CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Diretoria de Pesquisa e Pós-Graduação

Curso de Mestrado em Modelagem Matemática e Computacional

Arquitetura Multiagente Metaheurísticos Cooperativos para Resolução de Problemas de Otimização Combinatória

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

Aluno: Filipe Costa Fernandes
Bacharel em Ciência da Computação

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

Co-orientador: Prof. Dr. Henrique Elias Borges (CEFET/MG)

Belo Horizonte, setembro de 2009.

Dedico este trabalho à três pessoas de extrema importância em minha vida, meu pai, minha mãe e meu irmão. Natanael, Gildete e Thiago.

Agradecimentos

Segundo o Novo Dicionário Aurélio da Língua Portuguesa, agradecer, significa "Mostrar-se Grato", "Retribuir, recompensar". E neste momento de extrema felicidade, Deus sabe o quanto sou grato, às pessoas que contribuíram, de alguma foram, para a conclusão deste trabalho.

E por saber do tamanho de minha gratidão, primeiramente, agradeço a Deus. Agradeço por ter me proporcionado saúde, sabedoria, oportunidade, vida, amigos, família, uma namorada maravilhosa e um orientador fenomenal. Um conjunto de itens que foram fundamentais para a conclusão deste trabalho. Obrigado Senhor, por tudo.

Agradeço também a meus pais, Natanael e Gildete, que sempre estiveram do meu lado, apoiando em todas a minhas decisões e me orientando sempre que a certeza, de alguma forma, me faltava. Pai, mãe, sem vocês, com certeza, eu não teria conseguido. Obrigado por estarem sempre comigo. Amo muito vocês.

Ao meu irmão, Thiago, que sempre foi um exemplo de companheirismo, fidelidade e cumplicidade, e que a cada novo dia, me faz perceber o verdadeiro significado da palavra Irmão. Kiba, te amo meu irmão.

À minha namorada, Ivanna, que me apoiou e incentivou, desde o início, mesmo quando estive ausente, soube me entender e apoiar. Obrigado, meu amor, por tudo que vivemos juntos e por toda uma vida que ainda teremos pela frente.

Ao meu orientador, prof. Sérgio Ricardo, que me explicou em ações o por que dos orientados serem chamados de filhos de seus orientadores, sendo muito mais que um orientador, um pai que tenho na família Cefet. Obrigado pela paciência que sempre demonstrou ter, pela amizade construída e, não menos importante, por todo o conhecimento transmitido.

Ao meu co-orientador, prof. Henrique Elias, por todos os conselhos e intervenções, que foram fundamentais para a conclusão deste trabalho.

À minha Tia Gilzene e meus primos Cecília, Humberto e Roberta. Vocês sempre estiveram presentes em minha vida e foram o meu porto seguro nestes anos de mestrado, me receberam e me acolheram em todos os momentos, não tenho palavras para dizer o quanto sou grato e amo vocês. Obrigado.

Ao amigo Fábio Mourão, que mostrou ser o meu irmão na família Cefet. Obrigado por toda a contribuição fundamental para o meu desenvolvimento pessoal e profissional.

À amiga Maria Amélia, madrinha deste trabalho. Obrigado por todo apoio e auxílio no desenvolvimento deste trabalho.

Aos amigos do CEFET-MG, obrigado pelo companheirismo, amizade e auxílio em todos os momentos.

A todos os meus amigos e familiares que, de alguma forma, com certeza, contribuíram para o desenvolvimento deste trabalho.

E finalmente, ao Centro Universitário de Caratinga, pelo incentivo profissional e financeiro.

A todos vocês, meus sinceros agradecimentos.

"*Demorei 20 anos para fazer sucesso da noite para o dia.*"

Resumo

Este trabalho apresenta a Arquitetura MultiAgentes para Metaheurística versão 1.0 (AMAM v1.0), proposta para a solução de problemas de otimização combinatória utilizando metaheurísticas. Esta arquitetura combina diversas metaheurísticas, sendo cada uma delas definida como um agente autônomo, capaz de realizar a exploração do espaço de busca de um problema. O ambiente de atuação da arquitetura é o espaço de busca do problema em análise. Os agentes presentes na estrutura da arquitetura podem ser instanciados em diferentes configurações para a busca da solução do problema em questão. O comportamento da arquitetura é avaliado pela solução do Problema de Roteamento de Veículos com Janelas de Tempo. Para a solução deste problema, como forma de experimento, a arquitetura implementa, em forma de agentes, as metaheurísticas Algoritmo Genético, *Iterated Local Search* e *Variable Neighborhood Search*, bem como algumas variações de métodos de Busca Local, Construtores de Soluções e das técnicas de busca particulares de cada metaheurística implementada. Os resultados computacionais comprovam que a combinação de mais de um agente metaheurístico aumenta a eficiência da arquitetura, gerando, assim, resultados que são comparáveis aos melhores encontrados na literatura para a solução do problema tratado.

Palavras-Chave: Metaheurísticas Cooperativas, Problemas de Roteamento de Veículos, Multiagentes, Otimização Combinatória.

Abstract

This dissertation presents the Multi-Agent Architecture 1.0 version (MAM 1.0) , proposed for solving combinatorial optimization problems using metaheuristics. This architecture combines several metaheuristics, each one defined as an autonomous agent, capable to accomplish the exploration of the search space of a problem. The environment under which the architecture is acting is the search space of the problem under analysis. The agents in the structure of the Architecture can be set up in different ways for searching the solution of the problem in question. The proposed architecture is evaluated by solving the Vehicle Routing Problem with Time Windows. To solve this problem, the architecture implements, in the form of agents, the metaheuristics Genetic Algorithm, Iterated Local Search and Variable Neighborhood Search as well as some variations of methods of Local Search, Constructors of Solutions and of the search techniques peculiar of each implemented metaheuristic. The computational results show that the combination of more than one agent metaheuristic increases the efficiency of the architecture, generating, like this, results that are comparable to the best ones found in the literature for the addressed problem.

Keywords: Cooperative Metaheuristics, Vehicle Routing Problem, Multiagents, Combinatorial Optimization.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Problemas de otimização combinatória têm recebido grande atenção de pesquisadores, em especial devido a suas fortes ligações com aplicações do cotidiano. Ao lado disso, esta atenção de pesquisadores também se deve às características destes problemas, quanto à dificuldade de determinação de soluções de boa qualidade com um baixo custo computacional.

Assim, ao lado da continuidade do desenvolvimento de técnicas tradicionais de solução destes problemas, envolvendo métodos de programação matemática, novas técnicas de solução têm se estabelecido ou sido propostas, principalmente envolvendo conceitos inovadores no campo computacional.

Deste modo, técnicas heurísticas de solução desta classe de problemas de otimização são hoje fortemente utilizadas, com sucesso, para a busca de soluções de problemas de maior porte, envolvendo grande número de variáveis e restrições, nos quais as técnicas de programação exata ainda não obtém bons resultados. Ao lado disso, o uso de conceitos derivados de inteligência artificial têm se mostrado campo de pesquisa de forte interesse, com a aplicação, por exemplo, de estruturas computacionais na forma de agentes de software para a busca de soluções de problemas de otimização combinatória.

As metaheurísticas são algoritmos aproximados que combinam algoritmos construtivos, estratégias de busca local, estratégias para escapar de ótimos locais, dentre outras características. Estes algoritmos são amplamente utilizados na resolução de problemas de otimização combinatória, sendo que a maioria destes são considerados algoritmos difíceis de resolver, classificados como NP-Difícil (veja (Cormen et al., 2002), (Osman e Laporte, 1996) e (Ziviani, 2007)).

Desenvolvimentos importantes também se dão no uso de computação paralela e no desenvolvimento de algoritmos metaheurísticos que emulam conceitos da natureza, como as metaheurísticas de colônia de formigas, enxame de partículas, saltos de rãs e algoritmos evolutivos. Deve-se ressaltar que um dos problemas que têm sido mais estudados e utilizados como meio de desenvolvimento de novas técnicas de resolução de problemas é o Problema de Roteamento de Veículos, em especial em sua versão envolvendo Janelas de Tempo.

Os avanços recentes na área de sistemas multiagentes, agregado aos conceitos de agentes inteligentes e autômatos interagindo juntos com um objetivo comum, colocam a necessidade de simplificação e padronização destes conceitos visando melhorias e facilidades no desenvolvimento de sistemas de agentes inteligentes (Weiss,

1999).

Devido a grande dimensão dos problemas tratados, a tarefa de encontrar boas soluções no espaço de soluções factíveis do problema se torna muito mais fácil se estes agentes inteligentes cooperarem uns com os outros. Esta necessidade de cooperação vem se afirmando cada vez mais como uma técnica de grande interesse, principalmente em conjunto com a metodologia de sistemas multiagentes, que segundo (Wooldridge, 2002), são sistemas compostos de múltiplos elementos computacionais, definidos como agentes, em constante interação.

Em sistemas multiagentes, é extremamente importante que exista uma certa cooperação entre todos os agentes que atuam no mesmo ambiente, na tentativa de ampliar o espaço de busca individual e possibilitar melhor resultado global.

No presente trabalho, o interesse de estudo se concentra na aplicação de técnicas de cooperação entre Agentes metaheurísticas para a resolução de problemas de otimização combinatória. Deste modo, este trabalho se coloca como análise e continuidade de (Silva, 2007), em que a versão inicial da arquitetura AMAM foi apresentada.

Em seu trabalho intitulado "Modelagem de uma arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória" (AMAM), (Silva, 2007) define a versão inicial da arquitetura AMAM como um *framework* para combinar diversas metaheurísticas, sob a forma de Agentes de *software*, para a solução de problemas de otimização combinatória. O *framework* proposto proporciona uma estrutura híbrida de metaheurísticas, possibilitando a inserção de diversas metaheurísticas para trabalhar em um mesmo ambiente. Para a comunicação entre os Agentes presentes na Arquitetura AMAM, (Silva, 2007) propôs a definição de uma estrutura de simples trocas de sinais entre os mesmos, indicando caminhos que deveriam ser evitados, pois apresentam fortes evidências que levam à soluções ruins. Esta estratégia possibilita uma comunicação direta entre os Agentes metaheurísticas mas não proporciona cooperação entre os mesmos, no sentido de obter ganhos nos resultados encontrados através da troca de informações relevantes para a obtenção de boas soluções para problemas de otimização combinatória.

Para que os Agentes presentes na Arquitetura AMAM proposta por (Silva, 2007) trabalhem conjuntamente, na forma de uma equipe (uma das característica de Sistemas Multiagentes), é necessário a definição de uma estrutura eficiente de cooperação entre estes Agentes.

Assim, o presente trabalho apresenta, como contribuição, além de uma análise de inserção de novos Agentes Metaheurísticos na Arquitetura AMAM, uma estrutura cooperativa entre os Agentes, que define regras e negociações para a troca de informações relevantes para auxiliar os Agentes presentes na Arquitetura na busca por melhores soluções para problemas de otimização combinatória.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é a definição de uma estrutura cooperativa entre metaheurísticas, aplicada à Arquitetura MultiAgentes para Metaheurísticas

(AMAM), primeiramente proposta em (Silva, 2007), para resolução do Problema de Roteamento de Veículos com Janelas de Tempo. Neste sentido, este trabalho se coloca como uma continuidade e uma crítica ao trabalho de (Silva, 2007).

Geralmente, em sistemas de metaheurísticas cooperativas, cada metaheurística é definida como um agente em busca de regiões promissoras no espaço de busca do problema. Sendo assim, a estrutura proposta neste trabalho pode ser considerada como uma Arquitetura Multiagentes Metaheurísticos Cooperativos para Resolução de Problemas de Otimização Combinatória.

Esta Arquitetura Multiagentes Metaheurísticos Cooperativos tem como principal objetivo guiar estes agentes metaheurísticas na exploração e diversificação do espaço de busca, visando melhorar seus resultados e/ou tempo de busca por regiões ainda mais promissoras. Além disso, visa também integrar diversas metaheurísticas de forma cooperativa, e verificar até que ponto esta cooperação traz benefícios para a resolução do problema.

A idéia principal desta cooperação é o compartilhamento de informações, utilizando uma estratégia de memória adaptativa compartilhada, que, por sua vez, armazena, gerencia e distribui estas informações de forma inteligente e eficiente.

1.1.2 Objetivos Específicos

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

(i) Analisar e testar todos os aspectos de desenvolvimento da Arquitetura Multi-Agentes para Metaheurísticas apresentada em (Silva, 2007), com o objetivo de compreender todas as características de funcionamento e desenvolvimento desta arquitetura.

(ii) Uma revisão de literatura sobre os temas envolvidos neste trabalho: análise, projeto e desenvolvimento das principais técnicas de metaheurísticas cooperativas usadas na resolução de problemas de otimização combinatória;

(iii) Conhecer e implementar as principais metaheurísticas, mas principalmente as mais utilizadas na resolução do problema de roteamento de veículos com janela de tempo.

(iv) Implementação desta estrutura de comunicação/cooperação de metaheurísticas para testar e validar a estrutura.

(v) Análise e comparação dos resultados com os melhores resultados encontrados na literatura para este tipo de problema.

1.2 Justificativa

Os problemas de otimização combinatória de classe NP-difícil são problemas complexos para os quais não existem, até o momento, algoritmos que os resolvam em tempo polinomial. A abordagem mais amplamente utilizada na literatura para

resolver este tipo de problemas é por métodos heurísticos, que possuem fácil implementação e conseguem encontrar "boas" soluções com tempo computacional reduzido.

Com a grande evolução dos computadores e conseqüentemente das redes de computadores, áreas como Sistemas Distribuídos e Inteligência Artificial Distribuída passam a ter importância significativa no estabelecimento de novos paradigmas de solução de problemas. Uma destas formulações é a utilização simultânea de diferentes técnicas para a busca de melhores soluções de problemas como os citados problemas de classe NP-difícil.

No presente trabalho, cada técnica de solução, na forma de metaheurística, é tratada como um agente de *software*. Problemas reais, de forma geral, possuem grandes dimensões e alta complexidade para serem resolvidos por um único agente, pois estes são limitados em sua "esfera de influência". A adoção de sistemas multiagentes busca suprir estas dificuldades, em especial quando na forma de sistemas cooperativos.

Em sistemas multiagentes, para que estas ações possam ser organizadas e o objetivo comum alcançado, devem existir arquiteturas de coordenação e cooperação. É com a cooperação que os agentes poderão ampliar seus espaços de busca, através da troca de informações que cada agente consegue individualmente através de sua esfera de influência. Com a utilização de técnicas cooperativas os agentes conseguem visualizar uma área de busca muito maior, obtendo mais diversificação nas soluções encontradas. E, neste sentido, este trabalho propõem a definição de uma estrutura cooperativa entre metaheurísticas.

1.3 Metodologia

Para o desenvolvimento deste trabalho e definição da estrutura de cooperação é preciso inicialmente identificar todos os componentes envolvidos neste processo. Neste sentido, as principais propostas de estruturas de cooperação devem ser analisadas para fornecer fundamentos e técnicas adequadas para a elaboração desta estrutura de cooperação.

Uma revisão de literatura, relacionada a metaheurísticas cooperativas, permitirá identificar e distinguir as diferentes técnicas de cooperação e as principais metaheurísticas utilizadas para este propósito.

Após análise e projeto da estrutura de metaheurísticas cooperativas, serão analisados seu correto funcionamento e performance com relação a resultados, para verificar sua eficiência e validação de seus objetivos. Na implementação e execução destas aplicações será utilizada a plataforma Java, versão J2SE 5.0, utilizando o IDE Eclipse, versão 3.4.

A análise dos resultados obtidos nos testes destas aplicações permitirá a avaliação do desempenho da arquitetura e sua possível validação.

1.4 Estrutura da Dissertação

O presente trabalho está organizado da seguinte forma. No capítulo 2 são descritas as características das principais heurísticas e metaheurísticas para resolução de

problemas de otimização combinatória; no capítulo 3 é realizada revisão de literatura a respeito de heurísticas cooperativas e hiperheurísticas; o capítulo 4 apresenta as principais características de Sistemas Multiagentes e seus principais componentes, bem como uma revisão de literatura a respeito de propostas que utilizam Sistemas Multiagentes; no capítulo 5 é apresentada a Arquitetura AMAM, suas derivações e a estrutura cooperativa proposta para esta arquitetura; o capítulo 6 descreve os experimentos computacionais realizados com a arquitetura AMAM, além dos seus resultados e análise. Finalmente, o capítulo 7 conclui este trabalho e apresenta perspectivas de trabalhos futuros.

Capítulo 2

Heurísticas e Metaheurísticas

2.1 Caracterização Geral

A grande maioria dos problemas que conhecemos e estudamos são classificados, em termos de complexidade computacional, como problemas polinomiais, pois podem ser solucionados através de algoritmos determinísticos em tempo polinomial, tendo, geralmente, função de complexidade computacional $\mathcal{O}(p(n))$, sendo $p(n)$ um polinômio, ou, por outro lado, como problemas não-polinomiais, que podem ser solucionados através de algoritmos não-determinísticos em tempo polinomial tendo, geralmente, função de complexidade $\mathcal{O}(c^n)$, para $c > 1$ (Ziviani, 2007) (Hoos e Stützle, 2005).

Os problemas de Otimização Combinatória são classificados, em sua maioria, como problemas não-polinomiais em termos da complexidade computacional dos algoritmos que os solucionam, sendo considerados algoritmos difíceis de resolver e classificados como problemas de classe NP-difícil (Osman e Laporte, 1996). Esta classe de problemas pode ser solucionada por algoritmos não-determinísticos em tempo polinomial.

Para a solução de problemas desta classe, uma ferramenta importante têm sido os procedimentos heurísticos. Diversas definições são encontradas na literatura a respeito desta classe de algoritmos.

Do ponto de vista etimológico, a palavra "Heurística" advém do verbo grego "*heuriskein*" ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$), que significa "buscar" ou "encontrar".

Em termos computacionais, as heurísticas são algoritmos não-determinísticos, com capacidade de tomar decisões sobre estratégias de busca, geralmente usadas para guiar o processo de busca de soluções para um determinado problema, ou seja, são algoritmos aproximados que combinam algoritmos construtivos com estratégias de busca local, e não garantem a solução ótima do problema, mas geralmente, encontram boas soluções (Hoos e Stützle, 2005).

As metaheurísticas são algoritmos aproximados que combinam algoritmos construtivos, estratégias de busca local, estratégias para escapar de ótimos locais, dentre outras características. Estes algoritmos são amplamente utilizados na resolução de problemas de otimização combinatória (Roli e Milano, 2001). Em linhas gerais, as metaheurísticas são compostas de estratégias de alto nível para explorar o espaço de busca usando diferentes técnicas de busca.

O termo Metaheurística deriva da composição de dois termos gregos: a palavra

"Heurística", proveniente do verbo "*heuriskein*", que significa "buscar" ou "encontrar"; e o prefixo "*Meta*", que significa "além de" ou "em um nível superior" (Pelta, 2005).

Osman e Laporte (1996) apresentam um conceito de metaheurísticas que se adequa corretamente aos interesses deste trabalho:

> "*Uma metaheurística é definida como um processo de geração iterativo, que guia uma heurística subordinada, pela combinação de diferentes conceitos inteligentes para explorar e expandir o espaço de busca. Estratégias de aprendizagem são usadas para estruturar informações com a finalidade de encontrar de forma eficiente soluções próximas do ótimo.*"

Em (Blum e Roli, 2003) são apresentadas algumas propriedades fundamentais acerca de metaheurísticas, que seguem:

(i) Metaheurísticas são estratégias que guiam o processo de busca;

(ii) O objetivo é explorar eficientemente o espaço de busca para encontrar soluções (próximas de) ótimas;

(iii) Técnicas que constituem algoritmos de metaheurísticas variam de um simples procedimento de busca local a processos complexos de aprendizagem;

(iv) Metaheurísticas são algoritmos aproximados e normalmente não-determinísticos;

(v) Elas devem incorporar mecanismos para evitar ficar presas em áreas limitadas do espaço de busca;

(vi) Metaheurísticas não são voltadas para a solução de problemas específicos;

(vii) Metaheurísticas podem usar estratégias específicas de heurísticas para atingir um nível superior na diversificação da busca;

(viii) Muitas metaheurísticas avançadas usam experiências de busca (baseadas em algum tipo de memória) para guiar sua busca.

Dois conceitos muito importantes em metaheurísticas são intensificação e diversificação. Segundo Blum e Roli (2003) e Hoos e Stützle (2005), estes são os pontos principais que determinam o comportamento de uma metaheurística, caso um equilíbrio dinâmico seja estabelecido entre eles. A diversificação é expressa pela exploração do espaço de busca, em termos do aumento do número de soluções diferentes encontradas através de variadas técnicas que são utilizadas. A intensificação implica na exploração exaustiva da vizinhança de uma solução, resultando, geralmente, em um ótimo local.

Assim, de forma objetiva, a principal finalidade das metaheurísticas é guiar a exploração do espaço de busca utilizando diversas estratégias. Neste sentido, considere um problema de otimização combinatória, definido por um conjunto de variáveis $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$; um conjunto de restrições sobre as variáveis; e uma função objetivo. O espaço de busca $\mathcal{S}$ é, então, o conjunto de todas as soluções s factíveis

para o problema, ou seja, o conjunto de todas as soluções que satisfazem o conjunto de restrições (Birattari et al., 2001).

Neste ponto, é importante diferenciar métodos heurísticos quanto à forma como geram soluções de um determinado problema, ou seja, distinguir entre heurísticas construtivas e heurísticas de busca local. Uma heurística construtiva tem a função de construir soluções para um determinado problema, situando-as em um ponto do espaço de busca, *i.e.*, tem a função de encontrar uma solução factível para o problema. Estas heurísticas, geralmente, constroem as soluções de forma incremental, ou seja, geram a solução adicionando a ela um elemento de cada vez, sendo este elemento, por sua vez, uma variável do problema. Já as heurísticas de busca local, também conhecidos como métodos de melhoramento iterativo, partem de uma solução inicial, pertencente ao espaço de busca de um determinado problema, e buscam substituir esta solução por uma solução melhorada, pertencente a uma vizinhança desta solução.

Uma vizinhança ou uma estrutura de vizinhança é uma função $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ que associa a cada $s \in \mathcal{S}$ um conjunto de vizinhos $\mathcal{N}(s) \subseteq \mathcal{S}$. Neste caso, o conjunto $\mathcal{N}(s)$ é chamado de vizinhança de s. Neste sentido, o conceito de vizinhança indica soluções passíveis de serem determinadas fruto de uma simples busca, ou seja, soluções que, através de um único movimento (adição, remoção ou modificação de um ou mais podem componentes da solução), podem ser convertidas umas nas outras. Por exemplo, no caso do Problema de Roteamento de Veículos, a troca de duas cidades clientes em uma solução determina a obtenção de uma solução vizinha à corrente (Hoos e Stützle, 2005) (Blum e Roli, 2003).

As metaheurísticas também possuem estratégias para escapar de ótimos locais. Estas estratégias são particularidades de cada metaheurística e serão tratadas nas seções 2.3 e 2.4. São estas estratégias para escapar de ótimos locais que diferem as metaheurísticas das heurísticas, sendo que estas últimas não possuem tal habilidade.

A diferenciação entre pontos de ótimo local e pontos de ótimo global de um dado problema de otimização está associada à vizinhança considerada. A figura 2.1 ilustra o comportamento de uma função contínua e um ponto de ótimo local e outro de ótimo global de um problema de minimização. Observe que o ponto de ótimo local é um valor mínimo para sua vizinhança, enquanto o ponto de ótimo global representa um valor mínimo para todo o conjunto de soluções factíveis do problema em tela.

2.2 Classificação de Metaheurísticas

As metaheurísticas podem ser classificados de várias formas, de acordo com características particulares a serem enfatizadas. Uma classificação importante, apresentada a seguir, se refere à forma como os métodos foram desenvolvidos e sua relação com processos biológicos:

- métodos "inspirados na natureza": neste caso se incluem metaheurísticas que têm origem em processos naturais (biológicos ou não) ou emulam tais processos. Podem ser incluídos nesta classe as metaheurísticas Algoritmo Genético, Colônia de Formigas, Enxame de Partículas, Saltos de Rãs e *Simulated Annealing*, dentre outras, além de suas diversas variações.

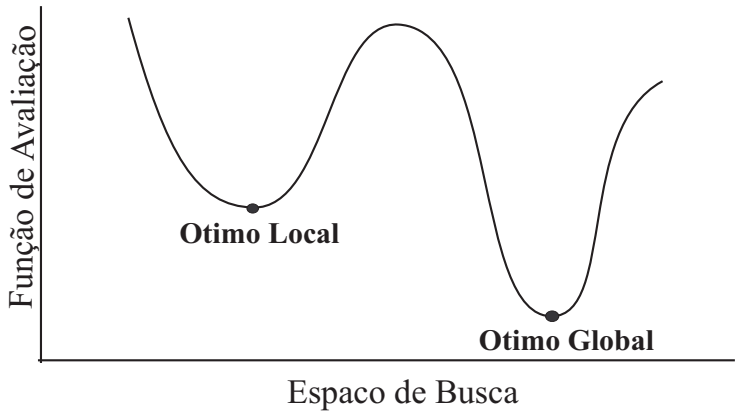

Figura 2.1: Ótimo Local e Ótimo Global de um problema de minimização de uma função contínua.

- métodos "não inspirados na natureza": neste caso são incluídas as metaheurísticas que não possuem relação ou não têm origem em processos naturais, quais sejam, metaheurísticas como Busca Tabu, *Greedy Randomized Adaptive Search Procedure* (GRASP), *Iterated Local Search* (ILS), *Variable Neighborhood Search* (VNS), dentre outras, incluindo suas variações.

Esta classificação, segundo Blum e Roli (2003), é frágil, pois, de um lado, torna-se difícil classificar um certo algoritmo como associado a uma destas classes; de outro, ela não consegue incluir o caso de algoritmos que possuem características tanto naturais quanto não-naturais. Nesta última situação, estão postos os algoritmos híbridos.

Uma outra classificação diz respeito à forma de evolução dos algoritmos, ou melhor, do número de soluções com as quais lidam ao mesmo tempo.

- métodos "baseados em população": utilizam de um conjunto de soluções (denominada "população de soluções") em seu processo de busca, aplicando suas características nestas soluções até o final de sua operação e determinando a evolução de um conjunto de pontos no espaço de busca. Estes métodos, geralmente, são inspirados na natureza. Alguns exemplos são as metaheurísticas Algoritmos Genéticos, Colônia de Formigas, Enxame de Partículas e Saltos de Rãs, dentre outras.

- métodos de "trajetória" ou de "ponto único": estes métodos utilizam uma única solução no processo de busca, aplicando todas as suas características nesta mesma solução até o final de sua operação. Incluem-se neste caso as metaheurísticas de Busca Tabu, *Greedy Randomized Adaptive Search Procedure* (GRASP), *Iterated Local Search* (ILS), *Variable Neighborhood Search* (VNS) e *Simulated Annealing*, dentre outras.

Esta forma de classificação é a que mais se adequa aos interesses deste trabalho, pois, além de permitir uma clara descrição do comportamento da metaheurística, abre-se também para o tratamento de algoritmos híbridos, incluindo características tanto de métodos populacionais quanto de métodos de trajetória.

A seguir, são descritos os principais métodos de busca para problemas de otimização, classificados de acordo com as definições postas acima.

2.3 Métodos de Trajetória

Esta seção revisa os principais métodos de trajetória. Ela se inicia com a apresentação de um método de busca local simples, que contém a formulação básica de um procedimento de busca na forma de trajetória. Em seguida, são apresentadas estruturas mais complexas, na forma de metaheurísticas para a solução de problemas de otimização.

2.3.1 Método de Busca Local Simples

O Método de Busca Local Simples procura, a cada iteração, uma solução melhor que a solução corrente do problema. Esta característica é normalmente chamada de "melhoramento iterativo", sendo que melhora a solução a cada iteração até que algum critério de parada seja satisfeito (Blum e Roli, 2003).

Este método é baseado na busca em vizinhança. Partindo de uma solução inicial, ele procura pelo melhor vizinho, de acordo com a função objetivo do problema. O pseudo-algoritmo deste método é apresentado na figura 2.2.

```
Algoritmo Busca_Local_Simples(·)
1  inicio
2      s ← Gera_Solucao_Inicial(·);
3      repita
4              s ← Melhora(s, N(s));
5      até que não consiga melhorar s;
6  fim.
```

Figura 2.2: Pseudo-algoritmo Busca Local Básica.

Neste pseudo-algoritmo, s recebe uma solução inicial. Em seguida, transformações de movimentos a partir de s na vizinhança $\mathcal{N}(s)$ são realizados pela função $Melhora(s, \mathcal{N}(s))$, para visitar outras soluções em busca de soluções melhores que a existente, até que um critério de parada seja satisfeito. A função $\mathcal{N}(s)$ é uma função de vizinhança do problema tratado.

2.3.2 *Simulated Annealing*

No processo de recozimento (*annealing*) de metais, o metal é aquecido a altíssimas temperaturas e resfriado posteriormente. O resfriamento rápido conduz a micro-estrutura do metal a um estado aleatóriamente instável, mas, por outro lado, se este resfriamento for lento, pode conduzir a um ponto de equilíbrio caracterizado por produtos mais estáveis e estruturalmente fortes (Kirkpatrick et al., 1983).

(Kirkpatrick et al., 1983) propuseram que a simulação desse processo poderia ser usada em problemas de otimização combinatória na busca por melhores soluções, onde os estados possíveis de um metal correspondem a soluções do espaço de busca do problema de otimização.

No processo de recozimento, a micro-estrutura do metal passa por vários estados possíveis e a energia em cada estado pode ser comparada com o valor da função

objetivo em problemas de otimização. A energia mínima (ou máxima) corresponde à solução ótima, local ou global. A figura 2.3 ilustra este procedimento para um problema de minimização.

```
Algoritmo Simulated Annealing(·)
1  inicio
2      s_0 ← Gera_Solucao_Inicial(·);
3      T ← T_0;
4      enquanto Critério de Parada não satisfeito;
5          s' ← Gera_Vizinho(s);
6          se f(s') < f(s)
7                  s ← s';
8          senao
9                  Aceita s' como nova solução, com probabilidade p(T, s', s);
10         fim se;
11         atualiza T;
12     fim enquanto;
13 fim.
```

Figura 2.3: Pseudo-algoritmo *Simulated Annealing*.

No pseudo-algoritmo apresentado na Figura 2.3, o processo de busca inicia com uma solução inicial para o problema de minimização a ser resolvido. A partir deste ponto, o pseudo-algoritmo entra em um *loop* que será executado até que algum critério de parada seja satisfeito, da seguinte forma: s' recebe um vizinho definido por uma estrutura de vizinhança do *Simulated Annealing* em s. Caso s' apresente uma melhora com relação à s, s' passa a ser a nova solução, caso contrário, s' pode vir a ser a nova solução de acordo com uma função probabilística que leva em consideração os valores de s, s' e o parâmetro de temperatura T. Após esta definição de nova solução, o parâmetro T é atualizado de acordo com características de movimentos e aceitação do método *Simulated Annealing*.

2.3.3 Busca Tabu

A metaheurística Busca Tabu - *Tabu Search* - tem como principais características, memória adaptativa e estratégias de busca baseadas em memória. Proposta por (Glover, 1986), a Busca Tabu aceita movimentos de piora da solução a fim de escapar das armadilhas dos ótimos locais.

A idéia da metaheurística Busca Tabu é utilizar, junto à busca local, uma Lista Tabu de movimentos para armazenar as últimas soluções visitadas ou soluções proibidas.

Como em todo processo de busca, este começa com uma solução inicial e, a cada iteração, explora a vizinhança da solução corrente. Novas soluções são geradas através da busca local e a cada iteração, usa-se uma Lista Tabu que armazena as soluções já visitadas, possibilitando, assim, uma espécie de perturbação na solução, sem permitir que soluções já visitadas se repitam, para diversificar as soluções e a tentativa de escapar de ótimos locais. Se o tamanho máximo da Lista Tabu for

```
Algoritmo BuscaTabu(·)
1  inicio
2      s_0 ← Gera_Solucao_Inicial(·);
3      Inicializa_Lista_Busca_Tabu(LT);
4      enquanto Critério de Parada não satisfeito;
5          Determine melhor vizinho que use um movimento não pertencente à LT;
6          s ← Melhor_Vizinho;
7          Atualiza_Lista_Tabu(LT);
8          Atualiza_Melhor_Solucao(s);
9      fim enquanto;
10 fim.
```

Figura 2.4: Pseudo-algoritmo Busca Tabu.

atingido, e não for encontrado um movimento de melhora, pode-se aceitar a solução corrente ou aumentar o tamanho da Lista Tabu, produzindo saltos no espaço de busca, na tentativa de encontrar melhores soluções.

Como ilustrado na figura 2.4, o pseudo-algoritmo Busca Tabu gera uma solução inicial e através dela a Lista Tabu. Enquanto algum critério de parada não for satisfeito (o critério de parada pode ser o número máximo de iterações ou a quantidade de iterações sem melhora), é gerada uma Lista de Movimentos Permitidos que pertencem à vizinhança da solução atual e que não violam a Lista Tabu. A nova solução atual será obtida da melhor solução encontrada na Lista de Movimentos Permitidos e por fim a Lista Tabu será atualizada, de acordo com a vizinhança da nova solução atual.

2.3.4 *Greedy Randomized Adaptive Search Procedure* (GRASP)

GRASP (*Greedy Randomized Adaptive Search Procedure* ou Procedimento de Busca Gulosa Adaptativa Aleatória) é uma metaheurística de soluções aproximadas para problemas de otimização combinatória. Este procedimento foi proposto em Feo e Resende (1995), referência na qual uma descrição mais detalhada deste método deve ser buscada.

A metaheurística GRASP é um processo iterativo, em que cada iteração consiste em duas fases distintas, a fase de construção e a fase de busca local. Na fase de construção, em que uma solução inicial factível é gerada, uma lista restrita de candidatos a fazer parte da solução é gerada iterativamente elemento por elemento. Esta lista contém um conjunto reduzido de elementos candidatos a pertencer à solução. Como os benefícios associados a cada elemento são atualizados a cada iteração, esta heurística é também chamada de heurística adaptativa (Blum e Roli, 2003). Na fase de busca local, é pesquisado um ótimo local na vizinhança da solução inicial que foi gerada por elementos aleatórios da lista restrita de candidatos. O método GRASP retorna como resultado a melhor solução encontrada dentre todas as iterações realizadas. A figura 2.5 apresenta um pseudo-algoritmo GRASP que ilustra este processo para um problema de minimização.

Em (Roli e Milano, 2001) é apresentada análise da metaheurística GRASP na

```
Algoritmo GRASP(·)
1  inicio
2  x ← ∞;
3  enquanto Critério de Parada não satisfeito;
4      x' ← Solucao_inicial_gulosa_aleatoria;
5      x'' ← Busca_Local(x');
6      se(x'' < x)entao
7          x ← x'';
8      fim se
9  fim enquanto;
10 retorne x;
11 fim.
```

Figura 2.5: Pseudo-algoritmo GRASP.

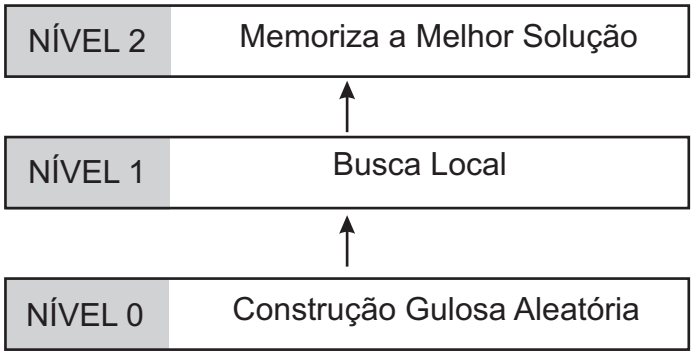

Figura 2.6: Arquitetura GRASP multi-nível (Roli e Milano, 2001).

forma de uma arquitetura multi-nível, que define três fases para este processo iterativo. Esta arquitetura multi-nível, mostrada na figura 2.6, define o nível 0 como sendo a fase de Construção Gulosa Aleatória, na qual uma solução inicial do problema é gerada; o nível 1 é definido pela busca local, aplicada à solução inicial gerada no nível 0; e, por fim, o nível 2 atualiza a solução atual, memorizando a melhor solução até o momento.

2.3.5 *Variable Neighborhood Search* (VNS)

O método de busca em vizinhança variável - VNS (*Variable Neighborhood Search*) - é uma metaheurística para resolução de problemas de otimização combinatória, sendo sua idéia básica uma busca local que explora o espaço de busca do problema realizando trocas sistemáticas da estrutura de vizinhança como estratégia de escapar de ótimos locais (Hansen et al., 2003).

Este método, a cada iteração, realiza uma busca local na solução corrente e compara o resultado desta busca local com a própria solução corrente em questão. Caso uma melhora não seja encontrada, a função de vizinhança é alterada (Hansen e Mladenovic, 2001).

Segundo (Hansen e Mladenovic, 1999), a metodologia VNS não é um método de trajetória, já que, através da troca de vizinhança, é capaz de explorar diferentes

partes do espaço de busca, realizando saltos durante a exploração, conseguindo assim, explorar vizinhanças gradativamente mais "distantes". O pseudo-algoritmo VNS, que demonstra este processo para problemas de minimização, é apresentado na figura 2.7.

```
Algoritmo VNS()
1  inicio
2      s ← Gera_Solucao_Inicial(·);
3      vizinhanca ← 1;
4      enquanto Critério de parada não satisfeito
5          s' ← Gera_Vizinho(s, vizinhanca);
6          s'' ← Busca_Local(s');
7          se f(s'') < f(s')
8              s ← s'';
9              vizinhanca ← 1;
10         senão
11             Troca_Vizinhanca(vizinhanca);
12         fim se;
13     fim enquanto;
14     retorne s;
15 fim.
```

Figura 2.7: Pseudo-algoritmo VNS.

2.3.6 *Iterated Local Search* (ILS)

A metaheurística *Iterated Local Search* (ILS) é um método de busca iterativo que utiliza uma técnica de perturbações (modificações ou troca de vizinhança) de solução para escapar de ótimos locais correntes, ou visitar outros ótimos locais. Este método constrói soluções iterativamente, realizando perturbações em algumas soluções já encontradas como principal objetivo a diversificação da busca (Lourenço et al., 2002).

O método ILS pode ser definido por quatro componentes básicos: Geração da Solução Inicial; Busca Local; Pertubação; e Critério de Aceitação. Na Geração da Solução Inicial, uma solução inicial factível ao problema é gerada e passa a ser o ponto inicial do processo de busca. O componente de Busca Local, refina uma determinada solução para encontrar seu ótimo local. A perturbação é a modificação da solução corrente, guiando-a a uma solução intermediária. Esta perturbação deve permitir que o método explore diferentes soluções no espaço de busca do problema. O Componente Critério de Aceitação seleciona as novas soluções e decide em qual delas serão aplicadas as próximas pertubações (Lourenço et al., 2002).

A figura 2.8, exemplifica o processo de busca da metaheurística ILS. Nesta figura uma solução sofre uma perturbação, ou seja, um salto no espaço de busca do problema, obtido pela troca de vizinhança, gerando uma solução s_1. Nesta nova solução, encontrada no processo de pertubação, é aplicada uma busca local, definida no método, e encontra-se uma nova solução s_2, que é um ótimo local na vizinhança atual.

O processo se repete até que um determinado critério de parada seja estabelecido e a cada iteração o melhor ótimo local é armazenado como melhor solução do problema. O pseudo-algoritmo da metaheurística ILS é mostrado na figura 2.9 e demonstra todos os componentes desta metaheurística.

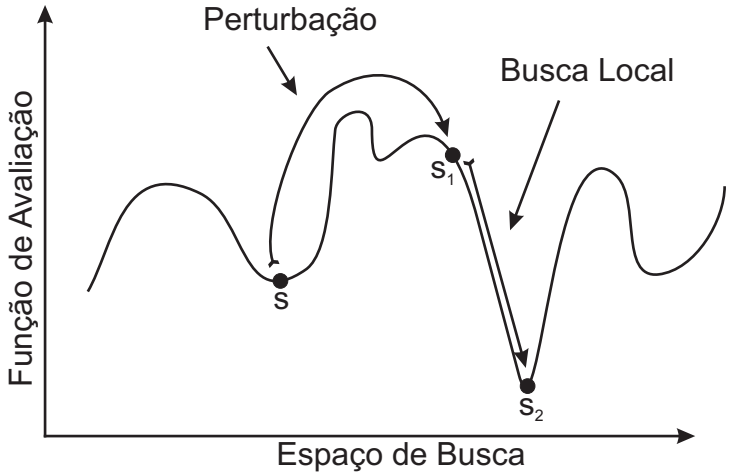

Figura 2.8: Funcionamento da Metaheurística ILS.

Algoritmo $ILS()$
1 **inicio**
2 $s_0 \leftarrow Gera\_Solucao\_Inicial(\cdot)$;
3 $s \leftarrow Busca\_Local(s_0)$;
4 **repita**
5 $s_1 \leftarrow Perturbacao(s, histrico)$;
6 $s_2 \leftarrow Busca\_Local(s_1)$;
7 $s \leftarrow Criterio\_Aceitacao(s_1, s_2, historico)$;
8 **ate que** *Critério de parada seja satisfeito*;
9 **retorne** s
10 **fim**.

Figura 2.9: Pseudo-algoritmo ILS.

2.4 Métodos baseados em População

2.4.1 Algoritmos Genéticos

John Holland acreditava que a incorporação das características naturais de evolução em um computador poderia produzir uma técnica para solucionar problemas da mesma maneira como funcionam na natureza os processos de seleção e adaptação das espécies. Baseado nessa idéia, John Holland, junto aos seus alunos e colegas da Universidade de Michigan, criaram o Algoritmo Genético (Holland et al., 1992).

Um algoritmo genético (AG) pode ser entendido como um algoritmo de pesquisa probabilístico "inteligente", que pode ser aplicado em uma grande variedade de problemas de otimização. Os AGs simulam o processo de evolução de uma população

de estruturas, sujeitas às forças competitivas prescritas nos princípios da seleção natural e da "sobrevivência do mais bem adaptado" de Charles Darwin, onde os indivíduos mais bem adaptados possuem uma melhor chance de sobreviver e de se reproduzir, enquanto os indivíduos com pior adaptação tendem a ser eliminados. Os genes dos indivíduos mais bem adaptados tendem a aparecer em um número maior de indivíduos nas gerações futuras, onde a combinação de suas características tende a gerar populações melhores. Sendo assim, o processo de evolução é aleatório, porém guiado por um mecanismo de seleção baseado na adaptação de estruturas individuais.

O processo de funcionamento de um AG consiste em gerar uma população inicial com N indivíduos e aplicar repetidamente os operadores genéticos de reprodução, cruzamento e mutação até que se encontre um critério para o fim da evolução. Em termos de otimização, estes indivíduos na população são codificados por uma *string* - cadeia de caracteres ou de bits - chamada cromossomo, que representa uma possível solução para o problema. A evolução dos indivíduos na população é guiada pela função de adaptação, também chamada *fitness function* ou função de avaliação, que determina o valor de cada cromossomo. O pseudocódigo do Algoritmo Genético é descrito na Figura 2.10.

```
Algoritmo AG()
1  inicio
2      t ← 0;
3      Gera_Populacao_InicialP(t);
4      avalia P(t);
5      enquanto Critério de parada não satisfeito
6          t ← t + 1;
7          P' ← Selecione pais de P(t − 1);
8          P'' ← Cruzamento(P');
9          P'' ← Mutação(P'');
10         avalia P'';
11         Atualiza P(t) a partir de P(t − 1) e P'';
12     fim enquanto;
13 fim.
```

Figura 2.10: Pseudo-algoritmo Algoritmo Genético.

Neste algoritmo, a população inicial, onde cada indivíduo é uma solução do problema, é gerada em $P(t)$, sendo t um determinado instante (ou geração).

A partir deste ponto o algoritmo entra em um *loop* que será executado até que algum critério de parada seja satisfeito. A cada iteração deste loop a população (t) (ou geração) é incrementada, para que os operadores genéticos gerem novos indivíduos, para a população $P(t)$, a partir da população anterior $P(t-1)$. Nesta fase são implementados *Crossover* e Mutação. A população é novamente avaliada e a nova população, chamada de PopulaçãoSobrevivente, passará a ser a população atual, para assim então repetir o processo.

Como dito anteriormente, após a fase de construção, que gera uma solução inicial para o problema, a população será avaliada. O processo de avaliação determina

o quanto um indivíduo está adaptado ao problema, diferenciando indivíduos que representam boas soluções daqueles que estão longe de ser uma boa solução do problema. Na população, cada indivíduo é avaliado utilizando uma função de avaliação que determina o grau de adaptação do mesmo, ela também é chamada de função objetivo em um grande número de problemas, já que em problemas de otimização a própria função objetivo pode ser usada como medida de avaliação.

O operador de reprodução, *Crossover*, atua como na biologia, onde os indivíduos mais bem adaptados possuem mais chance de sobreviver a predadores, a pragas e a outros obstáculos que dificultam os indivíduos chegarem à fase adulta para se reproduzirem. No Algoritmo Genético estes indivíduos, também chamados de pais, podem ser selecionados aleatoriamente, ou seja, a cada nova população são escolhidos dois entre todos os indivíduos existentes na população.

Destes dois indivíduos escolhidos, outros dois serão gerados, chamados de indivíduos filhos, esta é a fase de cruzamento. Um processo conhecido na biologia como crossover se define em cruzar dois indivíduos (pais) selecionados e gerar dois novos indivíduos (filhos) a partir da combinação dos genes dos pais.

Embora o seu uso não seja obrigatório, o operador de mutação é útil para garantir a biodiversidade da população, assegurando que o espaço de busca provavelmente será explorado em uma parte significativa de sua extensão.

O processo de atualização ou seleção determina entre os indivíduos novos e antigos quais farão parte da nova população.

O Algoritmo Genético repete estes processos (seleção de pais, *crossover*, mutação e atualização) até que o critério de parada seja satisfeito, que pode ser baseado no número de iterações ou a população atingir uma certa estagnação com relação à sua função objetivo, e seus indivíduos não conseguirem mais evoluir.

2.4.2 *Ant Colony Optimization*

A metaheurística *Ant Colony Optimization* (ACO), ou Otimização colônia de Formigas, é uma metaheurística inspirada na natureza, baseada no comportamento real das formigas. A idéia por trás desta técnica é que as formigas na sua busca por alimento espalham um rastro químico, conhecido como feromônio, por todo o seu caminho, influenciando na escolha do caminho de outras formigas (Dorigo e Blum, 2005) (Burke et al., 2005) (Dorigo e Gambardella, 1997).

(Dorigo, 1992) mostrou que as formigas conseguem encontrar o menor caminho entre o seu ninho e a fonte de comida sem uma orientação visual do seu espaço de busca, utilizando somente uma orientação química, ou seja, os rastros de feromônios que elas deixam por onde passam. Enquanto caminham, as formigas depositam feromônios e são guiadas por feromônios depositados anteriormente por onde passam. A eficiência desta técnica tem explicação simples: quanto menor for um certo caminho escolhido por um dada formiga, mais rápido ela irá retornar ao ninho e mais forte será o seu rastro de feromônio quando outra formiga for sair do ninho, induzindo esta na escolha deste caminho e fortalecendo ainda mais o rastro de feromônio depositado. Com o passar do tempo, a grande maioria das formigas estará utilizando este menor caminho (Dorigo et al., 1996).

O pseudo-algoritmo ACO, mostrado na figura 2.11, demonstra a idéia deste método. Neste algoritmo, dado um problema de otimização combinatória, uma formiga

artificial é posicionada em partes distintas do espaço de busca. Cada formiga segue um caminho aleatório seguindo uma determinada fórmula probabilística em função do feromônio presente (Rizzoli et al., 2004). Para atualizar a trilha de feromônio é calculada inicialmente a quantidade a ser depositada proporcional a qualidade das soluções (Dorigo e Di Caro, 1999).

```
Algoritmo ACO()
1  inicio
2      enquanto Critério de parada não satisfeito;
3          Constrói conjunto de solucoesS = {s_1, s_2, ..., s_n};
4          S' ← Para toda solução em S faça Busca_Local(s_i);
5          Atualiza rastro de feromônio;
6fim enquanto;
7fim.
```

Figura 2.11: Pseudo-algoritmo ACO.

2.5 Considerações Finais

Este Capítulo apresenta os principais métodos heurísticos e metaheurísticos tratados na literatura. Os conceitos apresentados permitem estabelecer as principais semelhanças e diferenças entre estes métodos, através de suas formas de tratamento de problemas, bem como suas classificações, de acordo com as características de cada método. Para os experimentos práticos deste trabalho, foram implementados os métodos de trajetória *Iterated Local Search* e *Variable Neighborhood Search* e o método baseado em população Algoritmo Genético.

Capítulo 3

Heurísticas Cooperativas

3.1 Introdução

Durante os últimos anos, o número de pesquisadores de otimização combinatória com interesse em técnicas cooperativas para metaheurísticas aumentou consideravelmente. Uma das razões desse aumento de interesse é a publicação de trabalhos, como (Gendron et al., 2003), (Bouthillier e Crainic, 2005) e (Cadenas et al., 2007), apresentando resultados de alta qualidade para diversos problemas de otimização combinatória através do uso de algoritmos cooperativos. Estes algoritmos são obtidos pela atuação em conjunto de algoritmos como Busca Local Básica, *Iterated Local Search*, Busca Tabu e Algoritmos Evolutivos, na busca de soluções de alta qualidade para os problemas tratados.

Neste Capítulo, é apresentada uma revisão de literatura sobre heurísticas e metaheurísticas cooperativas, adequáveis aos interesses deste trabalho.

3.2 Metaheurística Paralela Cooperativa

Bouthillier e Crainic (2005) introduzem um procedimento de busca metaheurística cooperativa paralela, baseada em uma técnica de Repositório de Soluções, para o Problema de Roteamento de Veículos com Janela de Tempo (PRVJT). Este procedimento utiliza uma abordagem *multi-thread*, em que cada *thread* independente segue uma estratégia de busca diferente. A figura 3.1 apresenta um diagrama de colaboração que ilustra este esquema.

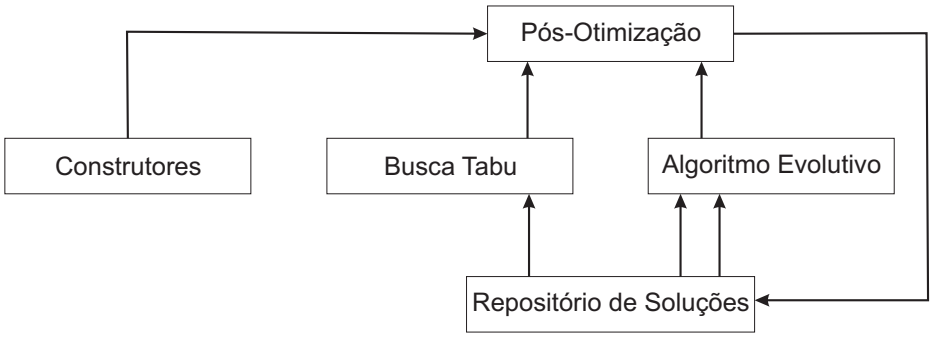

Figura 3.1: Diagrama de Colaboração (Bouthillier e Crainic, 2005).

A cooperação é alcançada, nos termos postos em (Bouthillier e Crainic, 2005), através de trocas assíncronas de informação. Neste processo, a informação é compartilhada por um Repositório de Soluções e, sempre que uma *thread* decide enviar uma informação, ela a envia para o Repositório. Similarmente, quando uma *thread* necessita de informação, ela a procura no Repositório de Soluções.

O Repositório de Soluções é, desse modo, para esta proposta, o mecanismo principal da cooperação. Ele mantém soluções boas e factíveis, e é atualizado dinamicamente pelos processos de busca independentes.

O Repositório de Soluções proposto em (Bouthillier e Crainic, 2005) contém dois tipos de soluções: as soluções *Adultas* e as soluções *Em Treinamento*. Todas as soluções recebidas dos processos independentes são armazenadas na área de soluções *Em Treinamento*. As soluções resultantes do processo de pós-otimização são armazenadas na área de soluções *Adultas*. Toda vez que um processo independente requerer uma solução inicial, esta será fornecida pelo Repositório de Soluções, e somente serão enviadas soluções *Adultas*.

Bouthillier e Crainic (2005) envolveram, no processo de cooperação, duas variações da metaheurística Busca Tabu, que são executadas seqüencialmente, nomeadas como *Unified Tabu* e *Tabu Route*; dois Algoritmos Evolutivos (Algoritmos Genéticos), um realizando o operador *Crossover OX* e outro o operador *Crossover ER*; além de métodos de pós-otimização, na forma de heurísticas de busca local, que foram usados para reduzir o número de veículos e a distância total percorrida. Foram usados, também, quatro algoritmos de construção, para gerar soluções iniciais para a população de soluções.

No método *Tabu Route*, segundo Bouthillier e Crainic (2005), movimentos de vizinhança são realizados por um algoritmo de inserção generalizada, que consiste em uma realocação inter-rota. As inserções são realizadas somente quando não violam a janela de tempo da solução destino e a comunicação é realizada através do Repositório de Soluções. Neste processo, o método envia soluções promissoras para o Repositório de Soluções e recebe soluções deste Repositório antes da ocorrência do processo de diversificação. O método aceita somente soluções melhores que a sua solução atual e, caso a solução recebida não seja melhor, o método utilizará sua própria solução, ou seja, realizará uma iteração simples, equivalente a uma iteração sem cooperação.

O método *Unified Tabu*, apresentado por Bouthillier e Crainic (2005), se baseia nas mesmas técnicas do método *Tabu Route*, porém, a busca é realizada somente em uma solução inicial. Além disso, não é realizada a fase de intensificação do método. Bouthillier e Crainic (2005) adicionaram este método no mecanismo de cooperação, realizando comunicações com o Repositório de Soluções a cada W iterações do método, ou seja, a cada W iterações, o método *Unified Tabu* é reiniciado com uma solução do Repositório de Soluções (se no mesmo existir uma solução melhor que a gerada pelo método *Unified Tabu*) e envia para o Repositório de Soluções suas melhores soluções.

Outra técnica utilizada por Bouthillier e Crainic (2005) em seu processo de cooperação é um Algoritmo Evolutivo na forma de um Algoritmo Genético. Foram desenvolvidas, segundo os autores, duas especializações deste algoritmo, uma implementando o operador *Crossover OX* (*Order Crossover*) e outra implementando o operador *Crossover ER* (*Edge Recombination*). Neste algoritmo genético, todos os

filhos gerados são enviados para o Repositório de Soluções. O operador *Crossover OX* é baseado em uma representação de caminho em cada rota e tenta preservar a ordem relativa dos clientes do segundo pai. Este operador copia uma sub-cadeia do primeiro pai para o filho e as demais posições são preenchidas por clientes do segundo pai que ainda não fazem parte da solução. O operador *Crossover ER*, por outro lado, preserva o número máximo de arcos dos pais e introduz um número mínimo de arcos novos. Uma tabela de adjacências é construída para representar, para cada nó, os nós adjacentes em cada pai. A geração do filho é feita selecionando-se caminhos com um número mínimo de arcos entre dois nós adjacentes que não estão na solução.

Na proposta apresentada por Bouthillier e Crainic (2005), o principio de pós-otimização é utilizado principalmente para explorar vizinhanças amplas e complexas na metaheurística Busca Tabu e corresponde, em geral, a aplicar uma heurística de busca local na solução. Dois tipos de heurísticas de busca local são utilizadas, por Bouthillier e Crainic (2005), para a solução do PRVJT. O primeiro tipo é chamado de processo de troca multi-nó e envolve somente uma rota por vez. Todas as trocas de nós são executadas dentro da mesma rota e o último nó da cadeia recebe a posição do primeiro nó. O segundo tipo é chamado de processo de inserção multi-nó e envolve rotas executando trocas de nó entre duas rotas ao mesmo tempo. Um nó é retirado da primeira rota e inserido na segunda, da qual outro nó é conseqüentemente retirado para ser inserido em outra rota, e assim por diante, até que um determinado número de trocas seja atingido.

O procedimento de busca proposto por Bouthillier e Crainic (2005) é baseado em um processo multi-nó e tenta eliminar uma rota, chamada de rota origem, por remoções seqüenciais de nós que são inseridos em outras rotas. Um nó removido não pode, porém, substituir um nó existente. Preferencialmente, ele é simplesmente inserido em uma rota destino, entre dois nós existentes. Se a inserção deste nó provocar que a rota receptora torne-se infactível, outros nós desta rota são removidos até que a sua factibilidade seja recuperada. Este processo é repetido para as rotas destino subseqüentes, até que um número máximo de trocas seja atingido, enquanto que, em um processo de tentativa e erro, escolhe-se rotas destino alternativas se os critérios de factibilidade da rota não forem satisfeitos.

A rota de origem é escolhida de acordo com o caminho mais curto em distância ou o caminho com o menor número de clientes ou, ainda, o caminho com a menor soma de distância para toda distância de cada cliente para seu vizinho mais próximo. Os nós no caminho de origem escolhidos são removidos um por vez, para maximizar a economia total em distancia e a folga na capacidade avaliada na rota destino.

Várias heurísticas clássicas também foram incluídas para gerar soluções iniciais e ajudar na diversificação. Quatro heurísticas construtivas simples foram adaptadas para a solução do PRVJT: (i) menor sucessor; (ii) mais próximo *double-ended*, o qual começa de um nó semente para cada caminho e seqüencialmente adiciona o melhor nó no caminho atual até que este esteja completo; (iii) fragmentos múltiplos, que adiciona seqüencialmente os arcos mais curtos para formar caminhos factíveis; e (iv) arcos híbridos mais curtos, similar à heurística anterior, porém adiciona arcos probabilisticamente de acordo com o seu tamanho (arcos de comprimento mais curtos têm maior probabilidade de serem escolhidos).

3.2.1 Descrição do Repositório de Soluções

O Repositório de Soluções é o item principal do mecanismo de cooperação proposto em Bouthillier e Crainic (2005). Ele mantém boas soluções factíveis e é atualizado dinamicamente por processos de busca independentes. A população de soluções forma, então, uma população elite, da qual os procedimentos independentes requererão soluções em várias fases de execução. Representação de caminho é utilizado para manter soluções no Repositório de Soluções. Estas soluções são ordenadas primeiramente pelo número de veículos (em ordem crescente) e, então, de acordo com soma do tempo total de viagem, distância total, tempo de espera e tempo ocioso para todos clientes.

Segundo (Bouthillier e Crainic, 2005), o Repositório de Soluções é dividido em duas sub-populações: população em treinamento e população adulta. Todas as soluções recebidas dos processos independentes são colocadas nas soluções em treinamento. O processo de pós-otimização é então aplicado e o resultado é movido para a população adulta, tendo as soluções duplicadas eliminadas. Todas as requisições de solução feitas por um processo independente serão atendidas por uma sub-população adulta. Soluções são selecionadas aleatoriamente, de acordo com um mecanismo guloso de seleção, utilizando a mesma função de aptidão usada para ordenar o Repositório de Soluções.

No diagrama de colaboração relativo à arquitetura proposta em (Bouthillier e Crainic, 2005), mostrado na Figura 3.1, cada método independente é encapsulado em um processo MPI (*Message Passing Interface*). Um processo MPI contém um algoritmo particular, um mecanismo de comunicação para troca de soluções e um mecanismo de controle para configuração de parâmetros, inicialização, terminação, e aprovação de soluções. O método de cooperação trabalha junto com o Repositório de Soluções, que é o ponto central da comunicação. Ele fornece soluções iniciais diversificadas para o procedimento de Busca Tabu e pais para os algoritmos evolucionários. O tamanho da população no Repositório de Soluções é fixado em um valor relativo à dimensão do problema (definido em (Bouthillier e Crainic, 2005) como 10 vezes a dimensão do problema) e os piores resultados são eliminados, quando necessário. Nenhuma comunicação direta acontece entre os processos, de forma que os torne independentes entre si e com uma comunicação assíncrona. Este esquema torna o processo de cooperação simples e, eventualmente, permite modificação fácil do sistema paralelo, adicionando novos métodos ou removendo métodos ineficientes.

O estudo experimental de (Bouthillier e Crainic, 2005) teve dois objetivos: (*i*) comparar o Repositório de Soluções proposto com os melhores resultados de métodos propostos na literatura para a solução do PRVJT e, assim, validar esta proposta confirmando que o método proposto oferece desempenho comparável em termos de qualidade de solução e esforço computacional; (*ii*) avaliar o comportamento da cooperação e a contribuição dos métodos individuais envolvidos na cooperação, em particular o impacto da qualidade dos métodos individuais no desempenho da busca global.

Um método de busca diferente é executado para cada um dos quatro processadores (dois para Busca Tabu e dois para Algoritmos Evolucionários). O Armazém de Soluções, a pós-otimização, e os métodos de construção são executados em outro processador, totalizando cinco processadores neste estudo de (Bouthillier e Crainic,

2005).

(Bouthillier e Crainic, 2005) realizaram experimentos em duas versões da busca cooperativa, para estudar a influência dos métodos individuais no desempenho global e a interação entre os métodos, quando executados em cooperação. O primeiro experimento (identificado como LC02) consiste de dois métodos *Tabu Route* (TS1 e TS2) e dois Algoritmos Evolucionários (OX e ER). No segundo experimento (LC03), o método *Unified Tabu* (UT1) substitui o segundo método *Tabu Route*. Os Algoritmos Evolucionários não são alterados. Os testes foram realizados no conjuntos de instâncias testes de Solomon (Solomom, 1987).

Neste modelo proposto por (Bouthillier e Crainic, 2005), as soluções no Repositório de Soluções adultas são classificadas, primeiramente, pelo número de vértices e, em seguida, por uma soma dos pesos de atributos, dada por:

$$\begin{aligned} C(p) &= W_1 * TempoTotal + W_2 * DistânciaTotal + \\ &\quad + W_3 * TempoDeEspera + W_4 * FolgaTotal \end{aligned} \quad (3.1)$$

sendo:

- $TempoTotal$: o tempo total necessário para atender todos os clientes;
- $DistânciaTotal$: a distância total associada;
- $TempoDeEspera$: tempo total de espera em cada cliente;
- $FolgaTotal$: soma das folgas em cada janela de tempo;
- W_1, W_2, W_3, W_4: parâmetros de controle.

Em todos os experimentos, (Bouthillier e Crainic, 2005) definiram os parâmetros W_1, W_2, W_3 e W_4 com o valor 1. Segundo os autores, esta medida, combinada com o número de veículos, fornece uma idéia global da qualidade da solução (Tempo Total e Distância Total) e flexibilidade (Tempo de Espera e Tempo de Folga da Janela de Tempo). As duas últimas medidas indicam a quantidade de folga existente na solução e possíveis soluções vizinhas que podem ser exploradas facilmente.

3.3 *COSEARCH*: Metaheurística Cooperativa Paralela com Memória Adaptável

Talbi e Bachelet (2006) propuseram um modelo de metaheurística cooperativa paralela denominado *COSEARCH*. O *COSEARCH* é uma proposta de métodos heurísticos cooperativos, utilizando uma memória adaptável para manter um histórico de busca. O modelo geral do ambiente *COSEARCH* foi proposto para a resolução do Problema Quadrático de Atribuição (*QAP - Quadratic Assignment Problem*). Para definições e informações a respeito dessa classe de problemas, podem ser consultadas as referências (Çela, 1998) e (Pardalos et al., 1994). A proposta apresentada por Talbi e Bachelet (2006) para a resolução utiliza uma forma original de memória adaptável, focada em regiões de busca promissoras, evitando regiões atraentes, porém, enganosas.

Para a solução do problema objeto de interesse, (Talbi e Bachelet, 2006) utilizaram um modelo híbrido com três agentes heurísticos complementares: um agente na forma de uma implementação da metaheurística Busca Tabu é utilizada como principal algoritmo de busca; um outro agente posto como um Algoritmo Genético é utilizado para diversificar a solução; e terceiro agente, um operador de "troca", é aplicado para intensificar a busca.

Neste projeto, Talbi e Bachelet (2006) propõem uma estratégia de busca heurística em que dois critérios diferentes têm que ser equilibrados: a diversificação do espaço de busca e a exploração das soluções que já foram encontradas. Enquanto cada algoritmo heurístico, com seus métodos particulares, diversifica e intensifica o espaço de busca, é difícil projetar um método bem balanceado. Então, alguns algoritmos são melhores para diversificar, e outros para intensificar os espaços de busca. Em vez de tentar melhorar a diversificação ou a intensificação de alguns algoritmos, o *COSEARCH* utiliza a combinação de heurísticas híbridas, fazendo com que heurísticas diferentes trabalhem em conjunto, cada uma em sua área de melhor atuação.

A figura 3.2 mostra o esquema de metaheurísticas cooperativas do ambiente *COSEARCH*. Neste esquema, as três heurísticas citadas anteriormente (Busca Tabu, Algoritmo Genético e operador de "troca") são definidas como "Agente de Busca", "Agente Diversificador" e "Agente Intensificador", respectivamente.

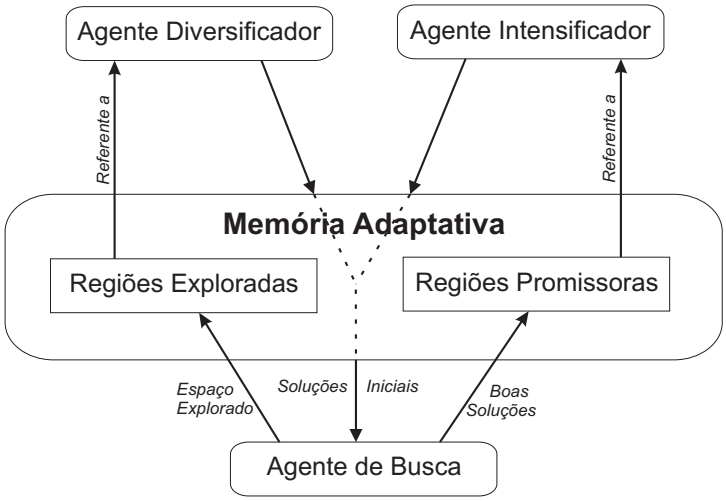

Figura 3.2: Esquema da Metaheurística Cooperativa *COSEARCH* (Talbi e Bachelet, 2006).

Os três agentes implementados por (Talbi e Bachelet, 2006) trocam informações através de um coordenador passivo, que é a própria "memória adaptativa". Esta estrutura de memória possui um histórico de busca, armazenando informações sobre áreas já visitadas do espaço de busca e sobre a qualidade de boas soluções já encontradas. Quando atuando na forma de diversificação, o agente Busca Local recebe soluções iniciais de regiões ainda não exploradas do espaço de busca; quando atuando na forma de intensificação, o agente Busca Local recebe soluções iniciais de regiões promissoras no espaço de busca. O agente diversificador refere-se à área da memória adaptável (informações sobre áreas exploradas) que pode levar a soluções de regiões ainda não exploradas; o agente intensificador refere-se à área da memória adaptável

(informações sobre regiões promissoras) que contem soluções iniciais promissoras.

No agente diversificador do *COSEARCH*, o Algoritmo Genético gera indivíduos em regiões não exploradas da memória adaptativa, sendo eficiente na tarefa de diversificação, pois gera soluções dispersas (Talbi e Bachelet, 2006).

O agente diversificador inicializa com soluções geradas pela metaheurística Busca Tabu. O agente intensificador utilizado no *COSEARCH* é o operador de "troca". Para gerar novas soluções, este operador utiliza o conjunto de soluções elite, escolhendo aleatoriamente uma solução elite e a perturbando, com o objetivo de realizar um pequeno salto no espaço de busca. Se a perturbação realizada pelo operador for muito grande, pode ser gerada uma solução fora do espaço de busca promissor das soluções elite. Assim, a perturbação deve ser pequena o suficiente para evitar que este operador se torne um operador de diversificação.

3.3.1 Descrição da Memória Adaptativa do *COSEARCH*

A memória adaptativa é o coordenador da metaheurística *COSEARCH*. Ela mantém, em tempo real, um histórico de ações da busca tabu. Segundo (Talbi e Bachelet, 2006), a memória adaptativa é composta de três partes principais: a memória de freqüência global, o conjunto de soluções iniciais da busca tabu e o conjunto de soluções elite.

A memória de freqüência global e o conjunto de soluções iniciais são utilizados pelo Algoritmo Genético na tarefa de diversificação. A memória de freqüência global é a soma das memórias de freqüência local geradas pela busca tabu. Para a tarefa de intensificação, são utilizadas soluções dos conjunto de soluções elite. Este conjunto armazena as melhores soluções geradas nas iterações da busca tabu. O número de soluções elite é um parâmetro do método *COSEARCH*.

Para evitar que o conjunto elite fique preso em uma única região interessante, (Talbi e Bachelet, 2006) propuseram uma maneira de atualizar o conjunto elite de soluções, utilizando uma estratégia bi-critério (qualidade e diversidade) e um processo de seleção guiado por soluções *Pareto-dominadas* (Talbi e Bachelet, 2006). Quando não for possível determinar o conjunto *Pareto* de soluções, uma estratégia *FIFO* (*First in, first out*) é utilizada para organizar e selecionar as soluções na Memória Adaptativa.

3.4 Sistema Cooperativo de Metaheurísticas com Regras *Fuzzy*

Em Cadenas et al. (2007) é proposto um sistema cooperativo de metaheurísticas, como um sistema multiagente, em que cada metaheurística é definida como sendo um agente. Três metaheurísticas são combinadas como um sistema híbrido de cooperação, baseado em mineração de dados, para resolução do problema da mochila, tendo um agente coordenador para controlar e modificar o comportamento dos agentes.

Para a cooperação entre as metaheurísticas utilizadas, Cadenas et al. (2007) utilizaram um modelo de quadro-negro adaptado (veja seção 3.5). Neste modelo,

cada metaheurística controla uma parte do quadro-negro no qual pode escrever informações sobre espaços de busca promissores e, periodicamente, escrever soluções encontradas. O agente coordenador verifica o quadro-negro para monitorar o comportamento das metaheurísticas e decidir como e quando cada uma poderá modificar suas ações. Este processo pode ser visualizado na figura 3.3.

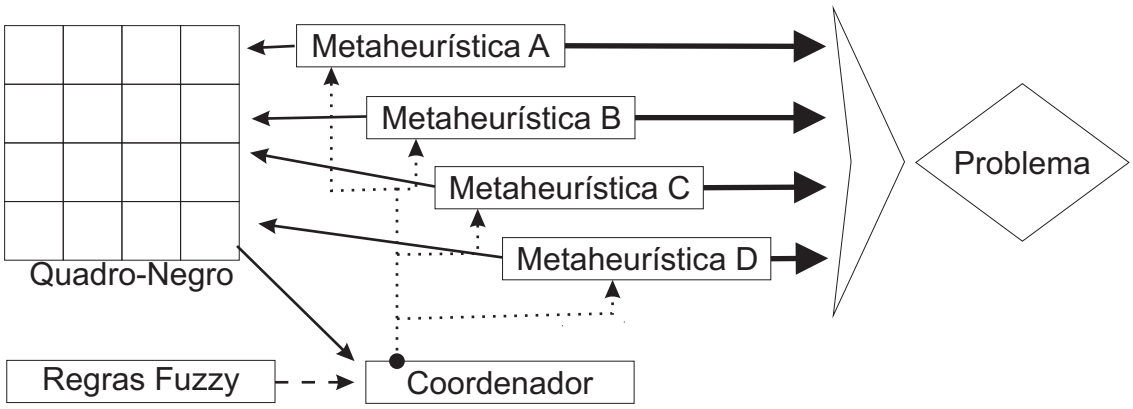

Figura 3.3: Sistema Metaheurístico Multiagente (Cadenas et al., 2007).

Na proposta apresentada em Cadenas et al. (2007), ao agente coordenador é adicionado um conjunto de regras *fuzzy*, de modo a auxiliá-lo em seu mecanismo de tomada de decisões. O primeiro passo desta metodologia é utilizar um processo de extração de conhecimento, do qual é obtido o conjunto de regras *fuzzy*, e então, estas regras são organizadas em conjunto com o coordenador. O processo de extração de conhecimento é supervisionado e constituído de três fases distintas: (i) a fase de preparação dos dados; (ii) a fase de mineração dos dados (*Data Mining*); e, por fim, (iii) a fase de Avaliação do Modelo. Estas três fases têm, como principais objetivos, obter um banco de dados de informações úteis ao sistema sobre o conjunto de regras, aplicar técnicas de mineração de dados para refinar o modelo de coordenador do sistema e testar a eficiência do conjunto de regras fuzzy deste modelo de coordenador, respectivamente.

Na fase de preparação dos dados, todas as informações iniciais, como, por exemplo, informações sobre o problema a ser tratado e heurísticas a serem utilizadas, são coletadas. Após este processo, segundo Cadenas et al. (2007), é realizada uma calibragem de parâmetros, resolvendo um conjunto de instâncias teste, com soluções já conhecidas, e utilizando uma árvore de decisão *fuzzy*, para transformar este conjunto de soluções em diversas regras *fuzzy*, que, por sua vez, constituem o primeiro modelo de coordenador.

Estas regras que definem o comportamento do agente coordenador devem ter características que o guiem para realizar uma cooperação efetiva entre as metaheurísticas envolvidas no problema:

- quando e como uma metaheurística pode usar soluções de outras de forma positiva;
- quais os parâmetros iniciais para uma metaheurística, baseado nas instâncias a serem resolvidas;

- quando e como devem ser alterados os parâmetros de uma metaheurística que estejam gerando resultados piores que as outras;
- que regra deve ser escolhida se mais de uma for ativada;

A ultima fase deste processo, a fase de avaliação do sistema, é utilizada para verificar a eficiência deste modelo de coordenação e seu custo computacional.

Cadenas et al. (2007) mostra duas implementações de estruturas multiagentes. A primeira foi realizada com as metaheurísticas Algoritmo Genético e Busca Tabu. A segunda, além das mesmas metaheurísticas consideradas na primeira, acrescenta a metaheurística *Simulated Annealing*, com a finalidade de verificar a capacidade de obter maior qualidade com a adição de uma nova metaheurística. Conclui-se que o aumento do número de metaheurísticas aumenta o tempo de busca, mas não altera o desempenho do sistema, pois cada metaheurística atua de forma independente da outra.

No entanto, deve ser ressaltado que a estrutura multiagente utilizada em Cadenas et al. (2007) não utiliza o conceito de autonomia para cada agente em particular.

3.5 Sistema de Quadro-Negro

(Corkill et al., 1986) descreveu as principais características de sistemas de quadro-negro (*Blackboard Systems*). Nesta proposta, (Corkill et al., 1986) discute o que é um sistema de quadro-negro e as principais justificativas de sua utilização. Esta proposta pode ser visualizada como a ação de um grupo de especialistas (agentes), sentados próximos a um grande quadro-negro, trabalhando para resolver cooperativamente um certo problema, usando o quadro-negro como o local de trabalho para desenvolver a solução do problema. A resolução do problema se inicia quando o problema e os dados iniciais são escritos no quadro-negro. Os agentes verificam as questões postas no quadro-negro, enquanto procuram apresentar suas contribuições para a solução em desenvolvimento.

Quando um agente encontra uma informação que possa contribuir, ele registra esta informação no quadro-negro, e os outros agentes também apresentam suas contribuições. Este processo de somar contribuições para o que está escrito no quadro-negro continua, até que o problema seja resolvido (Corkill, 1991).

No contexto de metaheurísticas cooperativas, a descrição posta para o sistema de quadro-negro, pode-se considerar cada especialista, ou agente, como sendo uma metaheurística que busca resolver o problema. Como tem-se várias metaheurísticas e cada uma tem características particulares, ou seja, cada uma tem uma forma diferente de tratar o problema, elas devem tratar as informações do problema e estabelecer uma forma de contato comum com o sistema de quadro-negro. A Figura 3.4 exibe os componentes básicos deste modelo de cooperação.

De acordo com a Figura 3.4, o modelo de quadro-negro é composto de três componentes básicos: as Fontes de Conhecimento, o Quadro-Negro e o componente de Controle.

As Fontes de Conhecimento são módulos independentes que contém conhecimento (ou informações) necessário para resolver o problema. Estas Fontes de Conhecimento podem ser extensamente diversas em representação e técnicas de trata-

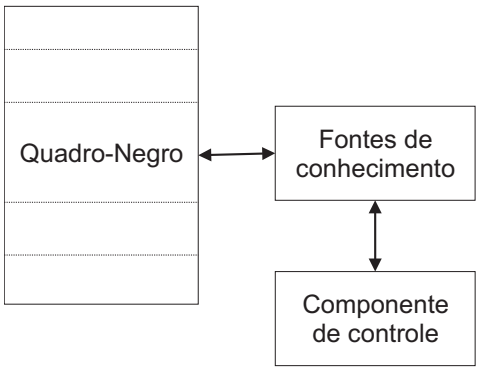

Figura 3.4: Componentes Básicos do Modelo Quadro-Negro (Corkill, 1991).

mento do problema. Cada Fonte de Conhecimento é implementada individualmente e independente de outras Fontes de Conhecimento. Uma Fonte de Conhecimento não precisa conhecer as demais Fontes de Conhecimentos, mas deve ter conhecimento suficiente do problema e de suas formas de representação. Estas Fontes de Conhecimento podem ser vistas como sendo os agentes descritos anteriormente.

O Quadro-Negro é um banco de dados global, que contém dados de contribuições, soluções parciais e outros dados que estão em vários estados de resolução. O Quadro-Negro é uma estrutura que está disponível a todas as Fontes de Conhecimento e serve como:

- uma memória compartilhada de dados, soluções parciais, soluções finais e controle sobre todas estas informações;
- um meio de comunicação e armazenamento;
- um mecanismo de ativação das Fontes de Conhecimento.

O Quadro-Negro deve interagir diretamente com as Fontes de Conhecimento. Estas não podem ter outras formas de comunicação ou possuir mecanismos de controle separados, ou seja, distintos das Fontes de Conhecimento.

O Componente de Controle toma decisões em tempo real sobre o comportamento do sistema. Ele consiste em um mecanismo de controle que guia as Fontes de Conhecimento na resolução de problemas, gerenciando sua comunicação com o Quadro-Negro. Baseado no estado atual do Quadro-Negro e no conjunto de Fontes de Conhecimento ativas, o Componente de Controle define estratégias de ação a serem executadas. O Componente de Controle utiliza uma metodologia de filas para gerenciar as Fontes de Conhecimento, de modo que o Componente de Controle sempre tratará da Fonte de Conhecimento que for a próxima da fila de Fontes, em um formato *first in-first out*. Este processo é repetido até que algum critério de parada seja satisfeito, que pode ser um número máximo de iterações ou uma solução aceitável encontrada (Corkill, 1991).

Corkill (1991) propõem que os Sistemas de Quadro-Negro devem ser flexíveis e permitirem que diversas Fontes de Conhecimento sejam adicionadas e/ou removidas da arquitetura. Mas, para que isso seja possível, deve existir um entendimento comum da representação da informação tratada por todas as partes. Portanto, os

Quadros-Negros devem implementar uma padronização dos dados referentes ao problema tratado, para que, independente da forma com que cada Fontes de Conhecimento trate o problema, os dados escritos e lidos do Quadro-Negro sejam padronizados. Esta padronização agiliza o processo de leitura e escrita das informações e permite a adição e remoção de Fontes de Conhecimento. Por sua vez, as Fontes de Conhecimento em Sistemas de Quadro-Negro devem ser capazes de interpretar as informações do Quadro-Negro e de escrever neste mesmo padrão.

Sendo o problema a ser resolvido extenso e complexo e o número de contribuições feitas no Quadro-Negro crescentes, a busca por informações pertinentes pode se tornar lenta e complexa. Um agente não deve ter que examinar todo o Quadro-Negro na tentativa de encontrar informações relevantes à sua procura. Uma solução, neste caso, é subdividir o Quadro-Negro em regiões, cada uma correspondendo a um tipo particular de informações. Outra abordagem válida, semelhante à anterior, é organizar as informações dentro do Quadro-Negro , utilizando formas de ordenação ascendente, descendente, ou por relevância (Corkill, 1991).

Segundo Corkill (1991), outro aspecto muito importante a ser observado no desenvolvimento de Sistemas de Quadro-Negro é a utilização de uma memória de grupo, que armazena, por um determinado tempo, informações que, *a priori*, parecem não ter relevância significativa na resolução do problema, mas que mais tarde possam vir a ser muito úteis.

Conforme já dito, os agentes especialistas não se comunicam diretamente. Cada um verifica o Quadro-Negro à procura de uma oportunidade para contribuir para a solução do problema em análise (Corkill, 1991). Tais oportunidades surgem quando um evento acontece (uma mudança é feita no Quadro-Negro), permitindo ao agente agir. O grande problema com esta abordagem é que vários agentes podem ser acionados e desejarem acessar o Quadro-Negro ao mesmo tempo, ocasionando problemas como a não garantia da integridade das informações. Existem várias abordagens sobre as soluções para este tipo de problema, sendo uma delas criar uma fila da prioridades, em que os agentes têm, cada um, a sua vez de acessar o Quadro-Negro. Outra forma de solução de conflitos é a criação de um agente coordenador, para gerenciar a comunicação entre os demais agentes e o Quadro-Negro.

3.6 Hiperheurísticas

O termo "Hiperheurística" é devido a Edmond Burke e colaboradores, em seus trabalhos particularmente associados ao Problema de Quadro de Horários (Burke et al., 2003). (Burke et al., 2005) apresentam uma definição de hiperheurísticas, que se adequa aos interesses desta dissertação:

> *"Hiperheurísticas podem ser descritas como estratégias definidas para controlar a aplicação de um conjunto de heurísticas para resolver um problema ."*

Ou seja, uma hiperheurística pode ser definida como, basicamente, uma heurística para a escolha de heurísticas, visando a melhor estratégia para a resolução de problemas de otimização (Burke et al., 2003).

Hiperheurísticas podem ser definidas de acordo com a particularidade de cada tipo de problema tratado, e podem implementar diversos tipos de regras para a

escolha de que heurística, e como esta vai atuar no espaço de busca do problema tratado. A questão principal do funcionamento das hiperheurísticas é baseada no desempenho das heurísticas envolvidas no processo de busca, ou seja, a cada instante (ou iteração, este tempo pode ser definido por uma regra) durante a busca, a seleção (ou escolha) da próxima heurística a ser utilizada é baseada no histórico do desempenho de cada heurística sobre o problema. Segundo (Burke et al., 2003), uma importante característica de hiperheurísticas é que o conjunto de heurísticas aplicadas durante a busca pode ser desde uma simples busca local até algoritmos mais elaborados, como metaheurísticas.

(Burke et al., 2003) propuseram uma técnica hiperheurística baseada em Busca Tabu, para a solução de um problema de distribuição espacial bi-objetivo. Para este problema, foram projetadas algumas heurísticas de busca local básica, baseadas em movimentos simples de troca e realocação. Nesta proposta, (Burke et al., 2003) definem uma estratégia hiperheurística simples, baseada em Busca Tabu, para escolher a heurística de busca local que irá explorar uma determinada solução durante o processo de busca. Esta abordagem gera um conjunto de soluções iniciais, e então, iterativamente, melhora estas soluções selecionando as heurísticas mais apropriadas para a busca local na vizinhança corrente.

Nesta hiperheurística Busca Tabu, todas as heurísticas de busca local competem entre si para serem selecionadas. Esta competição é feita por um princípio de aprendizagem e no inicio da busca cada heurística tem associado um valor zero a cada objetivo do problema tratado. Quando uma heurística de busca local é aplicada a uma vizinhança do problema, o seu valor associado a um determinado objetivo é incrementado, se neste obtiver ganho, e decrementado caso contrário. Estes valores são mantidos e gerenciados pela hiperheurística Busca Tabu, que mantém uma lista (tabu) de heurísticas de busca local que não estão se comportando bem ao problema, evitando que estas sejam escolhidas por um determinado tempo.

Uma hiperheurística baseada em Algoritmo de Colônia de Formigas (veja seção 2.4.2) é proposta em (Burke et al., 2005). Neste artigo, os autores propõem uma estratégia de heurísticas em forma de grafo para a resolução de um problema de alocação. Nesta proposta, (Burke et al., 2005) definem cada vértice do grafo como sendo uma heurística que resolve o problema, e estes vértices, por sua vez, são conectados por arestas que representam se uma heurística pode ser aplicada imediatamente após a outra, ou seja, dados dois vértice v_i e v_j e uma aresta a_{ij}, os vértices v_i e v_j representam, cada um, uma heurística que resolve o problema, e a aresta a_{ij} a ligação entre estes dois vértices. Neste sentido, a aresta a_{ij} significa que a heurística representada pelo vértice v_j pode ser aplicada ao problema, logo após a aplicação da heurística representada pelo vértice v_i.

Com estas definições, é criado um número constante de formigas, cada uma representando um agente hiperheurístico e com capacidade de acesso às heurísticas e suas funções de avaliação. Estas formigas caminham pelos vértices do grafo, em busca das melhores heurísticas para resolver o problema a cada passo. Uma solução hiperheurística para o problema é um ciclo (veja (Ziviani, 2007)), ou um conjunto de ciclos, sendo que cada formiga traça um ciclo de heurísticas, baseadas na quantidade de feromônio depositados em cada heurística (ou vértice do grafo) e na função de avaliação destas heurísticas.

A quantidade de feromônio depositada em cada vértice (heurística) do grafo é definida de acordo com a avaliação da solução, baseado no melhoramento iterativo da qualidade da solução durante o ciclo percorrido por cada formiga, que por sua vez, atualiza este rastro de feromônio após a inserção de uma certa quantidade de vértices adicionados à seu ciclo. Podem existir tantos ciclos quantos forem necessários ao problema.

3.7 Considerações Finais

A cooperação entre métodos de busca pode ser uma ferramenta muito útil na resolução de problemas de grande porte, possibilitando ganhos em tempo computacional e qualidade das soluções encontradas. Diversas técnicas são propostas na literatura para possibilitar um ambiente cooperativo e integrado de métodos resolvedores de soluções. Este capítulo apresentou os principais métodos cooperativos para metaheurísticas, no âmbito do interesse desta dissertação.

São apresentados, também, conceitos sobre Hiperheurísticas, que são importantes para o entendimento de, por exemplo, estratégias de escolha de diversas metodologias de busca diferentes no sentido de afinidade de cooperação, ou seja, escolher os métodos que conseguem cooperar melhor entre si.

Capítulo 4

Sistemas Multiagentes

4.1 Introdução

Diversas técnicas de solução de problemas têm sido desenvolvidas utilizando os conceitos de Inteligência Computacional (IC). Dentre estas, pode-se destacar métodos heurísticos e metaheurísticos (Blum e Roli, 2003), algoritmos evolutivos (Bäck, 1996), redes neurais (Bleale e Jackson, 1990), sistemas *fuzzy* (Driankov et al., 1990), dentre outros.

Entretanto, com o surgimento das redes de computadores e seu crescimento, questões relacionadas aos processos de interação e de comunicação entre os sistemas computacionais foram levantadas. Adicionalmente, a complexidade das tarefas que são automatizadas e delegadas aos computadores cresce continuamente. Fatores como estes insinuam a necessidade de sistemas computacionais que possam cooperar, tendo, além disso, a habilidade de negociar acordos (ou até mesmo competir) com outros sistemas que têm interesses diferentes (Wooldridge, 2002).

Esta abordagem de sistemas computacionais conduziu ao aparecimento de um novo área de pesquisas em Ciência da Computação, denominada Sistemas Multiagentes (Wooldridge, 2002).

Um Sistema Multiagente pode ser visto como uma rede fracamente acoplada de sistemas solucionadores de problemas, que interagem para solucionar problemas que estão além das capacidades ou conhecimento de cada solucionador de problemas individualmente. Estes solucionadores de problemas são tratados como agentes.

Os Sistemas Multiagente (SMA) visam garantir que os agentes têm habilidades de comunicação e, de certa forma, influenciam uns nas ações dos outros (Shuai e Feng, 2005).

Em Sistemas Multiagentes, os agentes operam de forma assíncrona e com um certo nível de autonomia, cooperando entre si para alcançar um conjunto de objetivos. Esta autonomia está relacionada à habilidade de um agente em tomar suas próprias decisões (Lesser, 1999).

Neste capítulo, são apresentados, na seção 4.2, conceitos e características de Agentes de Software. Na seção 4.3, são apresentados os conceitos básicos sobre agentes e sistemas multiagentes. A seção 4.4 introduz o conceito de Ambiente de atuação dos agentes. As duas próximas seções finalizam o capítulo e apresentam duas propostas de estruturas para implementação de ambientes multiagentes, encontradas na literatura, denominadas MAGMA, introduzida por Milano e Roli (2004), e *A-*

Teams, introduzida por Talukdar e de Souza (1990).

4.2 Agentes

Na busca de sistemas computacionais cada vez mais aprimorados e capazes de desempenhar atividades como pensar, tomar decisões, comunicar e aprender, a Inteligência Computacional busca cada vez mais se inspirar no comportamento individual do ser humano, bem como em outros sistemas inspirados na natureza. Estes sistemas computacionais são definidos como agentes.

Usar uma definição única para agentes é uma tarefa difícil, pois existem várias abordagens de diferentes autores sobre o tema. Além disso, devido às suas mais diversas aplicações, uma definição única torna-se muito complicada e variada.

Pode-se dizer que um agente é simplesmente algo que faz (a palavra agente vem do termo latim *agere*, que significa fazer). No entanto, espera-se que um agente computacional tenha outros atributos que possam distingui-lo de simples "programas", tais como operar sob controle autônomo, perceber seu ambiente, persistir por um período de tempo prolongado, adaptar-se a mudanças e ser capaz de assumir metas de outros agentes.

Além disso, os agentes podem decidir por si próprios o que é necessário fazer para satisfazer o objetivo para o qual foram projetados. Agentes inteligentes operam em ambientes imprevisíveis, abertos e em constante mudanças, onde há elevada possibilidade de que ações possam falhar (Russell e Norvig, 2004).

Wooldridge (2002) propõe a seguinte definição para agentes:

> "*Um agente é um sistema de computador que é situado em um ambiente, e é capaz de ação autônoma neste ambiente para conhecer seus objetivos de projeto.*"

Milano e Roli (2004) apresentam uma definição, no contexto de metaheurísticas, que se adapta melhor aos interesses deste trabalho:

> *"Um agente pode ser entendido como um sistema capaz de construir uma solução, mover sobre um espaço de busca, comunicar com outros agentes, ser ativo e, possivelmente, adaptativo."*

Assim, agentes são entidades de *software* autônomas que podem decidir por si próprios o que é necessário fazer para satisfazer o objetivo para o qual foram projetados. Os agentes atuam em determinados ambientes, interagem com este e com outros agentes ali existentes, atuando de forma assíncrona e com um certo nível de autonomia. Os agentes têm a capacidade de perceber e agir sobre um ambiente sem a necessidade de intervenções humanas (van der Hoek e Wooldridge, 2007).

Wooldridge e Jennings (1995) apresentam propriedades que distinguem agentes de outros sistemas computacionais. Estas propriedades são essenciais para melhor caracterização de agentes. A primeira propriedade é a autonomia: o agente inteligente deve ser capaz de tomar suas próprias decisões sobre as atividades a serem desempenhadas sem a necessidade da interferência do ambiente. Ele deve ser capaz de agir sozinho e continuar ativo mesmo quando outros agentes falharem. Outra propriedade de agente, que está associado à autonomia, é a pró-atividade, que é a

capacidade de tomar iniciativas, ou seja, os agentes possuem objetivos bem definidos e têm o seu comportamento baseado nestes objetivos.

Reatividade, outra propriedade posta como essencial, é a capacidade do agente de perceber o ambiente e responder a esta percepção de modo oportuno. Por fim, a habilidade de comunicação, que é a capacidade de comunicação dos agentes, seja com o ambiente, seja com outros agentes.

Associado a estas propriedades, Lesser (1999) afirma que os Agentes além de autônomos devem ser adaptáveis ao problema, percebendo suas características e atuando sobre estas para solução do mesmo. Ainda segundo este mesmo autor, o grau de autonomia e a capacidade de adaptação dos Agentes são normalmente associados com o nível de inteligência que eles possuem.

Agentes inteligentes devem ter, além de todas estas características, a capacidade de cooperação, de modo que possam trabalhar em conjunto para perceber uma região mais ampla do ambiente e aumentar a probabilidade de serem bem sucedidos. Os agentes trabalhando em conjunto, cooperando uns com os outros para alcançar algumas metas comuns dão origem aos Sistemas Multiagentes, que de forma simples podem ser classificados como um conjunto de agentes trabalhando em conjunto em um mesmo Ambiente.

4.3 Sistemas Multiagentes

Sistemas Multiagentes são sistemas computacionais que possuem dois ou mais agentes interagindo ou trabalhando juntos para realizar um conjunto de tarefas ou para satisfazer um conjunto de objetivos (Lesser, 1999).

Segundo (Wooldridge, 2002), um Sistema Multiagentes é caracterizado por um conjunto de agentes autônomos, cada um com uma tarefa específica e um conjunto de objetivos a serem alcançados em um ambiente. A cooperação entre estes agentes visando a solução conjunta do problema define o comportamento do sistema. Ainda segundo este mesmo autor, os sistemas multiagentes consistem de aplicações distribuídas compostas por um conjunto de processos autônomos, heterogêneos, distribuídos e inteligentes (agentes) que cooperam entre si para a solução de um problema complexo que está além de suas capacidades individuais.

A figura 4.1 ilustra o esquema de funcionamento de um Sistema Multiagentes. Nesta figura, são exibidos doze agentes operando em um ambiente. Neste caso, os agentes estão se comunicando com o seu grupo, enquanto percebem e atuam no ambiente. Cada agente possui uma "esfera de influência", que é a área do problema (ambiente) em que ele é capaz de atuar. Os agentes trocam informações sobre aspectos de suas esferas de influência. Esta técnica possibilita aos agentes, além do conhecimento de suas esferas de influência, ter conhecimento de outras áreas do espaço de busca do problema (ambiente), se limitando às informações que são trocadas entre os agentes.

Sistemas Multiagentes são amplamente utilizados na resolução de problemas de natureza distribuída e que requerem uma visão mais ampla do espaço de busca do problema a ser tratado, como por exemplo, problemas de otimização combinatória, em que a exploração exaustiva do espaço de busca do problema é computacionalmente inviável (Ziviani, 2007). Em problemas como os de otimização combinatória,

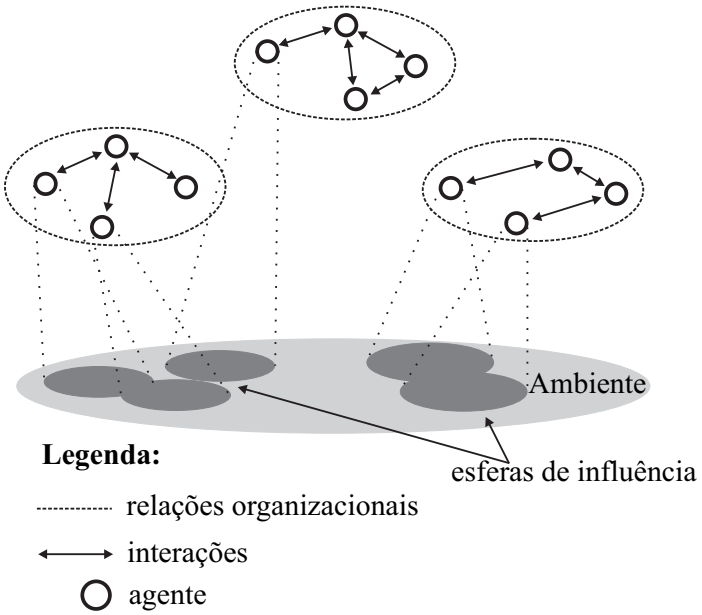

Figura 4.1: Sistema Multiagente (Wooldridge, 2002).

cada entidade que integra o processo pode ser representada como sendo um agente, que interage com o ambiente a fim de atingir seus objetivos (Blum e Roli, 2003).

Um dos problemas principais em sistemas multiagentes é como conseguir que os agentes cooperem efetivamente. A necessidade de cooperação, em tais sistemas, acontece devido à grande dimensão dos problemas tratados, aliada ao fato de que os agentes resolvem subproblemas, ou seja, conseguem abranger somente uma pequena área do problema, limitada à sua esfera de influência. As principais formas de cooperação em Sistemas Multiagentes, tratadas na literatura, foram tratadas no Capítulo 3 e nas seções 4.5 e 4.6.

4.4 Ambiente

Um Ambiente é uma estrutura onde os agentes podem obter informações completas e precisas a respeito de um determinado problema. No Ambiente devem estar presentes todas as características necessárias para a solução de um determinado problema, para que uma Agente que atue neste Ambiente possa ter conhecimento total do problema em questão (Wooldridge, 2002).

Existe uma relação de dependência entre os agentes e o ambiente em que eles atuam. O Ambiente interfere no comportamento dos agentes e os agentes realizam ações neste ambiente, de acordo com suas características. A figura 4.2 apresenta uma visão da relação entre agente e ambiente. O agente pode perceber, através de sensores o ambiente, e como resposta, realizar ações, através de atuadores que afetam este ambiente.

Segundo (Wooldridge, 2002) e (Russell e Norvig, 2004, pag. 200), a eficiência dos agentes está diretamente relacionada com a capacidade de seus sensores de entrada e com o tipo de ambiente em que está inserido. Este mesmo autor apresenta propriedades de um ambiente em Sistemas Multiagente, que seguem:

- Acessível × Inacessível: em um ambiente acessível, o agente pode obter infor-

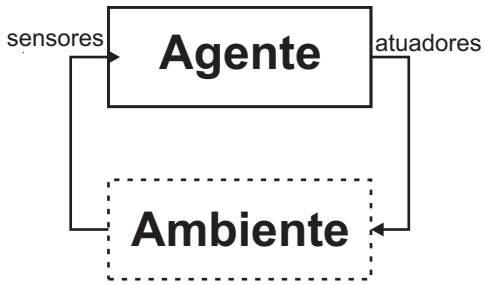

Figura 4.2: Relação Agente X Ambiente (Wooldridge, 2002).

mações completas e atualizadas sobre o estado do ambiente. A maioria dos ambientes do mundo real (incluindo, por exemplo, o mundo físico cotidiano e a internet) não é acessível neste sentido.

- Determinístico × Não-Determinístico: em um ambiente determinístico, qualquer ação tem um efeito simples garantido - não há nenhuma incerteza sobre qual será o resultado da execução de uma determinada ação. Por outro lado, o não determinismo captura o fato de que o agente tem uma "esfera de influência" limitada e que suas ações podem falhar, não obtendo o efeito desejado.

- Estático × Dinâmico: um ambiente estático é aquele que permanece inalterado, exceto pela execução de ações do agente. Em contraste, um ambiente dinâmico é aquele que tem outros processos operando sobre ele, e o qual conseqüentemente muda além do controle do agente.

- Discreto × Contínuo: um ambiente é discreto se existe um número fixo e finito de ações e percepções nele. Em um ambiente contínuo, existem incontáveis estados.

4.5 Arquitetura MAGMA

(Milano e Roli, 2004) introduzem uma arquitetura multiagentes metaheurísticos - *MultiAGent Metaheuristics Architecture* (MAGMA), em que uma metaheurística pode ser vista como o resultado da interação entre vários agentes presentes na arquitetura. Em sua arquitetura básica, a MAGMA é dividida em três níveis, de modo que cada nível contém um ou mais agentes, podendo também, no modelo conceitual de sua versão cooperativa, existir um quarto nível na arquitetura, com o objetivo de coordenar as ações e as trocas de informações de todos os outros níveis da arquitetura, ou seja, gerenciar o modelo cooperativo da arquitetura MAGMA.

Assim, na Arquitetura MAGMA, uma metaheurística não é implementada diretamente em um único agente, mas em um conjunto de agentes com propósitos específicos, classificados e separados pelos níveis da arquitetura. A Figura 4.3 ilustra o esquema Multi-Níveis proposto pela arquitetura.

O Nível 0 contém os agentes responsáveis por gerar soluções iniciais para o problema tratado na arquitetura. Estes agentes têm a função de gerar estas soluções iniciais e factíveis, podendo se basear em diversas estratégias de construção (como aleatória, gulosa ou probabilística) e enviá-las aos agentes do nível imediatamente superior, ou seja, aos agentes do Nível 1.

Figura 4.3: Esquema Multi-Níveis da Arquitetura MAGMA.

No Nível 1, os agentes têm a função de melhorar a solução fornecida pelos agentes do nível anterior, o Nível 0, a partir da exploração do espaço de busca, podendo ser constituído por diversas estratégias de busca local.

Os agentes do Nível 2 têm a tarefa de guiar a busca para soluções mais promissoras. Estes agentes possuem estratégias de busca mais elaboradas, para possibilitar escapar de ótimos locais, utilizando a combinação de diferentes conceitos inteligentes para explorar e expandir o espaço de busca.

No quarto nível da arquitetura, o Nível 3, os agentes são responsáveis pelo processo de coordenação/cooperação na arquitetura. Este nível só existe na versão cooperativa da arquitetura MAGMA e tem a função de coordenar todo o processo de busca, isto é, coordenar o comportamento de todos os agentes dos níveis inferiores, definindo quando e como a comunicação ocorrerá.

Neste sentido, um modelo prático da Arquitetura MAGMA pode ser visto como a decomposição de uma metaheurística em vários agentes de diferentes níveis. Em (Milano e Roli, 2004), são propostas cinco especializações da arquitetura MAGMA: a primeira, implementando a metaheurística *GRASP*; a segunda, implementando a metaheurística *Ant Colony Optimization*; a terceira, implementando a metaheurística *Iterated Local Search*, a quarta, uma especialização da arquitetura implementando uma metaheurística AG com busca local, ou seja, algoritmo memético; e, por fim, na quinta e ultima especialização, uma versão cooperativa da arquitetura, combinando duas das especializações simples desenvolvidas, GRASP e AM (Algoritmo Memético). Estas especializações são descritas a seguir.

A primeira especialização da arquitetura MAGMA descreve a metaheurística *GRASP*. Nesta especialização, mostrada na figura 4.4, são utilizados três níveis, sendo esta metaheurística decomposta por estes três níveis. Os agentes do Nível 0 geram soluções iniciais com uma heurística de construção gulosa aleatória; os agentes do Nível 1 utilizam de um algoritmo de busca local para melhorar a solução gerada pelos agentes do nível anterior; e, por fim, os agentes do Nível 2 selecionam e armazenam a melhor solução encontrada no processo de busca local.

Figura 4.4: Especialização da arquitetura MAGMA para GRASP.

A segunda especialização da arquitetura descreve a utilização da metaheurística ACO (*Ant Colony Optimization*). De acordo com o algoritmo *ACO* genérico, os níveis da arquitetura, para esta especialização, são definidos da seguinte forma (Milano e Roli, 2004): o Nível 0 possui os agentes responsáveis por construir a solução inicial, utilizando um procedimento construtivo probabilístico, baseado no rastro de feromônio; o Nível 1 implementa agentes de busca local para melhorar a solução; e os agentes do Nível 2 armazenam a melhor solução e atualizam os níveis de feromônio. Este processo é ilustrado pela Figura 4.5.

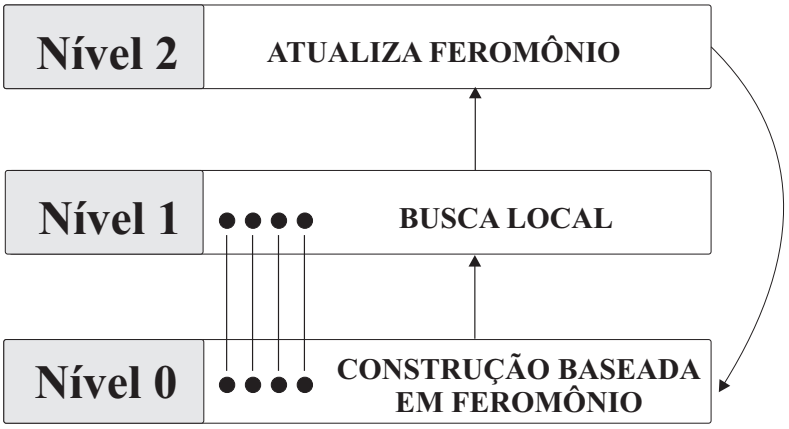

Figura 4.5: Especialização da arquitetura MAGMA para ACO.

A terceira especialização da arquitetura MAGMA é ilustrada na Figura 4.6. Esta especialização descreve a utilização da metaheurística ILS (*Iterated Local Search*) e se difere das anteriores somente no Nível 2. Como nas especializações anteriores da arquitetura MAGMA, (Milano e Roli, 2004) constroem a solução inicial com os agentes do Nível 0 e utilizam agentes algoritmos de busca local no Nível 1. O Nível 2 da especialização ILS possui agentes responsáveis por realizar a perturbação na solução atual, armazenar a melhor solução e enviar esta solução perturbada para os agentes do Nível 1, no qual o processo é reiniciado.

A quarta e ultima especialização básica da arquitetura MAGMA descreve a implementação de um Algoritmo Memético, ou seja, um algoritmo genético com busca local. O Nível 0 desta especialização implementa os agentes responsáveis por gerar uma população de soluções iniciais, que será enviada para os agentes do Nível 1,

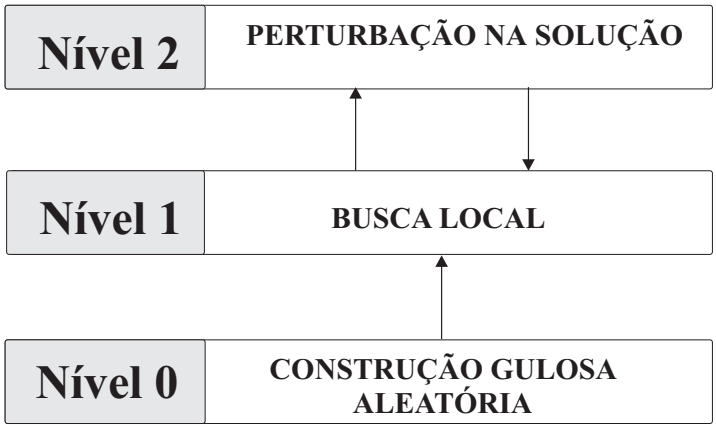

Figura 4.6: Especialização da arquitetura MAGMA para ILS.

que, por sua vez, aplicam um método de busca local em todas as soluções desta população. Os agentes do Nível 2 são responsáveis por aplicar todos os operadores genéticos (seleção, *crossover* e mutação) nas soluções recebidas dos agentes do Nível 1 e enviar a população sobrevivente para estes mesmos agentes, nos quais o processo é reiniciado. Esta especialização é ilustrada na Figura 4.7

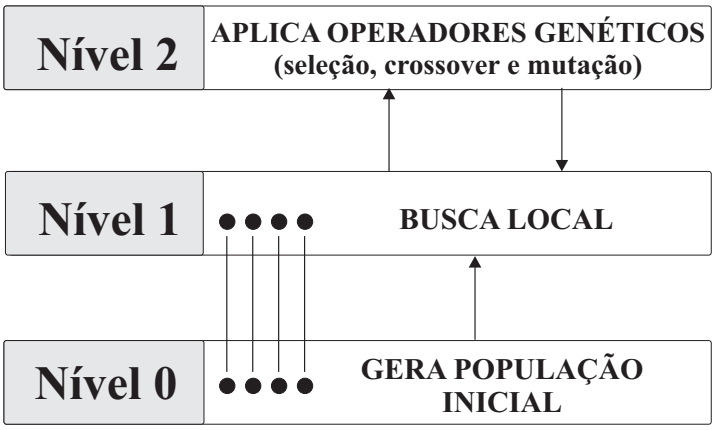

Figura 4.7: Especialização da arquitetura MAGMA para Algoritmo Memético.

Na versão cooperativa da arquitetura MAGMA, (Milano e Roli, 2004) propõem a combinação da especialização GRASP com a especialização Algoritmo Memético. Nesta especialização, ilustrada na Figura 4.8, o Nível 0 contém os agentes responsáveis por gerar a solução inicial para o GRASP e uma população de soluções para o Algoritmo Memético, da mesma forma como são geradas as soluções iniciais em suas respectivas especializações básicas. Os agentes do Nível 1 aplicam a busca local nestas mesmas soluções. O Nível 2 desta nova especialização também funciona de acordo com o Nível 2 destas duas especializações.

O diferencial desta especialização de cooperação para com as demais é a existência do quarto nível da arquitetura MAGMA, que possui agentes responsáveis pela coordenação e troca de informações entre os demais agentes da arquitetura, caracterizando, assim, um ambiente cooperativo na arquitetura. Esta cooperação é expressa por simples trocas de soluções entre os agentes, sendo que as melhores soluções encontradas pelos agentes de Nível 2 do GRASP são enviadas para a

população de soluções atuais do Algoritmo Memético e os melhores indivíduos do Algoritmo Memético constituirão a lista restrita de candidatos do GRASP.

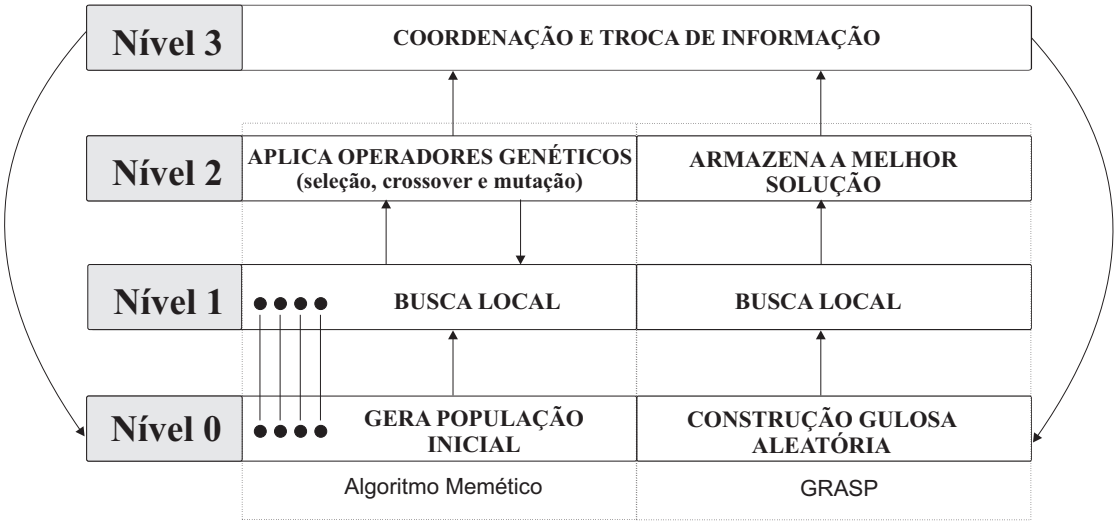

Figura 4.8: Especialização da arquitetura MAGMA Cooperativa.

Os agentes, na arquitetura MAGMA, são definidos por tuplas $T =< M, S >$, em que S é a estratégia usada pelo agente e M é o modelo de problema. Assim, todos os agentes, de todos os níveis da arquitetura, devem possuir uma definição do modelo do problema. Da forma como é proposta, então, a arquitetura está fortemente ligada ao problema a ser tratado, já que, para que dois problemas diferentes sejam solucionados, devem ser feitas alterações nos agentes, no que diz respeito à definição do problema.

4.6 *Asynchronous Teams*

(Talukdar e de Souza, 1990) propuseram uma Arquitetura Multiagente para solução de problemas, chamada de *Asysnchronous Teams* (*A-Teams*). Esta arquitetura possui agentes autônomos e cooperativos, com habilidades para modificar as soluções uns dos outros.

A arquitetura *A-Teams* tem sido usada na solução de diversos problemas complexos, envolvendo áreas como, por exemplo, projetos, diagnósticos, controle e otimização. Desta forma, vários trabalhos que implementam este formalismo podem ser citados: (Passos e Fonseca, 2005) utilizam *A-teams* na solução de problemas de seqüencialmento de produção; em (Lukin et al., 1997), este é usado no problema de tratamento de dados para ressonância da coluna vertebral; em (Chen, 1992), é usado no diagnóstico de falhas em redes elétricas; em (Gorti et al., 1996), é aplicado a problemas de satisfação de restrições; em (de Souza, 1993), na solução do Problema do Caixeiro Viajante; (Chen et al., 1993) utiliza-o na programação de horários de trabalho em lojas.

A estrutura do *A-Teams* pode ser definida, segundo (Talukdar e de Souza, 1990), como um Hipergrafo $\mathcal{DF} = (\mathcal{V}, \mathcal{W})$, sendo $\mathcal{V}$ um conjunto de nós, chamados na arquitetura de memórias, e $\mathcal{W}$ um conjunto de arestas (ou arcos), representando os agentes da arquitetura.

Como ilustrado pela Figura 4.9, os nós $M1$ e $M2$ são as memórias que armazenam conjuntos de soluções para o problema tratado. Este conjunto de soluções é também chamado de população de soluções. As arestas $(C1-C5)$ são os agentes construtores, ou seja, agentes que irão gerar soluções para a população de soluções, tanto as soluções iniciais quanto as intermediárias no processo de busca dos agentes são consideradas soluções da população de soluções. As arestas $D1$ e $D2$ são também agentes, porém destrutores.

As soluções presentes nas memórias, populações de soluções, podem variar com o tempo, pois os agentes estão a todo momento inserindo e removendo soluções das mesmas. Os agentes construtores, que podem implementar diversos tipos de heurísticas, ou seja, cada agente encapsula um método particular de solução do problema, que age modificando as soluções existentes (Passos e Fonseca, 2005), são responsáveis por gerar soluções iniciais para o problema, aplicar suas técnicas de busca nas soluções existentes na memória em que atuam e enviar novas soluções para esta memória. Já os agentes destrutures examinam uma população e apagam os resultados que julgarem ruins, baseados em uma lista de resultados que devem ser evitados. Este processo é semelhante as listas utilizadas em Busca Tabu (Talukdar et al., 1998).

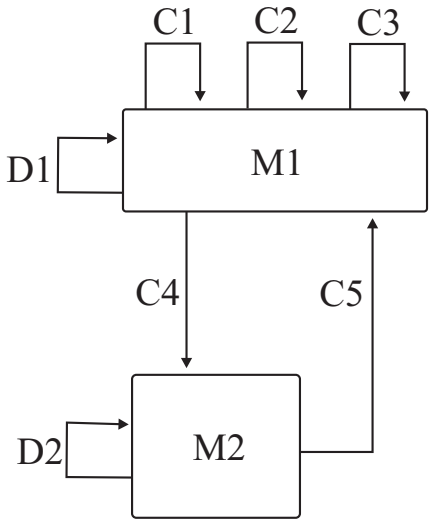

Figura 4.9: *Asyncronous Teams*: C1-C5 são agentes construtores; D1-D2 são agentes destrutores; e M1-M2 são memórias.

Todos os agentes do *A-Teams* são autônomos. O controle não é centralizado e não há hierarquia de supervisão entre os agentes.

A cooperação na arquitetura é dada por troca de informações entre os agentes, através da população de soluções, neste caso os agentes podem trabalhar nos resultados uns dos outros, já que todos os resultados são enviados para as memórias. Pelo fato dos agentes serem autônomos, a cooperação é assíncrona e nenhum agente pode ser forçado a esperar por resultados dos outros.

4.7 Considerações Finais

Neste capítulo são apresentadas as principais características de Agentes e Sistemas Multiagentes, que são sub-áreas de Inteligência Artificial Distribuída, bem como, dois dos principais sistemas multiagentes que se adequam aos interesses deste trabalho.

Projetar Sistemas Multiagentes é uma tarefa árdua, além de todas as dificuldades e detalhes a serem observados no projeto de um sistema para resolver um determinado problema, surge ainda, os conceitos e detalhes a respeitos de agentes a serem observados, como por exemplo, a dificuldade de integrar vários agentes de software, cooperando entre si, em um único sistema.

Em um sistema multiagente, como cada agente tem sua área de atuação, a atuação em conjunto permite que o sistema tenha um maior conhecimento do seu ambiente. O ambiente e as interações são elementos importantes em um sistema multiagente e podem exercer influência na forma de atuação dos agentes.

Vários trabalhos propõem o uso de agentes de softwares trabalhando em conjunto, em forma de sistemas multiagentes, na tentativa de melhorar a qualidade dos resultados obtidos através da cooperação entre estes diversos agentes de software, e também, para proporcionar arquiteturas dinâmicas e flexíveis, visando utilizar o maior número possível de conceitos para resolver um conjunto de problemas com o mínimo de esforço computacional.

Dentre estes trabalhos, destacam-se as arquiteturas híbridas, que reúnem diferentes metaheurísticas, cooperando entre si, para a resolução de um determinado problema, visando o ganho em qualidade das soluções e ganho em tempo computacional, como é o caso da arquitetura MAGMA e *A-Teams*. Estas arquiteturas propõe a utilização das abordagens multiagentes para o tratamento de metaheurísticas. Nestas propostas cada agente pode ser entendido como uma estratégia de busca diferente, atuando em um ambiente e compartilhando soluções de alguma forma.

As características de Sistemas Multiagentes apresentadas neste Capítulo, são amplamente utilizadas na definição da Arquitetura proposta neste trabalho.

Capítulo 5

Arquitetura Multiagente para Metaheurísticas

5.1 Introdução

O objetivo deste capítulo é apresentar a versão 1.0 da arquitetura AMAM - Arquitetura MultiAgente para Metaheurísticas, objeto central de desenvolvimento da presente dissertação. A versão inicial dessa arquitetura foi originalmente proposta em (Silva et al., 2007) para a solução de problemas de otimização combinatória.

Este capítulo, então, se inicia pela apresentação, na seção 5.2, da versão inicial da AMAM, conforme Silva et al. (2007), em seus aspectos de definições gerais. Em seguida, a seção 5.3 apresenta o modelo conceitual dessa arquitetura. A seção 5.5 introduz o conceito de cooperação para aplicação à arquitetura AMAM. A seção 5.6 apresenta, então, a proposta da arquitetura AMAM aplicando o conceito de cooperação entre multiagentes, sendo a principal contribuição dessa dissertação.

5.2 Visão Geral da AMAM

O modelo conceitual da arquitetura AMAM - Arquitetura MultiAgente para Metaheurísticas - apresentado em (Silva et al., 2007) e em (Silva, 2007), propõe o acoplamento de diversas metaheurísticas em uma estrutura geral, na qual diversos agentes buscam por melhores soluções para problemas de otimização através do trabalho cooperativo. A Figura 5.1 mostra a estrutura geral da arquitetura AMAM, composta por: Ambiente e Agentes, denominados Agente Construtor, Agente Busca Local, Agente Coordenador, Agente Analisador de Soluções e Agente Metaheurística.

Com a Arquitetura AMAM, (Silva, 2007) demonstra que a cooperação entre agentes metaheurísticos, acoplados em um mesmo ambiente, pode alcançar resultados melhores do que com estes agentes trabalhando individualmente. Nesta arquitetura, cada agente implementa uma certa metaheurística específica, e ainda, tem autonomia de ação em sua exploração do espaço de busca do problema. Através desta exploração, os agentes, em conjunto, buscam encontrar a solução ótima, ou pelo menos, uma solução de "boa" qualidade para o problema tratado (Silva, 2007).

Segundo (Silva, 2007), os agentes presentes na arquitetura AMAM compartilham

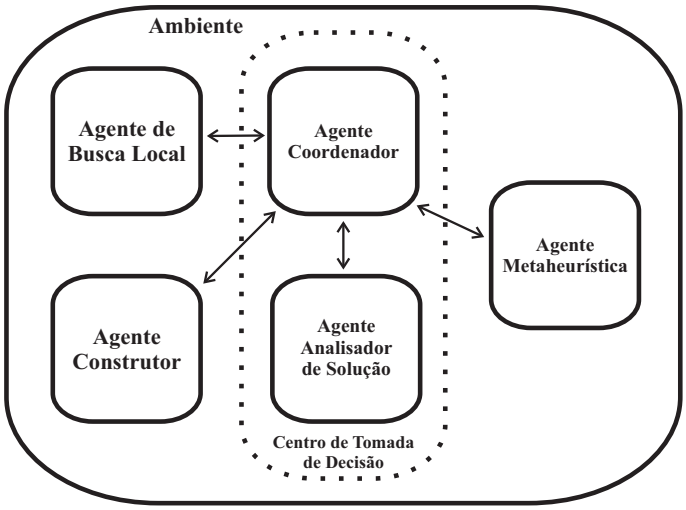

Figura 5.1: Estrutura Geral da AMAM.

informações acerca do ambiente e sobre o seu estado, isto é, trocam informações já encontradas e avaliadas. O trabalho cooperativo visa fundamentalmente melhorar a qualidade das soluções encontradas.

O modelo conceitual da AMAM propõe uma estrutura (*framework*), na qual agentes (heurísticas e metaheurísticas) podem ser adicionados e diferentes problemas de otimização podem ser solucionados sem grande esforço adicional. A estrutura define os tipos de agentes que podem ser utilizados, mas não limita quais e quantos agentes atuarão na busca pela solução. O ambiente, no qual os agentes atuam, faz parte da estrutura e representa o problema a ser solucionado. Assim, caso se altere o ambiente da arquitetura, altera-se também o problema a ser solucionado. A interação (comunicação) prevista na arquitetura acontece de duas formas distintas:

- agente-agente: através do Agente Coordenador, que centraliza as trocas de mensagens entre os agentes, mantendo-os independentes uns dos outros;
- agente-ambiente: através da troca de estímulos entre o ambiente e o agente, tornando, então, o ambiente, (também) um mediador da comunicação entre os diversos agentes da Arquitetura.

Cada agente possui sensores e efetores capazes se sentir/agir no ambiente. Os sensores permitem ao agente perceber as características do problema e as alterações do ambiente, como, por exemplo, o surgimento de novas soluções, que podem ser úteis na busca destes agentes. Os efetores possibilitam a construção de soluções, assim como a exploração do espaço de busca. Nesta arquitetura, uma solução é considerada um estímulo para os agentes.

Os sensores e efetores dos Agentes na Arquitetura AMAM, são definidos como estruturas de Entrada e Saída de dados. Para que um Agente "perceba" as características do Ambiente, por exemplo, ele deve "ler" as informações relacionadas a este Ambiente. Os Agentes agem (estrutura efetor) no Ambiente modificando a estrutura previamente percebida do mesmo, ou seja, a aplicação de uma estrutura de vizinhança, provocando um salto do Agente em questão no espaço de busca, carac-

teriza uma ação do Agente no Ambiente. Mais detalhes sobre ação e percepção dos Agentes com o Ambiente são apresentados na seção 5.4.

Um estímulo, na Arquitetura AMAM, é definido como um "gatilho" que aciona o item estimulado. Um Agente pode enviar e receber estímulos. Quando enviando estímulos, um Agente pode estar invocando um método em questão, ou agindo sobre o Ambiente. Quando recebendo estímulos, o Agente tem o acionamento de algum método, que irá receber algum tipo de informação de um outro Agente da Arquitetura.

Em termos práticos, o acionamento de algum método dentro de um Agente X, feito por outro Agente Y, caracteriza um estímulo enviado do Agente Y para o Agente X.

A arquitetura AMAM é uma arquitetura com alto nível de abstração, podendo facilmente serem adicionados ou removidos agentes da arquitetura, e ainda, esta arquitetura é flexível o bastante para permitir sua utilização em qualquer classe de problema de otimização combinatória, com o mínimo impacto no restante da arquitetura.

O modelo conceitual original da arquitetura AMAM, conforme proposto em (Silva et al., 2006) e (Silva et al., 2007), não admite uma estrutura cooperativa entre os agentes. É implementado um modelo de Lista Tabu de soluções que guiam os protótipos de agentes metaheurísticas no processo de descarte de sua soluções. Ou seja, é criada uma Lista Tabu de soluções consideradas ruins, que devem ser evitadas pelos agentes metaheurísticas. A cada vez que um agente metaheurística encontra uma solução que está presente nesta Lista Tabu, ele a evita e procura por outra para realizar seu processo de busca.

5.3 Modelo Conceitual da AMAM

Na arquitetura AMAM cada agente, em conjunto com os demais na exploração do espaço de busca do problema, busca encontrar a solução ótima, ou, pelo menos, uma solução de boa qualidade para o problema tratado. Os agentes colaboram uns com os outros, trocando e compartilhando informações sobre seu estado e acerca do ambiente entre, por exemplo, regiões promissoras do espaço de busca, regiões não recomendadas à exploração, soluções guias, soluções iniciais, parâmetros da busca, dentre outras informações. A seguir, cada componente da Arquitetura, ilustrada na Figura 5.1 é descrito em detalhes.

5.3.1 Ambiente

Segundo (Silva, 2007), o ambiente no qual os agentes metaheurísticos existem e atuam é parte integrante da arquitetura AMAM, devendo ser modelado para cada problema particular de otimização. A arquitetura também define e incorpora os mecanismos de comunicação entre os agentes e entre estes e seu ambiente. O esforço de modelagem, assim, para a criação de uma nova instância da AMAM, que permita o tratamento de novos problemas de otimização, é minimizado pela simples troca do ambiente. Este componente é, portanto, aquele que proporciona a flexibilidade da arquitetura. A forma geral do ambiente, apresentada na Figura 5.2,

define o modelo básico de um problema de otimização combinatória. Esta forma é definida considerando-se que, na modelagem de um problema de otimização, é necessário definir a forma como a solução do problema será representada (estrutura Solução), assim como todas as informações que precisam ser conhecidas para a busca de soluções (estrutura Ambiente).

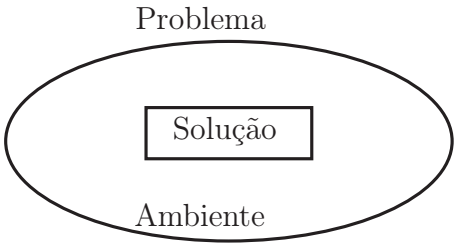

Figura 5.2: Modelo do Ambiente.

5.3.2 Agente Construtor

O Agente Construtor de Solução é constituído por uma heurística construtiva (Blum e Roli, 2003) e é responsável pela construção de soluções iniciais. O processo de construção das heurísticas envolve a soma de componentes em uma solução - inicialmente vazia - até que esta esteja completa. O estabelecimento de uma solução inicial para certo agente é equivalente a se estabelecer a localização desse agente no ambiente (espaço de busca), a partir do qual o agente iniciará seus movimentos exploratórios (Silva, 2007).

5.3.3 Agente Busca Local

Agentes de busca local são responsáveis por refinar soluções já encontradas, a partir de algoritmos de Busca Local (Blum e Roli, 2003). O objetivo do Agente Busca Local é caminhar, de vizinho a vizinho, a cada iteração, visando melhorar uma solução já obtida por outro agente, procurando por um ótimo local. Este agente tem movimentos limitados, o que o mantém dentro de sua vizinhança.

5.3.4 Agente Metaheurística

Os Agentes Metaheurísticas são agentes que utilizam algoritmos baseados em metaheurísticas para encontrar boas soluções para problemas de otimização combinatória (Silva, 2007). O objetivo principal destes agentes é realizar uma melhor exploração do espaço de busca, na tentativa de escapar dos ótimos locais por meio de diferentes estratégias. Estes agentes tem mais capacidade de se movimentar no ambiente (espaço de busca) e podem, eventualmente, realizarem um movimento de salto que os leve para outro ponto do espaço, fora de sua vizinhança corrente. Além disso, estes agentes podem, também, atuar de modo cooperativo, na tentativa de alcançarem seus objetivos. Mais detalhes sobre os agentes metaheurísticas e suas técnicas são demonstrados no Capítulo 6.

5.3.5 Agente Coordenador

(Silva, 2007) define um agente centralizador, Agente Coordenador, que tem a função de coordenar as atividades e a comunicação entre os agentes no sistema multiagente. Em seu trabalho de coordenação, este agente, além de realizar trocas de informações, tem também a responsabilidade por realizar negociações, acordos e tradução das mensagens entre os diferentes agentes. A comunicação entre os agentes acontece através da troca de mensagens. Desta forma, o agente coordenador conhece todos os outros agentes da arquitetura, e é responsável, quando recebe uma mensagem com alguma solicitação de serviço, por encaminhá-la para o agente que pode realizar a operação solicitada. Outros tipos de comunicação também podem ser realizadas através do ambiente.

5.3.6 Agente Analisador de Soluções

Agente responsável pela análise e tomada de decisão com relação às soluções obtidas pelos Agentes Metaheurísticos, selecionando, dentre elas, a melhor solução. A análise das soluções realizadas por este agente pode variar de acordo com a estratégia implementada pelo Agente Metaheurística, podendo ser, por exemplo, probabilística (Silva, 2007). Uma importante contribuição, obtida a partir do Agente Analisador, está na possibilidade de utilização de diferentes formas de escolha da "melhor" solução, baseada em técnicas mais complexas, sem que grande esforço de implementação seja feito.

5.4 Interação Agentes-Ambiente

As possibilidades de interação entre agentes e ambiente são determinadas pela capacidade do agente de perceber as características do problema (estabelecer uma localização no espaço de busca do problema) e de agir através da construção de soluções e da exploração do espaço de busca. Neste sentido, as funções de vizinhança representam uma das principais formas de ação sobre o ambiente, permitindo a construção de novas soluções a partir da modificação de soluções já existentes. É através destas funções de vizinhança (que na prática são movimentos) que os agentes são capazes de percorrer o espaço de busca, caminhando, de solução em solução, na busca pela melhor solução do problema.

Da mesma forma, na construção de soluções iniciais, são utilizadas diferentes formas de adição de elementos, que permitem que soluções vazias sejam incrementadas, a partir de um determinado critério de inserção de elementos, até que estejam completas e possibilitem o início do processo de busca. Estas formas de adição de elementos, denominadas como incrementos, capacitam o agente a agir sobre o ambiente construindo soluções. Entretanto, estas operações (movimentos e incrementos), que permitem ao agente atuar sobre a solução (no ambiente), são específicas para cada modelo de problema e precisam ser definidas dentro do ambiente.

As ações efetuadas por agentes no ambiente são possíveis através de estímulos enviados, pelos agentes, ao ambiente e que ativam as operações disponíveis neste. Da mesma forma, o ambiente emite estímulos (na forma de novas regiões no espaço

de busca do problema) que podem ser captados pelos agentes, caracterizando a percepção das características do ambiente (Silva, 2007).

A interação agente-agente na arquitetura acontece através de um modelo baseado em sistemas de quadro-negro. Esta interação consiste em uma lista tabu de soluções ruins, podendo os agentes ler e escrever informações nesta lista. Os agentes adicionam soluções consideradas ruins na lista e sempre que um agente obtém uma nova solução, ele a descarta se estiver na lista tabu de soluções ruins. Este método pode evitar que um agente trabalhe com uma solução que outro agente já considerou ruim, consequentemente, impede a tentativa de aplicar diferentes estratégias de busca nesta solução.

A interação entre os Agentes e entre estes e o Ambiente acontece a todo momento na Arquitetura AMAM. Os Agentes podem utilizar informações compartilhadas de outros Agentes, a todo momento. Mas no caso de falha de algum Agente na Arquitetura, os demais podem conseguir realizar seus processos de busca individualmente.

Esta importante característica da Arquitetura AMAM, a tolerância a falhas, ou seja, alguns Agentes podem falhar, e até mesmo parar de funcionar, que a Arquitetura continuará funcionando com os demais Agentes, proporciona um certo grau de independência para alguns Agentes.

Esta tolerância é muito importante e útil, mas tem suas limitações. O Agente Coordenador é o centro de funcionamento da Arquitetura AMAM, fazendo o papel de, em termos práticos, "programa principal" da Arquitetura, sendo o seu perfeito funcionamento, vital para toda a Arquitetura. Os Agentes Construtores de Solução Inicial são responsáveis por gerar todas as soluções iniciais para a Arquitetura, ou seja, eles devem ser capaz de gerar todas as soluções iniciais para que os Agentes Metaheurísticos sejam capazes de iniciar seus processos de busca. Após gerar as soluções iniciais, estes Agentes não são mais necessários.

Os Agente Metaheurísticas não dependem uns dos outros para funcionarem, ou seja, se algum Agente Metaheurística falar, os demais continuaram funcionando perfeitamente, porem sem as contribuições deste Agente que falhou. Os Agentes Metaheurísticas conseguem, também, funcionar sem a presença do Agente Coordenador, mas como dito anteriormente, este é necessário para o funcionamento geral da Arquitetura.

5.5 Estrutura Cooperativa para a Arquitetura AMAM

O objetivo desta seção é apresentar a estrutura de metaheurísticas cooperativas, proposta para a arquitetura AMAMv1.0.

Uma estrutura de multiagentes metaheurísticos cooperativos tem como principal objetivo guiar estes agentes metaheurísticos na exploração e diversificação do espaço de busca do problema tratado, visando melhorar seus resultados e/ou tempo de busca por regiões mais promissoras, e, ainda, visa integrar diversos agentes metaheurísticas de forma cooperativa, e verificar até que ponto esta cooperação traz benefícios para a resolução do problema (Roli e Milano, 2001).

A idéia principal desta estrutura de cooperação na arquitetura AMAMv1.0 é o compartilhamento de informações, utilizando uma estratégia de memória adaptativa compartilhada, que, por sua vez, através do agente coordenador, armazena, gerencia

e distribui estas informações de forma inteligente e eficiente, visando a interação, para solucionar problemas que estão além da capacidade ou conhecimento de cada agente metaheurística individualmente.

Nesta estrutura de cooperação, proposta através da arquitetura AMAMv1.0, podem ser implementados um número independente de agentes metaheurísticas, cada um executando uma estratégia de busca diferente, ou, até mesmo, as mesmas estratégias de busca, ou seja, mais de um agente implementando a mesma metaheurística. Na AMAMv1.0, os agentes metaheurísticas são implementados como processos independentes, na forma de *threads*, e por isso podem ser executados em paralelo, no mesmo processador, compartilhando informações através da estrutura de cooperação (Crainic e Toulouse, 2009). O modelo de cooperação paralela entre os agentes metaheurísticas envolvidos é alcançado através de trocas assíncronas de informação, e esta informação, pode ser, por exemplo, dada na forma de soluções do problema de otimização tratado.

O mecanismo principal da cooperação é o "repositório de soluções", ou, "*pool* de soluções". O repositório, ou *pool*, de soluções funciona como um armazém de soluções, no qual as informações serão armazenadas, organizadas e distribuídas entre todos os agentes envolvidos no processo de busca. Este repositório é diretamente ligado ao agente coordenador, que, por sua vez, é responsável por gerenciar o fluxo das informações no processo de cooperação. Neste esquema, sempre que um agente deseja compartilhar uma informação, este agente envia a informação para o repositório, através do agente coordenador. Consequentemente, sempre que um agente necessita de alguma informação compartilhada, ele solicita esta informação ao agente coordenador, que a solicita ao *pool* de soluções. Este processo se dá pela troca de estímulos entre os agentes presentes na arquitetura AMAMv1.0 e em caso de falhas, ou seja, na ausência do agente coordenador, os agentes metaheurísticas trabalham sozinhos, com suas próprias soluções.

A comunicação entre os agentes da arquitetura acontece sempre intermediada pelo agente coordenador, e pode ser realizada de duas formas, mas sempre compartilhando informações que dizem respeito às soluções do problema tratado. A primeira forma de comunicação é entre os agente presentes na arquitetura. Neste modelo, os agentes compartilham seus recursos, como por exemplo, uma estratégia de busca local, ou uma estratégia de construção de soluções. Quando um agente metaheurística necessita realizar uma busca local em sua solução, ele faz esta requisição ao agente coordenador, ou seja, envia um estimulo a este agente, solicitando uma busca local, que, por sua vez, estimula o agente que implementa uma heurística de busca local, recebendo, deste, uma solução ótima local do espaço de busca em questão. Por fim, o agente coordenador envia esta solução, que é um ótimo local, para o agente metaheurística que o solicitou. Um estímulo para a arquitetura é uma simples invocação de métodos, por exemplo quando um método $m1$ faz uma chamada a um método $m2$ da arquitetura, entende-se que o método $m1$ enviou um estímulo com seus parâmetros de chamada para o método $m2$ e o método $m2$, por sua vez, reponde a este estímulo com um possível retorno de sua execução.

A segunda forma de comunicação é entre os agentes e o *pool* de soluções, através de envio e requisição de novas soluções, sempre feitas por intermédio do agente coordenador. Esta forma de cooperação é detalhada na Seção 5.5.1.

As possibilidades de interação entre os agentes metaheurísticas e o *pool* de

soluções são determinadas pela capacidade destes agentes de agir através da construção de soluções e da exploração do espaço de busca. Neste sentido, as funções de vizinhança representam uma das principais formas de ação sobre o problema, permitindo a construção de novas soluções a partir da modificação de soluções já existentes. É através destas funções de vizinhança (denominados como movimentos) que os agentes metaheurísticas são capazes de percorrer o espaço de busca, caminhando, de solução em solução, na busca pela melhor solução do problema. Da mesma forma, na construção de soluções iniciais, os agentes construtores utilizam diferentes formas de adição de elementos, que permitem que soluções vazias sejam incrementadas, a partir de um determinado critério de inserção de elementos, definidos em cada um destes agentes, até que estejam completas e possibilitem o início do processo de busca. Estas formas de adição de elementos, denominadas como incrementos, capacitam os agentes construtores a agir sobre o problema construindo soluções.

Para caracterizar completamente o processo de cooperação, deve-se especificar, segundo (Bouthillier et al., 2005):

(i) A informação que será compartilhada;

(ii) Os métodos que compõem a busca cooperativa;

(iii) Quando as comunicações ocorrerão; e

(iv) Como cada agente tratará a informação compartilhada.

A informação compartilhada entre os agentes cooperativos tem que ser significativa, ou seja, tem que ser útil para os agentes que receberão estas informações. Nas implementações descritas neste trabalho, agentes compartilham recursos e soluções encontradas. Quando um agente melhora a solução compartilhada ou quando encontra uma nova solução, de boa qualidade, envia esta nova solução ao *pool* de soluções, por intermédio do agente coordenador.

5.5.1 A Estrutura de Cooperação

O *pool* de soluções, ou repositório de soluções, é o principal mecanismo da cooperação da arquitetura AMAMv1.0. Este repositório mantém soluções boas e factíveis para o problema tratado pela arquitetura AMAMv1.0, e é atualizado e controlado pelo agente coordenador, utilizando as informações recebidas por cada agente presente na arquitetura.

No *pool* de soluções é formado, então, um conjunto elite de soluções, mantendo as melhores soluções encontradas e descartando as piores, podendo cada agente solicitar informações sempre que necessário. Quando uma nova solução é enviada ao *pool* por algum agente, esta é organizada dentre todas as outras já existentes e a pior é eliminada. Esta estratégia é implementada no agente coordenador, e, portanto o *pool* de soluções se torna, efetivamente, um repositório de soluções, servindo simplesmente como uma central de armazenamento e distribuição de soluções, sem nenhuma inteligência ou técnica heurística implementada.

O tamanho da população no *pool* é proporcional à dimensão do problema tratado, na forma de P vezes a dimensão do problema, e, sempre que for necessário, as piores

soluções são eliminadas. Por questões de definição da arquitetura, não existe uma comunicação direta, no que diz respeito a trocas de soluções, entre os agentes metaheurísticas, ou seja, sempre que um agente metaheurística necessitar de alguma informação deve solicitar ao agente coordenador, que, por sua vez, irá solicitar a quem for necessário, inclusive ao *pool* de soluções, este procedimento visa garantir que toda comunicação realizada na arquitetura seja mediada pelo agente coordenador. A Figura 5.3 ilustra o processo de cooperação descrito.

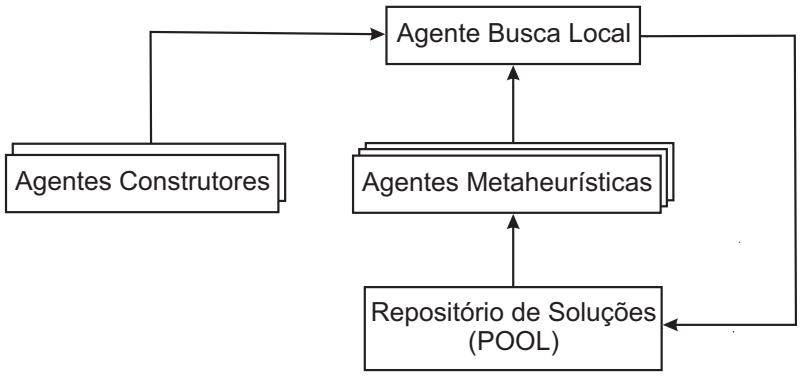

Figura 5.3: Diagrama de Cooperação para a AMAM v1.0

No diagrama apresentado pela Figura 5.3, os agentes construtores geram soluções iniciais para o problema. A escolha de qual heurística será utilizada é totalmente aleatória, escolhendo-se, por exemplo, uma dentre as citadas na seção 6.2.1. Logo após a geração de uma nova solução inicial para o problema, esta solução é enviada para um agente de busca local. O agente de busca local, que implementa o método da descida completa, envia suas soluções refinadas para o repositório de soluções, lembrando sempre que todas as trocas de informações na arquitetura aqui proposta são feitas através do agente coordenador e este processo demonstra o fluxo básico das soluções dentro da arquitetura.

O critério de seleção de soluções no *pool* implementado neste trabalho, se baseia no método de seleção da roleta, muito utilizado nos algoritmos evolutivos. Assim, as soluções são escolhidas com probabilidade na sua função de avaliação (veja Seção 6.2.6), ou seja, a mesma técnica utilizada para a avaliação de uma população nos algoritmos evolutivos, é utilizada para selecionar as soluções candidatas a sair do *pool*, sendo que as melhores soluções tem maior probabilidade de serem escolhidas.

Cada componente implementado na arquitetura AMAMv1.0 é descrito em detalhes na Seção 5.6 a seguir, e detalhes de implementação dos agentes heurísticos e metaheurísticos implementados são apresentados na Seção 6.2.

5.6 Arquitetura Multiagentes Cooperativos para Metaheurísticas - AMAM versão 1.0

Nesta seção, será detalhada a versão 1.0 da Arquitetura AMAM, ou seja, a proposta de desenvolvimento objeto deste trabalho.

5.6.1 Aspectos de Desenvolvimento da Arquitetura AMAM versão 1.0

O desenvolvimento de uma arquitetura de *software* consiste na tarefa de se construir um *framework* a partir de um modelo conceitual proposto. Neste sentido, a modelagem e desenvolvimento da versão 1.0 da arquitetura AMAM tomou por base o modelo conceitual proposto por (Silva, 2007), sumarizado na Seção 5.1.

Neste modelo conceitual, foram tratados os principais aspectos de um sistema multiagentes cooperativos, em uma arquitetura bem definida e flexível. A arquitetura AMAM possibilita a comunicação, a inserção e a remoção de agentes com muita facilidade, sem grandes impactos no restante da arquitetura. Outra característica fundamental deste modelo conceitual é a possibilidade, no sentido de flexibilidade, de se alterar o problema tratado com a troca do ambiente em que os agentes atuam. Para que esta arquitetura flexível seja, também, e efetivamente, uma arquitetura cooperativa, deve-se definir um modelo de cooperação entre os agentes existentes na arquitetura.

No modelo proposto neste trabalho, para a arquitetura AMAM, são acrescentados à estrutura original novas implementações de heurísticas e metaheurísticas, e também uma estrutura efetiva de cooperação. Esta proposta, ilustrada pela Figura 5.4, se diferencia da proposta original da arquitetura AMAM por possuir um maior número de agentes construtores de solução inicial e pela existência do Repositório de Soluções, que é responsável pela cooperação entre os agentes envolvidos no processo.

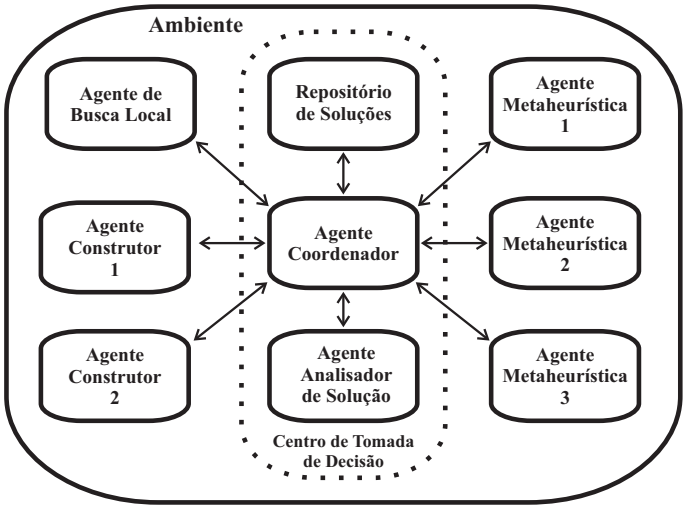

Figura 5.4: Estrutura de Cooperação da Arquitetura AMAM v1.0

A partir da idéia conceitual do trabalho de (Silva, 2007), o desenvolvimento da Arquitetura AMAM versão 1.0 segue, em todos os aspectos, as características e conceitos propostos em sua versão inicial, sendo que a principal proposta desta dissertação é o desenvolvimento de uma modelo cooperativo para a versão original da Arquitetura AMAM. Neste sentido, toda e qualquer comunicação na arquitetura é centralizada e controlada pelo agente coordenador, sendo este responsável por receber estímulos e enviá-los aos agentes presentes na arquitetura, ou seja, quando um agente que participa do processo de busca necessita de uma busca local, este solicita ao agente coordenador, e o agente coordenador, por sua vez, solicita esta

tarefa ao agente que implementa a operação em questão. Em seguida, encaminha o resultado para o agente que originou o estímulo para a busca local.

A Figura 5.4 apresenta o esquema de relacionamento dos agentes presentes na Arquitetura AMAMv1.0. Seguindo as características da arquitetura, todos os agentes atuam no ambiente do problema a ser tratado e estes se relacionam, sempre, através do agente coordenador. Neste sentido, atuam no ambiente, um agente de busca local, dois agentes construtores de soluções iniciais e n agentes metaheurísticas, podendo neste caso, ser removido algum agente existente ou, adicionados outros agentes. O centro de tomada de decisão é composto pelos agentes: Coordenador e Analisador de solução, além do Repositório de Soluções. O agente Analisador de solução é responsável por analisar soluções e fornecer suporte para que os demais agentes possam tomar decisões a respeito da qualidade das soluções tratadas. Assim como em todas as definições da arquitetura, o agente Coordenador é responsável por coordenar e centralizar toda a comunicação na arquitetura, é este agente que controla, através das regras definidas pelo mecanismo de cooperação, o Repositório de Soluções.

5.6.2 Modelo Conceitual da Metaheurística *Iterated Local Search* como um Agente da Arquitetura AMAM versão 1.0

A metaheurística ILS - *Iterated Local Search* - pode ser descrita por meio da arquitetura AMAM v1.0, na forma apresentada na Figura 5.5. Considerando a forma geral do algoritmo ILS, uma solução inicial é obtida do *pool* de soluções e o refinamento de soluções é feito, através do agente coordenador, pelo agente de busca local. A estratégia utilizada pelo ILS para escapar de ótimos locais é a execução de saltos no gráficos de soluções, através da estrutura de perturbação. Para a perturbação da solução corrente, é usado uma estrutura de vizinhança que modifica a solução, levando-a para outra região do espaço de busca.

Na arquitetura proposta, isto é possível por meio dos movimentos disponíveis no ambiente. Assim, a cada iteração, o Agente ILS emite estímulos ao ambiente, ativando um movimento escolhido para a perturbação da solução (Silva, 2007). Mais detalhes sobre o funcionamento da metaheurística ILS, bem como, seus parâmetros, estrutura de perturbação e busca local, para o PRVJT, são apresentados na Seção 6.2.4. Em seguida, a solução obtida é enviada para o agente coordenador, que, após melhorá-la através do agente de busca local, decidirá por enviá-la ou não para o *pool* de soluções. O agente coordenador toma todas as decisões a respeito de qualidade das soluções através do agente analisador de soluções, ou seja, o agente analisador de soluções avalia e calcula o quão boa é cada solução analisada e atribui estes valores às respectivas soluções, possibilitando definir quais as melhores soluções de acordo com vários critérios que podem ser aplicados à esta escolha.

5.6.3 Modelo Conceitual da Metaheurística Algoritmo Genético como um Agente da Arquitetura AMAM versão 1.0

A arquitetura AMAM v1.0 pode ser também utilizada para descrever a metaheurística Algoritmo Genético - AG. A Figura 5.6 mostra os agentes necessários para representar a metaheurística AG na arquitetura. Assim, o Agente Construtor, neste caso, não será sobrecarregado, visto que deve construir uma população de soluções

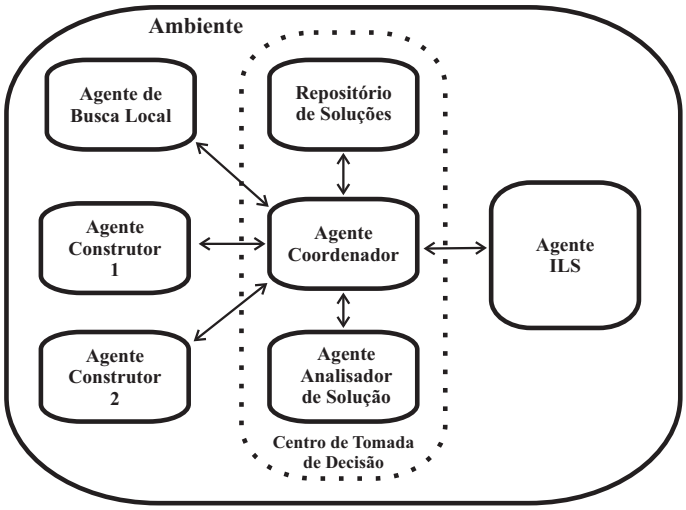

Figura 5.5: Arquitetura AMAM v1.0 para a Metaheurística ILS

iniciais para o agente AG, pois este recebe suas soluções do *pool* de soluções, que por sua vez, já recebeu todas as soluções necessárias para seu funcionamento inicial. Pode-se observar a existência do agente de busca local na especialização da Arquitetura AMAM v1.0 para a Metaheurística AG. Isto se dá pelo fato de que a busca local aqui implementada não é utilizada diretamente pelo agente AG e, sim, para refinar as soluções que entram no *pool* de soluções. Portanto, para este modelo cooperativo, a presença do agente de busca local será sempre necessária. Mais detalhes do funcionamento e implementações do agente metaheurística AG são apresentados na Seção 6.2.6.

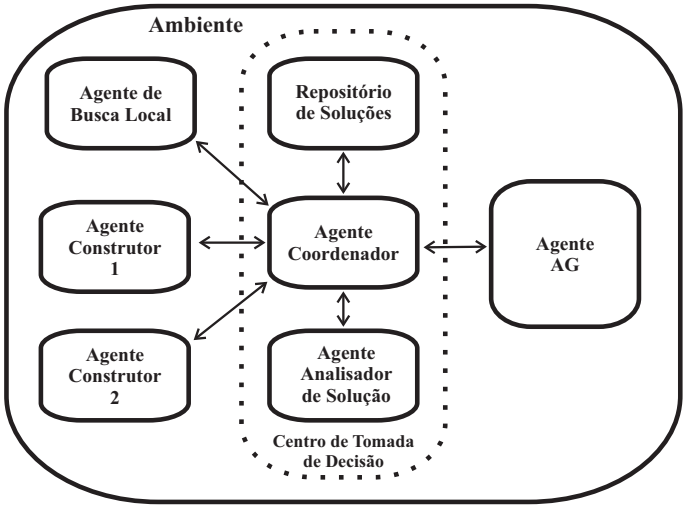

Figura 5.6: Arquitetura AMAM v1.0 para a Metaheurística AG

5.6.4 Modelo conceitual da Metaheurística *Variable Neighborhood Search* como um Agente da Arquitetura AMAM versão 1.0

A metaheurística VNS, *Variable Neighborhood Search*, assim como as demais metaheurísticas apresentadas, é facilmente descrita a partir da arquitetura AMAM v1.0. A Figura 5.7 mostra os agentes necessários para representar esta metaheurística na arquitetura. Melhores detalhes do funcionamento do agente metaheurística VNS são apresentados na Seção 6.2.5.

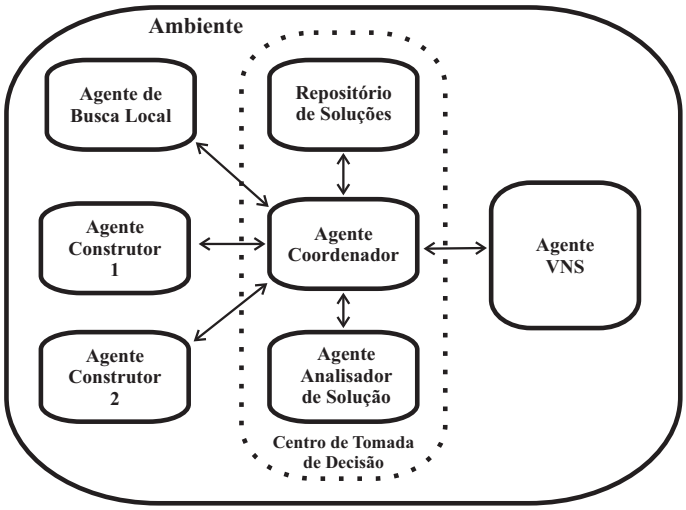

Figura 5.7: Arquitetura AMAM v1.0 para a Metaheurística VNS

5.7 Considerações Finais

A arquitetura AMAM combina diversas metaheurísticas, sendo que cada uma delas é definida como um agente autônomo, capaz de realizar a exploração do espaço de busca de um problema de otimização combinatória. Desta forma, o espaço de busca do problema é definido como o ambiente da arquitetura multiagente proposta (Silva, 2007).

A estrutura geral da arquitetura AMAM é apresentada com os principais elementos que a define. Logo em seguida é apresentada a proposta de desenvolvimento da Arquitetura AMAM versão 1.0, com suas principais contribuições e modificações da proposta inicial da Arquitetura. Os agentes presentes em ambas estruturas podem ser instanciados em diferentes configurações para a busca da solução do problema tratado, definido no ambiente da arquitetura.

Toda comunicação realizada na arquitetura é intermediada pelo agente coordenador. Este por sua vez, tem total controle e conhecimento de todos os demais agentes presentes na arquitetura. Sempre que algum agente necessita de alguma informação, seja uma nova solução ou um serviço de outro agente, este solicita ao agente coordenador que, através de sua influência na arquitetura, encaminha esta solicitação a um agente capaz de resolve-la. Todas as tomadas de decisão do agente coordenador são realizadas pelo agente analisador de soluções.

A principal diferença entre a primeira proposta da Arquitetura AMAM e sua versão 1.0 é a presença do Repositório de Soluções e o acréscimo de mais um Agente Construtor de soluções na versão 1.0 da arquitetura. O repositório de soluções transforma a arquitetura AMAM em um modelo efetivo de cooperação, deixando ainda mais evidente as características de um sistema MultiAgente.

Neste sentido, também foi definido, o modelo cooperativo utilizado nesta proposta da Arquitetura AMAMv1.0, bem como, suas principais características e técnicas adotadas para, efetivamente, possibilitar a cooperação entre todos os componentes da arquitetura.

Capítulo 6

Experimentos Computacionais

Neste capítulo, são apresentados os experimentos computacionais relativos à aplicação da Arquitetura AMAM v1.0 para a solução de instâncias de um problema clássico de otimização combinatória. Esta apresentação tem o objetivo de validar o funcionamento desta arquitetura, mostrando as possibilidades de sua utilização na resolução de problemas de otimização combinatória, de forma competitiva à resolução destes mesmos problemas já encontrada na literatura.

O problema utilizado para este estudo é o Problema de Roteamento de Veículos com Janela de Tempo. Para estes experimentos, foram utilizadas as especializações da AMAM v1.0 descritas na seção 5.6.

Este capítulo, então, é organizado como segue: a próxima seção apresenta o Roteamento de Veículos com Janela de Tempo (PRVJT). A seção 6.2 descreve as implementações particulares das especializações apresentadas na seção 5.6, quando utilizadas para a solução do PRVJT. As seções 6.3 e 6.4 apresentam os resultados relativos à aplicação da arquitetura para a solução do PRVJT.

6.1 Problema de Roteamento de Veículos com Janela de Tempo

6.1.1 Descrição Geral

O Problema de Roteamento de Veículos com Janela de Tempo, identificado pela sigla PRVJT, é uma generalização do Problema do Caixeiro Viajante (PCV). Neste problema, o objetivo é atender à demanda de um conjunto de clientes, geograficamente distribuídos, com um tempo determinado para o inicio do atendimento, chamado de janela de tempo, utilizando uma frota de veículos capacitados, ou seja, com limite de carga.

Problemas de Roteamento de Veículos (PRV) são problemas de otimização combinatória de grande relevância econômica, principalmente em sistemas logísticos e de transporte. O PRV é um problema de transporte que busca encontrar, dentre os possíveis caminhos, a melhor rota, partindo de um ponto central chamado de depósito, para atender clientes espalhados geograficamente por meio de uma frota de veículos. O PRV pode ser modelado para vários problemas do mundo real, como, por exemplo, um centro de distribuição de produtos. Uma excelente revisão do

Problema de Roteamento de Veículos pode ser encontrada em Toth e Vigo (2000).

O PRV aqui considerado pode ser definido na forma de um grafo, constituído de um conjunto de vértices e um conjunto de arestas, conectando pares de vértices. Este grafo pode ser representado na forma $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, sendo $\mathcal{G}$ um grafo completo (ou seja, existem arestas ligando todos os vértices); $\mathcal{V}$ o conjunto de vértices, cada um representando um cliente, além do depósito; e $\mathcal{A}$ o conjunto de arestas (arcos), demonstrando a ligação entre os clientes, ou seja, os caminhos existentes entre os pares de clientes, como também clientes com o depósito, além de suas respectivas distâncias. A figura 6.1 representa uma solução para um conjunto de 10 clientes e 1 depósito, na qual pode-se observar que uma demanda não negativa q_i é associada a cada vértice e um custo c_{ij} é associado a cada aresta, ou seja, cada cliente i da solução tem uma demanda de serviço q_i e cada viagem de cliente para cliente, ou cliente para depósito, ou depósito para cliente tem um custo de deslocamento c_{ij}, sendo i a cidade atual e j a cidade de destino. Dentre os vários objetivos que podem ter como guia para a solução deste tipo de problema, o mais comum é a minimização do custo de transporte, acumulando a distância total percorrida e o número total de veículos utilizados. O principal objetivo, assim, é o de minimizar o custo operacional total, maximizar o lucro e fornecer o melhor serviço para os clientes em questão.

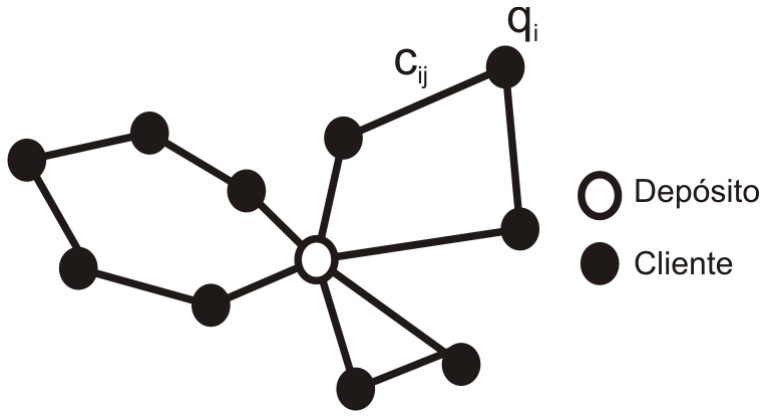

Figura 6.1: Grafo representando uma solução do Problema de Roteamento de Veículos.

Existem diversas variantes do PRV, que se diferenciam pelas restrições adicionadas à versão básica do problema. O Problema de Roteamento de Veículos com Janelas de Tempo (PRVJT) é a variante de interesse nesta dissertação, na qual estão presentes restrições de capacidade dos veículos e restrições de janela de tempo. No PRVJT, a cada cliente i está associado um período de tempo $[a_i, b_i]$ de atendimento, chamado de janela de tempo, sendo que a_i é o tempo de abertura da janela de tempo do cliente i e b_i é tempo de fechamento da janela de tempo deste mesmo cliente i. Associado a esta janela de tempo, existe também um tempo s_i, que determina o tempo de atendimento ou tempo de serviço para cada cliente i do problema.

6.1.2 Descrição da Função de Avaliação

A função de avaliação adotada para o PRVJT no presente trabalho tem, como principais objetivos, minimizar o custo associado, dado pela soma das distâncias

percorridas nas rotas da solução, e minimizar o número de rotas (veículos) que constituem cada solução. Esta função é responsável por avaliar o quão boa é uma solução, definida por um somatório de todas as distâncias percorridas; a soma dos excessos de capacidade de todos os veículos da solução; a soma das violações referentes às restrições de janela de tempo; e a soma do total de veículos da solução. Esta função é apresentada na expressão (6.1):

$$f(s) = \sum_{(i,j)\in A} d_{ij} + \alpha Q(s) + \beta J(s) + \gamma K(s) \tag{6.1}$$

sendo:

- s: uma determinada solução do problema;
- $\mathcal{A}$: conjunto dos arcos pertencentes à solução s;
- d_{ij}: distância entre os clientes (i, j);
- $Q(s)$: soma dos excessos de capacidade de todos os veículos da solução s;
- $J(s)$: soma das violações referentes às restrições de janela de tempo da solução s;
- $K(s)$: total de veículos da solução s;
- α, β e γ: fatores de penalidade não negativos, sendo definidos empiricamente como 1, 50 e 100, respectivamente.

A qualidade de uma solução para o PRVJT é constituída basicamente em termos do Número Total de Veículos utilizados nesta solução, ou seja, a quantidade de rotas que a constitui; e a Distância Total Percorrida, ou seja, a soma das distâncias de todas as rotas pertencentes à solução. Estes valores são utilizados como parâmetros de comparação para estabelecer uma relação de qualidade entre as soluções.

6.1.3 Problemas Testes

Para testar e validar a proposta da Arquitetura AMAM v1.0 na solução do PRVJT, foram utilizadas as 56 instâncias descritas em (Solomom, 1987), padrão para testes de desempenho de soluções do Problema de Roteamento de Veículos com Janelas de Tempo. Nestas instâncias, a classe de problemas C apresenta consumidores agrupados (veja Figuras 6.2 e 6.3); a classe de problemas R apresenta consumidores dispostos aleatoriamente (veja Figuras 6.4 e 6.5); e a classe de problemas RC é um misto das duas, apresentando consumidores aleatoriamente agrupados (veja Figuras 6.6 e 6.7). Nestas instâncias, as classes do tipo 1 ($C1$, $R1$ e $RC1$) são instâncias que permitem poucos consumidores atendidos de uma só vez, ou seja, poucos clientes em cada rota da solução, necessitando de um número maior de veículos. As classes do tipo 2 ($C2$, $R2$ e $RC2$) são instâncias que permitem um número maior de clientes atendidos de uma só vez, ou seja, permitem vários clientes em cada rota da solução, possibilitando um número menor de veículos em suas soluções.

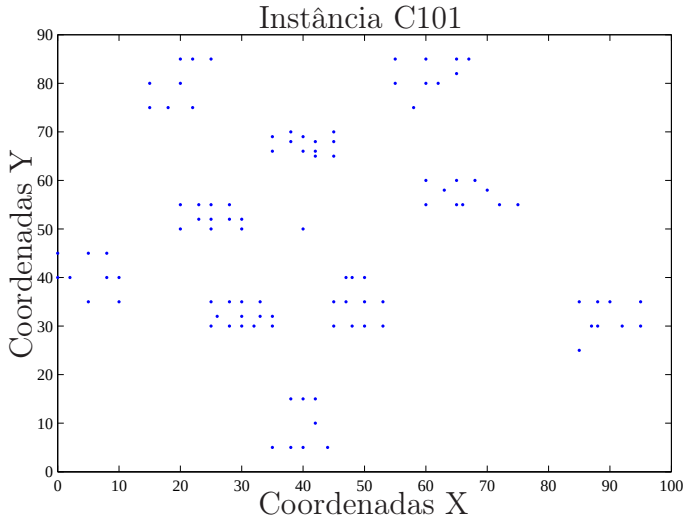

Figura 6.2: Distribuição dos clientes da instância C101 (Solomom, 1987).

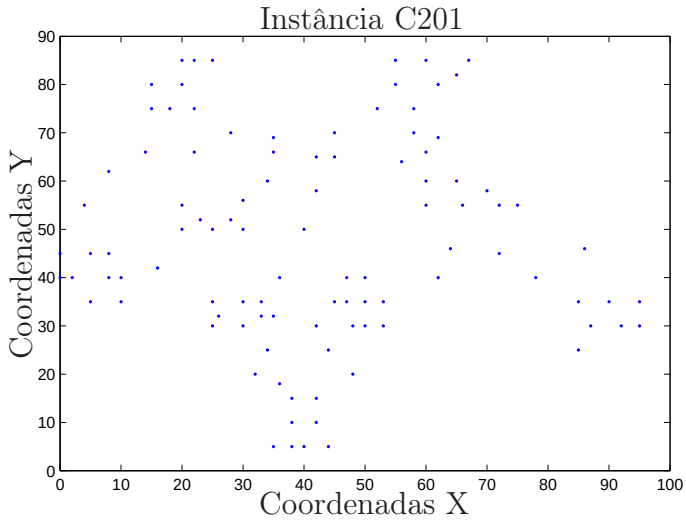

Figura 6.3: Distribuição dos clientes da instância C201 (Solomom, 1987).

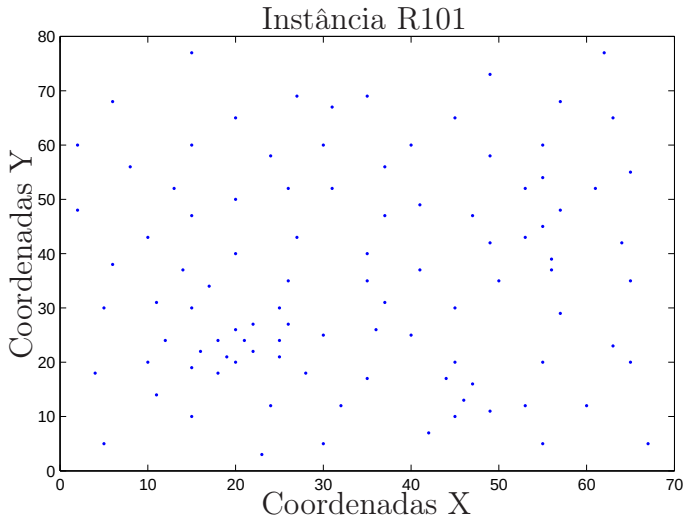

Figura 6.4: Distribuição dos clientes da instância R101 (Solomom, 1987).

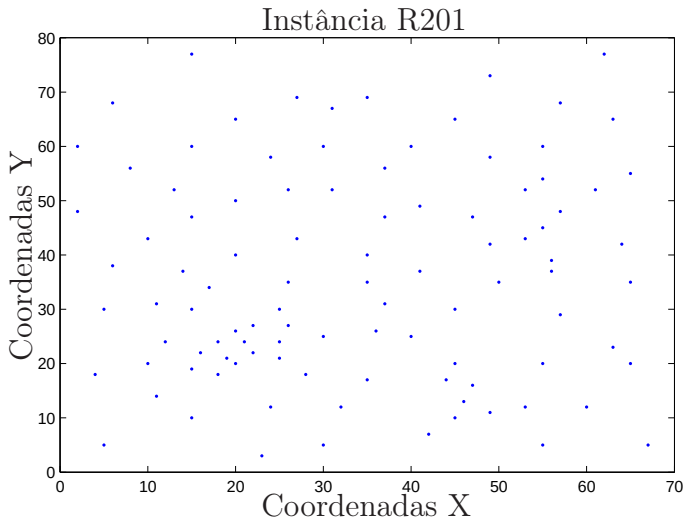

Figura 6.5: Distribuição dos clientes da instância R201 (Solomom, 1987).

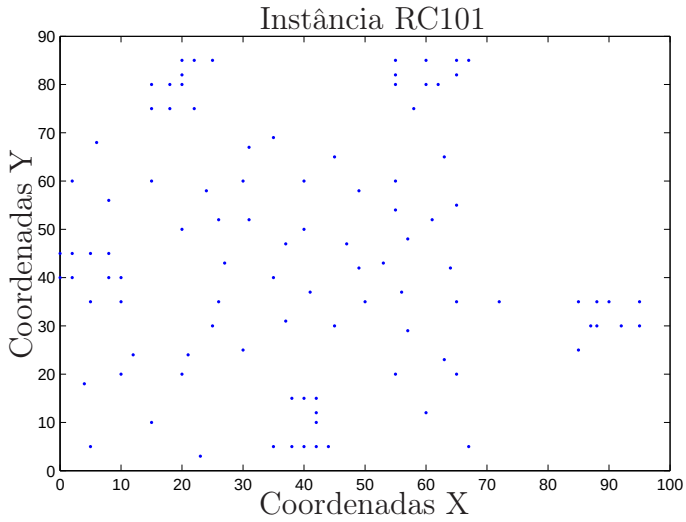

Figura 6.6: Distribuição dos clientes da instância RC101 (Solomom, 1987).

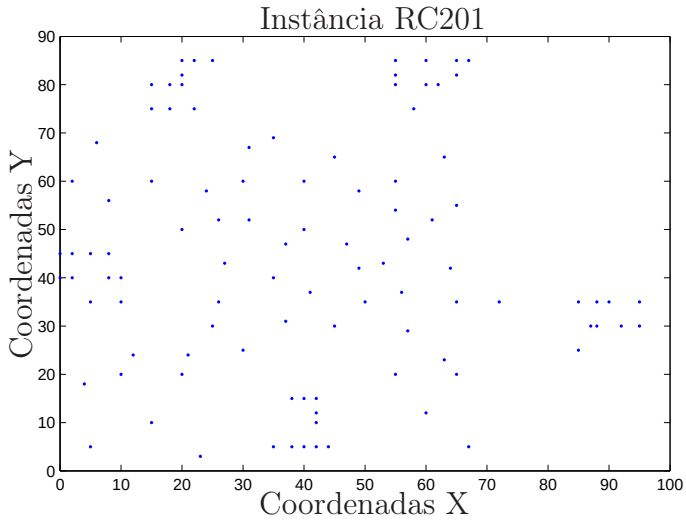

Figura 6.7: Distribuição dos clientes da instância RC201 (Solomom, 1987).

Cada uma das instâncias teste citadas é expressa por um arquivo texto contendo todas as informações pertinentes ao problema. A Figura 6.8 apresenta um modelo do arquivo das instâncias teste de Solomon. Estes arquivos contém, na primeira linha, dois números inteiros, sendo o primeiro referente ao número de clientes do problema, e o segundo referente à capacidade de carga de cada um dos veículos utilizados para este problema-teste. Nas instâncias em questão, todos os veículos apresentam a mesma capacidade de carga.

As demais linhas do arquivos são referentes ao Número de Identificação do Cliente; Coordenada X, referente à posição geográfica do cliente; Coordenada Y, referente à posição geográfica do cliente; Demanda do cliente, ou seja, a quantidade de itens a ser entregue neste cliente; o instante inicial da Janela de Tempo para o início do atendimento ao cliente; o instante final da Janela de Tempo para o início do atendimento ao cliente; e, por fim, o Tempo de Serviço gasto para atender o cliente. Para todas as instâncias teste, o cliente 0 (zero) é considerado como o depósito.

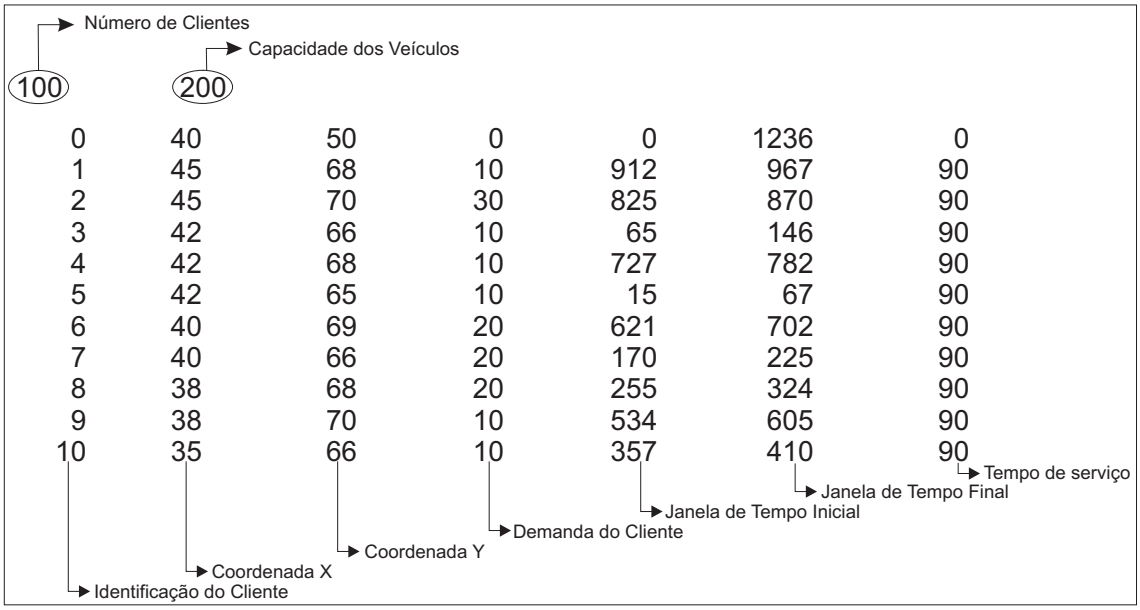

Figura 6.8: Arquivo da Instância C101.

6.2 Aplicação das Heurísticas e Metaheurísticas implementadas na Arquitetura AMAMv1.0

6.2.1 Heurísticas Construtivas

A Heurística construtiva é responsável pela construção de soluções iniciais (Blum e Roli, 2003). O processo de construção das heurísticas envolve a soma de componentes em uma solução - inicialmente vazia - até que esta esteja completa. O estabelecimento de uma solução inicial, para uma metaheurística, é equivalente a se estabelecer o ponto do espaço de busca do problema, a partir do qual a metaheurística iniciará seus movimentos exploratórios.

Para este trabalho, foram implementadas duas heurísticas construtivas distintas, na tentativa de diversificar as soluções iniciais e, consequentemente, diversificar os espaços de busca de atuação das metaheurísticas.

Na primeira implementação do agente construtor, foi desenvolvida uma heurística gulosa em relação à janela de tempo de cada cliente, de modo que o melhor cliente de cada incremento é escolhido e adicionado na solução, até que todos os clientes sejam atendidos. Esta forma de construção de solução gera soluções factíveis, porém, em geral, de qualidade ruim. Esta heurística gulosa, aqui implementada, é guiada pela função de avaliação do problema de Roteamento de Veículos com Janelas de Tempo, conseguindo assim, construir soluções factíveis para este problema, devido ao seu grande número de restrições. Para evitar que esta estratégia gulosa gere sempre soluções idênticas, é utilizada uma técnica heurística em conjunto com esta estratégia gulosa, selecionando 5% dos melhores clientes e escolhendo um de forma totalmente aleatória. Com esta abordagem, consegue-se gerar soluções diferentes a cada tentativa e, com isso, diversificar a população de soluções, e.g, para o Algoritmos Genético.

Para a segunda implementação de um agente construtor, foi desenvolvida uma heurística aleatória em conjunto com uma técnica de melhor inserção, baseada nas técnicas da heurística PFIH (a heurística PFIH é descrita em detalhes na seção 6.2.3), na qual um cliente é, a cada incremento, escolhido aleatoriamente e inserido na melhor posição na rota atual, utilizando, como guia, a função objetivo do problema em questão. Quando não for possível a adição de um novo cliente escolhido, na rota atual, uma nova rota será criada, e o processo se repete até que todos os clientes sejam atendidos. Esta heurística gera soluções factíveis diversificadas e, em geral, melhores que a versão gulosa.

6.2.2 Heurísticas de Busca Local

A heurística de refinamento utilizada pelos Agentes Busca é o método de descida. O método de descida analisa todos os vizinhos de uma dada solução inicial e aceita apenas as soluções que representem melhora na solução atual. Assim, como este método explora toda a vizinhança da solução inicial, ele para quando encontra um ótimo local. No método implementado, apresentado na Figura 6.9, a solução vizinha é obtida a partir da aplicação do movimento realocação sobre a solução inicial. Este movimento de realocação, que consiste em remover um cliente de uma rota e inseri-lo em outra posição na solução, não permite que novas rotas sejam geradas, e, assim, é possível diminuir o número de rotas durante o processo de refinamento da solução. A aplicação do método de descida com o movimento de realocação permitiu a redução significativa do número de rotas em relação a outras implementações realizadas.

6.2.3 Heurística *Push-forward Insertion*

Na atuação das metaheurísticas implementadas neste trabalho, é utilizada uma adaptação da heurística *Push-forward Insertion Heuristic* (PFIH), descrita inicialmente por (Solomom, 1987). A heurística PFIH é um eficiente método para inserir novos clientes em uma solução do problema ((de Oliveira, 2007), (Larsen, 1999)).

```
Algoritmo Descida(solucaoInicial)
1  inicio
2      solucaoCorrente = solucaoInicial;
3      para(todavizinhacadasolucao_inicial)
4              solucaoVizinha = realocacao(solucaoInicial);
5              se (solucaoVizinha < solucaoCorrente)
6                  solucaoCorrente = solucaoVizinha;
7      fim para;
8  fim.
```

Figura 6.9: Algoritmo de Busca Local.

Considere uma determinada rota $R_q = C_1, \ldots, C_{m+1}$, em que C_1 é o primeiro cliente atendido e C_m é o último, e o depósito central é definido como $C_0 = C_{m+1}$. A viabilidade de inserir um novo cliente na rota R_q é determinada inserindo-se o cliente entre todos os clientes já existentes nesta rota e escolhendo a inserção de menor aumento na distância total percorrida na rota R_q escolhida. A viabilidade de inserir um cliente C_i entre C_0 e o cliente C_1 é computada ajustando as janelas de tempo de todos os demais clientes pertencentes à rota R_q. Neste caso, a Janela de Tempo Inicial (tempo de chegada) do cliente C_1 será ajustada, podendo sofrer um deslocamento para frente e, consequentemente, todos os outros clientes pertencentes à rota R_q também poderão sofrer este deslocamento. Este processo é chamado de *Push Forward* ou, em uma tradução literal, "empurrar adiante".

Esta ação de "empurrar adiante" dos clientes pertencentes à rota R_q em alguns casos poderá ser nulo, pois podem existir folgas em algumas janelas de tempo. Este processo de reajuste de janelas de tempo termina quando é encontrado o primeiro caso que não sofra este deslocamento de janela de tempo, sendo que, no pior caso, o algoritmo verifica todos os clientes da rota escolhida R_q.

6.2.4 Metaheurística ILS

A metaheurística *Iterated Local Search* (ILS) é um método de busca iterativo que utiliza uma técnica de perturbações (modificações ou troca de vizinhança) da solução para escapar de ótimos locais correntes, ou visitar outros ótimos locais. Este método constrói soluções iterativamente, realizando perturbações em algumas soluções já encontradas, tendo, como principal objetivo, a diversificação da busca (Lourenço et al., 2002).

O processo de busca da metaheurística ILS consiste na perturbação de uma solução, obtida pela troca de vizinhança, gerando uma solução s'. Nesta nova solução, encontrada no processo de perturbação, é aplicada uma busca local, definida no método, e encontra-se uma nova solução s'', que é um ótimo local na vizinhança atual. O processo se repete até que um determinado critério de parada seja estabelecido. A cada iteração, o melhor ótimo local é armazenado como melhor solução do problema. O pseudo-algoritmo da metaheurística ILS é mostrado na Figura 2.9.

A definição da estrutura de vizinhança para o algoritmo ILS implementado neste trabalho utiliza uma adaptação do algoritmo *Push-Forward*, chamada de movimento k_{Otimo}, sendo que, neste caso, uma rota R_q é escolhida aleatoriamente e em seguida

K clientes são removidos desta rota, com K variando de 1 a N, sendo N o número de clientes pertencentes à rota R_q e K definido pela estrutura de vizinhança da metaheurística ILS. Os clientes removidos são reinseridos na solução utilizando o principio de inserção do algoritmo *Push-Forward Insertion Heuristic* - PFIH, escolhendo a melhor inserção, na solução atual, para cada um dos clientes removidos no passo anterior.

6.2.5 Metaheurística VNS

O método de busca em vizinhança variável (*Variable Neighborhood Search* - VNS) - é uma metaheurística para resolução de problemas de otimização combinatória, que consiste em uma busca local que explora o espaço de busca do problema, realizando trocas sistemáticas da função de vizinhança como estratégia para escapar de ótimos locais (Hansen et al., 2003). A cada iteração, é realizada uma busca local na solução corrente e comparado o resultado desta busca local com a própria solução corrente em questão. Caso uma melhora não seja encontrada, a função de vizinhança é alterada. O pseudo-algoritmo VNS, que mostra este processo, é apresentado na figura 2.7.

A definição da estrutura de vizinhança para o agente VNS implementado neste trabalho utiliza uma adaptação do algoritmo *Push-Forward Insertion Heuristic* - PFIH, conforme (Solomom, 1987). Neste algoritmo, uma rota R_q é escolhida aleatoriamente e, em seguida, K clientes são removidos desta rota, com K variando de 1 a N, sendo N o número de clientes pertencentes a rota R_q e K definido pela estrutura de vizinhança da metaheurística VNS. Os clientes removidos são reinseridos na solução, utilizando o principio de inserção do algoritmo *Push-Forward*, escolhendo a melhor inserção, na solução atual, para cada um dos clientes removidos no passo anterior.

6.2.6 Metaheurística AG

Um Algoritmo Genético (AG) pode ser entendido como um algoritmo de pesquisa probabilístico "inteligente", que pode ser aplicado em uma grande variedade de problemas de otimização combinatória. Os AGs simulam o processo de evolução de uma população de estruturas sujeitas às forças competitivas prescritas nos princípios da seleção natural e da "sobrevivência do mais bem adaptado" (Holland et al., 1992). Os genes dos indivíduos mais bem adaptados tendem a aparecer em um número maior de indivíduos nas gerações futuras, de modo que a combinação de suas características tende a gerar populações melhores. Assim, o processo de evolução é aleatório, porém, guiado por um mecanismo de seleção baseado na adaptação de estruturas individuais. O funcionamento de um AG consiste em gerar uma população inicial com N indivíduos e aplicar repetidamente os operadores genéticos de reprodução, cruzamento e mutação até que se encontre um critério para o fim da evolução. O pseudocódigo do Algoritmo Genético é descrito na Figura 2.10.

Na implementação do AG para este trabalho, a população inicial, na qual cada indivíduo é uma solução do problema, é gerada pela heurística construtiva, e o tipo da heurística construtiva (veja seção 6.2.1) utilizada para cada indivíduo da população é escolhido aleatoriamente, gerando assim soluções bem diversificadas.

Esta população é avaliada através da função de adaptação, definida na expressão (6.1), que determina o valor de cada indivíduo. A partir deste ponto, a cada iteração, conforme mostrado pela Figura 2.10, a população é incrementada, de modo a gerar novos indivíduos a partir da população anterior, utilizando as operações de *Crossover* e Mutação. A população é novamente avaliada através da função de adaptação e dessa serão escolhidos os novos indivíduos que passarão a ser a população atual, e o processo é repetido.

A escolha dos indivíduos que farão parte da nova população, ou seja, a definição da população sobrevivente, é feita pelo AG utilizando o método da roleta (Michalewicz, 1996). O método de seleção da roleta atribui a cada indivíduo de uma população uma probabilidade de aceitação ($F(x_i)$) proporcional ao valor da função de avaliação $f(x_i)$, dada em (6.1), em relação à somatória do valor da função de avaliação de todos os indivíduos da população em questão, como mostrado pela expressão (6.2). Assim, quanto maior o valor da função de avaliação de um dado indivíduo, maior a probabilidade dele ser escolhido. Sendo assim, a seleção de indivíduos pelo método da roleta pode fazer com que o melhor indivíduo seja perdido, ou seja, que este não seja escolhido em momento algum.

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{n} f(x_i)} \tag{6.2}$$

O operador de reprodução é utilizado para escolher quais os indivíduos mais adaptados irão passar as suas características para a nova população. No Agente AG, estes indivíduos, também chamados de pais, são selecionados aleatoriamente, ou seja, a cada nova população são escolhidos dois dentre todos os indivíduos existentes na população.

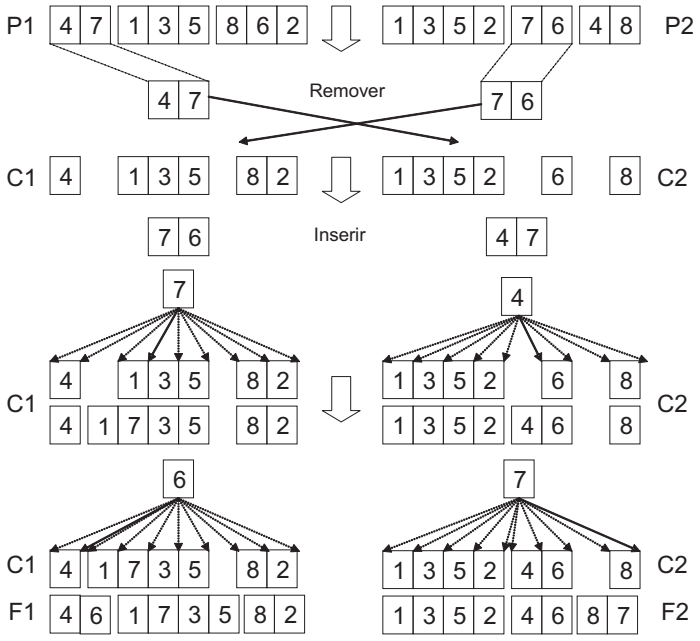

Figura 6.10: Demonstração da técnica BCRC (Ombuki et al., 2006).

Destes dois indivíduos escolhidos, outros dois serão gerados, chamados de in-

divíduos filhos. Esta é a fase de *Crossover*, que consiste em cruzar dois indivíduos (pais) selecionados e gerar dois novos indivíduos (filhos), a partir da combinação dos genes dos pais. O operador de *Crossover* utilizado é uma adaptação da técnica *Best Coast Route Crossover* - BCRC, descrita em (Ombuki et al., 2006), pois permite a garantia de factibilidade das janelas de tempo para o PRVJT. A adaptação do *Crossover* BCRC implementada neste trabalho consiste em escolher dois indivíduos pais, P_1 e P_2, para o cruzamento. Neste momento, é selecionada, aleatoriamente, uma rota R_1 de P_1 e outra rota R_2 de P_2, e todos os clientes pertencentes a rota R_1 são removidos de P_2 e reinseridos nesta mesma solução, utilizando-se os princípios básicos de inserção da heurística PFIH. O processo é o mesmo para o individuo P_1, do qual são removidos todos os clientes pertencentes à rota R_2 de P_2 e logo após reinseridos com a heurística adaptada PFIH. A Figura 6.10 demonstra a utilização deste método. Os resultados computacionais mostram que esta metodologia proporciona boas soluções e em tempo computacional baixo.

O operador de mutação é utilizado com a finalidade de proporcionar a diversidade da população, assegurando que o espaço de busca será explorado em uma parte significativa de sua extensão. O operador de mutação implementado neste trabalho utiliza uma adaptação do algoritmo PFIH, de modo que uma rota, escolhida aleatoriamente, é removida do indivíduo e logo após re-inserida no mesmo, utilizando o principio de inserção do algoritmo PFIH. Esta adaptação do operador de mutação mostrou-se bastante eficiente para o PRVJT.

6.3 Análise da ação da Arquitetura AMAMv1.0

A arquitetura AMAM v1.0, proposta neste trabalho, foi implementada e executada sobre a plataforma Java, versão J2SE 5.0, utilizando o IDE Eclipse, versão 3.4, no sistema operacional Windows XP SP3. Para os experimentos, cada uma das especializações dos agentes metaheurísticas utilizadas teve, como critério de parada, o número máximo de iterações, sendo configurados para executar 2.000 iterações cada. Este critério de parada foi definido empíricamente, após série de testes experimentais envolvendo as estruturas analisadas.

Foram realizados testes com os Agentes metaheurísticas AG, ILS e VNS, em separado e, também, em uma série de combinações destes três Agentes, apresentadas nas subseções seguintes. Os resultados obtidos com estes experimentos são discutidos na Seção 6.4. Justifica-se o uso destas metaheurísticas para efeito de avaliação dos resultados obtidos com a Arquitetura AMAM 1.0 devido à larga utilização de cada um, em particular, na solução de problemas gerais de otimização combinatória, bem como sua ampla utilização na solução do problema específico objeto de experimentação nessa dissertação, qual seja, o Problema de Roteamento de Veículos com Janela de Tempo (veja Seção 6.1).

Todos os experimentos seguem um padrão de comportamento da Arquitetura AMAMv1.0, conforme descrito a seguir, e, desta forma, se diferem somente na estratégia implementada por cada Agente Metaheurística.

O comportamento geral da Arquitetura pode ser descrito como segue. Após a inicialização da Arquitetura, o Agente coordenador reconhece e identifica o Ambiente, os Agentes Construtores, o Agente de Busca Local, o Agente Analisador de

Soluções, o Repositório de Soluções e os Agentes Metaheurísticas presentes. Para isso, ele recebe uma referência destes agentes, indicando que estão presentes na arquitetura. Esta referência de situação indica os canais de comunicação entre os agentes e o Agente Coordenador

Após o reconhecimento de todos os Agentes da Arquitetura, o processo de busca é iniciado, com o envio de soluções para o Repositório de Soluções. Estas soluções são solicitadas aos Agentes Construtores de Soluções pelo Agente Coordenador.

No caso da implementação realizada nesta dissertação para a validação da arquitetura, que envolve a solução de instâncias do Problema de Roteamento de Veículos com Janela de Tempo, serão enviadas para o Repositório de Soluções até $10N$ soluções a serem armazenadas, sendo N o número de clientes da solução da instância tratada. O tamanho máximo do Repositório é determinado pela memória alocada para esse fim e definido, portanto, pelo *hardware* utilizado para a implementação da Arquitetura (veja Seção 5.5.1).

Após a inicialização do Repositório de Soluções, o Agente Coordenador envia estímulos aos Agentes Metaheurísticas, permitindo o inicio de suas operações.

O primeiro passo de um Agente Metaheurística é a obtenção de solução(ões) inicial(is) para o seu processo de busca (veja Seção 2.1. Para a obtenção desta(s) solução(ões), o Agente Metaheurística envia um estímulo de solicitação para o Agente Coordenador, que, por sua vez, obtém as soluções necessárias do Repositório de Soluções e as envia para o Agente Metaheurística solicitante. Este estímulo significa que é acionada, pelo Agente Metaheurística, uma funcionalidade do Agente Coordenador, funcionalidade essa que realiza as ações solicitadas, quais sejam, obter as soluções necessárias do Repositório e enviá-las para o Agente Metaheurística solicitante. Cabe ressaltar que as regras para a seleção de soluções no Repositório de Soluções foram discutidas na Seção 5.5)

A seguir, os Agentes Metaheurísticas existentes na Arquitetura iniciam seus processos de busca, de acordo com suas características. A cada iteração de um determinado Agente, soluções armazenadas no Repositório de soluções são solicitadas ao Agente Coordenador e comparadas, pelo Agente Analisador de Soluções, com as soluções atuais desta iteração, para determinar que soluções farão parte do processo de busca da próxima iteração. Sempre que um Agente Metaheurística conclui uma iteração do seu processo de busca, ele envia a solução para o Agente Coordenador, que decidirá por adicioná-la, ou não, ao Repositório de Soluções. Cabe ressaltar, novamente, que as regras para a inserção ou não de uma solução no Repositório de Soluções foram discutidas na Seção 5.5). Este processo é repetido até que todas as iterações dos Agentes Metaheurísticas sejam concluídas.

6.3.1 Aplicação de 1 Agente AG na Arquitetura AMAMv1.0

O experimento prático da aplicação de 1 Agente AG para a solução do PRVJT equivale ao desempenho do próprio algoritmo genético implementado, pois, neste caso, não existe nenhuma forma de comunicação, no sentido de cooperação, entre agentes metaheurísticas, com troca de informações relevantes para o sucesso do processo de busca, uma vez que há um único agente sendo utilizado. Neste experimento, toda a Arquitetura AMAMv1.0 funciona segundo os padrões definidos anteriormente, tendo todos os demais agentes presentes na configuração em questão

participando do processo de busca.

Nesta aplicação prática, participam do processo de busca os seguinte componentes: (1) Ambiente; (2) Agentes Construtores; (3) Agente de Busca Local; (4) Agente Coordenador; (5) Agente Analisador de Soluções; (6) Repositório de Soluções; e (7) Agente Algoritmo Genético. O modelo conceitual da Arquitetura AMAMv1.0 para a aplicação de 1 Agente AG é ilustrado na Figura 5.6. Os resultados computacionais desta aplicação são apresentados nas Tabelas 6.1, 6.2 e 6.3 na Seção 6.4.

O esquema de funcionamento desta aplicação prática pode ser visto como o funcionamento normal de qualquer Algoritmo Genético em um processo de busca. Ao iniciar o seu processo de busca, o Agente AG solicita uma população de soluções iniciais ao Agente Coordenador, e este, por sua vez, retira estas soluções no Repositório de Soluções e as envia para o Agente AG solicitador. Ao término de cada iteração, o Agente AG envia sua população sobrevivente para o Agente Coordenador, e este decidirá por enviá-las, ou não, para o repositório de Soluções.

A cada nova iteração, o Agente AG solicita ao Agente Analisador de Soluções uma comparação entre os pais (soluções) selecionados para o processo de busca do AG, com novas soluções obtidas do Repositório de Soluções, através do Agente Coordenador. Este processo faz parte da estratégia de cooperação da Arquitetura AMAMv1.0 e tem como objetivo estabelecer a comunicação (troca de informações) entre todos os Agente Metaheurísticas existentes na Arquitetura. Mas, como, neste caso, só existe 1 Agente AG presente na Arquitetura, não existe efetivamente, esta comunicação.

6.3.2 Aplicação de 1 Agente ILS na Arquitetura AMAMv1.0

Assim como o experimento anterior, o experimento prático com a aplicação de 1 Agente ILS não oferece ganhos com relação à cooperação entre os agentes metaheurísticas, pois, neste caso, é implementado somente 1 Agente ILS como principal ferramenta de busca. Equivale, do mesmo modo que ressaltado quanto ao AG, à aplicação da metaheurística ILS implementada para a solução do PRVJT.

Nesta aplicação prática, participam do processo de busca os seguinte componentes: (1) Ambiente; (2) Agentes Construtores; (3) Agente de Busca Local; (4) Agente Coordenador; (5) Agente Analisador de Soluções; (6) Repositório de Soluções; e (7) Agente ILS. O modelo conceitual da Arquitetura AMAMv1.0 para a aplicação de 1 Agente ILS é ilustrado na Figura 5.5. Os resultados computacionais desta aplicação são apresentados nas Tabelas 6.1, 6.2 e 6.3 na Seção 6.4.

No inicio de seu funcionamento, o Agente ILS solicita uma solução inicial ao Agente Coordenador e este envia uma solução extraída do Repositório de Soluções a este Agente solicitador. Ao final de cada iteração, o Agente ILS envia a melhor solução desta iteração para o Agente Coordenador decidir se adiciona ou não esta solução ao Repositório de Soluções. No inicio de cada iteração, o Agente ILS solicita ao Agente Analisador de Soluções uma comparação entre a solução atual desta nova iteração e uma nova solução obtida do Repositório de Soluções, para verificar qual a melhor opção de solução para esta nova iteração, e este processo se repete até que todas as iterações sejam concluídas.

6.3.3 Aplicação de 1 Agente VNS na Arquitetura AMAMv1.0

O experimento prático da aplicação de 1 Agente VNS na Arquitetura AMAM v1.0, assim como os dois anteriores, não oferece ganhos com relação à cooperação entre os agentes metaheurísticas, pois neste caso, é implementado somente 1 Agente VNS como principal ferramenta de busca, não tendo, neste processo, outros agentes implementando estratégias eficientes de busca para compartilhar informações sobre o espaço de busca do problema. Também é equivalente à aplicação desta estrutura de metaheurística VNS para a solução do PRVJT.

Nesta aplicação prática, participam do processo de busca os seguinte componentes: (1) Ambiente; (2) Agentes Construtores; (3) Agente de Busca Local; (4) Agente Coordenador; (5) Agente Analisador de Soluções; (6) Repositório de Soluções; e (7) Agente VNS. O modelo conceitual da Arquitetura AMAMv1.0 para a aplicação de 1 Agente VNS é ilustrado na Figura 5.7. Os resultados computacionais desta aplicação são apresentados nas Tabelas 6.1, 6.2 e 6.3 na Seção 6.4.

O esquema de funcionamento do Agente VNS na arquitetura AMAMv1.0 é similar ao esquema do Agente ILS descrito na Seção 6.3.2.

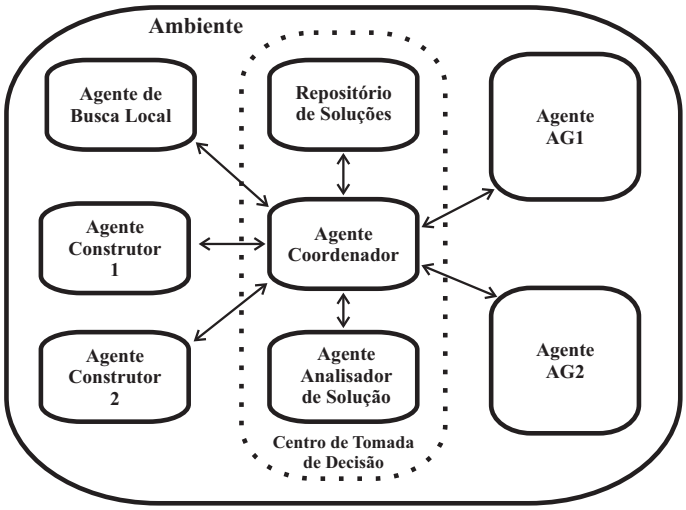

Figura 6.11: Modelo Conceitual da Arquitetura AMAMv1.0 para 2 Agentes AG

6.3.4 Aplicação de 2 Agentes AG na Arquitetura AMAMv1.0

Este experimento prático se caracteriza pela existência de 2 Agentes Algoritmo Genético na Arquitetura AMAMv1.0. Nesta aplicação prática, participam do processo de busca os seguintes componentes: (1) Ambiente; (2) Agentes Construtores; (3) Agente de Busca Local; (4) Agente Coordenador; (5) Agente Analisador de Soluções; (6) Repositório de Soluções; e (7) Agente AG1; e (8) Agente AG2. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura 6.11.

Neste caso, a cooperação existe entre os Agentes AG1 e AG2. Como em todos os outros agentes metaheurísticas, quando os agentes AG necessitam de soluções iniciais, enviam estímulos para o Agente Coordenador e este, por sua vez, retorna com estas soluções, retiradas do Repositório de Soluções. Estas soluções constituem as populações iniciais dos Agentes AG.

Os Agentes AG utilizam estas soluções, chamadas de população inicial, em seu processo de busca e, ao término de cada iteração, os Agentes AG enviam suas populações sobreviventes para o Agente Coordenador, e este decidirá por enviá-las ou não para o repositório de Soluções. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.4, 6.5 e 6.6 na Seção 6.4.

A cada nova iteração, cada um dos Agentes AG solicita ao Agente Analisador de Soluções uma comparação entre os pais (soluções) selecionados para o processo de busca do AG, com novas soluções obtidas do Repositório de Soluções, através do Agente Coordenador. Neste processo é que se dá a cooperação da arquitetura, pois, nesta possível troca de soluções, um Agente AG pode utilizar uma solução compartilhada por outro Agente AG, possibilitando, assim, a melhora das soluções utilizadas no processo de busca destes Agentes Metaheurísticas.

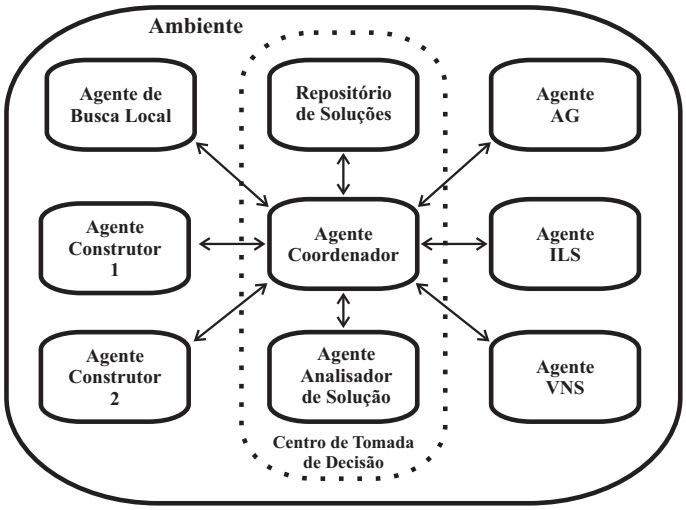

Figura 6.12: Especialização da Arquitetura AMAM v1.0 para a combinação dos 3 agentes, AG, ILS e VNS

6.3.5 Aplicação da combinação de todos Agentes (AG, ILS e VNS) na Arquitetura AMAMv1.0

A aplicação dos três Agentes implementados neste trabalho, trabalhando em conjunto na Arquitetura AMAMv1.0, é uma das formas mais completas, dentre todos os testes realizados. Este experimento prático envolve os agentes AG, ILS e VNS em uma busca cooperativa paralela (em um único processador), trocando soluções pertinentes ao problema tratado. Neste experimento, participam do processo de busca os seguintes componentes: (1) Ambiente; (2) Agentes Construtores; (3) Agente de Busca Local; (4) Agente Coordenador; (5) Agente Analisador de Soluções; (6) Repositório de Soluções; (7) Agente AG; (8) Agente ILS; e (9) Agente VNS. Esta instância da Arquitetura AMAMv1.0 é ilustrado na Figura 6.12.

Neste modelo de busca cooperativa, a cada iteração, os agentes recebem soluções do Repositório de Soluções, via Agente Coordenador, para escolher (via Agente Analisador de Soluções) dentre estas e sua soluções atuais. A cada nova iteração, enviam suas soluções para o Agente Coordenador enviá-las ao Repositório, se este for o caso.

Assim como em todas as outras configurações da Arquitetura AMAMv1.0, nesta configuração todos os agentes podem trabalhar, em algum momento, com soluções geradas pelos outros agentes presentes na Arquitetura. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.10, 6.11 e 6.12.

6.3.6 Outras configurações de teste da Arquitetura AMAMv1.0

Como pode-se observar nas seções anteriores, todas as configurações de teste da Arquitetura AMAMv1.0 seguem um mesmo padrão de comportamento, diferindo umas das outras apenas pelo número de Agentes Metaheurísticas presentes nas mesmas.

A configuração dos demais agentes e da própria arquitetura não é alterada para nenhum dos testes realizados e até mesmo a configuração dos agentes metaheurísticas não se altera para estes testes, sendo estes simplesmente adicionados e/ou removidos da arquitetura em cada bateria de testes.

Os modelos destas diversas configurações de testes da arquitetura AMAMv1.0, que não foram apresentados neste capítulo, são apresentados no Apêndice A.

6.4 Apresentação dos Resultados dos Experimentos

Esta seção apresenta os resultados obtidos para a solução das instâncias do PRVJT, já descritas na seção 6.1.3, utilizando as configurações da Arquitetura AMAMv1.0 apresentadas nas seções 6.3.1 a 6.3.5.

Tabela 6.1: Resultado para a aplicação individual, na Classe C, dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG		1ILS		1VNS		Melhor	
C101	827,29	**10**	827,29	**10**	852,60	**10**	828,94	10
C102	827,29	**10**	827,29	**10**	827,30	**10**	828,94	10
C103	826,29	**10**	826,29	**10**	1010,70	**10**	828,06	10
C104	830,00	**10**	830,00	**10**	1023,59	**10**	824,78	10
C105	827,29	**10**	827,29	**10**	904,90	**10**	828,94	10
C106	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C107	827,29	**10**	827,29	**10**	886,90	11	828,94	10
C108	827,29	**10**	827,29	**10**	927,19	11	828,94	10
C109	827,29	**10**	827,29	**10**	948,89	**10**	828,94	10
C201	589,09	**3**	589,09	**3**	589,09	**3**	591,56	3
C202	589,09	**3**	589,09	**3**	589,10	**3**	591,56	3
C203	588,69	**3**	588,69	**3**	600,90	**3**	591,17	3
C204	588,09	**3**	588,09	**3**	605,69	**3**	590,60	3
C205	586,39	**3**	586,39	**3**	586,39	**3**	588,88	3
C206	585,89	**3**	585,89	**3**	586,00	**3**	588,49	3
C207	585,79	**3**	585,79	**3**	585,79	**3**	588,29	3
C208	585,79	**3**	585,79	**3**	585,79	**3**	588,32	3

Tabela 6.2: Resultado para a aplicação, na Classe R, individual dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG		1ILS		1VNS		Melhor	
R101	1658,39	20	1668,90	**19**	1714,09	**19**	1645,79	19
R102	1485,90	18	1519,00	**17**	1529,00	18	1486,12	17
R103	1270,29	14	1236,39	14	1265,29	14	1292,68	13
R104	1002,10	10	988,20	10	1033,39	11	1007,24	9
R105	1405,40	**14**	1366,79	15	1424,20	16	1377,11	14
R106	1285,50	13	1259,19	13	1341,39	14	1251,98	12
R107	1105,20	11	1083,70	11	1181,70	12	1104,66	10
R108	964,00	10	963,50	10	1004,09	10	960,88	9
R109	1191,10	12	1171,60	12	1234,20	13	1194,73	11
R110	1114,79	11	1094,80	12	1170,10	13	1118,59	10
R111	1153,10	11	1071,29	11	1148,80	12	1096,72	10
R112	1000,40	10	1034,00	10	1007,80	11	982,14	9
R201	1251,29	**4**	1261,80	**4**	1276,90	**4**	1252,37	4
R202	1100,90	4	1089,79	4	1151,59	4	1191,70	3
R203	969,30	**3**	948,69	**3**	1008,29	**3**	939,50	3
R204	754,40	3	766,59	3	778,09	3	825,52	2
R205	1044,70	**3**	1048,60	**3**	1077,49	**3**	994,42	3
R206	944,50	**3**	950,39	**3**	986,30	**3**	906,14	3
R207	810,69	3	872,79	3	858,60	3	890,61	2
R208	732,10	**2**	755,60	**2**	792,09	**2**	726,75	2
R209	940,19	**3**	948,20	**3**	1004,60	**3**	909,16	3
R210	975,30	**3**	991,09	**3**	980,20	**3**	939,34	3
R211	818,50	3	807,60	3	852,69	3	885,71	2

As Tabelas 6.1, 6.2 e 6.3 apresentam os resultados da aplicação individual dos Agentes AG, ILS e VNS em todas as 56 instâncias testes descritas na Seção 6.1.3. Estes resultados são comparados com os melhores resultados encontrados na literatura, segundo o sítio `http://www.sintef.no/Projectweb/TOP/Problems/`.

Em todas estas tabelas, são exibidos, na primeira coluna, as instâncias-teste aplicadas e na última coluna, os melhores resultados encontrados na literatura para as instâncias em questão, nos termos postos. Nas demais colunas das tabelas, são apresentados os resultados encontrados pelas diversas configurações da Arquitetura AMAMv1.0, sendo que, neste caso, o número que antecede o nome do Agente Metaheurística aplicado na configuração representa a quantidade de Agentes deste tipo que foi aplicado, ou seja, na coluna $1AG$, foi aplicado somente 1 Agente Algoritmo Genético neste teste; para a coluna $1AG - 1ILS$ foi aplicado 1 Agente Algoritmo Genético em conjunto com 1 Agente *Iterated Local Search*. Este padrão é seguido por todas as tabelas de resultados apresentadas neste capítulo.

Cada coluna que representa uma configuração de teste da Arquitetura AMAM v1.0 ou os melhores resultados da literatura, se divide em duas outras colunas, sendo a primeira referente ao melhor valor encontrado para a função objetivo, em distância total percorrida, e a segunda, referente ao número de veículos obtidos para a solução

Tabela 6.3: Resultado para a aplicação, na Classe RC, individual dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG		1ILS		1VNS		Melhor	
RC101	1684,79	15	1748,70	15	1762,90	17	1696,94	14
RC102	1561,79	14	1513,00	14	1587,89	15	1554,75	12
RC103	1346,99	12	1340,99	12	1430,29	13	1261,67	11
RC104	1182,70	**10**	1226,20	11	1343,19	11	1135,48	10
RC105	1569,89	15	1626,69	14	1628,29	16	1629,44	13
RC106	1439,40	13	1414,50	13	1536,29	14	1424,73	11
RC107	1332,90	12	1253,80	12	1394,30	13	1230,48	11
RC108	1164,69	**10**	1180,00	11	1244,00	12	1139,82	10
RC201	1435,50	**4**	1455,09	**4**	1581,49	**4**	1406,91	4
RC202	1160,00	4	1198,40	4	1227,19	4	1365,65	3
RC203	1094,90	**3**	1011,29	4	1249,89	**3**	1049,62	3
RC204	804,50	**3**	882,80	**3**	858,89	**3**	798,41	3
RC205	1324,80	**4**	1376,39	**4**	1641,29	**4**	1297,19	4
RC206	1176,20	**3**	1131,79	4	1324,49	**3**	1146,32	3
RC207	1143,40	**3**	1055,10	4	1078,59	4	1061,14	3
RC208	850,39	**3**	848,29	**3**	978,59	**3**	828,14	3

em questão.

Para melhor visualização e entendimento, em todas as Tabelas foi adotada o seguinte procedimento: os valores encontrados pela Arquitetura AMAM v1.0 que superam os melhores resultados encontrados na literatura tiveram suas células preenchidas na cor cinza; os valores que se igualam aos melhores valores da literatura foram expressos em negrito; e os demais valores, que não superaram os melhores resultados encontrados na literatura, estão escritos com formatação padrão.

As Tabelas 6.4, 6.5 e 6.6 apresentam os resultados da aplicação em pares do mesmo agente, para os Agentes AG, ILS e VNS, em todas as instâncias teste.

As Tabelas 6.7, 6.8 e 6.9 apresentam os resultados da aplicação de combinações em pares combinados, para os Agentes AG, ILS e VNS em todas as instâncias teste.

As Tabelas 6.10, 6.11 e 6.12 apresentam os resultados da aplicação dos três agentes (AG, ILS e VNS) de uma só vez, e de 2 Agentes AG com 1 Agente ILS.

A porcentagem de resultados alcançados e/ou superados pelas configurações da Arquitetura AMAMv1.0, baseados nos melhores resultados encontrados na literatura, é apresentada na Tabela 6.13. Nesta tabela, na primeira coluna é apresentada a configuração do experimento realizado; na segunda, terceira e quarta colunas são exibidos, em porcentagem, a quantidade de resultados alcançados e/ou superados pelas configurações da Arquitetura AMAMv1.0, listadas na primeira coluna, para as classes de problemas C, R e RC, respectivamente. A última coluna da tabela, apresenta, também em termos de porcentagem, a quantidade total de bons resultados (alcançados e/ou superados) das diversas configurações apresentadas da Arquitetura AMAMv1.0.

Pode-se observar na Tabela 6.13 que as configurações com mais de um Agente Metaheurística superaram os resultados para as configurações dos Agentes acionados

Tabela 6.4: Resultado para a aplicação, na Classe C, de dois agentes AG, dois ILS e dois VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	2AG		2ILS		2VNS		Melhor	
C101	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C102	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C103	826,29	**10**	826,29	**10**	950,39	**10**	828,06	10
C104	828,39	**10**	829,00	**10**	940,90	**10**	824,78	10
C105	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C106	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C107	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C108	827,29	**10**	827,29	**10**	854,29	**10**	828,94	10
C109	827,29	**10**	827,29	**10**	854,29	**10**	828,94	10
C201	589,09	**3**	589,09	**3**	589,09	**3**	591,56	3
C202	589,09	**3**	589,09	**3**	589,09	**3**	591,56	3
C203	588,69	**3**	588,69	**3**	588,69	**3**	591,17	3
C204	588,09	**3**	588,09	**3**	597,19	**3**	590,60	3
C205	586,39	**3**	586,39	**3**	586,39	**3**	588,88	3
C206	585,89	**3**	585,89	**3**	586,00	**3**	588,49	3
C207	585,79	**3**	585,79	**3**	585,80	**3**	588,29	3
C208	585,79	**3**	585,79	**3**	585,79	**3**	588,32	3

individualmente, comprovando-se, com isso, que a cooperação entre os agentes metaheurísticas presentes na Arquitetura pode ter um impacto significativo no resultado global dos métodos aplicados.

Porém, como ilustrado nesta mesma tabela, a qualidade dos resultados está condicionada à qualidade individual dos Agentes participantes do processo de busca. Este fato pode ser observado, em particular, no tocante à aplicação do Agente *Variable Neighborhood Search* que, isoladamente, não produziu bons resultados e, quando acionado em conjunto com os demais, também não proporcionou bons resultados no modelo cooperativo. No entanto, mesmo neste caso os resultados em cooperação superam os resultados individuais para o VNS.

A configuração envolvendo os três agentes implementados (AG, ILS e VNS) atuando em conjunto obteve a melhor porcentagem de resultados alcançados e/ou superados. Esta configuração superou, em muito, as demais configurações, e demonstra que a cooperação entre diferentes técnicas de busca (cooperação híbrida) pode proporcionar resultados melhores que as configurações simples (configurações pouco diversificadas, ou com agentes iguais) entre os Agentes da Arquitetura.

A Tabela 6.14 apresenta os valores em termos de soma total dos resultados. A primeira coluna desta tabela apresenta as configurações da Arquitetura AMAMv1.0 apresentadas, sendo as duas últimas linhas expressando os melhores resultados. Na última linha (M. Literatura) são apresentados, em Total de Distância percorridas e Total de Veículos utilizados, os melhores resultados da literatura.

Para esta análise, foram considerados todos os resultados encontrados e estes somados, representando a quantidade total de distância e de veículos utilizados para cada configuração de teste. Através destes dados, pode-se observar que a Arquitetura

Tabela 6.5: Resultado para a aplicação, na Classe R, de dois agentes AG, dois ILS e dois VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	2AG		2ILS		2VNS		Melhor	
R101	1631,49	20	1667,59	**19**	1783,90	**19**	1645,79	19
R102	1515,39	**17**	1505,30	**17**	1579,90	**17**	1486,12	17
R103	1268,39	14	1220,20	14	1271,49	14	1292,68	13
R104	1002,10	10	988,00	10	1080,60	11	1007,24	9
R105	1400,00	**14**	1391,10	15	1422,39	15	1377,11	14
R106	1276.59	13	1256,10	13	1291,90	13	1251,98	12
R107	1104,89	11	1073,60	11	1126,10	11	1104,66	10
R108	993,49	10	945,89	10	1037,59	10	960,88	9
R109	1191,10	12	1162,39	12	1205,09	13	1194,73	11
R110	1111,29	11	1099,50	11	1267,50	12	1118,59	10
R111	1121,80	11	1070,90	11	1110,89	12	1096,72	10
R112	989,63	10	967,89	**9**	1017,20	11	982,14	9
R201	1251,29	**4**	1261,90	**4**	1299,00	**4**	1252,37	4
R202	1098,30	4	1116,10	4	1131,49	4	1191,70	3
R203	968,60	**3**	1061,09	**3**	1017,00	**3**	939,50	3
R204	744,59	3	762,19	3	766,29	3	825,52	2
R205	1044,30	**3**	1024,59	4	1167,59	**3**	994,42	3
R206	934,10	**3**	946,70	**3**	1021,49	**3**	906,14	3
R207	809,40	3	869,00	3	874,60	3	890,61	2
R208	738,89	**2**	739,30	3	755,10	**2**	726,75	2
R209	926,80	**3**	895,70	4	986,50	**3**	909,16	3
R210	972,89	**3**	952,30	4	1001,59	**3**	939,34	3
R211	777,90	3	805,00	3	838,50	3	885,71	2

AMAM superou em algumas configurações os melhores resultados encontrados na literatura em termos de distância total percorrida. Porém, em contrapartida, em nenhuma configuração a Arquitetura AMAMv1.0 conseguiu superar os melhores resultados com relação à soma total em número de veículos utilizados.

As Figuras 6.13, 6.14, 6.15, 6.16 e 6.17 mostram o comportamento da Arquitetura AMAMv1.0 nas configurações: $1AG$, $1ILS$, $1VNS$, $2AG$, $2ILS$ e $2VNS$. Nestas figuras, o melhor valor da função objetivo (eixo y) é plotado em termos do número de iterações (eixo x). Cada configuração teste é identificada em legenda.

A Figura 6.13 mostra o comportamento da Arquitetura AMAMv1.0 nas configurações: $1AG$, definido na legenda somente como AG e $2AG$, definido na legenda como $AGAG$. Esta figura mostra que a evolução das soluções para 2 Agentes AG se comporta melhor que a evolução das soluções para somente 1 Agente AG, ou seja, obtém melhores resultados em um número menor de iterações, conseguindo convergir para um ótimo mais rapidamente.

A Figura 6.14 mostra o comportamento da Arquitetura AMAMv1.0 nas configurações: $1ILS$, definido na legenda como ILS; e $2ILS$, definido na legenda como $ILSILS$. Os resultados obtidos são quase idênticos nas primeiras iterações para ambas as configurações (ILS e ILSILS), mas a partir da quinta iteração a configuração

Tabela 6.6: Resultado para a aplicação, na Classe RC, de dois agentes AG, dois ILS e dois VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	2AG		2ILS		2VNS		Melhor	
RC101	1501,70	15	1668,90	15	1766,70	16	1696,94	14
RC102	1561,79	14	1530,49	13	1559,90	15	1554,75	12
RC103	1342,80	12	1339,60	12	1362,69	12	1261,67	11
RC104	1190,90	**10**	1163,59	11	1275,70	11	1135,48	10
RC105	1551,00	15	1576,20	14	1639,49	15	1629,44	13
RC106	1436,19	13	1414,90	13	1446,89	13	1424,73	11
RC107	1285,30	12	1268,29	12	1332,89	13	1230,48	11
RC108	1147,59	**10**	1157,29	11	1217,20	12	1139,82	10
RC201	1430,40	**4**	1375,60	5	1525,29	**4**	1406,91	4
RC202	1157,39	4	1178,40	4	1218,49	4	1365,65	3
RC203	1085,49	**3**	1107,60	**3**	1167,39	**3**	1049,62	3
RC204	795,50	**3**	844,09	**3**	851,79	**3**	798,41	3
RC205	1312,30	**4**	1272,30	**4**	1384,49	**4**	1297,19	4
RC206	1160,90	**3**	1130,49	4	1156,90	4	1146,32	3
RC207	1081,79	**3**	1124,00	4	1049,79	4	1061,14	3
RC208	826,60	**3**	848,00	**3**	821,29	5	828,14	3

Tabela 6.7: Resultado para a aplicação, na Classe C, da combinação dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS		1ILS-1VNS		1AG-1VNS		Melhor	
C101	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C102	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C103	826,29	**10**	868,29	**10**	854,49	**10**	828,06	10
C104	842,69	**10**	845,29	**10**	842,69	**10**	824,78	10
C105	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C106	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C107	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C108	827,29	**10**	827,29	**10**	827,29	**10**	828,94	10
C109	868,30	**10**	827,29	**10**	827,29	**10**	828,94	10
C201	589,09	**3**	589,09	**3**	589,09	**3**	591,56	3
C202	589,09	**3**	589,10	**3**	589,10	**3**	591,56	3
C203	588,69	**3**	588,69	**3**	588,69	**3**	591,17	3
C204	588,09	**3**	606,79	**3**	600,10	**3**	590,60	3
C205	586,39	**3**	586,39	**3**	586,39	**3**	588,88	3
C206	585,89	**3**	585,90	**3**	585,90	**3**	588,49	3
C207	585,79	**3**	585,79	**3**	585,79	**3**	588,29	3
C208	585,79	**3**	585,79	**3**	585,79	**3**	588,32	3

com 2 Agentes ILS começa a apresentar um certo ganho com relação à configuração com apenas 1 Agente ILS, gerando bons resultados em um menor intervalo de tempo.

Tabela 6.8: Resultado para a aplicação, na Classe R, da combinação dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS		1ILS-1VNS		1AG-1VNS		Melhor	
R101	1670,09	**19**	1650,89	20	1645,39	20	1645,79	19
R102	1491,10	18	1478,50	18	1480,00	18	1486,12	17
R103	1225,90	14	1255,89	14	1225,90	14	1292,68	13
R104	1024,79	11	1033,20	10	1030,29	11	1007,24	9
R105	1374,00	15	1399,00	14	1375,10	15	1377,11	14
R106	1280,19	13	1251,30	13	1278,30	13	1251,98	12
R107	1107,30	11	1089,20	11	1100,19	11	1104,66	10
R108	995,20	10	971,09	10	988,50	10	960,88	9
R109	1210,50	13	1160,10	12	1201,49	13	1194,73	11
R110	1107,80	12	1115,90	12	1117,00	12	1118,59	10
R111	1089,30	11	1110,50	11	1090,30	11	1096,72	10
R112	991,20	10	979,00	11	993,29	10	982,14	9
R201	1250,79	**4**	1310,19	**4**	1273,20	**4**	1252,37	4
R202	1107,29	4	1134,79	4	1110,29	4	1191,70	3
R203	996,49	**3**	960,80	**3**	995,10	**3**	939,50	3
R204	787,60	3	788,70	3	788,63	3	825,52	2
R205	1071,20	**3**	1030,50	**3**	1073,19	**3**	994,42	3
R206	955,59	**3**	935,80	**3**	935,80	**3**	906,14	3
R207	815,19	3	884,90	3	836,99	3	890,61	2
R208	772,59	**2**	749,10	**2**	763,29	**2**	726,75	2
R209	909,10	**3**	948,10	**3**	929,29	**3**	909,16	3
R210	959,19	**3**	1046,40	**3**	1043,10	**3**	939,34	3
R211	785,00	3	809,20	3	800,39	3	885,71	2

As configurações da Arquitetura AMAMv1.0 com 1 e 2 Agentes VNS seguem o mesmo padrão de comportamento das demais configurações apresentadas. A Figura 6.15 mostra a evolução das soluções para estas configurações da Arquitetura, definidas na legenda como VNS para a aplicação se somente 1 Agente VNS, e VNSVNS para a aplicação de 2 Agentes VNS. Como ilustrado nesta figura, a configuração de 2 Agentes VNS inicia com um resultado pior que a configuração com somente 1 Agente VNS, mas em poucas iterações este quadro se inverte e a configuração de 2 Agentes apresenta um certo ganho com relação às soluções encontradas pela configuração com somente 1 Agente VNS.

As Figuras 6.16 e 6.17 apresentam o comportamento da Arquitetura AMAMv1.0 para as configurações com 1 e 2 Agentes, respectivamente. Nestas Figuras os resultados foram plotados em conjunto para melhor visualização e análise do comportamento das metaheurísticas aplicadas.

Na Figura 6.16 pode-se observar que a configuração com 1 Agente ILS gera resultados piores que as configurações com 1 Agente AG até um determinado número de iterações e a partir deste ponto consegue uma convergência mais rápida, ou seja, a metaheurística ILS melhora seus resultados gradativamente, e a metaheurística AG não tem um padrão de comportamento e com isso, a metaheurística ILS tende

Tabela 6.9: Resultado para a aplicação, na Classe RC, da combinação dos agentes AG, ILS e VNS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS		1ILS-1VNS		1AG-1VNS		Melhor	
RC101	1671,19	15	1645,10	15	1666,20	16	1696,94	14
RC102	1525,09	14	1498,40	14	1530,00	14	1554,75	12
RC103	1369,90	12	1340,39	12	1362,29	12	1261,67	11
RC104	1217,50	11	1184,90	11	1220,50	10	1135,48	10
RC105	1555,39	14	1571,90	15	1560,10	15	1629,44	13
RC106	1402,39	12	1446,09	13	1445,10	13	1424,73	11
RC107	1249,69	**11**	1269,29	12	1250,29	12	1230,48	11
RC108	1154,90	**10**	1165,19	11	1163,50	11	1139,82	10
RC201	1442,29	**4**	1525,79	**4**	1515,10	**4**	1406,91	4
RC202	1158,09	4	1261,30	4	1179,69	4	1365,65	3
RC203	1044,79	**3**	976,10	4	975,39	4	1049,62	3
RC204	798,50	**3**	851,19	**3**	801,09	**3**	798,41	3
RC205	1341,10	**4**	1355,90	**4**	1345,20	**4**	1297,19	4
RC206	1118,90	**3**	1118,30	4	1119,19	4	1146,32	3
RC207	1075,30	**3**	996,19	4	1070,10	**3**	1061,14	3
RC208	845,40	**3**	864,10	**3**	863,29	**3**	828,14	3

Tabela 6.10: Resultado para a aplicação, na Classe C, simultânea dos agentes AG, ILS e VNS e de dois agentes AG com um agente ILS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS-1VNS		2AG-1ILS		Melhor	
C101	827,29	**10**	827,29	**10**	828,94	10
C102	827,29	**10**	827,29	**10**	828,94	10
C103	826,29	**10**	885,39	**10**	828,06	10
C104	830,00	**10**	865,69	**10**	824,78	10
C105	827,29	**10**	827,29	**10**	828,94	10
C106	827,29	**10**	827,29	**10**	828,94	10
C107	827,29	**10**	827,29	**10**	828,94	10
C108	827,29	**10**	827,29	**10**	828,94	10
C109	827,29	**10**	853,29	**10**	828,94	10
C201	589,09	**3**	589,09	**3**	591,56	3
C202	589,09	**3**	589,09	**3**	591,56	3
C203	588,69	**3**	588,69	**3**	591,17	3
C204	588,09	**3**	589,49	**3**	590,60	3
C205	586,39	**3**	586,39	**3**	588,88	3
C206	585,89	**3**	585,89	**3**	588,49	3
C207	585,79	**3**	585,79	**3**	588,29	3
C208	585,79	**3**	585,79	**3**	588,32	3

a convergir mais rapidamente. Estas mesmas observações são válidas para os testes com 2 Agentes do mesmo tipo, apresentados na Figura 6.17.

Tabela 6.11: Resultado para a aplicação, na Classe R, simultânea dos agentes AG, ILS e VNS e de dois agentes AG com um agente ILS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS-1VNS		2AG-1ILS		Melhor	
R101	1646,40	**19**	1662,30	**19**	1645,79	19
R102	1476,50	**17**	1496,20	**17**	1486,12	17
R103	1236,50	**13**	1292,40	14	1292,68	13
R104	1003,30	10	1032,49	10	1007,24	9
R105	1374,80	15	1374,29	15	1377,11	14
R106	1244,30	**12**	1271,90	13	1251,98	12
R107	1080,40	11	1091,90	11	1104,66	10
R108	948,00	**9**	964,20	10	960,88	9
R109	1082,30	12	1172,80	12	1194,73	11
R110	1096,00	11	1103,00	12	1118,59	10
R111	1087,60	11	1087,90	11	1096,72	10
R112	971,90	10	1034,60	10	982,14	9
R201	1196,40	5	1249,29	**4**	1252,37	4
R202	1061,80	4	1141,79	4	1191,70	3
R203	945,69	**3**	954,20	**3**	939,50	3
R204	761,59	3	747,59	3	825,52	2
R205	955,40	4	1063,69	**3**	994,42	3
R206	933,69	**3**	946,69	**3**	906,14	3
R207	820,49	3	857,80	3	890,61	2
R208	719,40	**2**	752,10	**2**	726,75	2
R209	871,00	**3**	939,49	**3**	909,16	3
R210	953,30	**3**	964,29	**3**	939,34	3
R211	804,10	3	819,50	3	885,71	2

Este fato também é observado no tempo de execução dos algoritmos, que apesar de não ser discutido neste trabalho, as configurações com Agentes ILS convergiram mais rapidamente, ou seja, gastaram menor tempo. Os testes com Agentes VNS foram piores que os demais Agentes em todos os casos, mas mesmo assim, conseguiu gerar bons resultados.

A Tabela 6.15 apresenta os resultados da cooperação, em quantidade de vezes que a cooperação ocorreu, entre os Agentes de todas as configurações da Arquitetura AMAMv1.0, para as Instâncias C101, R101 e RC101.

Para todos os resultados, a primeira coluna identifica a quantidade de vezes que o Agente em questão enviou para o Repositório de Soluções melhores soluções do que todas as existentes no Repositório até o momento. A segunda coluna identifica a quantidade de vezes que o Agente em questão utilizou soluções de outros agentes que estão executando na mesma configuração da Arquitetura, não sendo computadas as vezes em que este Agente recebeu do Repositório de Soluções, soluções enviadas por ele mesmo. Nesta tabela não são exibidas as configurações com os Agentes individuais (1AG, 1ILS e 1VNS) porque nestes casos não existe um modelo efetivo de cooperação entre agentes, sendo que eles enviam e recebem suas próprias soluções.

Tabela 6.12: Resultado para a aplicação, na Classe RC, simultânea dos agentes AG, ILS e VNS e de dois agentes AG com um agente ILS na arquitetura AMAMv1.0 para a solução do PRVJT.

Instância	1AG-1ILS-1VNS		2AG-1ILS		Melhor	
RC101	1684,50	15	1663,70	16	1696,94	14
RC102	1494,60	14	1524,10	13	1554,75	12
RC103	1230,00	**11**	1327,49	12	1261,67	11
RC104	1074,00	11	1173,80	**10**	1135,48	10
RC105	1581,39	15	1568,10	15	1629,44	13
RC106	1382,60	13	1394,69	12	1424,73	11
RC107	1260,90	**11**	1289,19	12	1230,48	11
RC108	1154,70	11	1162,89	11	1139,82	10
RC201	1465,09	**4**	1513,49	**4**	1406,91	4
RC202	1181,10	4	1310,79	4	1365,65	3
RC203	910,30	4	1092,90	**3**	1049,62	3
RC204	795,50	**3**	815,40	**3**	798,41	3
RC205	1240,29	5	1361,29	**4**	1297,19	4
RC206	1081,60	4	1096,10	**3**	1146,32	3
RC207	1008,90	4	1072,00	**3**	1061,14	3
RC208	782,80	4	842,70	**3**	828,14	3

Tabela 6.13: Porcentagem de Melhores soluções encontradas para cada classe de problemas

Experimento	C (%)	R (%)	RC (%)	TOTAL (%)
1AG	94,12	43,48	18,75	51,79
1ILS	94,12	47,83	43,75	60,71
1VNS	47,06	21,74	12,50	26,79
2AG	94,12	43,48	31,25	55,36
2ILS	94,12	56,52	50,00	66,07
2VNS	70,59	21,74	18,75	35,71
1AG-1ILS	88,24	43,48	43,75	57,14
1ILS-1VNS	82,35	47,83	43,75	57,14
1AG-1VNS	82,35	47,83	37,50	55,36
1AG-1ILS-1VNS	94,12	82,61	81,25	85,71
2AG-1ILS	82,35	47,83	37,50	55,36

Para facilitar o entendimento, ainda na Tabela 6.15, os valores referentes aos resultados da aplicação de 2 Agentes Algoritmo Genético para a instância teste $C101$ receberam destaque. Esta coluna (2AG), dividi-se em duas outras (AG1 e AG2) referentes aos dois Agentes Algoritmo Genético presentes nesta configuração da Arquitetura. Por sua vez, as colunas AG1 e AG2 dividem-se em outras duas, cada uma.

Estes resultados não foram analisados no sentido de justificar os valores encontrados, e sim com o propósito de avaliar e validar a cooperação entre os agentes pre-

Tabela 6.14: Somatório das Distâncias e Total de veículos para todos os experimentos, comparados com os melhores encontrados na literatura

Experimento	Total Distância	Total Veículos
1AG	57397,02	430
1ILS	57307,67	434
1VNS	60626,57	452
2AG	55608,80	429
2ILS	56927,21	435
2VNS	59274,95	444
1AG-1ILS-1VNS	55745,27	433
1AG-1ILS	57147,66	431
1AG-1VNS	57341,11	438
1ILS-1VNS	57385,33	437
2AG-1ILS	57497,37	430
M. Literatura	57180,84	405

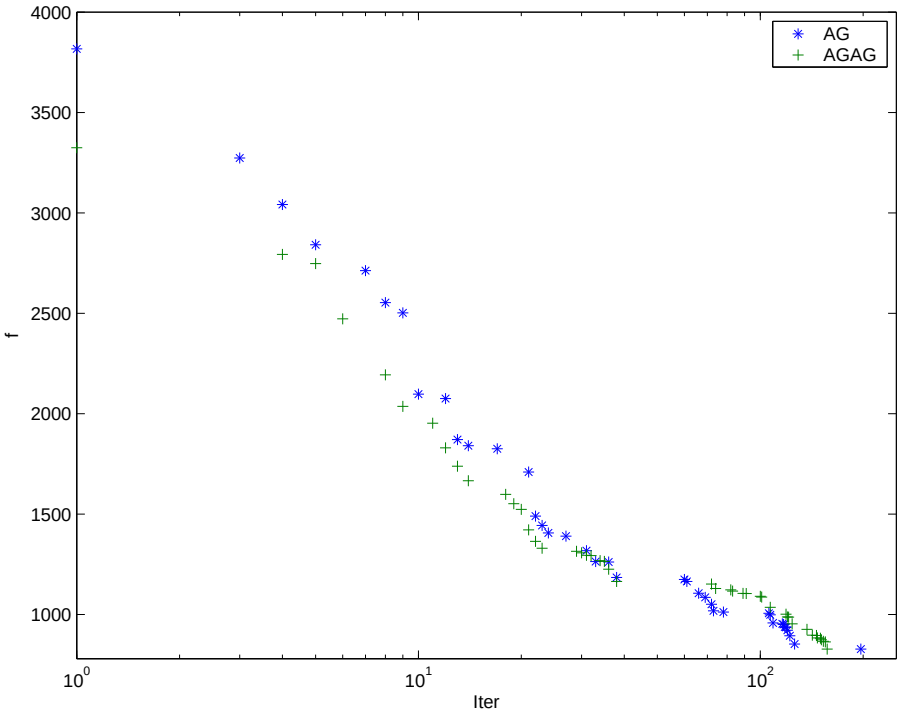

Figura 6.13: Evolução das Soluções de 1 Agente AG e de 2 Agentes AG (AGAG) para a instância C101

sentes em cada uma das configurações da Arquitetura AMAMv1.0. Os valores em destaque representam a cooperação, em quantidade de vezes, entre os dois Agentes AG presentes na configuração em questão. A primeira célula do Agente AG1, que contém o valor 60, é referente à quantidade de vezes que este Agente enviou para o Repositório de Soluções uma solução e esta foi melhor que todas as soluções já existentes no Repositório até o momento de seu envio. A segunda célula do Agente AG1, que contém o valor 167, é referente à quantidade de vezes que este Agente

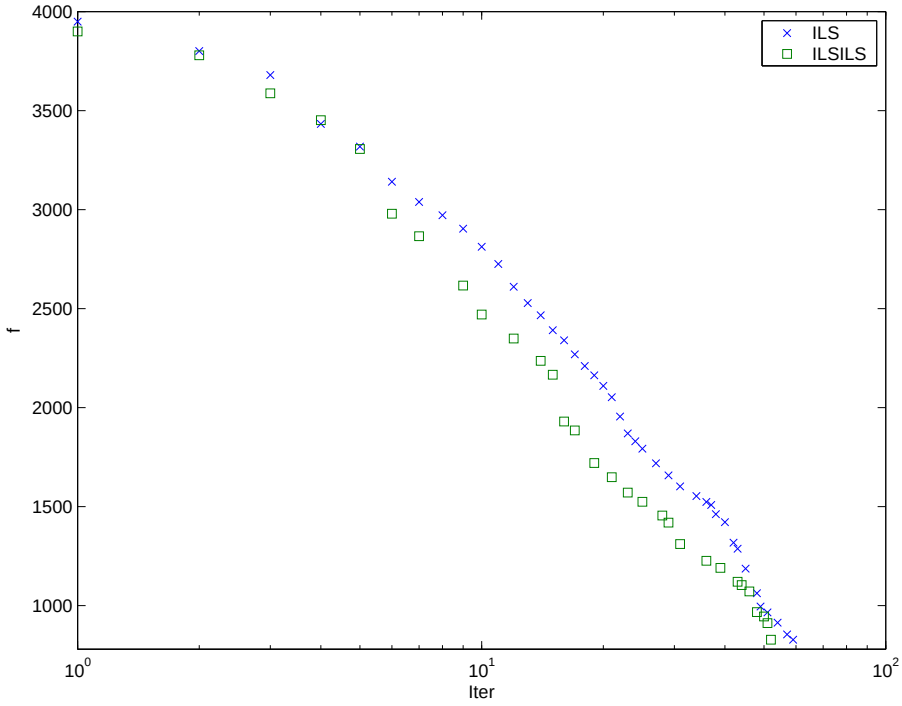

Figura 6.14: Evolução das Soluções de 1 Agente ILS e de 2 Agentes ILS (ILSILS) para a instância C101

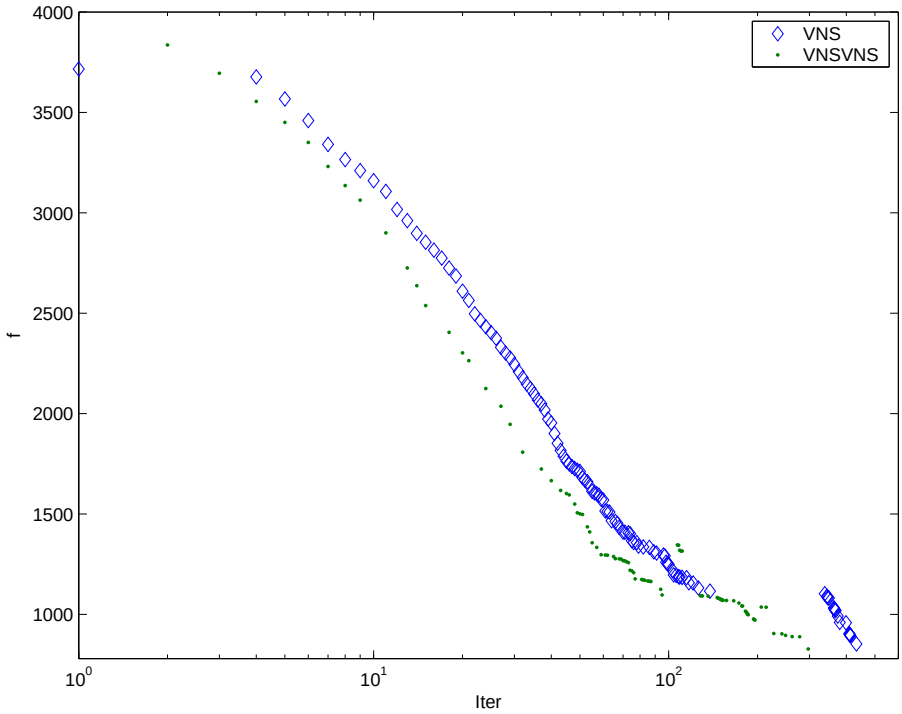

Figura 6.15: Evolução das Soluções de 1 Agente VNS e de 2 Agentes VNS (VNSVNS) para a instância C101

utilizou solução compartilhadas, através do Repositório de Soluções, pelo Agente AG2. Este processo é similar para todas as configuração ilustradas na Tabela 6.15.

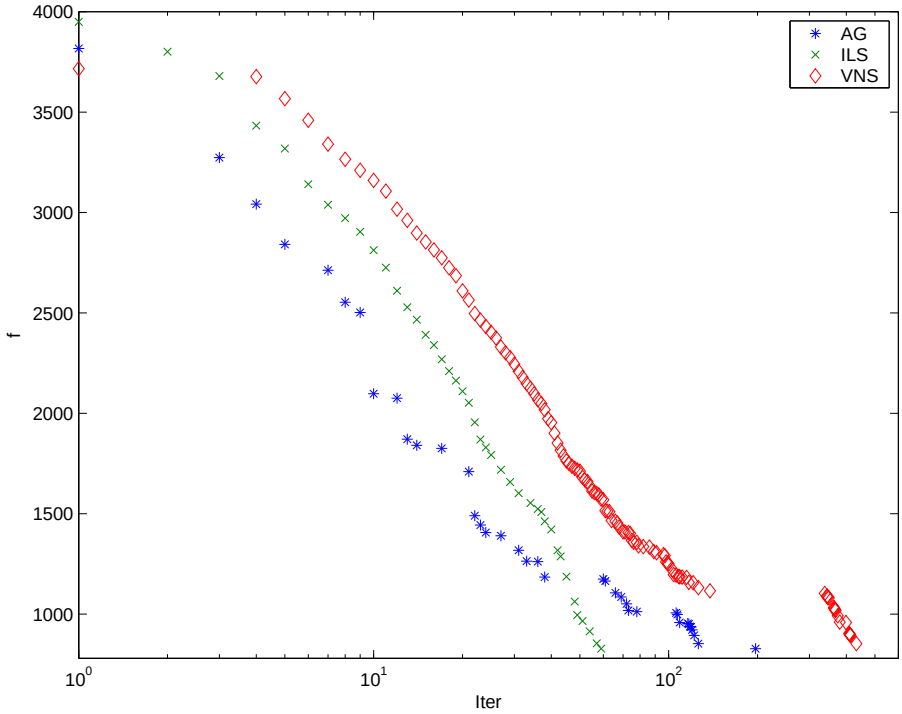

Figura 6.16: Evolução das Soluções de 1 Agente AG, 1 Agente ILS e de 1 Agente VNS para a instância C101

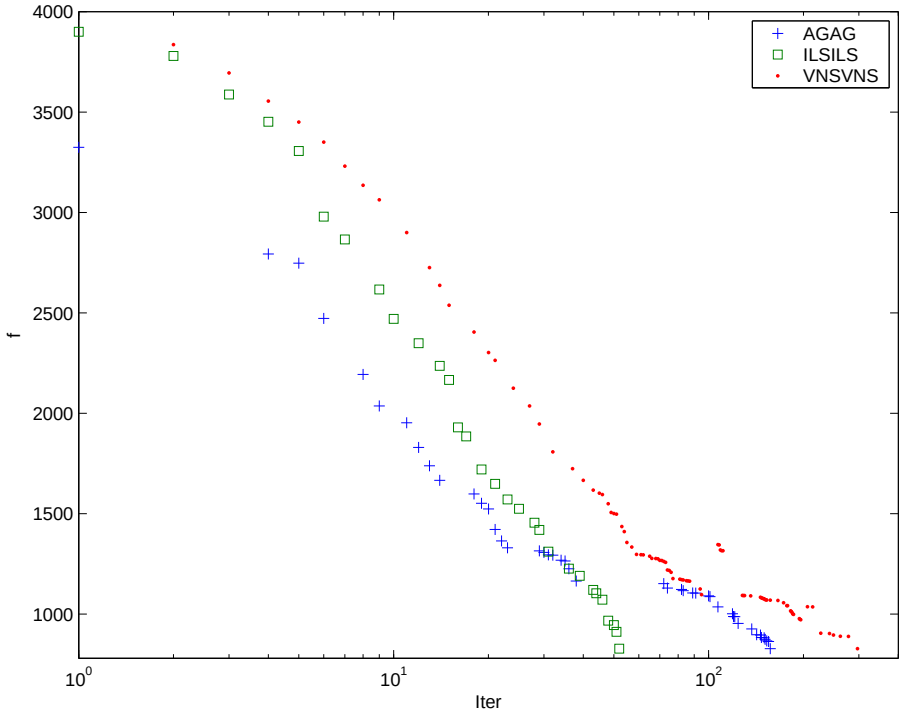

Figura 6.17: Evolução das Soluções de 2 Agente AG, 2 Agente ILS e de 2 Agente VNS para a instância C101

6.5 Considerações Finais

Embora a Arquitetura AMAMv1.0 tenha o propósito de ser uma arquitetura flexível, isso irá depender da forma com que os Agentes Metaheurísticos foram implementa-

Tabela 6.15: Resultados da Cooperação entre os Agentes para as Instâncias C101, R101 e RC101

INST.	2AG				2ILS				2VNS			
	AG1		AG2		ILS1		ILS2		VNS1		VNS2	
C101	60	167	76	297	32	134	39	162	156	357	197	439
R101	50	247	63	116	60	372	18	383	87	421	28	545
RC101	77	1039	85	75	32	340	57	337	18	457	102	538
INST.	1AG-1ILS				1AG-1VNS				1ILS-1VNS			
	AG		ILS		AG		VNS		ILS		VNS	
C101	35	21	51	247	64	253	115	154	6	634	97	118
R101	0	56	65	59	27	123	88	215	32	52	73	233
RC101	2	168	101	109	12	196	97	253	31	62	64	198
INST.	1AG-1ILS-1VNS						2AG-1ILS					
	AG		ILS		VNS		AG1		AG2		ILS	
C101	36	23	0	1217	62	651	0	220	41	24	55	289
R101	2	333	41	859	69	916	6	429	0	184	69	119
RC101	23	343	28	876	89	494	207	35	41	137	75	56

dos, ou seja, se eles estão ou não presos a algum problema em particular, e ainda, se possuem adaptabilidade suficiente para usufruir de todas as características do Ambiente da Arquitetura.

Neste sentido, ao projetar Agentes Metaheurísticos para a Arquitetura AMAM v1.0, deve-se observar e utilizar as características oferecidas pelo Ambiente desta Arquitetura, conseguindo, assim, manter o propósito de flexibilidade da mesma.

Apesar destes conceitos de flexibilidade, ou seja, a Arquitetura AMAMv1.0 não é específica para um determinando problema de otimização combinatória, especializações da arquitetura foram desenvolvidas para a solução do Problema de Roteamento de Veículos com Janelas de Tempo, com o objetivo de demonstrar o correto funcionamento da arquitetura e, também, para demonstrar suas características de escalabilidade e encapsulamento. Estes experimentos realizados, e com razoável sucesso, com as onze especializações desenvolvidas ($1AG$, $1ILS$, $1VNS$, $2AG$, $2ILS$, $2VNS$, $1AG-1ILS$, $1AG-1VNS$, $1ILS-1VNS$, $1AG-1ILS-1VNS$ e $2AG-1ILS$) da arquitetura, demonstram, que todas estas características estão presentes e como podem ser utilizadas.

A escalabilidade, que corresponde ao aumento do desempenho a partir da adição de novos agentes, foi confirmada, considerando a melhora efetiva nos valores das funções de avaliação, obtida nas especializações com dois, e principalmente, com três agentes metaheurísticas trabalhando em conjunto, em relação às especializações com apenas um agente.

Capítulo 7

Conclusões e Trabalhos Futuros

Neste capítulo, são apresentadas as conclusões relativas ao desenvolvimento e aplicações da Arquitetura MultiAgentes Cooperativos para Metaheurísticas, denominada AMAMv1.0, e são apresentadas também as perspectivas de trabalhos futuros utilizando os conceitos desta mesma Arquitetura.

7.1 Conclusões

Esta dissertação apresenta a Arquitetura MultiAgentes Cooperativos para Metaheurísticas (AMAM), em sua versão 1.0 (AMAMv1.0). Assim, o modelo conceitual da arquitetura AMAM nesta versão é apresentado, bem como os detalhes de seus componentes, além das principais modificações realizadas para a implementação de uma arquitetura cooperativa, nos termos propostos para a versão 1.0 e que a diferenciam da versão inicial apresentada por Silva (2007). Neste sentido, uma revisão de literatura a respeito de Sistemas Multiagentes e de Metaheurísticas Cooperativas é apresentada, afim de destacar as principais características destes sistemas e expor a correlação com a arquitetura proposta.

A arquitetura AMAMv1.0 destaca-se pela flexibilidade, permitindo, através de seu ambiente e das interações dos agentes com este ambiente, uma mesma estrutura na solução de diferentes problemas de otimização, isto é, possibilita que uma metaheurística implementada seja utilizada na solução de diferentes problemas. Da mesma forma, a arquitetura AMAMv1.0 foi modelada para ser flexível o bastante para solucionar diferentes problemas de otimização, sem grande impacto no restante da arquitetura.

Cada metaheurística é implementada como um agente da arquitetura. Estes agentes metaheurísticos trabalham de forma autônoma e assíncrona, possibilitando maior facilidade na inserção e remoção dos mesmos, sem causar grandes impactos no que diz respeito ao custo computacional para sua implementação. Esta facilidade de adicionar e remover agentes, unida ao trabalho assíncrono dos mesmos e à flexibilidade da estrutura cooperativa, centralizada pelo Agente Coordenador, permite que a arquitetura AMAMv1.0 se comporte de forma híbrida, podendo trabalhar com diversas metaheurísticas de diversas proposições ao mesmo tempo.

A característica central da estrutura proposta é a cooperação entre os Agentes Metaheurísticas. Esta cooperação é assegurada pela estrutura de Repositório de

Soluções utilizada, que permite a cada Agente Metaheurística comunicar aos demais suas soluções encontradas e recolher ou obter soluções encontradas pelos demais Agentes Metaheurísticas para sua própria utilização. Desse modo, cada agente influencia nos trabalhos dos demais e é por eles influenciado. Esta ação de cooperação é evidenciada pelas Tabelas 6.13 e 6.15, para o caso do exemplo de solução aqui tratado.

A validação da Arquitetura AMAMv1.0 proposta é feita através da solução das instâncias de Solomon do Problema de Roteamento de Veículos com Janela de Tempo. Os resultados encontrados, mostrados na Seção 6.4, tornam claro que:

- há uma forte correlação entre a qualidade do Agente Metaheurística implementado e os resultados alcançados com o uso do mecanismo de cooperação entre Agentes. Ou seja, caso o Agente, individualmente considerado, não encontre bons resultados na solução das instâncias do problema objeto de interesse, há dificuldades para a obtenção de resultados de boa qualidade na estrutura cooperativa. No entanto, observa-se que, neste caso, o mecanismo de cooperação entre Agentes leva a uma melhoria dos resultados alcançados.

- o mecanismo de cooperação não se prende a classes de metaheurísticas, no sentido que permite que diferentes classes de metaheurísticas (populacional, de trajetória) possam cooperar e garantir melhorias nos resultados individuais alcançados. Isso pode ser visto na cooperação entre os Agentes Metaheurísticas AG e ILS.

- a adição de maior número de Agentes Metaheurísticas leva a uma melhoria significativa dos resultados individuais alcançados, em especial no caso em que diferentes classes de metaheurísticas convivem no mesmo ambiente de solução, como pode ser observado na configuração envolvendo os Agentes AG, ILS e VNS.

- não só a qualidade dos resultados é afetada pela adição de maior número de Agentes Metaheurísticas. Também a velocidade de obtenção de resultados é afetada, pois, conforme mostram as Figuras 6.13 a 6.17, a adição de Agentes leva a obtenção de melhores resultados em menor número de iterações dos Agentes Metaheurísticas.

No tocante às soluções das instâncias de Solomon do Problema de Roteamento de Veículos com Janela de Tempo, é importante, primeiramente, ressaltar que os resultados obtidos pela implementação realizada são fortemente competitivos com os melhores resultados encontrados na literatura para a solução destas instâncias, conforme apresentados em `http://www.sintef.no/Projectweb/TOP/Problems/`. Assim:

- para as instâncias de classe C e a aplicação de apenas um Agente, foram encontradas as melhores soluções em 16 das 17 instâncias, para os Agente AG e ILS, e em 8 instâncias, para o Agente VNS.

- para as mesmas instâncias e a aplicação de dois Agentes de cada tipo implementado, os mesmos resultados foram encontrados com relação aos Agentes

AG e ILS. No caso do Agente VNS, foram obtidas as melhores soluções em 12 instâncias. Deve-se ressaltar que, neste caso, todos os melhores resultados quanto ao número mínimo de veículos foram alcançados.

- para as instâncias de classe R e a aplicação de apenas um Agente, foram encontradas as melhores soluções em 10 das 23 instâncias para o Agentes AG; em 11 das 23 instâncias para o Agente ILS; e em 5 instâncias, para o Agente VNS.

- para as mesmas instâncias e a aplicação de dois Agentes de cada tipo implementado, foram encontradas as melhores soluções em 10 das 23 instâncias para o Agentes AG; em 13 das 23 instâncias para o Agente ILS; e em 5 instâncias, para o Agente VNS.

- para as instâncias de classe RC e a aplicação de apenas um Agente, foram encontradas as melhores soluções em 3 das 16 instâncias para o Agente AG; em 7 das 16 instâncias para o Agente ILS; e em 2 instâncias, para o Agente VNS.

- para as mesmas instâncias e a aplicação de dois Agentes de cada tipo implementado, foram encontradas as melhores soluções em 5 das 16 instâncias para o Agentes AG; em 8 das 16 instâncias para o Agente ILS; e em 3 instâncias, para o Agente VNS.

- para a aplicação simultânea de três Agentes (AG, ILS e VNS) para a solução de instâncias da classe RC, foram foram encontradas as melhores soluções em 13 das 16 instâncias; em outra aplicação de três Agentes (AG, AG e ILS), foram encontradas as melhores soluções em 6 das 16 instâncias.

- no que diz respeito à soma total dos resultados alcançados, mostrada na Tabela 6.14, deve-se ressaltar que os resultados apresentados são inferiores aos melhores da literatura tendo em vista as dificuldades do Agente VNS. Contudo, em 4 das 11 configurações apresentadas, os resultados são superiores aos melhores da literatura, no que diz respeito à distância total percorrida.

- Por fim, os resultados encontrados têm dificuldades apenas na classe de instâncias RC.

Deve-se destacar, além disso, que o ponto principal destas implementações para a solução de instâncias do Problema de Roteamento de Veículos com Janela de Tempo é a adaptação do algoritmo *PFIH*.

7.2 Trabalhos Futuros

Esta etapa de desenvolvimento da Arquitetura AMAMv1.0 venceu diversos desafios e tornou evidente que a proposta apresentada é uma alternativa de interesse para a solução de problemas de otimização combinatória. Além disso, possibilitou a abertura de novas frentes de pesquisa para a continuidade desse desenvolvimento. Cabe destacar algumas destas questões:

- discussão a respeito da existência, ou não, do Agente Coordenador, de modo a efetivamente garantir autonomia aos Agentes Metaheurísticas;
- desacoplar as ações de movimento em um certo Agente Metaheurística, transferindo-as para um novo Agente;
- realizar testes computacionais com maior número de Agentes Metaheurísticas;
- realizar testes computacionais, para o caso do PRVJT, com instâncias de maior porte;
- diferenciar Agentes com características de autonomia e agentes com características de dependência, sem, no entanto, definir hierarquias de agentes;
- implementação na Arquitetura de outras metaheurísticas, além das já postas;
- implementação de novas estratégias de busca local e novas estratégias de construtores de soluções iniciais;
- outras implementações da estrutura Ambiente, para a resolução de diferentes problemas de otimização combinatória;
- desenvolvimento de novas técnicas de cooperação entre os Agentes;
- utilização de hiperheurísticas, para possibilitar, além de um ambiente cooperativo, um ambiente de heurísticas adaptativas ao problema tratado;
- estudar a implementação da Arquitetura em ambientes computacionais paralelos e em ambientes computacionais distribuídos.

7.3 Trabalhos Publicados

De todo trabalho de pesquisa espera-se bons resultados, e graças ao esforço e dedicação dos envolvidos na confecção deste trabalho estes resultados foram alcançados. Assim, o reconhecimento e aceitação deste trabalho foi confirmado com a publicação de artigos em importantes eventos da área, que seguem:

- artigo completo publicado nos Anais do IX Simpósio Brasileiro de Automação Inteligente 2009 - SBAI 2009; (Fernandes et al., 2009a);
- artigo completo publicado no 30 CILAMCE - Iberian-Latin-American Congress on Computational Methods in Engineering; (Fernandes et al., 2009b);
- artigo completo publicado no IEEE International Conference on Systems, Man, and Cybernetics 2009; (Fernandes et al., 2009d);
- artigo completo publicado no XLI Simpósio Brasileiro de Pesquisa Operacional 2009; (Fernandes et al., 2009c);

Apêndice A

Análise de eficácia da Arquitetura AMAMv1.0

A.1 Aplicação de 2 Agentes ILS

No experimento prático da Arquitetura AMAMv1.0 envolvendo 2 Agentes ILS, participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente ILS1 e (8) Agente ILS2. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.1.

Aqui a cooperação existente entre os dois Agentes ILS (ILS1 e ILS2) acontece da mesma forma que em todos os outros experimentos práticos da Arquitetura, por troca de soluções, através do Agente Coordenador. Quando os agentes ILS necessitam de soluções iniciais ele enviam estímulos para o Agente Coordenador e este por sua vez retorna com estas soluções, retiradas do Repositório de Soluções. Estas soluções constituem as soluções iniciais dos Agentes ILS.

Os Agentes ILS trabalham com estas soluções iniciais e ao término de cada iteração, em seu processo de busca, os Agentes ILS, enviam suas soluções atuais para o Agente Coordenador, e este decidirá por enviá-las ou não para o repositório de Soluções. A cada nova iteração, cada um dos Agentes ILS solicita ao Agente Analisador de Soluções uma comparação entre as soluções atuais desta nova iteração com novas soluções obtidas do Repositório de Soluções, através do Agente Coordenador. Neste instante os Agentes ILS podem utilizar de suas próprias soluções ou utilizar da solução que lhes foi enviadas do Repositório de Soluções, possibilitando com isso, que cada um dos Agentes envolvidos utilizem soluções compartilhadas uns pelos outros. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.4, 6.5 e 6.6 na Seção 6.4.

A.2 Aplicação de 2 Agentes VNS

No experimento prático da Arquitetura AMAMv1.0 envolvendo 2 Agentes VNS, participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente VNS1 e (8) Agente VNS2. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.2.

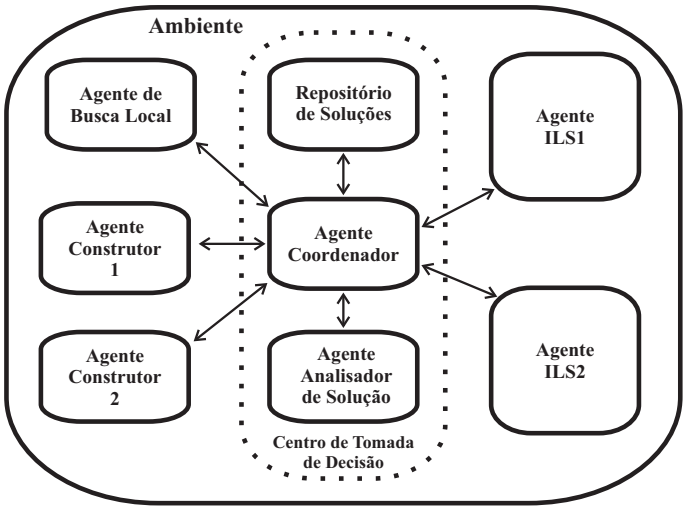

Figura A.1: Modelo Conceitual da Arquitetura AMAMv1.0 para 2 Agentes ILS

Aqui a cooperação existente entre os dois Agentes VNS (VNS1 e VNS2) acontece da mesma forma que em todos os outros experimentos práticos da Arquitetura, por troca de soluções, através do Agente Coordenador. Quando os agentes VNS necessitam de soluções iniciais eles enviam estímulos para o Agente Coordenador e este por sua vez retorna com estas soluções, retiradas do Repositório de Soluções. Estas soluções constituem as soluções iniciais dos Agentes VNS.

Os Agentes VNS trabalham com estas soluções iniciais e ao término de cada iteração, em seu processo de busca, os Agentes VNS, enviam suas soluções atuais para o Agente Coordenador, e este decidirá por enviá-las ou não para o repositório de Soluções. A cada nova iteração, cada um dos Agentes VNS solicita ao Agente Analisador de Soluções uma comparação entre as soluções atuais desta nova iteração com novas soluções obtidas do Repositório de Soluções, através do Agente Coordenador. Neste instante os Agentes VNS podem utilizar de suas próprias soluções ou utilizar da solução que lhes foi enviadas do Repositório de Soluções, possibilitando com isso, que cada um dos Agentes envolvidos utilizem soluções compartilhadas uns pelos outros. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.4, 6.5 e 6.6 na Seção 6.4.

A.3 Aplicação da combinação de 1 Agente AG com 1 Agente ILS

No experimento prático da Arquitetura AMAMv1.0 envolvendo 1 Agente AG e 1 Agente ILS, participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente AG e (8) Agente ILS. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.3.

O processo de cooperação que existe entre o Agente AG e o Agente ILS acontece da mesma forma que em todos os outros experimentos práticos da Arquitetura, por troca de soluções, através do Agente Coordenador. Quando o Agente AG necessita de soluções iniciais ele envia estímulos para o Agente Coordenador e este por sua vez

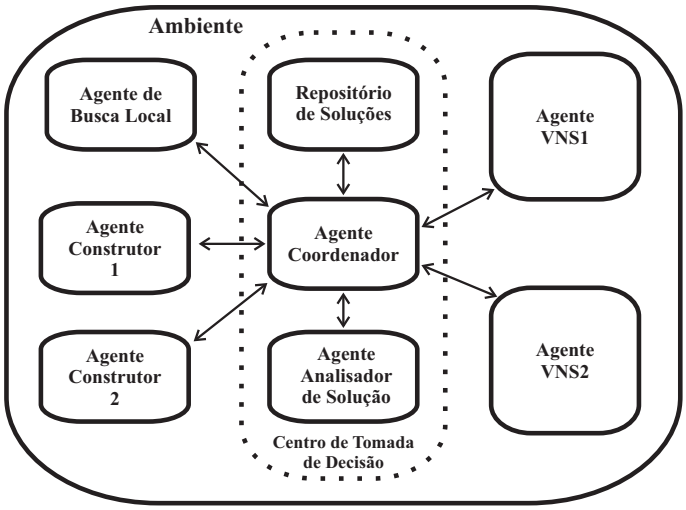

Figura A.2: Especialização da Arquitetura AMAM v1.0 para 2 Agentes VNS

retorna com estas soluções, retiradas do Repositório de Soluções. De forma idêntica acontece com o Agente ILS.

Os Agentes trabalham com estas soluções iniciais e ao término de cada iteração, em seu processo de busca, os Agentes, enviam suas soluções atuais para o Agente Coordenador, e este decidirá por enviá-las ou não para o repositório de Soluções. A cada nova iteração, cada um dos Agentes (AG e ILS) solicita ao Agente Analisador de Soluções uma comparação entre as soluções atuais desta nova iteração com novas soluções obtidas do Repositório de Soluções, através do Agente Coordenador. Neste instante os Agentes podem utilizar de suas próprias soluções ou utilizar da solução que lhes foi enviadas do Repositório de Soluções, possibilitando com isso, que cada um dos Agentes envolvidos utilizem soluções compartilhadas uns pelos outros. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.7, 6.8 e 6.9 na Seção 6.4.

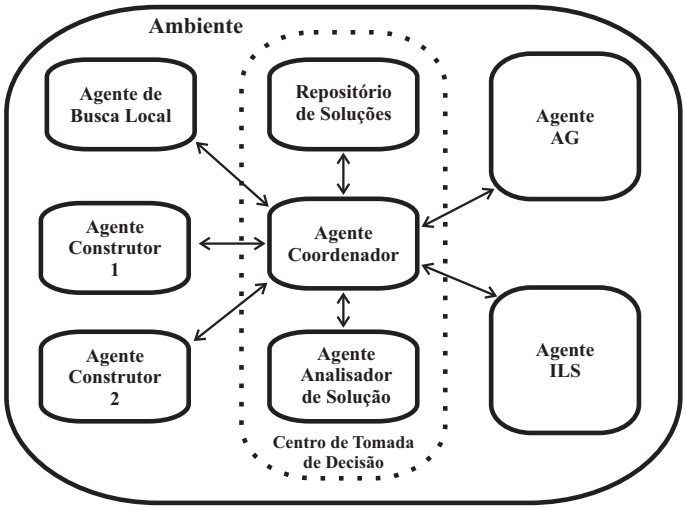

Figura A.3: Especialização da Arquitetura AMAM v1.0 para a combinação de 1 agente AG com 1 agente ILS

A.4 Aplicação da combinação de 1 Agente AG com 1 Agente VNS

No experimento prático da Arquitetura AMAMv1.0 envolvendo 1 Agente AG e 1 Agente VNS, participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente AG e (8) Agente VNS. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.4.

Esta configuração da Arquitetura AMAMv1.0 tem seu funcionamento semelhante ao apresentado na Seção A.3, sendo diferenciado somente pela existência do Agente VNS no lugar do Agente ILS. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.7, 6.8 e 6.9 na Seção 6.4.

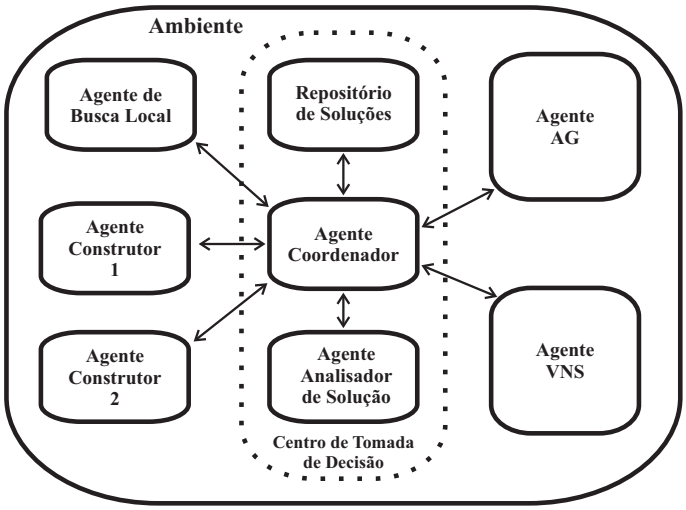

Figura A.4: Especialização da Arquitetura AMAM v1.0 para a combinação de 1 agente AG com 1 agente VNS

A.5 Aplicação da combinação de 1 Agente ILS com 1 Agente VNS

No experimento prático da Arquitetura AMAMv1.0 envolvendo 1 Agente AG e 1 Agente VNS, participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente ILS e (8) Agente VNS. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.5.

Esta configuração da Arquitetura AMAMv1.0 tem seu funcionamento semelhante ao apresentado na Seção A.4, sendo diferenciado somente pela existência do Agente ILS no lugar do Agente AG. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.7, 6.8 e 6.9 na Seção 6.4.

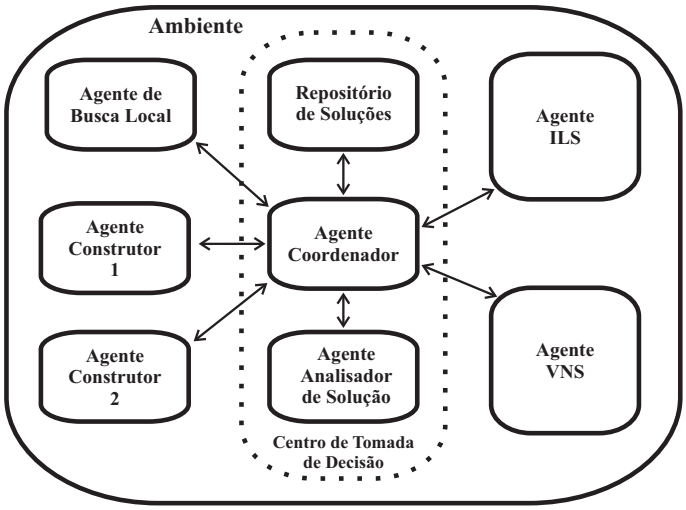

Figura A.5: Especialização da Arquitetura AMAM v1.0 para a combinação de 1 agente ILS com 1 agente VNS

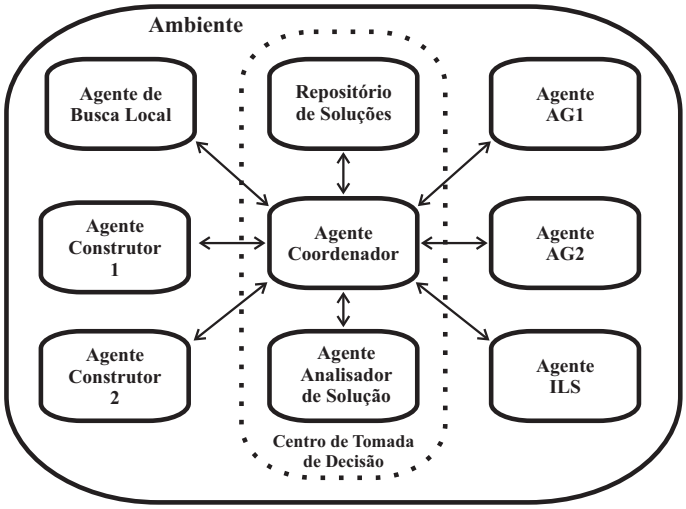

Figura A.6: Especialização da Arquitetura AMAM v1.0 para a combinação de 2 agentes AG com 1 agente ILS

A.6 Aplicação da combinação de 2 Agentes AG com 1 Agente ILS

A aplicação da combinação de 2 Agentes AG e 1 Agente ILS, trabalhando em conjunto na Arquitetura AMAMv1.0, é também, uma das formas mais completas, dos testes realizados. Este experimento prático envolve os 2 Agentes AG e 1 Agente ILS em uma busca cooperativa paralela (em um único processador), trocando soluções pertinentes ao problema tratado. Neste experimento participam do processo de busca os seguintes componentes: (1) Ambiente, (2) Agentes Construtores, (3) Agente de Busca Local, (4) Agente Coordenador, (5) Agente Analisador de Soluções, (6) Repositório de Soluções, (7) Agente AG1, (8) Agente AG2 e (9) Agente ILS. Este modelo da Arquitetura AMAMv1.0 é ilustrado na Figura A.6.

Assim como em todas as outras configurações da Arquitetura AMAMv1.0, nesta

configuração, todos os agentes podem trabalhar, em algum momento, com soluções cooperadas pelos outros agentes presentes na Arquitetura, e o comportamento da Arquitetura e dos Agentes presentes nesta configuração é o mesmo para os casos citados anteriormente. Os resultados computacionais desta aplicação são demonstrados nas Tabelas 6.10, 6.11 e 6.12 na Seção 6.4.

Referências Bibliográficas

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, NY, EUA.

Birattari, M.; Paquete, L.; Stützle, T. e Varrentrapp, K. (2001). Classification of metaheuristics and design of experiments for the analysis of components. Relatório Técnico AIDA-01-05, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt, Darmstadt, Alemanha.

Bleale, R. e Jackson, T. (1990). *Neural Computing: An Introduction*. Institute of Physics Publishing, Bristol, Reino Unido.

Blum, Christian e Roli, Andrea. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308.

Bouthillier, Alexandre Le; Crainic, Teodor G. e Kropf, Peter. (2005). A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems*, v. 20, n. 4, p. 36–42. ISSN 1541-1672.

Bouthillier, Alexandre Le e Crainic, Teodor Gabriel. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Compututers & Operations Resource*, v. 32, n. 7, p. 1685–1708. ISSN 0305-0548.

Burke, E. K.; Kendall, G.; Silva, D. L. e Soubeiga, R. O'Brien E. (2005). An ant algorithm hyperheuristic for the project presentation scheduling problem. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 3, p. 2263– 2270, (2005).

Burke, E. K.; Silva, J. D. Landa e Soubeiga, E. (2003). Hyperheuristic approaches for multi-objective optimisation. *Selected Papers from the 2003 Metaheuristics International Conference*, Kyoto Japan.

Cadenas, J. M.; Garrido, M. C. e Munoz, E. (2007). A cooperative system of metaheuristics. *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, p. 120–125, Washington, DC, EUA. IEEE Computer Society. ISBN 0-7695-2946-1.

Chen, C. L. *Bayesian Nets and A-Teams for Power System Fault Diagnosis*. PhD thesis, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, EUA, (1992).

Chen, S. Y.; Talukdar, S. N. e Sadeh, N. M. (1993). Job-shop-scheduling by a team of asynchronous agents. *Proceedings of Workshop on Knowledge-Based Production, Scheduling and Control (IJCAI-93)*, Chambery, France.

Corkill, Daniel D. (1991). Blackboard systems. *AI Expert*, v. 9, n. 6, p. 40–47.

Corkill, Daniel D.; Gallagher, Kevin Q. e Murray, Kelly E. August(1986). Gbb: A generic blackboard development system. *Proceedings of National Conference on Artificial Intelligence*, p. 1008–1014, August(1986).

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. e Stein, Clifford. (2002). *Algoritmos: Teoria e Prática*. Campus Elsevier, 2 edição.

Crainic, Teodor Gabriel e Toulouse, Michel. (2009). Parallel meta-heuristics. Relatório Técnico CIRRELT-2009-22, CIRRELT, Université de Montréal.

de Oliveira, Humberto Cesar Brandão. (2007). Um modelo híbrido estocástico para tratamento do problema de roteamento de veículos com janela de tempo. Dissertação de mestrado, Universidade Federal de Pernambuco, Recife - PE.

de Souza, P. S. *Asynchronous Organizations for Multi-Algorithm Problems*. PhD thesis, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, EUA, (1993).

Dorigo, M. *Optimization, learning and natural algorithms (in Italian)*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Itália, (1992).

Dorigo, Marco e Blum, Christian. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, v. 344, n. 344, p. 243–278.

Dorigo, Marco e Di Caro, Gianni. (1999). The ant Colony Optimization meta-heuristic. Corne, David; Dorigo, Marco e Glover, Fred, editors, *New Ideas in Optimization*, Capítulo 2, p. 11–32. McGraw-Hill, London, Reino Unido.

Dorigo, Marco e Gambardella, Luca Maria. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 53–66.

Dorigo, Marco; Maniezzo, Vittorio e Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, v. 26, n. 1, p. 29–41.

Driankov, D.; Hellendoorn, H. e Reinfrank, M. (1990). *An Introduction to Fuzzy Control*. Springer-Verlag, Berlin, Alemanha, 1ª edição.

Çela, E. (1998). *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer.

Feo, T. e Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, n. 2, p. 109–133.

Fernandes, Filipe Costa; de Souza, Sérgio Ricardo; Borges, Henrique Elias e Silva, Maria Amélia Lopes. (2009)a. Arquitetura multiagentes baseada em metaheurísticas para solução de problemas de otimização combinatória. *Anais do IX Simpósio Brasileiro de Automação Inteligente*, (2009)a.

Fernandes, Filipe Costa; de Souza, Sérgio Ricardo; Borges, Henrique Elias e Silva, Maria Amélia Lopes. (2009)b. Metaheurísticas cooperativas para solução do problema de roteamento de veículos com janela de tempo. *Anais do 30o CILAMCE - Iberian-Latin-American Congress on Computational Methods in Engineering*, (2009)b.

Fernandes, Filipe Costa; de Souza, Sérgio Ricardo; Borges, Henrique Elias e Silva, Maria Amélia Lopes. (2009)c. Uma arquitetura multiagentes para solução de problemas de otimização combinatória: aplicação ao problema de roteamento de veículos com janela de tempo. *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, p. 2181–2192, (2009)c.

Fernandes, Filipe Costa; de Souza, Sérgio Ricardo; Borges, Henrique Elias; Silva, Maria Amélia Lopes e Ribeiro, Fabio Fernandes. (2009)d. A multiagent architecture for solving combinatorial optimization problems through metaheuristics. *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics, 11-14 October, 2009*, p. 3071–3076, San Antonio - Texas, EUA.

Gendron, Bernard; Potvin, Jean-Yves e Soriano, Patrick. (2003). A parallel hybrid heuristic for the multicommodity capacitated location problem with balancing requirements. *Parallel Computing*, v. 29, n. 5, p. 591–606. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(03)00044-9.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, v. 13, p. 533–549.

Gorti, S. R.; Humair, S; Sriram, R. D.; Talukdar, S. e Murthy, S. (1996). Solving constraint satisfaction problems using a-teams. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, v. 10, n. 1, p. 1–19.

Hansen, P. e Mladenovic, N. (1999). An introduction to variable neighborhood search. Voß, S.; Martello, S.; Osman, I. e Roucairol, C., editors, *Metaheuristics: Advances and trends in local search paradigms for optimization*, p. 433–438. Kluwer Academic Publishers, Norwell, MA, EUA.

Hansen, P. e Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, v. 130, p. 449–467.

Hansen, Pierre; Mladenovic, Nenad e Pérez, José Andrés Moreno. (2003). Variable neighbourhood search. *Revista Iberoamericana de Inteligencia Artificial*, v. 19, p. 77–92. ISSN 1137-3601.

Holland, J.H.; Langton, C. e S.W.Wilson,. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, EUA.

Hoos, Holger H. e Stützle, Thomas. (2005). *Stochastic Local Search Foundations and Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann - Elsevier, San Francisco, CA, USA.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680.

Larsen, J. *Parallelization of the vehicle routing problem with time windows*. PhD thesis, Department of Mathematical Modeling, Technical University of Denmark, (1999).

Lesser, Victor R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, v. 11, n. 1, p. 133–142.

Lourenço, H. R.; Martin, O. e Stutzle, T. (2002). Iterated local search. Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 321–353. Kluwer Academic Publishers, Norwell, MA, EUA.

Lukin, J. A.; Gove, A. P.; Talukdar, S. N. e Ho, C. (1997). Automated probabilistic method for assigning backbone resonances of (13c, 15n)-labelled proteins. *Journal of Biomolecular NMR*, v. 9, p. 151–166.

Michalewicz, M.Z. e Schoenauer. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, v. 4, p. 1–32.

Milano, M. e Roli, A. (2004). Magma: a multiagent architecture for metaheuristics. *IEEE Transactions on Systems Man and Cybernetics*, v. 34, n. 2.

Ombuki, Beatrice; Ross, Brian J. e Hanshar, Franklin. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, v. 24, n. 1, p. 17–30.

Osman, I. H. e Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, v. 63, p. 513–628.

Pardalos, P.; Rendl, F. e Wolkowicz, H. (1994). The quadratic assignment problem: a survey and recent developments. *DIMACS Ser. Discrete Mathematics Theory Computation Science*, v. 16, p. 1–42.

Passos, C. A. S. e Fonseca, S. L. A. (2005). Uma arquitetura multiagente para a solução de problemas de sequenciamento da produção. *INFOCOMP Journal of Computer Science*, v. 4, n. 2, p. 38–45.

Pelta, David Alejandro. *Algoritmos Heurísticos en Bioinformática*. PhD thesis, Universidad de Granada, Granada, Espanha, (2005).

Rizzoli, A. E.; Oliverio, F.; Montemani, R. e Gambardella, C. M. (2004). Ant colony optimization fo vehicle routing problem: from theory to application. Relatório Técnico IDSIA-15-04, IDSIA - University of Lugano.

Roli, Andrea e Milano, Michela. (2001). Metaheuristics: a multiagent perspective. Relatório Técnico 50, Universidade de Bolonha.

Russell, S. e Norvig, P. (2004). *Inteligência Artificial*. Elsevier, Rio de Janeiro, Brasil.

Shuai, Dianxun e Feng, Xiang. (2005). Distributed problem solving in multiagent systems: A spring net approach. *IEEE Intelligent Systems*, p. 66–74.

Silva, M. A. L.; de Souza, S. R.; Borges, H. E.; Oliveira, S. M. e Temponi, E. C. C. (2007). Amam: Arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória. *Anais do VII Simpósio Brasileiro Automação Inteligente (SBAI 2007)*, Florianópolis, Brasil.

Silva, M. A. L.; de Souza, S. R. e Oliveira, S. M. (2006). Uma proposta de arquitetura multiagente para a solução via metaheurísticas do problema de roteamento de veículos com janela de tempo. *Anais do IX Encontro de Modelagem Computacional (EMC 2006)*, Belo Horizonte, Brasil.

Silva, Maria Amélia Lopes. (2007). Modelagem de uma arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória. Dissertação de mestrado, Mestrado em Modelagem Matemática e Computacional, CEFET-MG, Belo Horizonte - MG.

Solomom, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, v. 2, n. 35, p. 254–264.

Talbi, El-Ghazali e Bachelet, Vincent. (2006). Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, v. 5, p. 5–22.

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. N. e de Souza, P. S. (1990). Asynchronous teams. *Proceedings of the Second SIAM - Conference on Linear Algebra: Signals, Systems and Control*, San Francisco, CA, EUA.

Toth, P. e Vigo, D. (2000). *The Vehicle Routing Problem.* SIAM - Society for Industrial and Applied Mathematics.

van derHoek, W. e Wooldridge, M. (2007). Multi-agent systems. Porter, B.; Lifschitz, V. e vanHarmelen, F., editors, *Handbook of Knowledge Representation*, Capítulo 24, p. 1–41. Elsevier, Amsterdam, NL.

Weiss, Gerhard. (1999). *Multiagent Systems: A modern Approach to Distributed Artificial Intelligence*. MIT Press.

Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. John Wiley & Sons, Cambridge, MA.

Wooldridge, M. e Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, v. 10, n. 2, p. 41–59.

Ziviani, Nivio. (2007). *Projeto de Algoritmos com implementações em Java e C++*. Thomson, Rio de Janeiro, Brasil.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo