 Função *maximumRoulette*

```
1: function MAXIMUMROULETTE()
2:     max_value = fraction_roulette[0];
3:     for (i ← 1 to max_neighborhoods) do
4:         if (fraction_roulette[i] > max_value) then
5:             max_value ← fraction_roulette[i];
6:             pos ← i;
7:         end if
8:     end for
9:     return pos;
10: end function
```

Algoritmo 10 Função *rouletteWheel*

```
1: function ROULETTEWHEEL()
2:     pos = 0;
3:     i = 0;
4:     roulette_value ← random(roulette_total);
5:     while (roulette[i] ≤ roulette_value) do
6:         i++;
7:         pos = 1;
8:     end while
9:     return pos;
10: end function
```

função utilizada na definição da próxima vizinhança é aquela com maior probabilidade no vetor de probabilidades (`maximumRoulette()`). Se a solução não foi melhorada, é aplicada a função `rouletteWheel()`, que escolhe a vizinhança a partir de um valor aleatório. As funções `maximumRoulette()` e `rouletteWheel()` são apresentadas, respectivamente, nos Algoritmo 9 e 10.

Sempre que uma melhor solução é encontrada, a probabilidade da vizinhança que gerou esta solução é atualizada no vetor de probabilidades (veja linha 13 do Algoritmo 7). A função de atualização do vetor de probabilidades, denominada `updateRoulette(k)`, é apresentada no Algoritmo 11. Nesta função, uma recompensa (*weigth*) é atribuída à probabilidade da vizinhança k. Naturalmente, as probabilidades das demais vizinhanças devem diminuir. O valor da recompensa atribuída à vizinhança k é dividido e retirado igualmente das demais vizinhanças. Entretanto, para que uma vizinhança não seja completamente removida da roleta, uma probabilidade mínima é definida.

A busca local ALS-LA só encerra o seu processo de busca a partir de um número de iterações sem melhora ou quando todas as vizinhanças foram usadas e uma nova melhor solução não foi encontrada.

4.4.1.2 Agente Adaptativo Baseado em *Q-Learning*

Esta seção mostra o agente adaptativo baseado em *Q-Learning*. A capacidade adaptativa é atribuída ao agente por meio de uma Busca Local Adaptativa, denominada aqui como ALS-QLearning, *Adaptive Local Search - Q-Learning*, baseada na heurística Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND) (Mladenović e Hansen,

Algoritmo 11 Função *updateRoulette*

```
1: function UPDATEROULETTE(k, weight, minimum_probability)
2:     fraction_roulette[k] = fraction_roulette[k] + weight;   ▷ atualiza a fração da vizinhança k na roleta - somando o peso (recompensa)
3:     cont ← 0;
4:     for (i ← 0 to max_neighborhoods) do
5:         if (k ≠ i) then
6:             if (fraction_roulette[i] < minimum_probability) then
7:                 cont + +;
8:             end if
9:         end if
10:    end for
11:    weight_to_subtract ← weight/(max_neighborhoods − 1 − cont);
12:    for (i ← 0 to max_neighborhoods) do ▷ atualiza a fração das demais vizinhanças na roleta - subtraindo o peso adicionado a vizinhança k
13:        if (fraction_roulette[i] ≥ minimum_probability) then
14:            if (i ≠ k) then
15:                fraction_roulette[ji] ← fraction_roulette[ji] − weight_to_subtract;
16:            end if
17:        end if
18:        if (j = 0) then
19:            roulette[i] ← fraction_roulette[i];
20:        else
21:            roulette[i] ← roulette[−1] + fraction_roulette[i];
22:        end if
23:    end for
24:    return roulette;
25: end function
```

1997) e nos conceitos de Aprendizado por Reforço (*Reinforcement Learning* – RL) (Sutton e Barto, 1998), mais especificamente no método *Q-Learning* (Watkins e Dayan, 1992).

O objetivo é permitir que o agente modifique suas ações com base nas experiências adquiridas na interação com os demais agentes e com o meio ambiente. Esta experiência é obtida usando os conceitos de Aprendizado por Reforço, mais especificamente, através do algoritmo *Q-Learning* (Watkins e Dayan, 1992). Aqui, o conceito de Aprendizado por Reforço também é usado para definir a ordem de aplicação das estruturas de vizinhança da busca local. Uma tabela Q é utilizada para armazenar os valores dos pares de estado-ação no decorrer do processo de aprendizagem. A função que atualiza a tabela Q determina a recompensa que será usada como reforço do aprendizado e pode definir a *qualidade* de uma ação tomada em um determinado estado.

Nesta proposta, a sequência em que as vizinhanças são aplicadas é definida através do aprendizado por reforço, baseado no algoritmo *Q-Learning*. O foco principal é avaliar o ganho obtido com a aplicação de uma sequência de duas vizinhanças e, a partir daí, recompensar as melhores sequências e maximizar a recompensa acumulada. Cada vizinhança a ser usada pelo método de busca é considerada, neste artigo, como um estado de aprendizado.

O Processo de Decisão de Markov (MDP) para esta proposta é definido da seguinte

forma:

- Conjunto de estados S: estados são as funções de vizinhança, disponíveis para o problema a ser tratado pelo *framework*. Estar em um estado, no contexto do agente adaptativo, implica em aplicar uma busca local à solução, utilizando a função de vizinhança definida no estado. No caso dos problemas de teste usados aqui, são utilizadas as mesmas funções de vizinhança da proposta ALS-LA:

  (a) Para o estudo de caso do VRPTW: o conjunto de estados é formado pelas 8 funções de vizinhança listadas na Seção 3.1.2:

  $$S_{vrptw} = \{\textit{Intra-Route Swap}, \textit{Inter-Route Swap}, \textit{Intra-Route Shift}, \textit{Inter-Route Shift}, \textit{Two Intra-Route Swap}, \textit{Two Intra-Route Shift}, \textit{Eliminates Smaller Route}, \textit{Eliminates Random Route}\} \quad (4.3)$$

  (b) Para o estudo de caso do UPMSP-ST: o conjunto de estados é formado pelas 4 funções de vizinhança listadas na Seção 3.2.2:

  $$S_{upmsp-st} = \{\textit{Insertion in the Different Machines}, \textit{Insertion in the Same Machines}, \textit{Swap between Different Machines}, \textit{Swap between Same Machines}\} \quad (4.4)$$

- Conjunto de Ações $A(s)$: uma ação é definida como a mudança de um estado para outro (*ir para*). Desta forma, o conjunto de ações pode ser representado por um grafo completo, no qual cada ação é representada por um arco conectando dois estados (nós do grafo). Um exemplo de um grafo representando a relação entre os estados (funções de vizinhança) e as ações possíveis é mostrado na Figura 4.13(a). Neste exemplo, apenas quatro funções de vizinhança do VRPTW são usadas para facilitar a visualização. A tabela Q, referente a este exemplo, é mostrada na Figura 4.13(b). A tabela Q tem dimensões dadas por $M \times M$, em que M é o número de estados, isto é, o número de funções de vizinhança do problema tratado.

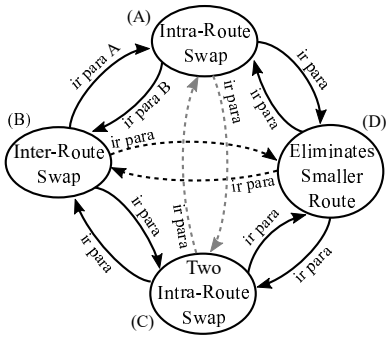
- Recompensa R: valor obtido empiricamente. Inicialmente, foi utilizado um valor baseado na função objetivo. Este valor era obtido subtraindo-se o valor da função objetivo encontrado pela primeira vizinhança do valor da função objetivo encontrado pela segunda vizinhança. Após a realização de testes, observou-se que esta estratégia gerava valores de recompensa maiores no início da busca, e valores muito menores no avançar da busca. Portanto, o procedimento de se considerar valores diferentes de recompensa beneficia vizinhanças em determinados instantes da busca e, assim, não se mostrou como uma estratégia adequada. Desta forma, foi utilizado um valor fixo de recompensa em toda a busca.

Como as funções de vizinhança são parâmetros específicos do problema a ser resolvido, o propósito de usar aprendizado por reforço é permitir que o *framework* se adapte melhor às características próprias deste problema.

O algoritmo de busca local adaptativa proposto é apresentado no Algoritmo 12. Nesse algoritmo, em primeiro lugar, a tabela Q é inicializada. Essa tabela é responsável por armazenar os valores dos pares estado-ação no decorrer do processo de aprendizado. Nesta proposta, cada elemento da tabela Q é inicializado com o valor zero; assim, o processo de aprendizado é realizado de maneira não tendenciosa.

Algoritmo 12 Busca Local Adaptativa baseada em *Q-learning* - ALS-*QLearning*

```
1: function SOLUTION ADAPTIVELOCALSEARCHQLEARNIG(x_0, w)
2:     initialize(Q(state, action));
3:     improved ← true;
4:     no_improvement ← 0;
5:     x* ← x_0;
6:     x ← x_0;
7:     next_state ← chooseAnAction(0, 2); ▷ 2: initial state defined by random function
8:     x ← bestNeighbor(next_state, x);
9:     if (x is better than x*) then
10:        x* ← x;
11:    else
12:        no_improvement ← 1;
13:    end if
14:    repeat                                                    ▷ for each episode
15:        states_visited_count ← 0;
16:        state ← next_state;
17:        while (not reached the state goal) do      ▷ state goal: improving the solution
18:            if (no_improvement = 0) then
19:                next_state = chooseAnAction(state, 1);     ▷ 1: epsilon greedy function
20:            else
21:                next_state = chooseAnAction(0, 2);         ▷ if not improved, the greedy
   function should not be used
22:            end if
23:            x = bestNeighbor(next_state, x);
24:            if (x is better than x*) then                  ▷ reached the state goal
25:                x* ← x;
26:                improved ← true;
27:                no_improvement ← 0;
28:                calculateQValue(state, next_state, w);
29:            else
30:                no_improvement + +;
31:                if (the state has been visited) then
32:                    states_visited_count + +;
33:                end if
34:                if ((no_improvement > max_iterations_without_improvement) and
   (states_visited_count = q_size)) then
35:                    improved ← false;
36:                end if
37:            end if
38:        end while
39:        ε ← ε * decay_rate;
40:    until (not(improved))
41:    return x;
42: end function
```



(a) Grafo com quatro estados (funções de vizinhança do VRPTW) e as respectivas ações do MDP.

Tabela Q

estado/ação	A	B	C	D
A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

(b) Tabela Q inicial para as quatro funções de vizinhança.

Figura 4.13: Grafo representando a relação entre estados (funções de vizinhança e possíveis ações.

O estado inicial é definido na linha 7 do Algoritmo 12. A função *chooseAnAction(state, type_function)*, apresentada no Algoritmo 13, permite que diferentes formas de seleção do próximo estado sejam facilmente utilizadas, ou, se necessário, novas funções sejam facilmente inseridas no *framework*. Sendo assim, neste caso, o estado inicial é determinado com *type-function* = 2, ou seja, de forma aleatória, como mostrado na linha 6 do Algoritmo 13. Funções específicas são também usadas para determinar os próximos estados, sendo *type-function* = 1 utilizado para a função ϵ-*greedy*, como mostrado na linha 3 do Algoritmo 13. A função ϵ-*greedy* é descrita a seguir.

Algoritmo 13 Função *chooseAnActionQL*

```
1:  function CHOOSEANACTIONQL(state, type_function)
2:      next_state ← 0;
3:      if type_function = 1 then
4:          next_state ← ε-greedy(state);
5:      else
6:          if type_function = 2 then
7:              next_state ← randomAction();
8:          end if
9:      end if
10:     return next_state;
11: end function
```

O estado inicial é aplicado à solução através da estratégia Descida/Subida Completa (função *bestNeighbor(next-state, x)*, na linha 8 do Algoritmo 12).

Para cada episódio relacionado ao algoritmo *Q-Learning*, o aprendizado é associado ao processo de busca da solução. A cada episódio, iniciado na linha 14 do algoritmo 12, o número de estados já visitados é reiniciado, e o estado aplicado na iteração anterior é registrado (linhas 15 e 16 do algoritmo 12).

Um episódio é executado até que a meta seja alcançada (linhas 17 a 38 do Algoritmo 12). A meta é encontrar uma solução melhor que a atual e, se alcançada, o episódio corrente é

encerrado. Os outros estados a serem visitados no episódio são definidos usando a função ϵ-*greedy* (Watkins, 1989), apresentada no Algoritmo 14, se na iteração anterior houve melhora na solução. A função ϵ-*greedy* seleciona uma ação aleatória com uma probabilidade ϵ ou uma ação que retorna a recompensa mais alta com uma probabilidade de $1 - \epsilon$. Se, na iteração anterior, não houve melhora na solução, a função usada é a aleatória.

Algoritmo 14 Função ϵ-*greedy*

```
1: function ε-GREEDY(current_state, ε, q_size)
2:     p ← random();
3:     if p ≤ ε then
4:         action ← random(q_size);
5:     else
6:         action ← maxAction(current_state);
7:     end if
8:     return action;
9: end function
```

O próximo estado também é aplicado à solução através da estratégia Descida/Subida Completa (função *bestNeighbor (next-state, x)*, na linha 23 do Algoritmo 12). Quando a solução melhora, a recompensa é atribuída, seu valor na matriz Q é calculado, o valor de ϵ diminui com a taxa de decaimento e o episódio termina.

A recompensa usada foi definida empíricamente e o mesmo valor foi usado em todas as situações. Como o objetivo é definir o ganho obtido pela aplicação de uma sequência de duas vizinhanças, a atualização da matriz Q leva em consideração o estado corrente (selecionado na iteração atual - $next\_state$) e o estado da iteração anterior ($state$). Desta forma, se a aplicação de uma vizinhança m_2 após a aplicação de uma vizinhança m_1 resultar em melhora da solução corrente, a função $calculateQValue(state, next\_state, reward)$ é utilizada para atualizar a matriz Q, modificando, assim, o valor de $Q(m_1, m_2)$.

Considerando as características específicas dos problemas de otimização, uma vizinhança precisa ser aplicada a uma solução, a partir de uma heurística de busca local de Descida/Subida, apenas uma vez. Em consequência, os estados visitados (vizinhanças) são registrados. Quando todos os estados tiverem sido visitados, a busca local estará concluída. O mesmo acontece após um certo número de iterações sem melhoria.

4.5 Dimensão Social

A dimensão Social envolve as estruturas que permitem a interação e cooperação entre os agentes. A comunicação entre agentes ocorre através da troca de informações no espaço de busca do problema. O objetivo dessa estrutura cooperativa é orientar os agentes no espaço de soluções em direção às áreas mais promissoras e, assim, melhorar o resultado final e reduzir o tempo necessário para solucionar o problema.

Uma estratégia de memória adaptativa chamada *Pool* de Soluções é usada para compartilhar as informações. As soluções disponíveis são armazenadas neste *pool* de soluções, localizado no ambiente do sistema multiagente. As soluções são compartilhadas pelos agentes no final de cada iteração. A comunicação é regida pelas regras de acesso ao *pool*, tanto para escrita quanto para leitura, visando garantir diversidade no compartilhamento de informações de busca.

O tamanho máximo do *pool* de soluções é pré-definido. A inserção de novas soluções é regulada por uma função de avaliação, dada por:

$$g(\phi_i) = \sum_{j=1}^{P} \phi(\lambda_{ij}) \tag{4.5}$$

na qual P é o número de soluções no *pool* e λ_{ij} é a distância entre as soluções i e j. A função $g(\phi_i)$ estima a concentração de soluções na vizinhança da solução i por meio da distância entre esta solução i e as demais soluções contidas no *pool*. Esta função de avaliação é baseada nas técnicas de nicho (Li et al., 2017) para coordenar arquivos de solução.

Na expressão (4.5), a função $\phi(\lambda_{ij})$ é definida como:

$$\phi(\lambda_{ij}) = \begin{cases} 1 - \dfrac{\lambda_{ij}}{pr}, & \text{if } \lambda_{ij} \leq pr \\ \\ 0, & \text{if } \lambda_{ij} > pr \end{cases} \tag{4.6}$$

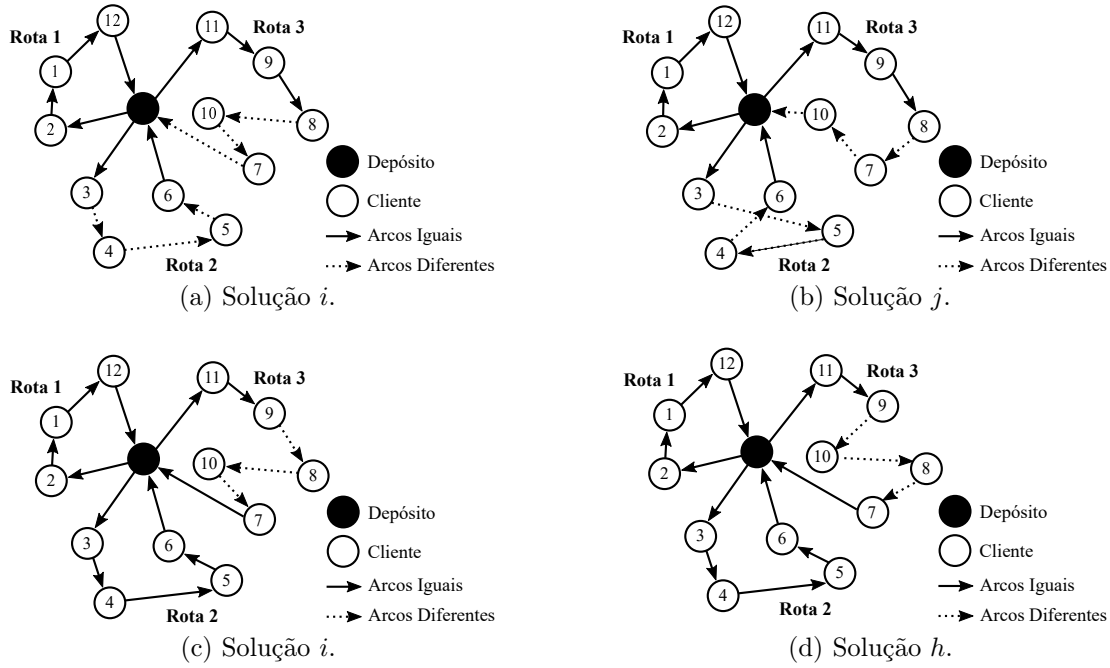
O fator pr é o raio do *pool* e controla o grau de dispersão das soluções, sendo um parâmetro do problema. Por exemplo, para o Problema de Roteamento de Veículos com Janelas de Tempo (VRPTW), usado como um dos estudos de caso neste trabalho, o valor do fator pr é dado pelo número mínimo de arcos comuns para que uma solução seja considerada muito próxima à outra. Portanto, seu valor depende diretamente da dimensão da instância a ser resolvida para o problema abordado.

O valor λ_{ij} mede o quanto as soluções i e j são semelhantes e depende fundamentalmente do problema a ser tratado. Como exemplo, considerando também o VRPTW, a distância entre duas soluções é calculada em relação ao número de arcos que não são comuns às duas soluções. A Figura 4.14 apresenta dois exemplos de cálculo da distância entre as soluções do VRPTW. Para o primeiro exemplo, a distância entre a solução i, mostrada na Figura 4.14(a), e a solução j, mostrada na Figura 4.14(b), é igual a $\lambda_{ij} = 12$, ou seja, existem 12 arcos não comuns entre essas soluções. Para o segundo exemplo, a distância entre a solução i, mostrada na Figura 4.14(c), e a solução h, mostrada na Figura 4.14(d), é igual a $\lambda_{ih} = 6$, ou seja, existem 6 arcos não comuns entre essas soluções.

Como consequência, após avaliar o valor $g(\cdot)$ de cada solução i no *pool*, a solução que possui a pior avaliação a partir desta análise é excluída para a inserção de uma nova solução no *pool*. A nova solução a ser inserida deve atender a dois critérios: (i) ainda não está no *pool*; (ii) tem um valor de função objetivo melhor que a pior solução do *pool*.

O principal objetivo desta função de avaliação é manter a diversidade do *pool*, evitando manter soluções muito semelhantes ou até mesmo iguais. Ao mesmo tempo, a melhor solução existente no *pool* é sempre armazenada em um atributo específico e atualizada a cada inserção, evitando que essa melhor solução encontrada seja eliminada.

A classe *Solution* está associada a duas classes utilizadas no processo de cooperação: *Sender* e *Receiver*. Estas classes armazenam a identificação dos agentes que enviam soluções ao *Pool* de Soluções ou utilizam as soluções lá disponíveis. A classe *Sender* identifica o agente que enviou a solução para o *Pool* e a classe *Receiver* identifica os agentes que utilizaram a solução do *Pool* em seu processo de busca. Estas informações permitem a análise do processo de cooperação após a realização de experimentos computacionais.



(a) Solução i. (b) Solução j.

(c) Solução i. (d) Solução h.

Figura 4.14: Exemplos de cálculo da distância entre as soluções para o VRPTW.

4.6 Pacote *Experiment*

A inclusão do pacote *Experiment* facilita a realização de experimentos no *framework*. A partir deste elemento é possível gerenciar os parâmetros tanto do sistema, quanto das metaheurísticas, e a execução dos agentes no sistema multiagente.

O pacote *Experiment* inclui todas as classes que permitem gerenciar os experimentos (veja Figure 4.4). Um experimento, aqui, é definido por n_try execuções de uma dada configuração do sistema multiagente, com n agentes, sobre uma instância teste específica de um dado problema.

A Figura 4.15 apresenta o diagrama de sequência que descreve o funcionamento do experimento. Em linhas gerais, com a execução de um experimento, o sistema opera da seguinte forma (veja Figure 4.15): a cada experimento executado, n_try execuções são disparadas (primeiro *loop* do diagrama apresentado na Figura 4.15). Para cada execução, um dado número de agentes é criado pela classe $AgentFactory$ (segundo *loop* do diagrama). O processo de criação de um agente inclui:

(i) a instanciação de um novo agente;

(ii) a criação de uma classe construtora MethodBuilder;

(iii) a construção da metaheurística (*method*), de acordo com os parâmetros previamente definidos; e

(iv) a atribuição desta metaheurística ao agente que irá utilizá-la.

Após instanciar todos os agentes, cada um é atribuído a uma *thread* e inicia sua ação no sistema (terceiro *loop* do diagrama). A classe $MainThread$ é responsável por gerenciar as *threads* dos agentes de uma dada execução. Destacamos aqui que o diagrama apresentado

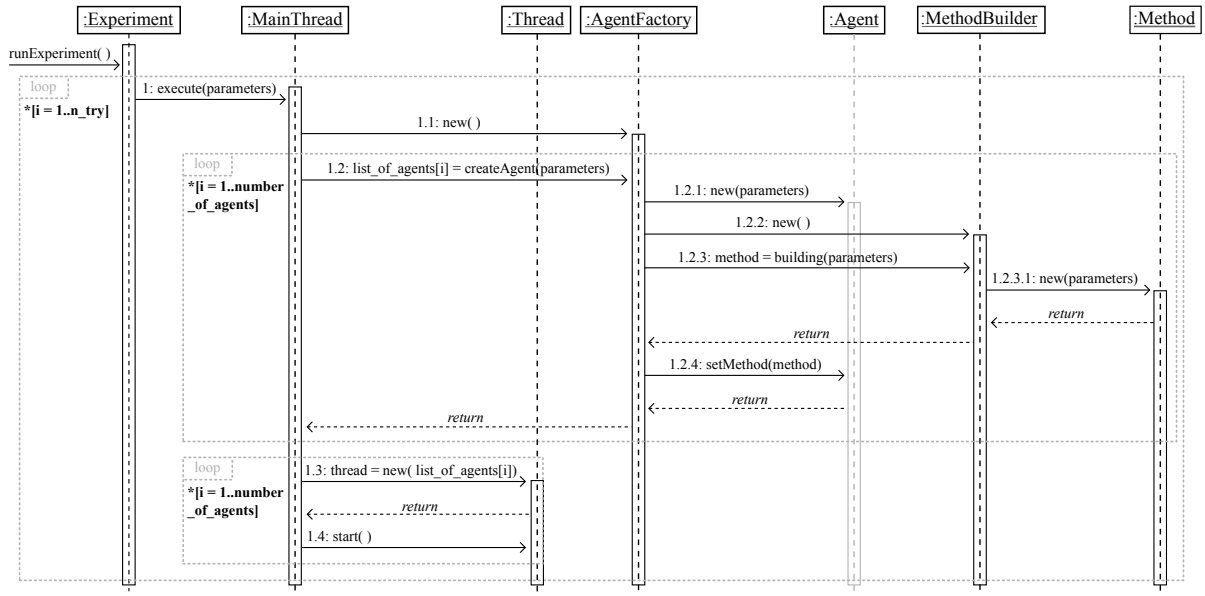


Figura 4.15: Comportamento do sistema quando executando um experimento no *Framework* AMAM.

na Figura 4.15 se limita a apresentar a execução do experimento, sem detalhar a ação do agente e como o método é construído.

Nos experimentos realizados neste artigo, por exemplo, são realizadas 30 execuções ($n\_try = 30$) de cada cenário de teste com 1, 2, 4 e 8 agentes. Os agentes instanciados e disparados em um cenário de teste são executados simultaneamente.

Capítulo 5

Aplicações e Resultados

Neste Capítulo são descritos os detalhes de implementação, instâncias teste, parâmetros utilizados nos experimentos computacionais realizados, assim como os resultados obtidos e a discussão destes. Para os experimentos apresentados aqui, foram desenvolvidas instâncias do *framework* AMAM voltadas para os dois problemas clássicos de otimização combinatória apresentados no Capítulo 3: Problema de Roteamento de Veículos com Janelas de Tempo (*Vehicle Routing Problem with Time Windows* – VRPTW) e Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempos de Configuração Dependentes da Sequência (*Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times* – UPMSP-ST). O propósito destas instanciações é analisar a flexibilidade do *framework* AMAM, aplicando-o a diferentes problemas de otimização, avaliar sua performance e mostrar seu potencial.

Este Capítulo está organizado da seguinte forma: a Seção 5.1 detalha os experimentos realizados e a Seção 5.2 apresenta os resultados obtidos e a discussão destes.

5.1 Experimentos

Os experimentos computacionais aqui apresentados foram realizados com o objetivo de testar e avaliar o *framework* AMAM. Para este propósito, o *framework* proposto foi instanciado para os dois problemas de Otimização Combinatória já apresentados: Problema de Roteamento de Veículos com Janela de Tempo (VRPTW) e Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempos de Configuração Dependentes da Sequência (UPMSP-ST).

O *framework* proposto aqui foi implementado em Java, com o `JDK 8`, utilizando a `IDE Elipse`. Os resultados foram obtidos usando um computador com processador Intel `i7-4500U`, 1,8 GHz, 16 GB de RAM DDR3, com sistema operacional Windows 7. É importante ressaltar que os testes aqui apresentados foram executados em um *notebook* comum, com um único processador, mostrando a eficácia do *framework* exposto através do uso de *threads*. É também digno de nota que competir com os melhores resultados da literatura para as instâncias dos problemas instanciados está fora do escopo deste experimento.

Das instâncias do VRPTW com 100 clientes propostas por Solomon (1987), foram usadas 16 instâncias nos experimentos para esta instanciação do *framework*. São elas: C107, C108, C109, C206, C207, C208, R110, R111, R112, R209, R210, R211, RC106, RC107, RC108, RC206, RC207, RC208. Essas instâncias são formadas por três diferentes conjuntos de clientes (*C-Cluster*, *R-Random* e *RC-Random-Cluster*), de acordo com a distribuição geográfica considerada. Posições geográficas dos clientes são geradas aleatoriamente nos

Algoritmo 15 Algoritmo *Iterated Local Search* (ILS)

```
1: procedure ILS
2:     x_o ← pfihInitialSolution();
3:     x ← localSearch(x_o);
4:     while (not reached the stopping condition) do
5:         x' ← perturbation(x, level_perturbation);
6:         x'' ← localSearch(x');
7:         if (acceptationCriteria(x, x'')) then
8:             x ← x'';
9:             level_perturbation ← 1;
10:        else
11:            level_perturbation + +;
12:        end if
13:    end while
14: end procedure
```

conjuntos de problemas R1 e R2. Nos conjuntos de problemas C1 e C2, as posições geográficas são agrupadas. Além disso, uma mistura de estruturas aleatórias e agrupadas é usada nos conjuntos de problemas RC1 e RC2.

O conjunto de instâncias usadas para testes computacionais associados ao UPMSP-ST foi proposto por Vallada e Ruiz (2011) e está disponível em `http://soa.iti.es/problem-instances`. Para avaliar o AMAM, 16 instâncias deste conjunto de dados foram escolhidas. Estes problemas de teste utilizados envolvem combinações de 50 tarefas, com 10, 15, 20 e 25 máquinas. Assim como no VRPTW, o principal objetivo deste experimento não é superar os melhores resultados da literatura para UPMSP-ST.

No ambiente multiagente das instanciações do *framework* utilizadas, cada agente é executado em sua própria *thread*. Os elementos do ambiente são compartilhados entre todos os agentes. Para garantir esse compartilhamento, o ambiente é passado por referência aos agentes, o que significa que todos os agentes têm acesso ao mesmo local na memória, e que a modificação realizada por um agente imediatamente está disponível aos demais.

Os experimentos foram conduzidos com o objetivo de analisar o desempenho dos agentes do *framework*, tanto individualmente, quanto em equipe; tanto em relação ao agentes adaptativos propostos aqui, quanto em relação à estrutura cooperativa apresentada. O objetivo principal é avaliar se a forma de aprendizagem, embutida no agente, tem influência direta no desempenho do *framework* em relação à qualidade dos resultados obtidos, tanto do ponto de vista individual quanto do ponto de vista do trabalho em equipe. Além disso, avaliar se o comportamento cooperativo influencia diretamente a qualidade das soluções.

Os agentes usados neste experimento implementam uma variação das metaheurísticas *Iterated Local Search* (ILS) (Lourenço et al., 2003), uma bem conhecida metaheurística de trajetória. Este método é apresentado no Algoritmo 15. Neste algoritmo, a perturbação da solução, realizada a partir de mudanças na solução atual, é implementada em níveis, ou seja, em cada iteração a função de perturbação é alterada se não houver melhoria na solução (linha 11), e retorna ao seu primeiro nível (linha 9), se uma melhor solução é encontrada.

A função $localSearch(s')$ (linha 6) implementa três versões diferentes da metaheurística de busca local VND:

(i) VND Clássico, apresentado no Algoritmo 6;

(ii) ALS-LA (*Adaptive Local Search – Learning Automata*), descrita na Seção 4.4.1.1;

(iii) ALS-*QLearning* (*Adaptive Local Search – QLearning*), descrita na Seção 4.4.1.2.

Estas três versões são utilizadas para a avaliação do desempenho de cada uma das propostas apresentadas aqui (ALS-LA e ALS-*QLearning*), em relação à versão clássica do VND.

Para os testes computacionais, os valores dos parâmetros de aprendizado por reforço para o algoritmo *Q-Learning* (veja Algoritmo 1) usados nos experimentos são $\gamma = 0,9$; $\alpha = 0,1$; e $\epsilon = 0,05$; com taxa de decaimento dada por $0,999$.

A composição do sistema multiagente, usada nos experimentos, envolveu agentes ILS idênticos. O processo de cooperação utilizado é o proposto neste trabalho e é descrito na Seção 4.5. Quatro cenários de agentes são usados para avaliar o *framework*:

(i) um único agente ILS;

(ii) dois agentes ILS idênticos;

(iii) quatro agentes ILS idênticos;

(iv) oito agentes ILS idênticos.

Nesse contexto, o *framework* foi analisado de 5 maneiras:

(i) Desempenho da proposta de aprendizado ALS-LA: análise específica dos resultados obtidos pela proposta ALS-LA, comparando o desempenho do cenário em que há um único agente, que realiza a busca de forma isolada, com o desempenho dos demais cenários (2 agentes, 4 agentes e 8 agentes), em que um conjunto de agentes coopera na busca da solução. Este item nos permitirá avaliar a eficácia da cooperação e a escalabilidade da estrutura usando a proposta ALS-LA;

(ii) Desempenho da proposta de aprendizado ALS-*QLearning*: análise específica dos resultados obtidos pela proposta de ALS-*QLearning*, com os mesmos objetivos do item anterior;

(iii) Comparação entre as duas propostas de aprendizado e o VND clássico: comparação das soluções obtidas pelas duas propostas de aprendizado (ALS-LA e ALS-*QLearning*) e pelo VND Clássico. O VND em sua forma clássica foi testado de duas maneiras para o VRPTW: com ordem crescente de vizinhanças, baseada na complexidade destas; e com ordem inversa à primeira, decrescente em relação à complexidade. Estas duas versões de teste do VND serão referenciadas a seguir como VND O1 (Ordem 1), referenciando a ordem crescente de complexidade das vizinhanças, e VND O2 (Ordem 2), referenciando a ordem decrescente de complexidade das vizinhanças. O UPMSP-ST foi testado apenas utilizando a ordem crescente de complexidade. As ordens de vizinhanças utilizadas para o VRPTW são:

- Ordem 1 (O1): *Intra-Route Swap*, *Inter-Route Swap*, *Intra-Route Shift*, *Inter-Route Shift*, *Two Intra-Route Swap*, *Two Intra-Route Shift*, *Eliminates Smaller Route*, *Eliminates Random Route*;
- Ordem 2 (O2): *Eliminates Random Route*, *Eliminates Smaller Route*, *Two Intra-Route Shift*, *Two Intra-Route Swap*, *Inter-Route Shift*, *Intra-Route Shift*, *Inter-Route Swap*, *Intra-Route Swap*;

(iv) Comparação do desempenho individual do agente com e sem aprendizagem: comparação do desempenho de um único agente usando cada uma das propostas (ALS-LA e ALS-*QLearning*) e um único agente usando o algoritmo VND clássico com as ordens de vizinhança já citadas;

(v) Cooperação: avaliação da cooperação usando como base a trajetória das soluções no *Pool* de Soluções.

Uma vez que os algoritmos utilizados são de natureza estocástica, cada um dos cenários avaliados nestes experimentos foi executado 30 vezes para cada instância. Os resultados obtidos em cada um desses cenários foram comparados utilizando um teste de hipóteses não paramétrico, com um nível de confiança de 95%. O teste não paramétrico utilizado foi o teste Kruskal-Wallis (Montgomery e Runger, 2013). Este teste verifica se existem diferenças entre as médias das populações de soluções avaliadas. Desta forma, as hipóteses devem ser formuladas da seguinte forma.

Considerando os cenários de teste analisados, as seguintes hipóteses são levantadas para comparar as soluções médias obtidas em cada um deles:

(i) Hipótese nula (H_0): a média das soluções obtidas nos cenários de teste analisados é igual. Se a hipótese nula não for rejeitada, não há diferença estatística significativa entre as soluções obtidas por estes cenários;

(ii) Hipótese alternativa (H_1): a média das soluções obtidas pelos cenários de teste analisados são diferentes. Se a hipótese nula for rejeitada, então há evidências estatísticas (com nível de confiança de 95%) que há diferença significativa entre as soluções obtidas por estes cenários.

O teste estatístico Kruskal-Wallis foi aplicado da seguinte forma:

(i) na comparação dos quatro cenários (1 agente, 2 agentes, 4 agentes e 8 agentes) de cada uma das propostas apresentadas (separadamente); e

(ii) na comparação de todos os algoritmos implementados (ALS-QL, ALS-LA, VND O1 e VND O2). Esta comparação buscou identificar se há diferença estatística entre as populações, ou seja, se a hipótese nula é rejeitada.

Após verificar se há diferença estatística em cada comparação realizada, o teste de Wilcoxon (Montgomery e Runger, 2013) foi utilizado para comparar pares de populações, buscando identificar se há diferença estatísticas, e consequentemente, qual a população obteve melhores resultados. Finalmente, gráficos de boxplot foram utilizados para demonstrar os resultados.

5.2 Resultados Computacionais e Discussão

Esta seção apresenta os resultados obtidos nas 30 execuções de cada algoritmo testado para cada problema instanciado, seguindo as diretrizes já descritas. Adicionalmente, esta seção apresenta a análise e discussão destes resultados.

Tabela 5.1: Número de vezes que cada cenário da proposta ALS-LA foi melhor do que os demais cenários para o VRPTW - valores obtidos pelo teste não paramétrico

Conjunto de Instâncias	Total de instâncias por conjunto	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
C1	**3**	2	2	2	3
C2	**3**	3	3	3	3
R1	**3**	0	1	2	3
R2	**3**	0	0	0	3
RC1	**3**	0	0	2	3
RC2	**3**	0	1	1	3
Total	**18**	**5**	**7**	**10**	**18**

Tabela 5.2: Número de vezes que cada cenário da proposta ALS-LA foi melhor do que os demais cenários para o UPMSP-ST - valores obtidos pelo teste não paramétrico

Conjunto de Instâncias	Total de instâncias por conjunto	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
50 tarefas - 10 máquinas	**4**	0	1	3	4
50 tarefas - 15 máquinas	**4**	0	2	2	4
50 tarefas - 20 máquinas	**4**	1	1	3	4
50 tarefas - 25 máquinas	**4**	0	1	2	4
Total	**16**	**1**	**5**	**10**	**16**

5.2.1 Proposta de Agente Adaptativo ALS-LA

A análise é iniciada apresentando os resultados obtidos nos quatro cenários da proposta ALS-LA para os dois problemas instanciados. As Tabelas 5.1 e 5.2 mostram o número de vezes em que cada cenário da proposta ALS-LA foi melhor que os demais cenários da mesma, para as instâncias VRPTW e UPMSP-ST, respectivamente, com base nos resultados do teste não paramétrico utilizado. O objetivo aqui é avaliar a escalabilidade da proposta, ou seja, se o aumento do número de agentes envolvidos na solução influencia o desempenho do sistema multiagente.

Para as instâncias VRPTW, os resultados obtidos mostraram que há evidências estatísticas que, em $100,00\%$ das instâncias, os cenários com 2 ou mais agentes foram melhores que o cenário com 1 único agente. Isso considerando que, em 5 das instâncias testadas, as melhores soluções foram encontradas em todas as 30 execuções dos 4 cenários; sendo assim, elas foram retiradas do cálculo da porcentagem.

Por sua vez, para as instâncias UPMSP-ST, os resultados obtidos mostraram que há evidências estatísticas que o uso de 2 ou mais agentes foi melhor em $93,75\%$ das instâncias. Observa-se, nas Tabelas 5.1 e 5.2, que o número de vezes em que cada cenário é melhor cresce com o número de agentes. Estas tabelas confirmam que, quando o número de agentes usados para resolver os dois problemas é aumentado, a função de avaliação diminui também para os dois problemas.

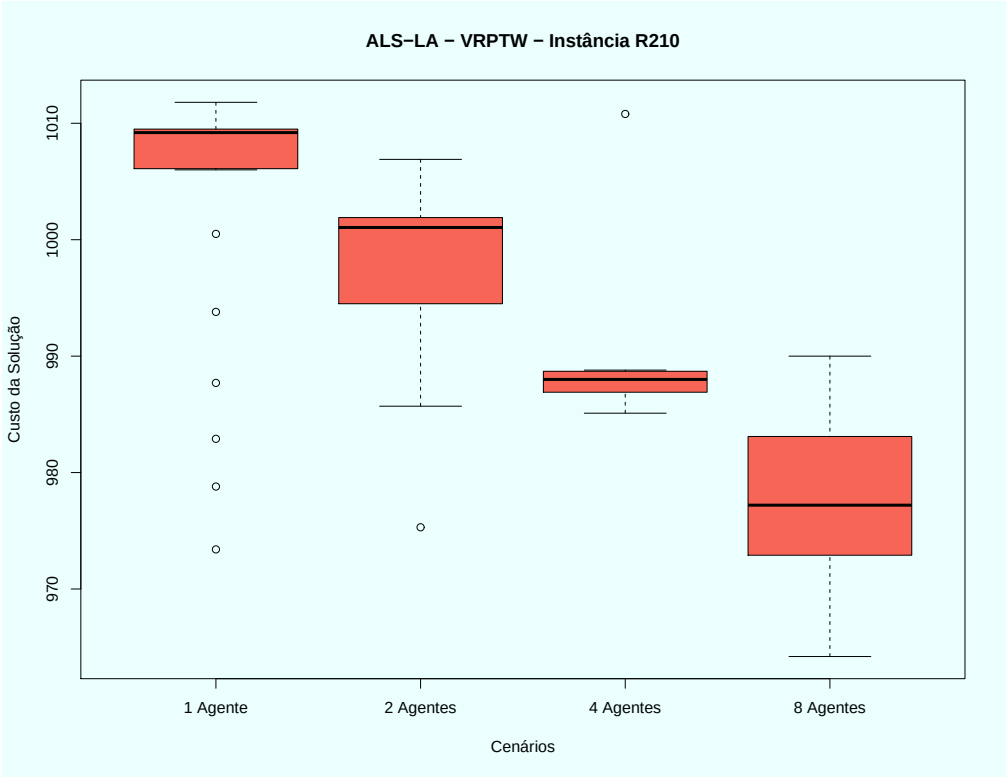


Figura 5.1: Comparação dos cenários da proposta ALS-LA para o VRPTW em relação à distância viajada - instância R210.

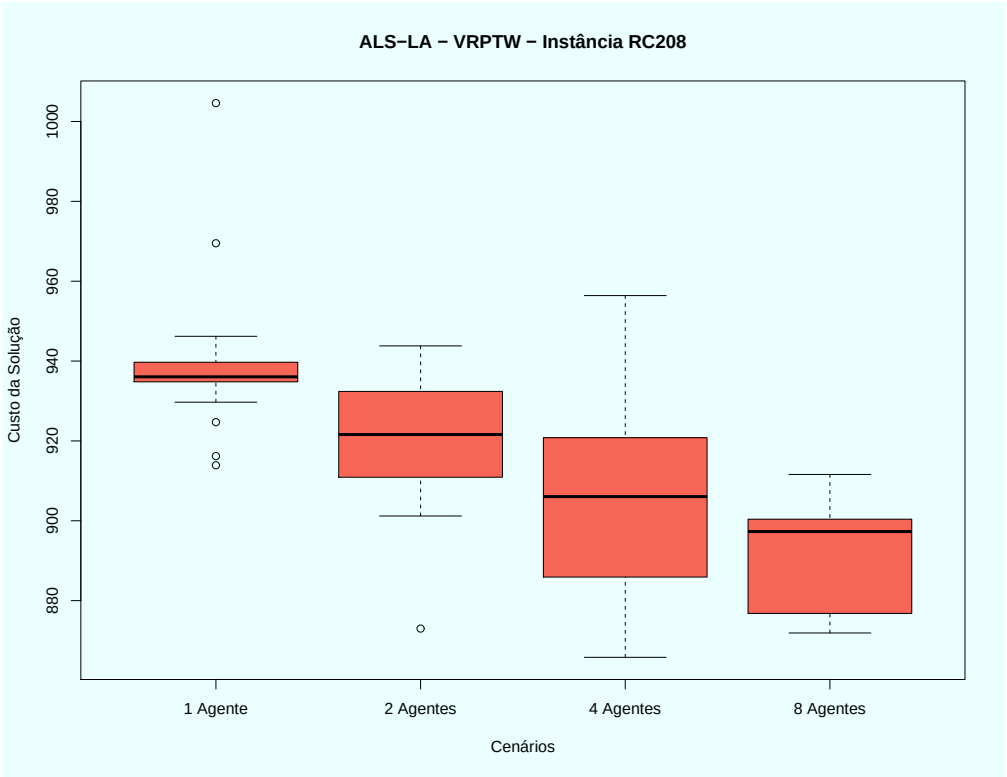


Figura 5.2: Comparação dos cenários da proposta ALS-LA para o VRPTW em relação à distância viajada - instância RC208.

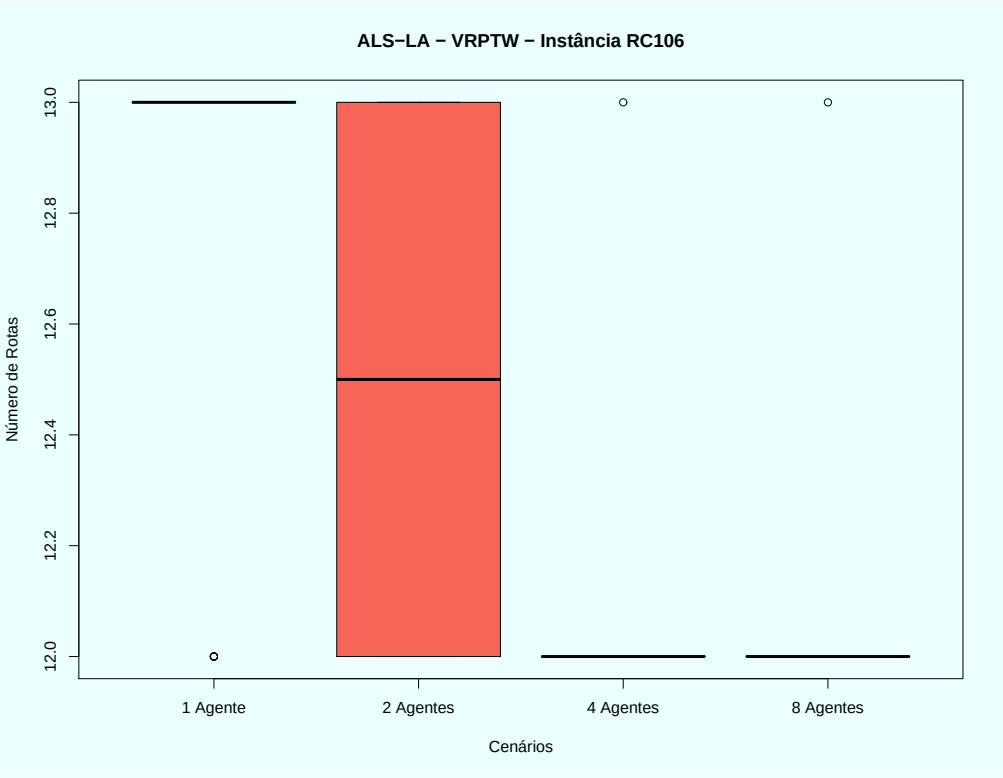


Figura 5.3: Comparação dos cenários da proposta ALS-LA para o VRPTW em relação ao número de rotas - instância RC106.

As Figuras 5.1, 5.2 e 5.3 usam gráficos boxplot para ilustrar a melhora na qualidade da solução com a adição de agentes no processo de solução, através dos resultados obtidos nas 30 execuções das instâncias R210, RC208 e RC106 do VRPTW, respectivamente. Observa-se, nos gráficos das Figuras 5.1 e 5.2, que a distância percorrida diminui à medida que o número de agentes aumenta. O gráfico da Figura 5.3 mostra como o número de rotas também reduz à medida que o número de agentes aumenta. O mesmo resultado pode ser observado nos gráficos de boxplot apresentados nas Figuras 5.4, 5.5 e 5.6, usados para ilustrar os resultados obtidos nas 30 execuções das instâncias I_50_10_S_1-49_1, I_50_10_S_1-99_1 e I_50_15_S_1-9_1, respectivamente, do UPMSP-ST. Estes exemplos demonstram como o *makespan* diminui à medida que o número de agentes aumenta.

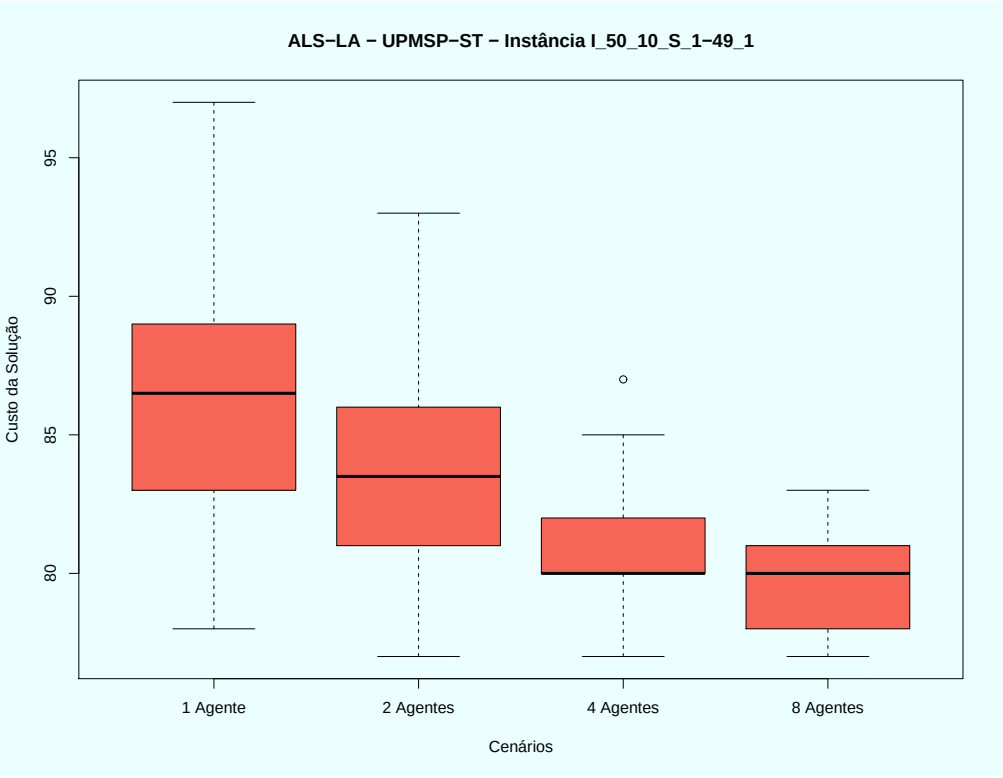


Figura 5.4: Comparação dos cenários da proposta ALS-LA para o UPMSP-ST em relação ao *makespan* - instância I_50_10_S_1-49_1.

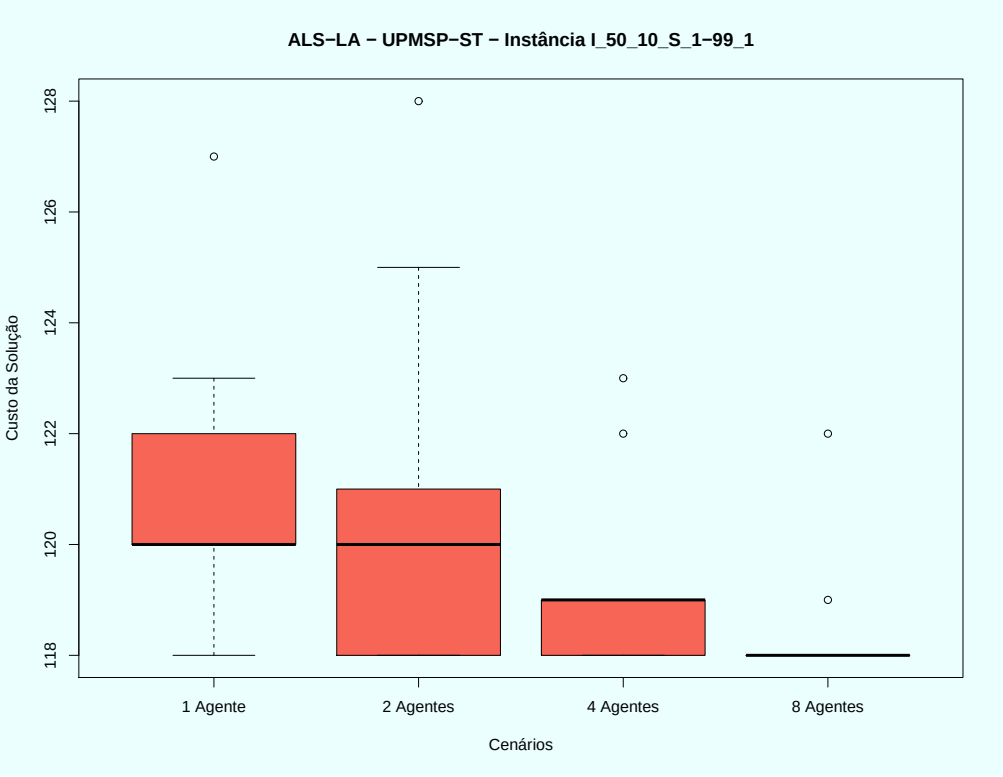


Figura 5.5: Comparação dos cenários da proposta ALS-LA para o UPMSP-ST em relação ao *makespan* - instância I_50_10_S_1-99_1.

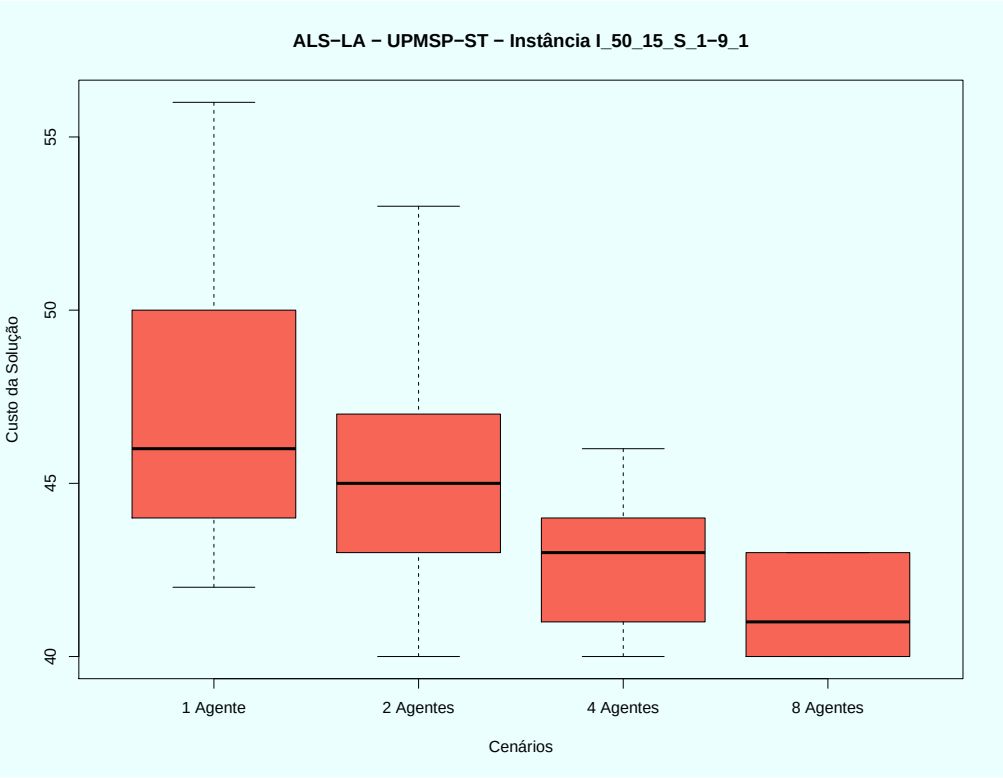


Figura 5.6: Comparação dos cenários da proposta ALS-LA para o UPMSP-ST em relação ao *makespan* - instância I_50_15_S_1-9_1.

É importante destacar que, em alguns casos, os cenários estão vinculados às melhores soluções para as instâncias, ou seja, não houve diferença estatística entre as médias das soluções. Nestes casos, todos os cenários que correspondiam aos melhores resultados foram contados nas Tabelas 5.1 e 5.2, embora eles tenham sido retirados do cálculo das porcentagens.

5.2.2 Proposta de Agente Adaptativo ALS-*QLearning*

Nesta seção, o foco é dado aos resultados obtidos pela aplicação da proposta ALS-*QLearning* para os dois problemas instanciados.

Em relação ao VRPTW, assim como na proposta ALS-LA, das 18 instâncias analisadas, em 5 delas, a proposta do ALS-*QLearning* obteve a melhor solução da literatura em todos os cenários (incluindo o cenário com um único agente) nas 30 execuções. Portanto, nesses casos, não houve diferença estatística para comparação e, consequentemente, foram excluídos desta análise. Para as instâncias restantes, há evidências estatísticas de que, em $92,30\%$ delas, os cenários com 2 ou mais agentes foram melhores que o cenário com 1 único agente.

Em relação ao UPMSP-ST, das 16 instâncias analisadas, em 2 delas, a proposta do ALS-*QLearning* obteve a melhor solução da literatura em todos os cenários (incluindo o cenário com um único agente) nas 30 execuções. Portanto, assim como no VRPTW, não houve diferença estatística para comparação e, consequentemente, foram excluídos desta análise. Das 14 instâncias restantes, há evidências estatísticas que, em $92,85\%$, os cenários com 2 ou mais agentes foram melhores que o cenário com 1 agente único.

Tabela 5.3: Número de vezes que cada cenário da proposta ALS-*QLearning* foi melhor que os demais cenários para o VRPTW - valores obtidos pelo teste não paramétrico.

Conjunto de Instâncias	Total de instâncias por conjunto	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
C1	**3**	2	3	3	3
C2	**3**	3	3	3	3
R1	**3**	1	1	1	3
R2	**3**	0	0	0	3
RC1	**3**	0	0	2	3
RC2	**3**	0	1	1	3
Total	**18**	**6**	**8**	**10**	**18**

Tabela 5.4: Número de vezes que cada cenário da proposta ALS-*QLearning* foi melhor do que os demais cenários para o UPMSP-ST - valores obtidos pelo teste não paramétrico

Conjunto de Instâncias	Total de instâncias por conjunto	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
50 tarefas - 10 máquinas	**4**	0	0	2	4
50 tarefas - 15 máquinas	**4**	1	2	3	4
50 tarefas - 20 máquinas	**4**	1	1	3	4
50 tarefas - 25 máquinas	**4**	1	1	3	4
Total	**16**	**3**	**4**	**11**	**16**

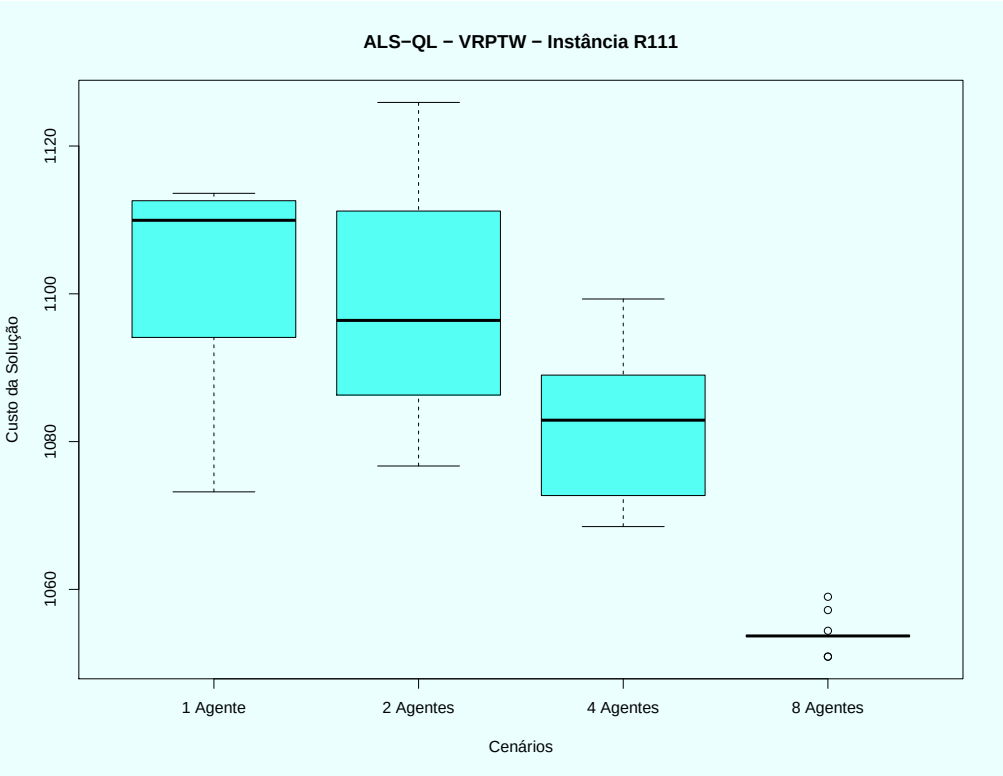


Figura 5.7: Comparação dos cenários da proposta ALS-*QLearning* para o VRPTW em relação à distância viajada - instância R111.

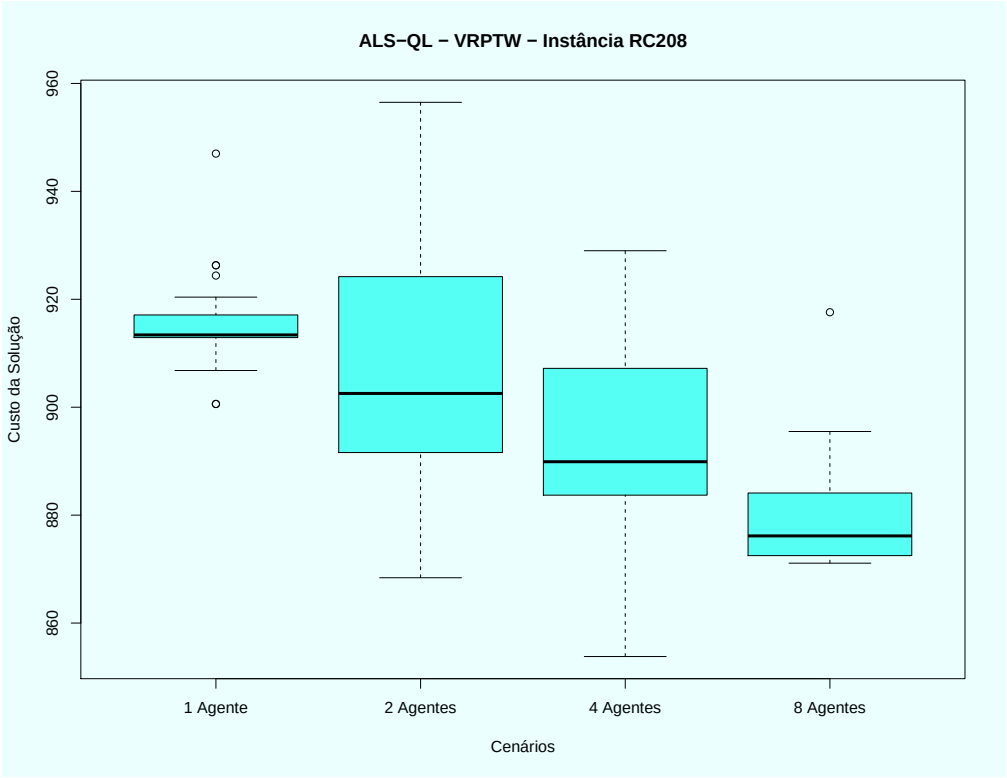


Figura 5.8: Comparação dos cenários da proposta ALS-*QLearning* para o VRPTW em relação à distância viajada - instância RC208.

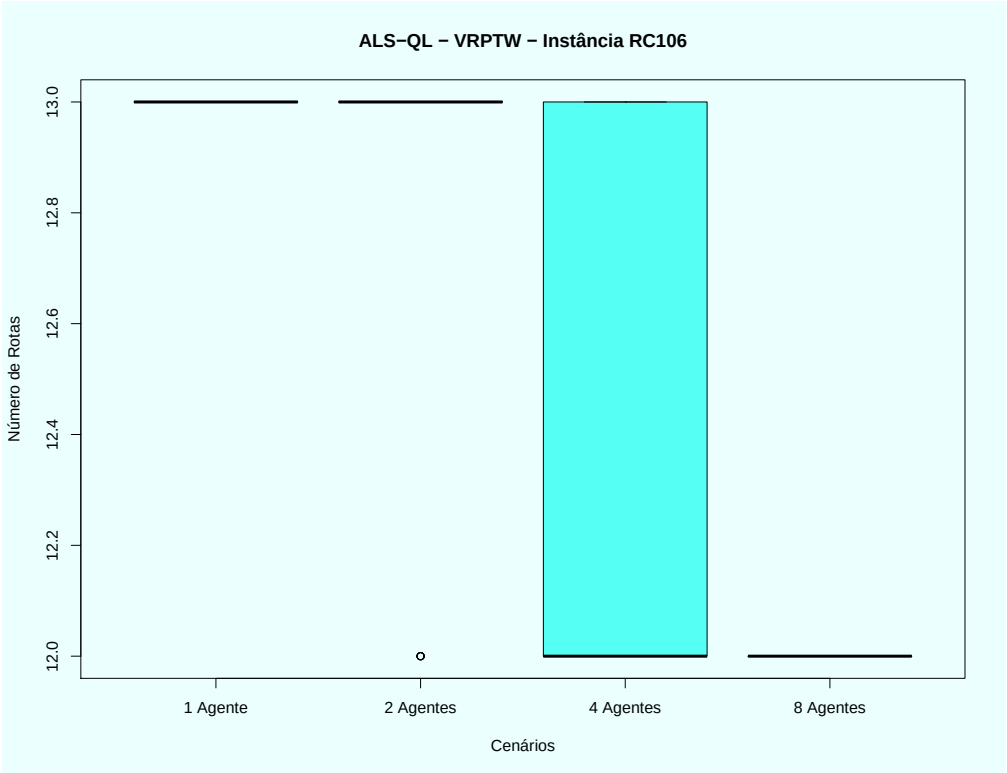


Figura 5.9: Comparação dos cenários da proposta ALS-*QLearning* para o VRPTW em relação ao número de rotas - instância RC106.

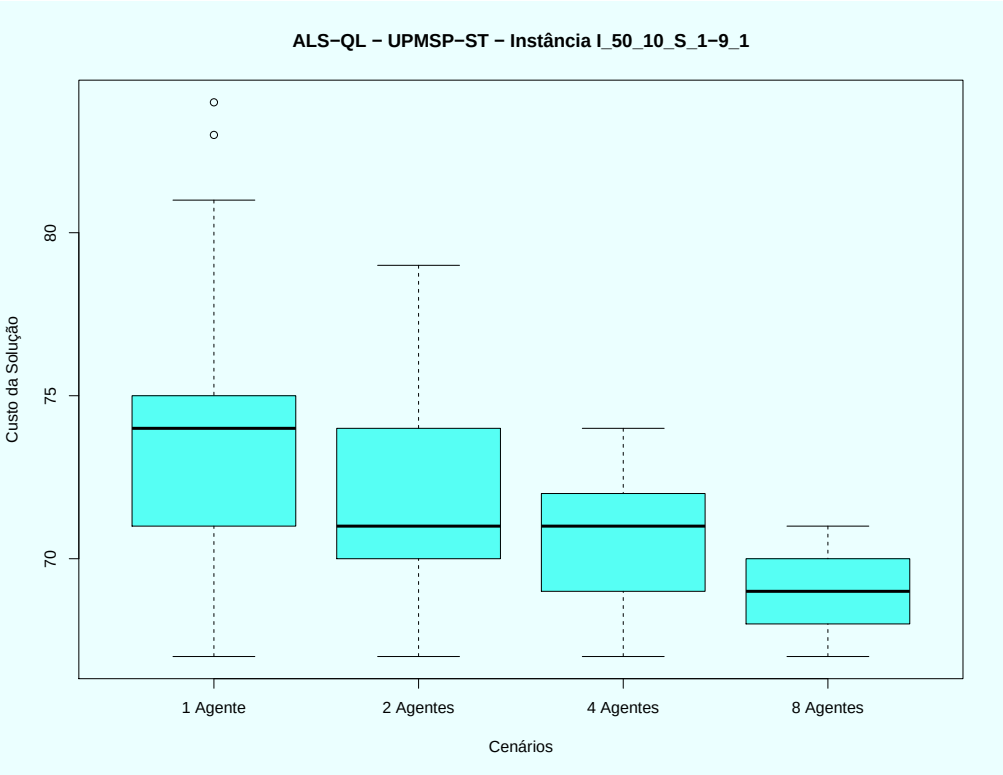


Figura 5.10: Comparação dos cenários da proposta ALS-*QLearning* para o UPMSP-ST em relação ao *makespan* - instância I_50_10_S_1-9_1.

As Tabelas 5.3 e 5.4 mostram o número de vezes que cada cenário da proposta ALS-*QLearning* foi melhor, para as instâncias VRPTW e UPMSP-ST, respectivamente, com base nos resultados do teste não paramétrico utilizado. Como na Seção 5.2.1, o objetivo é avaliar a escalabilidade da proposta. De maneira semelhante aos resultados mostrados nas Tabelas 5.1 e 5.2, nas Tabelas 5.3 e 5.4, conclui-se que cresce o número de vezes que cada cenário é melhor, na medida em que o número de agentes também cresce.

As Figuras 5.7, 5.8 e 5.9 mostram exemplos do efeito da adição de agentes no processo de solução. Estas figuras exibem as soluções das instâncias R111, RC208 e RC106, respectivamente, para o VRPTW. As Figuras 5.7 e 5.8 apresentam a análise do custo da solução em relação à distância percorrida, que se reduz na medida em que cresce o número de agentes. A Figura 5.9 apresenta o número de rotas obtido nas execuções da instância RC106. Observa-se que o número de rotas é também reduzido com o uso de recursos do ambiente cooperativo. O cenário com 8 agentes consegue reduzir o número de rotas em todas as execuções.

As Figuras 5.10, 5.11 e 5.12 também mostram exemplos do efeito da adição de agentes na qualidade das soluções para as instâncias I_50_10_S_1-9_1, I_50_15_S_1-9_1 e I_50_25_S_1-9_1 do UPMSP-ST, respectivamente. Estes gráficos revelam a melhora das soluções na medida em que o número de agentes cresce.

Conforme afirmado em relação aos resultados da proposta de ALS-LA, para o VRPTW, os resultados obtidos em alguns cenários da proposta de ALS-*QLearning* também estão atrelados aos melhores resultados da literatura. Nestes casos, como na análise da proposta de ALS-LA, todos os cenários que correspondiam aos melhores resultados foram contados nas Tabelas 5.3 e 5.4.

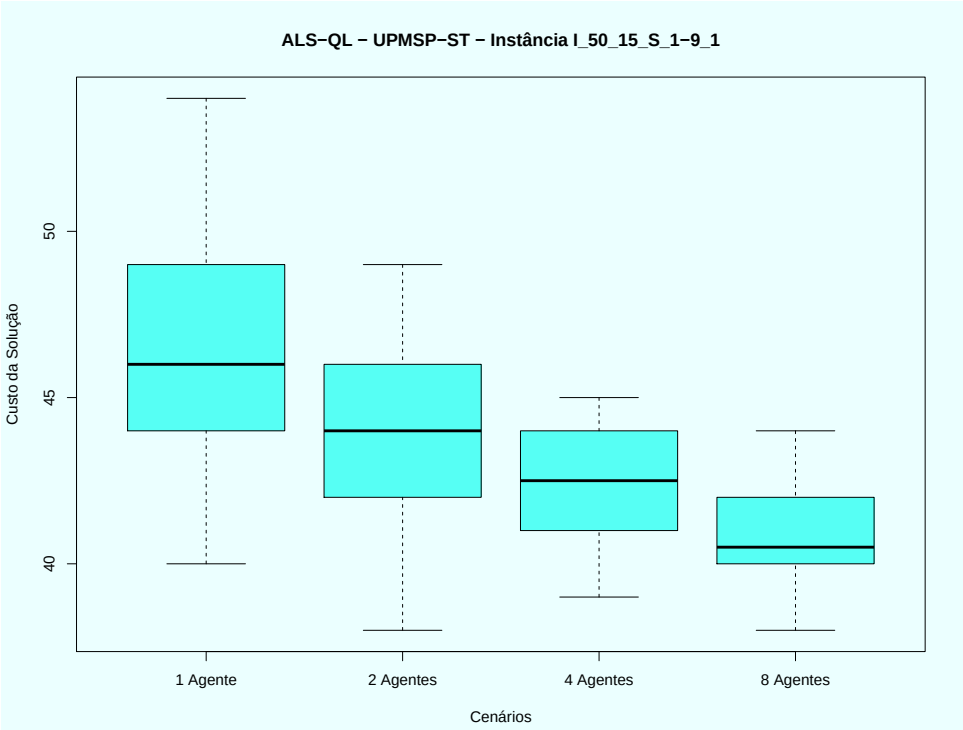


Figura 5.11: Comparação dos cenários da proposta ALS-*QLearning* para o UPMSP-ST em relação ao *makespan* - instância I_50_15_S_1-9_1.

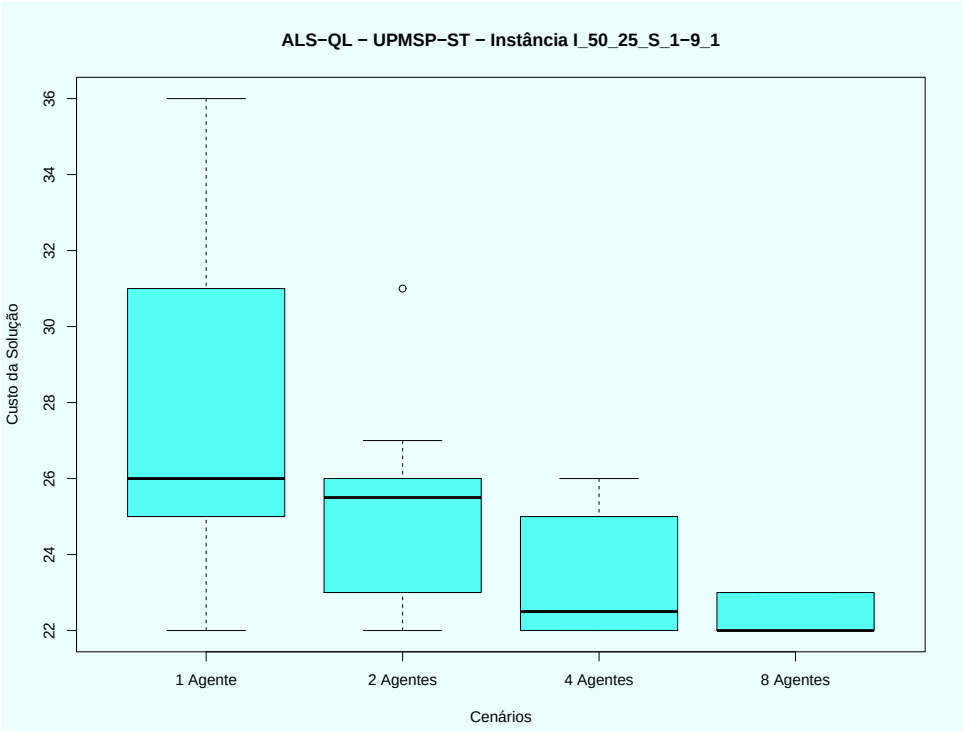


Figura 5.12: Comparação dos cenários da proposta ALS-*QLearning* para o UPMSP-ST em relação ao *makespan* - instância I_50_25_S_1-9_1.

5.2.3 ALS-LA × ALS-*QLearning* × VND Clássico

Esta seção apresenta a comparação entre as soluções encontradas para as propostas ALS-LA e ALS-*QLearning* e para o algoritmo VND clássico, avaliados neste experimento, com base nos resultados do teste não paramétrico utilizados. Essa comparação abordou os dois problemas instanciados. As Seções 5.2.3.1 e 5.2.3.2 mostram a análise desses resultados.

5.2.3.1 VRPTW

O custo de uma solução para VRPTW é calculado de acordo com a Expressão (3.1). Nesta expressão, a prioridade é minimizar o número de rotas da solução, ou seja, o número de veículos que são usados. Em alguns casos, as soluções geradas nos cenários avaliados podem ter diferentes números de rotas. Se o número de rotas for diferente, não é possível comparar o custo dessas soluções. Nestes casos, a comparação é feita através do número de rotas. As instâncias que obtiveram o mesmo número de rotas foram comparadas em relação à distância percorrida. O mesmo acontece para as instâncias que obtiveram números diferentes de rotas, mas não apresentaram diferença estatística nestes valores. Neste caso, as soluções com número de rotas que não influenciam o resultado foram removidas e, portanto, a análise estatística foi feita com as demais soluções, em relação à distância percorrida.

A primeira comparação realizada entre as propostas apresentadas aqui (ALS-LA e ALS-*QLearning*) e o VND Clássico, avalia o desempenho dos agentes em equipe, ou seja, os cenários com 2 ou mais agentes, para o VRPTW.

Neste contexto, o número de vezes que cada uma das versões de Busca Local implementadas obteve o melhor resultado com 2 ou mais agentes foi analisado. Caso o melhor resultado seja obtido por mais de um Algoritmo de Busca Local, ou seja, quando não há diferença estatística entre eles, todos os Algoritmos que obtiveram estes melhores resultados foram contados. A Tabela 5.5 mostra estes valores referentes às 16 instâncias testadas do VRPTW. Como pode ser observado, a proposta ALS-*QLearning* obteve um maior número de melhores resultados em relação às demais. A proposta ALS-*QLearning* obteve o melhor resultado em $92,30\%$ destas instâncias, sendo que, em $53,84\%$, a proposta ALS-*QLearning* obteve o melhor resultado entre todos os algoritmos. Esta porcentagem excluiu as instâncias em que todos os algoritmos obtiveram a melhor solução conhecida em todos os cenários.

A proposta ALS-LA se destacou na maioria das instâncias em que foi avaliado o número de rotas, tendo comportamento similar à proposta ALS-*QLearning*. Nestes casos, as duas propostas obtiveram os melhores resultados.

A Figura 5.13 ilustra a situação em que a proposta ALS-*QLearning* alcança resultados melhores do que a implementação clássica do VND utilizando as duas ordens de vizinhança testadas. Nesta figura, os resultados da instância RC208 mostram que, para cada cenário (1, 2, 4 e 8 agentes), a proposta ALS-*QLearning* obtém soluções melhores que as duas versões clássicas do VND. Neste caso específico, nos valores encontrados com 8 agentes, há uma redução na distância percorrida de $889,9$, obtida pelo VND O1, e $889,9$, obtida pelo VND O2, para o valor de $871,10$ obtida pelo ALS-*QLearning*.

O objetivo dos testes com o VND utilizando duas ordens de vizinhança (VND O1 e VND O2) é demonstrar a diferença de soluções obtidas, para cada ordem, de acordo com a instância utilizada e verificar a adaptabilidade das propostas apresentadas em relação a ordens de vizinhança tão diferentes. Neste sentido, a Figura 5.14 ilustra a situação em

Tabela 5.5: Número de vezes em que cada algoritmo implementado obteve o melhor resultado nos cenários com 2 ou mais agentes para o VRPTW - valores obtidos pelo teste não paramétrico

Conjunto de Instâncias	Total de Instâncias por conjunto	Total de Instâncias por conjunto com diferentes número de rotas	Buscas Locais			
			ALS-LA	**ALS-QL**	**VND O1**	**VND O2**
C1	**3**	**0**	2	3	3	2
C2	**3**	**0**	3	3	3	3
R1	**3**	**1**	2	3	1	2
R2	**3**	**0**	0	3	0	0
RC1	**3**	**3**	2	3	0	1
RC2	**3**	**1**	2	2	1	1
Total de Instâncias	**18**	**5**	**11**	**17**	**8**	**9**

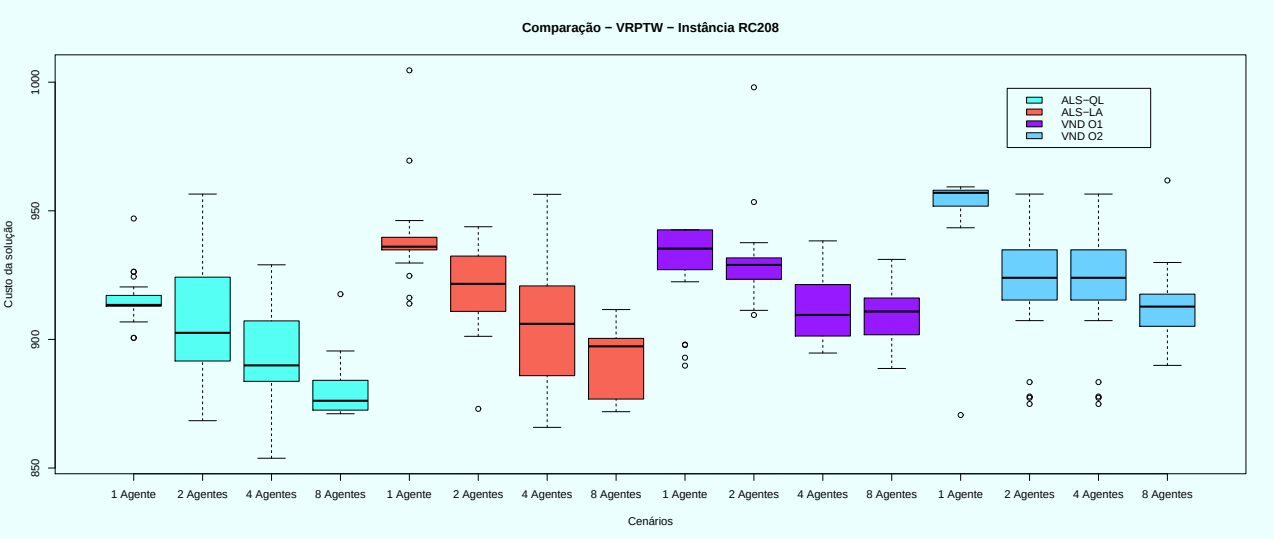


Figura 5.13: Comparação entre as propostas (ALS-LA e ALS-*QLearning*) e o VND clássico com as duas ordens de vizinhança (VND-O1 e VND-O2) em relação a distância viajada - VRPTW - intância RC208

que a versão clássica do VND com a ordem 1 (O1), ordem mais utilizada atualmente, é a única versão a não conseguir reduzir o número de rotas em todos os cenários. A ordem 2 (O2) obtém número de rotas melhores nos cenários com 4 e 8 agentes. Neste caso, a proposta adaptativa ALS-LA consegue reduzir, em todas as execuções, o número de rotas nos cenários com 4 e 8 agentes. A Figura 5.15 mostra mais um exemplo em que os valores das soluções do ALS-*QLearning* são melhores do que os demais algoritmos testados.

O desempenho do aprendizado individual também foi avaliado. A Tabela 5.6 apresenta esta análise. Como pode ser observado, a proposta ALS-*QLearning* obteve um maior número de melhores resultados em relação às demais no que diz respeito à utilização de um único agente. A proposta ALS-*QLearning* obteve melhor resultado em $84,61\%$ das instâncias. Diferentemente do resultado obtido no trabalho em equipe (2 ou mais agentes), com um único agente as duas propostas aqui apresentadas (ALS-LA e ALS-*QLearning*) tiveram desempenho semelhante, obtendo, em $61,53\%$ das instâncias, melhores resultados que as duas versões do VND testadas. Esta porcentagem excluiu as instâncias em que todos os algoritmos obtiveram a melhor solução conhecida para o cenário com 1 agente.

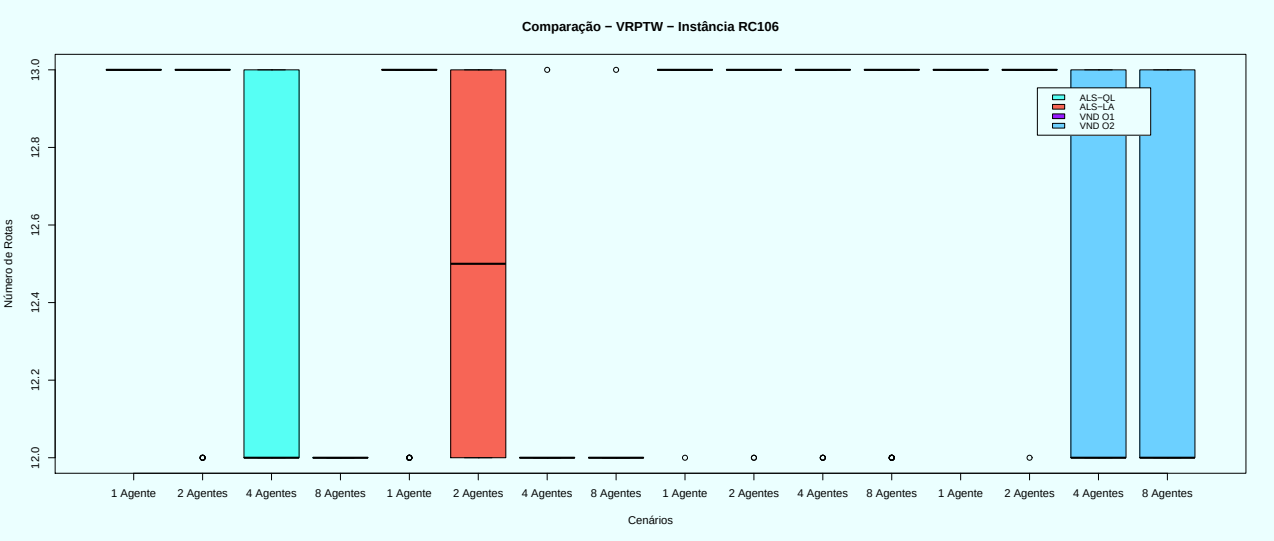


Figura 5.14: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico com as duas ordens de vizinhança (VND-O1 e VND-O2) em relação ao número de rotas - VRPTW - RC106

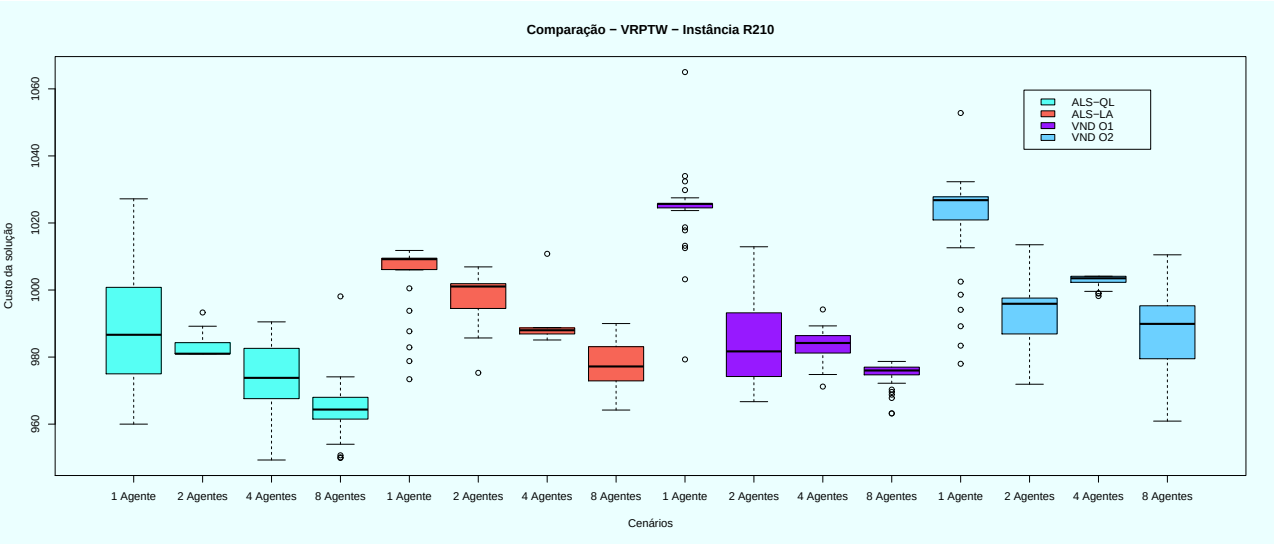


Figura 5.15: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico com as duas ordens de vizinhança (VND-O1 e VND-O2) em relação a distância viajada - VRPTW - R210.

Tabela 5.6: Número de vezes em que cada Algoritmo implementado obteve o melhor resultado no cenário com um único agente para o VRPTW - valores obtidos pelo teste não paramétrico.

Conjunto de Instâncias	Total de Instâncias por conjunto	Total de Instâncias por conjunto com diferentes número de rotas	**Buscas Locais**			
			ALS-LA	**ALS-QL**	**VND O1**	**VND O2**
C1	**3**	**0**	2	3	2	2
C2	**3**	**0**	3	3	3	3
R1	**3**	**1**	3	3	0	1
R2	**3**	**0**	1	2	1	0
RC1	**3**	**3**	2	3	2	2
RC2	**3**	**1**	2	2	0	2
Total de instâncias	**18**	**5**	**13**	**16**	**8**	**10**

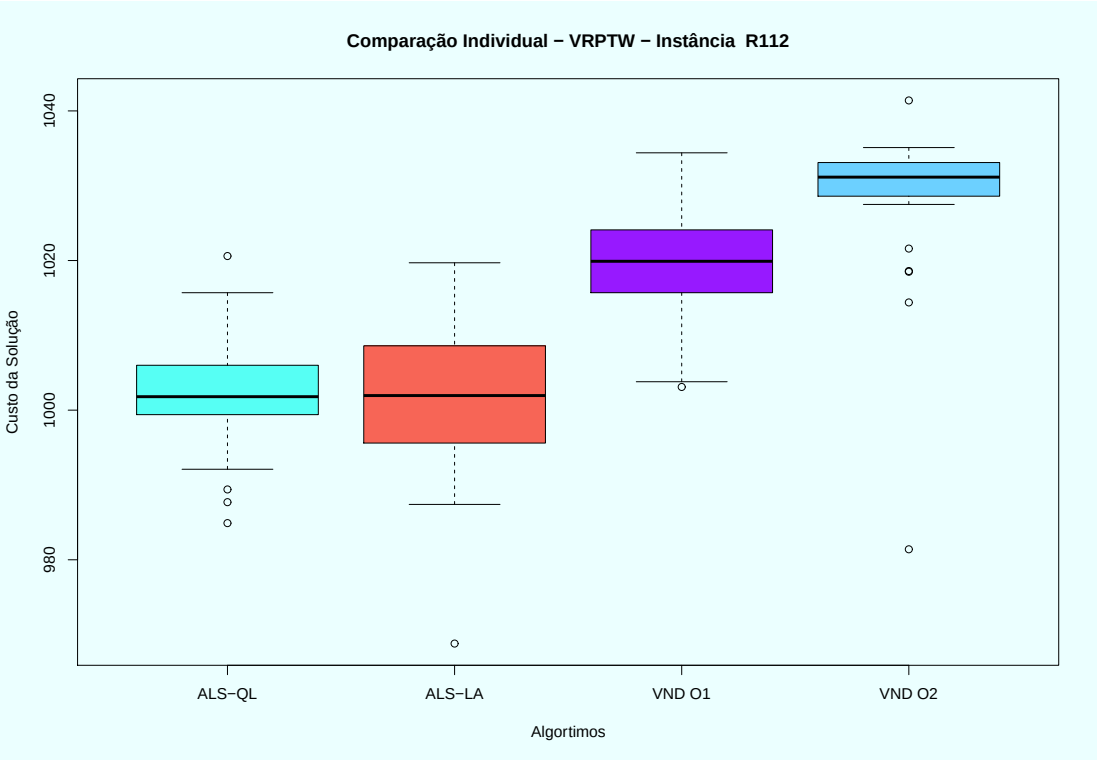


Figura 5.16: Comparação do desempenho individual das duas propostas de aprendizado com o VND clássico com duas ordens de vizinhanças diferentes para a instância R112.

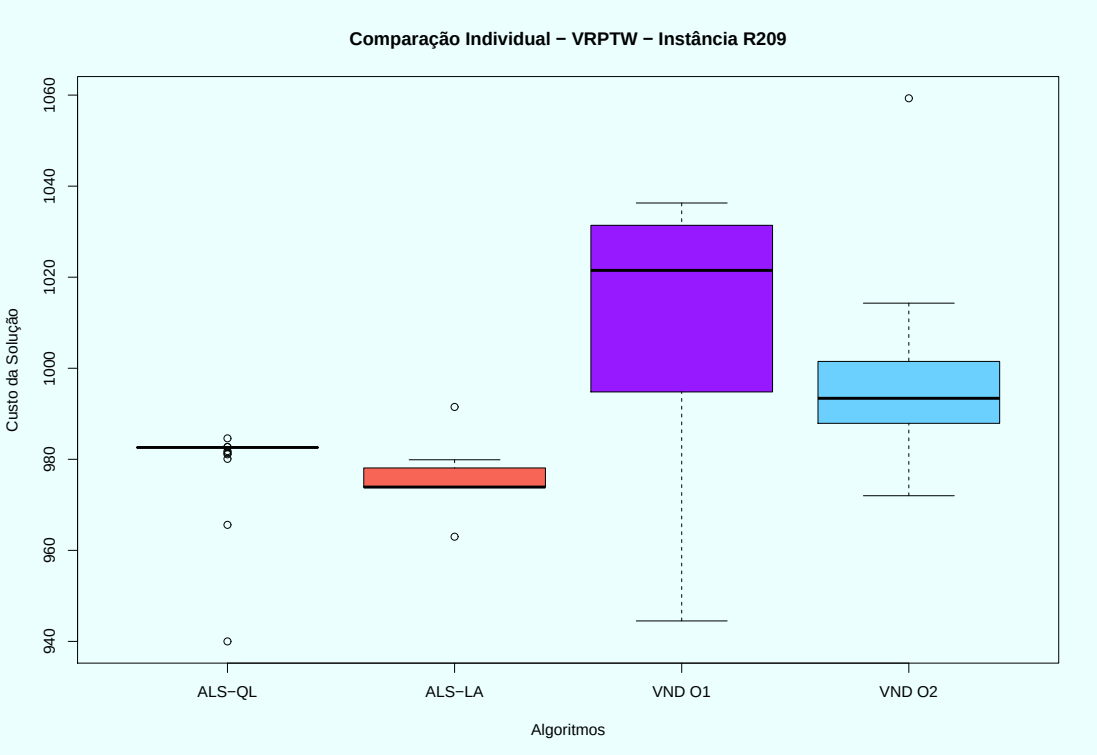


Figura 5.17: Comparação do desempenho individual das duas propostas de aprendizado com o VND clássico com duas ordens de vizinhanças diferentes para a instância R209.

A comparação das propostas apresentadas aqui com o VND clássico permite avaliar o comportamento do agente no ambiente multiagente do *framework* com aprendizado e sem aprendizado. Os resultados apresentados indicam, com base no teste não paramétrico utilizado, que existe evidência estatística de que a incorporação de aprendizado no agente melhora a qualidades das soluções para o VRPTW. É importante ressaltar que em nenhum dos testes realizados as propostas ALS-LA e ALS-*QLearning* obtiveram soluções inferiores às obtidas pelo VND nas versões testadas, o que indica que as duas propostas de aprendizado se adaptam bem ao problema, não sendo necessário definir a ordem de aplicação das vizinhanças para a busca local.

As Figuras 5.16, 5.17, 5.18 e 5.19 apresentam exemplos do comparativo entre as quatro versões de Busca Local para o cenário com um único agente.

A Figura 5.16 apresenta os valores obtidos nas 30 execuções do cenário com um agente para as 4 versões de busca local testadas para a instância R112. Neste caso, os melhores valores são obtidos pelas duas propostas apresentadas neste trabalho (ALS-LA e ALS-*QLearning*). As duas propostas obtiveram valores que não apresentam diferença estatística entre elas. Na Figura 5.17 é possível observar que os valores atingidos pela proposta ALS-LA são melhores do que os encontrados pelas outras versões, se considerar um único agente, para a instância R209. Da mesma forma, nas Figuras 5.18 e 5.19 é possível observar que os valores atingidos pela proposta ALS-*QLearning* são melhores do que os encontrados pelas demais versões, se considerar um único agente, para as instâncias R210 e RC108, respectivamente.

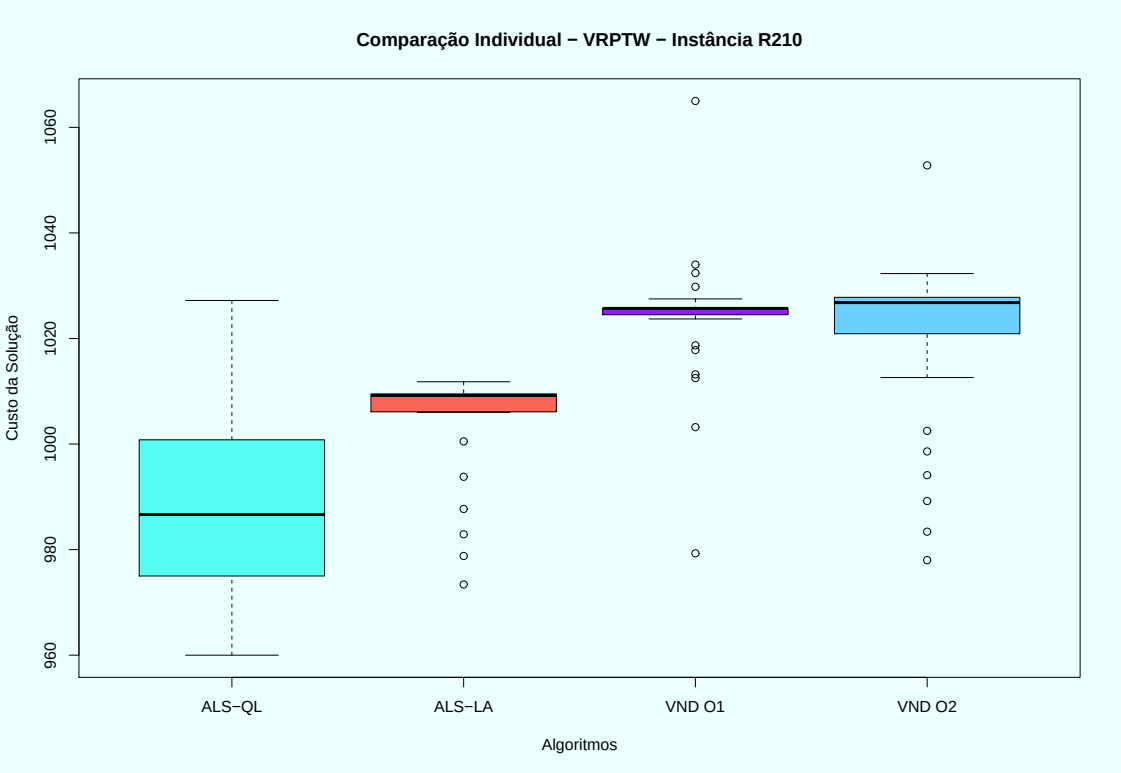


Figura 5.18: Comparação do desempenho individual das duas propostas de aprendizado com o VND clássico com duas ordens de vizinhanças diferentes para a instância R210.

É importante notar como os valores atingidos pelo ALS-*QLearning* são melhores do que os demais algoritmos, se considerarmos um único agente ou dois ou mais agentes.

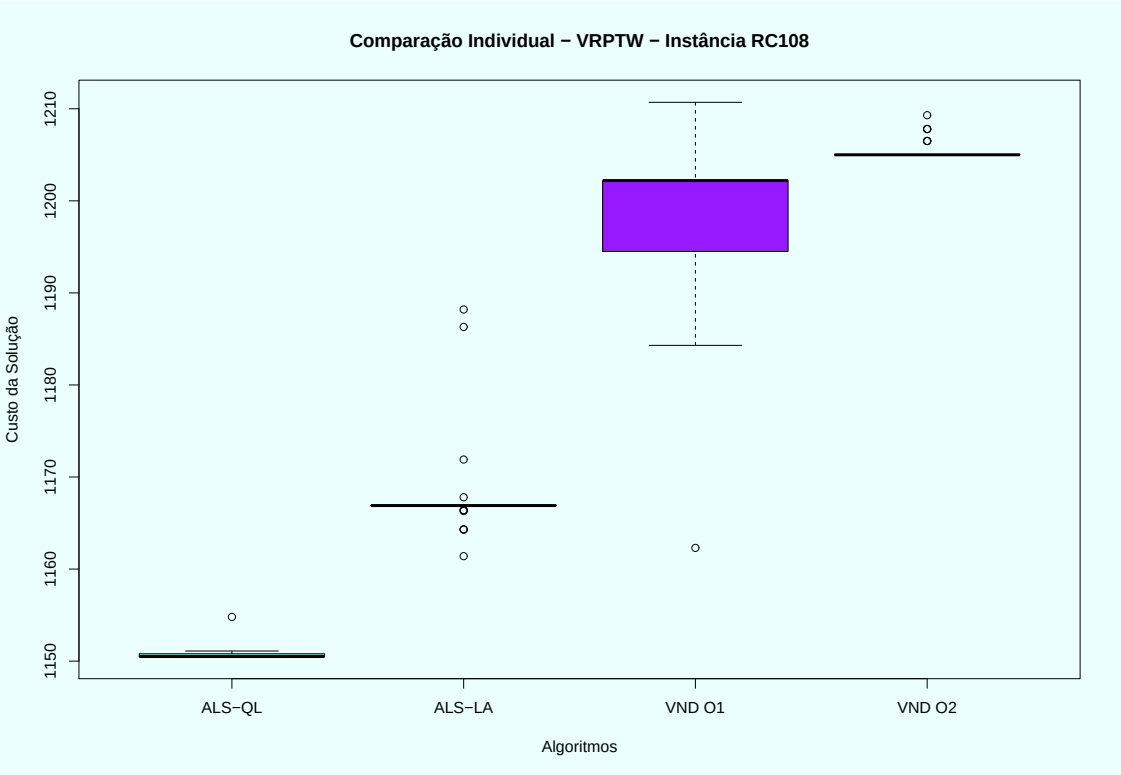


Figura 5.19: Comparação do desempenho individual das duas propostas de aprendizado com o VND clássico com duas ordens de vizinhanças diferentes para a instância RC108.

Adicionalmente, os valores melhoram de acordo com que o número de agentes envolvidos na busca pela solução cresce, o que demonstra a escalabilidade do *framework*.

5.2.3.2 UPMSP-ST

A comparação entre as propostas ALS-LA e ALS-*QLearning* e o VND Clássico é também realizada para o UPMSP-ST.

Nesse contexto, o número de vezes em que cada um dos algoritmos de Busca Local implementados obteve o melhor resultado com 2 ou mais agentes foi analisado para este problema. Assim como na seção anterior, caso o melhor resultado seja obtido por mais de um Algoritmo de Busca Local, ou seja, quando não há diferença estatística entre eles, todos os Algoritmos que obtiveram estes melhores resultados foram contados.

A Tabela 5.7 mostra os valores referentes ao UPMSP-ST, considerando os cenários com 2 ou mais agentes. Como pode ser visto, a proposta do ALS-*QLearning* obteve melhores resultados na maioria das vezes para as instâncias analisadas. Em $93,75\%$ de todas as instâncias, a proposta ALS-*QLearning* obteve o melhor resultado para os cenários com 2 ou mais agentes, sendo que, em $37,5\%$, ela obteve o melhor resultado entre todos os algoritmos testados.

As Figuras 5.20 e 5.21 mostram exemplos em que os valores de soluções da proposta ALS-*QLearning* são melhores que o VND clássico, para cada cenário.

O desempenho do aprendizado individual também foi avaliado para o UPMSP-ST. A Tabela 5.8 apresenta esta análise. Neste caso, a proposta ALS-*QLearning* alcançou melhor desempenho entre os algoritmos avaliados. A proposta ALS-LA obteve o melhor

Tabela 5.7: Número de vezes em que cada algoritmo implementado obteve o melhor resultado nos cenários com 2 ou mais agentes para o UPMSP-ST - valores obtidos pelo teste não paramétrico

Conjunto de Instâncias	Total de Instâncias por conjunto	Cenários		
		ALS-LA	ALS-QL	VND
50 tarefas - 10 máquinas	**4**	1	4	1
50 tarefas - 15 máquinas	**4**	3	4	2
50 tarefas - 20 máquinas	**4**	2	4	2
50 tarefas - 25 máquinas	**4**	2	3	1
Total	**16**	**8**	**15**	**6**

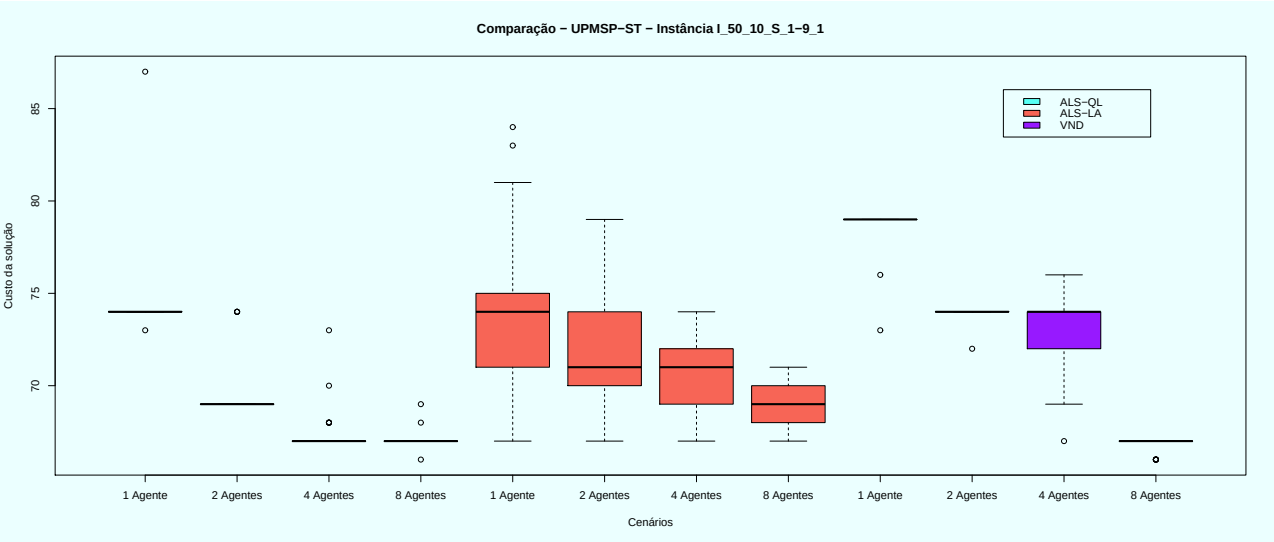


Figura 5.20: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico em relação ao *makespan* - instância I_100_10_S_1-9_1

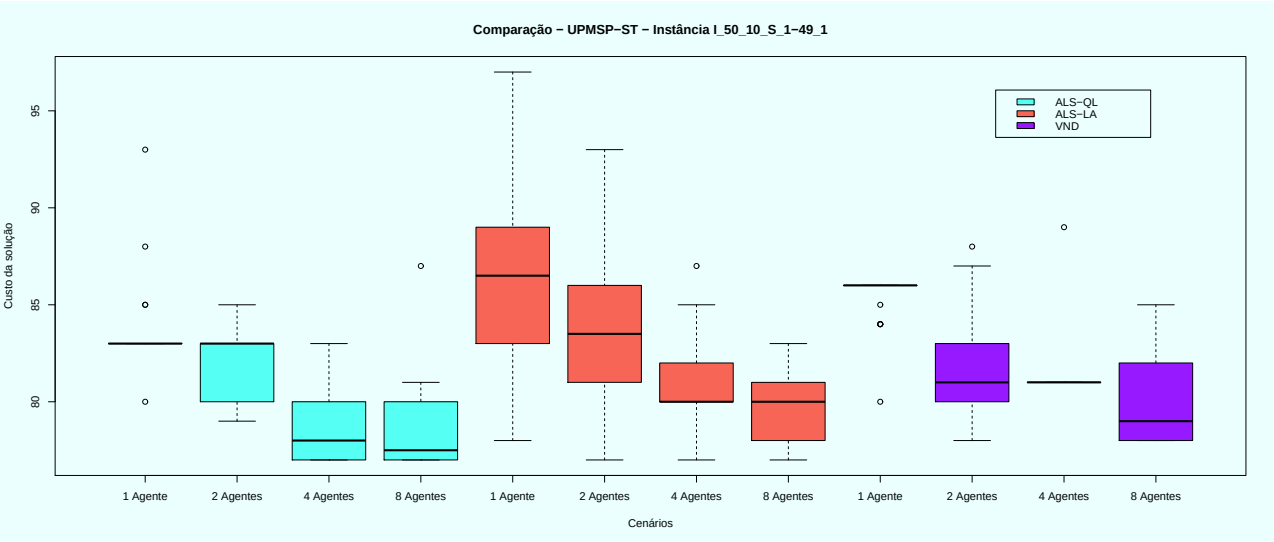


Figura 5.21: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico em relação ao *makespan* - instância I_100_10_S_1-49_1

Tabela 5.8: Número de vezes em que cada Algoritmo implementado obteve o melhor resultado no cenário com um único agente para o UPMSP-ST - valores obtidos pelo teste não paramétrico.

Conjunto de instâncias	Total de instâncias por conjunto	Cenário		
		ALS-LA	ALS-QL	VND
50 tarefas - 10 máquinas	**4**	2	4	1
50 tarefas - 15 máquinas	**4**	1	4	0
50 tarefas - 20 máquinas	**4**	2	4	2
50 tarefas - 25 máquinas	**4**	3	3	2
Total	**16**	**8**	**15**	**5**

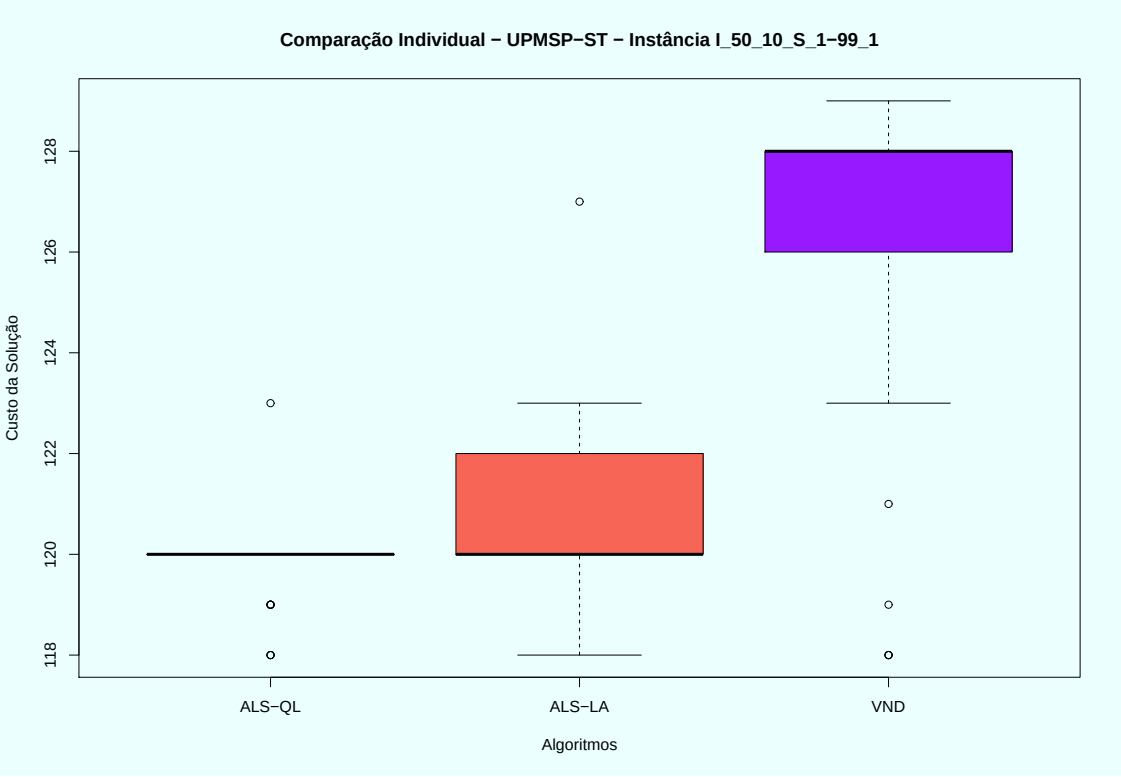


Figura 5.22: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico em relação ao *makespan* para os cenários com um único agente - instância I_100_10_S_1-99_1

resultado entre todos os algoritmos testados em 50% das instâncias testadas e a proposta ALS-*QLearning* obteve o melhor resultado entre todos os algoritmos testados em $93,75\%$.

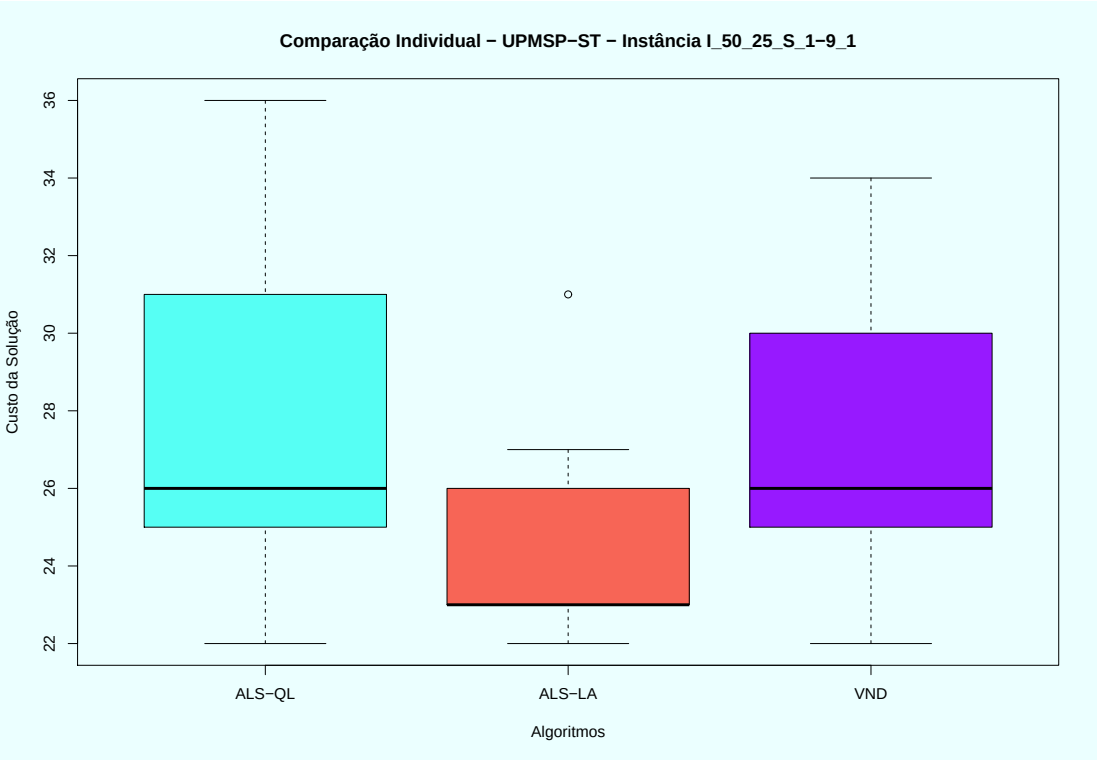


Figura 5.23: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico em relação ao *makespan* para os cenários com um único agente - instância I_100_25_S_1-9_1

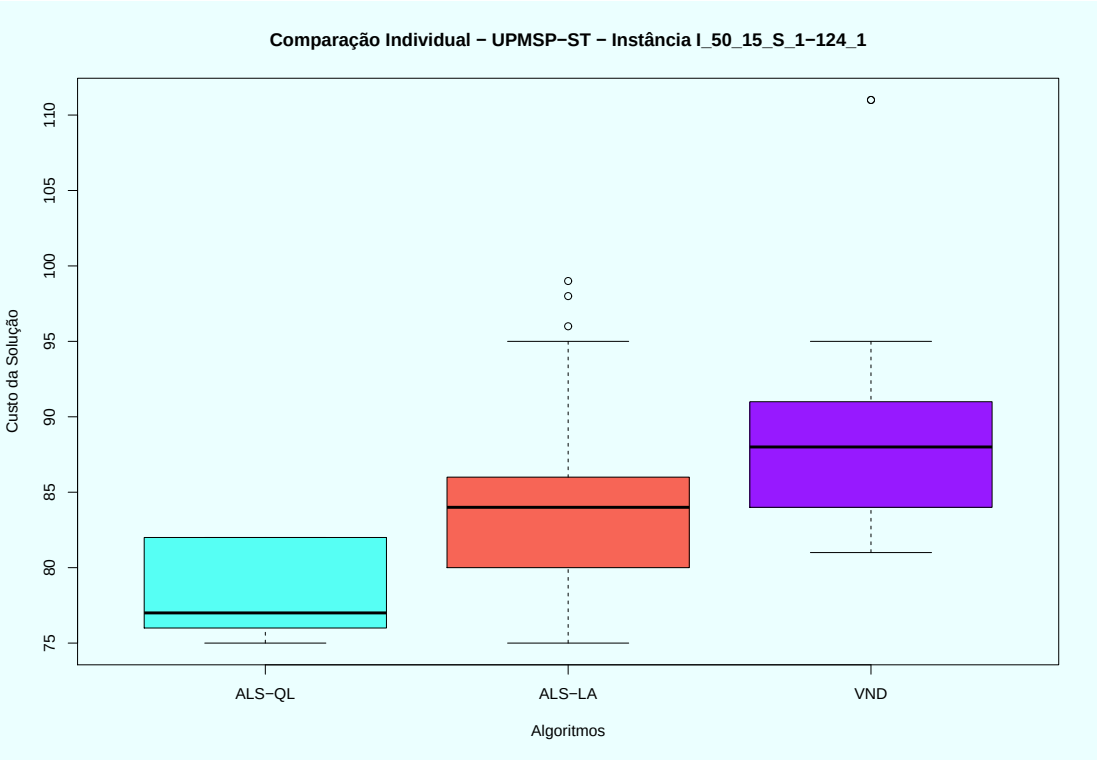


Figura 5.24: Comparação entre as propostas ALS-*QLearning* e ALS-LA e o VND clássico em relação ao *makespan* para os cenários com um único agente - instância I_100_15_S_1-124_1

As Figuras 5.22, 5.23, 5.24 apresentam exemplos de soluções obtidas em 30 execuções das instâncias I_100_10_S_1-99_1, I_100_25_S_1-9_1 e I_100_15_S_1-124_1, respectivamente, para um único agente, com as buscas locais ALS-LA, ALS-*QLearning* e VND clássico. No gráfico da Figura 5.22, os melhores valores são obtidos pelas duas propostas apresentadas neste trabalho (ALS-LA e ALS-*QLearning*). As duas propostas obtiveram valores que não apresentam diferença estatística entre elas. Na Figura 5.23, os melhores valores são obtidos pela proposta ALS-LA, enquanto que, na Figura 5.24, os valores alcançados pela proposta ALS-*QLearning* são melhores do que os obtidos pelos demais algoritmos.

Destacamos que, assim como no VRPTW, para todos os testes com diferenças estatísticas, os valores das soluções obtidas melhoram de acordo com o crescimento do número de agentes envolvidos na busca pela solução, confirmando, assim, a escalabilidade do *framework*.

Assim como para o VRPTW, nota-se que os valores atingidos pelo ALS-*QLearning* são melhores do que os demais algoritmos, caso se considere dois ou mais agentes. Neste caso, os valores melhoram de acordo com que o número de agentes envolvidos na busca pela solução cresça, o que demonstra a escalabilidade do *framework*. Nos cenários com 1 único agente, as propostas ALS-LA e ALS-QL apresentam desempenho semelhante e superior ao VND clássico.

Os resultados apresentados nesta seção demonstram também a adaptabilidade das propostas que utilizam aprendizado ao UPMSP-ST. Desta forma, fica claro que estas propostas não dependem da ordem de aplicação das vizinhanças pré-definida para a busca local.

5.2.4 Média das Soluções

Para fins de completude na apresentação dos resultados computacionais, os valores associados às soluções médias também são apresentados.

As Tabelas 5.9, 5.10, 5.11 e 5.12 apresentam os valores médios para o VRPTW. A Tabela 5.9 apresenta os custos médios da distância percorrida (DP) e o número de rotas (NR) das soluções obtidas com as 30 execuções da proposta da ALS-*QLearning* para o VRPTW. A Tabela 5.10 mostra os custos médios das soluções obtidas com as 30 execuções da proposta ALS-LA. A Tabela 5.11 mostra os custos médios das soluções obtidas com as 30 execuções do VND clássico utilizando a Ordem 1 de vizinhanças. A Tabela 5.12 mostra os custos médios das soluções obtidas com as 30 execuções do VND clássico utilizando a Ordem 2 de vizinhanças.

As Tabelas 5.13, 5.14 e 5.15 apresentam os valores médios para o UPMSP-ST. A Tabela 5.13 mostra o valor médio do *makespan* para as soluções obtidas com as 30 execuções com a proposta ALS-*QLearning*. Por sua vez, a Tabela 5.14 inclui o valor médio do *makespan* para as soluções obtidas com as 30 execuções com a proposta ALS-LA. A Tabela 5.15 inclui o valor médio do *makespan* para as soluções obtidas com as 30 execuções usando o VND clássico.

A partir da análise dessas tabelas, algumas observações surgem. A primeira observação, válida para ambos os problemas e propostas, é o efeito gerado pelo aumento do número de agentes em uso no *framework*. Há uma melhoria nos resultados obtidos aumentando-se o número de agentes em ação no *framework*. No caso do VRPTW, esse fato é válido tanto em relação ao número total de veículos quanto à distância total percorrida, e independentemente da classe de instância avaliada. Claramente, portanto, há uma identificação de um efeito de escalabilidade no número de agentes. Este efeito, a propósito, também é relatado

em aplicações da arquitetura A-Teams (Barbucha et al., 2010).

A segunda observação é sobre a comparação entre as duas técnicas de aprendizagem e seus resultados. Independentemente da classe do problema avaliado, a superioridade dos resultados obtidos com as propostas de ALS-*QLearning* e ALS-LA em relação àqueles obtidos pelo VND clássico, já previamente comprovada na análise estatística, também é demonstrada pela apresentação direta dos resultados nestas tabelas.

Tabela 5.9: Custos médios das soluções obtidas pela proposta ALS-*QLearning* para o VRPTW

Classe	**BKS**		**ALS-*QLearning***							
			1 Agente		**2 Agentes**		**4 Agentes**		**8 Agentes**	
	DP	**NR**	**DP**	**NR**	**DP**	**NR**	**DP**	**NR**	**DP**	**NR**
C107	**828.93**	**10**	828.93	10.00	828.93	10.00	828.93	10.00	828.93	10.00
C108	**828.93**	**10**	828.93	10.00	828.93	10.00	828.93	10.00	828.93	10.00
C109	**828.94**	**10**	832.72	10.00	829.91	10.00	828.94	10.00	828.94	10.00
C206	**588.49**	**3**	588.49	3.00	588.49	3.00	588.49	3.00	588.49	3.00
C207	**588.29**	**3**	588.29	3.00	588.29	3.00	588.29	3.00	588.29	3.00
C208	**588.32**	**3**	588.32	3.00	588.32	3.00	588.32	3.00	588.32	3.00
R110	**1118.83**	**11**	1140.60	11.00	1131.79	11.00	1124.48	11.00	1115.20	11.00
R111	**1096.70**	**10**	1102.52	11.00	1099.37	11.00	1081.47	11.00	1053.83	11.00
R112	**982.14**	**9**	1002.30	10.00	997.72	10.00	1994.09	10.00	988.49	10.00
R209	**909.16**	**3**	980.49	3.00	962.66	3.00	960.11	3.00	944.08	3.00
R210	**939.37**	**3**	989.13	3.00	983.03	3.00	974.36	3.00	964.80	3.00
R211	**885.71**	**2**	832.55	3.00	833.67	3.00	824.55	3.00	786.15	3.00
RC106	**1424.70**	**11**	1430.65	13.00	1414.34	12.83	1408.5	12.43	1400.23	12.00
RC107	**1230.50**	**11**	1288.18	12.00	1286.78	11.53	1275.46	11.17	1270.50	11.03
RC108	**1139.80**	**10**	1167.96	11.00	1184.89	10.90	1183.40	10.50	1161.95	10.23
RC206	**1146.30**	**3**	1245.6	3.00	1236.79	3.00	1220.04	3.00	1204.95	3.00
RC207	**1061.1**	**3**	1201.34	3.77	1192.11	3.00	1137.22	3.00	1144.32	3.00
RC208	**828.14**	**3**	915.86	3.00	907.05	3.00	892.63	3.00	879.89	3.00

Tabela 5.10: Custos médios das soluções obtidas pela proposta ALS-LA para o VRPTW

Classe	**BKS**		**ALS-AL**							
			1 Agente		**2 Agentes**		**4 Agentes**		**8 Agentes**	
	DP	**NR**	**DP**	**NR**	**DP**	**NR**	**DP**	**NR**	**DP**	**NR**
C107	**828.94**	**10**	828.94	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C108	**828.94**	**10**	828.94	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C109	**828.94**	**10**	860.82	10.00	849.41	10.00	846.71	10.00	834.83	10.00
C206	**588.49**	**3**	588.49	3.00	588.49	3.00	588.49	3.00	588.49	3.00
C207	**588.29**	**3**	588.29	3.00	588.29	3.00	588.29	3.00	588.29	3.00
C208	**588.32**	**3**	588.32	3.00	588.32	3.00	588.32	3.00	588.32	3.00
R110	**1118.8**	**10**	1129.96	11.00	1140.87	11.00	1142.75	11.00	1121.72	11.00
R111	**1096.7**	**10**	1108.45	11.00	1099.52	11.00	1095.44	11.00	1077.20	11.00
R112	**982.14**	**9**	1001.67	10.00	991.36	10.00	989.83	10.00	989.80	10.00
R209	**909.16**	**3**	975.43	3.00	970.21	3.00	975.65	3.00	960.16	3.00
R210	**939.37**	**3**	1004.73	3.00	998.03	3.00	988.48	3.00	977.69	3.00
R211	**885.71**	**2**	856.31	3.00	853.90	3.00	826.46	3.00	813.04	3.00
RC106	**1424.7**	**11**	1429.07	12.80	1414.30	12.50	1410.11	12.03	1407.41	12.03
RC107	**1230.5**	**11**	1268.59	11.97	1270.77	11.57	1253.57	11.00	1250.00	11.03
RC108	**1139.8**	**10**	1197.93	11.00	1184.50	11.00	1172.73	10.97	1165.4	10.57
RC206	**1146.3**	**3**	1241.13	3.00	1236.73	3.00	1233.51	3.00	1221.28	3.00
RC207	**1061.1**	**3**	1136.13	3.00	1137.09	3.00	1136.70	3.00	1120.45	3.00
RC208	**828.14**	**3**	938.67	3.00	921.04	3.00	905.20	3.00	892.23	3.00

Tabela 5.11: Custos médios das soluções obtidas pelo VND clássico utilizando a Ordem 1 de vizinhanças para o VRPTW

Classe	BKS		VND O1							
			1 Agente		2 Agentes		4 Agentes		8 Agentes	
	DP	NR	DP	NR	DP	NR	DP	NR	DP	NR
C107	**828.94**	**10**	828.94	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C108	**828.94**	**10**	828.94	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C109	**828.94**	**10**	859.28	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C206	**588.49**	**3**	588.49	3.00	588.49	3.00	588.49	3.00	588.49	3.00
C207	**588.29**	**3**	588.29	3.00	588.29	3.00	588.29	3.00	588.29	3.00
C208	**588.32**	**3**	588.32	3.00	588.32	3.00	588.32	3.00	588.32	3.00
R110	**1118.8**	**10**	1127.60	11.83	1126.73	11.33	1121.45	11.13	1113.08	11.00
R111	**1096.7**	**10**	1135.40	11.00	1108.30	11.00	1088.25	11.00	1088.30	11.00
R112	**982.14**	**9**	1019.47	10.00	986.10	10.00	990.73	10.00	983.98	10.00
R209	**909.16**	**3**	1010.40	3.00	977.10	3.00	970.01	3.00	965.97	3.00
R210	**939.37**	**3**	1023.83	3.00	983.36	3.00	983.19	3.00	974.54	3.00
R211	**885.71**	**2**	836.18	3.00	839.54	3.00	822.74	3.00	814.88	3.00
RC106	**1424.7**	**11**	1422.15	12.97	1427.89	12.93	1422.12	12.87	1410.52	12.77
RC107	**1230.5**	**11**	1286.65	12.00	1290.12	11.97	1277.79	11.53	1272.88	11.40
RC108	**1139.8**	**10**	1150.77	11.00	1150.40	11.00	1159.10	10.97	1155.74	10.87
RC206	**1146.3**	**3**	1151.87	3.87	1260.62	3.00	1226.21	3.00	1235.45	3.03
RC207	**1061.1**	**3**	1143.21	3.50	1143.00	3.00	1134.00	3.03	1104.50	3.00
RC208	**828.14**	**3**	930.06	3.00	929.96	3.00	911.71	3.00	908.72	3.00

Tabela 5.12: Custos médios das soluções obtidas VND clássico utilizando a Ordem 2 de vizinhanças para o VRPTW

Classe	BKS		VND O2							
			1 Agente		2 Agentes		4 Agentes		8 Agentes	
	DP	NR	DP	NR	DP	NR	DP	NR	DP	NR
C107	**828.94**	**10**	828.94	0.00	828.94	0.00	828.94	0.00	828.94	0.00
C108	**828.94**	**10**	828.94	10.00	828.94	10.00	828.94	10.00	828.94	10.00
C109	**828.94**	**10**	861.97	10.00	856.10	10.00	842.78	10.00	838.03	10.00
C206	**588.49**	**3**	588.49	3.00	588.49	3.00	588.49	3.00	588.49	3.00
C207	**588.29**	**3**	588.29	3.00	588.29	3.00	588.29	3.00	588.29	3.00
C208	**588.32**	**3**	588.32	3.00	588.32	3.00	588.32	3.00	588.32	3.00
R110	**1118.8**	**10**	1139.97	11.00	1135.99	11.00	1123.20	11.00	1120.40	11.00
R111	**1096.7**	**10**	1123.88	11.00	1118.37	11.00	1109.20	10.97	1092.02	11.00
R112	**982.14**	**9**	1028.54	10.00	1003.78	10.00	991.52	10.00	987.90	10.00
R209	**909.16**	**3**	995.79	3.00	970.30	3.00	963.30	3.00	966.91	3.00
R210	**939.37**	**3**	1019.95	3.00	992.88	3.00	1002.84	3.00	987.54	3.00
R211	**885.71**	**2**	870.67	3.00	846.33	3.00	835.21	3.00	830.00	3.00
RC106	**1424.7**	**11**	1418.66	13.00	1425.64	12.97	1420.29	12.47	1420.70	12.27
RC107	**1230.5**	**11**	1288.18	11.93	1275.44	11.37	1261.92	11.00	1260.00	11.03
RC108	**1139.8**	**10**	1205.57	11.00	1177.06	10.87	1177.08	10.83	1170.42	10.77
RC206	**1146.3**	**3**	1249.84	3.00	1243.34	3.00	1240.35	3.00	1239.55	3.00
RC207	**1061.1**	**3**	1265.19	3.00	1215.29	3.00	1178.62	3.00	1171.65	3.00
RC208	**828.14**	**3**	952.06	3.00	920.85	3.00	920.85	3.00	912.57	3.00

5.2.5 Cooperação

A cooperação é avaliada usando como base a trajetória das soluções no *Pool* de Soluções. Como já definido na Seção 4.5, esta avaliação é possível através da análise dos dados armazenados nas estruturas *Sender* e *Receiver*. Estas estruturas estão associadas às soluções e são responsáveis por armazenar a identificação dos agentes que enviam e que acessam soluções no *Pool* de Soluções. Desta forma, a cooperação é observada em situações em que a solução inserida por um agente no *pool* é utilizada por outro agente em seu processo de busca, definindo, assim, uma nova direção para este agente. Esta análise é ilustrada nas Figuras 5.25 e 5.26.

Tabela 5.13: Custos médios da proposta ALS-*QLearning* para o UPMSP-ST

Instância	BKS	Scenarios			
		1 agente	2 agentes	4 agentes	8 agentes
I_50_10_S_1-9_1	**67.00**	74.40	70.00	67.00	67.00
I_50_10_S_1-49_1	**77.00**	83.53	82.03	78.67	78.50
I_50_10_S_1-99_1	**118.00**	119.77	120.40	119.20	118.00
I_50_10_S_1-124_1	**114.00**	115.60	116.43	115.63	114.07
I_50_15_S_1-9_1	**36.00**	46.47	44.17	42.33	41.07
I_50_15_S_1-49_1	**59.00**	59.03	59.00	59.00	59.00
I_50_15_S_1-99_1	**78.00**	78.73	78.73	78.57	78.00
I_50_15_S_1-124_1	**75.00**	78.20	76.93	76.40	75.40
I_50_20_S_1-9_1	**31.00**	39.83	38.53	36.67	35.53
I_50_20_S_1-49_1	**39.00**	63.03	60.23	56.93	56.03
I_50_20_S_1-99_1	**49.00**	65.40	64.03	63.30	62.73
I_50_20_S_1-124_1	**52.00**	66.97	64.50	62.34	61.20
I_50_25_S_1-9_1	**22.00**	27.23	24.83	23.27	22.33
I_50_25_S_1-49_1	**25.00**	46.53	43.13	40.00	37.70
I_50_25_S_1-99_1	**35.00**	56.23	55.01	52.43	49.80
I_50_25_S_1-124_1	**37.00**	61.00	57.33	54.30	52.30

Tabela 5.14: Custos médios das soluções obtidas com 30 execuções da proposta ALS-LA para o UPMSP-ST

Instância	BKS	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
I_50_10_S_1-9_1	**67.00**	73.87	71.83	70.50	69.23
I_50_10_S_1-49_1	**77.00**	85.97	83.50	80.53	79.60
I_50_10_S_1-99_1	**118.00**	120.60	120.33	118.90	118.17
I_50_10_S_1-124_1	**114.00**	118.50	117.70	117.43	116.63
I_50_15_S_1-9_1	**36.00**	46.93	44.90	42.87	41.47
I_50_15_S_1-49_1	**59.00**	62,60	61.77	60.53	59.40
I_50_15_S_1-99_1	**78.00**	80.40	78.77	78.17	78.03
I_50_15_S_1-124_1	**75.00**	84.47	80.20	78.17	75.87
I_50_20_S_1-9_1	**31.00**	41.40	37.80	36.20	34.97
I_50_20_S_1-49_1	**39.00**	61.17	59.67	59.17	56.33
I_50_20_S_1-99_1	**49.00**	65.87	65.53	66.27	65.60
I_50_20_S_1-124_1	**52.00**	67.00	66.23	66.00	65.57
I_50_25_S_1-9_1	**22.00**	24.17	23.87	22.87	22.23
I_50_25_S_1-49_1	**25.00**	54.10	48.90	45.93	43.43
I_50_25_S_1-99_1	**35.00**	59.37	56.10	54.30	50.17
I_50_25_S_1-124_1	**37.00**	61.77	59.80	59.17	56.23

Tabela 5.15: Custos médios da proposta VND clássica para o UPMSP-ST

Instância	BKS	Cenários			
		1 agente	2 agentes	4 agentes	8 agentes
I_50_10_S_1-9_1	**67.00**	78.70	73.93	73.20	66.77
I_50_10_S_1-49_1	**77.00**	85.50	81.67	81.27	80.30
I_50_10_S_1-99_1	**118.00**	126.07	122.17	121.00	119.33
I_50_10_S_1-124_1	**114.00**	120.97	118.13	116.23	114.33
I_50_15_S_1-9_1	**36.00**	45.60	43.93	42.53	40.90
I_50_15_S_1-49_1	**59.00**	62.97	59.23	59.00	59.00
I_50_15_S_1-99_1	**78.00**	81.63	78.00	78.43	78.17
I_50_15_S_1-124_1	**75.00**	88.70	81.20	79.03	78.30
I_50_20_S_1-9_1	**31.00**	39.33	38.27	35.40	34.80
I_50_20_S_1-49_1	**39.00**	63.00	62.33	58.87	57.87
I_50_20_S_1-99_1	**49.00**	66.33	64.77	65.33	65.67
I_50_20_S_1-124_1	**52.00**	65.77	66.93	66.27	65.60
I_50_25_S_1-9_1	**22.00**	26.50	25.30	23.47	22.37
I_50_25_S_1-49_1	**25.00**	52.30	46.97	46.17	42.70
I_50_25_S_1-99_1	**35.00**	60.07	58.37	57.13	52.77
I_50_25_S_1-124_1	**37.00**	60.47	59.63	58.17	54.37

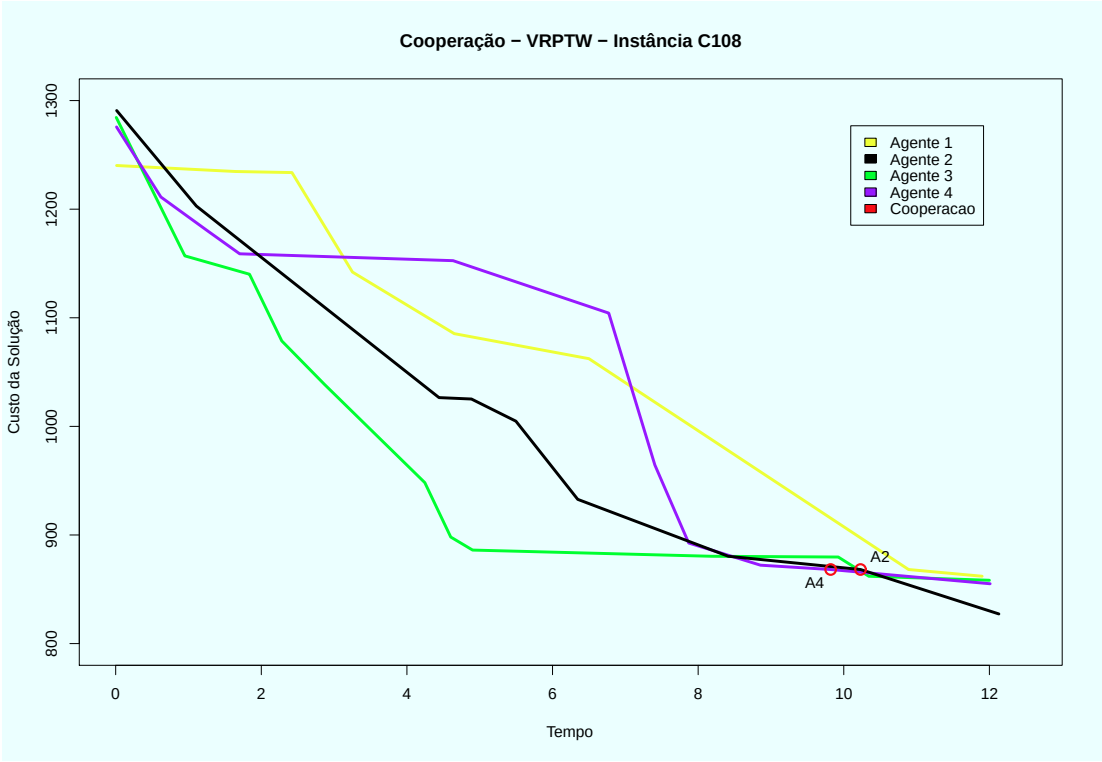


Figura 5.25: Cooperação C108 ALS-QL.

A Figura 5.25 apresenta a trajetória de busca de 4 agentes, em conjunto, para a solução da instância C108 do VRPTW, usando a proposta ALS-QL. Nesta figura, observa-se que cada agente percorre, simultaneamente, seu caminho para resolver o problema, agindo de forma independente. Todas as melhores soluções, encontradas ao final de cada iteração dos métodos implementados pelos agentes, foram compartilhadas no *Pool* de Soluções, e são identificadas no gráfico como pontos nas trajetórias de cada agente. Um bom exemplo de cooperação efetiva acontece entre os agentes 4 e 2 da Figura 5.25. A solução com distância total $868,20$ é encontrada e compartilhada pelo agente 4 no instante de tempo $9,821$ segundos. Em seguida, esta solução, nomeada na figura como $A4$, é acessada pelo Agente 2 (no instante de tempo $10,234$ segundos) e usada para prosseguir sua busca (solução $A2$ na Figura 5.25). A partir dessa cooperação entre os agentes, o Agente 2 pode alcançar, como solução final, o melhor resultado entre os 4 agentes, que corresponde ao valor da melhor solução para essa instância.

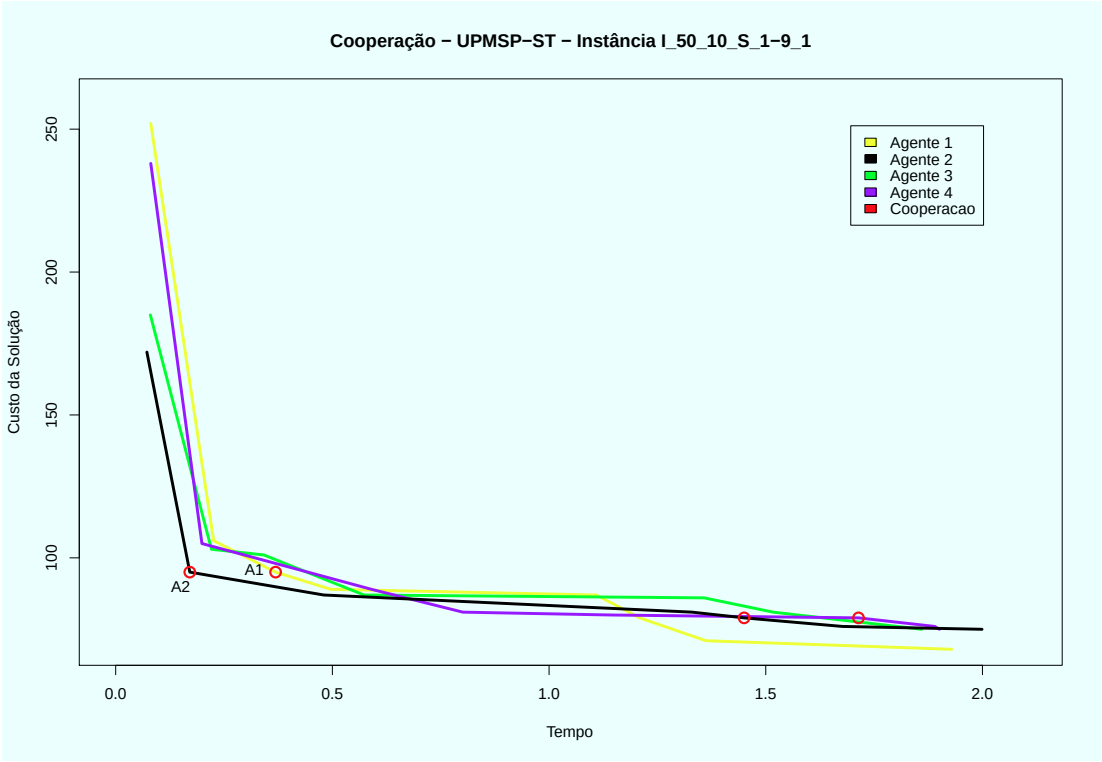


Figura 5.26: Cooperação 50_10_9 ALS-QL.

A Figura 5.26 mostra a mesma análise para a instância I_50_10_S_1-9_1 do UPMSP-ST, usando a proposta ALS-QL. A trajetória gerada pelas soluções inseridas no *pool*, por cada agente, é apresentada. Nesta execução, assim com na anterior, é possível notar que cada agente desempenha sua busca independentemente, traçando caminhos diferentes. Os círculos vermelhos indicam os pontos nos quais houve cooperação. Neste caso, destacamos, como um exemplo de ação cooperativa, a solução encontrada pelo Agente 2, denominada $A2$. O Agente 2 compartilha no *pool*, no instante de tempo $0,171$ segundos, a solução com *makespan* 95 (solução $A2$ na Figura 5.26). Esta solução é acessada pelo Agente 1 (solução $A1$ na Figura 5.26), no instante de tempo $0,369$ segundos, e usada para prosseguir sua busca. Assim como no exemplo para o VRPTW, a partir da cooperação entre estes agentes, o Agente 1 pode alcançar, como solução final, o melhor resultado entre os 4 agentes, que, aqui, também corresponde ao valor da melhor solução para essa instância.

A partir desta análise, a cooperação e sua habilidade de influenciar a qualidade das soluções dos agentes envolvidos no processo de busca são demonstradas. Esta análise permite entender e confirmar os resultados já apresentados nas análises das Seções 5.2.1, 5.2.2 e 5.2.3, nos quais há uma redução nos custos das soluções com o uso de agentes cooperativos.

Capítulo 6

Conclusão Finais e Direções Futuras

Neste Capítulo, inicialmente são apresentadas as conclusões finais deste trabalho (Seção 6.1). Em seguida, são apresentadas as propostas de trabalhos futuros, mostrando os caminhos que esta pesquisa pode seguir a partir do presente trabalho (Seção 6.2). Adicionalmente, são apresentadas as publicações resultantes desta pesquisa em eventos científicos e periódicos especializados (Secão 6.3).

6.1 Conclusões Finais

O presente trabalho abordou o *Framework* Multiagente para Otimização Combinatória usando Metaheurísticas denominado AMAM (**A**rquitetura **M**ulti**A**gente para **M**etaheurísticas). O AMAM foi inicialmente proposto em Silva (2007). O *framework* AMAM é uma estrutura genérica e flexível que tem, como principal força, a facilidade de hibridização de metaheurísticas, a partir da utilização de conceitos relacionados a sistemas multiagentes.

Esta tese propôs uma revisão detalhada do *framework* AMAM, assim como propôs a incorporação de novos recursos que permitam dinamizar e aperfeiçoar o processo de solução.

Para este fim, inicialmente, foi realizada uma revisão detalhada das principais ferramentas disponíveis na literatura, com o propósito de identificar as lacunas existentes e as características desejáveis a um *framework* para otimização usando metaheurísticas. Esta investigação do estado da arte permitiu listar e comparar os principais trabalhos correlatos a este. O foco desta análise foi direcionado à hibridização de metaheurísticas, especialmente aquelas que utilizam os conceitos de metaheurísticas cooperativas e metaheurísticas paralelas.

A análise comparativa apresentada na Seção 2.3 mostrou que a hibridização ainda não é uma característica forte dos *frameworks* existentes, como já havia sido afirmado por Parejo et al. (2012). Adicionalmente, esta análise mostrou que os *frameworks* que utilizam as abordagens multiagentes são uma boa alternativa para o desenvolvimento de estruturas que possibilitam a hibridização de metaheurísticas, permitindo, com considerável facilidade, o uso de técnicas cooperativas, como também, na grande maioria dos casos, o uso de recursos computacionais paralelos. A flexibilidade na implementação de métodos híbridos é proporcionada principalmente pela cooperação e ação autônoma dos agentes presentes nestas propostas, que utilizam o conceito de sistemas multiagentes. Além disso, o impacto de novas tecnologias de suporte a computação paralela e distribuída aponta para novas possibilidades de pesquisa que ainda precisam ser exploradas pelos desenvolvedores de *frameworks*.

Dentre as lacunas identificadas, recursos comuns em sistemas multiagentes, como, por exemplo, o aprendizado, não são utilizados pelos *frameworks* avaliados. Da mesma forma, faltam ferramentas para o suporte ao processo de otimização, como por exemplo, o auto-ajuste de parâmetros.

A análise comparativa realizada demonstrou que o *framework* AMAM é um candidato importante para suprir as lacunas no desenvolvimento de *frameworks* para otimização usando metaheurísticas.

A principal contribuição desta tese está na proposta de incorporação de capacidades auto-adaptativas nos agentes do *framework*, com o objetivo de permitir ao agente se adaptar às características específicas do problema. Esta proposta surge a partir de um dos principais questionamentos em relação ao uso de *frameworks*, que está na generalização dos métodos para que atuem em diferentes problemas. Esta generalização pode, em alguns casos, levar à perda da qualidade das soluções geradas. Desta forma, o objetivo é que, através da experiência com o ambiente, o agente seja capaz de se adaptar e buscar o melhor ajuste do seu comportamento de acordo com as características específicas deste ambiente, proporcionando um melhor desempenho do *framework*. A capacidade adaptativa é, neste trabalho, incorporada ao agente através de uma busca local adaptativa que utiliza os conceitos de Aprendizagem de Máquina, mais especificamente, utilizando o algoritmo *Q-Learning*, através da proposta ALS-*QLearning*, e Autômatos de Aprendizagem, através da proposta ALS-LA.

Além da adição de capacidades auto-adaptativas, a estrutura geral do *framework* AMAM foi revisada neste trabalho, sendo as principais contribuições desta revisão listadas a seguir:

(i) Aumento da autonomia dos agentes, removendo todo tipo de coordenação explícita entre eles e os elementos que intermediavam a comunicação;

(ii) Aprimoramento da cooperação entre os agentes, buscando maior diversidade nas soluções disponíveis na estrutura cooperativa, através da definição de novos critérios de inserção de novas soluções;

(iii) Divisão da estrutura geral do *framework* em dimensões, a partir da visão deste considerando diferentes perspectivas. O objetivo aqui foi facilitar o entendimento da arquitetura apresentada;

(iv) Consolidação do *framework* AMAM como uma ferramenta capaz de resolver diferentes problemas de otimização combinatória usando metaheurísticas.

O desempenho do *framework* AMAM foi avaliado através da sua instanciação para dois problemas clássicos de otimização combinatória: VRPTW e UPMSP-ST. As instâncias do *framework* desenvolvidas para os problemas citados utilizaram Agentes ILS com três variações de busca local: Busca Local Adaptativa ALS-LA, Busca Local Adaptativa ALS-*QLearning* e VND no seu formato clássico. Quatro cenários de teste também foram utilizados: com 1 agente atuando individualmente e com 2, 4 e 8 agentes realizando a busca em conjunto. A partir dessas instanciações foi possível avaliar: (i) a flexibilidade do *framework* AMAM aplicado a diferentes problemas de otimização; (ii) o desempenho da proposta ALS-LA quanto à cooperação e à escalabilidade; (iii) o desempenho da proposta ALS-*QLearning*, também em relação à cooperação e à escalabilidade; (iv) o desempenho das duas propostas (ALS-LA e ALS-*QLearning*) em relação ao VND clássico, tanto do

ponto de vista do trabalho do agente individualmente quanto do trabalho em equipe; e (v) a efetividade da cooperação.

Os resultados obtidos nos experimentos realizados foram comparados utilizando teste de hipóteses paramétrico, com nível de confiança de 95%. Os resultados mostraram a efetiva redução nos custos das soluções com o uso de agentes cooperativos, para todos os algoritmos testados e para os dois problemas instanciados. Essa redução é ainda mais significativa quando se aumenta o número de agentes. Desta forma, a capacidade de cooperação e sua influência na qualidade das soluções dos agentes envolvidos é confirmada pelos experimentos.

A comparação das propostas apresentadas neste trabalho com o VND clássico permitiu avaliar o comportamento do agente com e sem aprendizado. A comparação entre estes algoritmos de busca local implementados foi realizada considerando os dois problemas separadamente. Da mesma forma, esta análise foi feita considerando o comportamento individual e em equipe, também separadamente.

A proposta ALS-*QLearning* se destacou na análise dos cenários com dois ou mais agentes para o VRPTW. Neste caso, a proposta ALS-*QLearning* obteve o melhor desempenho, gerando o maior número de melhores resultados entre os algoritmos avaliados. Já no cenário com um único agente, as duas propostas apresentadas neste trabalho tiveram desempenho semelhante, sendo melhores mais vezes do que o VND clássico. Aqui, é importante ressaltar que, em nenhum dos testes realizados, as propostas ALS-LA e ALS-*QLearning* obtiveram soluções inferiores às obtidas pelo VND nas versões testadas, o que indica que as duas propostas de aprendizado se adaptam bem ao problema, não sendo necessário definir a ordem de aplicação das vizinhanças em sua busca local.

As duas ordens de aplicação das vizinhanças testadas no VND clássico foram utilizadas para demonstrar que há diferença nas soluções obtidas, para cada ordem, de acordo com a instância teste utilizada e para verificar a adaptabilidade das propostas apresentadas. Como afirmado acima, as duas propostas mostraram obter desempenho melhor na maioria das vezes ou, no mínimo, compatível, em relação à versão do VND que teve melhor desempenho em cada instância testada.

A proposta ALS-*QLearning* se destaca também na análise dos cenários com dois ou mais agentes para o UPMSP-ST, obtendo o melhor resultado na maioria das instâncias testadas. Assim como no caso do VRPTW, para o UPMSP-ST, nos cenários com um único agente, as duas propostas se comportaram de forma semelhante, obtendo os melhores resultados na maior parte das instâncias testadas. A adaptabilidade das duas propostas apresentadas neste trabalho é novamente confirmada para o UPMSP-ST, demonstrando que as técnicas de aprendizado utilizadas conseguem superar a necessidade de conhecimento das características específicas do problema a ser tratado.

A incorporação de capacidades auto-adaptativas no agente do *framework* AMAM reforça a consolidação deste como uma ferramenta de *software* capaz de atuar na solução de diferentes problemas de otimização usando metaheurísticas. No contexto de *frameworks*, esta característica é um diferencial que permite subjugar um dos principais questionamentos realizados em relação ao uso deste tipo de ferramenta.

A efetividade da cooperação é também avaliada nos experimentos realizados. A análise da trajetória das soluções no *Pool* permitiu distinguir o exato momento em que uma solução inserida por um agente é utilizada por outro(s) agente(s) em seu processo de busca, definindo, assim, uma nova direção para a busca deste agente e levando, em alguns casos, a melhores soluções entre os demais agentes do sistema. Esta análise confirma o que os resultados já mostraram: a cooperação entre os agentes influencia, efetivamente, a

qualidade das soluções obtidas. Cabe ressaltar que esta influência cresce à medida que o número de agentes envolvidos no processo de busca também aumenta.

Finalmente, esta tese buscou as respostas para as quatro questões colocadas no Capítulo 1, aqui retomadas para facilidade de leitura: (i) os *frameworks* atuais facilitam o desenvolvimento de aplicações de metaheurísticas híbridas?; (ii) eles permitem o desenvolvimento de aplicações com cooperação (troca de informações) entre métodos que são executados independentemente e simultaneamente?; (iii) qual é a melhor maneira de coordenar as metaheurísticas quando o objetivo é explorar o potencial de hibridização?; (iv) ao mesmo tempo, como poderia ser desenvolvida uma estrutura mais robusta capaz de lidar com diferentes problemas com mudanças mínimas? Sendo assim, em relação à primeira questão, foi identificado que os *frameworks* atuais não facilitam o desenvolvimento de aplicações para metaheurísticas híbridas. Em relação à segunda questão, os *frameworks* atuais têm dificuldades em permitir o desenvolvimento de aplicações com cooperação (troca de informações) entre métodos que são executados independente e simultaneamente. Quanto à terceira questão, a resposta para a melhor maneira de se coordenar metaheurísticas, quando o objetivo é explorar o potencial de hibridização, é o uso de *frameworks* baseados em sistemas multiagentes. Finalmente, a resposta à quarta pergunta é o uso de *frameworks* multiagentes para metaheurísticas com características adaptativas, uma vez que permitem uma estrutura mais robusta, capaz de lidar com diferentes problemas.

6.2 Trabalhos Futuros

A demanda atual para o aprimoramento do *framework* está na incorporação de novos recursos que permitam dinamizar e aperfeiçoar o processo de solução, buscando cada vez mais melhorar a qualidade das soluções obtidas e tornando-o cada vez mais competitivo.

Sendo assim, pode-se afirmar que as lacunas identificadas pela análise comparativa, realizada no Capítulo 2, fornecem importantes oportunidades para trabalhos futuros no desenvolvimento de *frameworks* para otimização utilizando metaheurísticas. São elas:

(i) A utilização dos conceitos de aprendizagem de máquina na definição de novos recursos adaptativos para auto-ajuste de parâmetros específicos do problema e em metaheurísticas populacionais;

(ii) Implementação de recursos de computação distribuída, para que o *framework* possa ser realmente executado em paralelo, através, principalmente, de GPUs;

(iii) Implementação de ferramentas de suporte a implementação, como análise estatística e interface;

(iv) Incorporação no *framework* do aprendizado multiagente, no qual o agente aprende seu comportamento ótimo em conjunto com outros agentes;

(v) Implementação de Problemas de Otimização Multiobjetivo no *framework*.

6.3 Publicações derivadas desta pesquisa

A seguir são listadas as publicações geradas pela presente pesquisa.

1. Título: **An Agent-Based Metaheuristic Approach applied to the Vehicle Routing Problem with Time-Windows**
   - Autores: Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Sabrina Moreira de Oliveira e Marcone Jamilson Freitas Souza
   - Evento: 2014 Brazilian Conference on Intelligent Systems - Encontro Nacional. de Inteligência Artificial e Computacional (BRACIS-ENIAC 2014)
   - Local: São Carlos, SP, Brasil
   - Período: 18 a 22 de outubro de 2014
   - Trabalho completo apresentado em evento
2. Título: **A Multi-agent Metaheuristic Optimization Framework with Cooperation**
   - Autores: Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza e Sabrina Moreira de Oliveira
   - Evento: 2015 Brazilian Conference on Intelligent Systems (BRACIS)
   - Local: Natal, Rio Grande do Norte, Brasil
   - Período: 4 a 7 de novembro de 2015
   - Trabalho completo apresentado em evento
3. Título: **Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis**
   - Autores: Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza e Moacir Felizardo de França Filho
   - Jornal: Applied Soft Computing
   - Editora: Elsevier
   - Volume: 71
   - Páginas: 433–459
   - Ano: 2018
   - DOI: `https://doi.org/10.1016/j.asoc.2018.06.050`
   - URL: `http://www.sciencedirect.com/science/article/pii/S1568494618303867`
4. Título: **A Reinforcement Learning-based Multi-Agent Framework applied for solving routing and scheduling problems**
   - Autores: Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza e Ana Lúcia C. Bazzan
   - Jornal: Expert Systems With Applications

- Editora: Elsevier
- Volume: 131
- Páginas: 148–171
- Ano: 2019
- DOI: `https://doi.org/10.1016/j.eswa.2019.04.056`
- URL: `http://www.sciencedirect.com/science/article/pii/S0957417419302866`

Referências Bibliográficas

Agerbeck, C. e Hansen, M. O. (2008). A multi-agent approach to solving NP-complete problems. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark (DTU), Denmark.

Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience.

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.; Luna, J.; Moreno, Luz Marina; 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).

Alirezaei, E.; Vahedi, Z. e Ghaznavi-Ghoushchi, M. May(2012). Parallel hybrid meta heuristic algorithm for university course timetabling problem (PHACT). *Proceedings of the 20th Iranian Conference on Electrical Engineering (2012 ICEE)*, p. 673–678, May(2012).

Allahverdi, Ali. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, v. 246, n. 2, p. 345 – 378. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.04.004. URL `http://www.sciencedirect.com/science/article/pii/S0377221715002763`.

Allahverdi, Ali; Ng, C.T.; Cheng, T.C.E. e Kovalyov, Mikhail Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, v. 187, n. 3, p. 985–1032. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2006.06.060. URL `http://www.sciencedirect.com/science/article/pii/S0377221706008174`.

Amaya, J. E.; Cotta, C. e Leiva, A. J. F. (2010). Hybrid cooperation models for the tool switching problem. González, J. R.; Pelta, D. A.; Cruz, C.; Terrazas, G. e Krasnogor, N., editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, volume 284 of *Studies in Computational Intelligence*, p. 39–52. Springer Berlin Heidelberg.

Applegate, David; Bixby, Robert; Chvátal, Vasek e Cook, William. (2007). *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2nd edição.

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.

Barbucha, Dariusz. (2010). Cooperative solution to the vehicle routing problem. Jędrzejowicz, Piotr; Nguyen, Ngoc Thanh; Howlet, Robert J. e Jain, Lakhmi C., editors, *Agent and Multi-Agent Systems: Technologies and Applications: 4th KES International Symposium, KES-AMSTA 2010, Gdynia, Poland, June 23-25, 2010, Proceedings. Part II*, p. 180–189. Springer Berlin Heidelberg, Berlin, Heidelberg.

Barbucha, Dariusz. (2014). Team of A-Teams approach for vehicle routing problem with time windows. Terrazas, German; Otero, Fernando E. B. e Masegosa, Antonio D., editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, volume 512 of *Studies in Computational Intelligence*, p. 273–286. Springer International Publishing.

Barbucha, Dariusz; Czarnowski, Ireneusz; Jędrzejowicz, Piotr; Ratajczak-Ropel, Ewa e Wierzbowska, Izabela. (2006). JABAT-an implementation of the A-Team concept. *Proceedings of the International Multiconference Computer Science and Information Technology, Wisła*, volume 1, p. 235–241, Wisla, Polônia. Polskie Towarzystwo Informatyczne.

Barbucha, Dariusz; Czarnowski, Ireneusz; Jędrzejowicz, Piotr; Ratajczak-Ropel, Ewa e Wierzbowska, Izabela. (2009). e-JABAT – an implementation of the web-based A-Team. Nguyen, Ngoc Thanh e Jain, Lakhmi C., editors, *Intelligent Agents in the Evolution of Web and Applications*, p. 57–86. Springer Berlin Heidelberg, Berlin, Heidelberg.

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.

Bellifemine, Fabio; Poggi, Agostino e Rimassa, Giovanni. (2001). Developing multi-agent systems with JADE. Castelfranchi, Cristiano e Lespérance, Yves, editors, *Intelligent Agents VII Agent Theories Architectures and Languages*, volume 1986 of *Lecture Notes in Computer Science*, p. 89–103. Springer Berlin Heidelberg.

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

Bellman, Richard. (1957). *Dynamic Programming*, volume 89. Princeton University Press.

Bertsekas, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.

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.; Puchinger, J.; Raidl, G. R. e Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, v. 11, n. 6, p. 4135 – 4151.

Blum, Christian e Roli, Andrea. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308.

Blum, Christian e Roli, Andrea. (2008). Hybrid metaheuristics: An introduction. Blum, Christian; Aguilera, María José Blesa; Roli, Andrea e Sampels, Michael, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, p. 1–30. Springer, Berlin Heidelberg.

Boussaid, I.; Lepagnot, J. e Siarry, P. Jul(2013). A survey on optimization metaheuristics. *Information Sciences*, v. 237, p. 82–117.

Burke, E.; Curtois, T.; Hyde, M.; Kendall, G.; Ochoa, G.; Petrovic, S.; Vázquez-Rodríguez, J. A. e Gendreau, M. July(2010). Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. *IEEE Congress on Evolutionary Computation*, p. 1–8, July(2010). doi: 10.1109/CEC.2010.5586064.

Burke, E. K.; Hart, E.; Kendall, G.; Newall, J.; Ross, P. e Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 457–474. Springer US.

Burke, E. K.; McCollum, B.; Meisels, A.; Petrovic, S. e Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, v. 176, n. 1, p. 177–192.

Burke, Edmund K.; Gendreau, Michel; Ochoa, Gabriela e Walker, James D. (2011). Adaptive Iterated Local Search for cross-domain optimisation. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, p. 1987–1994, New York, NY, USA. ISBN 978-1-4503-0557-0.

Butterfield, Andrew e Ngondi, Gerard Ekembe. (2016). *Oxford Dictionary of Computer Science*. Oxford University Press, 7 edição.

Byrski, Aleksander; Dreżewski, Rafał; Siwik, Leszek e Kisiel-Dorohinicki, Marek. (2015). Evolutionary multi-agent systems. *The Knowledge Engineering Review*, v. 30, n. 2, p. 171–186. doi: 10.1017/S0269888914000289.

Byrski, Aleksander e Kisiel-Dorohinicki, Marek. (2017). *Evolutionary multi-agent systems: From Inspirations to Applications*, volume 680 of *Studies in Computational Intelligence*. Springer.

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.

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

Carle, M.; Martel, A. e Zufferey, N. Aug(2012). Collaborative agent teams (CAT) for distributed multi-dimensional optimization. *Cirrelt*, v. 43.

Cetnarowicz, Krzysztof; Kisiel-Dorohinicki, Marek e Nawarecki, Edward. (1996). The application of evolution process in multi-agent world (MAW) to the prediction system. Tokoro, M., editor, *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96)*, p. 26–32. AAAI Press, (1996).

Chakhlevitch, Konstantin e Cowling, Peter. (2008). Hyperheuristics: recent developments. Cotta, Carlos; Sevaux, Marc e Sörensen, Kenneth, editors, *Adaptive and multilevel metaheuristics*, volume 136, p. 3–29. Springer, Berlin, Heidelberg.

Clancey, W. J. (1997). *Situated Cognition: on human knowledge and computer representations*. Cambridge University Press.

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. *Proceedings of the VII ALIOEURO 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. *Anais do XLII Simpósio Brasileiro de Pesquisa Operacional*, p. 1887–1898, Sept(2010).

Coelho, I. M.; Ribas, S.; Souza, M. J. F.; Coelho, V. N. e Ochi, L. S. (2009). A hybrid heuristic algorithm based on GRASP, VND, ILS and Path Relinking for the open-pit-mining operational planning problem. *Proceedings of the XXX Iberian-Latin-American Congress on Computational Methods in Engineering–CILAMCE, Búzios, Brazil*, (2009).

Coelho, V. N.; Coelho, I. M.; Coelho, B. N.; Cohen, M. W.; Reis, A. J. R.; Silva, S. M.; Souza, M. J. F.; Fleming, P. J. e Guimarães, F. G. (2016)a. Multi-objective energy storage power dispatching using plug-in vehicles in a smart-microgrid. *Renewable Energy*, v. 89, p. 730–742.

Coelho, V. N.; Coelho, I. M.; Coelho, B. N.; Reis, A. J. R.; Enayatifar, R.; Souza, M. J. F. e Guimarães, F. G. (2016)b. A self-adaptive evolutionary fuzzy model for load forecasting problems on smart grid environment. *Applied Energy*, v. 169, p. 567–584.

Coelho, V. N.; Grasas, A.; Ramalhinho, H.; Coelho, I. M.; Souza, M. J. F. e Cruz, R. C. (2016)c. An ILS-based algorithm to solve a large-scale real heterogeneous fleet VRP with multi-trips and docking constraints. *European Journal of Operational Research*, v. 250, n. 2, p. 367–376.

Coelho, V. N.; Oliveira, T. A.; Coelho, I. M.; Coelho, B. N.; Fleming, P. J.; Guimarães, F. G.; Ramalhinho, H.; Souza, M. J. F.; Talbi, El-Ghazali e Lust, T. (2017). Generic Pareto local search metaheuristic for optimization of targeted offers in a bi-objective direct marketing campaign. *Computers & Operations Research*, v. 78, p. 578–587.

Corkill, Daniel D. (1991). Blackboard systems. *AI Expert*, v. 9, n. 6, p. 40–47.

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.

Cowling, Peter; Kendall, Graham e Soubeiga, Eric. (2001). A hyperheuristic approach to scheduling a sales summit. Burke, Edmund e Erben, Wilhelm, editors, *Practice and Theory of Automated Timetabling III*, volume 2079, p. 176–190. Springer.

Crainic, T. G. e Gendreau, M. nov(2002). Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, v. 8, n. 6, p. 601–627.

Crainic, T. G. e Toulouse, M. (2003). Parallel strategies for meta-heuristics. Glover, Fred e Kochenberger, Gary A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 475–513. Springer US.

Crainic, Teodor Gabriel e Toulouse, Michel. (2010). Parallel meta-heuristics. Gendreau, Michel e Potvin, Jean-Yves, editors, *Handbook of Metaheuristics*, p. 497–541. Springer US, Boston, MA. ISBN 978-1-4419-1665-5. doi: 10.1007/978-1-4419-1665-5_17. URL `http://dx.doi.org/10.1007/978-1-4419-1665-5_17`.

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 Oliveira, Renata Barbosa. Desenvolvimento de uma arquitetura multiagentes baseada em metaheurísticas com um abordagem adaptative learning search. Dissertaccão de mestrado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brazil, (2008).

Di Gaspero, L. e Schaerf, A. (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.

Di Gaspero, L. e Urli, T. July(2011). A reinforcement learning approach for the cross-domain heuristic search challenge. *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, Udine, Italy.

Di Gaspero, Luca 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.

Di Gaspero, Luca 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.

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).

El-Abd, M. e Kamel, M. (2005). A taxonomy of cooperative search algorithms. Blesa, María José; Blum, Christian; Roli, Andrea e Sampels, Michael, editors, *Hybrid Metaheuristics*, volume 3636 of *Lecture Notes in Computer Science*, p. 32–41. Springer Berlin Heidelberg.

El-Mihoub, Tarek A.; Hopgood, Adrian A.; Nolle, Lars e Battersby, Alan. (2006). Hybrid genetic algorithms: A review. *Engineering Letters*, v. 13, n. 2, p. 124–137.

Elyasaf, Achiya e Sipper, Moshe. (2014). Software review: the HeuristicLab framework. *Genetic Programming and Evolvable Machines*, v. 15, n. 2, p. 215–218.

Fernandes, F. C. 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, sep(2009).

Fernandes, F. C.; de Souza, S. R.; Silva, M. A. L.; Borges, H. E. e Ribeiro, F. F. (2009). A multiagent architecture for solving combinatorial optimization problems through metaheuristics. *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics (SMC 2009)*, p. 3071–3076, (2009).

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.

Gamma, E.; Helm, R.; Johnson, R. e Vlissides, J. (1995). *Design Patterns: Elements of Object-Oriented Software*. Addison Wesley.

Gendreau, Michel e Potvin, Jean-Yves, editors. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2nd edição.

Gong, Yue-Jiao; Chen, Wei-Neng; Zhan, Zhi-Hui; Zhang, Jun; Li, Yun; Zhang, Qingfu e Li, Jing-Jing. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, v. 34, p. 286–300.

González-Álvarez, D. e Vega-Rodríguez, M. (2013). A parallel cooperative team of multi-objective evolutionary algorithms for motif discovery. *The Journal of Supercomputing*, v. 66, n. 3, p. 1576–1612.

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.

Hubner, Jomi F.; Sichman, Jaime S. e Boissier, Olivier. December(2007). Developing organised multiagent systems using the MOISE+ Model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, v. 1, n. 3/4, p. 370–395.

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.

Jin, J.; Crainic, T. G. e Løkketangen, A. (2014). A cooperative parallel metaheuristic for the capacitated vehicle routing problem. *Computers & Operations Research*, v. 44, n. 0, p. 33 – 41.

Jin, Xiaolong e Liu, Jiming. (2002). Multiagent SAT (MASSAT): Autonomous pattern search in constrained domains. Yin, Hujun; Allinson, Nigel; Freeman, Richard; Keane, John e Hubbard, Simon, editors, *Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'02)*, p. 318–328, London, UK. Springer Berlin.

Johnson, R. e Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, p. 22–35.

Jourdan, L.; Basseur, M. e Talbi, E.-G. (2009). Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, v. 199, n. 3, p. 620–629.

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.

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.

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.

Krzywicki, D.; Turek, W.; Byrski, A. e Kisiel-Dorohinicki, M. (2015). Massively concurrent agent-based evolutionary computing. *Journal of Computational Science*, v. 11, p. 153 – 162.

Landa-Silva, D. e Burke, E. K. nov(2007). Asynchronous cooperative local search for the office-space-allocation problem. *INFORMS Journal on Computing*, v. 19, n. 4.

Li, X.; Epitropakis, M. G.; Deb, K. e Engelbrecht, A. (2017). Seeking multiple solutions: An updated survey on niching methods and their applications. *IEEE Transactions on Evolutionary Computation*, v. 21, n. 4, p. 518–538.

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.

Liu, JyiShane e Sycara, Katia. (1994). Distributed problem solving through coordination in a society of agents. *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, p. 169–185, (1994).

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. C. e Stützle, T. (2003). Iterated local search. Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, Capítulo 11, p. 321–353. Kluwer Academic Publishers.

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.

Luke, Sean. (2017). ECJ then and now. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO'17, p. 1223–1230, New York, NY, USA. ACM. ISBN 978-1-4503-4939-0.

Lukin, Jonathan A.; Gove, Andrew P.; Talukdar, Sarosh N. e Ho, Chien. (1997). Automated probabilistic method for assigning backbone resonances of (13C,15N)-labeled proteins. *Journal of Biomolecular NMR*, v. 9, n. 2, p. 151–166.

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).

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.

Montgomery, D. C. e Runger, G. C. (2013). *Applied statistics and probability for engineers*. Wiley.

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.

Nebro, Antonio J.; Durillo, Juan J. e Vergne, Matthieu. (2015). Redesigning the jmetal multi-objective optimization framework. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion

'15, p. 1093–1100, New York, NY, USA. ACM. ISBN 978-1-4503-3488-4. doi: 10.1145/2739482.2768462. URL http://doi.acm.org/10.1145/2739482.2768462.

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.

Özcan, Ender; Bilgin, Burak e Korkmaz, Emin Erkan. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, v. 12, n. 1, p. 3–23.

Özcan, Ender e Kheiri, Ahmed. (2012). A hyper-heuristic based on random gradient, greedy and dominance. Gelenbe, Erol; Lent, Ricardo e Sakellari, Georgia, editors, *Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences*, p. 557–563. Springer London, London.

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, *Proceedings of the International Conference on Computational Science (ICCS 2003): Melbourne, Australia and St. Petersburg (Russia), June 2–4, 2003, Part IV*, volume 2660, p. 886–895. Springer, Berlin, Heidelberg.

Parejo, J. A.; Ruiz-Cortés, A.; Lozano, S. e Fernandez, P. (2012). Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, v. 16, n. 3, p. 527–561.

Potter, Mitchell A. e De Jong, Kenneth A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, v. 8, n. 1, p. 1–29.

Puchinger, J. e Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. Mira, José e Álvarez, José R., editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, p. 41–53. Springer Berlin Heidelberg.

Puterman, Martin L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, New York, NY.

Rabadi, Ghaith; Moraga, Reinaldo J. e Al-Salem, Ameer. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, v. 17, n. 1, p. 85–97.

Rabak, Cesar Scarpini e Sichman, Jaime Simão. (2003). Using A-Teams to optimize automatic insertion of electronic components. *Advanced Engineering Informatics*, v. 17, n. 2, p. 95–106.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. Almeida, Francisco; Aguilera, María J. Blesa; Blum, Christian; Vega, José Marcos Moreno; Pérez, Melquíades Pérez; Roli, Andrea e Sampels, Michael, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, p. 1–12. Springer Berlin Heidelberg.

Ramírez, Aurora; Romero, José Raúl; García-Martínez, Carlos e Ventura, Sebastián. (2019). JCLEC-MO: A java suite for solving many-objective optimization engineering problems. *Engineering Applications of Artificial Intelligence*, v. 81, p. 14 – 28. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2019.02.003. URL `http://www.sciencedirect.com/science/article/pii/S0952197619300260`.

Ramírez, Aurora; Romero, José Raúl e Ventura, Sebastián. (2015). An extensible JCLEC-based solution for the implementation of multi-objective evolutionary algorithms. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, p. 1085–1092, New York, NY, USA. ACM. ISBN 978-1-4503-3488-4. doi: 10.1145/2739482.2768461. URL `http://doi.acm.org/10.1145/2739482.2768461`.

Rodriguez, F. J.; Garcia-Martinez, C. e Lozano, M. (2012). Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: Taxonomy, comparison and synergy test. *IEEE Transactions on Evolutionary Computation*, v. 16, n. 6, p. 787–800.

Russell, S. e Norvig, P. (1995). *Artificial Intelligence: a Modern Approach*. Prentice-Hall.

Santos, Bruno André. Aspectos conceituais e arquiteturais para a criação de linhagens de agentes de *software* cognitivos situados. Dissertaccão de mestrado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brasil, (2003).

Silva, M. A. L.; de Souza, S. R.; Borges, H. E.; de Oliveira, S. M. e Temponi, E. C. C. (2007). Amam: Arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização. *Proceedings of the 8th Brazilian Symposium on Intelligent Automation (Simpósio Brasileiro de Automação Inteligente - SBAI)*, Florianópolis, 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 - Encontro Nacional 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. Modelagem de uma arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória (in portuguese). Dissertaccão de mestrado, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte, Brazil, (2007).

Silva, Maria Amélia Lopes; de Souza, Sérgio Ricardo; Souza, Marcone Jamilson Freitas e Bazzan, Ana Lucia 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. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2019.04.056. URL `http://www.sciencedirect.com/science/article/pii/S0957417419302866`.

Silva, Maria Amélia Lopes; de Souza, Sérgio Ricardo; Souza, Marcone Jamilson Freitas e de França 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. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2018.06.050. URL `http://www.sciencedirect.com/science/article/pii/S1568494618303867`.

Sislak, D.; Rehak, M. e Pechoucek, M. Sept(2005). A-globe: multi-agent platform with advanced simulation and visualization support. *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, p. 805–808, Sept(2005).

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, v. 2, n. 35, p. 254–264.

Souza, M. J. F.; Coelho, M. Igor; Ribas, S.; Santos, H. G. e Merschmann, L. H. C. (2010). A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, v. 207, n. 2, p. 1041–1051.

Subramanian, A.; Drummond, L. M.; Bentes, C.; Ochi, L. S. e Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computer & Operations Research*, v. 37, n. 11, p. 1899–1911.

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

Talbi, El-Ghazali. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, v. 8, n. 5, p. 541–564.

Talbi, El-Ghazali. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 1st edição.

Talukdar, S.; Baerentzen, L.; Gove, A. e de Souza, P. S. (1998). Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, v. 4, n. 4, p. 295–321.

Talukdar, S. e de Souza, P. S. (1990). Asynchronous Teams. *Proceedings of the Second SIAM Conference on Linear Algebra: Signals, System and Control, San Francisco, CA, USA, November 5-8, 1990*, (1990).

Talukdar, S.; Murthy, S. e Akkiraju, R. (2003). Asynchronous teams. Glover, Fred e Kochenberger, Gary A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 537–556. Springer US.

Toth, Paolo e Vigo, Daniele. (2002). *The Vehicle Routing Problem*. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, USA.

Toth, Paolo e Vigo, Daniele. (2014). *Vehicle Routing: Problems, Methods, and Applications*. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition edição.

Toutouh, J. e Alba, E. (2012). Parallel swarm intelligence for VANETs optimization. *Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, p. 285–290, (2012).

Trienekens, H. W. J. M. e Bruin, A. (1992). Towards a taxonomy of parallel branch and bound algorithms. Relatório Técnico Report EUR-CS-92-01, Erasmus University Rotterdam, Department of Computer Science.

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.

Vallada, Eva e Ruiz, Rubén. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 211, n. 3, p. 612–622. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.01.011. URL `http://www.sciencedirect.com/science/article/pii/S0377221711000142`.

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.

Villaverde, A.; Egea, J. e Banga, J. (2012). A cooperative strategy for parameter estimation in large scale systems biology models. *BMC Systems Biology*, v. 6, n. 1, p. 75.

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, M. (2009). *An Introduction to Multiagent Systems*. John Wiley & Sons, 2nd edição.

Yamaguchi, T.; Tanaka, Y. e Yachida, M. (1997). Speed up reinforcement learning between two agents with adaptive mimetism. *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1997 (IROS'97)*, volume 2, p. 594–600, (1997).

Zheng, Y.; Xu, X.; Chen, S. e Wang, W. Oct(2012). Distributed agent based cooperative differential evolution: A master-slave model. *Proceedings of the 2012 IEEE 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS)*, volume 1, p. 376–380, Oct(2012).

Żurek, Dominik; Piętak, Kamil; Pietron, Marciń e Kisiel-Dorohinicki, Marek. (2017). Toward hybrid platform for evolutionary computations of hard discrete problems. *Procedia Computer Science*, v. 108, p. 877 – 886. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.