Vítor de Albuquerque Torreão

ABORDAGENS EVOLUCIONÁRIAS PARA O PROBLEMA DA PATRULHA MULTIAGENTE TEMPORAL

Trabalho de Graduação

Universidade Federal Rural de Pernambuco
secretaria@preg.ufrpe.br
http://www.ufrpe.br/br/graduacao

RECIFE
2015

Vítor de Albuquerque Torreão
Abordagens Evolucionárias para o problema da Patrulha Multiagente Temporal/ Vítor de Albuquerque Torreão. – RECIFE, 2015-
67 p. : il. (algumas color.) ; 30 cm.

Orientador Pablo Azevedo Sampaio

Trabalho de Graduação – Universidade Federal Rural de Pernambuco, 2015.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática
Bacharelado em Ciência da Computação

Vítor de Albuquerque Torreão

ABORDAGENS EVOLUCIONÁRIAS PARA O PROBLEMA DA PATRULHA MULTIAGENTE TEMPORAL

Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Departamento de Estatística e Informática da Universidade Federal Rural de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Pablo Azevedo Sampaio*

RECIFE
2015

Trabalho de Conclusão de curso apresentado por **Vítor de Albuquerque Torreão** ao programa de Bacharelado em Ciência da Computação do Departamento de Estatística e Informática da Universidade Federal Rural de Pernambuco, sob o título **Abordagens Evolucionárias para o problema da Patrulha Multiagente Temporal**, orientada pelo **Prof. Pablo Azevedo Sampaio** e aprovada pela banca examinadora formada pelos professores:

Prof. Pablo Azevedo Sampaio
Departamento de Estatística e Informática/UFRPE

Prof. Péricles Miranda
Departamento de Estatística e Informática/UFRPE

Prof. <Nome>
Centro de Informática/UFPE

RECIFE
2015

Dedico este trabalho a todos os meus familiares, amigos e professores que me deram o suporte necessário para chegar aqui.

Agradecimentos

Agradeço aos meus pais, Roberto e Cláudia por terem investido tudo em mim e me ensinado desde cedo a valorizar a minha educação.

Agradeço a minha irmã, Maria da Graça, pela ajuda com o seu silêncio dentro de casa enquanto redigia e pelo seu super poder de me deixar mais confiante.

Agradeço a minha querida e amada companheira, Marília, que me aturou nos momentos de estresse, incerteza e ansiedade durante todo a faculdade e especialmente enquanto trabalhava nesta monografia.

Meus sinceros agradecimentos a todos os professores do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco que me ajudaram nesta caminhada fantástica pelo ensino superior.

Finalmente, agradeço a Deus pela calma e clareza de raciocínio que me foram dadas quando mais precisava, não só nesta, mas em todas as provações que passei.

Not everything that counts can be counted, and not everything that can be counted counts.

—ALBERT EINSTEIN

Resumo

A Patrulha Multiagente Temporal é uma complexa tarefa multiagente, a qual requer que um grupo de agentes coordene as ações uns dos outros a fim de obter os melhores resultados para todo o grupo. Alguns exemplos de patrulha são: patrulhar as fronteiras de um país, vigiar os corredores de um prédio, monitorar frotas marítimas e inspecionar áreas que possam ser sujeitas a vazamento de gás ou incêndio. Uma solução eficiente para a Patrulha Multiagente pode contribuir em uma variedade de domínios, tais quais, administração de redes de computadores, motores de busca da Web e fiscalização do tráfego.

Trabalhos recentes propuseram diversas soluções para a patrulha eficiente feita por um grupo de agentes. Agentes heurísticos, agentes baseados em teoria dos jogos, mecanismos de negociação, técnicas de aprendizado com reforço, estratégias gravitacionais, agentes baseados em colônia de formigas e abordagens híbridas evolucionárias com *Ant Colony Optimization* já foram aplicados para solucionar o problema da patrulha multiagente. O presente trabalho visa contribuir para o estudo do problema da patrulha através do desenvolvimento de estratégias puramente evolucionárias e sua comparação, através de simulação, com outras estratégias presentes na literatura.

A avaliação empírica das estratégias será realizada utilizando benchmarks propostos por outros pesquisadores em trabalhos anteriores, software disponível de graça na internet e mantido pela comunidade de pesquisadores da área, e o simulador chamado *Simple Patrol* disponibilizado pelos pesquisadores da Universidade Federal Rural de Pernambuco.

Palavras-chave: agentes autônomos, sistemas multiagentes, coordenação e patrulha, algoritmos evolucionários

Abstract

The Multiagent Temporal Patrolling is a complex multi-agent task, which requires a group of agents to coordinate each other's actions in order to obtain optimal results for the whole group. Patrolling a country's borders, watching the corridors of a building, monitoring maritime fleets, inspecting areas that may be subject to gas leakage or fires are examples of such a patrolling task. An efficient solution to the Multiagent Patrolling can contribute in a variety of domains such as computer network administration, web search engines and traffic inspection.

Previous works have proposed many solutions to patrolling efficiently with a group of agents. Heuristic agents, game theory based agents, negotiation mechanisms, reinforced learning techniques, gravitational strategies, ant colony based agents and hybrid evolutionary and ant colony optimization approaches have all been applied to solve the multi agent patrolling problem. This work aims to contribute to the study of the patrolling task by developing pure evolutionary approaches and comparing them to other strategies proposed in the literature through simulations.

The empirical evaluation of the strategies will be made using benchmarks proposed by other researchers in previous publications, software freely available on the internet and maintained by the community of researchers, and the patrolling simulator named Simple Patrol provided by researchers from the Universidade Federal Rural de Pernambuco.

Keywords: autonomous agents, multi agent systems, coordination and patrolling, evolutionary algorithms

Lista de Figuras

Lista de Quadros

Lista de Pseudocódigos

Lista de Acrônimos

TMAP	Timed Multiagent Patrolling
TSP	Problema do Caixeiro Viajante
TSPM	Problema do Caixeiro Viajante com Múltiplas Visitas
EA	Algoritmo Evolucionário
ACO	*Ant Colony Optimization*
ES	Estratégia Evolucionária
GA	Algoritmo Genético
MQI	Média Quadrática dos Intervalos
MQIN	Média Quadrática dos Intervalos Normalizada

Sumário

1

Introdução

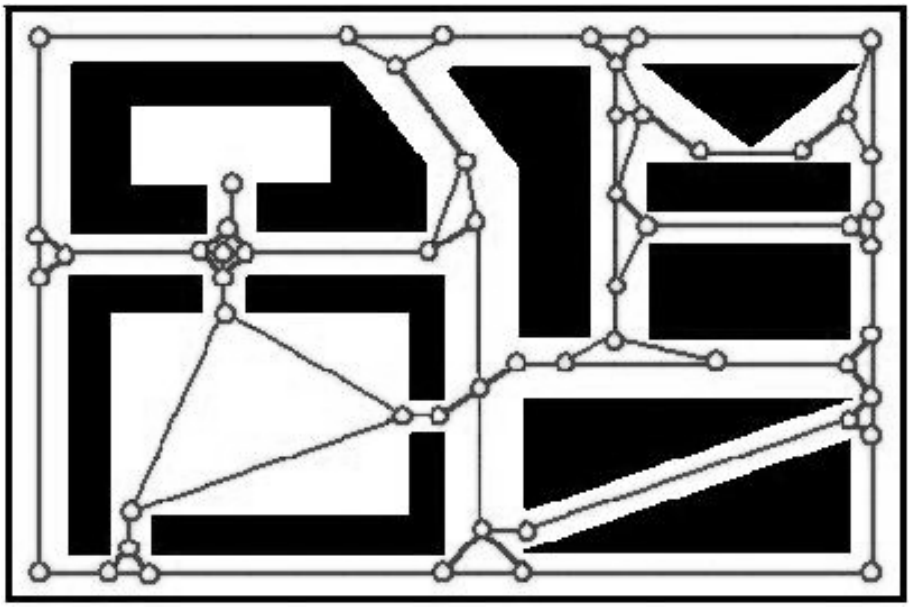
Atividades relacionadas à vigilância, inspeção e controle, que hoje são realizadas por seres humanos, são candidatas a serem executadas por sistemas autônomos no futuro (HERNÁNDEZ; CERRO; BARRIENTOS, 2013). Alguns exemplos dessas tarefas são: patrulhar fronteira de países ou muros de uma área civil, vigiar os corredores de um prédio, monitorar frotas marítimas e inspecionar áreas sujeitas a vazamentos de gás ou incêndios (SAMPAIO, 2013).

Segundo a publicação (HERNÁNDEZ; CERRO; BARRIENTOS, 2013), esses sistemas de segurança quando operados por humanos são, em sua maioria, previsíveis e inflexíveis, pois sua performance pode ser influenciada por fatores como o tédio, a distração ou a fadiga. Dessa forma, os autores afirmam que é importante melhorar os elementos de segurança desses sistemas para auxiliar seres humanos e destacam a Patrulha Multiagente (CHEVALEYRE; SEMPE; RAMALHO, 2004) como um dos sistemas que pode fazer esse papel.

A Patrulha Multiagente pode ser definida como a tarefa de um agente que deve perceber uma porção limitada do ambiente e detectar eventos ou anomalias (ALBERTON et al., 2012). Mais especificamente, pode ser definida como um problema no qual um time de indivíduos (agentes), visita tão frequentemente quanto possível, pontos de interesse contidos em uma área (POULET; CORRUBLE; SEGHROUCHNI, 2012a). Ou ainda como um problema de vigilância, onde deve-se minimizar o tempo entre visitas dos agentes aos locais importantes de um ambiente conhecido (PIPPIN; CHRISTENSEN; WEISS, 2013).

Existem algumas versões do problema que estendem essa definição, variando as características dos agentes e do ambiente. Dentre elas destacam-se: problema da patrulha em ambientes dinâmicos, isto é, o ambiente onde o agente se move muda ao longo da execução do agente (DOI, 2013). Outra variação leva em consideração que os agentes podem ter velocidades diferentes (LAURI; KOUKAM, 2014). Alguns autores trabalharam com o problema da patrulha onde os agentes não tem a mesma capacidade e operam em diferentes níveis de qualidade (PIPPIN; CHRISTENSEN; WEISS, 2013). Outros trabalham com agentes que possuem restrições de mobilidade (ALBERTON et al., 2012). E há ainda uma variação onde a quantidade de agentes muda ao longo da execução, isto é, um determinado agente pode sair ou ser substituído por outro:

Figura 1.1: Exemplo de ambiente modelado em um grafo



Fonte: (MACHADO, 2002)

são os chamados sistemas abertos (POULET; CORRUBLE; SEGHROUCHNI, 2012a).

Para estudar o problema, no entanto, esta pesquisa utilizará uma definição mais formal, precisa e abrangente do problema, que auxilie o pesquisador a enxergar a patrulha multiagente do ponto de vista matemático e a implementar soluções em aplicações de software. Assim, a definição técnica empregada no decorrer deste trabalho será a proposta por Sampaio (SAMPAIO, 2013).

Segundo o autor, uma instância da Patrulha Multiagente Temporal, do inglês Timed Multiagent Patrolling (TMAP), é uma 7-tupla, $\langle E, P, S, s_0, A, I, M \rangle$. O ambiente que o agente percebe é representado por um grafo (ROSEN, 2006), E, onde os vértices representam os pontos de interesse, P, a serem visitados e as arestas representam os caminhos entre esses pontos. Na Figura 1.1, visualiza-se um exemplo de ambiente para o problema da patrulha modelado como um grafo. O modelo demanda também o conjunto de estados possíveis da sociedade de agentes, S, e um estado inicial, s_0, tal que $s_0 \in S$. É necessário ainda um conjunto de ações possíveis, A, que alteram o estado da sociedade de agentes. Essas ações podem tanto ser individuais (realizadas por um dado agente) quanto coletivas (realizada pela sociedade inteira). Cada ação é descrita segundo uma lista de pré-condições, um custo de tempo e as alterações que são aplicadas no estado após a ação concluir. É preciso também explicitar o intervalo de medição, ou o intervalo de tempo, I, em que os agentes terão seu desempenho mensurado. E, por fim, se faz necessário um conjunto de métricas, M, para avaliar os agentes. Existem diversas métricas que podem compor este conjunto, como por exemplo o intervalo máximo (CHEVALEYRE; SEMPE; RAMALHO, 2004) e a ociosidade média (HERNÁNDEZ; CERRO; BARRIENTOS, 2013).

Sendo assim, é possível concluir que o problema da Patrulha Multiagente é, fundamen-

talmente, um problema de otimização, onde é preciso escolher as ações dos agentes com a finalidade de minimizar (ou maximizar) uma dada métrica (SAMPAIO, 2013).

A Computação Evolucionária engloba uma classe de algoritmos que realiza uma busca no espaço de soluções à procura daquela que seja ótima (LUKE, 2013). Esses algoritmos são baseados na Teoria da Evolução das Espécies proposta por Charles Darwin [1]. Eles são chamados de Algoritmos Evolucionários (EA), pois buscam, dentro de uma população, o indivíduo melhor adaptado através de operadores chamados de mutação, recombinação (também conhecido como cruzamento) e seleção. Dessa forma, será mostrado no presente trabalho que EAs podem ser utilizados para resolver a TMAP.

Apesar de possuir diversas soluções com diferentes abordagens para o problema (CHEVALEYRE; SEMPE; RAMALHO, 2004), (MACHADO et al., 2003), (ALMEIDA et al., 2004), (ELMALIACH; AGMON; KAMINKA, 2007), (HERNÁNDEZ; CERRO; BARRIENTOS, 2013), não existem soluções ótimas ou claramente dominantes para a TMAP. Embora os algoritmos evolucionários sejam amplamente utilizados em outros problemas de otimização, soluções para a TMAP que utilizam esses algoritmos são pouco encontradas na literatura (LAURI; KOUKAM, 2008), (LAURI; KOUKAM, 2014) e, por tanto, serão o objeto de pesquisa do presente trabalho.

1.1 Objetivos Gerais

Motivado pelos problemas descritos acima, este trabalho tem como objetivo geral a proposição e o estudo de soluções para TMAP utilizando algoritmos evolucionários. Outro objetivo é comprovar a eficácia e avaliar a relevância de abordagens evolucionárias para o problema da patrulha através de experimentos empíricos e da comparação destas soluções com as obtidas através de abordagens de destaque publicadas na literatura.

1.2 Objetivos Específicos

Tendo em vista tais objetivos gerais, este projeto visa atingir os seguintes objetivos específicos:

- Investigar estratégias evolucionárias e como elas podem ser aplicadas ao problema da patrulha multiagente temporal;
- Desenvolver uma estratégia da TMAP baseada em algoritmos evolucionários;
- Implementar as estratégias desenvolvidos em um simulador capaz de computar diversas métricas para uma dada solução da TMAP.

[1] http://en.wikipedia.org/wiki/Evolution

- Avaliar o desempenho das estratégias no simulador e verificar eficácia da abordagem através de comparações empíricas com algumas estratégias de destaque publicadas na literatura.

2

Trabalhos Relacionados

Neste capítulo serão revisitados alguns trabalhos realizados sobre a TMAP. Primeiramente, serão feitas algumas considerações sobre os diferentes nomes, escopos e métricas utilizados para o problema pelos autores. Depois, serão revisadas duas das definições formais mais utilizadas para o problema. Em seguida, serão apresentadas as classificações para estratégias que visam resolver a TMAP. Por fim, serão descritas as soluções já presentes na literatura.

2.1 Considerações Iniciais

Um dos grandes obstáculos para o estudo do Problema da Patrulha Multiagente é a falta de concordância, dentre os pesquisadores, sobre nomenclatura, escopo e critérios de avaliação para as soluções propostas. Isso pode se dever ao fato do problema da patrulha estar presente em diferentes áreas como Inteligência Artificial, Sistemas Multiagentes e até Robótica (SAMPAIO, 2013).

Há, na literatura, nomes diferentes para o problema da Patrulha Multiagente. Por exemplo, alguns autores (HERNÁNDEZ; CERRO; BARRIENTOS, 2013) chamam o problema de "patrulha multirobô" (ou Multi-Robot Patrolling, em inglês), outros (POULET; CORRUBLE; SEGHROUCHNI, 2012a) se referem ao problema pelo nome de "patrulha temporal" (tradução de timed patrolling), já alguns pesquisadores (KOENIG; LIU, 2001) utilizam o termo "cobertura de terreno" (terrain coverage, em inglês). No entanto, no decorrer do presente trabalho será utilizado o termo "Patrulha Multiagente", pois ele é considerado mais apropriado por diversos pesquisadores (LAURI; KOUKAM, 2014), (SAMPAIO, 2013), e (ALBERTON et al., 2012).

O escopo é outro fator que varia entre os trabalhos. (DOI, 2013), por exemplo, faz uma análise do problema da patrulha onde o ambiente é dinâmico. Isto é, o grafo onde os agentes patrulham sofre alterações ao longo da execução do agente. Em um dado momento, o grafo pode ter vértices adicionados ou removidos. Já (LAURI; KOUKAM, 2014) trabalham levando em conta que agentes podem ter velocidades diferentes. Os autores de (PIPPIN; CHRISTENSEN; WEISS, 2013) levam em consideração que agentes podem agir com eficiência abaixo do esperado, isto é, o agente não pode ser confiado para realizar a tarefa que lhe foi passada sem falhas. Em

(ELMALIACH; AGMON; KAMINKA, 2007), os autores consideram restrições de frequência, isso significa que cada nó tem a si designado um valor de frequência com a qual o nó **deve** ser visitado. Outro exemplo seria o trabalho feito em (POULET; CORRUBLE; SEGHROUCHNI, 2012a) e (POULET; CORRUBLE; EL FALLAH SEGHROUCHNI, 2012b), onde é feita uma análise do problema em uma configuração de sistema aberto. Sistemas abertos foram definidos em (POULET et al., 2011) como aqueles onde agentes podem entrar e sair a qualquer momento da execução.

Na presente pesquisa, será utilizado o escopo para a Patrulha Multiagente doravante referido como "padrão", onde os agentes possuem eficiência idêntica, são confiáveis e não são retirados nem adicionados ao longo da patrulha. Quanto ao ambiente, o objeto de estudo deste trabalho compreende apenas ambientes que permanecem estáticos durante a execução dos agentes, e os pontos que devem ser patrulhados não possuem restrições de frequência.

Existem diversas métricas que podem ser utilizadas para medir a eficiência de cada solução para a TMAP. Diferentes pesquisadores utilizam métricas distintas. Em (MACHADO et al., 2003) foram propostas as métricas mais utilizadas para a TMAP. São elas: ociosidade instantânea do nó, ociosidade instantânea do grafo, ociosidade máxima e tempo de exploração. Depois deste trabalho, (SAMPAIO, 2013) propôs uma nova família de métricas baseadas nos intervalos entre visitas. As métricas utilizadas nos trabalhos mais recentes variam: (LAURI; KOUKAM, 2014) e (PIPPIN; CHRISTENSEN; WEISS, 2013) utilizam o intervalo máximo, já (ELMALIACH; AGMON; KAMINKA, 2007) fazem uso da frequência mínima e (HERNÁNDEZ; CERRO; BARRIENTOS, 2013) compara as frequências mínimas.

Dessa forma, este capítulo visa, através de pesquisa bibliográfica, explorar os trabalhos relacionados para construir a base de conhecimento necessária para formular uma proposta compatível com o objetivo deste projeto. Serão discutidas as soluções para o problema da patrulha presentes na literatura classificando-as por seu escopo e métricas utilizadas na avaliação dos agentes.

2.2 Levantamento Bibliográfico

2.2.1 Definições

O problema da Patrulha é tipicamente modelado por um Grafo (ROSEN, 2006). Segundo (ALMEIDA et al., 2004), isso se deve ao fato de que representar o problema por meio de um grafo faz com que o problema possa ser facilmente adaptado para um variedade de domínios desde terrenos até redes de computadores. (SAMPAIO, 2013) também considera que os grafos sejam o modelo preferencial para os ambientes da TMAP, pois são suficientemente poderosos para capturar as características do terreno mais relevantes para o problema. Essa capacidade do grafo pode ser confirmada por estudos que adotam grafos como modelo e posteriormente aplicam suas soluções em ambientes realistas contínuos (PIPPIN; CHRISTENSEN; WEISS,

2013).

Em (CHEVALEYRE; SEMPE; RAMALHO, 2004), o problema é matematicamente definido da seguinte forma:

O território onde os agentes patrulham é representado por um grafo $G(V,E)$, onde V é o conjunto dos pontos de interesse que precisam ser patrulhados e E representa o conjunto de arestas, $E \in V^2$, de G. Para toda aresta $(i,j) \in E$, onde $i,j \in V$, corresponde um peso $c_{i,j}$ representando a distância entre o vértices i e j em G.

Nesse cenário, uma solução monoagente para o problema seria uma função $\pi : \mathbb{N} \longrightarrow V$, tal que $\pi(j)$ é o j-ésimo vértice visitado pelo agente, desde que $\pi(j+1) = x$ se, e somente se, $(\pi(j),x) \in E$. Analogamente, uma solução multiagente seria um conjunto $\Pi = \{\pi_1 ... \pi_r\}$, onde r é o número de agentes.

Dessa forma, o problema seria encontrar o conjunto Π que obtivesse os melhores resultados de acordo com um certo critério de avaliação.

Já (SAMPAIO, 2013) contribuiu com uma definição mais abrangente da TMAP. Ele aponta que uma instância da TMAP pode ser completamente definida pela seguinte tupla:

$$\langle E,P,S,s_0,A,I,M \rangle$$

O elemento E representa o ambiente (do inglês, *environment* onde estão os pontos de interesse a serem patrulhados, preferencialmente E deve ser um grafo $G(V,E)$. O elemento P é o conjunto dos pontos de interesse do ambiente. No caso de um grafo, $P = V$.

S é um elemento um pouco mais complexo, pois ele representa o conjunto de possíveis estados da sociedade de agentes. Cada estado dentro desse conjunto pode conter informações tais como, quantos agentes estão ativos, qual a posição atual de cada agente no ambiente E, o tempo decorrido desde o início da patrulha, orientação e energia de cada agente e características globais do conjunto de agentes. O estado inicial da sociedade é representado em s_0, ou seja, $s_0 \in S$. Esta modelagem para o grupo de agentes permite ao modelo de (SAMPAIO, 2013) englobar diversos escopos da TMAP, como por exemplo os citados na Seção 2.1.

O quinto elemento da tupla, A, é o conjunto de ações que alteram o estado da sociedade, S. Essas ações podem ser individuais, de cada agente, ou coletivas, da sociedade inteira. O autor afirma que A deve conter, no mínimo, duas ações individuais definidas para cada agente: movimentação entre os pontos de P e visitação aos elementos do conjunto.

I representa o intervalo de medição, dentro do qual o desempenho dos agentes é mensurado através das métricas definidas no conjunto M, último elemento da tupla.

O problema da patrulha multiagente seria, então, definir um conjunto A de ações a serem tomadas pelos agentes, que estão inicialmente no estado s_0, para minimizar as métricas presentes em M durante o período de patrulha, I.

Uma vez que a TMAP está bem definida, a seção seguinte irá abordar as diferentes formas de classificar soluções para a TMAP, de forma que ao final, as abordagens a serem

propostas neste trabalho podem ser classificadas.

2.2.2 Classificações para a TMAP

São notórios alguns trabalhos muito citados na literatura que visaram classificar as soluções para a TMAP.

Em (CHEVALEYRE; SEMPE; RAMALHO, 2004), as abordagens propostas até então foram classificadas em **cíclicas** (ou de ciclo único) e **baseadas em particionamento**. As soluções de ciclo único são aquelas onde é calculado um ciclo que cobre todos os vértices do grafo, e então, os agentes são colocados para caminhar nesse ciclo indefinidamente. As abordagens baseadas em particionamento são aquelas onde o território a ser patrulhado é dividido em r regiões, onde r é o número de agentes. Os agentes devem, então, patrulhar dentro de suas respectivas regiões.

Já (MACHADO et al., 2003) faz uma extensa classificação das arquiteturas de soluções para a TMAP. O trabalho divide os agentes em: **reativo** ou **cognitivo**; com comunicação **permitida** ou **proibida**; com coordenação **central** ou **descentralizada**; com percepção **local** ou **global**; com tomada de decisão **aleatória** ou **orientada a objetivo**. Os agentes reativos são aqueles que agem baseados apenas na sua percepção atual do território, enquanto que os agentes cognitivos podem perseguir um determinado objetivo. Enquanto os agentes cognitivos tem uma visão do grafo completo, os reativos só podem enxergar os nós adjacentes ao que eles se encontram. A comunicação se refere a possibilidade dos agentes trocarem informações. A percepção se refere ao quanto de informação o agente pode acessar sobre o ambiente ao seu redor e sobre os outros agentes. Finalmente, no esquema de coordenação centralizado, uma entidade central escolhe nó-objetivo de cada agente, enquanto que no esquema descentralizado, a coordenação emerge da interação entre os agentes.

As abordagens propostas na presente pesquisa podem ser classificadas como estratégias **baseadas em particionamento**, com agentes **cognitivos**, de comunicação **proibida**, com coordenação **centralizada**, de percepção **global**, e com tomada de decisão **orientada a objetivo**.

Com o problema bem descrito, pode-se explorar as soluções que já estão presentes na literatura e verificar se há espaço para a proposição de mais estratégias para a TMAP.

2.2.3 Soluções já propostas

(CHEVALEYRE; SEMPE; RAMALHO, 2004) propuseram uma solução para a TMAP baseada no Problema do Caixeiro Viajante (TSP). Segundo os autores, a solução mais simples seria encontrar um ciclo que percorresse todos os pontos de interesse do terreno (de preferência sem repetição de pontos dentro do ciclo) e então fazer o agente percorrer esse ciclo indefinidamente. Esta estratégia corresponde a encontrar um ciclo que percorre todos os nós do grafo sem repeti-los, um Ciclo Hamiltoniano. O trabalho demonstra que para um único agente a solução ótima, para a métrica da ociosidade máxima, é esta estratégia cíclica baseada no TSP.

No entanto, o TSP é conhecidamente um problema NP-Completo. Dessa forma, os autores propõem uma solução utilizando o algoritmo de Christofides (CHRISTOFIDES, 1976) para obter uma aproximação do TSP em tempo polinomial. A pesquisa ainda aponta que, para o caso multiagente, a solução seria calcular o ciclo, posicionar os agentes com um certo intervalo entre um e outro e colocá-los para percorrer o ciclo indefinidamente. Os autores, no entanto, não estudam a aplicação dessa solução ao problema com outras métricas que não a da ociosidade máxima.

Posteriormente, (ALMEIDA et al., 2004) realizam um estudo comparativo entre estratégias centralizadas e descentralizadas. Eles concluem que a solução proposta por (CHEVALEYRE; SEMPE; RAMALHO, 2004) é a mais eficiente dentre as estudadas por eles. Os autores também chegam à conclusão de que as estratégias centralizadas estudadas, apesar de terem obtido resultados melhores nos experimentos, devem sofrer mais com problemas de escalabilidade a medida que o grafo cresce.

A partir das conclusões tiradas nesses estudos, (ELMALIACH; AGMON; KAMINKA, 2007) propõem uma solução centralizada e cíclica, onde ao invés de utilizar o algoritmo de Christofides, se utiliza o método *Spanning Tree Coverage*, introduzido em (GABRIELY; RIMON, 2001), para gerar um Ciclo Hamiltoniano em um grafo com formato de *grid*. Assim como a solução de (CHEVALEYRE; SEMPE; RAMALHO, 2004), após o ciclo ser calculado, os vários agentes são posicionados no grafo e colocados para percorrer o ciclo indefinidamente.

Além dessas soluções de ciclo único, também foram propostas, na literatura, soluções baseadas em particionamento. Os autores (PIPPIN; CHRISTENSEN; WEISS, 2013) propõem uma solução, onde, na inicialização, os nós do grafo são particionados entre os agentes e esses calculam um ciclo fechado (*closed cycle*) dentro da respectiva partição. Uma entidade de monitoramento central fica responsável por verificar, ao longo da execução, a performance dos agentes para a métrica de intervalo máximo entre visitas e identificar agentes que estejam com baixo desempenho. Então, alguns nós do agente de baixo rendimento são oferecidos aos outros agentes e redesignados através de um protocolo baseado em leilão.

Visando a escalabilidade apontada por (ALMEIDA et al., 2004), foram propostas diversas soluções descentralizadas. Dentre elas, destacam-se as baseadas em colônias de formigas (KOENIG; LIU, 2001), (ELOR; BRUCKSTEIN, 2010), (DOI, 2013). Nessas soluções, os agentes não possuem uma visão completa do grafo, mas apenas das suas vizinhanças. A coordenação dos agentes é emergente (MACHADO et al., 2003), pois cada agente pode deixar marcações em nós do grafo que podem ser sentidas pelos outros agentes.

Apesar de serem amplamente utilizados na literatura para resolver problemas de otimização, os Algoritmos Evolucionários (LUKE, 2013) foram pouco utilizados no problema da Patrulha. (LAURI; KOUKAM, 2008) propõe uma solução híbrida composta de dois passos: no primeiro, um algoritmo genético para posicionar os agentes o mais distante possível dentro do território, e então, utiliza *Ant Colony Optimization* (ACO) para calcular partições no grafo por onde os agentes irão patrulhar durante a execução. Dessa forma, pode-se constatar que o

algoritmo evolucionário não está sendo usado para resolver o problema da patrulha em si, mas para encontrar os r nós mais distantes uns dos outros dentro do grafo, onde r é o número de agentes.

2.3 Considerações Finais

Uma vez apresentados, neste capítulo, os conceitos básicos dos trabalhos relacionados, é possível fazer algumas observações sobre tendências dentre as publicações. Há uma predileção pela Simulação como método de comparação e de demonstração de soluções. É possível também perceber que muitos trabalhos utilizam as métricas propostas por (MACHADO et al., 2003) e (SAMPAIO, 2013).

Os trabalhos comparativos feitos por (ALMEIDA et al., 2004) e (SAMPAIO, 2013) mostram que não existem estratégias claramente dominantes ou ótimas para TMAP. Percebe-se também que falta à literatura um trabalho que investigue uma solução para a TMAP utilizando algoritmos evolucionários, tais como as Estratégia Evolucionárias (ESs) e os Algoritmo Genéticos (GAs). Dessa forma, nos próximos capítulos, serão apresentados alguns algoritmos evolucionários que podem ser utilizados para resolver a TMAP, em seguida, serão propostas formas de aplicar esses algoritmos no contexto do problema e, finalmente, será demonstrada a eficácia dessas estratégias perante as abordagens de destaque propostas por outros autores.

O Quadro 2.1 resume as soluções apresentadas nesta sessão de acordo com o escopo do problema, o método de demonstração das soluções e as métricas utilizadas.

Quadro 2.1: Resumo dos trabalhos relacionados

Trabalho	Escopo	Método	Métrica(s)
(MACHADO et al., 2003)	Patrulha Multiagente "Padrão"	Simulação	Ociosidade instantânea do nó, ociosidade instantânea do grafo, ociosidade do grafo, ociosidade máxima e tempo de exploração
(CHEVALEYRE; SEMPE; RAMALHO, 2004)	Patrulha Multiagente "Padrão"	Análise Matemática	Pior ociosidade (*worst idleness*)
(ALMEIDA et al., 2004)	Patrulha Multiagente "Padrão"	Simulação	Ociosidade do grafo, Ociosidade média normalizada e Desvio padrão da ociosidade média
(ELMALIACH; AGMON; KAMINKA, 2007)	Patrulha Multiagente com restrição de frequência	Análise Matemática	Frequência máxima uniforme
(LAURI; KOUKAM, 2008)	Patrulha Multiagente "Padrão"	Simulação	Ociosidade Máxima
(POULET; CORRUBLE; SEGHROUCHNI, 2012a)	Patrulha Multiagente como um sistema aberto	Simulação	Intervalo médio entre visitas, Intervalo quadrático médio, tempo de estabilização e amplitude das variações
(PIPPIN; CHRISTENSEN; WEISS, 2013)	Patrulha Multiagente com agentes que possuem performances distintas	Simulação e Experimentação com robôs reais	Intervalo Máximo entre visitas
(DOI, 2013)	Patrulha Multiagente em ambientes dinâmicos	Simulação	Intervalo Máximo entre visitas
(HERNÁNDEZ; CERRO; BARRIENTOS, 2013)	Patrulha Multiagente "Padrão"	Simulação	Ociosidade total

Fonte: O autor

3

Algoritmos Evolucionários

Sendo o objetivo do presente trabalho apresentar abordagens evolucionárias para a TMAP, este capítulo tem como finalidade apresentar ao leitor alguns conceitos e terminologias da área da Computação Evolucionária, para que ele esteja familiarizado antes do capítulo sobre a aplicação de algoritmos evolucionários no contexto da TMAP. Este capítulo também vai apresentar os algoritmos que foram utilizados como base para esta pesquisa.

Segundo (BÄCK; SCHWEFEL, 1993), várias pesquisas mostraram que modelar o processo de busca de forma similar à evolução pelo qual os seres vivos passaram pode render algoritmos robustos, mesmo que estes modelos sejam apenas representações imperfeitas do verdadeiro processo biológico. O resultado desses modelos são chamados de Algoritmos Evolutivos ou Evolucionários. Essa busca pode ser aplicada para encontrar não apenas uma solução qualquer, mas aquela que minimize ou maximize uma dada métrica. Dessa forma, algoritmos evolucionários podem (e são) utilizados em problemas de otimização.

Tais algoritmos são baseados no processo de aprendizado coletivo pelo qual passa uma população de indivíduos. A cada geração, os indivíduos mais aptos passam para os filhos uma combinação das características que os tornam aptos. Então, ao longo das gerações, além dos indivíduos menos aptos serem extintos, novos indivíduos ainda mais aptos que os pais são formados por esse processo de reprodução (BÄCK; SCHWEFEL, 1993).

Na Computação Evolutiva, os indivíduos são as soluções para um determinado problema, ou um ponto no espaço de possíveis soluções. Na TMAP, por exemplo, um indivíduo seria uma sociedade de agentes e suas estratégias de patrulha. O ambiente seria o problema em si. Então, a aptidão de um indivíduo seria calculada através de uma medida de avaliação adequada ao problema.

Um algoritmo evolucionário funciona, genericamente, da seguinte forma: uma população inicial é arbitrariamente inicializada; esses indivíduos têm sua adaptação ao ambiente medida; eles são, posteriormente, recombinados para formar uma nova população, podendo também sofrer mutação; finalmente, um subconjunto dessas populações (antiga e nova, pais e filhos) é selecionado de alguma forma definida pelo algoritmo e se torna a população da próxima geração. Esse ciclo se repete tipicamente até que parem de surgir melhores indivíduos que aqueles já

presentes na população, evento chamado de convergência do algoritmo (BÄCK; SCHWEFEL, 1993).

A população inicial pode ser criada de forma aleatória, ou pode-se utilizar conhecimentos sobre o problema para inicializar uma população onde os indivíduos estão em uma região "boa"(de acordo com a métrica) do espaço de soluções (LUKE, 2013).

O Pseudocódigo 1 abaixo exemplifica de forma genérica um algoritmo evolucionário.

Pseudocódigo 1 Algoritmo Evolucionário Genérico

```
Procedimento EA
    P ← Constrói-População-Inicial
    Repita
        P' ← Recombina(P)
        P'' ← Aplica-Mutação(P')
        P ← Seleciona(P'')
    Até que não tenhamos mais tempo
Fim
```

Fonte: Adaptado de (BÄCK; SCHWEFEL, 1993)

O Quadro 3.1 revisa alguns dos termos comumente utilizados nos EAs.

No restante do presente capítulo, serão apresentados os algoritmos utilizados nesta pesquisa, as Estratégias Evolucionárias e os Algoritmos Genéticos. Este trabalho utilizou duas variações, propostas na literatura, para cada um deles. Estas variações também serão apresentadas nas demais seções.

3.1 Estratégias Evolucionárias

As duas estratégias evolucionárias usadas no presente trabalho diferem entre si pela forma como fazem a composição entre a população de pais e a população de filhos para construir a nova geração, que será utilizada na iteração seguinte do algoritmo.

A primeira ES é conhecida como (μ, λ). Tipicamente, começa-se com uma população de λ indivíduos gerados de forma arbitrária. Nessa ES, o μ representa o número de pais cujos filhos serão usados para compor a próxima população que também deve ter λ indivíduos no total. Note que λ tem que ser um múltiplo de μ. Então, os μ indivíduos mais aptos são escolhidos, processo chamado de Seleção por Truncamento (em inglês, *Truncate Selection*). Os indivíduos selecionados sofrem mutação para gerar λ/μ filhos cada . O que acarretará em uma nova população de λ indivíduos que será a geração utilizada na próxima iteração do algoritmo (LUKE, 2013).

O Pseudocódigo 2 exemplifica a Estratégia Evolucionária (μ, λ).

A segunda ES utilizada nesta pesquisa é chamada de $(\mu + \lambda)$. Enquanto que na estratégia (μ, λ) cada pai é substituído pelos seus λ/μ filhos, na estratégia $(\mu + \lambda)$, os μ pais permanecem

Pseudocódigo 2 Estratégia Evolucionária (μ, λ)

```
Procedimento μ_λ_ES(μ, λ)
    P ← {}
    Para 1...λ Faça
        p ← {novo indivíduo gerado de forma arbitrária}
        CalculaAptidão(p)
        P ← P ∪ {p}
    Fim
    Melhor ← nulo
    Repita
        Ordena(P)                          ▷ Ordena a população de acordo com a aptidão
        Q ← P_{1...μ}                      ▷ Inicia Q com os μ indivíduos mais aptos
        P ← {}
        Para Q_j ∈ Q Faça
            Para 1...λ/μ Faça
                P ← P ∪ {Aplica-Mutação(Q_j)}
            Fim
        Fim
        Para P_i ∈ P Faça
            CalculaAptidão(P_i)
            Se Melhor = nulo ou Aptidão(P_i) > Aptidão(Melhor) Então
                Melhor ← P_i
            Fim
        Fim
    Até que não tenhamos mais tempo
    Retorne Melhor
Fim
```

Fonte: Adaptado de (LUKE, 2013)

Quadro 3.1: Termos comuns na Computação Evolucionária

Termo	**Significado**
Indivíduo	Uma solução candidata
Filha e Pai	Uma solução Filha é uma cópia alterada de uma solução Pai
População	Um conjunto de soluções candidatas
Aptidão	Qualidade de um indivíduo (Solução)
Seleção	Coletar indivíduos baseados na sua aptidão
Mutação	Realização de pequenas alterações em indivíduos. Também chamada de reprodução assexuada
Recombinação (em inglês, *crossover*)	Uma forma de alteração especial que recebe como parâmetro duas soluções pais e (normalmente) produz duas soluções filhas
Reprodução	O ato de produzir uma ou mais soluções filhas a partir de uma solução pai
Geração	Um ciclo de medida de aptidão ou de reprodução de uma população

Fonte: Adaptado de (LUKE, 2013)

para competir com seus λ filhos na iteração seguinte. (LUKE, 2013) aponta que isso geralmente faz com que a estratégia $(\mu + \lambda)$ explore mais os ótimos locais em comparação com a estratégia (μ, λ), já que um pai suficientemente apto pode fazer com que a ES fique "presa" em seus descendentes imediatos, causando uma convergência das populações para o ótimo local ao redor do pai.

O Pseudocódigo 3 ilustra a Estratégia Evolucionária $(\mu + \lambda)$.

É importante notar que as Estratégias Evolucionárias não utilizam a recombinação para gerar novos indivíduos. Para isso, elas utilizam apenas a Mutação cuja implementação depende do tipo de dados envolvido no problema que está sendo estudado. No próximo capítulo, serão propostos alguns operadores de Mutação para a TMAP, que irão agir sobre caminhos e ciclos de um grafo. Também será tratado, no próximo capítulo, como foram implementadas as funções de gerar indivíduos e calcular suas aptidões dentro do contexto da TMAP.

3.2 Algoritmos Genéticos

Os Algoritmos Genéticos são similares às ESs: o seu *loop* principal consiste em selecionar indivíduos da polução de acordo com as respectivas aptidões, reproduzir os indivíduos selecionados e iterar sobre a nova população para calcular as novas aptidões (LUKE, 2013).

No entanto, eles são diferentes na forma como selecionam e reproduzem os indivíduos. Enquanto que nas Estratégias Evolucionárias, todos os pais são selecionados simultaneamente e passam por mutação em seguida, os Algoritmos Genéticos selecionam os pais e geram filhos ao

Pseudocódigo 3 Estratégia Evolucionária $(\mu + \lambda)$

```
Procedimento μ + λ_ES(μ, λ)
    P ← {}
    Para 1...λ Faça
        p ← {novo indivíduo gerado de forma arbitrária}
        CalculaAptidão(p)
        P ← P ∪ {p}
    Fim
    Melhor ← nulo
    Repita
        Ordena(P)                          ▷ Ordena a população de acordo com a aptidão
        Q ← P_{1...μ}                      ▷ Inicia Q com os μ indivíduos mais aptos
        P ← Q                              ▷ A diferença está aqui
        Para Q_j ∈ Q Faça
            Para 1...λ/μ Faça
                P ← P ∪ {Aplica-Mutação(Q_j)}
            Fim
        Fim
        Para P_i ∈ P Faça
            CalculaAptidão(P_i)
            Se Melhor = nulo ou Aptidão(P_i) > Aptidão(Melhor) Então
                Melhor ← P_i
            Fim
        Fim
    Até que não tenhamos mais tempo
    Retorne Melhor
Fim
```

Fonte: Adaptado de (LUKE, 2013)

poucos até que se tenha uma população de filhos suficiente. A reprodução em si também é bem diferente das ESs: nos Algoritmos Genéticos, inicia-se com uma população de filhos vazia. O Algoritmo então seleciona dois pais de forma arbitrária, recombina esses pais em dois novos indivíduos filhos e então faz a mutação deles (as ESs só realizam a mutação). Esses dois novos indivíduos são adicionados à população. Esse processo se repete até que a população de filhos esteja completamente preenchida (LUKE, 2013).

O pseudocódigo 4 deve ajudar o leitor a compreender o algoritmo.

Pseudocódigo 4 Algoritmo Genético

```
Procedimento ALGORITMOGENETICO(tamanhoPopulao)
    P ← {}
    Para 1... tamanhoPopulao Faça
        p ← {novo indivíduo gerado de forma arbitrária}
        CalculaAptidão(p)
        P ← P ∪ {p}
    Fim
    Melhor ← nulo
    Repita
        Q ← {}
        Para 1... tamanhoPopulao/2 Faça
            Pai_a ← Seleciona(P)
            Pai_b ← Seleciona(P)
            Filho_a, Filho_b ← Recombina(Pai_a, Pai_b)
            Q ← Q ∪ {Aplica-Mutação(Filho_a), Aplica-Mutação(Filho_b)}
        Fim
        P ← Q
        Para P_i ∈ P Faça
            CalculaAptidão(P_i)
            Se Melhor = nulo ou Aptidão(P_i) > Aptidão(Melhor) Então
                Melhor ← P_i
            Fim
        Fim
    Até que não tenhamos mais tempo
    Retorne Melhor
Fim
```

Fonte: Adaptado de (LUKE, 2013)

Esta é a forma tradicional como o Algoritmo Genético é apresentado nos livros textos (LUKE, 2013), também conhecido como Algoritmo Genético Geracional. No entanto, existem outras variações, como por exemplo, o Algoritmo Genético de Estado Estável, do inglês, *Steady State Genetic Algorithm*. O GA Geracional é assim conhecido pois nele a população é completamente atualizada de uma vez. O GA de Estado Estável, por sua vez, introduz um ou dois filhos, obtidos através de recombinação e mutação, diretamente na população de indivíduos (matando outros indivíduos para abrir espaço) e segue para a próxima geração.

Dessa forma, assim como a ES $(\mu + \lambda)$, os indivíduos pais se mantém na população e disputam com os filhos nas gerações seguintes. Por isso, este algoritmo pode sofrer do mesmo problema e ficar "preso" a ótimos locais ao redor dos pais (LUKE, 2013).

O pseudocódigo 5 detalha as características do Algoritmo Genético de Estado Estável.

Pseudocódigo 5 Algoritmo Genético de Estado Estável

```
Procedimento ALGORITMOGENETICOESTADOESTAVEL(tamanhoPopulao)
    P ← {}
    Para 1... tamanhoPopulao Faça
        P ← P ∪ {novo indivíduo gerado de forma arbitrária}
    Fim
    Para P_i ∈ P Faça
        CalculaAptidão(P_i)
        Se Melhor = nulo ou Aptidão(P_i) > Aptidão(Melhor) Então
            Melhor ← P_i
        Fim
    Fim
    Repita
        Pai_a ← Seleciona(P)
        Pai_b ← Seleciona(P)
        Filho_a, Filho_b ← Recombina(Pai_a, Pai_b)
        Filho_a ← Aplica-Mutação(Filho_a)
        Filho_b ← Aplica-Mutação(Filho_b)
        CalculaAptidão(Filho_a)
        Se Aptidão(Filho_a) > Aptidão(Melhor) Então
            Melhor ← Filho_a
        Fim
        CalculaAptidão(Filho_b)
        Se Aptidão(Filho_b) > Aptidão(Melhor) Então
            Melhor ← Filho_b
        Fim
        p_c ← SelecionaParaMorrer(P)  ▷ Seleciona um indivíduo para remover da população
        p_d ← SelecionaParaMorrer(P)                                   ▷ p_c ≠ p_d
        P ← P − {p_c, p_d}
        P ← P ∪ {Filho_a, Filho_b}
    Até que não tenhamos mais tempo
    Retorne Melhor
Fim
```

Fonte: Adaptado de (LUKE, 2013)

O algoritmo genético de estado estável utilizado na presente pesquisa utiliza o operador de seleção chamado Seleção da Pior Aptidão (LUKE, 2013) para escolher os indivíduos da antiga geração que darão lugar aos novos. Esse operador simplesmente escolhe o indivíduo que possuir a menor aptidão dentro da população.

No próximo capítulo, serão explorados as aplicações dos algoritmos acima apresentados

para a TMAP. Serão propostos operadores de mutação e de recombinação que possam ser aplicados ao modelo utilizado, além de um método para calcular a aptidão de um indivíduo. No entanto, pode-se, ainda neste capítulo, tratar de um operador utilizado nos algoritmos genéticos que é definido sem interferência do problema ou da estrutura de dados no qual ele está modelado. Os operadores de Seleção podem ser descritos para uma população qualquer de indivíduos modelados de forma arbitrária.

Na presente pesquisa, foi utilizado o operador de mutação comumente chamado de Torneio (do inglês, *Tournament*) (LUKE, 2013). Ele consiste em coletar, da população, t indivíduos de forma aleatória. Destes, o algoritmo retorna apenas o indivíduo mais apto. Segue o pseudocódigo para ilustrar esse operador.

Pseudocódigo 6 Torneio

```
Procedimento TORNEIRO(P,t)
    Melhor ← indivíduo escolhido aleatoriamente de P        ▷ De forma que não possa ser escolhido uma segunda vez
    Para i de 2...t Faça
        Proximo ← indivíduo escolhido aleatoriamente de P
        Se Aptidão(Proximo) > Aptidão(Melhor) Então
            Melhor ← Proximo
        Fim
    Fim
    Retorne Melhor
Fim
```

Fonte: Adaptado de (LUKE, 2013)

Neste capítulo foram revisados quatro algoritmos evolucionários presentes na literatura que foram utilizados nesta pesquisa. Todos esses algoritmos são bases para resolver problemas de otimização discretos ou contínuos. No entanto, eles precisam ser complementados com operadores para: criação de indivíduos, mutação e recombinação. Estes operadores dependem do problema, influenciam diretamente na busca realizada por esses algoritmos e podem ser um fator determinante para o sucesso ou fracasso da abordagem evolucionária (LUKE, 2013).

Dessa forma, no capítulo seguinte, serão apresentados alguns operadores que podem ser utilizados para mutação, recombinação e criação de indivíduos no contexto da TMAP.

4

Abordagens Evolucionárias para a TMAP

No último capítulo, foram apresentados os algoritmos evolucionários usados nesta pesquisa. No entanto, algumas lacunas ficaram abertas: não foram apresentadas formas de gerar indivíduos, isto é, soluções para a TMAP, de forma arbitrária para a primeira população, além de como aplicar recombinação e mutação neles. A definição desses operadores passa primeiro pela forma escolhida para representar indivíduos. Tipicamente, na literatura de Algoritmos Evolucionários, indivíduos são representados como um vetor de bits ou de números reais (LUKE, 2013). No entanto, tal como foi apresentado na Seção 2.2.1, a TMAP não é representada por essas estruturas de dados.

Considerando estas questões de representação, esta pesquisa adota uma abordagem baseada em particionamento do grafo para resolver a TMAP, como foi sugerido por (CHEVALEYRE; SEMPE; RAMALHO, 2004) e aplicado por (PIPPIN; CHRISTENSEN; WEISS, 2013) e (LAURI; KOUKAM, 2008).

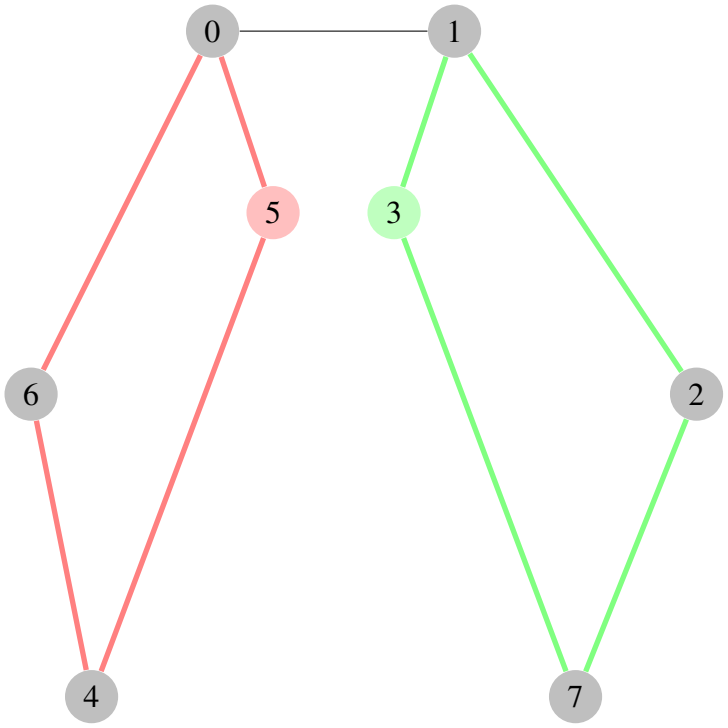
Cada agente pode ser representado por um ciclo que percorre todos os vértices de uma partição do grafo. Desta forma, uma solução para a TMAP seria representada (de forma finita) por um conjunto de r ciclos que percorrem r partições do grafo, onde r é o número de agentes. Com isso, é possível trocar agentes entre soluções, que como será mostrado na Seção 4.3, é a base para a recombinação.

A partir deste modelo, foi adicionado o conceito de centro das partições para que a recombinação possa trocar agentes semelhantes. Se duas soluções possuem agentes com os mesmos centros de partição, será possível trocar os agentes que possuam o mesmo centro (o que não quer dizer partições iguais).

Finalmente, a representação proposta para uma solução da TMAP é: um conjunto de r agentes, onde cada agente é representado por um ciclo que passa por todos os vértices de uma partição do grafo e um vértice de centro. A Figura 4.1, representa visualmente uma dessas soluções. Nela, dois agentes patrulham o grafo. Cada um ficou sob responsabilidade de uma partição, onde irão percorrer um ciclo indefinidamente. O agente vermelho, ficou com o centro 5, enquanto que o verde com o centro 3.

Utilizando esta representação, serão propostos, neste capítulo, operadores para construção

Figura 4.1: Exemplo de solução da TMAP



Fonte: O autor

de indivíduos, mutação e recombinação.

4.1 Criação de Indivíduos

A criação de indivíduos foi dividida em três etapas. A primeira é o cálculo de r centros, onde r é o número de agentes. A segunda tarefa é a escolha dos vértices que compõem a partição, dentre os quais está o centro definido para ela. Por fim, falta o cálculo de um ciclo, para cada partição, que passe por todos os seus vértices. Essas tarefas foram chamadas respectivamente de *centering*, *partitioning* e *path building*. No final, o resultado será um indivíduo (ou solução para a TMAP) com r agentes, onde cada agente tem um vértice de centro e um caminho para percorrer durante a patrulha. Estes operadores são utilizados pelos algoritmos evolucionários no momento em que se precisa de um "novo indivíduo gerado de forma aleatória"(vide os pseudocódigos no Capítulo 3).

Nas subseções seguintes, cada uma dessas três tarefas será tratada separadamente, mas é importante notar que uma tarefa depende da resposta dada pela sua antecessora. Serão apresentados mais de um operador que executa cada tarefa, assim ao montar um algoritmo evolucionário, o leitor pode construir seus indivíduos com qualquer combinação deles.

4.1.1 *Centering*

A primeira forma proposta para se calcular os centros é chamada de *Random Centering*, e consiste em simplesmente escolher nós aleatórios do grafo para ser os centros. Veja o Pseudocódigo 7 abaixo.

Pseudocódigo 7 *Random Centering*

```
Procedimento RANDOM-CENTERING(V, r)
▷ V é o conjunto de vértices do grafo e r é o número de agentes
    Vertices ← Cópia(V)
    Centros ← {}
    Para 1... r Faça
        Centros ← Centros ∪ {Seleciona-Aleatoriamente-Remove(Vertices)}
    Fim
    Retorne Centros
Fim
```

Fonte: O Autor

Uma outra forma de escolher os centros é tentar encontrar os vértices do grafo que estejam mais distantes uns dos outros. O objetivo neste operador é iniciar indivíduos de forma um pouco mais inteligente, para aumentar as chances de serem indivíduos com alta aptidão, mas sem perder completamente algum fator de aleatoriedade que é considerado importante nas inicializações de indivíduos nos algoritmos evolucionários. (LUKE, 2013) alerta para os perigos de utilizar heurísticas que aparentam gerar melhores indivíduos, mas que podem estar, inadvertidamente, condenando os algoritmos evolucionários a ficarem "presos" em mínimos locais.

O pseudocódigo 8 mostra como seria a implementação deste operador.

O operador começa selecionando r centros aleatoriamente, onde r é o número de agentes. Depois, ele calcula o somatório das distâncias entre todos os r vértices. Então, a cada iteração, um centro é escolhido aleatoriamente. Todos os sucessores do centro escolhido são testados no lugar do centro, se essa troca fizer com que os somatórios das distâncias entre os centros aumente, o sucessor fica no lugar do centro.

4.1.2 *Partitioning*

Seguindo a mesma ideia de ter uma opção completamente aleatória de criar os indivíduos, uma das formas propostas para calcular as partições é chamada de *Random Partitioning*. O pseudocódigo 9 exemplifica como isso pode ser implementado.

O *Random Partitioning* consiste em criar uma lista de partições para cada centro. A cada iteração do *loop* principal, um centro, c, é escolhido aleatoriamente. Então, um outro vértice, v, do grafo também é escolhido de forma aleatória (desde que não seja um centro). Depois, é

Pseudocódigo 8 *Approximated Maximum Distance Centering*

Procedimento APPROXIMATED-MAXIMUM-DISTANCE-CENTERING($G(V,E), r$)
▷ $G(V,E)$ é o grafo e r é o número de agentes
  $Vertices \leftarrow$ Cópia(V)
  $Centros \leftarrow$ RANDOM-CENTERING(V, r)
  $d \leftarrow$ Soma-Distancias($G(V,E)$, $centros$) ▷ Soma as distâncias entre os vértices
  Repita
    $centro \leftarrow$ escolhe um vértice aleatório de $centros$
    Para $V_i \in$ Sucessores($G(V,E)$, $centro$), onde $V_i \notin Centros$ **Faça**
      Substitui($Centros$, $centro$, V_i) ▷ Substitui $centro$ por V_i em $Centros$
      $d' \leftarrow$ Soma-Distancias($G(V,E)$, $centros$)
      Se $d > d'$ **Então**
        Substitui($Centros$, V_i, $centro$)
      Senão
        $d \leftarrow d'$
      Fim
    Fim
  Até que não tenhamos mais tempo
  Retorne $Centros$
Fim

Fonte: O Autor

Pseudocódigo 9 *Random Partitioning*

Procedimento RANDOM-PARTITIONING($centros, G(V,E)$)
▷ G é o o grafo e $centros$ é a lista de centros calculados anteriormente
  $Particoes \leftarrow \{\}$
  $verticesAdicionados \leftarrow \{\}$
  Repita
    $centro \leftarrow$ escolhe aleatoriamente um elemento de $centros$
    $candidato \leftarrow$ Seleciona-Aleatoriamente($V - centros$) ▷ $candidato \notin centros$
    $caminho \leftarrow$ Menor-Caminho(G, $centro$, $candidato$)
    $Particoes(centro) \leftarrow Particoes(centro) \cup caminho$
    $verticesAdicionados \leftarrow verticesAdicionados \cup caminho$
  Até que $|verticesAdicionados| = |V|$
  Retorne $Particoes$
Fim

Fonte: O Autor

adicionado à partição do centro c todo o menor caminho entre ele mesmo e o vértice v escolhido. O menor caminho inteiro é adicionado para que as partições estejam conectadas, isto é, para que seja possível traçar um caminho, dentro do grafo G, entre seus vértices. Finalmente, o *Random Partitioning* para quando todos os nós do grafo estiverem em alguma partição.

Como o próximo operador de particionamento, o *Heuristic Graph Partitioning* é mais complexo, primeiro será apresentado o pseudocódigo ao leitor, e então será feita uma discussão de suas propriedades. Observe o pseudocódigo 10.

Pseudocódigo 10 *Heuristic Graph Partitioning*

```
1: Procedimento GRAPH-PARTITIONING(centros, G(V,E))
   ▷ G é o o grafo e centros é a lista de centros calculados anteriormente
2:     ParticaoPorVertice ← {}
3:     ListaDeVerticesPorCentro ← {}
4:     ParticaoDoCentro ← {}
5:     Para i ∈ V Faça
6:         Se i ∈ centros Então
7:             ParticaoPorVertice[i] ← i
8:         Senão
9:             ParticaoPorVertice[i] ← −1
10:        Fim
11:    Fim
12:    Para centro ∈ centros Faça
13:        ListaDeVerticesPorCentro(centro) ← lista dos nós do grafo ordenados pelas suas
   distâncias ao centro
14:        ParticaoDoCentro(centro) ← {}
15:    Fim
16:    Repita
17:        Para C_i ∈ centros Faça
18:            Enquanto ListaDeVerticesPorCentro(C_i) ≠ {} Faça
19:                n ← REMOVE-PRIMEIRO(ListaDeVerticesPorCentro(C_i))
20:                Se ParticaoPorVertice(n) = −1 Então
21:                    ParticaoPorVertice(n) ← C_i
22:                    Pare o Laço Enquanto
23:                Fim
24:            Fim
25:        Fim
26:    Até que −1 ∉ ParticaoPorVertice        ▷ Até que todo vértice esteja em uma partição
27:    Para v ∈ V Faça
28:        ParticaoDoCentro(ParticaoPorVertice(v))∪ o menor caminho entre ParticaoPor-
   Vertice(v) e v
29:    Fim
30:    Retorne as partições em ParticaoDoCentro
31: Fim
```

Fonte: O Autor

O operador começa criando uma lista chamada *ParticaoPorVertice*. Nessa lista, cada índice representa um vértice, v, do grafo. O valor armazenado em $ParticaoPorVertice(v)$ corresponde ao centro ao qual o vértice v está associado. No final, todos os vértices associados a um centro formarão uma partição diferente. Na linha seguinte é iniciada uma lista bidimensional. O índice da primeira dimensão representa um centro, c, calculado na etapa anterior da construção de indivíduos, e $ListaDeVerticesPorCentro(c)$ representa uma lista de todos os vértices do grafo ordenados pelas suas distâncias a c. Na linha 4 é criada outra lista bidimensional, cujos índices também representam cada centro já calculado. *ParticaoDoCentro* é uma lista de partições por centro, e é a resposta do operador. $ParticaoDoCentro(c)$, então, vai guardar uma lista de todos os vértices que foram associados a c. Em breve, será mostrado porque é necessário manter ambas as listas *ParticaoDoCentro* e *ParticaoPorVertice*.

O operador segue para popular a lista *ParticaoPorVertice* com os elementos triviais. Cada centro, c, tem seu valor em *ParticaoPorVertice* configurado para si mesmo, enquanto que os outros vértices tem o valor configurado para -1. Na linha 13 é calculada a lista de vértices do grafo ordenadas pela distância ao centro e na linha 14 é iniciada a lista *ParticaoDoCentro* para cada um dos centros.

O *loop* iniciado na linha 16 só para quando todos os vértices estiverem associados a algum centro, dentro da lista *ParticaoPorVertice*. A cada iteração deste *loop*, é encontrado, para cada centro, c, o primeiro vértice na sua *ListaDeVerticesPorCentro* que ainda não possui um centro associado a si. Uma vez que esse vértice é encontrado, seu valor em *ParticaoPorVertice* é configurado para o centro c.

Quando todos os vértices possuem um valor diferente de -1 em *ParticaoPorVertice*, o operador entra em seu último *loop*. Nele, para cada vértice, v, do grafo é concatenada a lista que consiste no menor caminho entre v e o centro, c, a ele associado em *ParticaoPorVertice* com a lista de vértices na partição de c. Isso acontece para todas as partições possuam os vértices necessários para serem conectadas. E essa é a diferença entre as listas *ParticaoPorVertice* e *ParticaoDoCentro*.

4.1.3 *Path Building*

Uma vez de posse dos centros e das partições, falta construir um caminho, mais especificamente um ciclo, que permita ao agente caminhar por todos os nós de sua partição. A primeira abordagem a ser apresentada para tal finalidade, segue o mesmo princípio de ter uma opção totalmente aleatória, e é chamada de *Random Path Building*. Observe o pseudocódigo 11. Este procedimento deve ser executado para cada uma das partições calculadas na etapa de *Partitioning*.

Perceba que, como Q é uma lista cíclica, na última iteração do *loop* da linha 6, $Q[i+1]$ será o primeiro elemento de Q. Assim, o menor caminho entre o último e o primeiro elemento de Q será adicionado ao caminho final P. Isto fará com que P seja um ciclo, já que ele irá começar e terminar no primeiro elemento de Q.

Pseudocódigo 11 *Random Path Building*

```
1: Procedimento RANDOM-PATH-BUILDING(G(V,E), particao)
   ▷ G é o o grafo, particao é a lista de vértices que formam a partição de um agente
2:     G' ← subgrafo de G induzido por particao
3:     particao ← EMBARALHA(particao)  ▷ Embaralha a lista de vértices aleatoriamente
4:     Q ← LISTA-CICLICA(particao)
5:     P ← { }                                                    ▷ P é o caminho resultante
6:     Para i = 1 até |Q| Faça
7:         Se existe aresta em G' entre Q[i] e Q[i+1] Então
8:             ADICIONA-AO-CAMINHO(P, Q[i])
9:             ADICIONA-AO-CAMINHO(P, Q[i+1])
10:        Senão
11:            p ← menor caminho entre Q[i] e Q[i+1]
12:            Para v ∈ p Faça
13:                ADICIONA-AO-CAMINHO(P, v)
14:            Fim
15:        Fim
16:    Fim
17:    Retorne P
18: Fim
19: Procedimento ADICIONA-AO-CAMINHO(P, v)
20:    ultimo ← PEEK-LAST(P)          ▷ Retorna o último elemento de P sem removê-lo
21:    Se ultimo ≠ nulo e ultimo = v Então
22:        Retorne
23:    Senão
24:        P ← P ∪ {v}
25:    Fim
26: Fim
```

Fonte: O Autor

Já que o objetivo dos operadores de *Path Building* é construir um ciclo para o agente que passe por todos os nós da partição sob sua responsabilidade, é possível argumentar que o ciclo ótimo seria um Ciclo Hamiltoniano de custo mínimo. Dessa forma, os outros dois operadores para construção de caminhos são **baseados** em heurísticas para o Problema do Caixeiro Viajante com Múltiplas Visitas (TSPM) (GUTIN; PUNNEN, 2006). São elas: *Nearest Insertion Method* e *Nearest Neighbor Method* (NILSSON, 2003). Assim, os nomes dos operadores são: *Nearest Insertion Path Building* e *Nearest Neighbor Path Building*.

Algumas adaptações foram necessárias, já que o TSPM é definido sobre grafos completos. Foram calculados os menores caminhos entre todos os vértices do grafo, de forma que todas as referências às arestas do grafo feitas pelas heurísticas, foram convertidas, nos operadores aqui apresentados, para referências aos menores caminhos.

O pseudocódigo 12 ilustra o operador *Nearest Neighbor Path Building*. O operador começa por escolher um vértice aleatório da partição. Este será o primeiro elemento do caminho resultante, T. Então, até que todos os nós da partição estejam no caminho, ele irá escolher o vértice, v_s, ainda não adicionado com menor distância para o último ou primeiro elemento do caminho. Como mostram as linhas 27 a 38, o menor caminho entre v_s e o primeiro ou último elemento de T será adicionado a T. No final, como é necessário que T seja um ciclo, é adicionado o menor caminho entre o último e o primeiro elementos de T (linhas 41 a 44).

O pseudocódigo 13 ilustra uma possível forma de implementar o *Nearest Insertion Method*. O operador começa escolhendo um vértice, v, aleatório da partição. Depois, ele encontra outro vértice da partição, v_s, que tenha a menor distância para v e calcula um caminho que vá de v até v_s e volte para v. Após essa etapa, o operador entra em *loop*, até que todos os vértices da partição tenham sido adicionados ao caminho, T. A cada iteração, o operador encontra o vértice v_k, com menor distância para o caminho sendo construído. Depois, ele encontra a melhor aresta, (v_i, v_j), de T que possa ser quebrada para que o caminho vá de v_i até v_k e volte para v_j. A "melhor"aresta é dada pela equação $\Delta f = d(v_i, v_k) + d(v_k, v_j) - d(v_i, v_j)$, onde d é uma função que calcula a menor distância entre dois vértices. Observe que Δf corresponde ao aumento no custo do caminho caso se venha a quebrar a aresta (v_i, v_j) para ir até v_k e voltar. Assim, no final, o operador retorna um ciclo que vai de v, passa por todos os vértices da partição e volta.

4.2 Mutação

Uma vez que os operadores para criação arbitrária de indivíduos estejam apresentados, fica faltando os operadores de mutação e recombinação para que se possa executar um algoritmo evolucionário completo no contexto da TMAP. Nesta seção, por tanto, serão apresentados dois operadores diferentes para aplicar mutação em indivíduos.

No entanto, antes de descrever os operadores, será mostrado um procedimento que é utilizado em ambos. Ao final de sua operação, as mutações aqui propostas aplicam um proce-

Pseudocódigo 12 *Nearest Neighbor Path Building*

```
1: Procedimento NEAREST-NEIGHBOR-PATH-BUILDING(G(V,E), particao)
   ▷ G é o o grafo, particao é a lista de vértices que formam a partição de um agente
2:     T ← {}                                                    ▷ P é o caminho resultante
3:     P ← Cópia(particao)
4:     v ← vértice aleatório de P
5:     P ← P − {v}
6:     T ← T ∪ {v}
7:     Enquanto P ≠ {} Faça
8:         primeiro ← T[0]
9:         ultimo ← T[|T|]
10:        v_s ← nulo                                            ▷ O vértice a ser adicionado
11:        d_min ← ∞
12:        addNoFinal? ← falso
13:        Para j ∈ 1 ... |P| Faça
14:            v_j ← P[j]
15:            d_j,primeiro ← distância entre o vértice v_j e o vértice primeiro
16:            d_j,ultimo ← distância entre o vértice v_j e o vértice ultimo
17:            Se d_j,primeiro < d_min Então
18:                d_min ← d_j,primeiro
19:                v_s ← v_j
20:            Fim
21:            Se d_j,ultimo < d_min Então
22:                d_min ← d_j,ultimo
23:                v_s ← v_j
24:                addNoFinal? ← verdadeiro
25:            Fim
26:        Fim
27:        Se addNoFinal? Então
28:            P_ultimo,s ← menor caminho entre ultimo e v_s
29:            Para v_p ∈ P_ultimo,s Faça
30:                P ← P − {v_p}
31:                T ← T ∪ {v_p}
32:            Fim
33:        Senão
34:            P_s,primeiro ← menor caminho entre primeiro e v_s
35:            Para v_p ∈ P_s,primeiro Faça
36:                P ← P − {v_p}
37:                T ← {v_p} ∪ T
38:            Fim
39:        Fim
40:    Fim
41:    primeiro ← T[0]
42:    ultimo ← T[|T|]
43:    P_ultimo,primeiro ← menor caminho entre ultimo e primeiro
44:    T ← T ∪ P_ultimo,primeiro
45: Fim
```

Fonte: O Autor

Pseudocódigo 13 *Nearest Insertion Path Building*

1: **Procedimento** NEAREST-INSERTION-PATH-BUILDING($G(V,E)$, *particao*)
▷ G é o o grafo, *particao* é a lista de vértices que formam a partição de um agente
▷ Exclusivamente neste pseucódigo, será assumido que listas são indexadas a partir de zero
2: $T \leftarrow \{\}$ ▷ P é o caminho resultante
3: $P \leftarrow$ Cópia(*particao*)
4: $v \leftarrow$ vértice aleatório de P
5: $P \leftarrow P - \{v\}$
6: $v_s \leftarrow v' \in P$ tal que $d(v',v) = \min_{v_i \in P} d(v_i, v)$ ▷ A função d retorna a distância entre v_i e v
7: $P \leftarrow P - \{v_s\}$
8: $P_{v,v_s} \leftarrow$ menor caminho de v para v_s
9: $T \leftarrow P_{v,v_s} \cup P_{v,v_s}^{-1}$ ▷ T é um ciclo que parte de v, visita v_s e volta a v
10: **Enquanto** $P \neq \{\}$ **Faça**
11: $v_k \leftarrow v' \in P$ tal que $d(v',T) = \min_{v_i \in P} d(v_i, T)$ ▷ d calcula a distância entre um vértice e um caminho
12: $\Delta f_{min} \leftarrow \infty$
13: $e \leftarrow 0$
14: **Para** $i \in 0 \ldots (|T|-1)$ **Faça** ▷ -1 porque o primeiro e o último são iguais
15: $v_i \leftarrow T[i]$
16: $v_{i+1} \leftarrow T[i \bmod (|T|-1)]$
17: $\Delta f_i \leftarrow d(v_i, v_k) + d(v_k, v_{i+1}) - d(v_i, v_{i+1})$
18: **Se** $\Delta f_i < \Delta f_{min}$ **Então**
19: $e \leftarrow i$
20: $\Delta f_{min} \leftarrow \Delta f_i$
21: **Fim**
22: **Fim**
23: $C_a \leftarrow$ menor caminho entre $T[e]$ e v_k
24: $C_b \leftarrow$ menor caminho entre v_k e $T[e+1]$
25: Inserir $C_a \cup C_b$ em T entre os índices e e $e+1$
26: $P \leftarrow P - C_a$
27: $P \leftarrow P - C_b$
28: **Fim**
29: **Fim**

Fonte: O Autor

dimento baseado no $k-change$ (MARX, 2008), que será chamado de *melhorar*. O $k-change$, que também pode ser chamado de $k-opt$ (NILSSON, 2003), é uma heurística utilizada para encontrar uma solução aproximada para o TSP. O $k-change$ é descrito como uma heurística de busca local que tenta encontrar um caminho de menor custo através de leves alterações em um caminho original. A alteração é a substituição de até k arestas no caminho. Os valores para k mais utilizados são 2, 3 e 4. Como o objetivo do operador de mutação é realizar uma leve mudança no indivíduo (LUKE, 2013), esta pesquisa utilizou $k = 2$.

O pseudocódigo 14 ilustra o procedimento de *melhorar*, que engloba o uso do *2-change*.

Pseudocódigo 14 Melhorar (*2-change*)

```
1:  Procedimento MELHORAR(G(V,E),P)
    ▷ G é o grafo e P é a partição do Agente escolhido para sofrer mutação
2:      Repita
3:          p ← RANDOM-2-CHANGE(G(V,E), P)
4:          Se CUSTO(p) < CUSTO(P) Então
5:              P ← p
6:          Fim
7:      Até que não tenhamos mais tempo
8:  Fim
9:  Procedimento RANDOM-2-CHANGE(G(V,E),P)
10:     edge_1 ← 0
11:     edge_2 ← 0
12:     edge_1, edge_2 ← duas arestas aleatórias diferentes e não consecutivas de P
13:     Retorne 2-CHANGE(G(V,E), P, edge_1, edge_2)
14: Fim
```

Fonte: O Autor

Primeiramente, é importante perceber que o *2-change* não necessariamente irá retornar um caminho de custo menor. Então, o procedimento *melhorar* só vai usar o caminho retornado pelo *2-change* se houver ganho real. O procedimento *RANDOM-2-CHANGE* vai selecionar duas arestas aleatoriamente. Contudo, o caminho do agente, P, está representado em uma lista de vértices onde, se V_i e V_{i+1} são dois vértices nas posições i e $i+1$ dentro do caminho do agente, então existe uma aresta (V_i, V_{i+1}) no Grafo onde o agente patrulha. Então, quando o procedimento escolhe um valor $edge_1$, esse valor está representando, na verdade, a aresta $(V_{edge_1}, V_{edge_1+1})$.

Uma vez que o procedimento tem certeza de ter escolhido duas arestas diferentes, ele faz uma chamada para o *2-change* em si, que está descrito no pseudocódigo 15.

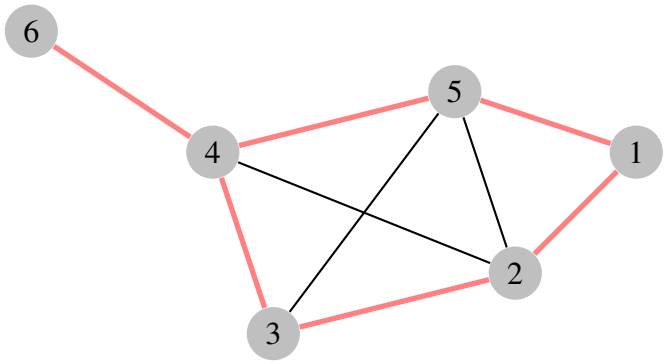
O procedimento *2-change* funciona da seguinte forma: seja P o conjunto de pares

Pseudocódigo 15 *2-change*

```
1: Procedimento 2-CHANGE(G(V,E), P, edge_1, edge_2)
   ▷ G é o grafo e P é a partição do Agente escolhido para sofrer mutação
2:     P' ← { }                                                    ▷ Novo caminho
3:     Para k ∈ 1 ... |P| Faça                    ▷ Lembrando que edge_1 < edge_2
4:         Se k < edge_1 Então
5:             ADICIONA-AO-CAMINHO(P', P[k])                  ▷ Vide pseudocódigo 11
6:         Senão Se k = edge_1 Então
7:             p ← menor caminho entre P[edge_1] e P[edge_2]
8:             Para cada p_i ∈ p Faça
9:                 ADICIONA-AO-CAMINHO(P', p_i)
10:            Fim
11:        Senão Se k < edge_2 Então
12:            v ← P[edge_1 + edge_2 − k]      ▷ lembrando que, nesse ponto, k > edge_1
13:            ADICIONA-AO-CAMINHO(P', v)
14:        Senão Se k = edge_2 Então
15:            p ← menor caminho entre P[edge_1 + 1] e P[edge_2 + 1]
16:            Para cada p_i ∈ p Faça
17:                ADICIONA-AO-CAMINHO(P', p_i)
18:            Fim
19:        Senão
20:            ADICIONA-AO-CAMINHO(P', P[k])
21:        Fim
22:    Fim
23:    Retorne P'
24: Fim
```

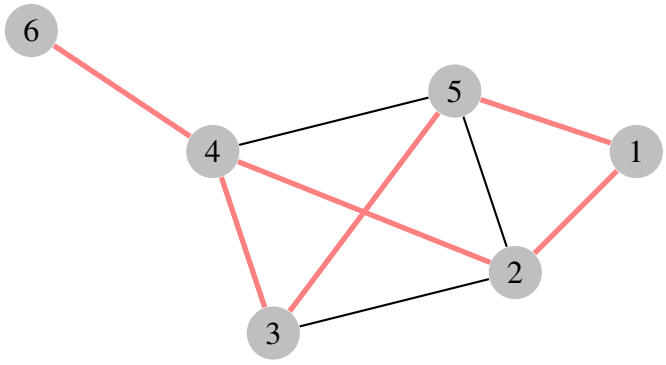
Fonte: O Autor

Figura 4.2: Caminho original do Agente de exemplo para o procedimento *2-change*



Fonte: O autor

Figura 4.3: Caminho do Agente de exemplo após o procedimento *2-change* aplicado a $(4,5)$ e $(2,3)$



Fonte: O autor

ordenados (já que é uma lista) dos vértices no caminho do agente A. P será algo como:

$$\{< V_1, 1 >, ..., < V_{e_1}, e_1 >, < V_{e_1+1}, e_1+1 >, ..., \\ < V_{e_2-1}, e_2-1 >, < V_{e_2}, e_2 >, < V_{e_2+1}, e_2+1 >, ..., < V_{|P|}, |P| >\}$$

Onde e_1 e e_2 são as duas arestas escolhidas para serem substituídas. Após a execução do *2-change*, o caminho retornado, P', deverá ser:

$$\{< V_1, 1 >, ..., < V_{e_1}, e_1 >, ..., < V_{e_2}, e_2 >, < V_{e_2-1}, e_2-1 >, ..., \\ < V_{e_1+1}, e_1+1 >, ..., < V_{e_2+1}, e_2+1 >, ..., < V_{|P|}, |P| >\}$$

Veja um exemplo. Suponha que um agente, A, que percorre as arestas vermelhas no seu subgrafo induzido ilustrado na Figura 4.2. O seu ciclo seria representado pela lista $P = [6,4,5,1,2,3,4,6]$. Suponha agora que esse agente passe pelo *2-change* e que as arestas escolhidas foram $(4,5)$ e $(2,3)$, ou seja, $e_1 = 2$ e $e_2 = 5$. Então, P' será $[6,4,2,1,5,3,4,6]$. E é possível visualizar P' na Figura 4.3.

Com o operador *Melhorar* devidamente explicado, fica faltando os operadores de mutação em si, que vão fazer uma chamada para o procedimento de *Melhorar*. O primeiro operador de mutação é chamado de *Half Add Half Sub Small Changes*. O nome vem do fato de que existe uma change de 50% do operador adicionar ou subtrair um vértice do caminho do agente selecionado para mutação, sempre de forma a modificar levemente o caminho. O pseudocódigo 16 ilustra o operador.

Quando o operador executa a adição de um vértice, v, ele procura pelo vértice, v' que já está no caminho do agente com menor distância para v. Depois, ele inclui no caminho do agente uma rota que vai do vértice v para o vértice v' e volta para v. Por isso, há duas operações de adição nas linhas 22 e 23: o operador está adicionando o caminho de ida e de volta ao mesmo tempo. Já para remover um vértice, v, do caminho do agente, o operador, primeiro, verifica se o vértice a ser removido é o primeiro (e último, já que é um ciclo). Em caso positivo, o primeiro e último elementos da lista que representa o caminho do agente são removidos e então o menor caminho entre os novos último e primeiro vértices é adicionado ao final do ciclo do agente (linhas 32 a 35). Caso o caminho do agente parta de um vértice v_i, visite o vértice escolhido e volte para o mesmo vértice v_i, então o operador simplesmente apaga v e uma das cópias de v_i (linhas 36 a 38). Por fim, caso nenhuma dessas condições seja verdadeira, o operador irá remover o vértice escolhido e substituí-lo pelo menor caminho entre seu antecessor e seu sucessor (linhas 39 a 41).

Para concluir os operadores de mutação, falta o operador *Half Add Half Sub Rebuild*. Ele compartilha com o *Half Add Half Sub Small Changes* a chance de 50% de adicionar ou remover um vértice do caminho do agente. No entanto, este operador tem como objetivo causar alterações mais drásticas no caminho do agente, pois ele o reconstrói integralmente. O pseudocódigo 17 ilustra este operador.

Caso o *Half Add Half Sub Rebuild* opere para adicionar um novo vértice, v, à lista de vértices patrulhados pelo agente, A, ele irá simplesmente adicionar v à partição associada a A. Depois, o caminho que A tem de percorrer é completamente reconstruído usando qualquer um dos operadores apresentados na Seção 4.1.3. Por isso, o *Half Add Half Sub Rebuild* deve gerar alterações mais bruscas nos indivíduos quando comparado com o *Half Add Half Sub Small Changes*.

Caso a operação seja de remoção, o operador irá procurar um vértice da partição do agente que possa ser removido sem desconectar o seu subgrafo induzido. Quando este vértice é encontrado ele é removido e um novo caminho é calculado para o agente. De forma análoga à operação de adição, o novo caminho pode ser construído utilizando qualquer um dos operadores de *path building* apresentados neste capítulo.

4.3 Recombinação

Por fim, será apresentado um operador para aplicar recombinação em dois indivíduos pais gerando dois indivíduos filhos. Esse operador é chamado de *Simple Random Crossover*,

Pseudocódigo 16 *Half Add Half Sub Small Changes*

```
1: Procedimento HALF-ADD-HALF-SUB-SMALL-CHANGES(Solucao)
   ▷ Solucao é o indivíduo selecionado para mutação
2:     G(V,E) ← o grafo associado à Solucao
3:     A ← um agente escolhido aleatoriamente
4:     d ← um número Real aleatório entre 0 e 1
5:     Se d ≥ 0.5 Então
6:         ADICIONA-COM-POUCAS-MUDANÇAS(G(V,E),A)
7:     Senão
8:         REMOVE-COM-POUCAS-MUDANÇAS(G(V,E),A)
9:     Fim
10:    Configurar caminho do agente A para MELHORAR(G(V,E), A)
11: Fim
12: Procedimento ADICIONA-COM-POUCAS-MUDANÇAS(G(V,E),A)
13:    P ← o caminho do agente A
14:    v ← um vértice aleatório tal que v ∈ V e v ∉ P
15:    d ← ∞
16:    v' ← nulo
17:    Para i ∈ 1 ... |P| Faça d_i ← DISTANCIA(G(V,E), v, P[i])
18:        Se v' = nulo ou d' < d Então v' ← P[i]
19:            d ← d_i
20:        Fim
21:    Fim
22:    P' ← menor caminho entre v' e v
23:    Para k ∈ 1 ... (|P'| − 1) Faça ADICIONA-AO-CAMINHO(P, i+k, P'[k])
24:        ADICIONA-AO-CAMINHO(P, i+k, P'[k])
25:    Fim
26:    ADICIONA-AO-CAMINHO(P, i+k+1, P'[k+1])
27: Fim
28: Procedimento REMOVE-COM-POUCAS-MUDANÇAS(G(V,E),A)
29:    P ← o caminho do agente A
30:    indice ← um número aleatório entre 1 e |P|
31:    P' ← { }
32:    Se indice = 1 ou indice = |P| Então REMOVE-PRIMEIRO-ELEMENTO(P)
33:        REMOVE-ULTIMO-ELEMENTO(P)
34:        P' ← menor caminho entre P[|P|] e P[1]
35:        indice ← |P|
36:    Senão Se P[indice − 1] = P[indice + 1] Então
37:        REMOVE-ELEMENTO-NO-INDICE(P, indice)
38:        REMOVE-ELEMENTO-NO-INDICE(P, indice)
39:    SenãoP' ← menor caminho entre P[indice − 1] e P[indice + 1]
40:        REMOVE-ELEMENTO-NO-INDICE(P, indice)
41:    Fim
42:    Para cada v_j ∈ P' Faça ADICIONA-AO-CAMINHO(P, indice, v_j)
43:        indice ← indice + 1
44:    Fim
45: Fim
```

Fonte: O Autor

Pseudocódigo 17 *Half Add Half Sub Rebuild*

```
1: Procedimento HALF-ADD-HALF-SUB-REBUILD(Solucao)
   ▷ Solucao é o indivíduo selecionado para mutação
2:     G(V,E) ← o grafo associado à Solucao
3:     A ← um agente escolhido aleatoriamente
4:     d ← um número Real aleatório entre 0 e 1
5:     Se d ≥ 0.5 Então
6:         ADICIONA-E-RECONSTROI(G(V,E),A)
7:     Senão
8:         REMOVE-E-RECONSTROI(G(V,E),A)
9:     Fim
10:    Configurar caminho do agente A para MELHORAR(G(V,E), A)
11: Fim
12: Procedimento ADICIONA-E-RECONSTROI(G(V,E),A)
13:    P ← o caminho do agente A
14:    v ← um vértice aleatório tal que v ∈ V e v ∉ P
15:    particao ← a partição por onde patrulha o agente A
16:    particao ← particao ∪ {v}
17:    P ← RECONSTRÓI-O-CAMINHO(G(V,E), particao)
18: Fim
19: Procedimento REMOVE-E-RECONSTROI(G(V,E),A)
20:    P ← o caminho do agente A
21:    particao ← a partição por onde patrulha o agente A
22:    lista ← EMBARALHA(particao)          ▷ Embaralha de forma aleatória
23:    Para v ∈ lista Faça
24:        particao' ← particao − {v}
25:        G' ← subgrafo de G induzido por particao'
26:    Fim
27:    particao ← particao'
28:    RECONSTRÓI-O-CAMINHO(G(V,E), particao)
29: Fim
```

Fonte: O Autor

ou Recombinação Aleatória Simples, e está ilustrado no pseudocódigo 18. O operador consiste em tentar trocar um agente de um indivíduo por um agente do outro indivíduo selecionado para recombinação.

Pseudocódigo 18 *Simple Random Crossover*

```
1:  Procedimento SIMPLE-RANDOM-CROSSOVER(Solucao_a, Solucao_b, r)
    ▷ Os parâmetros são dois indivíduos e r, o número de agentes
2:      r_i ← escolhe um número aleatório entre 1 ... r
3:      C_a ← Cópia(Solucao_a)                    ▷ Este é o filho do indivíduo a
4:      C_b ← Cópia(Solucao_b)                    ▷ Este é o filho do indivíduo b
5:      Agente_a ← RETORNA-AGENTE(C_a, r_i)
6:      Agente_b ← RETORNA-AGENTE(C_b, r_i)
7:      Se o centro do Agente_a for igual ao centro do Agente_b Então
8:          CONFIGURA-AGENTE(C_a, r_i, Agente_b)
9:          CONFIGURA-AGENTE(C_b, r_i, Agente_a)
10:         Retorne C_a, C_b
11:     Senão
12:         Para j ∈ 1 ... r Faça
13:             agente ← RETORNA-AGENTE(C_b, j)
14:             Se o centro do Agente_a está no caminho do agente Então
15:                 Agente_b ← agente
16:                 CONFIGURA-AGENTE(C_a, r_i, Agente_b)
17:                 CONFIGURA-AGENTE(C_b, j, Agente_a)
18:                 Retorne C_a, C_b
19:             Fim
20:         Fim
21:         CONFIGURA-AGENTE(C_a, r_i, Agente_b)
22:         CONFIGURA-AGENTE(C_b, r_i, Agente_a)
23:         Retorne C_a, C_b
24:     Fim
25: Fim
```

Fonte: O Autor

O operador começa escolhendo um índice, r_i, de um agente aleatoriamente. Depois, uma cópia dos indivíduos é atribuída a dois novos indivíduos filhos. A seguir, o operador verifica se os agentes no índice r_i dos indivíduos possuem o mesmo vértice como centro. Em caso positivo, o operador imediatamente troca um agente pelo outro entre os dois indivíduos filhos. Em caso negativo, o operador tenta encontrar o agente do indivíduo b cujo caminho passe pelo centro do agente posicionado no índice r_i do indivíduo a. Encontrando este agente, o operador faz a troca. Caso não seja encontrado nenhum agente, o operador vai simplesmente trocar os agentes que estão no mesmo índice, r_i, em seus respectivos indivíduos.

Quadro 4.1: Resumo dos operadores apresentados

Operador	**Operação**	**Referência**
Random Centering	Criação de Indivíduos	Proposto neste trabalho
Approximated Maximum Distance Centering	Criação de Indivíduos	Proposto neste trabalho
Random Partitioning	Criação de Indivíduos	Proposto neste trabalho
Heuristic Graph Partitioning	Criação de Indivíduos	Proposto neste trabalho
Random Path Building	Criação de Indivíduos	Proposto neste trabalho
Nearest Neighbor Path Building	Criação de Indivíduos	(NILSSON, 2003)
Nearest Insertion Path Building	Criação de Indivíduos	(NILSSON, 2003)
Melhorar	Mutação	Proposto neste trabalho
2-change	Mutação	(MARX, 2008)
Half Add Half Sub Small Changes	Mutação	Proposto neste trabalho
Half Add Half Sub Rebuild	Mutação	Proposto neste trabalho
Simple Random Crossover	Recombinação	Proposto neste trabalho

Fonte: O autor

4.4 Considerações Finais

Neste capítulo, foram apresentadas e detalhadas formas de modelar a TMAP que possibilitem a aplicação de Algoritmos Evolucionários tradicionais para encontrar soluções que minimizem uma dada métrica.

Os operadores apresentados ao longo deste capítulo foram desenvolvidos pelo autor em conjunto com Diogo Felipe Félix de Melo[1] para a disciplina de Tópicos Avançados em Inteligência Artificial no curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco ministrada pelo professor Pablo Azevedo Sampaio[2].

O Quadro 4.1 resume os operadores propostos, indicando a operação realizada e se foram baseados em trabalhos já presentes na literatura.

Os operadores descritos neste trabalho foram implementados na linguagem de programação Java[3]. A principal razão para a adoção desta linguagem foi o *Simple Patrol*, simulador da TMAP, desenvolvido pelo grupo de pesquisa sobre Patrulha Multiagente da Universidade Federal Rural de Pernambuco, utilizado para realizar os experimentos desta pesquisa. Os operadores foram desenvolvidos de forma compatível com a biblioteca jMetal[4] (DURILLO; NEBRO, 2011), que disponibiliza diversos algoritmos evolucionários facilmente adaptáveis para qualquer problema que possa ser modelado em classes Java.

[1] http://lattes.cnpq.br/2213650736070295
[2] http://lattes.cnpq.br/8865836949700771
[3] http://java.com/pt_BR/
[4] http://jmetal.sourceforge.net/

Os operadores propostos nesta pesquisa podem ser combinados de diferentes formas, construindo diversas heurísticas que podem encontrar soluções distintas para TMAP. No próximo capítulo, serão descritos experimentos feitos para encontrar as melhores combinações além de compará-las com abordagens propostas na literatura.

5

Experimentos

Como mostrado no Capítulo 4, os operadores propostos podem ser combinados de diferentes formas, para construir heurísticas distintas para a TMAP. Assim, este capítulo tem dois objetivos. O primeiro é encontrar as melhores combinações de operadores. O segundo é comparar as heurísticas que obtiveram melhores resultados com abordagens propostas por outros autores.

Uma vez que os operadores estavam implementados e importados no simulador *Simple Patrol*, foram realizados diversos experimentos divididos em dois grupos. O primeiro com o objetivo de encontrar as melhores abordagens evolucionárias para TMAP e o segundo com a finalidade de comparar estas abordagens com as publicadas por outros autores.

A Métrica utilizada para avaliar os indivíduos se manteve constante em todos os experimentos. Foi selecionada a média quadrática dos intervalos, pois segundo (SAMPAIO, 2013) ela reflete o equilíbrio entre as métricas de intervalo médio, intervalo máximo e desvio padrão dos intervalos e, por isso, é uma boa métrica para a TMAP.

Abaixo, estão listados alguns parâmetros que, embora não influenciem diretamente nos algoritmos evolucionários, tem grande impacto nas respostas obtidas.

- Todos os algoritmos evolucionários descritos no Capítulo 3, executam seu *loop* principal até "não haja mais tempo". Esse tempo é representado pelo parâmetro **número máximo de avaliações**. Toda vez que o algoritmo evolucionário avalia a aptidão de um indivíduo, ele incrementa o contador do número de avaliações. Quando esse contador chegar no número máximo de avaliações, o algoritmo evolucionário para.

- Cada vez que uma avaliação de aptidão de um indivíduo é requisitada, uma instância do *Simple Patrol* é iniciada. Essa instância simula a solução correspondente ao indivíduo e retorna o valor da métrica, que é utilizado como aptidão. A solução é simulada por um número pré-definido de unidades de tempo, que corresponde ao parâmetro **número de turnos**.

- A **quantidade de agentes** é outro número que influi no desempenho das soluções

para a TMAP. A medida que este número cresce, fica cada vez mais simples patrulhar uma mesma quantidade de pontos de interesse com alta frequência. (SAMPAIO, 2013) descobriu experimentalmente que não vale a pena manter a relação entre a quantidade de agentes e a quantidade de nós acima de um terço, pois, a partir desse número, estratégias bem distintas passam a ter desempenhos muito próximos.

Outro ponto importante são os mapas que foram utilizados nos experimentos. Esta pesquisa se valeu de três grafos diferentes para testar e comparar os algoritmos evolucionários. Eles estão ilustrados na Figura 5.1.

Nas seções seguintes, serão discutidos os dois grupos de experimentos realizados. Eles foram chamados de Experimentos de *Tuning* e de comparação.

5.1 Experimentos de *Tuning*

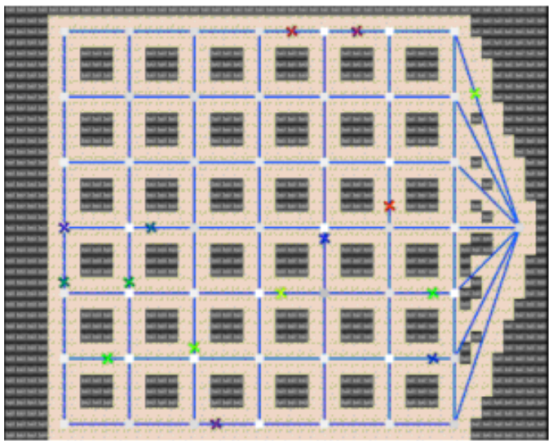
Tuning pode ser traduzido, do inglês, para ajuste. Esta seção tem como finalidade descrever e apresentar os resultados dos experimentos desenvolvidos para buscar o ajuste dos operadores que compõem as melhores heurísticas evolucionárias estudadas nesta pesquisa. Por exemplo, qual seria a melhor forma de inicializar um indivíduo? Existem ao todo doze formas diferentes de compor o operador de inicialização de indivíduos de uma heurística evolucionária utilizando os operadores propostos no Capítulo 4. Antes de realizar os experimentos de *tuning* em si, foram feitos alguns testes para os valores de **número máximo de avaliações**, **tamanho da população** e **número de tentativas do operador Melhorar**.

5.1.1 Número máximo de avaliações

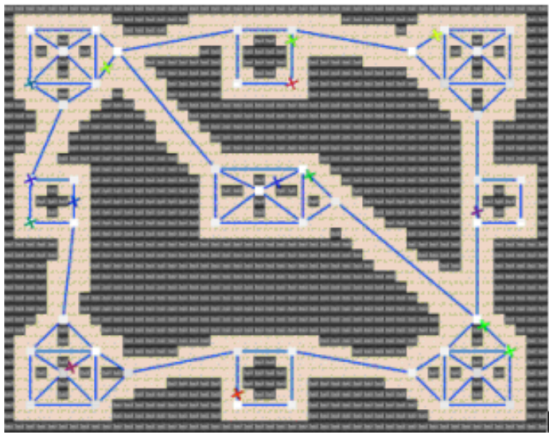
Primeiramente, foram realizados experimentos para determinar um número máximo de avaliações que permitisse distinguir os desempenhos das heurísticas nos demais experimentos. Enquanto o número máximo de avaliações variou entre 10 mil e 150 mil, os demais parâmetros permaneceram fixados:

- Algoritmo utilizado: Estratégia Evolucionária $(\mu + \lambda)$
- Número de turnos: *2mil*
- Número de agentes: *5*
- μ: *30*
- λ: *180*
- Mapa: *islands*

Figura 5.1: Mapas utilizados nos experimentos



Mapa *grid*



Mapa *islands*

Mapa *cicles corridor*

Fonte: (SAMPAIO, 2013)

- Criação de indivíduos: *Random Centering*, *Random Partitioning*, *Random Path Building*
- Mutação: *Half Add Half Sub Small Changes*

Após realizar os experimentos no mapa *islands*, com diferentes execuções da estratégia $(\mu + \lambda)$, variando o número máximo de avaliações entre *10 mil* e *150 mil*, os resultados indicaram que *30 mil* avaliações eram suficientes para as estratégias evoluírem de soluções notadamente ruins para soluções a ponto de convergir.

5.1.2 Tamanho da População

Depois, foram testados os tamanhos de população. Nos algoritmos genéticos, esse parâmetro diz respeito à quantidade de indivíduos a cada geração. Já nas estratégias evolucionárias existem dois parâmetros relacionados ao tamanho da população: μ e λ, como foi discorrido no Capítulo 3. Um experimento similar ao anterior foi realizado: as diferenças eram o número de avaliações agora fixado em *30 mil*, mas com μ e λ variando para as Estratégias Evolucionárias e o tamanho da população variando para o algoritmos genéticos. Todos os quatro algoritmos apresentados nesse trabalho foram testados individualmente. λ e o tamanho da população variaram entre *72* e *180*. μ variou entre $1/4$, $1/5$ e $1/6$ do valor de λ.

Os resultados obtidos foram:

- População de *96* indivíduos para o Algoritmo Genético
- População de *110* indivíduos para o Algoritmo Genético de Estado Estável
- $\mu = 18$ e $\lambda = 90$ para a Estratégia Evolucionária $(\mu + \lambda)$
- $\mu = 26$ e $\lambda = 156$ para a Estratégia Evolucionária (μ, λ)

5.1.3 Número de tentativas do operador Melhorar

Assim como os algoritmos evolucionários, o operador *Melhorar*, utilizado pelos operadores de mutação, também itera um número arbitrário de vezes. Também foram feitos experimentos para identificar qual o número de iterações que seria suficiente para causar melhoras significativas em soluções. O algoritmo evolucionário utilizado para estes testes foi a Estratégia Evolucionária $(\mu + \lambda)$, com $\mu = 18$ e $\lambda = 90$. Os demais parâmetros foram fixados com valores iguais ao experimento do máximo número de avaliações, enquanto que o número de iterações no *loop* do operador *melhorar* variou entre *0* e *100*, pois já que o operador leva bastante tempo para calcular uma resposta, valores acima de *100* se mostraram impraticáveis. Os resultados mostraram que a partir de *5* iterações, não havia variação significativa na qualidade dos indivíduos encontrados.

O Quadro 5.1 resume estes experimentos realizados antes do *tuning*.

Quadro 5.1: Resumo dos experimentos realizados antes do *tuning*

Parâmetro	***Variação***	**Resultado**
Número máximo de avaliações	Entre *10 mil* e *150 mil*	*30 mil*
Tamanho da População	λ e o tamanho da população variaram entre *72* e *180*. μ variou entre $1/4$, $1/5$ e $1/6$ do valor de λ.	$(\mu+\lambda)$: $\mu = 18$ e $\lambda = 90$. (μ,λ): $\mu = 26$ e $\lambda = 156$. GA: *96* indivíduos. GA de estado estável: *110* indivíduos
Número de tentativas do operador Melhorar	Entre *0* e *100*	*5*

Fonte: O autor

Finalmente, o experimento de *tuning* foi realizado para avaliar as melhores combinações dos operadores descritos no Capítulo 4 com os algoritmos apresentados no Capítulo 3. Ele foi configurado com os parâmetros de número máximo de avaliações, número máximo de iterações no operador *Melhorar* e o tamanho da população fixados (este último por algoritmo) nos valores acima reportados. O número de unidades de tempo em cada simulação foi mantido em *2 mil*. Os experimentos foram feitos em todos os três grafos propostos na Figura 5.1 e o tamanho da sociedade de agentes entre *1*, *5*, *10* e *15*. Para cada um desses foram simuladas cada umas das composições de operadores e algoritmos. Ao todo, foram *1728* simulações onde *144* estratégias evolucionárias diferentes foram testadas. No final, seus respectivos valores da Média Quadrática dos Intervalos (MQI) para o melhor indivíduo encontrado foram registrados.

Para comparar as estratégias, no entanto, foi utilizada a seguinte metodologia proposta por (SAMPAIO, 2013). Os valores das médias quadráticas dos intervalos foram normalizadas, para que se tenha um único valor para todas as sociedades de agentes por heurística, por mapa. Esta medida é obtida através da multiplicação da média quadrática dos intervalos pelo tamanho da sociedade de agentes que o obteve (no caso desses experimentos, 1, 5, 10 e 15). Depois, esse valor é somado. A equação abaixo mostra como obter a Média Quadrática dos Intervalos Normalizada (MQIN):

$$MQIN = \sum MQI_S * |S|$$

Onde S é a sociedade de agentes e MQI_S é a Média Quadrática dos Intervalos obtida pela sociedade S.

Depois, foi elaborado um *ranking*, para cada mapa, ordenado pela $MQIN$. Por fim, foi retirada uma média dos *rankings* para cada heurística e as 5 melhores estão listadas no Quadro 5.2.

Quatro das heurísticas presentes no *top 5* são Algoritmos Genéticos, que utilizam o operador de recombinação (Estratégias Evolucionárias só utilizam Mutação). Este é um forte indício da eficácia do *Simple Random Crossover* proposto nesta pesquisa.

Quadro 5.2: As 5 melhores heurísticas evolucionárias

Algoritmo Evolucionário	**Mutação**	***Centering***	***Partitioning***	***Path Building***	**Ranking Médio**
Algoritmo Genético	*Half Add Half Sub Small Changes*	*Approximated Maximum Distance Centering*	*Heuristic Graph Partitioning*	*Nearest Neighbor Path Building*	4,000
Algoritmo Genético de Estado Estável	*Half Add Half Sub Small Changes*	*Approximated Maximum Distance Centering*	*Heuristic Graph Partitioning*	*Nearest Neighbor Path Building*	4,000
Algoritmo Genético de Estado Estável	*Half Add Half Sub Small Changes*	*Random Centering*	*Heuristic Graph Partitioning*	*Nearest Neighbor Path Building*	5,333
Algoritmo Genético	*Half Add Half Sub Small Changes*	*Random Centering*	*Heuristic Graph Partitioning*	*Nearest Neighbor Path Building*	5,666
Estratégia Evolucionária (μ, λ)	*Half Add Half Sub Small Changes*	*Approximated Maximum Distance Centering*	*Heuristic Graph Partitioning*	*Nearest Neighbor Path Building*	8,333

Fonte: O autor

Outra importante informação do Quadro 5.2 é que, com exceção das 3ª e 4ª colocadas, as melhores heurísticas se valem dos operadores propostos no Capítulo 4 que não são exclusivamente aleatórios, mas se valem de informações sobre o problema para computar suas respostas. Notoriamente, os operadores *Approximated Maximum Distance Centering*, *Heuristic Graph Partitioning* e *Nearest Neighbor Path Building* obtiveram excelentes resultados. Por outro lado, as heurísticas com mais operadores aleatórios, como por exemplo, *Random Partitioning* e *Random Path Building* tiveram um baixo desempenho.

Com esses resultados, as duas primeiras heurísticas do Quadro 5.2 foram selecionadas para comparação com algoritmos propostos por outros autores.

5.2 Experimentos de Comparação

Para verificar a qualidade das heurísticas evolucionárias foram feitos experimentos comparando com as três melhores estratégias gravitacionais propostas por (SAMPAIO, 2013). São elas: $grav(Node, Ar, 2, sum)$, $grav(Node, Ar, 1, max)$ e $grav(Edge, Ar, 1, max)$. Além delas, a estratégia de ciclo único proposta por (CHEVALEYRE; SEMPE; RAMALHO, 2004) também foi utilizada na comparação. A estratégia de ciclo único obteve excelentes resultados nos experimentos realizados por (ALMEIDA et al., 2004), enquanto que as estratégias gravitacionais também tiveram bons resultados nos estudos conduzidos em SAMPAIO (2013).

A partir daqui, quando o texto mencionar "Algoritmo Genético", estará se referindo ao Algoritmo Genético do pseudocódigo 4, utilizando *Approximated Maximum Distance Centering*, *Heuristic Graph Partitioning* e *Nearest Neighbor Path Building* para criar indivíduos e *Half Add Half Sub Small Changes* para mutação. Da mesma forma, quando for mencionado o "Algoritmo Genético de Estado Estável", está se referindo ao Algoritmo Genético de Estado Estável descrito no pseudocódigo 5, utilizando *Approximated Maximum Distance Centering*, *Heuristic Graph Partitioning* e *Nearest Neighbor Path Building* para criar indivíduos e *Half Add Half Sub Small Changes* para mutação.

Os experimentos de comparação colocaram cada estratégia para gerar quatro sociedades de agentes para cada um dos mapas a serem patrulhados, ilustrados na Figura 5.1. O tamanho das sociedades variou entre 1, 5, 10 e 15, tal como nos experimentos de *tuning*.

O Quadro 5.3 mostra os resultados obtidos pela heurística do Algoritmo Genético (1º lugar no Quadro 5.2). Enquanto que o Quadro 5.4 mostra os resultados da heurística do Algoritmo Genético de Estado Estável. Os quadros 5.5, 5.6 e 5.7 mostram os resultados das estratégias gravitacionais $grav(Node, Ar, 2, sum)$, $grav(Node, Ar, 1, max)$ e $grav(Edge, Ar, 1, max)$, respectivamente. Por fim, o Quadro 5.8 exibe os resultados obtidos pela estratégia *Single Cycle*.

Seguindo a mesma abordagem proposta por (SAMPAIO, 2013), que foi utilizada na análise dos experimentos de *tuning*, os resultados das estratégias foram normalizados para todas as sociedades de agentes e a partir delas, um *ranking* foi construído para cada mapa. O Quadro 5.9 mostra os rankings médios obtidos por cada estratégia, enquanto que a Figura 5.2

Quadro 5.3: Resultados do Algoritmo Genético

Algoritmo Genético	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	150,007	32,834	17,445	11,561
Mapa *Grid*	208,876	44,393	23,293	15,617
Mapa *Islands*	188,480	38,340	16,179	10,619

Fonte: O autor

Quadro 5.4: Resultados do Algoritmo Genético de Estado Estável

Algoritmo Genético de Estado Estável	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	150,007	32,802	16,664	11,797
Mapa *Grid*	208,876	44,240	23,416	15,652
Mapa *Islands*	189,175	38,211	17,332	10,400

Fonte: O autor

Quadro 5.5: Resultados da Estratégia Gravitacional $grav(Node, Ar, 2, sum)$

grav(*Node*, *Ar*, 2, *sum*)	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	150,077	33,904	16,419	10,724
Mapa *Grid*	223,844	45,966	22,777	15,023
Mapa *Islands*	196,727	38,356	15,232	9,8

Fonte: O autor

Quadro 5.6: Resultados da Estratégia Gravitacional $grav(Node, Ar, 1, max)$

grav(*Node*, *Ar*, 1, *max*)	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	150,077	35,334	16,844	10,807
Mapa *Grid*	210,857	46,483	22,803	15,076
Mapa *Islands*	192,774	39,645	15,76	10,468

Fonte: O autor

Quadro 5.7: Resultados da Estratégia Gravitacional $grav(Edge, Ar, 1, max)$

grav(*Edge*, *Ar*, 1, *max*)	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	150,077	46,444	21,995	11,553
Mapa *Grid*	208,516	51,816	24,084	16,073
Mapa *Islands*	187,758	56,707	15,701	10,417

Fonte: O autor

Quadro 5.8: Resultados da Estratégia *Single Cycle*

Single Cycle)	**1 Agente**	**5 Agentes**	**10 Agentes**	**15 Agentes**
Mapa *Cicles Corridor*	149,932	31,386	15,739	10,583
Mapa *Grid*	229,122	46,645	23,415	15,667
Mapa *Islands*	193,783	38,652	19,341	12,862

Fonte: O autor

mostra mais detalhadamente o *ranking* de cada estratégia.

Um resultado inesperado foi o baixo desempenho da estratégia *Single Cycle* para um único agente no mapa de *grid*, já que está é baseada em uma solução aproximada do TSP e (CHEVALEYRE; SEMPE; RAMALHO, 2004) mostrou que a solução ótima do TSP também é a solução ótima para a patrulha com apenas um agente e com a métrica de Intervalo Máximo. Nesses experimentos, para a métrica de média quadrática dos intervalos a estratégia de ciclo único obteve o pior resultado no mapa de *grid*. No entanto, para o mapa de *Cicles Corridor*, a *Single Cycle* foi melhor em todos os tamanhos de sociedade.

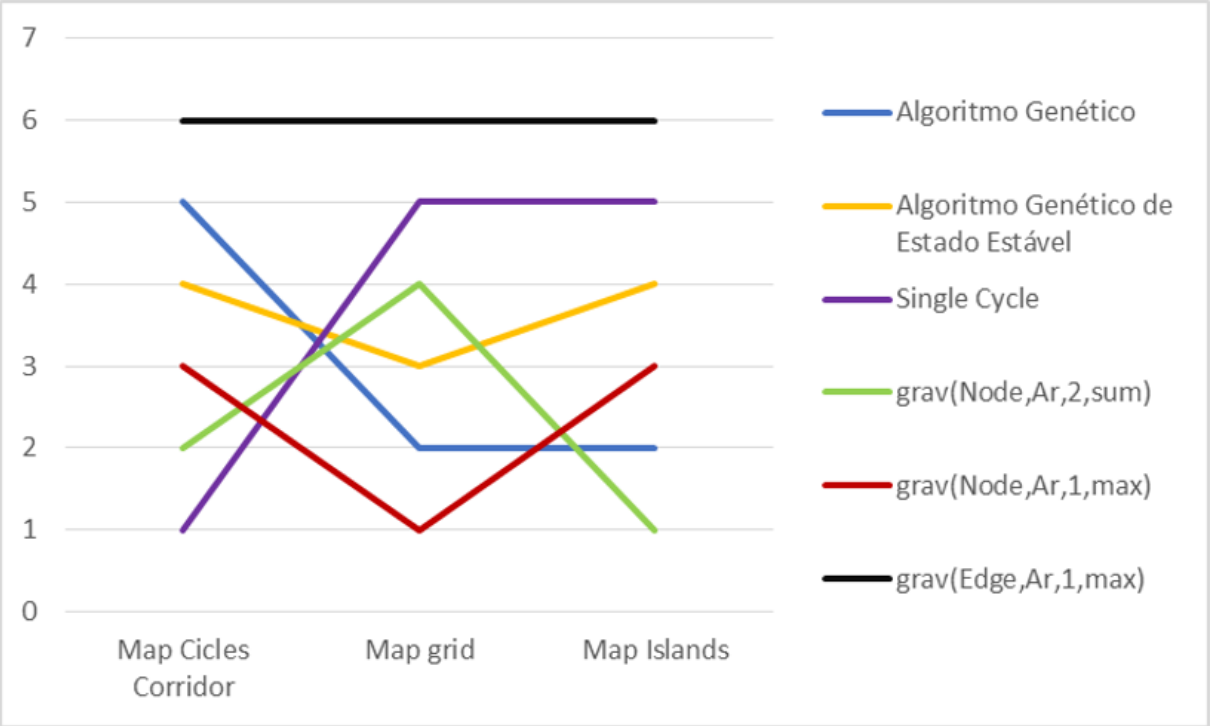
As duas heurísticas evolucionárias tiveram um desempenho muito similar em todos os experimentos, algo que já tinha sido observado quando elas obtiveram o mesmo ranking médio nos experimentos de *tuning*. Elas ficaram muito próximas dos melhores resultados para sociedades de tamanho 1 e 5 em todos os mapas, chegando, inclusive, a superar as estratégias *Single Cycle*, $grav(Node, Ar, 2, sum)$ e $grav(Node, Ar, 1, max)$ nos mapas de *grid* e *islands*. Nos

Quadro 5.9: Resultados da Estratégia *Single Cycle*

Estratégia	***Ranking* Médio**
grav(*Node*, *Ar*, 2, *sum*)	2,333
grav(*Node*, *Ar*, 1, *max*)	2,333
Algoritmo Genético	3,000
Algoritmo Genético de Estado Estável	3,667
Single Cycle	3,667
grav(*Edge*, *Ar*, 1, *max*)	6,000

Fonte: O autor

Figura 5.2: *Rankings* de cada estratégia por mapa



Fonte: O Autor

rankings médios, no entanto, o algoritmo genético levou uma pequena vantagem, principalmente devido ao seu excelente desempenho nos mapas de *grid* e *islands*, quando ficou com o segundo lugar.

Os gráficos das figuras 5.3, 5.4, 5.5 e 5.6 abaixo ajudam a compreender melhor os resultados encontrados.

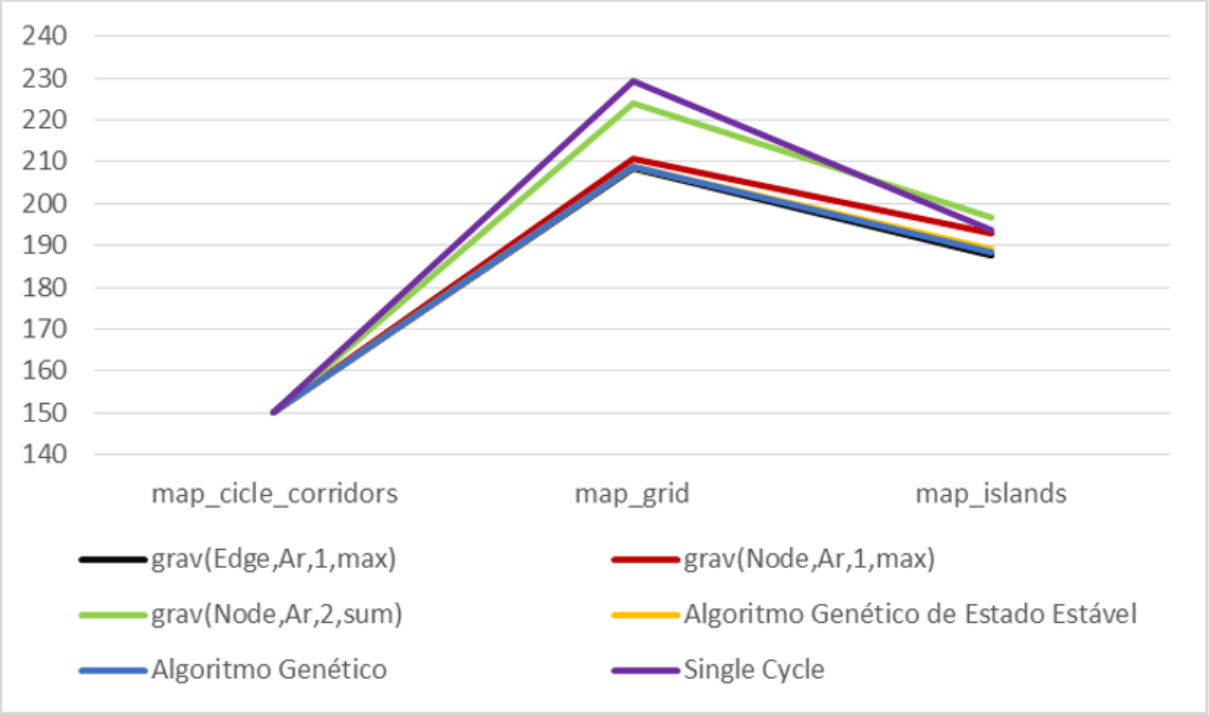
5.3 Considerações Finais

Primeiro, foram experimentadas *144* estratégias evolucionárias distintas, com quatro tamanhos de sociedade e em três mapas, para um total de *1728* simulações. Com o resultado, duas estratégias foram eleitas as melhores. Elas utilizam *Approximated Maximum Distance Centering*, *Heuristic Graph Partitioning* e *Nearest Neighbor Path Building* para criar indivíduos, *Half Add Half Sub Small Changes* para mutação e *Simple Random Crossover* para recombinação. Desses, apenas o *Nearest Neighbor Path Building* não foi contribuição deste trabalho.

Na comparação com outras estratégias propostas na literatura, foram simulados seis algoritmos diferentes, com quatro tamanhos de sociedades e em três mapas, resultando em 72 simulações.

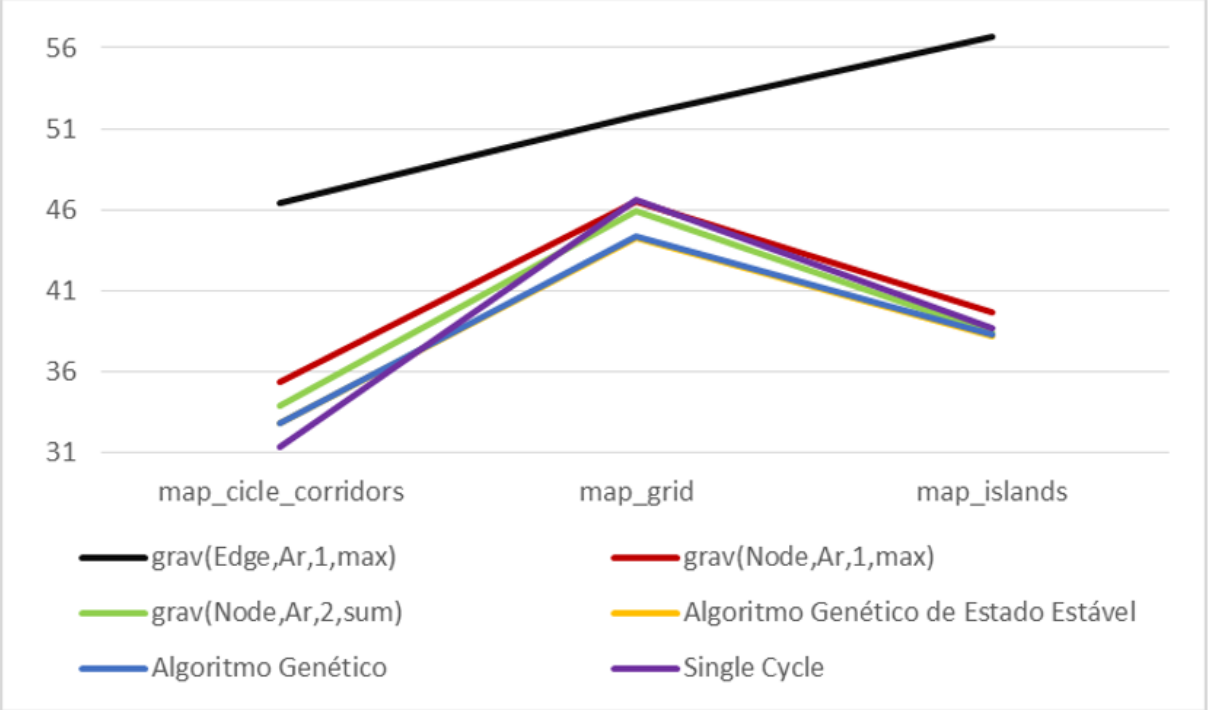
Ao menos para os grafos e sociedades de agentes testados nesses experimentos, as estratégias evolucionárias obtiveram resultados comparáveis com duas das principais estratégias presentes na literatura.

Figura 5.3: Resultado para sociedade de tamanho 1



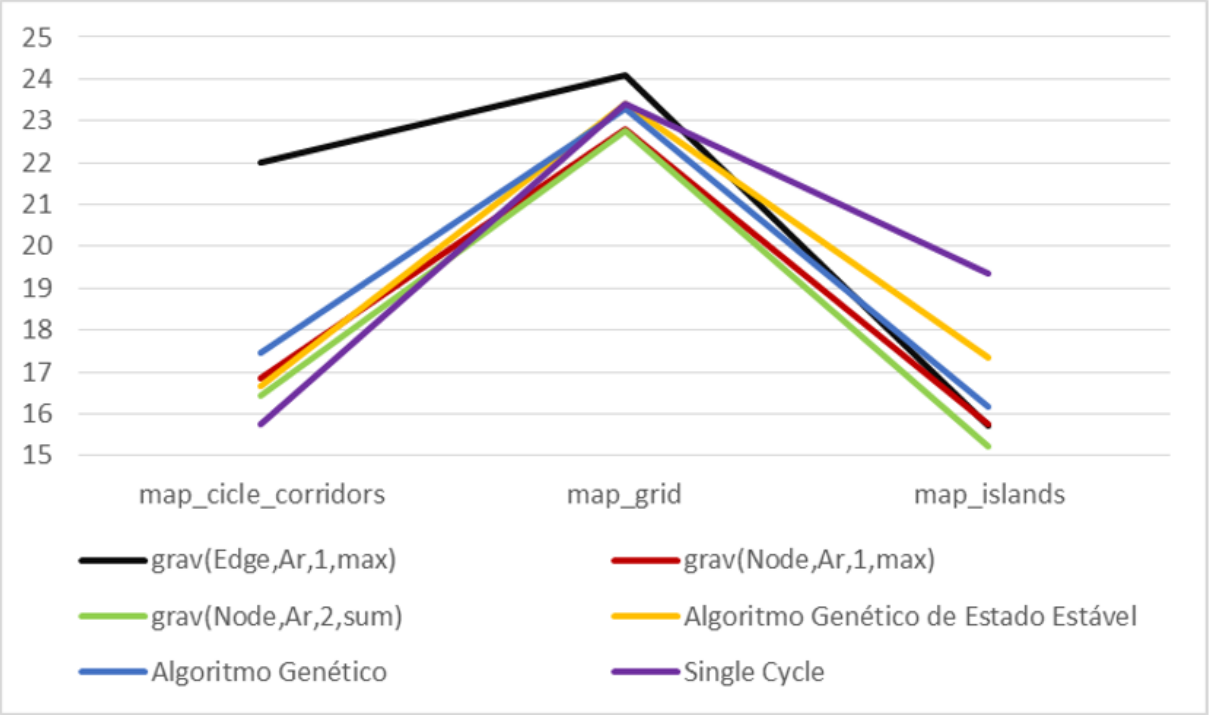
Fonte: O Autor

Figura 5.4: Resultado para sociedade de tamanho 5



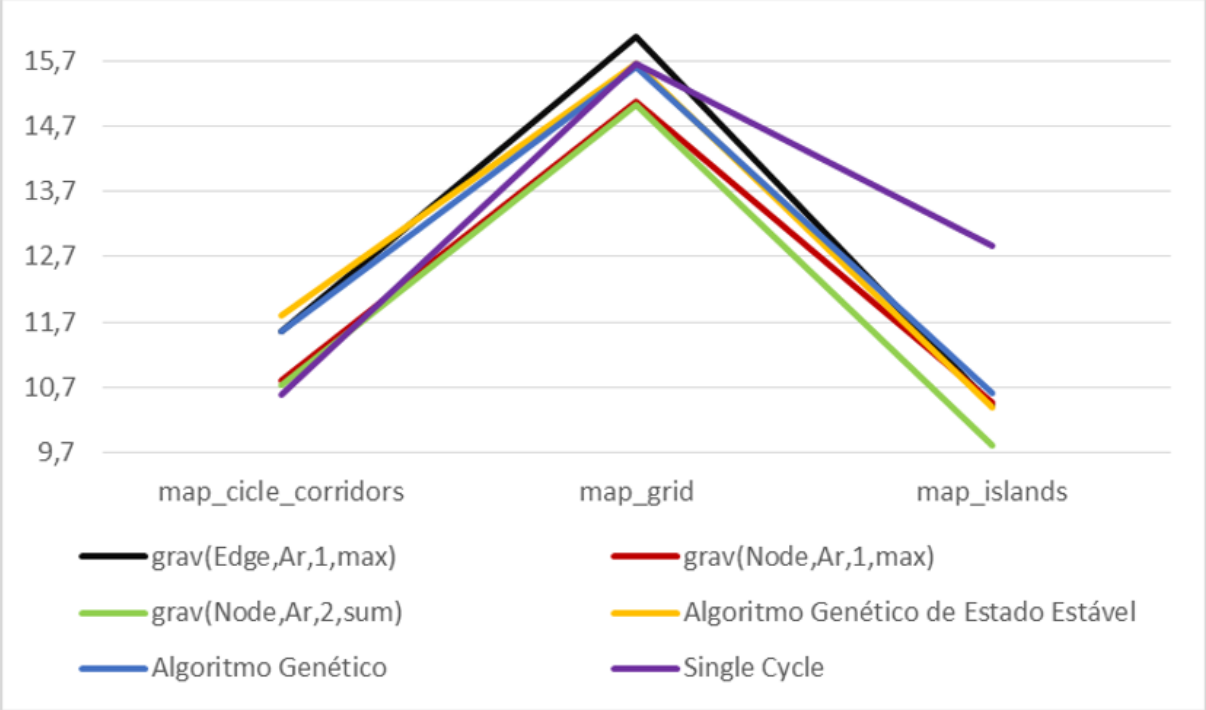
Fonte: O Autor

Figura 5.5: Resultado para sociedade de tamanho 10



Fonte: O Autor

Figura 5.6: Resultado para sociedade de tamanho 15



Fonte: O Autor

6

Conclusão

Como mostrado neste documento, a Patrulha Multiagente Temporal tem sido o objeto de estudo de diversas pesquisas, pois trata-se de um problema difícil e desafiador da área de Inteligência Artificial e Sistemas Multiagente com diversas aplicações em cenários reais.

No entanto, também foi verificado, que apesar de ser essencialmente um problema de otimização, não haviam sido estudadas aplicações diretas de algoritmos evolucionários para encontrar soluções para a TMAP.

Assim, após revisar quatro algoritmos bastante utilizados na Computação Evolucionária, a presente pesquisa apresentou dez operadores que podem ser combinados com esses algoritmos de diferentes maneiras para formar dezenas de heurísticas evolucionárias distintas capazes de resolver a TMAP.

Além disso, foi demonstrado através de experimentos que os operadores aqui apresentados são eficazes em produzir boas soluções. Também foram apontados os operadores e suas combinações que proporcionaram as melhores soluções. E foi mostrado empiricamente que a qualidade das soluções encontradas por estas heurísticas evolucionárias são comparáveis com a de algumas das melhores abordagens apresentadas na literatura.

6.1 Trabalhos Futuros

Este capítulo conclui o trabalho com algumas atividades que poderiam ter sido desenvolvidas, não fosse o período de tempo curto disponibilizado para sua conclusão.

Foi observado durante os experimentos que os operadores de mutação geravam indivíduos novos diferentes o bastante dos pais para permitir encontrar novas soluções e evoluir a população. No entanto, foi possível perceber ao longo dos experimentos realizados que esses operadores geravam indivíduos muito próximos uns dos outros. Fica como trabalho futuro investigar operadores de mutação mais aleatórios, que levem a uma diversidade maior da população, para que a busca no espaço de soluções seja mais global e não fique "presa"ao redor de mínimos locais.

Outra contribuição para o futuro seria investigar o impacto de outros operadores de

seleção nos algoritmos genéticos. Principalmente o algoritmo genético de estado estável que utiliza duas seleções distintas: uma para escolher os indivíduos que deverão sofrer recombinação e outra para escolher os indivíduos da geração mais velha que irão "morrer"para dar lugar às novas soluções recém-descobertas por meio da mutação e recombinação. Na presente pesquisa, esta última seleção foi configurada como Seleção da Pior Aptidão, ou seja, os indivíduos menos aptos seriam escolhidos para dar lugar. Uma outra seleção que pudesse eventualmente selecionar indivíduos bastante aptos, poderia fazer com que a busca do algoritmo genético de estado estável escapasse de mínimos locais em volta desses.

Nos experimentos de *tuning*, os operadores puramente aleatórios, como por exemplo, o *Random Centering* e *Random Partitioning*, tiveram um desempenho bem abaixo dos seus concorrentes. Seria de grande valia investigar se, dado mais tempo de execução para os algoritmos evolucionários, os operadores puramente aleatórios poderiam ser vantajosos, proporcionando uma busca mais global para os algoritmos.

Referências

ALBERTON, R. et al. Multi-agent perimeter patrolling subject to mobility constraints. **American Control Conference (ACC), 2012**, Montreal, QC, CAN, p.4498–4503, June 2012.

ALMEIDA, A. et al. Recent Advances on Multi-agent Patrolling. **Advances in Artificial Intelligence – SBIA 2004**, [S.l.], v.3171, p.474–483, 2004.

BÄCK, T.; SCHWEFEL, H.-P. An Overview of Evolutionary Algorithms for Parameter Optimization. **Evolutionary Computation**, Cambridge, MA, USA, v.1, n.1, p.1–23, Mar. 1993.

CHEVALEYRE, Y.; SEMPE, F.; RAMALHO, G. A Theoretical Analysis of Multi-Agent Patrolling Strategies. **Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3**, Washington, DC, USA, p.1524–1525, 2004.

CHRISTOFIDES, N. **Worst-case analysis of a new heuristic for the travelling salesman problem**. Pittsburgh, PA, USA: Carnegie-Mellon University, 1976.

DOI, S. Proposal and evaluation of a pheromone-based algorithm for the patrolling problem in dynamic environments. **Swarm Intelligence (SIS), 2013 IEEE Symposium on**, Singapore, p.48–55, April 2013.

DURILLO, J. J.; NEBRO, A. J. jMetal: a java framework for multi-objective optimization. **Advances in Engineering Software**, [S.l.], v.42, p.760–771, 2011.

ELMALIACH, Y.; AGMON, N.; KAMINKA, G. Multi-Robot Area Patrol under Frequency Constraints. **Robotics and Automation, 2007 IEEE International Conference on**, [S.l.], p.385–390, April 2007.

ELOR, Y.; BRUCKSTEIN, A. M. Autonomous Multi-agent Cycle Based Patrolling. **Proceedings of the 7th International Conference on Swarm Intelligence**, Berlin, Heidelberg, p.119–130, 2010.

GABRIELY, Y.; RIMON, E. Spanning-tree based coverage of continuous areas by a mobile robot. **Annals of Mathematics and Artificial Intelligence**, [S.l.], v.31, n.1-4, p.77–98, 2001.

GUTIN, G.; PUNNEN, A. **The Traveling Salesman Problem and Its Variations**. [S.l.]: Springer US, 2006. (Combinatorial Optimization).

HERNÁNDEZ, E.; CERRO, J. d.; BARRIENTOS, A. Game theory models for multi-robot patrolling of infrastructures. **International Journal of Advanced Robotic Systems**, [S.l.], v.10, p.181–189, March 2013.

KOENIG, S.; LIU, Y. Terrain Coverage with Ant Robots: a simulation study. **Proceedings of the Fifth International Conference on Autonomous Agents**, New York, NY, USA, p.600–607, 2001.

LAURI, F.; KOUKAM, A. A two-step evolutionary and ACO approach for solving the multi-agent patrolling problem. **Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on**, [S.l.], p.861–868, June 2008.

LAURI, F.; KOUKAM, A. Hybrid ACO/EA algorithms applied to the multi-agent patrolling problem. **Evolutionary Computation (CEC), 2014 IEEE Congress on**, Beijing, China, p.250–257, July 2014.

LUKE, S. **Essentials of Metaheuristics**. second.ed. [S.l.]: Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

MACHADO, A. et al. Multi-agent Patrolling: an empirical analysis of alternative architectures. **Proceedings of the 3rd International Conference on Multi-agent-based Simulation II**, Berlin, Heidelberg, p.155–170, 2003.

MACHADO, A. P. **Patrulha Multiagente**: uma análise empírica e sistemática. 2002. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco, Recife, PE, Brazil.

MARX, D. Searching the K-change Neighborhood for TSP is W[1]-hard. **Oper. Res. Lett.**, Amsterdam, The Netherlands, The Netherlands, v.36, n.1, p.31–36, Jan. 2008.

NILSSON, C. **Heuristics for the traveling salesman problem**. [S.l.]: Tech. Report, Linköping University, Sweden, 2003.

PIPPIN, C.; CHRISTENSEN, H.; WEISS, L. Performance Based Task Assignment in Multi-robot Patrolling. **Proceedings of the 28th Annual ACM Symposium on Applied Computing**, New York, NY, USA, p.70–76, 2013.

POULET, C.; CORRUBLE, V.; EL FALLAH SEGHROUCHNI, A. Auction-Based Strategies for the Open-System Patrolling Task. **PRIMA 2012: Principles and Practice of Multi-Agent Systems**, [S.l.], v.7455, p.92–106, 2012b.

POULET, C.; CORRUBLE, V.; SEGHROUCHNI, A. Working as a Team: using social criteria in the timed patrolling problem. **Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on**, [S.l.], v.1, p.933–938, Nov 2012a.

POULET, C. et al. The Open System Setting in Timed Multiagent Patrolling. **Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on**, [S.l.], v.2, p.373–376, Aug 2011.

ROSEN, K. H. **Discrete Mathematics and Its Applications**. 6th.ed. [S.l.]: McGraw-Hill Higher Education, 2006.

SAMPAIO, P. A. **Patrulha Temporal Taxonomia, Métricas e Novas Soluções**. 2013. Tese (Doutorado em Ciência da Computação) — Universidade Federal de Pernambuco, Recife, PE, Brazil.