Uma imagem com Tipo de letra, Gráficos, design gráfico, captura de ecrã

Descrição gerada automaticamente

**Relatório Sprint C - ALGAV**

**Turma 3DC-Grupo 14**

1211439 - Pedro Costa

1211134 - Pedro Viana

1211436 - David Pinho

1210825 - Pedro Ferreira

**Professor:**

David Freitas, DCF

**Unidade Curricular:**

Algoritmia Avançada

Índice

[Adaptação e Otimização do Algoritmo Genético 3](#_Toc155111758)

[Avaliação do Tempo entre tarefas 3](#_Toc155111759)

[Evitar sequência de cruzamentos sempre entre os mesmos pares de tarefas 3](#_Toc155111760)

[Passar os melhores e método de Seleção não puramente elitista 4](#_Toc155111761)

[Condição de término 6](#_Toc155111762)

[Complexidade 7](#_Toc155111763)

[Estado de Arte – US 1440 8](#_Toc155111764)

[Pros e Contras 11](#_Toc155111765)

[Bibliografia 13](#_Toc155111766)

# Adaptação e Otimização do Algoritmo Genético

## Avaliação do Tempo entre tarefas

Para a avaliação dos indivíduos adaptamos ao nosso projeto, para que o predicado do cálculo do tempo recebesse sequencialmente um par de tarefas, para ambas as tarefas, vai buscar as respetivas coordenadas, mais precisamente as coordenadas de destino da primeira e as coordenadas de origem da segunda e chama o predicado feito no sprint passado do cálculo do caminho com o nome **find\_caminho/5**. Este predicado vai então realizar o cálculo do caminho entre as duas tarefas e também retornar o tempo total do caminho que vai ser o valor usado depois no algoritmo genético.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamentePara o algoritmo também fizemos uma otimização para este cálculo do tempo, em vez de fazermos o cálculo a cada geração, basicamente a cada execução, fazemos de uma vez todas as transições possíveis entre tarefas antes de executarmos o algoritmo e geramos dinamicamente essas transições com o :**-dynamic transita/3**. Isto vai diminuir drasticamente o tempo de execução do algoritmo, fazendo com que a execução seja quase instantânea em alguns casos em vez de demorar vários minutos, isto é conseguido através do predicado **gera\_transicoes**.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

## Evitar sequência de cruzamentos sempre entre os mesmos pares de tarefas

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteO algoritmo genético fornecido, para os cruzamentos tenta sempre cruzar o 1º com o 2º, o 3º com o 4º e assim respetivamente, para evitar este problema basta mudar o algoritmo para antes implementar um predicado que randomize esse acontecimento. Para realizar uma permutação aleatória entre os indivíduos de uma população utilizamos o predicado **random\_permutation/2** do SWI Prolog.

## Passar os melhores e método de Seleção não puramente elitista

Uma das maiores otimizações do algoritmo genético é a possibilidade de passar os melhores indivíduos para a próxima geração e fazer com que a seleção de indivíduos não seja puramente elitista, isto resolve problemas como o risco de se perder indivíduos com uma avaliação muito boa e também no caso da seleção de dar oportunidades a indivíduos que obtiveram uma avaliação pior que passem á próxima geração.

Para implementar estas otimizações, reformulou-se o predicado **gera\_geracao/3** do original por completo, primeiro juntamos os **N** indivíduos da população atual com os seus descendentes obtidos por cruzamento e mutação e de seguida removemos todos os elementos repetidos. Para esta operação utilizamos dois predicados do **SWI-Prolog,** o predicado **append/3** e o **list\_to\_set/2,** sendo que o primeiro vai juntar todos os indivíduos de ambas as listas e o segundo vai remover todos os repetidos.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteApós obtermos a lista sem elementos repetidos, vamos ordená-los por ordem crescente de acordo com a sua avaliação através do predicado **ordena\_populacao/2** que já vinha no algoritmo originale a seguir selecionamos os **P** melhores indivíduos dessa lista com o predicado **seleciona\_melhores/2**, no nosso caso utilizamos um valor de **20%.**

Obtidos os melhores indivíduos, retiramo-los dessa respetiva lista com o predicado **remove\_melhores/3** pois vão para geração seguinte respetivamente, obtemos assim uma nova lista contendo **T – P** indivíduos onde começamos por associar a cada um deles o produto da avaliação por um número aleatório gerado entre 0 e 1 através do predicado **associa\_produto\_avaliacao/2** e consequentemente ordenamos essa lista por ordem crescente de acordo com o seu produto com o predicado **ordena\_populacao\_produto/2.**

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamentePara finalizar, passamos os **N – P** primeiros elementos dessa lista para a geração seguinte com o predicado **restantes\_melhores/4**, sendo que para isso primeiro temos de remover o valor do produto associado e ficar só com a respetiva avaliação, para isto utilizamos o predicado **remover\_produtos/2.** Com a lista de melhores individuos conseguido inicialmente e também com a lista de restantes melhores conseguidos anteriormente podemos então juntar ambos numa única lista, ordenar através dos predicados **append/3 e ordena\_populacao/2** e passar então para a geração seguinte.

## Condição de término

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteO algoritmo genético fornecido executa respetivamente para o número de gerações que o utilizador introduzir como input, mas para além desta condição adicionamos também mais uma condição em que o algoritmo corre até atingir um determinado individuo com uma avaliação específica, isto é, até atingir um valor igual ao menor ao estabelecido pelo utilizador. Para além disto, caso este não encontre porque o valor indicado é inferior á s9lução ideia do problema, implementamos também uma condição que estabelece um tempo limite de execução para evitar ciclos infinitos. Isto é conseguido através de verificações efetuadas imediatamente no início do predicado **gera\_geracao/3,** com os predicados **verifica\_condicao\_termino/2 e verifica\_tempo\_limite/2** sendo que o primeiro então verifica se há algum individuo com uma avaliação igual ou inferior ao valor estabelecido e depois verifica se o atual tempo de execução do algoritmo já ultrapassou o limite. Para esta condição de tempo limite utiliza-se o predicado do Prolog **get\_time/1** para obter esse tal valor do tempo de execução atual do algoritmo e depois verifica-se então se esse valor é maior que o valor introduzido para o limite.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

## Complexidade

Para realizar o estudo da complexidade, desenvolvemos um predicado que cria tarefas aleatórias. O número de tarefas aleatórias é determinado pelo valor de entrada fornecido pelo usuário.

Na primeira fase do estudo de complexidade, começamos por criar apenas quatro tarefas aleatórias. Essas quatro tarefas aleatórias foram usadas para determinar a melhor sequência entre elas, a fim de compreender qual é o caminho ideal para o robot seguir.

Observamos que, nessas quatro tarefas geradas aleatoriamente, os algoritmos não encontraram dificuldades em aplicar a melhor sequência. Portanto, decidimos testar com cinco tarefas. Mais uma vez, não houve problemas na execução do algoritmo, embora tenhamos notado um aumento no tempo de execução em comparação com o exemplo anterior com quatro tarefas.

Espera-se que esse tempo de execução aumente gradualmente à medida que o número de tarefas aumenta. No entanto, após testar com diversos valores de tarefas, notamos que, a partir de dez tarefas, o SWI-Prolog não consegue executar a operação corretamente.

O erro descrito na console do SWI-Prolog é o seguinte:



# Estado de Arte – US 1440

**Efetuar um estudo do estado da arte na geração de trajetórias de robots. Poderão combinar pesquisa bibliográfica própria com ferramentas de IA Generativa (por exemplo ChatGPT ou Bing AI), indicando os pedidos e deixando bem claro o que foi produzido via ferramentas de IA Generativa.**

A geração de trajetórias de robots é uma parte fundamental da nossa aplicação. Para este estudo do estado de arte baseamo-nos fundamentalmente na consulta de artigos, guias disponibilizados pelos docentes e também algumas ferramentas de IA como o **GitHub Copilot** ou o **ChatGPT**.

Para chegar á solução pretendida foi usado um algoritmo genético (AG) que é uma técnica de otimização heurística inspirada no processo de seleção natural. O objetivo é encontrar a melhor sequência de tarefas para execução, minimizando o tempo total de transição entre as tarefas.

O AG começa a gerar uma população inicial de sequências aleatórias de tarefas. De seguida, avalia a aptidão de cada sequência com base na soma dos tempos de transição entre as tarefas. O processo evolutivo envolve a seleção dos melhores indivíduos, que são submetidos a operadores genéticos, como cruzamento e mutação, para gerar uma nova geração. Esse processo repete-se por um número predefinido de gerações ou até que a condição de término seja alcançada, como alcançar a avaliação pretendida ou ultrapassar o limite de tempo. *(Jorge Nachtigall-2020).*

Depois de explicado como funciona o algoritmo genético vamos usar uma das ferramentas de IA mencionadas em cima, no caso o **ChatGPT**, para verificar e comparar a rentabilidade deste algoritmo com outros parecidos. Para tal fizemos o seguinte pedido:

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

De seguida vamos utilizar uma ferramenta de IA diferente para saber quais são os mais eficientes. A ferramenta a utilizar vai ser o **Bing AI**:

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

A conclusão a que chegamos é que cada algoritmo tem os seus pros e contras, mas em resumo, dentro desta lista de algoritmos os mais eficientes são os seguintes pois são os mais usados em problemas de otimização:

* Otimização por Enxame de Partículas (PSO).
* Algoritmos Genéticos Locais (Memetic Algorithms).

## Pros e Contras

Depois de escolhidos os algoritmos vamos agora fazer uma análise dos pros e contras de cada um:

**Pontos Fortes do Algoritmo de Otimização por Enxame de Partículas (PSO):**

* **Exploração Eficiente:** O PSO é eficaz na exploração de espaços de procura complexos, pois as partículas colaboram para explorar diferentes regiões e encontrar soluções promissoras. *(André Pacheco-2016).*
* **Simplicidade e Implementação Simples:** A simplicidade do algoritmo facilita a sua implementação e compreensão. Isso torna-o uma escolha eficaz na otimização de diversas aplicações.
* **Adaptação Dinâmica:** O PSO é capaz de se adaptar dinamicamente a mudanças no ambiente de otimização, ajustando as suas trajetórias com base na experiência acumulada pelas partículas.
* **Aplicações em Problemas Contínuos e Discretos:** Pode ser aplicado eficientemente tanto a problemas de otimização contínua quanto discreta.
* **Paralelização Simples:** A natureza paralela do PSO permite uma fácil paralelização, tornando-o adequado para implementações em ambientes de computação distribuída ou GPU. *(Andrey Dik-2023).*

**Pontos Fracos do Algoritmo de Otimização por Enxame de Partículas (PSO):**

* **Sensibilidade aos Parâmetros:** O desempenho do PSO pode ser sensível à escolha dos parâmetros, como coeficientes de inércia ou outros fatores. Encontrar a combinação ideal pode ser um desafio**.**
* **Convergência Prematura:** Em algumas situações, o PSO pode convergir prematuramente para uma solução ótima, especialmente se o tamanho da população ou o número de iterações não for ajustado adequadamente.
* **Limitações em Problemas Multimodais:** O PSO pode ter dificuldades em lidar com problemas multimodais (com várias soluções ótimas), pois as partículas podem se concentrar em uma única região do espaço de busca.
* **Dificuldade com Espaços de Busca Dinâmicos:** O PSO pode ter dificuldades em se adaptar eficientemente a espaços de busca que mudam dinamicamente durante a otimização.
* **Influência da Topologia:** A escolha da topologia de comunicação entre as partículas pode afetar o desempenho do PSO, e encontrar a topologia ideal pode ser um desafio em problemas complexos.
* **Não Garantia de Convergência Global:** Não há garantia de convergência global para a

solução ótima, e o PSO pode ficar preso em mínimos locais em certas situações.

**Pontos Fortes dos Algoritmos Genéticos Locais (Memetic Algorithms):**

* **Combinação de Diversidade e Busca Local:** Os Algoritmos Genéticos Locais (AGL) combinam estratégias de algoritmos genéticos, que são eficazes no que toca a explorar amplamente o espaço de procura, com procedimentos de procura local, aumentando a capacidade de refinar soluções promissoras. *(Márcio Miranda-2014).*
* **Manutenção da Diversidade:** A inclusão de operadores genéticos tradicionais permite a manutenção da diversidade na população, o que é crucial para explorar diferentes regiões do espaço de procura e evitar convergência prematura.
* **Adaptação a Diferentes Topologias e Estruturas de Problemas:** Os AGLs são flexíveis e podem ser adaptados para lidar com diferentes topologias e estruturas de problemas, tornando-os aplicáveis a uma ampla gama de cenários de otimização.
* **Exploração e Explotação Equilibradas:** Ao combinar a exploração global proporcionada pelos operadores genéticos e a exploração local por meio de técnicas de procura local, os AGLs procuram um equilíbrio entre explorar novas regiões e aprimorar soluções já conhecidas.

**Pontos Fracos dos Algoritmos Genéticos Locais (Memetic Algorithms):**

* **Sensibilidade a Parâmetros:** A eficácia dos AGLs pode depender da escolha adequada de parâmetros, como taxas de cruzamento, mutação e a intensidade da procura local. A sintonia desses parâmetros pode ser problemática em alguns casos.
* **Complexidade Computacional:** A incorporação de procedimentos de procura local pode aumentar a complexidade computacional dos AGLs, especialmente em problemas nos quais a avaliação da função objetivo é de elevado custo.
* **Potencial de Convergência Prematura:** Em determinadas situações, os AGLs podem estar suscetíveis à convergência prematura, especialmente se a procura local não for suficientemente explorada ou se a população perder a diversidade necessária.
* **Dificuldades em Problemas com Múltiplos Locais Ótimos:** Em cenários com múltiplos locais ótimos, a combinação de estratégias genéticas e procura local pode não ser suficiente para garantir a exploração adequada de diferentes soluções ótimas.
* **Desafios na Definição de Operadores Genéticos e Métodos de Procura Local:** A escolha de operadores genéticos e métodos de procura local apropriados para um problema específico pode ser um desafio e requer conhecimento especializado.

# Bibliografia

André Pacheco, 2016, “Otimização por enxame de partículas - PSO”, Computação Inteligente.com.br.

Andrey Dik, 2023, “Algoritmos de otimização populacionais: Enxame de partículas (PSO)”, mql5.com.

Márcio Miranda, 2014, “Algoritmos Genéticos: Fundamentos e Aplicações”, nce.ufrj.br.

Jorge Nachtigall, 2020, “Primeiros passos com algoritmos genéticos - uma aplicação prática”, linkedin.com.

Edirlei Lima, “Aula 06 – Algoritmos Genéticos”, edirlei.com.