Algoritmo Genético para o Orienteering Problem

Hugo Araújo de Sousa
Robótica Móvel
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
hugosousa@dcc.ufmg.br

1 INTRODUÇÃO

Os avanços em métodos computacionais e na tecnologia do *hardware* disponível têm permitido que a área da Robótica Móvel avance cada vez mais. Dentro da subárea de planejamento de caminhos para robôs (e times de robôs), por exemplo, muitos algoritmos já foram propostos, como os algoritmos Bug [15], Campos Potenciais [21], *Rapidly-exploring Random Tree* [13], entre outros. Todos esses algoritmos apresentam bons resultados, dependendo do cenário e do problema, de forma que a escolha do algoritmo deve levar em consideração o conhecimento sobre o ambiente e as necessidades por eficiência (tanto em tempo de execução quanto em qualidade dos caminhos encontrados) e eficácia.

Entretanto, existem muitos problemas para os quais ainda não existem algoritmos que sejam ótimos e razoavelmente eficientes (em termos de tempo de execução) simultaneamente. Essa observação pode ser vista como resultado direto da possibilidade de se modelar muitos desses problemas como problemas conhecidamente NP-Completos [12]. É esse o caso do *Orienteering Problem* (OP) [9], um problema de otimização onde o objetivo é coletar o maior número de recompensas em um ambiente, dado que o caminho do robô deve respeitar algum limite de orçamento. Tanto as recompensas no ambiente quanto o orçamento do robô são abstrações para, em aplicações reais, entidades e medidas mais concretas, como algum tipo de coleta de material, para a primeira, e a bateria do robô, para a segunda. A Figura 1 ilustra o problema.

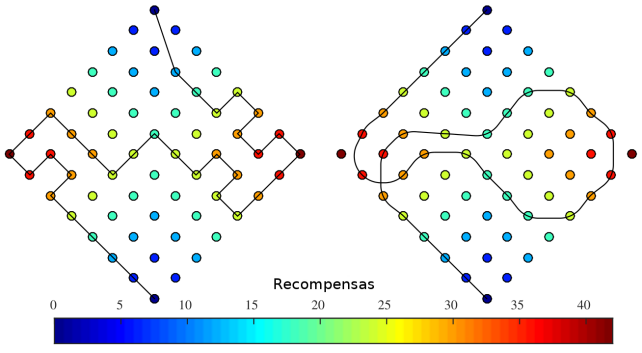

Figura 1: Exemplo de duas instâncias do *Orienteering Problem* [16]. As instâncias são representadas por uma grade de pontos, os chamados *sampling points*, cuja cor representa a recompensa coletada ao visitar o ponto correspondente. O ponto mais acima nos mapas representa o ponto de início e o ponto mais abaixo representa o ponto de término do caminho do robô.

Podemos definir o OP mais formalmente da seguinte forma:

Orienteering Problem: *Dado um conjunto de pontos em um ambiente, a recompensa $r(p) \geq 0$ associada a cada um desses pontos, um ponto de início p_i, um ponto de término p_f e um custo máximo $custo_{max}$, deseja-se encontrar um caminho c que começa em p_i e termina em p_f tal que seu custo $custo_c \leq custo_{max}$ e a soma das recompensas individuais de cada ponto de c seja a maior possível.*

Muitas aplicações reais em Robótica Móvel têm sido reportadas na literatura como instâncias do OP, tais como monitoramento de floração de algas, derramamento de óleo e poluição química [19], além da já citada exploração de emergência sob restrição de bateria. Diante de todos esses cenários e da ausência de algoritmos ótimos e eficientes para o problema, se torna evidente a necessidade de seu estudo. Neste trabalho, uma breve revisão bibliográfica sobre o tema é apresentada, além da implementação de um algoritmo genético que pode fornecer resultados competitivos em tempo hábil.

2 TRABALHOS RELACIONADOS

Por ser um problema NP-Difícil, ao longo dos anos muitos trabalhos procuraram desenvolver heurísticas para o OP. Dessa forma, o compromisso compromisso que esses trabalhos assumem é o de tentar equilibrar a qualidade das soluções encontradas com o custo computacional necessário para encontrar tais soluções. Nesta seção, será apresentada uma visão geral sobre alguns dos trabalhos relevantes sobre esse tema, obtida através de algumas compilações sobre OP na literatura [10, 20].

2.1 Algoritmos aproximados

Justamente pela ausência de um algoritmo que resolva problemas NP-Completos ou NP-Difíceis otimamente em tempo polinomial, os algoritmos aproximados se tornaram populares. Eles são heurísticas geralmente simples que geram soluções razoavelmente boas em tempo polinomial [11]. Para uma definição mais formal, podemos dizer que um algoritmo é dito de p-aproximação quando retorna uma solução que é p vezes o valor ótimo para a instância do problema sendo resolvida [8]. Dessa forma, mesmo não conseguindo uma solução tão boa quanto a ótima, a qualidade da solução é compensada pelo custo do algoritmo. Além disso, muitas vezes uma solução desse tipo já é suficiente para as necessidades do problema. Com relação ao OP, dois exemplos de trabalhos que utilizaram essa abordagem são de Chekuri, Korula e Pál [3] e Gavalas et al. [7], que modelaram o problema através de grafos.

2.2 Busca em vizinhança variável

O algoritmo de busca em vizinhança variável é baseado em buscas locais, através de métodos de descida que o levam a algum ponto de mínimo local dentro do espaço de soluções. Uma vez que atinge esses mínimos locais, o algoritmo prossegue com a busca, através da

extensão da busca para exploração de vizinhanças maiores ao redor do mínimo local encontrado. A cada nova busca, pontos ao redor do mínimo encontrado previamente são usados como soluções iniciais para começar a busca [17]. Na literatura, Lau et al. [14] e Zhang et al. [23] são exemplos de trabalhos que utilizaram essa abordagem com o OP.

2.3 *Branch-and-cut*

Por ser um problema de otimização, o OP pode ser formulado através da notação de programação linear. Em particular, os trabalhos que exploraram esse tipo de abordagem na literatura têm modelado o problema como uma programação inteira. Dessa forma, algoritmos já conhecidos para esse tipo de modelagem podem ser utilizados, como é o caso do algoritmo *branch-and-bound* que resolve instâncias de programação linear e adiciona novas restrições à modelagem inicial a fim de manter a restrição de que a solução seja composta somente por números inteiros. O algoritmo *branch-and-cut* utiliza essa ideia para, em adição ao método de planos de corte, ser capaz de melhorar as soluções encontradas gradualmente à medida em que divide a instância do problema em subproblemas. Na literatura, podemos encontrar exemplos desse tipo de abordagem para resolver o OP no trabalho de Angelelli et al. [1] e Dang et al. [4].

2.4 Algoritmos evolutivos

Finalmente, uma das abordagens mais populares atualmente, não só para resolver o OP, mas problemas NP-Difíceis e NP-Completos em geral, os algoritmos evolutivos são métodos inspirados nos mecanismos de evolução biológica [2]. Esses algoritmos representam soluções candidatas para um problema como indivíduos, que formam uma população. Através de um processo iterativo, esses indivíduos são refinados através de processos que simulam o processo evolutivo, tais como recombinação genética e mutação. Essa categoria de algoritmos inclui os Algoritmos Genéticos, a Programação Genética, o Enxame de Partículas, entre outros. Muitos desses já foram utilizados para prover soluções para o OP, como é o caso do trabalho de Dang et al. [5] e Ferreira et al. [6]. Neste trabalho, essa abordagem também será seguida e um arcabouço de algoritmo genético será implementado para resolver o OP.

3 IMPLEMENTAÇÃO

O trabalho foi implementado utilizando-se a linguagem Python 2.7 como referência. Além disso, ele foi desenvolvido para ser utilizado com o *framework* ROS [18, 22]. Esta seção detalha a implementação e apresenta as decisões tomadas ao longo do desenvolvimento.

3.1 Modelagem do mapa

O primeiro passo na implementação do trabalho foi definir uma representação para o mapa que desse suporte às operações necessárias pelo algoritmo genético implementado e que fornecesse uma interface simples para conversão entre coordenadas do mundo real e índices de célula no mapa. Dessa forma, os mapas onde o robô é inserido são grades de duas dimensões de células. Cada uma das células representa $1m^2$ do mapa no mundo real. Essa resolução (tamanho das células) foi fixada para manter a simplicidade. Independentemente das dimensões especificadas para o ambiente do robô (e, consequentemente para a grade que o representa), a origem do eixo de coordenadas (x, y) do ambiente é sempre posicionada no centro geométrico da grade. Sendo assim, a primeira célula da grade (linha 0 e coluna 0, de acordo com a representação de matrizes utilizada na maioria das linguagens de programação) representa a área mais à esquerda e acima no ambiente. A Figura 2 ilustra essa modelagem.

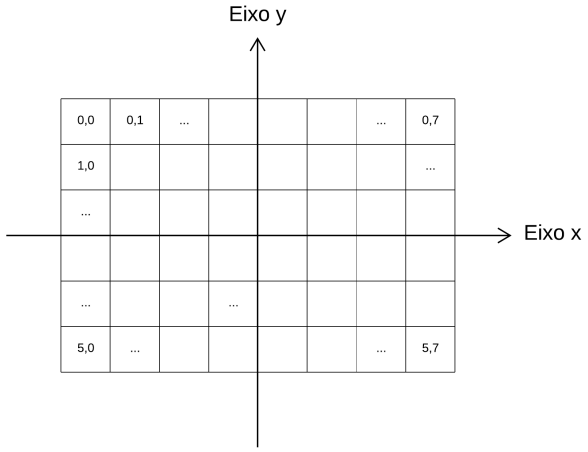

Figura 2: Modelagem de um mapa 6x8 como uma grade de células.

Para realizar a conversão entre índices de células da grade para as posições do centro de massa de tais células no ambiente, e entre posições no ambiente e suas células correspondentes na grade, duas funções foram criadas, ambas no arquivo `grid.py`: a função `index_to_center_of_mass` e `position_to_index`. Além disso, também foi criada a função `get_neighbours` para, dado o índice de uma célula na grade, obter as posições na grade ao seu redor. Essa função é necessária para o processo de mutação descrito na Seção 3.2.

Com a modelagem do ambiente em mãos, foi necessário criar um mapa de recompensas para servir de entrada para o programa. É através desse mapa que os caminhos serão encontrados, equilibrando a soma das recompensas obtidas por visitar cada célula da grade com o custo de fazê-lo. Para a tarefa de criação de um mapa de recompensas aleatório, foi criada a função `get_random_sampling_points` que está presente em um Jupyter Notebook chamado `projeto_final`, na pasta `input` do projeto. Além de imprimir as recompensas geradas em um arquivo, a função também salva uma visualização das recompensas geradas, como ilustrado na Figura 3.

3.2 O algoritmo genético

O algoritmo genético implementado neste trabalho se baseia principalmente no trabalho de Tsiogkas e Lane [19], com algumas simplificações e modificações, uma vez que esse trabalho se propõe a resolver o *Team Orienteering Problem*, uma variação do OP que utiliza múltiplos robôs. A estratégia mais comum para algoritmos genéticos é seguida neste trabalho e detalhada a seguir:

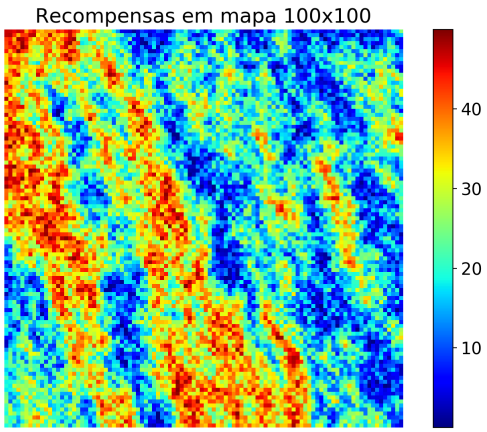

Figura 3: Exemplo de mapa de recompensas gerado para um mapa modelado como uma grade de 100x100 posições. A resolução de cada célula é 1m.

- Inicializar a população de indivíduos aleatoriamente.
- Enquanto uma condição de parada não é satisfeita, faça:
  - Inicialize uma nova população inicialmente vazia.
  - Enquanto o tamanho da população é menor que o tamanho da população inicial, faça:
    - Selecione dois indivíduos da população.
    - Com uma certa probabilidade de cruzamento, realize o cruzamento dos dois indivíduos selecionados, gerando dois novos indivíduos filhos. Do contrário, os indivíduos filhos são iguais aos pais.
    - Com uma certa probabilidade de mutação, realize a mutação dos dois indivíduos filhos. Do contrário, os indivíduos filhos mutados serão iguais aos indivíduos filhos.
    - Adicione os indivíduos filhos à nova população.
- Retorne o melhor indivíduo da população.

Dessa forma, precisamos definir como será a representação dos nossos indivíduos, qual é a condição de parada, como selecionar dois indivíduos da população para cruzamento e mutação, como são feitos os processos de cruzamento e mutação, e como um indivíduo é avaliado.

Dado que indivíduos em algoritmos genéticos são soluções candidatas para o problema sendo resolvido, neste trabalho, os indivíduos representam caminhos que o robô poderia seguir para coletar recompensas respeitando o limite de orçamento. Esses caminhos são representados como uma lista ordenada de vértices - coordenadas (x, y) - do ambiente. Com essa representação de indivíduo, a população inicial é criada iterativamente. Todo novo indivíduo já começa com os vértices definidos como iniciais e finais. Em seguida, tenta-se adicionar novos vértices entre esses dois, de forma que o custo total do caminho não ultrapasse o limite definido. Quando esse limite é atingido, o novo indivíduo está completo e é adicionado à população. Para avaliar a qualidade (*fitness*) de cada indivíduo, obtém-se primeiro seu custo total c e sua recompensa total r. Assim, sua qualidade é definida como $fitness = \frac{r^3}{c}$.

Como dito, precisamos selecionar indivíduos da população durante a fase de geração de nova população, a cada iteração. Isso é feito com base na qualidade dos indivíduos, de forma que aqueles com maior qualidade tendem a terem mais chances de se reproduzirem e sofrerem mutações. Mais especificamente, a seleção implementada neste trabalho é a seleção por torneio. Para cada seleção, k indivíduos são selecionados da população aleatoriamente. Desses k indivíduos, é selecionado aquele com maior qualidade.

Com relação aos dois operadores genéticos, cruzamento e mutação, temos o seguinte esquema. Para cruzamento, primeiro checa-se se os indivíduos selecionados como pais têm algum vértice em comum. Caso seja o caso, ambos têm seus caminhos divididos em duas partes, separando-se assim a cauda do caminho (do vértice em comum até o fim) do resto. Em seguida, as caudas dos indivíduos são trocadas (o primeiro recebe a cauda do segundo e vice-versa). Caso os indivíduos não possuam vértices em comum, os seus filhos serão apenas cópias de si mesmos. Já para efetuar a mutação, escolhe-se um vértice aleatório do indivíduo. Em seguida, todos os vértices vizinhos ao vértice escolhido no mapa são analisados e, caso seja mais atraente (recompensa) inserir o vizinho no lugar do vértice escolhido e a restrição de custo seja mantida, o vizinho é colocado no caminho no lugar do vértice escolhido anteriormente. A Figura 4 ilustra os operadores genéticos.

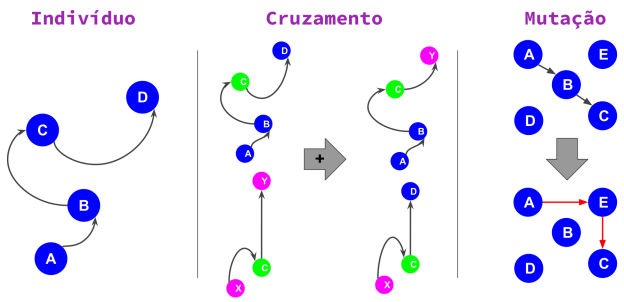

Figura 4: Ilustração do indivíduo e dos operadores genéticos utilizados.

Também foi utilizada um porcentagem de elitismo, isto é, uma porcentagem da população é passada para a próxima geração de indivíduos automaticamente, desde que esteja entre os indivíduos mais qualificados.

Por fim, o algoritmo genético considera a busca como terminada quando qualquer uma das duas coisas acontece: (i) a qualidade do melhor indivíduo da população se manteve estável por 10 gerações consecutivas; ou (ii) o número de gerações atingiu o limite especificado pelo usuário.

4 EXECUÇÃO E ESTRUTURA DO PROJETO

Como mencionado anteriormente, o projeto foi implementado para ser executado no *framework* ROS. Assim, é necessário criar um pacote ROS com os arquivos do projeto. A estrutura do diretório do projeto deve ser similar àquela mostrada na Figura 5.

Considerando que o usuário tenha criado o pacote do projeto com nome `projetofinal`, por exemplo, o programa pode ser executado através do utilitário `rosrun` (após a execução dos nodos `roscore`

```
.
├── CMakeLists.txt
├── maps
│   ├── bitmaps
│   │   └── empty.png
│   ├── holonomic_map.world
│   ├── holonomic_robot.inc
│   └── map.inc
├── package.xml
├── projetofinal.launch
└── src
    ├── ga.py
    ├── grid.py
    ├── pf.py
    └── robot.py
```

Figura 5: Arquivos necessários na árvore do diretório do projeto.

e `stage_ros`) ou através do utilitário `roslaunch`, que utiliza um arquivo `launch` do ROS criado para o projeto. Os comandos são mostrados abaixo.

```
rosrun projetofinal pf.py <r> <c> <ix> <iy> <fx> <fy>
                          <g> <p> <n>
     roslaunch projetofinal projetofinal.launch
     world_file:=<w> rewards:=<r> max_cost:=<c>
start_x:=<ix> start_y:=<iy> end_x:=<fx> end_y:=<fy>
                          navigate:=<n>
```

Onde `<w>` se refere ao arquivo .world utilizado pelo simulador Stage; `<r>` se refere ao arquivo gerado anteriormente com as recompensas no ambiente do robô; `<c>` indica o custo máximo que os caminhos para o robô devem ter; `<ix>` indica a coordenada x da posição inicial; `<iy>` indica a coordenada y da posição inicial; `<fx>` indica a coordenada x da posição final; `<fy>` indica a coordenada y da posição final; `<g>` indica o número de gerações máximo para o algoritmo genético; `<p>` indica o número de indivíduos na população; e `<n>` indica se o robô deve iniciar navegação (1) ou não (0) após o caminho ser encontrado pelo algoritmo.

5 TESTES

Após a implementação do algoritmo, foi necessário efetuar uma série de testes para verificar se seu comportamento estava de acordo com o esperado e, caso não estivesse, tentar entender os motivos. Primeiramente, foi testada a corretude do algoritmo em mapas de três tamanhos diferentes. Em seguida, um mapa pequeno foi utilizado para verificar a velocidade de convergência do algoritmo sob uma situação específica. Depois, uma mesma configuração foi testada duas vezes, variando-se o número de indivíduos da população. E, por fim, o cálculo da *fitness* dos indivíduos foi modificado e comparado com o resultado da mesma configuração do teste com o cálculo de *fitness* apresentado anteriormente. Os resultados são apresentados a seguir.

5.1 Teste 1: verificação de corretude do algoritmo

Para verificar se o algoritmo está se comportando de forma razoável, ele foi testado em 3 mapas de tamanhos diferentes: (i) pequeno, 5x5 células; (ii) médio, 15x15 células; e (iii) grande, 30x30 células. Em todos os casos o número de gerações máximo foi configurado para 100. Além disso, o número de indivíduos foi configurado como uma função das dimensões do mapa, sendo o número de indivíduos $num_{ind} = largura\ x\ altura\ x\ 10$. Os pontos iniciais sempre foram o centro de massa das células mais à esquerda e abaixo, enquanto os pontos finais sempre foram o centro de massa das células mais à direita e acima. O custo máximo dos caminhos, para cada mapa, foi configurado como o mínimo possível para um caminho (tamanho da reta que liga os dois pontos) e, em seguida, como o dobro desse valor. Os resultados são apresentados nas Figuras de 6 a 8. A linha tracejada em cada mapa representa o caminho encontrado. O robô somente para (e coleta recompensas) nos vértices marcados com um 'x' branco.

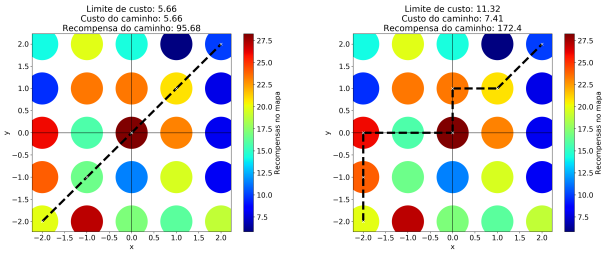

Figura 6: Teste de corretude para mapa 5x5.

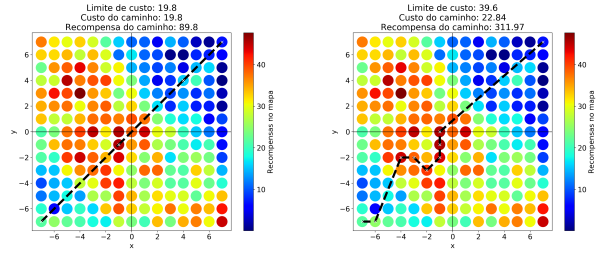

Figura 7: Teste de corretude para mapa 15x15.

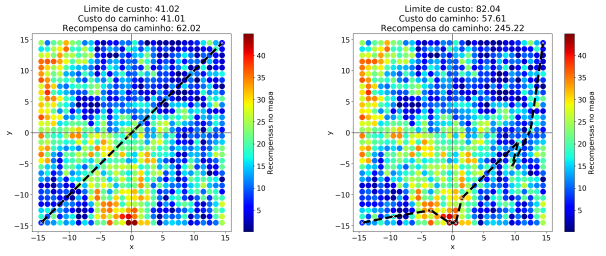

Figura 8: Teste de corretude para mapa 30x30.

Em todos os casos, vemos que o comportamento do robô foi razoável: ele segue em linha reta em direção à posição final quando tem um orçamento limitado, mas tende a ir em direção a áreas de maior recompensa, caso tenha orçamento para isso.

5.2 Teste 2: convergência em mapa pequeno

O segundo teste realizado procurou verificar a velocidade de convergência do algoritmo para a seguinte situação: todas as configurações da execução são idênticas àquelas da execução do teste anterior para o mapa pequeno, entretanto, o custo máximo é configurado para 30 e o número de gerações máximo e de indivíduos foi configurado para 10000. A intenção é de que o algoritmo somente termine de executar quando o melhor indivíduo da população se manteve estável por 10 gerações (convergência). O resultado é mostrado na Figura 9

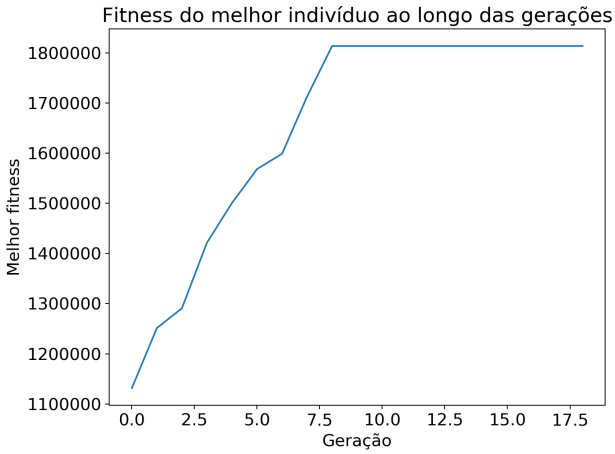

Figura 9: Teste de convergência para o mapa pequeno 5x5.

5.3 Teste 3: variedade de indivíduos em mapa pequeno

Em seguida, nosso terceiro teste procurou verificar a influência do número de indivíduos (diversidade) na população do algoritmo. Novamente utilizando o mapa pequeno, um custo máximo foi fixado e o número de indivíduos foi alterado. Os resultados são mostrados na Figura 10

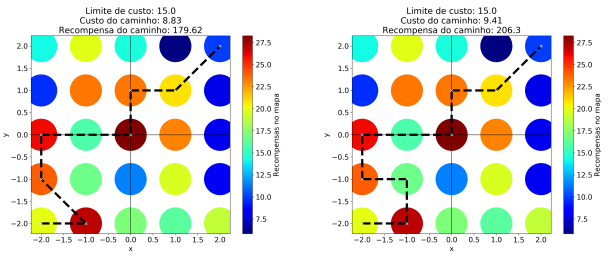

Figura 10: Teste de diversidade de indivíduos para o mapa pequeno 5x5. Vemos que com maior número de indivíduos o caminho resultante foi levemente melhor.

5.4 Teste 4: cálculo de *fitness*

Por fim, vamos verificar como o cálculo da qualidade dos indivíduos influencia no tipo de solução retornada pelo algoritmo. Como mencionado na Seção 3.2, a qualidade de cada indivíduo é definida como $fitness = \frac{r^3}{c}$, onde r representa a soma das recompensas ao visitar cada um de seus vértices e c é o custo total de fazê-lo. Vemos que esse cálculo dá maior 'peso' à recompensa obtida, entretanto, leva em consideração tanto o custo quanto a recompensa. Dessa forma, durante os testes a recompensa e o custo dos melhores indivíduos de cada população variavam independentemente. Em casos onde deseja-se maximizar a recompensa obtida, por exemplo, podemos simplesmente remover o custo do caminho do cálculo, ainda que respeitando o custo máximo durante os procedimentos do algoritmo. Este último teste compara uma execução do algoritmo com o cálculo de *fitness* apresentado e outra com uma *fitness* definida simplesmente como a recompensa do caminho. O resultado é mostrado na Figura 11.

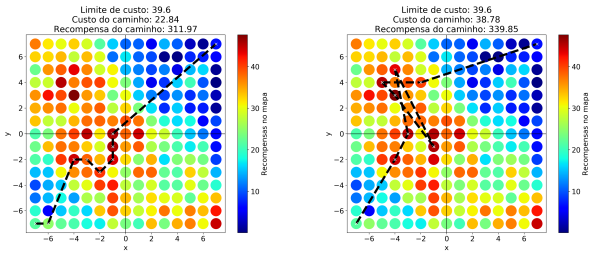

Figura 11: Teste de mudança de cálculo de *fitness* para o mapa médio 15x15. À esquerda, o caminho resultante com o cálculo de *fitness* definido na Seção 3.2. À direita, o caminho resultante com o cálculo de *fitness* que somente leva em consideração a recompensa total do caminho.

6 CONCLUSÃO

Neste trabalho foi implementada uma abordagem de Algoritmo Genético para resolver o *Orienteering Problem*. Testes foram executados para verificar o comportamento do algoritmo e suas particularidades. Considero que a implementação do trabalho se deu sem maiores problemas e que o objetivo foi alcançado: realizar um estudo mais aprofundado de um problema não visto durante as aulas da disciplina Robótica Móvel. Além disso, foi muito interessante poder explorar um problema com tantas aplicações reais na área da robótica.

Com relação às dificuldades encontradas, posso citar a dificuldade de teste da implementação, uma vez que, já que o OP é NP-Difícil, não conseguimos encontrar a solução ótima para uma de suas instâncias a fim de comparar com a solução encontrada pela implementação do Algoritmo Genético. Além disso, os próprios algoritmos evolutivos são complicados de serem testados, uma vez que existem vários parâmetros que influenciam seus resultados, como as probabilidades de aplicação de cada operador genético, o número de indivíduos utilizado, o número de gerações, etc.

Enquanto este trabalho focou no OP, existem diversas variações desse problema que seriam de grande interesse para trabalhos futuros dentro da solução proposta, tais como: *Correlated Orienteering*

Problem (COP), onde recompensas são coletadas também para posições vizinhas às visitadas; *Team Orienteering Problem* (TOP), onde caminhos devem ser encontrados para um time de robôs; *Correlated Team Orienteering Problem*, que combina o TOP e o COP; e o *Dubins Orienteering Problem* (DOP), que adiciona a restrição da cinemática de veículos *Dubins*.

REFERÊNCIAS

[1] Enrico Angelelli, Claudia Archetti, and Michele Vindigni. 2014. The clustered orienteering problem. *European Journal of Operational Research* 238, 2 (2014), 404–414.

[2] Jason Brownlee. 2011. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee.

[3] Chandra Chekuri, Nitish Korula, and Martin Pál. 2012. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms (TALG)* 8, 3 (2012), 23.

[4] Duc-Cuong Dang, Racha El-Hajj, and Aziz Moukrim. 2013. A branch-and-cut algorithm for solving the team orienteering problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 332–339.

[5] Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. 2011. A pso-based memetic algorithm for the team orienteering problem. In *European Conference on the Applications of Evolutionary Computation*. Springer, 471–480.

[6] João Ferreira, Artur Quintas, José A Oliveira, Guilherme AB Pereira, and Luis Dias. 2014. Solving the team orienteering problem: Developing a solution tool using a genetic algorithm approach. In *Soft Computing in Industrial Applications*. Springer, 365–375.

[7] Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, Grammati Pantziou, and Nikolaos Vathis. 2015. Approximation algorithms for the arc orienteering problem. *Inform. Process. Lett.* 115, 2 (2015), 313–315.

[8] Michel X Goemans and David P Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)* 42, 6 (1995), 1115–1145.

[9] Bruce L Golden, Larry Levy, and Rakesh Vohra. 1987. The orienteering problem. *Naval Research Logistics (NRL)* 34, 3 (1987), 307–318.

[10] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research* 255, 2 (2016), 315–332.

[11] David S Johnson. 1974. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences* 9, 3 (1974), 256–278.

[12] David S. Johnson. 1985. The NP-completeness column: an ongoing guide. *J. algorithms* 6, 3 (1985), 434–451.

[13] James J Kuffner and Steven M LaValle. 2000. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, Vol. 2. IEEE, 995–1001.

[14] Hoong Chuin Lau, William Yeoh, Pradeep Varakantham, Duc Thien Nguyen, and Huaxing Chen. 2012. Dynamic stochastic orienteering problems for risk-aware applications. *arXiv preprint arXiv:1210.4874* (2012).

[15] James Ng and Thomas Bräunl. 2007. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems* 50, 1 (2007), 73–84.

[16] Robert Penicka, Jan Faigl, Petr Vána, and Martin Saska. 2017. Dubins Orienteering Problem. *IEEE Robotics and Automation Letters* 2, 2 (2017), 1210–1217.

[17] José Andrés Moreno Pérez, Nenad Mladenović, Belén Melián Batista, and Ignacio J. García del Amo. 2006. *Variable Neighbourhood Search*. Springer US, Boston, MA, 71–86. https://doi.org/10.1007/0-387-33416-5_4

[18] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[19] Nikolaos Tsiogkas and David M Lane. 2018. An Evolutionary Algorithm for Online, Resource-Constrained, Multivehicle Sensing Mission Planning. *IEEE Robotics and Automation Letters* 3, 2 (2018), 1199–1206.

[20] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209, 1 (2011), 1–10.

[21] Charles W Warren. 1989. Global path planning using artificial potential fields. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE, 316–321.

[22] Leon Jung Darby Lim Yoonseok Pyo, Hancheol Cho. 2017. *ROS Robot Programming (English)*. ROBOTIS. http://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51

[23] Shu Zhang, Jeffrey W Ohlmann, and Barrett W Thomas. 2014. A priori orienteering with time windows and stochastic wait times at customers. *European Journal of Operational Research* 239, 1 (2014), 70–79.