UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

RONNEESLEY MOURA TELES

Um estudo de técnicas da Inteligência Artificial aplicadas na distribuição de recursos em áreas geográficas

Goiânia
2011

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Um estudo de técnicas da Inteligência Artificial aplicadas na distribuição de recursos em áreas geográficas

Autor(a): Ronneesley Moura Teles

Goiânia, 28 de Abril de 2011.

Ronneesley Moura Teles – Autor

Cedric Luiz de Carvalho – Orientador

RONNEESLEY MOURA TELES

Um estudo de técnicas da Inteligência Artificial aplicadas na distribuição de recursos em áreas geográficas

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Inteligência Artificial.

Orientador: Prof. Cedric Luiz de Carvalho

Goiânia
2011

RONNEESLEY MOURA TELES

Um estudo de técnicas da Inteligência Artificial aplicadas na distribuição de recursos em áreas geográficas

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 28 de Abril de 2011, pela Banca Examinadora constituída pelos professores:

Prof. Cedric Luiz de Carvalho
Instituto de Informática – UFG
Presidente da Banca

Profa. Telma Woerle de Lima Soares
Instituto de Informática – UFG

Prof. Ivan Nunes da Silva
Engenharia Elétrica – USP

Ronneesley Moura Teles

Graduou–se em Sistemas de Informação na Universo - Universidade Salgado de Oliveira. Durante sua graduação, atuou como desenvolvedor em Visual Basic, PHP e Java para o Tribunal de Justiça do Estado de Goiás. Especializou-se em Tecnologia da Informação e Negócios Eletrônicos pela Universidade Salgado de Oliveira.

Dedico este trabalho a todas as pessoas que me acompanham na vida. Em especial minha mãe Maria Vilani Pereira de Moura, meu pai Sebastião Teles da Silva, meu irmão Ronneery Moura Teles e todas pessoas que amo, pois me motivam a superar os desafios encontrados a cada dia.

Agradecimentos

Agradeço a companhia Celg de Participações (CELGPAR) por permitir o estudo de seu problema e a todos meus colegas que opinaram e apoiaram meu trabalho.

Resumo

Teles, Ronneesley Moura. **Um estudo de técnicas da Inteligência Artificial aplicadas na distribuição de recursos em áreas geográficas**. Goiânia, 2011. 136p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Este trabalho estuda um problema que consiste na distribuição de recursos em áreas geográficas. Muitas organizações, desde cooperativas de táxi até as forças armadas enfrentam este problema em suas operações diárias. Em essência, tenta-se responder a pergunta: "Quais os melhores lugares em que devo posicionar meus recursos na área geográfica X, de acordo com um conjunto de restrições Y?". Para responder a pergunta, foram estudados modelos de representação do problema, foram aplicadas técnicas conhecidas da Inteligência Artificial e da Pesquisa Operacional, tais como: teste de todas combinações, algoritmos gulosos, heurísticas e algoritmos genéticos. Foram criados algoritmos e foram realizadas simulações. Além disso, foi desenvolvido um Sistema Multiagente que implementa um dos algoritmos criados. Na avaliação das propostas feitas nesta dissertação, foi considerado o domínio das Companhias Elétricas, na tarefa de distribuir viaturas em uma cidade.

Palavras–chave

Sistemas Multiagentes, Inteligência Artificial, Simulações

Abstract

Teles, Ronneesley Moura. **Resource Allocation in Geographical Areas**. Goiânia, 2011. 136p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

This paper studies a problem that is the distribution of resources in geographical areas. Many organizations, from taxis, to the military faces this problem in their daily operations. In essence, it tries to answer the question: "What are the best places in which I place my assets in the geographic area X, according to a set of constraints Y?". To answer the question, we studied representation models of the problem, known techniques have been applied in Artificial Intelligence and Operations Research, such as: testing all combinations, greedy algorithms, heuristics and genetic algorithms. Algorithms were created and simulations were performed. Furthermore, Multiagent System was developed that implements one of algorithms developed. In evaluating the proposals made in this dissertation, it was considered the field of Electric Companies in the task of distributing vehicles in a city.

Keywords

Multiagent Systems, Artificial Intelligence, Simulations

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

Muitas organizações lidam diariamente com a distribuição de recursos em zonas geográficas como cidades, estados ou países. Em algumas destas organizações, estes recursos são distribuídos manualmente sem analisar todos os dados das regiões.

Uma distribuição mal feita pode ter consequências terríveis. Em alguns casos, pode causar mortes pela demora da chegada do recurso no local onde ele é necessário.

Sabe-se que a distribuição destes recursos, feita manualmente, não garante sua chegada rápida e nem a melhoria do processo de distribuição, causando a insatisfação de quem precisa deles.

Este problema pode ser encontrado em muitas organizações, dentre elas:

- cooperativas de táxi que posicionam os veículos numa cidade;
- departamentos de polícia que distribuem as viaturas em uma cidade, buscando-se minimizar crimes;
- clubes de futebol que posicionam os jogadores visando uma melhor defesa ou ataque;
- empresas aéreas que devem decidir em quais cidades construir aeroportos;
- multinacionais para decidir em quais cidades criar novas filiais.
- empresas de segurança para definir o posicionamento dos guardas e viaturas;
- distribuição de componentes na engenharia civil, por exemplo: lâmpadas e irrigadores;
- forças armadas para distribuição dos componentes navais, terrestres e aéreos;
- companhias elétricas[1] para distribuição de viaturas numa cidade.

Neste trabalho, serão apresentados algoritmos criados para resolução do problema da distribuição de recursos e um sistema multiagente que utiliza um algoritmo guloso.

[1] Neste trabalho será utilizado o termo companhias elétricas para referenciar as companhias de fornecimento de energia elétrica.

1.1 Motivação

A definição clara do problema da distribuição de recursos num espaço geográfico ajuda a melhorar as distribuições futuras e, consequentemente, melhora o atendimento de quem precisa dos recursos.

O desenvolvimento de algoritmos que resolvem, parcialmente ou totalmente, o problema é uma forma de documentar como a distribuição é feita. Por isto, estes algoritmos permitem manter e melhorar continuamente o método de distribuição.

Sem uma definição formal, quando os responsáveis pela distribuição saem da organização ou falecem, todo o conhecimento é levado junto com eles.

Além de serem uma definição formal, os algoritmos permitem a análise uniforme, julgando a necessidade real dos recursos numa região.

Desta forma, caso o cenário mude, o algoritmo receberá novos parâmetros e realizará a distribuição dos recursos de acordo com a nova realidade.

A importância da distribuição bem planejada dos recursos no espaço de atendimento para algumas organizações que lidam com vidas, tais como: companhias elétricas, polícia e as forças armadas, são os tópicos que motivam este estudo.

1.2 Objetivo

O objetivo geral do trabalho é criar algoritmos de distribuição de recursos, objetivando posicioná-los, segundo critérios preestabelecidos, nos lugares em que estes recursos são mais necessários dentro de uma área geográfica.

Para isto, é necessário definir matematicamente o problema da distribuição, definir uma forma de avaliação das distribuições, verificar quais técnicas da Inteligência Artificial podem ser utilizadas para distribuição de recursos e procurar técnicas que resultam nas melhores distribuições possíveis.

1.3 Estrutura do Trabalho

Este trabalho está dividido em 6 capítulos, além desta introdução. O Capítulo 2 apresenta alguns trabalhos relacionados que esclarecem vertentes do problema, e mostram que, existem outros grupos preocupados na resolução do problema apresentado neste trabalho.

O Capítulo 3 apresenta a definição matemática proposta para o problema da distribuição de recursos e uma forma de avaliação das distribuições.

O Capítulo 4 apresenta técnicas da Inteligência Artificial que foram aplicadas na resolução do problema. Neste capítulo, são criados algoritmos para resolução do problema da distribuição de recurso.

Foram utilizadas várias técnicas na concepção dos algoritmos apresentados no Capítulo 4, dentre elas destacam-se: algoritmos gulosos, algoritmos genéticos e heurísticas. A partir de simulações, foram avaliadas as performances dos algoritmos apresentados.

O Capítulo 5 apresenta um sistema multiagente (SMA) criado com base num algoritmo guloso proposto no Capítulo 4. Este SMA delega aos agentes o cálculo das melhores posições em uma subárea de atendimento e, posteriormente, outro agente se responsabiliza da união dos resultados. Com base na característica de autonomia dos agentes, o SMA pode ser complementado com recursos que o permitam funcionar em uma grade computacional.

O Capítulo 6 apresenta a análise dos resultados de cada algoritmo, apresentados no Capítulo 4 e no Capítulo 5, e uma comparação entre seus resultados quanto ao tempo gasto nas simulações e quanto a qualidade dos resultados, segundo a função de avaliação, definida no Capítulo 3.

Finalmente, o Capítulo 7 apresenta as considerações finais, contribuições deste trabalho, possíveis trabalhos futuros e as conclusões do trabalho. Em especial, este capítulo apresenta uma questão que não foi respondida, incentivando matemáticos e outros pesquisadores a pensarem sobre a representação do problema em programação linear.

Neste capítulo foi apresentado superficialmente o problema da distribuição e os motivos que levaram ao estudo deste trabalho. O capítulo a seguir apresenta os trabalhos relacionados ao problema da distribuição de recursos. Na grande maioria, são apresentados Sistemas Multiagentes e estratégias específicas para solucionar problemas similares ao da distribuição de recursos.

CAPÍTULO 2

Trabalhos Relacionados

Este capítulo apresenta trabalhos encontrados na literatura relacionados ao problema da distribuição de recursos em áreas geográficas.

2.1 Reorganização de Viaturas Policiais

Segundo Melo et al [12], a distribuição das viaturas de polícia numa cidade é feita de forma descentralizada por cada departamento de polícia. Estes departamentos de polícia trabalham para diminuir a taxa de criminalidade numa região.

Melo et al apresentam a hipótese de que, com a informação de onde ocorrem e as razões associadas aos crimes, é possível fazer uma distribuição otimizada e, consequentemente, diminuir a taxa de criminalidade.

No entanto, os gestores que distribuem as viaturas policiais na subárea de atendimento possuem grandes dificuldades de entender as relações entre os crimes e os fatores que motivaram o acontecimento.

Segundo Melo et al, destacar os agrupamentos criminais, mesmo utilizando um sistema de informação geográfica, não é uma tarefa trivial. Também é destacado que, não é fácil fazer experimentos neste domínio, pois podem resultar em perdas de vidas humanas. Por isto, sistemas de simulação são primordiais para a tomada de decisões e testes dos algoritmos. Desta forma, Melo et al criaram um simulador para realizar experimentos.

No simulador criado, a principal informação que a viatura de polícia possui é o posicionamento geográfico ou a rota que a viatura deve fazer ou está fazendo.

Para realizar simulações, Melo et al criaram uma sociedade de agentes, composta pelos agentes:

- Pontos notáveis: lugares de atividade financeira, tais como: bancos, loterias, postos de gasolina e *shoppings*;
- Central de emergência: responsável pelo recebimento das ocorrências e direcionamento para a viatura mais próxima;
- Polícia: responsável por resolver as ocorrências criminais;

- Bandidos: executam crimes na área de atendimento.

Para completar a definição dos agentes, eles determinam o comportamento dos bandidos da seguinte forma: se a taxa de satisfação do atendimento policial for maior do que a taxa ideal, os bandidos não fazem nada e voltam para suas casas, mas se for menor que a taxa ideal, os bandidos cometem crimes.

Melo et al propõem três tipos de personalidade para os agentes bandidos, são elas: perigoso, intermediário e novato. De acordo com a personalidade, foram determinadas probabilidades de ocorrer um crime para cada tipo de zona notável. A Tabela 2.1 apresenta as probabilidades de cometer crimes de cada personalidade de acordo com o ponto notável.

Tabela 2.1: *Probabilidades de ocorrer crimes de acordo com o ponto notável.*

Tipo de Alvo	**Personalidade**		
	Perigoso	**Intermediário**	**Novato**
Praça	10%	20%	50%
Drogaria	15%	30%	30%
Casa Lotérica	15%	30%	20%
Posto de Gasolina	10%	20%	
Shopping	15%		
Banco	35%		

Nota-se que Melo et al [12] não destacam como estas probabilidades foram calculadas e nem qual a fonte desses valores. Então, podem estar incorretos ou terem sido preenchidos empiricamente.

Para criação do simulador, Melo et al utilizaram a ferramenta Repast [14], desenvolvida pela University of Chigaco. Essa ferramenta permite a definição das regras graficamente e executa a simulação.

Melo et al definem estratégias nas rotas das viaturas de polícia para combater a criminalidade. São definidas rotas curtas quando a viatura é acionada numa ocorrência criminal, rotas longas quando a viatura está patrulhando pontos notáveis e rotas críticas quando as viaturas devem cobrir pontos de alta criminalidade. No entanto, os posicionamentos iniciais das viaturas são aleatórios.

Os agentes policiais mantém o número de crimes ocorridos e o número de crimes evitados de cada ponto notável. Essas propriedades representam a punição ou recompensa da viatura. Como estratégia adota-se que uma boa rota é onde a viatura policial mais evita crimes.

Melo et al apresentam uma simulação de um ambiente com 15 bandidos, sendo 5 perigosos, 5 intermediários e 5 novatos. Na simulação apresentada por Melo et al, existem 40 pontos notáveis e a iluminação pública segue a distribuição: 50% boa, 25% mediana e 25% ruim. Nessa simulação, a polícia não possui nenhuma estratégia de organização, ou seja, esses agentes não conseguem prever como o mundo evolui [13].

Nota-se que Melo et al não definem claramente a influência do grau de iluminação na ação dos bandidos. Apenas é definido que iluminações fracas aumentam as chances de ação dos bandidos.

O resultado dessa simulação apresentou que a quantidade de crimes aumentava à medida em que o tempo passava. Segundo Melo et al, a explicação para esse fenômeno é que, à medida em que o tempo de simulação aumenta, os bandidos vão adquirindo mais experiência e a quantidade de crimes aumenta. A Figura 2.1 apresenta os resultados da simulação.

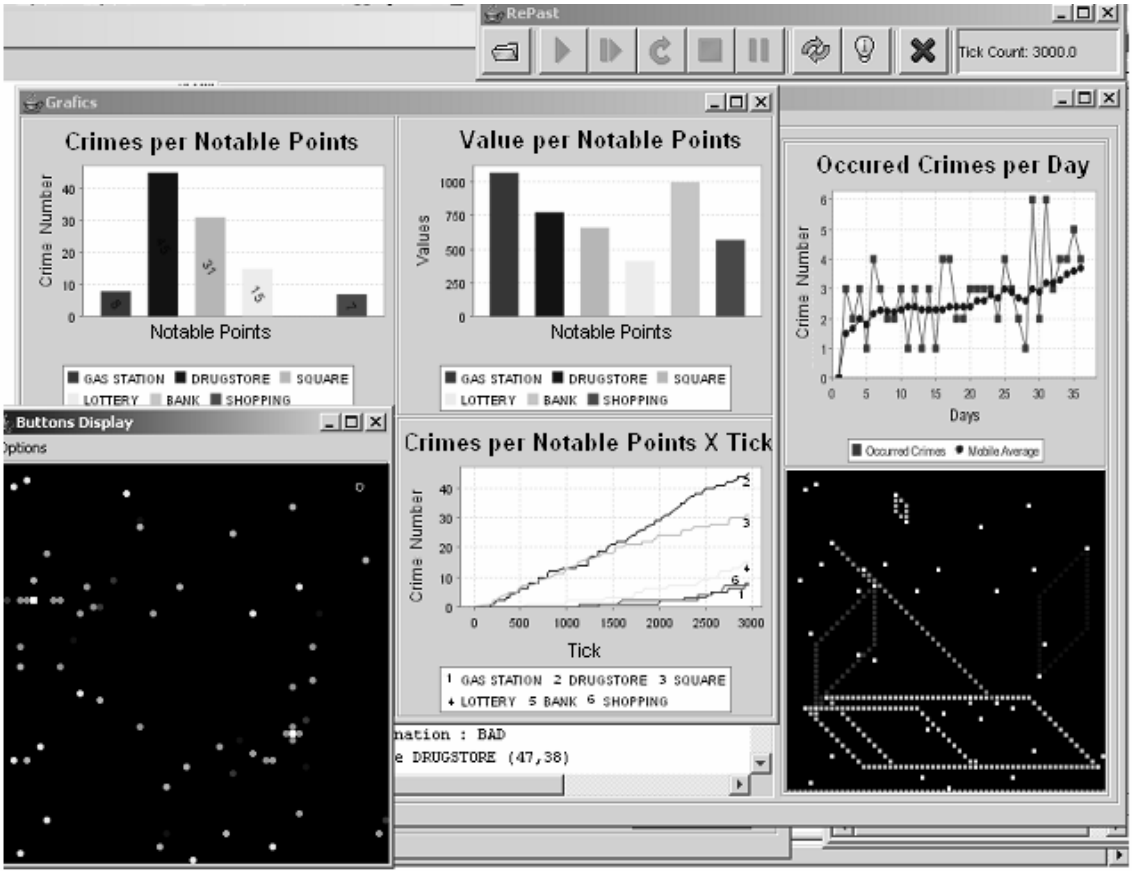


Figura 2.1: *Resultados da simulação.*

Na parte superior direita da Figura 2.1, é apresentado o gráfico de crimes por dia (eixo y). Nota-se que a média de crimes desse gráfico aumenta à medida que o tempo de simulação, eixo x, passa.

A média de crimes por dia aumenta porque os bandidos adquirem experiências ao longo da simulação. Dessa forma, bandidos novatos passam a ser intermediários, que

por sua vez, passam a ser perigosos. Consequentemente, as rotas previamente calculadas pela polícia não são mais eficazes à medida em que esses bandidos evoluem. Por esse motivo, existe a necessidade de reorganização da polícia para evitar que a criminalidade aumente.

A Figura 2.2 apresenta os resultados da simulação quando é utilizada uma reorganização centralizada com base em rotas curtas.

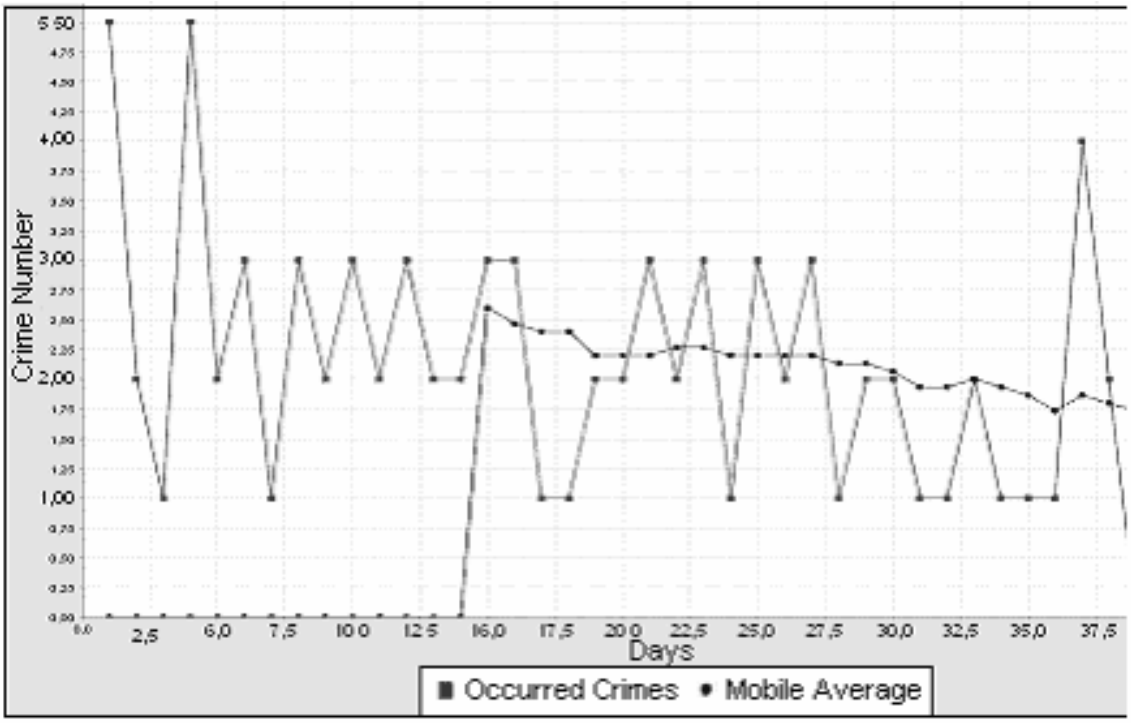


Figura 2.2: *Resultados da simulação feita por Melo et al. com reorganização centralizada.*

Na Figura 2.2, pode-se notar que a média de crimes diminui ao longo da simulação.

Finalmente, Melo et al apresentam o resultado de 15 simulações feitas sem reorganização das viaturas policiais e com a reorganização utilizando as estratégias de rotas curtas, rotas longas e rotas críticas. A Figura 2.3 apresenta os resultados de cada estratégia.

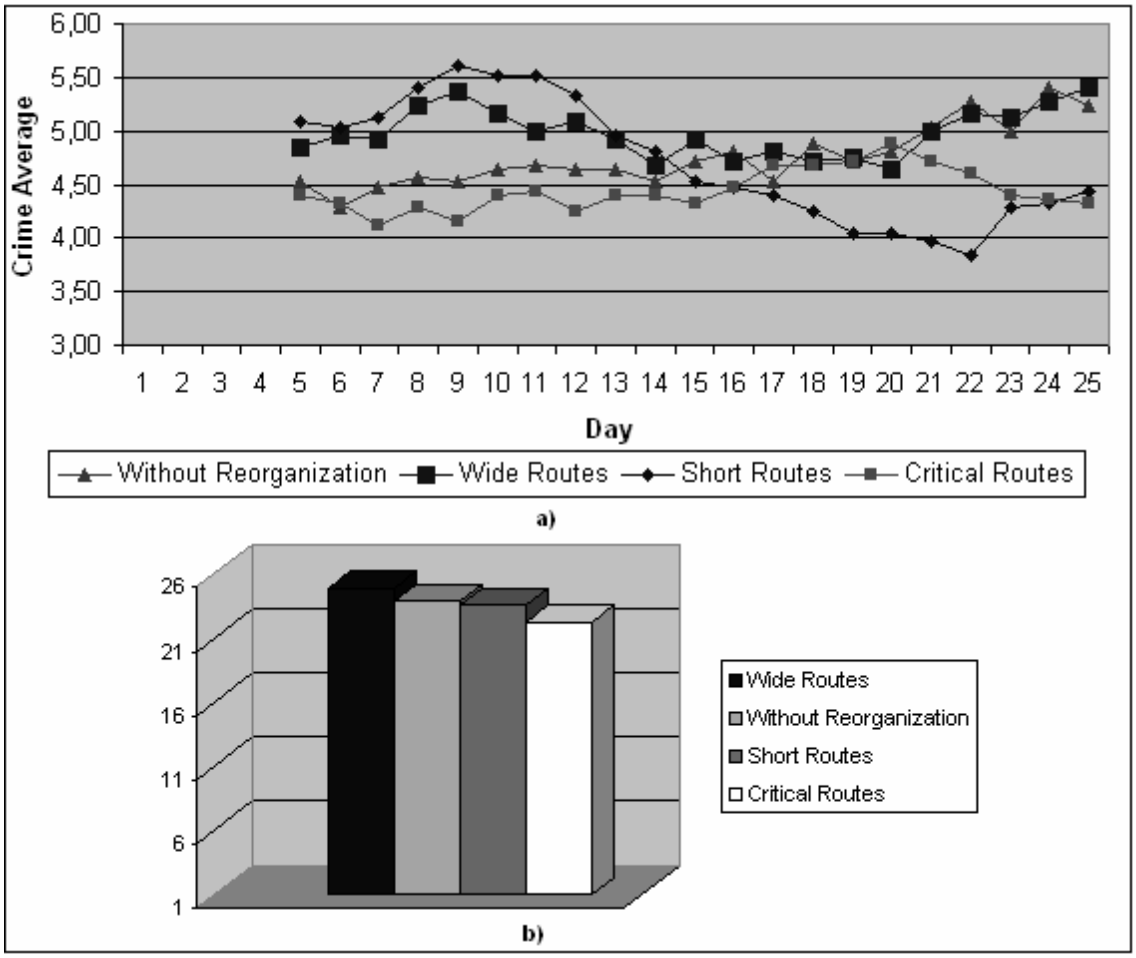


Figura 2.3: *Resultados finais das simulações feitas por Melo et al. com e sem reorganização.*

A partir da Figura 2.3, pode-se notar que, sem reorganização e utilizando a estratégia de rotas longas, não foi possível controlar a criminalidade, enquanto que, utilizando as estratégias de rotas curtas e rotas críticas a criminalidade foi controlada.

2.2 Patrulhamento de Terrenos

Machado et al [11] apresentam formas de construir um Sistema Multiagentes (SMA) para auxiliar no patrulhamento de terrenos. Segundo eles, patrulhamento é o ato de vistoriar regularmente uma determinada área.

Um exemplo concreto de patrulhamento pode ser encontrado em jogos de guerra, onde grupos de patrulha devem ser coordenados para detectar a presença dos inimigos.

A técnica de representação do terreno proposta em Machado et al é chamada de *skeletonization*. Essa técnica consiste na representação do terreno através de grafos representando os possíveis caminhos por onde pode-se passar.

A definição do problema proposto por Machado et al é dada da seguinte forma: Um conjunto de N agentes e K vértices conectados por arestas de pesos iguais. O objetivo é diminuir o intervalo de visita dos vértices.

Machado et al propõem a utilização de agentes reativos [13, página 46] e agentes cognitivos [13, página 48] para resolução do problema. Desta forma são apresentadas as arquiteturas dos agentes de acordo com a Tabela 2.2.

Tabela 2.2: *Arquiteturas propostas por Machado et al.*

Nome da Arquitetura	**Tipo Básico**	**Comunicação**	**Método de Escolha do Próximo Vértice**	**Estratégia de Coordenação**
Random Reactive	reativo	nenhuma	localmente aleatória	emergente
Conscientious Reactive	reativo	nenhuma	ociosidade local individual	emergente
Reactive with Flags	reativo	sinais / marcações (*flags*)	ociosidade local compartilhada	emergente
Conscientious Cognitive	cognitivo	nenhuma	ociosidade individual global	emergente
Blackboard Cognitive	cognitivo	*blackboard*	ociosidade compartilhada global	emergente
Random Coordinator	cognitivo	mensagens	globalmente aleatória	central
Idleness Coordinator	cognitivo	mensagens	ociosidade compartilhada global	central

Segundo Machado et al, os agentes reativos poderiam visualizar apenas os nós adjacentes à sua posição na representação do terreno, enquanto que os agentes cognitivos poderiam visualizar profundidades maiores que 1, ou seja, seria possível traçar caminhos.

Segundo Russell e Norvig [13, páginas 47 a 49] a distinção de agentes reativos para cognitivos se dá na execução de suas ações e no planejamento delas. Por isso, a capacidade de traçar caminhos não pode ser uma característica que distingue as duas

categorias de agentes. Sendo assim, não há motivos para um agente reativo não possuir os dados do grafo e não poder traçar rotas nele.

Machado et al também propõem dois métodos para coordenação dos agentes. O primeiro método é a utilização de uma coordenação centralizada escolhendo os destinos de cada agente. O segundo método é a coordenação descentralizada, onde os agentes se comunicam para estabelecerem os destinos.

Eles destacam que, nos resultados obtidos de seus experimentos, agentes com um comando centralizado tiveram desempenhos melhores do que os respectivos agentes sem um comando centralizado.

A Figura 2.4 apresenta os cenários utilizados por Machado et al. O mapa A representa um terreno onde não existem becos, ou seja, é um grafo onde cada vértice possui dois ou mais vizinhos. O mapa B representa um terreno com becos, ou seja, um grafo onde alguns vértices possuem apenas um vizinho.

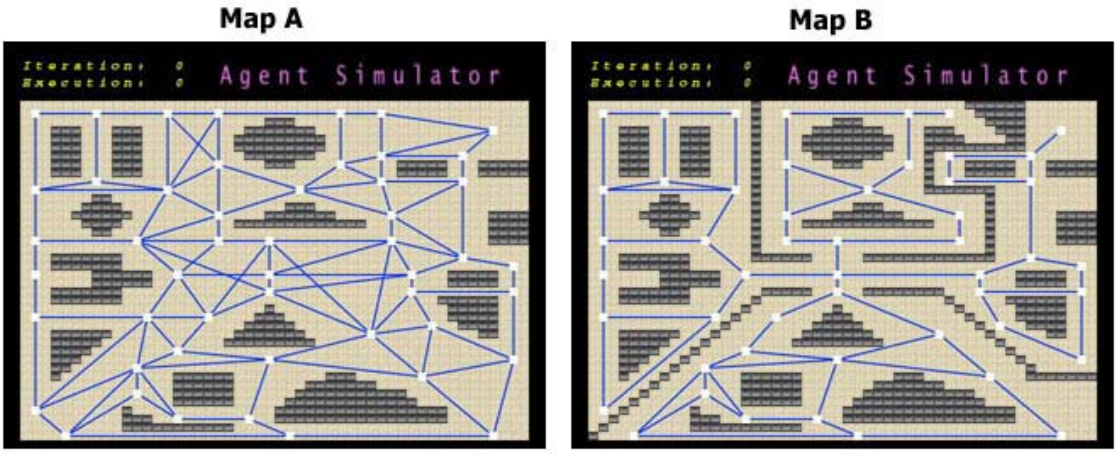
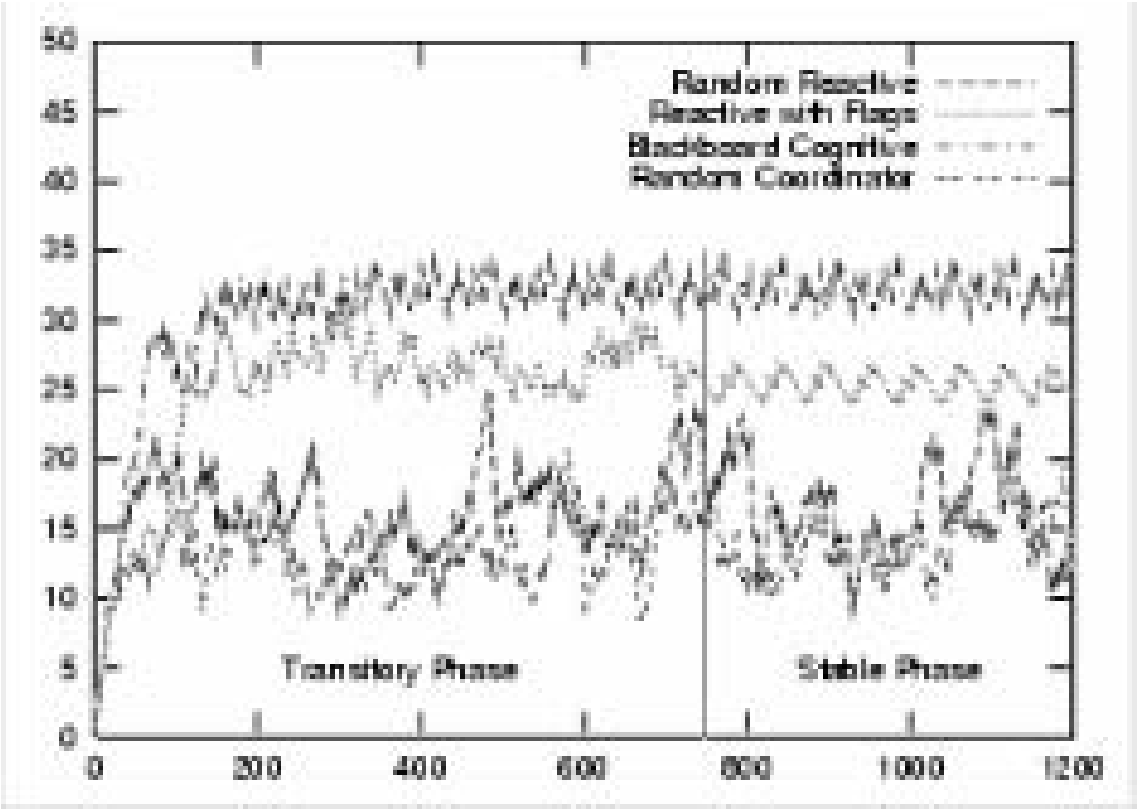


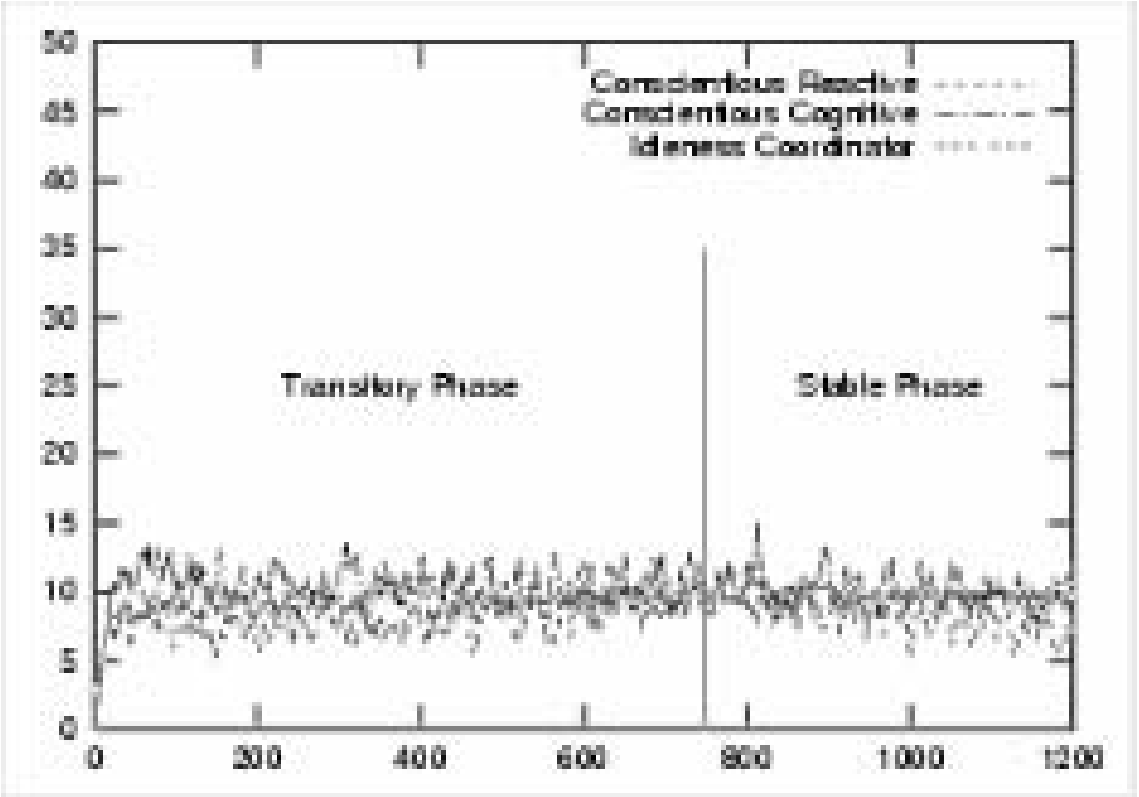
Figura 2.4: *Cenários utilizados por Machado et al em suas simulações.*

Com os mapas apresentados na Figura 2.4, Machado et al fizeram 360 simulações, com seis números diferentes de agentes (1, 2, 5, 10, 15 e 25) e 30 diferentes pontos de inicialização. O simulador utilizado foi criado por eles em C++ com OpenGL. Os pontos de inicialização dos agentes foram aleatoriamente selecionados. Cada simulação foi composta de 3 mil ciclos, sendo que um ciclo é o tempo necessário para um agente chegar até o vértice adjacente.

A Figura 2.5 apresenta os resultados da simulação com 5 agentes. O eixo y do gráfico representa a ociosidade total dos vértices e no eixo x a quantidade de ciclos da simulação.



(a) *Agentes: Random Reative, Reactive with Flags, Blackboard Cognitive e Random Coordinator.*



(b) *Agentes: Conscientious Reactive, Conscientious Cognitive e Idleness Coordinator.*

Figura 2.5: *Resultados das simulações feitas por Machado et al com 5 agentes.*

Os experimentos de Machado et al mostraram que a fase de adaptação dos agentes acaba no ciclo 750, exceto para os agentes reativos aleatórios (*Random Reactive Agents*) e de coordenação aleatória (*Random Coordinator*) que acaba próximo do ciclo 2.250.

Após os experimentos, foi possível agrupar os resultados pela arquitetura dos agentes. A Tabela 2.3 apresenta esses grupos.

Tabela 2.3: *Grupos de arquitetura do SMA.*

Grupo	**Tipo de Agente**
Random Group	*Random Reactive Agent* *Random Coordinator*
Non-coordinated Group	*Reactive Agent with Flags* *Blackboard Cognitive Agent*
Top Group	*Conscientious Reactive Agent* *Conscientious Cognitive Agent* *Idleness Coordinator*

Finalmente, por causa dos becos encontrados no mapa B (Figura 2.4) a performance dos agentes foi pior do que no mapa A em todos os experimentos. O grupo *Top Group* obteve os melhores resultados de todas as métricas e foi pouco afetado por causa dos becos do mapa B.

2.3 ExpertCop

ExpertCop [20] é um simulador tutorial multi-agente para o treinamento da alocação de equipes policiais. O ExpertCop é um trabalho complementar ao apresentado na seção 2.1.

Nele foi construido um sistema multi-agente (SMA) utilizando a plataforma JADE (*Java Agent Develop Framework*) [9] juntamente com um sistema GIS (*Geographical Information Sistems*) para simulação da criminalidade [7] num ambiente urbano.

O SMA é composto por três grupos de agentes: agentes de controle, agentes de domínio e agente pedagógico.

Os agentes de controle são responsáveis pela comunicação e pelo fluxo do sistema. Para intermediar o sistema GIS e o SMA existe um agente chamado agente GIS.

O agente pedagógico ajuda os estudantes (policiais) na compreensão dos crimes ocorridos. Esse agente utiliza árvores de prova para descrever os passos de raciocínio que levaram aos crimes cometidos pelos agentes criminosos.

Os agentes de domínio são os pontos notáveis, as equipes policiais e os criminosos.

A Figura 2.6 apresenta a interface que possibilita a alocação e definição das rotas das viaturas policiais.

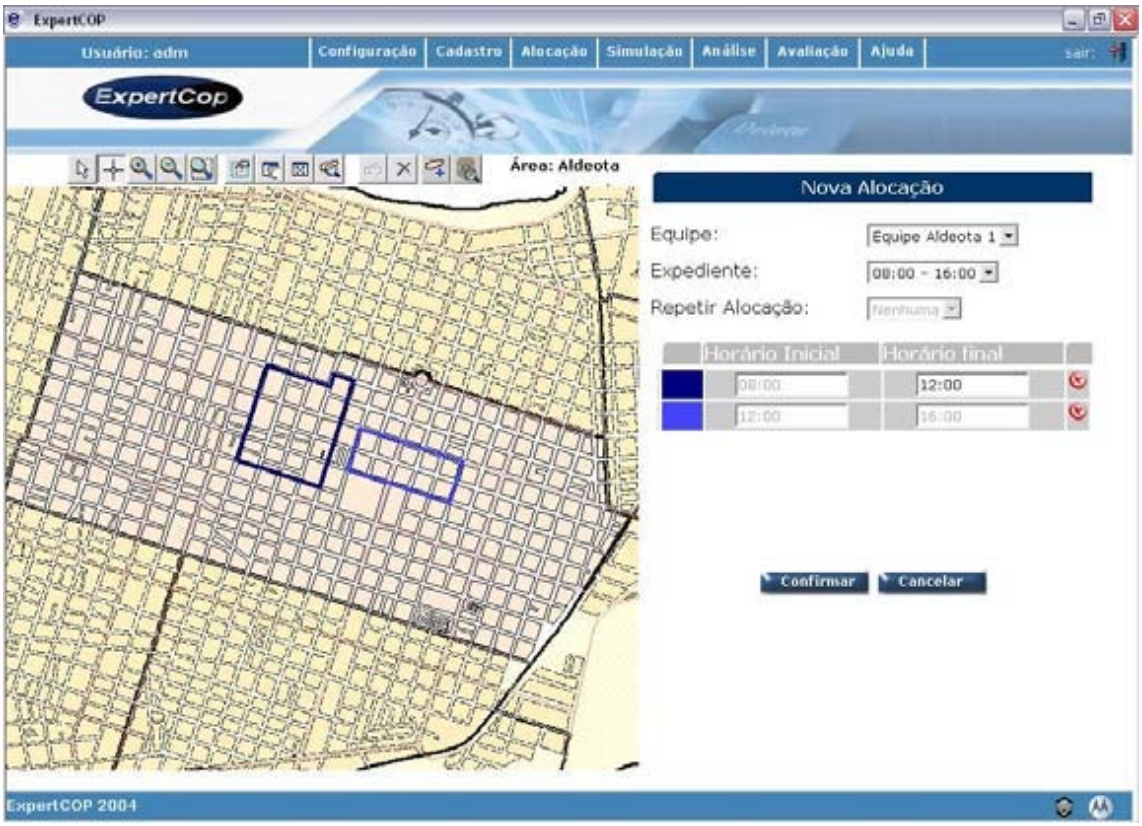


Figura 2.6: *Interface de alocação das viaturas policiais.*

Na interface apresentada na Figura 2.6, é possível definir as rotas das viaturas com seus respectivos horários através do GIS. Nota-se que, cada viatura policial possui um expediente de trabalho, normalmente 8 horas.

Após a simulação, a interface apresentada na Figura 2.7 ajuda a análise dos motivos que levaram aos crimes.

A análise dos motivos que levaram aos crimes fazem com que os estudantes mudem suas crenças a respeito de como as rotas devem ser definidas. Dessa forma, as próximas alocações, feitas pelos estudantes, possivelmente serão melhores que as anteriores.

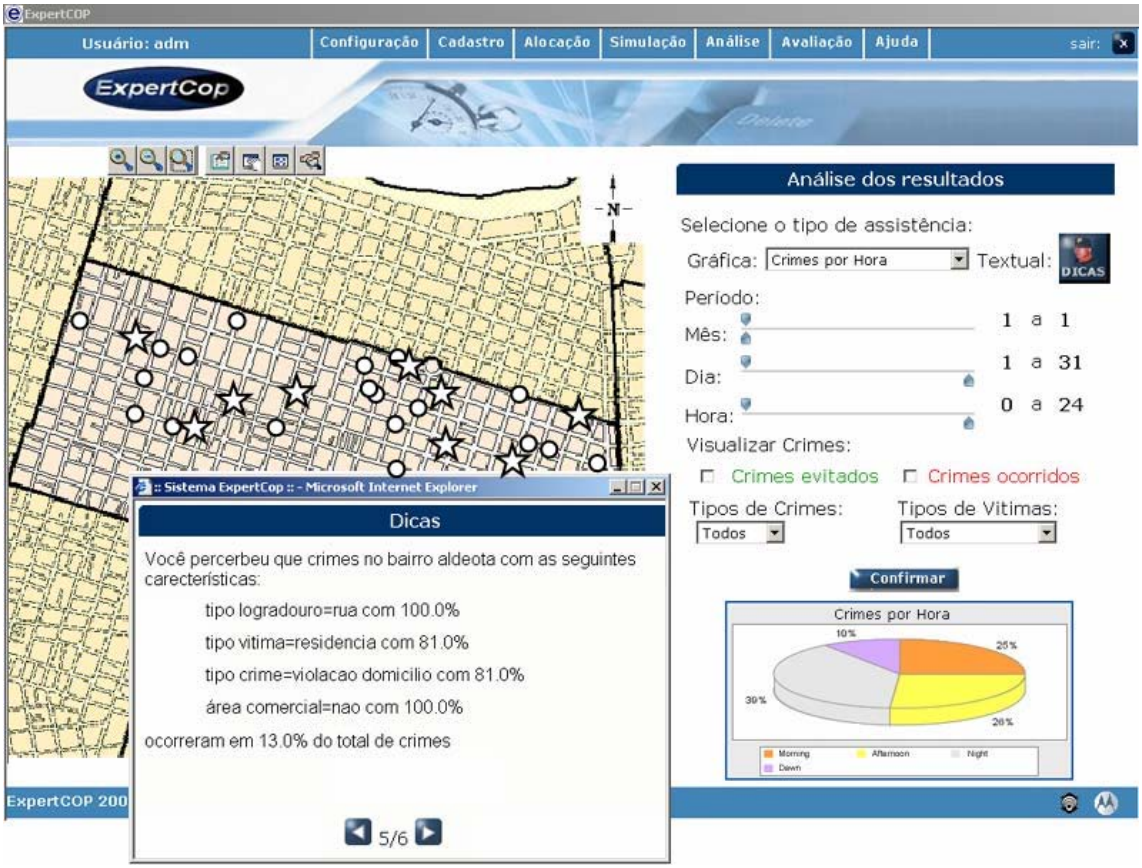


Figura 2.7: *Interface de análise e visualização. As cores (verde e vermelho) da interface foram modificadas por motivo de impressão.*

Na Figura 2.7, os pontos marcados por círculos representam os crimes ocorridos e os pontos marcados por estrelas os crimes evitados. Ao selecionar qualquer um desses pontos, o agente pedagógico apresenta a explicação dos motivos que levaram, ou não, ao crime.

Um experimento feito com noventa profissionais da área de segurança pública, mostrou que policiais militares com prática na tarefa de alocação apresentaram baixa modificações em suas crenças. Enquanto que, estudantes com pouca experiência alteraram fortemente suas crenças.

Talvez esse comportamento da mudança das crenças dos estudantes mostre que os agentes criminosos possuem uma boa representação do que realmente são os criminosos humanos.

2.4 Robocup Rescue

Robocup Rescue [19] é um simulador de desastres que incentiva o desenvolvimento de agentes heterogêneos que cooperam tentando socorrer as vítimas e refugiados

do desastre.

Dentre os variados tipos de agentes existem os: policiais, bombeiros, ambulâncias e pessoas.

A Figura 2.8 apresenta o simulador Robocup Rescue. Nessa figura, existem casas e prédios representados por retângulos e as vias por onde as viaturas de bombeiros e policiais circulam.

Figura 2.8: *Interface gráfica do simulador Robocup Rescue.*

Em caso de incêndio, as cores das casas e prédios apresentadas na Figura 2.8 informam a gravidade do problema.

Nesse simulador, o incêndio se espalha como acontece no mundo real, por isso o quanto antes uma viatura de bombeiros chega, mais rápido o incêndio será contido.

Ao final de uma simulação, uma nota é dada ao comportamento dos agentes (bombeiros, policiais, etc). Muitos fatores influenciam a avaliação do simulador, dentre eles, a condição das vítimas e das construções.

Devido à avaliação do simulador, os desenvolvedores de agentes criaram competições buscando aprimorar os comportamentos dos agentes.

Este trabalho pode ajudar a posicionar estrategicamente as viaturas policiais e de bombeiros numa simulação do Robocup Rescue.

2.5 DEFACTO

O sistema DEFACTO (*Demonstration Effective Flexible Agent Coordination of Teams through Omnipresence*) [16] apresenta um ambiente tridimensional onde o usuário acompanha os acontecimentos numa cidade.

A Figura 2.9 apresenta a aplicação do sistema DEFACTO numa cidade com ocorrências de incêndio.

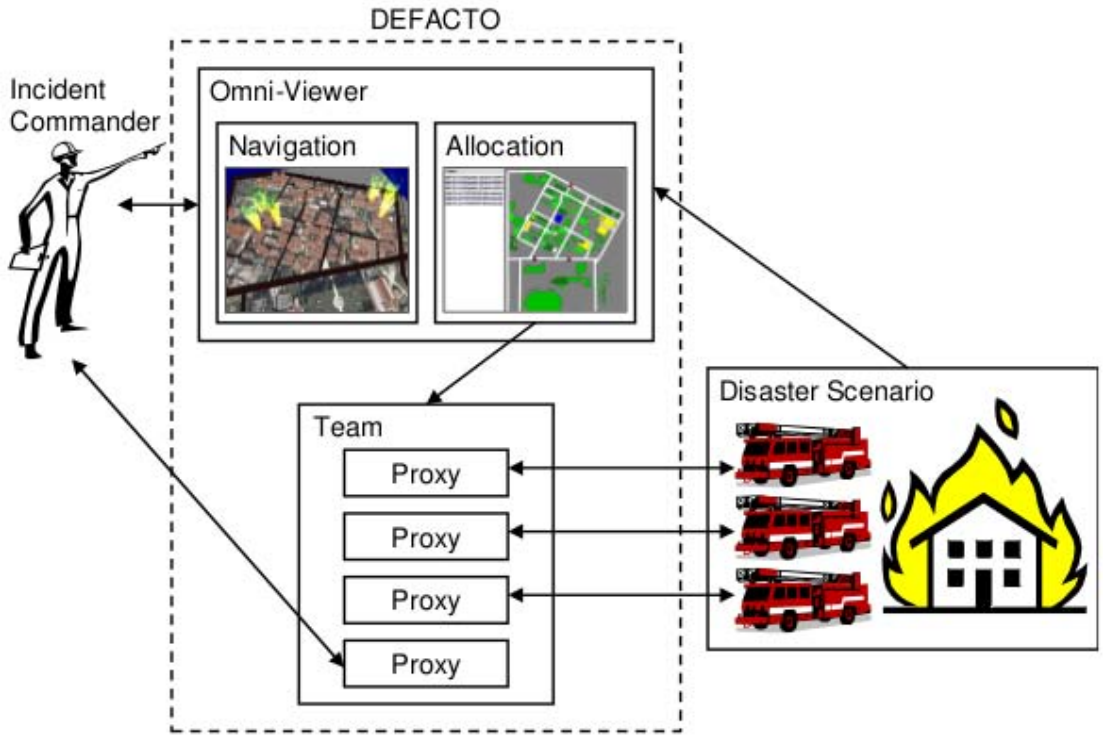


Figura 2.9: *Interface gráfica do sistema Defacto.*

Nesse sistema, uma pessoa age orientando as viaturas de bombeiros sobre a sequência das ocorrências que devem ser atendidas. Para facilitar a visualização do cenário, o ambiente é representado em três dimensões e aparecem chamas nos lugares das ocorrências. Como o simulador Robocup Rescue, apresentado na seção 2.4, o sistema DEFACTO também é um simulador, mas nele só existe o agente socorrista de bombeiros. Nesse sistema, o operador regula o grau de autonomia dos agentes. Dessa forma, o operador pode dar total ou nenhuma autonomia aos agentes.

2.6 Posicionamento Inicial de Ambulâncias

Brotcorne et al [2] apresentam uma revisão do problema do posicionamento de ambulâncias utilizando planos de posicionamento e análise probabilística. Eles representam a área de atendimento com um grafo. Nesse grafo, os vértices representam a demanda e os lugares das ambulâncias são representados por vértices numa variável chamada W.

O tempo de movimento de um vértice i para um vértice j é previamente conhecido e representado por t_{ij}. Um vértice i é considerado coberto, se e somente se, existe uma ambulância no ponto j e $t_{ij} < r$, onde r é um fator de ajuste pré-determinado.

Nesse trabalho, foram citados os modelos *Location Set Covering Model* (LSCM) e *Maximal Covering Location Problem* (MCLP) [3]. O primeiro determina o número de ambulâncias necessárias para atender toda demanda, enquanto que, o segundo tenta fazer o melhor uso possível das ambulâncias disponíveis.

Nesse capítulo foram apresentados trabalhos relacionados ao problema da distribuição. O capítulo a seguir apresenta um estudo detalhado do problema que pretende-se resolver.

CAPÍTULO 3

O Problema

Este capítulo apresenta formulações matemáticas para o problema da distribuição de recursos e uma forma de avaliar as distribuições. Com esse estudo, é possível resolver o problema e averiguar se as soluções foram boas ou ruins.

3.1 Representação e Métricas

Em essência, o problema possui duas variáveis: uma área geográfica e uma quantidade k de viaturas.

Até esse momento, pode-se criar inúmeras variações do problema em função da forma de representação da área geográfica de atendimento. As formas mais comuns de representação da área geográfica são vetores, matrizes e grafos.

Nesse trabalho, será tratada a forma de representação da área geográfica de atendimento por uma matriz M de n linhas e m colunas. O peso da célula indica o quanto é necessário um recurso naquele local.

A somatória de todos os pesos da matriz deve ser 100%, ou simplesmente 100. Essa definição obriga que os pesos da matriz estejam relacionados entre si. A expressão 3-1 apresenta esta definição:

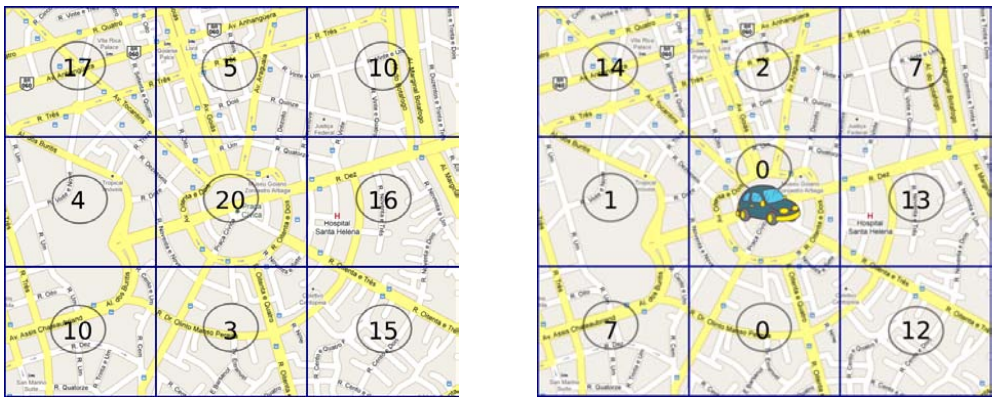
$$\sum_{i=1}^{i=n} \sum_{j=1}^{j=m} M(i,j) = 100 \tag{3-1}$$

Para avaliar uma distribuição é necessária uma função $f : (M,S) \rightarrow AV$, onde S é a posição de todas as viaturas no cenário e AV é a nota total da avaliação.

Cada organização pode definir sua própria função de avaliação, mas essa função deve seguir a seguinte regra: quando não houver recurso no ambiente a nota deve ser 0 (zero) e, quando houver pelo menos um recurso em cada célula da matriz M, a nota deve ser 1.

Para criação da função de avaliação, pode-se pensar que, à medida que recursos são adicionados na área de atendimento, os pesos das células da matriz M diminuem.

Por exemplo, a Figura 3.1 (a) apresenta uma matriz M com os pesos em cada célula, a soma destes pesos é igual a 100, obedecendo a restrição apresentada na expressão 3-1. Já na Figura 3.1 (b), os pesos da matriz M foram influenciados por causa do recurso alocado na célula (2, 2).



(a) *Matriz M que representa a área de atendimento com os pesos em cada célula.*

(b) *Matriz M com os pesos afetados pelo recurso distribuído na célula (2, 2).*

Figura 3.1: *Exemplo de influência de um recurso na matriz M.*

No exemplo apresentado na Figura 3.1 (b), o peso da célula onde o recurso foi alocado foi reduzido a zero, enquanto que cada célula adjacente teve seu peso subtraído de 3 unidades.

Nota-se que a forma como o recurso interfere na necessidade de outro recurso pode variar em cada área de aplicação do problema. No entanto, espera-se que muitos casos tenham o comportamento que será apresentado na seção 3.3.1.

Para calcular a nota de uma distribuição, pode-se calcular a diferença entre a somatória dos pesos da matriz original M e a matriz com os pesos afetados pelos recursos distribuídos, que será chamada de M'. Essa diferença, indica o quanto foi reduzido da necessidade inicial, mediante o posicionamento dos recursos. A expressão 3-2 apresenta o cálculo desta diferença.

$$AV = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} M(i,j) - M'(i,j) \tag{3-2}$$

No caso do exemplo apresentado na Figura 3.1, a nota seria: $17-14+5-2+10-7+4-1+20-0+16-13+10-7+3-0+15-12$. Desta forma, a nota da distribuição feita na Figura 3.1 (b) seria 44.

Sendo assim, o problema contém três entradas:

- M: matriz com os pesos que determinam a necessidade de um recurso;
- k: quantidade de recursos disponíveis;

- f: função de avaliação da distribuição criada.

Nesse problema, deseja-se obter a distribuição com melhor avaliação possível utilizando todos os k recursos, ou seja, pretende-se maximizar o resultado da função de avaliação $f(M,S)$, onde M é a matriz com os pesos e S é um conjunto que contém as posições dos recursos, resultante do algoritmo de distribuição.

3.2 Montando a Matriz M

Esta seção propõe uma forma da definição dos pesos da matriz M, em função da probabilidade da subárea de atendimento necessitar do recurso.

Nesta seção, será utilizado o domínio das companhias de fornecimento de energia elétrica e a área de uma parte da cidade de Goiânia como "estudo de caso".

A Figura 3.2 apresenta a área geográfica que servirá de exemplo para as demonstrações. Nota-se que a área geográfica deve possuir formato retangular porque é representada por uma matriz.

Figura 3.2: *Mapa de uma subárea da região central da cidade de Goiânia.*

Na área geográfica apresentada na Figura 3.2, acontecem problemas elétricos que geram ocorrências a serem resolvidas. Para resolução dessas ocorrências, existem viaturas que se locomovem até o lugar da ocorrência, verificam qual o problema elétrico e o resolve.

Diariamente, essas viaturas são posicionadas na área geográfica levando em consideração fatores ambientais e experiências dos recursos. No entanto, o posicionamento é feito manualmente, de maneira não uniforme, não ótimo e nem sempre segue o mesmo processo.

Devido à necessidade de locomoção das viaturas até as ocorrências, quanto mais próxima uma viatura estiver da ocorrência, menor será a distância percorrida pela viatura e mais rápido será a resolução da ocorrência, pois as concessionárias podem ser penalizadas se o tempo para restabelecer o sistema for elevado. Por isto, um bom planejamento das posições iniciais das viaturas na área geográfica é fundamental para uma boa qualidade de atendimento.

As seções seguintes apresentam como dividir a área de atendimento e a forma de se calcular as probabilidades de uma subárea necessitar de uma viatura.

3.2.1 Estratégia da Divisão e Conquista

Como apresentado na seção 3.1, a área geográfica é dividida em $n \cdot m$ partes que serão subáreas de atendimento.

Determinar a quantidade de partes da divisão também é uma tarefa complexa e está diretamente relacionada à qualidade da distribuição das viaturas no cenário. No entanto, este estudo foge do escopo deste trabalho. A Figura 3.3 apresenta a área geográfica de exemplo dividida numa matriz de três linhas e três colunas, ou seja, $n = 3$ e $m = 3$.

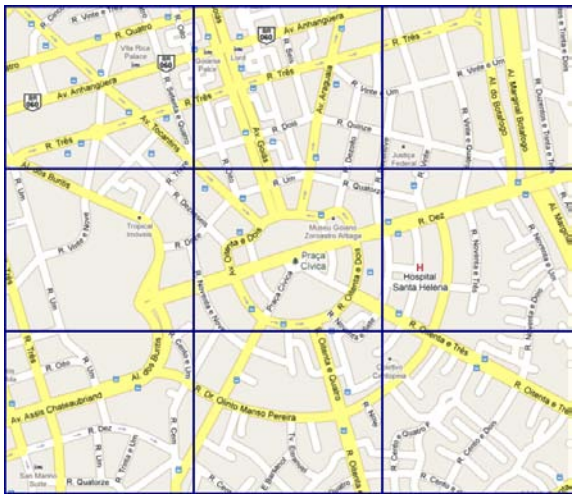


Figura 3.3: *Área de atendimento dividida numa matriz 3 por 3.*

3.2.2 Probabilidade de Ocorrer um Problema

Existem várias formas de determinar a probabilidade de ocorrer um problema numa subárea de atendimento. Normalmente, o que varia nessas formas de determinar a probabilidade é a quantidade de fatores que ela utiliza para o cálculo e a quantidade de informações sobre a subárea de atendimento.

Por exemplo, uma forma de determinar essa probabilidade pode levar em conta a quantidade de ocorrências na subárea no mês anterior. Também pode-se utilizar esse

fator juntamente com a quantidade de árvores na subárea. Da mesma maneira, poderia-se utilizar a previsão de chuva, ventos e raios na subárea. Sendo assim, é possível derivar várias maneiras de determinar essa probabilidade.

Além da quantidade de fatores, pode-se utilizar de informações sobre a subárea de atendimento. No caso das companhias elétricas, poderiam ser utilizadas informações como: está chovendo, está nublado e faz sol. Essas informações poderiam ser utilizadas como evidências numa Rede Bayesiana [13, página 447] para o cálculo da probabilidade condicional [13, páginas 452 e 457].

A forma como a probabilidade é calculada influencia diretamente a qualidade da distribuição e por isto devem ser estudadas várias formas de cálculo da probabilidade para cada domínio onde o problema da distribuição acontece.

3.2.3 Tratando Probabilidades

Para expressar a necessidade de um recurso numa subárea em relação a outra, é necessário submeter as probabilidades de cada subárea, já calculadas, à definição matemática 3-1, feita na seção 3.1.

Por exemplo, a Figura 3.4 apresenta a área geográfica dividida em nove subáreas. Cada subárea possui a probabilidade de ocorrer um problema num determinado dia.

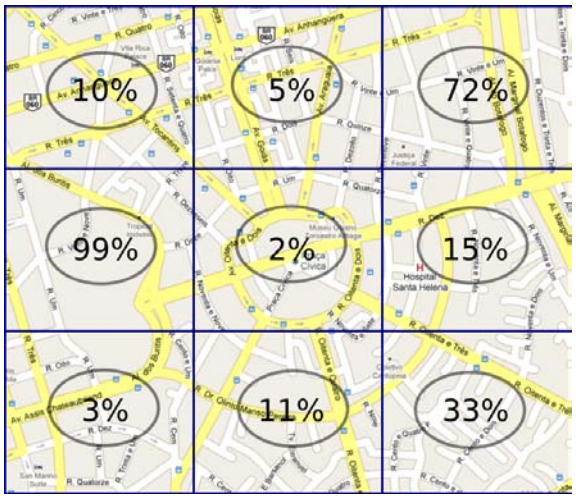


Figura 3.4: *Área de atendimento com probabilidades calculadas para cada subárea de forma não relativa.*

Na Figura 3.4, o somatório das probabilidades de cada célula da matriz é superior a 100. Neste caso, a soma de $10+5+72+99+2+15+3+11+33$ é igual a 250. Por isto a definição 3-1, feita na seção 3.1, foi violada.

Devido à violação da definição 3-1, deve-se fazer um segundo cálculo para tornar as probabilidades relativas entre si.

Para facilitar o entendimento, a probabilidade de ocorrer um problema na subárea (1, 1), que na Figura 3.4 é 10%, será chamada de $p_{(1,1)}$ e da subárea (i, j), será chamada de $p_{(i,j)}$.

Para iniciar a conversão é necessário efetuar a soma de todas as probabilidades:

$$ProbTotal = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} p_{(i,j)} \tag{3-3}$$

No caso da Figura 3.4, a soma das probabilidade é 250, ou seja, $ProbTotal = 250$.

Em seguida, é necessário recalcular a probabilidade de cada célula tornando-a relativa às outras. Esta nova probabilidade será chamada de $P_{(i,j)}$:

$$P_{(i,j)} = \frac{p_{(i,j)}}{ProbTotal} \tag{3-4}$$

Desta forma, as probabilidades exibidas na Figura 3.4 ficaram iguais ao da Figura 3.5 após a conversão.

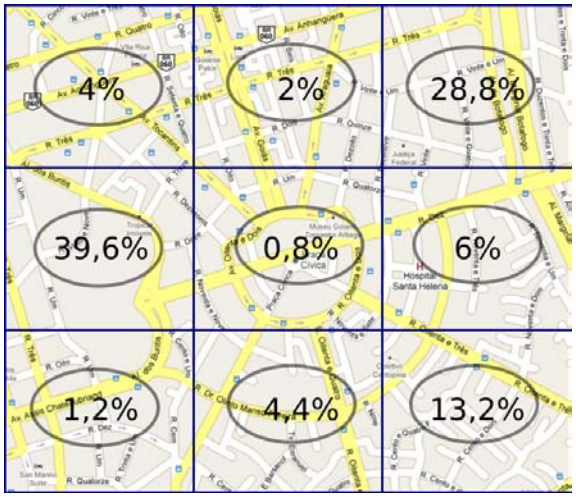


Figura 3.5: *Área de atendimento com probabilidades relativas entre si.*

Ao realizar a soma das probabilidades $4+2+28.8+39.6+0.8+6+1.2+4.4+13.2$, tem-se o resultado 100, respeitando a definição 3-1.

Desta forma, a probabilidade 4% da célula (1, 1) é duas vezes maior que a probabilidade 2% da célula (1, 2). Em outras palavras, a necessidade do recurso na subárea de atendimento (1, 1) é duas vezes maior que na subárea (1, 2).

3.2.4 Probabilidade Calculada pelo Histórico

Nas companhias elétricas, pode-se montar a matriz M por meio da análise do histórico de ocorrências de cada subárea de atendimento. Para isto, os dados do mês

anterior ou do mesmo mês no ano anterior, podem ser utilizados.

A Figura 3.6 apresenta a área geográfica dividida em nove subáreas com a quantidade de ocorrências retiradas do histórico da subárea.

Figura 3.6: *Área de atendimento com a quantidade de ocorrências do histórico.*

A quantidade de ocorrências na subárea (i, j) será representada por $O_{(i,j)}$. O primeiro passo para calcular a probabilidade é somar todas as ocorrências:

$$TotalOcorrencias = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} O_{(i,j)} \tag{3-5}$$

Neste caso, o total de ocorrências é $10+15+20+40+5+30+3+4+12$, ou seja, $TotalOcorrencias = 139$. O segundo passo é calcular a probabilidade de acontecer um problema em cada subárea:

$$P_{(i,j)} = \frac{O_{(i,j)}}{TotalOcorrencias} \tag{3-6}$$

A Figura 3.7 apresenta as nove subáreas e a probabilidade de ocorrer um problema em cada subárea após o cálculo.

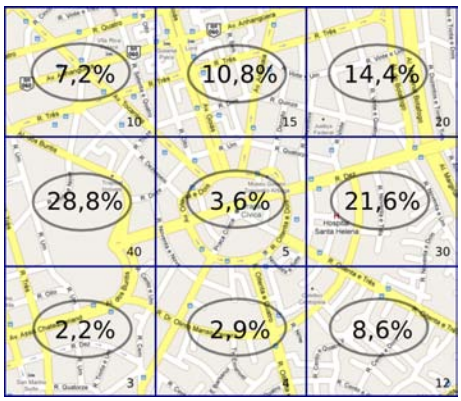


Figura 3.7: *Área de atendimento com a probabilidade relativa a quantidade de ocorrências do histórico.*

Neste caso, a probabilidade de cada subárea de atendimento é relativa às outras subáreas. Por isto, não é necessária a normalização apresentada na subseção 3.2.3.

Somando as probabilidades de cada subárea, ou seja, $7,2+10,8+14,4+28,8+3,6+21,6+2,2+2,9+8,6$ que é igual a $100,1$. Trabalhando com mais casas decimais, este somatório é igual a 100, seguindo a definição 3-1 da seção 3.1.

3.3 Função de Avaliação

A criação da função de avaliação está diretamente ligada ao modo como os recursos influenciam a área de atendimento. Por isto, a seção 3.3.1 e a seção 3.3.3 apresentam duas formas de influência do recurso na área de atendimento.

3.3.1 Influência por Hipérbole

Alocar um recurso numa subárea de atendimento influencia diretamente o cenário, pois em alguns casos, evita-se colocar outro recurso no mesmo local ou nas subáreas adjacentes. A influência de um recurso deve atingir todo o cenário, mesmo quando esse está distante, onde a influência tende a zero. Essa influência será chamada de "fator de proteção".

O fator de proteção de uma subárea (i, j) será associado a função $PRO_{(i,j)}$. Antes de definir a função $PRO_{(i,j)}$, é necessário definir a influência de uma viatura no cenário.

Por exemplo, ao colocar um recurso na subárea (2, 3), como apresentado na Figura 3.8, deve-se definir a sua influência no resto do cenário.

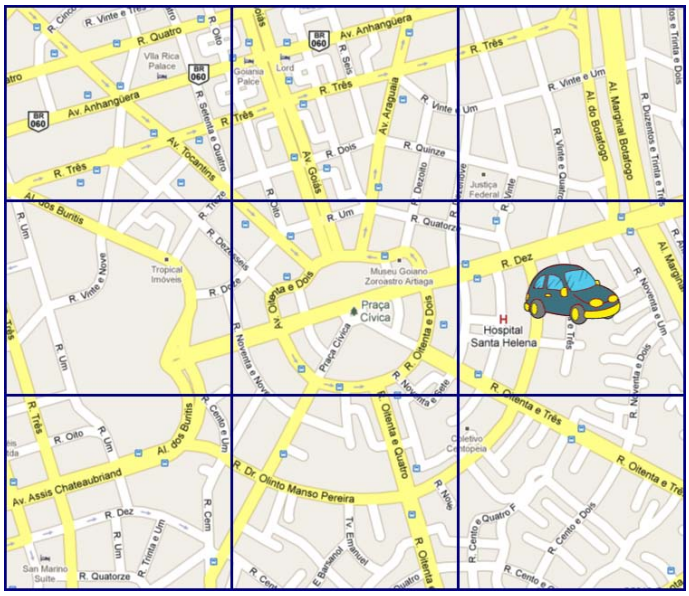


Figura 3.8: *Influência de uma viatura no cenário.*

Dado que, quando não há viaturas no cenário, as proteções de cada subárea são iguais a zero, uma forma simples de representar a proteção feita por um recurso é:

- a proteção é 1, ou seja, 100% na subárea onde o recurso foi alocado;
- a proteção é acrescida de $\frac{1}{2^d}$, onde d é a distância de vizinhança entre o recurso e a subárea, ou seja, quanto mais distante, menor será a proteção exercida pelo recurso.

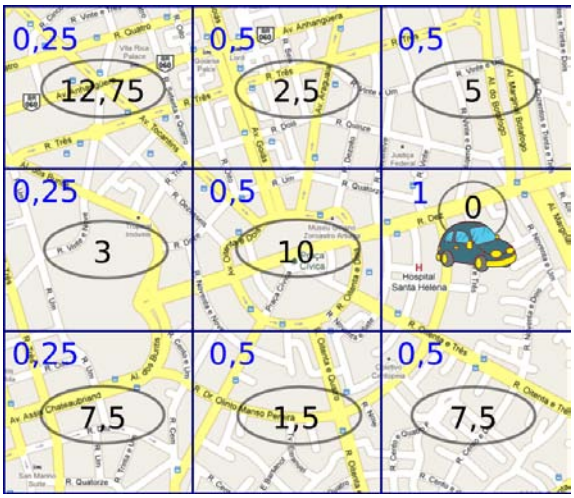
Esta função de influência foi chamada de hiperbólica devido à forma como a proteção reduz em função da distância.

A Figura 3.9 apresenta o fator de proteção, calculado utilizando esta definição. Nota-se que o fator de proteção de todas as subáreas no início é zero.

Figura 3.9: *Fator de proteção calculado em cada subárea de atendimento.*

As proteções indicam o quanto, em percentual, deve-se reduzir dos pesos da matriz M. Por exemplo, a Figura 3.10 (a) apresenta a matriz M com seus respectivos pesos. Após o posicionamento do recurso na posição (2, 3), o fator de proteção indica o percentual de quanto será retirado do peso naquela subárea.

(a) *Matriz M com os pesos em cada subárea.*



(b) *Matriz M com os pesos afetados pelo recurso distribuído na célula (2, 3).*

Figura 3.10: *Influência do recurso na matriz M.*

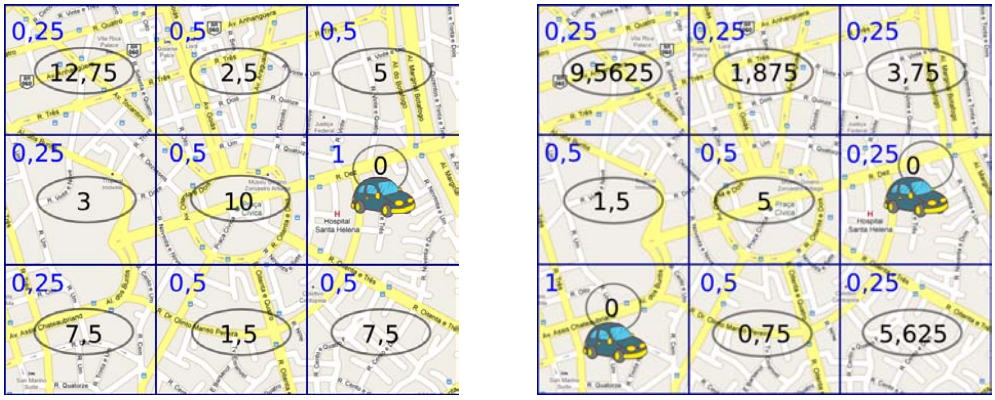
Na Figura 3.10 (b), o números não circulados são as proteções e os números circulados são os pesos das subáreas.

Na subárea (1, 1) da Figura 3.10 (b), a proteção igual 0,25 indica que 25% do peso da subárea (1, 1) será retirado. Como o peso da subárea (1, 1), na Figura 3.10 (a), era 17, então o peso final será: $(1-0,25)\cdot 17$, ou seja, apenas 75% do peso irá continuar.

Casos como da reparação da rede elétrica exigem comunicação entre as viaturas que por meio do Centro de Operação e Distribuição (COD) evitam de ligar linhas elétricas enquanto outras pessoas estão trabalhando nelas. Nesses casos, o gráfico do tempo em função da quantidade de recursos é uma hipérbole [6].

Em outras palavras, uma viatura poderia resolver dois problemas em duas horas. No entanto, duas viaturas não resolvem os mesmos dois problemas em uma hora porque existe a necessidade de comunicação entre elas para execução do serviço.

Devido a este tipo de comportamento do domínio das Companhias Elétricas, propõe-se que, quando adicionado o segundo recurso, a proteção feita por ele influencie os pesos da matriz corrente, que já foi modificada pela influência do primeiro recurso alocado. A Figura 3.11 é a continuação da distribuição iniciada na Figura 3.10.



(a) *Matriz M com os pesos afetados pelo recurso distribuído na célula (2, 3).*

(b) *Matriz M com os pesos afetados pelo segundo recurso distribuído na célula (3, 1).*

Figura 3.11: *Influência do segundo recurso na matriz M.*

A Figura 3.11 (a) apresenta a matriz M modificada após a inclusão do primeiro recurso. Então, quando o segundo recurso é adicionado, na Figura 3.11 (b), sua influência afeta a matriz M corrente.

Dessa forma, quando o segundo recurso é adicionado na subárea (3, 1), como apresentado na Figura 3.11 (b), o peso da subárea (1, 1), que na Figura 3.11 (a) era 12,75, será reduzido em 25%, ou seja, o peso final será $(1-0,25)\cdot 12,75$.

Além de representar a necessidade de comunicação, esse método de influência não permite que todos os recursos sejam adicionados num só ponto de maior influência.

Para formalizar, a função que determina o fator de proteção individual, proposta no início desta seção, será chamada de $h(d)$, onde d é a distância entre o recurso e a subárea. Desta forma, tem-se que:

$$h(d) = \begin{cases} 1 & \text{se} \quad d = 0 \\ \frac{1}{2^d} & \text{se} \quad d > 0 \end{cases} \tag{3-7}$$

Como 2^0 é 1, então a função $h(d)$ pode ser simplificada em:

$$h(d) = \frac{1}{2^d}, d \geqslant 0 \; e \; d \in \mathbb{N} \tag{3-8}$$

Se colocada num gráfico, a função $h(d)$ é uma hipérbole. A Figura 3.12 apresenta o gráfico da função $h(d)$.

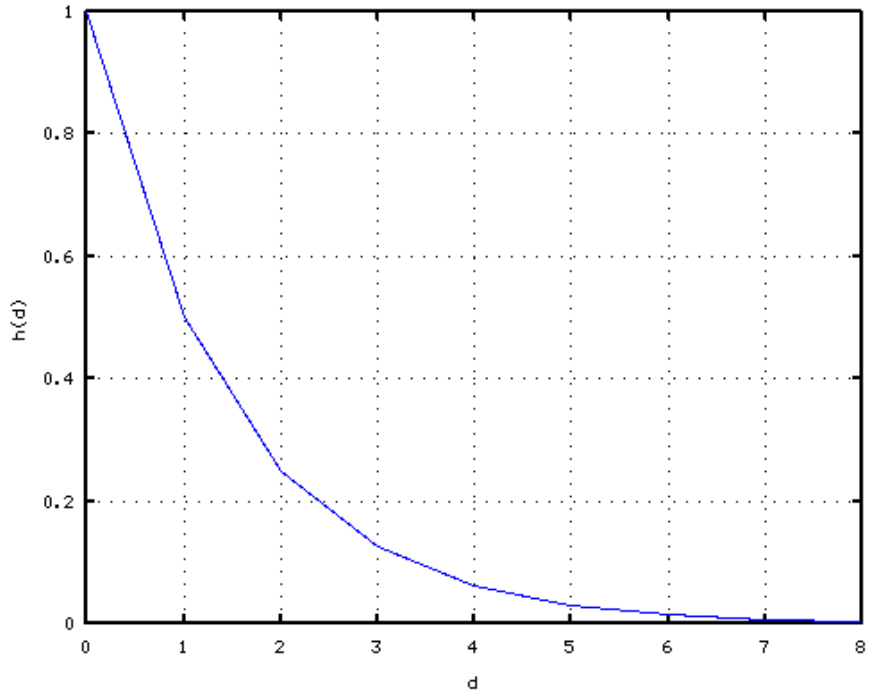


Figura 3.12: *Gráfico da função* $h(d)$.

Para calcular o fator de proteção individual, a função $h(d)$ necessita de um argumento que é, a distância entre o recurso e a subárea. Neste trabalho, será utilizada como distância, a quantidade de células, da matriz M, nas quais, deve-se passar até que se chegue a uma outra célula.

Desta forma, a função a seguir calcula a distância que um determinado recurso, posicionado na célula $(x1,\ y1)$, deve percorrer, para chegar em uma célula, localizada em $(x2,\ y2)$.

$$d(x1,\ y1,\ x2,\ y2) = max(|x1 - x2|, |y1 - y2|) \tag{3-9}$$

Dado que, todo recurso possui as informações de suas coordenadas geográficas, $(x,\ y)$, então pode-se criar a seguinte função:

$$dr(recurso, x, y) = d(recurso.x, recurso.y, x, y) \tag{3-10}$$

Finalmente, se existem K recursos e R é o vetor de recursos, para calcular a proteção numa subárea utiliza-se a expressão:

$$PRO_{(i,j)} = 1 - \prod_{l=1}^{l=K} \Big(1 - h(dr(R_l, i, j))\Big) \tag{3-11}$$

O produtório da função $PRO_{(i,j)}$ representa as sucessivas subtrações de pesos na matriz M ao adicionar um novo recurso, como apresentado na Figura 3.11. A expressão 3-

11 representa os cálculos feitos nas figuras 3.10 e 3.11. As figuras 3.13 e 3.14 apresentam como este cálculo é realizado num único passo.

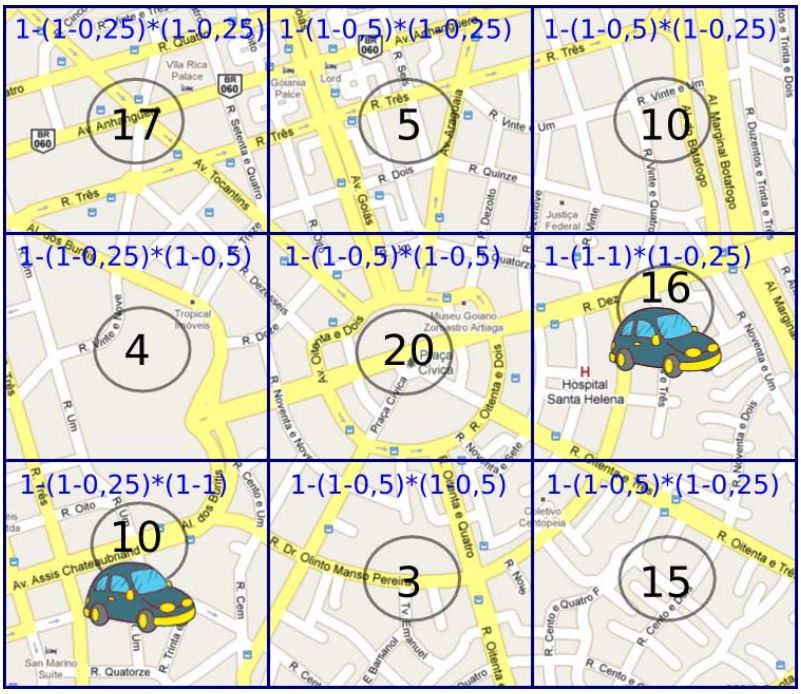


Figura 3.13: *Matriz M com os pesos iniciais e os dois recursos posicionados.*

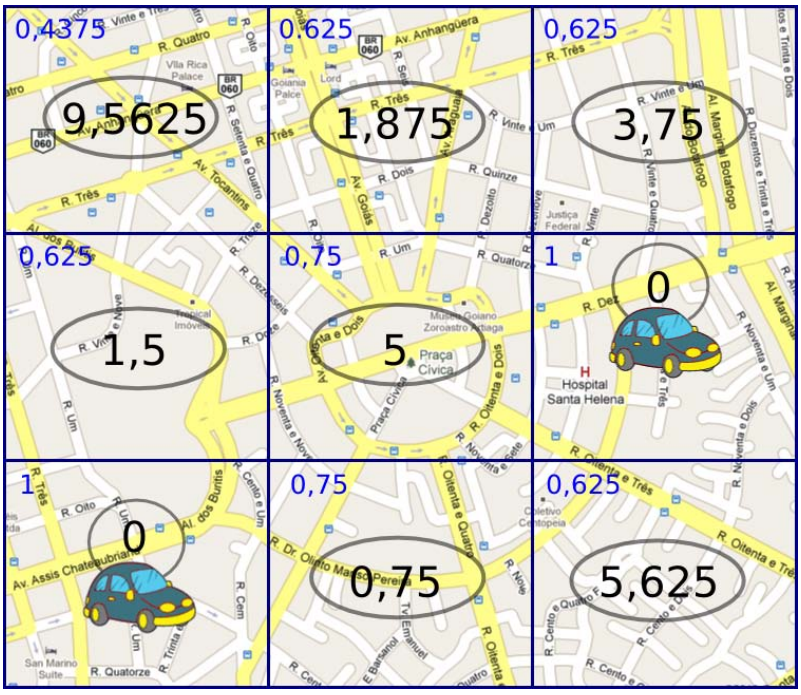


Figura 3.14: *Matriz M com os pesos afetados e a influência das duas viaturas calculada.*

Com a expressão de proteção 3-11 é possível descobrir a proteção final de uma

subárea. Por exemplo, na Figura 3.14 a proteção final da subárea (1, 1) é 43,75%, ou seja, será reduzido 43,75% do peso inicial devido a essa distribuição. Como o peso inicial da subárea (1, 1) era 17 na Figura 3.13, então o peso final é: $(1-0.4375)\cdot 17$.

Nota-se que os pesos finais calculados através da expressão 3-11 da matriz M apresentada na Figura 3.14 são iguais aos pesos calculados passo a passo da matriz M apresentada na Figura 3.11 (b).

3.3.2 A Função de Avaliação

A função de avaliação deve ser composta pelas variáveis: peso e proteção de cada subárea. Uma forma de representar a função de avaliação é:

$$fa = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} P_{(i,j)} \cdot PRO_{(i,j)} \tag{3-12}$$

Essa função é uma outra forma de calcular a diferença de pesos da matriz inicial e da matriz final, já apresentada na expressão 3-2.

Desta forma, uma solução ótima tenta maximar fa, ou seja, $solucao\_otima \leftarrow max(fa)$.

3.3.3 Influência Linear

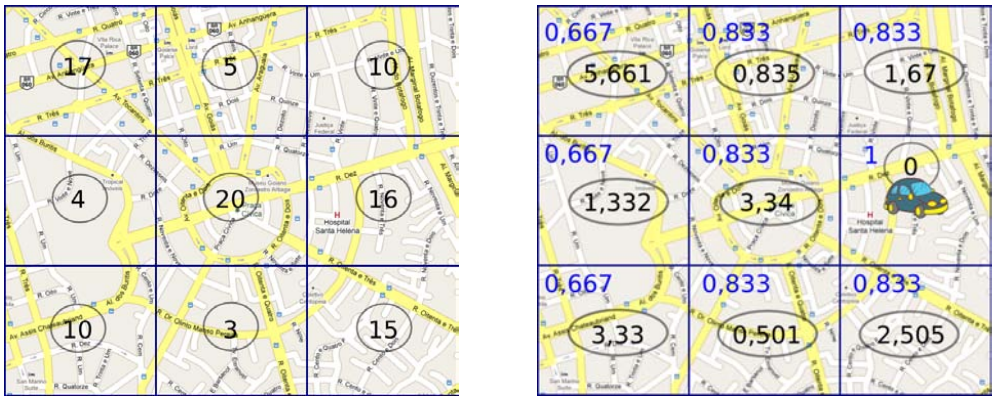
Há casos em que não existe necessidade de comunicação entre os recursos para resolução dos problemas. Por exemplo, no posicionamento de vendedores de cosméticos. Nesses casos, a influência do recurso pode ser expressa de forma linear. Essa influência é dita linear porque reduz linearmente o grau de proteção em função da distância.

É interessante utilizar um raio máximo de influência, já que a proteção é expressa em uma função decrescente. A função a seguir, define o raio máximo de influência igual a 6, devido a estudos que serão apresentados na seção 4.5:

$$h(d) = \begin{cases} 0 & \text{se} \quad d \geqslant 6 \\ 1 - \frac{d}{6} & \text{se} \quad d < 6 \end{cases} \tag{3-13}$$

Na expressão 3-13, o dividendo da variável d, nesse caso 6, especifica em que distância a influência é igual a zero. Nota-se que, o número 6 é o limite de influência. Esse número pode variar de acordo a precisão de cálculo requerida.

Neste exemplo, quando a distância for maior ou igual a seis, não haverá mais influência. A Figura 3.15 apresenta a influência linear do recurso distribuído na subárea (2, 3) numa matriz M.



(a) *Matriz M com os pesos em cada subárea.*

(b) *Matriz M com os pesos afetados pelo recurso distribuído na célula (2, 3).*

Figura 3.15: *Influência linear do recurso na matriz M.*

Na Figura 3.15 (b), a proteção da subárea (1, 1) é 66,7%. Como o peso dessa subárea era 17 na Figura 3.15 (a), então para calcular o peso final deve-se realizar o cálculo: $(1 - 0,667) \cdot 17$.

Quando um segundo recurso é adicionado, segue-se os mesmos cálculos apresentados na seção 3.3.1. A função de avaliação apresentada na seção 3.3.2 também pode ser utilizada para obter a nota de uma distribuição.

3.3.4 Considerações Finais do Capítulo

Neste capítulo foram apresentadas duas formas de definição da função de influência, são elas: hiperbólica e linear.

No entanto, é possível definir várias outras funções de influência, tais como: quadráticas, exponenciais, entre outras. Na seção 3.3.3, foi citado que, é interessante utilizar um raio de influência porque diminui a quantidade de cálculos necessários num algoritmo, mais detalhes serão apresentados na seção 4.5.

Neste capítulo foram apresentadas expressões que definem o problema da distribuição e como avaliar uma distribuição feita. O capítulo a seguir apresenta algoritmos criados para resolução do problema, detalhado neste capítulo.

CAPÍTULO 4

Resolução do Problema

Este capítulo apresenta os algoritmos criados para resolução do problema da distribuição de recursos numa área geográfica. Nos exemplos, será utilizado o domínio das companhias elétricas.

Como comentado na seção 3.1, do Capítulo 3, podem haver várias formas de representação da área de atendimento, desde grafos até matrizes. Neste trabalho, é feito um estudo relacionado à representação por uma matriz.

4.1 Representação por Matriz da Área de Atendimento

A representação por matriz consiste na divisão da região de atendimento numa matriz de *n* linhas e *m* colunas. A Figura 4.1 apresenta a Região Metropolitana de Goiânia dividida numa matriz.

Figura 4.1: *Exemplo de divisão da área de atendimento.*

Para visualização de representações por matriz, foi criado um *software* em JavaFX [1] que exibe os pesos de cada célula da matriz, seguindo o modelo apresentado na seção 3.1. Após a exibição dos pesos, o *software* criado é capaz de apresentar os resultados dos algoritmos de distribuição como apresentado na Figura 4.2.

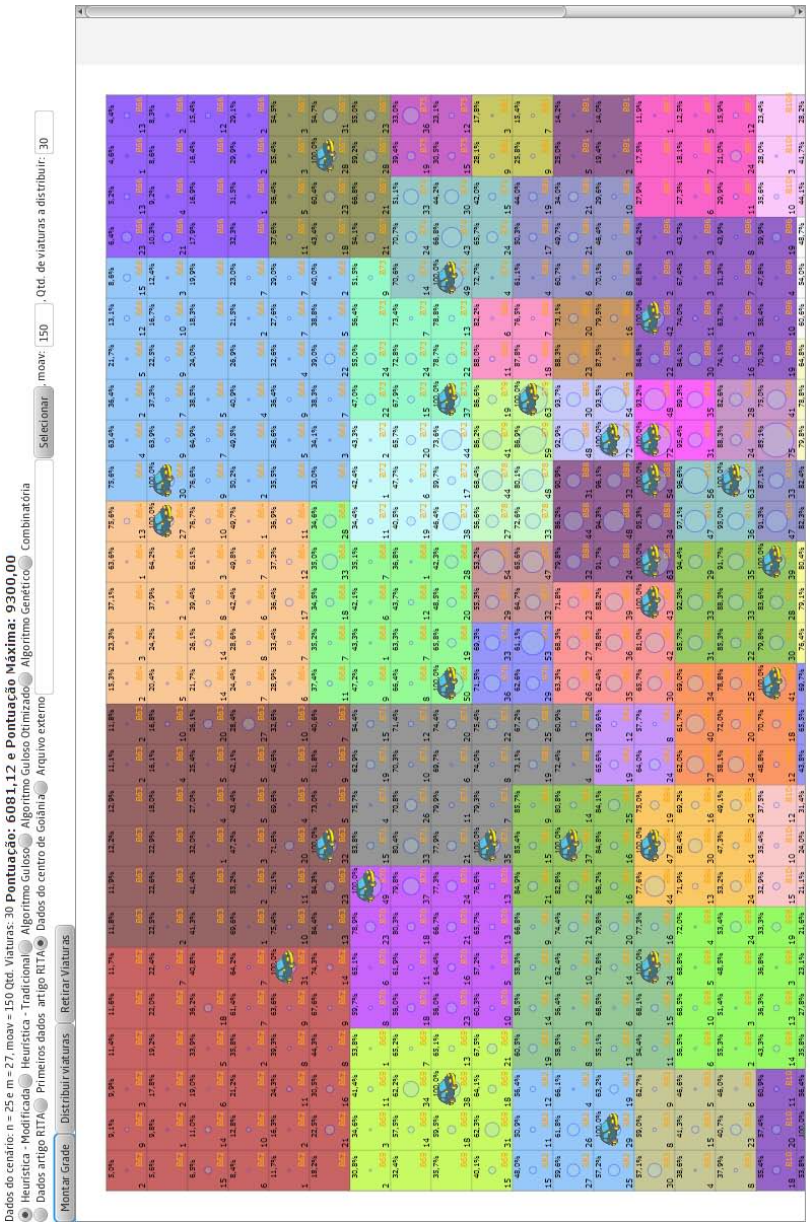


Figura 4.2: *Ferramenta desenvolvida para o desenho da representação por matriz.*

Na Figura 4.2, o centro de Goiânia foi dividido em blocos e distribuiu-se os recursos de acordo com o peso total de cada bloco. Essa heurística será apresentada com mais detalhes na seção 4.7. Também foi possível efetuar os cálculos da função de avaliação apresentada na seção 3.3.2. Dessa forma, o *software* apresenta uma pontuação na parte superior da distribuição feita pelo usuário.

O Exemplo 4.1 apresenta um arquivo no formato CSV (*Comma-Separated Values*) que é utilizado para montar uma matriz de oito linhas e dez colunas.

Exemplo 4.1: *Exemplo de arquivo no formato CSV dos pesos da matriz.*

```
1 0;3;8;2;2;1;4;4;3;2
2 6;3;5;1;4;3;1;5;0;1
3 0;0;7;3;1;2;1;5;4;5
4 2;6;0;1;3;4;4;5;8;3
5 5;2;6;0;7;8;1;0;0;4
6 1;2;1;1;4;9;6;7;3;0
7 6;6;3;7;8;0;3;2;0;8
8 7;5;8;7;4;4;1;1;8;1
```

4.2 Criação de Esqueletos das Vias

Seguindo o modelo de representação utilizado em Machado et al [11], apresentado na seção 2.2, foi criado um *software* em JavaFX [1] que permite o desenho da representação *skeletonization* num mapa fornecido pelo serviço Google Static Maps [8]. A Figura 4.3 apresenta a ferramenta criada.

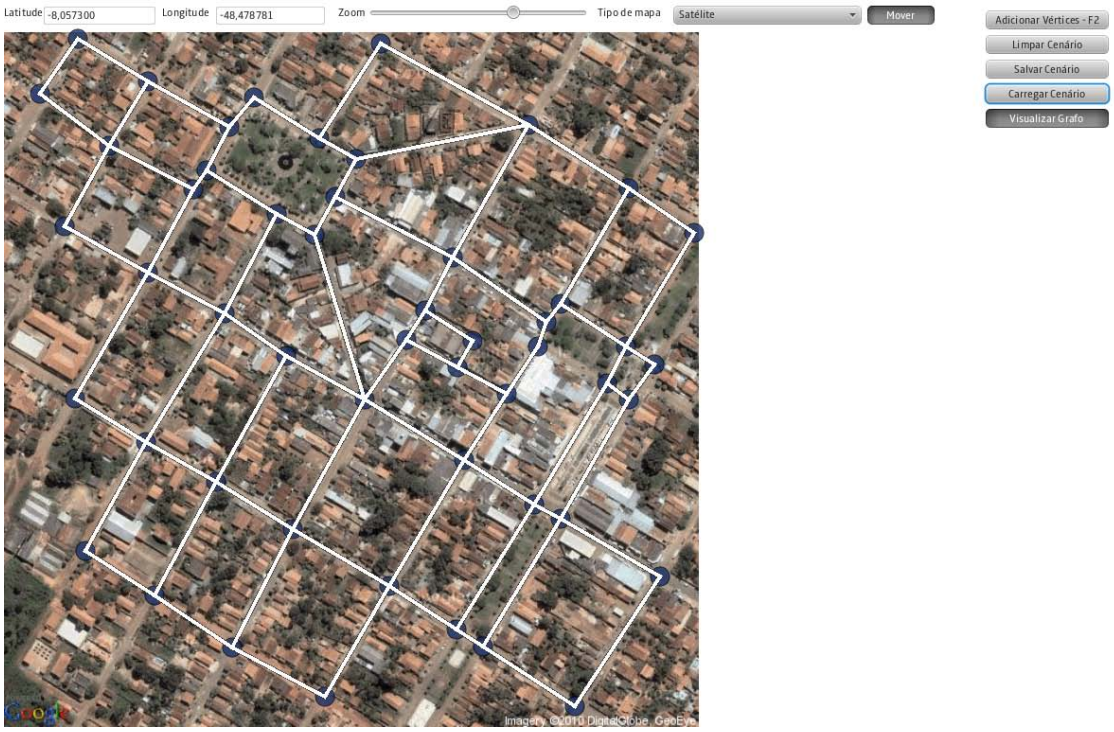


Figura 4.3: *Ferramenta desenvolvida para o desenho da representação skeletonization.*

Sobre o mapa da cidade de Colinas do Tocantins, apresentado na Figura 4.3, foram marcados vários vértices que representam as interseções entre vias. Cada vértice é representado por círculos roxos. As arestas representam uma seção da via e sempre são ligadas a dois vértices.

Na ferramenta apresentada na Figura 4.3, é possível:

- Alterar a latitude e longitude da visualização;
- Aumentar e diminuir o *zoom* do mapa;
- Alternar entre os tipos de mapa: satélite, desenho das ruas, híbrido e terreno;
- Mover a latitude e longitude através do clique e movimento do cursor;
- Adicionar e remover vértices e arestas;
- Limpar os vértices e arestas adicionadas no cenário;
- Salvar os vértices e arestas num arquivo no formato JSON (*JavaScript Object Notation*) [4];
- Carregar um arquivo no formato JSON no cenário.

O Exemplo 4.2 apresenta um arquivo no formato JSON para um cenário com seis vértices e seis arestas. Nota-se que a sintaxe dos arquivos JSON são bem simples e, seus conteúdos, podem ser lidos em qualquer linguagem de programação.

Exemplo 4.2: *Exemplo de arquivo no formato JSON de um cenário.*

```
{ "vertices": [
    {"id":1,"y":421,"x":383},
    {"id":2,"y":276,"x":468},
    {"id":3,"y":207,"x":355},
    {"id":4,"y":245,"x":329},
    {"id":5,"y":377,"x":300},
    {"id":6,"y":334,"x":231}],
  "arestas": [
    {"idDestino":2,"idOrigem":1},
    {"idDestino":3,"idOrigem":2},
    {"idDestino":4,"idOrigem":3},
    {"idDestino":1,"idOrigem":4},
    {"idDestino":5,"idOrigem":1},
    {"idDestino":6,"idOrigem":5}]
}
```

No Exemplo 4.2 pode-se notar que os dados armazenados da representação *skeletonization* são apenas as posições x e y de cada vértice e os vértices que cada aresta é ligada. Nota-se que o atributo “id” de cada vértice é criado no momento em que o cenário é salvo somente para relacioná-los às arestas.

4.3 Algoritmos

Esta seção apresenta os algoritmos relacionados à representação por matriz, apresentada no Capítulo 3.

Para criar o algoritmo da função de avaliação, definida na seção 3.3.2, é necessário criar várias outras funções, são elas: o cálculo da influência de um recurso, distâncias entre pontos e proteção das células. Os algoritmos dessas funções são apresentados no Apêndice A, na seção A.1.

O primeiro algoritmo, é mais fácil de ser abstraído e implementado. Ele consiste em testar exaustivamente todas combinações possíveis de distribuições e apresentar a melhor distribuição entre todas. No entanto, na grande maioria das vezes, testar todas as combinações é muito oneroso e, em determinadas instâncias do problema, pode ser tornar um processo inviável. Por este motivo, o estudo do algoritmo que testa todas as combinações é apresentado apenas no Apêndice A, na seção A.1.1.

4.4 Algoritmo Guloso

Utilizando a representação de matriz apresentada na seção 4.1, juntamente com a função de avaliação apresentada na seção 3.3.2, pode-se criar um algoritmo guloso para resolução do problema.

O Algoritmo 4.1 é guloso [13, página 95] [17, página 296] porque escolhe a célula da matriz M que maximiza momentaneamente a função de avaliação $fa()$.

Algoritmo 4.1: $Guloso(M,k)$

Entrada: M matriz de n linhas e m colunas.
k quantidade de recursos a serem distribuídos.
Saída: S conjunto de pontos (x,y).

```
1  S ← {}.
2  para l ← 1 até k faça
3      melhorPonto ← (1,1).
4      avalicaoMelhorPonto ← fa(M,S,melhorPonto).
5      para i ← 1 até n faça
6          para j ← 1 até m faça
7              p ← (i,j).
8              avaliacao ← fa(M,S,p).
9              se avaliacao > avaliacaoMelhorPonto então
10                 melhorPonto ← p.
11                 avaliacaoMelhorPonto ← avaliacao.
12             fim
13             j ← j+1.
14         fim
15         i ← i+1.
16     fim
17     S ← S∪melhorPonto.
18     l ← l+1.
19 fim
20 retorna S.
```

No Algoritmo 4.1, foi utilizada, nas linhas 4 e 8, uma variação do Algoritmo A.4 que determina a função de avaliação *fa()*. Nessa variação, a função *fa()* recebe três argumentos, sendo que os dois primeiros foram definidos no Algoritmo A.4.

O segundo e terceiro argumento, da função *fa()*, utilizada no Algoritmo 4.1, facilita a compreensão de que, existe um conjunto S de coordenadas já escolhidas e uma coordenada p a ser avaliada. Sendo assim, a função *fa()*, utilizada no Algoritmo 4.1, pode unir o conjunto S com a coordenada p para se adequar a definição da função $fa()$ feita no Algoritmo A.4.

O Algoritmo 4.1 tem complexidade $k \cdot (O(fa) + n \cdot m \cdot O(fa))$. Utilizando a complexidade de *fa()* calculada na seção 4.3, tem-se a seguinte complexidade para o Algoritmo 4.1:

$$O(Guloso) = k \cdot \Big(O(fa) + n \cdot m \cdot O(fa) \Big) \tag{4-1}$$

$$= k \cdot \Big(O(fa) \cdot (1 + n \cdot m) \Big) \tag{4-2}$$

$$= k \cdot \Big(n \cdot m \cdot (k+1) \Big) \cdot (1 + n \cdot m) \tag{4-3}$$

$$= n \cdot m \cdot (k^2 + k) \cdot (1 + n \cdot m) \tag{4-4}$$

$$= (k^2 + k) \cdot (n \cdot m + n^2 \cdot m^2) \tag{4-5}$$

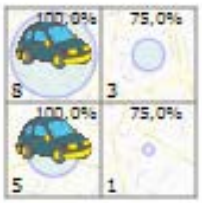
É importante ressaltar que a complexidade do Algoritmo 4.1 obtida na expressão 4-5 é polinomial.

Com uma implementação feita em Java do Algoritmo 4.1, foram feitas simulações com matrizes quadradas de duas a cinquenta linhas. A Tabela 4.1 apresenta o tempo gasto na execução, a quantidade de recursos utilizados e a nota da função de avaliação.

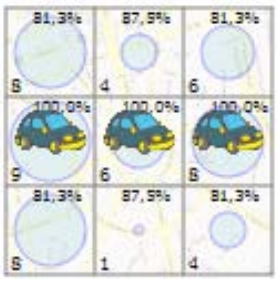
Tabela 4.1: *Simulação com o algoritmo de guloso com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	1 milisseg.	17	16	94,12%
(3 x 3) - 3	1 milisseg.	54	48	88,89%
(4 x 4) - 4	4 milisseg.	89	78,28	87,96%
(5 x 5) - 5	6 milisseg.	115	100,43	87,33%
(6 x 6) - 6	5 milisseg.	161	133,54	82,94%
(7 x 7) - 7	19 milisseg.	240	192,62	80,26%
(8 x 8) - 8	39 milisseg.	319	251,39	78,81%
(9 x 9) - 9	103 milisseg.	426	328,14	77,03%
(10 x 10) - 10	146 milisseg.	499	383,73	76,90%
(11 x 11) - 11	278 milisseg.	617	454,75	73,70%
(12 x 12) - 12	592 milisseg.	728	526,38	72,30%
(13 x 13) - 13	730 milisseg.	860	604,20	70,26%
(14 x 14) - 14	1 seg. 109 milisseg.	987	672,25	68,11%
(15 x 15) - 15	1 seg. 759 milisseg.	1144	756,40	66,12%
(25 x 25) - 25	35 seg. 547 milisseg.	3055	1598,45	52,32%
(30 x 30) - 30	1 min. 45 seg. 907 milisseg.	4511	2111,69	46,81%
(40 x 40) - 40	10 min. 12 seg. 852 milisseg.	8131	3178.62	39,09%
(50 x 50) - 50	38 min. 3 seg. 369 milisseg.	12455	4081.82	32.77%

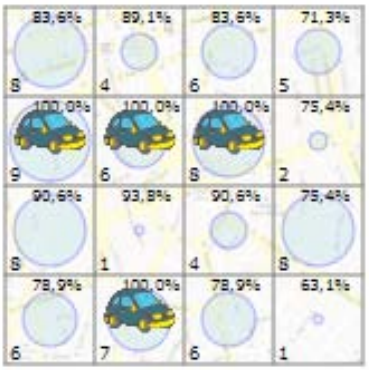
As figuras 4.4 e 4.5 apresentam as distribuições feitas, de 2 a 9 linhas, pelas simulações, apresentadas na Tabela 4.1.



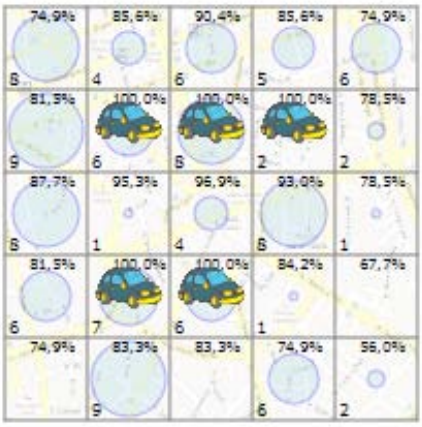
(a) *2 linhas, 2 colunas e 2 recursos*



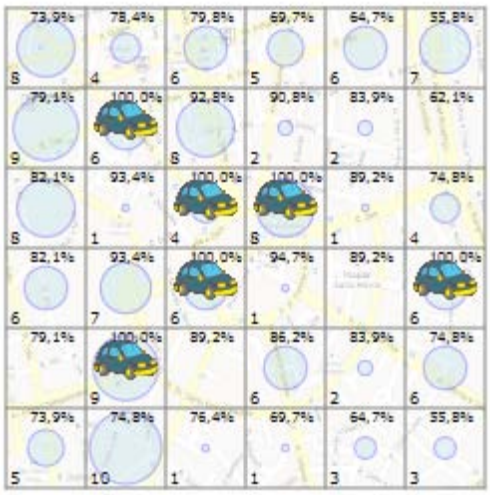
(b) *3 linhas, 3 colunas e 3 recursos*



(c) *4 linhas, 4 colunas e 4 recursos*

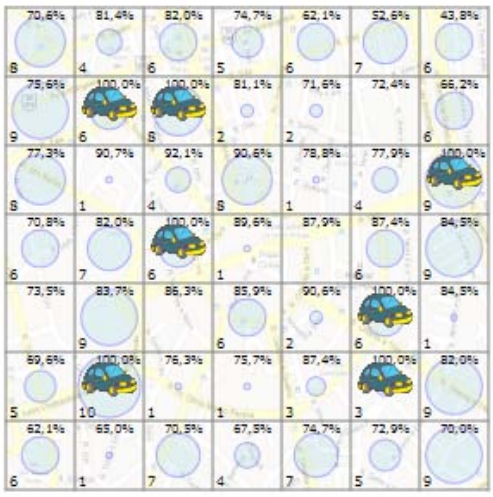


(d) *5 linhas, 5 colunas e 5 recursos*

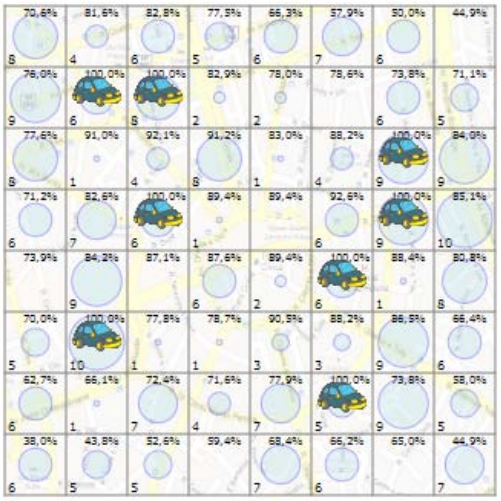


(e) *6 linhas, 6 colunas e 6 recursos*

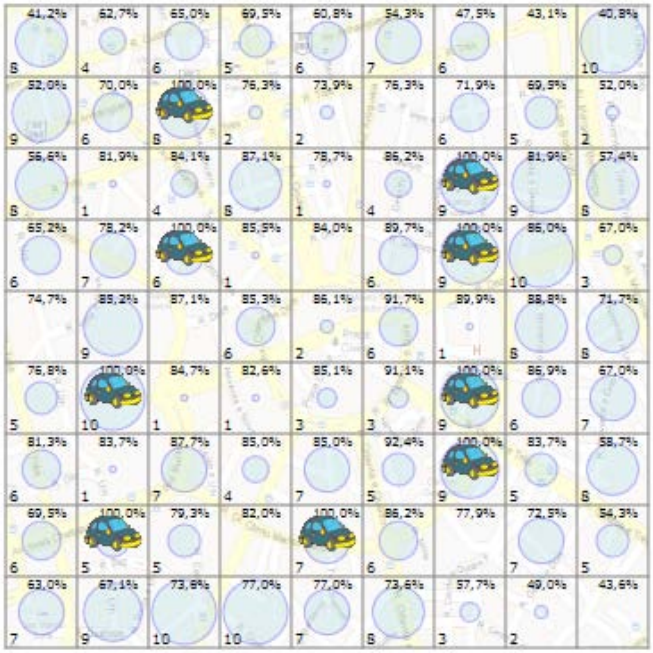
Figura 4.4: *Resultados das simulações das matrizes quadradas de 2 a 6 linhas com o algoritmo guloso*



(a) *7 linhas, 7 colunas e 7 recursos*



(b) *8 linhas, 8 colunas e 8 recursos*



(c) *9 linhas, 9 colunas e 9 recursos*

Figura 4.5: *Resultados das simulações das matrizes quadradas de 7 a 9 linhas com o algoritmo guloso*

Como feito na Tabela A.3 para o teste de todas as combinações, também é possível realizar estimativas da duração máxima do Algoritmo 4.1 utilizando a expressão definida em 4-5. A Tabela 4.2 apresenta as previsões de duração máxima do Algoritmo 4.1.

Tabela 4.2: *Previsões da duração máxima do algoritmo de guloso com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	***O(Guloso)***	**Tempo**
(2 x 2) - 2	120	0 milisseg.
(3 x 3) - 3	1.080	0 milisseg.
(4 x 4) - 4	5.440	2 milisseg.
(5 x 5) - 5	19.500	7 milisseg.
(6 x 6) - 6	55.944	21 milisseg.
(7 x 7) - 7	137.200	51 milisseg.
(8 x 8) - 8	299.520	112 milisseg.
(9 x 9) - 9	597.780	224 milisseg.
(10 x 10) - 10	1.111.000	417 milisseg.
(11 x 11) - 11	1.948.584	732 milisseg.
(12 x 12) - 12	3.257.280	1 seg. 224 milisseg.
(13 x 13) - 13	5.228.860	1 seg. 965 milisseg.
(14 x 14) - 14	8.108.520	3 seg. 48 milisseg.
(15 x 15) - 15	12.204.000	4 seg. 587 milisseg.
(25 x 25) - 25	254.312.500	1 min. 35 seg. 606 milisseg.
(30 x 30) - 30	754.137.000	4 min. 43 seg. 510 milisseg.
(40 x 40) - 40	4.201.024.000	26 min. 19 seg. 332 milisseg.
(50 x 50) - 50	15.943.875.000	1 hora 39 min. 53 seg. 937 milisseg.
(100 x 100) - 100	1.010.101.000.000	4 dias 9 horas 28 min. 57 seg. 218 milisseg.
(110 x 110) - 110	1.787.813.841.000	7 dias 18 horas 41 min. 50 seg. 466 milisseg.
(120 x 120) - 120	3.011.076.288.000	13 dias 2 horas 26 min. 23 seg. 566 milisseg.
(130 x 130) - 130	4.864.226.107.000	21 dias 3 horas 57 min. 36 seg. 431 milisseg.
(140 x 140) - 140	7.583.705.304.000	32 dias 23 horas 56 min. 57 seg. 31 milisseg.
(150 x 150) - 150	11.467.072.125.000	49 dias 21 horas 28 min. 49 seg. 370 milisseg.

A Figura 4.6 apresenta a comparação entre as tabelas 4.1 e 4.2.

			Milissegundos	
n	m	k	O(Guloso)	Simulação
2	2	2	0	1
3	3	3	0	1
4	4	4	2	4
5	5	5	7	6
6	6	6	21	5
7	7	7	52	19
8	8	8	113	39
9	9	9	225	103
10	10	10	418	146
11	11	11	733	278
12	12	12	1.225	592
13	13	13	1.966	730
14	14	14	3.048	1.109
15	15	15	4.588	1.759
25	25	25	95.606	35.547
30	30	30	283.510	105.907
40	40	40	1.579.332	612.852
50	50	50	5.993.938	2.283.369
100	100	100	379.737.218	

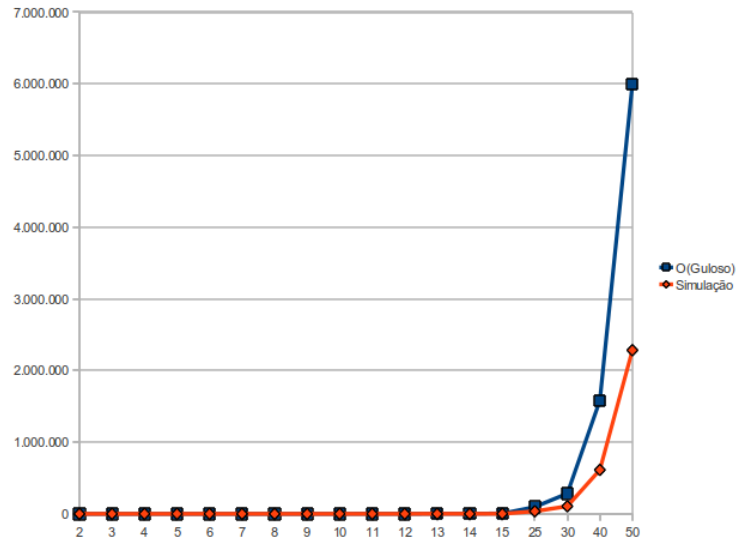


Figura 4.6: *Comparação entre as tabelas 4.1 e 4.2.*

A partir da Figura 4.6, pode-se notar que, a função $O(Guloso)$, apresentada na expressão 4-5, é o limite superior do Algoritmo 4.1.

4.5 Algoritmo Guloso Dinâmico com Avaliações Limitadas

Mesmo parecendo simples, o Algoritmo 4.1, apresentado na seção 4.4, pode ser inviável para instâncias superiores a 50 linhas, 50 colunas e 50 recursos, assim como apresentado nas tabelas 4.1 e 4.2.

No entanto, pode-se utilizar técnicas da programação dinâmica [17, página 259] e uma simplificação da função de avaliação apresentada no Algoritmo A.4.

A simplificação da função de avaliação diminui a qualidade dos resultados mas diminui significativamente a complexidade do algoritmo. Deve-se destacar que a diminuição da complexidade aumenta a performance do algoritmo.

Essa simplificação consiste em verificar onde a influência do recurso é muito baixa, por exemplo, menor que 1%. Através desta simplificação não é necessário calcular a influência do recurso na área de atendimento inteira, mas somente num raio em torno do recurso.

Utilizando a função de influência definida na expressão 3-8 pode-se fazer os cálculos apresentados na Tabela 4.3 para encontrar a distância onde a influência se torna menor que 1%.

Tabela 4.3: *Cálculos para encontrar h(d) < 1%.*

distância	*h(d))*
0	1
1	$\frac{1}{2^1} = 0.5$
2	$\frac{1}{2^2} = 0.25$
3	$\frac{1}{2^3} = 0.125$
4	$\frac{1}{2^4} = 0.0625$
5	$\frac{1}{2^5} = 0.03125$
6	$\frac{1}{2^6} = 0.015625$
7	$\frac{1}{2^7} = 0.0078125$

Como apresentado na Tabela 4.3, distâncias superiores a 6 unidades possuem influências menores que 0,015625, ou seja, menor que 1%. Dessa forma, pode-se calcular a influência do recurso somente para distâncias menores ou iguais a 6 unidades.

Fixar a distância limite no algoritmo restringiria suas aplicações, pois a função *h(d)* poderia ser expressa de outra maneira. No entanto, pode-se fazer uma verificação nas primeiras instruções do algoritmo guloso. O Algoritmo 4.2 apresenta a verificação de distância limite.

Algoritmo 4.2: $VerificacaoLimite(h,n,m)$

Entrada: h função de influência do recurso.
n quantidade de linhas e m quantidade de colunas da matriz M.
Saída: Distância limite de influência.

```
1  distanciaMaxima ← max(n,m).
2  i ← 0.
3  para i ← 1 até distanciaMaxima − 1 faça
4  |   influencia ← h(i).
5  |   se influencia < 0,01 então
6  |   |   retorna i − 1.
7  |   fim
8  |   i ← i + 1.
9  fim
10 retorna i.
```

No Algoritmo 4.2, considera-se que a distância zero é sempre maior que 1%. Na linha 3, a variável i inicia com o valor 1 e é incrementada até chegar em

$distanciaMaxima - 1$. É necessário subtrair 1 da distância máxima porque o recurso sempre ocupa 1 célula. Desta forma, se uma matriz M possuir o valor 10 como a maior dimensão, por exemplo: 3x10, 10x4 e 6x10, então a distância máxima poderá ser apenas 9. A complexidade do Algoritmo 4.2 é $O(VerificacaoLimite) = max(n,m)$.

A Figura 4.7 apresenta a simplificação proposta da função de avaliação.

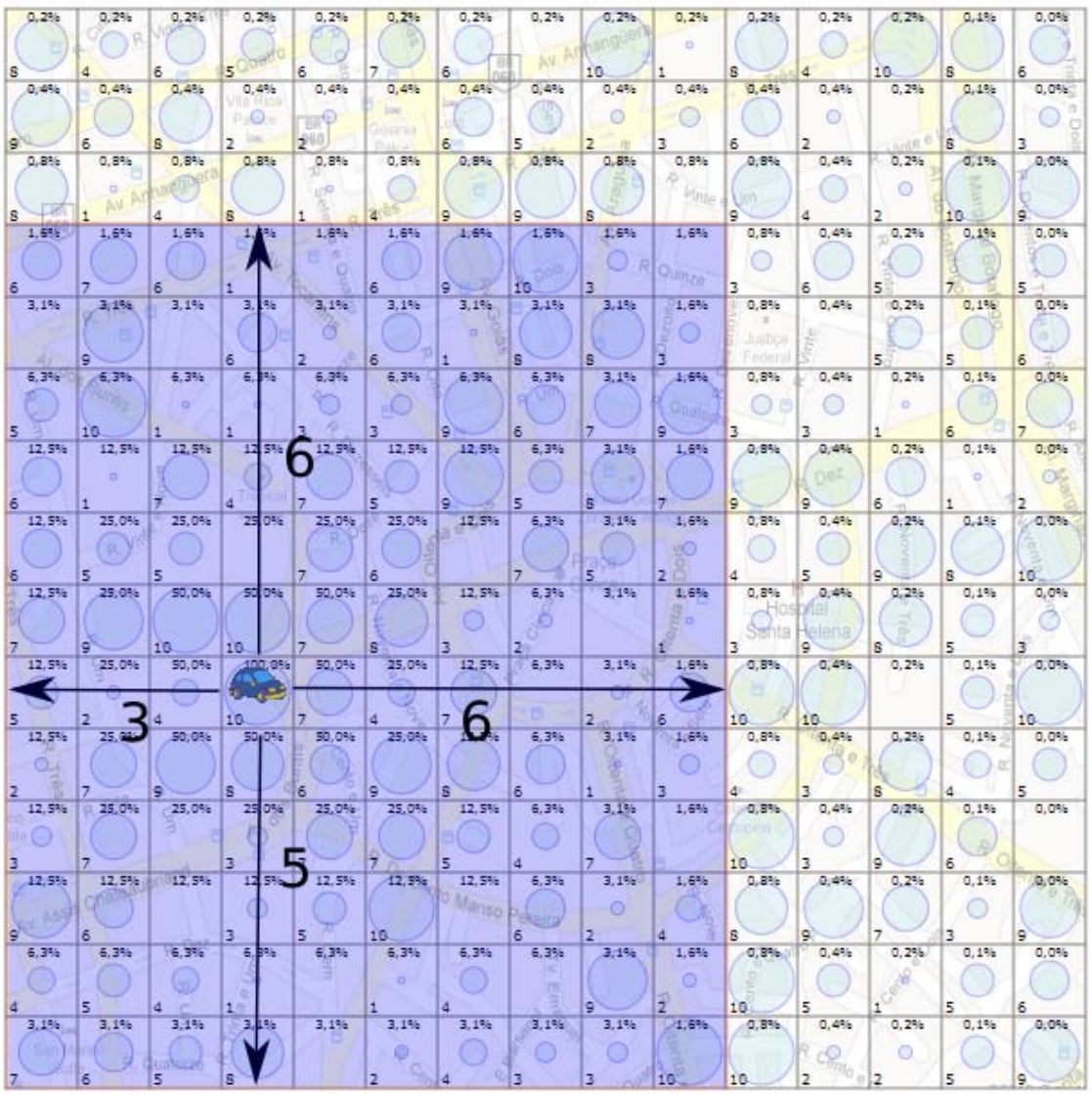


Figura 4.7: *Simplificação da função de avaliação com análise da influência mínima.*

Na Figura 4.7, pode-se visualizar um retângulo hachurado iniciado na linha 4 e coluna 1 que termina na linha 15 e coluna 10. Esse retângulo representa a área na qual a influência do recurso é relevante. Com a simplificação, as demais células da matriz não serão influenciadas pelo recurso evitando vários cálculos.

Como apresentado na Figura 4.7, no lado esquerdo e no lado inferior do recurso, nem sempre a influência é um quadrado, isto acontece quando o recurso está próximo

à borda da matriz. No lado esquerdo do recurso na Figura 4.7, só é possível analisar 3 colunas e, na parte inferior, somente 5 linhas.

O Algoritmo 4.3 apresenta como o retângulo que representa a área de influência é calculado.

Algoritmo 4.3: $RegiaoInfluencia(distanciaMaxima, i, j, n, m)$

Entrada: $distanciaMaxima$ distância máxima de influência.
Linha i e coluna j são as posições do recurso.
n quantidade de linhas e m quantidade de colunas da matriz M.
Saída: Um conjunto com a posição inicial (io, jo) e final (if, jf) do retângulo.

1 $io \leftarrow i - distanciaMaxima$.
2 $jo \leftarrow j - distanciaMaxima$.
3 $if \leftarrow i + distanciaMaxima$.
4 $jf \leftarrow j + distanciaMaxima$.
5 **se** $io < 1$ **então**
6 | $io \leftarrow 1$.
7 **fim**
8 **se** $jo < 1$ **então**
9 | $jo \leftarrow 1$.
10 **fim**
11 **se** $if > n$ **então**
12 | $if \leftarrow n$.
13 **fim**
14 **se** $jf > m$ **então**
15 | $jf \leftarrow m$.
16 **fim**
17 **retorna** $\{(io, jo), (if, jf)\}$.

O Algoritmo 4.3 tem complexidade 1. Nas linhas 1 a 4, é feito o cálculo da posição inicial e final do retângulo sem analisar os extremos da matriz M. Nas linhas 5 a 16, é verificado se a posição inicial ou final excedeu algum dos extremos da matriz de n linhas e m colunas.

O Algoritmo 4.4 apresenta o cálculo da proteção utilizando a área de influência limitada.

Algoritmo 4.4: $ProtecaoAreaLimitada(PROAnterior, p, io, jo, if, jf)$

Entrada: *PROAnterior* matriz de proteção calculada num passo anterior, com n linhas e m colunas.
p novo ponto onde pretende-se colocar um recurso.
(io, jo) e (if, jf) são as coordenadas de início e fim do retângulo de influência.
Saída: Matriz *PRO* de $(if - io + 1)$ linhas e $(jf - jo + 1)$ colunas com os valores das proteções.

```
1  Declara uma matriz PRO com (if − io + 1) linhas e (jf − jo + 1) colunas.
2  para i ← io até if faça
3      para j ← jo até jf faça
4          d ← DistanciaMatriz(p.i, p.j, i, j).
5          se d = 0 então
6              perigo ← 0.
7          senão
8              protecao ← ProtecaoLocal(d).
9              perigo ← (1 − PROAnterior[i, j]) ∗ (1 − protecao).
10         fim
11         PRO[i − io, j − jo] ← 1 − perigo.
12         j ← j + 1.
13     fim
14     i ← i + 1.
15 fim
16 retorna PRO.
```

De fato o Algoritmo 4.4 efetua cálculos somente com a região influenciada do recurso que pretende-se adicionar. O resultado desse algoritmo é uma matriz com o tamanho do retângulo da região influenciada, ou seja, uma matriz de $(if - io + 1)$ linhas e $(jf - jo + 1)$ colunas.

O Algoritmo 4.4 tem complexidade $O(ProtecaoAreaLimitada) = (if - io + 1) \cdot (jf - jo + 1)$. A expressão $(if - io + 1) \cdot (jf - jo + 1)$ representa os laços das linhas 2 e 3 para calcular as proteções do retângulo de influência. Como este retângulo pode possuir, no máximo, $2 \cdot dM + 1$ linhas e $2 \cdot dM + 1$ colunas, onde dM é o resultado do algoritmo 4.2 que calcula a distância máxima relevante, então a complexidade, no pior caso é:

$$T(ProtecaoAreaLimitada) = (2 \cdot dM + 1) \cdot (2 \cdot dM + 1) \quad (4\text{-}6)$$

$$= (2 \cdot dM + 1)^2 \quad (4\text{-}7)$$

Como o retângulo que representa a área de influência é uma submatriz da matriz M que possui n linhas e m colunas, tem-se que a complexidade do Algoritmo 4.4 é inferior a complexidade do Algoritmo A.3.

Pelo fato do cálculo da proteção das células da matriz M ter mudado, também é necessário modificar a função de avaliação. O Algoritmo 4.5 apresenta a função de avaliação otimizada.

Algoritmo 4.5: $faOtimizada(avAnterior, M, p, dM, PROAnterior)$

Entrada: $avAnterior$ pontuação obtida na última avaliação.
M matriz de n linhas e m colunas.
dM distância máxima do retângulo de influência.
p ponto que deseja-se adicionar.
$PROAnterior$ matriz de proteção calculada num passo anterior, com n linhas e m colunas.
Saída: Um conjunto com a avaliação da distribuição e o novo retângulo de proteções na área influenciada.

```
1  {(io, jo), (if, jf)} ← RegiaoInfluencia(dM, p.i, p.j, n, m).
2  pri ← 0. {Pontuação na região de influência}
3  pari ← 0. {Pontuação anterior na região de influência}
4  protecoes ← ProtecaoAreaLimitada(PROAnterior, p, io, jo, if, jf).
5  para i ← io até if faça
6      para j ← jo até jf faça
7          pri ← pri + M[i, j] * protecoes[i − io, j − jo].
8          pari ← pari + M[i, j] * PROAnterior[i, j].
9          j ← j + 1.
10     fim
11     i ← i + 1.
12 fim
13 pontuacao ← avAnterior + (pri − pari).
14 retorna {pontuacao, protecoes}.
```

O Algoritmo 4.5 tem complexidade igual à soma da complexidade do Algoritmo 4.3 com a complexidade do Algoritmo 4.4 e com a complexidade dos laços criados nas linhas 5 e 6. Assim, como no Algoritmo 4.4, a complexidade dos laços criados nas linhas 5 e 6 é $(2 \cdot dM + 1)^2$. A complexidade do Algoritmo 4.5 é:

$$T(faOtimizada) = 1 + (2 \cdot dM + 1)^2 + (2 \cdot dM + 1)^2 \quad (4\text{-}8)$$

$$= 1 + 2 \cdot (2 \cdot dM + 1)^2 \quad (4\text{-}9)$$

Dessa forma, a complexidade do Algoritmo 4.5 também é melhor que a complexidade do Algoritmo A.4.

Finalmente, após definir os novos algoritmos de cálculo da proteção e da função avaliação otimizada, pode-se construir um novo algoritmo guloso que aproveita estes recursos. O Algoritmo 4.6 apresenta o algoritmo guloso responsável pela busca da melhor escolha.

Algoritmo 4.6: $BuscaGulosaOtimizada(M, avAnterior, dM, PRO)$

Entrada: M matriz de n linhas e m colunas.
$avAnterior$ avaliação anterior.
dM distância máxima do retângulo de influência.
PRO matriz de proteções.
Saída: Conjunto com o melhor ponto, a melhor avaliação e a matriz de proteção do melhor ponto.

```
1  melhorPonto ← (1,1).
2  {avalicaoMelhorPonto, PROInfluMelhor} ←
   faOtimizada(avAnterior, M, melhorPonto, dM, PRO).
3  para i ← 1 até n faça
4      para j ← 1 até m faça
5          p ← (i, j).
6          {avaliacao, PROp} ←
           faOtimizada(avAnterior, M, p, distanciaMaxima, PRO).
7          se avaliacao > avaliacaoMelhorPonto então
8              melhorPonto ← p.
9              avaliacaoMelhorPonto ← avaliacao.
10             PROInfluMelhor ← PROp.
11         fim
12         j ← j + 1.
13     fim
14     i ← i + 1.
15 fim
16 retorna {melhorPonto, avaliacaoMelhorPonto, PROInfluMelhor}.
```

O Algoritmo 4.6 percorre todas as n linhas e m colunas da matriz M na busca da melhor escolha local. Por isto, a complexidade deste algoritmo é $n \cdot m$ vezes a complexidade da função de avaliação *faOtimizada()* apresentada no Algoritmo 4.5 somado com esta complexidade:

$$T(BuscaGulosaOtimizada) = T(faOtimizada) + n \cdot m \cdot T(faOtimizada) \quad (4\text{-}10)$$

$$= T(faOtimizada) \cdot (1 + n \cdot m) \quad (4\text{-}11)$$

$$= \left(1 + 2 \cdot (2 \cdot dM + 1)^2\right) \cdot (1 + n \cdot m) \quad (4\text{-}12)$$

$$= 1 + n \cdot m + 2 \cdot (2 \cdot dM + 1)^2 \cdot (1 + n \cdot m) \quad (4\text{-}13)$$

O Algoritmo 4.7 apresenta o algoritmo guloso completo com as otimizações.

Algoritmo 4.7: $GulosoO(M,k)$

Entrada: M matriz de n linhas e m colunas.
k quantidade de recursos a serem distribuídos.
Saída: S conjunto de pontos (x,y).

```
1  S ← {}.
2  avCorrente ← 0.
3  distanciaMaxima ← VerificacaoLimite(h(),n,m).
4  Declara uma matriz PRO de n linhas e m colunas.
5  Inicializa todas as células da matriz PRO com o valor 0 (zero).
6  para l ← 1 até k faça
7      {melhorPonto, avaliacaoMelhorPonto, PROInfluMelhor} ←
       BuscaGulosaOtimizada(M, avCorrente, distanciaMaxima, PRO).
8      S ← S ∪ melhorPonto.
9      {(io, jo), (if, jf)} ←
       RegiaoInfluencia(distanciaMaxima, melhorPonto.i, melhorPonto.j, n, m).
10     para i ← io até if faça
11         para j ← jo até jf faça
12             PRO[i, j] ← PROInfluMelhor[i − io, j − jo].
13             j ← j + 1.
14         fim
15         i ← i + 1.
16     fim
17     avCorrente ← avaliacaoMelhorPonto.
18     l ← l + 1.
19 fim
20 retorna S.
```

Após obter o melhor ponto através da busca gulosa no Algoritmo 4.7, as linhas 8 a 14 atualizam a matriz de proteção somente na área influenciada e na linha 15 a avaliação corrente é atualizada.

Dado que dM é a distância máxima da região de influência relevante, a complexidade do Algoritmo 4.7 é:

$$O(GulosoO) = k \cdot [T(BuscaGulosaOtimizada) + 1 + 2 \cdot (2 \cdot dM + 1)^2] \quad (4\text{-}14)$$
$$= k \cdot \left[\left(1 + 2 \cdot (2 \cdot dM + 1)^2\right) \cdot (1 + n \cdot m) + 1 + 2 \cdot (2 \cdot dM + 1)^2 \right] \quad (4\text{-}15)$$
$$= k \cdot \left[\left(1 + 2 \cdot (2 \cdot dM + 1)^2\right) \cdot \left((1 + n \cdot m) + 1\right) \right] \quad (4\text{-}16)$$
$$= k \cdot \left(1 + 2 \cdot (2 \cdot dM + 1)^2\right) \cdot (n \cdot m + 2) \quad (4\text{-}17)$$

Nota-se que a expressão 4-17 que determina a complexidade do Algoritmo 4.7 é polinomial.

Com uma implementação feita em Java do Algoritmo 4.7, foram feitas simulações com matrizes quadradas de duas a cem linhas. A Tabela 4.4 apresenta o tempo gasto na execução, a quantidade de recursos utilizados e a nota da função de avaliação.

Tabela 4.4: *Simulação com o algoritmo guloso otimizado com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	1 milisseg.	17	16	94,12%
(3 x 3) - 3	1 milisseg.	54	47,5	87,96%
(4 x 4) - 4	1 milisseg.	89	75,11	84,39%
(5 x 5) - 5	1 milisseg.	115	95,99	83,47%
(6 x 6) - 6	1 milisseg.	161	122,51	76,09%
(7 x 7) - 7	3 milisseg.	240	179,60	74,83%
(8 x 8) - 8	59 milisseg.	319	251,39	78,81%
(9 x 9) - 9	27 milisseg.	426	326,44	76,63%
(10 x 10) - 10	32 milisseg.	499	383,73	76,90%
(11 x 11) - 11	38 milisseg.	617	454,75	73,70%
(12 x 12) - 12	44 milisseg.	728	526,52	72,32%
(13 x 13) - 13	77 milisseg.	860	604,84	70,33%
(14 x 14) - 14	92 milisseg.	987	672,25	68,11%

(15 x 15) - 15	117 milisseg.	1144	752,59	65,78%
(25 x 25) - 25	610 milisseg.	3055	1596,66	52,26%
(30 x 30) - 30	979 milisseg.	4511	2118,20	46,96%
(40 x 40) - 40	2 seg. 460 milisseg.	8131	3173,77	39,03%
(50 x 50) - 50	5 seg. 259 milisseg.	12455	4077,98	32,74%
(100 x 100) - 100	47 seg. 371 milisseg.	50190	9238,11	18,41%
(200 x 200) - 200	6 min. 43 seg. 733 milisseg.	201054	19621,71	9,76%
(300 x 300) - 300	23 min. 37 seg. 421 milisseg.	449778	29884,18	6,64%
(400 x 400) - 400	56 min. 49 seg. 886 milisseg.	799981	40360.92	5,05%

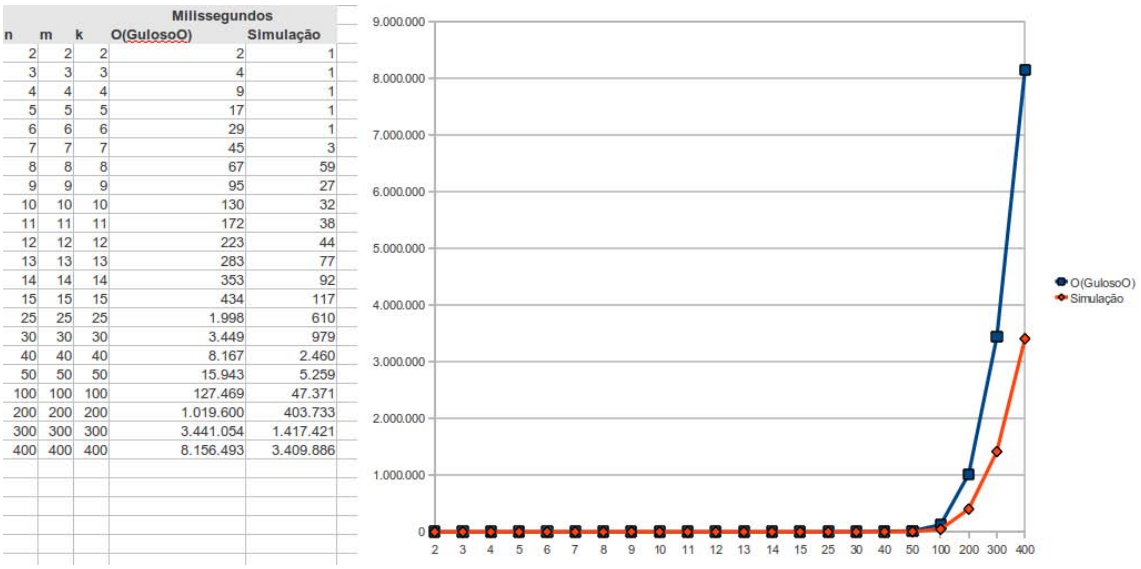
É possível realizar estimativas da duração máxima do Algoritmo 4.7 utilizando a expressão definida em 4-17. A Tabela 4.5 apresenta as previsões de duração máxima do Algoritmo 4.7 com distância máxima igual a 6.

Tabela 4.5: *Previsões da duração máxima do algoritmo de guloso otimizado com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	***O(GulosoO)***	**Tempo**
(2 x 2) - 2	4.068	1 milisseg.
(3 x 3) - 3	11.187	4 milisseg.
(4 x 4) - 4	24.408	9 milisseg.
(5 x 5) - 5	45.765	17 milisseg.
(6 x 6) - 6	77.292	29 milisseg.
(7 x 7) - 7	121.023	45 milisseg.
(8 x 8) - 8	178.992	67 milisseg.
(9 x 9) - 9	253.233	95 milisseg.

(10 x 10) - 10	345.780	129 milisseg.
(11 x 11) - 11	458.667	172 milisseg.
(12 x 12) - 12	593.928	223 milisseg.
(13 x 13) - 13	753.597	283 milisseg.
(14 x 14) - 14	939.708	353 milisseg.
(15 x 15) - 15	1.154.295	433 milisseg.
(25 x 25) - 25	5.313.825	1 seg. 997 milisseg.
(30 x 30) - 30	9.173.340	3 seg. 448 milisseg.
(40 x 40) - 40	21.723.120	8 seg. 166 milisseg.
(50 x 50) - 50	42.408.900	15 seg. 943 milisseg.
(100 x 100) - 100	339.067.800	2 min. 7 seg. 469 milisseg.
(200 x 200) - 200	2.712.135.600	16 min. 59 seg. 599 milisseg.
(300 x 300) - 300	9.153.203.400	57 min. 21 seg. 53 milisseg.
(400 x 400) - 400	21.696.271.200	2 horas 15 min. 56 seg. 492 milisseg.
(500 x 500) - 500	42.375.339.000	4 horas 25 min. 30 seg. 578 milisseg.
(1000 x 1000) - 1000	339.000.678.000	1 dia 11 horas 24 min. 3 seg. 863 milisseg.

A Figura 4.8 apresenta a comparação entre as tabelas 4.4 e 4.5.



			Milissegundos	
n	m	k	O(GulosoO)	Simulação
2	2	2	2	1
3	3	3	4	1
4	4	4	9	1
5	5	5	17	1
6	6	6	29	1
7	7	7	45	3
8	8	8	67	59
9	9	9	95	27
10	10	10	130	32
11	11	11	172	38
12	12	12	223	44
13	13	13	283	77
14	14	14	353	92
15	15	15	434	117
25	25	25	1.998	610
30	30	30	3.449	979
40	40	40	8.167	2.460
50	50	50	15.943	5.259
100	100	100	127.469	47.371
200	200	200	1.019.600	403.733
300	300	300	3.441.054	1.417.421
400	400	400	8.156.493	3.409.886

Figura 4.8: *Comparação entre as tabelas 4.4 e 4.5.*

A partir da Figura 4.8, pode-se notar que, a função $O(GulosoO)$, apresentada na expressão 4-17, é o limite superior do Algoritmo 4.7.

4.6 Algoritmo Genético

Também é possível utilizar algoritmos genéticos (AG) [5, página 6] para resolução do problema da distribuição de recursos. A modelagem é bastante parecida com a modelagem do problema das oito rainhas apresentada por Russell e Norvig [13, página 116].

A grande diferença da modelagem utilizada no problema das oito rainhas está na codificação do cromossomo que representa cada indivíduo. Russell e Norvig [13, página 116] definiram que, cada rainha deve ficar numa coluna do tabuleiro de xadrez, variando apenas, a linha onde a rainha está.

Para o problema da distribuição de recursos, tanto a coluna quanto a linha onde o recurso está, devem fazer parte da informação contida no cromossomo. Nota-se que, Russell e Norvig não utilizaram essas duas informações como parte do cromossomo porque duas, ou mais, rainhas na mesma coluna não é uma solução válida para o problema das oito rainhas.

O fato do cromossomo conter as informações da coluna e linha do recurso, permitirá a movimentação do recurso em todas as posições da matriz que representa a área de atendimento.

A Figura 4.9 apresenta a codificação do cromossomo utilizado no AG.

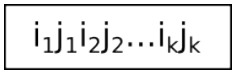
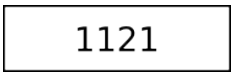


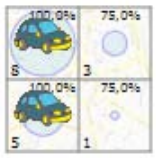
Figura 4.9: *Codificação do cromossomo.*

Dado que serão distribuídos k recursos, o cromossomo terá o dobro de alelos. Isto acontece porque cada grupo de dois alelos armazena a posição i e j de um recurso.

A Figura 4.10(a) apresenta o cromossomo [1121] e a Figura 4.10(b) apresenta a representação do cromossomo [1121].



(a) *Cromossomo.*



(b) *Representação do cromossomo.*

Figura 4.10: *Exemplo de representação do cromossomo.*

Nota-se que, as duas primeiras posições [11] da Figura 4.10 (a) representam a posição do primeiro recurso na Figura 4.10 (b). As duas últimas posições [21] representam a posição do segundo recurso.

A Figura 4.11 apresenta o processo geral dos algoritmos genéticos [17, página 116].

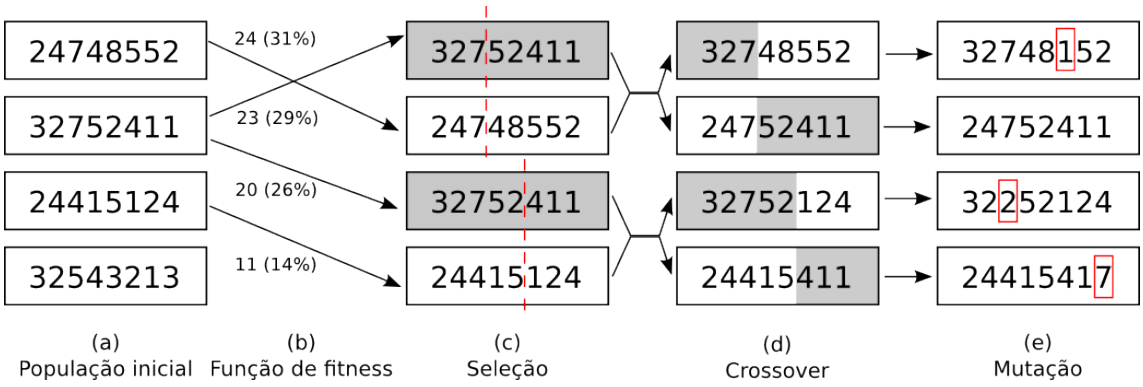


Figura 4.11: *Processo dos algoritmos genéticos.*

O passo (a), na Figura 4.11, representa a população inicial. Nesse AG, a população inicial será gerada aleatoriamente. A função de adaptação (do inglês *fitness*) do passo (b) será a função de avaliação definida na seção 3.3.2. Lembrando que esta função retorna as pontuações mais altas para as melhores distribuições, ou seja, os melhores indivíduos. No passo (c), será utilizada a seleção por roleta, onde os indivíduos mais adaptados possuem maior changes de reprodução. E, finalmente, no passo (d) será utilizado o *crossover* de um ponto.

Devem ser feitas várias simulações, buscando encontrar bons valores para as probabilidades de mutação, de *crossover* e a quantidade de indivíduos da população inicial. No entanto, este estudo foge do escopo deste trabalho.

O Algoritmo 4.8 apresenta a estrutura geral de um algoritmo genético [17, página 117].

Algoritmo 4.8: $AlgoritmoGenetico(populacao, FN\_FITNESS)$

Entrada: *populacao* um conjunto de indivíduos.
$FN\_FITNESS$ uma função que mede a adaptação de um indivíduo.
Saída: melhor indivíduo.

```
1  repita
2      nova_populacao ← ∅
3      para i ← 1 até TAMANHO(populacao) faça
4          x ← SELECAO_ALEATORIA(populacao, FN_FITNESS)
5          y ← SELECAO_ALEATORIA(populacao, FN_FITNESS)
6          filho ← REPRODUZ(x, y)
7          se pequena probabilidade aleatória então
8              filho ← MUTACAO(filho)
9          fim
10         Adicionar filho a nova_populacao
11     fim
12     populacao ← nova_populacao
13 até algum indivíduo estar adaptado o suficiente ou até ter decorrido tempo
   suficiente
14 retorna O melhor indivíduo em populacao, de acordo com FN_FITNESS
```

Com uma implementação feita em Java utilizando o arcabouço JGAP (*Java Genetic Algorithms Package*) [15] do Algoritmo 4.8, foram feitas simulações com matrizes quadradas de duas a cem linhas. A Tabela 4.6 apresenta o tempo gasto na execução, a quantidade de recursos utilizados e a nota da função de avaliação.

Tabela 4.6: *Simulação com o algoritmo genético utilizando uma população máxima de 100 indivíduos e 300 gerações com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	799 milisseg.	17	16	94.12%
(3 x 3) - 3	345 milisseg.	54	48,5	89.81%
(4 x 4) - 4	570 milisseg.	89	78,28	87.96%
(5 x 5) - 5	1 seg. 061 milisseg.	115	100,65	87.52%
(6 x 6) - 6	1 seg. 967 milisseg.	161.0	135,17	83.96%
(7 x 7) - 7	3 seg. 290 milisseg.	240	192,85	80.35%

(8 x 8) - 8	5 seg. 191 milisseg.	319	254,55	79.80%
(9 x 9) - 9	7 seg. 867 milisseg.	426	331,86	77.90%
(10 x 10) - 10	11 seg. 321 milisseg.	499	385,58	77.27%
(11 x 11) - 11	15 seg. 560 milisseg.	617	456,36	73.96%
(12 x 12) - 12	20 seg. 796 milisseg.	728	529,38	72.72%
(13 x 13) - 13	27 seg. 248 milisseg.	860	607,51	70.64%
(14 x 14) - 14	34 seg. 851 milisseg.	987	675,89	68.48%
(15 x 15) - 15	43 seg. 723 milisseg.	1144	760,15	66.45%
(25 x 25) - 25	3 min. 44 seg. 469 milisseg.	3055	1582,94	51.81%
(30 x 30) - 30	6 min. 38 seg. 424 milisseg.	4511	2093,32	46.40%
(40 x 40) - 40	15 min. 59 seg. 586 milisseg.	8131	3064,72	37.69%
(50 x 50) - 50	31 min. 24 seg. 60 milisseg.	12455	3911,42	31.40%
(100 x 100) - 100	3 horas 59 min. 48 seg. 869 milisseg.	50190	8483,41	16,90%

4.7 Heurística da Diagonal

A heurística da diagonal é uma forma de dividir uma área de atendimento em áreas menores buscando facilitar o processo de distribuição de recursos. Para o funcionamento, é necessária a matriz M com os pesos de cada célula, a quantidade k de recursos disponíveis e a média de pesos que um recurso consegue atender.

Com estes três argumentos, a heurística da diagonal é capaz de dividir uma matriz M num conjunto SM de submatrizes como apresentado na Figura 4.12.

Na Figura 4.12, a média de pesos que um recurso consegue atender é 20. Pode-se chegar nesse número através da divisão do somatório dos pesos pela quantidade de

recursos disponíveis ou pela análise de históricos. A expressão 4-18 apresenta o modo como o cálculo é executado:

$$media = \frac{\sum_{i=1}^{i=n} \sum_{j=1}^{j=m} M[i,j]}{k} \tag{4-18}$$

	1	2	3	4	5	6	7
1	0	10	9	4	10	4	3
2	2	10	1	9	6	1	4
3	0	8	4	2	7	0	0
4	8	10	7	6	8	6	10
5	4	3	5	6	7	4	10
6	2	4	10	2	8	8	5
7	3	7	6	5	8	6	2

moav = 20

	Iteração		
Agrupamento - (i, j)	1ª	2ª	3ª
1 - (1, 1)	0	22	
2 - (1, 3)	9	23	
3 - (1, 5)	10	21	
4 - (3, 1)	0	26	
5 - (3, 3)	4	19	52
6 - (3, 6)	0	16	
7 - (5, 1)	4	13	
8 - (5, 6)	4	27	
9 - (6, 3)	10	23	

Figura 4.12: *Submatrizes formadas pela heurística da diagonal.*

Na Figura 4.12, foram criadas nove submatrizes. O peso da célula (i, j) será representado por $p_{(i,j)}$.

Basicamente, a heurística da diagonal é o oposto do raio de influência fixo de um recurso. Nessa heurística, o recurso possui uma média de resolução e o seu raio de atendimento aumenta buscando chegar na média de resolução.

Para a formação das nove submatrizes na Figura 4.12, deve-se percorrer a matriz M a partir do canto esquerdo superior. Desta forma, inicia-se na linha 1 e coluna 1 verificando se o peso desta célula é superior à média calculada, neste caso a média é 20. Como $p_{(1,1)} < 20$, então a submatriz que será formada pode ser expandida. Esta expansão é feita na diagonal, por isto o nome "Heurística da Diagonal".

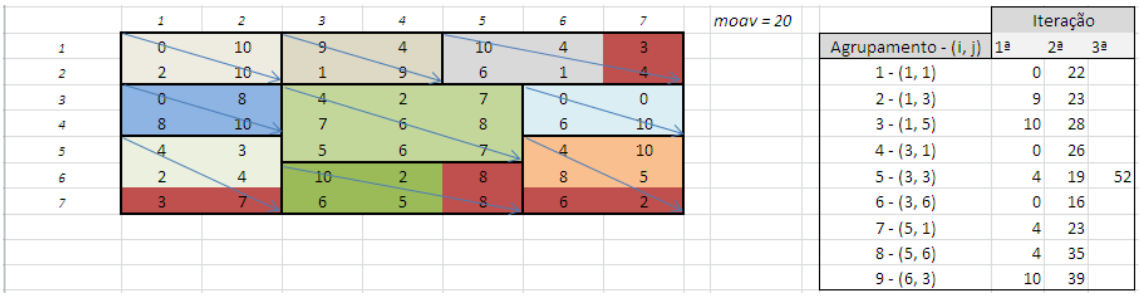
A expansão forma uma submatriz retangular iniciada na célula $(1,1)$ até a célula $(2,2)$. A Figura 4.12 apresenta a submatriz formada. Essa submatriz possui os pesos $p_{(1,1)}$, $p_{(1,2)}$, $p_{(2,1)}$ e $p_{(2,2)}$, a soma destes pesos é $0+10+2+10=22$. Como essa soma é superior à média, então essa submatriz não precisa mais ser expandida.

Em seguida, inicia-se o mesmo processo de formação de submatriz a partir da célula $(1,3)$. Após a formação da segunda e terceira submatrizes, o processo é iniciado a partir da primeira linha que não faz parte de uma submatriz. Nesse caso, a terceira linha ainda não faz parte de uma submatriz. Desta forma, o processo é iniciado a partir da linha 3 e coluna 1.

No término desse procedimento, a matriz M fica dividida em várias submatrizes. Mesmo após essa divisão, sobram células que não fazem parte de nenhuma submatriz. A Figura 4.12 apresenta que as células $(1,7)$, $(2,7)$, $(6,5)$, $(7,1)$, $(7,2)$, $(7,5)$, $(7,6)$ e $(7,7)$ não fazem parte de nenhuma submatriz. Este problema foi nomeado como "Problema

das fronteiras" porque acontece geralmente nas extremidades da matriz M ou entre duas submatrizes.

A Figura 4.13 apresenta como ficam as submatrizes após a resolução do problema das fronteiras.



	1	2	3	4	5	6	7
1	0	10	9	4	10	4	3
2	2	10	1	9	6	1	4
3	0	8	4	2	7	0	0
4	8	10	7	6	8	6	10
5	4	3	5	6	7	4	10
6	2	4	10	2	8	8	5
7	3	7	6	5	8	6	2

moav = 20

	Iteração		
Agrupamento - (i, j)	1ª	2ª	3ª
1 - (1, 1)	0	22	
2 - (1, 3)	9	23	
3 - (1, 5)	10	28	
4 - (3, 1)	0	26	
5 - (3, 3)	4	19	52
6 - (3, 6)	0	16	
7 - (5, 1)	4	23	
8 - (5, 6)	4	35	
9 - (6, 3)	10	39	

Figura 4.13: *Submatrizes formadas pela heurística da diagonal após a resolução do problema das fronteiras.*

Para resolução do problema das fronteiras, deve-se percorrer todas as linhas e colunas da matriz M. Quando encontrada, uma célula desagrupada será adicionada à submatriz à esquerda. Quando a célula desagrupada ocorrer na primeira coluna ou na última linha da matriz M, ela deve ser adicionada a submatriz que estiver acima.

Após a resolução do problema das fronteiras, a matriz M estará subdividida como apresentado na Figura 4.13. O Algoritmo 4.9 retorna *verdadeiro* quando uma submatriz deve parar de ser expandida na diagonal.

Algoritmo 4.9: $DeveParar HD(acumulado, media, i, j, n, m, tamanho, SM)$

Entrada: *acumulado* somatório de todas as células da submatriz.
media média de pesos que um recurso consegue atender.
i linha inicial da submatriz e j coluna da expansão atual da submatriz.
n quantidade de linhas e m quantidade de colunas da matriz M.
tamanho tamanho na diagonal da última expansão.
SM matriz com os números dos agrupamentos das submatrizes.
Saída: *verdadeiro* quando a expansão deve parar.

```
1  deveParar ← falso.
2  se acumulado ⩾ media então
3  |  deveParar ← verdadeiro.
4  senão
5  |  se j + 1 > m ou i + tamanho > n então
6  |  |  deveParar ← verdadeiro.
7  |  senão
8  |  |  se SM[i, j + 1] <> 0 então
9  |  |  |  deveParar ← verdadeiro.
10 |  |  fim
11 |  fim
12 fim
13 retorna deveParar.
```

As três condições em que a expansão deve parar, expressas pelo algoritmo 4.9, são:

1. A soma dos pesos da submatriz formada excede a média de pesos que um recurso consegue atender;
2. A expansão excede o limite direito ou inferior da matriz M;
3. A direita já possui uma submatriz.

O Algoritmo 4.10 realiza a soma dos pesos da última expansão. Nesse algoritmo, soma-se apenas a nova a linha e a nova coluna que foi adicionada na submatriz.

Algoritmo 4.10: $AcumuladoLinhaColuna(M, i, j, colunaInicial, tamanho, SM, nA)$

Entrada: M matriz de n linhas e m colunas.
i linha inicial da submatriz e j coluna da expansão atual da submatriz.
$colunaInicial$ coluna inicial da submatriz formada.
$tamanho$ tamanho na diagonal da última expansão.
SM matriz com os números dos agrupamentos das submatrizes.
nA número do agrupamento atual.
Saída: Acumulado parcial.

```
1  acumulado ← 0.
2  para k ← colunaInicial até j faça
3      acumulado ← acumulado + M[i + tamanho, k].
4      acumulado ←
       acumulado + M[i + k − colunaInicial, colunaInicial + tamanho].
5      SM[i + tamanho, k] ← nA.
6      SM[i + k − colunaInicial, colunaInicial + tamanho] ← nA.
7      k ← k + 1.
8  fim
9  acumulado ← acumulado + M[i + tamanho, colunaInicial + tamanho].
10 SM[i + tamanho, colunaInicial + tamanho] ← nA.
11 retorna acumulado.
```

Cada célula somada no Algoritmo 4.10 é marcada com o número do agrupamento informado nas linhas 5, 6 e 10.

O Algoritmo 4.11 é responsável por formar as várias submatrizes de uma mesma linha.

Algoritmo 4.11: $FormarSubMatriz(M, SM, mapSM, colunaLivre, media, i, nA)$

Entrada: M matriz de n linhas e m colunas.
SM matriz com os números dos agrupamentos das submatrizes.
$mapSM$ vetor extensível da lista de submatrizes.
$colunaInicial$ coluna inicial da submatriz que formada.
$media$ média de pesos que um recurso consegue atender.
i linha inicial da submatriz.
nA número do agrupamento atual.

```
1  para colunaFinal ← colunaLivre até m − 1 faça
2      colunaInicial ← colunaFinal.
3      acumulado ← 0.
4      tamanho ← 0.
5      para j ← colunaInicial até m faça
6          se j = colunaInicial então
7              acumulado ← acumulado + M[i, j].
8              tamanho ← 1.
9              SM[i, j] ← nA.
10         senão
11             acumulado ← acumulado +
               AcumuladoLinhaColuna(M, i, j, colunaInicial, tamanho, SM, nA).
12             tamanho ← tamanho + 1.
13             colunaFinal ← colunaFinal + 1.
14         fim
15         deveParar ← DeveParararHD(acumulado, media, i, j, n, m, SM).
16         se deveParar = verdadeiro então
17             mapSM[nA] ← {i, colunaInicial, tamanho, acumulado}.
18             Interromper laço
19         fim
20         j ← j + 1.
21     fim
22     colunaFinal ← colunaFinal + 1.
23     enquanto colunaFinal ⩽ m − 1 e não (SM[i, colunaFinal] = 0 e
       SM[i, colunaFinal + 1] = 0) faça
24         colunaFinal ← colunaFinal + 1.
25     fim
26     nA ← nA + 1.
27 fim
```

Primeiramente, o Algoritmo 4.11 inicia um laço na linha 1, na primeira coluna e linha livre da matriz M. Nas linhas 2 a 4, as variáveis *colunaInicial*, *acumulado* e *tamanho* são inicializadas. A variável *colunaInicial* recebe sempre a última coluna livre encontrada. Neste passo do algoritmo, a última coluna livre está armazenada na variável *colunaFinal*.

Na linha 5 do Algoritmo 4.11, um laço é inicializado para formação de uma submatriz. Este laço começa na coluna inicial e pode ir até a coluna m da matriz M, ou seja, a submatriz pode conter todas as colunas da matriz M.

A condição na linha 6 do Algoritmo 4.11 é verdadeira quando a matriz não possui nenhuma célula. Neste passo, o valor acumulado é incrementado com o valor da primeira célula, é atribuído o valor 1 para o tamanho da submatriz e o número do agrupamento é marcado na matriz SM.

Caso a condição da linha 6 do Algoritmo 4.11 seja falsa, é iniciada uma expansão na diagonal, adicionando-se mais uma linha e uma coluna na submatriz. Após cada expansão, a variável *tamanho* é incrementada, ou seja, ela armazena quantas expansões foram feitas.

Nas linhas 15 a 19 do Algoritmo 4.11, é feita a verificação se as expansões devem parar.

Nas linhas 22 a 25 do Algoritmo 4.11, verifica-se qual a próxima coluna, da mesma linha, onde uma nova submatriz pode ser formada. A condição para formação de uma nova submatriz na mesma linha é:

- Não estar na última coluna, pois neste caso, certamente, será formada uma submatriz de tamanho 1;
- Haja duas colunas adjacentes livres, ou seja, a submatriz possa ter tamanho mínimo igual a 2.

A variável *colunaFinal* do Algoritmo 4.11 é incrementada, na linha 13, a cada expansão da submatriz anterior. Por isto, a busca da próxima posição viável para formação de uma nova submatriz, começa na última coluna da última submatriz formada. Por este motivo, o laço declarado na linha 1 não é incrementado de 1 em 1.

O Algoritmo 4.12 especifica a lógica completa da formação de todas submatrizes apresentadas na Figura 4.12.

Algoritmo 4.12: $HeuristicaDiagonal(M, media)$

Entrada: M matriz de n linhas e m colunas.
$media$ média de pesos que um recurso consegue atender.
Saída: Conjunto com as submatrizes e suas informações.

```
1  Declara uma matriz SM de n linhas e m colunas.
2  Inicializa todas as células da matriz SM com o valor zero.
3  Declara um vetor mapSM extensível para armazenar a lista das submatrizes
   criadas.
4  colunaLivre ← 0.
5  numeroAgrupamento ← 1.
6  para i ← 1 até n − 1 faça
7      FormarSubMatriz(M, SM, mapSM, colunaLivre, media, i, numeroAgrupamento).
8      linhaDisponivel ← falso.
9      enquanto linhaDisponivel = falso e i ⩽ n faça
10         i ← i + 1.
11         se i ⩾ n então
12             Interromper laço.
13         fim
14         para j ← 1 até m − 1 faça
15             se SM[i, j] = 0 e SM[i, j + 1] = 0 então
16                 linhaDisponivel ← verdadeiro.
17                 colunaLivre ← j.
18                 Interromper laço.
19             fim
20             j ← j + 1.
21         fim
22     fim
23 fim
24 retorna {SM, mapSM}.
```

Nas linhas 4 a 5 do Algoritmo 4.12, são feitas as inicializações das variáveis auxiliares. Quando a matriz não possui nenhuma submatriz, a primeira coluna livre sempre será a primeira e o número do primeiro agrupamento será sempre 1.

Na linha 6 do Algoritmo 4.12, um laço é iniciado na primeira linha da matriz até a penúltima.

Nas linhas 8 a 22 do Algoritmo 4.12, é feita a busca da próxima linha disponível. Esta busca só acaba quando é encontrada uma linha com duas colunas consecutivas disponíveis ou quando o busca chega na última linha.

Nota-se que as variáveis *SM*, *mapSM* e *numeroAgrupamento* são alteradas no Algoritmo 4.11.

O Algoritmo 4.13 apresenta como o problema das fronteiras é solucionado.

Algoritmo 4.13: $ProblemaFronteira(M, SM, mapSM)$

Entrada: M matriz de n linhas e m colunas.
SM matriz com os números das submatrizes.
$mapSM$ vetor extensível da lista de submatrizes.
Saída: Submatrizes.

```
1  para i ← 1 até n faça
2      para j ← 1 até m faça
3          se SM[i, j] = 0 então
4              auxI ← i.
5              auxJ ← j.
6              se i < n e j > 1 então
7                  auxJ ← j − 1.
8              senão
9                  auxI ← i − 1.
10             fim
11             SM[i, j] ← SM[auxI, auxJ].
12             numeroAgrupamento ← SM[auxI, auxJ].
13             {i, j, tamanho, acumulado} ← mapSM[numeroAgrupamento].
14             acumulado ← acumulado + SM[i, j].
15             mapSM[numeroAgrupamento] ← {i, j, tamanho, acumulado}.
16         fim
17         j ← j + 1.
18     fim
19     i ← i + 1.
20 fim
21 retorna SM.
```

Quando uma célula é encontrada sem agrupamento no Algoritmo 4.13, verifica-se se esta célula está na última linha ou na primeira coluna. Caso esteja, na linha 9, esta célula é agrupada na submatriz que estiver acima. Caso não esteja, na linha 7, esta célula é agrupada na submatriz que estiver a esquerda.

Nas linhas 11 a 15 do Algoritmo 4.13, os dados das submatrizes em *SM* e *mapSM* são atualizados. Isto é necessário porque a submatriz $SM[auxI, auxJ]$ aumentou sua quantidade de pesos na linha 11.

Finalmente, o Algoritmo 4.14 apresenta a heurística da diagonal completa. Efetuando as distribuições dos recursos de acordo com as submatrizes criadas.

Algoritmo 4.14: $DistribuicaoHeuristicaDiagonal(M, k, media)$

Entrada: M matriz de n linhas e m colunas.
k quantidade de recursos a distribuir.
$media$ média de pesos que um recurso consegue atender.
Saída: Conjunto de pontos S.

```
1  {SM, mapSM} ← HeuristicaDiagonal(M, media).
2  SM ← ProblemaFronteira(M, SM, mapSM).
3  S ← {}.
4  para contador ← 1 até k faça
5      Busca a submatriz que possui maior número acumulado de pesos e
       armazena o resultado na variável indiceMaiorAcumulado.
6      {iAgrupamento, jAgrupamento, tamanho, acumulado} ←
       mapSM[indiceMaiorAcumulado].
7      iMax ← iAgrupamento.
8      jMax ← jAgrupamento.
9      para i ← iAgrupamento até iAgrupamento + tamanho faça
10         para j ← jAgrupamento até jAgrupamento + tamanho faça
11             se M[i, j] > M[iMax, jMax] então
12                 iMax ← i.
13                 jMax ← j.
14             fim
15             j ← j + 1.
16         fim
17         i ← i + 1.
18     fim
19     S ← S ∪ (iMax, jMax).
20     se M[iMax, jMax] > media então
21         M[iMax, jMax] ← M[iMax, jMax] − media.
22     senão
23         M[iMax, jMax] ← 0.
24     fim
25     mapSM[contador] ←
       {iAgrupamento, jAgrupamento, tamanho, acumulado − media}.
26 fim
27 retorna S.
```

Nas linhas 1 e 2 do Algoritmo 4.14, são feitas as chamadas dos algoritmos 4.12 e 4.13. Na linha 3, um conjunto é inicializado como vazio para armazenar os pontos onde os recursos serão alocados.

Nas linhas 4 a 26 do Algoritmo 4.14, é feito um laço para distribuir os k recursos. Neste laço, é criado um algoritmo guloso para distribuir os recursos na submatriz que possuir o maior acumulado.

Nas linhas 6 a 18 do Algoritmo 4.14, é feito uma busca da célula de maior peso da submatriz que possui o maior acumulado.

Na linha 19 do Algoritmo 4.14, o ponto de maior peso da submatriz é adicionado ao conjunto de pontos. Nas linhas 20 a 25, os dados da matriz são alterados para que ocorra uma distribuição de recurso no mesmo lugar somente se for necessário, ou seja, quando o acumulado da submatriz é muito grande em relação aos demais e o peso da célula também.

A Tabela 4.7 apresenta os resultados das simulações feitas com uma implementação em Java do Algoritmo 4.14.

Tabela 4.7: *Simulação com a heurística da diagonal com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	5 milisseg.	17	14,75	86,76%
(3 x 3) - 3	0 milisseg.	54	46,75	86,57%
(4 x 4) - 4	0 milisseg.	89	77,66	87,26%
(5 x 5) - 5	6 milisseg.	115	99,75	86,74%
(6 x 6) - 6	1 milisseg.	161	131,13	81,44%
(7 x 7) - 7	0 milisseg.	240	189,94	79,14%
(8 x 8) - 8	1 milisseg.	319	249,00	78,06%
(9 x 9) - 9	1 milisseg.	426	311,41	73,10%
(10 x 10) - 10	0 milisseg.	499	370,03	74,15%
(11 x 11) - 11	0 milisseg.	617	429,82	69,66%
(12 x 12) - 12	1 milisseg.	728	508,83	69,89%
(13 x 13) - 13	0 milisseg.	860	582,97	67,79%
(14 x 14) - 14	0 milisseg.	987	652,26	66,08%
(15 x 15) - 15	1 milisseg.	1144	715,59	62,55%
(25 x 25) - 25	2 milisseg.	3055	1512,38	49,51%
(30 x 30) - 30	1 milisseg.	4511	1946,88	43,16%
(40 x 40) - 40	3 milisseg.	8131	2901,79	35,69%
(50 x 50) - 50	3 milisseg.	12455	3820,46	30,67%

(100 x 100) - 100	64 milisseg.	50190	8221,70	16,38%
(200 x 200) - 200	119 milisseg.	201054	17107,49	8,51%
(300 x 300) - 300	279 milisseg.	449778	25956,00	5,77%
(400 x 400) - 400	15 milisseg.	799981	34964,69	4,37%
(500 x 500) - 500	25 milisseg.	1251456	43371,98	3,47%
(1000 x 1000) - 1000	102 milisseg.	4999064	88765,14	1,78%

4.8 Heurística da Diagonal Precipitada

A Heurística da Diagonal Precipitada (HDP) é uma variação da Heurística da Diagonal que tenta resolver o problema das fronteiras no momento em que as submatrizes são formadas. No capítulo 6, serão apresentadas comparações entre esta modificação na heurística e a heurística original.

A Tabela 4.8 apresenta a simulação feita com a Heurística da Diagonal Precipitada.

Tabela 4.8: *Simulação com a heurística da diagonal precipitada com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	5 milisseg.	17	14,75	86,76%
(3 x 3) - 3	0 milisseg.	54	40,23	74,51%
(4 x 4) - 4	0 milisseg.	89	71,64	80,50%
(5 x 5) - 5	0 milisseg.	115	96,79	84,17%
(6 x 6) - 6	0 milisseg.	161	126,20	78,39%
(7 x 7) - 7	0 milisseg.	240	181,70	75,71%
(8 x 8) - 8	0 milisseg.	319	227,66	71,37%
(9 x 9) - 9	1 milisseg.	426	313,51	73,60%
(10 x 10) - 10	1 milisseg.	499	373,61	74,87%
(11 x 11) - 11	0 milisseg.	617	444,63	72,06%
(12 x 12) - 12	1 milisseg.	728	508,97	69,91%
(13 x 13) - 13	0 milisseg.	860	582,47	67,73%
(14 x 14) - 14	0 milisseg.	987	640,35	64,88%

(15 x 15) - 15	0 milisseg.	1144	718,62	62,82%
(25 x 25) - 25	1 milisseg.	3055	1479,33	48,42%
(30 x 30) - 30	2 milisseg.	4511	1917,45	42,51%
(40 x 40) - 40	12 milisseg.	8131	2821,62	34,70%
(50 x 50) - 50	4 milisseg.	12455	3706,67	29,76%
(100 x 100) - 100	63 milisseg.	50190	8169,37	16,28%
(200 x 200) - 200	71 milisseg.	201054	16914,61	8,41%
(300 x 300) - 300	149 milisseg.	449778	25872,00	5,75%
(400 x 400) - 400	45 milisseg.	799981	34728,16	4,34%
(500 x 500) - 500	52 milisseg.	1251456	43732,70	3,49%
(1000 x 1000) - 1000	336 milisseg.	4999064	88222,06	1,76%

Neste capítulo foram criados algoritmos que, com base nas definições formuladas, no Capítulo 3, tentam resolver o problema da distribuição, na grande maioria, utilizando uma heurística. O capítulo a seguir apresenta a modelagem de um SMA para melhorar o desempenho do algoritmo criado na seção 4.5.

CAPÍTULO 5

Sistema Multiagente

Este capítulo propõe um Sistema Multiagente (SMA) [13] para o problema da distribuição de recursos. Pelo fato da modelagem e implementação de um SMA, na grande maioria das vezes, ser grande e complexa, foi dedicado este capítulo inteiro a este SMA.

O SMA que será apresentado neste capítulo utiliza o algoritmo 4.5, apresentado no Capítulo 4, como cerne dos agentes do SMA.

O SMA será modelado utilizando a metodologia Prometheus [21, página 21]. Os diagramas da metodologia Prometheus serão criados utilizando a ferramenta Prometheus Design Tool [18].

Será utilizada a plataforma JADE (*Java Agent DEvelopment Framework*) [9] para a implementação dos agentes. Nota-se que não existe nenhum conflito entre a metodologia Prometheus, por ter base na arquitetura BDI (*Belief-Desire-Intention*), com a implementação na plataforma JADE, segundo Padgham e Winikoff [22, página 3].

A metodologia Prometheus consiste em três fases [22, página 3]:

- Especificação do sistema: descrição dos objetivos, cenários e funcionalidades; a descrição das interfaces do sistema para o ambiente é feita em termos de ações, percepções e dados externos.
- Projeto arquitetural: identificação dos tipos de agentes; a estrutura do sistema é capturada em diagramas de alto nível; cenários são desenvolvidos em protocolos de interação.
- Projeto detalhado: os detalhes internos de cada agente são especificados e definidos em termos de habilidades, dados, eventos e planos; diagramas de processos são utilizados para sair dos protocolos de interação e chegar nos planos.

5.1 SMA de Distribuição Gulosa Dinâmica

Na seção 4.5, do Capítulo 4, foi apresentado um algoritmo guloso que utiliza conceitos de programação dinâmica e com um raio de influência limitado. Esse algoritmo

pode ser implementado na forma de um SMA, onde cada agente é responsável por calcular a melhor escolha de uma subárea do atendimento.

5.1.1 Especificação do sistema

A distribuição da busca da melhor posição onde o recurso deve ser colocado não reduz a qualidade do algoritmo guloso, se cada agente possuir as informações da área de atendimento da seguinte forma:

- A área de atendimento deve ser dividida em função do número de agentes, ou seja, a matriz M será dividia em nAg (número de agentes) partes;
- A divisão da área de atendimento deverá ser realizada com redundância nas extremidades da subárea de cada agente;
- A redundância de informações da área de atendimento deverá ser igual ao tamanho do raio de influência.

Por exemplo, uma matriz M com 10 linhas e 10 colunas pode ser dividida em 4 partes. A melhor posição de cada parte pode ser calculada por um agente. A Figura 5.1 apresenta como seria feita esta divisão.

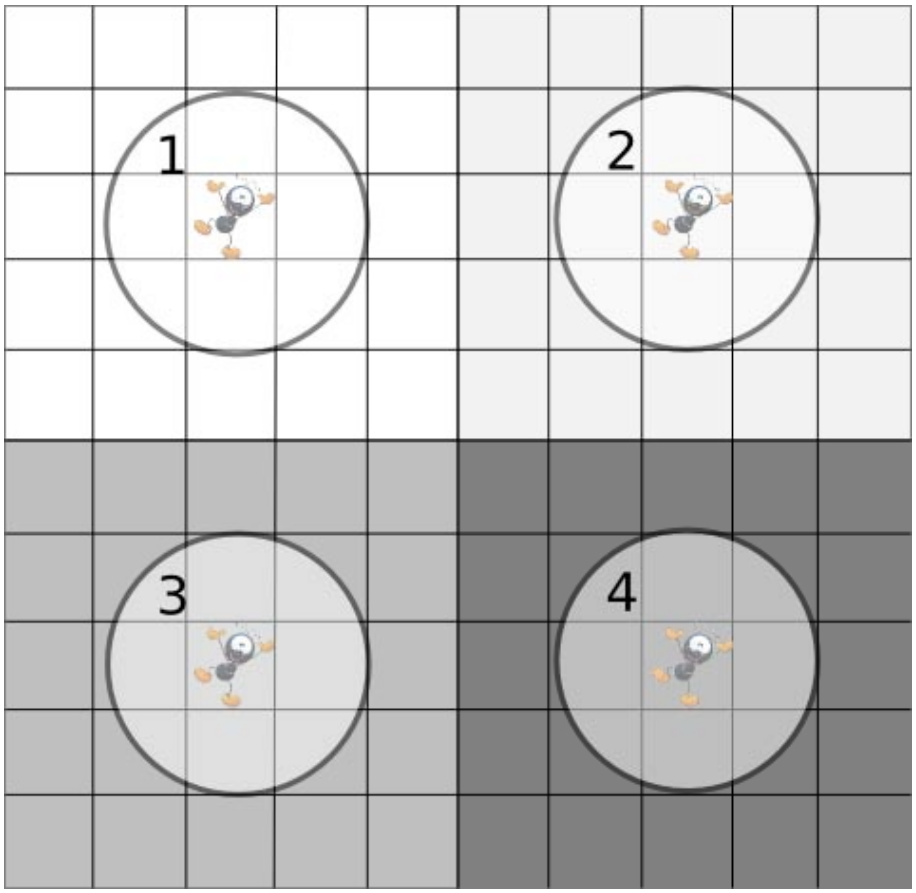


Figura 5.1: *Área de atendimento e as subáreas de cada agente do SMA.*

Na Figura 5.1, cada formiga representa um agente do SMA e as tonalidades das cores da matriz representam a área na qual o agente irá buscar a melhor posição para o recurso. Se o agente possuir apenas os pesos das células de sua subárea de atendimento, a precisão do algoritmo guloso irá diminuir porque o agente não saberá a influência correta quando estiver nas extremidades da submatriz.

Para resolver o problema do cálculo da influência nas extremidades da subárea de atendimento, deve-se informar para o agente uma matriz maior que sua subárea de busca. Por exemplo, se a distância máxima de influência, calculada pelo algoritmo 4.2, for igual a 2, então a matriz de cada agente deve ser aumentada em 2 linhas na extremidade superior e inferior e 2 colunas na extremidade esquerda e direita. A Figura 5.2 apresenta como esse aumento é realizado.

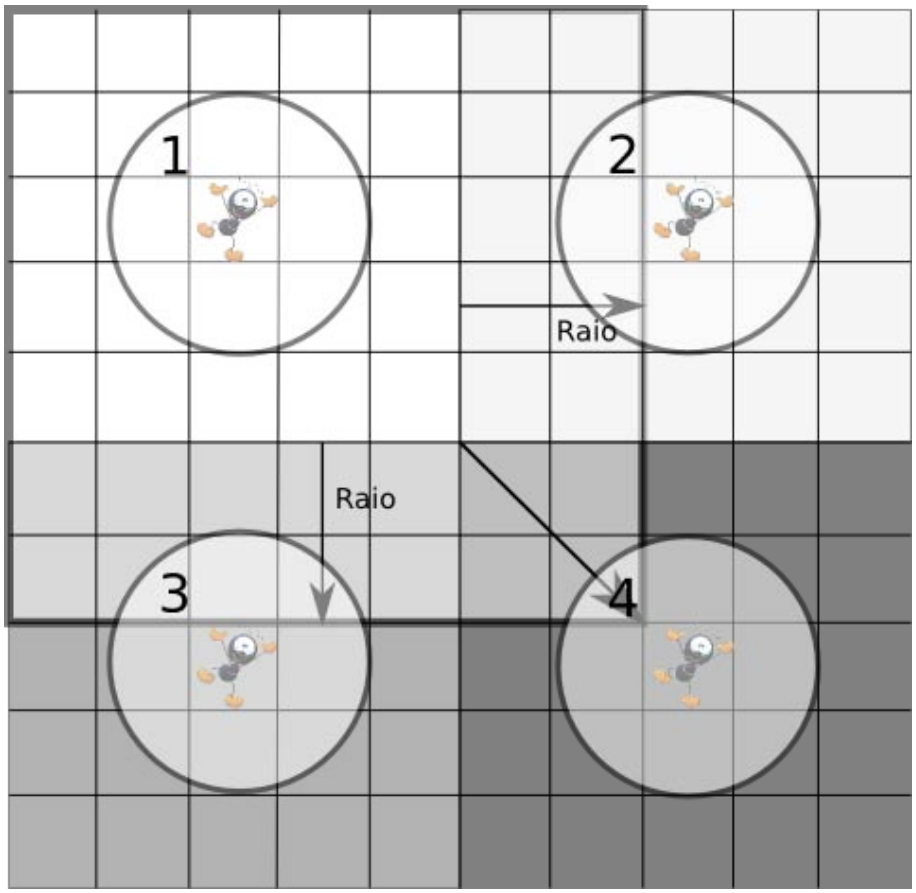


Figura 5.2: *Aumento da subárea do primeiro agente para evitar o problema do cálculo nas extremidades.*

A Figura 5.2 apresenta que, a matriz informada para o primeiro agente foi aumentada em duas unidades na diagonal, ou seja, duas unidades para direita e duas unidades para baixo. Não houve um aumento para cima e para esquerda porque a área de atendimento não possui dados nessas posições.

A mesma lógica acontece para o segundo, terceiro e quarto agente. Para o segundo agente, a matriz é aumentada para esquerda e para baixo. Para o terceiro agente,

a matriz é aumentada para direita e para cima. E, finalmente, para quarto agente, a matriz é aumentada para esquerda e para cima.

Mesmo possuindo uma matriz maior do que a inicial, os agentes não podem ultrapassar a área de busca. Por exemplo, o primeiro agente só pode calcular o melhor ponto até a linha 5 e coluna 5. Isto evita que os agentes respondam posições iguais na matriz M.

Objetivos

Essa especificação, feita na seção anterior, de como o sistema deve funcionar, é descrição do objetivo do sistema [21, página 34] na metodologia Prometheus. Após esta descrição, pode-se destacar o objetivo do SMA proposto:

- Distribuir recursos;
  - Encontrar melhor posição na matriz M';

Cenários

Com o objetivo do sistema, pode-se descrever os cenários de funcionamento [21, página 43]. A Tabela 5.1 apresenta o cenário de distribuição de recursos no qual um agente distribuidor coordena os agentes assistentes para distribuição dos k recursos na matriz M.

Tabela 5.1: *Cenário de Distribuição de Recursos.*

Cenário de Distribuição de Recursos	
Gatilho	Solicitação do usuário.
Descrição	Quando um usuário solicita a distribuição de k recursos utilizando x agentes, o sistema instancia os x agentes e divide a matriz M em várias submatrizes M'.
Passos	

1. ACTION: Criar x agentes assistentes
2. ACTION: Dividir a matriz M em x partes M';
3. GOAL: Encontrar melhor posição em M'
4. OTHER: Esperar melhores posições e respectiva avaliação de todos x agentes assistentes
5. PERCEPT: Melhores posições e avaliações recebidas
6. ACTION: Escolher a melhor posição dentre as melhores posições recebidas
7. OTHER: Alocar um recurso na melhor posição
8. OTHER: Atualizar a lista de posições e os dados da matriz M
9. OTHER: Repetir o processo a partir do passo 2 até distribuir os k recursos

A Tabela 5.2 apresenta o cenário no qual um agente assistente deve encontrar a melhor posição na submatriz.

Tabela 5.2: *Cenário de Encontrar Melhor Posição em M'.*

Cenário de Encontrar Melhor Posição em M'	
Gatilho	Solicitação do agente distribuidor.
Descrição	O agente distribuidor solicita ao agente assistente que encontre a melhor posição na submatriz M' utilizando o algoritmo guloso dinâmico.
Passos	
1. OTHER: Agente criado 2. OTHER: Aguardando dados da submatriz M' 3. PERCEPT: Dados da submatriz M' e área de busca recebida 4. OTHER: Utilizar o Algoritmo 4.7 na matriz M' informada; 5. ACTION: Informar melhor posição e respectiva avaliação	

Com os cenários, pode-se detalhar as percepções, ações e dados do SMA [21, página 47]. A Tabela 5.3 apresenta as percepções e ações do SMA.

Tabela 5.3: *Percepções e ações.*

A seguir a lista de percepções e ações do SMA de distribuição gulosa dinâmica distribuída:

Percepções
• Solicitação de distribuição de k recursos na matriz M com x agentes assistentes • Solicitação de cálculo para os assistentes • Solicitação de término de assistentes • Finalização da distribuição
Ações
• Distribuir k recursos

O SMA proposto possui os dados: a matriz M, a quantidade k de recursos que pretende-se distribuir e a quantidade x de agentes assistentes que devem ser utilizados. Nota-se que a quantidade x poderia ser obtida automaticamente através da análise de quantos processadores estão disponíveis na plataforma do SMA.

5.1.2 Projeto Arquitetural

O agrupamento de objetivos similares se tornam funcionalidades. As funcionalidades e os cenários podem ser avaliados de acordo com normas da engenharia de *software* sobre acoplamento e coesão.

Nos sistemas multiagentes, pode-se utilizar as definições de acoplamento e coesão da seguinte forma [21, página 55]:

- Acoplamento: é apresentado principalmente na comunicação entre agentes, mas também pode ser apresentado no uso compartilhado de dados;
- Coesão: capacidade de controlar várias ações que ocorrem ao mesmo tempo e a forma como objetivos do agente são relacionados.

Os diagramas de acoplamento de dados e de relacionamento dos agentes são ferramentas utilizadas para avaliar essas caraterísticas. Quando dois agentes se comunicam, pode-se utilizar o diagrama de interações de agentes para descrever as possíveis formas de interação e protocolos utilizados. Finalmente, existe um diagrama de visão geral que apresenta todas estas características juntas.

O diagrama de visão geral é considerado um dos mais importantes artefatos do projeto. Esse artefato junta informações sobre agentes, dados, entradas externas, saídas e a comunicação entre os agentes.

Nesse SMA, existem dois objetivos que foram o cenário de distribuição de recursos. Por se tratar de um SMA pequeno, a coesão é alta, mas em consequência tem-se um alto acoplamento.

Agentes

A primeira decisão a ser tomada no projeto arquitetural é os tipos de agentes que serão utilizados. Para decidir os tipos de agentes, deve-se agrupar funcionalidades similares e os dados utilizados nas funcionalidades.

A Figura 5.3 apresenta o diagrama de acoplamento de dados.

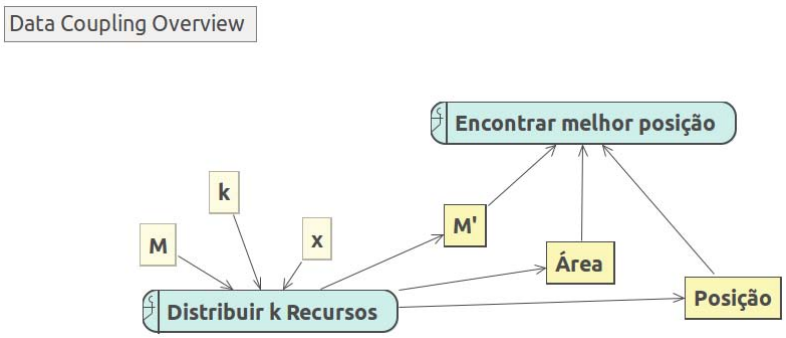


Figura 5.3: *Diagrama de acoplamento de dados.*

Na Figura 5.3, são apresentadas três fontes externas de dados que são: M, k e x. A funcionalidade "Distribuir k recursos" é responsável pela criação dos dados: a submatriz M' e o dado *Área* que determina o lugar de busca na submatriz M'.

Nesse SMA, serão criados dois agentes. O primeiro agente é o distribuidor guloso. Esse agente é responsável pela coordenação dos agentes assistentes gerenciando o cenário 5.1. O segundo agente é o assistente guloso. Esse agente é responsável por encontrar o melhor ponto numa submatriz M', apresentado no cenário da Tabela 5.2.

A Tabela 5.4 apresenta a descrição [21, página 66] do agente distribuidor guloso.

Tabela 5.4: *Descrição do agente distribuidor guloso.*

Nome	Distribuidor Guloso.
Descrição	Responsável pelo cálculo das melhores posições para os k recursos na matriz M e gerenciamento dos x assistentes.
Cardinalidade	Um por instância do problema.
Tempo de vida	Instanciado quando é feita uma solicitação direta. Morre quando os k recursos são distribuídos.
Inicialização	Obtém matriz.

Morte do agente	Finaliza os x agentes assistentes criados.
Objetivos	Distribuir recursos.
Usa os dados	Matriz M, número k de recursos e número x de assistentes.
Produz os dados	Melhores pontos de alocação dos k recursos.
Percepções respondidas	Solicitação do usuário.
Ações	Distribuir k recursos.
Protocolos e interações	Solicitação de Cálculo, Solicitação de Cálculo do Assistente, Finalização de Cálculo e Solicitação de Término de Assistente.

A Tabela 5.5 apresenta a descrição do agente assistente guloso.

Tabela 5.5: *Descrição do agente assistente guloso.*

Nome	Assistente Guloso.
Descrição	Responsável por encontrar o melhor ponto na submatriz M' dado os recursos já distribuídos.
Cardinalidade	número x informado por distribuidor.
Tempo de vida	Instanciado pelo distribuidor. Morre quando o distribuidor solicita.
Inicialização	Obtém matriz M'.
Objetivos	Encontrar melhor posição em M'.
Usa os dados	Matriz M', (io, jo) e (if, jf) retângulo que irá calcular, e S posições já determinadas.
Produz os dados	Melhor posição na matriz M'.
Percepções respondidas	Solicitação do distribuidor.
Ações	Encontrar melhor posição em M'.
Protocolos e interações	Solicitação de Cálculo do Assistente e Solicitação de Término de Assistente.

Percepções e Ações

Após a descrição dos agentes do SMA, deve-se descrever as percepções e ações destes agentes. As tabelas 5.6, 5.7, 5.8 e 5.9 apresentam as descrições das percepções do

SMA.

Tabela 5.6: *Percepção de Solicitação de Cálculo.*

Nome	Solicitação de Cálculo.
Descrição	Ocorre quando um programa instancia e solicita a um agente distribuidor o cálculo das k melhores posições onde os recursos devem ser distribuídos.
Informação transmitida	A matriz M que representa a área de atendimento, a quantidade k de recursos que pretende-se alocar, a quantidade x de assistentes que devem ser utilizados e a distância máxima (raio).
Conhecimentos atualizados	Todos os dados informados e o estado "aguardando resposta" do solicitante.
Origem	A informação é obtida através de uma mensagem.
Processamento	A mensagem é enviada para o agente distribuidor.
Responder para os agentes	Após o cálculo, deve ser respondido ao agente solicitante a lista dos melhores pontos encontrados para distribuição dos recursos.
Frequência esperada	Toda vez que um agente distribuidor for instanciado.

Tabela 5.7: *Percepção de Finalização de Cálculo.*

Nome	Finalização de Cálculo.
Descrição	Ocorre quando as respostas dos assistentes chegam na busca da k-ésima melhor posição.
Informação transmitida	Melhores posições.
Conhecimentos atualizados	Todos os dados informados.
Origem	A informação é obtida através de várias mensagens dos assistentes.
Processamento	A mensagem é enviada para o agente distribuidor.
Responder para os agentes	Após escolher a melhor posição, enviar mensagens solicitando o término dos assistentes (ver Tabela 5.9).
Frequência esperada	Uma vez após distribuir os k recursos.

Tabela 5.8: *Percepção de Solicitação de Cálculo do Assistente.*

Nome	Solicitação de Cálculo do Assistente.
Descrição	Ocorre após a criação do assistente. O agente distribuidor guloso envia os dados para o assistente calcular.
Informação transmitida	A matriz M' que representa a subárea de atendimento, a área de busca dentro da submatriz M', a distância máxima (raio) e a posição da matriz M' na matriz M.
Conhecimentos atualizados	Todos os dados informados e o estado "aguardando melhor ponto" do agente distribuidor.
Origem	A informação é obtida através de uma mensagem.
Processamento	A mensagem é enviada para o agente assistente.
Responder para os agentes	Após o cálculo, deve ser respondido ao agente distribuidor guloso a melhor posição na matriz M'.
Frequência esperada	k vezes para cada agente assistente do agente distribuidor guloso.

Tabela 5.9: *Percepção de Solicitação de Término de Assistente.*

Nome	Solicitação de Término de Assistente.
Descrição	Ocorre quando o agente distribuidor guloso termina a distribuição dos k recursos.
Informação transmitida	Solicitação de término.
Conhecimentos atualizados	O agente distribuidor mudou de estado para "concluído".
Origem	A informação é obtida através de uma mensagem do distribuidor.
Processamento	A mensagem é enviada para o agente assistente.
Responder para os agentes	Responder recebimento de mensagem para o agente distribuidor.
Frequência esperada	Uma vezes para cada agente assistente do agente distribuidor guloso.

As tabelas 5.10, 5.11, 5.12 e 5.13 apresentam as ações do SMA.

Tabela 5.10: *Ação de criação de x agentes assistentes.*

Nome	Criar x agentes assistentes.
Descrição	Esta ação é responsável pela criação de x agentes assistentes para o agente distribuidor.
Parâmetros	número x de assistentes.
Duração	Instantânea.
Detecção de Falha	A forma de detectar se tudo ocorreu bem é enviando uma mensagem ao agente assistente e aguardar uma resposta.
Alteração Parcial	O agente assistente não estará registrado na plataforma.
Efeitos colaterais	O cálculo não irá averiguar uma determinada área da matriz M.

Tabela 5.11: *Ação de dividir a matriz M em x partes.*

Nome	Dividir matriz M em x partes chamadas M'.
Descrição	Divide a matriz em x partes, as quais serão enviadas para os assistentes.
Parâmetros	número x de assistentes.
Duração	Instantânea.
Detecção de Falha	Falha se a matriz não foi dividida.
Alteração Parcial	As matrizes M' não foram dimensionadas.
Efeitos colaterais	Não é possível distribuir o cálculo.

Tabela 5.12: *Ação de escolha da melhor posição entre as melhores recebidas.*

Nome	Escolher melhor posição dentre as melhores posições recebidas.
Descrição	Após receber as melhores posições de cada assistente, deve-se buscar a melhor posição entre elas através da avaliação de cada posição.
Parâmetros	número x de assistentes.
Duração	Instantânea.
Detecção de Falha	Verificar se está dimensionado a melhor posição.

Alteração Parcial	Não foi encontrada nenhuma posição.
Efeitos colaterais	Não serão distribuídos os k recursos.

Tabela 5.13: *Ação de informar melhor posição e respectiva avaliação.*

Nome	Informar melhor posição e respectiva avaliação.
Descrição	Após o cálculo do assistente é enviada uma mensagem com a melhor posiçaõ.
Parâmetros	submatriz M' e área de busca.
Duração	Instantânea.
Detecção de Falha	Verificar se está dimensionado a melhor posição.
Alteração Parcial	Não foi encontrada nenhuma posição.
Efeitos colaterais	A área de busca não será avaliada.

Diagramas

A Figura 5.4 apresenta o diagrama da visão geral dos objetivos.

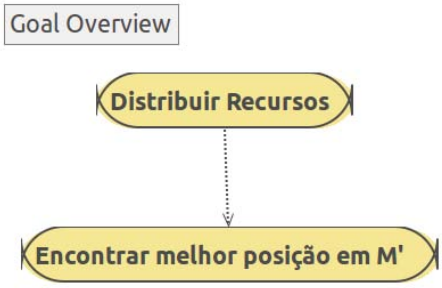


Figura 5.4: *Diagrama da visão geral dos objetivos.*

Na Figura 5.4, são apresentados dois objetivos. O objetivo "Encontrar melhor posição em M'" é um subobjetivo de "Distribuir Recursos".

A Figura 5.5 apresenta o diagrama da visão geral das funcionalidades do sistema.

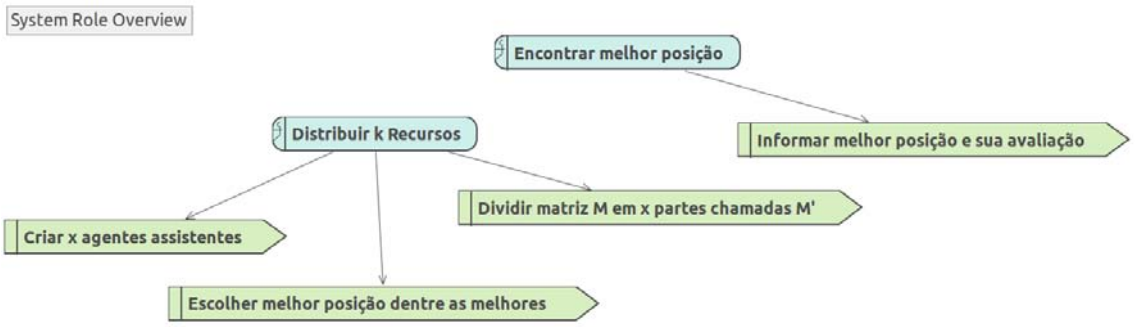


Figura 5.5: *Diagrama da visão geral das funcionalidades do sistema.*

Na Figura 5.5, cada funcionalidade está vinculada à ações. Foram definidas quatro ações:

- "Criar x agentes assistentes" (Tabela 5.10);
- "Escolher melhor posição dentre as melhores" (Tabela 5.12);
- "Dividir matriz M em x partes chamadas M'" (Tabela 5.11);
- "Informar melhor posição e sua avaliação" (Tabela 5.13).

A Figura 5.6 apresenta o agrupamento das funcionlidades para cada agente.

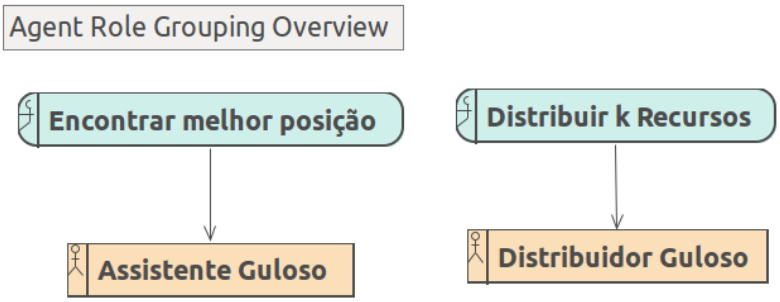


Figura 5.6: *Diagrama do agrupamento funcionalidades por agentes.*

Na Figura 5.6, o agente "Assistente Guloso" é responsável pela funcionalidade "Encontrar melhor posição" e o agente "Distribuidor Guloso" é responsável pela funcionalidade "Distribuir k Recursos".

A Figura 5.7 apresenta o diagrama da visão geral do sistema.

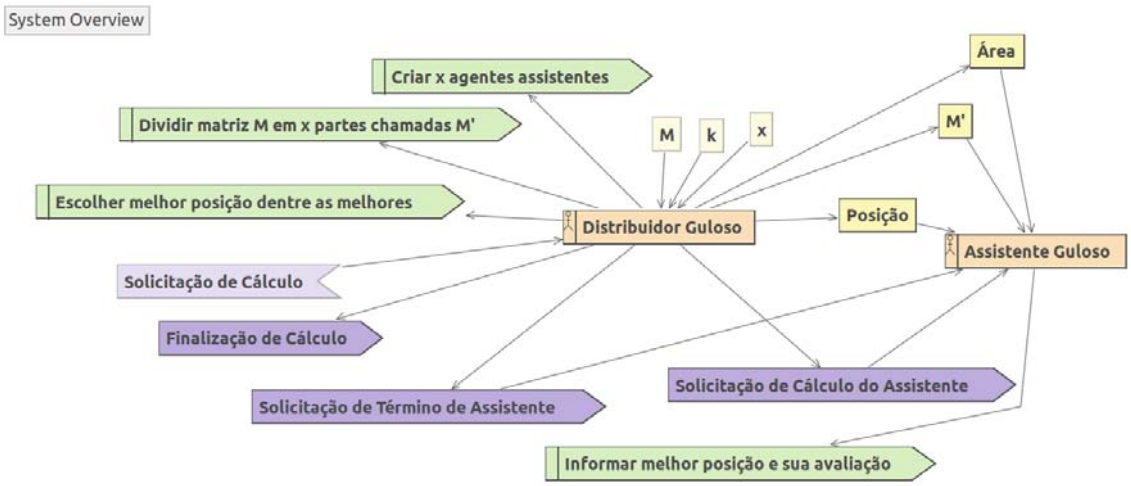


Figura 5.7: *Diagrama da visão geral do SMA.*

Na Figura 5.7, foram apresentados os agentes "Distribuidor Guloso" e "Assistente Guloso" descritos nas tabelas 5.4 e 5.5. O agente distribuidor guloso está vinculado a três ações: "Criar x agentes assistentes", "Dividir matriz M em x partes chamadas M" e "Escolher melhor posição dentre as melhores", enquanto que o agente assistente guloso está vinculado a apenas uma ação "Informar melhor posição e sua avaliação".

O agente distribuidor guloso efetua a leitura dos dados:

- M: matriz que representa a área de atendimento;
- k: quantidade de recursos que pretende-se distribuir;
- x: quantidade de assistentes que devem ser utilizados.

Esses três dados são providos por um sistema externo, por isto são representados de cores e contornos mais claros que os demais.

A partir da leitura dos dados e de algumas ações, o agente distribuidor guloso produz os dados:

- M': submatriz que o agente assistente irá utilizar;
- *Área*: representa a área onde o assistente deverá realizar os cálculos;
- *Posição*: representa a posição da primeira célula da submatriz M' na matriz M.

Para cada cálculo, será utilizado uma instância do agente distribuidor guloso e x instâncias do agente assistente guloso. Por isso, serão geradas x submatrizes M', x áreas de busca e x posições para cada solicitação de cálculo dos assistentes.

"Solicitação de Cálculo" representa uma percepção. Nesse caso, uma mensagem vinda de um agente externo a esse SMA. "Finalização de Cálculo", "Solicitação de Término de Assistente" e "Solicitação de Cálculo do Assistente" representam mensagens trocadas entre os agentes.

5.1.3 Projeto Detalhado

Após a definição do projeto arquitetural, pode-se iniciar o projeto detalhado. Nessa fase, são detalhadas:

- as capacidades dos agentes através de diagramas de visão geral de agentes e descritores das capacidades;
- os diagramas de processos através dos protocolos de interação;
- os planos para alcançar os objetivos.

Capacidades

Capacidade é uma forma de estruturação semelhante a módulos [22, página 7]. Uma capacidade pode conter planos, dados, eventos e também outras capacidades. Conter outras de si permite a divisão em capacidades menores facilitando a compreensão.

Em primeiro momento, cada funcionalidade pode se tornar uma capacidade. Após a criação de todas capacidades, elas podem ser agrupadas em capacidades mais gerais de acordo com a coesão entre elas.

Os diagramas de visão geral de agentes mostram as capacidades, funcionalidades e dados que cada agente contém ou interage. Esse diagrama é similar ao diagrama de visão geral do sistema exceto porque não aparecem agentes e normalmente não aparecem protocolos [22, página 7].

A Figura 5.8 apresenta o diagrama de visão geral do agente distribuidor guloso.

Figura 5.8: *Diagrama da visão geral do agente distribuidor guloso.*

Na Figura 5.8, foram apresentadas duas capacidades. A primeira capacidade, "Combinação de Cálculo com o Distribuidor", agrega os dados informados pelo solici-

tante do cálculo das k melhores posições e as mensagens de solicitação e finalização. A segunda capacidade, “Combinação de Cálculo com Assistentes”, agrega mensagens trocadas entre o distribuidor e os assistentes, ações dos dois agentes e os dados trocados entre eles durante o cálculo.

A Figura 5.9 apresenta o diagrama de visão geral do agente assistente guloso.

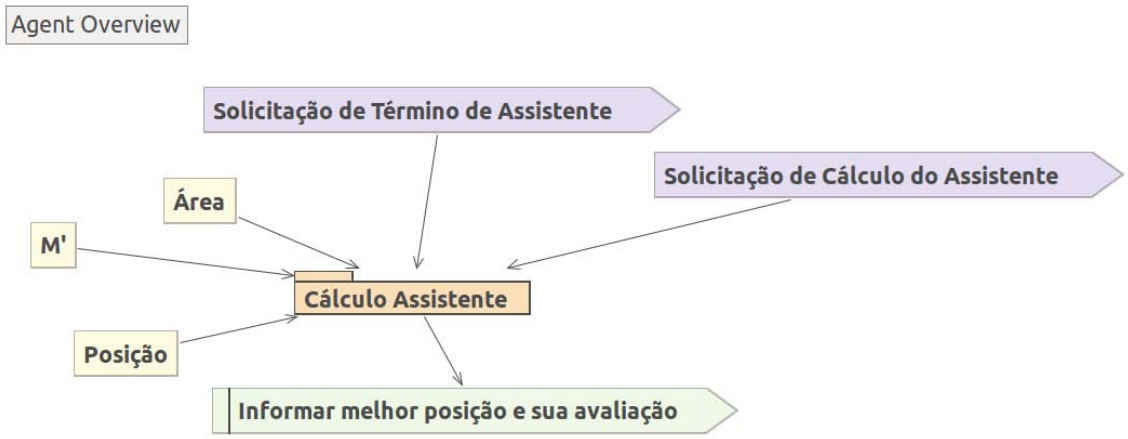


Figura 5.9: *Diagrama da visão geral do agente assistente guloso.*

Na Figura 5.9, foi apresentada uma capacidade chamada “Cálculo Assistente” que agrega as mensagens e dados trocados com o agente distribuidor guloso e a ação de informar o melhor ponto encontrado pelo agente assistente guloso.

A Tabela 5.14 apresenta o relacionamento das capacidades e dos objetivos que pretendem atender.

Tabela 5.14: *Objetivos alcançados por cada capacidade.*

Capacidade	**Objetivos**
Combinação Cálculo com Distribuidor	Distribuir Recursos
Combinação de Cálculo com Assistentes	Distribuir Recursos
Cálculo Assistente	Encontrar melhor posição em M

5.2 Detalhes da Implementação

Nesse trabalho, foi utilizada a versão 4.0 da plataforma JADE. Segundo Bellifemine et al, a plataforma JADE é constituída de contêineres que podem ser distribuídos numa grade computacional [9, página 49].

Para criar um agente na plataforma JADE deve-se criar uma classe que herda da classe *jade.core.Agent* [9, página 68].

Nesse SMA, foi criado um agente distribuidor guloso que recebeu o identificador [9, página 69] “DistribuidorGuloso” e, para cada agente deste tipo, são criados *x*, número informado na solicitação, agentes assistentes. Os agentes assistentes receberam os identificadores “DistribuidorGuloso_assistente_*i*”, onde *i* é o número do assistente, variando de 1 até *x*.

Também foi criado um agente chamado “Solicitador” para facilitar a comunicação com o agente distribuidor guloso.

A Figura 5.10 apresenta um diagrama de classes com os três tipos de agentes criados neste SMA.

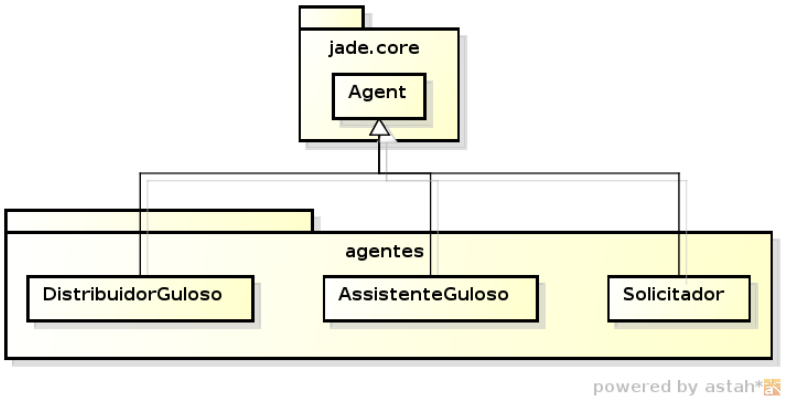


Figura 5.10: *Diagrama de classe dos agentes.*

Cada agente possui vários comportamentos [9, página 74] que refletem o estado atual do agente. Esses comportamentos representam as percepções e ações do agente num determinado momento.

A Figura 5.11 apresenta os comportamentos do agente distribuidor.

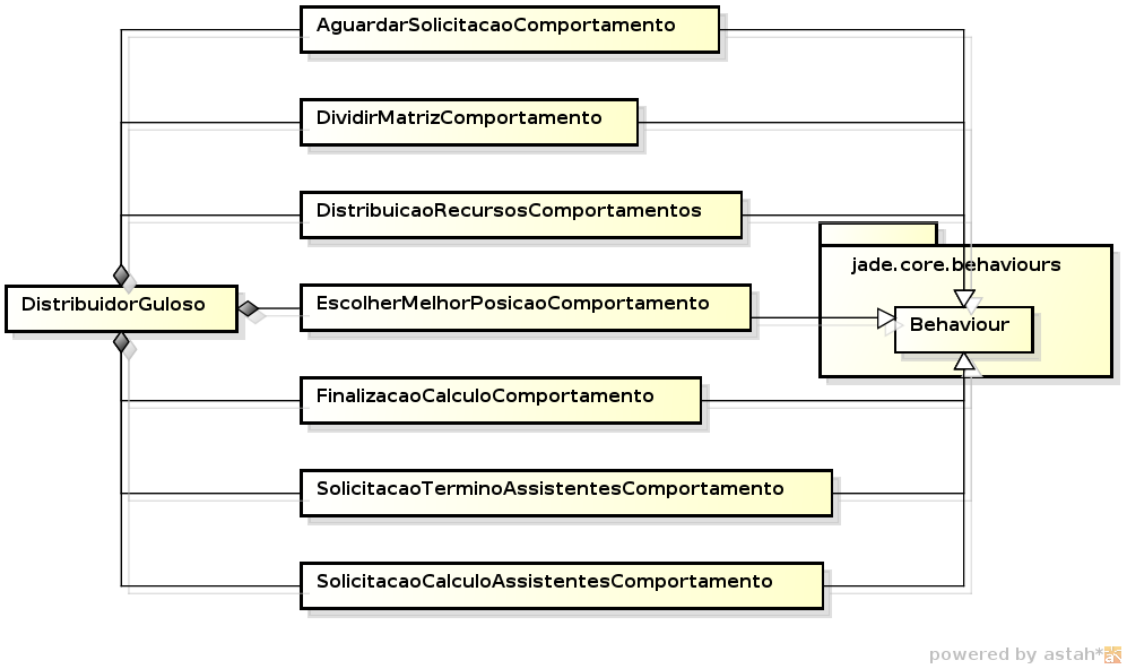


Figura 5.11: *Diagrama de classe dos comportamentos do agente distribuidor guloso.*

Na Figura 5.11, são apresentados sete comportamentos que herdam da classe *jade.core.behaviours.Behaviour*. Esses comportamentos são criados e destruídos durante a execução do agente distribuidor guloso.

A Figura 5.12 apresenta os comportamentos do agente assistente guloso.

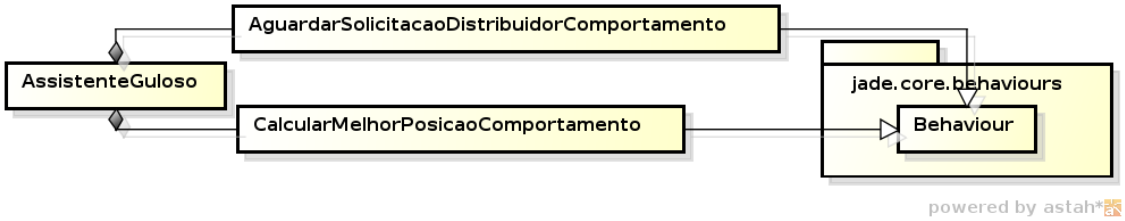


Figura 5.12: *Diagrama de classe dos comportamentos do agente assistente guloso.*

Na Figura 5.12, são apresentados dois comportamentos. O primeiro, *AguardarSolicitacaoDistribuidorComportamento*, representa a percepção da solicitação do distribuidor guloso. O segundo, *CalcularMelhorPosicaoComportamento*, representa a ação de cálculo do subproblema.

Os agentes, distribuidor e guloso, partilham uma ontologia [9, página 94] que permite e padroniza a comunicação através do envio de mensagens [9, página 83] [9, página 84]. A Figura 5.13 apresenta a ontologia criada.

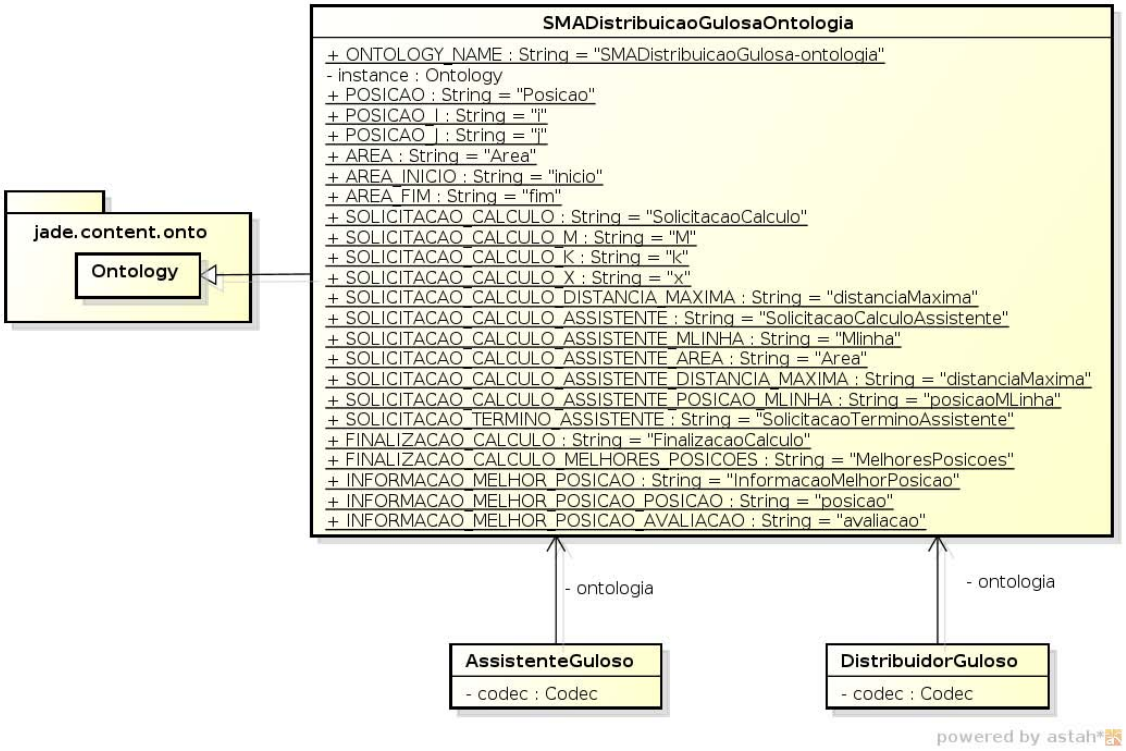


Figura 5.13: *Diagrama de classe da classe principal da ontologia.*

A classe *SMADistribuicaoGulosaOntologia* apresentada na Figura 5.13 representa a ontologia e os literais [9, página 99] que irão compor as mensagens trocadas entre os agentes.

A Figura 5.14 apresenta o diagrama de classe das entidades que compõem a ontologia.

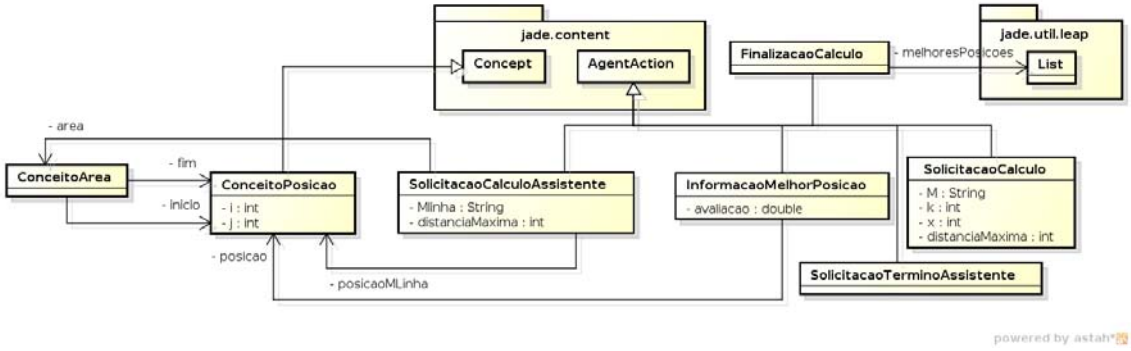


Figura 5.14: *Diagrama de classe das entidades que compõem a ontologia.*

Na Figura 5.14, são apresentados dois conceitos [9, página 101], as classes *ConceitoArea* e *ConceitoPosicao*.

Também são apresentadas cinco ações, as classes *SolicitacaoCalculoAssistente*, *InformacaoMelhorPosicao*, *SolicitacaoTerminoAssistente*, *SolicitacaoCalculo* e *Finaliza-*

caoCalculo. Cada uma destas classes representam ações de um agente e percepções dos destinatários.

Nas classes *SolicitacaoCalculoAssistente* e *SolicitacaoCalculo*, as matrizes são literais encapsulados no formato JSON [10].

Para possibilitar a troca de mensagens através da ontologia criada é necessário registrá-la, conforme apresentado por Bellifemine et al [9, página 105].

O envio e a recepção das mensagens entre os agentes é feita de acordo com o apresentado por Bellifemine et al [9, página 105 e 106].

Além de padronizar as mensagens trocadas entre os agentes, a utilização da ontologia facilita que outros agentes sejam criados para se comunicarem com os agentes distribuidores e assistentes.

5.3 Simulações

A Tabela 5.15 apresenta o tempo gasto na execução, a quantidade de recursos utilizados e a nota da função de avaliação obtidos através da execução do SMA com 2 agentes.

Tabela 5.15: *Simulação do SMA com 2 agentes e um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	324 milisseg.	17	16	94,12%
(3 x 3) - 3	149 milisseg.	54	48	88,89%
(4 x 4) - 4	104 milisseg.	89	78,28	87,96%
(5 x 5) - 5	173 milisseg.	115	100,43	87,33%
(6 x 6) - 6	183 milisseg.	161	133,53	82,94%
(7 x 7) - 7	226 milisseg.	240	192,23	80,10%
(8 x 8) - 8	395 milisseg.	319	251,39	78,81%
(9 x 9) - 9	391 milisseg.	426	326,44	76,63%
(10 x 10) - 10	439 milisseg.	499	383,73	76,90%
(11 x 11) - 11	406 milisseg.	617	454,75	73,70%
(12 x 12) - 12	580 milisseg.	728	526,52	72,32%

(13 x 13) - 13	599 milisseg.	860	604,84	70,33%
(14 x 14) - 14	606 milisseg.	987	672,25	68,11%
(15 x 15) - 15	743 milisseg.	1144	752,59	65,78%
(25 x 25) - 25	1 seg. 916 miliseg.	3055	1596,66	52,26%
(30 x 30) - 30	2 seg. 393 miliseg.	4511	2118,20	46,96%
(40 x 40) - 40	3 seg. 732 miliseg.	8131	3173,77	39,03%
(50 x 50) - 50	5 seg. 70 miliseg.	12455	4077,98	32,74%
(100 x 100) - 100	27 seg. 187 miliseg.	50190	9238,11	18,41%
(200 x 200) - 200	3 min. 25 seg. 521 miliseg.	201054	19621,71	9,76%
(300 x 300) - 300	12 min. 4 seg. 758 miliseg.	449778	29884,18	6,64%
(400 x 400) - 400	29 min. 55 seg. 682 miliseg.	799981	40360.92	5,05%

A Tabela 5.15 mostra que em relação a Tabela 4.4, os tempos de 2 a 40 recursos do SMA foram superiores ao do algoritmo guloso dinâmico, apresentado na seção 4.5, mas a partir de 50 recursos, os tempos são inferiores e se aproximam da metade do tempo gasto pelo algoritmo guloso dinâmico.

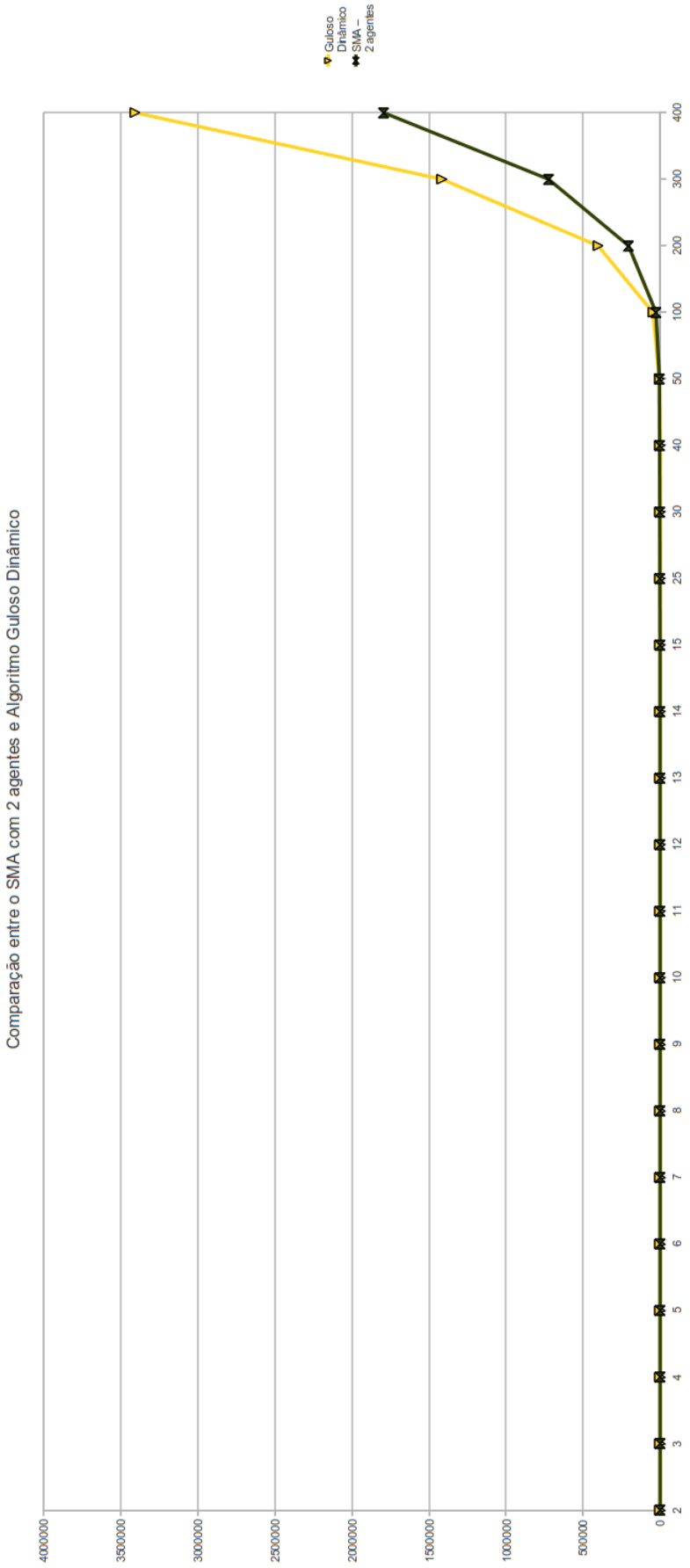


Figura 5.15: *Comparação entre os tempos do SMA com 2 agentes e o algoritmo guloso dinâmico.*

A Figura 5.16 apresenta a comparação entre a qualidade dos resultados do SMA com 2 agentes e do algoritmo guloso dinâmico.

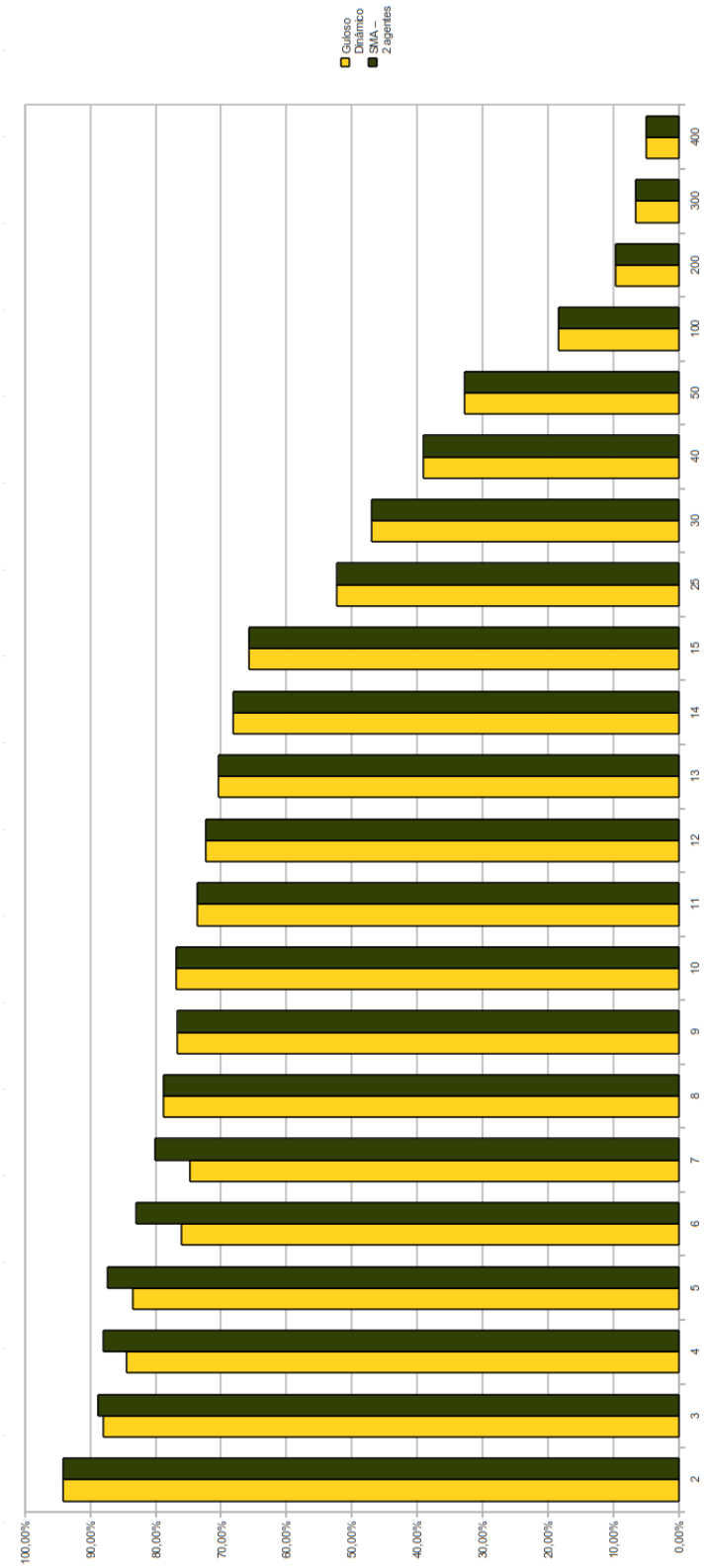


Figura 5.16: *Comparação entre a qualidade os resultados do SMA com 2 agentes e o algoritmo guloso dinâmico.*

A Figura 5.16 apresenta que no intervalo de 3 a 7 recursos a qualidade das distribuições do SMA com 2 agentes foram superiores ao do algoritmo guloso dinâmico. Em todas as outras simulações, a qualidade do SMA com 2 agentes foi a mesma do algoritmo guloso dinâmico.

Durante uma simulação, os agentes trocam mensagens em busca da solução do problema.

Pode-se utilizar o agente *Sniffer* da plataforma JADE para monitorar a troca de mensagens entre o distribuidor guloso e seus assistentes e o agente *DummyAgent* para enviar uma solicitação de cálculo ao agente Solicitador.

A Figura 5.17 apresenta o diagrama de sequência criado pela interface gráfica do agente *Sniffer*.

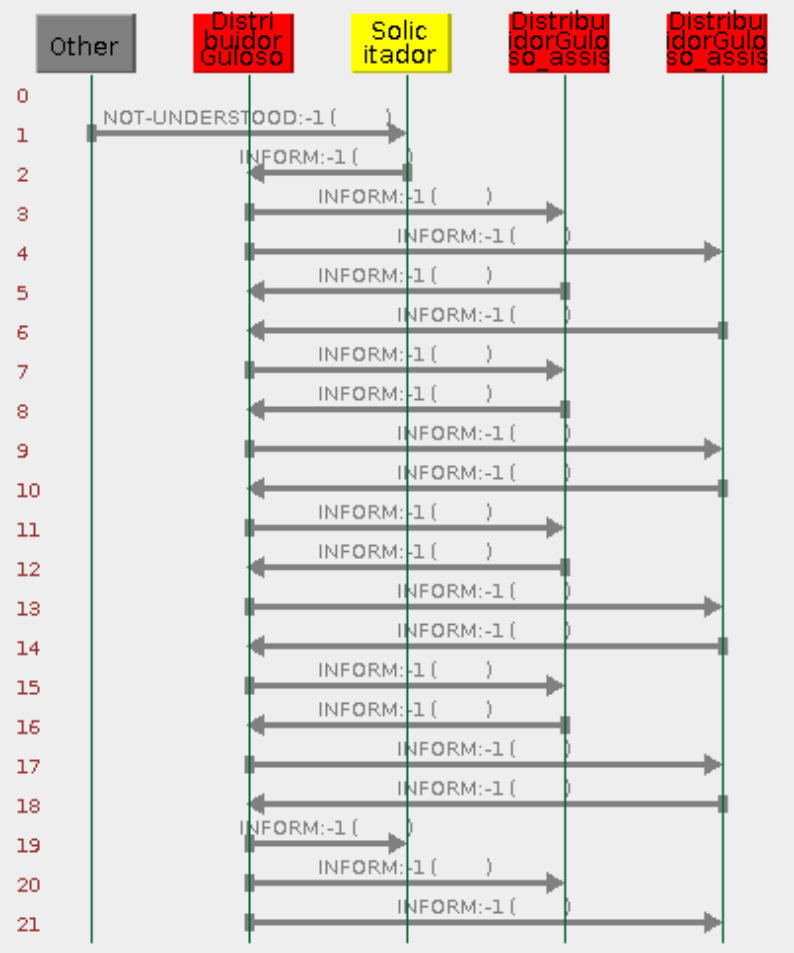


Figura 5.17: *Diagrama de sequência da comunicação dos agentes.*

No passo 1, da Figura 5.17, uma mensagem é enviada ao agente solicitador através do agente *DummyAgent*. Em seguida, no passo 2, o agente Solicitador utiliza a ontologia criada para enviar uma solicitação ao agente distribuidor guloso.

Na Figura 5.17, foi realizada uma simulação com uma matriz quadrada de 4 linhas com 4 recursos e 2 agentes ($n = m = 4$, $k = 4$ e $x = 2$). Por isto, nos passos 3 e

4 a matriz M é dividida em duas partes onde os assistentes 1 e 2 vão procurar a melhor posição. Em seguida, os agentes assistentes informam, nos passos 5 e 6, as melhores posições para o agente distribuidor guloso.

Esse mesmo procedimento se repete k vezes até que os k recursos sejam distribuídos. Os passos 3 a 6, 7 a 10, 11 a 14 e 15 a 18 representam esse laço de repetição.

Nota-se que a cada solicitação para o assistente, a matriz M é modificada pela distribuição de um recurso.

Finalmente, no passo 19 o agente Solicitador é informado das k melhores posições para os recursos. E, nos passos 20 e 21 acontece as solicitações de término dos assistentes.

Nesse capítulo foi criado um SMA que melhora o algoritmo criado na seção 4.5, do Capítulo 4. O capítulo a seguir apresenta uma análise das simulações feitas no Capítulo 4 e neste capítulo.

CAPÍTULO 6

Análise dos Resultados

Esse capítulo apresenta uma análise das soluções apresentadas no Capítulo 4 e 5. De fato, os algoritmos apresentados conseguem resolver o problema ou sugerir boas distribuições, segundo os critérios pré-estabelecidos. Este capítulo apresenta os limites de aplicação destes algoritmos, com base nas simulações realizadas no capítulo 4 e 5.

6.1 Tempo nas Simulações

Esta seção apresenta uma comparação entre os tempos gastos nas simulações de todos os algoritmos apresentados no Capítulo 4. A Figura 6.1 apresenta a comparação destes tempos gastos nas simulações.

O eixo x do gráfico apresentado na Figura 6.1 representa as dimensões n e m da matriz M e a quantidade de recurso k. Nota-se que, estas três variáveis são iguais em todas as simulações, por isto o eixo x possui apenas um valor. O eixo y do gráfico representa o tempo gasto em milissegundos.

Pelo gráfico apresentado na Figura 6.1, pode-se notar que a Heurística da Diagonal (HD) e sua variação a HD Precipitada gastaram tempos muito pequenos em relação aos demais algoritmos, pois possuem baixos valores no eixo y.

Nota-se também que, a implementação do algoritmo que testa todas as combinações tornou-se viável a partir do cálculo da matriz quadrada de 8 linhas com 8 recursos.

Continuando a análise da Figura 6.1, pode-se notar que a simulação do algoritmo genético demorou mais que o algoritmo guloso no intervalo de 15 recursos até o intervalo de 40 recursos. A partir de 50 recursos, a implementação do algoritmo genético foi mais rápida que a implementação do algoritmo guloso. No entanto, a partir 100 recursos o tempo de execução do algoritmo genético aumenta em grande escala.

Como esperado, a implementação do algoritmo guloso dinâmico teve um desempenho melhor do que a implementação do algoritmo guloso. Essa implementação, também foi mais rápido do que a implementação do algoritmo genético.

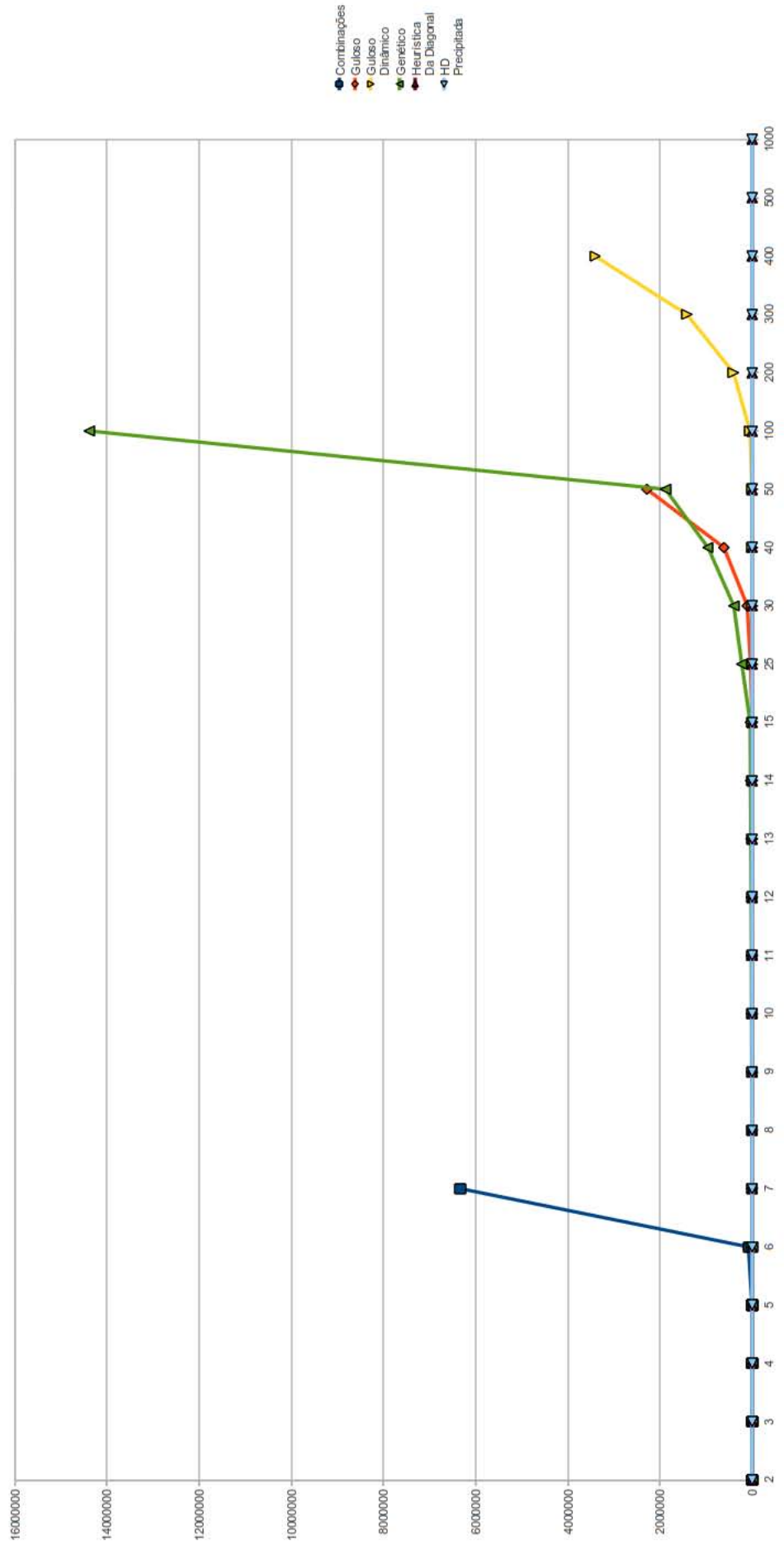


Figura 6.1: *Comparação entre tempos gastos nas simulações.*

A Figura 6.2 apresenta uma comparação entre os tempos gastos nas simulações dos algoritmos Heurística da Diagonal e a HD Precipitada, pois não é possível fazer comparações apenas com gráfico apresentado na Figura 6.1.

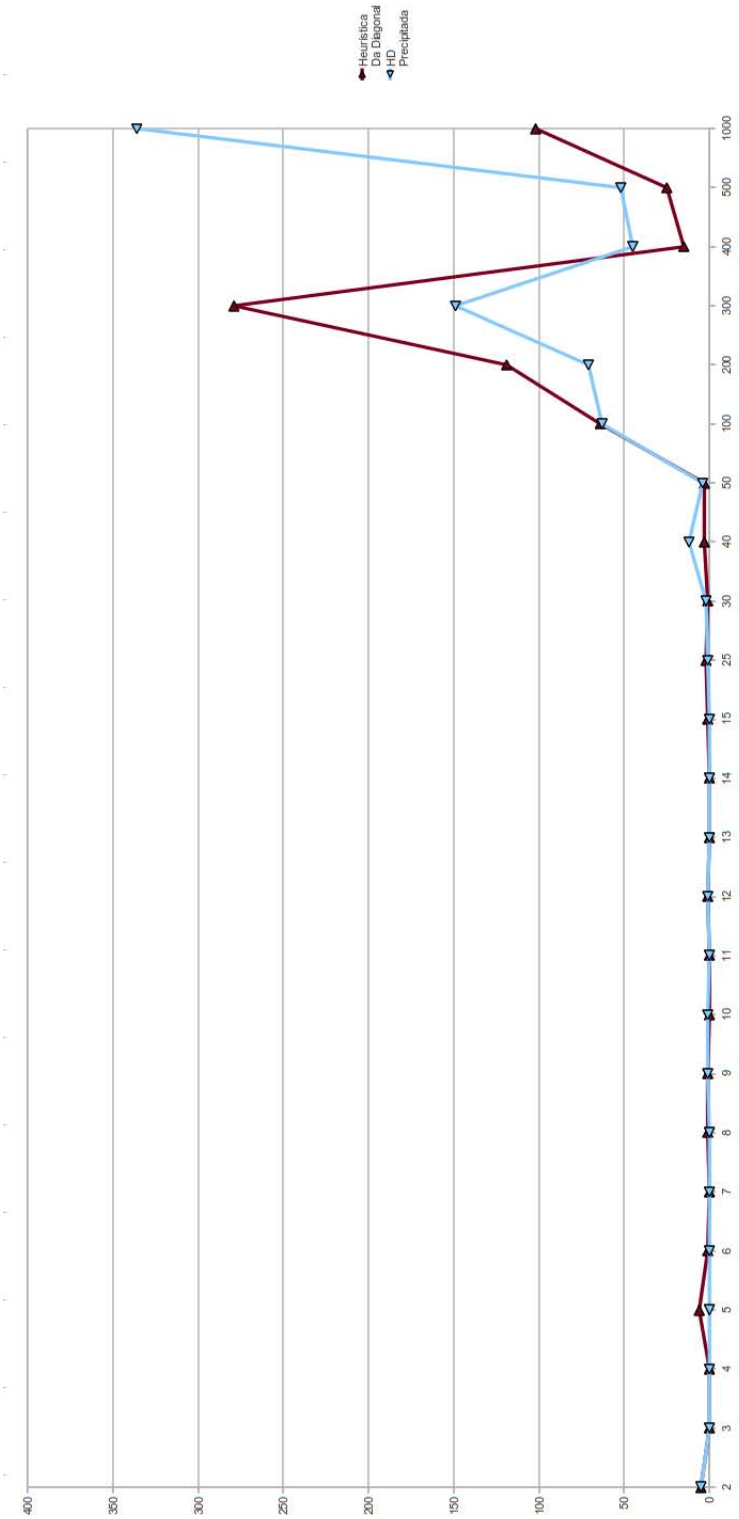


Figura 6.2: *Comparação entre os tempos dos algoritmos Heurística da Diagonal e HD Precipitada.*

No gráfico apresentado na Figura 6.2, observa-se que, com até cinquenta recursos, os tempos dos dois algoritmos são menores que 50 milissegundos. No intervalo de 100 a 300 recursos, a implementação do algoritmo HD Precipitada tem um desempenho maior que a implementação da Heurística da Diagonal. No entanto, no intervalo de 400 a 1.000 recursos a implementação da Heurística da Diagonal é melhor que a implementação da HD Precipitada.

A pequena diferença de desempenho entre as duas implementações apresentadas na Figura 6.2 é esperada, pois o algoritmo HD Precipitada é uma variação do algoritmo Heurística da Diagonal.

A HD Precipitada tem desempenho inferior do que a Heurística da Diagonal para instâncias maiores de 1.000 recursos do problema. A Figura 6.3 apresenta esta comparação entre estes algoritmos.

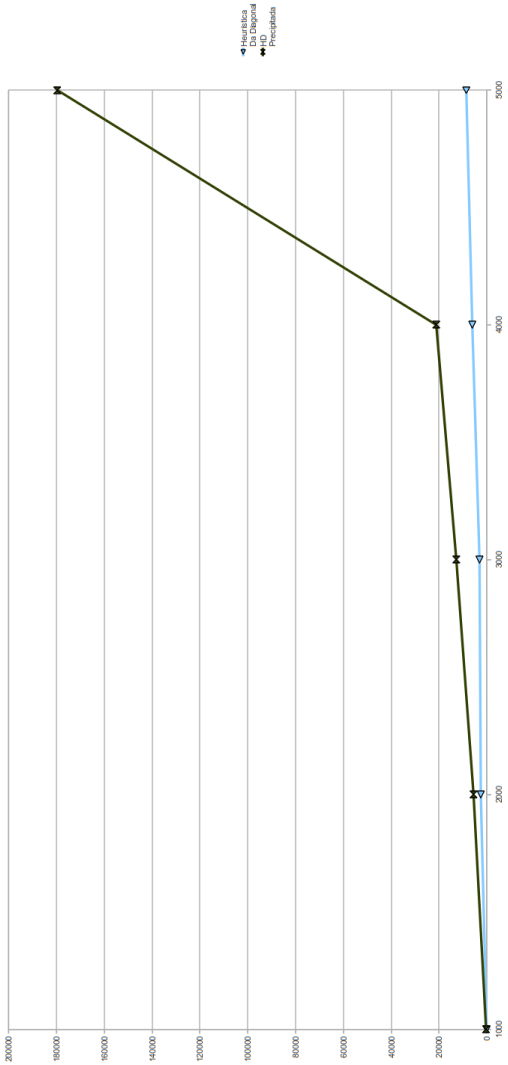


Figura 6.3: *Comparação entre os tempos dos algoritmos Heurística da Diagonal e HD Precipitada para instâncias com mais de 1.000 recursos.*

6.2 Qualidade dos Resultados dos Algoritmos

Analisar apenas os tempos das simulações não é suficiente para a escolha do melhor algoritmo para determinados domínios. Por isto, esta seção apresenta uma análise da qualidade dos resultados de cada algoritmo.

A Figura 6.4 apresenta a qualidade dos resultados de cada algoritmo apresentado no Capítulo 4.

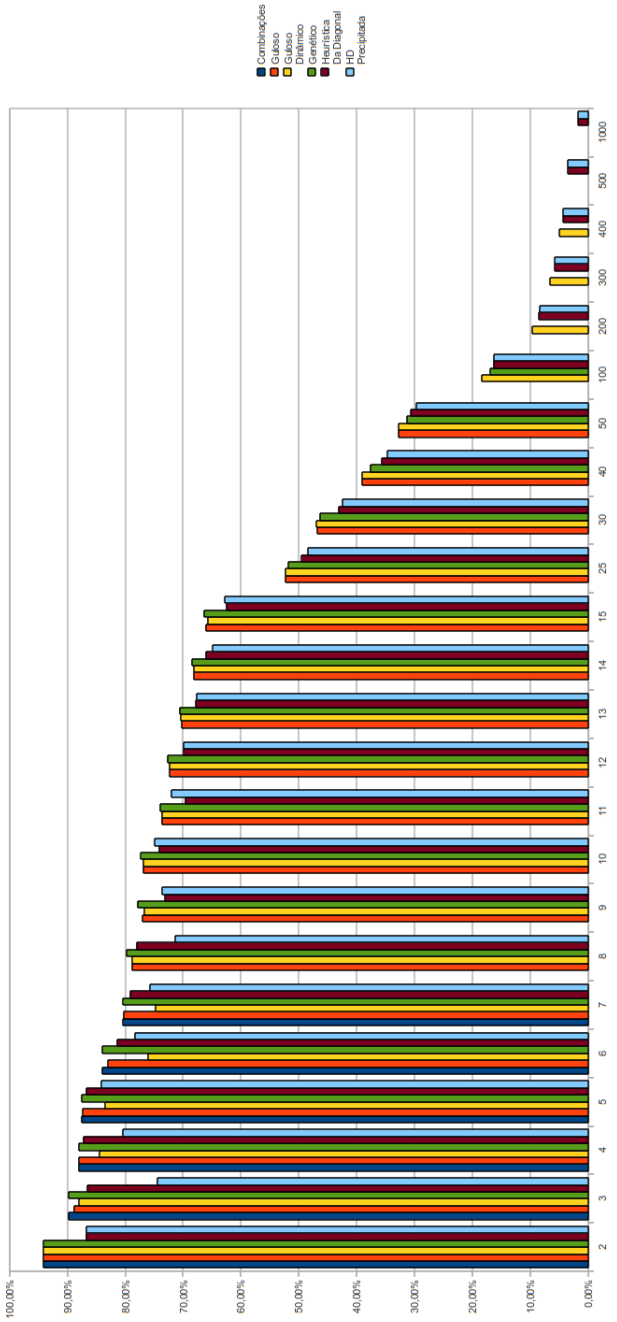


Figura 6.4: *Comparação entre as qualidades dos resultados dos algoritmos.*

O gráfico apresentado na Figura 6.4 mostra claramente que os algoritmos Heurística Diagonal e HD Precipitada obtiveram, na maioria das simulações, as piores avaliações. O algoritmo Heurística Diagonal, na maioria das simulações, obteve melhores resultados que o algoritmo HD Precipitada.

Nota-se que, na Figura 6.4, nenhum algoritmo atingirá 100% de qualidade, pois para isto seria necessário preencher cada subárea da matriz M com pelo menos um recurso.

Na Figura 6.4, o algoritmo Guloso apresenta, na grande maioria dos casos, resultados melhores que o algoritmo Guloso Dinâmico. Apenas com 12, 13 e 30 recursos, o algoritmo Guloso Dinâmico teve uma qualidade maior que o algoritmo Guloso.

Entre todos os algoritmos apresentados na Figura 6.4, o algoritmo genético obteve a maior qualidade nos resultados em todas as simulações no intervalo de 2 a 15 recursos. No intervalo de 2 a 7 recursos, pode-se afirmar que obteve as melhores avaliações possíveis, ou seja, o algoritmo conseguiu chegar no resultado ótimo. Isto pode ser afirmado porque o algoritmo genético tem a mesma qualidade de distribuição que o algoritmo que testa todas combinações no intervalo de 2 a 7 recursos. No entanto, o algoritmo genético começa a perder sua qualidade a partir de 25 recursos. Provavelmente, esse comportamento acontece porque a quantidade de gerações e a quantidade de indivíduos na população não é adequada para este intervalo.

Os resultados apresentados na Figura 6.4 mostra que, o algoritmo genético bem configurado retorna as melhores escolhas na grande maioria das simulações. Por isto, estudos mais detalhados devem ser realizados com esse algoritmo.

6.3 Dimensão da Área de Atendimento

Se for considerado que uma célula da matriz M possui 1 km^2, então para descobrir o tamanho da área que um dos algoritmos apresentados nos capítulos 4 e 5 pode trabalhar, basta multiplicar a quantidade de linhas e colunas da matriz M utilizada.

A Tabela 6.1 apresenta as dimensões máximas simuladas nos Capítulos 4 e 5.

Tabela 6.1: *Dimensões máximas simuladas.*

Algoritmo	**Área Máxima** (km^2)
Heurística da Diagonal	25.000.000
HD Precipitada	25.000.000
Guloso Dinâmico	160.000
Genético	10.000
Guloso	2.500
Combinações	49

Para se ter uma noção de dimensão real, o Brasil possui $8.514.876,599\ km^2$, o estado de Goiás possui $340.086,698\ km^2$, a Região Metropolitana de Goiânia possui $5.787\ km^2$ e Goiânia possui $739,492\ km^2$.

Desta forma, a Figura 6.5 apresenta os algoritmos que podem ser aplicados em zonas tão grandes quanto a brasileira.

Figura 6.5: *Algoritmos que podem ser aplicados em áreas tão grandes quanto o Brasil.*

A Figura 6.6 apresenta os algoritmos que podem ser aplicados em zonas tão grandes quanto a do estado de Goiás.

Figura 6.6: *Algoritmos que podem ser aplicados em áreas tão grandes quanto o Goiás.*

O algoritmo guloso dinâmico, apresentado na seção 4.5, também pode resolver instâncias com área de atendimento tão grandes quanto a do estado de Goiás, se e somente se, estiver trabalhando com vários computadores, como apresentado no SMA especificado no Capítulo 5, por isto, ele foi relatado na Figura 6.6. Estima-se que, sua execução com três agentes e processadores seja suficiente para áreas de atendimento com esta dimensão.

A Figura 6.7 apresenta os algoritmos que podem ser aplicados em zonas tão grandes quanto a Região Metropolitana de Goiânia.

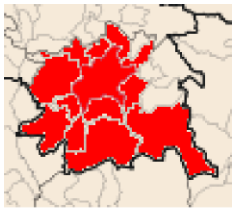


Figura 6.7: *Algoritmos que podem ser aplicados em áreas tão grandes quanto a Região Metropolitana de Goiânia.*

A Figura 6.8 apresenta os algoritmos que podem ser aplicados em zonas tão grandes quanto o município de Goiânia.

Figura 6.8: *Algoritmos que podem ser aplicados em áreas tão grandes quanto a Goiânia.*

Nota-se que, o algoritmo que analisa exaustivamente todas as combinações, de resultados possíveis, não foi citado em nenhum dos casos porque restringe-se a áreas menores que 49 km^2.

Neste capítulo os algoritmos criados, no Capítulo 4, e o SMA, criado no Capítulo 5, foram comparados através de simulações, feitas nos respectivos capítulos, quanto ao tempo de execução e a qualidade dos resultados. O capítulo a seguir apresenta as contribuições, trabalhos futuros e a conclusão deste trabalho.

CAPÍTULO 7

Considerações Finais

7.1 Contribuições do Trabalho

Este trabalho apresentou um estudo de trabalhos relacionados ao problema apresentado no Capítulo 2. Na grande maioria desses trabalhos, foi apresentado um problema que consiste na manutenção de um determinado serviço, seja ele, segurança, saúde ou controle de incêndio. Todos esses serviços têm o mesmo problema da distribuição inicial dos recursos. Na grande maioria dos trabalhos relacionados, utilizam-se Sistemas Multi-agentes para resolução, assim como foi apresentado no Capítulo 5.

No Capítulo 3 foi apresentado:

- Uma forma de representação do problema;
- Formas de preenchimento da matriz que representa a área de atendimento;
- Uma forma de avaliação de uma distribuição.

A função de avaliação apresentada, no Capítulo 3, foi:

$$fa = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} P_{(i,j)} \cdot PRO_{(i,j)} \tag{7-1}$$

A função de proteção apresentada, no Capítulo 3, foi:

$$PRO_{(i,j)} = 1 - \prod_{l=1}^{l=K} \Big(1 - h(d(R_l, i, j))\Big) \tag{7-2}$$

Foram desenvolvidos vários algoritmos para resolução do problema, utilizando a representação por matriz, no Capítulo 4. Dentre os algoritmos criados pode-se destacar o Algoritmo Genético (seção 4.6), a Heurística da Diagonal (seção 4.7) e o Algoritmo Guloso, que utiliza programação Dinâmica (seção 4.5).

Foi criada uma ferramenta em JavaFX para definição de áreas de atendimento representadas por matrizes. Essa ferramenta também permite a aplicação de todos os algoritmos criados no Capítulo 4.

No Capítulo 6, foi apresentada comparação dos resultados, de dezenas de simulações, feitas com os algoritmos criados no Capítulo 4.

Também foi criada uma ferramenta de construção de esqueletos de vias, apresentada na seção 4.2.

7.2 Relatórios e Artigos

Ao longo das pesquisas, foram publicados os relatórios técnicos:

- Linguagem Nice;
- Design by Contract;
- Integração de Java e Prolog com JPL.

Além destes relatórios técnicos, foram publicados os seguintes artigos:

- Um Sistema de Apoio à Decisão baseado em Agentes para a Companhia Elétrica do Estado de Goiás, no Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC), realizado em Rio Grande no Rio Grande do Sul;
- Um Sistema de Apoio à Decisão Baseado em Agentes para Tratamento de Ocorrências no Setor Elétrico, na Revista de Informática Teórica e Aplicada.

7.3 Trabalhos Futuros

Esta seção apresenta pesquisas que podem ser realizadas sobre o problema de distribuição de recursos em áreas geográficas:

- Definir matematicamente o problema da distribuição de recursos utilizando um grafo para representar a área de atendimento;
- Explorar algoritmos com base nesses grafos e compará-los com os algoritmos apresentados no Capítulo 4;
- Melhorar a ferramenta de representação dos esqueletos das vias apresentada na seção 4.2;
- Explorar o raio de influência do algoritmo guloso dinâmico apresentado na seção 4.5 que permite distribuí-lo numa grade computacional diminuindo o tempo de execução;
- Explorar ainda mais os algoritmos genéticos aplicados ao problema da distribuição;
- Propor uma Rede Bayesiana para representação da probabilidade de ocorrer um problema numa subárea para o domínio de companhias elétricas;

- Criar um simulador com base numa sociedade multiagente que permita verificar a real importância das distribuições para as áreas onde ocorrem o problema da distribuição;
- Tentar provar em qual classe P, NP, NP-completo ou outra em que o problema se classifica;

Finalmente, acredita-se que problemas dessa natureza podem ser expressos pela equação:

$$max\ z = \sum_{i=1}^{i=n} \left[w_i \cdot \left(1 - \prod_{j=1}^{j=n} \left(1 - h\Big(d(a_i, a_j)\Big) \cdot a_j \right) \right) \right] \tag{7-3}$$

$$s.a.: \sum_{i=1}^{i=n} a_i = k \tag{7-4}$$

Para todo a_i de 1 a n está sujeito a restrição $0 \leqslant a_i \leqslant 1$ tal que $a_i \in \mathbb{N}$. Desta forma, cabe a pergunta: "Seria possível converter a expressão 7-3 no formato padrão de programação linear?". Se esta resposta for sim, então pode-se utilizar o algoritmo Simplex para solucionar o problema.

A função z, apresentada na expressão 7-3, quando maximizada, retorna a maior pontuação da função de avaliação. Caso possa ser transformada em linear, é possível obter quais variáveis devem possuir valor 1, utilizando o algoritmo Simplex, para tornar a função z máxima.

Por exemplo, a Figura 7.1 apresenta um cenário do problema da distribuição, onde pretende-se distribuir 2 recursos, ou seja, $k = 2$.

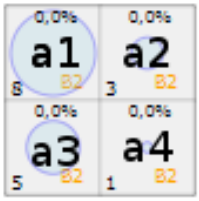


Figura 7.1: *Exemplo da expressão 7-3 no problema da distribuição.*

Na Figura 7.1, cada subárea é numerada de a_1 a a_4. Dessa forma, pode-se montar uma expressão com a expressão 7-3, utilizando a função $h()$, apresentada na expressão 3-8 do Capítulo 3:

$$\begin{aligned} max\ z = \quad & 8 \cdot \Big(1 - (1 - 1 \cdot a_1) \cdot (1 - 0.5 \cdot a_2) \cdot (1 - 0.5 \cdot a_3) \cdot (1 - 0.5 \cdot a_4)\Big) \\ + \quad & 3 \cdot \Big(1 - (1 - 1 \cdot a_2) \cdot (1 - 0.5 \cdot a_1) \cdot (1 - 0.5 \cdot a_3) \cdot (1 - 0.5 \cdot a_4)\Big) \end{aligned}$$

$$
\begin{aligned}
&+ \quad 5 \cdot \Big(1-(1-1\cdot a_3)\cdot(1-0.5\cdot a_1)\cdot(1-0.5\cdot a_2)\cdot(1-0.5\cdot a_4)\Big) \\
&+ \quad 1 \cdot \Big(1-(1-1\cdot a_4)\cdot(1-0.5\cdot a_1)\cdot(1-0.5\cdot a_2)\cdot(1-0.5\cdot a_3)\Big) \\
s.a: & \\
& \quad a_1+a_2+a_3+a_4 = 2 \\
& \quad 0 \leqslant a_1 \leqslant 1; 0 \leqslant a_2 \leqslant 1; 0 \leqslant a_3 \leqslant 1 \; e \; 0 \leqslant a_4 \leqslant 1 \\
& \quad a_1 \in \mathbb{N}; a_2 \in \mathbb{N}; a_3 \in \mathbb{N} \; e \; a_4 \in \mathbb{N}
\end{aligned}
$$

Se forem efetuados os cálculos com 2 recursos:

$$
\begin{aligned}
&a_1 = 0, a_2 = 0, a_3 = 1 \; e \; a_4 = 1. \; Tem-se: \; 14,25 \\
&a_1 = 0, a_2 = 1, a_3 = 0 \; e \; a_4 = 1. \; Tem-se: \; 13,75 \\
&a_1 = 0, a_2 = 1, a_3 = 1 \; e \; a_4 = 0. \; Tem-se: \; 14,75 \\
&a_1 = 1, a_2 = 0, a_3 = 0 \; e \; a_4 = 1. \; Tem-se: \; 15 \\
&a_1 = 1, a_2 = 0, a_3 = 1 \; e \; a_4 = 0. \; Tem-se: \; 16 \\
&a_1 = 1, a_2 = 1, a_3 = 0 \; e \; a_4 = 0. \; Tem-se: \; 15,5
\end{aligned}
$$

Como pode ser visto, o valor máximo 16 é obtido quando a_1 e a_3 possuem um recurso e as demais subáreas nenhum. A Tabela A.2 apresenta o valor máximo para 2 recursos na área de atendimento da Figura 7.1. Cada resultado dessas combinações, nesse caso 14,25, 13,75, 14,75, 15, 16 e 15,5, é a nota da função de avaliação.

Utilizando o algoritmo de combinações obteve-se o resultado da Figura 7.2.

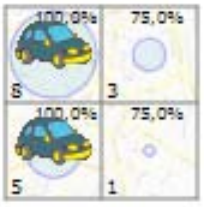


Figura 7.2: *Resultado utilizando o algoritmo de combinações com 2 linhas, 2 colunas e 2 recursos.*

Comparando o melhor resultado possível apresentado na Figura 7.2 e as subáreas a_1 e a_3 que devem receber recursos na Figura 7.1, pode-se concluir que, a função de maximizar z apresentada retorna a melhor opção para a instância do problema.

No entanto, para que esse problema seja resolvido utilizando o algoritmo Simplex é necessário transformar a expressão 7-3 numa função linear.

Neste capítulo foram apresentadas as contribuições deste trabalho e possíveis trabalhos futuros. O capítulo a seguir serão apresentadas as conclusões do trabalho.

7.4 Conclusão

Mesmo que o problema seja aparentemente simples, ele se mostrou difícil de ser definido, por meios matemáticos, e, ainda mais, de ser solucionado.

Diante dos algoritmos criados, nos capítulos 4 e 5, pode-se dizer que: o problema foi solucionado parcialmente porque não foi encontrado um algoritmo que o resolva de forma ótima.

As soluções apresentadas foram heurísticas que conseguem obter resultados em tempo polinomial. Adotou-se que, um algoritmo não consegue resolver instâncias a partir do momento em que, o tempo de execução fosse maior do que uma dia e meio.

Os resultados produzidos neste trabalho podem ser utilizados nos trabalhos relacionados, apresentados no Capítulo 2, como uma forma de distribuição inicial. Tendo em vista que, na maioria dos trabalhos apresentados foram definidas estratégias para atender as solicitações, dos recursos, criadas durante um período de atendimento.

Acredita-se que, os algoritmos apresentados podem ser futuramente melhorados e serviram de base para criação de novos algoritmos que resolvem o problema de forma ótima.

Referências Bibliográficas

[1] AND/OR ITS AFFILIATES, O. C. **Better experiences for end-users and developers with javafx 1.3.1**. http://javafx.com/, Outubro 2010.

[2] BROTCORNE, L.; LAPORTE, G.; SEMET, F. **Ambulance location and relocation models**. *European Journal of Operational Research*, 147(3):451 – 463, 2003.

[3] CHURCH, R.; REVELLE, C. **The maximal covering location problem**. *Papers in Regional Science*, 32:101–118, 1974. 10.1007/BF01942293.

[4] CROCKFORD, D. **Introducing json**. http://www.json.org/, Outubro 2010.

[5] DEB, K. **Nonlinear goal programming using multi-objective genetic algorithms**. *The Journal of the Operational Research Society*, 52(3):12, 2001.

[6] FREDERICK PHILLIPS BROOKS, J. **The mythical man-month: essays on software engineering**. 1975.

[7] FURTADO, V. **Modelagem e simulação multiagente da criminalidade**. *funcapciencia.funcap.ce.gov.br*, p. 13, 2008.

[8] GOOGLE. **Google static maps api**. http://code.google.com/intl/pt-BR/apis/maps/documentation/staticmaps/, Outubro 2010.

[9] GREENWOOD, F. L. B. G. C. D. **Developing Multi-Agent Systems with JADE**. John Wiley & Sons Ltd, 2007.

[10] JSON.ORG. **Introducing json**. http://www.json.org/, nov 2010.

[11] MACHADO, A.; RAMALHO, G.; ZUCKER, J.-D.; DROGOUL, A. **Multi-agent patrolling: An empirical analysis of alternative architectures**. In: Simão Sichman, J.; Bousquet, F.; Davidsson, P., editors, *Multi-Agent-Based Simulation II*, volume 2581 de **Lecture Notes in Computer Science**, p. 81–97. Springer Berlin / Heidelberg, 2003. 10.1007/3-540-36483-8_11.

[12] MELO, A.; BELCHIOR, M.; FURTADO, V. **Analyzing police patrol routes with the simulation of the physical reorganization of agents**. In: *In Proc. of the 6th Int. Workshop on Multi-Agent Based Simulation (MABS'05*. Springer-Verlag, 2005.

[13] NORVIG, S. R. P. **Inteligência Artificial**. Elsevier, 2004.

[14] OF CHICAGO, U. **Repast - recursive porus agent simulation toolkit**. http://repast.sourceforge.net/repast_3/, Outubro 2010.

[15] ROTSTAN, K. M. N. **Jgap: Java genetic algorithms package**. http://jgap.sourceforge.net, Outubro 2010.

[16] SCHURR, N.; MARECKI, J.; LEWIS, J.; TAMBE, M.; SCERRI, P. **The defacto system: Coordinating human-agent teams for the future of disaster response**. In: Weiss, G.; Bordini, R.; Dastani, M.; Dix, J.; Fallah Seghrouchni, A., editors, *Multi-Agent Programming*, volume 15 de **Multiagent Systems, Artificial Societies, And Simulated Organizations**, p. 197–215. Springer US, 2005. 10.1007/0-387-26350-0_8.

[17] STEIN, T. H. C. C. E. L. R. L. R. C. **Algoritmos Teoria e Prática**. Elsevier, 2002.

[18] SUN, H.; THANGARAJAH, J.; PADGHAM, L. **Eclipse-based prometheus design tool**. In: *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, p. 1769–1770, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[19] TADOKORO, H. K. S. **Robocup rescue: A grand challenge for multiagent and intelligent systems**. *AI Magazine*, 22:14, 2001.

[20] VASCONCELOS, V. F. E. **Geosimulation in education: A system for teaching police resource allocation**. *International Journal of Artificial Intelligence in Education*, p. 25, 2007.

[21] WINIKOFF, L. P. M. **Developing Intelligent Agent Systems: A Practical Guide**. John Wiley & Sons Ltd, 2004.

[22] WINIKOFF, M.; PADGHAM, L. **The prometheus methodology**. In: Weiss, G.; Bergenti, F.; Gleizes, M.-P.; Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems*, volume 11 de **Multiagent Systems, Artificial Societies, And Simulated Organizations**, p. 217–234. Springer US, 2004. 10.1007/1-4020-8058-1_14.

Apêndice A

A.1 Algoritmos Auxiliares

Esta seção apresenta os algoritmos das funções que auxiliam na resolução do problema da distribuição de recursos. Os algoritmos que serão apresentados a seguir, calculam:

- a distância entre duas células;
- a proteção numa determinada célula;
- a proteção de uma matriz;
- a avaliação de uma distribuição.

O primeiro algoritmo calcula a distância entre duas células numa matriz. Esta função foi apresentada na seção 3.3.1, na expressão 3-9. O Algoritmo A.1 define a função de cálculo da distância entre duas células numa matriz.

Algoritmo A.1: $DistanciaMatriz(x1, y1, x2, y2)$

Entrada: $x1$ e $y1$ são as coordenadas da primeira célula.
$x2$ e $y2$ são as coordenadas da segunda célula.
Saída: Distância entre as células $(x1, y1)$ e $(x2, y2)$ de uma matriz.

1 $dx \leftarrow |x1 - x2|$.
2 $dy \leftarrow |y1 - y2|$.
3 **retorna** $max(dx, dy)$.

Como apresentado no Algoritmo A.1, a distância entre duas células é a maior diferença vertical ou horizontal entre elas. Também poderia ser utilizada a distância euclidiana $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$.

O Algoritmo A.1 tem complexidade [13, página 942] [17, página 32] igual a 1, pois não executa nenhum laço para determinar a distância entre as duas células.

O segundo algoritmo define a função de influência do recurso na área de atendimento. Esta função foi apresentada na seção 3.3.1, na expressão 3-8. O Algoritmo A.2 define a função de influência.

Algoritmo A.2: $ProtecaoLocal(distancia)$

Entrada: $distancia$ distância entre as duas células.

Saída: Influência do recurso numa determinada distância.

1 **retorna** $1/2^{distancia}$.

Como o Algoritmo A.1, o Algoritmo A.2 também tem complexidade igual a 1.

O terceiro algoritmo calcula a proteção numa área de atendimento. Esta função foi apresentada na seção 3.3.1, na expressão 3-11. O Algoritmo A.3 define a função de proteção de uma área de atendimento.

Algoritmo A.3: $ProtecaoArea(M,S)$

Entrada: M matriz de n linhas e m colunas.

S conjunto de pontos dos recursos.

Saída: Matriz PRO de n linhas e m colunas com os valores das proteções.

```
1  Declara uma matriz PRO com n linhas e m colunas.
2  para i ← 1 até n faça
3      para j ← 1 até m faça
4          perigo ← 1.
5          para cada p em S faça
6              d ← DistanciaMatriz(p.i, p.j, i, j).
7              se d = 0 então
8                  perigo ← 0.
9                  Interromper laço.
10             senão
11                 protecao ← ProtecaoLocal(d).
12                 perigo ← perigo * (1 − protecao).
13             fim
14         fim
15         PRO[i, j] ← (1 − perigo).
16         j ← j + 1.
17     fim
18     i ← i + 1.
19 fim
20 retorna PRO.
```

O Algoritmo A.3 tem complexidade $O(ProtecaoArea) = n \cdot m \cdot k$, onde k é a quantidade de recursos em S. Ou seja, serão feitas $n \cdot m \cdot k$ repetições neste algoritmo, em função das entradas M e S.

Finalmente, com o algoritmo que calcula as proteções da área de atendimento, pode-se criar o algoritmo para a função de avaliação das distribuições apresentada na seção 3.3.2, na expressão 3-12. O Algoritmo A.4 apresenta a função de avaliação.

Algoritmo A.4: $fa(M,S)$

Entrada: M matriz de n linhas e m colunas.
S conjunto de pontos dos recursos.
Saída: Avaliação da distribuição.

```
1  protecoes ← ProtecaoArea(M,S).
2  pontuacao ← 0.
3  para i ← 1 até n faça
4      para j ← 1 até m faça
5          pontuacao ← pontuacao + M[i,j] * protecoes[i,j].
6          j ← j + 1.
7      fim
8      i ← i + 1.
9  fim
10 retorna pontuacao.
```

O Algoritmo A.4 tem complexidade igual a do Algoritmo A.3 somada a complexidade das linhas 2 a 8. Por isto, $O(fa) = n \cdot m \cdot k + n \cdot m = n \cdot m \cdot (k+1)$.

A.1.1 Teste de Todas Combinações

Com a representação por matriz, apresentada na seção 4.1, juntamente com a função de avaliação apresentada na seção 3.3.2, pode-se criar um algoritmo que efetua todas as combinações buscando a melhor combinação possível. No entanto, sabe-se que este algoritmo tem complexidade combinatória, ou seja, para o término do algoritmo são necessários no mínimo $C_p^n = \frac{n!}{(n-p)!\cdot p!}$ passos.

Na prática, este algoritmo é inviável, pois para obter boas soluções através da representação por matriz é necessário dividir a área de atendimento em muitas células. Por isto, o número n da expressão C_p^n se torna muito grande em relação ao número p. Isto faz com que a quantidade de passos C_p^n, seja um número grande, na casa de bilhões ou trilhões nos casos em que $n > 50$ e $8 < p < 42$.

Por exemplo, uma matriz M com 7 linhas e 7 colunas, tem-se 49 células, logo $n = 49$. Se devem ser distribuídos 10 recursos na área de atendimento, serão necessários $C_{10}^{49} = \frac{49!}{(49-10)!\cdot 10!}$ passos, isto equivale a 8.217.822.536 passos, ou seja, mais de 8 bilhões de passos para encontrar a melhor solução.

Mesmo que a função de avaliação das combinações seja linear, o tempo exigido para execução do exemplo seria enorme. A Tabela A.1 apresenta alguns exemplos de combinações com dez recursos, a quantidade de passos necessários para encontrar a solução ótima e uma estimativa de tempo utilizando um processador de 2,66 GHz, ou seja, 2.660.000 instruções por segundo.

Tabela A.1: *Exemplos de combinações com dez recursos e as estimativas de tempo para o término.*

Matriz (n x m)	**Combinações**	**Tempo (seg)**	**Tempo**
(8 x 8)	$C_{10}^{64} = 151.473.214.816$	56.945	56 seg. 945 milisseg.
(9 x 9)	$C_{10}^{81} = 1.878.392.407.320$	706.163	11 min. 46 seg. 163 milisseg.
(10 x 10)	$C_{10}^{100} = 17.310.309.456.440$	6.507.635	1 hora 48 min. 27 seg. 635 milisseg.
(11 x 11)	$C_{10}^{121} = 126.524.771.308.936$	47.565.704	13 horas 12 min. 45 seg. 704 milisseg.
(12 x 12)	$C_{10}^{144} = 767.464.189.477.128$	288.520.372	3 dias 8 horas 8 min. 40 seg. 372 milisseg.
(13 x 13)	$C_{10}^{169} = 3.992.397.569.688.528$	1.500.901.342	17 dias 8 horas 55 min. 1 seg. 342 milisseg.
(14 x 14)	$C_{10}^{196} = 18.257.282.924.056.176$	6.863.640.197	79 dias 10 horas 34 min. 197 milisseg.

O Algoritmo A.5 apresenta a busca da solução ótima para o problema da distribuição, através da análise de todas as combinações $C_k^{n \cdot m}$ de k recursos numa matriz M com n linhas e m colunas. Desta forma, serão realizados $\frac{(n \cdot m)!}{(n \cdot m - k)! \cdot k!}$ passos para encontrar a solução ótima.

Algoritmo A.5: *Combinacoes*(M,k)

Entrada: M matriz de n linhas e m colunas.
k quantidade de recursos a serem distribuídos.
Saída: S conjunto de pontos (x,y).

```
1  S ← {}.
2  combinacao ← {}.
3  avaliacaoMelhorPonto ← 0.
4  para i ← 1 até k faça
5      combinacao[i] ← i.
6      i ← i + 1.
7  fim
8  repita
9      P ← {}.
10     para i ← 1 até k faça
11         ponto ← ConverterNumeroEmPonto(m, combinacao[i]).
12         P ← P ∪ ponto.
13         i ← i + 1.
14     fim
15     avaliacao ← fa(M, P).
16     se avaliacao > avaliacaoMelhorPonto então
17         S ← p.
18         avaliacaoMelhorPonto ← avaliacao.
19     fim
20     Gera próxima combinação.
21 até Terminar todas combinações
22 retorna S.
```

O Algoritmo A.5 utiliza pouca memória para evitar problemas de falta de espaço. A variável S, declarada na linha 1, é iniciada com um conjunto vazio. Esta variável é responsável por armazenar o conjunto de pontos da melhor solução encontrada. Na linha 2, a variável *combinacao* também é inicializada com um conjunto vazio. Esta variável mantém a combinação que será avaliada. Na linha 3, a variável *avaliacaoMelhorPonto* que mantém a nota da melhor avaliação é inicializada com zero.

Nas linhas 4 a 7, do Algoritmo A.5, a variável *combinacao* é iniciada com a primeira combinação possível, ou seja, com o valor $(1,2,3,...,k)$. Desta forma, as combinações são geradas num vetor de k posições utilizando os números das células como apresentado na Figura A.1.

	A	B	C	D	E	F	G	H	I
1	0	1	2	3	4	5	6	7	8
2	9	10	11	12	13	14	15	16	17
3	18	19	20	21	22	23	24	25	26
4	27	28	29	30	31	32	33	34	35
5	36	37	38	39	40	41	42	43	44
6	45	46	47	48	49	50	51	52	53
7	54	55	56	57	58	59	60	61	62

Figura A.1: *Exemplo da numeração sequencial da matriz.*

A Figura A.1 apresenta uma matriz de sete linhas e nove colunas. Desta forma, as combinações com três elementos seriam $(0,1,2)$, $(0,1,3)$, $(0,1,4)$ até chegar na combinação $(60,61,62)$.

Devido à esta forma de geração das combinações, é necessário fazer uma conversão de um número numa coordenada (i,j) da matriz. Esta conversão é utilizada na linha 11 do Algoritmo A.5 como a função *converterNumeroEmPonto()*. Nas linhas 10 a 14 do algoritmo, o conjunto de pontos da combinação corrente são avaliados e, nas linhas 15 a 19, é feita a comparação com o melhor conjunto de pontos já encontrado. Caso a combinação corrente seja melhor que a atual, então ela se torna a melhor encontrada.

O Algoritmo A.6 apresenta a função *converterNumeroEmPonto()* que converte um número numa posição (i,j) de uma matriz com m colunas. Nota-se que o índice da matriz deve começar com 0 (zero), como apresentado na Figura A.1.

Algoritmo A.6: *ConverterNumeroEmPonto*$(m, numero)$

Entrada: m quantidade de colunas da matriz.
numero índice da célula.
Saída: posição (i,j) em uma matriz de m colunas.

```
1 i ← 0.
2 j ← 0.
3 se numero < m então
4 |   j ← numero.
5 senão
6 |   i ← ⌊numero/m⌋.
7 |   j ← numero − i · m.
8 fim
9 retorna (i, j)
```

A linha 3 do algoritmo A.6 verifica se o índice excedeu a quantidade de colunas da matriz. Na Figura A.1, isto aconteceria ao chegar na coluna “I”. Nas linhas 5 a 8, é

feita a conversão dos números que excedem a quantidade de colunas. Na Figura A.1, isto equivale aos índices 9 a 62.

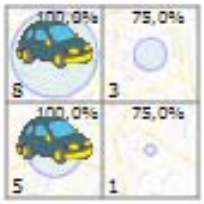
Com uma implementação feita em Java do Algoritmo A.5, foram feitas simulações com matrizes quadradas de duas a sete linhas. A Tabela A.2 apresenta o tempo gasto na execução, a quantidade de recursos utilizados e a nota da função de avaliação.

Tabela A.2: *Simulação com o algoritmo de combinações com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

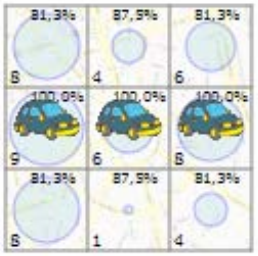
Dimensão (n x m) - recursos	**Tempo gasto**	**Nota máxima**	**Nota de *fa()***	**Cobertura**
(2 x 2) - 2	1 milisseg.	17	16	94,12%
(3 x 3) - 3	1 milisseg.	54	48,5	89,81%
(4 x 4) - 4	22 milisseg.	89	78,28	87,96%
(5 x 5) - 5	1 seg. 190 milisseg.	115	100,65	87,52%
(6 x 6) - 6	1 min. 20 seg. 711 milisseg.	161	135,17	83,96%
(7 x 7) - 7	1 hora 45 min. 34 seg. 876 milisseg.	240	192,85	80,35%

A Figura A.2 mostra as distribuições feitas na simulação apresentada na Tabela A.2. Os pesos das células foram sorteados no intervalo de 0 a 10.

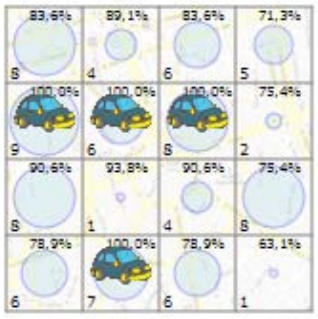
Cada exemplo de simulação que será apresentado utiliza números iguais para o número de linhas, o número de colunas e o número de recursos, ou seja, $n = m = k$. Por isto, o crescimento da área de atendimento é superior ao crescimento da quantidade de recursos.



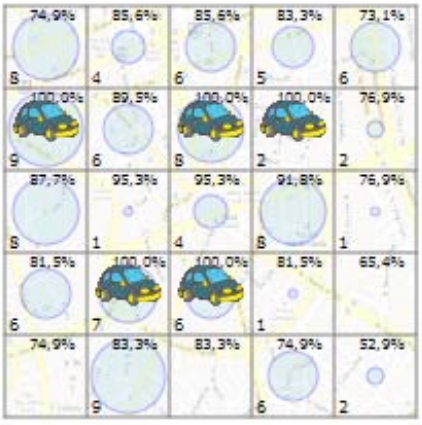
(a) *2 linhas, 2 colunas e 2 recursos.*



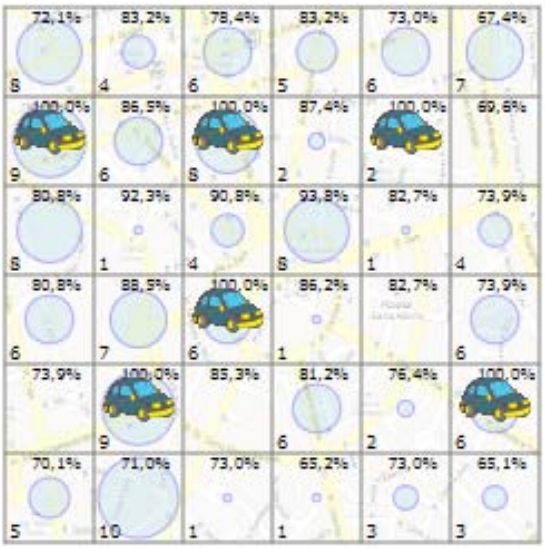
(b) *3 linhas, 3 colunas e 3 recursos.*



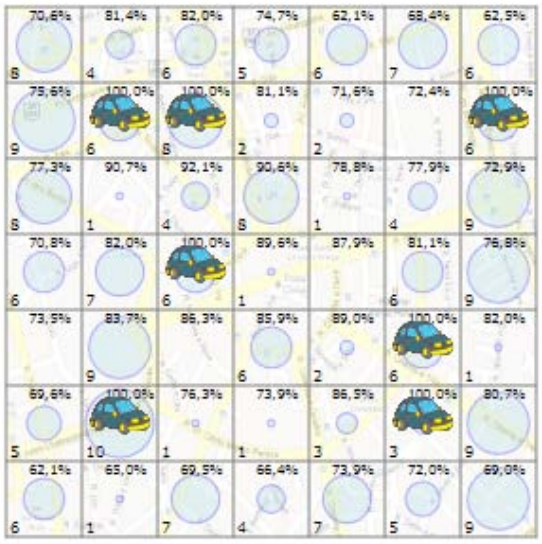
(c) *4 linhas, 4 colunas e 4 recursos.*



(d) *5 linhas, 5 colunas e 5 recursos.*



(e) *6 linhas, 6 colunas e 6 recursos.*



(f) *7 linhas, 7 colunas e 7 recursos.*

Figura A.2: *Resultados das simulações das matrizes quadradas de 2 a 7 linhas.*

Na Figura A.2, a parte superior de cada célula apresenta sua proteção e a parte inferior esquerda apresenta o peso da célula. Os círculos azuis desenhados em cada célula são proporcionais ao seu peso, sendo que o círculo do maior peso da matriz possui quase o mesmo tamanho da célula.

De fato, os tempos apresentados na Tabela A.2 são muito superiores do que os tempos apresentados na Tabela A.1. Isto acontece porque, na linha 20 do Algoritmo A.5, existe um custo c para geração da próxima combinação, nas linhas 10 a 14 acontece uma conversão de custo k e, na linha 15, existe o custo $n \cdot m \cdot (k+1)$ da função de avaliação *fa()* apresentada no Algoritmo A.4.

Por isto, a complexidade do Algoritmo A.5 é:

$$O(Combinacoes) = C_k^{n \cdot m} \cdot \Big(k + n \cdot m \cdot (k+1) + c\Big) \tag{A-1}$$

Onde c é o custo para gerar uma combinação. No caso da implementação feita em Java, c é igual a $2 \cdot k$ no pior caso, mais detalhes podem ser encontrados no Apêndice A na seção A.1.2, logo:

$$O(ImplCombinacoesJava) = C_k^{n \cdot m} \cdot \Big(k + 2k + n \cdot m \cdot (k+1)\Big) \tag{A-2}$$

$$= C_k^{n \cdot m} \cdot \Big(3k + n \cdot m \cdot (k+1)\Big) \tag{A-3}$$

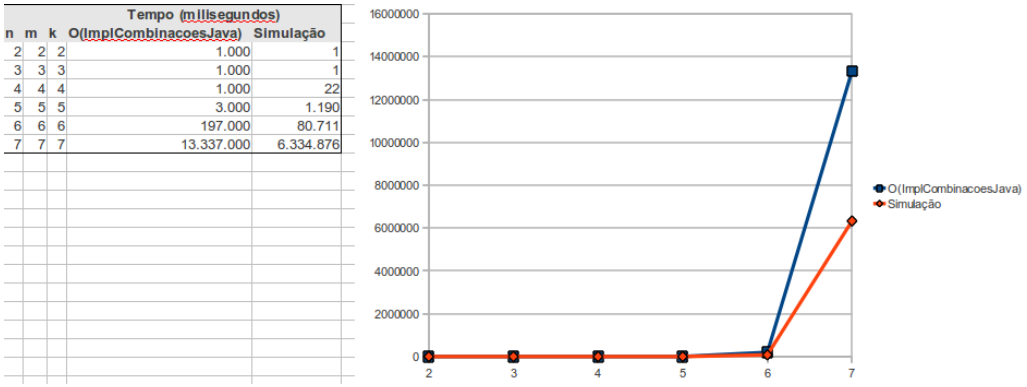
Desta forma, é possível realizar estimativas para a duração máxima dos cálculos feitos na Tabela A.2 e para distribuições feitas com outras dimensões e recursos. A Tabela A.3 apresenta as estimativas de durações máximas dos cálculos feitos na Tabela A.2 e as previsões para matrizes quadradas de 8 a 10 linhas.

Tabela A.3: *Previsões da duração máxima do algoritmo de combinações com um processador Intel(R) Core(TM)2 Duo de 2,66 GHz.*

Dimensão (n x m) - recursos	***O(ImplCombinacoesJava)***	**Tempo**
(2 x 2) - 2	108	1 seg.
(3 x 3) - 3	3.780	1 seg.
(4 x 4) - 4	167.440	1 seg.
(5 x 5) - 5	8.766.450	3 seg.
(6 x 6) - 6	525.903.840	3 min. e 17 seg.
(7 x 7) - 7	35.476.941.192	3 horas, 42 min. e 17 seg.
(8 x 8) - 8	2.655.699.220.800	11 dias 13 horas 19 min. 43 seg.

(9 x 9) - 9	218.363.117.350.950	950 dias 3 horas 9 min. 57 seg.
(10 x 10) - 10	19.560.649.685.777.200	24.855 dias 3 horas 14 min. 7 seg.

A Figura A.3 apresenta a comparação entre as tabelas A.2 e A.3.



			Tempo (milisegundos)	
n	m	k	O(ImplCombinacoesJava)	Simulação
2	2	2	1.000	1
3	3	3	1.000	1
4	4	4	1.000	22
5	5	5	3.000	1.190
6	6	6	197.000	80.711
7	7	7	13.337.000	6.334.876

Figura A.3: *Comparação entre as tabelas A.2 e A.3.*

Na Figura A.3, pode-se notar que, a expressão A-3, é o limite superior, da implementação em Java, do algoritmo A.5 porque, a partir de $n > 4$, todos os resultados da expressão A-3 são superiores aos da simulação.

A.1.2 Algoritmo de Geração das Combinações

Esta seção descreve o algoritmo que foi utilizado para geração de todas as combinações a partir da primeira. A Tabela A.4 apresenta uma matriz de índices, da mesma forma como foi apresentado na Figura A.1.

Tabela A.4: *Tabela de índices do exemplo.*

0	1	2	3
4	5	6	7
8	9	10	11

Neste exemplo, o objetivo é gerar combinações de 4 elementos. Nesta explicação, cada combinação será representada por: $(e1, e2, e3, e4)$, onde $e1$, $e2$, $e3$ e $e4$, são os elementos que fazem parte da combinação.

Para iniciar a geração de combinações, é preciso um ponto de partida. Neste caso, a primeira combinação, que é $(0, 1, 2, 3)$, será este ponto.

A ideia do algoritmo é incrementar sempre o último elemento, neste caso $e4$, até que ele atinga o último índice da matriz de índices, neste caso 11.

Como descrito, o valor máximo para o último elemento é o valor do último índice da matriz de índices. Para o penúltimo, a regra é similar, ou seja, o valor máximo para o penúltimo elemento é o valor do penúltimo elemento da matriz de índices. Esta regra vale para todos os elementos da combinação.

Desta forma, tem-se que, no exemplo apresentado, o valor máximo para $e4$ é 11, para $e3$ é 10, para $e2$ é 9 e para $e1$ é 8.

Sempre que, o último elemento, neste caso $e4$, atinge seu valor máximo, neste caso 11, deve-se verificar se outro elemento, da direita para a esquerda, neste caso $e3$, depois $e2$ e depois $e1$, ainda não atingiu o valor máximo. Por exemplo, na combinação $(0,1,2,11)$, o elemento $e3$ ainda não atingiu seu valor máximo, que é 10.

Quando encontrado, o elemento que não atingiu o valor máximo, deve ser incrementado. Na combinação $(0,1,2,11)$, $e3$ deve receber o valor de $2+1=3$, tornando a combinação igual a $(0,1,3,11)$. Em seguida, deve-se posicionar o cursor no primeiro elemento a direita, neste caso $e4$, e a partir deste, atribuir em seu valor a soma do elemento anterior mais 1, neste caso $e4 = e3+1$. Desta forma, a próxima combinação irá ficar igual a $(0,1,3,4)$.

Quando todos os elementos da combinação atingirem o valor máximo, neste caso, $e1$ igual a 8, $e2$ igual a 9, $e3$ igual a 10 e $e4$ igual a 11, todas as combinações possíveis foram geradas.

No melhor caso, somente o último elemento da combinação é incrementado. Por exemplo, na combinação $(0,1,2,3)$ para a próxima $(0,1,2,4)$, somente incrementou-se o elemento $e4$.

No pior caso, deve-se procurar o elemento que ainda não atingiu o limite, incrementá-lo e recalcular os valores dos elementos mais a direita. Por exemplo, na combinação $(0,9,10,11)$, será encontrado o elemento $e1$, que ainda não atingiu o máximo, e recalculados os elementos $e2$, $e3$ e $e4$, no final a próxima combinação será $(1,2,3,4)$.

Se k é o tamanho da combinação, neste caso 4, então a busca do elemento que ainda não atingiu o máximo terá custo k e o recalculo dos elementos a direita, no pior caso, terá custo $k-1$.

Desta forma, pode-se dizer que o custo de gerar a próxima combinação é $O(proxima) = k+(k-1) = 2k = k$.