UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Rodolfo César Bispo

Planejador de Roteiros Turísticos: Uma Aplicação do Problema do Caixeiro Viajante na Cidade do Recife

Recife

2018

Rodolfo César Bispo

Planejador de Roteiros Turísticos: Uma Aplicação do Problema do Caixeiro Viajante na Cidade do Recife

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Gilberto A. de A. Cysneiros Filho

Recife

2018

À minha família, amigos e professores

Agradecimentos

Agradeço a minha mãe e meus irmãos por me incentivarem durante essa longa jornada de faculdade. Agradeço a minha querida namorada Indira, que compartilha comigo todos os momentos de felicidades e tristezas atuando como principal suporte e incentivo para não desistir. Você é incrível. Agradeço a Guilherme Magalhães que mesmo longe, fez diferença em toda minha trajetória até hoje. Aos meus verdadeiros amigos Bizurados e RID que me proporcionaram momentos incríveis na faculdade. Agradeço especialmente a Daniel, Raffael Vieira, José Clodoalves, Bruno Roberto, Gustavo, Demis, Raphael Alves, Robson César, Robson Ugo, Vinicius Acioly, Leonardo, Lisandra, Raissa, Gabi, Jorge, Edvan e João. Vocês são maravilhosos.

Meu muito obrigado aos meus amigos do GD: Lara, Gabriel,Isa, Gabby, Thiago Chesus, Karla, Bia, Brenda, Amanda Amorim e Dan, que incentivaram nessa reta final me motivando até aqui. Vocês não sabem o quanto são especiais.

Agradeço a todos os meus professores, em especial ao meu orientador Gilberto que foi importantíssimo com seu apoio integral durante o meu trabalho de conclusão, desde o início na escolha do tema até aqui. Agradeço ao professor Gabriel Alves por tudo que foi me ensinado nesses longos árduos anos. Todas as reprovações/ aprovações foram motivos de reflexão, crescimento pessoal e profissional.

Agradeço a UFRPE pelos 7 anos de acolhimento e todo esforço máximo para dar o seu melhor. Muito obrigado a excelentíssima Carol Bezerra, por ser a melhor coordenadora que já tive na vida.

“...and remember one thing, the second day is always better than the first”
(Billie Joe Armstrong)

Resumo

Um aplicativo móvel (prova de conceito) foi desenvolvido fornecendo recomendações de rota para turistas que visitam Recife a pé. O turista seleciona os pontos de interesse (POI) que ele deseja visitar e a aplicação recomenda uma rota. O turista pode escolher os pontos de interesse de uma lista de pontos e visualiza-los em um mapa. A aplicação também fornece informações detalhadas sobre os pontos de interesse para auxiliar na escolha. Três algoritmos foram implementados para recomendação da rota. Os algoritmos Força Bruta, Vizinho Mais Próximo e Vizinho Mais Próximo combinado com 2-OPT foram comparados em termos de tempo de execução, impacto no tamanho total do percurso gerado, uso de memória e CPU. O algoritmo da Força Bruta apresentou um tempo de execução hábil em até 8 pontos escolhidos. O Vizinho Mais Próximo afastou-se cada vez mais do roteiro ótimo a medida que a quantidade de pontos aumentava, enquanto que sua combinação com o 2-OPT resultou em uma otimização de até 50 minutos na duração do roteiro.

Palavras-chave: Planejador de roteiros, Recomendação de Roteiros Turisticos, Problema do Caixeiro Viajante, Aplicativo Móvel no Dominio Turístico.

Abstract

A mobile application (proof of concept) has been developed providing route recommendations for tourists visiting Recife by walking. The tourist selects the points of interest (POI) he wants to visit and the application recommends a route. The tourist can choose the points of interest from a list of points and view them on a map. The application also provides detailed information on points of interest to aid in the choice. Three algorithms were implemented to recommend the route. The algorithms were compared in terms of execution time, impact on the total length of the route generated, memory and CPU usage. The Brute Force algorithm presented a skillful execution time in up to 8 chosen points. Nearest Neighbor moved away more and more of the optimal solution as the number of points increased,while its combination with 2-OPT resulted in an optimization of up to 50 minutes in the route duration.

Keywords: Route Planner, Recommendation of Touristic Routes, Travelling Salesman Problem, Mobile Application in The Tourist Domain

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

POI	Pontos de Interesse
PCV	Problema do Caixeiro Viajante
VMP	Vizinho Mais Próximo
FB	Força Bruta
SDK	Software Development Kit
API	Application Programming Interface

Sumário

1 Introdução

O Aeroporto Internacional de Recife/Gilberto Freyre atingiu a marca dos 7,77 milhões de passageiros em 2017, batendo a marca registrada em 2014 (ano da Copa do Mundo da FIFA que teve Recife como umas das cidades-sede). Esse dado é um indicador da quantidade de pessoas que visitam Recife e mostra que esse número vem crescendo. A Infraero também destacou que houve um aumento de 50% nos voos internacionais comparando os anos de 2016 e 2017.

Vários são os motivos para visitar Recife: clima ensolarado, culinária regional, arquitetura (influências portuguesas e holandesas), festas regionais (Carnaval e São João), música, centro histórico, artesanato, museus, praias entre outros. A prefeitura do Recife investe em projetos como o Olha! Recife[1], um programa que dá oportunidade às pessoas de conhecer mais sobre a cidade de forma gratuita. As pessoas participantes do projeto têm a chance de obter uma nova perspectiva sobre a cidade. O projeto contribui para atividade turística do Recife, como também para preservação do patrimônio histórico e cultural da cidade. O projeto consiste em ofertar vários passeios com guias turísticas divididos em roteiros. Os passeios podem ser a pé, de ônibus, de catamarã e de bicicleta.

O aplicativo desenvolvido nesta pesquisa foi inspirado no Olha! Recife e tem como objetivo auxiliar os turistas que não podem participar do projeto ou que preferem escolher os pontos que desejam visitar e procuram a ajuda de um aplicativo para escolher um roteiro otimizado.

1.1 Motivação

O planejamento de um roteiro turístico é uma tarefa desafiadora para as pessoas que visitam destinos urbanos desconhecidos. Em primeiro lugar, os turistas precisam restringir-se a um conjunto potencial de pontos de interesse(POI), entre os muitos disponíveis, alinhados com seus interesses pessoais e restrições de viagem. Dentre as dificuldades apresentadas, uma delas é como planejar uma rota para maximizar a experiência e satisfação do turista considerando o tempo.

Para fins desta pesquisa, o roteiro foi construído baseado em uma restrição: garantir que a rota comece de um POI e termine no mesmo POI. A problemática de otimização de rotas com esse tipo de limitação tem sido estudada na literatura por diversos autores, comumente relacionado ao Problema do Caixeiro Viajante (PCV). Esta pes-

[1] http://www.olharecife.com.br/

quisa é motivada pela aplicação do PCV no domínio turístico. Busca-se paralelamente uma mesclagem com recomendação de POIs. Entende-se que planejar e organizar um roteiro requer um conhecimento prévio dos locais, bem como adaptar o tempo do roteiro de acordo com a atratividade disponível do local. Por este motivo, a aplicação desenvolvida fornece informações como as comodidades do ambiente, tempo médio de permanência, como chegar, entre outras funcionalidades para auxiliar na escolha dos POIs. É importante também ressaltar que a redução no tempo total do percurso permite aos turistas ampliar a quantidade de POI visitados e aproveitar ao máximo o tempo disponível. Sendo assim, justifica-se a análise de métodos que pretendem minimizar o tempo total do roteiro, contemplando todos os POIs sem repeti-los.

1.2 Objetivos

1.2.1 Objetivo Geral

O trabalho tem como objetivo planejar uma rota de forma a minimizar o tempo total do roteiro. Além disso, mira-se a maximização da experiência e satisfação do turista, fornecendo informações para agregar na escolha das atrações turísticas.

1.2.2 Objetivo Específico

1. Obter informações sobre as atrações, identificando as comodidades, tempo de permanência e localização.
2. Coletar as distâncias entre cada POI.
3. Implementar os métodos necessários para criação do roteiro.
4. Testar cada método.
5. Comparar a eficiência dos métodos e suas peculiaridades.

1.3 Estrutura do Trabalho

Os demais capítulos deste trabalho estão divididos da seguinte maneira:

Capitulo 2: Apresenta toda fundamentação teórica por trás da minimização do roteiro, abordando os estudos do PCV tanto quanto os métodos para sua resolução.

Capitulo 3: Apresenta a metodologia do trabalho, a arquitetura e o fluxo do aplicativo móvel desenvolvido para aplicação dos métodos.

Capítulo 4: Faz uma avaliação comparativa dos algoritmos que montaram o roteiro e seu impacto no uso de memória e CPU.

Capitulo 5: Traz uma conclusão dos resultados obtidos, expõe os produtos gerados pela pesquisa e propostas para trabalhos futuros.

2 Referencial Teórico

Dada uma lista de cidades e os custos (distância, tempo, preço, etc.) entre cada par de cidades, qual é a rota mais curta possível que visita cada cidade exatamente uma vez e retorna ao ponto de origem? O PCV, do inglês *Travelling Salesman Problem* (TSP), procura resolver exatamente este questionamento. Para resolver esse problema, uma variedade de estudos tem sido amplamente abordado na esfera de problemas de otimização combinatória.

Pesquisadores de diferentes áreas como na Matemática, Pesquisa Operacional, Biologia e Inteligência Artificial, foram atraídos para resolver esse problema devido a aplicabilidade em diversos contextos do mundo real. Ainda sem o nome de Problema do Caixeiro Viajante, o problema foi primeiramente introduzido por volta dos anos 1930, com o Icosian game também chamado de jogo hamiltoniano (COXETER; BALL, 1960), que consiste em encontrar um caminho em um dodecaedro tal que todo vértice é visitado uma única vez e o ponto final é o mesmo que o inicial. A primeira vez que foi intitulado PCV foi por Hassler Whitney, em 1934, em um seminário na Princenton University (GOLDBARG, 2005). A partir daí, os trabalhos relacionados posteriores foram desenvolvidos com essa nomenclatura.

Durante o trajeto utilizando métodos exatos em busca de soluções ótimas, os testes de sucesso foram lembrados como instâncias. As instâncias continham as coordenadas e o método usado, o nome do autor do algoritmo ou região que estava propondo resolver, e a quantidade de cidades inseridas. No ano de 1954, destacaram-se os trabalhos de DANTZIG et al.,1954, publicando um artigo que propôs uma “Solução de larga escala do problema do caixeiro-viajante” no jornal da Sociedade de Pesquisa Operacional da América envolvendo 49 cidades. Nele foi proposta uma solução para o PCV utilizando-se métodos de programação linear, sendo esse um evento importante na história da otimização combinatória(ZAMBONI, 1997). Em 1954, heurísticas como algoritmo do Vizinho Mais Próximo e 2-OPT foram discutidas(FLOOD, 1956).HELD e KARP utilizaram uma abordagem de programação dinâmica para uma instância aleatória com 64 cidades.(GRÖTSCHEL, 1980) propôs um estudo computacional para 120 cidades através da programação linear. Um ano depois,(PADBERG; HONG, 1980) aplicaram o método de plano de corte para resolução de 318 cidades.

Seguindo nesta direção, os trabalhos foram evoluídos como mostra na tabela 1. As instâncias foram armazenadas na TSPLIB, biblioteca criada por (REINELT, 1991) contendo várias instâncias do PCV que foram testadas e discutidas na literatura. Importante ressaltar que estes trabalhos citados acima ganharam força ao longo de sua

história graças à grande evolução do poder computacional.

Ano	Instância	Tamanho	Primeiros Pesquisadores a Obter Solução Ótima
1954	dantzig42	42	Dantzig,Fulkerson e Johnson
1962	aleatória	64	Held e Karp
1974	aleatória	67	Camerine,Fratta e Maffioli
1980	gr120	120	Grötschel
1980	lin318	318	Crowder e Paderg
1987	att532	532	Paderg e Rinaldi
1991	gr666	666	Grötschel e Holland
1991	pr2392	2392	Paderg e Rinaldi
1995	pla7397	7397	Applegate,Bixby, Chvátal e Cook
1998	usa13509	13509	Applegate,Bixby, Chvátal e Cook
2001	d15112	15112	Applegate,Bixby, Chvátal e Cook
2004	sw24978	24978	Applegate,Bixby, Chvátal,Cook e Helsgaun

Tabela 1 – Soluções ótimas para o problema do caixeiro viajante (GOUVÊA et al., 2006)

Apesar do imenso esforço na literatura a respeito do PCV, ele está longe de ser resolvido de forma absoluta que satisfaça qualquer condição. Devido à sua complexidade computacional dada pela utilização de recursos como tempo e memória RAM, algoritmos até os dias atuais são escritos para apresentar uma solução factível, dado o critério específico do problema. Para chegar à solução ótima da instância sw24978, por exemplo, foram necessários 3 meses de processamento em um *cluster* com 10 supercomputadores.

2.1 Complexidade

A teoria da complexidade computacional (COOK, 1971), mostra a complexidade de problemas, classificando-os em duas classes genéricas conhecidas como P (*Polynomial time*) e NP (*Non-Deterministic Polynomial time*). A classe de complexidade P trata o conjunto de todos os problemas que aceitam o tempo polinomial no pior caso, representada pela função O (p (n)), onde p (n) é um polinômio. Por outro lado, a classe NP é formada por problemas onde as instâncias de médio e grande porte são intratáveis pelos algoritmos determinísticos conhecidos em tempo polinomial, devido ao consumo exagerado de recursos computacionais. Em 1972, Karp através de uma formulação matemática, mostrou que o PCV pertence à classe de problemas NP-Completos. Para cada N coleções de cidades, existem (N)! possibilidades de se planejar um roteiro (fixando a cidade inicial e permutando as restantes (N-1)!), causando uma explosão de possíveis combinações para cada N cidades adicionadas ao planejamento da rota, tornando-se inviável a resolução em tempo hábil até mesmo por supercomputadores.

Desta forma, podemos justificar a implementação de algoritmos heurísticos. Apesar das heurísticas não possuírem a garantia de encontrar a solução ótima, são capazes de encontrar soluções em um tempo apropriado para um problema em questão. Held e Karp apresentaram uma relaxação no modelo exato para o PCV, transformando-o num problema de programação linear cuja solução não representa uma solução viável, mas o valor da função objetivo reproduz um limite inferior para o valor ótimo do problema original. O problema linear obtido é polinomial e possível de ser resolvido. A partir de então, é comum os pesquisadores avaliarem se soluções encontradas em metodologias heurísticas estão próximas da solução ótima para grandes instâncias do PCV por meio desse limite inferior.

2.2 Formulação do Problema

Existem uma variedade de problemas relacionados a otimização combinatória que podem ser formulados como problemas em grafos. O PCV é um dos que fazem parte deste universo. Diversos autores mencionam alguns aspectos motivadores para o estudo do PCV, entre eles estão a facilidade com que o problema é descrito, a dificuldade em resolvê-lo por ser NP-Completo e sua vasta aplicabilidade (KARP, 1972).

Dado um grafo G = (N, E) onde N = {1, ..., *n*} é o conjunto de vértices e E = {1, ..., *m*} é o conjunto de arestas de G. O custo C_{ij} representa o custo associado a aresta que liga os vértices *i* e *j*. O problema consiste em determinar o menor ciclo hamiltoniano do grafo G, sendo que o tamanho do ciclo é dado pelo somatório dos custos das arestas que o compõem (GOUVÊA et al., 2006).

Transportando para realidade do turismo em recife, se usarmos os termos “pontos” ao invés de vértices e “tempo” relativo à distância entre dois pontos no lugar de aresta o problema poderia ser reescrito como: determinar o trajeto com menor tempo de uma sequência de visitas partindo de um determinado ponto, visitando todos os outros, uma única vez e, ao final, retornando ao ponto de partida.

Um grafo pode ser classificado como: simétrico (não direcionado) e assimétrico (direcionado). Será simétrico se o custo C_{ij} e C_{ji} forem iguais. Exemplo: ir caminhando da casa da cultura até o paço do frevo custa o mesmo tempo que sair do paço do frevo até a casa da cultura. Caso o tempo seja diferente, é dito assimétrico. Um grafo G = (N, E) é chamado de completo se para todo vértice $u, v \in N$ contém uma aresta conhecida uv. A figura 1 mostra um exemplo de grafo completo.

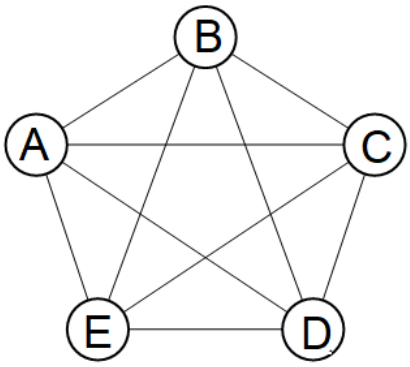

Figura 1 – Exemplo de um grafo completo
(REDHAT, 2018)

2.3 Metodologias para o PCV

Determinada a dimensão do PCV face à sua larga aplicabilidade em situações do dia a dia, várias abordagens têm sido desenvolvidas com o objetivo de resolver um conjunto cada vez maior de instâncias desse problema. As abordagens aplicadas ao PCV podem ser divididas em pelo menos duas principais condições: métodos exatos e métodos heurísticos. Nesta divisão, são apresentados alguns dos principais meios, tanto exatos como heurísticos.

2.3.1 Métodos Exatos

Um algoritmo é dito como exato quando consegue obter a solução ótima do problema e prova a otimalidade da solução obtida em um tempo de execução finito ou prova que a solução viável não existe (DUMITRESCU; STÜTZLE, 2003). Como exemplo, podem-se citar método da Força Bruta, *Branch and Bound,Branch and Cut* , Programação Inteira e Programação Dinâmica.

Dumitrescu e Stützle faz uma avaliação destacando as vantagens e desvantagens de algoritmos exatos na solução de problemas de otimização combinatória. A primeira vantagem deve-se a caso a execução do algoritmo tenha sucesso, é provado que não existe nenhum outro caminho menor que além do encontrado (solução ótima). Utilizando-se da Programação Inteira, através dos seus limites inferiores e superiores, ainda que o algoritmo não seja executado completamente, são adquirido informações capazes de criar abordagens aproximativas caso seja definido um critério de parada que anteceda o fim do algoritmo.

Em contrapartida, uma de suas desvantagens está no custo computacional muito alto. A medida que as possibilidades aumentam, o tempo para executar cresce fortemente. Devido a este comportamento, pode haver um consumo demasiado de memória causando o fim prematuro do programa.

2.3.1.1 Método da Força Bruta

O algoritmo de Força Bruta (FB) também conhecido como busca exaustiva é uma forma natural e intuitiva de se resolver o problema do caixeiro viajante. Nesta operação, é testada todas as possibilidades candidatas a solução do problema. Este método normalmente é implementado quando o número de instâncias é relativamente pequeno; alguma heurística foi aplicada antes diminuindo o tamanho do espaço de busca; o tempo de execução satisfaz aquele que está utilizando o programa, ou quando não vale a pena o esforço de implementar uma heurística vista que usando o algoritmo de FB já satisfaz as restrições em questão.

2.3.1.2 *Método Branch & Bound e Branch & Cut*

Proposto por (LAND; DOIG, 1960), o algoritmo *Branch and Bound* tem como fundamentação a ideia de dividir para conquistar. Atuando inicialmente com o problema maior, é dividido em vários subproblemas permitindo que sejam resolvidos, de maneira que este conjunto resulta em uma solução ótima para sua condição original. Um segundo pilar de sua abordagem é separar as possíveis soluções candidatas à solução ótima. Nesta direção, o método estaria equiparado à FB. Para escapar disso, o método utiliza-se de limites inferiores e superiores, descartando as soluções que estão fora dessa margem, reduzindo o espaço de busca. Os limites superiores podem ser encontrados através de alguma heurística eficiente e que possa ser executada em um curto período de tempo. Consequentemente, os valores obtidos acima deste limite são descartados pois já são conhecidos resultados melhores para a solução. Por outro lado, o limite inferior é dado pela relaxação do problema removendo uma ou mais restrições. Repetindo o processo de dividir o problema maior em subproblemas, espera-se terminar o algoritmo quando o limite inferior for igual ao superior ou quando o limite inferior for maior que a melhor solução encontrada durante o tempo de execução.

O algoritmo Branch and Cut é uma variação do método Branch and Bound, combinado com técnicas de plano de corte (*cutting plane*) destinado a melhorar a etapa de relaxamento estabelecido pelas soluções viáveis do problema. Devido ao grande número de restrições, nem sempre é possível resolver o problema aplicando programação linear. Segundo (GOUVÊA et al., 2006), se uma solução ótima associada à relaxação linear é inviável, um novo problema de separação deve ser tratado, buscando identificar uma ou mais restrições violadas pela relaxação corrente (*cutting procedure*). O problema no qual foi selecionado deverá ser novamente resolvido via programação linear até o momento em que não seja violada mais nenhuma restrição. Este método tem sido aplicado com sucesso em diversas classes de problemas de otimização combinatória, tais como: em PCVs (FISCHETTI et al., 1997), em problemas de Steiner(LUCENA; BEASLEY, 1998) e em problemas de particionamentos de grafos

(KARISCH; RENDL, 1998).

2.3.2 Métodos Heurísticos

Os métodos chamados heurísticos são técnicas no qual buscam resolver um determinado problema, sem necessariamente obter a solução ótima. É possível ainda encontrar heurísticas que não tem como saber o quão distante o seu resultado está da melhor solução. O que se espera, entretanto, é dado as suas restrições, encontrar um resultado que satisfaça no quesito de sua aplicabilidade. Tempo de execução e maleabilidade a respeito de quanto varia em relação à solução ótima são exemplos de fatores decisivos para escolha de uma determinada heurística.

2.3.2.1 Heurística do Vizinho Mais Próximo

Esta heurística funciona em sua construção adicionando ponto a ponto até formar o roteiro completo. A ideia consiste em definir um vértice de partida, e em seguida escolher o próximo vértice que possui o menor peso e também não tenha sido visitado previamente, ou seja, ainda não faz parte da solução final. Dessa maneira, este passo é executado repetidamente até que todos os vértices mais próximos (com pesos minimizados) sejam adicionados sem repetição. Por fim, é adicionado o ponto inicial para fechar o circuito Hamiltoniano. A tabela 2 mostra uma instância para n=5, onde n é a quantidade de vértices. Os valores preenchidos representam o tempo para se deslocar de um ponto turístico ao outro.

Matriz T	c_1	c_2	c_3	c_4	c_5
c_1	0	15	6	12	9
c_2	15	0	18	6	2
c_3	6	18	0	11	5
c_4	12	6	11	0	9
c_5	9	2	5	9	0

Tabela 2 – Heurística do VMP - Matriz pontos turísticos x tempo
Fonte: O Autor

Aplicando o método VMP para o PCV, utilizando a tabela 2, obtém-se uma rota R, com o percurso c1 - c3 - c5 - c2 - c4 - c1, totalizando um tempo (6+5+2+6+12 = 31). Abaixo, a figura 2 ilustra a inserção dos pontos até formar o caminho final, sendo c1 escolhido para ser o ponto de partida.

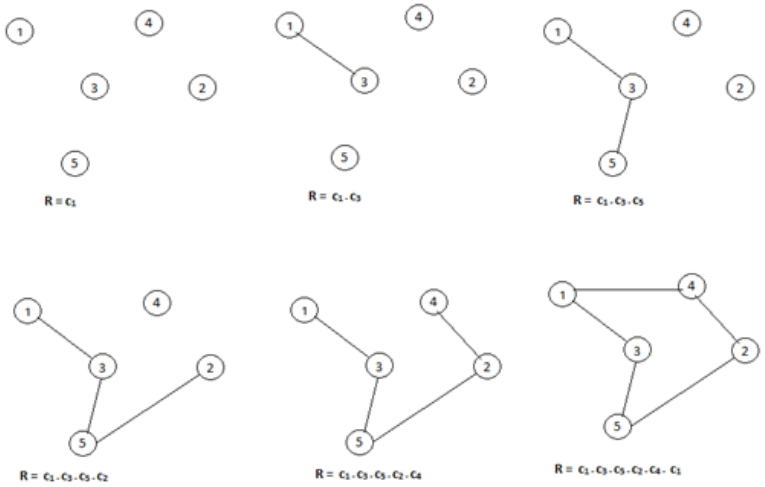

Figura 2 – Heurística do VMP - Passo a passo na inserção dos pontos
Fonte: O Autor

2.3.2.2 K-OPT

O método K-OPT é uma forma generalizada do algoritmo 2-OPT proposto por Croes (CROES, 1958) para resolver o PCV. Neste procedimento, é recebido uma solução inicial, e em seguida, uma tentativa de melhoramento é executado. Duas arestas são desconectadas formando dois sub percursos, e então religa-se as suas extremidades formando uma nova solução. Se esta for de melhor resultado em comparação a função objetivo, aceita-se o resultado e o processo é refeito para outras arestas na tentativa de conseguir um valor ainda mais satisfatório. Normalmente um critério de parada é definido, de modo que limite o número de iterações, mesmo que ainda seja possível de encontrar uma solução melhorada.

O nome 2-OPT é dado justamente pela troca de duas arestas, valores acima desta abordagem é considerado K-OPT.Caldas e Santos destacaram que é esperado ao crescer o valor de K, uma melhor solução seja encontrada, porém consumindo mais recursos computacionais. Valores de K maiores que 5 são impraticáveis na literatura, visto que o tempo para execução deste procedimento representa tempos exponenciais à medida que K e N crescem. Os valores mais frequentes encontrados na literatura para K são 2 e 3. Este método espera receber uma solução inicial, obtidas previamente por heurísticas como Busca Tabu, *Simulated Annealing*, Algoritmos genéticos, entre outras. A figura 3 mostra soluções geradas a partir do movimento 2-opt, sobre uma solução inicial.

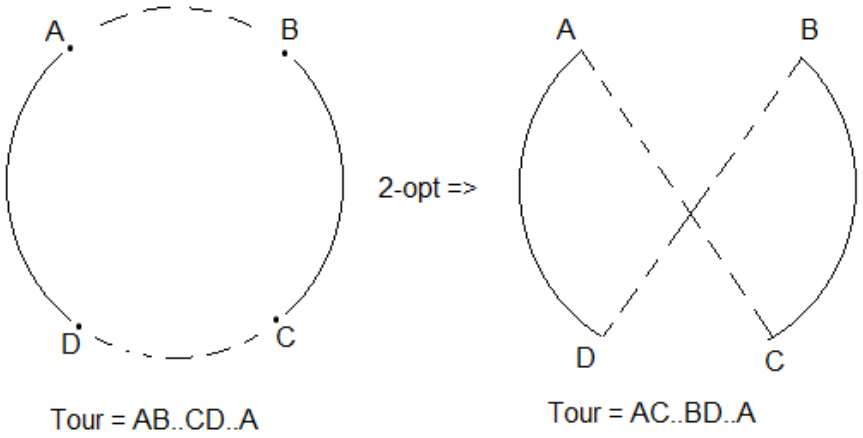

Figura 3 – K-OPT - movimento de troca
(ĐORđEVIĆ, 2008)

Como pode ser observado na figura 3, uma nova solução foi gerada, removendo as arestas (A, B) e (D, C) e adicionando as arestas (A, C) e (B, D). Caso esta satisfaça melhor o valor da função objetivo, no caso do PCV minimizar o peso total do roteiro, esta então passa ser a nova solução corrente para o problema.

2.3.3 Meta-Heurísticas

Ao longo de décadas, os estudos sobre heurísticas foram aprofundados, de modo que foi observado quais eram as características que levavam ao êxito dos métodos. Construindo estratégias genéricas com esqueletos de algoritmos, surgiu neste sentido a ideia de Meta-Heurísticas. Embora ainda não tenha um conceito comumente bem definido[1]:

> "Uma Meta-Heurística é um conjunto de conceitos que pode ser utilizado para definir métodos heurísticos aplicáveis a um extenso conjunto de diferentes problemas. Em outras palavras, uma meta-heurística pode ser visto como uma estrutura algorítmica geral que pode ser aplicada a diferentes problemas de otimização com relativamente poucas modificações que possam adaptá-la a um problema específico. Alguns exemplos de Meta-Heurísticas são: *Simulated Annealing*, *Busca Tabu*, *Iterated Local Search*, Algoritmos Evolutivos e *Ant Colony Optimization*."

Uma de suas principais características está na sua capacidade de explorar diferentes campos de soluções, de forma a evitar paradas em locais que a princípio foram

[1] http://www.metaheuristics.net

considerados como ótimos. Paradas prematuras são evitadas pelas Meta-Heurísticas através do processo chamado de fuga, operando escolhas aleatórias ou observando o histórico de soluções anteriores por meio da evolução do próprio método durante a sua execução. Esta prática atua em outros espaços de busca de modo a explorar possíveis soluções ótimas.

(VIANA, 1998) mencionou que as Meta-Heurísticas utilizam funções de probabilidade, onde não há garantia de obter o mesmo resultado para uma determinada instância do problema, diferenciando-se das heurísticas.(SUCUPIRA, 2004) relatou que dificilmente as Meta-Heurísticas alcançam o mesmo desempenho de métodos heurísticos especializados em um tipo de problema. A sua importância, contudo, está no fato delas descreverem métodos adaptáveis, proporcionando ideias que contribuam para problemas de otimização no qual não é conhecido algoritmos ou heurísticas eficientes. Ainda em sua citação, Sucupira expõe que o uso dessas abordagens comercialmente, entretanto, tem sido menos que o esperado pois a adaptação de uma Meta-Heurística para um tratamento eficaz e eficiente é uma tarefa custosa, dependendo de um forte conhecimento sobre o problema específico.

2.3.3.1 Busca Tabu

A busca tabu é uma Meta-Heurística proposta por Fred Glover para resolver problemas de otimização combinatória. É um método ajustável com a capacidade de fazer uso de outros métodos, tais como algoritmos de programação linear e heurísticas especializadas com o objetivo de evitar as limitações de ótimos locais (GLOVER, 1997).

O funcionamento da busca tabu concentra-se em receber uma solução e progredi-la ao longo de suas iterações, utilizando o conceito de movimento, vizinhança e memória até que uma condição de parada seja satisfeita. O processo de passagem da solução corrente para outra com melhor vizinhança ou dentre as melhores visitadas, é chamado de movimento. A cada iteração, é gerada uma vizinhança que é uma configuração modificada da solução inicial com intuito de melhorá-la. A memória possui um papel importante para a busca Tabu. Com o propósito de manter registrado o histórico de soluções anteriores, a busca “lembra” dos locais visitados, conferindo a oportunidade de explorar outros locais de busca e evitando criar soluções anteriormente visitado.

Tendo em vista a possibilidade de retornos a movimentos já visitados, uma estrutura chamada de lista tabu é inicializada. A lista tabu contém os movimentos reversos aos últimos movimentos realizados, assim como em uma fila. Ao adicionar um novo movimento à lista, é recebido um status de proibido em suas configurações, indicando não ser possível realizá-lo novamente. Sempre que isso acontece, o movimento mais antigo é retirado da lista tabu. Nesta lógica, a lista tabu reduz o risco de ciclagem, garan-

tindo o não retorno por N iterações a uma solução visitada anteriormente. Este modelo, todavia, abre margem para a proibição de movimentos para soluções que ainda não foram visitadas. Relaxando a rigidez da lista tabu, um mecanismo chamado função de aspiração foi desenvolvido para retirar o status de tabu relacionado a um movimento sob algumas circunstâncias. De modo geral, o critério adotado é quando a solução classificada na lista for melhor que a solução corrente.

Ao fim da busca Tabu, no que tange o critério de parada, é adotado na literatura uma de duas formas, sendo elas: quando é atingido um número máximo de iterações sem melhora no valor da melhor solução; ou o valor da melhor solução alcança um limite inferior conhecido ou próximo a ele. Este segundo critério tem a finalidade de evitar a execução desnecessária na ocasião em que uma solução ótima é encontrada ou a solução é julgada suficientemente satisfatória. Um pseudocódigo mostrando o funcionamento da busca tabu é apresentado na figura 4.

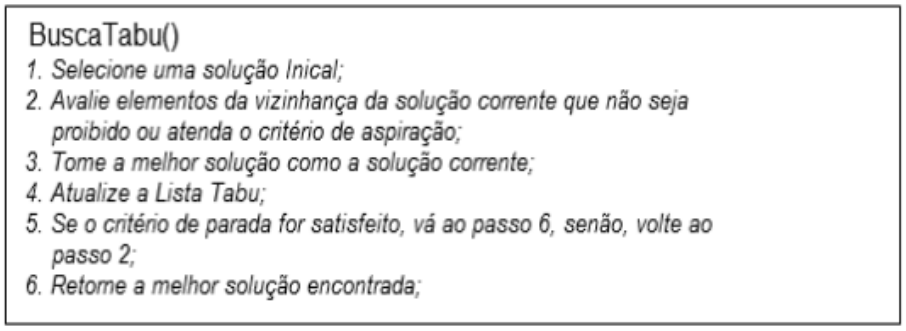
BuscaTabu()
1. *Selecione uma solução Inical;*
2. *Avalie elementos da vizinhança da solução corrente que não seja proibido ou atenda o critério de aspiração;*
3. *Tome a melhor solução como a solução corrente;*
4. *Atualize a Lista Tabu;*
5. *Se o critério de parada for satisfeito, vá ao passo 6, senão, volte ao passo 2;*
6. *Retorne a melhor solução encontrada;*

Figura 4 – Pseudocódigo da Meta-Heurística Busca Tabu
(KORZENOWSKI, 2018)

2.3.3.2 Colônia de Formigas

A Meta-Heurística colônia de formigas foi proposta por Marco Dorigo(DORIGO; BIRATTARI, 2011) com a proposta de simular o comportamento das formigas na natureza. Elas são capazes de encontrar o caminho mais curto entre seu ninho até a fonte de alimento. Ao sair em busca de alimento, as formigas liberam uma substância química chamada feromônio. Pelo fato da maioria delas serem cegas, elas são guiadas pelo cheiro da substância, escolhendo com maior probabilidade o caminho com o cheiro mais forte do componente químico, ou seja, percebendo maior quantidade.

Aplicado para o problema do caixeiro viajante, são criadas formigas artificiais usando heurísticas construtivas. Elas por sua vez, constroem soluções de forma probabilística através de uma trilha de feromônio artificial alterando-a dinamicamente durante a execução do programa com objetivo de refletir a experiência já adquirida durante a busca. (CARBAJAL et al., 2010) aplicaram esta Meta-Heurística no Desafio de Mona

Lisa. O Desafio de Mona Lisa foi apresentado por Robert Bosch em 2009 e refere-se a uma instância do PCV com n=100.000, cujo sua resolução procura reconstruir o quadro Mona Lisa de Leonardo da Vinci. A figura 5 apresenta um Pseudocódigo para a Meta-Heurística Colônia de Formigas.

```
1: inicialize
2: Execute /* neste nível cada loop é chamado de iteração
3:          Cada formiga é posicionada no nó inicial
4:          Execute /* neste nível cada loop é chamado passo
5:                  Cada formiga aplica a regra de transição de estado
para gerar a solução de forma incremental
6:                  Aplicação de regra de atualização local de feromônio
7:          Até que todas as formigas tenham contruído uma solução
8:          Aplicar regra de atualização global de feromônio
9: Até que condição de parada seja satisfeita
```

Figura 5 – Pseudocódigo da meta-heurística colônia de formigas (GAMBARDELLA; DORIGO, 1996)

No início, as formigas são colocadas em um ponto de partida diferente, ou posicionando-as aleatoriamente. Em seguida, move-se probabilisticamente para outro vizinho possível de ser visitado. A escolha do candidato é feita envolvendo a distância e a quantidade de feromônio envolvido. Durante a execução do algoritmo, dois procedimentos são executados: evaporação e depósito. A evaporação evita a acumulação indefinidamente do feromônio, permitindo esquecer decisões erradas do passado pela busca. O depósito é justamente a atualização do feromônio, fator indicativo para a formiga continuar a seguir por aquele caminho.

A ideia é que cada uma das formigas que inicialmente foram posicionadas em diferentes pontos, construa seu próprio caminho. Ao passo em que as iterações acontecem, a formiga que obteve um caminho eficiente, volta para o ponto inicial mais rapidamente influenciando a escolha das demais. Desta forma, as formigas não saem mais de maneira desordenada pelo fato da presença do feromônio liberado, convergindo para o melhor caminho ao longo do tempo. A meta heurística tem como condição de parada os seguintes fatores: o número máximo de iterações foi alcançado, ou na situação em que todas as formigas seguem o mesmo percurso. A ação cooperativa entre as formigas é o fator marcante desta Meta-Heurística, pois a qualidade da solução tende a aumentar de acordo com esse trabalho em conjunto para resolver o mesmo problema.

2.3.3.3 Algoritmos Genéticos

Estudados na década de 60, e posteriormente formalizado por Holland (HOLLAND, 1992), os algoritmos genéticos foram introduzidos. Inicialmente, seu principal

objetivo não era focado em otimização combinatória, mas sim estudar de maneira formal os fenômenos das espécies na natureza, como o de adaptação, seja eles naturais ou artificiais. Por conseguinte, havia um desejo de importar estes conceitos para o mundo computacional.

Os algoritmos genéticos foram inspirados na teoria Neo-Darwiniana. Três fenômenos naturais originaram esta teoria, sendo eles: a evolução das espécies abordada por Darvin, seleção natural dos indivíduos (Weismann) e a transmissão da informação genética estudada por Wendel. De acordo com a teoria Neo-Darwiniana, a vida é propagada através de 4 processos, sendo eles reprodução, mutação, competição e seleção (FOGEL; FOGEL, 1995) e são intrinsecamente paralelos. Na reprodução, envolve dois indivíduos dando origem a um novo, o qual herdará características genéticas de seus pais. A mutação é inserida neste contexto para diversificar a população (conjunto de soluções), ou seja, testar diferentes possibilidades de obter indivíduos. O processo é repetido a cada geração (iteração) produzindo um conjunto de população com indivíduos melhor adaptados ao longo do tempo.

No PCV, de forma análoga, o algoritmo genético gera inicialmente um conjunto de soluções candidatas aleatoriamente. Os indivíduos representados pelas cidades são avaliados, selecionados e recombinados para uma próxima geração. A aptidão dos indivíduos é chamada de *fitness* nos quais os mais promissores são escolhidos para fazer parte da nova geração. Os pesos (distância, tempo, etc.) são fatores que determinam o *fitness*. Durante o processo de seleção dos indivíduos, pode ser escolhido algum dos métodos tais como o método da Roleta, Torneio, Escalada Sigma entre outros. De acordo com Reyes (REYES et al., 2011), o torneio e a roleta são os métodos de seleção mais utilizados em AGs para o PCV. O método da roleta funciona dividindo em setores semelhante a uma roleta, distribuindo-a proporcionalmente em relação a sua aptidão ou *fitness*. Ao girar a roleta, um indivíduo é selecionado para participar da nova população. Já no Torneio, dois indivíduos são escolhidos aleatoriamente dentre um subconjunto da população, e posteriormente colocados para competir entre si e aquele que possuir melhor *fitness* é escolhido como um dos pais. Em seguida, é obtido o segundo pai da mesma maneira. O procedimento de mutação é realizado originando um indivíduo mais promissor para a próxima população.

2.3.3.4 Algoritmos Meméticos

Os algoritmos meméticos também conhecidos como algoritmo genético híbrido (MOSCATO et al., 2004) além de utilizar-se da evolução genética, trata o conceito de evolução cultural. (BURIOL et al., 2000) citou que diferentemente do algoritmo genético, a informação cultural não está associada ao processo de recombinação, mas sim pela comunicação entre indivíduos. Os genes como ocorre no processo evolutivo,

transmite sua informação através dos pais por gerações. Em contrapartida, os memes, unidade que replica a informação cultural também chamados de agentes, não possui ligação de hereditariedade, transmitindo a informação através de um indivíduo para um ou mais, podendo repassar até para uma população inteira. Por este motivo, a informação é transmitida de forma mais rápida e flexível que a genética (BURIOL et al., 2000). Alvejando transformar o algoritmo genético em memético, é introduzido um procedimento chamado de busca local. Este mecanismo tenta otimizar a solução inicial, aplicando a mutação entre os agentes, visando minimizar a função objetivo. Caso não consiga obter nenhuma melhora na aptidão dos indivíduos, a solução de entrada é retornada como saída dando continuidade aos outros procedimentos. Um pseudocódigo ilustrando a estrutura geral de um MA é a apresentado na figura 7.

```
Início
    InicializePopulação Pop usando InicialPop();          /*Geração de População Inicial */
    paraCada indivíduo i ∈ Pop faça i = OperadorDeBuscaLocal(i);   /* Otimização da população*/
    para Cada indivíduo i ∈ Pop faça Avalie(i);           /*Avaliação da população */
    repita
        Para j=1 até #recombinações faça                  /* fase de recombinação*/
            selecioneParaRecombinação um conjunto Spar ⊆ Pop;
            filho = Recombine( Spar,inst);
            filho = OperadorDeBuscaLocal(filho);
            Avalie(filho);
            adicionarNaPopulação indivíduo filho;
        fimPara;                                          /* fim da fase de recombinação*/
        Para j = 1 até #mutações faça
            selecioneParaMutar um indivíduo i ∈ Pop;
            m = Mutação (i);
            m=OperadorDeBuscaLocal(m);
            Avalie(m);
            adicionarNaPopulação indivíduo m;
        fimPara;                                          /* fim da fase de mutação*/
        Pop = SelecionaPop(Pop);                          /*seleção*/
        se convergênciaPop então Pop = RestartPop(Pop);   /* verifica diversidade */
    até criterioDeParada = true;
fim;
```

Figura 6 – Estrutura básica de um algoritmo memético
(BURIOL et al., 2000)

Destrinchando os procedimentos do algoritmo memético, a primeira função utilizada é **'inicializePopulacao'**. A população inicial dos agentes é escolhida através de alguma heurística construtiva, de forma aleatória ou ainda combinando ambos, visando uma boa solução logo ao seu início. O segundo procedimento **'operadorDeBuscaLocal'** tenta melhorar a solução inicial com o propósito de minimizar a função objetivo. Em seguida os agentes são escolhidos para sofrer o processo de recombinação, operado pelo procedimento **'selecioneParaRecombinação'**. Os métodos da Roleta ou torneio podem ser adotados para escolha neste momento. A função **recombine** fica responsável pelo cruzamento dos pais anteriormente selecionados. Adiante, o procedimento

‘adicionarNaPopulacao’ fica responsável por adicionar o indivíduo a população caso este melhore a função objetivo ou ainda não faça parte de uma população. Para cada geração, é selecionado para mutar um número de indivíduos expressos por #mutações e modificados por **Mutacao()**. Estes agentes são otimizados e possivelmente adicionados a população. Por fim, o procedimento **SelecionaPop()** vai escolher a próxima população. Um detalhe é que após a recombinação pode haver uma crise de diversidade, pelo fato da população possuir indivíduos muito similares. Diante deste cenário, o comando **‘convergenciaPop’** verifica a possível crise, e caso seja detectada o procedimento **RestartPop()** é acionado. Os indivíduos considerados similares são substituídos, preservando os mais bem avaliados. O processo continua otimizando e avaliando a cada geração. O critério de parada assim como no algoritmo genético pode ser a estagnação da população (não há melhora), ou o número de iterações chegou ao limite.

A seguir, é apresentado um comparativo das principais características dos algoritmos apresentados, finalizando o capítulo.

Algoritmo	Garante solução ótima	Custo computacional alto	Pode ocorrer interrupção prematura	Possui limites inferiores e superiores	Distancia-se da solução ótima ao adicionar novos vértices	Melhora a solução de outros algoritmos	Evita criar soluções já analisadas
Força Bruta	SIM	SIM	SIM	NÃO	NÃO	NÃO	SIM
Branch and Bound	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO
Branch and Cut	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO
Vizinho Mais Próximo	NÃO	NÃO	NÃO	NÃO	SIM	NÃO	SIM
2-OPT	NÃO	NÃO	NÃO	NÃO	N/A	SIM	SIM
Busca Tabu	NÃO	NÃO	NÃO	NÃO	SIM	NÃO	SIM
Colônia de Formigas	NÃO	NÃO	NÃO	NÃO	SIM	NÃO	SIM
Algoritmos Genéticos	NÃO	NÃO	NÃO	NÃO	SIM	NÃO	SIM
Algoritmos Meméticos	NÃO	NÃO	NÃO	NÃO	SIM	NÃO	SIM

Figura 7 – Resumo dos algoritmos e suas características.
Fonte : O Autor

3 Metodologia

Nesse capítulo é descrito o desenvolvimento de um aplicativo móvel (prova de conceito) que fornece recomendações de rota para turistas que visitam Recife a pé. O turista seleciona os pontos de interesses que ele deseja visitar e a aplicação recomenda uma rota. O turista pode escolher os pontos de interesse de uma lista de pontos e visualizá-los no mapa. A aplicação também fornece informações detalhadas sobre os pontos de interesse para auxiliar na escolha. Três algoritmos foram implementados para recomendação da rota: FB, VMP e 2-OPT. Uma análise mais profunda sobre o desempenho de cada um deles é apresentada no capítulo 4.

3.1 PCV na Cidade do Recife

A cidade do Recife possui um vasto acervo cultural onde o visitante pode fazer caminhadas pelo centro e desfrutar de sua beleza e história. Visando ampliar a experiência do turista neste cenário, o aplicativo ***Journey Planner*** (ou planejador de roteiro) foi implementado para melhor assistir os turistas que desejam visitar vários pontos de interesse e retornar ao seu início, semelhando-se ao PCV discorrido no capítulo anterior. Seja N = {1, ..., *n*} um conjunto de pontos da cidade do Recife e T = {1, ..., *m*} o tempo entre dois pontos, T_{ij} representando o custo simétrico (T_{ij}=T_{ji}) associado aos pontos *i* e *j*,T_{max} o tempo disponível do turista para realizar o passeio, o aplicativo recomenda uma rota aproximando-se da rota ótima, limitando-se à restrição de tempo concebido pelo turista.

3.2 Arquitetura da Aplicação

Nesta seção é discutido com mais detalhes o aplicativo. Uma visão geral da arquitetura da aplicação é ilustrada pela figura 8. Os conceitos foram divididos em 3 partes, sendo estes apresentados brevemente a seguir:

Repositório. O *Shared Preferences* é uma forma simples de armazenar os dados no próprio dispositivo móvel. A aplicação mantêm o histórico dos roteiros e os pontos favoritos escolhidos pelo usuário. As matrizes com o tempo entre os POI's são carregadas do repositório ao iniciar a aplicação.

Serviços. As API's do Google para Android foram utilizadas para obtenção dos dados necessários na aplicação. A API *Google Maps Distance Matrix* fornece informações como tempo e distância entre dois pontos no mapa. O aplicativo informa a

latitude e longitude de dois locais, como também o modo “a pé”, modo escolhido para percorrer o trajeto. A API retorna a distância relativa em km ou milhas e o tempo em segundos. Os pontos no mapa podem ser reconhecidos através de um ID. O Google fornece este ID através da API *Google Places* ou pelo *PlaceID Finder*. Esta identificação traz informações relevantes como o nome, endereço e classificação do local. Por fim, ainda foram utilizadas mais duas API’s, sendo elas: *Maps SDK Android* que provê a exibição dos pontos no mapa e *Google Directions*, responsável em obter o tracejado do roteiro.

Apresentação. Os dados fornecidos por todas as API’s e pelo repositório são processados e exibidos na tela do aplicativo.

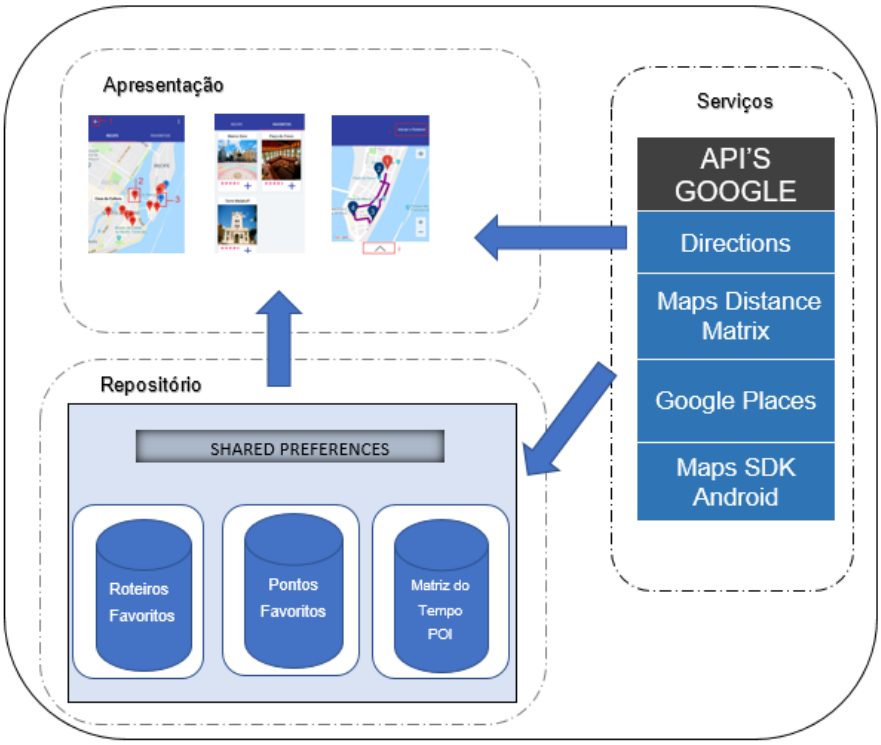

Figura 8 – Arquiterura da aplicação Journey Planner

3.3 Desenvolvimento

A aplicação *Journey Planner* foi desenvolvido na plataforma Android nativo, com ambiente de desenvolvimento no Android Studio versão 3.1.2 e linguagem de programação Java. A versão da API level compilada foi a versão 26.

O sistema conta com algumas funcionalidades, sendo elas:

- Ver pontos disponíveis
- Ver pontos recomendados
- Ver pontos mais populares

- Adicionar/remover pontos favoritos
- Visualizar pontos no mapa
- Adicionar restrição de tempo ao passeio
- Visualizar o roteiro prévio
- Criar roteiro
- Ver roteiros criados

A figura 9 mostra o fluxo das telas no aplicativo. Em seguida, é apresentado os detalhes sobre o funcionamento de cada uma delas.

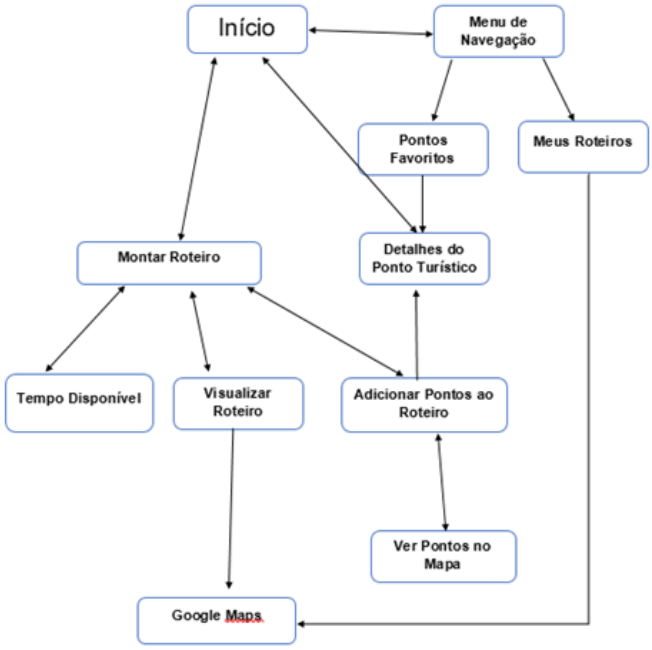

Figura 9 – Fluxo de telas - aplicação Journey Planner

3.3.1 Tela Inicial

1. O primeiro item da figura 10 representa um menu de navegação. Ao tocá-lo, o menu é expandido como mostra na figura 11.
2. Na tentativa de auxiliar o turista na escolha dos pontos turísticos, o aplicativo sugere os pontos mais populares do Recife para visita. Uma lista com rolagem horizontal é apresentada. O critério de utilizado foi a seleção dos pontos mais votados, dado fornecido pela API Google Places. Caso o usuário toque em algum item da lista, será direcionado para tela com os detalhes do ponto turístico, representada pela figura 12.

Figura 10 – Tela inicial do aplicativo

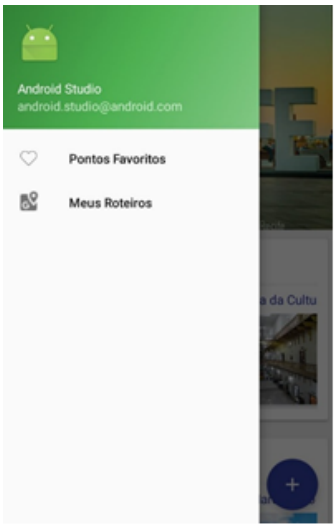

Figura 11 – Expansão do Menu de Navegação

3. Seguindo ainda o propósito de aproximar o turista na linha de recomendação, uma nova seção abaixo é mostrada com os pontos turísticos mais recomendados. O critério de utilizado foi a seleção dos pontos mais bem avaliados, dado fornecido pela API Google Places. Caso o usuário toque em algum item da lista, será direcionado para tela de detalhes do ponto turístico representada pela figura 12.

4. Um botão de mais (+) é mostrado. Ao tocá-lo, o aplicativo lançará para a tela representada pela figura 14, onde possibilita a criação de um roteiro.

3.3.2 Detalhes do Ponto Turístico

1. Na tela de detalhes do ponto turístico, o primeiro item abaixo da imagem do local e ao lado do nome, é representado por um coração. Diante de uma possível quantidade extensa de locais, para melhor encontrar o local posteriormente, uma função de favoritos foi implementada. Ao tocar no item, o símbolo do coração é preenchido e adicionado aos seus favoritos, como mostra a figura 13.
2. Buscando novamente auxiliar o usuário na escolha do ponto turístico, algumas comodidades do local foram desenvolvidas. O local pode ser:
   - Ao ar livre ou ambiente fechado
   - Possuir alimentação no local
   - Bom para levar criança ou não
3. O terceiro item mostra uma breve descrição, familiarizando o usuário com o local.
4. Entende-se que o tempo do turista é limitado. Almejando ajudá-lo a escolher locais que encaixe no seu tempo, o item 4 exibe o tempo máximo que as pessoas ficam naquele local.
5. O endereço referente ao local.
6. O Marcador do ponto turístico. Ao tocá-lo, o usuário é direcionado para o aplicativo Google Maps.

Figura 12 – Detalhes do ponto turístico

Figura 13 – Detalhes do ponto turístico, com o botão marcado como favoritos.

3.3.3 Criação do Roteiro

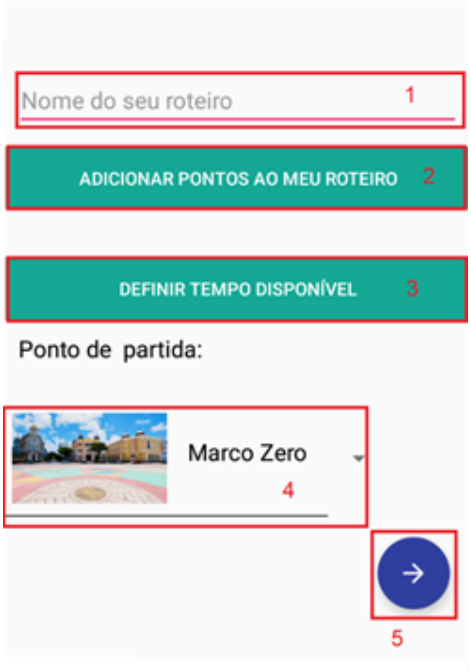

Figura 14 – Tela referente a construção do roteiro.

1. Nome do roteiro. Caso nenhum nome seja fornecido, é adicionado programaticamente a quantidade de roteiros previamente feitos mais um.
2. Botão para adicionar pontos ao roteiro. Ao tocá-lo, o aplicativo direciona para a tela com todos os pontos cadastrados, como mostra na figura 17.

3. Botão para definir o tempo que o turista tem disponível para realizar o roteiro. Ao tocá-lo, um componente do Android chamado TimePicker é expandido na tela, como mostra na figura 15.

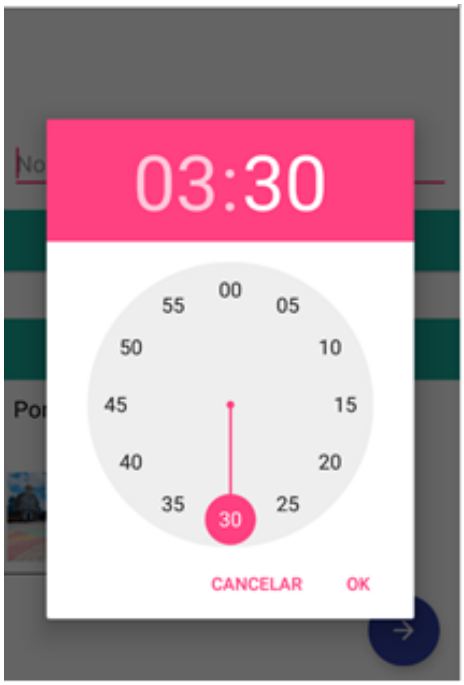

Figura 15 – TimePicker – escolher tempo disponível para o roteiro.

4. O turista tem a possibilidade de escolher o ponto de partida do roteiro. Um componente do Android chamado Spinner, podendo ser entendido como uma lista suspensa, é implementado para exibir a lista de pontos escolhidos pelo usuário. Caso não tenha adicionado nenhum ponto, nada será carregado na lista.

5. Botão de criar o roteiro. Ao toca-lo, O algoritmo FB é utilizado caso a quantidade de pontos escolhidos seja inferior a 8. Caso seja maior ou igual a 8, o algoritmo VMP é executado, e em seguida, com objetivo de melhorar ainda mais o caminho, é chamado o procedimento 2-OPT. Caso o tempo total do percurso ultrapasse o limite estabelecido pelo usuário, um alerta é exibido como mostra na figura 16.

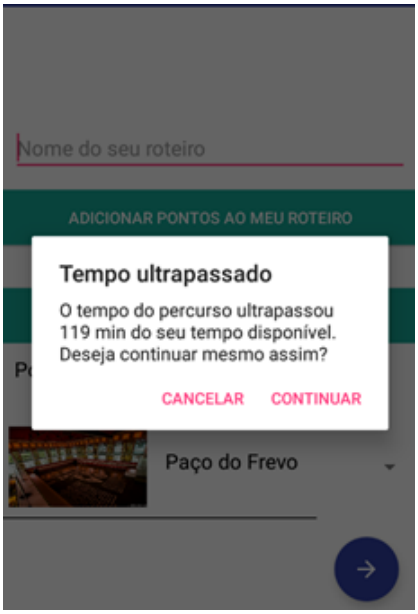

Figura 16 – Alerta de tempo ultrapassado.

Ao clicar em cancelar, abre a possibilidade para remover algum ponto turístico. Caso o turista opte por continuar o roteiro com o tempo excedido permanece e a aplicação direciona para a tela de pré-visualização do roteiro como mostra a figura 20.

3.3.4 Seleção de Pontos Turísticos

Uma lista de pontos da aba Recife é apresentada na figura 17. Caso o usuário toque no item da lista, como por exemplo na imagem, ele será direcionado para tela de detalhes do local.

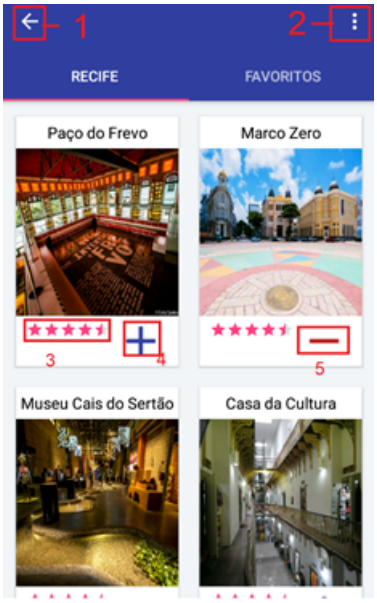

Figura 17 – Escolher pontos turísticos cadastrados em Recife.

1. Botão para voltar para a tela de construção do roteiro.
2. Menu. Ao tocá-lo, uma opção ‘ver no mapa’ é expandida. Caso o usuário toque neste item, a lista com os pontos turísticos é mostrada em forma de mapa, como mostra a figura 19.
3. Avaliação do local fornecida pelo Google Places API. A nota do local pode variar de 0 até 5 estrelas.
4. Botão de mais (+) para adicionar o local ao roteiro.
5. Botão de menos (-) para remover o local do roteiro.

Semelhante à figura 17, A aba de favoritos exibe os pontos que foram adicionados a favoritos.

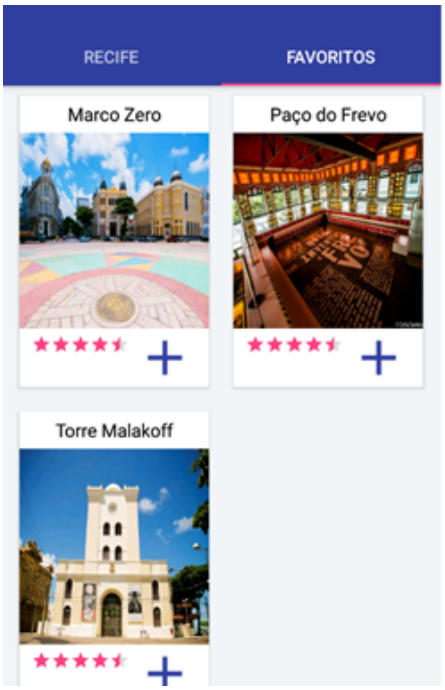

Figura 18 – Escolher Pontos Turísticos dos Favoritos.

3.3.5 Visualizar Lista de Pontos no Mapa

1. Voltar a visualizar em formato de lista.
2. Ponto turístico ainda não adicionado ao roteiro.
3. Ponto turístico selecionado para o roteiro.

Figura 19 – Mapa com lista de pontos turísticos em formato de geolocalização.

3.3.6 Pré-visualização do Roteiro

Um mapa é exibido com o trajeto dos pontos turísticos. Para apresentar obter o tracejado do caminho, foi utilizado a API do Google Directions.

Figura 20 – Tela de pré-visualização do roteiro.

1. Ao tocar em Iniciar o roteiro, o aplicativo finaliza seu ciclo lançando o usuário para o aplicativo Google Maps, com os pontos do roteiro definidos na ordem em

que foram estabelecidos. O *Journey Planner* salva o roteiro criado armazenando o histórico dos roteiros.

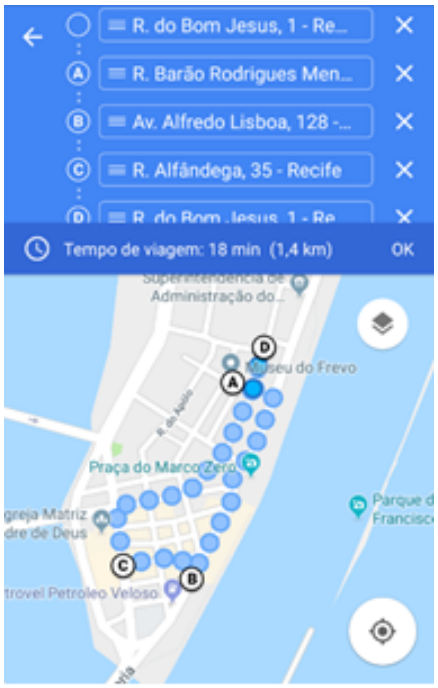

Figura 21 – Trajeto fornecido pelo Journey Planner no Google Maps.

2. Componente BottomSheet do Android, funcionando com uma gaveta. Ao deslizar para cima, é exibido o roteiro em detalhes.

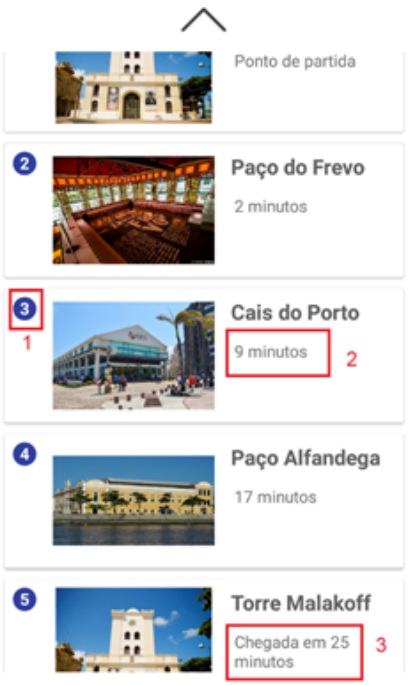

Figura 22 – BottomSheet expandida.

Na figura 22, é enumerado os pontos começando pelo ponto de partida até o de chegada. A cada ponto seguinte, é destacado quanto tempo já foi percorrido em relação ao início.

3.3.7 Visualizar Histórico de Roteiros e Pontos Favoritos

Após adicionar um ponto a favorito, ele pode ser encontrado no menu de navegação, figura 11. Ao expandir o menu e tocar em ‘Pontos Favoritos, o usuário é levado para Tela de Pontos Favoritos. A figura 23 ilustra os pontos, observando a possibilidade de remoção ao tocar no item “Remover” do menu suspenso.

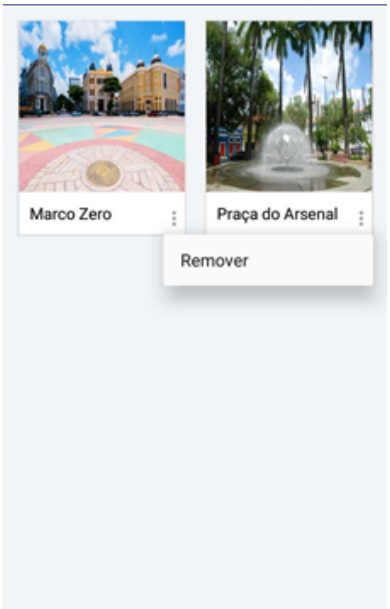

Figura 23 – Tela de pontos favoritos.

Após criar um roteiro e inicia-lo, o aplicativo guarda o roteiro, que poderá ser futuramente acessado em ‘Meus Roteiros’ – Figura 11.

A figura 24 exibe uma lista com os roteiros anteriores. Ao tocar no item da lista, a aplicação direciona o turista para o Google Maps, da mesma forma ilustrado pela figura 20.

Figura 24 – Tela do histórico de roteiros

4 Avaliação dos Algoritmos

Nesta seção serão mostrados os testes realizados com os algoritmos FB, VMP e VMP combinado à 2-OPT. Para cada seção de teste foram realizados 30 experimentos como recomenda o teste t de distribuição(LOVELAND, 2011) e obtido uma média dos valores. Os testes têm como principal objetivo mostrar o tempo de execução dos algoritmos, o impacto no tamanho total do roteiro, o gasto de CPU e de memória.

Para tal avaliação, foi utilizado um celular Motorola G4 Play com 1.2 GHz Quad Core de processamento, 2 GB de memória RAM e Chipset Cortex-A53 Qualcomm Snapdragon 410 MSM8916. As informações referentes à processamento e memória foram obtidas através da ferramenta de análise Android Profiler, disponível no Android Studio versão 3.0. O tempo de execução dos algoritmos foi obtido através da função *System.currentTimeMillis()* fornecida pelo código nativo.

4.1 Tempo de execução

O algoritmo FB foi testado com roteiros de 4 até 11 pontos turísticos. A figura 25 mostra o tempo médio de execução em segundos para cada quantidade de pontos escolhidos. A escolha dos pontos foi feita de forma aleatória.

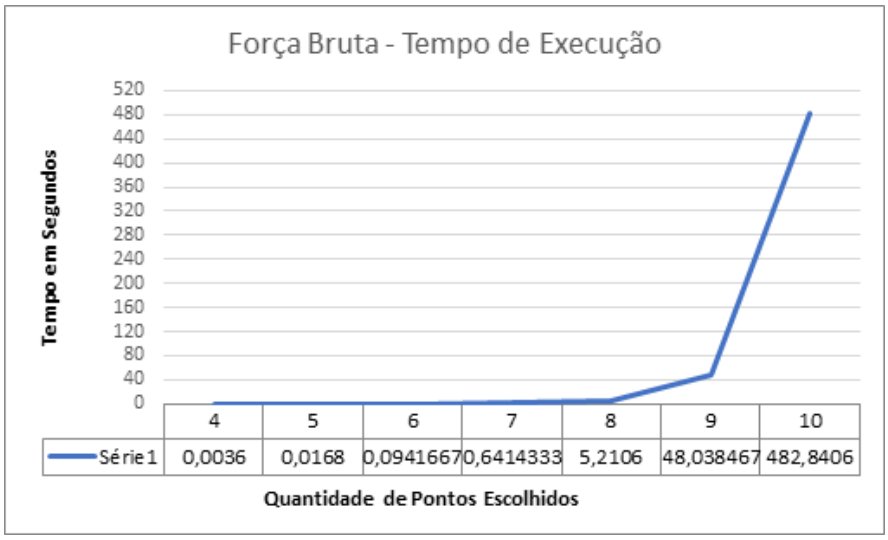

Figura 25 – FB – Tempo de execução

O gráfico aponta um aumento expressivo no tempo a partir de **8** pontos, com cerca de **5,21** segundos até **8,047** minutos para executar. Um teste com 11 pontos foi realizado, porém o limite de memória reservado é estourado levantando o erro ***OutOfMemoryError*** abortando o procedimento.

Uma seção de teste foi feita para efeitos comparativos de tempo de execução utilizando o algoritmo do VMP, e VMP combinado ao 2-OPT. O gráfico apresentado na figura 26 expressa um aumento no tempo de execução em relação ao tamanho do roteiro, porém não ultrapassando dos **0,202** segundos.

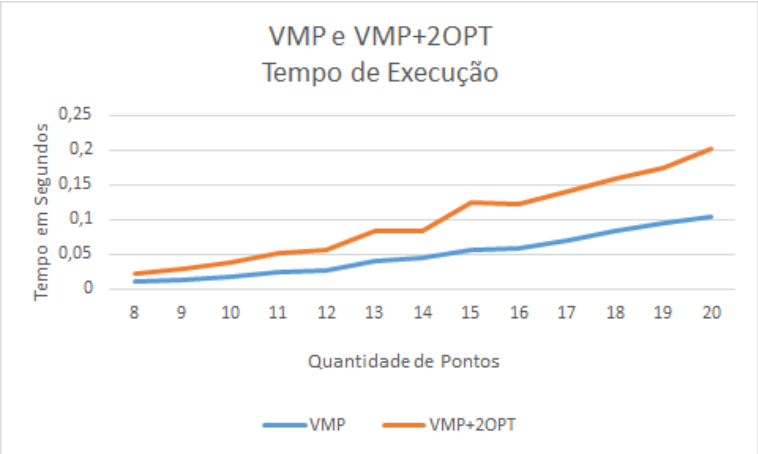

Figura 26 – Comparativo VMP + 2-OPT em tempo de execução

4.2 Tamanho do Percurso

Como foi discutido na revisão da literatura, o algoritmo FB no PCV testa todos os possíveis roteiros e apresenta aquele com menor tamanho. Servindo-se deste parâmetro, a figura 27 apresenta a porcentagem do quão maior foi o roteiro feito pela heurística VMP comparado a FB. É possível notar uma tendência do algoritmo do VMP em distanciar-se do roteiro ótimo ao aumentar a quantidade de pontos.

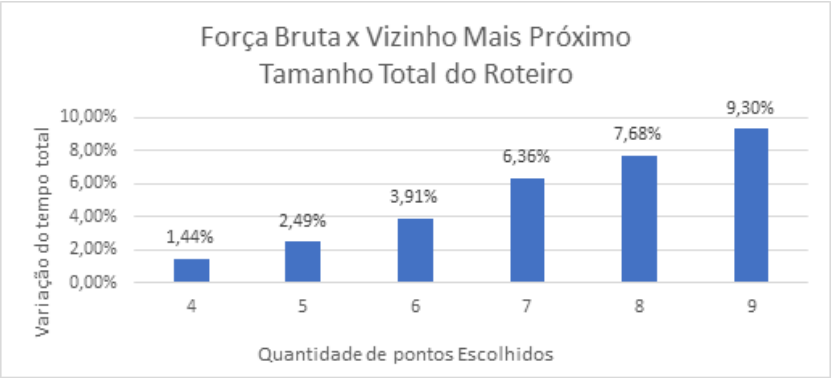

Figura 27 – FB X VMP – Comparativo na variação do percurso total gerado pelos algoritmos.

Observando a crescente diferença entre o tamanho do roteiro dado pelo algoritmo FB e VMP, foi implementado e testado o procedimento 2-OPT. Primeiramente é executado o algoritmo VMP, em seguida o roteiro é dado como entrada para o 2-OPT

na tentativa de melhorá-lo. O gráfico da figura 28 mostra o ganho em média obtido em porcentagem para determinada quantidade de pontos turísticos escolhidos.

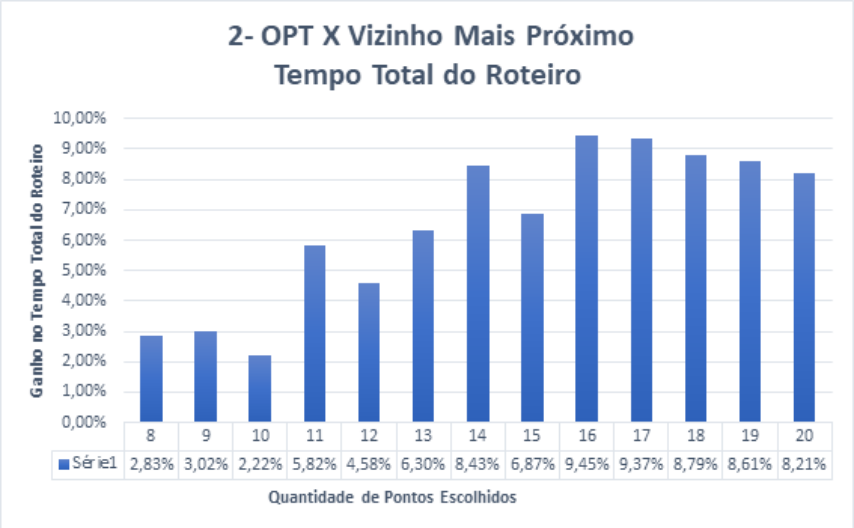

Figura 28 – 2 OPT x VMP - Ganho no tempo total do roteiro.

QP	AM	% MÁXIMA
8	21,65	13,98
9	35,83	17,98
10	22,27	12,00
11	38,73	18,32
12	38,7	17,75
13	39,35	19,05
14	39,47	18,75
15	44,7	18,81
16	42,3	18,51
17	50,4	20,73
18	37,48	15,96
19	41,97	18,98
20	37,97	15,84

Tabela 3 – Comparativo no ganho dos algoritmos 2-OPT x VMP.

A tabela 3 mostra o ganho obtido pelo algoritmo 2-OPT em relação ao VMP, onde QP representa a quantidade de pontos e AM o aproveitamento máximo em minutos. Por exemplo, para 8 pontos escolhidos, dentre os 30 testes realizados com esta quantidade, existiu ao menos um roteiro que foi otimizado em **21,65** minutos, bem como **13,98**% foi a porcentagem que representou o melhor aproveitamento. Destaca-se na tabela 3 o teste realizado com 17 pontos, onde a rota foi melhorada em **50,4** minutos chegando a diminuir **20,74**% do tamanho total do roteiro gerado pelo VMP.

4.3 Avaliação de CPU e Memória

A avaliação de CPU e memória foi realizada mostrando o pico máximo quando os algoritmos FB e VMP combinado à 2-OPT é executado. A figura 29 mostra o algoritmo FB com 11 pontos escolhidos. Nota-se um constante uso de processamento durante sua execução, fazendo com que o aplicativo Journey Planner tomasse até **32,7**% do processamento total do celular. O consumo de memória atingiu o seu máximo de **222 MB**, até despencar com o travamento da aplicação.

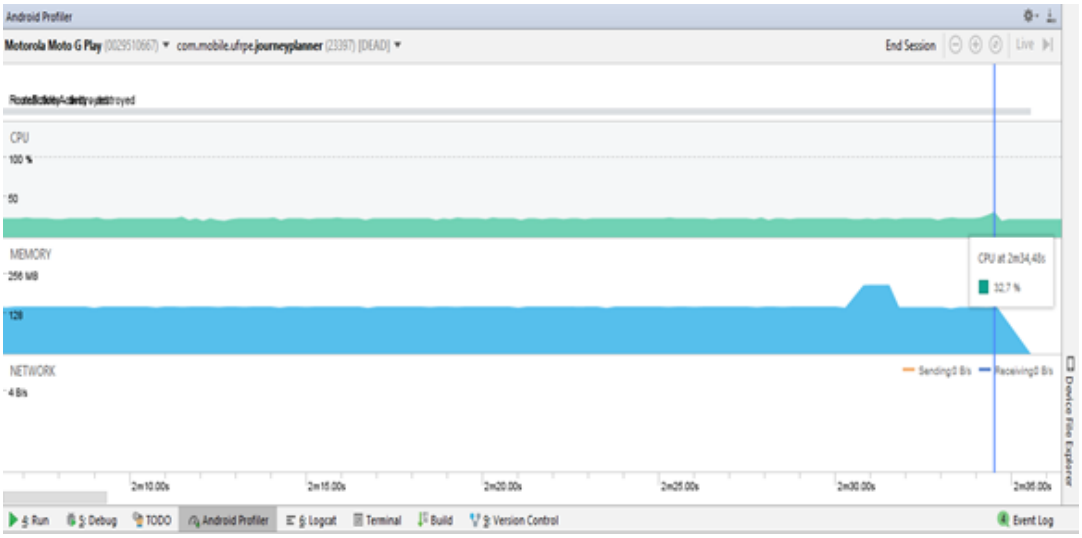

Figura 29 – Algoritmo FB com 11 Pontos – Pico máximo de CPU.

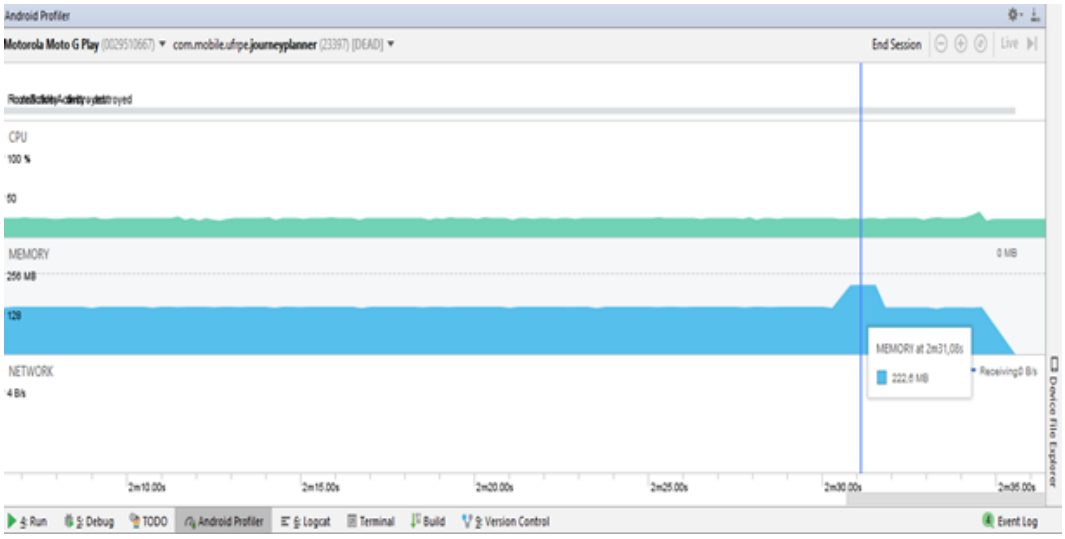

Figura 30 – Algoritmo FB com 11 Pontos – Pico máximo de memória.

Ademais, o algoritmo da FB foi testado com 7 pontos, pois foi observado em testes anteriores um tempo de execução menor que um segundo, tempo tolerável para o uso no aplicativo. A figura 31 manifesta uma oscilação de **1,3MB** na memória e uma

máxima de **73,6 MB**. Constata-se também um uso de CPU em um tempo mais curto, sendo necessário apenas de **44,39**s até **46,34**s. O máximo de processamento consumido pelo aplicativo neste intervalo foi de **26,9**%.

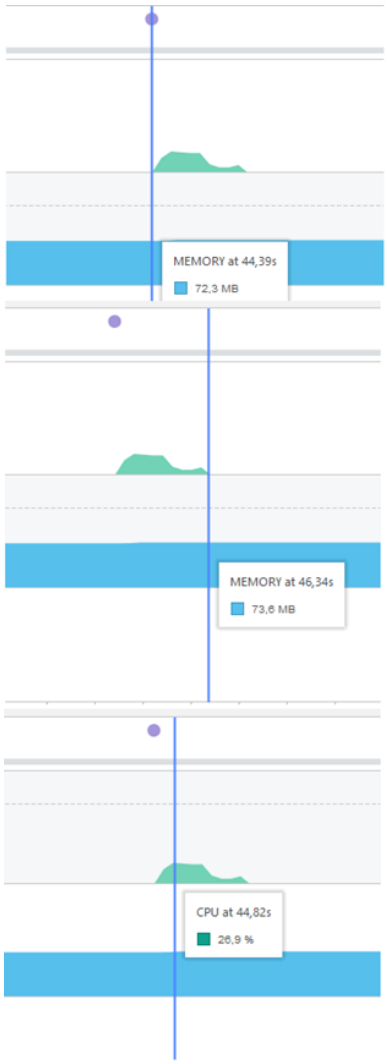

Figura 31 – Algoritmo da força bruta com 7 pontos – Uso da memória e CPU.

O algoritmo 2-OPT foi testado recebendo como entrada um roteiro traçado pelo VMP com 20 pontos. A figura 32 revela o consumo de memória antes de executar o procedimento foi de **87,8MB** com seu termino em **89,4MB**, variando **1,6MB**.

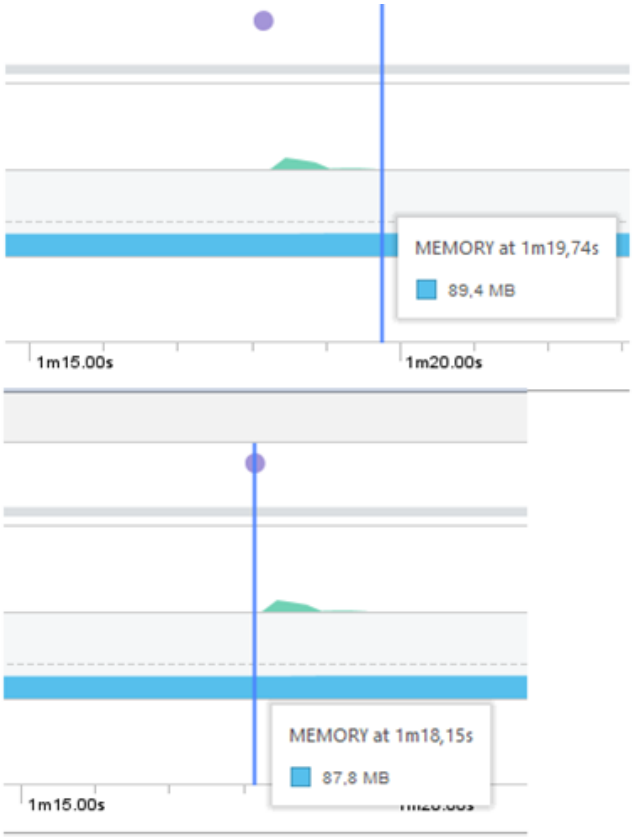

Figura 32 – VMP + 2 OPT - Variação da memória com 20 pontos.

5 Conclusão

O objetivo desse trabalho é auxiliar os turistas a planejar rotas de forma a minimizar o tempo do roteiro bem como auxiliá-los na escolha dos pontos turísticos. O problema da roteirização foi fundamentado no PCV e testados 3 métodos para sua resolução.

Os algoritmos FB, VMP+ VMP+2-OPT mostraram-se dentro do esperado em seu comportamento. Os resultados apresentados na sessão anterior mostraram que o algoritmo FB tem uma tendência de crescer o tempo de execução de forma explosiva conforme a quantidade de pontos aumenta, sendo viável para o turista escolher até 8 pontos, considerando 5 segundos um tempo de espera razoável. O teste feito com 11 pontos mostrou-se impraticável visto que há um consumo muito elevado de memória e CPU, interrompendo a aplicação.

Em contraste com o algoritmo FB, o VMP desempenhou rapidamente o seu papel, porém a cada ponto adicionado o roteiro ficou mais distante do ótimo. Entretanto, é plausível destacar que este comportamento é natural de métodos heurísticos, os quais intentam resolver em tempo ágil, mas sem garantias de melhor roteiro. O custo com memória e CPU foi inferior comparado ao FB.

Por fim, o algoritmo 2-OPT revelou um ganho de até 50,4 minutos e conseguiu melhorar em até 20% a rota gerada pelo VMP. Os testes de CPU e memória foram efetuados em cima de 20 pontos turísticos, o máximo cadastrado no aplicativo. Houve um aumento de memória em 1.6 MB após o toque de gerar roteiro. Mesmo assim, o processo perdurou pouco mais de 1 segundo sendo considerado o mais rápido dentre os executados.

5.1 Artefatos Gerados

Para uma melhor análise e entendimento do aplicativo Journey Planner, foi criado um vídeo mostrando seu funcionamento suas funcionalidades. O aplicativo está disponível na PlayStore e pode ser visualizado através desse link.

5.2 Dificuldades Encontradas

Algumas dificuldades foram encontradas neste presente trabalho, sendo elas:

• Planejar o roteiro com o tempo entre dois pontos sendo assimétrico. Por este motivo, uma abordagem mais controlada foi escolhida com tempo simétricos.

• Planejar o roteiro com o modo de passeio “carro”. O aplicativo recomendaria uma rota, porém ao iniciá-la as condições de transito afetariam o tempo total do percurso.

• Apresentar a pré-visualização dos roteiros. A aplicação requisita o caminho total para desenhar o percurso no mapa, porém o comportamento assíncrono faz com que muitas vezes o aplicativo tente desenhar antes da resposta ser recebida, falhando ao exibir um trecho do caminho.

• A ferramenta de análise Android Profiler não informa exatamente o quanto de CPU e memória foi gasto com determinado método, sendo possível apenas observar um comportamento mais generalizado.

5.3 Trabalhos Futuros

Para trabalhos futuros, tem-se alguns desejos como:

- Explorar algoritmos de recomendação.
- Melhorar a experiência do usuário focando menos no caminho ótimo e mais nos locais de visita.
- Maximizar a quantidade de locais cadastrados e permitir que os usuários possam escolher o modo de passeio desejado.
- Programar uma sequência de roteiros personalizados em dias diferentes
- Considerar os dias e horários de abertura dos pontos turísticos.

Referências

BURIOL, L. S. et al. Algoritmo memetico para o problema do caixeiro viajante assimetrico como parte de um framework para algoritmos evolutivos. [sn], 2000. Citado 2 vezes nas páginas 26 e 27.

CARBAJAL, S.; CORNE, D.; REID, F. Solving monalisa tsp challenge with parallel ant colony optimization. 2010. Citado na página 24.

COOK, S. A. The complexity of theorem-proving procedures. In: ACM. *Proceedings of the third annual ACM symposium on Theory of computing*. [S.l.], 1971. p. 151–158. Citado na página 16.

COXETER, H. S. M.; BALL, W. W. R. *Mathematical Recreations Essays*. [S.l.]: Macmillan Reference, 1960. Citado na página 15.

CROES, G. A. A method for solving traveling-salesman problems. *Operations research*, INFORMS, v. 6, n. 6, p. 791–812, 1958. Citado na página 21.

DORIGO, M.; BIRATTARI, M. Ant colony optimization. In: *Encyclopedia of machine learning*. [S.l.]: Springer, 2011. p. 36–39. Citado na página 24.

DUMITRESCU, I.; STÜTZLE, T. *A survey of methods that combine local search and exact algorithms*. [S.l.], 2003. Citado na página 18.

FISCHETTI, M.; GONZÁLEZ, J. J. S.; TOTH, P. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, INFORMS, v. 45, n. 3, p. 378–394, 1997. Citado na página 19.

FLOOD, M. M. The traveling-salesman problem. *Operations Research*, INFORMS, v. 4, n. 1, p. 61–75, 1956. Citado na página 15.

FOGEL, D. B.; FOGEL, L. J. An introduction to evolutionary programming. In: SPRINGER. *European Conference on Artificial Evolution*. [S.l.], 1995. p. 21–33. Citado na página 26.

GAMBARDELLA, L. M.; DORIGO, M. Solving symmetric and asymmetric tsps by ant colonies. In: IEEE. *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. [S.l.], 1996. p. 622–627. Citado na página 25.

GLOVER, F. Tabu search and adaptive memory programming—advances, applications and challenges. In: *Interfaces in computer science and operations research*. [S.l.]: Springer, 1997. p. 1–75. Citado na página 23.

GOLDBARG, C. Luna; hpc otimização combinatória e programação linear. *Rio de Janeiro: Editora Campus*, 2005. Citado na página 15.

GOUVÊA, E. F.; RAMOS, I. C. d. O.; GOLDBARG, M. C.; OLIVEIRA, C. S. et al. *Uma análise experimental de abordagens heurísticas aplicadas ao problema do caixeiro viajante*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2006. Citado 3 vezes nas páginas 16, 17 e 19.

GRÖTSCHEL, M. On the symmetric travelling salesman problem: solution of a 120-city problem. In: *Combinatorial Optimization*. [S.l.]: Springer, 1980. p. 61–77. Citado na página 15.

HOLLAND, J. H. Genetic algorithms. *Scientific american*, JSTOR, v. 267, n. 1, p. 66–73, 1992. Citado na página 25.

KARISCH, S. E.; RENDL, F. Semidefinite programming and graph equipartition. *Topics in Semidefinite and Interior-Point Methods*, v. 18, p. 77–95, 1998. Citado na página 20.

KARP, R. M. Reducibility among combinatorial problems. In: *Complexity of computer computations*. [S.l.]: Springer, 1972. p. 85–103. Citado na página 17.

KORZENOWSKI, A. *Pseudo código da Busca Tabu*. 2018. <http://www.revistaespacios.com/a15v36n21/15362108.html>. Accesso: 20-07-2018. Citado na página 24.

LAND, A. H.; DOIG, A. G. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, JSTOR, p. 497–520, 1960. Citado na página 19.

LOVELAND, J. L. Mathematical justifcation of introductory hypothesis tests and development of reference materials. 2011. Citado na página 42.

LUCENA, A.; BEASLEY, J. A branch and cut algorithm for the steiner problem in graphs. *Networks: An International Journal*, Wiley Online Library, v. 31, n. 1, p. 39–59, 1998. Citado na página 19.

MOSCATO, P.; COTTA, C.; MENDES, A. Memetic algorithms. In: *New optimization techniques in engineering*. [S.l.]: Springer, 2004. p. 53–85. Citado na página 26.

ĐORđEVIĆ, M. *Influence of grafting a hybrid searcher into the Evolutionary Algorithm*. [S.l.: s.n.], 2008. Citado na página 22.

PADBERG, M. W.; HONG, S. On the symmetric travelling salesman problem: a computational study. In: *Combinatorial Optimization*. [S.l.]: Springer, 1980. p. 78–107. Citado na página 15.

REDHAT. *Complete Graph*. 2018. <https://access.redhat.com/webassets/avalon/d/Red_Hat_JBoss_A-MQ-6.1-Using_Networks_of_Brokers-en-US/images/9a8669a7f1123c990dd5403d4bc729bd/broker_networks_12.gif>. Accesso: 20-07-2018. Citado na página 18.

REINELT, G. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, INFORMS, v. 3, n. 4, p. 376–384, 1991. Citado na página 15.

REYES, F.; CERPA, N.; CANDIA-VÉJAR, A.; BARDEEN, M. The optimization of success probability for software projects using genetic algorithms. *Journal of Systems and Software*, Elsevier, v. 84, n. 5, p. 775–785, 2011. Citado na página 26.

SUCUPIRA, I. R. Métodos heurísticos genéricos: metaheurísticas e hiper-heurísticas. *USP: São Paulo*, p. 32, 2004. Citado na página 23.

VIANA, G. V. R. *Meta-heurísticas e programação paralela em otimização combinatória*. [S.l.]: UFC, 1998. Citado na página 23.

ZAMBONI, L. d. S. *Técnicas de roteirização de veículos aplicadas ao transporte escolar*. Tese (Doutorado) — Dissertação de Mestrado, UFPR, 1997. Citado na página 15.