Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Angelo Furtado Picanço

Uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos

Rio de Janeiro
2012

Angelo Furtado Picanço

Uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia de Computação da Universidade do Estado do Rio de Janeiro. Área de concentração: Geomática.

Orientador: Prof. Dr. Orlando Bernardo Filho

Rio de Janeiro

2012

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

P586 Picanço, Angelo Furtado.
Uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos / Angelo Furtado Picanço. - 2012.
155 f.

Orientador: Orlando Bernardo Filho.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia de Computação. 2. Engenharia de software – Dissertação. 3. Sistemas de Informação Geográfica — Dissertação. I. Bernardo Filho, Orlando. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 004.41

Assinatura

Data

Angelo Furtado Picanço

Uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia de Computação da Universidade do Estado do Rio de Janeiro. Área de concentração: Geomática.

Aprovado em: 11 de setembro de 2012.

Banca Examinadora:

_____

Prof. Dr. Orlando Bernardo Filho (Orientador)
Faculdade de Engenharia da UERJ

_____

Prof. Dr. Oscar Luiz Monteiro de Farias
Faculdade de Engenharia – UERJ

_____

Prof. Dr. Eduardo Bezerra da Silva
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
Departamento de Informática - CEFET/RJ

Rio de Janeiro

2012

DEDICATÓRIA

À minha amada esposa, Marina.

AGRADECIMENTOS

À minha esposa e eterna namorada Marina, a quem dedico tudo o que faço. Agradeço por ser minha inspiração de vida, meu porto seguro para qualquer parada e maior incentivadora.

À minha mãe, Teresinha (*in memoriam*), minha primeira e mais admirada professora. Agradeço por dedicar toda sua vida à educação e à felicidade de seus filhos. Agradeço especialmente também a meu pai, Cinibalde, e irmão, Ubaldo, por contribuírem em meu amadurecimento.

À minha amada família, simplesmente porque amo a todos. Agradeço especialmente ao Rogério, à Roberta, ao Marcos e à Luciana pela amizade mais que sincera, e à Sandra e ao Ricardo, por me acolherem como um filho.

Aos professores Orlando Filho, Oscar Farias, João Araújo, Guilherme Mota, José Carlos Vasconcellos, Marcelo Sperle e Flávio de Souza, meus tutores durante o Mestrado. Agradeço por compartilharem seu tempo e conhecimento e por me ajudarem imensamente na elaboração dessa dissertação.

Aos meus amigos da Eletrobras, por proporcionarem momentos alegres em tempos de muito trabalho. Agradeço especialmente ao Sérgio Guieiro, ao Sant'ana, ao Pedro Celso, ao Rafael, ao Ricardo Balbi, ao Fabrício Souto e ao Leonardo Almeida pelos cafés, conselhos e ideias.

Aos meus colegas do curso de mestrado, por se mostrarem solidários à disseminação de conhecimento. Agradeço sinceramente a todos do programa pela motivação em aprender e ensinar com aqueles que buscam o conhecimento.

À Universidade do Estado do Rio de Janeiro (Uerj), por ser uma universidade pulsante e cheia de diversidade e estar muito além das letras e das leituras.

RESUMO

PICANÇO, Angelo Furtado. **Uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos**. 2012. 155f. Dissertação (Mestrado em Engenharia da Computação)–Faculdade de Engenharia, Universidade Estadual do Rio de Janeiro, Rio de Janeiro, 2012.

Este trabalho está inserido no campo da Geomática e se concentra, mais especificamente, no estudo de métodos para exploração e seleção de rotas em espaços geográficos sem delimitação prévia de vias trafegáveis. As atividades que poderiam se beneficiar de estudos desse tipo estão inseridas em áreas da engenharia, logística e robótica. Buscou-se, com as pesquisas realizadas nesse trabalho, elaborar um modelo computacional capaz de consultar as informações de um terreno, explorar uma grande quantidade de rotas viáveis e selecionar aquelas rotas que oferecessem as melhores condições de trajetória entre dois pontos de um mapa. Foi construído um sistema a partir do modelo computacional proposto para validar sua eficiência e aplicabilidade em diferentes casos de estudo. Para que esse sistema fosse construído, foram combinados conceitos de sistemas baseados em agentes, lógica nebulosa e planejamento de rotas em robótica. As informações de um terreno foram organizadas, consumidas e apresentadas pelo sistema criado, utilizando mapas digitais. Todas as funcionalidades do sistema foram construídas por meio de software livre. Como resultado, esse trabalho de pesquisa disponibiliza um sistema eficiente para o estudo, o planejamento ou a simulação de rotas sobre mapas digitais, a partir de um módulo de inferência nebuloso aplicado à classificação de rotas e um módulo de exploração de rotas baseado em agentes autônomos. A perspectiva para futuras aplicações utilizando o modelo computacional apresentado nesse trabalho é bastante abrangente. Acredita-se que, a partir dos resultados alcançados, esse sistema possa ajudar a reduzir custos e automatizar equipamentos em diversas atividades humanas.

Palavras-Chave: Engenharia de software; Sistemas baseados em agentes; Lógica nebulosa; Planejamento de rotas em robótica; Sistema de Informações Geográficas; SIG.

ABSTRACT

This research is placed in the field of Geomatics and focuses more specifically on the study of methods for exploration and route selection in geographic areas without prior definition of trafficable roads. Activities that could benefit from such studies are embedded in areas of engineering, logistics and robotics. This study aimed to develop a computational model able to select information from a terrain, explore a lot of viable routes and select those routes that offer the best possible path between two points on a map. It was built a system from the proposed computational model to validate its effectiveness and applicability in different case studies. For this system to be built concepts of agent-based systems, fuzzy logic and route planning in robotics were combined. The information about land were organized, presented and consumed by the system created using digital maps. All features of the system were built using open source. As a result, this research provides an efficient system for the study, planning or route simulation on digital maps, using a fuzzy inference module applied to the classification of routes and a module to operate routes based on autonomous agents. The perspective for future applications using the computational model presented in this study is quite comprehensive. It is believed that from the results, this system can help reduce costs and automate equipment in various human activities.

Keywords: Software engineering; Systems-based agents; Fuzzy logic; Route planning in robotics; Geographic Information System; GIS.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

INTRODUÇÃO

Descobrir rotas entre dois pontos não é uma atividade recente. Na verdade, é uma atividade primitiva, praticada desde as mais remotas civilizações. Entretanto, dependendo das características do terreno sobre o qual se pretende traçar uma rota, da quantidade de caminhos possíveis e do nível de eficiência que se deseja atingir na escolha de um percurso, a pesquisa de rotas nos dias de hoje ainda é uma tarefa difícil.

Atualmente, é comum encontrarmos aplicações na Internet para seleção de rotas. Nessas aplicações, é possível, por exemplo, escolher dois pontos sobre um mapa com ruas e avenidas de uma cidade e obter um percurso para se chegar de um ponto a outro, inclusive com a possibilidade de escolher o meio de transporte mais indicado para realizar o trajeto (ônibus, bicicleta etc.). Portanto, podemos considerar que grande parte dos problemas mais comuns envolvendo a descoberta de rotas em espaços urbanos já está resolvida.

As ferramentas para seleção de rotas disponíveis na Internet normalmente definem caminhos em um espaço previamente delimitado por bairros e vias públicas. Esse fato simplifica bastante o problema, reduz enormemente o processamento de dados e, consequentemente, o tempo de resposta de um computador.

Por outro lado, procurar caminhos em terrenos onde não existem delimitações (ou onde elas não estão definidas explicitamente) agrega bastante complexidade ao problema de seleção de rotas. A utilização de técnicas que possam colaborar para definir rotas em espaços sem delimitações de vias explícitas de forma eficiente é o principal objetivo desse trabalho.

Em espaços geográficos onde não existem vias públicas para restringir as possibilidades de rotas, o esforço computacional ou o tempo de resposta de uma máquina pode inviabilizar a utilização das ferramentas de seleção de rotas mais populares. Quando tratamos de um problema mais simples, como encontrar o melhor caminho entre nossa casa e uma escola de um bairro próximo, por exemplo, praticamente todas as ferramentas de seleção de rotas atuais apresentam uma resposta adequada. Isso porque é explorada uma pequena quantidade de caminhos possíveis (todos demarcados por vias urbanas).

Entretanto, sem a simplificação introduzida pela delimitação de vias urbanas, as possibilidades de caminhos podem ser infinitas. Na seleção de áreas para construção de torres de telecomunicação, por exemplo, as ondas eletromagnéticas são transmitidas pelo ar e sem as restrições impostas por ruas e logradouros. Portanto, seria preciso avaliar as possibilidades de rotas em praticamente todas as direções.

Essa dificuldade imposta na escolha de rotas em espaços sem a delimitação de vias explícitas justifica um estudo mais detalhado do problema. Pode-se considerar que o espaço (um terreno qualquer) oferece diferentes níveis de resistência nas áreas que o compõem. A atribuição desses níveis de resistência depende da atividade abordada. A construção de uma estrada implica em distribuições de esforço diferentes da construção de linhas de telecomunicação.

No caso de uma linha de telecomunicação, a resistência estaria associada à elevação de uma determinada área do espaço estudado, pois os sinais transmitidos são enfraquecidos ao colidir com áreas montanhosas. Já no caso de uma rota de avião, além dos pontos de elevação, outro problema seria o posicionamento de radares e, nesse caso, poderia ser necessário classificar as áreas que compõem um mapa de acordo com a distribuição dos radares em operação.

Seja qual for o problema, o fato é que um espaço geográfico pode ser segmentado por áreas com diferentes níveis de resistência ou esforço para transpô-las. Em outras palavras, um mapa é composto por áreas que demandam níveis de esforço específicos para trafegá-las (níveis de esforço que podem ser determinantes para a escolha de uma rota).

A priori, a melhor trajetória interligando dois pontos de um mapa seria uma linha reta. Entretanto, nem sempre é possível realizar uma trajetória reta. Quando o nível de resistência de uma determinada área do espaço se torna inviável, devemos considerá-la um segmento intransponível e tentar um trajeto alternativo (realizar um contorno). Nesse caso, uma área do espaço pode ser configurada como um obstáculo, como algo que impede uma trajetória reta. Um obstáculo ou uma área intransponível para a construção de uma rodovia, por exemplo, seria uma área de preservação ambiental. Isso porque o esforço para transpor uma área desse tipo pode inviabilizar o empreendimento.

Assim como as áreas de um mapa com níveis de esforço infinitos tornam-se intransponíveis, as áreas com menores níveis de esforço configuram-se como preferenciais. Porém, para que as melhores rotas sejam selecionadas, não basta percorrer as áreas com menor nível de esforço entre os pontos de origem e destino. Isso porque selecionar as menores distâncias entre dois pontos de um mapa também é um fator importante na descoberta de rotas. Neste sentido, para se atingir um resultado eficiente, é preciso encontrar rotas com a melhor relação entre o esforço a ser realizado e a distância a ser percorrida.

A partir destas considerações, esse trabalho pretende apresentar uma combinação dos conceitos de sistemas baseados em agentes e de lógica nebulosa para solucionar problemas relacionados ao descobrimento de rotas em espaços geográficos com obstáculos. Em outras

palavras, busca-se propor um método para classificação das rotas possíveis entre dois pontos no espaço, com base nas medidas “distância” e “esforço”.

Motivação

A principal motivação para esse trabalho advém das inúmeras possibilidades de aplicação que emergem da combinação da cartográfica digital com modelos computacionais de tomada de decisão. Algumas das aplicações possíveis a partir desse trabalho podem contribuir para a realização de simulações e estudos de viabilidade de projetos de estradas e linhas de transmissão. Em alguns casos, poderiam ser realizadas explorações de trajetos sobre mapas e enriquecer o modelo em estudo, até se atingir perspectivas de custos adequados para a execução de um projeto.

A execução de simulações de trajetórias visando à escolha das melhores possibilidades de percurso sobre um terreno não seria útil apenas em empreendimentos de engenharia. Outro exemplo de aplicação seria a construção de máquinas ou robôs capazes de planejar e selecionar caminhos de forma independente (autônoma). Através de um sistema de tomada de decisão, uma máquina seria capaz de executar e escolher um entre diversos caminhos possíveis sobre um mapa. Esse tipo de planejamento automático de rota pode colaborar para a exploração de regiões de difícil acesso, insalubres ou com alto nível de periculosidade.

Outras simulações destinadas ao entendimento de características sociais e econômicas também poderiam ser feitas a partir de dados históricos de um terreno. Nesse caso, uma simulação considerando a sobreposição de informações relacionadas à cartografia de um terreno pode contribuir para a descoberta de novos conhecimentos de rotas ou para estudos em diversas áreas de conhecimento.

Muitas outras aplicações envolvendo simulações, empreendimentos e projetos de engenharia, robótica e cartografia poderiam surgir dos estudos de rotas em espaços sem a delimitação de vias.

Neste sentido, as principais motivações desse trabalho são:

a) Combinar técnicas que viabilizem os estudos de rotas de uma forma prática e eficiente; e
b) Oferecer uma ferramenta de exploração de rotas com utilidade para diversas áreas de conhecimento.

Nota-se que, a partir das possibilidades apresentadas, seria possível se obter grandes benefícios com os estudos de rotas. Entretanto, o desenvolvimento do método para processar essas informações pode demandar bastante esforço. Sendo assim, o sucesso na busca de tais benefícios depende de um processo flexível e com alta aplicabilidade[1].

Justificativa

Os estudos de rotas em espaços abertos são apresentados com considerável importância já há algum tempo. Em alguns casos são utilizadas ferramentas de geoprocessamento na definição de rotas, com o objetivo de elevar a qualidade técnica e operacional em projetos de linhas de transmissão (Belém, et al., 2009). Outra utilidade prevista para tecnologias desse tipo seria estabelecer uma avaliação da qualidade e da confiabilidade de novos projetos em comparação aos projetos concebidos no passado.

A utilização de mapas digitais e de técnicas de sobreposição (*overlay*) na solução de problemas de rotas é considerada simples e eficiente (Belém, et al., 2009). Os mapas digitais normalmente compõem uma importante base de informações cartográficas que pode ser utilizada por um algoritmo capaz de definir o traçado para o caminho de menor custo sobre um terreno.

Alguns dos algoritmos utilizados para definir traçados em mapas digitais são: o algoritmo de Dijkstra (W. Dijkstra, 1982), a Programação Dinâmica (Monteiro, et al., 2005) e o A* (S. Mata & S. B. Mitchell, 1997). Diferentes níveis de eficiência podem ser obtidos de acordo com as configurações do cenário sobre o qual os algoritmos executam. Além disso, diversos caminhos podem ser gerados de acordo com os critérios definidos para a classificação de rotas. Entretanto, apesar das diferenças nos traçados dos caminhos, todos os resultados apresentam o mesmo custo de execução (Belém, et al., 2009).

A combinação entre algoritmos para traçados de rotas e mapas digitais, consolidando informações e características temáticas relacionadas à vegetação, ao relevo, aos riscos e às restrições ambientais de um terreno, é comprovadamente uma técnica eficiente para seleção de uma rota de custo mínimo (Belém, et al., 2009). Um mapa consolidado, neste caso, terá uma representação do tipo matricial (*raster*), onde cada *pixel* estará associado a um custo unitário. Essa estratégia permite que algoritmos de busca em grafos sejam empregados para selecionar uma rota de menor custo.

[1] Aplicabilidade nesse caso foi utilizada com o sentido de ter utilidade em diferentes aplicações.

Nota-se, nas soluções apresentadas, que apenas uma variável definirá o critério de seleção da melhor rota: o custo. Essa definição sugere os seguintes questionamentos a respeito da distância de um caminho traçado:

a)Deve-se assumir que os algoritmos irão selecionar sempre as rotas de menor distância e menor custo?
b)O custo de uma célula do mapa matricial está relacionado com alguma medida de distância?
c)Como serão processados os caminhos com uma longa distância e um baixo custo?
d)Como serão processados os caminhos com uma curta distância e um alto custo?
e)Como poderiam ser selecionados os caminhos com a melhor relação entre o custo e a distância?

Além das questões relacionadas acima, deve ser considerada também a possibilidade de se explorar a maior quantidade de caminhos possíveis. Um dos maiores desafios desse tipo de problema é processar um grande número de possibilidades de rotas com rapidez e eficiência. Para isso, é preciso utilizar modelos computacionais flexíveis e capazes de processar grandes volumes de dados. Os algoritmos de busca em grafo citados apresentam diferentes níveis de eficiência no processamento de um mapa de custos e a diversidade de caminhos explorada no mapa está diretamente relacionada ao tempo de processamento de cada algoritmo.

A programação dinâmica percorre todos os pontos da matriz de custo, o que significa uma exploração mais ampla do mapa. Entretanto, dependendo do tamanho da área a ser explorado, o tempo de processamento desse algoritmo pode inviabilizar uma simulação. A programação dinâmica tem ordem de complexidade $O(n)$, onde n representa a quantidade de elementos da matriz de custos. A programação dinâmica é capaz de informar o caminho ótimo entre dois pontos de uma matriz de custos (Monteiro, et al., 2005).

Para evitar o processamento pesado de todos os elementos de uma matriz de custos, o algoritmo A* poderia ser empregado no lugar da programação dinâmica. Esse algoritmo pode executar, com ordem de complexidade $O(n\ log(n))$, onde n representa a quantidade de elementos da matriz de custos. Isso significa dizer que o caminho ótimo entre dois pontos poderá ser encontrado sem que todos os elementos da matriz sejam visitados. Embora essa característica seja positiva para o tempo de processamento, ela compromete a diversidade de caminhos explorados no mapa.

Portanto, seja qual o for o algoritmo escolhido para selecionar o melhor caminho em uma matriz de custos, a diversidade de caminhos explorados implicará em um maior tempo de processamento. Além disso, empregando a técnica de sobreposição de mapas para consolidar a matriz de custos, será preciso planejar uma forma de relacionar as medidas de distância ao estudo do problema. As principais justificativas para o desenvolvimento de uma pesquisa mais abrangente sobre a seleção de rotas utilizando mapas digitais nesse contexto são:

a)Criar um processo capaz de absorver as informações geradas a partir de mapas digitais;
b)Propor uma nova abordagem para a seleção de rotas em mapas digitais;
c)Gerar resultados em tempo hábil, mesmo nos casos em que uma grande quantidade de rotas precisa ser analisada; e
d)Apresentar um método flexível e com alta aplicabilidade.

Com base nas justificativas apresentadas, as metas desse trabalho estão fundamentadas na pesquisa de modelos computacionais que possam ser combinados para processar uma grande quantidade de informações cartográficas com o objetivo de gerar classificações de rotas sobre mapas digitais.

Objetivo

Um computador é capaz de processar muitas informações e trabalhar incessantemente durante horas, dias ou até anos. Porém, nem sempre um computador é capaz de tomar decisões rapidamente. Em muitos casos, como na seleção de rotas, a quantidade de alternativas possíveis para se chegar a uma solução pode levar um computador a extrapolar o tempo hábil para a tomada de decisão.

Para resolver a demora no processamento e não extrapolar o tempo disponível durante a tomada de decisão, é necessário buscar modelos computacionais capazes de encontrar soluções sem necessariamente processar todas as informações disponíveis. Nesses casos, é preciso encontrar uma solução eficiente em um tempo hábil, mesmo que ela não seja a melhor solução.

Qualquer decisão, seja computacional ou humana, vai sempre depender do tempo disponível e do nível de eficiência exigido para o resultado que virá da decisão tomada. Por exemplo, quando precisamos escolher um casaco para usar, olhamos pela janela para ver

como está o clima e decidimos o tipo de casaco que será adequado. Talvez, uma pessoa muito precavida consulte ainda informações meteorológicas. No caso de uma viagem, pode ser realmente importante ter informações precisas sobre o clima previsto para se tomar a decisão mais acertada. De qualquer forma, o fato é que não podemos ficar por tempo indefinido analisando alternativas e possibilidades, nem tomar decisões que levem a consequências indesejadas.

Alguns modelos computacionais são ligeiramente parecidos com a forma como nós tomamos decisões. Para resolver problemas de seleção de rotas em espaços com obstáculos, é importante considerar modelos:

a)Capazes de tomar decisões sobre medidas imprecisas e parametrizáveis;
b)Capazes de agir de forma autônoma para explorar uma grande diversidade de rotas; e
c)Simples de construir.

Cabe enfatizar que, quaisquer que sejam os modelos computacionais utilizados, as informações cartográficas necessárias durante a exploração de rotas sempre poderão ser consolidadas através da sobreposição de mapas digitais. Portanto, os principais objetivos desse trabalho são:

a)Propor uma técnica híbrida de modelos computacionais para classificação de rotas em espaços com obstáculos (sistemas baseados em agentes e lógica nebulosa);
b)Apresentar um processo de auxílio à tomada de decisão flexível, de alta aplicabilidade e que utilize mapas digitais como informações de entrada; e
c)Selecionar rotas eficientes entre dois pontos de um mapa digital.

Para atingir esses objetivos foram realizadas pesquisas sobre uma grande variedade de aplicações, envolvendo técnicas de cartografia digital e computação. Os estudos desse trabalho podem ser resumidos a partir de uma combinação de conceitos das seguintes áreas de conhecimento:

a)Sistemas baseados em agentes;
b)Lógica nebulosa e sistemas de inferência nebulosos;
c)Algoritmos de planejamento de rotas;

d)Engenharia de software; e

e)Cartografia digital.

Organização do trabalho

Esse trabalho foi organizado nos seguintes capítulos:

a)Fundamentos teóricos;

b)Abordagem proposta;

c)Implementação;

d)Resultados; e

e)Conclusão.

No capítulo “Fundamentos teóricos”, serão apresentados os principais conceitos de sistemas baseados em agentes, de lógica nebulosa e de algoritmos de planejamento de rotas. Esse capítulo apresentará informações básicas para o entendimento dos assuntos pesquisados e dos resultados alcançados.

Após a apresentação das principais bases teóricas a serem utilizadas nesse trabalho, será descrita a abordagem proposta para solucionar os problemas relacionados à seleção de rotas em espaços com obstáculos. Nesse capítulo, serão detalhados com mais profundidade os fluxos de informação e a arquitetura utilizada na solução proposta. Em outras palavras, será apresentada uma solução conceitual para o método de trabalho. As contribuições feitas a partir de adaptações nos modelos computacionais também serão explicadas nesse capítulo.

No capítulo “Implementação” serão detalhados em termos técnicos todos os aspectos descritos na solução conceitual do trabalho. Além disso, serão fornecidas explicações sobre as ferramentas utilizadas, os modelos de dados empregados, as rotinas utilizadas na implementação de processos etc. Dessa forma, todas as estruturas utilizadas no sistema construído para avaliar a solução dada ao problema de seleção de rotas serão explicadas nesse capítulo.

O capítulo de resultados descreverá os testes e as avaliações feitas a partir da execução do sistema construído. Nesse capítulo, os resultados de cada um dos algoritmos de planejamento de rotas empregados serão confrontados e detalhadas suas particularidades em cada cenário de testes. Nesse momento, serão enfatizadas variações na eficiência de cada algoritmo de acordo com a mudança de cenário de teste.

O último capítulo apresentará informações conclusivas sobre o trabalho: quais objetivos foram alcançados com sucesso, quais objetivos não foram atingidos, em que pontos o método proposto precisa ser melhorado e onde pode ser expandido. No capítulo de conclusões, serão relatadas informações a respeito do que pode ser obtido desse trabalho, qual sua aplicabilidade, qual sua utilidade e quais aspectos podem ser explorados ou ampliados futuramente.

1. **FUNDAMENTOS TEÓRICOS**

Nesse capítulo, serão apresentados conhecimentos essenciais para o entendimento de todo o trabalho.

1.1.**Apresentação**

Para realizar a descoberta de rotas em espaços geográficos abertos, propõe-se nesse trabalho a fundamentação teórica em três disciplinas:

a)Sistemas baseados em agentes;
b)Lógica nebulosa; e
c)Algoritmos para o planejamento de rotas.

Essas disciplinas combinadas compõem o ferramental necessário para resolver questões relacionadas ao percurso e à classificação de grandes quantidades de rotas. Com base na teoria de sistemas baseados em agentes, pretende-se aplicar a autonomia necessária para explorar um grande número de possibilidades de pontos sobre um mapa. Os algoritmos para o planejamento de rotas servem nessa abordagem como uma fonte de informação importante para definir o comportamento desses agentes exploradores de rotas. Finalmente, a lógica nebulosa servirá como base teórica para a construção de um sistema de inferência capaz de classificar as melhores rotas exploradas pelos agentes.

Nesse capítulo, primeiramente serão descritos os conceitos de sistemas baseados em agentes. Serão abordadas as principais propriedades e definições de um agente e do ambiente no qual ele executa. A partir dessas definições essenciais, serão descritas as notações formais para os conceitos que compõem a teoria. Por fim, serão apresentados os principais tipos de sistemas baseados em agentes e a evolução dos conceitos mais importantes em cada sistema.

Após a apresentação dos sistemas baseados em agentes, será apresentada a lógica nebulosa. Incialmente serão apresentados os conjuntos nebulosos através de uma analogia com a teoria dos conjuntos clássica. A partir da teoria dos conjuntos nebulosos, será descrito o funcionamento de uma de suas principais aplicações: o sistema de inferência nebuloso. Serão apresentados os principais módulos envolvidos no funcionamento de um sistema de inferência nebuloso, os tipos de funções de pertinência para tratamento da entrada de dados, o método de

inferência e base de regras e as funções para consolidar os resultados das inferências realizadas em um valor de saída do sistema.

Para finalizar a fundamentação teórica, serão apresentados os algoritmos para o planejamento de rotas. Os conceitos apresentados serão organizados em quatro classificações básicas: algoritmos de planejamento por decomposição celular, *roadmaps* e campo potencial. Para cada tipo de algoritmo, serão apresentadas suas principais características, seu funcionamento e os problemas para os quais cada um se aplica.

1.2.**Sistemas baseados em agentes**

Um agente é uma entidade computacional capaz de ações autônomas, a fim de atingir os objetivos para o qual foi projetado (Wooldridge, 2002). Também pode ser explicado como uma entidade que executa em um ambiente computacional multiprocessado, no qual cada participante (ou componente do ambiente) tem autonomia para decidir quando deve ou não responder às requisições que recebe.

Já um sistema multiagente (*multiagents system*) pode ser definido como um sistema com um número qualquer de agentes que interagem entre si e a fim de solucionar um problema, ou de atender aos objetivos de um projeto, porém, sem compartilhar de objetivos e motivações comuns (Wooldridge, 2002). Em outras palavras, em um sistema multiagentes, espera-se que a solução de um problema surja (emerja) dos conflitos de objetivos de diversos agentes em interação, o que sugere um ambiente de cooperação, coordenação e negociação.

Uma propriedade importante de um agente e que distingue os sistemas baseados em agentes de outros sistemas é o fato de um agente ter consciência sobre o mundo no qual está inserido. Um agente é capaz de fazer uso dessa consciência para executar ações. Podemos assumir que, de forma generalizada, o principal objetivo de qualquer agente é escolher a ação mais correta a ser executada sobre um determinado domínio de aplicação (Wooldridge, 2002).

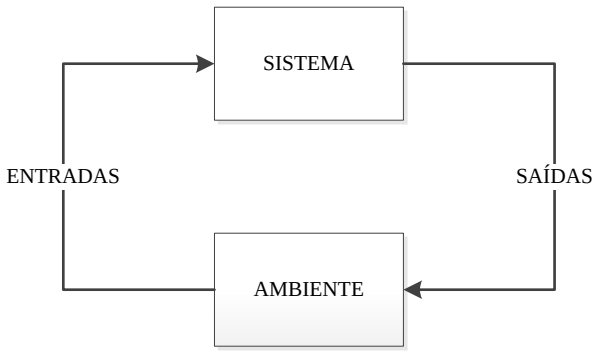


Figura 1: Fluxograma básico de um sistema baseado em agentes.

Isso implica em incorporar determinadas características similares as de entidades do mundo real, tais como (Wooldridge, 2002):

a)Reatividade (*Reactivity*): No mundo real, as coisas mudam o tempo todo e as informações podem ser transmitidas com erros ou incompletas. Um ambiente real é bastante dinâmico. O conceito de reatividade está relacionado à capacidade de um agente interagir e reagir às mudanças de seu ambiente. Um agente é reativo quando se mantém interagindo com seu ambiente e respondendo às mudanças que ocorrem em tempo útil (enquanto a resposta ainda pode ser útil);

b)Proatividade (*Proactiveness*): Além de reagir aos estímulos que recebe, é importante para um agente manter seus comportamentos direcionados aos seus objetivos. Portanto, não basta ser orientado pelos eventos, é necessário ter iniciativa e identificar oportunidades. Esse é o conceito de proatividade; e

c)Sociabilidade (*Social Ability*): Assim como no mundo real, no ambiente em que um agente está inserido, muitas vezes não é possível atingir objetivos sem a colaboração de outros agentes. Dessa forma, em diversas situações torna-se importante a capacidade de um agente se comunicar e cooperar com outros para atingir seus objetivos.

Além das propriedades descritas acima, também fazem parte do contexto de discussões sobre agentes a capacidade de mobilidade de um agente (*Mobility*), a capacidade de mentir (*Veracity*), a capacidade de colaborar em negociações (*benevolence*), a capacidade de não agir deliberadamente e tomar atitudes que possam comprometer seus objetivos (*Rationality*) e a capacidade de melhorar seu rendimento ao longo do tempo (*Learning/adaption*).

Nota-se que as ações de um agente dependem muito das propriedades e do estado do ambiente no qual ele está inserido. Alguns dos principais tipos de ambiente são (Wooldridge, 2002):

a)Acessível (*Accessible versus inaccessible*): Um ambiente é acessível quando permite que um agente obtenha informações (completas, com precisão e quando necessitar) sobre seu estado. Quanto mais acessível um ambiente, mais simples a construção do agente para operar nele. Entretanto, quanto mais características de ambientes reais um sistema baseado em agentes incorporar, mais inacessível será seu ambiente.

Situações reais raramente estão relacionadas com informações precisas, completas e disponíveis;

b)Determinístico (*Deterministic versus non-deterministic*): Um ambiente é determinístico quando o resultado de uma ação resulta em um estado previsível e sem incertezas. Mais uma vez, quanto mais próximo de um ambiente real, mais incertezas terá um ambiente baseado em agentes, o que implicará em mais dificuldade para se projetar agentes que executem nele;

c)Episódico (*Episodic versus non-episodic*): O conceito de episódio está relacionado à diversidade de cenários que podem ser afetados pela ação de um agente. Em um ambiente episódico, um agente depende de um número pequeno de episódios para tomar decisões e não existem ligações entre as ações desse agente em cenários diferentes. Em outras palavras, em um ambiente episódico, um agente toma decisões apenas para o cenário corrente, sem ter preocupações com o impacto de suas ações em cenários futuros;

d)Estático (*Static versus dynamic*): Ambientes estáticos têm seu estado alterado apenas pelas ações dos agentes inseridos nele. Ambientes dinâmicos, por outro lado, podem sofrer mutações provenientes de ações de outros processos, além das executadas por seus agentes; e

e)Discreto (*Discrete versus continuous*): Os ambientes discretos comportam um número finito de ações e percepções sobre ele, enquanto os ambientes contínuos permitem que as interações com um agente perdurem infinitamente.

A partir dos conceitos apresentados anteriormente, serão descritas as notações formais utilizadas para representar os componentes (ou a arquitetura abstrata) de um sistema baseado em agentes (Wooldridge, 2002). Um ambiente pode ser descrito formalmente como um conjunto finito de estados possíveis, conforme expresso abaixo:

$$E = \{e_0, e_1, \ldots\}$$

Esses estados instantâneos são configurados de acordo com as ações tomadas por um agente. Portanto, um agente possui um conjunto de ações capazes de modificar o estado de um ambiente. Esse conjunto de ações pode ser expresso conforme abaixo:

$$Ac = \{\alpha_0, \alpha_1, \ldots\}$$

As ações são tomadas por um agente durante sua execução. A execução de um agente pode ser descrita como uma sequência de estados de um ambiente encadeados pelo conjunto de ações tomadas por esse agente. Assim sendo, uma execução r pode ser expressa conforme apresentado abaixo:

$$r: e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_{u-1}} e_u$$

Tanto um ambiente quanto um agente podem ser representados formalmente por funções. O comportamento de um ambiente pode ser representado por uma função de transformação onde um subconjunto de sequências de estados e ações possíveis, finalizados por uma ação, leva a um novo estado do ambiente, conforme expresso abaixo:

$$\mathrm{T}: \mathcal{R}^{\mathrm{Ac}} \rightarrow \wp(E)$$

Onde:

$\mathcal{R}^{\mathrm{Ac}}$ *é o conjunto de estados e ações finalizados por uma ação; e*
$\wp(E)$ *é o estado obtido a partir da ação tomada, no conjunto de estados possíveis* $\wp$*.*

Portanto, podemos representar um ambiente através dos três componentes apresentados abaixo:

$$Env = \langle E, e_0, \mathrm{T} \rangle$$

Onde:

E *representa o conjunto de estados possíveis;*
e_0 *o estado inicial sendo* $e_0 \in E$*; e*
T *a função de transformação.*

Um agente toma uma decisão de executar uma ação com base nas mudanças de estado do sistema. Portanto, um agente pode ser representado por uma função de transformação onde um subconjunto de sequências de estados e ações possíveis, finalizados por um estado, leva a uma nova ação do agente, conforme expresso abaixo:

$$Ag: \mathcal{R}^{\mathrm{E}} \rightarrow Ac$$

Onde:

$\mathcal{R}^{E}$ *é o conjunto de estados e ações finalizados por um estado; e*
Ac *é a ação tomada com base no histórico do sistema.*

Assim sendo, podemos concluir que um sistema é uma composição do par agente Ag e ambiente Env, associado a um conjunto de execuções possíveis e finitas, conforme expresso abaixo:

$$\mathcal{R}(Ag, Env)$$

Onde:

$\mathcal{R}$ *é o conjunto de todas as sequências finitas possíveis geradas a partir de Env e Ag.*

Feita a apresentação formal dos principais conceitos envolvidos na arquitetura de um sistema baseados em agentes, a seguir serão apresentados os tipos de sistemas mais comuns (Wooldridge, 2002).

Um tipo de sistema bastante simples é o chamado puramente reativo (*purely reactive*). Em sistemas desse tipo, um agente toma decisões baseadas apenas em sua situação atual, sem a preocupação com informações históricas. As ações de agentes em sistemas puramente reativos podem ser representadas pela relação entre o conjunto de estados possíveis de um ambiente e o conjunto de ações tomadas por um agente, conforme expresso abaixo:

$$action: E \rightarrow Ac$$

Onde:

E é o conjunto de estados do ambiente no qual executa um agente; e
Ac é o conjunto de ações tomadas a partir de um conjunto de estados.

Um bom exemplo para esse tipo de sistema é um termostato[2]. Seu funcionamento é puramente reativo porque, quando em execução, um termostato simplesmente verifica o último estado do ambiente (a temperatura corrente) e toma a decisão de resfriá-lo, ou não, dependendo da temperatura lida.

Um tipo de sistema um pouco mais sofisticado que o puramente reativo é o chamado sistema de percepção (*perception system*). Nesse tipo de sistema, são incorporadas habilidades aos agentes para que uma ação seja tomada com base, não apenas na situação corrente do agente, mas a partir da análise de um conjunto de percepções feitas durante sua execução.

[2] Termostato é um dispositivo utilizado no controle de temperatura, comum em equipamentos de refrigeração.

Para analisar as situações de seu ambiente, um agente de um sistema de percepção possui funções para observar e agir. A ilustração abaixo apresenta o modelo de um sistema de percepção com suas funções *see* e *action*:

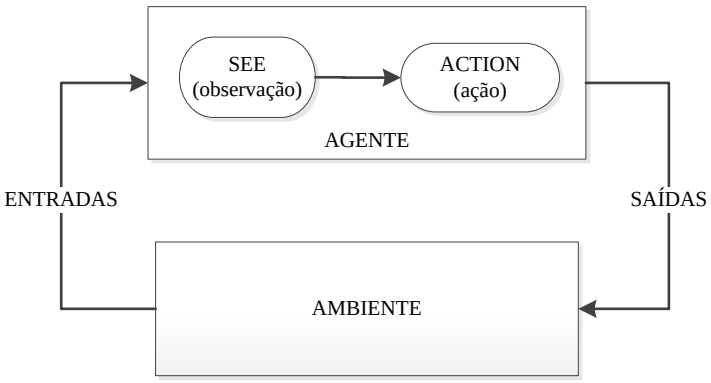


Figura 2: Fluxograma de um sistema de percepção.

A função *see* de um sistema com percepção é responsável pela coleta de informações provenientes das observações feitas por um agente sobre seu ambiente. O resultado obtido através da função *see* é a transformação do conjunto de estados de um ambiente em um conjunto de percepções feitas por um agente, conforme expresso abaixo:

$$see: E \rightarrow Per$$

Onde:

E é o conjunto de estados do ambiente no qual executa um agente; e
Per é o conjunto de percepções feitas a partir dos estados de um ambiente.

Cada estado do ambiente relacionado no conjunto de percepções feitas por um agente servirá de insumo para as decisões a serem tomadas e implicarão nas ações executadas pelo agente. A função *action* representa, nesse contexto, essa tomada decisão e relaciona uma sequência de percepções às consequentes ações que afetarão o ambiente, conforme expresso abaixo:

$$action: Per* \rightarrow A$$

Onde:

Per é uma sequência de percepções; e*
A o conjunto de ações relacionadas às percepções.

Nos sistemas de percepção, embora várias percepções sejam geradas a cada ciclo de execução, as percepções de um agente não são preservadas (armazenadas) pelo sistema. Em outras palavras, um agente faz suas observações sobre o ambiente, toma uma decisão e altera seu ambiente, mas não se preocupa com a possibilidade de posteriormente necessitar reavaliar suas observações passadas.

Outro tipo de sistema um pouco mais sofisticado que o de percepção é o chamado sistema de agentes com estado (*agents with states*). Nesse tipo de sistema, as percepções obtidas são armazenadas e utilizadas em decisões futuras. Portanto, um conjunto de estados do ambiente no qual o agente está inserido é armazenado e preservado a cada ciclo de observação. Dessa forma, as ações são tomadas com base não apenas no último conjunto de percepções, mas a partir de um histórico de percepções adquiridas ao longo da vida de um agente. Segue uma ilustração de um sistema de agentes com estado:

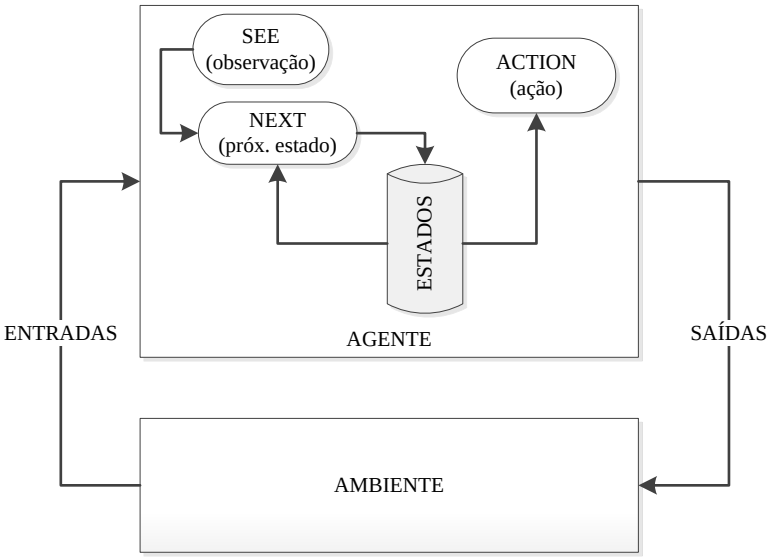


Figura 3: Sistema de agentes com estado.

Para representar um sistema de agentes com estado são necessárias duas alterações no sistema de percepção: primeiramente a função *action* é alterada para consultar a base histórica de percepções e, por fim, uma nova função chamada *next* é incorporada ao sistema para relacionar o conjunto das últimas percepções feitas com as informações da base histórica.

A expressão abaixo representa a relação entre o estado interno do sistema e uma percepção feita, onde I corresponde ao estado interno do sistema.

$$next: I \times Per \rightarrow I$$

Onde:

I o é conjunto de todos os estados internos de um agente; e
Per é o conjunto de percepções geradas pela função see.

Por fim, segue a alteração feita na função *action*, utilizada na apresentação dos sistemas de percepção, para refletir a nova forma de tomada de decisão dos sistemas de agentes com estado, onde a base histórica é utilizada. A função *see* permanecerá idêntica àquela utilizada nos sistemas de percepção.

$$action: I \rightarrow Ac$$

Onde:

I é o conjunto dos estados internos de um agente; e
Ac é o conjunto de ações tomadas a partir de um conjunto de estados.

Em suma, a função *action* gera seus resultados a partir dos estados internos *I* gerados pela função *next*, a partir das percepções geradas pela função see, conforme representado abaixo:

$$action(next(i_0, see(e)))$$

Onde:

i^0 *é estado inicial de um agente;*
e é um estado do ambiente no qual o agente executa;
$see(e)$ *gera uma nova percepção; e*
$next(i_0, see(e))$ *atualiza o estado interno de um agente.*

Generalizando todos esses tipos de sistemas apresentados, pode-se concluir que todo agente segue um fluxo de controle bastante simples que inicia em um estado zero, a partir do qual são feitas observações sobre o ambiente onde o agente executa. O estado interno do agente é então atualizado com informações sobre seu ambiente e novas ações são tomadas.

Agentes são construídos para executar tarefas com autonomia. É possível delegar tarefas a um agente especificando apenas o que precisa ser feito, sem necessariamente dizer como se fazer. Esse fato, associado à redução dos custos de componentes computacionais, ao crescimento das redes de computadores e ao crescimento da complexidade de tarefas que nos tornamos capazes de automatizar, é o que viabiliza projetos de sistemas capazes de tomar decisões sem intervenção humana (Wooldridge, 2002).

Entretanto, os modelos apresentados até aqui ainda carecem de interações mais ricas, com informações que auxiliem a tomada de decisão de um agente. Uma proposta seria

estabelecer critérios para a escolha de um estado entre os vários que compõem o conjunto de estados E possíveis em um ambiente. Essa é exatamente a ideia por trás do conceito de função de utilidade (*functions over states*) (Wooldridge, 2002). O objetivo dessa abordagem é relacionar um nível de utilidade (um número real) com cada estado possível de um ambiente e, dessa forma, possibilitar que um agente maximize a escolha de estados. Segue a função que associa um número real aos estados de um ambiente:

$$u : E \rightarrow \mathbb{R}$$

Onde:

E é o conjunto dos estados possíveis de um ambiente; e
$\mathbb{R}$ é o conjunto dos números reais.

Nota-se que o conceito de utilidade atribuído a cada estado de um ambiente poderia ser generalizado para qualquer parâmetro que diferenciasse um estado de outro (por exemplo, o esforço em transitar entre estados). Com essa proposta, no entanto, introduzimos uma nova questão relacionada à eficiência da execução de um agente: como definir o resultado (ou valor) final de uma execução completa? A partir da soma, da média ou do valor mínimo dos valores avaliados?[3]

Uma abordagem possível seria criar uma relação de utilidade, não com estados individuais de um ambiente, mas sim com a execução de vários deles. Essa relação pode ser expressa conforme abaixo:

$$u : \mathcal{R} \rightarrow \mathbb{R}$$

Onde:

$\mathcal{R}$ é o conjunto execuções possíveis e finitas; e
$\mathbb{R}$ é o conjunto dos números reais.

Assumindo que o conceito de utilidade de uma execução esteja associado à relação entre o número de tarefas executadas com sucesso e o total de tarefas que precisam ser concluídas, pode-se denotar essa função de utilidade conforme abaixo:

[3] O problema citado foi observado durante a avaliação dos resultados obtidos na seleção de rotas em espaços geográficos com obstáculos. Para que uma rota fosse selecionada, foi levado em consideração o nível de esforço empregado por um agente ao realizá-la.

$$u(r) \cong \frac{tarefas\ bem\ sucedidas\ em\ r}{total\ de\ tarefas\ em\ r}$$

Onde:

u é a função de utilidade para r; e
r representa a execução de um agente.

Se a probabilidade de uma execução ocorrer é a relação entre um evento de uma execução qualquer e o conjunto de todas as possíveis execuções que emergem das interações entre agentes Ag e ambientes Env do sistema, pode-se assumir que a soma dessas probabilidades será igual a um, conforme expresso abaixo (Wooldridge, 2002):

$$\sum_{r \in R(Ag,Env)} P(r \mid Ag,Env) = 1$$

Onde:

r representa o evento de uma execução; e
$R(Ag, Env)$ *é o conjunto de todas as execuções possíveis do agente Ag no ambiente Env.*

Portanto, pode-se concluir que a utilidade esperada (*expected utility*) de um agente Ag inserido em um ambiente Env é estimada pelo produto entre sua função de utilidade u e a probabilidade de sua execução ocorrer.

$$EU(Ag,Env) = \sum_{r \in R(Ag,Env)} u(r).P(r \mid Ag,Env)$$

Onde:

r representa o evento de uma execução;
$R(Ag, Env)$ *é o conjunto de todas as execuções possíveis do agente Ag no ambiente Env; e*
$EU(Ag, Env)$ *é a utilidade esperada para um conjunto de execuções de Ag.*

Um agente é classificado como ideal (*optimal agent*) quando é capaz de maximizar a medida de utilidade esperada para sua execução. Ser um agente ideal não significa ser o agente com a melhor execução sempre, mas apenas estar entre os agentes com a possibilidade de melhor execução (atingir a média das melhores execuções). Possivelmente um agente ideal

terá a melhor execução em relação aos demais agentes de um ambiente. Um agente ideal é denotado pela expressão abaixo (Wooldridge, 2002):

$$Ag_{opt} = arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env)$$

Onde:

$EU(Ag, Env)$ *é a utilidade esperada para um conjunto de execuções de Ag; e*
$\mathcal{AG}$ *é um conjunto de agentes do ambiente Env.*

O conceito de *optimal agent* levanta uma questão relacionada à capacidade de processamento de um sistema computacional. Se, por um lado, existem agentes classificados com um bom desempenho (capazes de maximizar a medida de utilidade esperada), por outro lado podem existir agentes que não executem em determinados computadores. Isso porque a relação entre o conjunto de estados possíveis e as ações tomadas pelo agente ($Ag: \mathcal{R}^E \rightarrow Ac$) pode exigir mais memória do que o sistema é capaz de disponibilizar.

O planejamento da capacidade de processamento de um computador no contexto de sistemas baseados em agentes está relacionado ao conceito de *bounded optimal agents* (delimitação de agentes classificados como ideais). Essa delimitação pode ser denotada formalmente da seguinte maneira:

Sendo:

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ pode executar em } m\}$$

Onde:

m *representa o sistema computacional onde um agente deverá executar; e*
$\mathcal{AG}$ *é um conjunto de agentes do ambiente Env.*

Conclui-se que:

$$Ag_{bopt} = arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env)$$

Onde:

$EU(Ag, Env)$ *é a utilidade esperada para um conjunto de execuções de Ag;*
$\mathcal{AG}_m$ *é o conjunto de agentes do ambiente Env que podem executar em m; e*
Ag_{bopt} *representa o bouded optimal agent.*

Existem duas atribuições de utilidade especiais para a execução de agentes: zero e um (representando sucesso ou fracasso, respectivamente). A execução de uma agente recebe o status de bem sucedida, dependendo do grau de utilidade que alcançar em relação aos objetivos de projeto. Os status de sucesso e fracasso estão relacionados ao conceito de *predicate task specifications* (especificação de tarefas de predicado), denotado conforme abaixo:

$$\psi = \mathcal{R} \rightarrow \{0, 1\}$$

Onde:

$\mathcal{R}$ representa o conjunto de execuções possíveis;
0 representa o status de fracasso; e
1 o status de sucesso.

A partir da relação entre a especificação de tarefas de predicado e o ambiente de um sistema baseado em agentes, pode-se estabelecer um conjunto de ambientes de tarefas (*task environment*). Um ambiente de tarefas é denotado pelo par ambiente e predicado de execuções, conforme expresso abaixo (Wooldridge, 2002):

$$\mathcal{T}\varepsilon = \langle Env, \psi \rangle$$

Onde:

Env representa um ambiente; e
ψ é um predicado sobre execuções.

Basicamente, um ambiente de tarefas especifica:

a)As propriedades do ambiente no qual um agente irá habitar; e
b)Os critérios pelos quais um agente será julgado como bem sucedido ou mal sucedido.

Com isso, a partir do conjunto de execuções de um agente, será possível extrair o subconjunto de execuções que satisfazem especificações de tarefas de predicado. Esse subconjunto de execuções é denotado pela expressão que segue:

$$\mathcal{R}_{\psi}(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \; e \; \psi(r) = 1\}$$

Onde:

Env representa um ambiente;
Ag um agent;
r uma execução de Ag;
$\mathcal{R}$ o conjunto de execuções possíveis; e
ψ a especificação de uma tarefa (predicado).

Pode-se concluir, com isso, que um agente obtém status de bem sucedido quando o conjunto de todas as suas execuções possíveis é igual ao conjunto de execuções que satisfazem as especificações de tarefas, conforme denotado abaixo:

$$\mathcal{R}_{\psi}(Ag, Env) = \mathcal{R}(Ag, Env)$$

Onde:

Env representa um ambiente;
Ag um agent;
r uma execução de Ag;
$\mathcal{R}$ o conjunto de execuções possíveis; e
ψ a especificação de uma tarefa (predicado).

Se a probabilidade de um agente executar, quando inserido em um ambiente, pode ser denotada por $P(r \mid Ag, Env)$, então a probabilidade de um agente executar com sucesso (ou seja, a probabilidade de atender as tarefas de um ambiente com sucesso) será:

$$P(\psi \mid Ag, Env) = \sum_{r \in R_{\psi}(Ag, Env)} P(r \mid Ag, Env)$$

Onde:

Env representa um ambiente;
Ag um agent;
r representa uma execução de Ag;
$\mathcal{R}$ o conjunto de execuções possíveis; e
ψ a especificação de uma tarefa (predicado).

Existem, basicamente, dois tipos de tarefas em um ambiente:

a)Tarefas de conquista ou de realização (*achievement tasks*); e
b)Tarefas de manutenção (*maintenance tasks*).

As tarefas de conquista definem os estados que devem ser buscados por um agente, enquanto as tarefas de manutenção definem os estados que devem ser evitados por um agente.

Pode-se dizer que as tarefas de conquista levam à mudança do estado das coisas e as tarefas de manutenção preservam o estado das coisas (Wooldridge, 2002).

Uma tarefa de conquista é um subconjunto de estados de um ambiente cujos resultados são classificados como bons (ou como objetivos a serem atingidos pelos agentes). Em outras palavras, um agente é classificado como bem sucedido se levar o ambiente a pelo menos um dos estados de conquista. Já uma tarefa de manutenção é descrita por um subconjunto de estados de um ambiente cujos resultados são classificados como ruins (ou seja, resultados a serem evitados pelos agentes). Os agentes são classificados como bem sucedidos quando não levam o ambiente a um estado de manutenção.

Com base nos conceitos apresentados nesse capítulo, pode-se entender a existência de um agente como uma programação automática cujo objetivo é, inicialmente, gerar um programa que leve a um ambiente de tarefas e, posteriormente, a partir desse ambiente de tarefas, gerar um agente que deverá ser bem sucedido em seu ambiente. Esse ciclo de vida é denominado síntese de um agente (*agent synthesis*) e denotado conforme abaixo (Wooldridge, 2002):

$$syn: \mathcal{T}\varepsilon \rightarrow (\mathcal{AG} \cup \{\perp\})$$

Onde:

$\mathcal{T}\varepsilon$ *é um ambiente de tarefas;*
$\mathcal{AG}$ *é um conjunto de agentes do ambiente; e*
$\perp$ *é o conjunto vazio ou nulo.*

1.3.**Lógica nebulosa**

Muitas informações que observamos em nosso dia a dia são expressas com certo grau de imprecisão. As mudanças de temperatura, por exemplo, podem ser classificadas de forma imprecisa, a partir de observações feitas em um termômetro. Em muitos casos, um mesmo conceito pode ter avaliações diferentes quando observado em diferentes momentos. Nota-se que uma viagem longa em um dia de sol não é como uma viagem longa em uma noite com neblina e chuva. Além disso, a definição exata de quando a distância passa a ser longa é imprecisa. Em outras palavras, essa mudança normalmente não é avaliada de um quilômetro para outro.

A imprecisão está presente em diversas áreas de estudo. A teoria dos conjuntos nebulosos permite a manipulação dessas medidas imprecisas através de uma base matemática

associada a um sistema de raciocínio nebuloso (Cox, 1994). Algumas das aplicações da lógica nebulosa são:

a)Sistemas de controle de guindastes;

b)Sistemas de controle de metrôs;

c)Sistemas de tomada de decisão;

d)Freios ABS; e

e)Sistemas de análise de crédito financeiro.

1.3.1.Teoria dos conjuntos nebulosos

Na teoria dos conjuntos nebulosos, a noção de pertinência de um elemento em relação a um conjunto é um pouco diferente da noção conhecida na teoria clássica de conjuntos. Na teoria clássica, um elemento simplesmente pertence ou não a um conjunto (ou seja, faz parte ou não do conjunto). No caso dos conjuntos nebulosos, a pertinência de seus elementos será definida por uma função, chamada função característica ou de pertinência (Cox, 1994). Formalmente, um conjunto A de um universo U pode ser definido da seguinte maneira:

$$f_A(x): U \rightarrow A$$

Onde:

U é o universo de discurso;
A é um subconjunto do universo de discurso U;
x é um elemento do universo de discurso U; e
f *é a função que define o grau de pertinência dos elementos de U no conjunto A.*

Tal que:

$f_A(x) = 1$ *se e somente se* $x \in A$; *e*
$f_A(x) = 0$ *se e somente se* $x \notin A$.

O gráfico abaixo demonstra o grau de pertinência das medidas de um conjunto de distâncias longas com base na teoria clássica de conjuntos:

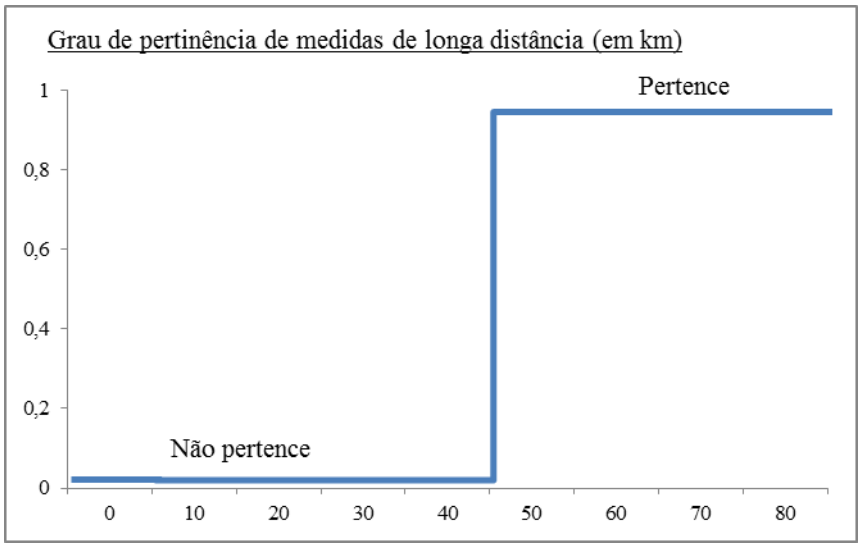


Figura 4: Relação de pertinência segundo a teoria clássica dos conjuntos.

Nota-se que, entre zero e 50, o grau de pertinência é zero e a partir de 50 é um. Com isso, pode-se assumir que uma medida de distância abaixo de 50 km é curta e, acima desse valor, é longa. Na teoria clássica, o grau de pertinência de um elemento está limitado a dois estados mutuamente exclusivos: não pertence (grau zero) ou pertence (grau um).

Na teoria dos conjuntos nebulosos, o grau de pertinência de um elemento é generalizado para assumir qualquer valor entre os graus zero e um. Em outras palavras, a teoria dos conjuntos nebulosos considera que um elemento pode assumir diferentes níveis de pertinência entre os estados não pertence e pertence (Cox, 1994). Esse grau de pertinência é atribuído pela função característica, conforme expresso abaixo:

$$\mu_A(x): U \rightarrow [0,1]$$

Onde:

U é o universo de discurso;
x é um elemento do universo de discurso U; e
μ é a função que define o grau de pertinência de um elemento de U.

Essa nova definição implica em uma grande melhoria na forma como representamos as mudanças de níveis das medidas que fazemos, conforme ilustra o gráfico abaixo:

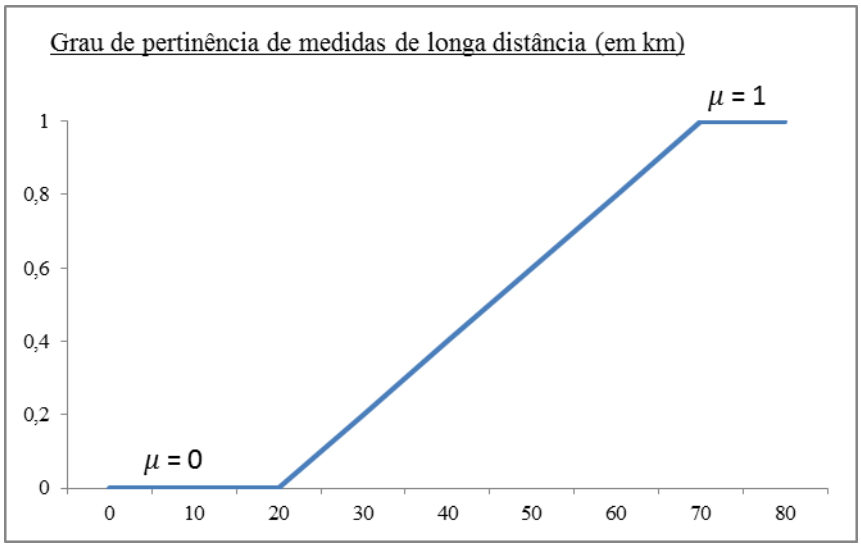


Figura 5: Relação de pertinência segundo a teoria dos conjuntos nebulosos.

Nota-se que, para atribuir o grau de pertinência a um determinado elemento de um conjunto, é preciso considerar o contexto em que o estudo é feito. No gráfico apresentado, as medidas de distâncias começam ser classificadas como longas a partir do quilômetro 20 e são consideradas realmente longas conforme se aproximam do quilômetro 70. Essa análise poderia ser válida ao se estudar a trajetória de um ciclista ou corredor, mas não a de um avião, que, certamente, não seria considerada longa nos primeiros 70 quilômetros.

Portanto, a mudança de contexto é determinante na configuração de um conjunto nebuloso e influencia diretamente na eficiência dos resultados de um modelo, conforme ilustra o gráfico abaixo:

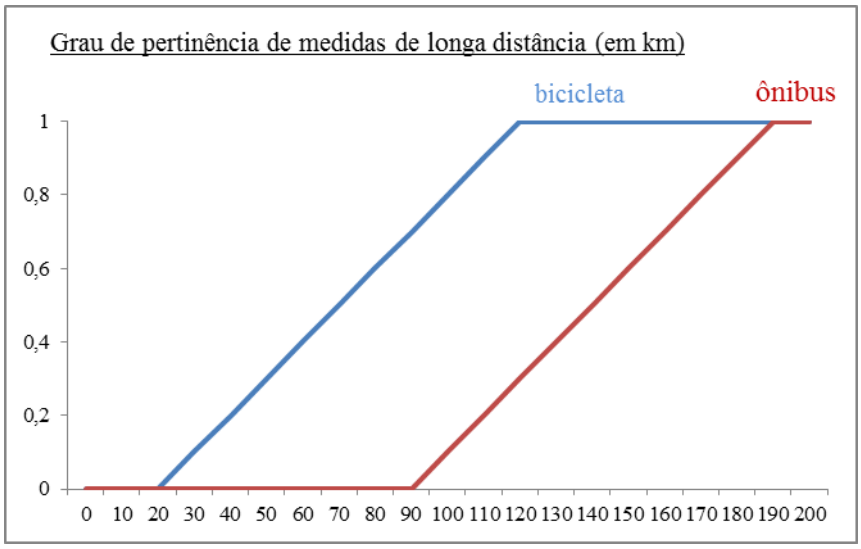


Figura 6: Importância da mudança de contexto ao avaliar conjuntos nebulosos.

Na teoria clássica, os elementos de um conjunto são expressos através de uma listagem simples, conforme demonstrado abaixo:

$$A = \{x_1, x_2, x_3, x_4, \dots, x_n\}$$

Onde:

A é um conjunto; e
x é um elemento de A.

Nesse caso, a operação de união pode ser denotada por:

$$A = \sum_{i=1}^{n} X_i$$

Ou, caso o conjunto represente um intervalo contínuo, poderia ser denotada por:

$$A = \int x \, dx$$

Tomando as definições da teoria clássica de conjuntos, serão apresentadas as definições da teoria de conjuntos nebulosos. Nesse caso, é necessário que a lista de elementos seja expressa com o grau de pertinência de cada elemento, conforme feito a seguir (Cox, 1994):

$$A = \{\mu_1/x_1, \mu_2/x_2, \mu_3/x_3, \mu_4/x_4, \dots, \mu_n/x_n\}$$

Onde:

A é um conjunto nebuloso; e
x é um elemento de A.

Nesse caso, a operação de união deverá ser denotada por:

$$A = \sum_{i=1}^{n} \mu_i(X_i)/X_i$$

E, para o caso de intervalo contínuo, por:

$$A = \int \mu_A(x)/x \, dx$$

Portanto, o conjunto nebuloso pode ser expresso como um conjunto de pares ordenados relacionados por uma função de pertinência, conforme apresentado abaixo:

$$A = \{(x, \mu_A(x)) | \ x \in X\}$$

Onde:

A é um conjunto nebuloso;
μ_A é a função de pertinência; e
X o universo de discurso.

Assim como na teoria clássica, os conjuntos nebulosos podem ser manipulados através de operações de união, interseção e negação (Cox, 1994). A operação de união é denotada da seguinte forma:

$$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x), \forall \ x \in X$$

Onde:

$\vee$ é o operador de máximo, que pode ser aplicado sobre um conjunto ou elemento.

O gráfico que segue ilustra a operação de união entre os conjuntos nebulosos *A* e *B*:

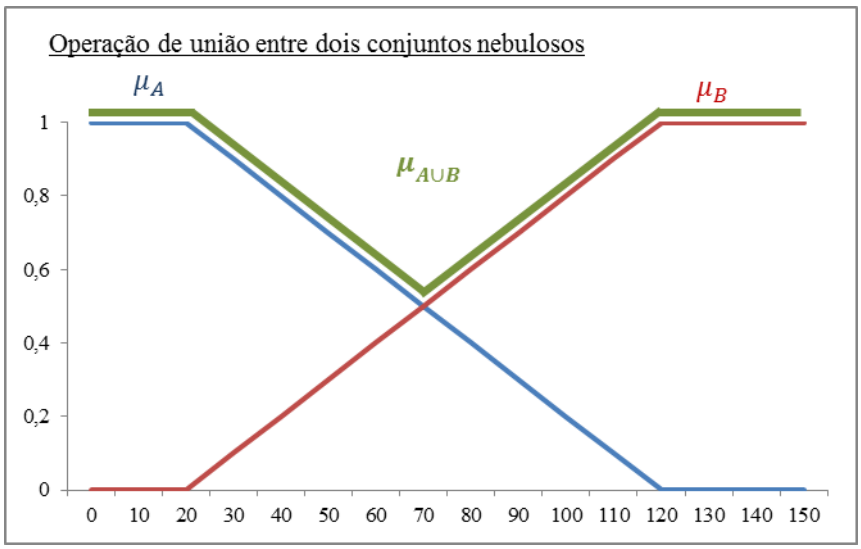


Figura 7: Resultado da operação de união entre conjuntos nebulosos.

A operação de interseção é denotada da seguinte forma:

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x), \forall \ x \in X$$

Onde:

$\wedge$ *é o operador de mínimo, que pode ser aplicado sobre um conjunto ou elemento.*

O gráfico abaixo ilustra a operação de interseção entre os conjuntos nebulosos *A* e *B*:

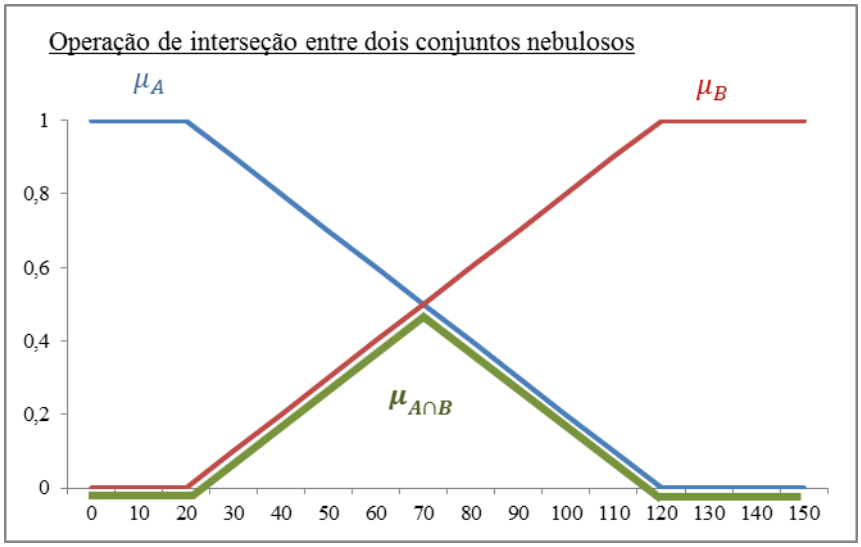


Figura 8: Resultado da operação de interseção entre conjuntos nebulosos.

A operação de complemento (ou negação) é denotada da seguinte forma:

$$\mu_A(x) = 1 - \mu_A(x), \forall\, x \in X$$

O gráfico abaixo ilustra a operação de complemento sobre o conjunto *A*:

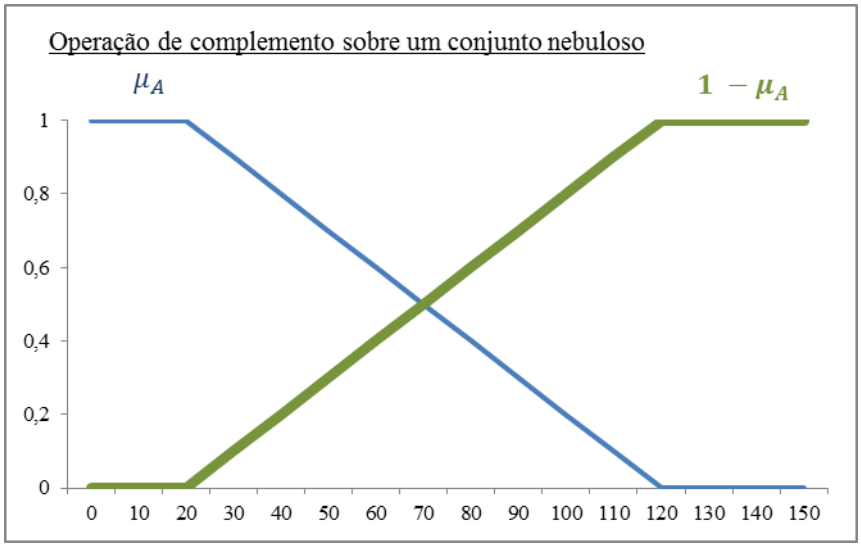


Figura 9: Operação de complemento sobre um conjunto nebuloso.

As inovações trazidas pela teoria dos conjuntos nebulosos acarretaram em algumas consequências sobre as leis das operações com conjuntos, algumas dessas consequências foram (Cox, 1994):

a) A lei da não contradição deixa de ser válida em operações com conjuntos nebulosos; e
b) A lei da exclusão mútua também deixa de ser válida em operações com conjuntos nebulosos.

A invalidade da lei da não contradição pode ser observada pela ilustração que segue. Nota-se no gráfico que a interseção entre os conjuntos nebulosos não resulta no conjunto vazio:

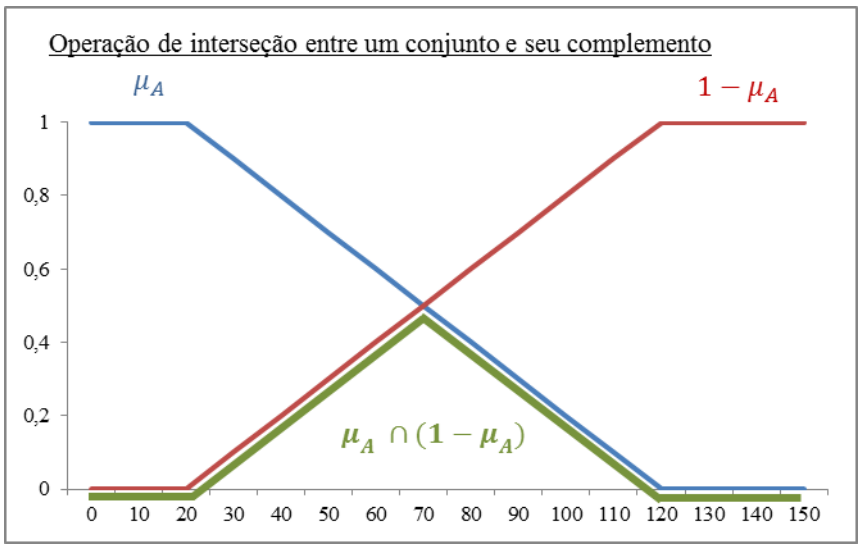


Figura 10: Invalidade da lei da não contradição para conjuntos nebulosos.

$$Teoria\ clássica: A \cap {\sim}A = \emptyset$$

$$Teoria\ nebulosa: A \cap {\sim}A \neq \emptyset$$

A invalidade da lei da exclusão mútua pode ser observada pela ilustração abaixo. Nota-se no gráfico que a união entre os conjuntos nebulosos não resulta no conjunto universo:

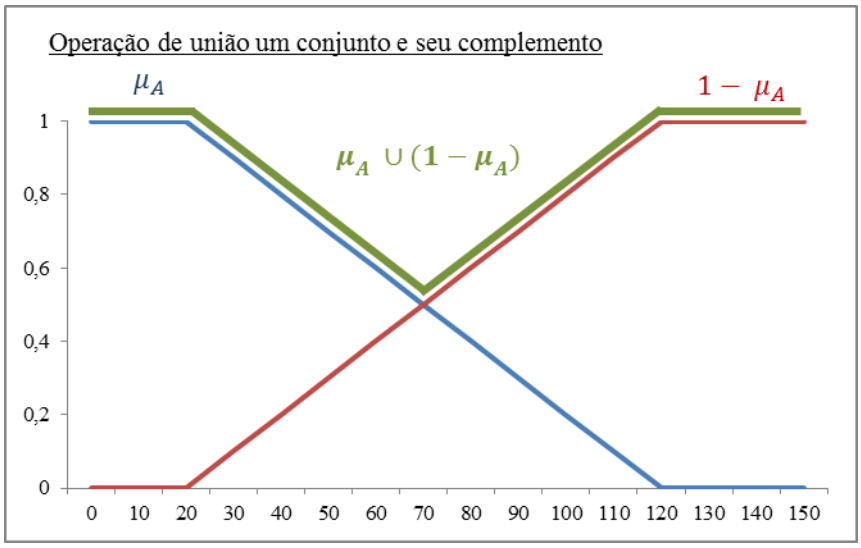


Figura 11: Invalidade da lei da exclusão mútua para conjuntos nebulosos.

$$Teoria\ clássica: A \cup {\sim}A = U$$

$$Teoria\ nebulosa: A \cup {\sim}A \neq U$$

Os conjuntos nebulosos são empregados na representação de variáveis linguísticas. Essas variáveis são utilizadas para descrever as informações de uma aplicação específica. Uma variável linguística pode ser caraterizada conforme denotado abaixo:

$$Variável = (x, T(x), U, G, M)$$

Onde:

x *é o nome da variável linguística;*
$T(x)$ *é o conjunto de termos linguísticos de x relacionados ao universo de discurso;*
U *é o universo de discurso;*
G *é uma regra sintática para gerar os nomes dos valores de x; e*
M *é uma regra semântica para dar um significado para cada valor de x.*

Tomando-se como exemplo de variável linguística a distância de uma rota, deve-se denotar a distância da seguinte forma:

$$x = distância$$

$$T(distância) = \{curta, longa\}$$

$$U = [0, 100]$$

Onde:

Cada termo T é caracterizado por um conjunto nebuloso contido no universo de discurso U.

Pode-se interpretar no exemplo acima que a distância deve assumir duas configurações distintas: curta e longa. Uma distância curta implica na atribuição de medidas entre zero e 50, enquanto uma distância longa implica na atribuição entre 50 e 100. Esses termos são caracterizados por conjuntos nebulosos e podem assumir diferentes níveis de imprecisão, de acordo com suas funções de pertinência.

1.3.2. Sistema de inferência nebuloso

A teoria dos conjuntos nebulosos pode ser aplicada na construção de sistemas de inferência nebulosos (Mendel, 1995). Esses sistemas recebem como entrada um conjunto de informações precisas (determinísticas) que são mapeadas em um conjunto nebuloso pelo seu módulo *fuzzifier*. O conjunto nebuloso gerado é então submetido a uma base de regras pelo módulo de inferência (*inference*) e, por fim, passado ao módulo *defuzzifier*, para que as informações nebulosas resultantes sejam transformadas novamente em informações precisas.

O esquema abaixo ilustra o fluxo de processamento de um sistema de inferência nebuloso:

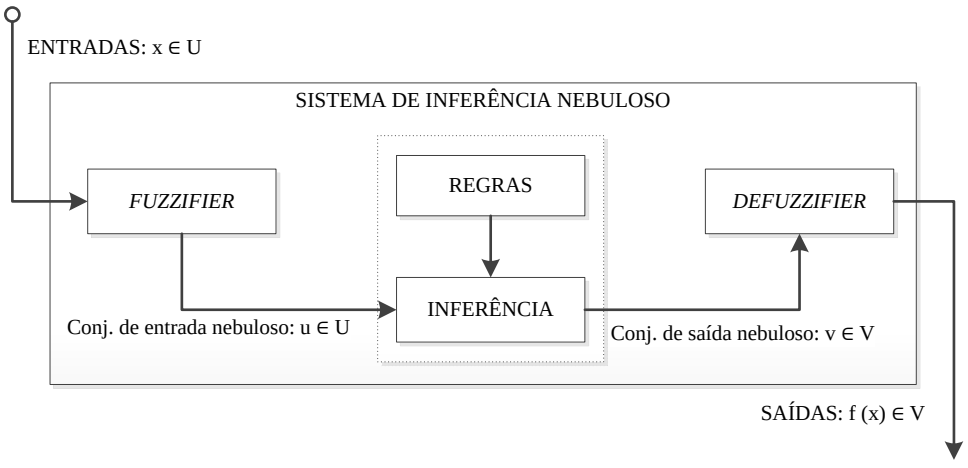


Figura 12: Fluxograma básico de um sistema de inferência nebuloso.

No esquema apresentado, o módulo *fuzzifier* gera um conjunto nebuloso a partir de um universo de discurso *U*. O grau de pertinência de cada elemento de *U* será mapeado por uma função de pertinência no conjunto nebuloso, conforme expresso abaixo:

$$\mu_A(x): U \rightarrow [0,1]$$

Onde:

U é o universo de discurso;
x é um elemento do universo de discurso U; e
μ é a função que define o grau de pertinência de um elemento de U.

O objetivo do módulo *fuzzifier* é computar o grau de pertinência com que as entradas precisas satisfazem os termos linguísticos do sistema. A relação entre um elemento do universo de discurso e seu grau de pertinência pode ser feita por diferentes tipos de função. Alguns formatos de funções de pertinência são (Mendel, 1995):

a)Triangular;
b)Trapezoidal;
c)Gausiana;
d)Sino;
e)Sigmoidal; e
f)Singleton.

O formato triangular é bastante simples e pode ser descrito com apenas três variáveis, conforme expresso abaixo:

$$\mu(x) = \frac{x - SL}{C - SL}, \qquad para\ SL \leq x \leq C;$$

$$\mu(x) = \frac{x - SR}{C - SR}, \qquad para\ C \leq x \leq SR; e$$

$$\mu(x) = 0, \qquad caso\ contrário.$$

Onde:

SL (spread left) é o domínio da função quando a imagem é linear e ascendente;
SR (spread right) é o domínio da função quando a imagem é linear e descendente;
x é um elemento do universo de discurso; e
μ é a função que define o grau de pertinência de um elemento x.

A figura abaixo ilustra um exemplo de uma função de pertinência do tipo triangular:

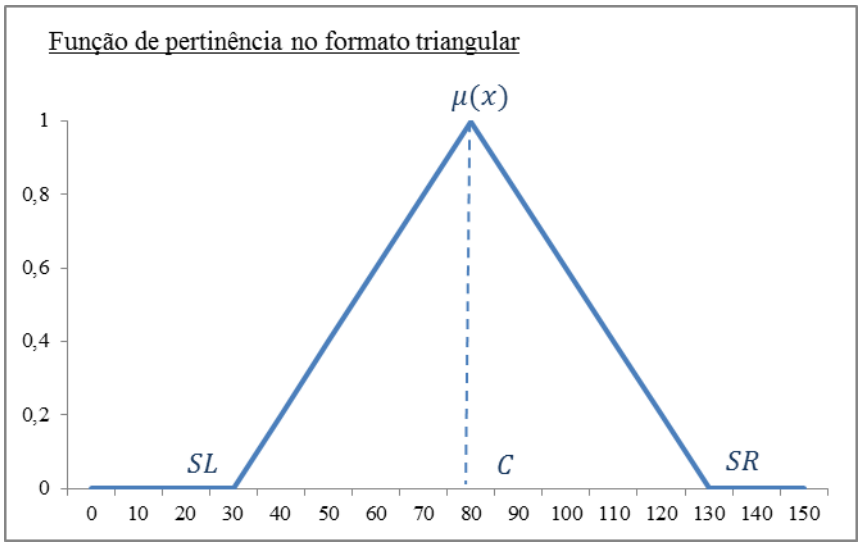


Figura 13: Gráfico apresentando função de pertinência do tipo triangular.

O formato trapezoidal também é bastante simples e pode ser descrito com três variáveis, conforme demonstrado abaixo:

$$\mu(x) = [1 - g(x - v, d) - g(u - x, d)]$$

Onde:

$$g(s, d) = \begin{cases} 1, & se\ s.d > 1 \\ s.d, & se\ 0 \leq s.d \leq 1 \\ 0, & se\ s.d \leq 0 \end{cases}$$

A figura abaixo ilustra um exemplo de uma função de pertinência do tipo trapezoidal:

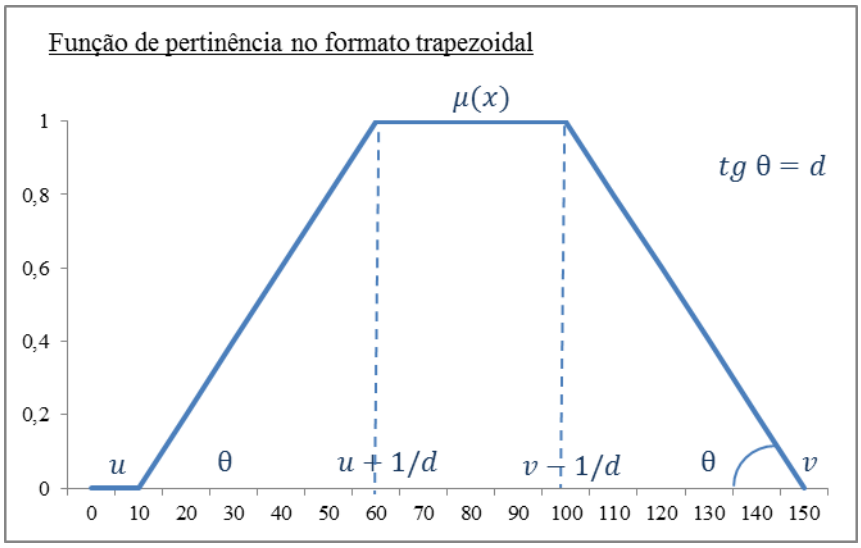


Figura 14: Gráfico apresentando função de pertinência do tipo trapezoidal.

O formato gaussiano é descrito pela expressão abaixo:

$$\mu(x) = e^{-\left(\frac{x-m}{v}\right)^2}$$

Onde:

m é a média dos elementos de domínio da função;
v é o desvio padrão; e
e uma constante.

Quanto mais próximo da média está o padrão, maior é o grau de pertinência. Nesse formato, padrões semelhantes terão pequenas distâncias entre si. A figura que segue ilustra um exemplo de uma função de pertinência do tipo gaussiano:

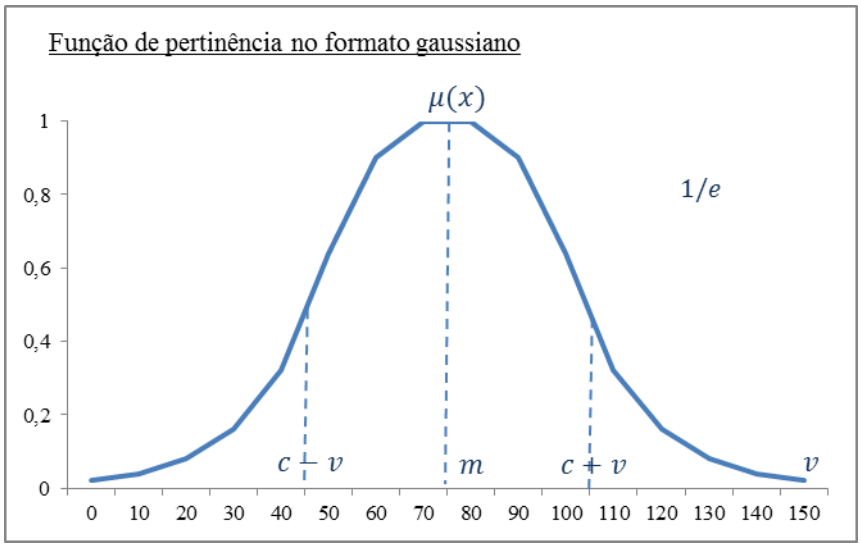


Figura 15: Gráfico apresentando função de pertinência gaussiana.

O formato sino é bastante semelhante ao formato gaussiano. Entretanto, sua utilização é mais fácil pelo fato de não envolver cálculos exponenciais. O formato sino é descrito pela expressão abaixo:

$$\mu(x) = \frac{1}{1 + \left(\frac{x - c}{a}\right)^{2b}}$$

Onde:

A variável c define o centro da função;
a define a largura; e
b define o decaimento.

O formato sigmoidal pode ser obtido a partir de uma ou duas funções sobrepostas (formato monotônico ou formato sino, respectivamente). O formato monotônico é obtido por duas variáveis: uma para definir o ponto de transição e outra para definir o grau de nebulosidade (inclinação no ponto de transição). O formato sigmoidal monotônico é descrito pela expressão abaixo:

$$\mu(x) = \frac{1}{1 + e^{-a(x-b)}}$$

Onde:

a define a inclinação a partir do ponto de transição; e
b define o ponto de transição.

A figura abaixo ilustra um exemplo de uma função de pertinência do tipo sigmoidal:

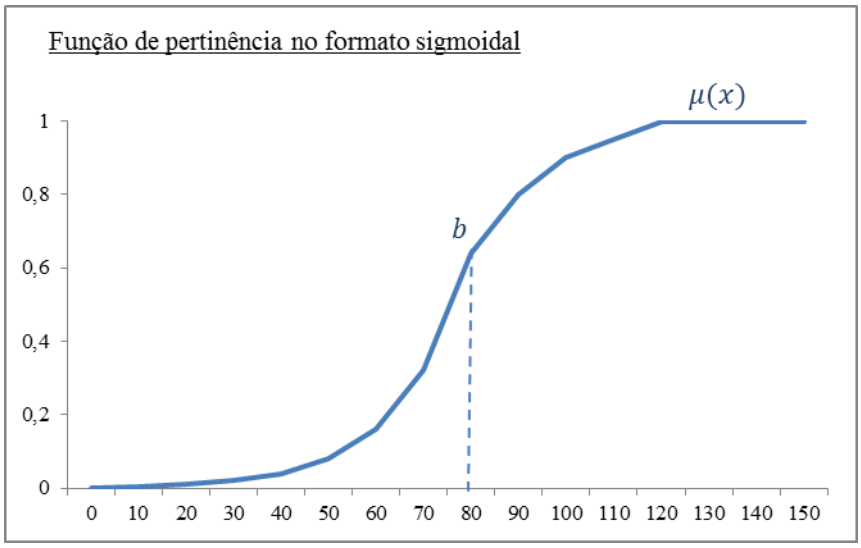


Figura 16: : Gráfico apresentando função de pertinência sigmoidal.

O formato *singleton* refere-se a um tipo de função de pertinência bastante particular. Nele, o grau de pertinência apresenta valor igual a um apenas em um único ponto de todo o domínio e zero nos demais pontos. Segue o gráfico de uma função do tipo *singleton*:

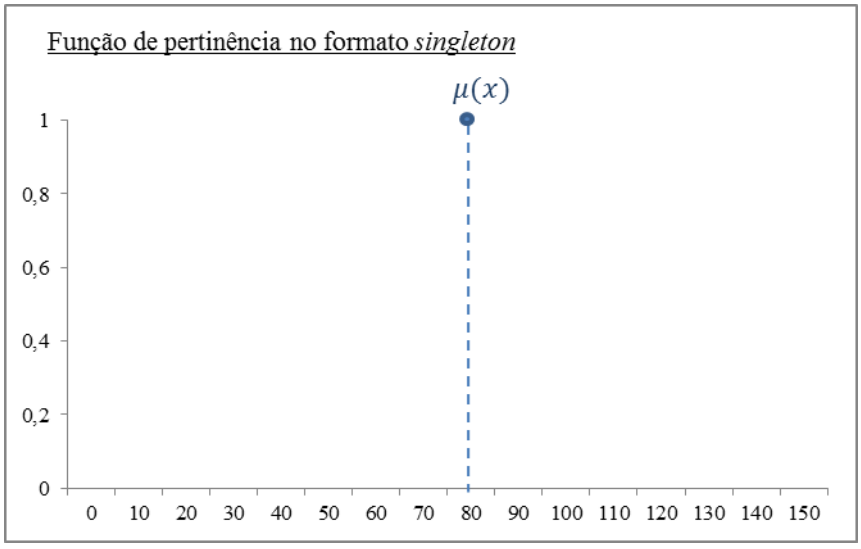


Figura 17: Gráfico apresentando função de pertinência do tipo singleton.

Os conjuntos resultantes das funções de pertinência do módulo *fuzzifier* são passados ao módulo de inferência do sistema nebuloso. Esses conjuntos resultantes são avaliados pelo módulo de inferência a partir da base de regras que compõe o sistema (Mendel, 1995). Essas regras são expressas como implicações lógicas que utilizam as informações geradas pelo módulo *fuzzifier* como termos (antecedente e consequente da regra), conforme ilustrado abaixo:

$$x = A \Rightarrow y = B$$

Onde:

x é a variável linguística do termo antecedente;
A é o conjunto nebuloso de x;
y é a variável linguística do termo consequente; e
B o conjunto nebuloso de y.

A unidade de inferência do sistema nebuloso tem o objetivo de derivar conclusões a partir de um conjunto de regras (Mendel, 1995). Essas regras são, basicamente, implicações compostas por uma ou mais condições. Pode-se dizer que a base de regras do sistema está relacionada ao conhecimento que se tem sobre um domínio de aplicação, enquanto a unidade de inferência determina o modo de se raciocinar e de tomar decisões sobre esse domínio de aplicação.

Uma regra representa um tipo especial de relação entre conjuntos nebulosos através de seus termos antecedente e consequente. Além da relação de implicação, também podem ser empregadas operações de conjunção, disjunção e negação entre termos de uma regra. Em outras palavras, os termos de uma regra podem ser conectados por operadores lógicos. Segue um exemplo onde o termo antecedente da regra é formado por uma conjunção:

$$x = A \wedge y = B \Rightarrow z = C$$

Onde:

x é a variável linguística do termo antecedente;
A é o conjunto nebuloso de x;
y é a variável linguística do termo antecedente;
B o conjunto nebuloso de y;
z é a variável linguística do termo consequente; e
C o conjunto nebuloso de z.

A lógica proposicional tem operações equivalentes às operações da teoria dos conjuntos. Portanto, é possível conceituar uma regra de um sistema nebuloso como um processo de combinar proposições (Cox, 1994). As proposições relacionam as variáveis de um modelo nebuloso com seus conjuntos. Em outras palavras, combinar proposições é equivalente a executar operações com conjuntos. O isomorfismo entre a teoria dos conjuntos e a lógica proposicional garante que essas teorias têm teoremas equivalentes. Seguem algumas operações equivalentes entre duas teorias:

$$p \wedge q \Leftrightarrow A \cap B$$

$$p \vee q \Leftrightarrow A \cup B$$

$$\sim p \Leftrightarrow \bar{A}$$

Uma implicação na lógica proposicional é verdadeira quando o termo consequente é verdadeiro sempre que o antecedente também é verdadeiro. Chegar a uma conclusão falsa a partir de uma proposição verdadeira é uma contradição. Já uma conjunção é verdadeira somente quando suas proposições são todas verdadeiras. Por fim, uma negação inverte o valor lógico de uma proposição. Com essas definições, podem ser apresentadas duas importantes regras de inferência (Filho, 2002):

a)*Modus Ponens*; e
b)*Modus Tollens*.

A regra de inferência *Modus Ponens* afirma que: se existe uma proposição verdadeira *p* e se essa proposição *p* implica em outra proposição *q*, podemos deduzir que resultado de *q* é verdadeiro. Essa regra é denotada da seguinte forma (Filho, 2002):

$$\frac{\begin{array}{c} p \\ p \rightarrow q \end{array}}{q}$$

De forma equivalente:

$$\frac{\begin{array}{c} x = A \\ x = A \Rightarrow y = B \end{array}}{y = B}$$

A regra de inferência *Modus Tollens* afirma que: se existe uma proposição verdadeira *q* e se outra proposição *p* implica em uma negação de *q*, podemos deduzir que *p* é falsa. Essa regra é denotada da seguinte forma (Filho, 2002):

$$\frac{\begin{array}{c} \sim q \\ p \rightarrow q \end{array}}{\sim p}$$

De forma equivalente:

$$\frac{\begin{array}{c} y \neq B \\ x = A \Rightarrow y = B \end{array}}{x \neq A}$$

Essas regras de inferência podem ser generalizadas para assumir não apenas dois estados (verdadeiro ou falso), mas qualquer valor entre zero e um (grau de veracidade ou de pertinência) (Cox, 1994). Na lógica proposicional uma regra de inferência *Modus Ponens* será válida se sua primeira premissa for exatamente igual ao antecedente de sua segunda premissa, sendo o resultado igual ao consequente da segunda premissa. Na lógica nebulosa essa mesma regra é generalizada e será válida se sua primeira premissa tiver um grau de pertinência similar ao grau do antecedente de sua segunda premissa, sendo o resultado o grau de pertinência do consequente da segunda premissa.

Segue uma ilustração com a aplicação de uma regra de inferência sobre três conjuntos nebulosos *A*, *B* e *C*:

Tomando a regra:

$$Se\ \mu(x_A) \in A\ e\ \mu(x_B) \in B\ então\ \mu(x_c) \in C$$

Portanto:

$$\mu(x_A) \in A\ \wedge\ \mu(x_B) \in B \rightarrow\ \mu(x_c) \in C$$

Nota-se que:

$$A \cap B \Rightarrow C$$

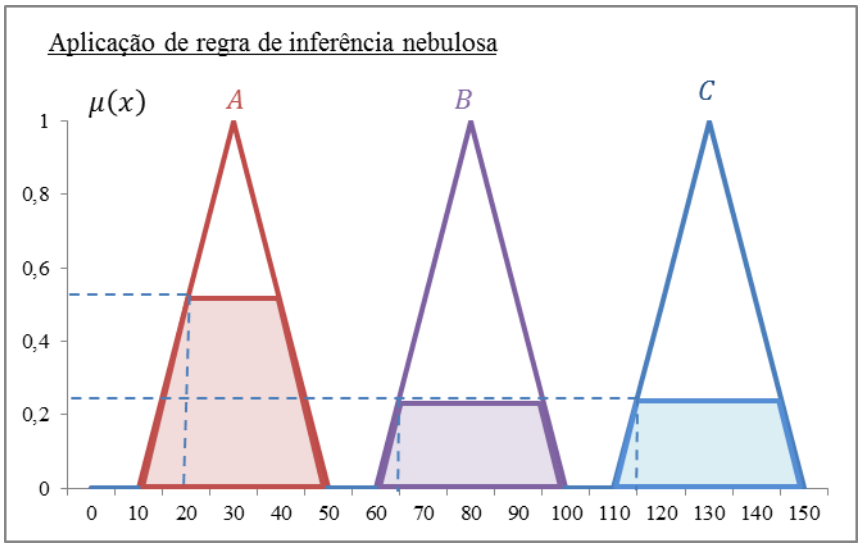


Figura 18: Aplicação de uma regra de inferência sobre três conjuntos nebulosos.

Como exemplo, seja *A* é o conjunto medidas de rotas de curta distância, *B* o conjunto de medidas de esforço médio e *C* o conjunto de medidas para rotas classificadas como difíceis (ou de alto custo). Nesse caso, x_A é a variável linguística distância, x_B a variável linguística esforço e x_C a variável linguística custo. Pela regra de inferência ilustrada anteriormente, pode-se concluir que: se a distância é curta e a rota exige um esforço mediano para ser realizada, então o custo final para concluir o percurso será alto.

Em outras palavras, o grau de pertinência de uma regra de um sistema de inferência nebuloso é o resultado da seguinte função:

$$\mu_{A \to B}(x_A, x_B) = \mu_C(x_C)$$

Onde:

x_A é a variável linguística do termo antecedente;
A é o conjunto nebuloso de x_A;
x_B é a variável linguística do termo antecedente;
B o conjunto nebuloso de x_B; e
x_C é a variável linguística do termo consequente;
C o conjunto nebuloso de x_C.

Para se chegar ao final do processamento nebuloso serão aplicadas diversas regras e diferentes resultados de inferência precisarão ser consolidados. A partir das decisões tomadas no módulo de inferência o módulo *defuzzifier* deve realizar a restauração do conjunto de informações precisas (ou determinísticas) para a saída do sistema, consolidando diversos

conjuntos resultantes das aplicações de regras ativadas pelo módulo de inferência (Mendel, 1995). Portanto, feitas as avaliações das proposições da base de regras e gerados os conjuntos nebulosos resultantes é necessário determinar o valor real de saída do sistema.

Executar o módulo *defuzzifier* significa aplicar uma função com o efeito inverso da função de pertinência sobre os conjuntos nebulosos resultantes do processo de inferência. Alguns dos métodos que podem ser utilizados pelo módulo *defuzzifier* de um sistema nebuloso são (Mendel, 1995):

a)Centro de Gravidade (COG) ou Centro de Área (COA);
b)Média Ponderada (MP); e
c)*Center of Sums*.

O método de cálculo do centro de gravidade é um dos mais utilizados. Como o próprio nome sugere o objetivo desse método é calcular o centro de gravidade da área formada pela composição das regras de inferência do sistema nebuloso, conforme demonstra a ilustração abaixo:

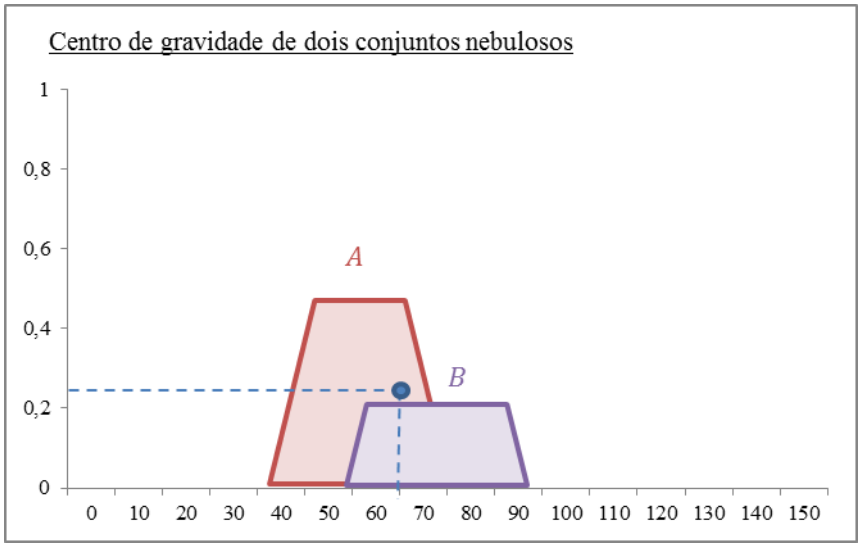


Figura 19: Cálculo do centro de gravidade consolidando dois conjuntos nebulosos.

Segue a fórmula matemática utilizada no cálculo do centro de gravidade:

$$Z = \frac{\sum_{i=0}^{m} \mu_c(z_i) * Z_i}{\sum_{i=0}^{m} \mu_c(z_i)}$$

Onde:

m é o número de intervalos de quantização da saída;
Z_i *é a saída para o intervalo de quantização i; e*
$\mu_c(z_i)$ *o grau de pertinência para Z.*

O método de média ponderada é mais indicado quando o consequente de uma regra é obtido através de uma função de pertinência do tipo *singleton*. Esse método combina os consequentes de uma regra de inferência (conjuntos resultantes) através da fórmula matemática denotada abaixo:

$$Z = \frac{\sum_{i=1}^{n} \mu_i . Z_i}{\sum_{i=1}^{n} \mu_i}$$

Onde:

n é o número de regras ativadas pelo módulo de inferência;
Z_i *é o valor do singleton i; e*
μ_i *é o nível de disparo (firing strengh) da regra i.*

O método *Center of Sums* é uma simplificação do método de cálculo do centro de gravidade. A saída do *Center of Sums* é dada pela equação abaixo:

$$Z = \frac{\sum_{i=1}^{n} C^u(B_i^{\alpha_i})}{\sum_{i=1}^{n} Área(B_i^{\alpha_i})}$$

Assumindo que:

$$C^u(B_i^{\alpha_i}) = Centróido(B_i^{\alpha_i}) . Área(B_i^{\alpha_i})$$

Onde:

B_i *refere-se aos termos linguísticos dos consequentes;*
α_i *é o nível de disparo da regra i; e*
$B_i^{\alpha_i}$ *é o conjunto nebuloso obtido pela aplicação de* α_i *em* B_i.

Conforme demonstra o gráfico abaixo:

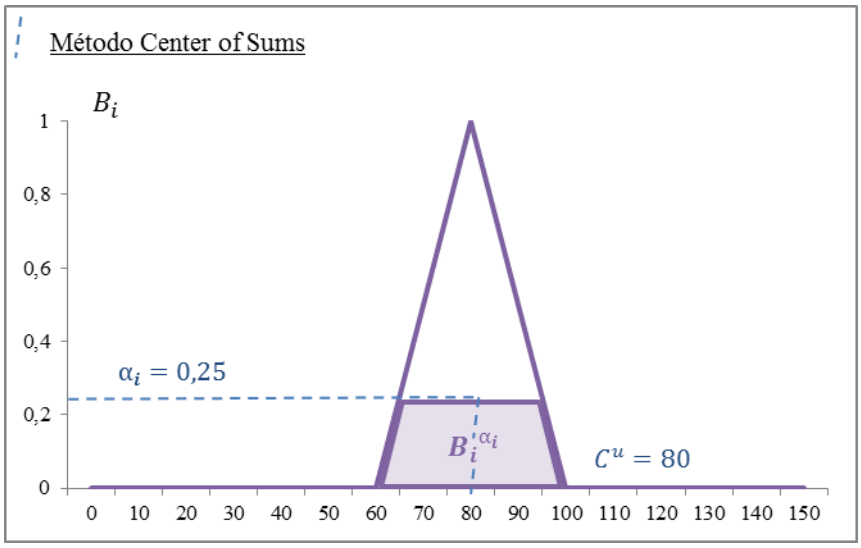


Figura 20: Gráfico exibindo variáveis do método center of sums.

Com os resultados gerados pelo módulo *defuzzifier*, um sistema de inferência nebuloso é capaz de apresentar em sua saída um valor determinístico para as inferências feitas sobre os conjuntos nebulosos. Com isso, o sistema de inferência nebuloso conclui seu processamento: após receber valores determinísticos e inferir sobre classificações imprecisas desses valores, computa um valor determinístico que consolida os resultados das inferências realizadas.

1.4.**Algoritmos para o planejamento de rotas**

O planejamento de trajetórias está relacionado à solução de diversos problemas espaciais que envolvem tarefas como a navegação e a escolha de caminhos. Esse planejamento depende do conceito de mapeamento. O mapeamento consiste em modelar o ambiente em estudo através de cartas e mapas. O planejamento viabiliza a escolha de caminhos sobre mapas e pode ser feito com ou sem a interferência humana.

Existem diversos tipos de planejamento de trajetória, sendo a maioria deles baseados em três tipos fundamentais de algoritmos (Latombe, 1991):

a)Algoritmo de decomposição em células;
b)*Roadmaps*; e
c)Campo potencial.

Um agente autônomo é capaz de realizar uma ou mais rotas sobre um mapa, desviando de possíveis obstáculos através da leitura de um planejamento de trajetória. Entretanto, esse

planejamento não é um problema bem delineado e simples de ser resolvido, mas um conjunto de problemas a serem resolvidos (Latombe, 1991).

O planejamento em navegação robótica, por exemplo, pode ser definido como a busca de um percurso a ser seguido, ou a sequência de ações a serem tomadas para que o robô possa sair de um ponto de partida e chegar a um ponto de destino sem colisão com obstáculos conhecidos (Gero, 1988).

O planejamento de um caminho tem duas vertentes principais: a global, relacionada ao mapeamento do ambiente onde se pretende traçar um percurso, e a local, relacionada à navegação curta (Firby, 1993). O planejamento global trata-se de um modelo simplificado, estático e pré-estabelecido (Crowley, 1985). Já o planejamento local é baseado nos valores coletados pelos sensores de um robô, por exemplo. Nesse trabalho será abordada a vertente global apenas. A seguir serão descritos os principais métodos de planejamento global de trajetória.

1.4.1.Roadmaps

A ideia essencial desse método é criar uma trajetória em linha reta da posição inicial à posição final. Essa trajetória em linha reta nem sempre é possível e, como alternativa, uma série de trajetórias retilíneas que tangenciam os obstáculos, chamadas *roadmaps*, são criadas para ligar os pontos de origem e destino. O planejamento é então reduzido à conexão entres essas *roadmaps*, a posição inicial e a posição final.

A ideia dessa abordagem consiste em reduzir as informações espaciais a um grafo representando os possíveis caminhos sobre um terreno qualquer (Ottoni, 2000). Alguns métodos baseados nessa abordagem são: os grafos de visibilidade e os diagramas de Voronoi (Voronoi, 2012).

Um grafo de visibilidade é obtido a partir de segmentos de reta entre os pares de vértices dos obstáculos. Todos os segmentos de reta que estiverem inteiramente na região do espaço livre são adicionados ao grafo. No planejamento de trajetória, as posições de origem e destino são representadas como vértices, gerando assim um grafo de conectividade onde um algoritmo de busca pode ser utilizado para encontrar um caminho livre.

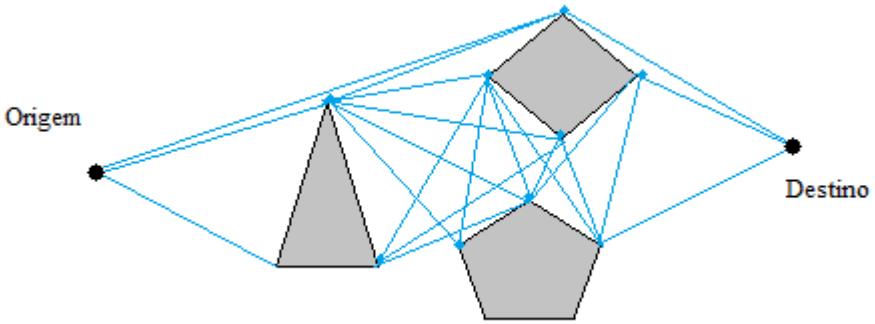


Figura 21: Grafo de visibilidade empregado para contornar trêsobstáculos.

Um grafo de visibilidade é formado pela ligação entre as posições (pontos) de um ambiente, de acordo com a visibilidade de uma posição à outra (Davidson, Watson, & Selman, 1993). No caso da navegação bidimensional, com diversos obstáculos poligonais, o grafo de visibilidade baseia-se na visibilidade mútua dos vértices dos polígonos contidos no ambiente em questão. Um grafo de visibilidade contém os menores trajetos existentes entre dois pontos do ambiente.

Como se pretende utilizar um grafo de visibilidade para gerar *roadmaps* a partir da conexão em linha reta dos vértices de obstáculos visíveis, cada nó no grafo representa um vértice de um obstáculo e cada aresta representa uma conexão entre dois nós. Essa conexão deve ser estabelecida pela seguinte regra de construção (Davidson, Watson, & Selman, 1993): Supondo que em um grafo *G* qualquer existam dois vértices: *X* e *Y*. Existirá uma aresta entre *X* e *Y*, se e somente se, for possível traçar uma linha reta que conecte esses dois vértice sem interceptar um obstáculos de um terreno.

O método do diagrama de Voronoi define um subconjunto do espaço de configuração composto por todas as posições livres que estão a uma distância mínima de qualquer obstáculo. Em outras palavras, dado um conjunto *S* de *n* pontos no plano, deseja-se determinar para cada ponto *p* de *S* qual é a região *V(p)* dos pontos do plano que estão mais próximos de *p* do que de qualquer outro ponto em *S*. As regiões determinadas por cada ponto formam uma partição do plano chamada de Diagrama de Voronoi (Voronoi, 2012).

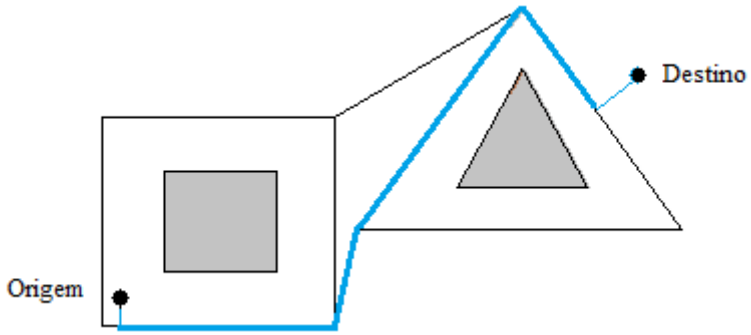


Figura 22: Diagrama de Voronoi gerado a partir de dois polígonos (obstáculos).

No planejamento de caminhos sobre uma região contendo obstáculos, cada obstáculo pode ser representado por polígonos côncavos ou convexos. Nesse caso, para encontrar o diagrama generalizado de Voronoi para essa coleção de polígonos, pode-se calcular o diagrama através de uma aproximação, convertendo os obstáculos em uma série de pontos. Feito isso, o próximo passo é calcular o diagrama para essa coleção de pontos. Em seguida, os segmentos do diagrama que interceptam algum obstáculo são eliminados.

1.4.2.Decomposição Celular

No método de decomposição celular o espaço geográfico é dividido com o objetivo de particionar um planejamento de trajetória em planos triviais. A partir dessa divisão do espaço a meta de planejamento se reduz a percorrer um grafo de células adjacentes. Em outras palavras, o método decompõe o espaço livre em regiões simples chamadas células, de forma que um caminho entre dois pontos em células diferentes seja facilmente gerado.

Segue uma ilustração com a divisão do espaço em células através do método de decomposição celular:

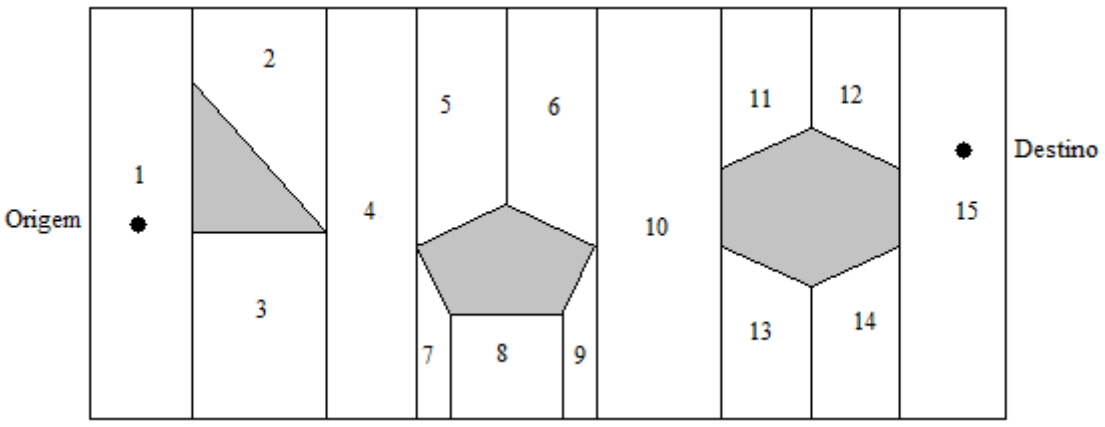


Figura 23: Divisão do espaço através do método de decomposição celular.

Um grafo não direcionado representando a relação de adjacência entre as células é então construído e sobre esse grafo a busca do caminho é realizada. Esse grafo é chamado de grafo de conectividade (Ottoni, 2000). Os nós desse grafo são as células extraídas do espaço livre. Dois nós estão conectados por uma aresta se e somente se as duas células correspondentes são adjacentes. O resultado da busca é uma sequência de células chamada de canal. Segue uma ilustração de um grafo de conectividade:

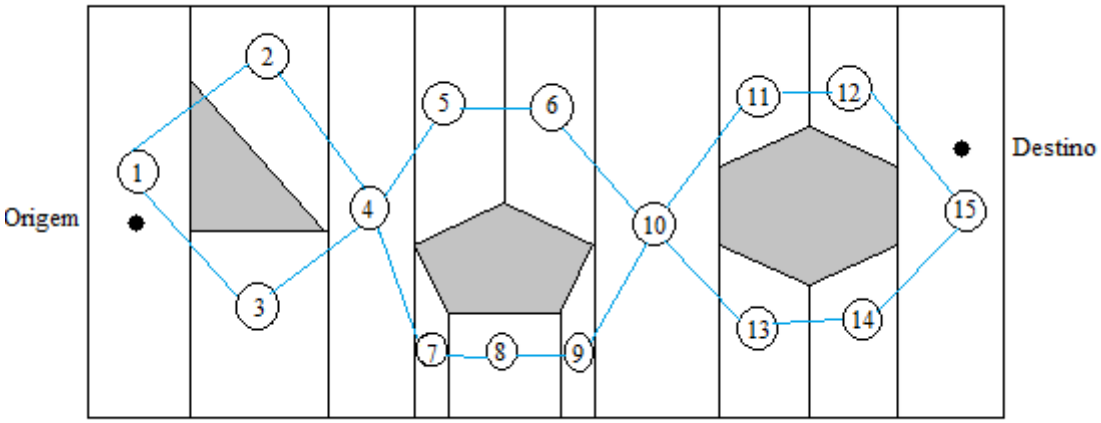


Figura 24: Canais gerados sobre um grafo de conectividade.

Portanto, um caminho livre pode ser computado a partir da sequência de células adjacentes do grafo de conectividade. Seguem as principais características de um grafo de conectividade:

a)Cada nó representa uma célula criada na divisão do terreno; e
b)Cada aresta representa a conexão entre duas células adjacentes.

A complexidade desse método de planejamento de trajetória está na divisão do espaço livre do terreno. Em um grafo de conectividade podem existir diversas trajetórias entre a posição inicial e a final. Cada uma dessas opções de trajetória representa um canal. O método de decomposição celular é subdividido em duas categorias:

a) Métodos exatos; e

b) Métodos aproximados.

A decomposição exata divide o espaço livre em células de forma que a união dessas células corresponde exatamente ao espaço disponível no universo de configuração. Em outras palavras, decompõe o espaço livre em células cuja união é exatamente o próprio espaço livre. A fronteira entre duas células adjacentes é um local crítico para o planejamento de trajetória já que podem ocorrer mudanças bruscas nas restrições aplicadas ao movimento de um agente.

No método da decomposição celular aproximada são produzidas células de formas pré-definidas cuja união está estritamente inclusa no espaço livre. Normalmente, os grafos gerados não são completos (ao contrário do método de decomposição exata), pois dependendo da precisão utilizada podem não contemplar todos os caminho possíveis entre duas configurações. Como consequência tem-se que em geral não é possível modelar o ambiente de forma mais real, sendo necessárias algumas aproximações, embora a precisão desses métodos possa ser ajustada arbitrariamente a custo de espaço de armazenamento e tempo de processamento.

De qualquer forma, métodos de decomposição aproximada são mais simples e, por isso, utilizados na prática com maior frequência. Além disso, geralmente dão origem a sistemas mais simples de planejamento de trajetórias do que os métodos de decomposição exata (Latombe, 1991).

1.4.3.Campo Potencial

O método do campo potencial consiste em calcular a direção e o sentido da trajetória de acordo com a direção e o sentido da resultante das forças que se aplicam ao agente autônomo em cada instante da navegação. A resultante das forças é calculada de acordo com os seguintes critérios: os obstáculos, de qualquer tipo, serão considerados como forças repulsivas e a posição final desejada será considerada como uma força atrativa.

O desempenho é uma vantagem desse método sobre os outros. Devido a pouca complexidade dos cálculos exigidos para a elaboração do planejamento de trajetória, esse método é recomendado em espaços de configuração dinâmica, visto que a alteração da configuração do espaço deve ser analisada da forma mais rápida possível para que eventuais choques com obstáculos dinâmicos sejam evitados.

Apesar dos bons resultados fornecidos pelos métodos de campos potenciais eles possuem uma série de problemas, tais como: situações de armadilha com mínimos locais (o

que implica no comportamento cíclico do agente) e dificuldades em terrenos com obstáculos muito próximos ou em corredores estreitos. Como existe a possibilidade da resultante movimentar o agente em direção a uma determinada posição na qual ele não é capaz de contornar, a trajetória planejada deverá ser considerada inválida em uma situação desse tipo.

1.5.**Comentários**

A partir do conteúdo apresentado nesse capítulo pode-se elaborar uma proposta para solucionar o problema de seleção e classificação de rotas em espaços geográficos com obstáculos. Os algoritmos de planejamento de rotas irão compor com os conceitos de sistemas baseados em agentes a unidade de exploração de rotas desse trabalho, enquanto o sistema de inferência nebuloso será utilizado para classificar as rotas geradas pela unidade de exploração.

A principal utilidade dos algoritmos de planejamento é direcionar os conceitos da teoria dos sistemas baseados em agentes para uma aplicação específica: a navegação. Combinando essas duas disciplinas será possível projetar agentes que de fato realizem suas trajetórias de forma consciente, buscando de maneira autônoma um objetivo bem definido por seu plano de rota. Além disso, como foi apresentado mais de um algoritmo de planejamento, torna-se possível estabelecer comparações entre eles, constatando vantagens ou desvantagens mútuas sobre diferentes características geográficas.

Com relação à lógica nebulosa, trata-se de uma teoria comprovadamente eficiente para a solução de problemas envolvendo tomada de decisão e classificação de dados. Portanto, pode-se assumir que o sistema de inferência nebuloso é uma ferramenta adequada para, integrada a um sistema baseado em agentes, selecionar rotas em um mapa digital. A base teórica da lógica nebulosa permitirá a criação de uma unidade lógica para seleção de rotas.

2. **ABORDAGEM PROPOSTA**

Nesse capítulo será apresentada a solução conceitual para o problema de seleção de rotas em espaços com obstáculos. Serão descritos detalhes sobre como os fundamentos teóricos apresentados no capítulo anterior foram empregados e combinados no processo proposto para gerar uma classificação de rotas.

2.1.**Apresentação**

Entre dois pontos de uma cidade pode existir uma quantidade razoável de ruas e avenidas que os interligam. Assim como, entre dois aeroportos pode existir uma infinidade de rotas aéreas. Entretanto, nota-se que nessas situações apenas uma pequena parte de todas as combinações possíveis é utilizada diariamente. Mesmo no caso do tráfego aéreo o espaço é segmentado de acordo com regras que buscam o melhor aproveitamento de recursos.

Embora o tráfego aéreo apresente características completamente distintas do tráfego sobre avenidas de uma cidade, mesmo em situações tão distintas, é possível identificar similaridades. Ao analisarmos as características mais simples e elementares dessas duas situações identificamos informações válidas em quaisquer estudos interligando dois pontos em um espaço plano. As características elementares selecionadas para esse trabalho foram:

a)Pontos de origem e destino;
b)Distância entre dos pontos; e
c)Segmentos com diferentes níveis de esforço sob trajetórias.

Generalizando os casos do tráfego aéreo e do tráfego urbano a partir da perspectiva apresentada, pode-se assumir que, em ambas as situações, os cenários de estudo terão aspectos e informações que poderão ser analisados sem qualquer noção do veículo que fará o trajeto. Todas as informações relevantes para tomada de decisão podem ser expressas em um modelo abstrato aqui denominado mapa de esforço.

Esse mapa de esforço pode ser gerado tomando como exemplo o mapa de uma cidade, conforme o mapa ilustrado a seguir:

Figura 25: Mapa com vias urbanas.

No mapa apresentado é possível observar uma grande quantidade de dados, tais como: nomes de ruas, sentido do tráfego, áreas alagadas etc. Entretanto, para escolher uma rota podemos simplificar as informações e considerar apenas a origem, o destino e os segmentos possíveis. Assumindo apenas dois níveis de esforço para os segmentos do mapa (branco como viável e cinza com intransponível), um mapa de esforço para o tráfego de automóveis poderia ser gerado conforme a ilustração abaixo:

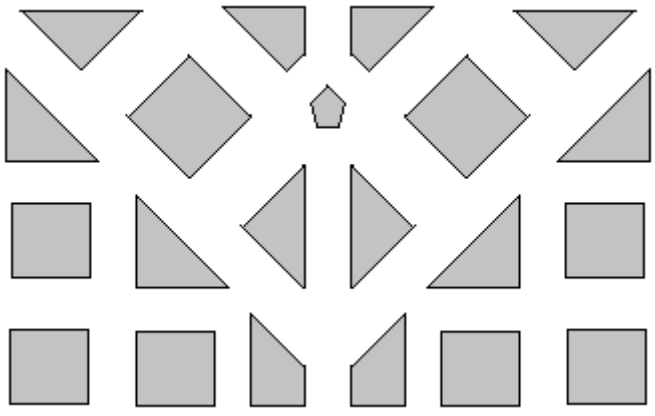


Figura 26: Representação de um mapa de esforço.

Nesse esquema abstrato apenas duas informações básicas são apresentadas para traçar uma rota: onde é possível trafegar (área branca representando vias urbanas) e onde não é (área cinza). Nesse caso, um segmento intransponível poderia ser entendido também como um

obstáculo. Diferentes níveis de esforço poderiam ser representados pelos segmentos do mapa, gerando rotas preferenciais. De qualquer forma, as informações necessárias estariam representadas no esquema e poderiam servir à tomada de decisão em um próximo passo do processo. A figura abaixo ilustra a possibilidade de três níveis de segmentos (o branco como caminho fácil, um cinza mais claro como caminho difícil e outro cinza mais escuro como caminho impossível):

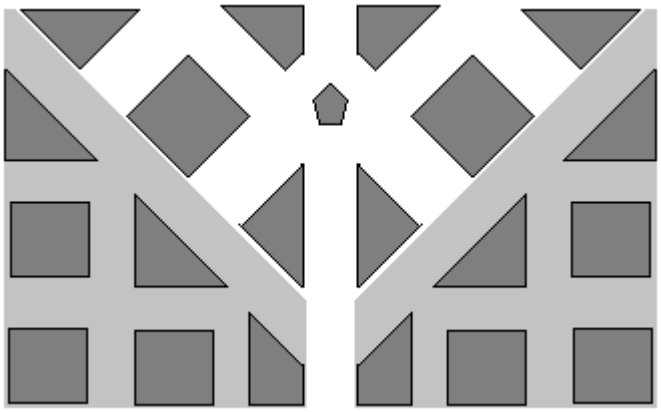


Figura 27: Mapa de esforço com três níveis de esforço.

A mesma abstração utilizada para delimitar rotas para automóveis se aplicaria para delimitar trajetórias possíveis para um avião ou um helicóptero. Evidentemente, os segmentos seriam completamente diferentes porque outros critérios de mapeamento seriam utilizados. Entretanto, a partir do mapeamento o procedimento de exploração de rotas seria o mesmo. Em outras palavras, através do modelo abstrato as informações para explorar rotas seriam resumidas a: origem, destino e segmentos com diferentes níveis de esforço.

Segue uma ilustração de um mapeamento para rotas aéreas com três níveis de esforço (branco fácil, cinza claro difícil e cinza escuro intransponível):

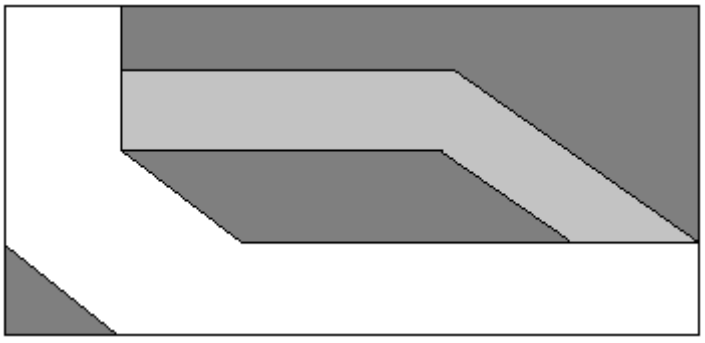


Figura 28: Mapa de esforço para rotas aéreas.

Essa abstração isola, para qualquer aplicação, a delimitação das áreas disponíveis para percurso (traçado) de rotas no nível da cartografia digital. Em outras palavras, a partir do mapeamento dos segmentos disponíveis e de seus níveis o problema para traçar e selecionar rotas passa a ser o mesmo, seja qual for a aplicação (tráfego aéreo, construção de estrada ou de linhas de transmissão).

A partir da distribuição de esforço sobre um mapa, passa a ser importante definir um processo para explorar rotas sobre os segmentos de transposição possível (ou trafegáveis) evitando os segmentos intransponíveis. Quanto maior a diversidade de caminhos explorados sobre os segmentos com níveis de esforço baixo maior a possibilidade de encontrar o caminho de menor esforço.

Além do esforço, também deve ser contabilizada a distância dos trajetos explorados, para que, após o levantamento da maior quantidade possível de rotas viáveis, possa ser gerada uma classificação das melhores rotas, com base no custo de cada uma. Esse custo é uma variável obtida a partir da relação entre a distância percorrida e o esforço realizado sobre uma rota.

Portanto, assumindo que para qualquer exploração de rotas sempre haverá pontos de origem e de destino e que a área contendo tais pontos poderá ser segmentada em diversos níveis de esforço, independente do problema abordado, propõe-se nesse trabalho um método para concretizar e testar a utilidade dessas ideias.

Assim sendo, a partir das informações armazenadas em um mapa digital pretende-se explorar uma grande diversidade de rotas e gerar uma classificação com o objetivo de apontar a rota de menor custo (melhor relação entre o esforço e a distância). Nesse capítulo será explicado como os conceitos apresentados até aqui foram utilizados para abordar o problema de seleção de rotas.

2.2.**Ambiente**

2.2.1.Mapas digitais

Um mapa digital pode ser representado por um arquivo no formato *shapefile* e armazenar informações sobre pontos, linhas e polígonos. Trata-se de uma especificação aberta desenvolvida para interoperabilidade de dados entre sistemas de informações geográficas (ESRI, 2012). Um arquivo desse tipo contém, além de dados espaciais em representação

vetorial, dados com informações temáticas que podem ser utilizadas para definir o nível de esforço de uma área de um mapa.

Um *shapefile* pode ser editado através de um aplicativo como o ArcView GIS 3.2 (ou qualquer outro disponível no mercado). Através desse aplicativo um usuário pode alterar os atributos temáticos ou criar novos atributos para uma camada de um mapa. Por exemplo, seja um mapa contendo os bairros de uma cidade, onde cada bairro é um polígono com um nome, uma área e um perímetro, através do ArcView seria possível alterar o nome de um bairro ou criar um novo atributo população para os bairros do mapa. Dessa forma, uma consulta sobre os bairros da cidade representada no mapa digital poderia exibir diversas informações sobre bairros combinando os atributos área, perímetro e população, por exemplo.

Segue uma ilustração de uma consulta realizada sobre um mapa de bairros de uma cidade:

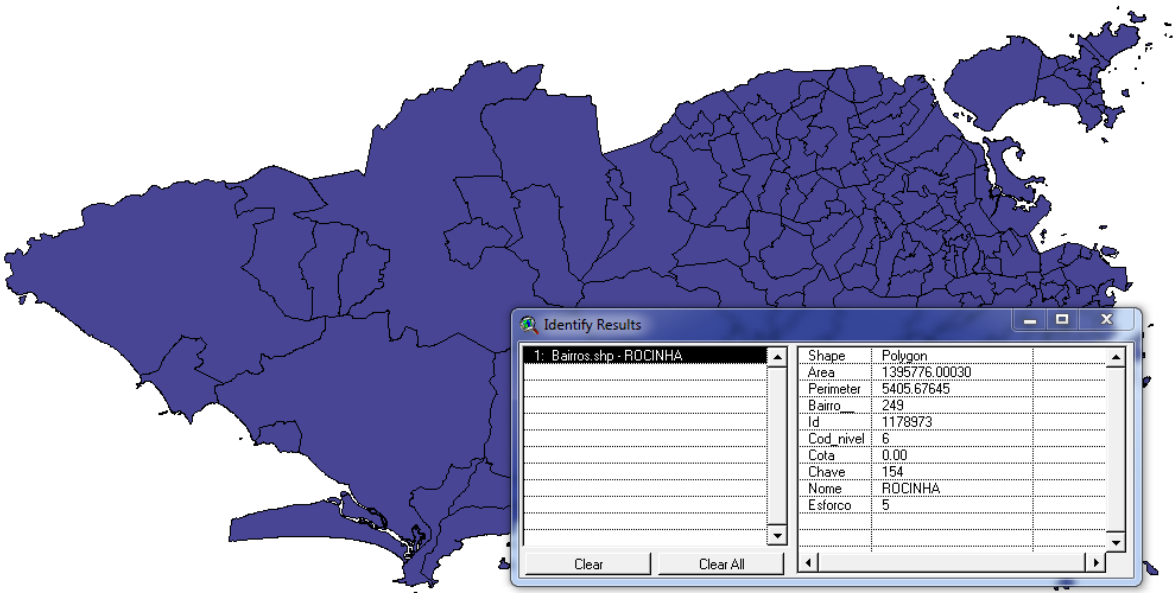
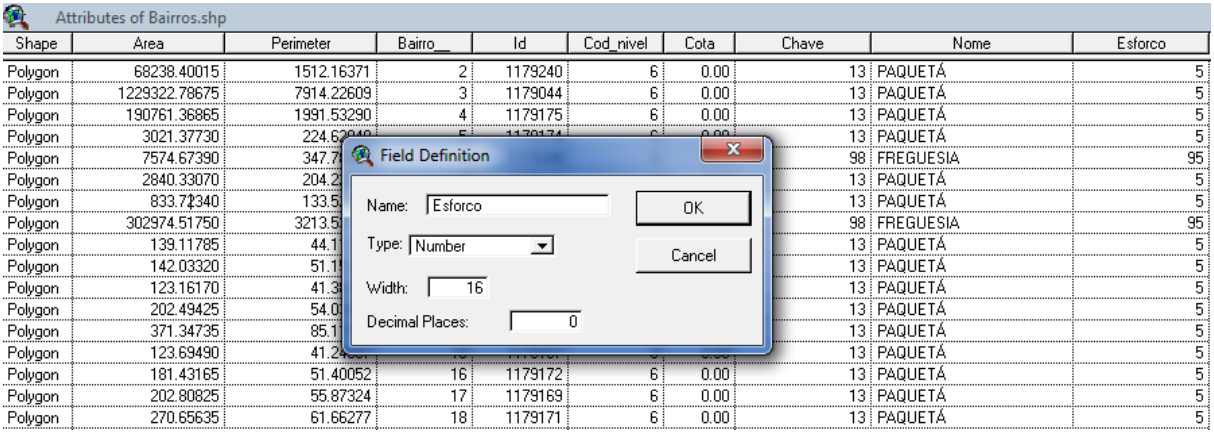


Figura 29: Consulta sobre um arquivo no formato shapefile.

As funcionalidades de um aplicativo como o ArcView GIS 3.2 resolvem a primeira etapa da solução proposta para a seleção de rotas desse trabalho: a elaboração de um mapa de esforço. Um mapa de esforço para o mapa de bairros apresentado acima, por exemplo, poderia ser definido a partir da criação de um novo atributo em sua tabela temática. No ArcView GIS 3.2 é possível criar um novo atributo através da opção adicionar campo do menu editar. Segue uma ilustração da criação de um novo atributo no ArcView:



Shape	Area	Perimeter	Bairro_	Id	Cod_nivel	Cota	Chave	Nome	Esforco
Polygon	68238.40015	1512.16371	2	1179240	6	0.00	13	PAQUETÁ	5
Polygon	1229322.78675	7914.22609	3	1179044	6	0.00	13	PAQUETÁ	5
Polygon	190761.36865	1991.53290	4	1179175	6	0.00	13	PAQUETÁ	5
Polygon	3021.37730	224.62	5	1179174	6	0.00	13	PAQUETÁ	5
Polygon	7574.67390	347.7					98	FREGUESIA	95
Polygon	2840.33070	204.2					13	PAQUETÁ	5
Polygon	833.72340	133.5					13	PAQUETÁ	5
Polygon	302974.51750	3213.5					98	FREGUESIA	95
Polygon	139.11785	44.1					13	PAQUETÁ	5
Polygon	142.03320	51.1					13	PAQUETÁ	5
Polygon	123.16170	41.3					13	PAQUETÁ	5
Polygon	202.49425	54.0					13	PAQUETÁ	5
Polygon	371.34735	85.1					13	PAQUETÁ	5
Polygon	123.69490	41.24					13	PAQUETÁ	5
Polygon	181.43165	51.40052	16	1179172	6	0.00	13	PAQUETÁ	5
Polygon	202.80825	55.87324	17	1179169	6	0.00	13	PAQUETÁ	5
Polygon	270.65635	61.66277	18	1179171	6	0.00	13	PAQUETÁ	5

Figura 30: Criação de atributo temático no ArcView.

O atributo esforço poderá ser preenchido de acordo com os critérios de esforço definidos para um projeto específico. Essa flexibilidade permitirá que uma grande diversidade de aplicações seja trabalhada através dessa mesma abordagem.

Por exemplo, a construção de uma via expressa interligando bairros de uma cidade exigiria critérios para classificação de esforço completamente diferentes da construção de uma linha de metrô. Porém, uma vez criado o atributo esforço na tabela temática do mapa digital todas as diferenças entre os dois projetos estarão resolvidas em uma única variável e os mapas de esforço de ambos poderão ser consultados da mesma forma.

A interoperabilidade de um arquivo do tipo *shapefile* permite que uma grande quantidade de mapas possa ser utilizada e garante um rico acervo de informações espaciais para se trabalhar. Além disso, os polígonos contendo as informações de esforço podem representar diversos tipos de feições espaciais e consolidar diversas camadas de mapas, o que garante ao modelo mais flexibilidade para a elaboração do mapa de esforço.

Portanto, pode-se assumir que a elaboração de mapas de esforço para compor a solução do problema de seleção de rotas com obstáculos poderá ser feita de forma eficiente pelo ArcView GIS 3.2. Dessa forma, a próxima demanda passa a ser a definição de uma ferramenta para exploração de rotas sobre mapas de esforço.

2.2.2.Ambiente baseado em agentes

Um sistema baseado em agentes pode ser entendido com um ambiente onde é possível explorar comportamentos de indivíduos (nível micro) com o objetivo de observar padrões que possam emergir de suas interações (nível macro). O NetLogo é uma ferramenta para

desenvolvimento de sistemas baseados em agentes disponibilizada livremente na Internet e utilizada por estudantes e professores em todo Mundo (NetLogo, 2012).

O NetLogo permite que centenas ou até milhares de agentes sejam criados e instruídos a trabalhar de forma autônoma e independente. Com isso, é possível através dessa ferramenta modelar, com certa facilidade, uma grande diversidade de fenômenos naturais e sociais. O NetLogo atende muito bem a necessidade de uma ferramenta para exploração de rotas. Nesse trabalho será utilizada a versão 4.1.1.

No NetLogo 4.1.1 tudo é composto por agentes. Em outras palavras, o NetLogo é um Mundo composto de agentes, onde um agente pode ser entendido como uma entidade capaz de seguir as instruções que lhe são passadas de maneira independente. Cada agente pode conduzir suas próprias atividades simultaneamente com outros.

Um agente no NetLogo não assume apenas um papel. Na realidade um agente no NetLogo pode ser um elemento que compõe o próprio ambiente de um modelo, pode ser uma ligação entre dois ou mais agentes ou simplesmente uma ser entidade que observa outros agentes em execução. Esses diferentes papéis de um agente no NetLogo podem ser resumidos nos seguintes tipos:

a)*Turtles*;
b)*Patches*;
c)*Links*; e
d)*Observer*.

Patches são agentes que compõem o ambiente de um modelo (de um mundo abstrato). Esse ambiente é representado por uma matriz bidimensional onde cada *Patch* é um elemento que ocupa uma posição de linha *m* e coluna *n*. O conjunto de *Patches* define a superfície sobre a qual um *Turtle* deve executar.

Um *Turtle* representa uma entidade com a capacidade de executar ações e, a partir dessas ações, transformar seu ambiente. Em outras palavras, um *Turtle* pode alterar o estado ou as propriedades das coisas que compõem um ambiente no NetLogo.

Um *Link* é um agente capaz de estabelecer a ligação entre outros agentes. Essa ligação viabiliza a comunicação entre agentes e também permite que o estado de um agente seja alterado por outro. É possível definir critérios para que determinada mensagem seja propagada para apenas determinado grupo de agentes dentre todos aqueles interligados por um *Link*.

Para que as ações e os estados dos agentes possam ser acompanhadas por usuários (ou por outros sistemas), existe um tipo de entidade especial no NetLogo que pode ver tudo o que acontece durante as interações entre *Turtles* e *Patches*, essa entidade é chamada de *Observer*. Com isso, é possível consultar ou enviar comandos para os agentes de um ambiente.

Qualquer agente pode ter seu estado consultado ou receber um comando para executar uma ação. Entretanto, para ter acesso ao estado de um agente ou lhe enviar um comando é preciso indexá-lo. Um agente *Turtle* é identificado (ou indexado) por um número chamado *who number*. Esse número é atribuído a um agente no instante em que ele é criado. Já um *Patch* é identificado por um par de coordenadas (sua posição na matriz que representa a superfície de um ambiente) e um *Link* é identificado pelo *who number* dos agentes conectados por ele.

Para facilitar a criação de modelos ou aplicações o NetLogo disponibiliza ao usuário uma variedade de componentes para interface gráfica. Alguns dos componentes mais comuns são:

a)*Buttons*: "*once*" ou "*forever*";
b)*Speed Slider*: para controlar a velocidade de execução;
c)*Sliders* e *Switches*: para alterar parâmetros de execução;
d)*Plots* e *Monitors*: gráficos e resultados de variáveis utilizadas durante uma execução; e
e)*Model Settings*: para configurar as dimensões (visão) do ambiente.

Componentes do tipo *Button once* executam apenas uma ação enquanto componentes do tipo *Button forever* executam infinitamente a mesma ação até que o usuário interrompa a execução. Segue uma ilustração com os tipos de componentes citados:

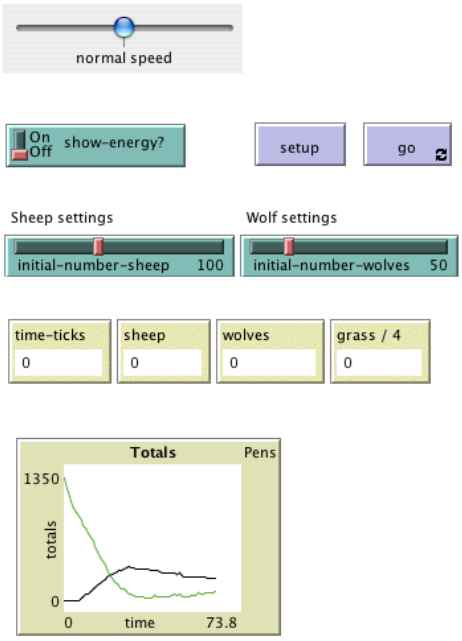


Figura 31: Componentes de interface gráfica disponíveis no NetLogo.

A superfície de um ambiente pode ser configurada através da janela *Model Settings*, conforme demonstrado na ilustração abaixo:

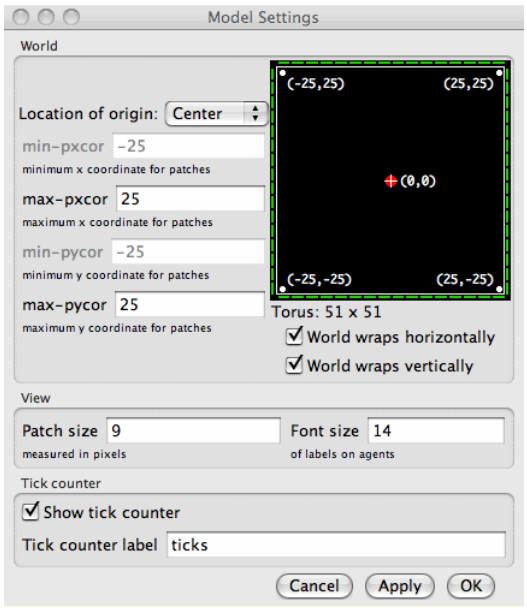


Figura 32: Tela de configuração de uma superfície no NetLogo.

No NetLogo é possível interagir diretamente com os agentes de um modelo através de uma funcionalidade chamada *Command Center*. Através dessa funcionalidade é possível enviar comandos inclusive enquanto os agentes executam. Apenas agentes do tipo *Observer*,

Turtle e *Patch* recebem comandos via *Command Center*. Na realidade, o *Command Center* pode ser entendido como a interface com o agente *Observer* de um modelo. Em outras palavras, os comandos são enviados ao *Observer* e repassados às demais entidades do ambiente. Segue uma ilustração do *Command Center*:

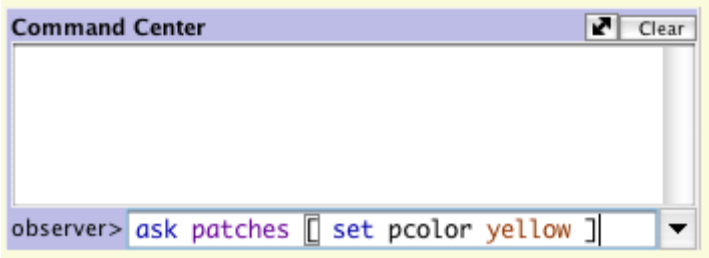


Figura 33: Janela Command Center do NetLogo.

Outro recurso bastante útil do NetLogo é o *Monitor*. Existe um monitor para cada tipo de agente. Os atributos de um agente do tipo *Turtle* podem ser consultados através de seu *who number* utilizando a janela *Turtle Monitor*. De forma semelhante, um *Patch* pode ser consultado através de suas coordenadas utilizando o *Patch Monitor* e um *Link* pode ser consultado através dos identificadores dos agentes conectados por ele (*end points*). Segue uma ilustração de consulta feita sobre um agente:

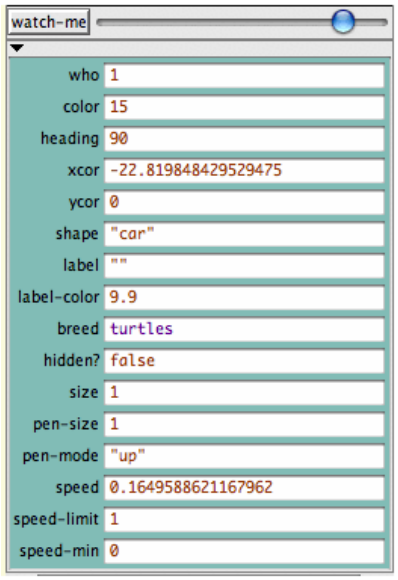


Figura 34: Consulta utilizando Turtle Monitor do NetLogo.

Além dos componentes de interface gráfica o NetLogo também possui um linguagem de programação. Nessa linguagem os blocos de instrução são constituídos basicamente de dois tipos de instrução:

a)*Commands*; e
b)*Reports*.

Tanto as instruções do tipo *Command* quanto as instruções do tipo *Report* dizem aos agentes o que fazer. A diferença é que um *Command* implica na ação de um agente enquanto um *Report* relata um resultado.

Um tipo específico de bloco de instrução é a *Procedure*. Uma *Procedure* combina uma série de *Commands* e *Reports* em uma unidade lógica identificada por uma assinatura. Estruturas desse tipo ajudam a organizar o fluxo de execução de um programa além de propiciar a reutilização de rotinas. Segue um exemplo de declaração de uma *Procedure*.

```
to setup                ; assinatura
 clear-all              ; sequência de comandos
 create-turtles 100
 ask turtles [ setxy random-xcor random-ycor ]
end                     ; fim da procedure
```

Uma *Procedure* também pode receber parâmetros como argumento (entrada). Nessas situações o parâmetro recebido é tratado como uma variável local pela *Procedure* e seu conteúdo perdido quando a *Procedure* encerrar sua execução. Segue um exemplo de uma *Procedure* recebendo um parâmetro como argumento:

```
 to draw-polygon [num-sides len]    ;declaração de procedure com parâmetro de entrada
  pen-down
  repeat num-sides
    [ fd len
      rt 360 / num-sides ]
 end
 ask turtles [ draw-polygon 8 who ]   ;chamada passado valor como parâmetro
```

Além de receber parâmetros, uma *Procedure* pode também devolver um valor quando ao encerrar sua execução. Nesse caso o comando *report* deve ser utilizado. Esse comando pode ser posicionado em qualquer ponto do corpo de uma *Procedure* e sua execução implica no encerramento da *Procedure*. Segue um exemplo de *Procedure* com retorno de valor:

```
to-report absolute-value [number]
  ifelse number >= 0
    [ report number ]              ;primeira possibildade de retorno
  [ report (- number) ]            ;segunda possibilidade de retorno
end
```

Os blocos de instrução apresentados acima acessam variáveis durante sua execução. Variáveis são áreas de memória utilizadas para armazenar valores relacionados a um agente. Essas áreas de memória podem ser de escopo global, uma propriedade de um agente ou uma variável local de uma *Procedure*.

Uma variável de escopo global está sempre visível para qualquer entidade de um ambiente. Já uma variável local de uma *Procedure* ou de um bloco de instrução não pode ser acessada de fora dessa *Procedure* ou bloco. Em outras palavras, variáveis locais são declaradas e utilizadas no contexto de uma *Procedure* enquanto as variáveis globais são utilizadas no contexto de um programa. Uma variável pode ser declarada conforme o exemplo abaixo:

```
global [result]              ;declaração de variável global
let score 50                 ;declaração de variável local
```

Quando uma variável é declarada como propriedade de um agente algumas particularidades devem ser consideradas. Primeiramente, é preciso que a variável seja declarada de acordo com o tipo de agente que irá contê-la (*Turtle*, *Pacth* ou *Link*). Além disso, o acesso à variável de um agente faz referência ao agente. Segue um exemplo de como utilizar variáveis de agentes:

```
turtles-own [energy speed]       ;declaração de variável de um tipo turtle
patches-own [friction]
links-own [strength]

show [color] of turtle 5         ;acesso à variável color do agente 5
```

Um agente pode receber instruções a partir do *Command Center* ou de qualquer ponto de um programa em execução. Para enviar uma instrução para um agente deve-se utilizar o comando *ask*.

Quando utilizado, o comando *ask* obriga que os agentes requisitados executem suas funções sequencialmente. Quando se faz necessária uma execução concorrente (ou seja,

quando é necessário que os agentes executem em paralelo) deve-se utilizar o comando *ask-concurrent*. Segue um exemplo de utilização do comando *ask*.

```
ask turtle 0              ;solicita ao primeiro agente turtle
   [ ask patch-at 1 0     ;que o patch de coordenada (1, 0) altere sua cor
     [ set pcolor red ] ]
```

O comando *ask* pode ser utilizado também para enviar requisições a um grupo de instâncias de agentes. Um conjunto de instâncias de agentes é chamado de *Agentset*. Esse conjunto pode ter instâncias dos tipos *Turtle*, *Pacth* ou *Link*. Entretanto, não poderá haver mais de um tipo diferente em um mesmo *Agentset*. Com isso é possível criar um grupo de agentes com características semelhantes (ou de uma mesma classe) e tirar proveito dessas características para estabelecer critérios de seleção, conforme demonstrado abaixo:

```
ask patches with [ pxcor > 10 and pycor > 20 ] [ set pcolor red ]
ask one-of patches with [ pxcor > 5 and pycor > 5 ] [ set pcolor green ]
```

No NetLogo é possível criar agentes com características e comportamentos comuns através de declarações do tipo *Breed*. Estruturas do tipo *Breed* podem ser definidas como classes ou espécies de agentes. Quando é definida uma nova espécie de agente com a declaração *breed* um *Agentset* é automaticamente incorporado a essa espécie, incorporando diversos recursos da linguagem para operações com conjuntos, tais como: *turtles-on*, *others* e *with*. Segue um exemplo do uso da declaração do tipo *Breed*:

```
breed [wolves wolf]          ;cria uma classe do tipo lobo
breed [sheep a-sheep]        ;cria uma classe do tipo ovelha
```

Outro aspecto muito importante da linguagem de programação do NetLogo é a capacidade de se a estabelecer comunicação entre agentes. Segue um exemplo que explora a possibilidade de se criar dependência entre conjunto de instâncias de agentes:

```
breed [ suns sun ]                          ;declara uma classe de estrelas
 breed [ planets planet ]                     ;declara uma classe de planetas
 to setup
  create-suns 1
  create-planets 5
    [ create-link-from one-of suns[ tie hide-link ]  ]    ;cria um link e associa os
movimentos
 end
```

Em muitas situações pode ser útil trabalhar com listas no NetLogo. Um lista no NetLogo pode ser composta por itens de diferentes tipos, tais como: números, cadeias de caracteres, agentes e, até mesmo, outras listas. Segue um exemplo simples de declaração de lista no NetLogo:

```
set mylist [2 4 6 8]
set mylist [[2 4] [3 5]]
```

No NetLogo 4.1.1 uma lista não pode ser alterada. Consequentemente, sempre que for necessário alterar um item de uma lista uma nova lista será criada sobrescrevendo os valores antigos. Essa limitação pode, em determinadas situações, impactar no desempenho de um programa.

Novos itens podem ser adicionados a uma lista através dos comandos *lput* e *fput*. O comando *lput* adiciona um item no final da lista e o comando *fput* adiciona no início. Segue um exemplo de utilização de listas no NetLogo:

```
set mylist [2 7 5 Bob [3 0 -2]]
set mylist replace-item 2 mylist 10   ; uma nova lista é criada com o terceiro item alterado
 set mylist lput 42 mylist            ; adiciona o número 42 no final da lista
```

Existem várias funcionalidades para se manipular elementos de uma lista no NetLogo. Para percorrer um item por vez deve-se utilizar o comando *foreach*. Para se aplicar um critério de ordenação deve-se utilizar o comando *sort-by*. Para se aplicar uma operação matemática ou lógica sobre todos os itens de uma lista deve-se utilizar o comando *map*. Seguem alguns exemplos de manipulação de listas no NetLogo:

```
foreach  [2 4 6]
   [ crt ?]                          ; cria 2, 4 e 6 agentes sequencialmente
show map [? < 0] [1 -1 3 4 -2 -10]   ; imprime [false true false false true true]
show sort-by [?1 < ?2] [4 1 3 2]     ; imprime [1 2 3 4]
```

Com as funcionalidades do NetLogo apresentadas até aqui é possível modelar uma grande quantidade de fenômenos naturais através do paradigma de programação baseada em agentes. Entretanto, será preciso explorar também a capacidade de interoperabilidade e integração do NetLogo com outros sistemas. Para atender a esses requisitos o NetLogo 4.1.1 disponibiliza as seguintes funcionalidades:

a)*Controlling* API;

b)*Extensions* API; e

c)*Applets*.

Através do *Controlling* API o NetLogo pode ser invocado e controlado por outro programa executando sobre a Máquina Virtual Java (*Java Virtual Machine*[4]). Em outras palavras, a *Controlling* API permite que um modelo criado no NetLogo seja embutido em outro aplicativo e que as funcionalidades de modelo sejam utilizadas pelo aplicativo. Por exemplo, um pequeno aplicativo poderia automatizar uma série de modelos de execução do NetLogo ou uma grande aplicação poderia embarcar um conjunto de modelos do NetLogo.

Também é possível incorporar novas funções ao ambiente do NetLogo. Essa perspectiva pode ser trabalhada através de *Extensions* API. O NetLogo permite que usuários desenvolvam novos *Commands* e *Reporters* para uso em seus modelos. Existem, inclusive, algumas extensões bastante úteis disponibilizadas na Internet. Dentre essas extensões pode-se destacar como importantes para esse trabalho as seguintes:

a)MySql *for* NetLogo 4.1: extensão para integração com o gerenciador de banco de dados MySQL;

b)Sql *for* NetLogo 4.1: extensão, baseada em JDBC[5], para integração com os gerenciadores de banco de dados MySQL e PostgreSQL; e

c)GIS *Extension*: extensão para manipulação de arquivos do tipo *shapefile*, disponibilizada com a versão 4.1.1.

Além das possibilidades de estender o ambiente de programação (*Extensions* API) e de controlar modelos embarcados em outros aplicativos (*Controlling* API), também é possível gerar um *applet*[6] a partir de um modelo NetLogo e executá-lo em um navegador WEB. Essas três possibilidades de integração disponíveis no NetLogo 4.1.1 garantem a flexibilidade necessária para resolver uma diversidade imensa de problemas.

Com isso, não apenas o problema de exploração de rotas poderá ser resolvido, também poderão ser integradas outras ferramentas ao NetLogo, tornando-o o módulo central de uma arquitetura sistêmica capaz consultar mapas digitais, processar informações de rotas,

---

[4] *Java Virtual Machine* (JVM) é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina.

[5] *Java Database Connectivity* (JDBC) é uma interface Java para envio de instruções SQL para gerenciadores de banco de dados relacional.

[6] *Applet* é um pequeno aplicativo JAVA que executa no contexto de outro programa.

classificá-las com base nas medidas de distância e esforço e apresentar os resultados dessa classificação em um mapa.

2.3.**Arquitetura**

A possibilidade de utilizar o paradigma de programação baseada em agentes e as funcionalidades de extensão do NetLogo 4.1.1 direcionam a solução do problema de seleção de rotas, abordado nesse trabalho, para uma arquitetura sustentada, em grande parte, pelo próprio NetLogo. Nessa arquitetura deve-se planejar a integração de três grandes processos:

a)A configuração de terrenos utilizando mapas digitais;
b)O planejamento e a exploração de rotas; e
c)A tomada de decisão (ou a seleção de rotas) sobre a exploração realizada.

A integração desses processos implica na definição de um modelo de sistema, aqui chamado de Híbrida, com o qual seja possível materializar uma técnica híbrida para geração de rotas em espaço geográfico com obstáculos, onde essa técnica híbrida significa uma combinação de conhecimentos de lógica nebulosa, sistemas baseados em agentes, planejamentos de rotas e mapas digitais.

Segue uma ilustração do funcionamento esquemático do sistema Híbrida:

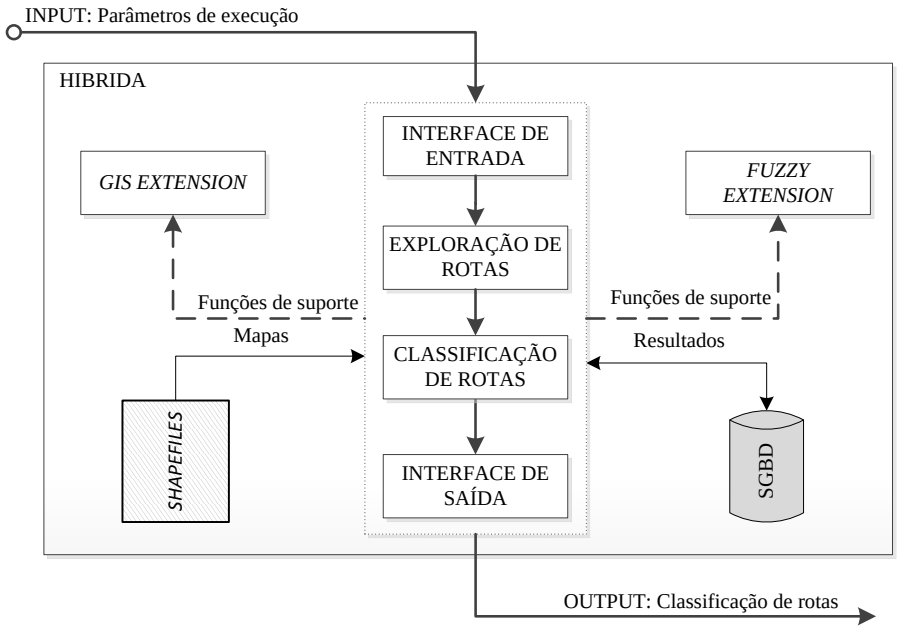


Figura 35: Esquema básico do sistema Híbrida.

O módulo principal da arquitetura do sistema deve atender aos processos de planejamento, exploração e classificação de rotas, sendo os demais módulos acoplados à estrutura simplesmente para suportar as operações executadas pelo módulo principal. Essas operações de suporte foram organizadas nos chamados módulos de extensão do sistema. O funcionamento básico do módulo principal executa quatro operações:

a)Interface de entrada;
b)Exploração de rotas;
c)Classificação de rotas; e
d)Interface de saída.

Já operações de extensão estão organizadas em dois módulos básicos:

a)Extensão SIG[7] (*Extension GIS*); e
b)Extensão nebulosa (*Extension Fuzzy*).

A interface de entrada deve tratar o conjunto de informações fornecido ao sistema com o objetivo de parametrizar seu funcionamento. Uma grande variedade de combinações deve ser prevista para que todo potencial do paradigma de programação baseada em agentes e dos mapas digitais sejam explorados. Algumas das entradas (definições) previstas são:

a)A quantidade de agentes que executarão em uma exploração;
b)A configuração de propriedades do ambiente sobre o qual executará um agente;
c)A configuração das características de um agente; e
d)Os aspectos que determinam os planos de rotas dos agentes.

As propriedades e variáveis do ambiente de execução devem ser definidas, em sua maioria, através da leitura de mapas digitais (*shapefiles*). Portanto, será mantido um acervo de mapas digitais pelo sistema para configuração do ambiente de execução dos agentes, conforme parametrização do módulo de entrada. Nota-se que, nesse cenário, a interface de entrada dependerá fortemente da extensão SIG para realizar a leitura de dados provenientes de mapas digitais.

[7] SIG: Sistema de Informações Geográficas

Para iniciar a atividade de exploração de rotas assume-se que o ambiente de execução e o conjunto de agentes necessários para a exploração já estejam devidamente instanciados e organizados para o processamento. O principal objetivo da atividade de exploração é registar as informações de distância e esforço dos agentes durante seus ciclos de vida.

Cada agente na atividade de exploração carregará consigo informações elementares para improvisar sua trajetória no ambiente. No final de sua execução um agente terá armazenado conhecimento sobre o custo de sua exploração no mapa. Dessa forma, a atividade de exploração, realizada por um conjunto de agentes, fornecerá insumos essenciais para a classificação de rotas.

Nota-se que os conhecimentos adquiridos nas explorações de rotas devem ser preservados e compartilhados com outros módulos do sistema. Essa necessidade será suportada na arquitetura proposta por uma integração com um Sistema Gerenciador de Banco de Dados (SGBD). A integração com o SGBD será encapsulada no módulo de extensão *fuzzy*. Dessa forma, a extensão *fuzzy* irá expor métodos (funções) para que as informações de rotas adquiridas pelos agentes sejam preservadas (registradas) em suas estruturas.

Para realizar a classificação de rotas, as informações geradas pela atividade de exploração serão processadas através de funções de um sistema de inferência nebuloso. Essas funções também estão organizadas no módulo de extensão *fuzzy*. A classificação dependerá do modelo de inferência e da base de regras definidos em estruturas da extensão *fuzzy*.

Outro aspecto importante da atividade de classificação é preservar as informações geradas e disponibilizá-las para a interface de saída. Nota-se mais uma vez a necessidade de integração com um SGBD. Essa integração também será encapsulada no módulo de extensão *fuzzy*, sendo disponibilizados apenas os métodos de consulta das classificações geradas.

Na interface de saída devem ser previstas algumas configurações de apresentação dos resultados obtidos sobre um mapa. Algumas possibilidades são:

a)Definir a quantidade de classificações a serem exibidas;
b)Carregar no mapa informações utilizadas em uma classificação; e
c)Apagar informações exibidas e iniciar nova consulta.

Com isso, a partir de uma grande quantidade de classificações disponíveis, o módulo de saída permitirá que as consultas possam ser feitas com certa flexibilidade para que um usuário possa analisar os resultados na apresentação mais adequada.

Dessa forma, pode-se resumir a arquitetura do sistema Híbrida proposta nesse trabalho da seguinte forma: um mapa digital fornecerá as informações necessárias para configuração de um terreno, os agentes do módulo de exploração de rotas executarão os algoritmos de planejamento de rotas para coletar informações de esforço e distância da maior diversidade de rotas possível sobre o terreno e o módulo de inferência nebuloso tomará as decisões necessárias para realizar uma classificação de rotas satisfatória.

2.4.**Algoritmos de planejamento de rotas adotados**

Um dos principais módulos da arquitetura proposta para o sistema Híbrida é o modulo de exploração de rotas. O funcionamento desse módulo é uma combinação dos conceitos de sistemas baseados em agentes e dos algoritmos de planejamento de rotas. Na realidade, alguns algoritmos de planejamento de rotas foram utilizados nesse trabalho para definir o funcionamento de agentes no ambiente do NetLogo. Os algoritmos adotados nesse trabalho para o módulo de exploração de rotas foram:

a)Planejamento de rotas por grafo de visibilidade;
b)Por decomposição celular; e
c)Por campo potencial.

Para adaptar esses algoritmos ao ambiente de programação baseado em agentes e, principalmente, às particularidades introduzidas pelos mapas digitais, serão necessárias algumas inovações para viabilizar a solução proposta.

No caso do grafo de visibilidade, onde os caminhos possíveis entre dois pontos do terreno são determinados pelos vértices dos obstáculos existentes entre esses pontos, o uso de mapas digitais introduz bastante complexidade ao problema. Isso porque os polígonos contidos em uma feição de um mapa digital podem conter centenas de vértices.

Apontar todos os vértices de um polígono de um mapa digital não é apenas trabalhoso, em alguns casos pode ser também inútil porque, dependendo da forma do polígono, alguns vértices não irão gerar caminhos viáveis (arestas) para um agente. A ilustração que segue demonstra que, embora a estratégia de traçar caminhos a partir dos vértices de um polígono (obstáculo) funcione bem para formas mais simples, algumas adaptações são necessárias na proposta do grafo de visibilidade quando polígonos com formas mais complexas fazem parte do problema.

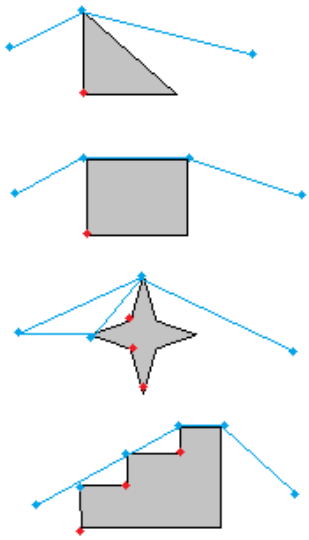


Figura 36: Formas de obstáculos e grafo de visibilidade.

Nota-se na ilustração acima que quando um obstáculo tem a forma de um triângulo o traçado de um caminho apontando para os vértices do polígono atende perfeitamente. Entretanto, quando o obstáculo assume a forma de um quadrado surge uma primeira questão: o melhor traçado deve ter a ligação de dois vértices de um mesmo obstáculo. Aumentando a complexidade do formato do obstáculo nota-se que alguns vértices não beneficiariam o traçado.

Em outras palavras, instruir um agente a obter todos os vértices de um obstáculo para planejar um traçado seria improdutivo. Se considerarmos que um polígono de um mapa digital pode ter centenas de vértices e que esses vértices seriam ligados a outros milhares para comporem as arestas de um grafo de visibilidade, muito processamento seria necessário e muito provavelmente isso inviabilizaria a solução.

Para contornar o problema introduzido por obstáculos de formas mais complexas propõe-se que as arestas do grafo de visibilidade sejam construídas a partir dos centroides de polígonos vizinhos a um obstáculo. Essa solução não apenas reduz enormemente a quantidade de vértices no grafo de visibilidade, mas também se beneficia do uso de mapas digitais, uma vez que polígonos vizinhos a um obstáculo e seus centroides podem ser obtidos com facilidade.

A ilustração a seguir demonstra um traçado contornando um obstáculo a partir dos centroides de polígonos vizinhos:

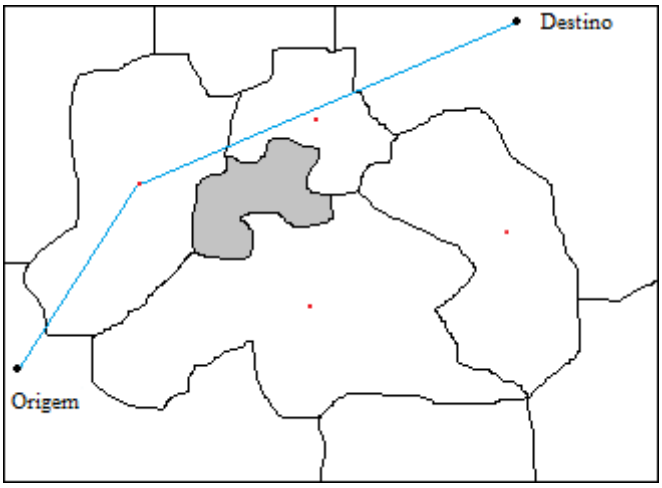


Figura 37: Grafo de visibilidade a partir de centroides da vizinhança.

A partir dos centroides pode-se criar uma infinidade de arestas para compor um grafo de visibilidade. Cada centroide pode ser combinado com qualquer outro e formar uma aresta válida, desde que essa aresta não atravesse um obstáculo. A combinação de centroides ao redor de um mesmo obstáculo também é possível e pode gerar caminhos sinuosos.

No caso do algoritmo de decomposição celular, onde o espaço geográfico é divido para que o traçado seja gerado a partir de células adjacentes, o uso de mapas digitais facilita bastante o trabalho. As células podem ser organizadas no próprio mapa digital. Durante a configuração do ambiente será necessário apenas apontar os centroides de cada polígono do mapa.

Feito o apontamento do centroide o funcionamento do algoritmo de decomposição celular funcionará exatamente como no caso do grafo de visibilidade. Na realidade, a ligação entre células adjacentes é simplesmente uma ligação de centroides. Em outras palavras, conectar células é como criar uma aresta para compor um grafo de visibilidade.

Como o objetivo desse trabalho é explorar a maior diversidade de rotas possível, as ligações entre células não serão limitadas apenas às células adjacentes. Ou seja, uma célula pode estabelecer uma conexão com outra célula distante. Essa adaptação pode ajudar na identificação de atalhos importantes para a seleção de rotas, mas também pode sobrecarregar os recursos do sistema.

Dependendo da quantidade de polígonos de um mapa a combinação de caminhos entre células pode demandar um esforço computacional que inviabilize a solução. Portanto, é adequado manter a possibilidade de criar ligações apenas entre células adjacentes.

No método de planejamento de rota por campo potencial a direção e o sentido da trajetória de um agente devem ser calculados de acordo com a resultante das forças de repulsão, imposta por obstáculos, e de atração, imposta pelo destino do agente. Esse algoritmo poderia ser executado por um único agente, mas um pequeno número de rotas seria explorado. Além disso, existe a possibilidade do agente permanecer em trajetória cíclica (em movimento circular), dependendo da configuração do ambiente.

As limitações do método de planejamento por campo potencial quando aplicado em problemas de exploração de rotas motivou o desenvolvimento de um método que se beneficiasse da simplicidade e do bom desempenho do método de campo potencial sem deixar de explorar uma maior diversidade de rotas. As principais questões levantadas sobre o método de campo potencial nesse trabalho foram:

a)Quando um agente encontra um obstáculo sua direção muda. Seria possível explorar mais de uma direção ao encontrar um obstáculo?

b)Um agente pode entrar em um movimento cíclico. Como isso poderia ser evitado?

c)Para realizar a trajetória do ponto de origem ao destino é necessário apenas um agente. Seria possível disparar um número qualquer de agentes em diferentes direções para explorar um número maior de rotas?

Para resolver a questão relacionada ao desvio de obstáculos propõe-se que o agente não apenas mude sua direção, mas que também seja duplicado e assuma o sentido oposto ao de sua réplica. Em outras palavras, sempre que um agente sofrer a ação de um campo potencial de um obstáculo ele será duplicado e cada réplica fará um desvio com sentido oposto ao de sua cópia. A ilustração abaixo demonstra o funcionamento original do algoritmo de campo potencial:

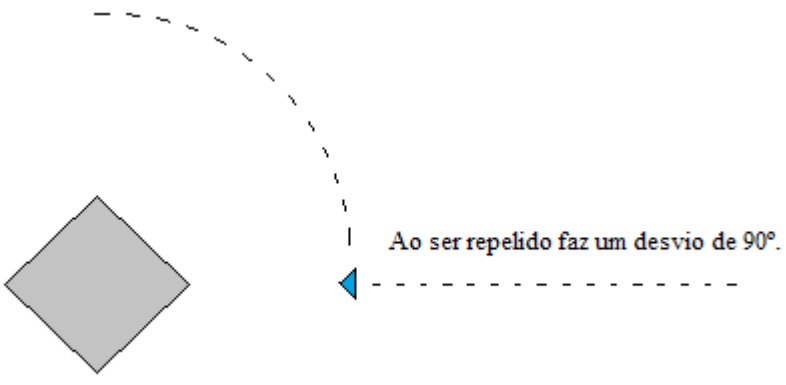


Figura 38: Funcionamento do algoritmo de rota por campo potencial.

Nota-se que apenas um sentido é explorado no método original, o que resolve o problema de encontrar um caminho até o destino, mas não ajuda muito a exploração de rotas. A ilustração abaixo demonstra a adaptação proposta nesse trabalho:

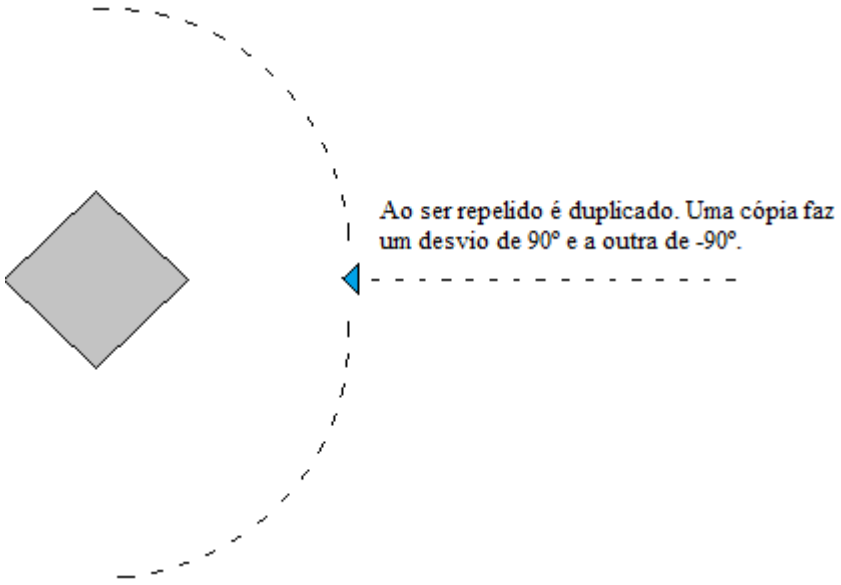


Figura 39: Funcionamento do algoritmo de rota por campo potencial alterado.

Com essa alteração no algoritmo de rotas por campo potencial surgem duas observações importantes:

a) A força de repulsão deve ser exercida apenas quando o agente identifica um obstáculo em sua trajetória para evitar que o agente continue se duplicando; e
b) Seria possível criar um número qualquer de réplicas e explorar maiores possibilidades de caminho.

Nesse trabalho a força de repulsão dos obstáculos irá influenciar a direção de um agente apenas quando existir a possibilidade de colisão. Dessa forma, um agente poderá passar entre dois obstáculos com mais facilidade, sem que esses obstáculos influenciem de forma prejudicial na trajetória do agente. Além de evitar duplicações e outras ações desnecessárias, essa abordagem ajudará a evitar também o problema de caminhos cíclicos. Isso porque a cada possibilidade de colisão será feito um único desvio e, após isso, o agente seguirá sobre a força de atração de seu destino apenas.

A possibilidade de criar um número maior de réplicas a cada possibilidade de colisão é uma forma interessante de melhorar a exploração de rotas. Nesse caso, propõe-se que as

réplicas compartilhem as possibilidades de direção mantendo certo distanciamento entre si. Caso a amplitude de direções seja superior a um módulo de 180º, de -90º a 90º, alguns agentes realização um pequeno retorno em direção à origem até que a atração exercida pelo ponto de destino os redirecione.

Uma possibilidade seria dividir um agente por quatro direções para evitar uma colisão. Cada réplica manteria um distanciamento de 45º totalizando uma amplitude de 180º, sendo 90º acima do traçado original e outros 90º abaixo do traçado original. Segue uma ilustração de um agente sendo replicado em quatro direções:

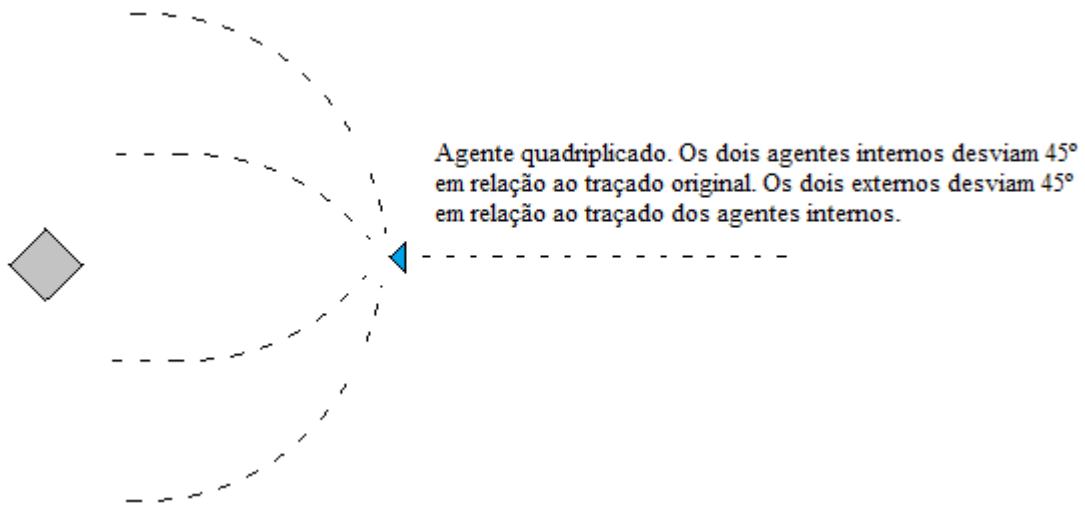


Figura 40: Maior diversidade rotas com o algoritmo de campo potencial alterado.

Para que rotas sejam exploradas utilizando o algoritmo de campo potencial alterado será necessário apenas demarcar as áreas caracterizadas como obstáculos nos mapas digitais. Ao contrário dos algoritmos que utilizam grafo de visibilidade ou decomposição celular, o método de campo potencial não necessita da delimitação do espaço por meio de grafos e os agentes executam com maior liberdade.

Com os algoritmos para planejamento de rotas apresentados e devidamente adaptados para a exploração de rotas será possível traçar um grande número de rotas sobre um mapa digital. Esses algoritmos direcionaram os agentes sobre os mapas entre os pontos de origem e destino e desviando de áreas classificadas com instransponíveis, por demandarem um esforço que inviabilizaria uma determinada rota. Durante suas trajetórias os agentes contabilizarão as medidas de distância e esforço, gerando as informações necessárias para classificação de rotas.

2.5.**Sistema de inferência nebuloso**

A partir das informações levantadas durante a exploração de rotas feitas deve-se iniciar o processo de classificação das rotas. Essa classificação será realizada pelo módulo de inferência nebuloso do sistema proposto. O modelo de inferência nebuloso utilizado será apresentado nessa seção de acordo com os passos listados abaixo:

a)Apresentação das variáveis linguísticas utilizadas;
b)Apresentação dos termos definidos para as variáveis linguísticas;
c)Descrição das funções de pertinência empregadas;
d)Descrição das regras empregadas; e
e)Explicação do método *defuzzifier* utilizado.

Para realizar a classificação das rotas exploradas pelos agentes o módulo de inferência nebuloso receberá os dados de distância e esforço de cada rota explorada. Esses dados deverão ser utilizados pelo módulo de inferência para calcular o valor do custo de uma rota e a partir desse custo apresentar uma classificação. Portanto, foram definidas três variáveis linguísticas no módulo de inferência nebuloso:

$$Distância = (x, T(x), U, G, M)$$

Onde:

x = *distância;*
$T(x)$ = *{curta, média e longa}; e*
U = *[0, 250].*

$$Esforço = (x, T(x), U, G, M)$$

Onde:

x = *esforço;*
$T(x)$ = *{pequeno, médio e grande}; e*
U = *[0, 100].*

$$Custo = (x, T(x), U, G, M)$$

Onde:

x = *custo;*
$T(x)$ = *{baixo, médio e alto}; e*
U = *[0, 100].*

Cada variável está relacionada a um conjunto de termos. Os termos da variável distância são curta, média e longa; os da variável esforço são pequeno, médio e grande; e os da variável custo são baixo, médio e alto. Cada variável será modelada por um conjunto nebuloso com o grau de pertinência definido por uma função do tipo trapezoidal.

Seguem os conjuntos nebulosos que representam os termos da variável linguística distância utilizando uma função de pertinência do tipo trapezoidal:

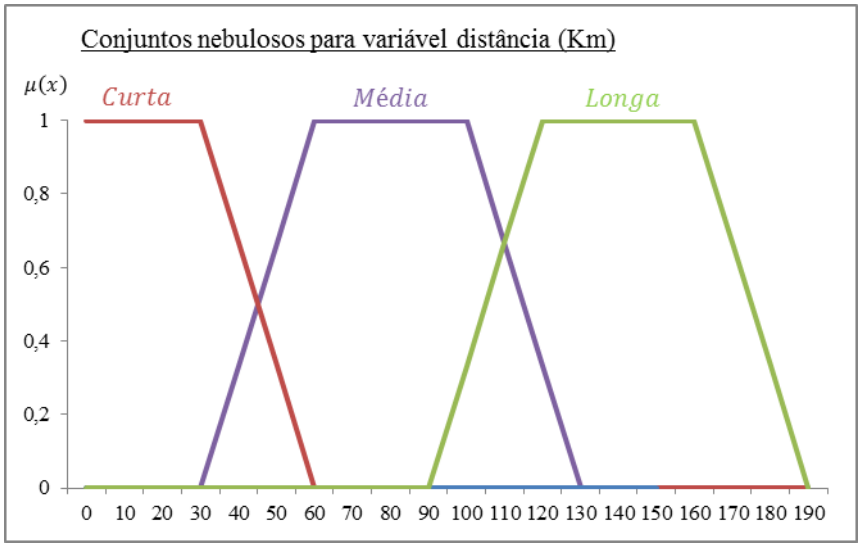


Figura 41: Conjuntos nebulosos para variável distância.

Seguem os conjuntos nebulosos que representam os termos da variável linguística esforço utilizando uma função de pertinência do tipo trapezoidal:

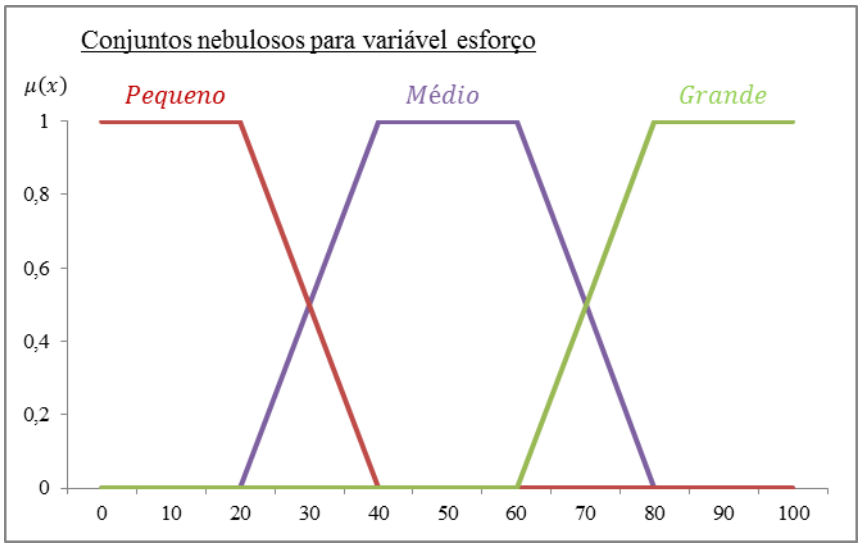


Figura 42: Conjuntos nebulosos para variável esforço.

Seguem os conjuntos nebulosos que representam os termos da variável linguística custo utilizando uma função de pertinência do tipo trapezoidal:

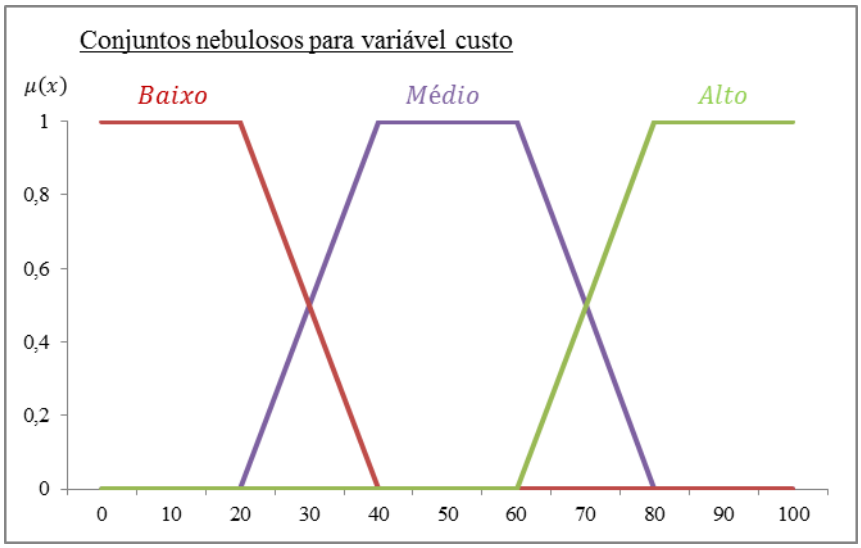


Figura 43: Conjuntos nebulosos para a variável custo.

Uma vez definidos os conjuntos nebulosos para os termos das variáveis linguísticas podem ser apresentadas as regras de inferência do modelo proposto nesse trabalho. As variáveis que representam a distância e o esforço irão compor os termos antecedentes e a variável que representa o custo irá compor o termo consequente de uma regra. O módulo de inferência tomará decisões para gerar uma classificação de rotas avaliando as medidas de distância e esforço com base numa sequência lógica de nove regras. Segue a lista de regras utilizada no módulo de inferência:

$$Se\ \mu(x_D) \in Curta\ e\ \mu(x_E) \in Pequeno\ então\ \mu(x_c) \in Baixo$$

$$Se\ \mu(x_D) \in Curta\ e\ \mu(x_E) \in Médio\ então\ \mu(x_c) \in Médio$$

$$Se\ \mu(x_D) \in Curta\ e\ \mu(x_E) \in Grande\ então\ \mu(x_c) \in Médio$$

$$Se\ \mu(x_D) \in Média\ e\ \mu(x_E) \in Pequeno\ então\ \mu(x_c) \in Baixo$$

$$Se\ \mu(x_D) \in Média\ e\ \mu(x_E) \in Médio\ então\ \mu(x_c) \in Médio$$

$$Se\ \mu(x_D) \in Média\ e\ \mu(x_E) \in Grande\ então\ \mu(x_c) \in Alto$$

$$Se\ \mu(x_D) \in Longa\ e\ \mu(x_E) \in Pequeno\ então\ \mu(x_c) \in Médio$$

$$Se\ \mu(x_D) \in Longa\ e\ \mu(x_E) \in Médio\ então\ \mu(x_c) \in Alto$$

$$Se\ \mu(x_D) \in Longa\ e\ \mu(x_E) \in Grande\ então\ \mu(x_c) \in Alto$$

Onde:

x_D é a variável linguística representando a distância;
x_E é a variável linguística representando o esforço;
x_C é a variável linguística representando o custo;
Curta, Média e Longa são conjuntos nebulosos representando termos de x_D;
Pequeno, Médio e Grande são conjuntos nebulosos representando termos de x_E; e
Baixo, Médio e Alto são conjuntos nebulosos representando termos de x_C.

A lista de regras pode ser representada também em formato tabular, conforme ilustrado abaixo:

Distância / Esforço	**Pequeno**	**Médio**	**Grande**
Curta	Baixo	Médio	Médio
Média	Baixo	Médio	Alto
Longa	Médio	Alto	Alto

Figura 44: Lista de regras de inferência.

Na tabela acima nota-se que o custo de uma rota é o resultado da relação entre a distância da rota (linhas da tabela) e o esforço empregado para trafega-la (colunas da tabela). Conforme as medidas de distância e de esforço são avaliadas, as regras de inferência descritas acima podem ser ativadas ou não.

Tomando como exemplo uma rota com distância total de 20 km e cujo esforço para trafegá-la seja de 10 pontos. Como 20 km resulta em um grau de pertinência maior que zero no conjunto nebuloso da variável linguística que representa a distância e 10 pontos de esforço resulta em um grau de pertinência maior que zero no conjunto nebuloso da variável linguística que representa o esforço, os termos antecedentes de uma regra são configurados com valor lógico igual a verdadeiro e consequentemente uma regra é ativada.

Cada par de antecedentes configurados com grau de pertinência maior que zero leva à ativação de uma regra de inferência e implicará na atribuição de um grau de pertinência para um conjunto nebuloso associado ao termo consequente dessa regra. No módulo de inferência do sistema proposto o grau de pertinência do conjunto nebuloso consequente é calculado por

uma função mínimo, cujo resultado é o menor valor dentre os graus de pertinência dos termos antecedentes da regra, conforme denotado abaixo:

$$\mu_{D\to E}(x_D, x_E) = Min[\mu_D(x_D), \mu_E(x_E)] = \mu_C(x_C)$$

Onde:

x_D *é a variável linguística do termo antecedente representando a distância;*
D é o conjunto nebuloso de x_D*;*
x_E *é a variável linguística do termo antecedente representando o esforço;*
E o conjunto nebuloso de x_E*; e*
x_C *é a variável linguística do termo consequente;*
C o conjunto nebuloso de x_C*.*

O resultado da função de mínimo é equivalente à interseção dos conjuntos nebulosos associados aos termos antecedentes de uma regra. Tomando como exemplo uma rota com distância igual a 45 km e assumindo a configuração dos conjuntos nebulosos utilizada no sistema proposto, pode-se representar o grau de pertinência da variável linguística representando a distância pelo gráfico abaixo:

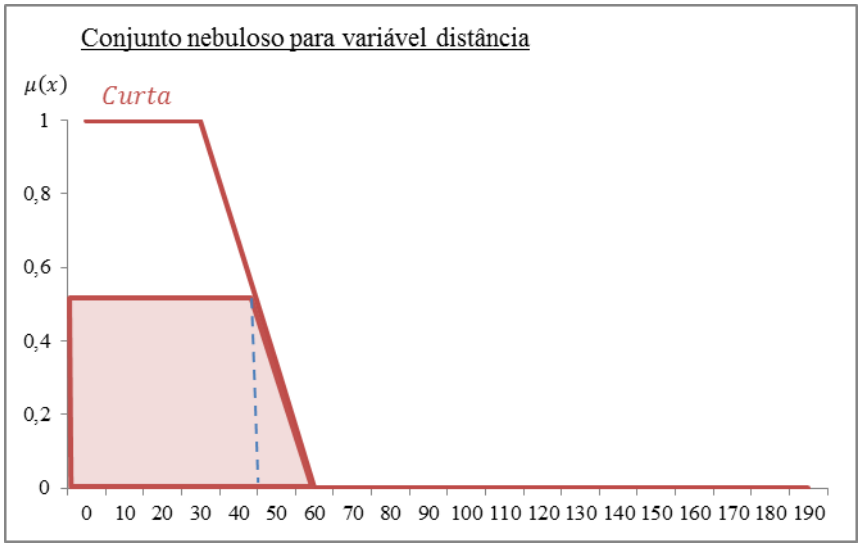


Figura 45: Grau de pertinência para distâncias curtas.

Nota-se no exemplo acima que apenas a variável linguística distância cujo termo é curta é considerada. Assumindo ainda nesse mesmo exemplo que para trafegar a referida rota sejam demandados 25 pontos de esforço, pode-se representar o grau de pertinência da variável linguística associada ao esforço pelo gráfico que segue:

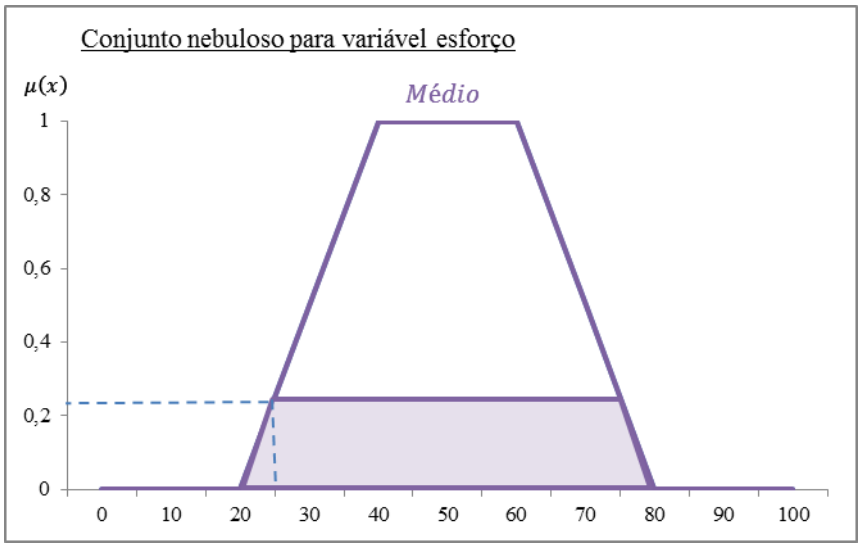


Figura 46: Grau de pertinência para esforço médio.

Aplicando as regras de inferência do sistema proposto sobre os antecedentes citados no exemplo acima é obtido o termo consequente cujo custo é médio. A função de mínimo determina o limite superior da variável linguística que representa o custo da rota, conforme ilustração abaixo:

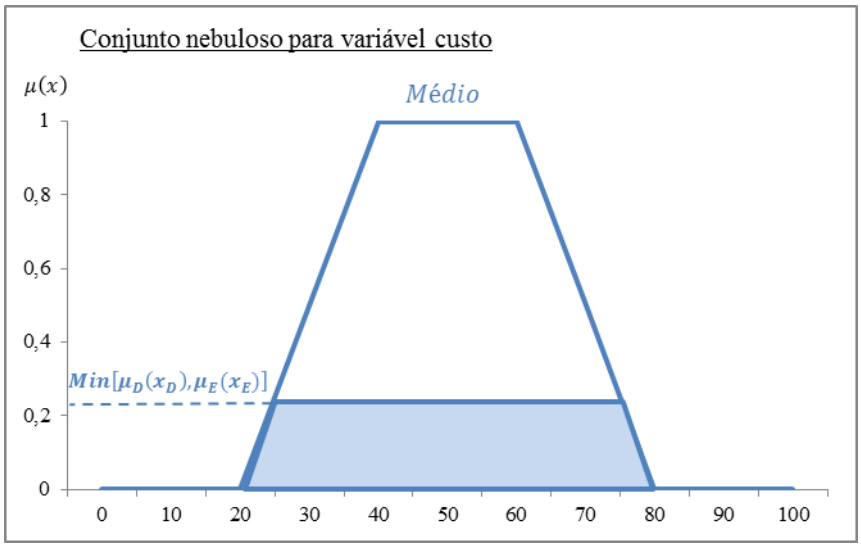


Figura 47: Limite superior de uma variável linguística consequente.

Nota-se no exemplo acima que, com distância igual a 45 km e esforço igual a 25 pontos, mais de uma regra de inferência será ativada. Isso porque uma distância de 45 km, pela configuração do sistema, não é interpretada apenas como curta, mas também como média. Assim como um esforço de 25 pontos não é interpretado apenas como médio, mas

também como pequeno. Portanto, quando a distância for 45 km e o esforço 25 pontos, todas as regras que seguem serão ativadas pelo sistema:

$$Se\ \mu(x_D) \in Curta\ e\ \mu(x_E) \in Pequeno\ então\ \mu(x_c) \in Baixo$$

$$Se\ \mu(x_D) \in Curta\ e\ \mu(x_E) \in Médio\ então\ \mu(x_c) \in Médio$$

$$Se\ \mu(x_D) \in Média\ e\ \mu(x_E) \in Pequeno\ então\ \mu(x_c) \in Baixo$$

$$Se\ \mu(x_D) \in Média\ e\ \mu(x_E) \in Médio\ então\ \mu(x_c) \in Médio$$

Onde:

x_D é a variável linguística representando a distância;
x_E é a variável linguística representando o esforço;
x_C é a variável linguística representando o custo;
Curta, Média e Longa são conjuntos nebulosos representando termos de x_D;
Pequeno, Médio e Grande são conjuntos nebulosos representando termos de x_E; e
Baixo, Médio e Alto são conjuntos nebulosos representando termos de x_C.

A lista de regras acima também pode ser representada em formato tabular. Segue uma ilustração com as regras ativadas em destaque:

Distância / Esforço	**Pequeno**	**Médio**	**Grande**
Curta	Baixo	Médio	Médio
Média	Baixo	Médio	Alto
Longa	Médio	Alto	Alto

Figura 48: Regras de inferências ativadas.

Em função da aplicação de mais de uma regra, mais de um conjunto nebuloso pode ter de ser avaliado para se calcular um valor determinístico para a saída do sistema. Por conta disso esse valor determinístico será gerado com base na área e no centro de gravidade dos termos consequentes das regras ativadas. A ilustração que segue demonstra o resultado obtido a partir da ativação de várias regras de inferência:

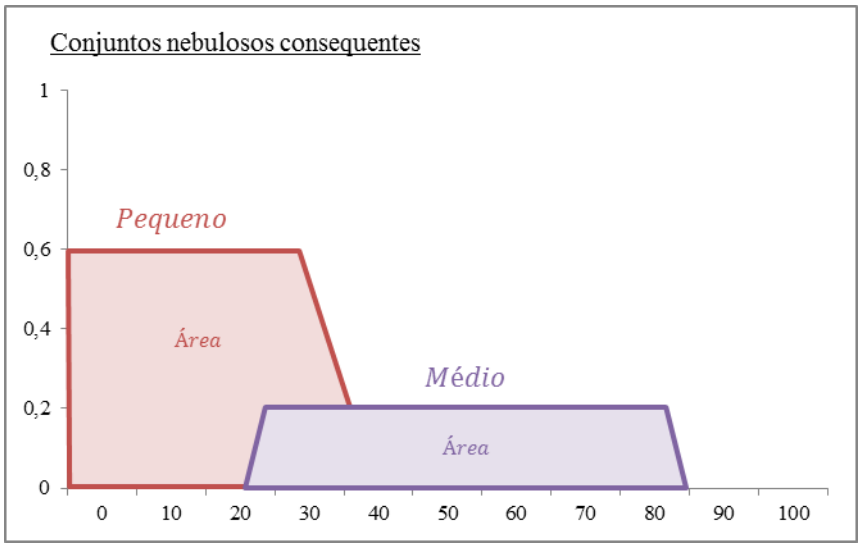


Figura 49: Áreas dos termos consequentes de regras.

Para cada regra ativada o sistema irá calcular a área da região limitada pela função de mínimo sobre os conjuntos nebulosos dos termos da variável linguística associada ao custo de uma rota. A medida de uma área formada sobre uma variável linguística será multiplicada pela medida de seu centroide. Além disso, o sistema também irá calcular a medida da área total formada (soma das áreas de todos os termos). O resultado para a saída do sistema proposto será dado em função do centroide e da área de cada região formata, conforme expresso abaixo:

$$custo = \frac{\sum centroide}{\sum área}$$

Onde:

centroide é o centro de massa de uma região formada por um termo consequente;
área é a medida da área de uma região formada por um termo consequente; e
custo e um valor determinístico obtido a partir de conjuntos nebulosos.

A expressão acima descreve o método *defuzzifier* utilizado pelo módulo de inferência nebuloso do sistema Híbrida. Com isso, conclui-se a explicação conceitual do método computacional empregado para gerar uma classificação de rotas com base nas medidas de distância e esforço coletados durante a exploração de rotas feitas por agentes.

2.6.**Comentários**

Nesse capítulo foram explicados detalhes de como os fundamentos teóricos de sistemas baseados em agentes, dos algoritmos de planejamento de rotas e dos sistemas de inferência nebulosos foram combinados para elaborar uma solução para a classificação de rotas em espaços com obstáculos. Também foram apresentados detalhes a respeito do uso de mapas digitais e de funcionalidades da ferramenta NetLogo para compor o ambiente de desenvolvimento da solução.

Conclui-se com os estudos realizados que o NetLogo pode ser utilizado não apenas para implementar os algoritmos de planejamento de rotas através de uma programação baseada em agentes, mas também pode ser estendido para trabalhar com informações estruturadas em mapas digitais. Além disso, também é possível acoplar ao NetLogo extensões para integrá-lo a um gerenciador de banco de dados ou para executar quaisquer outras funções necessárias (como funções de um módulo de inferência nebuloso).

Dessa forma, o NetLogo passou a ser o cerne de um sistema capaz de ler mapas digitais com distribuições de esforço, disparar agentes orientados por algoritmos de planejamento de rotas, registrar informações de esforço e de distância a partir de explorações de rotas, acionar um módulo de inferência nebuloso para classificação de rotas e apresentar as classificações obtidas em um mapa digital.

A arquitetura proposta e o ambiente para o desenvolvimento da solução apresentados nesse capítulo garantem uma grande diversidade de soluções. Como a elaboração de mapas digitais pode ser feita desacoplada do sistema, bastando apenas que sejam definidas feições de polígonos contendo a distribuição de esforço, é possível trabalhar com diferentes situações e com diferentes áreas de estudo, sendo necessário apenas definir o mapa adequado na entrada do sistema.

Além da facilidade de definir diferentes mapas na entrada do sistema, as funcionalidades do NetLogo permitem que uma aplicação seja configurada de diferentes maneiras a cada execução, o que contribui ainda mais para o estudo de um problema. Combinando essas funcionalidades com a possibilidade de se executar mais de um algoritmo de planejamento de rotas será possível explorar uma grande quantidade de caminhos sobre o mapa em estudo, o que ajudará a simular diversas possibilidades eficientes.

O modelo de inferência nebuloso proposto também funcionará com um nível baixo de acoplamento. Isso facilitará a adaptação de regras e de conjuntos nebulosos para abordar novos problemas. O fato do módulo de inferência nebuloso estar integrado a um gerenciador

de banco de dados permitirá não apenas que os resultados das inferências sejam melhor armazenados, mas também que a própria configuração da base de regras seja mais flexível às mudanças.

Dessa forma, a flexibilidade da arquitetura proposta combinada com as funcionalidades disponíveis no NetLogo viabiliza uma implementação adequada para a solução proposta ao problema de classificação de rotas em espaços com obstáculos.

3. **IMPLEMENTAÇÃO**

Esse capítulo explica de que forma a solução sistêmica arquitetada para o problema de seleção de rotas com obstáculos foi desenvolvida, quais os métodos e recursos de programação foram utilizados para criar um sistema real e capaz de colocar à prova a combinação de conceitos apresentada nos capítulos anteriores.

3.1.**Apresentação**

Para tornar real uma solução esboçada na arquitetura de um sistema é preciso descrever com detalhes uma série de artefatos responsáveis pela execução dos processos e serviços planejados para resolver um problema. No caso específico desse trabalho, ficou definida uma arquitetura sistêmica que combina técnicas de sistemas baseados em agentes, mapas digitais e um sistema de inferência nebuloso para resolver problemas de exploração de rotas. Agora se faz necessário o detalhamento dos componentes, fluxos de dados e estruturas de dados para implementar a solução proposta.

Na arquitetura apresentada nesse trabalho ficou explícita a necessidade de uso de um gerenciador de banco de dados para persistir informações de configuração ou para simplesmente armazenar dados utilizados para gerar resultados. Portanto, nesse capítulo serão explicados detalhes sobre o modelo de dados integrado ao sistema proposto.

Também serão detalhados os aspectos estáticos e dinâmicos da solução proposta, com as especificações dos fluxos de processamento implementados no NetLogo e nas extensões necessárias para trabalhar com mapas digitais e com um modelo de inferência nebuloso. Por fim, serão explicados os controles utilizados na interface de entrada do sistema com o objetivo de ampliar a aplicabilidade da arquitetura proposta.

3.2.**Banco de dados**

A arquitetura do sistema proposto, tanto nas rotinas de exploração de rotas quanto no módulo de inferência nebuloso, é fortemente integrada a um sistema gerenciador de banco de dados (SGBD). Nesse trabalho foi utilizado o SGBD mySql versão 6.0.0 (MySQL, 2012). O modelo de banco de dados empregado nesse trabalho deverá organizar informações de três processos distintos:

a)Informações utilizadas na exploração de rotas;

b)Configuração do módulo de inferência nebuloso; e

c)Resultados.

Para organizar as informações de exploração de rotas e de resultados foram criadas duas tabelas, conforme apresentado na relação abaixo:

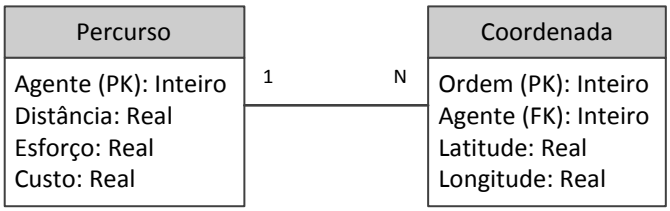


Figura 50: Relação entre percurso e coordenada.

Nota-se que os registros da tabela Percurso são identificados de forma única pelo campo Agente (*primary key*) e que na tabela Coordenada os registros são identificados pelo campo Ordem. Além disso, os registros da tabela Coordenada estão relacionados aos registros da tabela Percurso pelo campo Agente (*foreign key*).

Conforme um agente executa a exploração de rotas ele registra as coordenadas dos pontos sobre o mapa por onde trafega. O identificador do agente, a latitude e longitude dos pontos por onde ele passa e a ordem em que passa por cada ponto são valores armazenados na tabela Coordenada. Ao final de um percurso, quando atinge seu ponto de destino, um agente registra as informações de distância e esforço, contabilizadas durante sua trajetória, na tabela Percurso. O custo de um percurso é calculado e armazenado durante o processo de classificação de rotas.

O relacionamento entre as tabelas Percurso e Coordenadas, definido pelo identificador de um agente, permite consultar trajetórias realizadas por um agente a partir de registros da tabela Percurso. Em outras palavras, é possível selecionar um percurso de um agente e relacionar todas as coordenadas associadas a esse percurso. Essa possibilidade de consulta é particularmente útil na apresentação dos resultados da classificação de rotas, onde um pequeno grupo de agentes com percursos classificados (ou rotas com custo baixo) é apresentado no mapa.

Dessa forma, a tabela de coordenadas deverá armazenar os passos dos agentes durante a exploração de rotas, enquanto a tabela de percurso deverá armazenar as medidas de distância e esforço. As medidas de distância e esforço são então consultadas pelo módulo de

inferência nebuloso para classificação de rotas. Além dessas tabelas que armazenam informações coletadas por agentes, são necessárias estruturas para definir o funcionamento e a base de regras do módulo de inferência. Essas estruturas do módulo de inferência devem organizar as seguintes informações:

a)Lista de variáveis linguísticas do módulo de inferência nebuloso;
b)Lista de termos para cada variável linguística;
c)Definição das funções de pertinência para cada termo de uma variável linguística; e
d)Configuração das regras de inferência com seus termos antecedentes e consequentes.

Para organizar a lista de variáveis linguísticas foi criada uma tabela chamada Variável contendo o nome e o universo de discurso de cada variável. Essa tabela está relacionada a outra chamada Termo, contendo os termos de cada variável linguística utilizada. Segue uma ilustração com a relação das tabelas Variável e Termo:

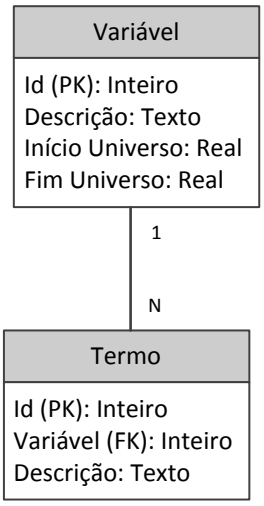


Figura 51: Relação entre variáveis linguísticas e termos.

Nota-se pela ilustração acima que cada variável com seus atributos é identificada de forma única em sua tabela pela coluna Id. Assim como cada termo é identificado em sua tabela pela coluna Id. Portanto, vários termos podem estar relacionados a uma única variável através da coluna chamada Variável pertencente à tabela Termo.

Cada termo de uma variável linguística é definido por um conjunto nebuloso, cuja função de pertinência é descrita nesse trabalho por um conjunto de segmentos de reta. Portanto, cada termo de uma variável linguística está relacionado na base de dados a um

conjunto de segmentos de reta, que combinados descreverão uma função de pertinência de um conjunto nebuloso.

Cada segmento de reta é descrito por um registro contendo os campos: coeficiente angular, coeficiente linear, intervalo no eixo das abcissas e intervalor no eixo das ordenadas. Segue uma ilustração com a relação entre termos e segmentos de reta de um conjunto nebuloso:

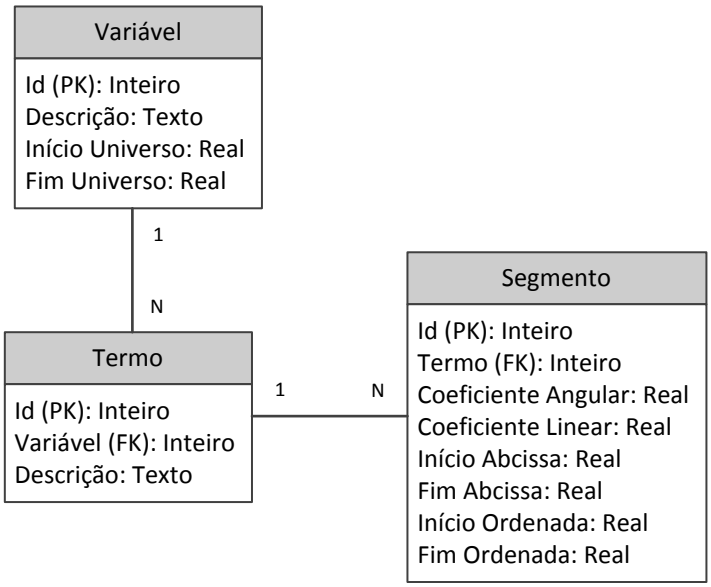


Figura 52: Relação entre termos e conjuntos nebulosos.

Nota-se que cada segmento de reta da tabela Segmento está relacionado a um único termo através da coluna Termo da tabela Segmento. Dessa forma, apesar de nesse trabalho terem sido utilizadas funções de pertinência do tipo trapezoidal, seria possível com o modelo de dados apresentado anteriormente associar a um termo um conjunto de segmentos de reta que combinados gerassem um formato aproximado aos de outras funções de pertinência, tais como: sigmoidal e gaussiando.

Independente do formato dos conjuntos nebulosos, as relações entre as variáveis linguísticas e seus termos são regidas no módulo de inferência por uma base de regras. Basicamente, são regras que definem relações entre termos antecedentes e consequentes no processo de inferência. Para resolver essas relações foram criadas duas tabelas: uma tabela chamada Antecedente da Regra e outra chamada Consequente da Regra. A tabela Antecedente relaciona dois termos através de uma regra enquanto a tabela Consequente relaciona um termo consequente a uma regra. Segue uma ilustração com as relações entre termos antecedentes e consequentes de regras de inferência:

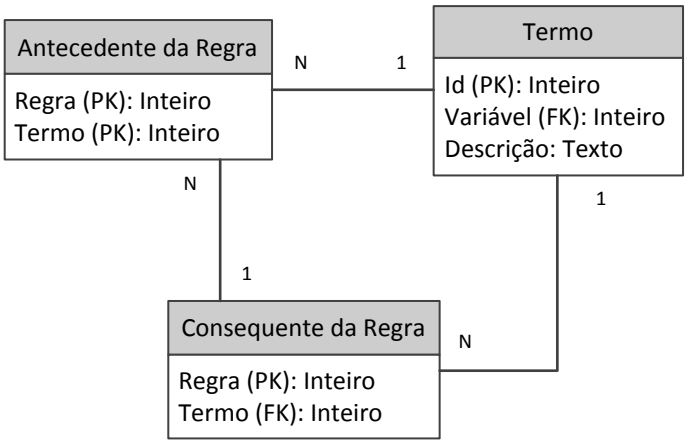


Figura 53: Relação entre termos antecedente e consequente.

Nota-se que um termo pode ser utilizado como antecedente em várias regras. Assim como pode ser utilizado como consequente em várias regras. Entretanto, cada regra terá apenas um único termo consequente. Essa estrutura garante que mais de um termo antecedente seja combinado para implicar em apenas um termo consequente.

Com os esquemas apresentados nessa seção conclui-se a descrição do modelo conceitual de banco de dados integrado à arquitetura de sistema proposta nesse trabalho. Segue uma ilustração com o modelo de dados completo:

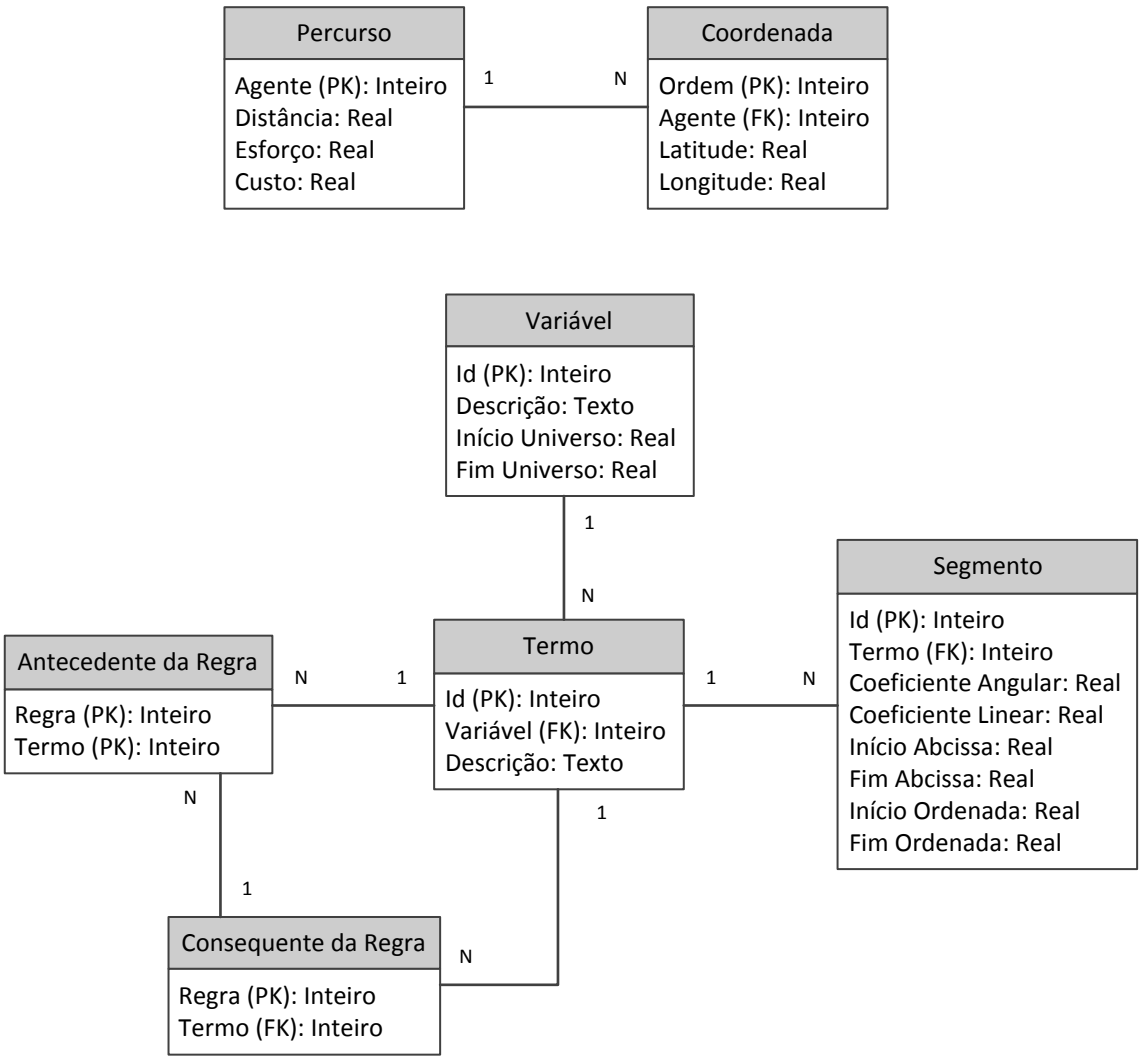


Figura 54: Modelo de dados proposto.

A partir do modelo de dados apresentado anteriormente, as rotinas de programação criadas no ambiente do NetLogo e no módulo de inferência nebuloso poderão organizar todas as informações obtidas na exploração rotas e todas as configurações necessárias para classificação dessas rotas.

3.3.**Estrutura**

Nessa seção serão explicadas as rotinas de programação criadas no NetLogo e as extensões utilizadas para compor a arquitetura do sistema proposto. Para descrever o funcionamento dessas rotinas de uma forma conceitual serão utilizados os seguintes diagramas da UML (*Unified Modeling Language*):

a)Diagrama de implantação;
b)Diagrama de atividades;
c)Diagrama de transição de estados;
d)Diagrama de classes; e
e)Diagrama de sequência.

A arquitetura básica do sistema proposto nesse trabalho é suportada pelo ambiente de desenvolvimento do NetLogo e suas funcionalidades de extensão. Para manipular mapas digitais foi necessário utilizar uma extensão GIS (*Geographic Information System*) e para processar as classificações de rotas foi construída uma extensão *fuzzy*. A arquitetura básica da solução proposta é apresentada na ilustração abaixo através de uma notação UML:

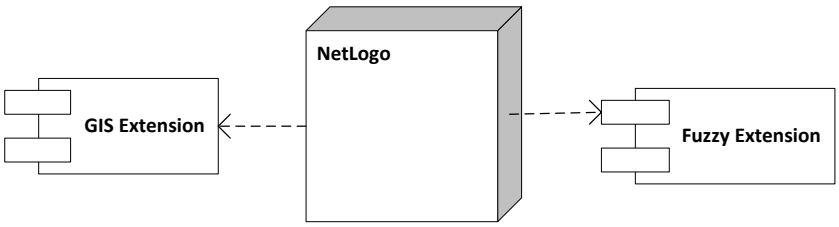


Figura 55: Diagrama de implantação da solução.

Nota-se no diagrama de implantação apresentado acima que existe um módulo de processamento principal que executa no próprio ambiente de programação do NetLogo e que as extensões de GIS e de lógica nebulosa são utilizadas por esse módulo principal. De fato as extensões são componentes acoplados ao ambiente de programação do NetLogo que passam a prover funcionalidades através de suas interfaces.

A extensão GIS é fornecida no pacote de instalação do NetLogo, versão 4.4.1. As principais funções e procedimentos da extensão GIS utilizadas nesse trabalho são:

a)*load-dataset*: para carregar um arquivo do tipo *shapefile*;
b)*apply-coverage*: para transferir valores de atributos do mapa para *patches*;
c)*find-one-feature*: para encontrar um polígono através do valor de um atributo;
d)*centroid-of*: para encontrar o centroide de um polígono;
e)*find-range*: para encontrar polígonos com um atributo entre dois valores;
f)*contains*: para verificar se um ponto está contido em um polígono; e
g)*intersects*: para verificar se ocorre a interseção entre polígonos;

O módulo principal deve executar um fluxo de atividades que inicia no processamento de configurações básicas, tais como: definição do diretório de um mapa digital, ponto de origem, destino e características de funcionamento dos agentes. Após essas configurações básicas, o fluxo de execução segue com a identificação de obstáculos, o planejamento de rotas e a exploração de caminhos no mapa. Por fim, as rotas devem ser classificadas e os resultados apresentados em um mapa. O fluxo de processamento do módulo principal é apresentado na ilustração abaixo através de um diagrama de atividades da UML:

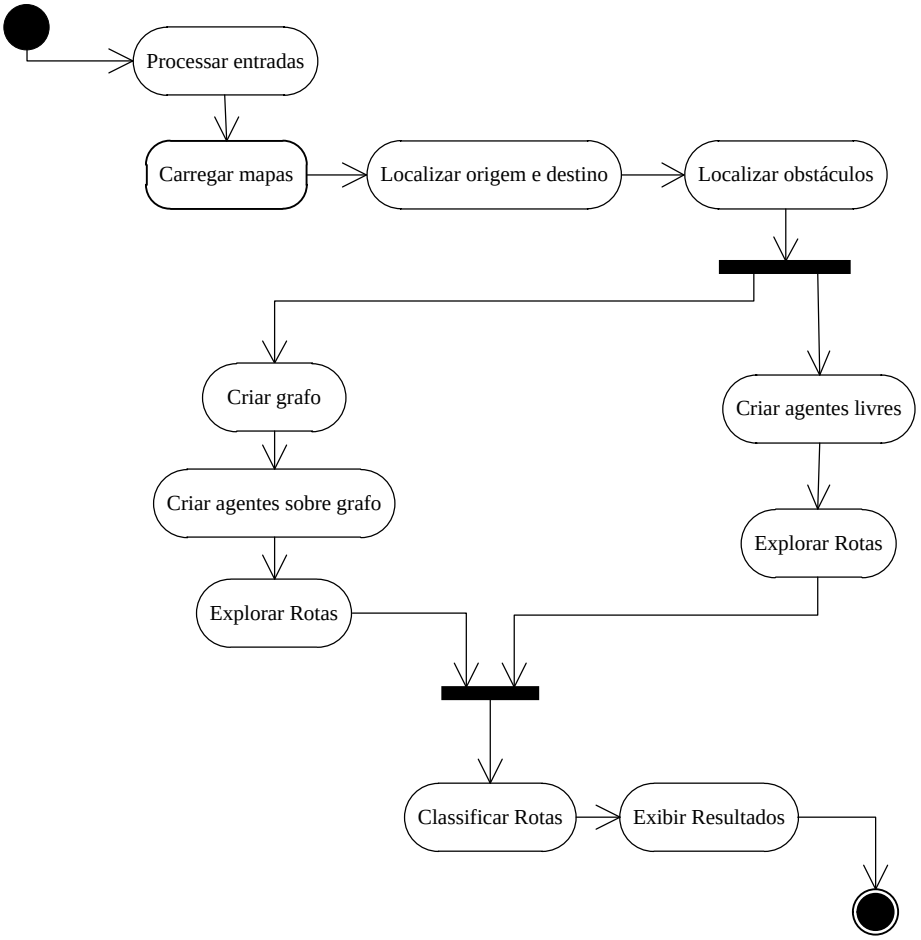


Figura 56: Diagrama de atividades da solução.

As entradas essenciais do sistema são a localização (diretório) do mapa digital que define o terreno em estudo e os pontos de origem e destino para os agentes. Mas, também são possíveis algumas configurações relacionadas às características dos agentes, tais como: distância em que um agente detecta um obstáculo, ângulo de desvio para evitar um obstáculo e tempo para alinhamento com o ponto de destino. Também será possível definir na entrada

do sistema a quantidade de rotas selecionadas a partir da classificação final e o intervalo de tempo para registrar coordenadas durante um percurso. Esses parâmetros serão fornecidos através de formulários construídos a partir de componentes de interface do próprio NetLogo.

A partir da definição do diretório de um arquivo *shapefile*, um mapa digital será carregado no ambiente do NetLogo através do seguinte comando:

set gb-map gis:load-dataset gb-map-path

Onde:

gis:load-dataset é uma função do pacote de extensão GIS;
gb-map é o objeto com o mapa digital carregado; e
gb-map-path é a variável que contém o caminho de um shapefile.

Quando um mapa e carregado, além dos polígonos também são consultadas e distribuídas as medidas de esforço sobre ambiente de execução do NetLogo. Essas informações são essenciais para o funcionamento do sistema. O mapa carregado em memória é utilizado pelos agentes para coletar os níveis de esforço demandados durante o percurso das rotas exploradas. Níveis de esforço maiores ou iguais a 100 pontos são caracterizados como obstáculos (áreas intransponíveis) no mapa e serão contornados pelos agentes.

O ambiente de execução do NetLogo é dividido em pequenos agentes retangulares chamados *patches*. A área total do ambiente e a fração ocupada por cada *patch* em um ambiente são configuradas através de funcionalidades do NetLogo pelo usuário. Ao atribuir as medidas de esforço de cada polígono de um mapa ao conjunto de *patches* de um ambiente de execução o resultado é algo similar ao apresentado na ilustração abaixo:

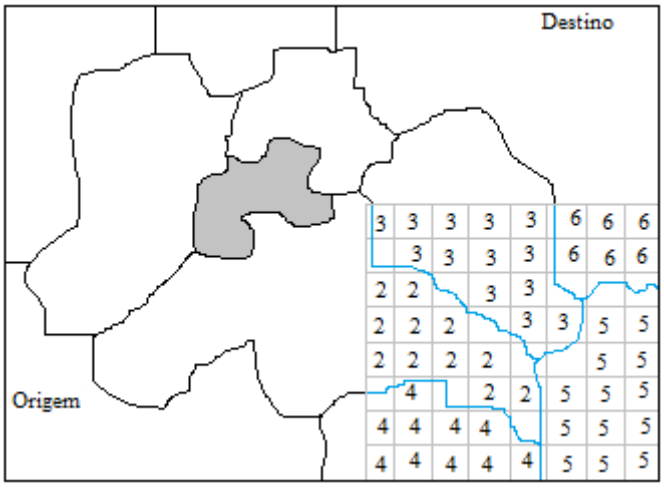


Figura 57: Distribuição de esforço em patches.

Nota-se na ilustração anterior que um *patch* contorna o perímetro de um polígono com certa imprecisão. Com isso, pode-se ter a sensação de um polígono receber a medida de esforço de outro. Esse problema pode ser minimizado reduzindo-se o tamanho de um *patch*. Essa solução aumenta a precisão da área ocupada por um *patch*, mas aumenta também a quantidade de *patches* do ambiente e, consequentemente, demanda maior capacidade de processamento.

A transferência das medidas de esforço dos polígonos de mapas para *patches* do ambiente de execução do NetLogo pode ser feita através do comando abaixo:

gis:apply-coverage gb-map gb-map-effort po-effort

Onde:

gis:apply-coverage é uma função do pacote de extensão GIS;
gb-map é o objeto com o mapa digital carregado;
gb-map-effort é o nome da coluna da tabela relacionada ao mapa contendo o esforço; e
po-effort é o nome do atributo do objeto patch que receberá a medida de esforço.

Após a atribuição das medidas de esforço ao ambiente de execução, os obstáculos podem ser identificados simplesmente consultando-se o atributo que indica o esforço alocado em um *patch*. Com relação aos pontos de origem e destino, serão criados dois agentes de posições fixas, situados nos centroides dos polígonos indicados no formulário de entrada. Feitas essas configurações, pode-se iniciar as ações para executar a exploração de rotas.

Nota-se no diagrama de atividades apresentado para o módulo principal (Figura 56: Diagrama de atividades da solução.) que, após o sistema carregar um mapa, definir obstáculos e identificar as posições de origem e destino, são processados dois caminhos paralelos: um com a criação de uma família de agentes direcionados por grafos e outro com a criação de uma família de agentes independentes de grafos. Os agentes direcionados por grafos são criados de acordo com os métodos de planejamento de rotas por *roadmaps* e por decomposição celular. Já os agentes livres realizam desvios baseados no planejamento de rotas por campo potencial.

Para que os agentes direcionados por grafos executem devem ser definidos os vértices que irão compor o grafo de visibilidade por onde eles irão trafegar. Esses vértices são agentes fixos, situados nos centroides de polígonos do mapa. Estão disponíveis no sistema três ações para se estabelecer os vértices de um grafo de visibilidade:

a)Definir vértices a partir dos centroides de polígonos vizinhos aos obstáculos;

b)Definir vértices a partir dos centroides de polígonos de origem e destino; e

c)Definir vértices a partir dos centroides de todos os polígonos do mapa, exceto de polígonos de obstáculos.

As duas primeiras formas são incrementais e podem funcionar combinadas. Na realidade, o objetivo de se criar vértices a partir dos centroides da vizinhança dos pontos de origem e de destino é prover algumas possibilidades de caminhos a mais. Em alguns casos esses vértices a mais propiciam melhorias na exploração de rotas, conforme se pode observar na ilustração abaixo:

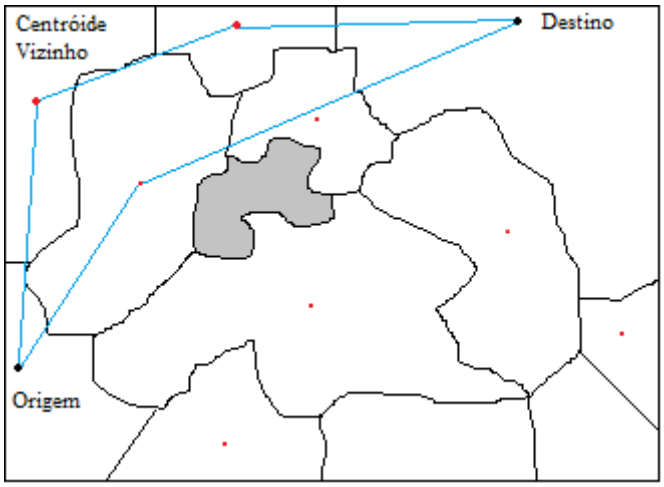


Figura 58: Vértices a partir de centroides na vizinhança da origem.

No caso da terceira opção, definir vértices a partir dos centroides de todos os polígonos do mapa, trata-se de uma exploração com uma diversidade muito maior de rotas e cujo objetivo é, além de abranger mais áreas do mapa, diminuir a influência do posicionamento de obstáculos, dado que os vértices serão criados em suas vizinhanças. Em outras palavras, dependendo do posicionamento dos obstáculos tem-se uma alternativa para manter uma distribuição mais adequada de caminhos explorados. Entretanto, essa alternativa implica normalmente em uma demanda de processamento muito maior.

Os vértices de um grafo de visibilidade podem ser criados no ambiente de execução do NetLogo através do comando abaixo:

create-vertex 1 [set xcor item 0 centroid set ycor item 1 centroid]

Onde:

create-vertex é um comando do NetLogo para se criar agentes;
vertex define a família de agentes criada com o comando breed;
xcor e ycor são as coordenadas do agente criado; e
item 0 centroid e item 1 centroid as coordenadas do centroide de um polígono.

Definidos os vértices dos grafos de visibilidade, as arestas podem ser estabelecidas ligando-se cada vértice criado a todos os demais, exceto nos casos em que uma aresta leva a um vértice mais próximo da origem do que do destino. Em outras palavras, as arestas podem ser criadas de forma a propiciar caminhos que se afastam da origem e se aproximam do destino. Essa abordagem tem como objetivo ligar o máximo de vértices possíveis evitando caminhos menos eficientes, conforme demonstra a ilustração abaixo:

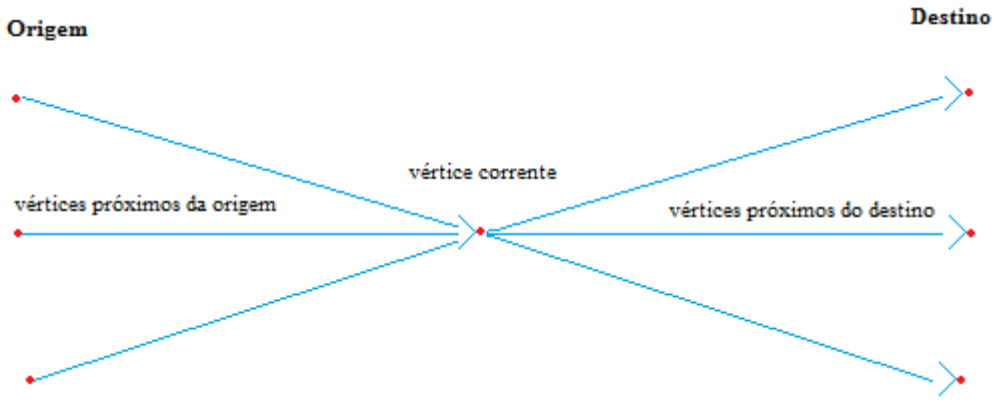


Figura 59: Arestas do ponto de origem para o destino.

As arestas do grafo de visibilidade serão criadas como agentes do tipo *link*, relacionando dois agentes da família de vértices. Seguem os comandos para criação de arestas no NetLogo:

ask vertex [create-edges-from other vertex with [vo-source < [vo-source] of myself and vo-dest > [vo-dest] of myself]]
ask vertex [create-edges-to other vertex with [vo-source > [vo-source] of myself and vo-dest < [vo-source] of myself]]

Onde:

ask vertex envia o comando de criação de arestas para todos os vértices;
create-edges-from cria arestas do um vértice corrente para os demais;
create-edges-to cria arestas de todos os vértices para o vértice corrente;
with define um critério para a seleção dos vértices;
vo-source < [vo-source] of myself seleciona os vértices mais distantes da origem; e
vo-dest > [vo-dest] of myself]]seleciona os vértices mais próximos do destino.

A partir das arestas estabelecidas, os agentes direcionados por grafos poderão se multiplicar e se distribuir para percorrer todas as possibilidades de caminhos possíveis sobre o grafo criado. Nessa abordagem é criada na origem uma quantidade de agentes igual à quantidade de arestas do vértice de origem. Cada agente seguirá por sua aresta até encontrar um vértice, aonde irá se multiplicar em igual número à quantidade de arestas menos um (ou seja, à quantidade de vértices de destino). Esse procedimento se repetirá até o destino final de um agente, conforme demonstra a ilustração abaixo:

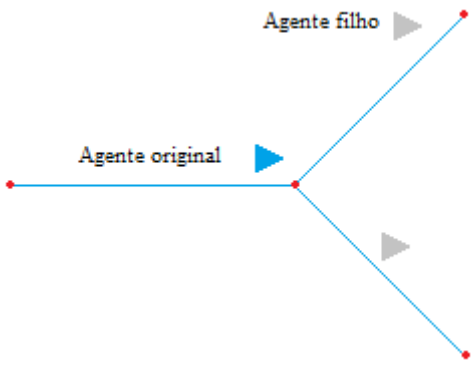


Figura 60: Multiplicação de agentes no vértice de um grafo.

Portanto, o ciclo de vida de um agente direcionado por grafo pode ser resumido pelas ações: se direcionar para um vértice de destino (podendo, inclusive, esse vértice ser o destino final), verificar colisões com obstáculos[8], verificar novas arestas e se multiplicar. Ao se multiplicar um agente encerra suas atividades, deixando para seus filhos a tarefa de encontrar o ponto de destino final. Evidentemente, todas as informações coletadas são passadas por herança aos agentes gerados.

O diagrama de transição de estados que segue ilustra o ciclo de vida de um agente direcionado por grafo (nota-se que para cada agente gerado deve-se considerar uma nova execução do diagrama):

[8] As colisões são situações possíveis porque as possibilidades de rodas, diferentemente dos casos de planejamentos por *roadmaps* clássicos, aqui podem ser criadas sem depender dos obstáculos, como é o caso da rota gerada diretamente do vértice de origem para o vértice de destino.

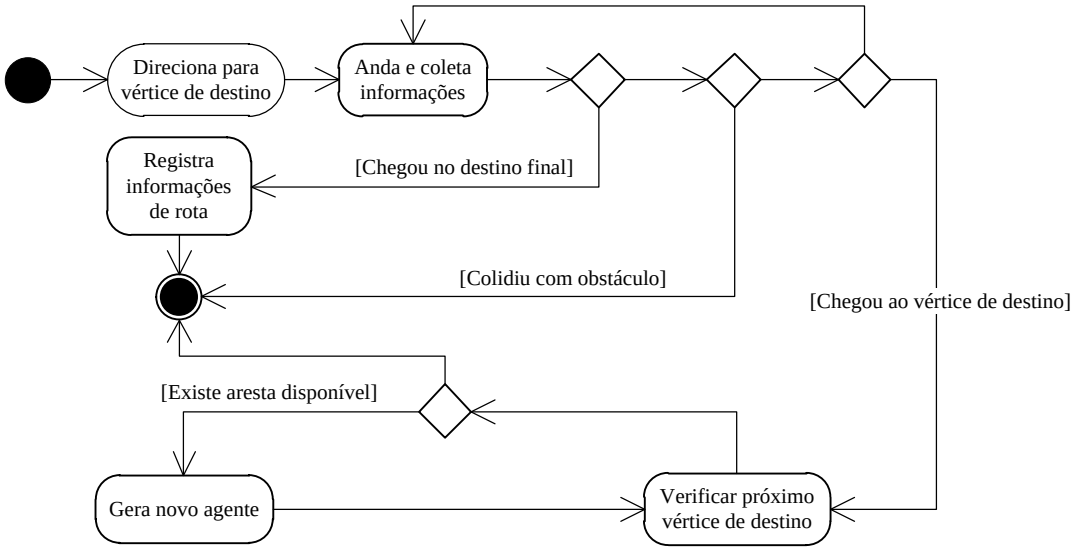


Figura 61: Transição de estados de agentes direcionados por grafos

Para direcionar-se para o vértice de destino um agente deve consultar a inclinação formada entre seus vértices de origem e destino. Isso pode ser feito no NetLogo através do comando abaixo:

ask agents [set heading ([link-heading] of edge who ([who] of ao-next-vertex))]

Onde:

ask agents solicita a execução de um comando para um grupo de agentes;
set heading atribui um valor para o atributo heading;
[link-heading] of edge acessa o atributo link-heading de um agente do tipo link; e
edge who ([who] of ao-next-vertex) é a aresta formada por um agente e seu próximo vértice.

A coleta de informações pode ser feita por um agente simplesmente acessando os atributos dos *patches* do ambiente de execução, o registro de informações de rotas é feito através da extensão *fuzzy* e o controle de vértices de destino é feito utilizando atributos do próprio agente. Por fim, a multiplicação de agentes através das arestas que os levam os aos vértices de destino pode ser feita utilizando o comando abaixo:

ask agent [hatch 1 [set ao-next ([end2] of my-out-link) create-edge-to ao-next]]

Onde:

ask agent solicita a execução de um comando para um grupo de agentes;
hatch 1 gera um agente filho com os mesmos atributos do pai;
set ao-next ([end2] of my-out-link) configura destino do filho com o fim da aresta;
[end2] of my-out-link é o vértice final de uma aresta de saída; e
create-edge-to ao-next cria um link do filho com seu vértice de destino.

Enquanto os agentes direcionados por grafos realizam desvios orientados pelas arestas de um grafo de visibilidade os agentes livres desviam apenas ao interagir com um obstáculo. Essa interação se dará por uma força potencial que age a partir de certa distância e que altera a direção do agente com certa intensidade. A distância de interação e o ângulo de redirecionamento são definidos pelo usuário através do formulário de entrada do sistema.

Ao atingir a distância limite para perceber um obstáculo um agente irá se multiplicar de forma a explorar direções possíveis a partir do ângulo de redirecionamento imposto pelo obstáculo. Em outras palavras, um agente exposto a um campo potencial de um obstáculo cuja intensidade implica em um desvio de 90º poderá se duplicar em duas direções opostas (ou seja, de sentidos contrários). A quantidade de agentes gerados a partir de uma interação com um obstáculo também é definida no formulário de entrada do sistema.

Cada agente gerado seguirá seu caminho e atualizará sua rota de acordo com a ação potencial exercida pelo ponto de destino final, conforme ilustrado abaixo:

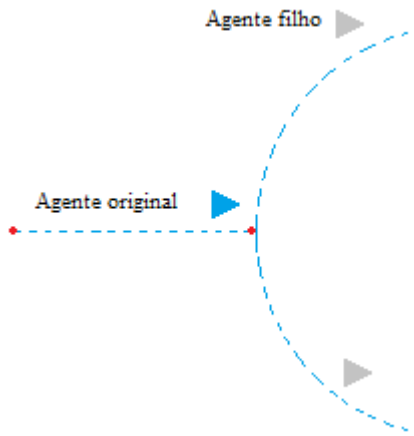


Figura 62: Multiplicação de agentes por campo potencial.

O ciclo de vida de um agente livre inicia com a definição de seu ângulo de partida. A partir do seu ângulo inicial um agente realiza ajustes contínuos em sua direção em função da posição de seu ponto de destino. Conforme caminha em direção ao ponto de destino um agente coleta informações sobre seu ambiente. Ao interagir com um obstáculo, novos agentes são gerados, conforme demonstra o diagrama de transição de estados que segue:

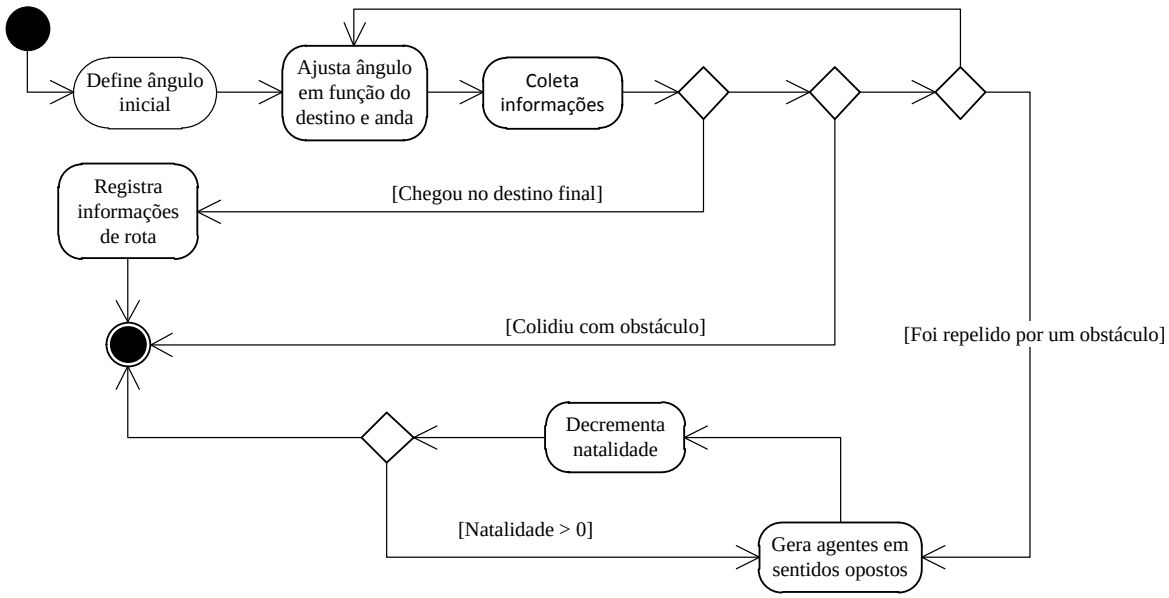


Figura 63: Transição de estados de agentes livres.

Assim como no caso dos agentes direcionados por grafo, os agentes livres registram suas informações de rotas utilizando a extensão *fuzzy*, coletam informações do ambiente acessando os atributos de *patches* e utilizam agentes do tipo *link* para controlar suas direções em função dos obstáculos e do ponto de destino. Cada nova geração de agentes deve executar o diagrama acima do início.

Tanto os agentes direcionados por grafo quanto os agentes livres executam com o objetivo de explorar possibilidades de rotas coletando informações de distância e de esforço para o módulo de inferência nebuloso. Portanto, após a execução desses agentes o sistema poderá executar a classificação das rotas exploradas por eles. Para integrar ao NetLogo as rotinas para classificação de rotas o módulo de inferência nebuloso foi construído utilizando uma abordagem baseada em componentes.

Como o NetLogo permite que novos comandos sejam incorporados ao seu ambiente de programação utilizando a linguagem Java (Oracle, 2012), foi gerado um componente com extensão JAR (*Java Archive*), compactando o conjunto de classes que implementa o modelo de inferência nebuloso proposto nesse trabalho. Para implementar as classes do modelo proposto e gerar o arquivo de extensão JAR foi utilizado o ambiente de desenvolvimento integrado do Eclipse (Eclipse, 2012), uma ferramenta de desenvolvimento *open source*.

O funcionamento básico do módulo de inferência nebuloso é descrito pelo diagrama de atividades que segue:

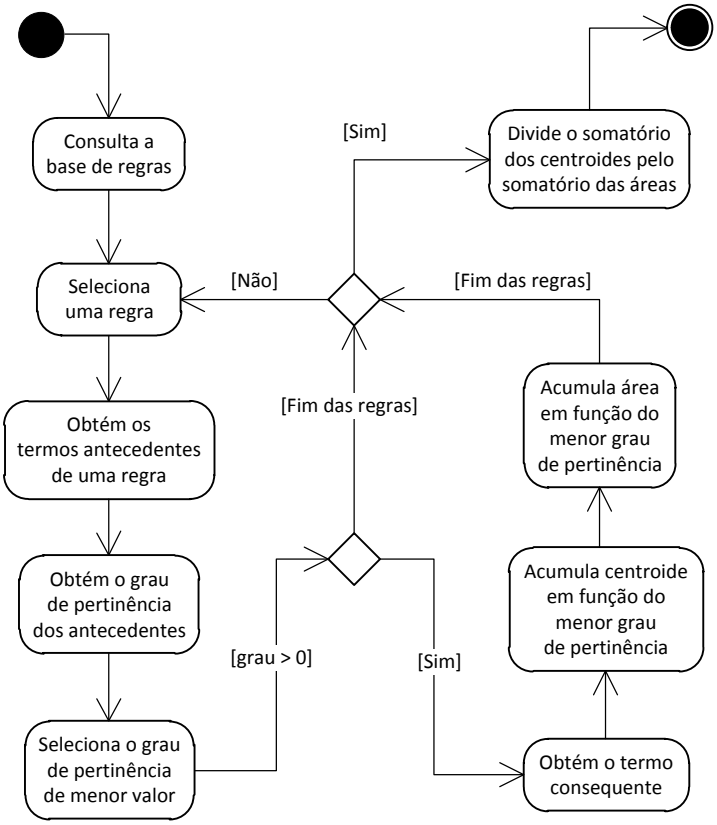


Figura 64: Diagrama de atividades do módulo de inferência.

Nota-se no diagrama acima que a base de regras do modelo de inferência é consultada regra a regra. Cada termo antecedente de uma regra é avaliado e quando o antecedente de menor grau de pertinência tem um valor maior que zero a regra é ativada. A partir das regras ativadas determina-se o termo consequente e calcula-se o centroide e a área do gráfico cujo ponto de máximo é igual ao ponto de mínimo obtido dos antecedentes. Cada resultado obtido com a ativação de uma regra é acumulado e após todas as regras serem avaliadas o resultado final é obtido pela razão entre o somatório dos centroides e o somatório das áreas.

As classes de objetos criados para implementar os processos descritos acima foram empacotadas conforme denotado abaixo:

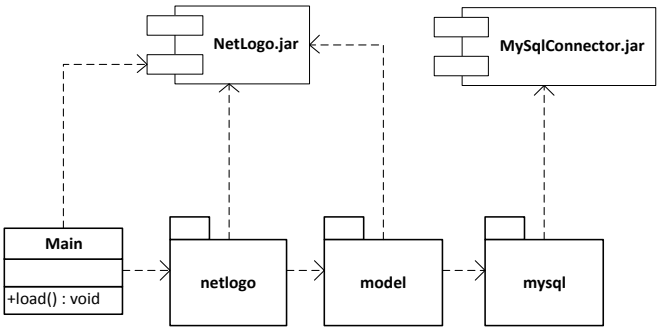


Figura 65: Empacotamento de classes da extensão fuzzy.

No modelo apresentado anteriormente, a classe *Main* é o artefato de ligação do NetLogo com o módulo de inferência nebuloso. Essa classe utiliza funcionalidades disponibilizadas por um componente fornecido no NetLogo versão 4.1.1 (NetLogo.jar) para agregar novas funções e procedimentos[9] ao ambiente de programação básico. As novas funções são incorporadas ao ambiente através da criação de um vínculo, no método *load* da classe *Main*, entre a definição de uma função (ou procedimento) e uma classe do módulo de extensão, conforme demonstrado abaixo:

primitiveManager.addPrimitive ("set-route", new RoutesRegistrator());

Onde:

addPrimitive método do componente NetLogo.jar cria uma nova função ou proc.;
set-route nome definição para a nova função um procedimento; e
RoutesRegistrator classe da extensão que implementa a nova função.

Foram definidas as seguintes funções e procedimentos no módulo de extensão *fuzzy*:

a)*clear-route*: Exclui todos os registros de rotas armazenados na base;
b)*set-route*: Registra na base a distância e o esforço de uma rota de um agente;
c)*set-route-coordinates*: Registra as coordenadas de um agente;
d)*update-routes-cost*: Calcula e atualiza o custo de todas as rotas da base; e
e)*get-routes-classification*: Retorna uma lista de classificação de rotas com o total de classificações solicitadas por parâmetro.

As classes que implementam as funções e os procedimentos definidos para a extensão *fuzzy* foram organizadas no pacote netlogo. Essas classes devem estender as classes *DefaultReporter* ou *DefaultCommand*, dependendo de quando se deseja criar funções ou procedimentos, respectivamente, conforme denotado no diagrama de classes que segue:

[9] Funções devem ser consideradas rotinas que retornam valor e procedimento rotinas que não retornam valor.

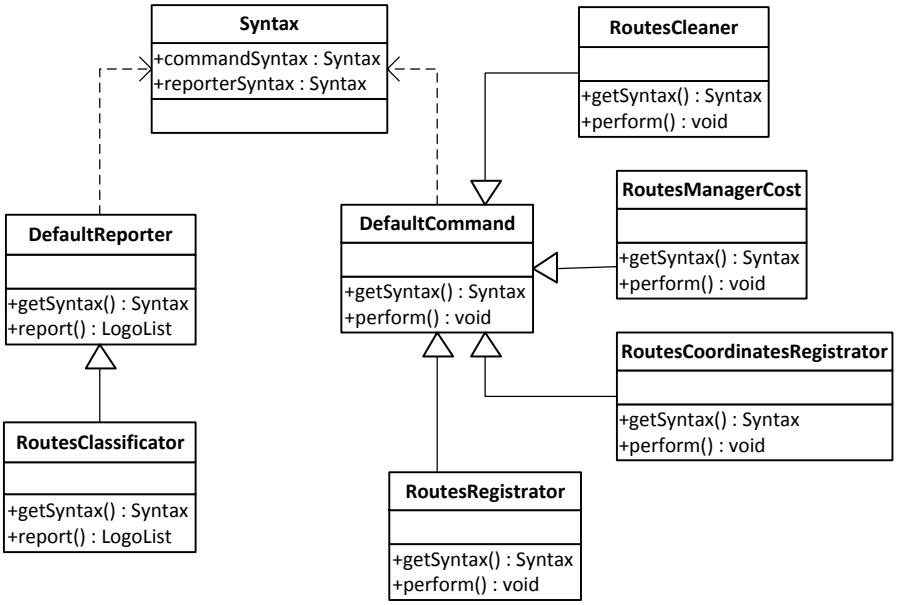


Figura 66: Diagrama de classes do pacote netlogo.

Nota-se no diagrama de classes acima que os métodos *getSyntax*, *perform* e *report* devem ser sobrescritos nas classes derivadas. O método *getSyntax* é responsável pela definição da assinatura de uma função ou procedimento e os demais responsáveis por implementá-los (*perform* para procedimentos e *report* para funções).

Outro artefato importante da arquitetura proposta é o pacote *model*. Segue o diagrama de classes que o compõe:

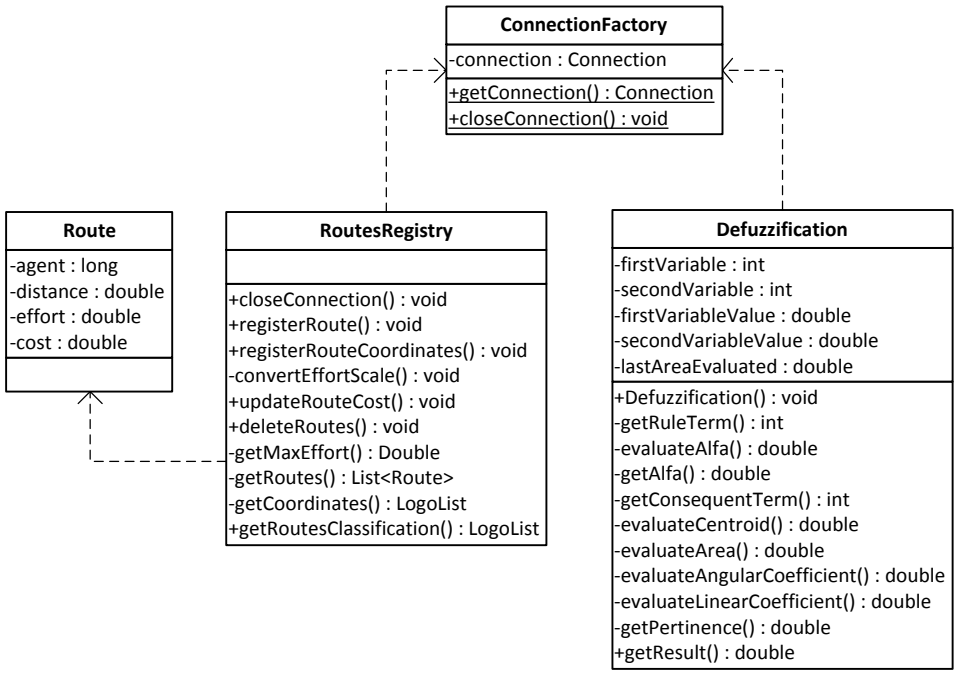


Figura 67: Diagrama de classes do pacote model.

Todas as rotinas necessárias às classes que implementam novas funções ou procedimentos para modelo proposto nesse trabalho foram organizadas no pacote *model*. Basicamente, são rotinas para operar com registros de banco de dados e para manipular a base de regras e os conjuntos nebulosos do módulo de inferência proposto. No diagrama anterior as operações com o banco de dados são gerenciadas pela classe *ConnectionFactory* do pacote *mysql*.

A extensão *fuzzy* é utilizada basicamente em dois pontos do fluxo de execução do módulo principal: na exploração e na classificação de rotas. Durante a exploração de rotas são executadas rotinas para iniciar estruturas de rotas (excluir registros da base), registrar informações de rotas (registrar distância e esforço) e registrar coordenadas de rotas. O diagrama de sequência abaixo denota a dinâmica dos objetos responsáveis pelo processo de exploração de rotas:

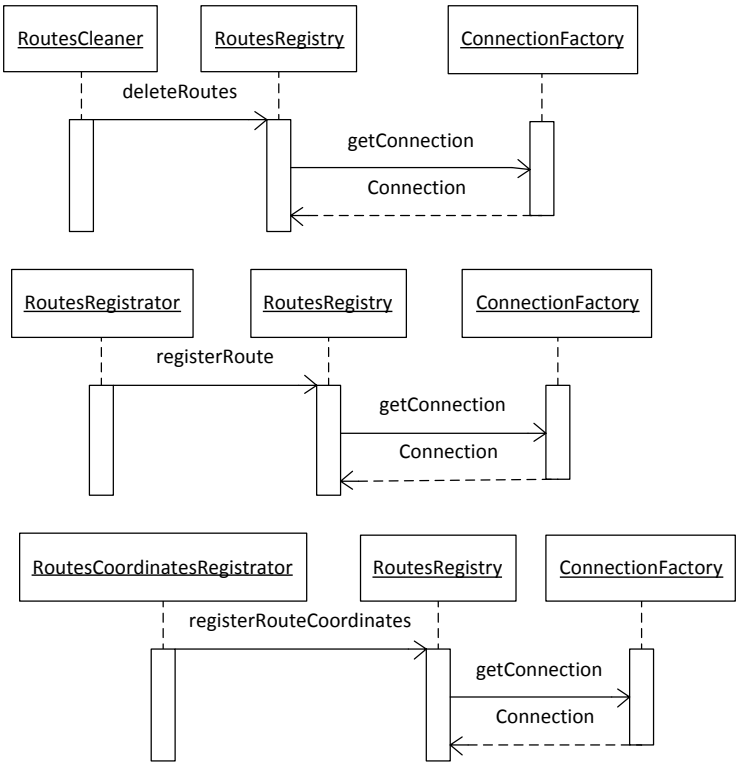


Figura 68: Diagrama de sequência para exploração de rotas.

Já o processo de classificação de rotas engloba rotinas para atualizar o custo das rotas exploradas (utilizando o modelo de inferência nebuloso proposto) e para selecionar uma quantidade qualquer de rotas ordenadas por custo. O diagrama de sequência que segue denota a dinâmica de objetos que compõem o processo de classificação de rotas:

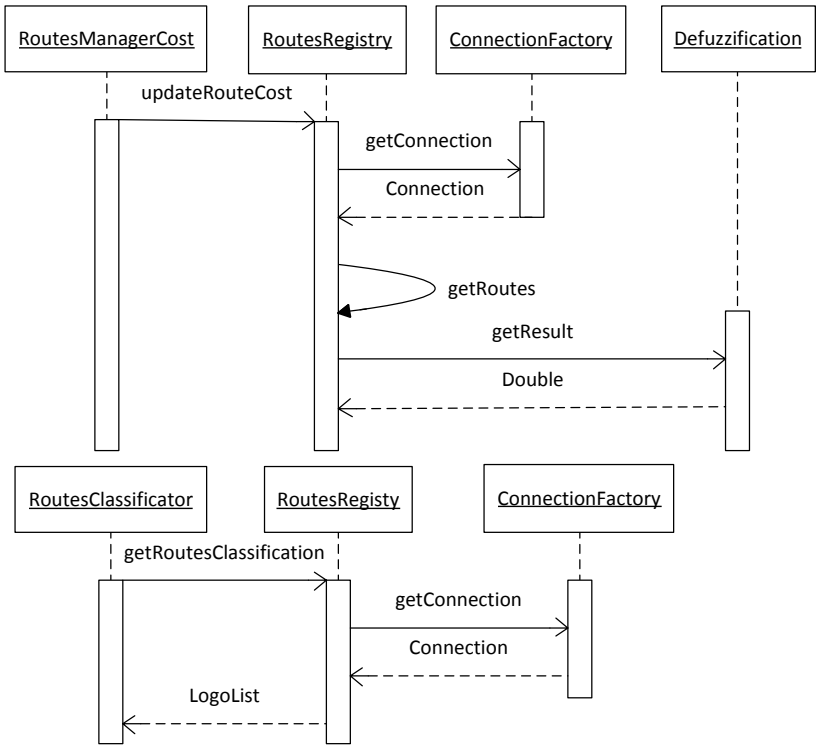


Figura 69: Diagrama de sequência para classificação de rotas.

Com isso, ficam definidos os fluxos de controle projetados para a implementação da arquitetura de sistema proposta nesse trabalho. Os modelos apresentados favorecem o desenvolvimento de estruturas flexíveis e extensíveis, além de propiciar a utilização de uma variedade razoável de controles e parametrizações nas telas de interface do sistema.

3.4.**Interface**

O principal objetivo da interface de entrada do sistema proposto é captar informações para parametrizar a execução de forma a maximizar o estudo de um problema de seleção de rotas. É fato que ao tratar de um problema no qual podem ser definidas rotas em qualquer direção o número de possibilidades de caminhos tende ao infinito. Portanto, a melhor abordagem encontrada para esse trabalho foi permitir que certas propriedades e variáveis envolvidas em um estudo fossem parametrizadas de acordo com as percepções obtidas pelos usuários.

Dessa forma, pretende-se utilizar uma interface em que, através de pequenos ajustes sobre os parâmetros de entrada, os usuários possam explorar melhor as funcionalidades do

sistema apresentado e atingir soluções mais eficientes, de acordo com a capacidade de processamento da plataforma computacional utilizada.

Normalmente, para se atingir resultados mais precisos ou para se explorar uma quantidade maior de possibilidades de caminhos são exigidas plataformas mais poderosas. Entretanto, nem sempre é possível atingir a solução ótima ou, em alguns casos, o melhor é tomar uma decisão rapidamente mesmo que não seja a decisão ótima.

Na interface proposta para o sistema apresentado buscou-se suportar uma variação ampla do consumo de recursos de processamento de uma plataforma com o objetivo de ampliar a aplicabilidade da solução desenvolvida. Segue uma ilustração do layout de interface de entrada utilizada:

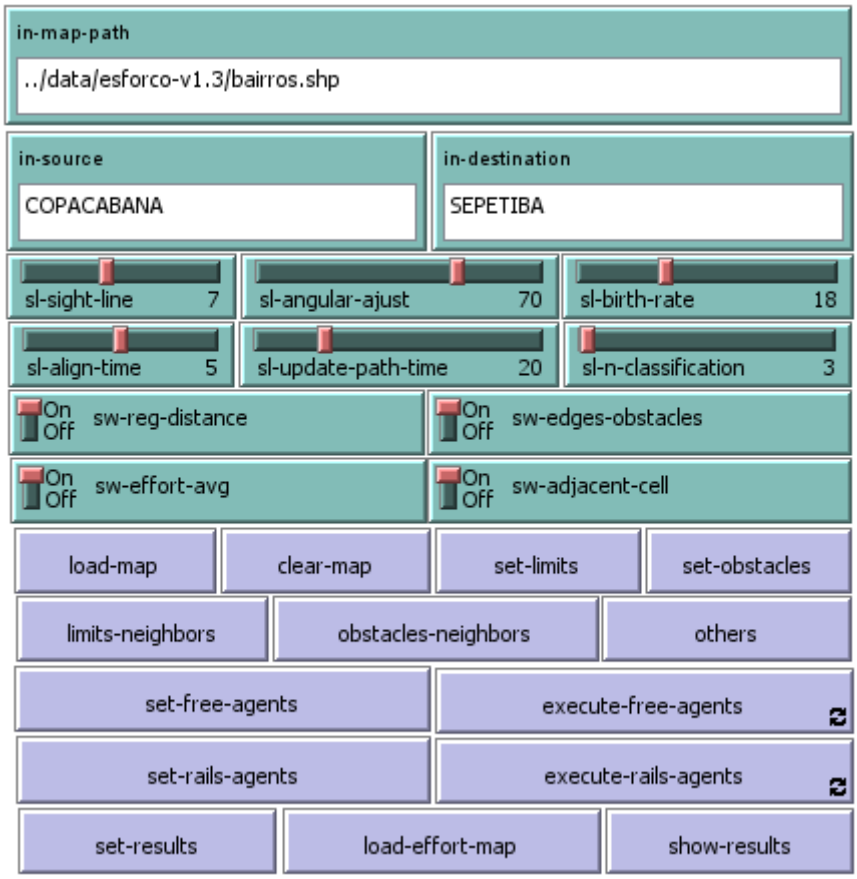


Figura 70: Layout de interface de entrada.

O campo *in-map-path* é preenchido pelo usuário com o caminho de um arquivo do tipo *shapefile*. Portanto, para carregar um mapa digital o usuário precisa definir o diretório em que o arquivo se encontra e pressionar o botão *load-map*. Com isso é possível analisar uma infinidade de problemas simplesmente alterando o mapa digital de trabalho.

Como serão avaliadas as rotas possíveis entre dois pontos de um mapa, é preciso definir esses pontos de origem e destino sobre o mapa. Esses pontos são definidos preenchendo-se os campos *in-source* e *in-destination* e pressionando o botão *set-limits*.

Para finalizar a definição básica do terreno sobre o qual se pretende selecionar rotas é preciso identificar as áreas caracterizadas como obstáculos (áreas intransponíveis). Essas áreas são polígonos do mapa cujo atributo esforço possui valor maior ou igual a 100 (ou seja, as medidas de esforço podem variar de zero a 100). Portanto, para identificar obstáculos basta pressionar o botão *set-obstacles*.

Após executar o controles descritos acima o que se tem é um mapa digital completamente carregado no ambiente de execução do NetLogo. O resultado dos parâmetros enviados através da interface gráfica é demonstrado na ilustração abaixo:

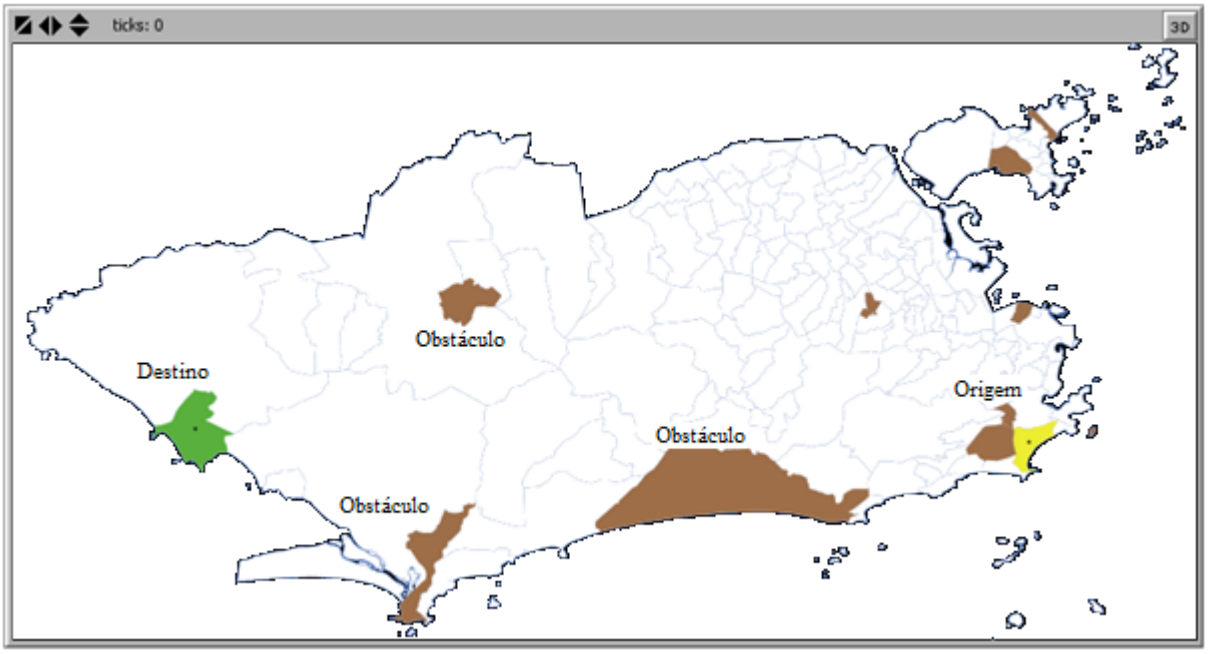


Figura 71: Mapa digital carregado no NetLogo.

A partir desse ponto, as próximas ações estão relacionadas à configuração dos vértices que irão compor um grafo de visibilidade. Essas ações devem ser executadas apenas quando se deseja utilizar agentes direcionados por grafos. São disponibilizados três controles para criação de vértices:

a)*limits-neighbors*: para criar vértices sobre os centroides de polígonos na vizinhança dos pontos de origem e destino;

b)*obstacles-neighbors*: para criar vértices sobre os centroides de polígonos na vizinhança de obstáculos; e

c)*others*: para criar vértices sobre os centroides de todos os polígonos do mapa, exceto obstáculos.

Os controles descritos acima podem ser executados individualmente ou em conjunto, deixando o usuário livre para decidir a melhor forma de distribuir rotas sobre o mapa. Nota-se que, com essa flexibilidade, fica a critério do usuário avaliar até que ponto pode-se demandar pelos recursos computacionais disponíveis ou quanto tempo de processamento pode-se dispor para se atingir uma solução eficiente.

Uma parametrização importante relacionada à criação do grafo de visibilidade diz respeito à forma como os vértices da vizinhança de um mesmo polígono se relacionam através de suas arestas. Os vértices da vizinhança de um mesmo polígono (obstáculo, por exemplo) podem ter ou não arestas entre si, dependendo do valor atribuído ao controle *sw-edges-cell* da interface de entrada. De forma similar, os vértices de polígonos vizinhos podem ser ligados ou não por arestas, dependendo do valor configurado no controle *sw-adjacent-cell*. Através dessas configurações é possível se estabelecer pequenos atalhos para se contornar obstáculos.

A partir da definição do grafo de visibilidade ou simplesmente da identificação dos obstáculos do mapa é possível criar agentes para exploração de rotas. Nesse momento é possível definir alguns parâmetros de entrada capazes de modificar consideravelmente as distribuições de rotas sobre um mapa. Alguns desses parâmetros são utilizados apenas por agentes livres, outros apenas por agentes direcionados por grafos e outros por ambos.

Os controles utilizados para caracterizar o comportamento de um agente livre são:

a)*sl-sight-line*: define o comprimento do raio do campo potencial (ou de visão) de um agente em relação a um obstáculo;

b)*sl-angular-ajust*: define o desvio angular sofrido por um agente ao interagir com um obstáculo;

c)*sl-birth-rate*: define a taxa de natalidade ou a quantidade de réplicas geradas por um agente ao desviar de um obstáculo;

d)*sl-align-time*: define o tempo (em milissegundos) que um agente leva para ajustar sua direção em função do campo potencial exercido pelo ponto de destino; e

e)*sl-update-path-time*: define o tempo (em milissegundos) que um agente leva para registrar sua coordenada corrente.

Das parametrizações descritas acima é possível concluir que quando menor forem o comprimento do raio do campo potencial e o ângulo de desvio de um agente mais próximo dos obstáculos serão os desvios. Além disso, quanto maior a taxa de natalidade maior a quantidade de rotas exploradas e quanto menor o tempo de ajuste de direção mais rápido o alinhamento com o destino. Finalmente, conclui-se que o tempo de registro de coordenadas influência diretamente na precisão dos caminhos selecionados, conforme demonstra a ilustração abaixo:

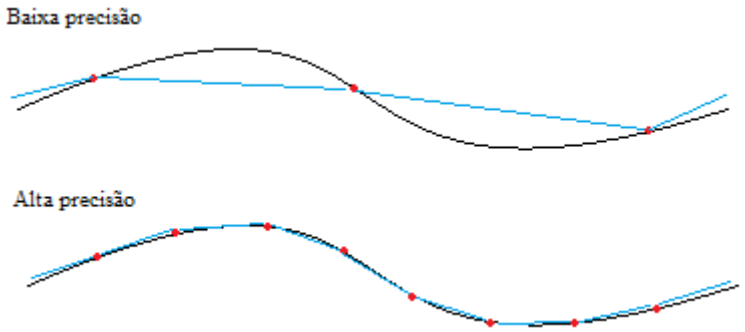


Figura 72: Tempo de registro de coordenadas de rotas.

Os controles utilizados para caracterizar o comportamento de um agente direcionado por grafo são:

a)*sl-birth-rate*: define o limite de natalidade ou o limite de réplicas geradas por um agente entre as arestas de um grafo; e

b)*sl-align-time*: define o tempo (em milissegundos) que um agente leva para alinhar sua direção com o vértice de destino.

Conclui-se com as parametrizações acima, que limitar as réplicas de agentes nos vértices de um grafo pode ser útil nos casos em que a quantidade de agentes trafegando atinja patamares que inviabilizem a execução do sistema. Portanto, pode ser necessário restringir o crescimento da população de agentes para se evitar a estagnação do sistema. Já o tempo para alinhamento dos agentes com vértices se faz necessário para minimizar a imprecisão do cálculo do ângulo de alinhamento, que diminui com a proximidade do vértice de destino. Em outras palavras, para garantir que o agente passará exatamente sobre o vértice é preciso ajustar sua direção continuamente.

Também é possível otimizar o percurso de um agente direcionado por grafo avaliando, durante sua execução, a posição do próximo vértice e a do ponto de destino. Se o controle *sw-reg-distance* estiver configurado com valor igual à *On*, um agente deverá comparar a distância até o próximo vértice com a distância até o destino e privilegiar a escolha do trecho de menor distância. Outra implicação do controle *sw-reg-distance* é não permitir que sejam criados vértices mais afastados do ponto de destino que do ponto de origem.

Nota-se mais uma vez que os controles disponibilizados para configuração de agentes buscam conciliar o melhor aproveitamento possível dos recursos computacionais em uso sem, no entanto, impedir que explorações mais modestas do mapa sejam possíveis. Uma questão importante nessas situações é: Até que ponto vale a pena aumentar a diversidade de rotas para se atingir um resultado melhor? O objetivo por trás desses controles que caracterizam o comportamento dos agentes é delegar a resposta para essa questão aos usuários do sistema.

Feitas as parametrizações para os comportamentos dos agentes, deve-se pressionar o botão *set-free-agents*, para criar agentes livres, e o botão *set-rails-agents*, para criar agentes direcionados por grafos. Esses controles podem ser utilizados individualmente ou em conjunto (ou seja, os dois tipos de agentes podem executar simultaneamente). Entretanto, para criar os agentes direcionados por grafo é preciso primeiro gerar os vértices do grafo. Após a criação dos agentes os botões *execute-free-agents* e *execute-rails-agents* devem ser pressionados para iniciar a exploração de rotas.

Uma questão importante sobre a exploração de rotas é como contabilizar a medida de esforço quando diversas áreas com diferentes níveis de esforço são atravessadas por um agente. Essas situações nem sempre podem ser resolvidas simplesmente somando-se as medidas de esforço porque o resultado final do somatório pode extrapolar o universo de discurso da variável linguística que representa o esforço (limitado entre zero e 100 pontos). Portanto, como alternativa para o somatório da medida esforço, a interface de entrada permite que o esforço resultante de uma rota seja calculado pela média dos esforços avaliados ao longo de uma trajetória, dependendo da configuração do controle *sw-effort-avg*.

Após finalizar a exploração de rotas pode-se executar o módulo de inferência nebuloso para atualizar o custo de cada rota registrada na base. A partir do cálculo do custo de cada rota pode ser obtida uma lista de classificação. Ao se pressionar o botão *set-result* o sistema atribui o custo para cada rota, exibe uma lista classificatória (legenda) com o total de itens definido no controle *sl-n-classification* e apresenta o traçado de cada rota classificada no mapa. Também é possível apresentar a distribuição de esforço no mapa, limpar o mapa e exibir novamente a classificação, a partir dos botões *load-effort-map*, *clear-map* e *show-results*,

respectivamente. Segue uma demonstração de um mapa com a distribuição de dois níveis de esforço (azul com 85 pontos e vermelho com 15):

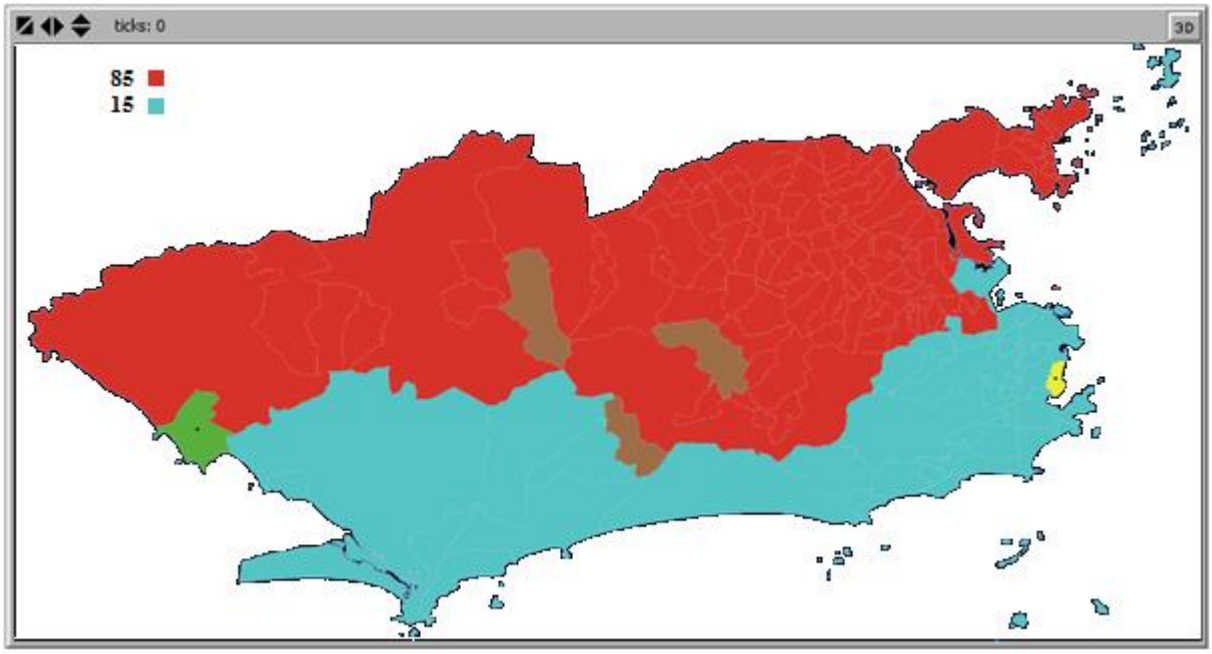


Figura 73: Distribuição de esforço.

Além dos controles da interface de entrada também é possível utilizar alguns controles disponibilizados pelo próprio ambiente de trabalho do NetLogo como o *Command Center*, para enviar comandos para um modelo em execução, e o *Model Settings*, para definir as dimensões do ambiente e o tamanho de *patches*. Essa segunda funcionalidade e particularmente importante para melhorar a precisão dos perímetros dos polígonos de um mapa. Com os recursos apresentados nessa seção é possível trabalhar com uma grande variedade de problemas e elaborar um grande número de simulações para o estudo de rotas.

3.5.**Comentários**

Com o conteúdo exposto nesse capítulo é possível compreender de maneira mais detalhada o funcionamento da arquitetura de sistema proposta. Dessa forma, não apenas a solução proposta foi apresentada, mas de que forma essa solução será viabilizada. Portanto, pode-se, a partir desse ponto, colocar finalmente a combinação dos conceitos abordados à prova.

Em outras palavras, será possível verificar através da execução de um sistema real se de fato as técnicas de sistemas baseados em agentes combinadas com mapas digitais e

modelos de lógica nebulosa podem gerar resultados eficientes para o problema de seleção de rotas em espaços com obstáculos.

4. **RESULTADOS**

Nesse capítulo serão apresentados e explicados os resultados obtidos a partir dos testes realizados com a implementação do sistema proposto para seleção de rotas. O principal objetivo desses testes é avaliar a aplicabilidade do sistema construído e demonstrar em quais situações o modelo apresentado nos capítulos anteriores pode ser mais eficiente.

4.1.**Apresentação**

Para avaliar que condições poderiam interferir na utilidade ou aplicabilidade do modelo computacional proposto nesse trabalho, foram elaborados casos de teste que enfatizassem a busca pela rota de menor custo (ou seja, com a melhor relação entre a distância percorrida e o esforço demandado). Como o principal objetivo do sistema é encontrar a rota de custo mínimo foram propostos cenários que demonstrassem a tendência ou o comportamento de aspectos relevantes à busca desse objetivo.

Portanto, foi definido um conjunto de variáveis que afetam direta ou indiretamente a eficiência de uma rota. Essas variáveis são: o custo de uma rota, a quantidade de obstáculos de um ambiente, a quantidade de agentes e o nível de distribuição do esforço sobre o mapa. Nota-se que cada variável pode ser analisada independentemente ou em conjunto com as demais.

Em cada caso de teste proposto a quantidade de variáveis independentes foi definida de acordo o objetivo do teste e o seu valor gradativamente ampliado para que os resultados fossem apresentados com um nível de detalhamento adequado. Foram elaborados quatro casos de teste:

a) Análise do custo mínimo de uma rota com a quantidade de agentes fixa, a distribuição de esforço fixa e a quantidade de obstáculos variável;
b) Análise do custo mínimo de uma rota com a distribuição de esforço fixa, a quantidade de obstáculos fixa e a quantidade de agentes variável;
c) Análise do custo mínimo de uma rota com a distribuição de esforço fixa, a quantidade de obstáculos variável e a quantidade de agentes variável; e
d) Análise do custo mínimo de uma rota com a quantidade de obstáculos fixa, a quantidade de agentes fixa e a distribuição de esforço variável.

Para simular as situações de teste propostas algumas variáveis em estudo, tais como a quantidade de obstáculos sobre o mapa ou a quantidade de agentes em execução no ambiente, precisaram assumir o comportamento de uma sequência crescente (ou seja, deviam receber valores que aumentassem gradativamente). Os valores atribuídos às variáveis de entrada nos casos de testes foram definidos de acordo com os termos de uma sequência de Fibonacci. Segue a descrição da sequência de Fibonacci:

$$F(n)\begin{cases} 0, se\ n = 0 \\ 1, se\ n = 1 \\ F(n-1) + F(n-2)\ se\ n > 1 \end{cases}$$

Onde:

n é o índice da sequência;
F(n) é o termo de índice n da sequência.

Portanto, os termos iniciais da sequência são:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Para cada caso de teste foram utilizados algoritmos de planejamento de rotas por *roadmaps* e por campo potencial. Seriam possíveis diversas combinações de parâmetros para se chegar às rotas de menor custo, mas, nesse trabalho, o principal objetivo com os testes propostos foi simplesmente analisar o comportamento das variáveis em estudo e identificar tendências que pudessem comprovar a aplicabilidade do sistema.

4.2.**Análise do custo mínimo de uma rota em função da quantidade de obstáculos**

Esse primeiro caso de teste é na realidade o caso mais simples de se analisar. Como o esforço é fixo o custo depende exclusivamente da distância (ou seja, apenas a distância influência o resultado). Além disso, apenas a quantidade de obstáculos muda no ambiente, aumentando a dificuldade de se traçar uma linha reta entre os pontos de origem e destino.

Logo, observa-se nesse teste o comportamento de uma função de uma única variável, onde a probabilidade de se encontrar a rota de custo mínimo é inversamente proporcional à quantidade de obstáculos do terreno. Em outras palavras, conforme aumenta a quantidade de obstáculos sobre o terreno diminuem as chances de se encontrar rotas de baixo custo.

Para realizar os testes foram utilizados mapas digitais nos quais a quantidade de obstáculos variou de acordo com os termos de uma sequência de Fibonacci e a posição de

cada obstáculo no mapa foi distribuída da forma mais homogênea possível (buscou-se espalhar obstáculos de forma a preencher gradativamente todo o mapa).

Foram elaborados mapas cuja quantidade de obstáculos varia de zero a 34 (ou seja, foram utilizados os dez primeiros termos da série de Fibonacci). Nota-se que o segundo e terceiro termos representam a mesma quantidade de obstáculos e, portanto, para esses termos, apenas um mapa foi elaborado.

Para cada mapa elaborado foram executados testes utilizando agentes direcionados por grafos e agentes livres. Cada custo obtido a partir das rotas exploradas foi então relacionado à quantidade de obstáculos existentes no mapa para que o efeito do aumento gradativo de obstáculo pudesse ser analisado. Além disso, com a execução isolada de cada algoritmo, foi possível observar também qual deles apresentava melhores resultados.

Na medida em que os obstáculos aumentavam sobre o mapa o espaço disponível para tráfego diminuía, levando os agentes a realizarem contornos cada vez mais longos. Além disso, o aumento de obstáculos proporcionou a incidência de polígonos aglutinados, formando blocos de obstáculos cada vez maiores, capazes de anular diversas possibilidades de rotas sobre o mapa. A ilustração abaixo demonstra a formação de obstáculos aglutinados impedindo que trajetórias fossem realizadas por áreas do centro do mapa:

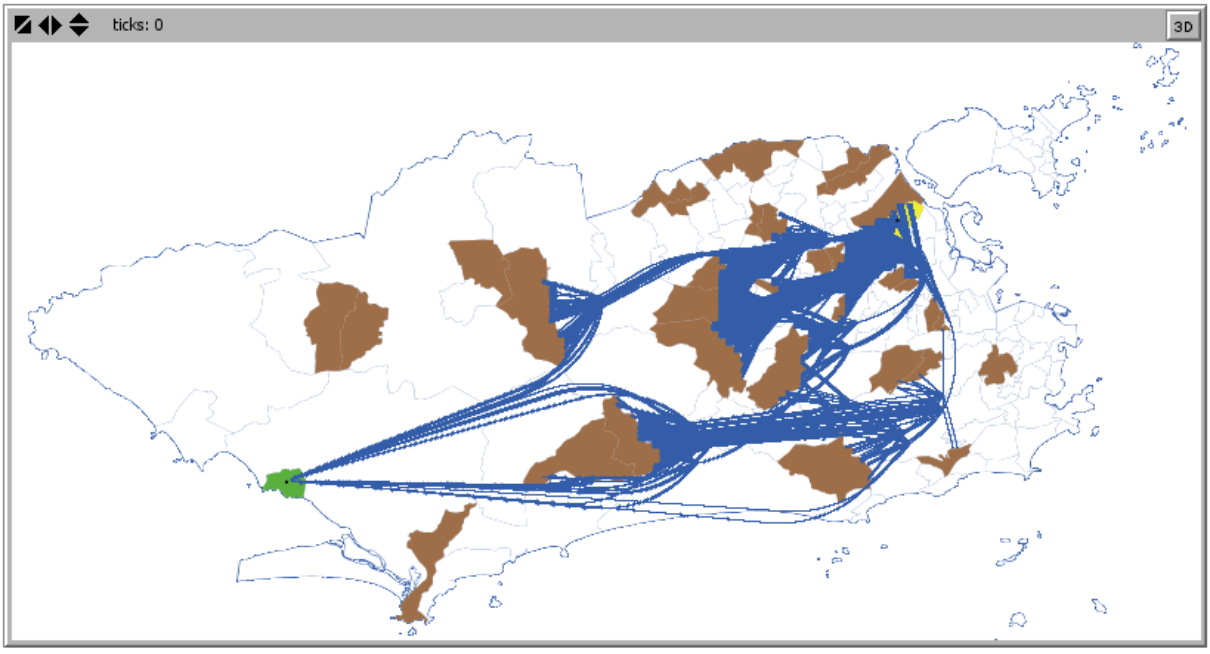


Figura 74: Agentes contornando obstáculos aglutinados.

Com isso, pode-se assumir que existe um limite para que a área ocupada por obstáculos não inviabilize a exploração de rotas em um mapa. Em outras palavras, deve

existir uma quantidade mínima de áreas interligadas e trafegáveis entre dois pontos do mapa para que pelo menos uma rota seja traçada. Dependendo da configuração de um mapa esse limite será atingido com uma quantidade maior ou menor de obstáculos, mas em qualquer configuração a quantidade de rotas possíveis irá diminuir gradativamente com o aumento das áreas definidas como obstáculos até que apenas uma única rota permaneça disponível. Nesse momento, qualquer área a mais configurada como obstáculo irá inviabilizar a exploração de rotas sobre o mapa em estudo.

Com a escassez de rotas exploradas a tendência observada foi obter distâncias cada vez mais longas interligando os pontos de origem e destino. Conforme previsto nesse primeiro cenário de testes, com a variável representando o esforço assumindo valor constante, a distância determinou o custo de uma rota e, consequentemente, o aumento de obstáculos influenciou diretamente no aumento desse custo. O quadro abaixo apresenta, para cada algoritmo, o custo mínimo obtido entre dois pontos relacionado à quantidade de obstáculos do mapa:

Quantidade de obstáculos	**Custo mínimo com agentes livres**	**Custo mínimo com agentes direcionados**
0	15,56	15,56
1	15,56	17,3
1	15,56	17,3
2	15,56	17,3
3	15,56	17,3
5	15,56	17,3
8	15,56	17,3
13	15,56	17,3
21	17,27	37,84
34	24,83	

Figura 75: Custo mínimo de uma rota em função da quantidade de obstáculos.

Nota-se que os agentes livres (implementação do algoritmo de campo potencial) suportaram o aumento de obstáculos melhor que os agentes direcionados (implementação do grafo de visibilidade), mantendo um custo igual a 15,56 até o oitavo termo da sequência de Fibonacci (ou seja, 13 obstáculos). Com os agentes direcionados o custo aumentou já no segundo termo da sequência e se manteve constante até o oitavo termo.

Outra observação importante é que os agentes direcionados atingiram o limite de caminhos disponíveis no nono termo da sequência e não encontraram caminhos interligando

os pontos de origem e destino quanto o mapa foi configurado com 34 obstáculos. Essa limitação poderia ser resolvida com o aumento do número de agentes. Mas, nesse caso de teste o número de agentes foi mantido para se enfatizar apenas o efeito gerado pelo aumento de obstáculos.

Apesar dos algoritmos apresentarem níveis de eficiência ligeiramente distintos, em ambos pode-se verificar a tendência de elevação dos custos de acordo com o aumento gradativo de obstáculos. Essa tendência pode ser bem observada através do gráfico abaixo:

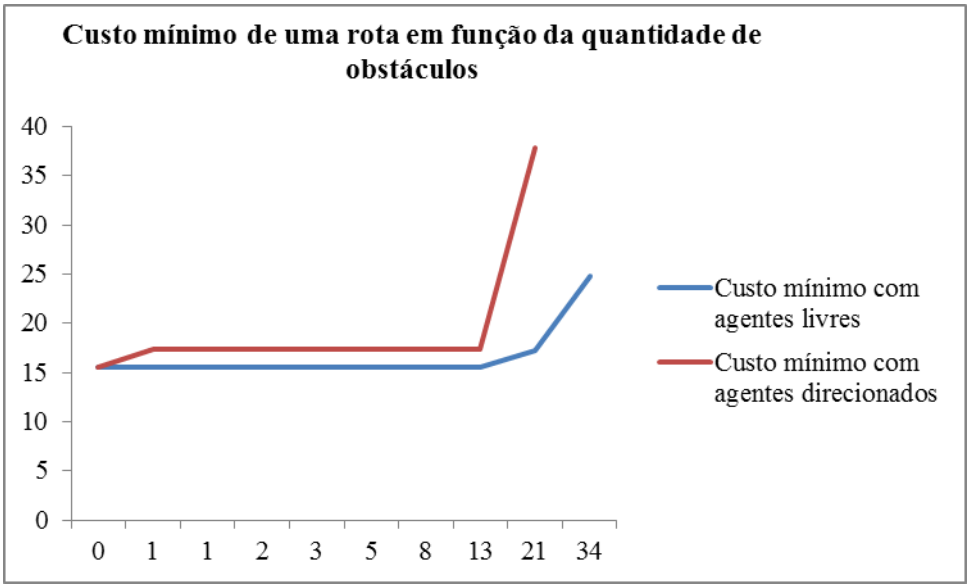


Figura 76: Gráfico do custo mínimo em função da quantidade de obstáculos.

Nota-se pelo gráfico acima que a tendência de elevação do custo de uma rota a partir do aumento do número de obstáculos sobre o mapa é maior quando empregado o algoritmo de agentes direcionados por grafo. Nesse primeiro caso de teste os agentes direcionados por grafo foram orientados por vértices das vizinhanças de obstáculos apenas. Apesar de poderem ser distribuídos vértices por todo o mapa, esse recurso não foi empregado para que o número de agentes permanecesse constante.

4.3.**Análise do custo mínimo de uma rota em função da quantidade de agentes**

O segundo caso de teste é bastante similar ao primeiro. Assim como no cenário apresentado anteriormente, a distribuição de esforço não varia e a variável que representa o custo de uma rota depende exclusivamente da medida de distância percorrida. Entretanto, nesse segundo caso, a quantidade de obstáculos não é alterada de uma execução para outra e a

quantidade de agentes é aumentada gradativamente. O objetivo é analisar o efeito desse aumento de agentes sobre a medida da rota de custo mínimo encontrada em cada execução.

Portanto, nesse segundo caso de teste, observa-se o comportamento de uma função de uma variável (a quantidade de agentes), onde a probabilidade de se encontrar rotas de baixo custo é diretamente proporcional à quantidade de agentes em execução no ambiente. Em outras palavras, conforme aumenta a quantidade de agentes para a exploração de rotas aumentam também as chances de se encontrar rotas de custos mais baixos entre dois pontos do mapa.

Para realizar os testes foram utilizados mapas digitais nos quais a quantidade de obstáculos não varia e a quantidade de agentes é alterada através do controle *sl-birth-rate* da interface de entrada do sistema. Foi escolhido um mapa com 21 obstáculos e os valores do controle *sl-birth-rate* variaram de acordo com os termos de uma sequência de Fibonacci.

Para cada taxa de natalidade definida no controle sl-birth-rate foram executados testes utilizando agentes livres (implementação do algoritmo de campo potencial) e agentes direcionados por grafos (implementação do algoritmo de grafo de visibilidade). Cada custo obtido a partir das rotas exploradas foi então relacionado à taxa de natalidade definida (configuração do controle sl-birth-rate), para que o aumento gradativo de agentes no ambiente pudesse ser analisado. Com a execução isolada de cada algoritmo foi possível observar qual deles apresentava melhores resultados.

Na medida em que a quantidade de agentes em execução aumentava, mais áreas do mapa podiam ser exploradas. Isso contribuiu para que custos de rotas cada vez menores fossem encontrados. Segue a ilustração de um teste com uma pequena quantidade de agentes em execução:

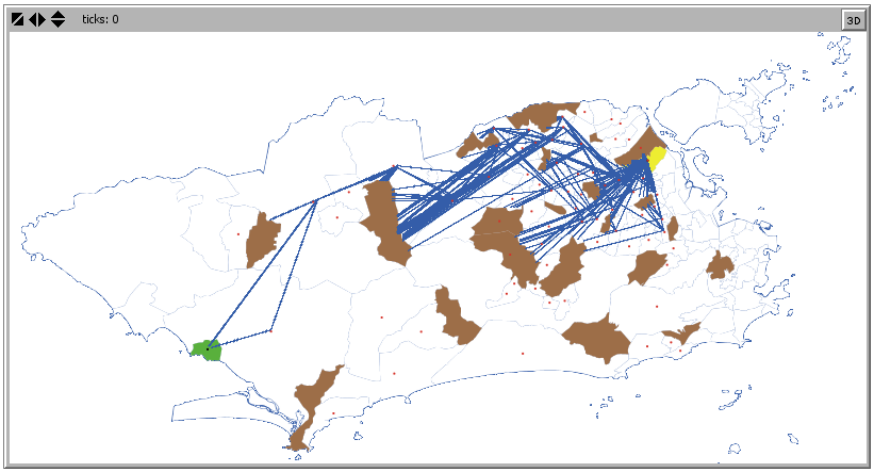


Figura 77: Exploração de rotas com uma pequena quantidade de agentes.

Nota-se na ilustração anterior que apenas a parte superior do mapa foi explorada devido ao baixo número de agentes. Segue uma nova ilustração demonstrando um teste com uma quantidade maior de agentes:

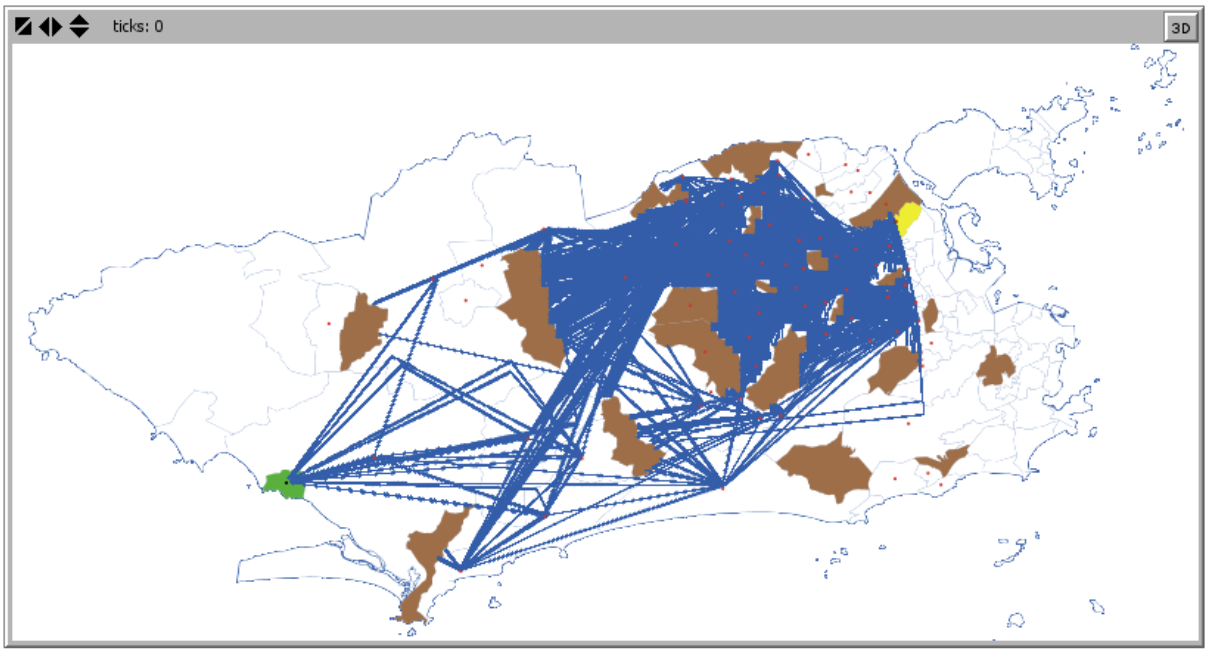


Figura 78: Exploração de rotas com uma grande quantidade de agentes.

Nota-se nessa nova ilustração, apresentada acima, que tanto a parte superior quanto a inferior do mapa foram bem mais exploradas devido ao aumento considerável de agentes em execução no ambiente.

Com isso, pode-se observar que o aumento do número de agentes em execução aumenta as chances de se explorar rotas com custos cada vez menores. Além disso, conforme a quantidade de agentes aumenta torna-se cada vez mais estreita a diferença entre os custos encontrados. Portanto, deve existir um valor limite, para a quantidade de agentes em execução, que determina a rota de menor custo ideal. A partir desse limite, ao aumentarmos a quantidade de agentes, os custos de rotas obtidos tendem a se manter praticamente equivalentes. O quadro abaixo apresenta, para cada algoritmo, o custo mínimo entre dois pontos relacionado à taxa de criação de agentes sobre o mapa:

Taxa de natalidade de agentes	**Custo mínimo com agentes livres**	**Custo mínimo com agentes direcionados**
0		
1	18,24	
1	18,24	

Taxa de natalidade de agentes	Custo mínimo com agentes livres	Custo mínimo com agentes direcionados
2	17,93	37,29
3	15,67	33,35
5	15,67	33,35
8	15,67	33,35
13	15,67	30,89
21	15,67	27,52
34	15,67	27,52
55	14,8	27,49

Figura 79: Custo mínimo de uma rota em função da quantidade de agentes.

Nota-se, assim como no primeiro cenário de testes, que os agentes livres obtiveram resultados melhores que os agentes direcionados por grafos. Os agentes livres conseguem encontrar rotas com custos bem menores já no primeiro termo da sequência de Fibonacci e no último termo da sequência obtêm um custo equivalente a pouco mais que a metade do custo obtido pelo algoritmo de agentes direcionados por grafos.

Outra observação importante foi que os agentes direcionados por grafo não conseguiram obter rotas com os três primeiros termos da sequência de Fibonacci. Em outras palavras, foi necessário configurar o controle *sl-birth-rate* com valor pelo menos igual a dois para que esse algoritmo encontrasse rotas interligando pontos de origem e destino no mapa.

Apesar do algoritmo de agentes livres (ou campo potencial) ser nitidamente mais eficiente nesse segundo caso de teste, nos dois algoritmos foi possível constatar a tendência de redução dos custos de rotas em consequência do aumento gradativo de agentes no ambiente. Essa tendência pode ser bem observada através do gráfico abaixo:

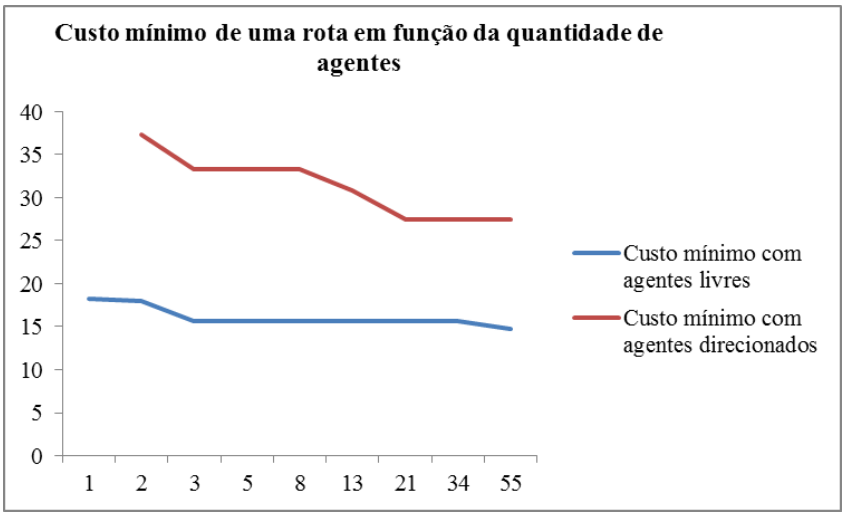


Figura 80: Custo mínimo em função da quantidade de agentes.

Nota-se também que, apesar do algoritmo implementado por agentes livres obter inicialmente menores custos, os custos obtidos através do algoritmo implementado por agentes direcionados por grafos decrescem mais rapidamente.

Outra importante observação nesse segundo cenário de teste foi que o aumento do número de agentes em execução poderia compensar um aumento da quantidade de obstáculos sobre o mapa. Nesse segundo caso de teste foi utilizado um mapa com 21 obstáculos enquanto a quantidade de agentes aumentava gradativamente. Nota-se que no primeiro caso de testes os custos mínimos obtidos com 21 obstáculos foram 17,27 e 37,84, para agentes livres e direcionados por grafos respectivamente. Foram atingidos custos menores no segundo teste a partir do terceiro termo da sequência de Fibonacci (ou seja, a partir do controle *sl-birth-rate* com valor igual a três). Essa redução nos custos se manteve até o último termo da série o que indica que o aumento de agentes pode reduzir os efeitos do aumento de obstáculos na exploração de rotas.

4.4.**Análise do custo mínimo em função das quantidades de obstáculos e de agentes**

O terceiro cenário de teste é uma combinação dos dois primeiros casos apresentados. No primeiro caso foram analisadas rotas de custo mínimo em função da quantidade de obstáculos sobre o mapa, enquanto no segundo caso foram analisadas rotas de custo mínimo em função da quantidade de agentes em execução. Nesse terceiro teste foram avaliadas rotas de custo mínimo em função das quantidades de obstáculos e de agentes.

Portanto, nesse terceiro cenário de teste observa-se o comportamento de uma função de duas variáveis (a quantidade de obstáculos e a quantidade de agentes), onde a probabilidade de se encontrar rotas de baixo custo é diretamente proporcional à quantidade de agentes e inversamente proporcional à quantidade de obstáculos. Em outras palavras, conforme aumenta a quantidade de agentes para exploração de rotas aumentam também as chances de se encontrar rotas de custo baixo entre pontos do mapa. Porém, conforme aumentam os obstáculos sobre o mapa diminuem as chances de se encontrar rotas de custo baixo. Dessa forma, a busca pela rota de custo mínimo ideal depende da relação entre a quantidade de agentes em execução e da quantidade de obstáculos sobre o mapa em estudo.

Para realizar os testes foram utilizados mapas digitais nos quais a quantidade de obstáculos variou de acordo com os termos de uma sequência de Fibonacci e a posição de cada obstáculo no mapa foi escolhida de forma a propiciar a dispersão e a distribuição homogênea de obstáculos, conforme no primeiro caso de teste. Para cada mapa digital

carregado a quantidade de agentes foi alterada através do controle sl-birth-rate da interface de entrada do sistema. Os valores do controle sl-birth-rate variaram de acordo com os termos de uma sequência de Fibonacci. A quantidade de obstáculos e os valores do controle sl-birth-rate variaram na mesma medida (ou seja, assumiram os mesmos termos da sequência para cada teste).

Para cada taxa de natalidade, definida no controle sl-birth-rate, e quantidade de obstáculos, estabelecidos sobre o mapa, foram executados testes utilizando agentes livres (algoritmo de campo potencial) e agentes direcionados por grafos (algoritmo de grafo de visibilidade). Cada custo obtido a partir das rotas exploradas foi então relacionado à taxa de natalidade e a quantidade de obstáculos definidos. Com a execução isolada de cada algoritmo foi possível observar também qual deles apresentava melhores resultados.

Quanto mais obstáculos eram configurados em um mapa mais reduzidas tornavam-se as áreas trafegáveis. Entretanto, com o aumento gradativo de agentes em execução mais áreas do mapa podiam ser exploradas. Dessa forma, o efeito causado pelo aumento de obstáculos sobre o mapa era compensado pelo aumento de rotas exploradas por agentes. O resultado desse equilíbrio pode ser observado na tabela abaixo, contendo os dados obtidos durantes os testes:

Taxa de natalidade de agentes	**Quantidade de obstáculos**	**Custo mínimo com agentes livres**	**Custo mínimo com agentes direcionados**
0	0		
1	1	16,31	17,3
1	1	16,31	17,3
2	2	15,56	17,3
3	3	15,56	17,3
5	5	15,56	17,3
8	8	15,56	17,3
13	13	15,56	17,3
21	21	15,67	27,52
34	34	28	29,3

Figura 81: Custo mínimo em função das quantidades de obstáculos e de agentes.

Nota-se que, nos dois algoritmos empregados, as medidas de custo mínimo se mantiveram constantes por vários termos da sequência de Fibonacci, o que demonstrou que, de certa forma, o aumento de obstáculos pode ser compensado pelo aumento de agentes. Outra observação importante foi que, assim como nos dois primeiros casos de teste, o algoritmo de agentes livres obteve rotas com custos menores nos primeiros termos da

sequência, mas os custos obtidos através do algoritmo de agentes direcionados por grafos decrescem mais rapidamente. Essa tendência pode ser observada no gráfico que segue:

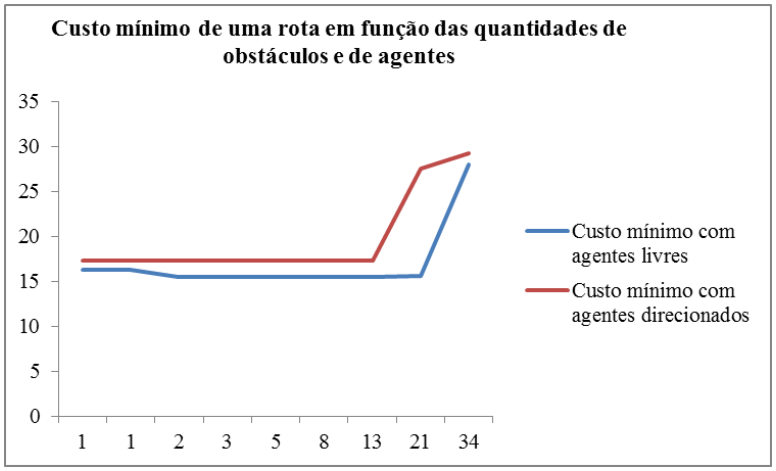


Figura 82: Custo mínimo em função das quantidades de obstáculos e de agentes.

Nota-se pelo gráfico anterior que a partir de certa quantidade de obstáculos (entre os valores oito e 21 do eixo das abscissas) ocorre uma mudança brusca no comportamento do gráfico e que a linha representando o algoritmo de agentes direcionados por grafos tende a se manter com custos bem próximos aos da linha representando o algoritmo de agentes livres. Na mesma faixa de valores em que as medidas de custos convergem para valores próximos nos dois algoritmos, observou-se que o aumento da quantidade de agentes não é capaz de compensar o aumento da quantidade de obstáculos. Ao verificar o mapa, notou-se que nesse instante de transição um grande obstáculo impediu que diversas possibilidades de trajetórias fossem realizadas pelo centro do mapa.

A ilustração abaixo apresenta as três rotas de menor custo encontradas em um mapa com 13 obstáculos:

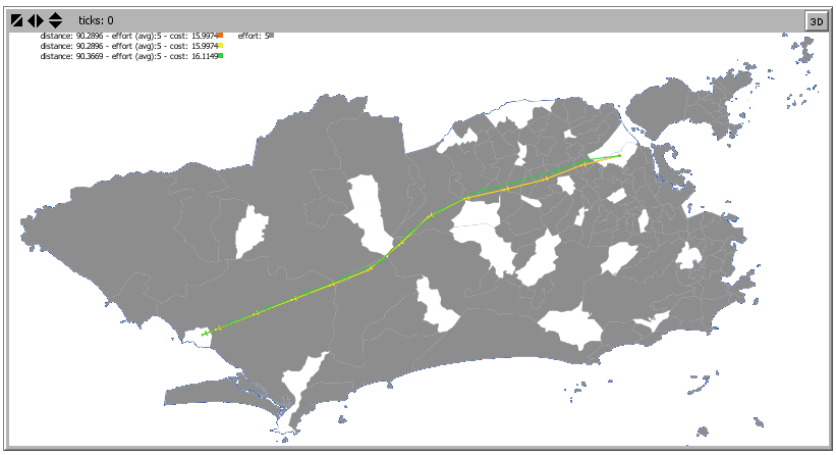


Figura 83: Rotas de menor custo sobre um mapa com 13 obstáculos.

Concluiu-se que enquanto foi possível trafegar pelo centro do mapa a diversidade de agentes em execução contribuiu para que esse espaço fosse bem explorado, buscando-se a rota de menor custo. Entretanto, quando a passagem pelo centro do mapa foi impedida o aumento do número de agentes tornou-se inútil à tentativa de obter rotas melhores, sendo necessário utilizar caminhos alternativos (momento em que ocorreu uma mudança brusca no gráfico apresentado anteriormente). O desvio imposto pelo aumento de obstáculos é demonstrado pela ilustração abaixo:

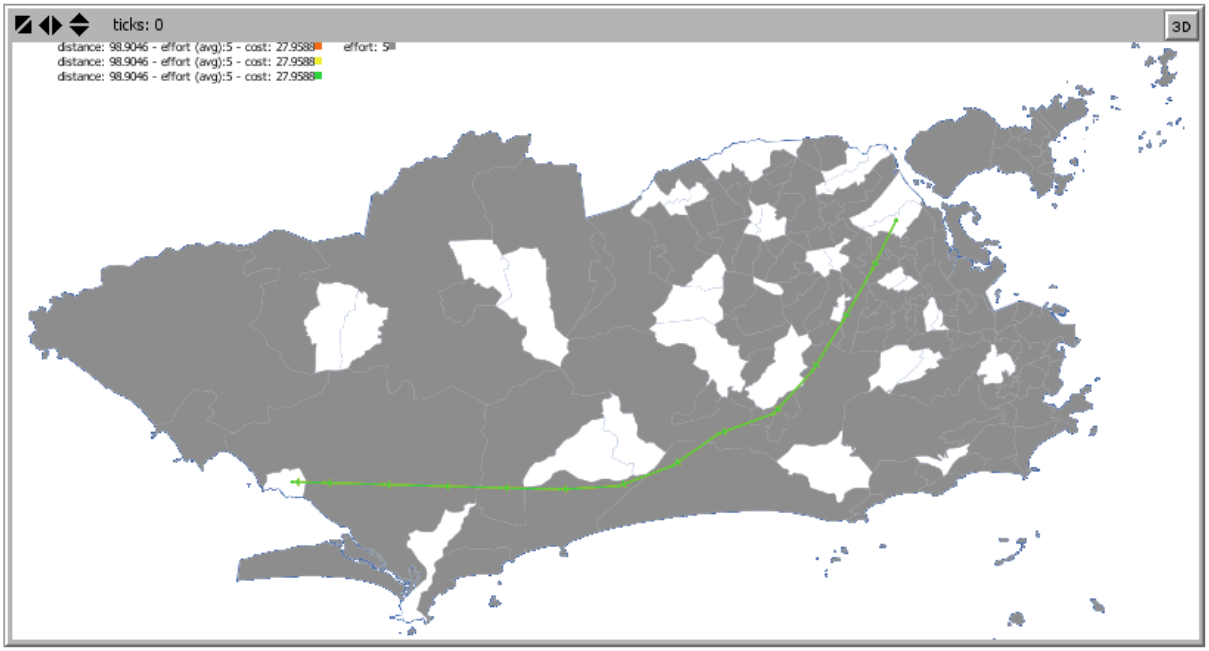


Figura 84: Rotas de menor custo sobre um mapa de 21 obstáculos.

4.5.**Análise do custo mínimo de uma rota em função da distribuição de esforço**

No quarto caso de testes foram avaliados aspectos relacionados à distribuição de esforço de um mapa. Os níveis de esforço são distribuídos sobre um mapa através de polígonos e consultados pelos agentes durante suas explorações de rotas. No final de um percurso cada agente terá coletado medidas de esforço de polígono sobre o qual trafegou e poderá consolidar essas medidas através de uma soma ou de uma média aritmética. Nesse quarto caso de teste o sistema foi configurado para consolidar as medidas de esforço através da média das medidas coletadas.

Durante os testes não foram alocados obstáculos sobre o mapa, permitindo que os agentes explorassem livremente um terreno em busca das rotas de menor esforço. Portanto, nesse quarto cenário de teste observa-se o comportamento de uma função de uma variável,

onde a diversidade de níveis de esforço distribuídos sobre um mapa tende a estabilizar as diferenças de esforço entre as rotas que ligam dois pontos do mapa. Em outras palavras, quanto maior a diversidade de níveis de esforço ou quanto maior a distribuição de diferentes níveis de esforço sobre um mapa, mais homogênea será a distribuição de esforço entre as rota exploradas, levando a rota de menor custo a aproximar-se da rota de menor distância.

Para se definir melhor o cenário proposto, pode-se considerar que inicialmente, quanto o espaço é divido em poucas áreas, onde são estabelecidos dois ou três níveis de esforço apenas, algumas rotas preferenciais são identificadas facilmente. Entretanto, após serem realizadas sucessivas divisões do espaço para distribuição de novos níveis de esforço o terreno tende a se tornar cada vez mais homogêneo. Até que, a partir de certa quantidade de níveis, qualquer rota terá um nível de esforço muito próximo dos níveis das demais.

Exemplificando, a configuração mais simples de distribuição de esforço sobre um mapa sem obstáculos é aquela em que existe apenas um nível de esforço. Nessa configuração, como todas as rotas exploradas terão a mesma medida de esforço, a rota de menor custo é uma linha reta entre os pontos de origem e destino (rota de menor distância). Ao se dividir o mapa em dois níveis diferentes de esforço o sistema de inferência nebuloso deve selecionar rotas de menor custo sobre as áreas do mapa com o menor esforço atribuído. A ilustração abaixo apresenta o resultado obtido durante um teste utilizando um mapa com a distribuição de dois níveis de esforço:

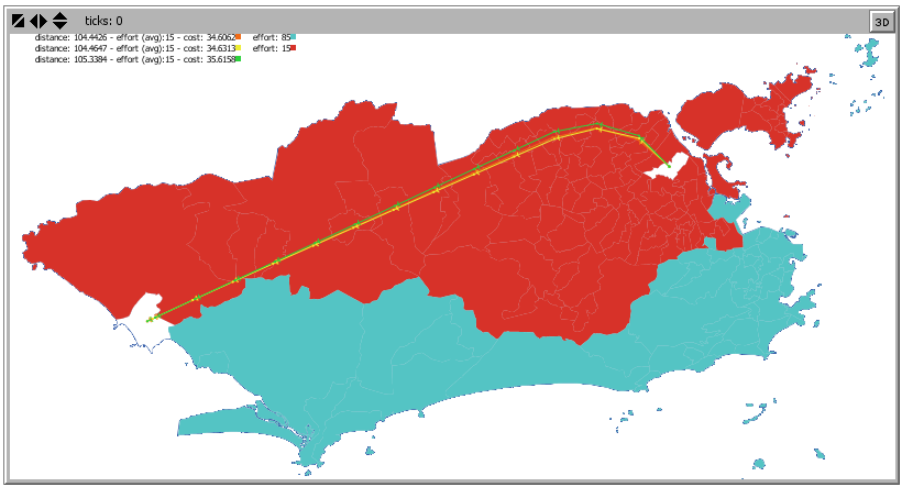


Figura 85: Rotas sobre áreas de menor esforço.

Nota-se que as rotas de menor custo são realizadas sobre pontos fora da área de maior esforço. Dessa forma, o aumento na distância consequente do desvio da área que demanda maior esforço, é compensado pela facilidade em trafegar sobre as rotas indicadas. Nos casos

como o ilustrado anteriormente é possível identificar as rotas de menor esforço facilmente. A ilustração abaixo apresenta mais uma situação onde as rotas preferências (ou seja, as rotas sobre áreas de menor esforço) podem ser identificadas com certa facilidade:

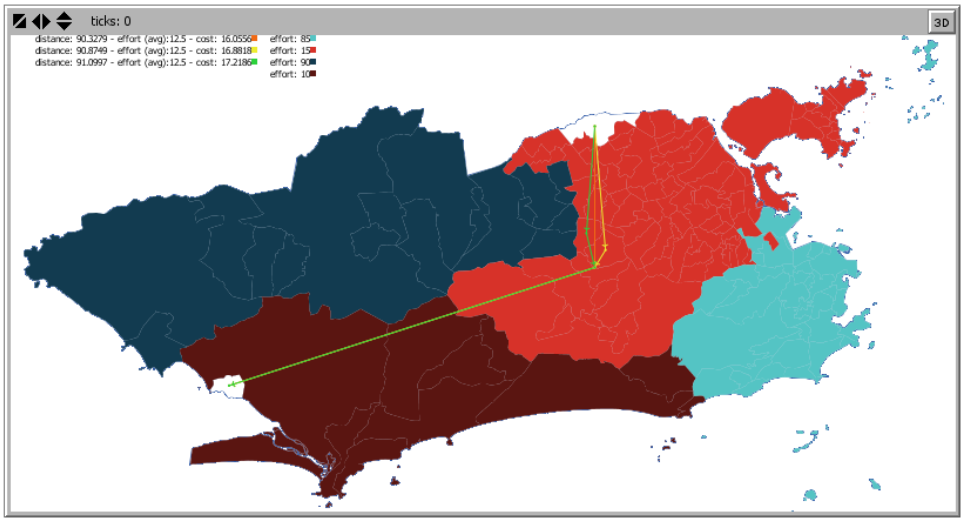


Figura 86: Rotas preferencias sobre um mapa com quatro níveis de esforço.

Nota-se que na ilustração acima o mapa possui quatro níveis de esforço distintos. Apesar disso, as áreas que proporcionam trajetórias de menor esforço são vizinhas e podem ser combinadas para interligar os pontos de origem e destino. Dessa forma, justifica-se o desvio realizado para evitar as áreas de maior esforço, mesmo com o aumento da distância a ser percorrida. Um último caso ilustrando a busca por rotas de menor esforço, com consequente aumento da distância percorrida, é apresentado abaixo:

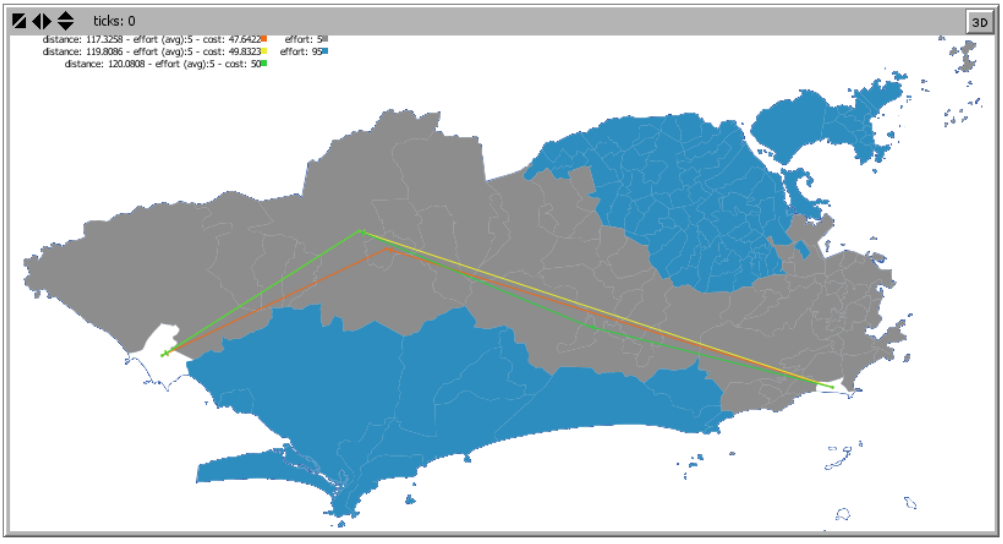


Figura 87: Rotas contornando áreas de maior esforço.

Na ilustração anterior, mais uma vez as rotas sobre áreas que demandam menor esforço implicam em percursos onde o custo final (relação entre a distância e o esforço) é menor. Assim como nos outros casos ilustrados nessa seção as áreas e níveis de esforço estão bem segregados no mapa, o que facilita a identificação das rotas de menor custo. Entretanto, podem existir configurações bem mais complexas, onde áreas com diferentes níveis de esforço são distribuídas de uma forma mais desorganizada sobre o mapa. Essas configurações mais desorganizadas são os principais objetos de testes dessa sessão.

Para os cenários de teste aqui propostos foram utilizados mapas digitais nos quais a quantidade de níveis de esforço variou de acordo com os termos de uma sequência de Fibonacci e a posição e tamanho de cada área, contento uma medida de esforço, variou aleatoriamente. Assim como nos casos de teste introdutórios, apresentados anteriormente nessa sessão, não foram configurados obstáculos sobres os mapas.

Para cada configuração de mapa (ou seja, para cada distribuição de esforço) foram executados testes utilizando agentes livres (algoritmo de campo potencial) e agentes direcionados por grafos (algoritmo de grafo de visibilidade com ênfase no método de decomposição celular). Cada custo obtido a partir das rotas exploradas foi então relacionado ao nível de distribuição de esforço definido para execução. Com a execução isolada de cada algoritmo foi possível observar que os resultados de ambos foram bastante próximos.

Quanto mais níveis de esforço eram distribuídos em um mapa mais semelhantes tornavam-se as rotas selecionadas. Na medida em que se aumentava a quantidade de níveis de esforço sobre um mapa diminuía a diferença entre a média dos custos das áreas definidas e os custos das rotas selecionadas. O resultado dessa homogeneização da distribuição de esforço sobre um mapa pode ser observado na tabela abaixo, contento os dados obtidos durantes os testes:

Quantidade de níveis de esforço	**Custo mínimo com agentes livres**	**Custo mínimo com agentes direcionados**	**Média dos níveis de esforço distribuídos no mapa**
0			
1	58,36	58,49	50
1	58,36	58,49	50
2	58,38	58,49	50
3	54,73	53,85	50
5	54,6	56,55	50
8	56,44	58,49	50
13	54,19	55,64	50

Figura 88: Custo mínimo em função da distribuição de esforço.

Nota-se que, nos dois algoritmos empregados, as medidas de custo mínimo se mantiveram com valores bem próximos, o que demonstra que a ausência de obstáculos torna a exploração de rotas bem similar em ambos os algoritmos. Essa similaridade entre as rotas exploradas nos dois algoritmos pode ser observada no gráfico abaixo:

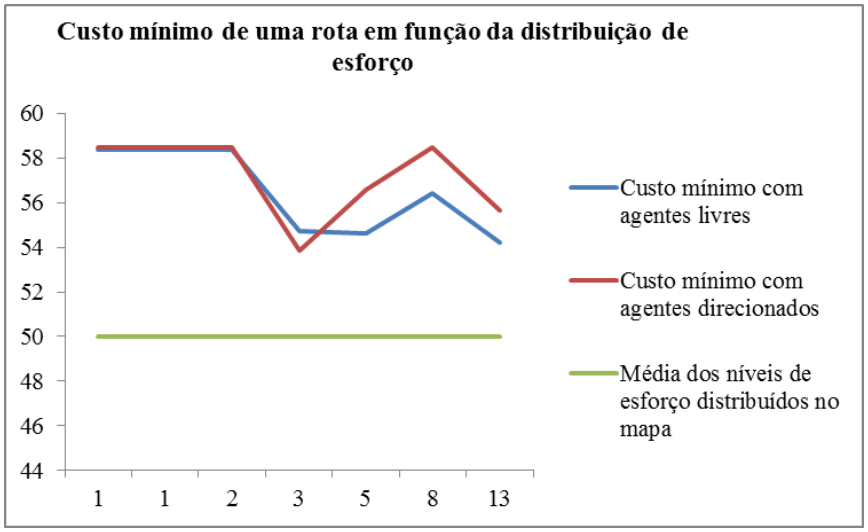


Figura 89: Custo mínimo em função da diversidade de níveis de esforço.

Nota-se também, conforme previsto, que após a distribuição de certa quantidade de níveis de esforço sobre o mapa, o custo das rotas selecionadas tende a se aproximar de valores próximos a 50. Justamente um valor que se aproxima da média aritmética dos níveis de esforço das áreas definidas no mapa em estudo. Portanto, pode-se presumir que quanto mais granular a distribuição de esforço sobre o mapa mais homogêneo será o nível de esforço constatado entre as rotas exploradas. Possivelmente ao se explorar rotas sobre um mapa com uma granularidade de esforço extremamente alta seriam obtidos resultados equivalentes aos de outro mapa com apenas uma única atribuição de esforço, cujo valor seria equivalente à média dos esforços do primeiro mapa.

4.6.**Avaliação da eficiência na descoberta de rotas**

A partir dos testes realizados nesse capítulo foi possível observar quais aspectos do sistema proposto seriam essenciais para que as rotas mais eficientes de um mapa fossem encontradas. Dentre os cenários abordados, dois foram mais importantes para enfatizar questões relacionadas à proximidade entre as rotas exploradas por agentes e as rotas entendidas como as mais eficientes de um mapa: a análise a partir do aumento de agentes com

nível de esforço fixo e a análise a partir do aumento da distribuição de esforço com quantidade de obstáculos fixa. No primeiro cenário, foi possível observar a capacidade do sistema em explorar rotas com desvios cada vez mais curtos. No segundo cenário, foi possível observar a priorização de caminhos que demandavam menor esforço e a tendência em obter distâncias cada vez menores a partir da homogeneização da distribuição de esforço.

Durante os testes em que o número de agentes era aumentado sistematicamente, constatou-se que o sistema possuía funcionalidades suficientes para explorar o mapa por vias mais ou menos estreitas, empregar desvios com ângulos mais fechados ou mais abertos e aumentar ou diminuir as opções de traçados sobre o mapa. Com isso, a cada configuração de teste, onde novos ângulos e novas divisões do espaço eram definidos, foi possível avaliar a capacidade do sistema de chegar mais próximo da rota de menor distância. Segue a ilustração que apresenta o esquema das rotas realizadas por agentes para desviar de um obstáculo, em contraste com a trajetória que seria da rota de menor distância:

Figura 90: Contorno de obstáculos em contraste com a rota mais curta.

Nesses esquemas enfatiza-se a tendência do sistema em reduzir as distâncias com o aumento de agentes, conforme demonstra a próxima ilustração:

Figura 91: Redução do desvio de obstáculos a partir do aumento de agentes.

Nota-se na ilustração anterior que, com o aumento da quantidade de agentes, foram traçadas rotas mais próximas do obstáculo e, consequentemente, com distâncias mais próximas da rota de menor distância. Apesar dessa tendência de se aproximar da rota de menor distância, a eficiência (ou a própria distância) da rota traçada por um agente dependerá de determinados fatores. Foram observadas nos testes duas situações de maior importância:

a)Caso o método de exploração de rotas utilize o algoritmo de campo potencial (agentes livres), a rota traçada por um agente jamais atingirá a rota de menor distância; e

b)Caso o método de exploração de rotas utilize o algoritmo de grafo de visibilidade (agentes sobre grafos), a rota de menor distância poderá ser obtida dependendo da distribuição dos centroides vizinhos aos obstáculos.

As constatações observadas nos testes estão diretamente relacionadas às ideias de rota mais curta e rota de desvio. A rota mais curta entre dois pontos de um mapa é uma linha reta. Quando um obstáculo impede o traçado de uma linha reta o desvio mais curto é formado pelos segmentos de reta que interligam o ponto de origem, o vértice do obstáculo e o ponto de destino, conforme demonstrado na ilustração que segue:

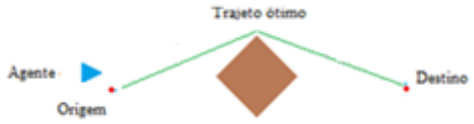


Figura 92: Demonstração do desvio mais curto de uma rota.

Na primeira situação, observada durante os testes com o algoritmo de agentes livres, como os traçados são realizados em curva, por menor que fosse a diferença entre a rota traçada e a rota de menor distância ela sempre existia. Isso porque, pela característica do algoritmo, os segmentos de reta necessários para traçar a rota de menor distância não podem ser obtidos, conforme demonstra a ilustração que segue:

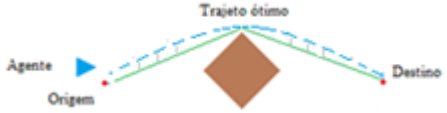


Figura 93: Demonstração do desvio de obstáculo em curva.

Apesar disso, nas avaliações feitas com esse algoritmo, notou-se que foram obtidas rotas com pontos muito próximos dos segmentos de reta que compõem a rota mais curta, conforme demonstram as ilustrações que seguem:

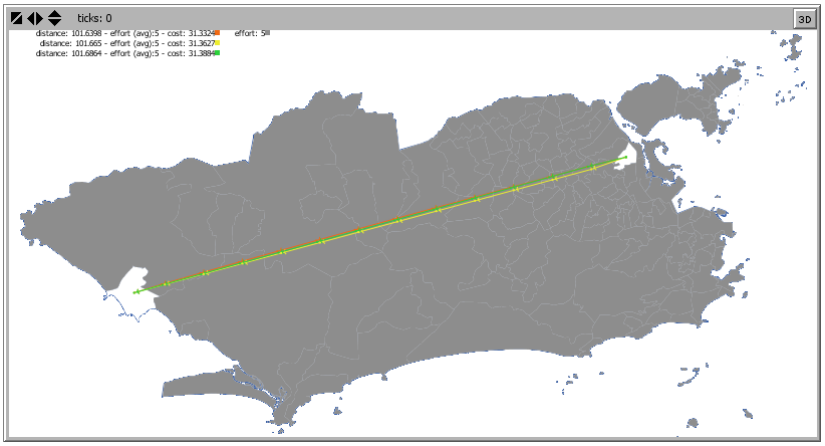


Figura 94: Trajeto em linha reta.

Figura 95: Contorno de um obstáculo.

Nas ilustrações apresentadas anteriormente, a trajetória em linha reta demandou um custo de 31,33 enquanto a trajetória com contorno demandou 31,46. As diferenças entre as respectivas distâncias foram também pequenas: de 101,64 para 101,74. Observou-se nos testes que tais diferenças, entre uma trajetória reta e a rota selecionada pelo sistema, tornavam-se mais estreitas na medida em que se reduzia o ângulo de desvio.

No caso do algoritmo de agentes sobre grafos os resultados podem ser ainda melhores. De fato, com esse algoritmo é possível atingir a rota de menor distância. Entretanto, a diferença entre a distância da rota traçada por um agente e a rota de menor distância dependerá da posição do centroide vizinho ao obstáculo. Quanto mais próximo do vértice do

obstáculo estiver o centroide vizinho, menor será a distância percorrida pelo agente, conforme demonstra a ilustração que segue:

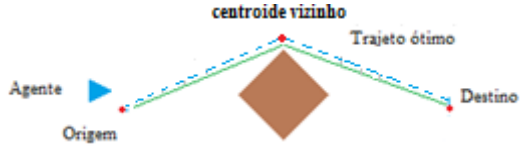


Figura 96: Demonstração do desvio de obstáculo por segmentos de reta.

Com isso é possível afirmar que, a partir da configuração adequada do mapa que representa o terreno em estudo, o sistema proposto é capaz de obter as rotas de menor distância que interligam pontos sobre esse mapa. Em outras palavras, através da configuração de um mapa digital os usuários poderão estabelecer desvios tão curtos quanto julgarem necessário. Isso garante que a rota de menor distância poderá ser obtida pelo sistema.

Além de comprovar ser possível utilizar o sistema para obter as distâncias mais curtas entre pontos de um mapa, foram avaliados também os efeitos da distribuição de esforço sobre o mapa, com o objetivo de validar a capacidade do sistema de escolher trajetórias sobre áreas de menor esforço.

No teste de distribuição de esforço, a expectativa era que o sistema selecionasse rotas em que o esforço baixo compensasse maiores distâncias percorridas, de acordo com as regras de inferências configuradas previamente no sistema. Essa expectativa foi comprovada com certa facilidade através de mapas onde os níveis de esforço foram distribuídos por duas ou três áreas bem segregadas.

Porém, para garantir que a seleção de rotas fosse realizada corretamente seria preciso não apenas visualizar seleções de rotas por caminhos de baixo custo, mas também avaliar o comportamento do módulo de inferência em situações onde a distribuição de esforço sobre o mapa fosse tão intensa que tornasse a superfície do mapa homogênea. Nesse caso extremo, a expectativa era que caminhos cada vez mais curtos fossem selecionados, já que a diferença entre esforço demandado por diferentes rotas exploradas se tornaria menor, em função da distribuição de esforço mais homogênea sobre o mapa.

Os efeitos do aumento na distribuição de esforço sobre um mapa podem ser bem compreendidos através das ilustrações que seguem. No caso mais simples são considerados apenas dois níveis de esforço sobre áreas bem segregadas no mapa.

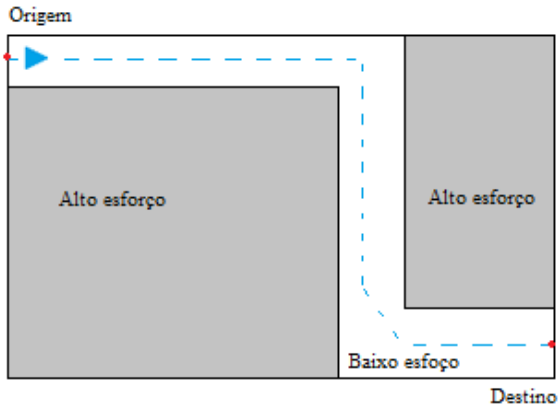


Figura 97: Baixa distribuição de esforço sobre um mapa.

Na ilustração anterior nota-se que, por existirem apenas dois níveis de esforço distribuídos por áreas bem segregadas, o caminho selecionado possuiria contornos bem acentuados, mantidos sobre áreas de menor esforço no mapa. Alterando o mapa explorado para outro com uma distribuição de esforço um pouco maior, o caminho selecionado já assumiria uma forma cujos pontos se aproximariam de uma reta, conforme demonstra a ilustração seguinte:

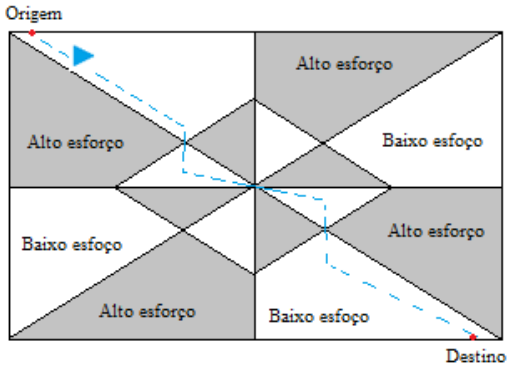


Figura 98: Média distribuição de esforço sobre um mapa.

Na medida em que a distribuição de esforço aumenta, a segregação entre as diferentes áreas do mapa diminui, assim como o tamanho de cada área também. Com o espaço cada vez mais homogêneo e os desvios dos agentes cada vez menores a expectativa durante os testes de distribuição de esforço passou a ser selecionar rotas cada vez mais curtas. Em outras palavras, nesse cenário, a distribuição de esforço deveria influenciar cada vez menos o cálculo do custo de uma rota, conforme demonstra a ilustração que segue:

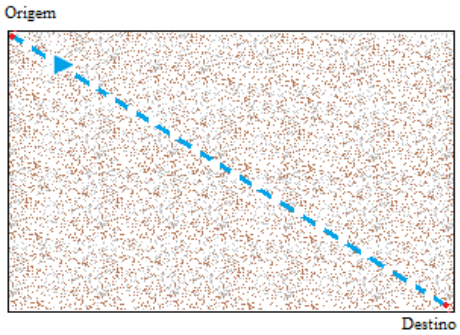


Figura 99: Alta distribuição de esforço sobre um mapa.

Concluiu-se que um bom indicador para avaliar a qualidade da seleção de rotas nos testes com distribuição de esforço deveria enfatizar a aproximação entre a rota selecionada pelo sistema e a rota de menor distância (no caso de um mapa sem obstáculos, uma linha reta). Portanto, sendo constatada nos testes uma tendência de redução da distância percorrida por um agente ao haver um aumento na distribuição de esforço sobre um mapa, seria possível admitir a capacidade do sistema de selecionar caminhos cada vez mais próximos da rota de esforço e distância ideais. Essa foi exatamente a observação feita durante os testes, conforme demonstram as ilustrações que seguem:

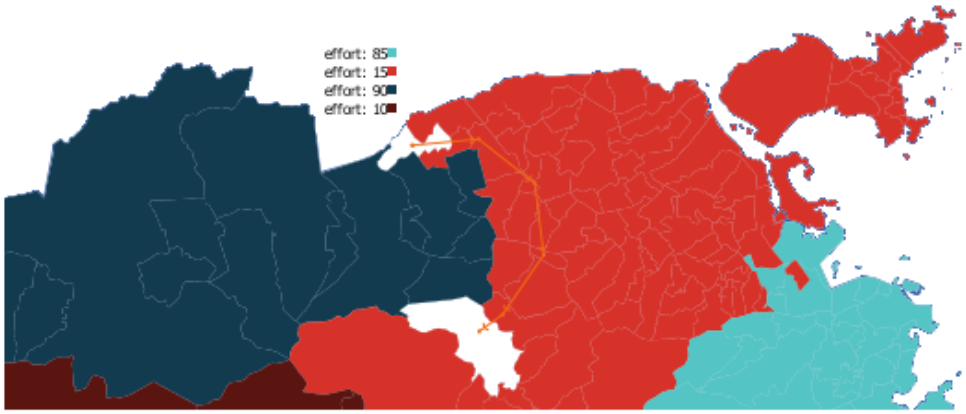


Figura 100: Mapa com baixa distribuição de esforço.

Na ilustração anterior a rota selecionada pelo sistema realiza um desvio para evitar o percurso sobre áreas onde o esforço demandado é maior. Dessa forma, a distância maior é compensada pelo menor esforço ao realizar o percurso selecionado. Ao aumentar a distribuição de esforço sobre o mapa observa-se que a distância da rota selecionada diminui (ou seja, a rota selecionada aproxima-se de uma linha reta). Observou-se nos testes que isso

ocorre quando não é possível realizar um percurso sobre um nível constante de esforço, conforme demonstra a ilustração que segue:

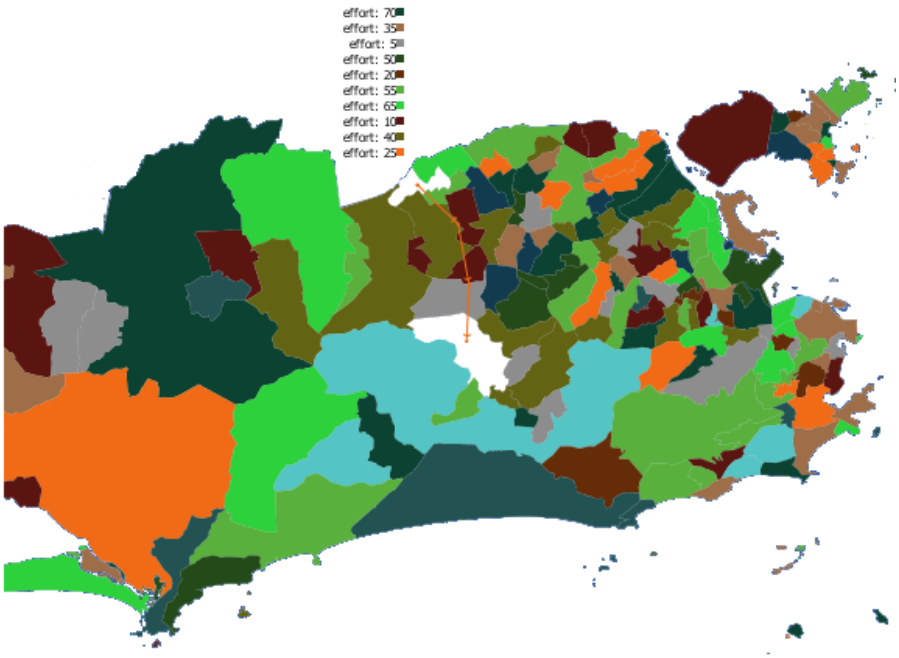


Figura 101: Mapa com alta distribuição de esforço.

4.7.**Comentários**

Através dos testes realizados com o sistema construído foi possível constatar sua eficiência em cenários variados de exploração de rotas em espaços com obstáculos ou com diferentes distribuições de nível de esforço. Ao combinar os cenários testados nesse capítulo observou-se a capacidade do sistema de suportar uma quantidade razoável de aplicações relacionadas ao estudo de rotas. Portanto, os testes realizados serviram adequadamente para validar os benefícios esperados para o modelo computacional proposto.

Na exploração de rotas sobre mapas com obstáculos, apesar das limitações impostas nos casos em que uma grande quantidade de obstáculos era configurada, os resultados foram satisfatórios, dado que nenhuma das configurações de mapas testadas impediu a exploração dos espaços disponíveis para tráfego. Em outras palavras, em todos os casos onde havia pelo menos um caminho disponível entre dois pontos no mapa houve descoberta de rotas em pelo menos um dos algoritmos implementados.

Nos casos em que a quantidade de agentes foi testada constatou-se que o sistema pode ampliar sua eficiência na exploração de rotas e, consequentemente, na busca de percursos com

custos cada vez menores através do aumento gradativo da propagação de agentes no ambiente de execução. A possibilidade de controlar a quantidade de agentes durante a exploração de rotas dá ao usuário do sistema a flexibilidade necessária para conciliar o tempo de processamento com a necessidade de resultados mais precisos, além de compensar a dificuldade imposta pelo aumento de obstáculos sobre o mapa em estudo.

Finalmente, nos casos de teste onde se avaliou a distribuição de níveis de esforço sobre um mapa constatou-se que o sistema foi capaz de responder eficientemente nas situações em que certos desvios poderiam levar às rotas de menor custo. Também foi avaliado o efeito causado pelo aumento da quantidade de níveis de esforço sobre o mapa (ou seja, o aumento da distribuição de diferentes níveis de esforço sobre o mapa). Constatou-se que esse aumento na distribuição de esforço pode homogeneizar o nível de esforço entre as rotas exploradas e, portanto, aproximar a rota de menor custo da rota de menor distância encontrada.

Os casos de testes utilizando mapas com obstáculos e com distribuição de níveis de esforço também apresentam resultados eficientes quando combinados. Ao realizar essas combinações observa-se que são mantidas as mesmas características observadas durante a exploração e a seleção de rotas de menor custo dos demais casos de testes. Em outras, palavras mantém-se a dificuldade imposta pelo aumento de obstáculos sobre o terreno, a melhora na exploração de espaços trafegáveis com o aumento de agentes em execução e a homogeneização do nível de esforço entre rotas com o aumento da distribuição de esforço sobre o mapa.

Dessa forma, os resultados foram satisfatórios e a aplicabilidade do sistema comprovada durante os testes. Provavelmente resultados ainda melhores poderiam ser atingidos se os recursos de configuração de entrada fossem mais explorados durante os testes ou se máquinas com maior capacidade de processamento fossem utilizadas. Mas, de qualquer forma, é possível perceber que uma grande quantidade de trabalho pode ser realizada automaticamente através do modelo computacional proposto.

5. **CONCLUSÕES**

Através dos estudos realizados durante esse trabalho de pesquisa, buscou-se propor um modelo computacional capaz de explorar e encontrar rotas de baixo custo em espaços geográficos sem delimitações de vias explícitas. Essas rotas de baixo custo foram definidas como rotas com a melhor relação entre a distância a ser percorrida e o esforço a ser empregado durante o percurso. Para atingir esse propósito, foi empregada uma combinação de técnicas computacionais que, ao serem aplicadas na avaliação de um terreno representado em um mapa digital, pudesse apresentar uma classificação de rotas interligando dois pontos sobre esse terreno.

Dessa forma, foi elaborado um modelo computacional capaz de ler um mapa, identificar os pontos de origem e destino, identificar a distribuição de esforço e os obstáculos do terreno e, finalmente, apresentar uma listagem das rotas de menor custo. Para isso, foram realizados estudos para integrar conceitos e técnicas relacionados a sistemas baseados em agentes, sistemas de inferência nebulosos, métodos de planejamento de rotas em robótica e cartografia digital. Os resultados obtidos apresentam não apenas algumas respostas para a aplicação dessas técnicas na Geomática, mas também muitas novas questões sobre a diversidade de aplicações importantes que poderiam ser exploradas a partir dessas combinações de conceitos abordadas.

Com relação aos objetivos definidos nos capítulos introdutórios dessa dissertação, considera-se que todos foram adequadamente atendidos, uma vez que:

a) O modelo computacional integrando técnicas de sistemas baseados em agentes e lógica nebulosa funcionou corretamente;
b) O processo de tomada de decisão, implementando um sistema de inferência nebuloso, é flexível e pode ser utilizado em uma grande quantidade de estudos de rotas, bastando, para isso, que os conjuntos nebulosos e os mapas digitais utilizados sejam configurados adequadamente; e
c) A seleção de rotas do sistema construído apresentou resultados eficientes durante os testes realizados.

Pode-se constatar, pelos resultados apresentados nesse trabalho, que o modelo computacional proposto é útil em estudos de rotas e pode ser empregado em simulações ou

em situações de tomada de decisão. Além disso, pelo fato do sistema apresentado abordar a seleção de rotas sem a delimitação de vias e generalizar a segmentação do espaço através da distribuição de níveis esforço, constata-se na solução proposta uma característica bastante particular e também uma vantagem, com relação à sua aplicabilidade, quando comparada às ferramentas de seleção de rotas disponíveis na Internet ou apresentadas em trabalhos recentes.

A combinação de conceitos da lógica nebulosa, de sistemas baseados em agentes, dos métodos de planejamento de rotas e da cartografia digital é algo que, certamente, pode trazer grandes benefícios em diversas áreas do conhecimento. Acredita-se que os resultados obtidos nesse trabalho são uma parcela infinitamente pequena em relação às grandes inovações que ainda podem emergir dessa abordagem no estudo de rotas.

Com o sistema construído a partir dessa pesquisa é possível poupar uma grande quantidade de tempo e esforço em projetos de engenharia e logística, processando uma grande quantidade de distâncias e de medidas de esforço entre dois pontos de um mapa automaticamente. Além disso, a forma como o problema de rotas foi abordado permite que a dificuldade imposta pela capacidade de processamento dos computadores atuais seja contornada, o que caracteriza um resultado importante em problemas desse tipo (onde são processadas grandes massas de dados).

Além da utilização do modelo proposto para simulação e exploração de rotas em projetos de estradas e linhas de transmissão, também existe a possibilidade de utilizar seu potencial em sistemas de automação e robótica. Seria viável, por exemplo, utilizar um computador central que enviasse coordenadas para robôs que dependessem da definição de rotas para executar uma atividade específica (por exemplo, agricultura ou mineração). Dessa forma, as atividades desses robôs poderiam ser gerenciadas através de um mapa digital que sofresse alterações de acordo com o clima, por mudanças geográficas ou, até mesmo, pelo resultado das interações dos próprios robôs com seu ambiente, representado no mapa digital.

Atualmente, robôs são empregados em diversas atividades que oferecem algum risco ou esforço prejudicial às pessoas, tais como: atividades em águas profundas ou de exposição ao calor intenso, radiação ou substâncias tóxicas. A utilização de mapas digitais para direcionar esses robôs combinada com um sistema para seleção de rotas poderia manter uma quantidade grande de equipamentos em funcionamento durante dias, meses ou até anos, com pouca ou até mesmo nenhuma intervenção humana. Existem diversas aplicações atualmente que testam ou uso de robôs para controle de incêndio e manutenção de tubulações, por exemplo, que poderiam se beneficiar da abordagem proposta nessa pesquisa.

Entretanto, para garantir que esses novos objetivos sejam atingidos com eficiência, será necessário estudar com maior profundidade os métodos de planejamento de rotas para propor melhorias ou para desenvolver métodos novos. Também seria muito importante otimizar a colaboração entre os agentes durante a exploração de rotas e melhorar as funcionalidades do sistema criado para reduzir o processamento com rotas pouco eficientes (rotas que acabam sendo descartadas no final da exploração). Por fim, os casos de testes precisariam ser ampliados e os dados coletados sobre o funcionamento do sistema mais detalhados para que funções matemáticas pudessem modelar com maior precisão o comportamento do sistema em situações extremas.

Portanto, essa pesquisa encerra uma etapa dos estudos necessários para a aplicação de sistemas baseados em agentes e de lógica nebulosa em problemas de seleção de rotas sobre mapas digitais e abre novas possibilidades para que o modelo computacional apresentado possa ser avaliado em outros estudos, tanto no campo da Geomática, quanto em outras áreas da ciência. Espera-se que as contribuições apresentadas nesse trabalho possam ajudar na exploração de novas perspectivas de estudo e na inspiração de soluções inovadoras. Que os pequenos passos dados nessa pesquisa possam abrir novos caminhos para a ciência.

REFERÊNCIAS

Belém, F. L., Cunha de Araújo Pimenta, L., Ramos Fonseca, A., Teofilo Rezende, D., Tanure Rocha, F., Mendonça Bosque, M., . . . Rodrigues de Souza, A. (2009). Ferramenta para Seleção de Corredor de Linha Aérea de Transmissão Utilizando Geoprocessamento. *XIV Simpósio Brasileiro de Sensoriamento Remoto. 3559-3566.* Natal: INPE.

Cox, E. (1994). *The fuzzy systems handbook: a practitioner's guide to building, using, and maintaining fuzzy systems.* New York: AP PROFESSIONAL.

Crowley, J. L. (1985). Navigation for an Intelligent Mobile Robot. *IEEE Journal of Robotics and Automation*, 31-41.

Davidson, D. A., Watson, A. I., & Selman, P. H. (1993). An evaluation of GIS as an aid to the planning of proposed developments in rural areas. *Geographical Information Handling: Research and Applications.* London: Wiley.

Eclipse. (2012). *Eclipse*. Acesso em 2012, disponível em http://www.eclipse.org/.

ESRI. (2012). *http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf*.

Filho, E. d. (2002). *Iniciação à lógica matemática.* São Paulo: Nobel.

Firby, R. J. (1993). An Architecture for a Synthetic Vacuum Cleaner. *AAAi Fall Symposium Series Workshop On Instantiating Real-World Agents.*

Gero, J. S. (1988). *Artificial intelligence in engineering: robotics and processes.* Elsiver.

Latombe, J. C. (1991). *Robot Motion Planning (The Springer International Series in Engineering and Computer Science).* Kluwer Academic; 1 edition.

Mendel, J. M. (1995). Fuzzy Logic Systems for Engineering: A Tutorial. *83*.

Monteiro, C., J. Ramírez-Rosado, I., Miranda, V., J. Zorzano-Santamaría, P., García-Garrido, E., & Fernández-Jiménez, L. (2005). GIS spatial analysis applied to electric line routing optimization. *IEEE Transactions on Power Delivery, 20*.

MySQL. (2012). *http://www.mysql.com/*. Acesso em 2012

NetLogo. (2012). *http://ccl.northwestern.edu/netlogo/index.shtml*. Fonte: NetLogo.

O'Hare, G. M., & Jennings, N. (1996). *Foundations of distributed artificial intelligence.*

Oracle. (2012). *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. (Oracle) Acesso em 2012, disponível em Oracle.

Ottoni, G. d. (2000). *Planejamento de trajetórias para robôs móveis.* Universidade Federal do Rio Grande.

S. Mata , C., & S. B. Mitchell, J. (1997). A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). *SCG '97 Proceedings of the thirteenth annual symposium on Computational geometry*.

Schalkoff, R. J. (1990). *Artificial intelligence: an engineering approach.* McGraw-Hill.

Voronoi. (2012). *pt.wikipedia.org/wiki/Diagrama_de_Voronoi*. Acesso em 2012.

W. Dijkstra, E. (1982). *A Note on Two Problems in Connexion with Graphs.* Amsterdam: Burroughs Corporation and Eindhoven University of Technology.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems.* John Wiley & Sons Ltd.