UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ana Carolina Ladeira Costa Queiroz

Sistemas Multiagentes para Coleta e Entrega Combinando Algoritmos Genéticos e Planejamento de Caminho

Juiz de Fora
2022

Ana Carolina Ladeira Costa Queiroz

Sistemas Multiagentes para Coleta e Entrega Combinando Algoritmos Genéticos e Planejamento de Caminho

Dissertação apresentada ao Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. D.Sc. Heder Soares Bernardino

Coorientador: Prof. D.Sc. Alex Borges Vieira

Juiz de Fora

2022

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Queiroz, Ana Carolina Ladeira Costa.
Sistemas Multiagentes para Coleta e Entrega Combinando Algoritmos Genéticos e Planejamento de Caminho / Ana Carolina Ladeira Costa Queiroz. – 2022.
55 f. : il.

Orientador: Heder Soares Bernardino
Coorientador: Alex Borges Vieira
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Pós-Graduação em Ciência da Computação, 2022.

1. Sistemas Multiagentes. 2. Alocação de Tarefas. 3. Planejamento de Caminho. 4. Algoritmo Genético. 5. Otimização Multiobjetivo. I. Bernardino, Heder Soares, orient. II. Vieira, Alex Borges, coorient. III. Título

Ana Carolina Ladeira Costa Queiroz

Sistemas Multiagentes para Coleta e Entrega Combinando Algoritmos Genéticos e Planejamento de Caminho

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 04 de março de 2022.

BANCA EXAMINADORA

Prof. Dr. Heder Soares Bernardino - Orientador

Universidade Federal de Juiz de Fora

Prof. Dr. Alex Borges Vieira

Universidade Federal de Juiz de Fora

Profª. Dra. Luciana Brugiolo Gonçalves

Universidade Federal de Juiz de Fora

Prof. Dr. Eduardo Krempser da Silva

Fundação Oswaldo Cruz

Juiz de Fora, 16/02/2022.

Documento assinado eletronicamente por **Mario Antonio Ribeiro Dantas**, **Professor(a)**, em 23/02/2022, às 18:36, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Alex Borges Vieira**, **Coordenador(a) em exercício**, em 04/03/2022, às 16:48, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Heder Soares Bernardino**, **Professor(a)**, em 04/03/2022, às 16:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Luciana Brugiolo Goncalves**, **Professor(a)**, em 23/03/2022, às 15:12, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Eduardo Krempser da Silva**, **Usuário Externo**, em 23/03/2022, às 15:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **0682217** e o código CRC **CDD14C10**.

À minha mãe.

AGRADECIMENTOS

Agradeço a minha família, que esteve sempre presente me ajudando e apoiando incondicionalmente.

Ao meu namorado João Marcos, por toda ajuda e compreensão durante todos esses anos juntos.

Aos meus amigos por todo o incentivo e apoio, especialmente ao Bráulio. Obrigada por estarem do meu lado durante todo o processo.

Aos amigos que eu fiz no PGCC pela troca de experiências, ajuda e momentos de companheirismo.

Ao Prof. Heder e Prof. Alex pela orientação e por confiar em mim para realização desse trabalho. Obrigada pelos ensinamentos, paciência e compreensão em todos os momentos.

Agradeço à todos os professores do DCC e do PGCC que buscaram transmitir seu conhecimento e ensinamentos de vida ao longo da minha vida acadêmica.

Aos funcionários do DCC e PGCC pelo suporte.

E agradeço à CNPq, FAPEMIG, CAPES e UFJF pelo apoio financeiro que foi de extrema importância para viabilizar a realização desse trabalho.

RESUMO

Centros de distribuição estão cada dia mais automatizados com a utilização de robôs móveis autônomos. Estes robôs desempenham a função de movimentar produtos, aumentando a produtividade nos armazéns. Vários trabalhos vêm sendo estudados na literatura que buscam capturar essas características e o *Multi-Agent Pickup and Delivery* (MAPD) é um exemplo de problema desta área. Neste problema, tarefas aparecem no sistema em diferentes instantes de tempo, e cada tarefa tem duas posições, uma posição de coleta e uma de entrega. Os agentes devem atender a esse fluxo de tarefas, se deslocando para a posição de coleta e depois de entrega da tarefa. Comumente, esse problema tem duas partes: (i) alocação de tarefas, em que o agente recebe uma sequência de tarefas a serem executadas, e (ii) planejamento de caminho, no qual é necessário encontrar o melhor caminho para o agente realizar sua tarefa sem colidir com outros agentes. O problema de encontrar caminhos para multiagentes também é conhecido como *Multi-Agent Path Finding* (MAPF). Neste trabalho, foram propostos diferentes algoritmos genéticos para resolver a parte de alocação de tarefas do MAPD. Também foi apresentado uma análise dos objetivos dos problemas e este problema foi tratado com uma abordagem multiobjetivo, utilizando o *Non-dominated Sorting Genetic Algorithm* (NSGA-II) a fim de minimizar duas funções objetivo, *makespan* e *service time*. Para o sub-problema MAPF, foram utilizados algoritmos de planejamento de caminhos já conhecidos na literatura: o *Prioritized Planning* (PP) e a *Conflict Based Search* (CBS). Também foi utilizada outra abordagem para o *Prioritized Planning*, denominado PP-E. Esta abordagem, PP-E, tem como fim evitar futuras colisões entre agentes, possibilitando que o agente se desloque para outra posição livre após chegar na sua posição de objetivo. Experimentos computacionais foram realizados em dois ambientes, com diferentes tamanhos, números de agentes, quantidade de tarefas e taxa de entrada de tarefas no sistema. Os resultados foram comparados com algoritmos da literatura e mostraram que a abordagem proposta alcança melhores resultados quando comparada a outras técnicas.

Palavras-chave: Sistemas Multiagentes. Alocação de Tarefas. Planejamento de Caminho. Algoritmo Genético. Otimização Multiobjetivo.

ABSTRACT

Distribution centers are increasingly automated with the use of autonomous mobile robots. These robots perform the function of moving products, increasing productivity in warehouses. Several works have been studied in the literature that try to capture these characteristics and the Multi-Agent Pickup and Delivery (MAPD) problem is an example of problem from this area. In this problem, tasks appear in the system at different times, and each task has two positions, a pickup and a delivery position. Agents attend to a stream of incoming tasks, moving to pickup position and then to delivery position. Commonly, this problem has two parts: (i) task allocation, in which the agent receives a sequence of tasks to be executed, and (ii) path planning, in which it is necessary to find the best way for the agent to perform its task without colliding with other agents. The problem of finding multi-agent paths is also known as Multi-Agent Path Finding (MAPF). In this work, different genetic algorithms were proposed to solve the task allocation part of the MAPD. An analysis of the objectives of the problems was also presented and this problem was treated with a multi-objective approach, using the Non-dominated Sorting Genetic Algorithm (NSGA-II) in order to minimize two objective functions, namely, makespan and service time. For the MAPF sub-problem, path planning algorithms from the literature were used: Prioritized Planning (PP) and Conflict Based Search (CBS). Another approach to Prioritized Planning was also proposed, called PP-E. This approach aims to avoid future collisions between agents, allowing the agent to move to another free position, after reaching its objective position. Computational experiments were carried out in two environments, with different sizes, number of agents, number of tasks and rate of entry of tasks in the system. The results were compared with algorithms from the literature and showed that the proposed approach achieves better results when compared to other techniques.

Keywords: Multiagent Systems. Task Allocation. Path Planning. Genetic Algorithm. Multi-Objective Optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
AGV	*Automated Guided Vehicles*
BHS	*Baggage handling systems*
CBS	*Conflict-Based Search*
CT	*Constraint tree*
MAPD	*Multi-agent pickup-and-delivery*
MAPF	*Multi-agent path finding*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
PDP	*Pickup and Delivery Problem*
SIC	*Sum of the costs*
TP	*Token passing*
TPTS	*Token passing with task swapping*
TSP	*Traveling Salesman Problem*

SUMÁRIO

1 INTRODUÇÃO

Os estoques e encomendas mundiais vem aumentando consideravelmente nos últimos anos. Um dos motivos é a tendência mundial com o aumento do número de compras feitas "online", seja no atacado ou no varejo (35). Os depósitos e centros de distribuição são os responsáveis pelo armazenamento dos itens e pelo fluxo das mercadorias. Esses depósitos recebem uma enorme quantidade diária de pedidos e precisam atender ao maior número possível desses pedidos em pouco tempo, otimizando os recursos. No sistema tradicional, esse serviço é feito manualmente por trabalhadores que retiram as encomendas e as alocam para as devidas transportadoras, o que demanda tempo, recursos e gera erros, como trocas ou perdas de objetos.

Uma solução para esse problema é a automatização de armazéns utilizando robôs móveis autônomos, também chamados de agentes, que realizam a tarefa de se movimentar pelo armazém, buscar os produtos e prepará-los para a entrega. A Kiva Systems (hoje em dia Amazon Robotics), em 2006, entrou em operação com uma instalação permanente de um sistema automatizado de controle de mercadorias em armazéns, confirmando a eficiência do projeto (35). Ao invés dos trabalhadores se movimentarem pelo armazém buscando produtos, o sistema fazia com que as prateleiras fossem até os trabalhadores, o que (pelo menos) dobrou a produtividade.

Esse tipo de sistema tem vários benefícios a destacar, como: (i) menos trabalhadores estão envolvidos, então as interdependências desaparecem, (ii) eliminação do processamento em lote, já que tudo pode ser feito em tempo real, (iii) nenhum ponto único de falha, se um robô falhar, todo o sistema não é interrompido, ao contrário de um sistema com transportadores, (iv) flexibilidade, já que o sistema pode funcionar em várias salas sem necessidade de adaptações significativas, e (v) permite expandir, se mais capacidade for necessária, pode se adicionar mais robôs (35).

Existem várias outras aplicações além de centros de distribuições que utilizam agentes para operar em um ambiente conhecido. Um exemplo são os sistemas de manuseio de bagagem em aeroportos, do inglês *Baggage handling systems* (BHS), que utilizam veículos guiados automatizados, do inglês *Automated Guided Vehicles* (AGV), que transportam malas por toda a sala de bagagens, coletando de forma automatizada as malas e encontrando caminhos ideais e sem conflitos para seus destinos (5). Outro caso são os sistemas de veículos autônomos de reboque de aeronaves que, sob comando, navegam autonomamente até uma aeronave designada, se conectam a ela, e rebocam a aeronave para um local designado (uma pista de decolagens, ou um portão de desembarque), se desconectam de forma autônoma e navegam para outro local designado, seja uma área de parada ou para atender outra aeronave (21). Outras aplicações também incluem: robôs de escritório, robôs de salvamento e personagens de video game (30, 17).

Um problema comum na computação é o desafio em encontrar o caminho de um ponto ao outro (22). Um modelo viável é através de um grafo, de modo que o agente se mova de vértice para vértice ao longo das arestas do grafo até chegar ao seu objetivo. Uma extensão desse problema é o planejamento de caminhos multiagente, do inglês *Multi Agent Pathfinding* (MAPF) (28).

Um sistema multiagente é constituído por diversos agentes autônomos com a capacidade de tomar suas próprias decisões, percebendo seu ambiente e agindo sobre ele (34, 12, 7). Nesse tipo de sistema, um conjunto de agentes interage com o ambiente e uns com os outros a fim de satisfazer seus objetivos, que são mais complexos de resolver com agentes isolados. Para interagir com sucesso, os agentes precisam cooperar, coordenar, negociar e se comunicar uns com os outros. Os sistemas multiagentes podem ser divididos em três partes: (i) o ambiente no qual os agentes estão localizados deve ser especificado, (ii) os agentes devem ser definidos, bem como o modo que percebem o ambiente e como processam as informações coletadas e (iii) as interações entre os agentes e o ambiente devem ser especificadas.

No problema MAPF, cada agente tem o seu objetivo e o sistema deve encontrar o melhor caminho, com o menor custo, para que todos alcancem seus objetivos sem colisões entre si. Isso é feito em um ambiente predeterminado, no qual os únicos obstáculos dinâmicos são os próprios agentes e cada agente tem o conhecimento da posição do seu objetivo, do ambiente e da localização de todos os demais agentes. O problema acaba quando todos os agentes chegam em seus objetivos, logo esse problema é descrito como "*one shot*". Portanto, o problema MAPF não captura características importantes do mundo real, no qual os agentes recebem novas posições de destino após chegarem em uma posição final.

Nesse trabalho será abordada uma variação desse problema, uma versão "online" chamada na literatura de *Multi-Agent Pickup and Delivery* (MAPD) (19). O problema de transportar mercadorias ou passageiros de diferentes origens para diferentes destinos, é chamado *Pickup and Delivery Problem* (PDP) e já foi abordado em (25). Na abordagem MAPD, as tarefas podem aparecer no sistema a qualquer momento e os agentes devem executar essas tarefas. Cada tarefa possui duas posições, chamadas de posição de coleta e posição de entrega. Para executar a tarefa, o agente deve primeiramente se deslocar para a posição de coleta e posteriormente para a posição de entrega da tarefa. Após isto, o agente está disponível para executar uma nova tarefa. Os agentes precisam escolher as melhores tarefas com o objetivo de terminar de executá-las o mais rápido possível.

Desse modo, esse problema pode ser dividido em dois sub-problemas: alocação de tarefas e planejamento de caminho. O primeiro problema consiste em determinar a ordem das tarefas que cada agente deve executar. Depois disso, o segundo problema é encontrar um caminho para cada agente até a sua tarefa. O problema acaba quando não há mais

nenhuma tarefa a ser executada no sistema.

Esse trabalho tem como objetivo apresentar uma solução eficiente para tratar o problema MAPD. Para isso são propostos algoritmos genéticos para alocar as tarefas para os agentes, e são utilizados dois diferentes algoritmos para o planejamento de caminho já vistos na literatura (24). Também realizamos um estudo sobre os métodos para evitar colisões no planejamento de caminhos. Com o propósito de entender melhor o problema, foi feita uma análise dos objetivos. Dessa forma, foi proposta uma otimização multiobjetivo para a parte de alocação de tarefas do problema (23).

Os testes foram realizados com instâncias disponíveis na literatura, em dois ambientes que variam no tamanho, número de agentes, quantidade de tarefas e taxa com que as tarefas entram no sistema (tarefas são adicionadas ao sistema em diferentes tempos). Essas diferentes características das instâncias tentam capturar melhor a natureza dos problemas do mundo real.

Os experimentos foram feitos utilizando diferentes algoritmos de planejamento de caminho, incluindo uma modificação em um desses algoritmos quando o agente chega em sua posição de destino, a fim de otimizar o tratamento quanto a futuras colisões. Para a parte da alocação de tarefas foram utilizadas 4 diferentes abordagens de algoritmos genéticos mono-objetivos e 3 algoritmos multiobjetivo baseados no *Non-dominated Sorting Genetic Algorithm* (NSGA-II) (4).

Os resultados obtidos neste trabalho foram comparados com algoritmos do estado da arte presentes na literatura. Testes estatísticos também foram feitos para comparar o desempenho dos algoritmos propostos. Os resultados gerais demonstraram a eficiência destas abordagens.

Esse trabalho está estruturado em 7 capítulos. No Capítulo 2 definiremos formalmente o problema de coleta e entrega de multiagentes. No Capitulo 3 introduzimos os trabalhos relacionados referentes ao tema. No Capitulo 4 são descritos os principais algoritmos utilizados como base para o desenvolvimento da pesquisa. O Capitulo 5, os métodos e algoritmos propostos utilizados. No Capitulo 6 são apresentados os experimentos computacionais e os resultados encontrados. Finalmente no Capitulo 7 uma discussão conclui o trabalho, com propostas para trabalhos futuros.

2 DEFINIÇÃO DO PROBLEMA

Nesse capítulo são explicados os conceitos e características que determinam o que é o problema MAPD. Na Seção 2.1, o problema MAPD é formulado e também são definidos os objetivos adotados para avaliar a solução e a interpretação da mesma no gráfico de Gantt. Na Seção 2.2, apresentamos os ambientes utilizados nesse trabalho junto com suas características.

2.1 *Multi-Agent Pickup and Delivery* (MAPD)

MAPD pode ser modelado em um grafo não direcionado $G = (V, E)$, em que V representa as posições e E representa as conexões entre as posições nas quais o agente pode se mover, dado um conjunto de k agentes $\{a_1...a_k\}$ e um conjunto de l tarefas $\{\tau_1...\tau_l\}$. Cada agente a_i tem um vértice de início $s_i \in V$, cada tarefa τ_j tem um vértice de início/coleta $p_j \in V$ e um vértice de final/entrega $g_j \in V$ (19).

Um agente pode se mover para um vértice adjacente ou esperar no vértice atual. No total, o agente tem 5 movimentos (4 direções e o ato de aguardar). A distância entre 2 vértices adjacentes é determinada pelo valor 1, sem unidade de medida, e o intervalo necessário para que um agente se desloque de seu vértice atual para um adjacente é de 1 espaço de tempo, ou *timestep*. O caminho do agente é uma sequência do par vértice e tempo, (v, t), representando a posição do agente em cada espaço de tempo.

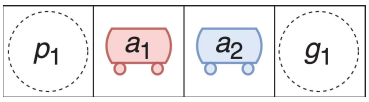
O agente precisa evitar colidir com outros agentes, as colisões são causadas por uma das duas situações que seguem:

- Dois agentes não podem estar na mesma posição no mesmo espaço de tempo. Isso é chamado de conflito de vértice, representado pela tupla $< a_i, a_j, v, t >$, a_i e a_j são agentes no mesmo vértice v no mesmo espaço de tempo t.
- Dois agentes não podem cruzar a mesma aresta no mesmo espaço de tempo. Isso é chamado de um conflito de aresta que é representado pela tupla $< a_i, a_j, u, v, t >$, em que a_i e a_j cruzam a mesma aresta (u, v) em direções opostas entre os tempos t e $t + 1$.

As tarefas são alocadas para os agentes executarem. O alocador de tarefas gera uma sequência de tarefas a serem executadas para cada agente. A tarefa só é considerada concluída quando o agente chega na posição de entrega da tarefa depois de ter visitado a posição de coleta da mesma. As tarefas podem entrar no sistema a qualquer momento e assim que entram no sistema já estão liberadas para serem executadas.

O MAPF pode ser categorizado em dois tipos (16): anônimos e não-anônimos. Nos anônimos, cada tarefa pode ser atribuído a qualquer agente e, sendo assim, os agentes

Figura 1 - Exemplo de uma instância não solucionável do problema MAPD .



Fonte: Adaptado de (18).

podem trocar de tarefas. Nos não-anônimos, cada tarefa só pode ser atribuído a um agente predeterminado. Logo, cada agente já tem um local de destino e portanto os agentes não podem trocar de tarefas.

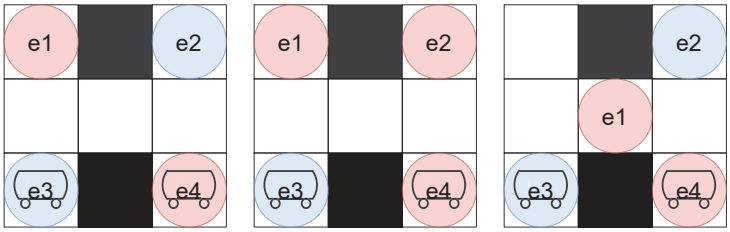
O MAPD, por sua vez, possui estágios anônimos e não-anônimos, tal que os agentes podem estar em dois estados: livres ou ocupados. As tarefas somente podem ser atribuídas a agentes livres. Um agente é dito livre quando não está executando nenhuma tarefa ou quando está se deslocando no caminho para um ponto de coleta de uma tarefa já atribuída a ele. Nesse estado, livre, o agente pode trocar de tarefa caso o algoritmo de alocação o aloque para outra tarefa. Esse estágio do problema é classificado como anônimo. O estágio não-anônimo acontece quando o agente muda para o estado ocupado. O agente torna-se ocupado quando chega na posição de coleta da tarefa e, sendo assim, não pode mais trocar de tarefa, devendo terminar de executar a tarefa atual, deslocando-se até o ponto de entrega da mesma. Quando o agente chega ao ponto de entrega da tarefa, ou seja, conclui a tarefa a que lhe foi atribuída, ele se torna livre novamente e pode receber uma nova tarefa.

Nem todas as instâncias do problema MAPD são solucionáveis. A Figura 1 mostra um exemplo de uma instância do problema MAPD em uma grade 2D com dois agentes livres a_1 e a_2 e uma tarefa τ_1 com vértice de coleta p_1 e vértice de entrega g_1 que é adicionada ao sistema. Os robôs coloridos são os agentes. Os círculos tracejados são vértices de coleta e entrega. Nenhum dos agentes consegue executar a tarefa τ_1, pois o outro agente está impedindo.

Características são definidas para uma instância MAPD ser solucionável, com a ideia de que os agentes só devem poder descansar (ou seja, permanecer indefinidamente em uma determinada posição) em vértices chamados *endpoints*. Um conjunto de *endpoints* é qualquer subconjunto de vértices que contém todos os vértices iniciais de agentes e todos os vértices de coleta e entrega de tarefas, podendo incluir vértices além dos citados. Os vértices de coleta e entrega são chamados de *endpoints* de tarefas. Os outros *endpoints* são chamados de *endpoints* que não estão associados a tarefas. Para uma instância MAPD ser considerada solucionável, ela precisa ser bem definida e, para isso, precisa respeitar os seguintes critérios (19):

- o número de tarefas deve ser finito;

Figura 2 - Três instâncias MAPD.



Fonte: Adaptado de (19).

- o número de *endpoints* que não estão associados a tarefas tem que ser igual ou maior ao número de agentes; e
- deve existir um caminho livre entre dois *endpoints* que não passe por nenhum outro *endpoint*.

A Figura 2 apresenta três exemplos de instâncias MAPD. As células pretas são bloqueadas. Círculos vermelhos são *endpoints* de tarefas. Círculos em azul são *endpoints* que não estão associados a tarefas. A instância da figura da esquerda é bem definida. A instância da figura do meio não é bem definida, pois há dois agentes e somente um *endpoint* que não está associado a tarefas. A instância da figura da direita não é bem definida pois todos os caminhos entre os *endpoints* e2 e e3 passam pelo *endpoint* e1.

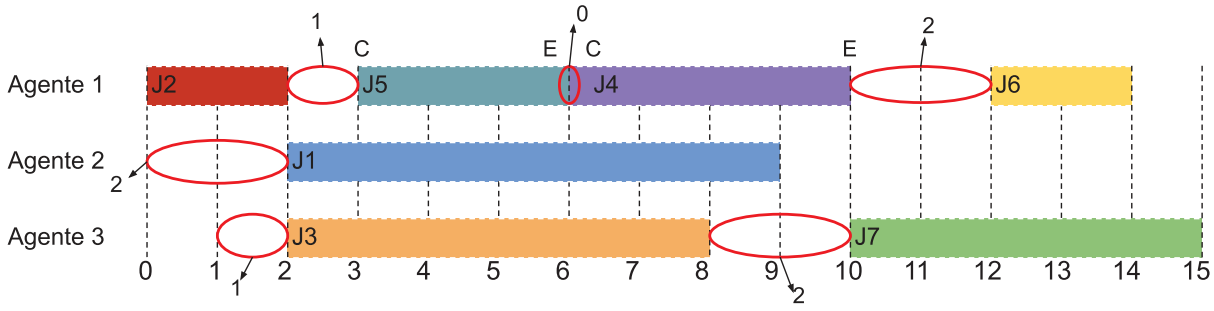
Existem variadas métricas utilizadas para avaliar uma solução. Para o planejamento de caminho os algoritmos são otimizados em relação ao custo dos caminhos (29), como a soma dos custos, do inglês *sum of costs* (SIC) e *makespan*, enquanto na atribuição de tarefas consideram principalmente o *makespan* como o objetivo a ser otimizado. O *makespan* é o tempo que leva para que todas as tarefas sejam entregues, desde a primeira a entrar até a última a ser entregue. Além do *makespan* outra métrica que pode ser utilizada é o *service time*, os trabalhos (19, 8) também utilizam essa métrica como objetivo. O *service time* considera as tarefas individuais, o tempo entre uma tarefa entrar no sistema e aquela tarefa ser entregue.

Neste trabalho para avaliar a solução, deseja-se minimizar duas funções objetivos: *makespan* ($f_{Makespan}$) e *service time* ($f_{Service\ Time}$). *Makespan*, que é considerado aqui o objetivo principal, dado a quantidade de trabalhos que utilizam esse objetivo para avaliar a solução (16, 33). Sua minimização implica que as tarefas serão realizadas o mais rápido possível. A função objetivo pode ser definida como

$$f_{Makespan} = C_{max} = \max_{j}\{C_j\},$$

onde C_j é o instante em que a tarefa j é concluída, $j = 1, ..., l$ e l é o número total de tarefas. *Service time*, o segundo objetivo, é a média da quantidade de espaços de tempo necessária para cada agente terminar de executar a tarefa depois da mesma ter sido

Figura 3 - Gráfico de Gantt de exemplo de uma solução para um problema MAPD.



Fonte: Extraído de (23).

adicionada. Sua minimização resulta nas tarefas sendo executadas tão logo são inseridas no sistema. A função objetivo pode ser definida como

$$f_{Service\ Time} = \frac{1}{l}\sum_{j=1}^{l}(C_j - r_j)$$

onde r_j é o instante de liberação (*release time*) da tarefa j.

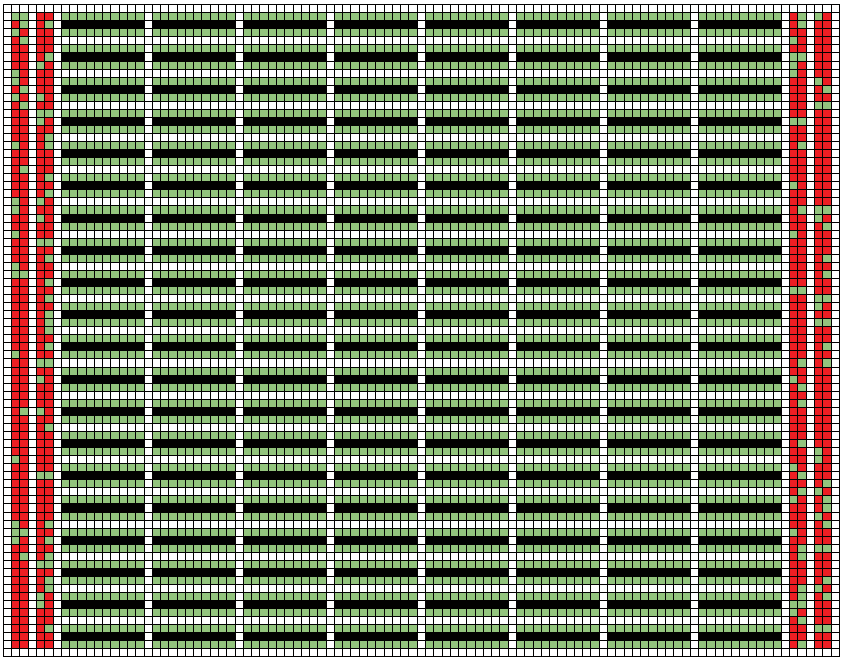
O gráfico de Gantt mostrado na Figura 3 representa um exemplo de solução para um problema MAPD com 7 tarefas e 3 agentes. No gráfico, as barras horizontais representam os espaços de tempo em que os agentes estão executando alguma tarefa, e o eixo vertical representa os agentes aos quais as tarefas foram atribuídas. Os círculos em vermelho representam os espaços de tempo necessários para que os agentes se desloquem de sua posição atual até a posição de entrega da tarefa.

Nesse exemplo, $l = 7$, sendo l o número total de tarefas. As tarefas são denominadas: J1, J2, J3, J4, J5, J6 e J7. O conjunto que contém o espaço de tempo no qual cada tarefa é inserida no sistema é dado por r, sendo, r_j= {0, 0, 1, 2, 2, 3, 4}. No exemplo da Figura 2, as tarefas J2, J5, J4, e J6 foram atribuídas ao Agente 1, a tarefa J1, foi atribuída ao Agente 2, e as tarefas J3 e J7 foram atribuídas ao Agente 3.

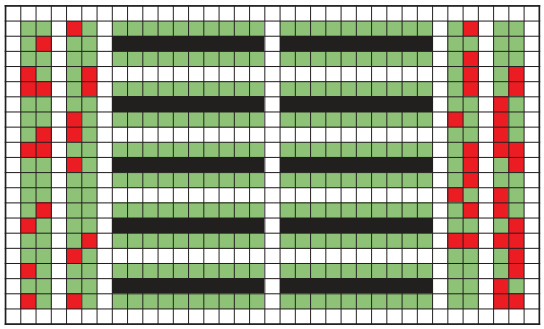
As tarefas começam a ser executadas quando o agente chega na posição de coleta da tarefa. Os círculos em vermelho na figura representam os espaços de tempo necessários para os agentes se deslocarem da sua posição atual até a posição de coleta da tarefa. Nesse exemplo a posição inicial do Agente 1 coincide com a posição de coleta da tarefa J2. Quando o Agente 1 termina a tarefa J2 (chega na posição de coleta da tarefa J2) ele leva 1 espaço de tempo para chegar na posição de coleta da tarefa J5. A posição de entrega da tarefa J5 é a mesma da posição de coleta da tarefa J4. O conjunto que contém o tempo no qual as tarefas são concluídas é dado por C, sendo, C_j = {9, 2, 8, 10, 6, 14, 15}. Nesse exemplo, o *makespan* da solução é 15, sendo este o valor máximo do conjunto C, e o *service time* é calculado como a média das diferenças entre os tempo de conclusão e os tempos de inserção das tarefas no sistema, logo $(9 + 2 + 7 + 8 + 4 + 11 + 11) \div 7 = 7,42$.

Figura 4 - Representação de um ambiente grande e pequeno.

(a) Ambiente grande com 500 agentes



(b) Ambiente pequeno com 50 agentes



Fonte: Adaptado de (19).

2.2 Ambientes

Os testes realizados aqui nos experimentos computacionais foram feitos em dois ambientes: um ambiente pequeno e em um ambiente grande, ambos representados na Figura 4 e definidos em (19). As células em preto são posições bloqueadas, as células em verde são *endpoints* de tarefas e as células em vermelho são os *endpoints* que não estão associados a tarefas. O ambiente pequeno possui tamanho 21 x 35 e é definido para problemas envolvendo 5 diferentes números de agentes: 10, 20, 30, 40 e 50 agentes. Ao todo, 500 tarefas chegam em diferentes frequências (nome dado na literatura em (19) à taxa de inserção de novas tarefas no sistema a cada janela de tempo). As frequências são 0.2, 0.5, 1, 2, 5 e 10. Uma frequência igual a 10 significa que 10 novas tarefas são adicionadas a cada espaço de tempo, enquanto uma frequência de 0.2 indica que é necessário 5 espaços de tempo para cada adição.

O ambiente grande tem como tamanho 81 x 81 células, e é definido para problemas envolvendo 5 diferentes números de agentes: 100, 200, 300, 400 e 500. No total, 1000 tarefas são adicionadas ao sistema com uma frequência fixa de 50, significando que no 20º espaço de tempo as últimas tarefas são adicionadas no sistema e nenhuma outra tarefa será criada depois.

3 TRABALHOS RELACIONADOS

Nesse capítulo, são apresentados trabalhos encontrados na literatura relacionados aos problemas apresentados nesta dissertação. Na Seção 3.1, são apresentados variantes do MAPF existentes. Na Seção 3.2, são apresentados trabalhos e algoritmos utilizados para resolver o problema MAPD.

3.1 *Multi-agent Path Finding*

Existem diferentes variações do MAPF que tentam capturar as características de problemas do mundo real. Em (20), é descrito o "MAPF *with Deadlines*", em que é definido um tempo limite para os agentes chegarem em suas posições finais. Esse tem como objetivo maximizar o número de agentes que chegam na posição final sem colidir uns com os outros. Em (11), o problema é modificado adicionando-se restrições de cinemática ao movimento do agente. O movimento que o agente pode executar depende não apenas de sua localização atual, mas também de parâmetros de estado como velocidade e orientação. Em outra versão do problema, discutida em (31), as ações dos agentes (o movimento de uma célula para outra) podem levar mais do que um espaço de tempo para serem completadas. Essa versão do problema pode ser chamada de "MAPF *with non-unit edge cost*".

Em algumas pesquisas envolvendo o MAPF, os agentes possuem formas geométricas e volumes específicos (13). O fato dos agentes terem volume levanta questões sobre como eles estão situados no grafo e como se movem nele. Em particular, se um agente estiver localizado em um vértice, pode proibir outros agentes de ocupar vértices próximos. Da mesma forma, se um agente se move ao longo de uma aresta, pode proibir outros agentes de se moverem ao longo das interseções ou ficarem em vértices próximos da aresta, o que pode introduzir novos tipos de conflitos.

3.2 *Multi-agent Pickup and Delivery*

Diferentemente do MAPF, nessa variação do problema, denominado MAPD, cada tarefa tem duas posições, e o agente tem que se deslocar para a posição inicial e depois para a posição final para concluir a tarefa. Os agentes também estão sempre recebendo novas tarefas. O MAPD está interessado em escolher melhores sequências de tarefas para cada agente executar, e executar o planejamento de caminho multiagentes (MAPF), até as posições das tarefas.

Um problema MAPD *offline* é tratado em (15), onde todas as tarefas são conhecidas a priori, mas só poderão ser executadas depois de liberadas. Os algoritmos utilizados realizam primeiramente a atribuição de tarefas para cada agente computando uma sequência

de tarefas usando um Problema do Caixeiro Viajante, do inglês *Traveling Salesman Problem* (TSP) especial. Para gerar as sequencias, a alocação de tarefas ignora as colisões e usa as distâncias estimadas com o objetivo de minimizar o *makespan*. O solucionador TFS usado, constrói um grafo ponderado direcionado em que planeja um ciclo hamiltoniano (*loop* fechado através de um grafo no qual cada vértice é visitado apenas uma vez), e, assim, gera as sequencias de tarefas.

No problema MAPD *online*, as tarefas só são conhecidas quando entram no sistema. Logo a alocação de tarefas aos agentes só pode ser feita depois que as tarefas entram no sistema. Os trabalhos a seguir tratam de MAPD *online*.

Em (19), são introduzidos dois algoritmos desacoplados para lidar com o problema MAPD: *Token Passing* (TP) e *Token Passing with Task Swapping* (TPTS). O *token* é um bloco de memória compartilhado no qual todos os caminhos atuais dos agentes, o conjunto de tarefas e as atribuições dos agentes são armazenados. O TP foi usado anteriormente em (2), no qual o algoritmo COBRA foi introduzido. O COBRA resolve o problema MAPF operando em um sistema no qual o usuário pode a qualquer momento obrigar os robôs a se deslocarem de sua localização atual para outra. No entanto, o COBRA não leva em consideração que os locais de coleta ou entrega possam ser ocupados por outros robôs que não estejam executando uma tarefa, resultando em *deadlocks* (19).

O TPTS torna o TP mais eficiente, permitindo que os agentes troquem de tarefas. Se o sistema identificar algum benefício na otimização dos objetivos, os agentes podem então trocar de tarefas desde que a nova tarefa ainda não esteja sendo executada por outro, ou seja, desde que nenhum agente tenha chegado ao seu ponto de coleta. A lista de tarefas que cada agente pode executar deixa de ser relacionada às tarefas não atribuídas e passa a observar todas as tarefas ainda não executadas.

O desempenho de ambos os algoritmos (TP e TPTS) é comparado a um algoritmo denominado CENTRAL, também desenvolvido em (19). Se tratando de um algoritmo centralizado, é possível uma maior comunicação entre os agentes. Desta forma, o CENTRAL gera soluções de melhor qualidade, mas requer mais tempo de processamento. O CENTRAL começa com a atribuição de *endpoints* a todos os agentes e, em seguida, resolve o problema MAPF, no qual cada agente deve se mover para seus *endpoints* atribuídos usando um algoritmo acoplado, o CBS (26). Deve-se notar que a parte MAPF tornou-se significativamente mais eficiente dividindo-a em duas etapas: primeiro, planejando caminhos para os agentes que receberam uma tarefa na etapa de tempo atual (enquanto trata os outros agentes como objetos em movimento); e depois, planejando os caminhos dos agentes livres para seus terminais atribuídos (novamente ao tratar os outros agentes como objetos em movimento).

Como o TPTS e o CENTRAL não permitem operações em tempo real, esses dois algoritmos não funcionam para ambientes grandes com muitos agentes. O TP mostrou ser

escalável, mantendo os tempos de execução baixos e resolvendo todos os problemas de MAPD bem formados.

Os algoritmos descritos acima (TP, TPTS e CENTRAL) não levam em consideração a parte da alocação de tarefas para os agentes, levando isso em consideração, foram propostos novos algoritmos que tratam dessa parte.

Um algoritmo *multi-label* A* para atribuir agentes disponíveis às tarefas foi introduzido em (8) junto com uma nova heurística centralizada para planejar o caminho dos agentes até as tarefas (HBH). Os resultados do algoritmo (HBH+MLA*) foram comparados com TP, e obtiveram uma melhora significativa na qualidade da solução, reduzindo na média o *makespan* em 30% e o *service time* em 43% e também no tempo de processamento, que reduziu em 98%.

4 MATERIAIS E MÉTODOS

O problema MAPD é dividido em dois sub-problemas: planejamento de caminho e alocação de tarefas. Neste capítulo são descritos os métodos utilizados como base para as propostas deste trabalho. Na Seção 4.1 são descritos os algoritmos para o planejamento de caminho. Na Seção 4.2 um Algoritmo Genético e o algoritmo multiobjetivo NSGA-II são apresentados. Esses são utilizados para realizar a alocação de tarefas aos agentes.

4.1 Algoritmos de Planejamento de Caminho

Os algoritmos MAPF são divididos em duas categorias (26): acoplados e desacoplado. Nas abordagens acopladas, os caminhos dos agentes são planejados todos juntos de uma só vez, de modo que as soluções de todos os agentes dependem um do outro. Isso significa que todo o espaço de busca deve ser percorrido para encontrar uma solução viável para todos os agentes. Devido a isso, o custo de encontrar uma solução cresce exponencialmente com o número de agentes. Já nas abordagens desacopladas, o problema é decomposto em vários subproblemas, encontrando caminhos para cada agente separadamente. A escolha entre essas duas abordagens resulta em um impacto no tempo de execução do algoritmo e na qualidade da solução. Nas próximas seções são apresentados os algoritmos usados aqui para solucionar os problemas MAPF: *Prioritized Planning* e o CBS. O primeiro, atua de forma desacoplada. O segundo, funciona em duas partes, uma de forma acoplada e outra desacoplada, combinando as características das duas abordagens.

4.1.1 *Prioritized Planning*

Prioritized Planning é um algoritmo que utiliza o *Cooperative* A* (27), que funciona criando uma lista de prioridades para os agentes, de forma que aqueles que apresentam maiores prioridades são os primeiros a executar o planejamento de caminho (6, 1). A lista é baseada na distância da posição atual de cada agente até a posição final predeterminada que o agente deve chegar. Agentes com a maior distância tem maior prioridade, resultando em um número menor de restrições no planejamento de caminhos. Com isso, é esperado um valor menor de *makespan*, visto que agentes com o maior caminho (maior custo) estão planejando o caminho primeiro e, assim, tendo um menor número de colisões no planejamento. Dessa forma, o custo da posição atual até a posição final fica próximo do custo estimado (15).

O *Prioritized Planning* utiliza o *Cooperative* A*, que cria uma tabela de reservas salvando o caminho de cada agente. Essa tabela tem a localização e o tempo de cada agente. Desse modo, cada agente ao executar o A* para encontrar o caminho entre dois pontos verifica se determinada posição está disponível num determinado tempo: caso

esteja, o agente estará livre para se mover para a posição; caso contrário, o agente deve encontrar outro caminho.

A* é um algoritmo de planejamento de caminho muito utilizado para encontrar o menor caminho entre dois pontos em um mapa conhecido (9). O algoritmo funciona mantendo uma lista de vértices chamada "Abertos", que inicialmente contém somente o vértice inicial. A cada iteração, o A* determina qual vértice vai ser escolhido, formando o caminho. O vértice v com o menor valor $f(v)$ é selecionado, no qual $f(v) = g(v) + h(v)$, é a soma do custo do caminho onde $g(v)$ é o custo do caminho até o vértice v, e $h(v)$ é a função heurística, que é o custo estimado necessário para chegar no vértice destino a partir do vértice v. O vértice v selecionado é removido e extendido. Extender significa que todos os vértices adjacentes são visitados e adicionados a lista "Abertos" caso já não tenham sido adicionados na lista. A* termina quando chegar no vértice de destino, ou não existir mais nenhum vértice para ser extendido.

4.1.2 *Conflict-Based Seach* (CBS)

Conflict-Based Seach (CBS) (26) é um algoritmo dividido em dois níves. No alto nível, é utilizada uma *constraint tree* (CT), ou árvore de restrições, com os nós incluindo localização e instante de tempo. No baixo nível, é realizada, para cada um desses nós, uma busca a fim de encontrar novos caminhos até o destino, levando em conta as restrições impostas pelo alto nível. O CBS tem ordem de complexidade computacional exponencial em relação ao número de conflitos (26).

A ideia por trás do CBS é gerar restrições para caminhos conflitantes para que, ao atender essas restrições, os novos caminhos gerados não entrem em conflito. O alto nível encontra esses conflitos e adiciona restrições para cada um dos agentes. O nível baixo atualiza os caminhos levando em consideração os conjuntos de restrições dos agentes para que conflitos sejam evitados.

O pseudo-código do alto nível do CBS é mostrado no Algoritmo 1. No alto nível, o CBS pesquisa uma árvore de restrições. Um CT é uma árvore binária e cada nó N no CT contém os seguintes dados:

- Um conjunto de restrições (N.restrições). A raiz do CT contém um conjunto vazio de restrições. O filho de um nó no CT herda as restrições do pai e adiciona uma nova restrição para um agente.
- Uma solução (N.solucao). Um conjunto de k caminhos, sendo um caminho para cada agente. O caminho para o agente a_i deve ser consistente com as restrições de a_i. Tais caminhos são encontrados pelo baixo nível.
- O custo total (N.custo) da solução atual. O SIC (*sum of the costs*) é a soma do custo de caminho de todos os agentes.

Algoritmo 1 CBS no alto nível.

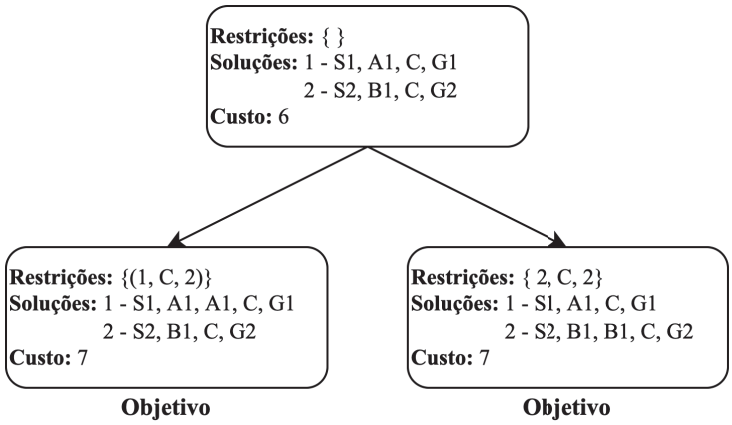
1: R.restricoes ← 0
2: R.solucao ← encontrar caminhos para cada agente individualmente usando a busca em baixo-nível()
3: R.custo ← SIC(R.solucao)
4: Insere R em "Abertos"
5: **enquanto** "Abertos" não está vazia **faça**
6: $P \leftarrow$ o nó com o menor custo de solução
7: Valida os caminhos de P até encontrar um conflito
8: **se** P não tem nenhum conflito **então**
9: retorna P.solucao
10: **fim se**
11: C ← primeiro conflito (a_i, a_j, v, t) em P
12: **para** todo agente a_i em C **faça** ▷ Para cada agente a_i envolvido na colisão C
13: A ← novo nó
14: A ← P.restricoes $\cup$ (a_i, s, t) ▷ A recebe as restrições herdadas do nó P, mais a nova restrição de C
15: A ← P.solucao
16: Atualiza A.solucao chamando a busca em baixo-nivel(a_i)
17: A.custo ← SIC(A.solucao)
18: Insere A em "Abertos"
19: **fim para**
20: **fim enquanto**

O nó N no CT é um nó final quando N.solucao é válido, ou seja, o conjunto de caminhos para todos os agentes não possui conflitos. O nível superior realiza uma busca onde os nós são ordenados por seus custos. O nó com o menor custo é a solução.

Dada a lista de restrições para um nó N do CT, chama-se a busca de baixo nível. O baixo nível recebe um agente, a_i, e um conjunto de restrições associadas ao agente. O A* é utilizado para encontrar um caminho ótimo para o agente a_i que satisfaça todas as suas restrições. Essa busca retorna o caminho com o menor custo para cada agente, a_i, que é consistente com todas as restrições associadas a a_i no nó N. Uma vez que um caminho consistente foi encontrado para cada agente em relação às suas restrições, esses caminhos são validados em relação aos outros agentes para verificar se novas colisões ocorreram. A validação é realizada simulando o conjunto de k caminhos. Se todos os agentes atingirem seu objetivo sem nenhum conflito, esse nó CT N é declarado como um nó final e a solução atual (N.solucao) que contém esse conjunto de caminhos é retornada. Se, no entanto, durante a validação for encontrado um conflito C = (a_i, a_j, v, t) para dois ou mais agentes a_i e a_j, a validação é interrompida e o nó é declarado como um nó não final.

Um exemplo de CT é mostrado na Figura 5, onde o agente a1 tem como objetivo chegar na posição G1 e o agente a2 tem como objetivo chegar na posição G2. É feita uma busca em baixo nível com um conjunto vazio de restrições, que retorna uma solução ótima para cada agente (linha 2 do Algoritmo 1) , < S1, A1, C, G1 > para a1 e < S2,

Figura 5 - Exemplo de uma *constraint tree* (CT), onde um conflito ocorre no vértice C.



Fonte: Adaptado de (26).

B1, C, G2 > para a2. Assim, o custo total deste nó é 6. Esta informação é mantida dentro deste nó. A raiz é então inserida na lista "Abertos" e será expandida em seguida. Ao validar a solução de dois agentes dada pelos dois caminhos individuais (linha 7) um conflito é encontrado quando ambos os agentes chegam ao vértice C no passo de tempo 2. Isso cria o conflito (a1, a2, C, 2). Como resultado, a raiz é declarada como não final e dois filhos são gerados para resolver o conflito (Linha 11). O filho da esquerda adiciona a restrição (a1, C, 2) enquanto o filho da direita adiciona a restrição (a2, C, 2). A busca de baixo nível é agora invocada (Linha 16) para encontrar um caminho ótimo que também satisfaça a nova restrição. Para o filho da esquerda, a1 deve esperar um passo de tempo em A1 e o caminho < S1, A1, A1, C, G1 > é retornado para a1. O caminho para a2, < S2, B1, C, G2 > permanece inalterado para o filho à esquerda. O custo total para o filho esquerdo agora é 7. De forma semelhante, o filho à direita é gerado, também com custo 7. Ambos os filhos são adicionados a "Abertos" (Linha 18). Na etapa final, o filho à esquerda é escolhido para expansão e os caminhos subjacentes são validados. Como não existem conflitos, o filho da esquerda é declarado como um nó final (Linha 9) e sua solução é retornada como uma solução ótima.

4.1.3 Controle de futuras colisões

O algoritmo de planejamento de caminho não faz distinção entre um ponto de coleta e um ponto de entrega de uma tarefa, considerando apenas a posição final que o agente deve atingir. Dessa forma, faz-se uma busca separada, chamando o algoritmo de planejamento de caminho duas vezes. Primeiro é utilizado o algoritmo para encontrar um caminho da posição atual do agente até o ponto de coleta. E depois, quando o agente chega na posição de coleta, utiliza o algoritmo para encontrar um caminho até a posição de entrega. Quando o agente chega na posição final, seja ela posição de coleta ou entrega, é necessário alocá-lo para outro local no qual: (i) pode permanecer indefinidamente; (ii)

Algoritmo 2 Pseudo-código do algoritmo de planejamento de caminho - PP-E.

```
1: Ordena lista de agentes em ordem descendente de acordo com a distância estimada da
   posição do atual do agente até a posição final definida anteriormente;
2: para todo agente a_i na lista ordenada faça
3:     para todo agente k ≠ a faça
4:         se agente k está bloqueando a posição p do agente a então
5:             Mova agente k para o endpoint mais próximo
6:         fim se
7:     fim para
8:     Obtém um caminho usando versão modificada do A* (a, p) (Algoritmo 3)
9: fim para
10: Returna a solução encontrada
```

pode receber uma nova tarefa (ir para o ponto de coleta da tarefa); ou (iii) continuar executando a tarefa a que está atribuído (ir para o ponto de entrega da tarefa).

Para evitar futuras colisões e possíveis *deadlocks*, depois de executado o algoritmo de planejamento de caminho, o agente volta a sua posição inicial, ou seja, posição de descanso, para "permanecer indefinidamente" como utilizado em (15). Isto evita futuras colisões, porém é muito custoso, tanto computacionalmente quanto em relação ao *makespan*, e algumas vezes desnecessário já que o planejamento desse caminho não é utilizado, uma vez que o agente já pode ter uma nova posição de destino. Essa posição de destino pode ser: a posição de entrega da tarefa que o agente esta executando, ou uma posição de coleta de uma nova tarefa, visto que ele esta sempre recebendo novas atribuições.

Outra abordagem, proposta em (8), é não realizar a busca separada. O algoritmo faz uma busca conjunta da posição de coleta da tarefa para posição de entrega da mesma. Assim sendo, evita que o agente vá para outra posição já sabendo que a sua próxima posição final é o ponto de entrega da tarefa.

Na fase em que o agente ainda não chegou na posição de coleta da tarefa, ele permanece livre. Esta busca conjunta também planeja caminhos desnecessariamente, visto que, com tarefas entrando no sistema a todo momento, o alocador de tarefas pode atribuir uma nova tarefa e assim ter que calcular um novo caminho para a nova posição de coleta.

Foi utilizada uma abordagem similar para o algoritmo *Prioritized Planning* chamado de PP-E. O pseudo-código do PP-E é mostrado no Algoritmo 2. Neste algoritmo, antes de executar o planejamento de caminho, é verificado se há outros agentes impedindo o determinado agente de chegar em sua posição objetivo (que pode ser coleta ou entrega da tarefa). Caso tenha, esse agente vai para o *endpoint* livre mais próximo (linha 4 e 5). *Endpoint* livre é quando nenhum agente tem como posição final (permanecer indefinidamente) este *endpoint*.

Uma versão modificada do A* é chamado na linha 8 para encontrar o caminho do agente entre dois pontos. No Algoritmo 3, depois de executado o algoritmo de planejamento

Algoritmo 3 A* Modificado - PP-E.

1: Cria um nó inicial da posição inicial e adiciona o nó a lista “Abertos”;
2: **enquanto** “Abertos” não está vazia **faça**
3: Obtém o nó com o menor valor de f da lista “Abertos”
4: **se** nó n == posição final **então**
5: **se** agente a consegue ir da sua posição final para um *endpoint* de tarefa livre **então**
6: Retorna o caminho planejado da posição inicial para a posição final + o caminho para a posição final + o caminho para um *endpoint* livre;
7: **se não**
8: Continua
9: **fim se**
10: **fim se**
11: Expande nó n para os nós adjacentes. Para cada nó adjacente, se a posição não está bloqueada, calcula g e h e adiciona esse nó a lista “Abertos”, se já não existir na lista;
12: **fim enquanto**

de caminho, o agente não volta para sua posição de descanso (linha 5 e 6), deslocando-se para o *endpoint* livre mais próximo. Esse *endpoint* pode ser até mesmo a própria posição objetivo do agente.

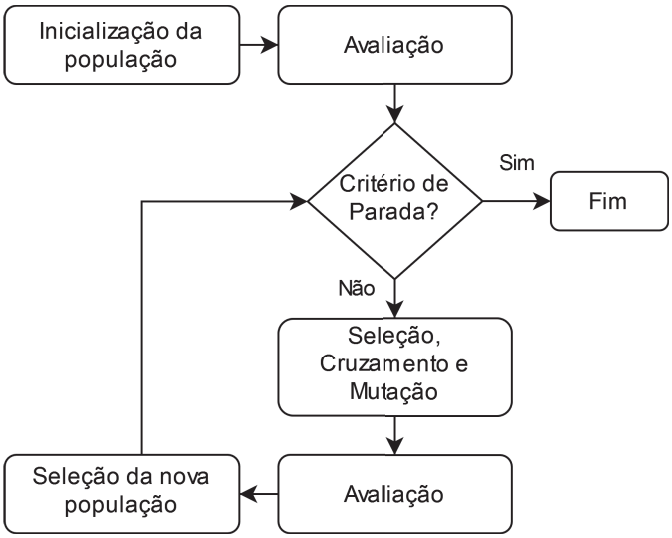
4.2 Algoritmos Genéticos

Algortimos Genéticos (AGs) são técnicas de otimização global baseadas nos mecanismos de seleção natural e da genética (10). Esses algoritmos são muito utilizados em problemas de otimização de tarefas (2, 14), já que permitem uma maior exploração do espaço de busca. Os AGs se baseiam em evoluir um conjunto de soluções, que são os indivíduos (codificados na forma de cromossomos) com o intuito de encontrar novas e melhores soluções a partir das existentes.

A Figura 6 fornece uma visão geral da AG. Os AGs inicializam gerando indivíduos que são as soluções para o problema tratado. Esta pode ser gerada aleatoriamente ou através de uma heurística de inicialização. Esses indivíduos são então inseridos na população e as soluções são avaliadas através de uma função objetivo. Com base nesse valor de aptidão gerado, seleciona-se os melhores indivíduos, denominados progenitores, para aplicar os operadores de cruzamento e mutação, assim dando origem a novos indivíduos denominados filhos. Os novos indivíduos também são avaliados e inseridos dentro da população, substituindo os indivíduos com menor valor de aptidão. O ciclo termina quando é satisfeito o critério de parada. As etapas de uma AG são:

- Cruzamento: selecionado dois indivíduos, as características desses são combinadas gerando outros dois indivíduos;

Figura 6 - Representação esquemática do fluxo de um Algoritmo Genético.



Fonte: Adaptado de (10).

- Mutação: aplica-se uma pequena modificação no indivíduo gerando uma nova solução;
- Avaliação: determina um valor de aptidão para cada indivíduo e este valor é baseado na função objetivo do problema;
- Seleção: seleciona os indivíduos nos quais serão aplicados os operadores de cruzamento e mutação, assim, gerando os filhos. Também são selecionados os indivíduos que permanecerão na população. Essas seleções são normalmente feitas com base no valor de aptidão de cada indivíduo.

Para selecionar os indivíduos nos quais será feito o cruzamento, é utilizada aqui a seleção por torneio. Nela um grupo de indivíduos são selecionados aleatoriamente da população e o indivíduo com melhor valor de aptidão nesse grupo é escolhido para ser o primeiro progenitor. O processo é repetido para selecionar o segundo progenitor e esses dois indivíduos passam pelo processo de cruzamento.

Na AG, cada indivíduo representa uma solução para o problema descrito. Nessa dissertação, temos duas representações de solução. A primeira solução é representada por um vetor de tarefas e um vetor de agentes. Cada tarefa está associada a um agente na ordem que aparecem, como mostrado na Figura 7, que exemplifica um problema com 10 tarefas e 5 agentes. Cada agente tem uma lista de tarefas a ser executada na ordem determinada pelo posicionamento da tarefa no vetor. No exemplo mostrado na Figura 7 o Agente 3 executa a tarefa 7 e depois a tarefa 5; Agente 2 = {1, 4, 6} o Agente 2 executa a tarefa 1, seguida pela tarefa 4 e depois a tarefa 6; Agente 0 = {3}; Agente 4 = {9, 8, 2}. A segunda representação é formada somente pelo vetor de tarefas, e a alocação das tarefas ao agente fica por conta de um escalonador.

Figura 7 - Representação de um solução candidata.

Tarefas	7	1	3	9	5	4	8	10	2	6
Agentes	3	2	0	4	3	2	4	1	4	2

Nessa dissertação, foi utilizado dois operadores para o cruzamento, são eles, o *one-point crossover* e *order one crossover*, e um operador o para a mutação o *pairwise interchange neighborhood mutation* (32).

Os operadores de cruzamento e mutação são feitos em cada vetor separadamente. No vetor de tarefas é feita uma permutação no cruzamento, *order one crossover*, e no vetor de agentes um *one point crossover*.

Para a mutação no vetor de tarefas é usado o *pairwise interchange neighborhood mutation* e, dado duas posições, as tarefas são trocadas de posição. Para o vetor de agentes é escolhida uma posição aleatoriamente no vetor e seu valor é trocado por outro agente escolhido aleatoriamente. Essa escolha aleatória é realizada com uma variação de valor α em uma lista circular com o número total de agentes. Por exemplo: considerando 10 agentes e a posição escolhida sendo o agente 5, o agente é trocado por um agente entre 2 e 8. Os operadores utilizados estão descritos abaixo:

- *One-point crossover* seleciona aleatoriamente um ponto de corte. Os agentes são copiadas do Progenitor 1 para o filho até esse ponto e, o resto dos genes vazios são completados com o restante das tarefas do Progenitor 2 (localizadas após o ponto de corte) como demonstrado na Figura 8. As cores azul e vermelho representam os genes copiados do Progenitor 1 e Progenitor 2, respectivamente. O ponto de corte, neste caso, é o terceiro gene.

- *Order one crossover* (OX) seleciona dois pontos de corte do Progenitor 1 e essa *substring* é copiada para o filho. Esses valores são marcados no Progenitor 2 para não serem copiados e, os genes vazios são copiados do Progenitor 2 na ordem em que aparecem. A Figura 9 mostra um exemplo de recombinação em um vetor de tarefas. A cor azul representa os genes copiados do Progenitor 1 e marcados no Progenitor 2, a cor vermelha representa os genes restantes, copiados do Progenitor 2.

- *Pairwise interchange neighborhood mutation* (IM) seleciona dois pontos aleatoriamente e suas posições são trocadas no cromossomo, como mostrados na Figura 10. As cores azul e vermelho indicam os genes que serão trocados entre si.

4.2.1 NSGA-II

A otimização multiobjetivo corresponde à situação em que se tem mais de um objetivo a ser otimizado. Não existe uma única solução ótima que otimiza todos os

Figura 8 - Exemplo de *one-point crossover* no indivíduo.

Progenitor 1	3	2	0	4	3	2	4	1	4	2
Progenitor 2	1	0	5	4	3	4	2	3	2	1
Filho 1	3	2	0	4	3	4	2	3	2	1

Fonte: Adaptado de (32).

Figura 9 - Exemplo de *order one crossover* no indivíduo.

Progenitor 1	7	1	3	9	5	4	8	10	2	6
Progenitor 2	~~9~~	10	~~5~~	~~3~~	1	~~4~~	6	8	2	7
Filho 1	10	1	3	9	5	4	6	8	2	7

Fonte: Adaptado de (32).

Figura 10 - Exemplo de *pairwise interchange neighborhood mutation* no indivíduo.

Progenitor 1	7	1	3	9	5	4	8	10	2	6
Filho 1	7	1	8	9	5	4	3	10	2	6

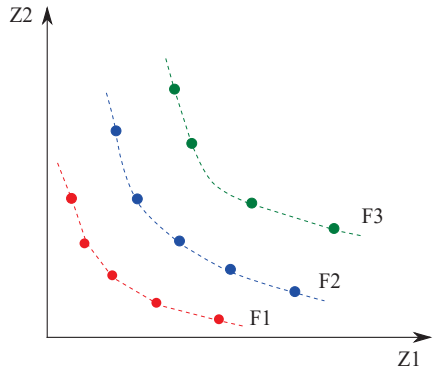
Fonte: Adaptado de (32).

objetivos ao mesmo tempo, dado que os objetivos são conflitantes. Cabe a um tomador de decisão selecionar qual solução escolher, considerando por exemplo os pesos dos objetivos a serem otimizados.

Nondominated Sorting Genetic Algorithm II (NSGA-II) é um algoritmo multiobjetivo bastante conhecido na literatura e proposto em (4), que utiliza uma classificação por dominância chamada de Ordenação de Pareto ou *Non-Dominated-Sorting*. Os indivíduos de uma população P são classificados em d subconjuntos conforme seu grau de dominância, tais que os indivíduos do primeiro posto tem preferência de seleção sobre os demais, $F_1 =$ { todos os indivíduos não-dominados de P} (*rank* 1). O mesmo acontece com os indivíduos do segundo *front* (f2) que tem preferência de seleção sobre os demais, exceto os indivíduos do primeiro *front* (f1), $F_2 = \{$todos os indivíduos não-dominados de $P \setminus F_1\}$(*rank* 2), ... $F_d = \{$todos os indivíduos não-dominados de $P \setminus \{F_1 \cup F_2 \cup ... \cup F_{d-1}\}$ $\}$ (*rank* d), onde $P = F_1 \cup F_2 \cup ... \cup F_d$. A Figura 11 mostra os indivíduos de uma população e suas determinadas fronteiras (f1, f2, f3), é assumido na figura que ambos os objetivos devem ser minimizados.

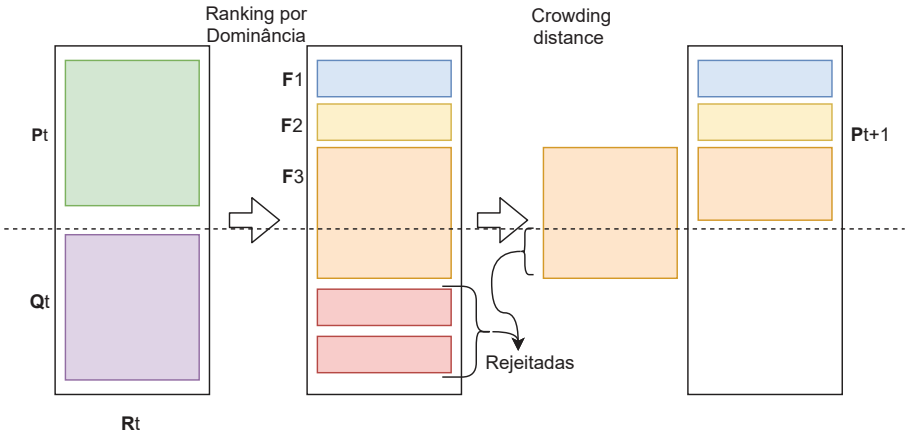
Na primeira iteração, existe uma população inicial P_0 de tamanho N. Após aplicar os operadores de cruzamento e mutação uma nova população filha Q_0 é gerada, do mesmo tamanho N. As duas populações são unidas em um conjunto R_0 de tamanho 2N e submetidas à classificação por dominância, gerando as fronteiras $F_1, F_2, ...F_d$. Para gerar a nova população P_1 com tamanho N são selecionados os melhores indivíduos de acordo com a sua classificação por dominância. Indivíduos que possuem os postos mais baixos continuam na população e indivíduos com postos maiores, que são mais dominados, são

Figura 11 - Exemplo da Ordenação de Pareto.



Fonte: Adaptado de (3).

Figura 12 - Procedimento de substituição do NSGA-II.

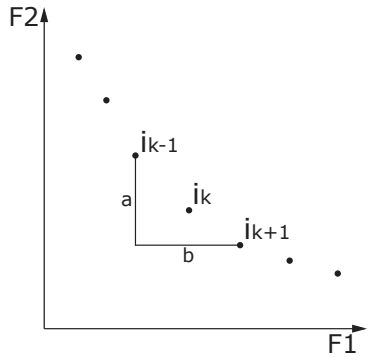


Fonte: Extraído de (23).

descartados. Quando for necessário selecionar um indivíduo entre 2 ou mais dentro do mesmo posto, o indivíduo com maior valor de *crowding distance* (CD) terá preferencia. A Figura 12 mostra o esquema de seleção do NSGA-II.

O *crowding distance* utiliza como métrica o semiperímetro do cubóide formado pelos vizinhos mais próximos, como mostrado na Figura 13. Quanto maior o cubóide formado (valor do CD), mais distante a solução se encontra de suas soluções vizinhas, o que aumenta sua preferência na seleção para a próxima geração da população. Para manter as soluções candidatas bem distribuídos ao longo da frente de Pareto os indivíduos extremos recebem um valor infinito em seu CD. O Algoritmo 4 calcula o valor do CD de cada indivíduo da população. $\mathcal{F}$ representa uma fronteira composta por $N = |\mathcal{F}|$ indivíduos. M é o número de funções objetivos, e $\mathcal{F}[i].m$ denota o valor da função objetivo m do indivíduo i no *rank* $\mathcal{F}$. Parâmetros f_m^{max} é o máximo valor para o objetivo m e o f_m^{min} é o mínimo.

Figura 13 - Cálculo do *Crowding Distance* do indivíduo i_k, CD(i_k) = a+b.



Fonte: Adaptado de (3).

Algoritmo 4 Cálculo do *Crowding Distance.*

$N = |\mathcal{F}|$
para $i = 1 \ldots N$ **faça**
    $\mathcal{F}[i]_{dist} = 0$
fim para
para $m = 1 \ldots M$ **faça**
    Orderna($\mathcal{F}, m$);
    $\mathcal{F}[1]_{dist} = \mathcal{F}[N]_{dist} = \infty$
    para $i = 2 \ldots N - 1$ **faça**
        $\mathcal{F}[i]_{dist} = \mathcal{F}[i]_{dist} + \frac{\mathcal{F}[i+1].m - \mathcal{F}[i-1].m}{f_m^{max} - f_m^{min}}$
    fim para
fim para

5 MÉTODOS PROPOSTOS

Neste capítulo são descritos os métodos propostos, e suas características. Na Seção 5.1 é apresentada uma descrição de como funciona a entrada de tarefas e a alocação das tarefas aos agentes, até chegar na parte do planejamento de caminho. Para isso apresentamos um algoritmo e seu fluxograma, para descrever os vários componentes utilizados em cada etapa. Na Seção 5.2 apresentamos a alocação de tarefas que é feita por meio de algoritmos genéticos. Na Seção 5.3 apresentamos a heurística de inicialização da população utilizada, e nas seções seguintes descrevemos as diferenças entre as abordagens de algoritmos genéticos mono-objetivos utilizados. Na Seção 5.8 discorremos a análise dos objetivos usados para avaliar o problema e, por fim, na Seção 5.9 apresentamos os algoritmos multiobjetivos que utilizam o NSGA-II.

5.1 Método Geral

O Algoritmo 5 mostra o pseudo-código do algoritmo geral e a Figura 14 apresenta o fluxograma do algoritmo. O algoritmo começa com tempo $t = 0$ e verifica se alguma tarefa entrou no sistema (linha 4). Se alguma tarefa foi adicionada ao sistema, é chamado o alocador de tarefas. Na primeira iteração, todos os agentes estão definidos como livre, com sua posição sendo sua posição inicial e com o tempo final sendo o mesmo que o tempo t que na primeira iteração é 0. Se o agente está definido como ocupado, sua posição final é definida como a posição final da tarefa que ele está executando e o instante de tempo que o agente está é definido como o tempo necessário pra o agente chegar na posição de entrega da tarefa.

O algoritmo verifica, para cada agente, se o dado agente terminou de executar a tarefa que está executando (linha 8). Se o agente tiver terminado, o estado do agente muda, sendo agora definido como livre, e dessa forma, o agente já pode começar a executar a próxima tarefa na lista definida pelo alocador de tarefas. A posição final dele é definida como a posição da coleta da tarefa e o agente é adicionado na lista de agentes para executar o algoritmo de planejamento de caminho. Se o agente chegou na posição de coleta da tarefa em que ele está atribuído (linha 15), este agente é então definido como ocupado, o que significa que ele deve terminar de executar a tarefa atual (chegar na posição de entrega da tarefa). Sua posição final é definida como a posição de entrega da tarefa e o agente é adicionado na lista de agentes para fazer o planejamento de caminho. O algoritmo de planejamento de caminho é aplicado aos agentes que estão na lista, a fim de encontrar um caminho da posição do agente no tempo t até a posição final do agente. Em cada iteração $t = t + 1$, e todos os agentes se deslocam para a posição que foi definida no planejamento de caminho. Com cada agente com uma nova posição e tempo final, o processo se repete até todas as tarefas terem sido executadas e não ter mais tarefas a serem adicionadas no

Algoritmo 5 Pseudo-código do algoritmo geral.

1: $t = 0$
2: **enquanto** o critério de parada não é atendido **faça**
3: **se** tarefa entrou no sistema **então**
4: Adiciona tarefa no conjunto de tarefas
5: Obtém a sequencia de tarefas para cada agente usando um alocador de tarefas. ▷ Entre os algoritmos pode ser um desses: GA-TA, GA-E, *Elitist*, *Steady State*, NSGA-M, NSGA-ST e NSGA-O
6: **fim se**
7: **para** todo agente a_i **faça**
8: **se** agente a_i terminou de executar a tarefa **então**
9: Define agente a_i como livre
10: Define a posição final do agente a_i como a posição de coleta da próxima tarefa.
11: Adiciona agente a_i na lista
12: **fim se**
13: **fim para**
14: **para** todo agente a_i **faça**
15: **se** agente a_i chegou na posição de coleta da tarefa **então**
16: Define agente a_i como ocupado
17: Define a posição final do agente a_i como a posição de entrega da tarefa, que o agente está executando.
18: Remove tarefa do conjunto de tarefas
19: Adiciona agente a_i na lista
20: **fim se**
21: **fim para**
22: Chama um algoritmo de planejamento de caminho ▷ Entre os algoritmos pode ser um desses: *Prioritized Planning*, PP-E, CBS (Algoritmo 1).
23: $t = t + 1$
24: **fim enquanto**

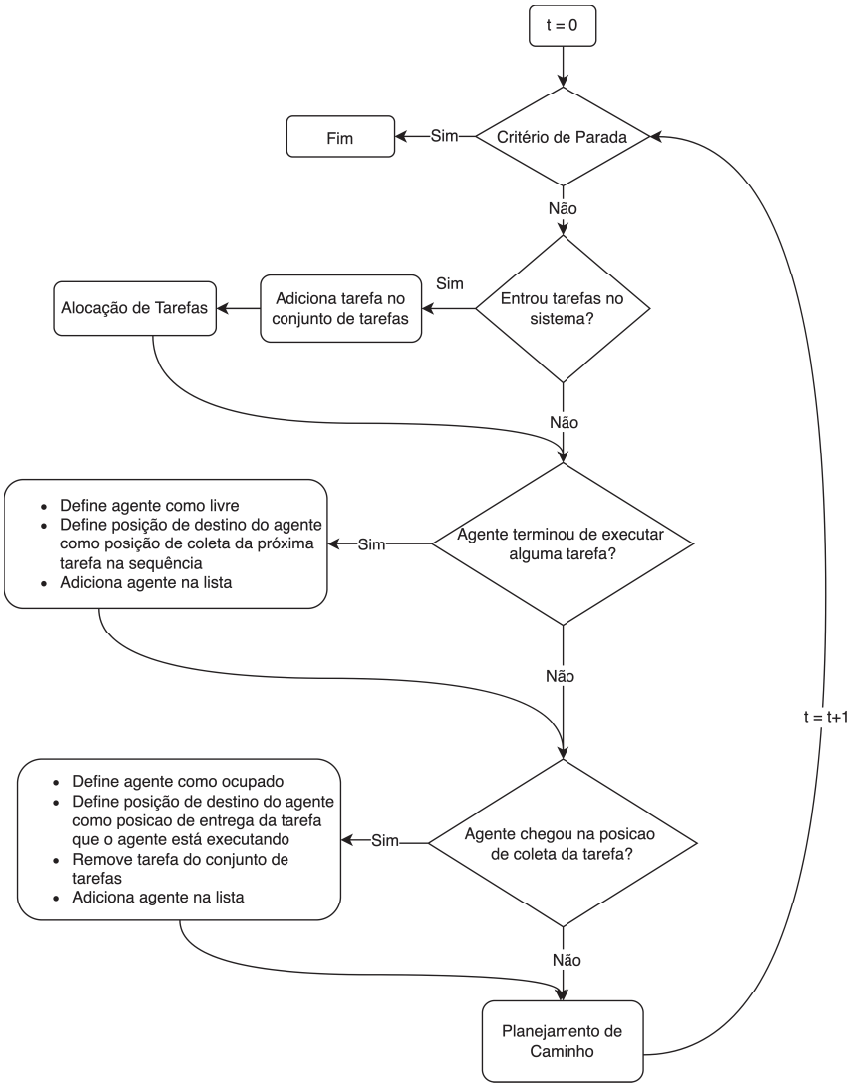
sistema.

5.2 Alocação de Tarefas

A alocação de tarefas é feita aqui por meio de AGs, que buscam pela melhor sequência de tarefas para cada agente executar. Para a função objetivo do algoritmo genético, foi utilizado um valor estimado do tempo gasto da posição do agente até a posição final (custo do caminho), não considerando as possíveis colisões, que só ocorrem depois que o planejamento de caminhos é chamado.

O algoritmo genético, que é o alocador de tarefas, é chamado toda vez que entra uma nova tarefa no sistema e, assim, utiliza alguma abordagem dos algoritmos genéticos propostos. Quando o alocador de tarefas é chamado (linha 5 do Algoritmo 5), cada agente recebe uma posição e um tempo final. Para os agentes livres, a posição é a posição atual e o tempo final é o tempo t. Para os agentes ocupados, a posição é a posição de entrega da

Figura 14 - Fluxograma do método geral.



Fonte: Elaborado pelo autor (2022).

tarefa que o agente está executando, e tempo final é o tempo no qual o agente chegará nessa posição. Dessa forma, quando o algoritmo genético gerar a melhor sequência para cada agente, o mesmo irá considerar a posição e o tempo que o agente estará livre pra poder receber uma nova tarefa. Com isso, o algoritmo genético consegue calcular o melhor caminho levando em conta que os agentes que estão com o status definido como ocupado só estarão livres quando chegarem ao ponto de entrega da tarefa que estão executando.

Foram testadas diferentes abordagens de algoritmo genético mono-objetivo: GA-TA, um algoritmo genético geracional; GA-E, que utiliza um escalonador para fazer a alocação das tarefas para os agentes; *Elitist*, um algoritmo que diferencia do GA-TA no processo de substituição da nova população; *Steady State*, um algoritmo de estado estacionário; e algoritmos multiobjetivo, baseados no NSGA-II: NSGA-M, NSGA-ST e NSGA-O. As diferenças entre eles serão discutidas nas seções seguintes.

5.3 Heurística de inicialização da população

Uma melhoria foi utilizada para auxiliar o processo de busca dos algoritmos genéticos: uma heurística gulosa inicializa a população. Um indivíduo é gerado com a heurística gulosa e o restante da população é gerado aleatoriamente.

A heurística pretende escolher uma sequência com tarefas mais próximas a eles utilizando um valor aproximado para esse custo. O custo é o tempo estimado para o agente percorrer da sua posição atual até a a posição de coleta da tarefa. Dessa forma, cada vez que o agente termina uma tarefa, ele passa para a próxima que estiver mais perto dele.

A heurística utiliza uma matriz de custo, da posição dos agentes naquele dado tempo para a posição de coleta das tarefas. Em cada execução, o agente e a tarefa com o menor valor de custo são adicionados ao cromossomo, a tarefa é removida dessa matriz e a posição do agente e seu tempo final é atualizado.

A posição do agente é definida para o ponto de entrega da tarefa escolhida, e o agente tem seu tempo final atualizado da seguinte forma: $TF = TF_{ant} + T(s_i, p_i) + T(p_i, d_i)$. O tempo final recebe o tempo final anterior somado ao tempo estimado para que o agente se desloque de sua posição inicial até o ponto de coleta da tarefa e ao tempo de deslocamento do ponto de coleta ao ponto de entrega.

A cada execução, a matriz de custo é atualizada levando em consideração a nova posição do agente. Em (24) experimentos computacionais demonstraram a eficácia em reduzir os valores de *makespan* das soluções usando esta heurística em comparação com a inicialização de toda a população de forma aleatória.

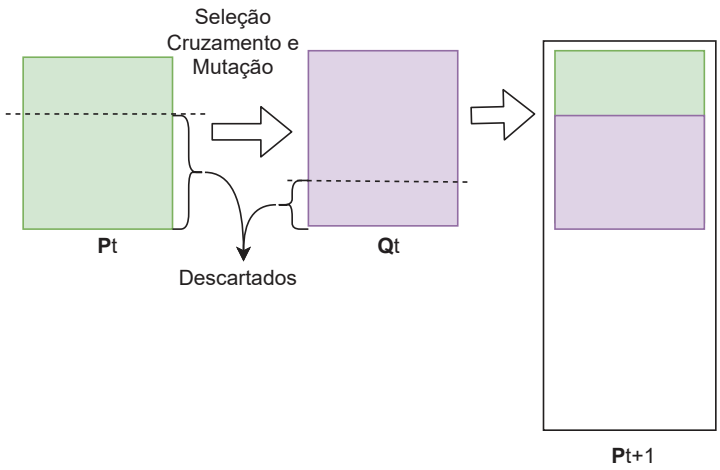
5.4 Mono-objetivo - GA-TA

A Figura 15 representa o processo de substituição da população desse algoritmo. Uma população inicial P_t é criada e avaliada. Os operadores de seleção, cruzamento e mutação são aplicados gerando uma nova população Q_t do mesmo tamanho N. São descartados os piores indivíduos da população atual P_t, de acordo com a sua aptidão e são adicionados os melhores indivíduos da nova população Q_t, formando a nova população P_{t+1} de mesmo tamanho N.

5.5 Mono-objetivo - GA-E

No algoritmo GA-E o indivíduo é composto somente do vetor de tarefas. O processo do algoritmo segue semelhante ao GA-TA, com a diferença que o cruzamento e a mutação ocorrem somente no vetor de tarefas. Para alocar os agentes nas tarefas é utilizado um escalonador que utiliza de uma heurística pra atribuir cada agente a uma tarefa. A

Figura 15 - Procedimento de substituição do GA-TA



Fonte: Extraído de (23).

heurística funciona de forma gulosa, percorrendo o vetor de agentes e escolhendo o agente para cada tarefa de acordo com dois critérios:

- Tempo final TF do agente; e
- Tempo que o agente gasta para percorrer a distância da posição s_i, até o ponto de coleta da tarefa p_i.

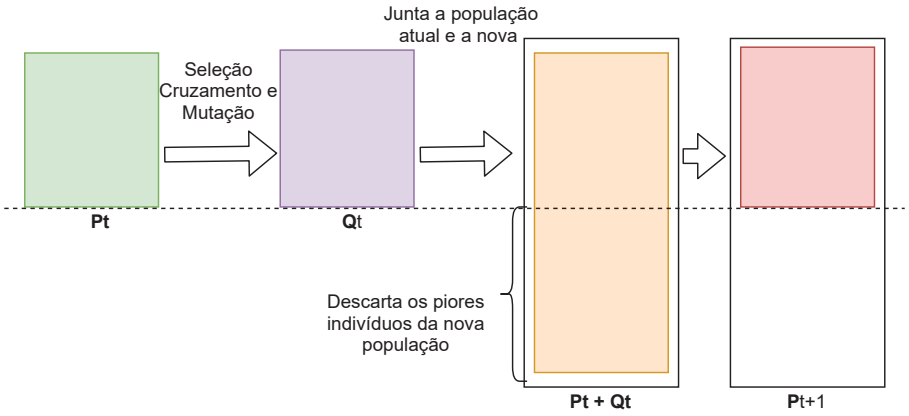
Para cada agente, esses dois valores são somados, e o agente com o menor valor é escolhido para executar a tarefa. A cada iteração, um agente é atribuído a uma tarefa e adicionado ao vetor de agentes, e os valores, de TF e s_i, são atualizados.

Dado que $T(u, v)$ é o tempo necessário para percorrer a distância de u até v. E s_i é a posição do agente no tempo TF. O tempo final do agente é atualizado da seguinte forma: $TF = TF_{ant} + T(s_i p_i) + T(p_i, d_i)$, ou seja, o tempo final do agente é o tempo final anterior do agente somado ao tempo gasto para percorrer a posição s_i para a posição de coleta p_i mais o tempo gasto para percorrer a posição de coleta p_i para a posição de entrega d_i. A posição s_i é atualizada a cada iteração da seguinte forma: $s_i = d_i$ a posição do agente no tempo final é igual a posição de entrega da tarefa, sendo p_i o ponto de coleta, g_i o ponto de entrega da tarefa τ_i. Depois de determinado qual agente vai executar qual tarefa, segue-se para a próxima tarefa.

5.6 Mono-objetivo *Elitist*

Elitist é um algoritmo genético geracional, como ilustrado na Figura 16. Na primeira iteração, é gerada uma população P_t, e o processo de seleção é quem escolhe os indivíduos em que vão ser aplicados os operadores de cruzamento e mutação. Após

Figura 16 - Procedimento de substituição do *Elitist*.



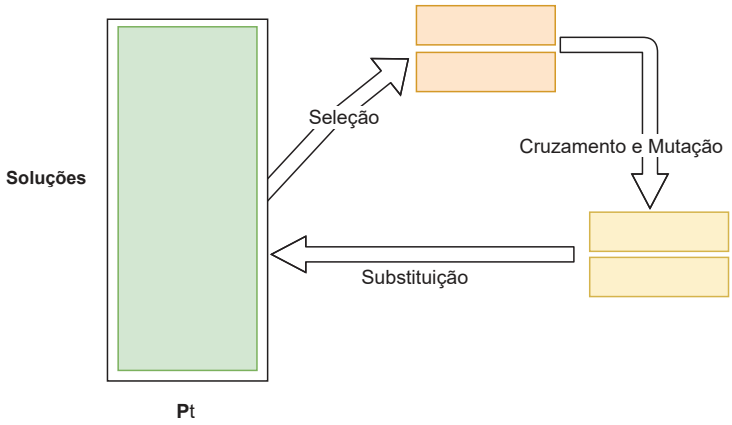
Fonte: Extraido de (23).

aplicar os operadores de cruzamento e mutação tem-se uma nova população filha Q_t do mesmo tamanho N. As duas populações são unidas, $P_t + Q_t$ de tamanho $2N$. Essa nova população é ordenada de acordo com o valor de aptidão e os indivíduos menos aptos são descartados, mantendo a população P_{t+1} com tamanho N, e a elite da combinação das populações é mantida. Isso se repete para as demais iterações até atingir o critério de parada. O *makespan* é utilizado como valor de aptidão de cada indivíduo.

5.7 Mono-objetivo *Steady State*

Os outros algoritmos genéticos apresentados aqui são conhecidos como geracionais. Uma estratégia de estado estacionário é usada no algoritmo apresentado nesta seção. Na abordagem geracional, toda a população, P_t, é substituída a cada geração, enquanto na abordagem de estado estacionário apenas alguns indivíduos da população são substituídos a cada geração. A seleção por torneio gera dois filhos e esses são adicionados a população P_t, resultando em uma nova população com tamanho $N + 2$, onde N é o tamanho da população original. A população é ordenada de acordo com os seus valores de aptidão e os dois piores indivíduos são removidos para que o tamanho da população permaneça constante (tamanho N). Ao contrário dos algoritmos genéticos geracionais, as abordagens de estado estacionário não geram uma nova população de tamanho N a cada geração. Nessa abordagem, somente dois novos indivíduos são gerados, o que resulta que em cada geração ocorre menos avaliações da função objetivo. A abordagem difere no processo de torneio, são passados no torneio dois indivíduos da população, $P[i]$ e $P[i + 1]$, se não forem encontrados indivíduos com valores melhores de aptidão na população, os indivíduos passado no torneio (P[i] e P[i+1]) são retornados como progenitores. Na primeira iteração $i = 0$, a cada iteração $i \leftarrow i + 1$ até i ser igual a N. Depois o processo começa de novo, dessa forma todos os indivíduos da população tem chances iguais de serem escolhidos como progenitores. Depois de atingido o critério de parada, é escolhido o indivíduo de

Figura 17 - Procedimento de substituição do *Steady State*.



Fonte: Extraído de (23).

acordo com a sua aptidão. Para a tomada de decisão o indivíduo com o menor valor de *makespan* é a solução. O processo é ilustrado na Figura 17.

5.8 Análise dos objetivos

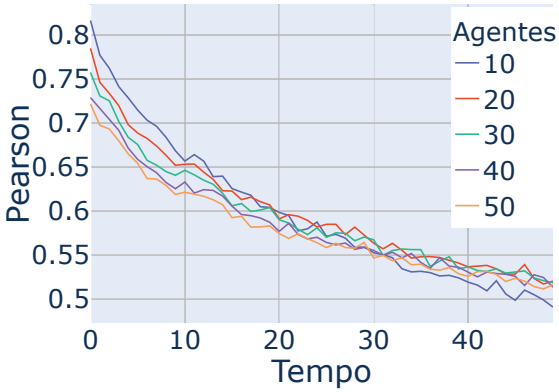
Foi realizada uma análise dos objetivos utilizados na literatura, *makespan* e *service time*, visto a importância desses objetivos para o problema (19). As análises foram feitas a fim de determinar se otimizando um objetivo também resultaria na melhora do outro. Os testes foram feitos em dois ambientes: um ambiente pequeno e um ambiente grande. O algoritmo GA-TA com o método de planejamento de caminho PP-E foi utilizado para gerar os dados parar fazer a análise de correlação. O coeficiente de correlação de Pearson (ρ), que é uma medida de associação linear entre as varáveis, foi utilizado para verificar a relação entre o par de objetivos. Foram gerados 1000 indivíduos aleatórios e 30 execuções independentes. Os testes foram feitos toda vez que o algoritmo genético foi chamado (quando entram novas tarefas no sistema). O coeficiente de correlação de Pearson está compreendido no intervalo fechado de -1 a 1. A correlação pode ser positiva e negativo, quanto maior o valor do coeficiente de correlação, em módulo, maior a correlação entre os objetivos.

Na Figura 18 temos o gráfico de correlação entre os dois objetivos. O eixo y é o coeficiente de correlação e o eixo x é o instante de tempo no qual entraram tarefas no sistema (e o algoritmo genético é chamado). A Figura 18a representa a correlação gerada em um ambiente pequeno com 10, 20, 30, 40 e 50 agentes. Ao todo entram 500 tarefas no sistema. A Figura 18b representa os resultados de um ambiente grande, com 100, 200, 300, 400 e 500 agentes, com a quantidade de total tarefas sendo 1000.

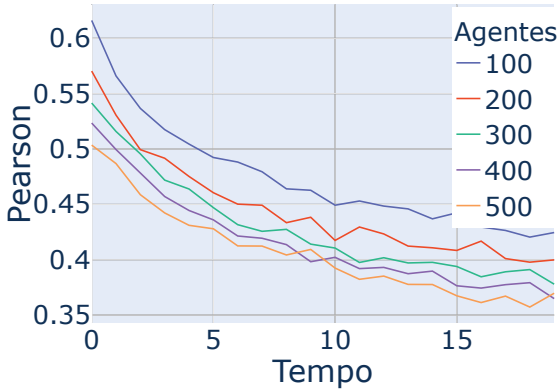
Pode ser visto na Figura 18 que, à medida em que o tempo aumenta, mais tarefas entram no sistema e vão sendo acumuladas. Neste contexto, o coeficiente de correlação

Figura 18 - Gráfico de correlação entre os dois objetivos: *makespan* e *service time*. O eixo y é o coeficiente de correlação de Pearson (ρ), e o eixo x é o instante de tempo que o algoritmo genético é chamado.

(a) Ambiente pequeno



(b) Ambiente grande



diminui. Para o ambiente pequeno não é possível notar uma clara diferença entre os valores obtidos de correlação para os diferentes valores de agentes. Por outro lado, para o ambiente grande, essa diferença fica clara. À medida que o número de agentes aumenta, a correlação entre os objetivos diminui. Para o ambiente pequeno, mesmo que a correlação entre os objetivos diminua com o passar do tempo, a correlação ainda não pode ser considerada baixa o suficiente para que se justifique a utilização de uma otimização multiobjetivo. No ambiente grande, devido a baixa correlação entre os indivíduos, há uma tendência de que esse problema, nesta instância, pode ser tratado como um problema multiobjetivo. Dessa forma, os resultados sugerem uma propensão de que uma melhoria em um dos objetos não acarreta, necessariamente, em uma melhoria no outro.

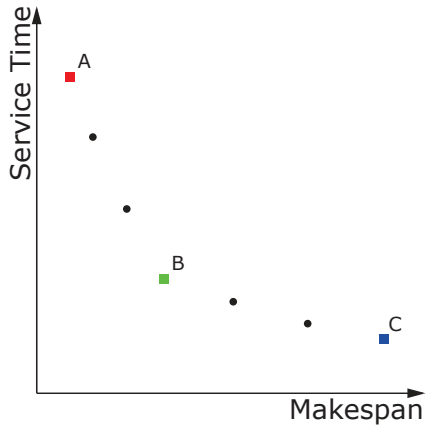
5.9 Otimização Multiobjetivo

Para a otimização multiobjetivo foi utilizado o NSGA-II, descrito na Seção **4.2.1**. Foram utilizadas três abordagens que se diferenciam na tomada de decisão de qual solução do NSGA-II considerar. Ao terminar a busca, tendo o algoritmo atendido ao critério de parada, são disponibilizadas ao tomador de decisão um conjunto de soluções. Um exemplo de soluções é o primeiro posto, como mostrado na Figura 19, em que cada ponto no gráfico representa uma solução, e os pontos A, B e C são amostras desse conjunto. Devido ao tamanho e a complexidade das soluções encontradas em grande parte dos problemas, é necessário um tomador de decisão. Esse tomador de decisão deve considerar qual solução escolher segundo determinadas preferências. As preferências especificas de cada algoritmo que são explicas no paragrafo seguinte.

frente aos conflitos decorrentes das funções objetivos otimizadas

O algoritmo NSGA-M escolhe o indivíduo de acordo com o melhor valor do

Figura 19 - Exemplo de população final.



Fonte: Adaptado de (23).

makespan, na Figura 19 representado pelo indivíduo A. O algoritmo NSGA-ST escolhe o melhor indivíduo de acordo com o objetivo do *service time*, na Figura 19 representado pelo indivíduo C. O algoritmo NSGA-O escolhe o indivíduo que está mais próximo da origem segundo uma distância euclidiana, representado pelo indivíduo B, com o intuito de escolher um indivíduo que possua melhores resultados em ambos os objetivos. Para encontrar tal indivíduo é necessário normalizar os resultados para os dois objetivos, visto que sua ordem de magnitude é diferente e, sem a normalização, resultaria em um peso maior de um objetivo em relação ao outro.

6 EXPERIMENTOS COMPUTACIONAIS

Neste capítulo, são exibidos os experimentos computacionais e resultados obtidos a partir das propostas apresentadas. Para cada proposta, foram feitos experimentos separadamente, sendo comparados com algoritmos da literatura bem como algoritmos do estado da arte.

Os experimentos foram feitos utilizando 3 algoritmos de planejamento de caminho: *Prioritized Planning* (PP), um algoritmo MAPF desacoplado; PP-E, o algoritmo *Prioritized Planning* com a diferença na abordagem para evitar *deadlocks*; e CBS um algoritmo MAPF que utiliza uma abordagem acoplada e desacoplada. Para a alocação de tarefas foram utilizados 7 diferentes algoritmos genéticos: GA-TA um algoritmo geracional padrão; GA-E em que a alocação de tarefas para os agentes é feita por meio de um escalonador; *Elitist*, que varia no processo de substituição da população; *Steady State*, um algoritmo de estado estacionário; e mais 3 algoritmos multiobjetivos que variam no processo de tomada de decisão do NSGA-II, sendo esses: NSGA-M, NSGA-ST e NSGA-O.

Os experimentos foram feitos executando cada método 30 vezes para cada instância. As instâncias utilizadas foram de (19), e estão disponíveis publicamente[1]. O código fonte foi desenvolvido em C++ e também está disponível publicamente[2]. Os experimentos foram realizados numa máquina Intel Xeon CPU X5650 2.67GHz com 16GB de memória.

Para todos os algoritmos genéticos, foi utilizada uma população de 20 indivíduos e o número de gerações foi definido como 50. Os parâmetros utilizados para a criação de uma nova população utilizando seleção por torneio foi de 1%. Foi utilizado um valor de 50% de chance para ocorrer o cruzamento. Para a mutação ocorrer também foi utilizado 50% no vetor de agentes e 50% no vetor de tarefas. No caso do algoritmo GA-E não ocorre mutação no vetor de agentes devido a sua não existência, resultando em apenas 50% de chance no vetor de tarefas. Quando a mutação ocorrer no vetor de tarefas, o valor utilizado para α é de 30%. Esses parâmetros foram escolhidos de forma empírica, de forma que nenhum procedimento formal foi utilizado.

Para o algoritmo GA-TA e GA-E o valor utilizado para a seleção por elitismo da nova população foi 60%. No algoritmo *Steady State*, para manter a mesma quantidade de avaliações de funções objetivos dos outros algoritmos, foram usadas 500 gerações.

Para avaliar a significância da diferença dos resultados dos algoritmos, foi usado o teste de *Wilcoxon* com a hipótese nula de que a distribuição das amostras tem medianas iguais. Os resultados foram considerados estatisticamente diferentes quando o p-valor é menor que 0,05. Para o ambiente pequeno, os testes foram feitos nos algoritmos CBS e *Prioritized Planning* em relação ao Central. Por exigência da implementação[3], os testes só

[1] https://github.com/carolladeira/MAPD-Instances.
[2] https://github.com/carolladeira/MAPD-NSGA.
[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html.

foram realizados quando o tamanho da amostra foi maior que 20. Para o ambiente grande, os testes foram feitos entre os 6 algoritmos implementados: GA-TA, *Elitist*, *Steady State*, NSGA-M, NSGA-ST e NSGA-O. Um $\star$ ao lado do resultado na Tabela 1, 2 e 3 indica que esses valores são estatisticamente semelhantes aos resultados do melhor algoritmo, representados em negrito (mesma distribuição).

6.1 Resultados em um ambiente pequeno

Para o ambiente pequeno, foram realizados experimentos variando o método de planejamento de caminho e o algoritmo genético para alocação de tarefas. Os experimentos foram realizados com os seguintes algoritmos:

- PP-E+GA-TA+a - PP-E com GA-TA e inicialização com heuristica
- PP+GA-TA+a - *Prioritized Planning* com GA-TA e inicialização com heuristica
- PP+GA-TA+b - *Prioritized Planning* com GA-TA e inicialização sem heuristica
- PP+GA-E+a - *Prioritized Planning* com GA-E e inicialização com heuristica
- PP+GA-E+b - *Prioritized Planning* com GA-E e inicialização sem heuristica
- CBS+GA-TA+a - CBS com GA-TA e inicialização com heuristica
- CBS+GA-TA+b - CBS com GA-TA e inicialização sem heuristica
- CBS+GA-E+a - CBS om GA-E e inicialização com heuristica
- CBS+GA-E+b - CBS com GA-E e inicialização sem heuristica

Na Tabela 1, são mostrados os resultados da média do *makespan*. Os algoritmos são comparados com o Central e TP, ambos usados em (19). Por ser um algoritmo computacionalmente custoso, para os testes com o CBS, foi definido um tempo limite de 5 minutos no planejamento de caminho, similar ao que foi adotado em (15). Se excedido o intervalo de 5 minutos, a solução é considerada expirada.

Ao lado de cada solução foi informado, entre parênteses, a quantidade de vezes em que a solução ultrapassou o limite de tempo de 5 minutos, denominado *timeout*. Caso o limite tenha sido excedido em todas as execuções, o valor da solução é substituído pelo termo *timeout* na tabela.

Pode-se observar que o CBS+GA-TA+a quando resolvendo o problema com frequência 0.2, não gerou *timeout* em nenhuma execução. Na frequência 10 com 20 agentes, 14 execuções geraram *timeout*. Com 30 agentes, 26 execuções geraram *timeout*, e com 50 agentes, 29 execuções. À medida que o número de agentes aumenta, a quantidade de vezes que o algoritmo ultrapassa o limite de tempo aumenta.

Tabela 1: Média dos resultados do *makespan* em um ambiente pequeno. "F" é a frequência de tarefas, "Ag" é o número de agentes, e "Central" e "TP" são técnicas propostas em (19). 'a' e 'b' se referem ao uso da heurística de inicialização. Os melhores valores para cada instância estão destacados em negrito.

				PP-E	*Prioritized Planning*				CBS			
F	Ag	TP	Central	GA-TA	GA-TA		GA-E		GA-TA		GA-E	
				a	a	b	a	b	a	b	a	b
	10	2532	**2513**	2517.7	2519.7	3476.9	2519.8	2520.3	2517.3(0)	3210.4(1)	2516.9(0)	2516.2(0)
	20	2540	**2513**	2516.5	2518.7	2621.7	2520.7	2520.2	2517.2(0)	2516.8(0)	2521.0(0)	2521.0(0)
0.2	30	2546	**2513**	2515.3	2518.4	2575.7	2521.1	2521.0	2516.8(0)	2517.0(0)	2520.0(0)	2520.0(0)
	40	2540	**2511**	2515.0	2516.4	2558.6	2515.0	2515.0	2516.6(0)	2516.1(0)	2515.0(0)	2515.0(0)
	50	2540	**2511**	2515.3	2517.1	2559.6	2515.0	2515.0	2515.3(0)	2516.3(0)	2515.0(0)	2515.0(0)
	10	1309	**1242**	1249.4	1255.0	2777.0	1334.1	1913.6	1250.6(2)	2743.6(7)	1326.4(1)	1903.7(1)
	20	1094	1031	1047.0	1049.8	1940.2	1025.2⋆	1025.4⋆	1051.4(15)	timeout	1025.1⋆(1)	**1025.0**⋆(0)
0.5	30	1069	1034	1024.9	1028.5⋆	1651.3	1021.9⋆	1021.9⋆	1026.8(18)	timeout	**1021.1**⋆(1)	1021.4⋆(0)
	40	1090	1034	1022.7	1025.4⋆	1512.1	1021.2⋆	1021.0⋆	1022.0⋆(9)	timeout	1020.1⋆(1)	**1020.0**⋆(0)
	50	1083	1031	1023.5	1025.4⋆	1421.9	1020.5⋆	1020.4⋆	1022.3⋆(9)	timeout	1019.9⋆(0)	**1019.8**⋆(0)
	10	1198	1143	1137.3	1144.0	2373.2	1174.8	1790.8	**1121.7**⋆(0)	2341.2(13)	1155.4(2)	1757.8(2)
	20	757	673	664.2	669.4⋆	1522.3	741.6	965.1	**656.0**⋆(9)	timeout	731.3(7)	939.1(7)
1	30	607	**557**	568.3	571.8	1241.4	595.0	623.6	567.1(21)	timeout	574.5(26)	584.0(26)
	40	624	556	543.7	546.2⋆	1101.0	527.8	527.8	549.7 (23)	timeout	527.0(12)	**526.7**(15)
	50	597	552	537.2	539.0⋆	1010.6	527.9	527.8	538.0 (23)	timeout	**526.5** (13)	**526.5**⋆(9)
	10	1167	1121	1113.3	1120.3	2136.5	1132.2	1626.1	**1093.9**⋆(1)	2103.5 (5)	1109.7⋆(0)	1597.6(0)
	20	683	628	605.0	611.3⋆	1288.4	634.7	844.5	**590.9**(10)	1239.5(28)	615.4⋆(7)	818.9(9)
2	30	529	466	441.3	445.8⋆	1003.2	496.4	587.3	**433.1**(21)	timeout	474.4(23)	564.1(19)
	40	464	385	368.7	372.3⋆	864.0	430.3	453.8	**361.2**(24)	timeout	419.6(27)	431.7(26)
	50	432	**320**	331.0	332.4	773.7	361.8	366.1	329.8 (26)	timeout	362.0(28)	timeout
	10	1162	1105	1099.9	1103.7	1994.1	1106.3	1527.8	**1081.5**⋆(1)	1968.5(4)	1084.7⋆(0)	1499.3(0)
	20	655	594	592.0	594.9	1151.7	599.5	770.4	**570.6**(15)	1110.1(14)	575.5⋆(7)	742.5(7)
5	30	478	426	421.5	430.3	866.0	473.4	529.2	**411.3**(26)	828.6(25)	413.8(22)	503.2(19)
	40	418	334	344.4	347.0	721.5	363.2	410.5	**318.0**(29)	698.0(29)	343.0(28)	378.0(29)
	50	395	295	291.3	294.4	637.7	321.7	339.2	**275.0**(29)	624.0(29)	299.0(29)	timeout
	10	1163	1090	1094.2	1103.7	1943.6	1103.7	1490.8	**1077.4**⋆(0)	1915.8(1)	1073.5⋆(2)	1468.6(1)
	20	643	607	582.8	588.0⋆	1099.0	590.8	742.9	**564.4**(14)	1062.5(6)	560.9(12)	719.7(13)
10	30	526	414	417.7	423.6	815.3	423.3	509.2	**399.3**(26)	790.5(14)	397.3(18)	482.0(21)
	40	407	341	335.7	344.8	678.4	342.3	394.4	timeout	636.8(21)	**315.3**(27)	360.5(28)
	50	333	277	285.8	286.4	591.0	301.9	326.3	**273.0** (29)	568.0(26)	timeout	timeout

A utilização de heurística aumentou significativamente a qualidade da solução. Como visto, comparando o PP+GA-TA+a ao PP+GA-TA+b, o PP+GA-TA+a obteve um ganho médio de makespan de 36,38%. Isso também pode ser visto com o algoritmo GA-E, comparando o PP+GA-E+a e o PP+GA-E+b, a utilização da heurística conseguiu obter uma melhora de 10,96% na média das soluções. O mesmo pode ser dito sobre os testes utilizando o planejamento de caminho do CBS. Dado que muitos dos resultados das variantes com sufixo b resultaram em *timeout*, os cálculos foram realizados somente nos valores de frequência que obtiveram resultados sem *timeout*, resultando em uma melhora de 30,81% refente ao GA-TA e 11,02% referente ao GA-E. Também observa-se que a utilização da heurística gerou melhores resultados no algoritmo GA-TA em relação ao GA-E, visto que a margem de ganho é superior no GA-TA.

Em frequências baixas, os resultados não tiveram grande variação. À medida que a frequência aumenta, percebe-se uma melhora do GA-TA+a em relação ao GA-E+a, com destaque para os resultados com maior numero de agentes, na qual a diferença chega a 13,48% nos casos com 40 agentes e frequência 2, e 10,21% nos casos com 30 agentes e frequência 2.

Como pode-se observar na Tabela 1, os melhores resultados, representados em

negrito, são os obtidos na utilização do CBS+GA-TA+a em frequências maiores que 1. Também pode-se perceber que a utilização de algoritmos genéticos não funciona tão bem com frequência baixa, que é quando poucas tarefas entram no sistema

Foi testado outro algoritmo para o planejamento de caminho, o PP-E. Esse algoritmo foi testado junto com o algoritmo GA-TA e a heurística de inicialização. O algoritmo GA-TA foi utilizado devido ao fato de obter melhores resultados comparados ao GA-E, e a heurística foi utilizada pois obteve melhores resultados quando comparada a não utilização da mesma.

Comparando os 3 algoritmos MAPF desacoplados (PP-E+GA-TA+a, PP+GA-TA+a e TP), nota-se que os algoritmos PP-E e PP ao se comparar com o TP obtiveram melhores resultados em todas as instâncias. O PP-E obteve um ganho médio de 5,88% e o PP 5,50%.

Quando compara-se o PP-E+GA-TA+a com o PP+GA-TA+a, percebemos um melhor resultado em todas as instâncias, obtendo um percentual médio de melhora de 0,41%.

6.2 Resultados em um ambiente grande

Para o ambiente grande, foram realizados experimentos utilizando a heurística de inicialização em todos os algoritmos uma vez que sua utilização obteve melhores resultados no ambiente pequeno para todas as instâncias. Dado que o CBS não é escalável, foi utilizado o *Prioritized Planning* modificando-o a fim de evitar conflitos do algoritmo de planejamento de caminho, visto que essa abordagem conseguiu melhores resultados. No algoritmo PP-E, quando o agente termina de executar uma tarefa, ele não retorna para sua posição inicial, deslocando-se para o *endpoint* livre mais próximo. Os algoritmos utilizados no ambiente grande foram:

- PP-E+GA-TA - *Prioritized Planning* com GA-TA
- PP-E+*Elitist* - *Prioritized Planning* com *Elitist*
- PP-E+SS - *Prioritized Planning* com SS
- PP-E+NSGA-M - *Prioritized Planning* com NSGA-M
- PP-E+NSGA-ST - *Prioritized Planning* com NSGA-ST
- PP-E+NSGA-O - *Prioritized Planning* com NSGA-O

Os resultados médios de *makespan* são mostrados na Tabela 2 e os resultados do *service time* são mostrados na Tabela 3. Para avaliar os algoritmos eles foram comparados com HBH+MLA* (8) e TP (19).

Tabela 2: Média dos resultados do *makespan* em um ambiente grande. Os melhores valores para cada instância estão destacados em negrito

Agentes	TP	HBH+MLA*	GA-TA	*Elitist*	*Steady State*	NSGA-M	NSGA-ST	NSGA-O
100	1018	800	771.60⋆	773.73	769.87⋆	**767.07**	769.47⋆	769.63⋆
200	745	606	460.07	450.73	455.20	**444.30**	447.10⋆	445.37⋆
300	746	507	359,03	352.93	357.43	346.77⋆	**345.63**	348.40⋆
400	788	491	304.37	301.83	303.77	300.33⋆	299.53⋆	**297.47**
500	946	501	285.03	275.87⋆	278.57	273.10⋆	276.20⋆	**272.67**

Tabela 3: Média dos resultados do *service time* em um ambiente grande. Os melhores resultados para cada instância estão destacados em negrito.

Agentes	TP	HBH+MLA*	GA-TA	*Elitist*	*Steady State*	NSGA-M	NSGA-ST	NSGA-O
100	463.25	363	354.51	351.86	352.84	350.27⋆	**350.17**	350.32⋆
200	330.18	208	200.98	199.15	199.65	**198.67**	198.69⋆	198.79⋆
300	301.96	157	153.06	151.20	151.55	150.65	**150.39**	150.55⋆
400	289.07	136	130.03	128.53	129.09	128.00⋆	**127.89**	127.99⋆
500	284.24	125	118.62	116.50	116.96	116.00⋆	116.11⋆	**115.97**

Dentre os algoritmos testados aqui, pode-se observar a vantagem dos algoritmos NSGA-II em relação aos demais. Os 3 algoritmos deste tipo apresentaram melhores resultados levando em conta a avaliação do *makespan* e do *service time*. Os resultados de cada um desses algoritmos apresentam pouca variação, e sem uma distribuição clara dessa variação em relação ao número de agentes.

Comparando os resultados do *makespan* na Tabela 2, tem-se que os resultados de todos os algoritmos NSGA-II são estatisticamente semelhantes. Os resultados dos demais algoritmos apresentam diferenças estatisticamente significativas, exceto: os algoritmos GA-TA e *Steady State* em testes com 100 agentes; e o algoritmo *Elitist* no caso de 500 agentes. Comparando os resultados do *service time* na Tabela 3, os algoritmos do NSGA-II são, em sua maioria, estatisticamente semelhantes. Uma exceção ocorre no caso do NSGA-M com 300 agentes, em que ele apresentou uma diferença significativa. Os demais algoritmos apresentaram resultados estatisticamente significativos em sua totalidade.

A Figura 20 apresenta os boxplots referentes ao *makespan*, *service time*, e tempo de execução dos algoritmos propostos. É possível observar que os métodos que utilizam o

NSGA-II apresentam melhorias tanto no *makespan* quanto no *service time*, ao passo que mantém o custo computacional baixo comparado com o TP, e outros métodos basedos em algoritmos genéticos aqui apresentados. Entre os algoritmos baseados em NSGA-II, os NSGA-M, NSGA-ST e NSGA-O, não houve diferença expressiva em relação ao custo computacional para um número de agentes acima de 200, como mostrado na terceira coluna dos boxplots (Figura 20).

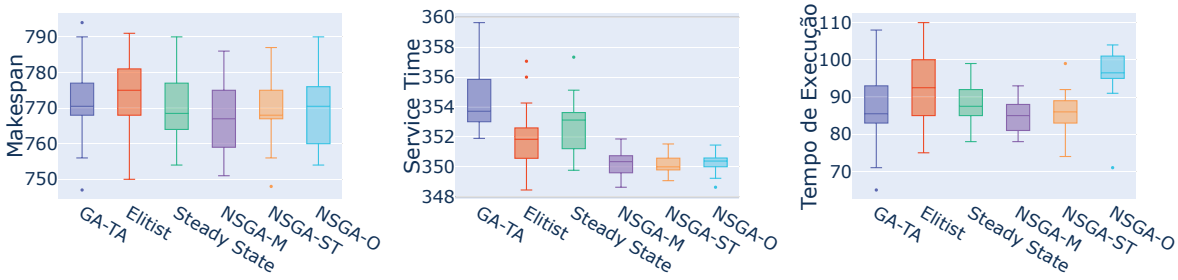
Visto que os resultados de *makespan* e *service time* apresentam grandes diferenças ao se variar o número de agentes, é realizada uma normalização dos valores da Tabela 2 e 3 a fim de se permitir uma melhor comparação dos métodos. Pode-se então perceber que, pela média dos resultados normalizados, o NSGA-M obteve o melhor resultado geral, seguido pelo NSGA-O e do NSGA-ST. O NSGA-M apresentou um resultado estatisticamente diferente comparado com os outros algoritmos no *service time* com 300 agentes e, por isso, o NSGA-O foi o que melhor se ajustou para ser comparado.

A partir dos resultados apresentados, é possível observar uma melhora considerável no desempenho do NSGA-O com o aumento do número de agentes. Pode-se destacar os resultados do *makespan* com 500 agentes, sendo 45,57% melhores em relação ao HBH+MLA*, mostrando sua eficiência e escalabilidade.

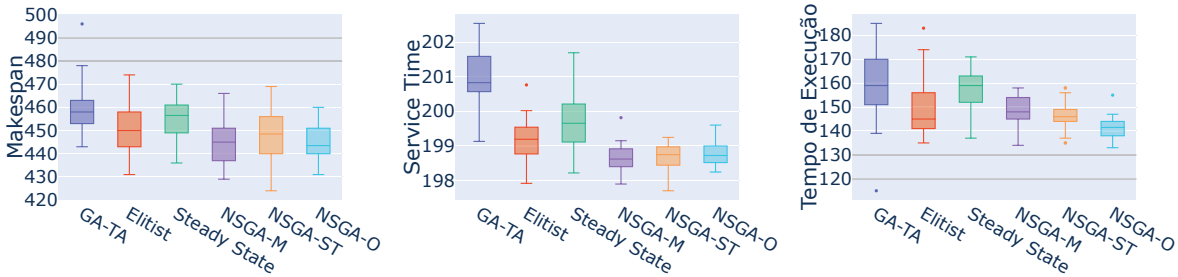
A Figura 21 apresenta os boxplots do *makespan*, *service time*, tempo de execução do NSGA-O e dos algoritmos da literatura TP e HBH+MLA*. Comparado com o HBH+MLA*, o algoritmo de planejamento de caminho PP-E, difere ao permitir que o agente troque de tarefas enquanto o agente não chega na posição de coleta da tarefa. Isso faz com que o agente tenha que reprocessar o caminho e, dessa forma, obtemos um melhor *makespan* e *service time*, ao custo do tempo de processamento. Ainda, deve-se levar em conta que não foi possível fazer uma comparação direta com o tempo de execução dos algoritmos devido as diferenças nos processadores utilizados.

Figura 20 - Boxplot dos resultados para o ambiente grande com os seguintes algoritmos: GA-TA, *Elitist*, *Steady State*, NSGA-M, NSGA-ST, NSGA-O.

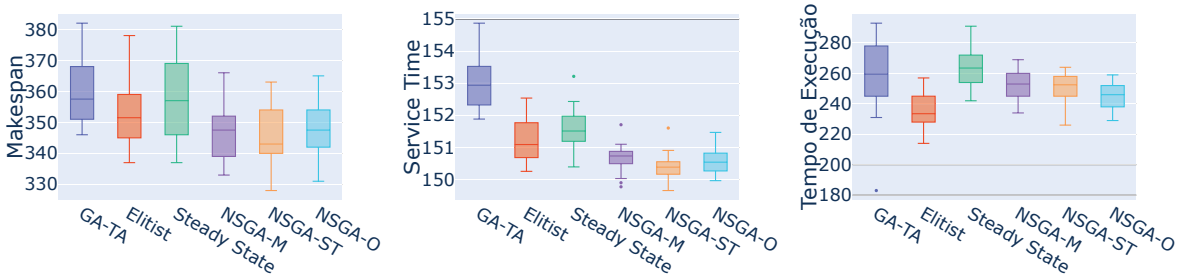
(a) 100 agentes



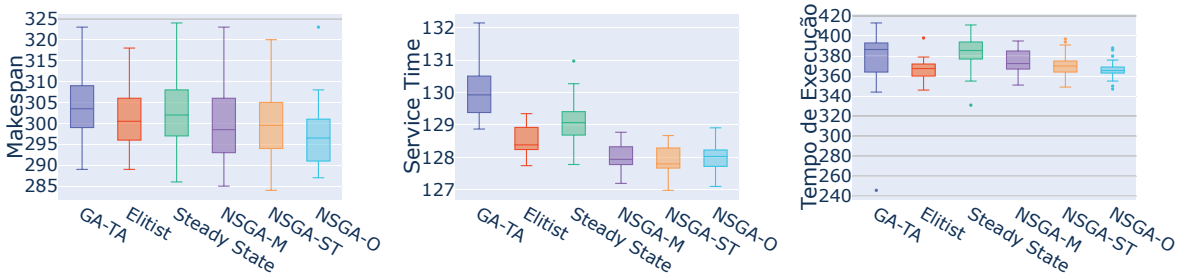
(b) 200 agentes



(c) 300 agentes



(d) 400 agentes



(e) 500 agentes

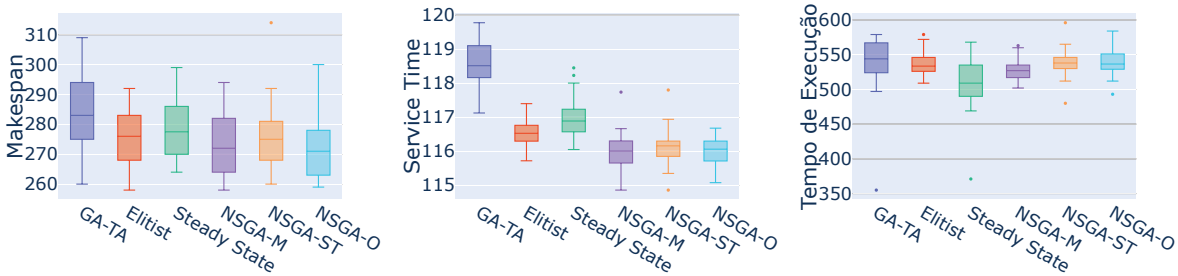
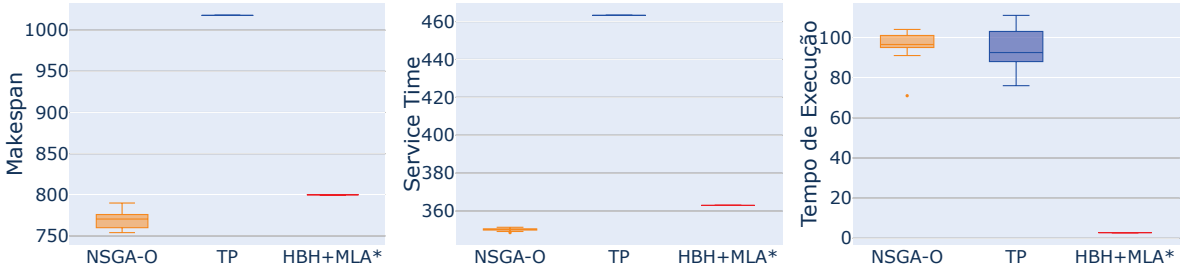
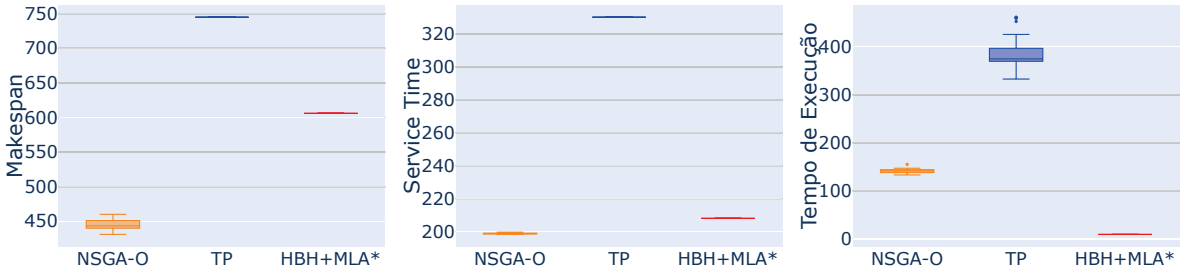


Figura 21 - Boxplot dos resultados para o ambiente grande com do algoritmo proposto NSGA-O, e os algoritmos da literatura TP e HBH+MLA*.

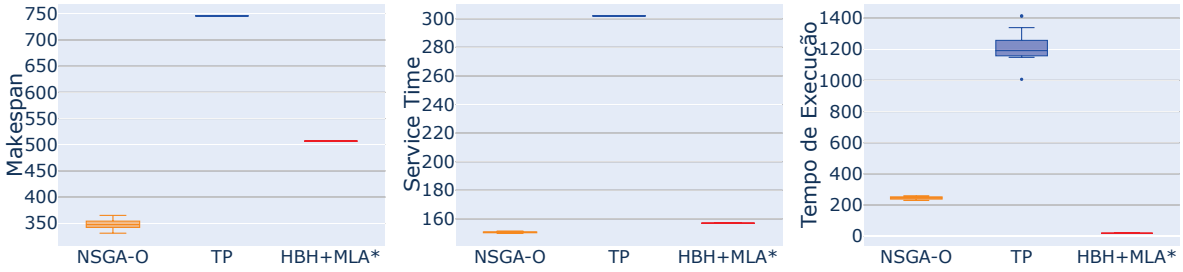
(a) 100 agentes



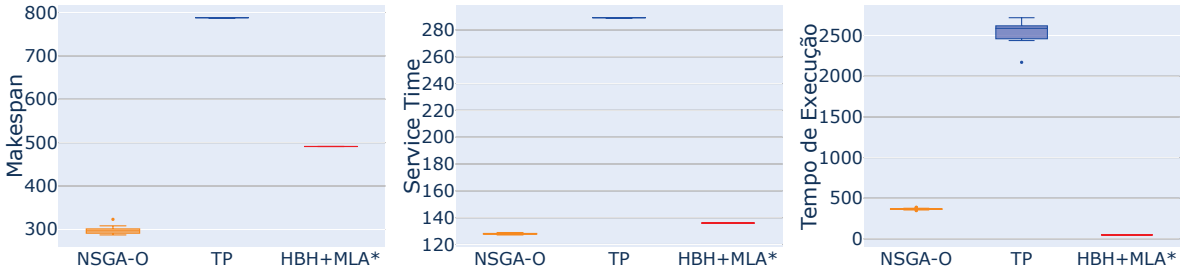
(b) 200 agentes



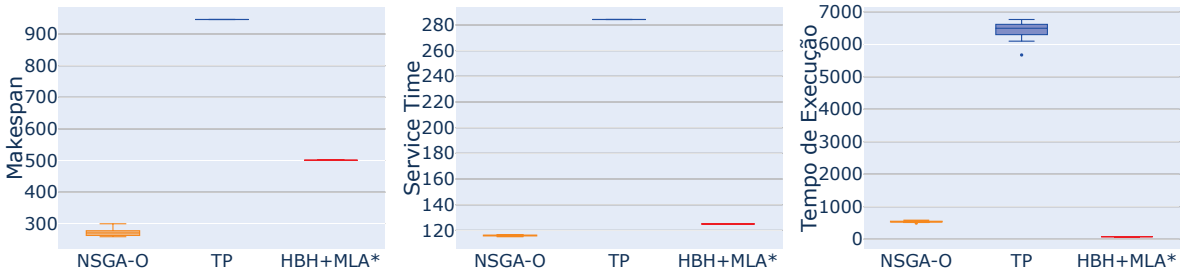
(c) 300 agentes



(d) 400 agentes



(e) 500 agentes



7 CONCLUSÕES E TRABALHOS FUTUROS

Em muitas aplicações de sistemas multiagentes do mundo real, os agentes autônomos são atribuídos a objetivos. Os agentes precisam planejar caminhos livres de colisão para seus alvos a fim de concluir as tarefas. Essas tarefas podem ser locais (posições) em um mapa. Um exemplo de aplicação é a automação de armazéns que emprega uma frota de robôs, denominados agentes, que se deslocam de um local para outro, movendo mercadorias, aumentando a produtividade e reduzindo custos.

Neste trabalho, foi tratado o problema *Multi-Agent Pickup and Delivery* (MAPD), que tem como fim capturar as características da automatização de armazéns. Nesse problema, existe um sistema multiagente em um ambiente conhecido, um conjunto de tarefas que podem ser inseridas nesse sistema a todo momento, e agentes que executam tais tarefas o mais rápido possível. O problema MAPD pode ser dividido em dois sub-problemas: alocação de tarefas e planejamento de caminho.

Para a alocação de tarefas, foi proposto o uso de algoritmos genéticos a fim de gerar a sequência de tarefas para cada agente. Essa sequência é gerada ignorando colisões e utilizando tempos estimados para o agente chegar na sua posição final. Também foram realizados experimentos utilizando uma heurística gulosa para a geração de um individuo da população inicial, mantendo o resto da população sendo gerada aleatoriamente. Após a geração dessa sequência, é realizado o planejamento de caminhos, dando continuidade ao processo.

Para o planejamento de caminho, foram utilizados três algoritmos: *Prioritized Planning* (PP), PP-E, e *Conflict Based Search* (CBS). *Prioritized Planning* e PP-E são, em sua base, o mesmo algoritmo, sendo que sua diferença se dá no tipo de abordagem em relação à tomada de decisão no momento em que um agente chega em uma posição final. No *Prioritized Planning*, o agente volta para a sua posição inicial, que é a sua posição de descanso, onde o agente iniciou o problema. No PP-E, após chegar na posição final do agente, o mesmo se desloca para o *endpoint* livre mais próximo. Este *endpoint* livre pode ser, inclusive, a própria posição na qual o agente se encontra no momento. O outro algoritmo, CBS, é um algoritmo custoso, que requer mais tempo de processamento, mas que gera bons resultados. Contudo, devido ao seu alto custo de processamento, ele não é viável para ambientes grandes ou com vários agentes.

Os experimentos foram feitos em dois ambientes: um pequeno (21x35) e um grande (81x81). No ambiente pequeno, foram utilizados dois algoritmos genéticos mono-objetivo: o GA-TA e o GA-E. Para o planejamento de caminho foram utilizados dois algoritmos: o CBS e o PP. Com os experimentos, foi confirmada a eficiência ao se utilizar a heurística de inicialização da população e o algoritmo CBS para o planejamento de caminho. Com a utilização do CBS, constatou-se que à medida em que o número de agentes aumenta, o

número de vezes que o algoritmo ultrapassa o tempo limite definido para o processamento também aumenta. Dentre os algoritmos genéticos, GA-TA e GA-E, o GA-TA foi o que obteve os melhores resultados. Testes foram realizados utilizando o planejamento de caminho PP-E com o algoritmo GA-TA e a heurística de inicialização. PP-E obteve melhores resultados quando comparado com o planejamento de caminho PP.

Como o CBS não é escalável e a eficiência do PP-E comparado com o *Prioritized Planning* foi observada, foram feitos os experimentos no ambiente grande utilizando somente o PP-E. Para a alocação de tarefas, foram testados mais dois algoritmos mono-objetivos além do GA-TA, sendo eles: *Steady State, Elitist*. Visto a melhora na solução com o uso da heurística de inicialização, os algoritmos genéticos utilizaram essa heurística para os ambientes grandes. Depois de fazer uma análise dos objetivos verificou-se que para o ambiente grande o problema pode ser tratado como uma otimização multiobjetivo. Então, foram propostas 3 abordagens baseadas no NSGA-II, em que a diferença dos algoritmos está na tomada de decisão de qual individuo escolher na população de soluções não dominadas obtida. Esses algoritmos são: NSGA-M, que escolhe o melhor individuo de acordo com o objetivo do *makespan*; NSGA-ST, de acordo com o objetivo do *service time*; e o NSGA-O, que escolhe o melhor individuo considerando os dois objetivos.

Os experimentos mostraram o sucesso de utilizar uma abordagem multiobjetivo. O NSGA-O obteve o melhor resultado quando comparado com os outros algoritmos e com os resultados da literatura. Os resultados também mostraram que aumentando o número de agentes, aumenta-se a margem de ganho do algoritmo, mostrando a robustez do método.

Para trabalhos futuros pretende-se realizar testes em outros ambientes, considerando restrições nas tarefas, como pesos diferentes, limites de tempo e capacidade. Também podem ser avaliadas outras formas de se aproximar o cálculo do *makespan* e do *service time* utilizado na função objetivo do algoritmo genético. Desse modo, também utilizar as técnicas em cenários diferentes e mais realistas.

Também pretende-se incluir outras heurísticas para inicialização das populações dos algoritmos genéticos. Além disso, propõe-se impedir que os agentes livres troquem as tarefas que lhes foram atribuídas, dessa forma, podendo avaliar o efeito da troca de tarefas nos resultados finais.

REFERÊNCIAS

1 Bennewitz, M., Burgard, W., Thrun, S. (2002). Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2-3), p. 89–99.

2 Čáp, M., Vokřínek, J., Kleiner, A. (2015). Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the international conference on automated planning and scheduling* (Vol. 25, pp. 324-332).

3 Deb, K. (2001). Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons.

4 Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective optimization: NSGA-II. *Proceedings of the 6th International Conference on Parallel Problem solving from Nature*, September, Berlin, Heidelberg: Springer.

5 Duchateau, Wouter. (2021). Multi-Agent Pickup-and-Delivery in a Distributed Baggage Handling System. Delft University of Technology.

6 Erdmann, M. A., Lozano-Pérez, T. (1987). On multiple moving objects. *Algorithmica*, 2, 477–521.

7 Franklin, S., Graesser, A.(1996). Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Budapest, Hungary, pp. 21-35.

8 Grenouilleau, F., van Hoeve, W. J., and Hooker, J. N. (2019). A multi-label A* algorithm for multi-agent pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 29, pp. 181-185).

9 Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

10 Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis ith applications to biology, control, and artificial intelligence.* University of Michigan Press

11 Hönig, W., Kumar, T. K. S., Cohen, L., Ma, H., Xu, H., Ayanian, N. and Koenig, S. (2017). Summary: Multi-Agent Path Finding with Kinematic Constraints.. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI (p./pp. 4869-4873),

12 Jennings, N. R., Sycara, K. and Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*. Vol. 1, pp. 7-38

13 Li, J., Surynek, P., Felner, A., Ma, H., Kumar, T. K. S. and Koenig, S. (2019). Multi-agent path finding for large agents. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. p. 7627-7634.

14 Liu, C. and Kroll, A. (2012). A centralized multi-robot task allocation for industrial plant inspection by using a* and genetic algorithms. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer, Berlin, Heidelberg, p. 466-474.

15 Liu, M., Ma, H., Li, J., and Koenig, S. (2019) Task and path planning for multi-agent pickup and delivery. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS.

16 Ma, H. and Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems*. AAMAS.

17 Ma, H., Koenig, S., Ayanian, N., Cohen, L., Honig, W., Kumar, T. K. S., Uras, T., Xu, H., Tovey, C., Sharon, G. (2016). Overview: Generalizations of multi-agent path finding to real- world scenarios. In IJCAI-16 Workshop on Multi-Agent Path Finding.

18 Ma, Hang. Target Assignment and Path Planning for Navigation Tasks with Teams of Agents. Diss. University of Southern California, 2020.

19 Ma, H., Li, J., Kumar, T. K. S. and Koenig, S. (2017). Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. AAMAS p. 837-845.

20 Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T. K. S. and Koenig, S. (2018). Multi-agent path finding with deadlines. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI.

21 Morris, R., Pasareanu, C. S., Luckow, K. S., Malik, W., Ma, H., Kumar, T. K. S. and Koenig, S. (2016). Planning, scheduling and monitoring for airport surface operations. In: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press.

22 Russell, S., Norvig, P. (2003). Artificial Intelligence: A modern approach. Upper Saddle River, NJ, USA:: Prentice Hall.

23 Queiroz, A. C. L.; Bernardino, H. S.; Vieira, A. B. Solving multi-agent pickup and delivery problems using multiobjective optimization (em revisão). Journal of Intelligent and Robotic Systems, Springer.

24 Queiroz, A. C. L. C., Bernardino, H. S., Vieira, A. B. and Barbosa, H. J. C. (2020). Solving Multi-Agent Pickup and Delivery Problems Using a Genetic Algorithm. In: *Brazilian Conference on Intelligent Systems.* BRACIS Springer. p. 140-153.

25 Savelsbergh, M. W. P. and Sol, M. (1995). The general pickup and delivery problem. *Transportation science*, 29.1: 17-29.

26 Sharon, G., Stern, R., Felner, A. and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, Elsevier, v. 219, p. 40-66.

27 Silver, D. (2005). Cooperative pathfinding. In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AIIDE. AAAI Press. p. 117-122.

28 Stern, Roni et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Proceedings of the Twelfth International Symposium on Combinatorial Search*. SOCS. p. 151-159

29 Surynek, P, A Felner, R Stern, and E Boyarski (2016).An Empirical Comparison of the Hardness of Multi-Agent Path Finding under the Makespan and the Sum of Costs Objectives. In: *Proceedings of the Symposium on Combinatorial Search* (SoCS), pp. 145–147.

30 Veloso, Manuela et al. (2015) Cobots: Robust symbiotic autonomous mobile service robots. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. IJCAI. AAAI Press. p. 4423–4429

31 Walker, T. T., Sturtevant, N. R., and Felner, A. (2018). Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI. AAAI Press. p. 534-540.

32 Werner, F. (2013). A survey of genetic algorithms for shop scheduling problems. In: *Heuristics: Theory and Applications*. p. 161–222.

33 Wilt, C. M., and Botea, A. (2014). Spatially distributed multiagent path planning. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

34 Wooldridge, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

35 Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine, v. 29, n. 1, p. 9-20,.