Lucas Teles Agostinho
Rodrigo Mendonça da Paixão

Titulo

São Paulo – Brasil

2017

Lucas Teles Agostinho
Rodrigo Mendonça da Paixão

Titulo

Pré-monografia apresentada na disciplina Trabalho de Conclusão de Curso I, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Centro Universitário Senac

Bacharelado em Ciência da Computação

Orientador: Eduardo Heredia

São Paulo – Brasil

2017

Lista de abreviaturas e siglas

AG	Algoritmos Genéticos
IA	Inteligência Artificial
CPU	Central Processing Unit
PCV	Problema do Caixeiro Viajante
PRV	Problema de Roteirização de Veículos

Sumário

1 Introdução

1.1 Motivação

No meio empresarial é essencial pensar na área logística, essa é a área que gerencia os recursos, matérias-primas, componentes, equipamentos, serviços e informação necessária para execução e controle das atividades da empresa. Ela tem como foco orquestrar estes itens de forma a encontrar a melhor condição de operação no menor tempo possível (DIAS, 2010).

Um dos principais pontos dentro da logística é o transporte, onde chega a custar até 60de seu custo total.(RODRIGUES, 2007) Logo é de interesse das empresas conseguir minimizar o custo de escoamento de seus produtos.

Graças a sua importância no processo produtivo a logística se tornou um grande fator competitivo entre empresas. Isso se deve ao fato que a cadeia de suprimento está relacionada com agregação de valores e disponibilidade dos seus bens e serviços para os clientes, fornecedores da empresa e os demais interessados. Independe do lugar que o interessado esteja um serviço ou produto apenas tem valor quando ele está disponível para ser consumindo (TSUDA, 2007).

No planejamento estratégico de logística o principal problema esta relacionado a roteirização de veículos (TSUDA, 2007) também conhecido como PRV, para encontrar a rota menos custosa, é necessário calcular as possíveis combinações de um determinado problema. Contudo, dependendo do numero de combinações pode requerer um processamento muito elevado, levando muito tempo para encontrar a solução ótima. Esse problema se encaixa na categoria NP-Difícil (CUNHA, 2007), Neste tipo de problema não existe uma nenhum algoritmo conhecido que consiga resolvê-lo em tempo polinomial.

1.2 Objetivos

Desenvolver uma solução que resolva o problema de PRV utilizando a metaheurística algoritmos genéticos. Um sistema capaz de calcular uma rota entre vários destinos levando em consideração restrições de tempo e notificando a quantidade de motoristas necessários para realizar todas as entregas ate uma data limite estipulada. Além de levar em consideração o tempo de transito entre estes pontos, permitindo que um motorista possa recalcular a sua rota para otimizar o tempo a qualquer momento.

1.2.1 Objetivos Específicos

-Utilizar API do Google Maps para adquirir informações sobre endereços ou pontos no mapas -Desenvolver um algorítimo genético capas de minimizar a rota entre todos os pontos (caixeiro -Introduzir janelas de tempo nas entregas e adaptar algoritmo genético para levar essas janelas de tempo em consideração -Definir dia/hora limite e dividir entrega em mais de um entregador para respeitar essa hora/data limite -Criar um aplicativo capaz de consultar e recalcular a rota Utilizar uma API de terceiros para adquirir informações sobre endereços ou pontos no mapa, alem do tempo de locomoção entre os pontos. Implementar um algorítimo genético capas de minimizar a rota entre todos os pontos. Introduzir janelas de tempo nas entregas e adaptar o algoritmo genético para levar essas janelas de tempo em consideração. Definir fata/hora limite e dividir entrega em mais de um entregador para respeitar essa hora/data limite caso seja necessário. Criar um aplicativo capaz de consultar e recalcular a rota.

1.3 Método de trabalho

Desenvolver uma implementação de algoritmos genéticos capaz de calcular rotas entre pontos geográficos. Desenvolver uma aplicação Web capaz de receber uma quantidade N de pontos no mapa uma data limite, e a partir deles utilizar a implementação previa para calcular as rotas e notificar o utilizador. Ter uma pagina onde um motorista possa recalcular sua rota em qualquer ponto.

1.4 Organização do trabalho

Este trabalho é dividido em 4 capítulos. O primeiro capitulo faz uma introdução geral do problema, descrever os objetivos e a motivação para a resolução do problema proposto.

O segundo capitulo trata do problema de forma separada, mostrando o que existe na literatura para uma possível solução. Também explica de forma mais detalhada o funcionamento de dois exemplos de busca heurística, demostrando uma aplicação em um trabalho da literatura e dos algoritmos genéticos, explicando seu funcionamento e aplicação na literatura. O terceiro capitulo é a proposta apresentada para a criação deste trabalho.

2 Revisão de Literatura

Nesse capítulo é feita uma revisão no estado da arte dos algoritmos de busca de caminhos, e a aplicação de algoritmos genéticos para mesma finalidade.

2.1 Busca Heurística

Busca heurística de caminhos é um subgrupo de algoritmos de busca em grafos (**??**), é um mecanismo geral de solução de problemas no ramo de IA. A sequência de passos necessários para solução da maioria dos problemas de IA usualmente não é conhecida, mas deve ser determinada por uma exploração de alternativas através de mecanismos de tentativa e erro. Um problema de busca típico é o PCV, que consiste em encontrar o caminho mais curto que as visite cada cidade de um conjunto cidades e retorne para a cidade de partida. Os algoritmos BFS e o A* são otimo exemplos de busca heurística. (**??**)

2.1.1 Aplicações

Aplicações para os algoritmos de busca podem ser diversos, como por exemplo, em um simulador de carro corrida (**??**). Utilizando o algoritmo do A* com duas modificações para encontrar o melhor caminho enquanto evita os obstáculos entre o ponto de início e o ponto de destino. A primeira modificação consiste em utilizar o teorema de Pitágoras, onde primeira calcula a distância entre dois pontos(1 e 2), verifica se existe algum obstáculo dentre eles, se existir, utiliza o terceiro ponto para calcular a hipotenusa e verifica se existe obstáculo entre a hipotenusa, se não existir, remove o ponto 2 e começa a considerar o caminho do ponto 1 para o ponto 3. A segunda modificação consiste em somente ir para frente, isso significa, procura somente os pontos a frente do carro, direita, esquerda e frente, simulando um controle de carro.

O projeto utiliza 3 pistas reais de corrida da Formula 1, Peru, Itália e Hungria, sendo cada pista, uma imagem de escala 1280x782 *pixels* retiradas do site oficial da Formula 1. O Carro é implementado em *Microsoft XNA Game Studio*, plataforma usada para desenvolver jogos para *Windows Phone, Xbox e Windows*. As Imagens das pistas são modificadas para serem mapas de detecção de colisão, a pista é pintada de preto, indicando onde o carro pode andar e o resto pintado de branco indicando os o carro não pode andar. O carro tem o tamanho de 18x12 *pixels* e uma movimentação de 2 *pixels* por segundo. Como o XNA trabalha com uma taxa de quadros de 60 quadros por segundo, o carro se movimenta a 120 *pixels* por segundo.A Figura 1 mostra um exemplo de rota gerada na pista do Peru.

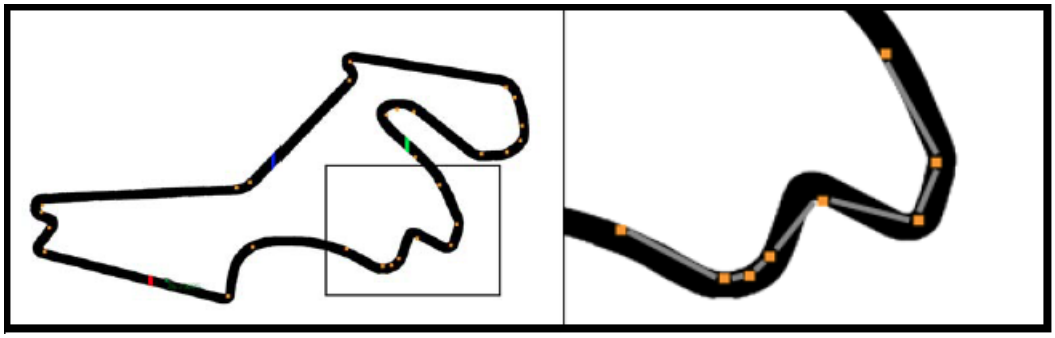

Figura 1 – Pisca de formula 1 do Peru (??)

Os resultados obtidos por (**??**) na primeira modificação conseguiu economizar ciclos de CPU, reduzindo o numero de pontos indicadores da pista em 97%. A segunda modificação tem a vantagem de obter o tempo de volta mais curto, por que reduz o numero de nós do algoritmo A* de 4 para 3 nós]. A desvantagem é que o carro pode balançar em curvas acentuadas. Em geral, as modificações garantiram uma melhoria em performance, economizando o numero de ciclos de CPU.

2.2 Algoritmos genéticos

AG é uma técnica amplamente utilizada de IA, que utilizam conceitos provenientes do princípio de seleção natural para abordar uma ampla série de problemas, geralmente de adaptação. (**??**)

2.2.1 Funcionamento

Inspirado na maneira como o seleção natural explica o processo de evolução das espécies, Holland (**??**) decompôs o funcionamento dos AG em sete etapas, essa são *inicialização*, *avaliação*, *seleção*, *cruzamento*, *mutação*, *atualização* e *finalização* conforme a Figura 2.

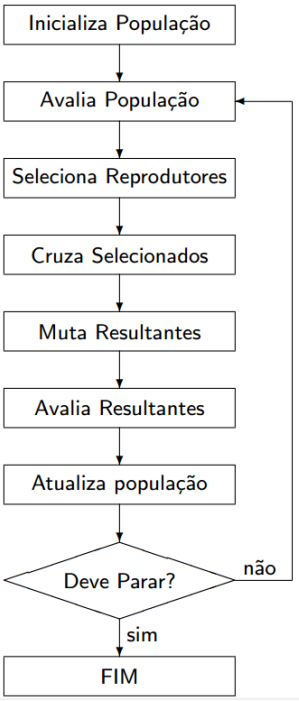

Figura 2 – Estrutura de um AG (??)

2.2.2 Inicialização

Criar uma população de possíveis respostas para um problema. É comum fazer uso de funções aleatórias para gerar os indivíduos, sendo este um recurso simples que visa fornecer maior diversidade.

2.2.3 Avaliação

Avalia-se a aptidão das soluções, os indivíduos da população, então é feita uma análise para que se estabeleça quão bem elas respondem ao problema proposto. A função de avaliação também pode ser chamada de função objetivo. Ela pode variar de acordo com problema, Calcular com exatidão completa o grau de adaptação dos indivíduos pode ser uma tarefa complexa em muitos casos, e se levarmos em conta que esta operação é repetida varias vezes ao longo do processo de evolução, seu custo pode ser consideravelmente alto. Em tais situações é comum o uso de funções não determinísticas, que não avaliam a totalidade das características do indivíduo, operando apenas sobre uma amostragem destas.

2.2.4 Seleção

Ela é a responsável pela perpetuação de boas características na espécie. Neste estágio que os indivíduos são escolhidos para posterior cruzamento, fazendo uso do grau de adaptação de cada um é realizado um sorteio, onde os indivíduos com maior grau de adaptação tem maior probabilidade de se reproduzirem. O grau adaptação é calculado a

partir da função de avaliação para cada individuo, determina o quão apto ele esta para reprodução relativo a sua população.

Selection Random: Gera um numero aleatório entre 0 e o tamanho total da população e retorna o indivíduo do índice escolhido.

Selection Roulette Wheel: Faz a soma de todos os valores da função de aptidão da população, depois calcula a porcentagem de cada indivíduo referente ao total e guarda em um vetor. Então é gerado um valor A aleatório entre 0 e 1 e multiplicado pelo valor total dos pesos. Para selecionar o indivíduo é feito um loop nos pesos e seus valores somados até que o valor A seja igual ou menor que zero, o índice do peso que fez a condição acontecer, se o índice do indivíduo selecionado. Desta forma aumentando a possibilidade de selecionar um indivíduo com maior aptidão.

2.2.5 Cruzamento

Características das soluções escolhidas na seleção são recombinadas, gerando novos indivíduos.

CrossOver Simple: Utiliza dois indivíduos selecionados, define dois números aleatórios de 0 até menor tamanho da lista de cromossomos entre os dois, sendo que o primeiro índice tem que ser menor que o segundo índice e os mesmos não podem ser iguais. Esse tamanho é utilizado para trocar cromossomos entre os dois indivíduos, ou seja, adicionar todos os cromossomos do primeiro indivíduo do índice igual ao primeiro numero, até o índice segundo numeSAro, e repete o processo contrario.

Crossover OBX (Order-Based Crossover): Utiliza dois indivíduos escolhidos na seleção, então define dois números aleatórios, de 0 até menor tamanho da lista de cromossomos entre os dois, sendo que o primeiro tem que ser menor que o segundo e não podem ser iguais. O primeiro numero até o segundo numero, são definidas posições aleatórias e são salvas em uma lista. Faz um loop na lista e troca o cromossomo da posição do primeiro indivíduo para o segundo e do segundo para o primeiro.

Crossover PBX (Position-Based Crossover): Utiliza dois indivíduos selecionados, então define dois números aleatórios, de 0 até menor tamanho da lista de cromossomos entre os dois, sendo que o primeiro índice tem que ser menor que o segundo índice e os mesmos não podem ser iguais. Entre esse tamanho são definidas posições aleatórias e guardadas em uma lista. Os indivíduos resultantes são zerados, e para cada posição é trocado do cromossomo principal para o resultante de mesma posição outro da mesma posição. As posições não preenchidas são completadas com os cromossomos restante, seguindo a ordem do cromossomo e adicionado se ele não ja existir na lista.

2.2.6 Mutação

Características dos indivíduos resultantes do processo de reprodução são alteradas, acrescentando assim variedade a população. A mutação opera sobre os indivíduos resultantes do processo de cruzamento e com uma probabilidade pré-determinada efetua algum tipo de alteração em sua estrutura. A importância desta operação é o fato de que uma vez bem escolhido seu modo de atuar, é garantido que diversas alternativas serão exploradas.

MutateEM (Exchange Mutation): Define duas das posições aleatórias distintas do segundo cromossomo até o ultimo, e troca os cromossomos do indivíduo.

MutateSM (Scramble Mutation): Define duas das posições aleatórias distintas do segundo cromossomo até o ultimo, e uma quantidade aleatória. Então faz um loop da quantidade aleatória e mistura os cromossomos que estão entre a posição inicial e final trocando aleatoriamente dois pontos entre eles.

MutateDM (Displacement Mutation): Define duas das posições aleatórias distintas do segundo cromossomo até o ultimo, e remove todos os cromossomo entre essa posições e recoloca a partir de uma posição aleatória.

MutateIM (Insertion Mutation): Define uma posição aleatória, remove o cromossomo da posição, reorganiza os cromossomos e insere o cromossomo removido em uma nova posição aleatória.

MutateIVM (Inversion Mutation): Define duas das posições aleatórias distintas do segundo cromossomo até o ultimo, e inverte todos os cromossomos que está entre as posições.

MutateDIVM (Displaced Inversion Mutation):Define duas das posições aleatórias distintas do segundo cromossomo até o ultimo, e remove todos os cromossomo entre essa posições e recoloca a partir de uma posição aleatória de forma invertida.

2.2.7 Atualização

Os indivíduos criados no processo de reprodução e mutação são inseridos na população.

Na forma mais tradicional deste a população mantém um tamanho fixo e os indivíduos são criados em mesmo número que seus antecessores e os substituem por completo.

Existem, porém, algumas alternativas, o número de indivíduos gerados pode ser menor ou o tamanho da população pode sofrer variações e o critério de inserção pode variar, por exemplo, nos casos em que os filhos substituem os pais, ou em que estes só são inseridos se possuírem maior aptidão que o cromossomo que sera substituído, ou o manter sempre o conjunto dos n melhores indivíduos.

2.2.8 Finalização

É testado se as condições de encerramento da evolução foram atingidas, retornando para a etapa de avaliação em caso negativo e encerrando a execução em caso positivo.

Os critérios para a parada podem ser vários, desde o número de gerações criadas até o grau de convergência da população atual.

Toda base dos AG se fundamenta nos indivíduos, eles são a unidade básica em qual o algoritmo se baseia, sua função é codificar as possíveis soluções do problema a ser tratado e partir de sua manipulação no processo evolutivo, a partir daí que são encontradas as respostas.

Esses indivíduos precisam de uma representação, essa será o principal responsável pelo desempenho do programa. É comum chamar de *genoma* ou *cromossomo* para se referir ao individuo. Por essa definição podemos resumir um indivíduo pelos genes que possui, ou seja seu *genótipo*.

Apesar de toda representação por parte do algoritmo ser baseada única e exclusivamente em seu genótipo, toda avaliação é baseada em seu fenótipo, o conjunto de características observáveis no objeto resultante do processo de decodificação dos genes do individuo, ver Tabela 1.

Tabela 1 – Exemplos de genótipos e fenótipos correspondentes em alguns tipos de problemas (??)

Problema	**Genótipo**	**Fenótipo**
Otimização numérica	0010101001110101	10869
Caixeiro viajante	CGDEHABF	Comece pela cidade C, depois passe pelas cidades G, D, E, H, A, B e termine em F
Regras de aprendizado para agentes	$C_1R_4C_2R_6C_4R_1$	Se condição 1 (C_1) execute regra 4 (R_4), se (C_2) execute (R_6), se (C_4) execute (R_1)

Para cada indivíduo é calculado o seu grau de adaptação, a partir de uma função objetivo, comumente denotada como na formula 2.1.

$$f_O(x) \tag{2.1}$$

Que vai representar o quão bem a resposta apresentada pelo individuo soluciona o problema proposto.

Também é calculado o grau de adaptação do indivíduo relativo aos outros membros da população a qual ele pertence, esse é chamado de grau de aptidão, para um indivíduo x temos seu grau de aptidão denotado pela fórmula 2.2.

$$f_A(x) = \frac{f_O(x)}{\sum_{i=1}^{n} f_O(i)} \tag{2.2}$$

Sendo n o tamanho da população.

A dinâmica populacional é a responsável pela evolução, ao propagar características desejáveis a gerações subsequentes no processo de cruzamento, enquanto novas são testadas no processo de mutação.

Algumas definições importantes relativo as populações de um AG são:

Geração: É o número de vezes em que a população passou pelo processo de seleção, reprodução, mutação e atualização.

Média de adaptação: É a taxa média que ao indivíduos se adaptaram ao problema, é definida pela formula 2.3.

$$M_A = \frac{\sum_{i=1}^{n} f_O(i)}{n} \tag{2.3}$$

Grau de convergência: define o qual próxima esta a media de adaptação desta população relativo as anteriores. O objetivo dos AG é fazer a população convergir para uma valor de adaptação ótimo. Um estado negativo que pode ocorrer relativo a esta medida é a *convergência prematura*, a mesma ocorre quando a população converge em uma média de adaptação sub-ótima, e dela não consegue sair por causa de sua baixa diversidade.

Diversidade: Mede o grau de variação entre os genótipos da população. Ela é fundamental para o tamanho da busca. Sua queda esta fortemente ligada ao fenômeno de *Convergência prematura.*

Elite: São os indivíduos mais bem adaptados da população. Uma técnica comum nos AG é p *elitismo*, onde são selecionados k melhores indivíduos que serão mantidos a cada geração.

2.2.9 Aplicações

Existem vários aplicações para os algoritmo genéticos, por serem uma inteligência artificial não supervisionada, de rápido aprendizado e podendo ser paralelizado.

O modelo m-PRC(Problema de Rotas de Cobertura multi-veículo) é uma aplicação de algoritmos genéticos para construção de rotas em uma região mapeada, para encontrar uma boa distribuição de viaturas para patrulhamento urbano usado por departamentos

de segurando como a policia, guardas municipais ou segurança privada (**??**). O Modelo é definido como um grafo não direcionado 2.4.

$$G = (V \cup W, E) \tag{2.4}$$

Onde 2.5:

$$V \cup W \tag{2.5}$$

Compõem o conjunto de vértices e E o conjunto de arestas, ou seja, o subgrafo induzido por E e um grafo completo cujo conjunto de nós é V. V são todos os vértices que podem ser visitados e é composto pelo subconjunto T, que são os vértices que devem ser visitados por algum veiculo. W é um conjunto de vértices onde todos os M veículos devem passar. M é o numero de rotas de veículos que começam no vértice base V_0.

O m-PRC atribui o conjunto de m rotas de veículos com as restrições: todas as m rotas de veículos começam e terminam na base V_0, Tem exatamente m rotas, cada vértice de V pertence a no máximo uma rota, cada vértice de T pertence a exatamente uma rota, com exceção a base, cada vértice de W deve ter uma rota que passa por ele e em uma distancia C de um vértice V visitado, O modulo da diferença entre o número de vértices de diferentes rotas não pode exceder um determinado valor R. A Figura 3 mostra o grafo da relação de V com W.

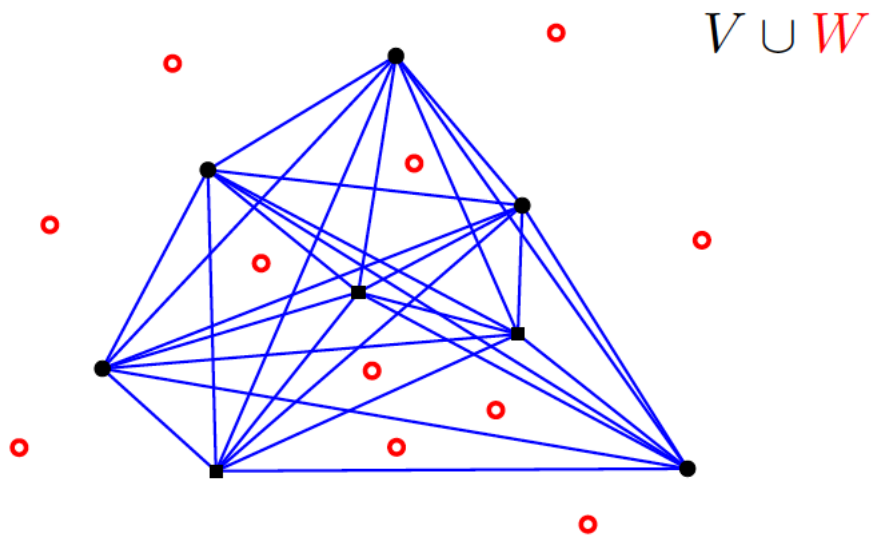

Figura 3 – Exemplo de gráfo não direcionado para V U W. (??)

Para utilizar o algoritmos genéticos com o modelo m-PRC, o trabalho propõem dois modelos. O AGS (Algoritmo genético sequencial), que utiliza heurísticas GENIUS e 2-opt balanceada para ajustes finais para tentar melhor a solução; O AGH(Algoritmos genéticos H-1-PRC), que utiliza heurísticas H-1-PRC-MOD e 2-opt balanceada em todo o

processo de resolução.

A conclusão de (**??**) é que a utilização de algoritmos genéticos para a resolução de de uma adaptação do problema de rotas de cobertura de veículos como bastante relevantes e de fácil manipulação. O modelo AGS resolve o problema de forma rápida e tem uma fácil implementação dentro dos critérios de comparação adotadas. O modelo AGH é mais lento e não conseguiu encontrar a solução para alguns exemplos.

2.3 Algoritmos genéticos para busca de caminhos

3 Proposta

4 Metodologia

Para realizar os testes precisamos de uma grande quantidade de mapas. Para isso desenvolvemos uma ferramenta que os gera de forma automática a partir de parâmetros previamente definidos.

Nestes parâmetros definimos um tamanho $N \times M$ para o mapa, a densidade percentual que o mapa será coberto por obstáculos, se haverá ou não movimentação diagonal e se queremos um mapa gerado de forma totalmente aleatória ou que seja gerado repetindo um padrão de bloco de tamanho $L \times L$. O tamanho L do bloco que será repetido também é definido na configuração do gerador, é gerado de forma aleatória nos mesmos padrões do mapa totalmente aleatório.

Também é definido o tamanho mínimo do caminho solução do mapa, ou seja, o tamanho do caminho entre o ponto inicial e final. Cada mapa gerado é utilizado o algoritmo A* para verificar se é solúvel e o tamanho do caminho maior ou igual ao tamanho de caminho mínimo definido nos parâmetros. O caminho mínimo é importante para evitar que seja gerado mapas onde o ponto inicial e final estão muito próximos.

Pelo fato de ter pouco tempo para a coleta dos dados, não foi possível analisar todas as possibilidades das configurações do GA. Como pode ser visto na tabela a baixo.

Em mapas 100x100 podem existir momentos que o GA não está encontrando uma solução, com isso faz com que ele demore para chegar no limite de gerações, que está definido em 1000. Com uma média de 30 segundos por mapa, todas as configurações existe na implementação se torna inviável para o período de tempo para a finalização do projeto.

Tabela 2 – Possibilidades do GA

Nome	**Total**	**Escolha**
Tamanho	1	2
Tipo Mapa	2	2
Movimentação	4	2
Qt. Mapas	100	50
Qt. Repetição GA	10	4
Heurísticas	4	4
Fitness	4	3
Cruzamento	3	2
Mutação	7	2
Seleção	2	1
Total	**5.376.000**	**76.800**
Tempo Médio	30	30
Dias	**1866,67**	**26,67**

Tabela 3 – Comparação Fitness

	Média de Tempo (ms)		**Não Encontrou**	
Fitness	**10x10**	**100x100**	**10x10**	**100x100**
FitnessHeuristic	**106**	62.542	**3**	**59**
FitnessWithCirclicValidation	991	**34.119**	91	191
FitnessWithCollisionDetection	1.467	**30.405**	105	202
FitnessWithCollisionDAndCV	**510**	44.943	**21**	**103**
Total	**768**	**43.002**	**220**	**555**

Tabela 4 – Comparação Mutação

	Média de Tempo(ms)		**Não encontrou**	
Mutações	**10x10**	**100x100**	**10x10**	**100x100**
MutateBitwise	938	42.755	50	76
MutateDIVM	**753**	**42.161**	**26**	**87**
MutateDM	**617**	44.924	**20**	**80**
MutateEM	781	43.362	35	74
MutateIM	728	43.324	25	73
MutateIVM	813	44.133	34	85
MutateSM	749	**40.356**	30	80
Total	**768**	**43.002**	**220**	**555**

Tabela 5 – Comparação Cruzamento

	Média de Tempo(ms)		**Não encontrou**	
Rótulos de Linha	**10x10**	**100x100**	**10x10**	**100x100**
CrossoverOBX	**167**	56.548	5	**121**
CrossoverPBX	1.777	**33.921**	200	259
CrossoverSimple	**362**	**38.537**	15	**175**
Total Geral	**768**	**43.002**	**220**	**555**

Para a decisão de como ficaram os parâmetros finais, utilizamos dois tipos mapas, pequeno 10x10 e grande 100x100, sendo gerados 10 mapas de cada 1 vez para as configurações do GA, como o tempo é o melhor problema, ele foi o parâmetro para a tomada de decisão.

Decidimos escolher 2 mais rápidas das 7 mutações , 2 mais rápidos dos 3 Cruzamentos e como a Fitness é o processo muito importante do GA, as 3 mais rápidas das 4 Fitness. Reduzimos de 10 para 4 o número de vezes que o GA vai rodar sobre o mesmo mapa e o numero todos de mapas para de 100 para 50. Como tivemos diferentes resultados com as Fitness para tamanho diferentes de mapas, decidimos por 2 tipos de tamanho de mapa, 20x20 e 100x100.

Para a análise foram gerados mapas de tamanho 100×100 com um caminho mínimo mínimo de 15 passos e uma densidade de obstáculos de 30%. No total foram gerados 400 mapas, sendo divididos em 200 gerados a partir de um padrão e 200 gerados aleatoriamente. Para cada mapa com padrão é gerado aleatoriamente um bloco de 5×5 que é repetido

até completar o tamanho total do mapa.

Os 200 mapas de cada tipo são divididos em 2, de forma que sejam 100 mapas para que permitem movimentação diagonal (figura:5) e 100 que não permitem (figura:4).

A configuração de movimentação diagonal interfere diretamente na heurística utilizada e na geração do mapa, o mesmo pode ou não ter uma solução dependendo do tipo de diagonal. Logo o mapa deve ser gerado levando isso em consideração.

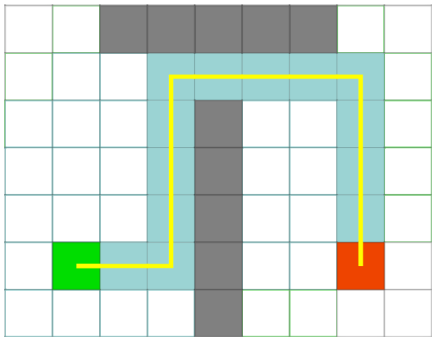

Figura 4 – Não permite andar nas verticais, podendo somente se movimentar para cima, baixo, direita e esquerda.

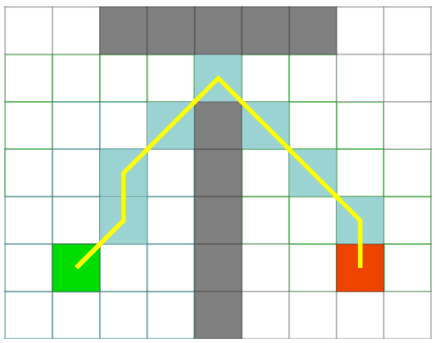

Figura 5 – É permitidos seguir na vertical, se um dos lados estiver livre.

Os algoritmos clássicos de busca que utilizamos para comparação são A*, BFS, Dijkstra e IDA*, sendo executados uma vez para cada uma das heurísticas selecionadas para os testes, essas são Manhattam, Euclideana, Octil e Chebyshev.

Devido à natureza não determinística do Algoritmo genético rodamos ele 5 vezes para cada mapa, e calculamos a média de tempo, custo de memória e tamanho de caminho resultado para comparação com os algoritmos clássicos.

As possíveis configurações do GA dependem dos operadores de cruzamento, mutação e seleção, alem da função de aptidão e a heurísticas utilizada por ela. Os operados de cruzamento implementados para a analise são *"Simples"*, OBX e o PBX. Os operadores de Mutação são Bitwise, DIVM, DM, EM, IM,IVM e SM. Na seleção utilizamos o algoritmo de Roleta. A função de aptidão é baseada nas heurísticas implementadas, sendo uma a heurística sem nenhuma alteração, outra implementando a heurística e penalizando caminhos cíclicos, outra implementando a heurística e penalizando encontro com paredes

e a ultima implementando a heurística e penalizando tanto caminhos cíclicos quanto encontro com paredes.

Para os testes de GA, utilizamos a heurística Manhattam para movimentação sem diagonal, e Octile para movimentação com diagonal, a escolha é devido ao fato da heurística ser fortemente ligada a forma de movimentação como é citado no artigo (**??**), as heurísticas selecionadas garantem melhor resultado para cada movimentação selecionada.

5 Implementação

Nesse capitulo será apresentado mais aprofundadamente as ferramentas e métodos que foram utilizados para realizar os testes e implementações dos modelos de busca de caminho.

5.1 Tecnologias

O projeto é separado em Pathfinder e Pathfinder.UI ambos desenvolvidos na linguagem C# o primeiro utilizando o .NET Standard Library 1.6 e o segundo no .NET Core. No projeto Pathfinder estão todas a implementações dos algoritmos de busca que temos para a comparação. Pathfinder.UI tem como objetivo consumir os recursos do projeto Pathfinder, dando opções de visualização ou geração de dados. Ambos rodam em sistemas Windows e *nix utilizando o .Net Core CLI 1.1 para execução.

5.2 Estrutura do Projeto

Essa seção tem como objetivo descrever como foram implementados os algoritmos

5.2.1 PathFinder

Projeto de implementação de algoritmos de busca. Sua estrutura de pastas está organizada em Abstraction, Core, Factories, Finders, GeneticAlgorithm, Heuristics e MapGenerators.

O **'project.csproj'** é o arquivo onde é definido as bibliotecas utilizadas e a versão do .NET Framework, as outras pastas agregam arquivos com informações relevantes a nossa implementação.

5.2.2 Abstraction

Nesta pasta estão todos os arquivos a nível de abstração dos algoritmos de busca, esses são:

IFactory: Essa interface tem como objetivo padronizar as "fabricas", ferramentas que decidir e instanciar toda dependência necessária.

IMap: Essa interface tem como objetivo abstrair o comportamento da classe de mapa utilizada nos arquivos de busca, assim sendo por padrão todo algoritmo espera uma implementação de IMap para rodar.

IHeuristic: Essa interface abstrai o comportamento das heurísticas.

IMapGenerator: Essa interface tem como objeto abstrair os gerador de mapas.

IFinder: Essa interface é a responsável por abstrair todo comportamento dos algoritmos de busca.

IGeneticAlgorithm: Essa interface herda de IFinder, ela compartilha a mesma assinatura de métodos, propriedades e eventos, porem acrescenta a abstração necessária para o utilização de GA.

5.2.3 Core

Nesta pasta são definidos as implementações e configurações bases.

Container: Esta classe é responsável por registar e resolver as implementações conhecidas das interfaces.

Enumerators: Contem as definições de enumerações, usados para usar nomes bem definidos ao invés de números avulsos no código.

Extensions: Arquivo com métodos auxiliares de lista para comportamento de uma estrutura de pilha.

FileTools: Classe responsável por toda manipulação de I/O de arquivos

Map: Implementação do IMap, tem como objetivo ser a estrutura de mapa base dos algoritmos de busca.

Node: Classe responsável por ser a representação de uma célula no mapa, ou seja, o mapa é uma matriz de ***"Node"***.

Settings: Contém toda configuração estática do projeto, do qual é carregado de um arquivo Json chamado "appsettings.json"

5.2.4 Factories

Nesta Pasta temos os arquivos responsáveis pelo instanciar as implementações de interfaces.

FinderFactory: Classe responsável por decidir e instanciar uma implementação IFinder.

HeuristicFactory: Classe responsável por decidir e instanciar uma implementação IHeuristic.

MapGeneratorFactory: Classe responsável por decidir e instanciar uma implementação IMapGenerator.

5.2.5 Finders

Nesta pasta temos definidas as implementações de todos os algoritmos de busca de caminho.

AStarFinder: Implementação do algoritmo de busca de caminho A* implementada em cima da interface IFinder.

BestFirstSearchFinder: Implementação do algoritmo de busca de caminho "Best First Search" implementada em cima da interface IFinder.

DijkstraFinder: Implementação do algoritmo de busca de caminho Dijkstra implementada em cima da interface IFinder.

IDAStarFinder: Implementação do algoritmo de busca de caminho IDA* implementada em cima da interface IFinder.

GAFinder: Implementação de um algoritmo genético para busca de caminhos implementada em cima da interface IFinder e IGeneticAlgorithm.

5.2.6 Heuristics

Nesta pasta são definidas as implementações de IHeuristic, responsáveis pelos cálculos de heurística.

Manhattan: Implementação da classe Manhattan implementada em cima da interface IHeuristic responsável por calcular a distancia Manhattam.

Euclidean: Implementação da classe Euclidean implementada em cima da interface IHeuristic responsável por calcular a distancia Euclideana.

Octile: Implementação da classe Octile implementada em cima da interface IHeuristic responsável por calcular a distancia Octile.

Chebyshev: Implementação da classe Chebyshev implementada em cima da interface IHeuristic responsável por calcular a distancia Chebyshev.

5.3 Genetic Algorithm

Nesta pasta são definidos todas as implementações referentes ao algoritmo genético, pela complexidade. do algoritmo ele possui uma estrutura própria de pastas para definições e configurações de injeção de dependência.

5.3.1 Abstraction

Nesta pasta estão todos os arquivos a nível de abstração das etapas do algoritmo genético.

ISelection: Interface é responsável por abstrair os algoritmos de seleção.

IGenome: Interface tem como funcionalidade abstrair a definição de genoma.

IFitness: Interface tem como objetivo abstrair o calculo de fitness.

IMutate: Interface tem como objetivo abstrair os operadores de mutação.

ICrossover: Interface tem como objetivo abstrair os operadores de cruzamento.

IRandom: Interface tem como objetivo abstrair a implementação de geração de números aleatórios.

AbstractMutate: Implementação base para operador de mutação.

AbstractCrossover: Implementação base para operador de cruzamento.

5.3.2 Core

Adaptation: Classe responsável para realizar a adaptação de um indivíduo novo após ser gerado.

Enumerators: Contem as definições de enumerações, usados para usar nomes bem definidos ao invés de números avulsos no código.

GARandom: Implementação responsável por gerar números aleatórios, implementa IRandom.

GASettings: Arquivo responsável por carregar configuração estática de GA, carrega do arquivo "GASettings.json".

Genome: Classe responsável por representar o genoma no algoritmo de GA, implementa a IGenome.

5.3.3 Selection

Nesta pasta estão todas as implementações dos algoritmos de seleção.

SelectionRandom: Implementação de seleção de indivíduos aleatório. **SelectionRouletteWheel**: Implementação de seleção roleta.

5.3.4 Crossover

Nesta pasta estão todas as implementações dos algoritmos de cruzamento.

CrossoverOBX: Implementação do operador de cruzamento OBX.

CrossoverPBX: Implementação do operador de cruzamento PBX.

CrossoverSimple: Implementação do operador de cruzamento simples.

5.3.5 Mutation

Nesta pasta estão todas as implementações dos algoritmos de mutação.

MutateBitwise: Implementação do operador de cruzamento Bitwise.

MutateDIVM: Implementação do operador de cruzamento DIVM.

MutateDM: Implementação do operador de cruzamento DM.

MutateEM: Implementação do operador de cruzamento EM.

MutateIM: Implementação do operador de cruzamento IM.

MutateIVM: Implementação do operador de cruzamento IVM.

MutateSM: Implementação do operador de cruzamento SM.

5.4 Projeto de UI

Foi desenvolvido um projeto com objetivo de consumir a biblioteca de busca de caminhos, e poder visualiza-los.

5.4.1 Abstraction

Nesta pasta estão todos os arquivos a nível de abstração.

IAppMode: Abstração que define de que forma o app ira rodar.

IViewer: Abstração do tipo de visualizador.

5.4.2 AppMode

Pode-se configurar diferentes modos de rodar os algoritmo, nesse pasta estão a implementação das diferentes formas.

SingleRunMode: O programa será executado e rodara uma vez usando as configurações do arquivo estático "appsettings.json"que é lido pela classe UISettings.

DynamicMode: O programa ira perguntar qual algoritmo, heurística, tipo de diagonal, forma de visualização e cada operador do GA antes de rodar.

BatchMode: O software ira rodar N vezes cada algoritmo selecionado no arquivo de configuração, onde N também é definido neste arquivo, ao final ira salvar os resultados e cada mapa numa pasta na raiz do projeto.

5.4.3 Core

Nesta pasta são definidos as implementações e configurações bases.

Enumerators: Contem as definições de enumerações, usados para usar nomes bem definidos ao invés de números avulsos no código.

RegisterConfig: Neste arquivo são configurados os binds do visualizador para injeção de dependência.

Settings: Onde são carregados as configurações estáticas do arquivo "appsettings.json", neste são configurações da forma de visualização e do Batch.

5.4.4 Factories

Nesta Pasta temos os arquivos responsáveis pelo instanciar as implementações de interfaces.

AppModeFactory: Classe responsável por decidir e instanciar uma Implementação de IAppMode.

ViwerFactory: Classe responsável por decidir e instanciar uma Implementação de IViewer.

5.4.5 Viewer

Nesta pasta estão as diferentes formas de exibir os resultados das buscas.

ConsoleViewer: Classe responsável por apresentar a busca de caminhos em ASC no Console da aplicação.

OpenGLViewer: Classe responsável por apresentar a busca em uma janela em OpenGL.

OpenGLWindow: Classe que é utilizada pela OpenGLViewer para mostrar a janela com uma grid que mostra o andamento dos algoritmos.

5.5 Estrutura do GA

A busca utilizando o GA, segue com as operações base de todo GA, que são seleção, cruzamento, adaptação e mutação. Para cada interação é gerada uma população, a função de aptidão é calculada para cada indivíduo da população, desses o indivíduo com o valor mais próximo de zero é selecionado como o melhor.

5.5.1 Função de Aptidão

Nesta pasta estão as implementações das funções fitness.

FitnessHeuristic: Para cada indivíduo da população é calculada uma aptidão com base em uma função heurística (Manhattam, Octile, Euclideana, Chebyshev) previamente

definida, o calculo é feito a partir do ponto final da lista do genoma do indivíduo até o ponto de destino do mapa a soma de todos os resultados é a função de aptidão da população.

FitnessWithCollisionDetection: Para cada indivíduo da população é calculada a função de aptidão idêntica a FitnessHeuristic, porém no processo de adaptação adaptação os caminhos que aumentarem para um caminho inválido, que colidam ou saiam do mapa, são marcados, posteriormente todos os indivíduos marcados são penalizados com a soma de um valor alto para diminuir o valor de sua aptidão na população.

5.5.2 Adaptação

Ela é importante para corrigir possíveis problemas nos indivíduos resultantes dos operadores de cruzamento ou mutação. Quando os cromossomos são reorganizados, o caminho novo que foi gerado pode levar para cima de um bloqueio ou para fora do mapa, então seguindo as direções indicadas no cromossomo, a posição no mapa é recalcula e só adiciona o cromossomo do indivíduo se for uma posição valida ou que não voltam para o mesmo lugar. Para complementar o caminho do indivíduo, uma nova direção valida é adicionada e calculada, fazendo com que cada interação de adaptação, o caminho cresça.

5.5.3 Mutação

Todas as mutações executam se o indivíduo tiver mais do que 3 cromossomos e não afeta o primeiro cromossomo. O primeiro cromossomo é a ligação com o ponto inicial e se for trocado de lugar, o caminho é quebrado. /citeMatBuckland

5.6 Modo Batch

O modo Batch do programa serve para poder gerar os dados necessários para a analise.

Para iniciar o processo do batch é necessário o mudar o *AppMode* no arquivo "appsettings.json"para 2 e iniciar a aplicação Pathfinder.UI.

5.6.1 Configuração

A definição das configurações para o Batch ficam no arquivo "appsettings.json"onde são carregadas na classe UISettings posteriormente na aplicação. As chaves que interessam ao modo batch são:

Width: Tamanho em largura dos mapas.

Height: Tamanho em altura dos mapas.

MinimumPath: Tamanho minimo entre o ponto inicial e final dos mapas.

RandomBlock: Tamanho que sera usado para gerar aleatoriamente uma parte do mapa e se repetira para completar os mapas com padrão.

Batch_map_origin: Define se você quer carregar os mapas previamente gerados ou gerar mapas novos para a execução.

Batch_map_qtd_to_generate: Quantidade de mapas que serão gerados para o teste.

Batch_generate_pattern: Este campo define se os mapas gerados automaticamente serão aleatórios (0) ou se terão algum padrão de repetição (1).

Batch_map_diagonal: Define qual tipo de movimentação diagonal sera usada na geração dos mapas, onde Never(0), OnlyWhenNoObstacles(1), IfAtMostOneObstacle(2), Always(3).

Batch_GATimesToRunPerMap: Define a quantidade de vezes que o GA ira rodar para cada configuração.

Batch_folder: Pasta aonde serão gerados os mapas e arquivo de log do batch.

Batch_list_finders: Uma lista que define quais algorítimos de busca serão rodados no modo batch, as opções são A*(0), BFS(1), IDA*(2), Dijkstra(3), GA(4).

Batch_list_heuristics: Uma lista que define quais heurísticas serão rodados no modo batch, as opções são Manhatam(0), Euclidean(1), Octile(2), Chebyshev(3).

Batch_list_Mutation: Uma lista que define quais operadores de mutação serão utilizados pelo GA no modo batch, as opções são *Exchange(0)*, *Displaced Inversion*(1), *Displacement*(2), *Insertion*(3), *Inversion*(4), *Scramble*(5) e *Bitwise*(6).

Batch_list_Crossover: Uma lista que define quais operadores de cruzamento serão utilizados pelo GA no modo batch, as opções são *Simple*(0), *Order-Based Crossover*(1), *Position-Based Crossover*(2).

Batch_list_Fitness: Uma lista que define quais funções de aptidão serão utilizados pelo GA no modo batch, as opções são Heurística(0), Heurística com penalidade em caminho cíclico(1), Heurística com penalidade em caminhos que colidem(2) e Heurística com penalidade em caminho cíclico e colisão(3).

Batch_list_Selection: Uma lista que define quais funções de seleção serão utilizados pelo GA no modo batch, as opções são Aleatório(0) e *Roulette Wheel Selection*(1).

5.7 Modo Visual

O modo Visual do programa serve para poder rodar um algorítimo em um mapa especifico e acompanhar visualmente como ele se comporta.

Para iniciar o modo visual é necessário o mudar o *AppMode* no arquivo "appsettings.json"para 0 e iniciar a aplicação Pathfinder.UI.

5.7.1 Configuração

A definição das configurações para o modo visual ficam no arquivo "appsettings.json"e "GASettings.json"onde são carregadas na classe UISettings posteriormente na aplicação. As chaves que interessam ao modo visual são:

MapViwer: Define o tipo de visualização que sera utilizada onde as opções são Console(0) e OpenGL(1).

MapOrigin: Define a origem do mapa utilizado no programa, onde as opções são Estático(0), De um arquivo(1), Aleatório(2), Com padrão(3).

Heuristic: Define qual heurística sera rodado no modo visual, as opções são Manhatam(0), Euclidean(1), Octile(2), Chebyshev(3).

AllowDiagonal: Define qual tipo de movimentação diagonal sera usada, onde Never(0), OnlyWhenNoObstacles(1), IfAtMostOneObstacle(2), Always(3).

Algorithm: Define qual algoritmo de busca sera rodar no modo visual, as opções são A*(0), BFS(1), IDA*(2), Dijkstra(3), GA(4).

IDAStarFinderTimeOut: Define em quanto tempo em milissegundos o IDA* ira parar caso não encontre o caminho.

IDATrackRecursion: Define se ira mostrar todos os passos recursivos do IDA* (pode demorar).

RandomSeed: Seed utilizada no gerador aleatório de mapas.

Width: Tamanho em largura do mapa.

Height: Tamanho em altura do mapa.

MinimumPath: Tamanho minimo entre o ponto inicial e final do mapa.

RandomBlock: Tamanho que sera usado para gerar aleatoriamente uma parte do mapa e se repetira para completar o mapa com padrão.

FileToLoad: Arquivo de mapa que sera carregado caso escolhe o MapOrigin de arquivo.

FolderToSaveMaps: Pasta onde serão salvas automaticamente os arquivos de

mapas gerados.

Start: Define o caractere de ponto inicial no modo de visualização console.

End: Define o caractere de ponto final no modo de visualização console.

Empty: Define o caractere de caminho livre no modo de visualização console.

Wall: Define o caractere de parede no modo de visualização console.

Path: Define o caractere de caminho no modo de visualização console.

Closed: Define o caractere de nó fechado no modo de visualização console.

Opened: Define o caractere de nó aberto no modo de visualização console.

OpenGlBlockSize: Tamanho em pixel dos blocos no modo OpenGL.

GenerationLimit: Define o numero máximo de gerações do GA antes de abortar o processo.

MutationRate: Chance de ocorrer mutação em cada individuo de uma nova população.

MutationAlgorithm: Define qual operador de mutação sera utilizados pelo GA no modo visual, as opções são *Exchange(0)*, *Displaced Inversion*(1), *Displacement*(2), *Insertion*(3), *Inversion*(4), *Scramble*(5) e *Bitwise*(6).

CrossoverRate: Chance de ocorrer cruzamento em cada par de indivíduos de uma nova população.

CrossoverAlgorithm: Define qual operador de cruzamento serão utilizados pelo GA no modo visual, as opções são *Simple*(0), *Order-Based Crossover*(1), *Position-Based Crossover*(2).

PopulationSize: Quantidade de indivíduos em cada nova população.

FitnessAlgorithm: Define qual função de aptidão sera utilizado pelo GA no modo visual, as opções são Heurística(0), Heurística com penalidade em caminho cíclico(1), Heurística com penalidade em caminhos que colidem(2) e Heurística com penalidade em caminho cíclico e colisão(3).

SelectionAlgorithm: Define qual função de seleção sera utilizado pelo GA no modo visual, as opções são Aleatório(0) e *Roulette Wheel Selection* (1).

BestSolutionToPick: Quantidade de indivíduos mais bem avaliados a serem colocados em uma nova geração a partir da anteriores.

Penalty: Valor de penalidade usado nas funções de aptidão.

6 Conclusão

6.1 Resultados

6.2 Trabalhos futuros

Referências

CUNHA, C. B. da. Aspectos práticos da aplicação de modelos de roteirização de veículos a problemas reais. transportes. 2007. Citado na página 5.

DIAS, M. A. P. *Administração de materiais: uma abordagem logística.* [S.l.: s.n.], 2010. Citado na página 5.

RODRIGUES, P. R. A. *Introdução aos sistemas de transporte do Brasil e à logística internacional.* [S.l.: s.n.], 2007. Citado na página 5.

TSUDA, D. S. Modelo de roteirização de veículos em uma empresa importadora de produtos japoneses. 2007. Citado na página 5.