Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Distribuição Otimizada de Tarefas de Coleta, Transporte e Entrega a Múltiplos Agentes Robóticos num Cenário de Restrição de Tempo e Recursos Energéticos

José Ronaldo Mouro

JUIZ DE FORA
DEZEMBRO, 2023

Distribuição Otimizada de Tarefas de Coleta, Transporte e Entrega a Múltiplos Agentes Robóticos num Cenário de Restrição de Tempo e Recursos Energéticos

José Ronaldo Mouro

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Heder Soares Bernardino

Coorientador: Alex Borges Vieira

JUIZ DE FORA

DEZEMBRO, 2023

Distribuição Otimizada de Tarefas de Coleta, Transporte e Entrega a Múltiplos Agentes Robóticos num Cenário de Restrição de Tempo e Recursos Energéticos

José Ronaldo Mouro

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Heder Soares Bernardino
Doutor em Modelagem Computacional

Alex Borges Vieira
Doutor em Ciência da Computação

Lorenza Leão Oliveira Moreno
Doutora em Informática

Luciana Brugiolo Gonçalves
Doutora em Ciência da Computação

JUIZ DE FORA

14 DE DEZEMBRO, 2023

Resumo

Deslocar múltiplos agentes autônomos confinados em um ambiente restrito, atribuindo a cada um deles tarefas de coleta e entrega de itens em posições e momentos diversos é um problema de otimização combinatória conhecido como *Multi-Agent Pickup and Delivery(MAPD)*. As restrições cinemáticas do ambiente, a quantidade de agentes disponíveis e as suas localizações físicas no ambiente, os locais de coleta e entrega das tarefas aliados ao instante temporal no qual essas se apresentam para serem executadas suscitam soluções diversificadas que impõem, cada qual, seu próprio custo em tempo de execução de todas as tarefas. *MAPD* é um problema que encontra residência em modernos sistemas de armazéns, portos, aeroportos e instalações industriais, onde frotas robôs se deslocam para atender tarefas transporte ou reboque de carga continuamente. Nesse contexto, este trabalho objetiva apresentar abordagens que incluem a minimização do consumo energético desses agentes como mais um objetivo a ser alcançado. O consumo de energia dessas máquinas é diferenciado em função do estado em que se encontram. Assim sendo, a abordagem realizada adota valores de consumo distintos para agentes estacionados, em movimento e em transporte. São apresentados, em dois eixos de desenvolvimento, quatro abordagens de algoritmos resolvedores, sendo duas heurísticas e duas meta-heurísticas. Esses abordagens foram submetidas a experimentos utilizando instâncias referenciadas na literatura e outra proposta neste trabalho. Foram realizadas comparações de resultados entre as abordagens propostas e, no que se refere ao *makespan*, com algoritmos considerados pertencer ao estado da arte. Os resultados alcançados evidenciam a eficácia dos algoritmos propostos e apontam caminhos de aprimoramento.

Palavras-chave: *Multi-Agent Pickup and Delivery(MAPD)*, Otimização Combinatória, Consumo de Energia, Algoritmo Genético.

Abstract

Displacing multiple autonomous agents confined in a restricted environment, assigning each of them tasks of collecting and delivering items at different positions and times is a combinatorial optimization problem known as *Multi-Agent Pickup and Delivery(MAPD)*. The kinematic restrictions of the environment, the number of available agents and their physical locations in the environment, the collection and delivery locations of tasks combined with the time in which they are presented to be executed give rise to diverse solutions that impose, each one, its own cost in execution time of all tasks. *MAPD* is a problem that finds residence in modern warehouse systems, ports, airports and industrial facilities, where fleets of robots move around to perform cargo transportation or towing tasks continuously. In this context, this work aims to present approaches that include minimizing the energy consumption of these agents as another objective to be achieved. The energy consumption of these agents differs depending on the state they are in. Therefore, the approach used here adopts different consumption values for stationary, moving and transport agents. Four approaches to solving algorithms are presented in two development axes, two heuristics and two meta-heuristics. These approaches were subjected to experiments using instances from the literature and another proposed in this work. Comparisons of results were made between the proposed approaches and, with regard to *makespan*, with algorithms considered to be state-of-the-art. The results achieved demonstrate the effectiveness of the proposed algorithms and point out paths for improvement.

Palavras-chave: *Multi-Agent Pickup and Delivery(MAPD)*, Combinatorial Optimization, Energy Consumption, Genetic Algorithm.

Agradecimentos

Agradeço, primeiramente, a Deus, por permitir minha caminhada até aqui, bem como tudo que se sucedeu nesse transcurso.

Aos meus queridos pais, por me amarem incondicionalmente, me educarem com exemplos sólidos de corretude, dispendendo de boa parte de suas existências nesse mister.

Aos meus filhos amados, pela compreensão, pelo apoio e brilho nos seus olhos ao verem seu velho pai lutando para dar-lhes bons ensinamentos e exemplos de vida.

Aos professores do Instituto de Ciências Exatas, especialmente aqueles do Departamento de Ciência da Computação, pela dedicação ao transmitir-me valiosos ensinamentos que levarei por toda minha vida pós-acadêmica.

À minha namorada e aos amigos, por me incentivarem e torcerem por mim desde o meu ingresso na Universidade.

Aos professores Heder e Alex, pelas seguras orientações que nortearam a realização deste trabalho de conclusão de curso, sem as quais nada teria a apresentar neste momento.

Aos colegas de curso, pela colaboração durante as atividades e trabalhos em grupo, e pela cordialidade e empatia no convívio acadêmico diário.

Conteúdo

Lista de Figuras

—-Glossário ————

Lista de Abreviações

BTT	*Backward Threshod Token*
CE	Caminho Estimado
CR	Caminho Real
GAT	*Genetic Algorithm Token*
GATP	*Genetic Algorithm Token Passing*
GPPT	*Greedy Path Planner Token*
GPPTP	*Greedy Path Planner Token Passing*
MAPD	*Multi-Agent Pickup and Delivery*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
RCE	Regime de Consumo Energético
TDC	Taxa de Distanciamento de Caminho
TP	*Token Passing*
TPTS	*Token Passing with Task Swaps*
TTP	*Threshold Token Passing*

1 Introdução

Nos últimos anos, segundo Salzman e Stern (2020), houve um grande aumento nas aplicações e pesquisas envolvendo problemas que incluem a movimentação de frota de agentes robôs. Os robôs de armazém desenvolvidos pela Amazon[1] são exemplos dessas iniciativas. Nesses armazéns tecnológicos, um grande número de robôs opera autonomamente coletando, transportando e entregando itens de estoque. Um volume cada vez maior de transações comerciais online, defende Oliveira et al. (2022), inviabiliza a operação manual de armazéns logísticos e exige o desenvolvimento e uso de tecnologias de automação. Esses Armazéns inteligentes, explica JVD e W (1999), compreendem a aplicação de técnicas de Inteligência Computacional, juntamente com tecnologias de Internet das Coisas (IoT), para realizarem abordagens analíticas que empregam equipes de robôs para realizarem a separação de pedidos.

A distribuição otimizada de tarefas de coleta, transporte e entrega de itens realizada por múltiplos agentes robóticos, que operam num ambiente conhecido e sob um cenário de restrição de recursos, é a temática principal deste trabalho. Os armazéns ou centros de distribuição inteligentes são um bom exemplo de local de residência para esse problema. No entanto, como alega Queiroz et al. (2020), existem vários outros onde agentes robóticos precisam operar eficientemente, coexistindo em um ambiente conhecido, frente a uma alta demanda por tarefas que envolvem percorrer caminhos de qualidade, sem a ocorrência de colisões e em tempo real. As frotas de veículos autônomos de reboque de aeronaves em aeroportos, segundo Morris et al. (2016), e os sistemas de robôs de escritório, segundo Veloso et al. (2015), são outros locais onde esse problema encontra aplicação.

Multi-Agent Pickup and Delivery (MAPD) é um conhecido problema da literatura no qual, explica Ma et al. (2017), os agentes devem atender a um fluxo de tarefas de entrega que podem surgir no sistema a qualquer momento. Tais tarefas são caracterizadas por um local de coleta e um local de entrega. Um agente ocioso, que não está executando

[1] https://www.amazon.jobs/en/teams/amazon-robotics (último acesso em 14/12/2023)

alguma tarefa, pode ser atribuído a uma ainda não executada, devendo primeiramente se mover de sua localização atual para o local de coleta e, depois, para o local de entrega da tarefa, evitando colisões com outros agentes e obstáculos fixos do ambiente.

MAPD, segundo Ma et al. (2017), quando comparado ao problema *Multi-Agent Pathfinding (MAPF)*, um problema predecessor e muito pesquisado, apresenta vantagem ao capturar características importantes de muitos domínios do mundo real, como no caso dos armazéns automatizados, onde os agentes estão constantemente engajados em novas tarefas. No *MAPF*, o número de agentes é igual ao número tarefas, e a instância do problema *MAPF* é resolvida quando todos os agentes alcançam seus destinos. O conceito de tafera, no *MAPF*, é mais simples que no *MAPD*, devendo o agente ir somente de um ponto a outro do ambiente em caminho único, não havendo, portanto, o conceito de deslocamento para o ponto de coleta e, depois, para o ponto de entrega da tarefa.

O conhecimento prévio sobre o ambiente permite o planejamento dos caminhos a serem percorridos pelos agentes. Isso recai num subproblema de busca pelo melhor caminho num ambiente co-habitado por múltiplos agentes. Nesse problema de busca, os agentes se deslocam simultaneamente livres de colisões, ou seja, além das restrições do ambiente, há restrições impostas pelo posicionamento dos próprios agentes.

A demanda *online* por execução das tarefas implica em imprevisibilidade e obriga o sistema a buscar melhores soluções considerando o estado atual dos agentes no ambiente e a especificidade das tarefas.

Neste trabalho, foi proposta e abordada uma versão do problema *MAPD* em que nele, adicionalmente, inclui-se a otimização do consumo energético. Foi considerado que os agentes, a depender do seu estado atual, consomem uma quantidade diferenciada de energia, o que desvincula parcialmente a otimização desse recurso do tempo de execução das tarefas. Ficou, portanto, estabelecido que os agentes consomem energia segundo um regime que inclui os seguintes estados: parado, deslocando e transportando (deslocando com carga).

Dois eixos principais para abordagem do problema foram seguidos com o intuito de resolvê-lo. O primeiro deles inclui duas heurísticas que estabelecem metas ou limites para de consumo energético que restrinjam a escolha das tarefas que os agentes irão

executar ou estabelece como será executada determinada tarefa. Nesse último caso, permitindo o fracionamento da tarefa de modo que ela possa ser executada em sub-tarefas que não comprometam as metas de consumo. Ambos os algoritmos apresentaram baixo custo computacional e atingiram médias de tempo de execução das tarefas próximas às apresentadas pelas abordagens de referência para o problema *MAPD*. Esse algoritmos, apesar de buscar a minimização em dois objetivos, se colocaram, em *makespan*, entre o *TP* e o *TPTS*, propostos em (MA et al., 2017). Como em (QUEIROZ et al., 2020), o segundo eixo buscou solucionar o problema combinando algoritmo genético multi-objetivo e heurística de planejamento de caminhos. O *NSGA-II* foi utilizado para escalonamento das tarefas e duas heurísticas distintas de execução desse escalonamento foram propostas. A primeira dessas heurísticas considera a possibilidade de substituição de uma tarefa anteriormente atribuída por outra; a segunda, segue o escalonamento anterior caso um novo seja apresentado, não permitindo substituições de tarefas atribuídas.

Foram propostos dois tipos de função-objetivo para avaliação das soluções geradas pelo algoritmo genético, qual sejam: aproximada e real. A função real, como foi implementada, apesar de retornar os custos das soluções precisamente, apresentou elevado custo computacional, o que a torna inviável para uso em sistemas de tempo real. As funções aproximadas foram consideradas de custo computacional aceito para o problema tratado, mas mostraram pouca precisão em avaliar as soluções. Os experimentos realizados geraram resultados que permitiram comparações entre as abordagens propostas, bem como comparação com outras presentes na literatura. Esses experimentos utilizaram instâncias propostas neste trabalho e instâncias existentes na literatura.

2 Fundamentação Teórica

Nesse capítulo são explicados fundamentos teóricos sobre o problema *MAPD*, as adaptações a ele propostas para atenderem os objetivos de estudo deste trabalho e conceitos sobre a meta-heurística usada na obtenção de soluções. Na Seção 2.1, o problema *MAPD* é formulado, sendo apresentadas as principais definições, condições e conceitos envolvidos. Na Seção 2.2, as instâncias do problema *MAPD* são explicadas, especialmente o que vem a ser uma instância bem formada para o problema. A Seção 2.3 discorre sobre alguns métodos que resolvem o problema *MAPD*. A Seção 2.4 trata sobre as métricas para avaliação das soluções dos problemas *MAPF* e *MAPD*. Na seção 2.5, é apresentado o trabalho de (DEB et al., 2002) no qual é proposto o *NSGA-II*, um algoritmo genético multi-objetivo baseado no conceito de fronteiras de dominância.

2.1 Definição do Problema *MAPD* e Adaptação Realizada

Uma instância do problema MAPD, segundo Ma et al. (2017), consiste em m agentes $A = a_1, a_2, ..., a_m$ e um grafo conectado não direcionado $G = (V, E)$ cujos vértices em V correspondem às localizações e cujas arestas em E correspondem às conexões entre localizações, pelas quais os agentes podem seguir em frente. Seja $l_i(t) \in V$ denotando a localização do agente a_i no *timestep* discreto t. O agente a_i inicia em sua localização inicial $l_i(0)$. A cada *timestep* t, o agente permanece em sua localização atual $l_i(t)$ ou move-se para uma localização adjacente, ou seja, $l_i(t+1) = l_i(t)$ ou $(l_i(t), l_i(t+1)) \in E$.

Os agentes, segundo Ma et al. (2017), precisam evitar colisões entre si, sendo essas de dois tipos:

1. dois agentes não podem estar no mesmo local no mesmo intervalo de tempo, ou seja, para todos os agentes a_i e $a_{i'}$ com $a_i \neq a_{i'}$ e todos os intervalos de tempo t: $l_i(t) \neq l_{i'}(t)$; e

2. dois agentes não podem se mover ao longo da mesma aresta em direções opostas no mesmo intervalo de tempo, ou seja, para todos os agentes a_i e $a_{i'}$ com $a_i \neq a_{i'}$ e todos os *timestepes* t: $l_i(t) \neq l_{i'}(t+1)$ e t: $l_i(t+1) \neq l_{i'}(t)$.

Um caminho é uma sequência de localizações associadas a intervalos de tempo, ou seja, um mapeamento de *timestepes* para localizações. Dois caminhos colidem se os dois agentes que se movem ao longo deles colidem. Considere um conjunto de tarefas T que contém o conjunto de tarefas não executadas. Em cada *timestep*, o sistema adiciona todas as novas tarefas ao conjunto de tarefas.

Cada tarefa $t_j \in T$ é caracterizada por um local de coleta $s_j \in V$ e um local de entrega $g_j \in V$.

Um agente é chamado de livre se não estiver executando tarefa no momento. Caso contrário, é chamado de ocupado. Um agente livre pode ser atribuído a qualquer tarefa $t_j \in T$. Ele, então, tem que se mover de sua localização atual, passando pelo local de coleta s_j até o local de entrega g_j da tarefa. Quando o agente chega ao local de coleta, ele começa a executar a tarefa e a tarefa é removida de T. Ao chegar ao local de entrega, termina a execução da tarefa, o que implica que fica novamente livre e não está mais atribuído à tarefa. Qualquer agente livre pode ser atribuído a qualquer tarefa no conjunto de tarefas. Um agente pode ser atribuído a uma tarefa diferente no conjunto de tarefas enquanto ainda está se movendo para o local de coleta da tarefa à qual está atribuído no momento mas, primeiro, deve concluir a execução da tarefa depois de atingir o local de coleta. Essas propriedades modelam tarefas de entrega em que os agentes podem frequentemente receber novas tarefas antes de pegarem um bem, mas, uma vez pego o bem precisam obrigatoriamente entregá-lo. O objetivo é terminar a execução de cada tarefa o mais rápido possível.

Consequentemente, a eficácia de um algoritmo *MAPD* é avaliada pelo número médio de *timestepes*, chamado de *service time*, necessário para concluir a execução de cada tarefa depois que ela foi adicionada ao conjunto de tarefas. Um algoritmo *MAPD* resolve uma instância *MAPD* se o *service time* resultante de todas as tarefas for limitado.

Neste trabalho, é seguida a definição original do problema *MAPD*, à exceção objetivo de terminar a execução de cada tarefa o mais rápido possível. Ao ser incluído

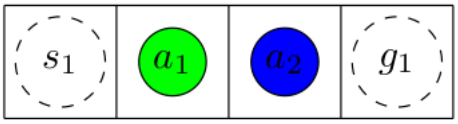

Figura 2.1: Instância *MAPD* (extraído de (MA et al., 2017)).

o objetivo adicional de minimizar o consumo energético, optou-se por uma formulação de duplo objetivo, qual seja: terminar todas as tarefas o mais rápido possível e com menor consumo energético por parte dos agentes. O trabalho também incorpora outra excepcionalidade ao considerar que uma tarefa pode ser concluída por mais de um agente mediante o seu fracionamento. Tal fracionamento se dá quando a etapa de transporte da tarefa é subdividida novas tarefas.

2.2 Instâncias para o problema *MAPD*

Nem toda instância *MAPD* é solucionável. A Figura 2.1 mostra um exemplo com dois agentes livres a_1 e a_2 onde nenhum agente pode terminar a execução da tarefa t_1 com local de coleta s_1 e local de entrega g_1.

Ma et al. (2017) forneceu, intuitivamente, uma condição suficiente que torna as instâncias *MAPD* solucionáveis, ou seja, instâncias bem formadas: os agentes só devem poder descansar (ou seja, permanecer para sempre) em localizações, chamadas *endpoints*, onde não bloqueiam outros agentes. O conjunto V_{ep} de *endpoints* de uma instância *MAPD* contém os locais iniciais dos agentes, os locais de coleta e entrega de tarefas, e talvez locais adicionais de estacionamento designados. Seja V_{tsk} o conjunto de todos os locais possíveis de coleta e entrega de tarefas, chamados de extremidades da tarefa, conjunto V_{ep}/V_{tsk} é chamado de conjunto de *endpoints* não relacionados à tarefa.

Ma et al. (2017) define que uma instância *MAPD* é bem formada se:

1. o número de tarefas for finito;
2. não houver menos *endpoints* não relacionados a tarefas do que o número de agentes;
3. para quaisquer dois *endpoints*, existe um caminho entre eles que não atravessa nenhum *endpoint*.

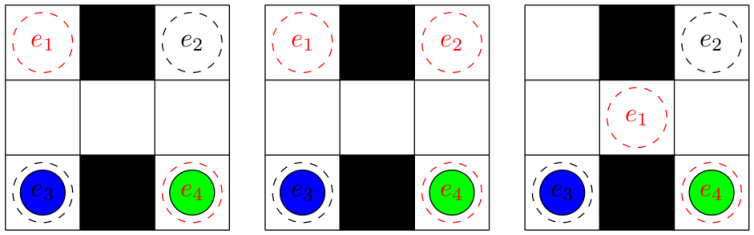

Figura 2.2: Instâncias *MAPD* (Extraído de (MA et al., 2017). A figura mostra três instâncias MAPD. Células pretas são bloqueadas. Os círculos azuis e verdes são as localizações iniciais dos agentes. Os círculos tracejados vermelhos são os pontos finais da tarefa. Os círculos tracejados pretos são pontos de extremidade não relacionados à tarefa.

Instâncias *MAPD* bem formadas (com pelo menos uma tarefa), portanto, têm pelo menos $m + 1$ *endpoints*.

A Figura 2.2 mostra três instâncias *MAPD*. A instância à esquerda está bem formada. Ao centro, encontra-se uma instância mal formada porque há dois agentes, mas apenas um terminal não-tarefa. A instância à direita é mal formada porque, por exemplo, todos os caminhos entre os pontos finais e_2 e e_3 atravessam o ponto final e_1.

2.3 Alguns métodos que resolvem o problema *MAPD*

Ma et al. (2017) apresentou dois algoritmos desacoplados que resolvem instâncias bem formadas do problema *MAPD*. O mais simples deles foi nomeado de *Token Passing (TP)*. Uma versão melhorada e mais eficaz que o *TP* recebeu o nome de *Token Passing with Task Swaps (TPTS)*. O termo "desacoplados"relacionado a esses algoritmos se referem ao fato de que cada agente atribui a si mesmo tarefas e calcula seus próprios caminhos livres de colisão com algumas informações globais.

2.3.1 *Token Passing (TP)*

Para Ma et al. (2017), o *TP* é baseado em uma ideia semelhante à *Cooperative pathfinding* proposto por Silver (2005), onde os agentes planejam seus caminhos um após o outro. Seu conjunto de tarefas contém todas as tarefas que não possuem agentes atribuídos a elas. O *token* é um bloco de memória compartilhado sincronizado que contém os caminhos atuais de todos os agentes, o conjunto de tarefas e as atribuições de agentes. O *TP* assume que

um agente permanece no último local de seu caminho no *token* quando chega ao final dele.

Como os caminhos de custo mínimo precisam ser encontrados apenas para os pontos finais, os custos do caminho de todos os locais para todos os pontos finais são calculados em uma fase de pré-processamento e, em seguida, usados como valores h para todas as pesquisas A*.

O algoritmo *TP* funciona da seguinte forma:

1. o sistema inicializa o *token* com os caminhos triviais onde todos os agentes ficam em suas localizações iniciais;

2. em cada *timestep*, o sistema adiciona todas as novas tarefas, se houver, ao conjunto de tarefas;

3. qualquer agente que tenha atingido o fim de seu caminho no *token* solicita o *token* uma vez por *timestep*;

4. o sistema então envia o *token* para cada agente que o solicita, um após o outro;

5. o agente com o *token* escolhe uma tarefa do conjunto de tarefas de forma que nenhum caminho de outros agentes no *token* termine no local de coleta ou entrega da tarefa;

6. se houver pelo menos uma dessas tarefas:

   (a) o agente atribui a si mesmo aquela com o menor valor h de sua localização atual para o local de coleta da tarefa e remove essa tarefa do conjunto de tarefas;

   (b) o agente então atualiza seu caminho no *token* com um caminho de custo mínimo que:

      i. se move de seu local atual através do local de retirada da tarefa para o local de entrega da tarefa; e

      ii. não colide com os caminhos de outros agentes armazenados no *token*.

7. se não houver tal tarefa:

(a) então o agente não atribui a si mesmo uma tarefa no *timestep* atual. Se o agente não estiver no local de entrega de uma tarefa no conjunto de tarefas, ele atualizará seu caminho no *token* com o caminho trivial onde ele permanece em seu local atual; e

(b) caso contrário, para evitar *deadlocks*, ele atualiza seu caminho no *token* com um caminho de custo mínimo que:

i. move de seu local atual para um *endpoint* de modo que os locais de entrega de todas as tarefas, no conjunto de tarefas, sejam diferentes do local *endpoint* escolhido e nenhum caminho de outros agentes no *token* termina no *endpoint* escolhido; e

ii. não colide com os caminhos de outros agentes armazenados no *token*.

2.3.2 *Token Passing with Task Swaps (TPTS)*

Para Ma et al. (2017), a passagem de *token* com troca de tarefas *(TPTS)* é semelhante ao *TP*, exceto que seu conjunto de tarefas agora contém todas as tarefas não executadas, em vez de apenas todas as tarefas que não têm agentes atribuídos. Isso significa que um agente com o *token* pode se atribuir não apenas a uma tarefa que não tem nenhum agente atribuído, mas também a uma tarefa que já está atribuída a outro agente, desde que esse agente ainda esteja se movendo para o local de coleta da tarefa. Isso pode ser benéfico quando o agente a ser atribuído à tarefa pode se mover para o local de coleta da tarefa em menos intervalos de tempo do que o agente que dela estava anteriormente atribuído. O último agente não é mais atribuído à tarefa e não precisa mais executá-la. O ex-agente, portanto, envia o *token* para o último agente para que o último agente possa tentar atribuir-se a uma nova tarefa.

2.3.3 Algoritmo Genético (AG) combinado a algoritmos definidores de melhor caminho

Em (QUEIROZ et al., 2020), se considerou uma versão *"online"* do *MAPF*, sendo proposta a combinação de um algoritmo genético (AG) com o *Improved Conflict Based Search*

(ICBS) e o *Prioritized Planning (PP)* para resolver o problema *MAPD*. Nessa proposta, as tarefas são alocadas pelo AG, enquanto *ICBS* e *PP* são usados para definir os caminhos. O *ICBS* é uma versão modificada do *Conflict-Based Search (CBS)* projetado para o planejamento de caminhos de vários agentes.

Sempre que novas tarefas são adicionadas, o AG inicia para definir quais tarefas os agentes irão executar. Duas variantes do AG foram propostas: GA-TA, onde as soluções candidatas são representadas por um vetor de índices de tarefas e um vetor de índices de agentes; e GA-E, onde apenas as tarefas são representadas. Para ambos, GA-TA e GA-E, o comprimento do cromossomo é o número de tarefas atualmente no problema.

O algoritmo inicia com *timestep* zero e todos os agentes estão livres. Enquanto o critério de parada não for satisfeito, o *timestep* é aumentado uma unidade cada vez que o *loop* é executado. O critério de parada é atendido quando não há mais tarefas a serem adicionadas. Quando uma tarefa é adicionada ao conjunto de tarefas, o AG é usado para atribuir as tarefas aos agentes. A posição final de um agente ocupado é o local de entrega da tarefa que está sendo executada. O AG considera para o agente ocupado o *timestep* e o local em que o agente está novamente livre, escolhendo a melhor tarefa para o agente executar após terminar sua tarefa. Além disso, o AG pode modificar as atribuições das tarefas não concluídas e das que não estão sendo executadas. O AG retorna uma sequência de tarefas a serem executadas para cada agente. O planejamento do trajeto ocorre toda vez que a posição final de um agente é modificada. A localização do objetivo de um agente livre é a posição de coleta da tarefa definida pelo AG. Para um agente ocupado, o local de destino é o local de entrega de sua respectiva tarefa atual. *ICBS* ou *PP* são executados apenas quando um local de destino é definido ou alterado. A cada iteração, o algoritmo verifica se o agente já concluiu uma tarefa ou se já atingiu a posição de retirada de uma tarefa. Os agentes podem ser atribuídos a qualquer tarefa quando chegam à posição de entrega. Caso contrário, o agente está ocupado e sua tarefa atual não pode ser modificada. O processo evolutivo considerado foi composto pelas estratégias de seleção parental, cruzamento, mutação e substituição. *Makespan* é a função objetivo. As soluções são selecionadas para reproduzir por uma seleção de torneio e o procedimento de substituição mantém os melhores indivíduos atuais na população (elitismo).

Prioritized Planning (PP)

PP é um algoritmo A* modificado para focar na solução de problema multiagente, onde cada agente tem pleno conhecimento dos outros agentes e suas rotas planejadas. O algoritmo resolve o problema de busca otimizando o caminho de cada agente. Após o planejamento, ele preenche uma "tabela de reservas" com informações sobre qual vértice está ocupado em determinado tempo para que outro agente da lista planeje seu caminho com eficiência, verificando tal tabela e evitando futuras colisões. A tabela de reservas é tratada como uma grade tridimensional, sendo duas dimensões espaciais e uma terceira dimensão representando o tempo (SILVER, 2005). O algoritmo escalona os agentes, priorizando aqueles com maiores *timesteps* estimados para executar a tarefa. O primeiro agente a escolher seu caminho tem maior prioridade, pois a "tabela de reservas" está vazia e, portanto, há menos obstáculos a serem evitados (menos colisões). Dessa forma, agentes com maior tempo estimado para concluir sua tarefa têm menos restrições, o que pode resultar em menor *makespan*.

2.4 Métricas para avaliação de resultados

2.4.1 Métricas usadas em trabalhos da Literatura

Em (STERN et al., 2019), para o problema *MAPF*, são apontadas duas funções mais comumente usadas pelos pesquisadores para avaliação de soluções, quais sejam: a minimização do *makespan* e do *flowtime*. Em (MA et al., 2017), para o problema *MAPD*, se considera usar como métrica para as funções de minimização o *makespan* e *service time*. Embora não tenha sido elencada para uso no problema *MAPD*, *flowtime* poderia ser igualmente nele empregado.

Para o problema *MAPF*, *makespan* é o número de intervalos de tempo para todos os agentes atingirem seu objetivo, ou seja, a localização ao final do deslocamento; e no *MAPD*, é o total de *timestep* necessário para que todas as tarefas sejam entregues, desde a primeira a entrar até a última a ser entregue.

Para o problema *MAPF*, *flowtime* é a soma dos *timestep* requeridos para que todos os agentes atinjam seus objetivos.

Para o problema *MAPD*, *service time* considera as tarefas individuais, a quantidade de *timestep* entre uma tarefa entrar no sistema e aquela tarefa ser entregue.

2.4.2 Métricas da utilizadas neste trabalho

Neste trabalho, são usados o *makespan* e consumo energético como métricas de avaliação das soluções encontradas. O consumo energético é um valor inteiro que representa uma quantidade de unidades de energia consumida por intervalo de tempo.

2.5 *Non-dominated Sorting Genetic Algorithm II (NSGA-II)*

O *NSGA-II* (DEB et al., 2002) é um algoritmo evolucionário multi-objetivo baseado em ordenação do conjunto de soluções em subconjuntos de soluções não dominadas também conhecidos como fronteiras de dominância. Uma fronteira de dominância contém soluções que não dominam umas às outras, mas são dominadas por soluções das fronteira anteriores a ela, à exceção da primeira fronteira, que contém todas as soluções não dominadas. Se diz que uma solução domina a outra, quando os valores dos seus objetivos de otimização são iguais ou melhores, devendo, pelo menos um deles, ser melhor. Na figura 2.3, é exemplificada essa relação de dominância, podendo se verificar, por exemplo, que as soluções $S_9 e S_1 0$ são dominadas pela solução S_7. Pode-se verificar também as soluções, que estão representadas por pontos coloridos, se encontram organizadas em fronteiras de dominância de acordo com a cor dos pontos. Ao ordenar o conjunto de soluções dessa forma, é garantido que a primeira fronteira, chamada de Fronteira de Pareto, contenha as melhores soluções para o problema, e as demais, à medida que se segue a ordenação, contêm, cada vez mais, piores soluções.

Na ausência de informações adicionais, não há como apontar uma melhor solução existente dentro de cada uma das fronteiras de dominância, e é, por esse motivo, que o *NSGA-II* busca encontrar não a melhor solução para o problema, mas sim o maior e mais diversificado possível conjunto de soluções não dominadas. Uma vez obtido esse conjunto de soluções não dominadas, um decisor é usado para a escolha da melhor solução. É

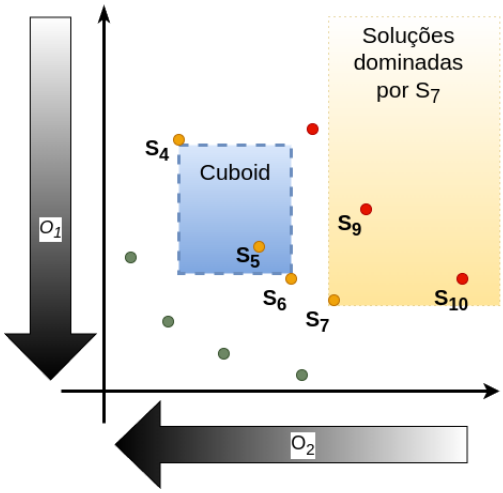

Figura 2.3: O plano cartesiano representa em seus eixos dois objetivos de minimização. Um conjunto de pontos de mesma cor representa fronteiras de dominância. Os pontos verdes representam a primeira fronteira de soluções não dominadas (Fronteira de Pareto). O retângulo azulado representa um cuboide formado por soluções vizinhas à S_5. O retângulo amarelado representa uma região onde se encontram soluções dominadas por S_7.

importante salientar que, ao se percorrer a fronteira de em busca de uma melhor solução, a escolha por melhores valores de um objetivo implicam em valores iguais ou piores dos demais objetivos.

Para definir e ordenar os subconjuntos de soluções correspondentes às fronteiras de dominância, é usado o algoritmo *Fast Non-dominated Sorting (FNS)* de complexidade $O(MN^2)$, onde M é o número de objetivos a serem otimizados e N, a quantidade de soluções existentes. A Figura 2.4 traz o pseudo código do *FNS*. O algoritmo primeiramente encontra a Fronteira de Pareto, inicializa um mapa de contadores de dominância para todas as soluções e um conjunto de soluções dominadas. As soluções cujo valor do contador for zero pertencem à primeira fronteira. Feito isso, o algoritmo entra em um ciclo e, nele, permanece enquanto houver fronteiras a serem definidas. O contador de dominância serve para atribuir as soluções à uma fronteira, o que é feito quando seu valor é zero. Nesse ciclo, a cada iteração, é usada a fronteira anterior para decrementar o contador de cada solução pertencente ao conjunto de solução dominadas, caso exista a solução dessa fronteira que domine a solução do conjunto de soluções dominadas.

Para garantir a diversidade de soluções é usado o algoritmo *Crowding Distance Sorting (CDS)*, que calcula a distância de aglomeração das soluções pertencentes à uma fronteira de dominância, como instrumento para rejeitar soluções pertencentes a essa fron-

```
fast-non-dominated-sort(P)
for each p ∈ P
  S_p = ∅
  n_p = 0
  for each q ∈ P
    if (p ≺ q) then                    If p dominates q
      S_p = S_p ∪ {q}                  Add q to the set of solutions dominated by p
    else if (q ≺ p) then
      n_p = n_p + 1                    Increment the domination counter of p
  if n_p = 0 then                      p belongs to the first front
    p_rank = 1
    F_1 = F_1 ∪ {p}
i = 1                                  Initialize the front counter
while F_i ≠ ∅
  Q = ∅                                Used to store the members of the next front
  for each p ∈ F_i
    for each q ∈ S_p
      n_q = n_q − 1
      if n_q = 0 then                  q belongs to the next front
        q_rank = i + 1
        Q = Q ∪ {q}
  i = i + 1
  F_i = Q
```

Figura 2.4: (Extraído de (DEB et al., 2002). A figura mostra em pseudo-código o algoritmo *Fast Non-dominated Sorting (FNS)*.

```
crowding-distance-assignment(I)
l = |I|                                number of solutions in I
for each i, set I[i]_distance = 0      initialize distance
for each objective m
  I = sort(I, m)                       sort using each objective value
  I[1]_distance = I[l]_distance = ∞    so that boundary points are always selected
  for i = 2 to (l − 1)                 for all other points
    I[i]_distance = I[i]_distance + (I[i + 1].m − I[i − 1].m)/(f_m^max − f_m^min)
```

Figura 2.5: (Extraído de (DEB et al., 2002). A figura mostra em pseudo-código o algoritmo *Crowding Distance Sorting (CDS)*.

teira. As soluções a serem rejeitadas são aquelas com menores distâncias de aglomeração. Para o cálculo da distância de aglomeração de uma solução, a fronteira é ordenada com base em valores de um objetivo e, para cada solução é tomada a distância entre os pontos referentes às soluções vizinhas. Isso feito para todos os objetivos, e essas distâncias para cada solução são somadas. Esses resultados servem como estimativas do perímetro do cubóide formado usando os vizinhos mais próximos como vértices. A figura 2.3 mostra um exemplo do que vem a ser esse cubóide.

Como ilustrado na figura 2.6, a cada iteração do *NSGA-II* a quantidade de soluções é dobrada mediante a seleção de soluções geradoras por torneio binário, recom-

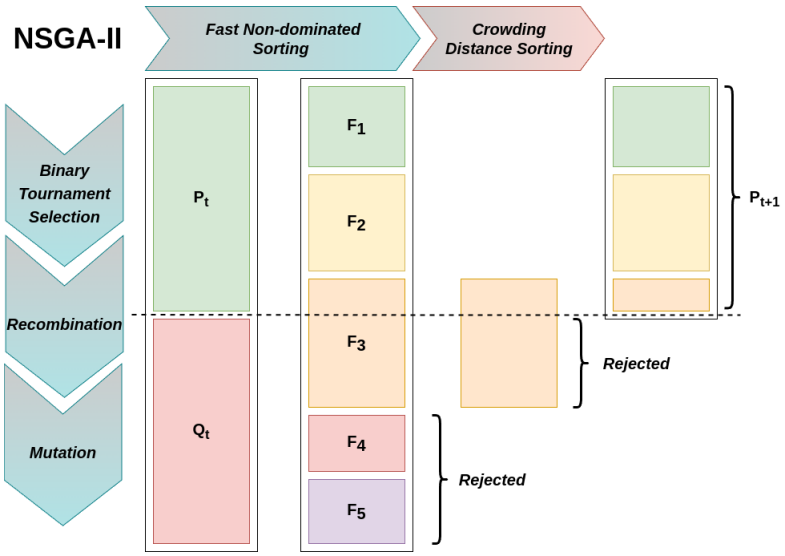

Figura 2.6: Ilustra as ações percorridos durante uma iteração do *NSGA-II* (DEB et al., 2002). As setas verticalmente descendentes representam ações de expansão do conjunto de soluções e as setas horizontais, ações necessárias à redução do conjunto de soluções.

binação das soluções geradoras para obtenção de novas soluções e mutação dessas novas soluções geradas. Após dobrar o número de soluções é aplicado o *FNS* para a definição dos subconjuntos de soluções não dominadas. Uma vez definidos esses subconjuntos, a população será reduzida à metade, eliminando-se as soluções pertencentes aos subconjuntos de soluções mais dominadas, e, caso seja necessário, para se atingir a redução desejada, é aplicado o algoritmo *CDS* para rejeitar as soluções com menores distâncias de aglomeração dentro de uma mesma fronteira de dominância.

3 Trabalhos Relacionados

Neste capítulo, são apresentados alguns trabalhos relacionados ao tema que são referenciais úteis para comparações de métodos que resolvem os problemas *MAPF* e *MAPD*. Na seção 3.1, é mencionado o trabalho de Hart, Nilsson e Raphael (1968) que apresentou o algoritmo de busca em grafos $A*$. Na seção 3.2, *Cooperative Pathfinding* de Silver (2005) é apresentado, demonstrando a possibilidade expandir a busca individual do algoritmo $A*$ de um ambiente espacial bidimensional para um ambiente tridimensional dinâmico, incorporando a dimensão tempo e a multiplicidade de buscas. Na seção 3.3, é apresentado o trabalho de Sharon et al. (2015), que propõe o algoritmo *Conflict Based Search (CBS)* para buscas em problemas *MAPF*, qual apresenta soluções ótimas. Na seção 3.4, é apresentado o trabalho de Svancara et al. (2019), que estuda as peculiaridades e complexidades da versão *online* de *MAPF*. Na seção 3.5, é apresentado o trabalho de Liu et al. (2019) o qual propõe dois novos algoritmos que melhoraram, segundo os autores, o estado da arte dos algoritmos *MAPD*. Na seção 3.6, é apresentado o trabalho de (MA et al., 2019) no qual foi proposto um algoritmo de busca combinatória que torna o *TP* ((MA et al., 2017)) mais eficiente.

3.1 *A Formal Basis for Heuristic Determination of Minimum Cost Paths*

Em (HART; NILSSON; RAPHAEL, 1968), foi discutido como informações a respeito do domínio de um problema podem ser matematicamente incorporadas à teoria de busca em grafos e apresentado um algoritmo geral para encontrar caminhos de custo mínimo em grafos. Nesse trabalho, restou provado que, sob leves pressupostos, esse algoritmo é ótimo, no sentido de que ele examina o menor número de nós do grafo necessário para garantir uma solução de custo mínimo.

A fim de buscar o caminho de custo mínimo, expandindo o menor número possível

de nós do grafo, segundo Hart, Nilsson e Raphael (1968), o algoritmo, que recebeu o o nome de *A**, deve constantemente manter-se informado, tanto quanto possível, sobre a decisão de qual próximo nó expandir. Caso seja expandido um nó que não pertença ao caminho mínimo, um esforço desnecessário foi realizado. Neste contexto, a função de avaliação sobre qual nó deve ser expandido considera o custo real do caminho percorrido até o nó atual somado uma estimativa de custo para o restante do caminho (função heurística). Hart, Nilsson e Raphael (1968) sustenta que muitos problemas possuem informações físicas que podem servir como função heurística admissível e usou, como exemplo para a sua análise, um grafo representando cidades e rodovias unindo-as. Como heurísticas admissíveis para o exemplo de estudo foram apontadas as distâncias euclidiana e de Manhattan, sendo discutido genericamente o custo computacional frente a admissibilidade das tais heurísticas.

3.2 *Cooperative Pathfinding*

Em (SILVER, 2005), foram apresentados três algoritmos para resolver o problema *MAPF* de forma mais robusta e efetiva em ambientes de tempo real, são eles: *Cooperative A*(CA*)*, *Hierarquical Cooperative A*(HCA*)* e *Windowed Hierarquical Cooperative A*(WHCA*)*. Os algoritmos assumem que os agentes se deslocam no ambiente, planejando cada qual seu caminho, tendo pleno conhecimento da posição de todos os outros agentes e suas rotas planejadas.

Cooperative A(CA*)*

O algoritmo *CA** decompõe a tarefa de deslocamento dos agentes como um todo em uma série de buscas individuais pelo melhor caminho. Para tal, usa o algoritmo A* e, como função heurística admissível a distância de Manhattan. Para que não haja colisão entre os agentes, essa busca ocorre num espaço-tempo tridimensional, cuja primeira e segunda dimensões representam o espaço e a terceira dimensão, o tempo. Uma ação de aguardar, ou seja, não se mover em determinado intervalo de tempo, foi incluída no conjunto de ações de movimento do agentes. Uma estrutura de dados chamada *reservation table* é usada para salvar o espaço de busca considerando o ambiente e a posição de cada agente

em determinado intervalo de tempo.

Silver (2005) adverte para o fato de que *CA** é sensível ao ordenamento do planejamento dos deslocamentos dos agentes, requerendo a adoção de prioridades para se alcançar boa performance. Em ambientes mais desafiadores, é apontada também a necessidade de adoção de melhores heurísticas que a distancia de Manhattan para melhoria da performance do algoritmo.

Hierarquical Cooperative A(HCA*)*

Segundo Silver (2005), há dois métodos para se melhorar heurísticas baseadas em abstrações do espaço de busca. O primeiro deles é se pre-computar todas as distâncias no espaço abstrato e armazená-las para uso futuro, o que não é viável para o problema *MAPF* porque o mapa do ambiente é dinâmico e os objetivos mudam ao longo do tempo. O segundo método é usar *Hierarquical A*(HA*)* proposto por Holte et al. (1996). Nessa abordagem, as distâncias abstratas são calculadas conforme a demanda, o que é mais apropriado em contextos dinâmicos. A hierarquia, nesse caso, refere-se a uma série de abstrações do espaço de estados, sendo cada uma mais genérica que a anterior.

*HCA** usa uma única hierarquia contendo uma abstração do domínio que ignora a dimensão tempo e a posição dos agentes, ou seja, considera apenas o mapa do ambiente bidimensional sem os agentes. A distância abstrata, nesse caso, é a estimativa perfeita da distância real para o deslocamento a ser feito pelo agente, o que é claramente uma heurística admissível com imprecisão apenas determinada pela dificuldade relacionada à iteração com os outros agentes.

Silver (2005) sustenta que uma das questões do *HCA** é qual a melhor forma de reusar os dados desse domínio abstrato. Nesse contexto, no foi adotada uma abordagem que usa o algoritmo *Reverse Resumable A*(RRA*)* para, sob demanda, guardar numa estrutura de dados as distâncias reais que vão sendo encontradas ao se aplicar o algoritmo *A** de forma reversa. Uma propriedade bem conhecida de *A** com uma heurística consistente é que a distância ideal do início até um nó é conhecida uma vez que o nó é expandido. O *RRA**, armazena, sob demanda, as distâncias reais como distâncias abstratas, e é requisitado somente quando não se conhece tais distâncias previamente.

Windowed Hierarquical Cooperative A*(WHCA*)

Silver (2005) apontou três problemas com os algoritmos *CA** e *HCA**, quais sejam:

1. alguns agentes, ao atingirem seu destino, podem bloquear a passagem dos demais em parte do mapa, por exemplo, em um corredor estreito do ambiente;

2. os algoritmos são sensíveis à ordem com qual as rotas dos agentes são calculadas, sendo necessária a adoção de estratégia mais robusta como a que dá a cada agente uma prioridade alta num curto espaço de tempo; e

3. os algoritmos calculam rotas para agentes de forma integral em um largo espaço tridimensional, sendo mais eficiente intercalar essas buscas em trechos menores até que os agentes atinjam seus objetivos, evitando, assim, a necessidade de planejar contingências de longo prazo que de fato podem até não ocorrer.

Para fazer frente aos problemas com os algoritmos *CA** e *HCA**, Silver (2005) apresentou uma simples solução que é "janelar"a busca, ou seja, limitar a busca cooperativa a uma profundidade especificada por uma janela de tempo w. No *WHCA**, cada agente procura uma rota parcial até seu destino e começa a segui-la. Em intervalos regulares (por exemplo, quando um agente está na metade de sua rota parcial), a janela é deslocada para frente e uma nova rota parcial é calculada. Para garantir que o agente siga na direção correta, apenas a profundidade de busca cooperativa é limitada a uma profundidade fixa, enquanto a busca abstrata é executada em profundidade total. Os agentes, então, são considerados apenas para w passos (através da *reservation table*) e são ignorados para o restante da busca. Um benefício adicional do janelamento é que o tempo de processamento pode ser distribuído por todos os agentes.

Silver (2005) testou seus algoritmos em uma série de 10 desafios caraterizados por ambientes construídos randomicamente contendo as localizações de início e objetivo dos agentes, como exemplificado na figura 3.1. Comparando-os também com a abordagem conhecida por *Local Repair A*(LRA*)*. *LRA** descreve uma família de algoritmos amplamente utilizados na indústria de videogames, na qual cada agente procura uma rota para o destino usando o algoritmo *A**, ignorando todos os outros agentes, exceto seus vizinhos atuais. Os agentes então começam a seguir suas rotas, até que uma colisão é

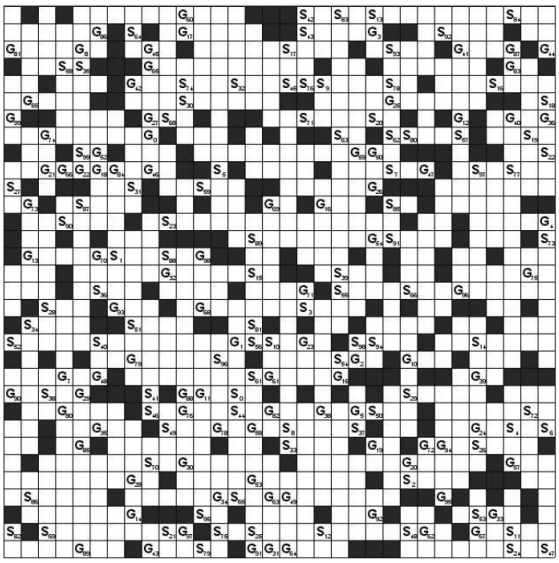

Figura 3.1: Exemplo de Ambiente. O agente i vai de S_i para G_i. (Extraído de (SILVER, 2005)).

iminente. Sempre que um agente está prestes a se mover para uma posição ocupada, ele recalcula o restante de sua rota.

Os métodos cooperativos de busca de caminhos são mais bem-sucedidos e encontram rotas de maior qualidade do que o A* com Local Repair. Infelizmente, o algoritmo CA* básico é caro para calcular, levando mais de um segundo para calcular 100 rotas. Usar uma heurística hierárquica reduz o custo de busca cooperativa, mas com uma sobrecarga no domínio abstrato. Embora ligeiramente inferior ao CA*, o custo inicial do HCA* ainda é muito alto para aplicações em tempo real. Janelando a pesquisa para uma profundidade fixa, o pré-cálculo pode ser reduzido para menos de 1ms por agente.

3.3 *Conflict-based search for optimal multi-agent pathfinding*

Em (SHARON et al., 2015), foi apresentado o *Conflict Based Search (CBS)*, um novo algoritmo ótimo de busca de caminhos multiagente. O *CBS* é um algoritmo de dois níveis em que, no mais alto nível, uma pesquisa é realizada em uma Árvore de Conflitos *Constraint Tree(CT)*, que é uma árvore baseada em conflitos entre agentes individuais. Cada nó dessa árvore representa um conjunto de restrições ao movimento dos agentes. No

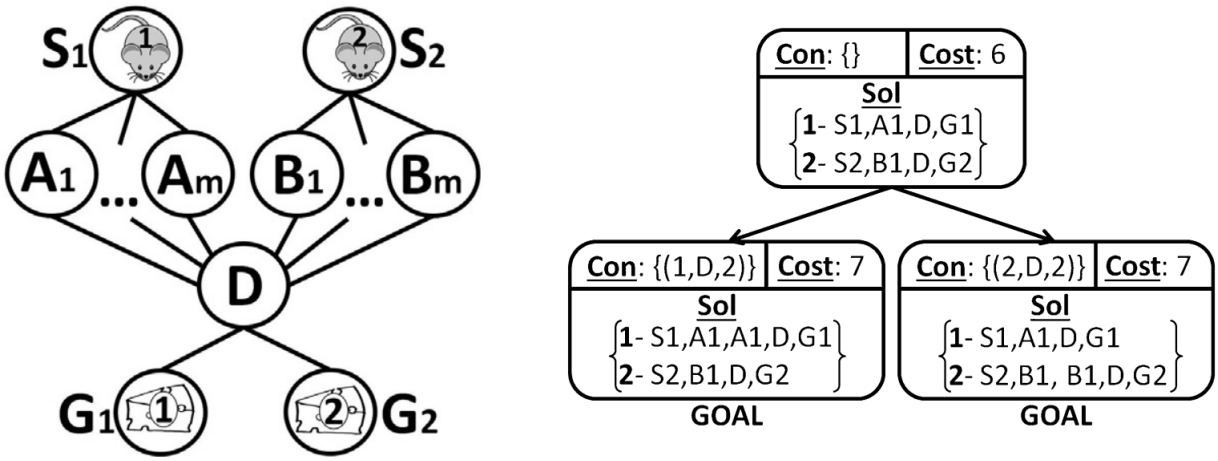

Figura 3.2: À esquerda, exemplo de instância contendo dois agentes(ratos), cada qual com suas localizações de saída e objetivo. À direita, a *CT* correspondente à instancia. (Extraído de (SHARON et al., 2015)).

nível baixo do algoritmo, pesquisas rápidas de agente único são realizadas para satisfazer as restrições impostas pela *CT*. Em muitos casos, essa formulação de dois níveis permite que o *CBS* examine menos estados do que *A**, mantendo a otimização.

Para Em Sharon et al. (2015), algoritmos para resolver o problema *MAPF* podem ser divididos em duas classes: solucionadores ótimos e sub-ótimos. Encontrar uma solução ótima é NP-difícil, pois o espaço de estados cresce exponencialmente com o número de agentes. Os solucionadores sub-ótimos geralmente são usados quando o número de agentes é grande e ótimos geralmente são aplicados quando o número de agentes é relativamente pequeno e a tarefa é encontrar uma solução ótima de custo mínimo. Naturalmente, os algoritmos de busca baseados em A* podem resolver o problema de maneira otimizada, mas podem ser executados por muito tempo ou esgotar a memória disponível.

No *CBS*, os agentes são inicializados com caminhos padrão, que podem conter conflitos. A busca de alto nível é realizada em *CT* cujos nós contêm restrições de tempo e localização para um único agente. Em cada nó de *CT*, uma busca de baixo nível é realizada para todos os agentes. A pesquisa de baixo nível retorna caminhos de agente único que são consistentes com o conjunto de restrições fornecidas em qualquer nó de *CT*. Se, após executar o nível baixo, ainda houver conflitos entre os agentes, ou seja, dois ou mais agentes estiverem localizados no mesmo local ao mesmo tempo, o nó de alto nível associado é declarado um nó sem objetivo e a pesquisa de alto nível continua

adicionando mais nós com restrições que resolvem o novo conflito. A ideia-chave do *CBS* é aumentar um conjunto de restrições e encontrar caminhos que sejam consistentes com essas restrições. Se esses caminhos tiverem conflitos e, portanto, forem inválidos, os conflitos serão resolvidos adicionando novas restrições. A Figura 3.2 mostra um exemplo de instância para o problema e a correspondente árvore de restrições.

3.4 *Online Multi-Agent Pathfinding*

Em (SVANCARA et al., 2019), foi estudada a versão *online* de *MAPF*, onde novos agentes aparecem ao longo do tempo. Algumas variantes de *online MAPF* foram definidas e analisadas teoricamente, mostrando que não é possível criar um resolvedor de *online MAPF* ótimo. Foi proposto, também, algoritmos *online MAPF* eficazes que equilibram a qualidade da solução, o tempo de execução e o número de alterações de plano que um agente faz durante a execução.

Segundo Svancara et al. (2019), pode-se imaginar muitas variantes de *online MAPF*, em particular com relação ao que acontece quando um agente atinge seu objetivo e o que acontece quando um novo agente aparece em sua localização inicial. Quando um agente atinge seu objetivo e, lá, permanece, isso resulta em um cenário semelhante mencionado em (MA et al., 2017). Uma opção diferente é que um agente desapareça ao atingir seu objetivo e, tal suposição faz sentido quando o objetivo está associado a algum local em que o agente possa realmente entrar e permanecer ali sem interferir nos caminhos de outros agentes. A decisão do que acontece com um agente quando ele aparece imediatamente em sua localização inicial também precisa ser considerada pois isso pode causar colisões com agentes já posicionados no ambiente. Nesse caso, se supôs que o agente precisa realizar uma ação de movimento para entrar em seu local de início, podendo aguardar o tempo oportuno a fim de evitar colisão com outro agente. Esta suposição corresponde a um cenário de vaga de estacionamento privado, onde o agente pode esperar nela, por exemplo, se perceber que sua localização inicial no grafo já está ocupada.

Para Svancara et al. (2019) uma maneira comum de analisar problemas e algoritmos *online* é considerar o comportamento de um solucionador ótimo *offline* para esse

mesmo problema. Um solucionador ideal *offline* é aquele que aceita todas as entradas para o problema *online* antecipadamente, sendo útil para análise, portanto, pois o solucionador *online* não pode fazer melhor do que o solucionador ideal *offline*. Nesse contexto, foi observado que:

1. se os agentes não desaparecem quando atingem seus objetivos, existem instâncias de problemas que podem ser resolvidas por um solucionador ideal *offline*, mas não podem ser resolvidas por nenhum solucionador *online MAPF* completo;

2. um problema *online MAPF*, onde os agentes desaparecem no objetivo e onde novos agentes podem esperar antes de entrar em sua localização inicial é solucionável se a parte *offline* do problema for solucionável, assumindo que existe um caminho para cada agente de sua localização inicial até sua localização final; e

3. não existe um solucionador *online MAPF* completo que possa garantir o retorno de uma solução com um custo igual ao solucionador *MAPF* ideal *off-line*.

Ao observar que não é possível garantir uma solução ótima com um solucionador completo para o problema *online MAPF*, Svancara et al. (2019) focou sua discussão sobre os algoritmos na parte *online* e assumiu que todos os algoritmos propostos começam com uma solução ótima para o problema *offline* inicial. Assim, descreveu apenas a função replanejar, que é chamada quando novos agentes aparecem. A entrada para essa função de replanejamento é sempre o conjunto de agentes atuais A, o conjunto de novos agentes $A+$ e o plano em andamento π_A, que é o plano que os agentes em A estão seguindo atualmente.

Primeiramente, foram, portanto, propostos dois algoritmos básicos, nos quais não é permitida a mudança no planejamento de caminhos dos agentes e, em decorrência disso, resultam em soluções de baixa qualidade, quais sejam:

1. *Replan Single (RS)* que busca um caminho ótimo para cada novo agente, um por vez, evitando todos os outros agentes já planejados;

2. *Replan Single Grouped (RSG)* procura caminhos ideais para todos os novos agentes de uma só vez;

Em seguida, Svancara et al. (2019) propôs um critério de qualidade de solução que os algoritmos *online MAPF* podem buscar em um esforço para alcançar uma melhor qualidade geral da solução. Esse critério recebeu o nome de *Snapshot Optimality*, que considera um planejamento ideal instantâneo é aquele com menor soma dos custos dos caminhos de todos os agentes até suas metas, assumindo que nenhum novo agente aparecerá no futuro. Com base nesse critério, foi proposto o algoritmo *Replan All (RA)*, que planeja de maneira ideal os caminhos de todos os agentes a partir de suas posições atuais sempre que um novo agente aparecer. O *RA* ignora totalmente o plano que os agentes existentes estavam seguindo. Isso pode ser um desperdício em termos de tempo de execução. Além disso, alterar a rota de um agente que já está em movimento, o que chamamos de reencaminhamento do agente, pode ser indesejável, pois requer comunicação com esse agente e modificar o plano do agente pode gerar alguma sobrecarga. Para fazer frente a isso, foi proposto um algoritmo que retorna soluções ótimas instantâneas, mas também tenta minimizar o número de reencaminhamentos. O algoritmo recebeu o nome de *Online Independence Detection (OID)*, uma vez que é baseado no algoritmo de *Independence Detection (ID)* em (STANDLEY, 2010). A ideia principal do *ID* é planejar cada agente separadamente, ignorando os outros agentes. Se houver um conflito entre os planos gerados, os agentes conflitantes serão mesclados em um grupo e replanejados juntos. Esse processo continua de forma iterativa até que não haja mais conflitos. A ideia principal do *OID* é muito semelhante, pois permite que os novos agentes planejem enquanto ignoram os outros agentes e, somente quando houver conflitos, os envolvidos serão agrupados para o replanejamento. *OID* pode resultar em planejamentos que não são ótimos. Com expectativa de aumentar a probabilidade de encontrar um planejamento ótimo, foi proposto o algoritmo parametrizável chamado *Suboptimal OID (SubID)* cuja heurística força a obtenção de soluções mais próximas da solução ótima.

3.5 *Task and Path Planning for Multi-Agent Pickup and Delivery*

Em (LIU et al., 2019), foi estudado o problema *MAPD offline*, no qual agentes, em um ambiente conhecido, executam um lote de tarefas, cada qual com o seu tempo de liberação, livres de colisões. No mesmo problema na versão *online*, pode-se utilizar essa versão *MAPD offline*, porém não se tem todas as informações disponíveis, uma vez que as tarefas entram no sistema de forma imprevisível. Nesse trabalho, foram apresentados dois novos algoritmos que melhoram o estado da arte de algoritmos *MAPD online*. São eles: *TA-Prioritized(TA-P)* e *TA-Hybrid(TA-H)*. Ambos, primeiro, atribuem as tarefas aos agentes e, então, calculam uma sequência de tarefas para cada agente resolvendo um especial Problema do Caixeiro Viajante, em que os agentes, ignorando colisões, visitam sequencialmente os locais de coleta e entrega de todas as tarefas a eles atribuídas. Os tempos de viagem estimados entre os diversos locais das tarefas são usados para minimizar o *makespan*.

3.5.1 *TA-Prioritized (TA-P)*

No *TA-P*, o planejando dos caminhos para cada agente é realizado em ordem decrescente dos tempos estimados de execução de suas sequências de tarefas, ou seja, priorizando os agentes com maiores tempos estimados de execução. Depois que um caminho foi planejado para um agente, os caminhos de todos os agentes restantes não podem colidir com ele, e, neste novo contexto do ambiente, são reavaliados os tempos de execução das tarefas do agentes restantes para a escolha do próximo agente a ter seu caminho calculado. Dessa forma, agentes com tempos de execução estimados maiores possuem menos restrições, o que pode resultar em um menor *makespan*.

3.5.2 *TA-Hybrid (TA-H)*

TA-H considera dois grupos de agentes para planejamento de caminhos e utiliza um método de planejamento de trajetos diferente para cada grupo, são eles:

1. Grupo 1 - Agentes de novas Tarefas. *TA-H* planeja sub-caminhos para os agentes

de seus locais atuais até os locais de entrega de suas tarefas atuais. Os agentes não podem trocar seus locais de entrega. Usa-se a busca baseada em conflito aprimorada (BOYARSKI et al., 2015), uma versão recente da *CBS* (SHARON et al., 2015), para executar o planejamento de caminho baseado em *MAPF*. Os caminhos resultantes permanecem inalterados até que os novos agentes de tarefa alcancem seus locais de entrega, pois o *TA-H* planeja caminhos apenas para novos agentes de tarefa, mas não para os outros agentes de tarefa.

2. Grupo 2 - Agentes Livres. *TA-H* planeja sub-caminhos para os agentes de seus locais atuais até os locais de coleta das próximas tarefas em suas sequências de tarefas. Pode-se ter planejado tais caminhos antes, no entanto, os caminhos podem ser melhorados enquanto os agentes os seguem. Nesse contexto, por exemplo, os agentes podem trocar seus locais de coleta. *TA-H* usa um algoritmo de fluxo mínimo de custo mínimo em tempo polinomial para executar o planejamento de caminho baseado em *AMAPF* para eles em cada etapa de tempo em que o conjunto de agentes livres foi alterado. O problema *AMAPF* é uma versão do problema *MAPF* onde agentes "anônimos" podem trocar seus locais de destino.

3.6 *Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery*

Em (MA et al., 2019), foi proposto um algoritmo de busca combinatória para planejamento de caminho de agente único, que torna o *TP* ainda mais eficiente e eficaz, chamado *Safe Interval Path Planning with Reservation Table (SIPPwRT)*. Esse algoritmo usa uma estrutura de dados avançada que permite atualizações e pesquisas rápidas dos caminhos atuais de todos os agentes em uma configuração online. Assim, o algoritmo *MAPD* resultante, *TP-SIPPwRT*, leva em consideração as restrições cinemáticas de robôs reais durante o planejamento, calculando movimentos contínuos de agentes com velocidades dadas em vez de movimentos discretos de agentes com velocidade uniforme.

Segundo Ma et al. (2019), *Space-time A** e *SIPP* (PHILLIPS; LIKHACHEV, 2011) são duas versões de *A** que planejam caminhos de tempo mínimo para os agentes

considerando os outros agentes como obstáculos dinâmicos. Ambas assumem movimentos discretos do agente com uma velocidade uniforme de tipicamente uma célula por unidade de tempo em uma grade. O *Space-time A** opera em pares de células e etapas de tempo, enquanto o *SIPP* agrupa etapas de tempo contíguas durante as quais uma célula não é ocupada em intervalos (de tempo) seguros para essa célula e, portanto, opera em pares de células e intervalos seguros. Assim, se a busca *A** do *SIPP* já encontrou um caminho que chega a alguma célula em algum momento durante algum intervalo seguro e depois descobre um caminho que chega à mesma célula em um momento posterior no mesmo intervalo seguro, então ele pode podar o último caminho sem perder a otimização.

SIPP representa o caminho de cada obstáculo dinâmico como uma lista ordenada cronologicamente de células ocupadas pelo obstáculo dinâmico, o que não é eficiente, pois o algoritmo precisa percorrer todas essas listas para calcular todos os intervalos seguros de uma determinada célula. Por outro lado, *Space-time A** mantém uma tabela de reserva indexada por uma célula e um intervalo de tempo, o que permite o cálculo eficiente de todos os intervalos seguros de uma determinada célula. O *SIPPwRT* melhora o *SIPP* usando uma versão de uma tabela de reserva que lida com movimentos contínuos de agentes com velocidades dadas e é indexada por uma célula.

Uma entrada na tabela de reserva de uma determinada célula é uma fila de prioridade que contém todos os intervalos reservados para essa célula em ordem crescente de seus limites inferiores. Um intervalo reservado para uma célula é um intervalo contíguo máximo durante o qual a célula é ocupada por algum obstáculo dinâmico. A tabela de reserva permite que o *SIPPwRT* implemente eficientemente todas as operações necessárias ao *TP-SIPPwRT* quais sejam:

1. calcular todos os intervalos seguros de uma determinada célula;
2. adicionar entradas na tabela de reservas após o cálculo de um novo caminho; e
3. excluir as entradas da tabela de reservas que se referem a intervalos passados, mantendo a tabela de reservas reduzida.

Segundo Ma et al. (2019), restou demonstrados os benefícios do *TP-SIPPwRT* para armazéns automatizados usando um simulador de agente com execução de caminho

perfeito e um simulador de robô padrão com execução de caminho imperfeito resultante de restrições cine-dinâmicas não modeladas e ruído de movimento pelos algoritmos *MAPD*.

4 Métodos Propostos

Os métodos propostos seguem dois eixos de abordagem distintos, quais sejam: meta-heurístico e heurístico. Neste capítulo são descritos os métodos propostos e suas características. Na Seção 4.1, são apresentados conceitos e decisões importantes a respeito dos métodos. Na Seção 4.2 é descrito o fluxo geral de execução dos algoritmos por eixo de abordagem, explicando como se dá a atribuição das tarefas pendentes de execução e o planejamento dos caminhos dos agentes no ambiente. Nas Seções 4.3 e 4.4, se discorre sobre as abordagens heurísticas e meta-heurísticas propostas, respectivamente.

4.1 Consumo Energético

Para atender o estudo realizado foi proposto o conceito de Regime de Consumo Energético (RCE), o qual estabelece que um agente consome energia segundo um dos seguintes estados: estacionado (em espera); em movimento sem carga (movimento); e em movimento com carga (transporte). O RCE, tentando simular o que normalmente acontece em casos reais, considera a seguinte regra: estacionado, o agente consome menos energia do que estando em movimento; e, transportando carga, mais energia do que movimentando-se sem carga. Essa regra busca similaridade observada no uso de máquinas que transportam materiais, onde estacionadas, elas não consomem energia para se movimentar, somente para manterem seus sistemas eletrônicos em funcionamento; movimentando-se, precisam manter seus sistemas e consumir energia para deslocarem sua massa; e transportando carga, precisam manter seus sistemas, deslocar sua massa e, adicionalmente, deslocar a massa da carga transportada.

4.1.1 Taxa de Distanciamento de Caminho (TDC)

Caminho Estimado (CE) é um caminho percorrido no ambiente desconsiderando a presença de agentes, sendo, portanto, sempre o menor caminho possível de ser realizado. Os CEs entre os *endpoints* conhecidos do ambiente são calculados na fase de pré-processamento,

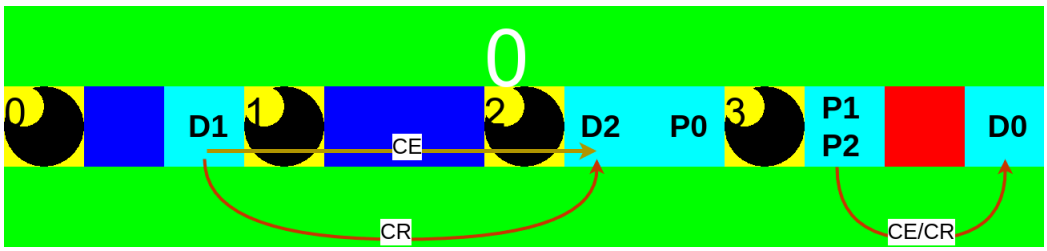

Figura 4.1: Exemplo de ambiente contendo agentes representados por círculos sólidos na cor preta. As cores amarela, azul claro e azul escuro representam *endpoints* e a cor vermelho, obstáculo do ambiente. As setas exemplificam caminhos no ambiente.

tornando os algoritmos propostos isentos desse custo computacional. Essa é mesma estratégia usada nos trabalhos que tratam do problema *MAPD*, e é razoável adotá-la pois, normalmente, num caso real de emprego dos algoritmos, se conhece, de projeto, essas localizações previamente.

Caminho Real (CR) é um caminho percorrido considerando a existência de agentes no ambiente, que se deslocam livres de colisões. O CR, portanto, é sempre um caminho maior ou igual ao CE, quando ambos possuem as mesmas localizações de início e término.

Taxa de Distanciamento de Caminho (TDC) é um valor correspondente à razão da distancia de um CE pela distância de um CR. Esse valor sempre se encontra no intervalo entre $0 e 1$, e serve de métrica de avaliação da qualidade do CR. Os algoritmos de propostos sempre calculam o menor caminho possível no ambiente, sendo esse caminho um CR. Se esse caminho calculado for igual ao CE, o valor da TDC será 1, ou seja esse caminho também é um CE. Dois algoritmos propostos vão usar essa TDC como informação para decisão a ser tomada.

A figura 4.1 ilustra um ambiente com a presença de agentes e dois exemplos comparativos entre CEs e CRs. No primeiro deles, há sinalizados com setas dois caminhos distintos entre as localizações $D1$ e $D2$, e nesse caso, a TDC é, aproximadamente, $0,55$. No segundo exemplo, é sinalizada a existência de um caminho entre as localizações $P1P2$ e $D0$, sendo verificado que, nesse caso, o CR é o mesmo que o CE, e a TDC será $1,0$.

4.2 Fluxo Geral de Execução dos Algoritmos Resolvedores

O fluxo geral de execução dos algoritmos propostos para resolverem o problema considera o ambiente populado por agentes que se encontram estacionados em *endpoints* em espera por tarefas, se deslocando para cumprir alguma tarefa, seja para um *endpoint* de coleta ou de entrega, ou, ainda, se deslocando para liberar um *endpoint* de tarefa no qual, eventualmente, se encontrava obstruindo o caminho de outro agente. A cada intervalo de tempo, sequencialmente, os algoritmos executam o seguinte:

1. verificação da entrada de novas tarefas no sistema com adição dessas num conjunto de tarefas pendentes;
2. atribuição de tarefas pendentes aos agentes;
3. planejamento dos caminhos dos agentes no ambiente; e
4. o deslocamento dos agentes para a sua próxima posição conforme os caminhos planejados.

4.2.1 Atribuição de Tarefas aos Agentes

A atribuição de tarefas aos agentes se dá de duas formas gerais distintas nos eixos de abordagem propostos: loteamento prévio de todas as tarefas pendentes entre os agentes ou seleção oportuna de tarefa a ser atribuída. Como o problema *MAPD* é um problema *online*, o loteamento prévio de tarefas necessita ser renovado quando novas tarefas entram no sistema. Nesse caso, há a possibilidade de que um loteamento realizado anteriormente seja diferente do atual em termos de ordem das tarefas atribuídas ou da própria atribuição das tarefas entres os agentes. A seleção oportuna de tarefa não demanda essa vinculação ao evento de entrada de novas tarefas no sistema, pois é feita diretamente sobre o conjunto de tarefas pendentes em momento oportuno.

A atribuição de tarefas pendentes difere nas abordagens propostas, sendo a heurística de cada uma dessas um dos principais fatores de diferenciação do funcionamento desses

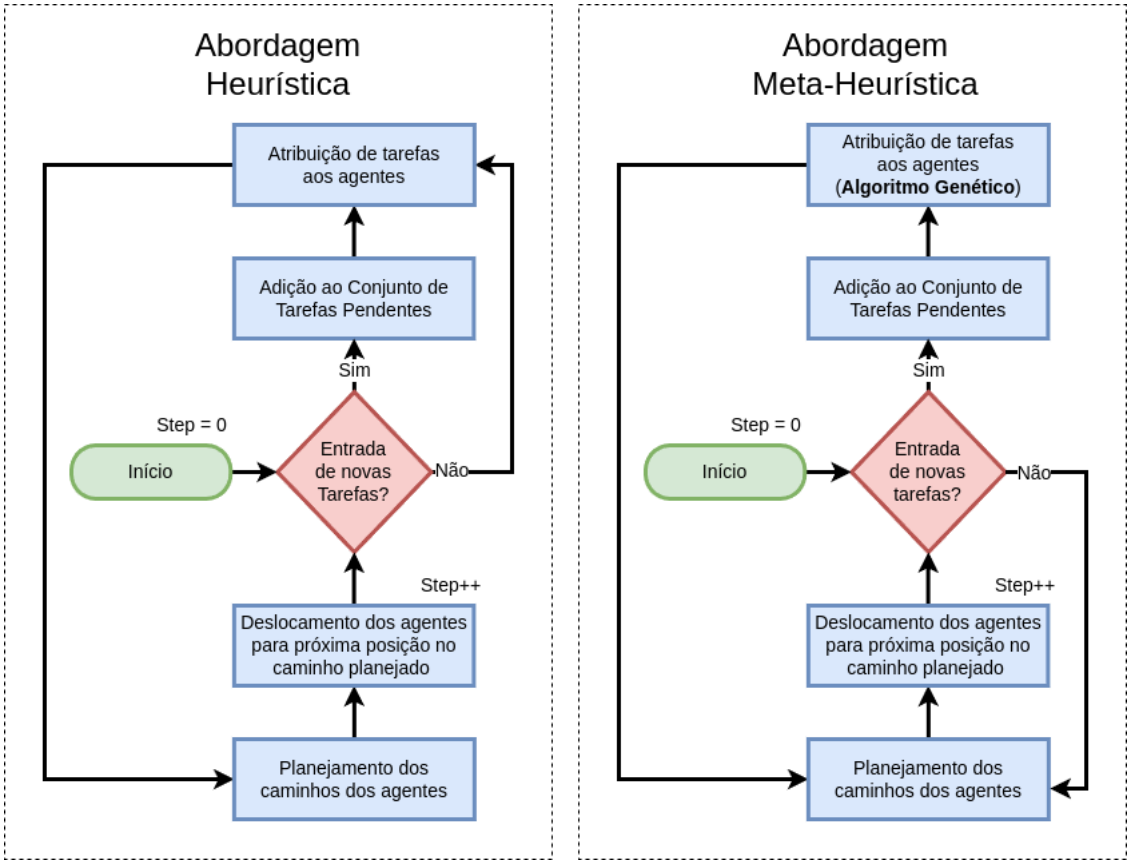

Figura 4.2: Fluxo Geral de Execução dos Algoritmos Resolvedores. O fluxo da esquerda corresponde ao eixo heurístico de desenvolvimento, e o da direita, ao eixo meta-heurístico.

algoritmos e, consequente, da qualidade das soluções encontradas. Os algoritmos que incluem meta-heurística buscam antecipar o processamento de atribuição de tarefas aos agentes, produzindo uma solução prévia a ser seguida até o final da execução de todas as tarefas pendentes, ou até que o conjunto de tarefas pendentes receba uma nova adição de tarefas. De outra forma, os algoritmos que usam heurísticas calculam, sob demanda, a atribuição dessas tarefas aos agentes, ou seja, quando o agente estiver livre, em determinado intervalo de tempo, essa atribuição será efetivada. A figura 4.2 ilustra os fluxos gerais de cada eixo de desenvolvimento, e essa diferença na forma como são feitas as atribuições de tarefas pode ser observada pelas saídas do componente decisor existente nesse fluxos.

4.2.2 Planejamento de Caminhos dos Agentes no Ambiente

Os algoritmos de planejamento de caminhos lidam não somente com os caminhos correspondentes às tarefas que vão sendo atribuídas, mas também com eventuais condutas para manter os agentes em suas posições atuais ou para que esses ocupem novas posições no ambiente, liberando assim a posição atual. Esse planejamento se faz atendendo uma ordenação entre os agentes, o que implica no fato de que os primeiros agentes a terem seus caminhos planejados tendem a executar caminhos de melhor qualidade, ou seja o CR se aproxima mais, em distância, do CE. O planejamento de caminhos, dependendo do algoritmo proposto, considera de forma distinta o conceito de agente livre. Alguns algoritmos, portanto, consideram que o agente está livre se ele não possui tarefa atribuída; outros consideram que o agente está livre se ele não está transportando carga, ou seja, executando de fato uma tarefa.

As heurísticas propostas planejam os caminhos dos agentes seguindo sempre a mesma ordenação rígida entre eles, e o fazem somente quando o agente atinge um *endpoint* em final de percurso do seu caminho atual. Assim sendo, nesse momento, os algoritmos decidem se o agente permanece na sua posição atual estacionado, se realiza deslocamento para cumprir uma tarefa atribuída ou se desloca para outro *endpoint* próximo, liberando a posição que se encontra. Esses algoritmos calculam os caminhos para realização das tarefas de uma só vez, ou seja, calculam um caminho até a posição de entrega da tarefa,

caminho esse que passa antes pela posição de coleta dessa mesma tarefa.

Os algoritmos que seguem o eixo meta-heurístico obtêm todo o planejamento de atribuição de tarefas previamente elaborado e planejam os caminhos dos agente seguindo a ordenação entre eles conforme estabelece o planejamento de atribuição em vigor. Assim, a cada novo planejamento, essa ordem entre os agentes pode mudar. Nesses algoritmos, como acontece nas heurísticas propostas, planeja-se, normalmente, em final de percurso dos caminhos em execução, os novos caminhos a serem realizados pelos agentes. Entretanto, há a possibilidade de modificação de um caminho em execução para atender um novo e diferenciado planejamento de atribuição de tarefas. Quando isso acontece, é calculado um novo caminho a partir da posição atual do agente para que ele realize outra tarefa atribuída ou para que ele se dirija para um *endpoint* próximo de sua localização onde permanecerá em estado de espera. Esses algoritmos que usam meta-heurística calculam os caminhos para realização das tarefas de duas vezes, ou seja, calculam primeiro um caminho até a posição de coleta da tarefa e, em momento futuro, quando o agente atingir essa posição, calculam o caminho para a posição de entrega da tarefa.

4.3 Abordagens Heurísticas

A fim de contemplar a necessidade de otimizar o *makespan* juntamente com o consumo energético, foram propostas duas abordagens inspiradas no *TP* que se utilizam de metas de TDC a serem prioritariamente atendidas. A ideia por trás de selecionar uma tarefa para ser atribuída cuja TDC atenda às metas é evitar caminhos muito congestionados por outros agentes, que resultam, consequentemente em maior tempo de execução das tarefas e maior consumo energético. Sabe-se que o RCE implica em consumo diferenciado para os agentes transportando carga ou movimentando-se sem carga, razão pela qual esse algoritmos possuem duas metas. Uma é aplicada em deslocamentos para coleta e a outra, em deslocamentos para entrega. Assim como no *TP*, as tarefas pendentes são ordenadas em função das distâncias dos CEs entre seus pontos de coleta da tarefa e a posição atual do agente a ser atribuído, e são também filtradas para evitar tarefas cujos seus *endpoints* estejam obstruídos pela presença de outros agentes. Cumprida essa etapa inicial de ordenamento e filtragem, as tarefas são, uma a uma, testadas segundo as metas

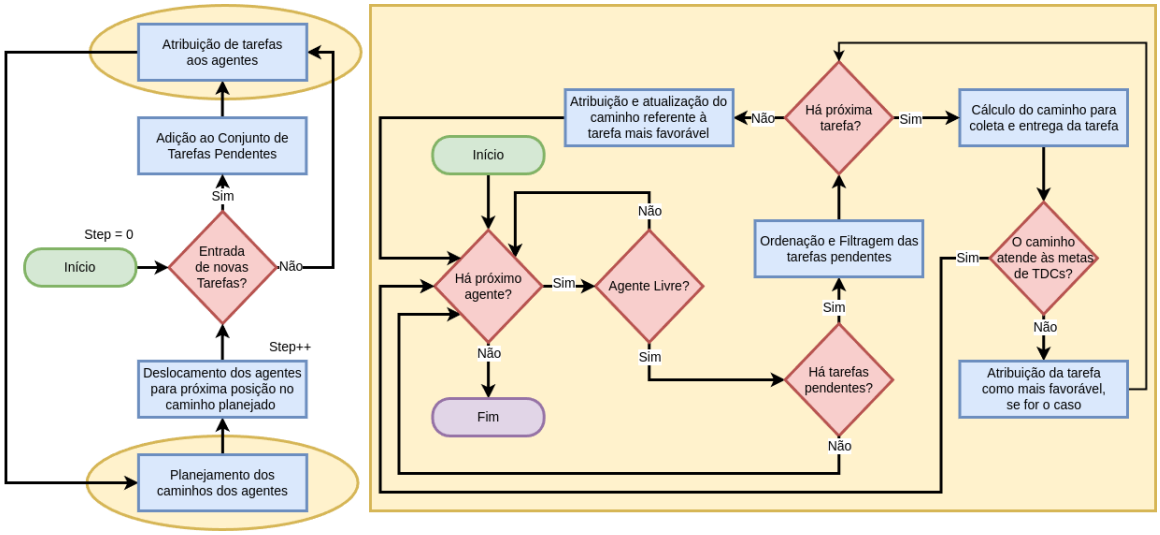

Figura 4.3: Fluxograma Geral do Resolvedor, à esquerda, e fluxograma relativo ao algoritmo *Threshold Token Passing (TTP)*, à direita.

de TDC impostas até que se encontre uma tarefa que as atenda plenamente. Caso não seja possível atender às metas, uma tarefa mais favorável será atribuída ao agente livre. É nessa fase de teste das metas e estratégia de escolha da tarefa a ser atribuída que os algoritmos *Threshold Token Passing (TTP)* e *Backward Threshod Token (BTT)* se diferem.

4.3.1 *Threshold Token Passing (TTP)*

O *Threshold Token Passing (TTP)* seleciona a tarefa a ser atribuída verificando se as metas impostas estão abaixo das TDCs correspondentes aos caminhos de coleta e de entrega da tarefa em questão. Caso alguma tarefa atenda integralmente às metas, a busca é interrompida, e a tarefa e seu caminho calculado serão atribuídos ao agente. Do contrário, a tarefa cujo o caminho é o mais favorável possível será atribuída. A tarefa de caminho mais favorável é aquela com maior valor de somatório dos TDCs de coleta e entrega. É importante salientar que para se obter a TDC, o algoritmo calcula o CR, e esse custo de calcular o CR até que se encontre uma tarefa compatível com as metas é o que diferencia o *TTP* do *TP* em termos de custo computacional. A figura 4.3 ilustra o fluxograma do algoritmo *TTP* e mostra, marcando com elipses na cor amarelada, a que estágios do fluxo geral do resolvedor o algoritmo corresponde.

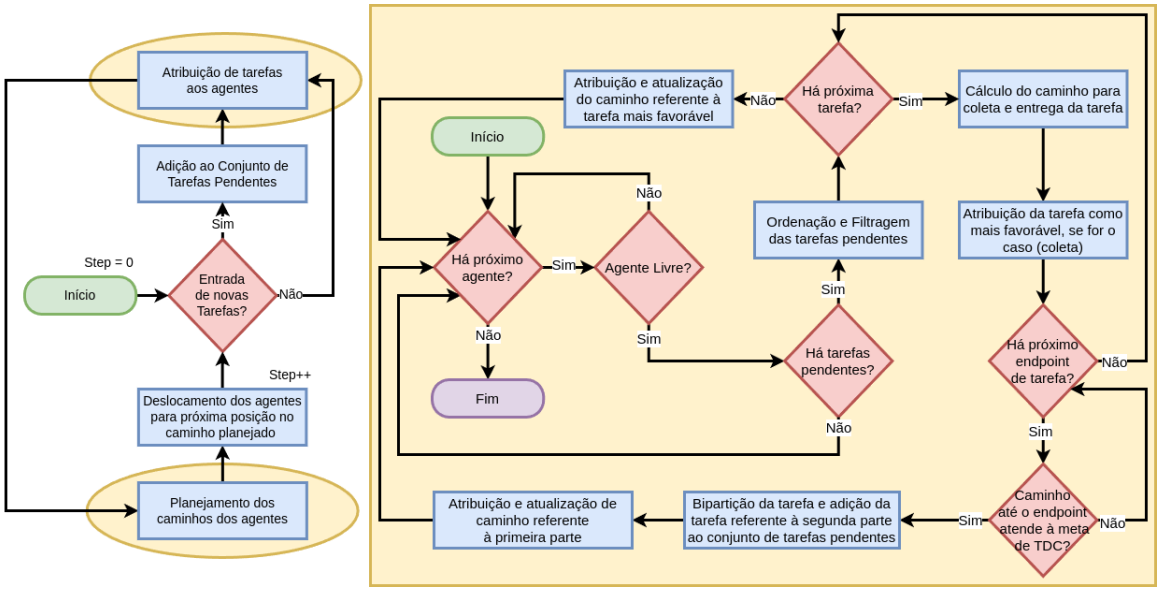

Figura 4.4: Fluxograma Geral do Resolvedor, à esquerda, e fluxograma relativo ao algoritmo *Backward Threshod Token (BTT)*, à direita.

4.3.2 *Backward Threshod Token (BTT)*

O *Backward Threshod Token (BTT)* é uma abordagem que muda o problema *MAPD* ao permitir que uma tarefa possa ser executada em várias etapas de transporte. A ideia por trás desse algoritmo é evitar o consumo de energia extra durante o transporte de carga, induzindo a escolha da tarefa a ser atribuída àquela que possa ser fracionada em duas para que, pelo menos, a primeira fração atenda a meta da TDC referente à entrega. Quando ocorre o tal fracionamento, uma tarefa complementar é inserida no sistema de forma que, no futuro, se efetive a finalização da tarefa original mediante a concretização dessa tarefa complementar. A ideia, portanto, é adiar passar por trechos congestionados por outros agentes durante a etapa de entrega, já que, nessa etapa, se gasta mais energia. Aliado a isso, aproximar o item a ser transportado do seu local de entrega.

O *BTT*, percorre, de trás pra frente, o CR de entrega das tarefas de interesse em busca de *endpoints* de tarefa. Ao encontrar algum desses *endpoints*, calcula a TDC até aquela localização. Caso esse *endpoint* não seja localização final de caminho de outros agentes e a meta imposta para entrega esteja abaixo da TDC calculada, essa localização será escolhida para o fracionamento da tarefa. O primeiro *endpoint* de tarefa encontrado nessa busca sempre será o *endpoint* de entrega da tarefa original a ser fracionada. A assim sendo, caso a TDC atenda nesta localização, a tarefa não será fracionada e será

atribuída integralmente ao agente. Caso, ao se percorrer todas as tarefas de interesse, não se encontre uma tarefa que atenda à política de fracionamento, a tarefa de caminho mais favorável possível será atribuída. Como no *TTP*, a tarefa de caminho mais favorável é aquela com maior valor de somatório dos TDCs de coleta e entrega, e há também o cálculo do CR para cada tarefa de interesse. Adicionalmente, há uma busca nos caminhos de entrega para selecionar *endpoints* candidatos a local de fracionamento da tarefa. A figura 4.4 ilustra o fluxograma do algoritmo *BTT* e mostra, marcando com elipses na cor amarelada, a que estágios do fluxo geral do resolvedor o algoritmo corresponde.

4.4 Abordagens Meta-Heurísticas

4.4.1 Algoritmo Genético

Alternativamente às abordagens heurísticas apresentadas neste trabalho, foi proposto também o emprego do *NSGA-II* para a obtenção de melhores soluções para o problema. Uma solução retornada pelo algoritmo genético define uma lista sequencial de tarefas a serem executadas para cada agente, bem como, a ordem entre os agentes através da qual será atualizado o caminho de cada um deles. Assim sendo, os algoritmos de planejamento de caminhos propostos nesta seção planejam com base na solução encontrada pelo algoritmo genético.

A figura 4.5 ilustra um problema e para esse, uma solução do algoritmo genético caracterizada como um indivíduo possuidor de genótipo e fenótipo. O problema contém um ambiente de tamanho 3×13 com quatro agentes (círculos sólidos na cor preta) e três tarefas pendentes sinalizadas com letras e números. As letras referem-se ao tipo de *endpoint* de tarefa, sendo a letra P *endpoint* de coleta, e a letra D de entrega. Os números referem-se ao identificador dessas tarefas. O genótipo é uma estrutura de dados que possui dois vetores, sendo um contendo os agentes presentes no ambiente e o outro as tarefas pendentes de execução. Os elementos desses vetores, seguindo a analogia biológica, seriam os genes, e os vetores o cromossomo. O tamanho do vetor de tarefas pendentes é um múltiplo do tamanho do vetor de agente, de tal forma que as tarefas estão associadas para atribuição aos agentes executores tomando-se múltiplos dos índices dos agentes no

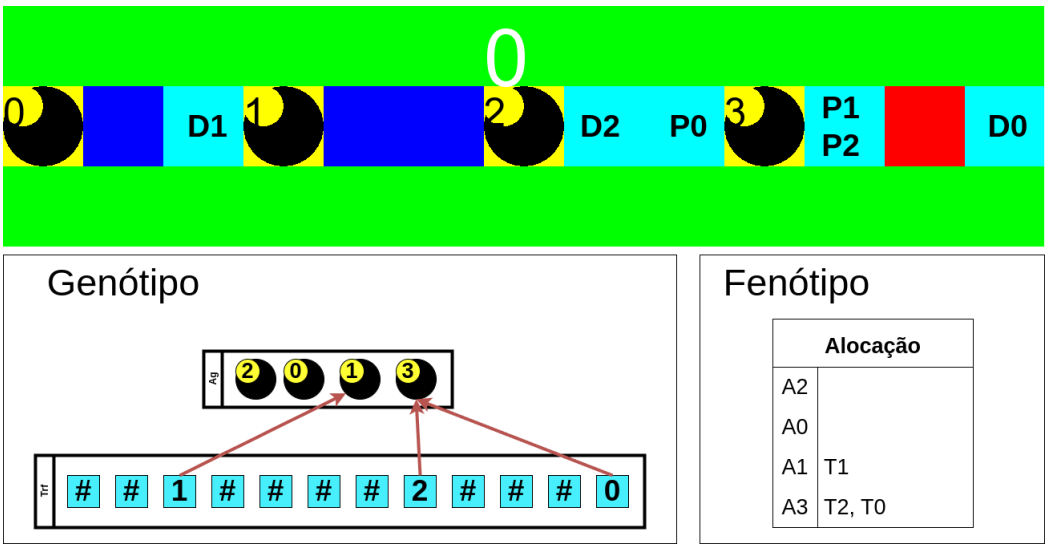

Figura 4.5: Ilustração de um problema, na parte de cima da figura, com um ambiente populado de agentes e *endpoints* de tarefa assinalados com letras e números indicando tarefas pendentes. Abaixo, se ilustra a representação de uma solução como um indivíduo possuidor de genótipo e fenótipo.

vetor de agentes.

A partir de duas soluções, como é mostrado na figura 4.6, por recombinação em uma única posição nos vetores de agentes e de tarefas, são gerados 4 (quatro) novas soluções da seguinte forma: a primeira parte dos vetores de agentes e de tarefas das soluções geradoras irão compor as primeiras partes dos vetores de duas das soluções geradas, ocorrendo o mesmo com as segundas partes das soluções geradoras envolvendo duas outras soluções geradas. Feito isso, as soluções geradas serão completadas com os agentes e tarefas que nelas faltam, seguindo a ordem dos elementos na solução geradora que não contribuiu, até então, com a construção da solução gerada. A posição nos vetores usada para a recombinação é definida por uma taxa que representa o tamanho da primeira fração dos vetores, de tal forma que, por exemplo, a taxa de $0,5$ indica que a posição de recombinação é exatamente a metade do vetor.

Algumas soluções geradas, após o processo de recombinação, sofrem mutação, que se dá por trocas de posição dos elementos dos vetores. Tanto a quantidade de soluções que sofrem mutação, quanto a quantidade de trocas de posição em cada um dos vetores são parametrizadas por percentuais sobre o número total de soluções e tamanho dos vetores de agentes e de tarefas.

O ciclo de gerações do algoritmo genético está ilustrado na figura 4.7, assim como

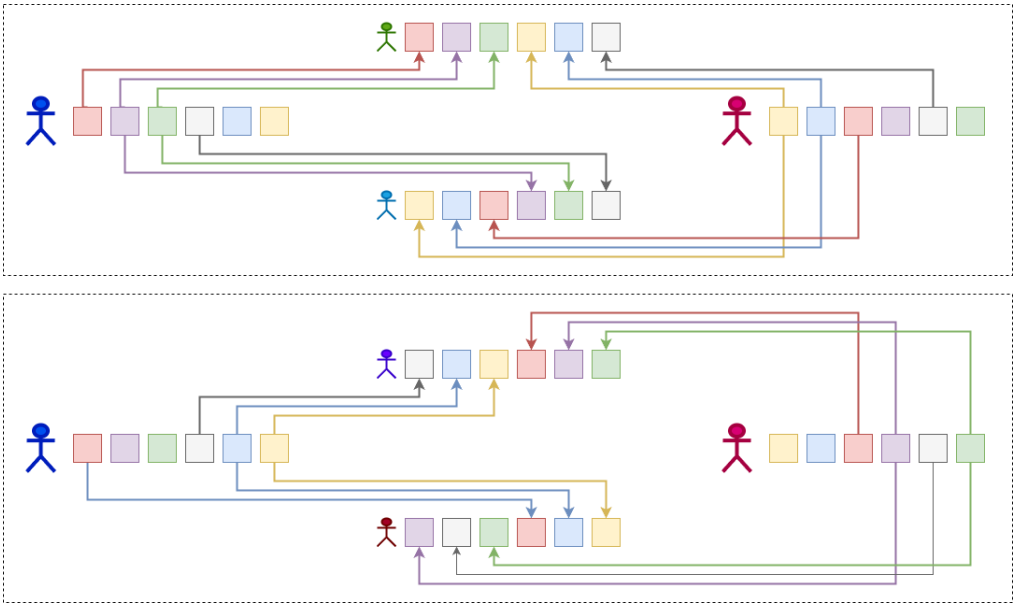

Figura 4.6: Ilustração do processo de recombinação de duas soluções geradoras para a criação de quatro novas soluções geradas.

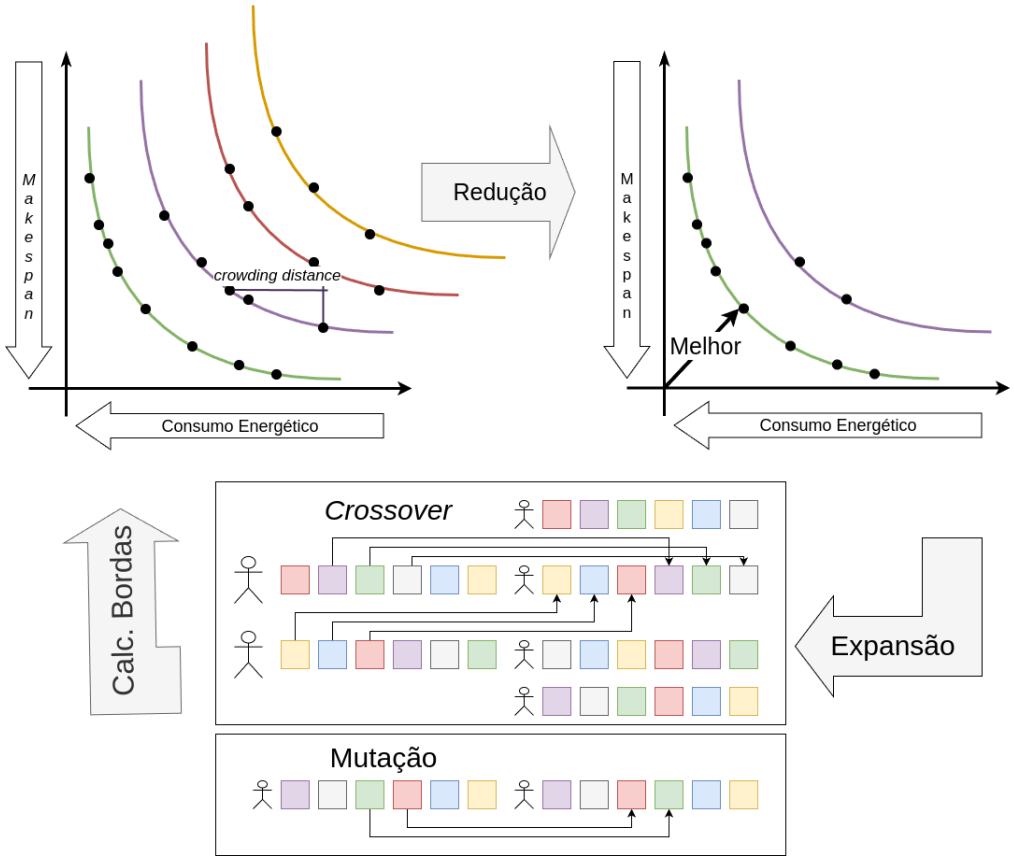

Figura 4.7: Ilustração do ciclo de gerações do algoritmo genético.

o processo de recombinação e mutação das soluções. O critério de parada do algoritmo utilizado é o número máximo de gerações, e, tendo alcançado esse número, o algoritmo retorna a melhor solução pelo critério da menor distância euclidiana do ponto formado pelo *makespan* e consumo energético no plano cartesiano e a origem do plano. Essas grandezas são primeiramente normalizadas para conferir um equilíbrio entre esses objetivos de minimização.

Antes e para se executar o ciclo de gerações do algoritmo genético, uma população inicial é gerada a partir de perturbações (mutação) realizadas em uma solução gulosa construída, que integrará também essa população. A quantidade de soluções (indivíduos) nessa população é um dos parâmetros a serem definidos.

A solução gulosa inicial é construída mediante a distribuição equitativa das tarefas aos agentes, considerando as proximidades desses aos *endpoints* de coleta das tarefas. A ordenação dos agentes no vetor de agentes, nessa solução construída, se dá priorizando os agentes que consomem mais energia nos deslocamentos transportando carga, sendo que os tamanhos dos CEs são utilizados para obtenção da estimativa desse consumo energético.

Funções Objetivo

Para o problema estudado neste trabalho, o consumo energético não está unicamente relacionado ao *makespan*. Caso os agentes sempre percorressem um CR idêntico ao CE, ou seja, um caminho sem outras restrições que não sejam aquelas impostas pelo ambiente, buscar soluções que atendam aos dois objetivos seria desnecessário. Bastaria tratar o problema como um problema mono-objetivo, buscando minimizar o *makespan*. Nesse contexto, a dificuldade é apresentar boas funções-objetivo capazes de avaliar eficaz e eficientemente as soluções. Este trabalho apresenta dois tipos de funções, quais sejam: aproximada e real. As funções objetivo aproximadas propostas buscam estimar o *makespan* e o consumo energético com base nos CEs percorridos pelos agentes, enquanto que a função objetivo real proposta, simulando a execução da solução a ser avaliada, obtém os CRs que os agentes percorrem e, dessa forma, os valores de *makespan* e consumo energético reais.

Os algoritmos das funções aproximadas, primeiramente, constroem para cada agente os CEs que eles poderão percorrer conforme o loteamento de tarefas definido na

solução e, dessa forma, tem-se estimado o *makespan* e o consumo de energia. Essa estimativa leva em conta o RCE. Como são CEs, é esperado que ocorra uma degradação dessa estimativa conforme o agente se desloca realmente no ambiente. Assim sendo, esses CEs podem divergir bastante dos CRs, sendo eventuais desvios e retardamentos para se evitar colisões entre os agentes responsáveis por isso. Um desvio é considerado quando o agente, ao se deslocar de um *endpoint* para outro, ocupa uma localização, em determinado intervalo de tempo, que não corresponde à localização no CE. Um retardamento é quando o agente permanece por mais de um intervalo de tempo seguidamente numa mesma posição do CE. O passo seguinte dos algoritmos é, tendo esses valores iniciais estimados, penalizar a estimativa feita, de alguma forma e, assim, buscar estimar também o impacto desses desvios e retardamentos de caminhos. A penalização se dá incrementando os valores estimativos inicialmente feitos com base em parâmetros definidos para essa finalidade. Essas funções aproximadas são parametrizadas com três valores de penalização, sendo um para *makespan*, e outros dois para deslocamentos com e sem carga.

Este trabalho testou quatro abordagens diferentes de funções aproximadas para tentar aproximar a ocorrência de desvios nos CEs, todas baseadas em colisões de caminhos estimados. Assim, quando um CE colide com outro no tempo e espaço, infere-se que ali haverá um desvio ou retardamento de caminho para se evitar uma colisão entre agentes. A forma como são contabilizadas essas colisões de CEs é o que diferencia as funções aproximadas umas das outras. São elas:

1. ***count***: conta todas as colisões ocorridas entre os CEs e penaliza conforme essa contagem, multiplicando os valores de penalização pela quantidade de colisões contada;

2. ***check***: verifica a ocorrência de colisões entre os CEs e penaliza uma única vez por CE verificado e tipo de colisão;

3. ***coll***: verifica a ocorrência de colisões na vizinhança das localizações dos CEs e penaliza uma única vez por CE verificado e tipo de colisão, aqui considerada essa vizinhança como as localizações adjacentes às localizações dos CEs; e

4. ***task***: conta ocorrência de colisões do CE com *endpoints* de tarefas executadas por outros agentes e penaliza conforme essa contagem, multiplicando os valores de

penalização pela quantidade de colisões contada.

A heurística das funções aproximadas que fazem contagem de colisões (*count* e *task*) busca quantificar o impacto das colisões estimadas, e a heurística das funções aproximadas que verificam somente a existência das colisões nos CEs (*check* e *coll*) busca qualificar esse impacto. A figura 4.8 exemplifica como se realizam os cálculos estimativos correspondentes às funções. Na parte superior da figura se encontram informações sobre os parâmetros de RCE e penalização usados e um problema com a respectiva solução. O problema contém um ambiente de tamanho 3×13 com quatro agentes (círculos sólidos na cor preta) e três tarefas pendentes sinalizadas com letras e números. As letras referem-se ao tipo de *endpoint* de tarefa, sendo a letra P *endpoint* de coleta, e a letra D de entrega. Os números referem-se ao identificador dessas tarefas. Seguindo para a parte inferior da figura, encontram-se os caminhos estimados e cálculos realizados correspondentes a cada função aproximada. Nos caminhos estimados, há elipses marcando as colisões de contabilizadas.

A função objetiva *real* clona as estruturas de dados que representam o estado atual do algoritmo de planejamento de caminhos e o aplica à solução a ser avaliada fazendo uso dessa estruturas clonadas, simulando assim uma execução real do planejamento de caminhos. A figura 4.9 mostra o fluxo de funcionamento da função e uma exemplificação da avaliação de uma solução mostrando os CRs planejados para os agentes no ambiente.

4.4.2 *Genetic Algorithm Token (GAT)*

O *Genetic Algorithm Token (GAT)* é um algoritmo planejador de caminhos que permite a substituição de tarefas anteriormente atribuídas por outra ou, simplesmente, anulação de atribuições feitas. Esse algoritmo, no momento da atribuição da tarefa, não a retira do conjunto de tarefas pendentes, só o fazendo após ser realizada a etapa de coleta dessa tarefa. Por conta disso, até o momento da coleta de uma tarefa, o agente é considerado livre e pode ser remanejado para cumprir outra tarefa ou ocupar uma posição de espera. O planejamento de caminhos para realizar tarefas é feito em duas etapas, sendo, primeiro calculado o deslocamento para coleta e, num momento futuro, quando o agente atinge o local de coleta, é feito o cálculo do caminho de entrega. Em situações em que agentes

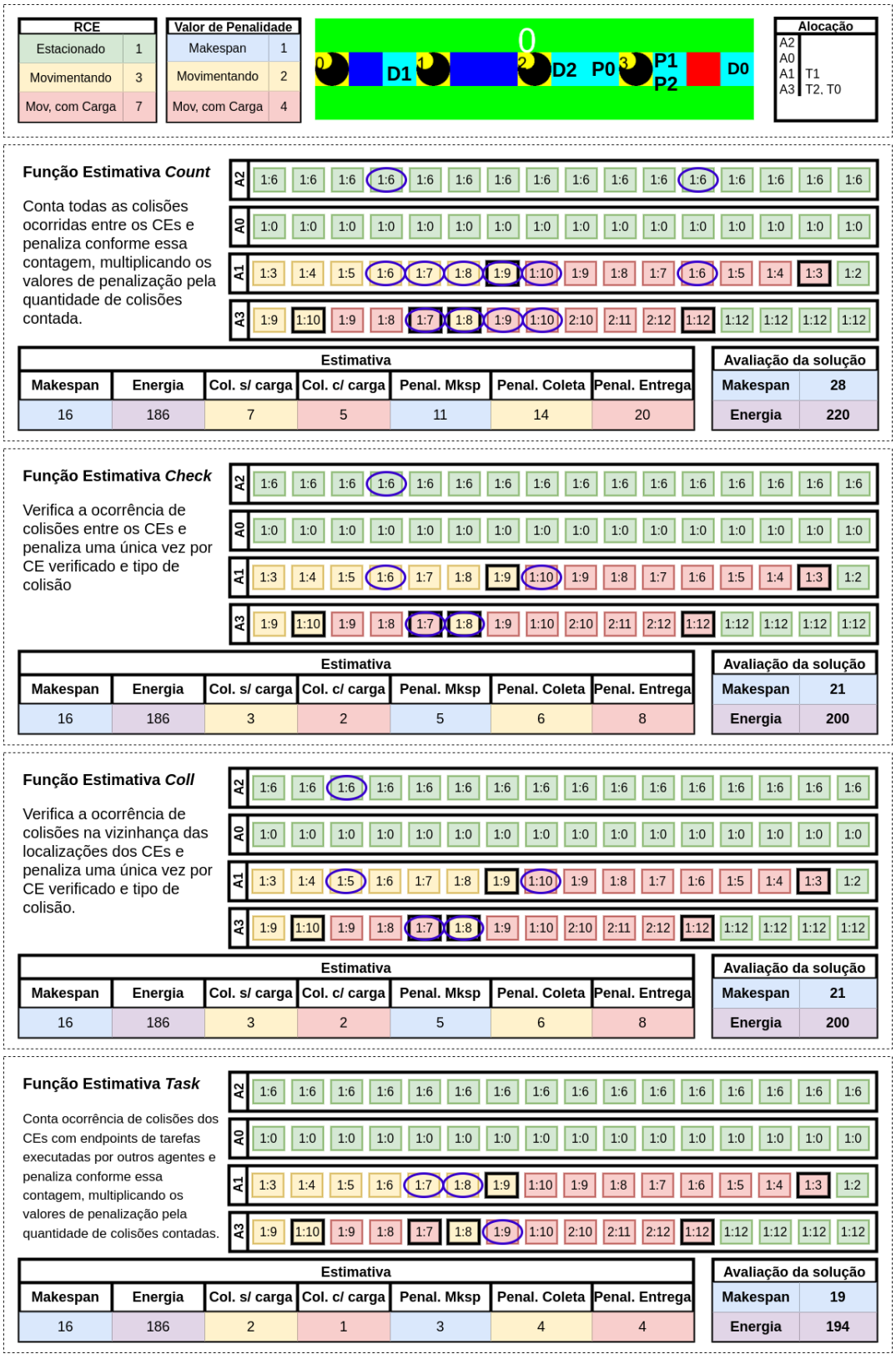

Figura 4.8: Ilustração dos processos de avaliação de uma solução pelas funções aproximadas.

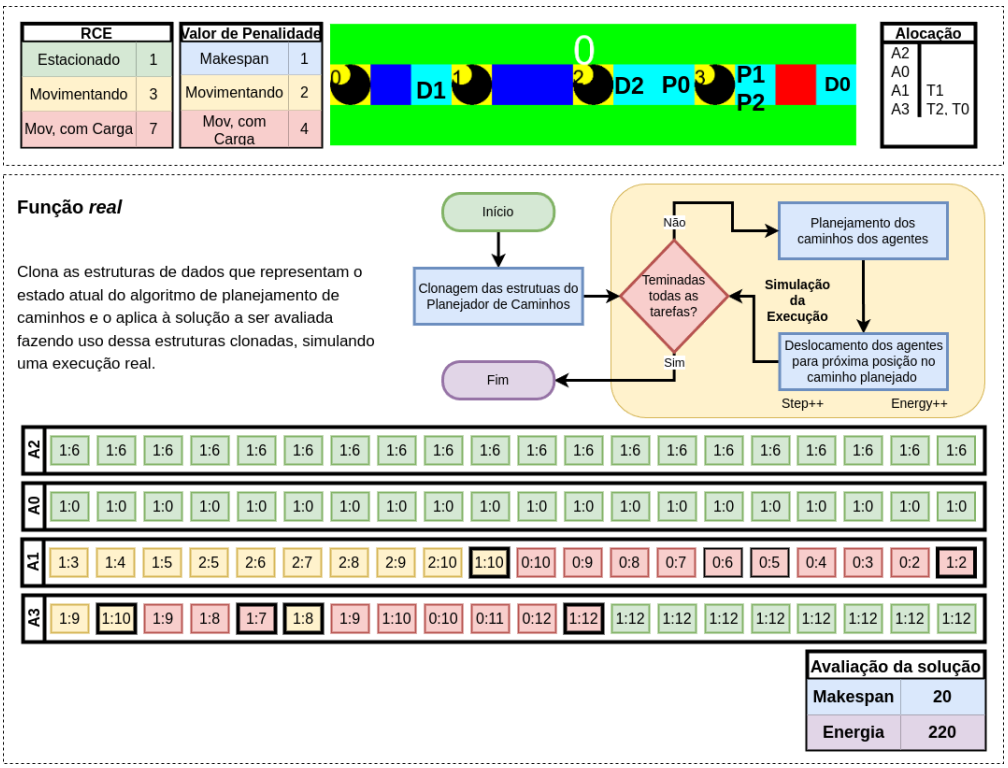

Figura 4.9: Ilustração do fluxo de execução da função objetivo *real* e um exemplo de problema, solução e avaliação da solução.

estejam ocupando *endpoints* de tarefa obstruindo assim o caminho dos agentes com tarefa atribuída, o algoritmo providencia a liberação dessas localizações encaminhando os agentes obstrutores para um outro *endpoint* próximo que não obstrua os caminhos planejados. A figura 4.10 ilustra o fluxograma do algoritmo *GAT* e mostra, marcando com elipse na cor amarelada, a que estágio do fluxo geral do resolvedor o algoritmo corresponde. O *GAT* quando aplicado diretamente com algoritmo construtor da solução gulosa inicial, ou seja, sem o uso do algoritmo genético, recebe o nome de *Greedy Path Planner Token (GPPT)*.

4.4.3 *Genetic Algorithm Token Passing (GATP)*

O *Genetic Algorithm Token Passing (GATP)* é um algoritmo planejador de caminhos que, diferentemente do *GAT*, não permite a substituição de tarefas anteriormente atribuídas por outra, nem anulação de atribuições de tarefas feitas, retirando, assim, a tarefa do conjunto de tarefas pendentes no momento da sua atribuição. Esse comportamento se assemelha ao *TP*. Por outro lado, assim como no *GAT*, o planejamento de caminhos para realizar tarefas é feito em duas etapas. A política de desobstrução de caminhos de tarefa

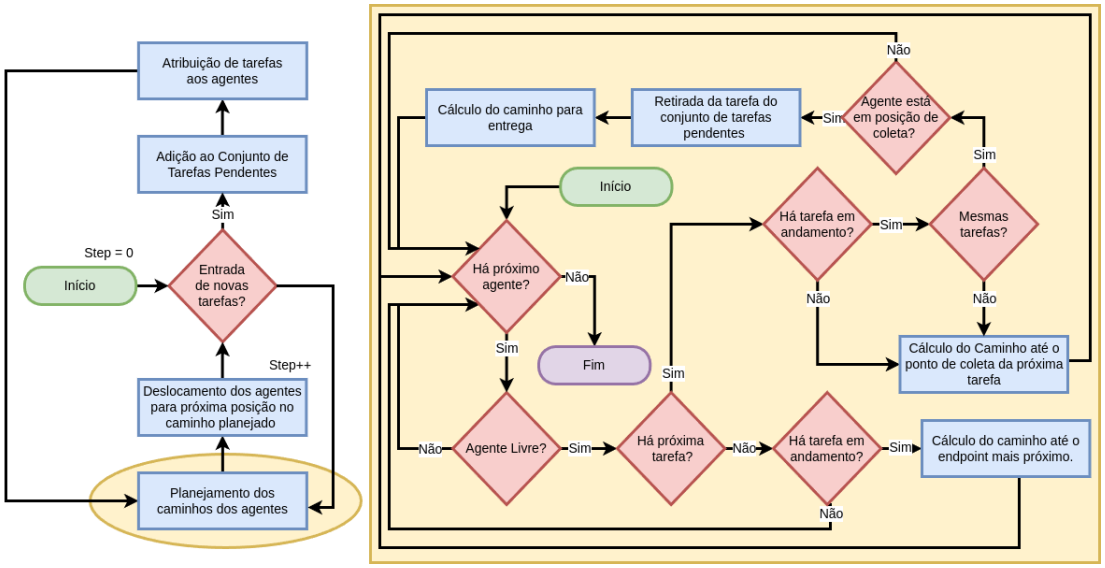

Figura 4.10: Fluxograma Geral do Resolvedor, à esquerda, e fluxograma relativo ao algoritmo *Genetic Algorithm Token (GAT)*, à direita.

também é a mesma que no *GAT*. A figura 4.11 ilustra o fluxograma do algoritmo *GATP* e mostra, marcando com elipse na cor amarelada, a que estágio do fluxo geral do resolvedor o algoritmo corresponde. O *GATP* quando aplicado diretamente com algoritmo construtor da solução gulosa inicial, ou seja, sem o uso do algoritmo genético, recebe o nome de *Greedy Path Planner Token Passing (GPPTP)*.

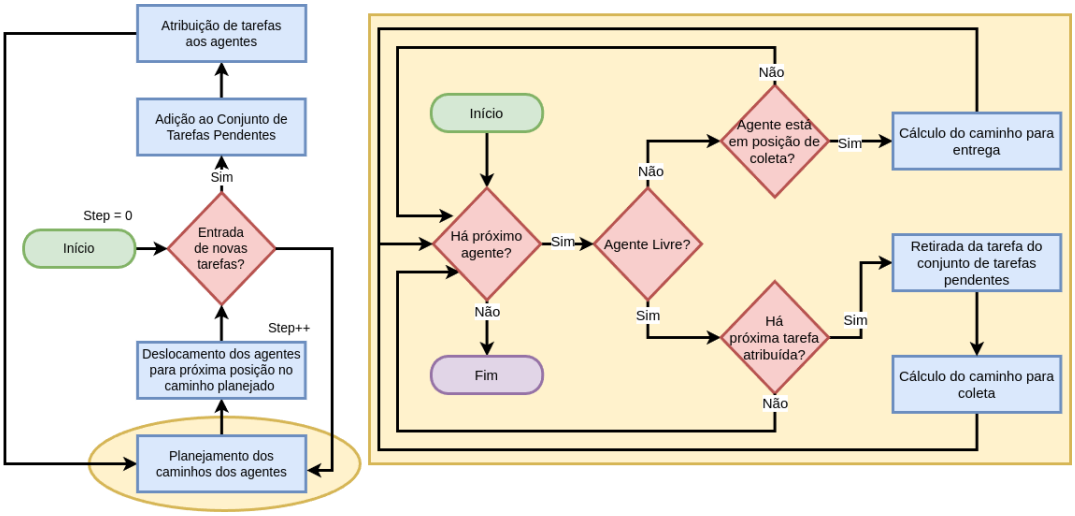

Figura 4.11: Fluxograma Geral do Resolvedor, à esquerda, e fluxograma relativo ao algoritmo *Genetic Algorithm Token Passing (GATP)*, à direita.

5 Experimentos Computacionais

Neste capítulo, são explanados detalhes sobre o desenvolvimento dos experimentos computacionais e resultados obtidos pelos algoritmos propostos. Na Seção 5.1, são descritos detalhes sobre o desenvolvimento do código e recursos utilizados. Na Seção 5.2, são apresentadas as instância utilizadas nos experimentos. Na Seção 5.3, é explicado como foram parametrizados os algoritmos. Na Seção 5.4, são apresentados os resultados dos experimentos.

5.1 Código Fonte e Recursos Utilizados

O desenvolvimento das abordagens apresentadas neste trabalho foi realizado na linguagem C++ e fazendo uso das bibliotecas padrões da linguagem. Fez-se uso, também, da biblioteca *Simple and Fast Multimedia Library (SFML)* apenas para se exibir graficamente o funcionamento dos algoritmos. O código fonte está disponível publicamente [2]. Os experimentos foram realizados em uma máquina Intel© Core™ i7-4790 CPU @ 3.60GHz × 4, com 16 Gb de RAM (4 x DDR3 1600 Mhz) e sistema operacional Linux Mint 21.2 Cinnamon.

5.2 Instâncias de Ambiente e Tarefas

5.2.1 Ambientes

A Figura 5.1 mostra um exemplo de ambiente utilizado neste trabalho. Os experimentos foram realizados em dois ambientes distintos, quais sejam:

1. um ambiente descrito na literatura e considerado pequeno, que possui tamanho de 21×35 e é definido para problemas envolvendo 5 diferentes quantidades de agentes (10, 20, 30, 40 e 50 agentes); e

[2] https://github.com/jrmouro/https—github.com-jrmouro-mapd

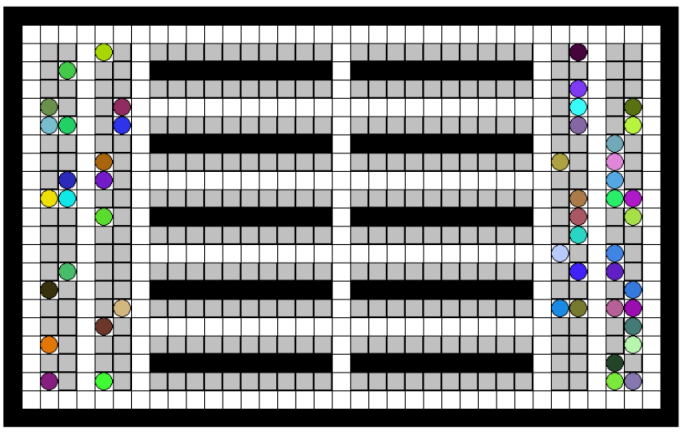

Figura 5.1: Exemplo de Ambiente *MAPD* (Extraído de (MA et al., 2017)).

2. um ambiente ainda menor, que possui tamanho em localizações de 5 x 9 e foi definido, neste trabalho, para problemas envolvendo 4 diferentes quantidades de agentes (2, 3, 4 e 5 agentes).

A definição feita neste trabalho de um ambiente mais reduzido permitiu a rápida execução dos experimentos, de forma que foi possível ter resultados comparáveis de todos os algoritmos.

5.2.2 Tarefas

As tarefas contidas nas instâncias usadas se apresentam segundo uma frequência, significando, por exemplo, a frequência 0,2 uma tarefa a cada 5 intervalos de tempo e a frequência 10, 10 tarefas a cada um intervalo de tempo. Os experimentos foram realizados com duas instâncias de tarefas, sendo uma para ser aplicada no ambiente de 21×35 e outra, no ambiente de 5×9, conforme se segue:

1. 500 tarefas que se apresentam com frequências 0,2, 0,5, 1, 2, 5 e 10; e

2. 28 tarefas que se apresentam com frequências 0,2, 0,5, 1, 2, e 5.

5.3 Parametrização dos algoritmos

Os algoritmos foram parametrizados empiricamente. Para os algoritmos determinísticos foi adotado um RCE segundo os seguintes valores de consumo : 1 unidade para agentes

espera; 2 unidades para agentes se deslocando sem carga; e 4 unidades para agentes se deslocando com carga.

Foram executados experimentos com seis pares de distintos de MCE correspondentes aos deslocamentos para coleta e entrega, quais sejam: (1,0; 1,0), (0,8; 0,8), (0,5; 0,5), (0,3; 0,3), (0,8; 0,3), (0,3; 0,8) e (0,0; 0,0).

Foram executados experimentos parametrizando o algoritmo genético NSGA-II da seguinte forma: 30 gerações para instâncias maiores e 50 gerações para as menores; tamanho mínimo da população de 10 indivíduos; tamanho máximo da população de 20 indivíduos; taxa de ponto de recombinação: 0,5; taxa de seleção para mutação: 0,5; taxa de mutação sobre o vetor de agentes: 0,3; e taxa de mutação sobre o Vetor de tarefas: 0,3.

Para as funções objetivo aproximadas foram definidos parâmetros de penalização para *makespan*, deslocamentos para coleta e para entrega conforme se segue: 1 para *makespan*, 2 para coleta e 3 para entrega.

5.4 Resultados

5.4.1 Abordagens Heurísticas

A figura 5.2 resume o desempenho das heurísticas propostas aplicados sobre o ambiente 21×35 em relação às médias dos objetivos de otimização, número de tarefas executadas e tempo de execução das instâncias. Os valores que representam um equilíbrio entre os objetivos de otimização foram obtidos pela média dos resultados de consumo energético e *makespan* dos experimentos, normalizados dentro de cada grupo de experimento. Um grupo de experimento é representado por uma instância de ambiente e uma de instância de tarefa. Os valores de tempo são valores médios dos tempos de execução dos experimentos em cada grupo. Os valores relativos ao número de tarefas executadas são valores médios das quantidades de tarefas executadas em cada grupo.

O algoritmo *TTP*, parametrizado com MCE 0,8 para deslocamentos sem carga e 0,8 para deslocamentos com carga, obteve melhor desempenho em equilíbrio de objetivos. Pode-se verificar que quando esse algoritmo foi parametrizado para valores inferiores a

Algoritmo	Eq. Objetivos	Tarefas Exec.	Tempo(s)
TTP(0.8:0.8)	0,8649	500,0	0,482
TTP(0.8:0.3)	0,8718	500,0	0,399
TTP(0.3:0.8)	0,8735	500,0	0,346
TTP(0.3:0.3)	0,8794	500,0	0,321
TTP(0.5:0.5)	0,8821	500,0	0,358
TTP(1:1)	0,8845	500,0	1,281
BTT(0.3:0.3)	0,8892	504,0	0,266
BTT(0.8:0.3)	0,8892	504,0	0,263
BTT(0:0)	0,8895	500,0	0,263
TTP(0:0)	0,8895	500,0	0,263
BTT(0.5:0.5)	0,8917	515,3	0,268
BTT(0.3:0.8)	0,9020	583,9	0,290
BTT(0.8:0.8)	0,9020	583,9	0,295
BTT(1:1)	0,9541	904,1	0,366

Figura 5.2: Resultado das Heurísticas (Ambiente 21×35).

esses, seu desempenho foi o pior, o que pode indicar uma coerência de sua heurística e da ideia de que é possível reduzir tempo de execução e consumo de energia optando-se por tarefas que, não só estejam mais próximas dos agentes, mas que tenham caminhos mais próximos do CE. Verifica-se, também, que parâmetros mais exigentes implicam em maior tempo de execução dos algoritmos. Isso porque, quanto mais rigorosas forem as metas de TDC, mais o algoritmo itera sobre o conjunto de tarefas de interesse e mais cálculos de CRs são feitos.

O algoritmo *BTT* apresentou um desempenho pior que o *TTP* em se tratando de equilíbrio de objetivos, no entanto, apresentou melhores tempos de execução. A razão para melhores tempos de execução em relação ao *TTP* deve-se ao fato de que o algoritmo é capaz de iterar menos sobre o conjunto de tarefas de interesse uma vez que usa o fracionamento de tarefas como uma forma auxiliar de adequar os CRs de entrega às metas de TDC impostas. O fracionamento das tarefas não surtiu o efeito esperado, levando a piores soluções à medida que o rigor das TDCs aumentavam, tanto em relação ao equilíbrio de objetivos quanto ao tempo de execução. O aumento do tempo de execução, nesse caso, deve-se ao fato de que o algoritmo inseriu mais tarefas no sistema à medida que o rigor das metas de TDC aumentavam, e mais tarefas implicaram em mais deslocamentos de coleta.

Frenq.	Agentes	BTT(0.5:0.5)	TTP(0.5:0.5)	BTT(0.5:0.5)	TTP(0.5:0.5)
0,2	10	0,9812	0,9824	0,9812	0,9824
	20	0,9437	0,9449	0,9437	0,9449
	30	0,9340	0,9269	0,9340	0,9269
	40	0,9243	0,9261	0,9243	0,9261
	50	0,9245	0,9277	0,9245	0,9277
0,5	10	0,8800	0,8734	0,8800	0,8734
	20	0,9583	0,9845	0,9583	0,9845
	30	0,9223	0,9281	0,9223	0,9281
	40	0,9293	0,9303	0,9293	0,9303
	50	0,8913	0,9061	0,8913	0,9061
1	10	0,9049	0,8914	0,9049	0,8914
	20	0,8883	0,8619	0,8883	0,8619
	30	0,8361	0,8355	0,8361	0,8355
	40	0,9040	0,9219	0,9040	0,9219
	50	0,8821	0,8767	0,8821	0,8767
2	10	0,8718	0,8573	0,8718	0,8573
	20	0,8501	0,8244	0,8501	0,8244
	30	0,8678	0,9232	0,8678	0,9232
	40	0,8903	0,8068	0,8903	0,8068
	50	0,8686	0,8080	0,8686	0,8080
5	10	0,9426	0,9242	0,9426	0,9242
	20	0,9115	0,8802	0,9115	0,8802
	30	0,8623	0,8623	0,8623	0,8623
	40	0,8615	0,8718	0,8615	0,8718
	50	0,7380	0,7206	0,7380	0,7206
10	10	0,9352	0,9303	0,9352	0,9303
	20	0,8501	0,8304	0,8501	0,8304
	30	0,9163	0,8555	0,9163	0,8555
	40	0,8797	0,8431	0,8797	0,8431
	50	0,8009	0,8085	0,8009	0,8085

Figura 5.3: Resultado das Abordagens Heurísticas (algoritmos parametrizados com valores médios de metas de TDC e Ambiente 21 × 35).

A figura 5.3 mostra desempenho das abordagens heurísticas propostas (algoritmos parametrizados com o valor médio de meta de TDC) aplicados sobre o ambiente 21 × 35 em relação aos objetivos de otimização por grupos de experimentos. Os valores representam o equilíbrio entre os objetivos de otimização normalizados dentro de cada grupo de experimento. Os algoritmos, nessa tabela, foram analisados sob dois aspectos relacionados às peculiaridades dos grupos de experimentos. Do lado esquerdo, as cores, que indicam uma gradação (de verde à vermelha) dos resultados, sinalizam que os algoritmos tendem a ter melhor desempenho quando o número de agentes no ambiente é maior. Do lado direito da tabela, as cores indicam que os algoritmos tendem a ter melhor desempenho com frequências de entrada de tarefas maiores. Como tendência geral, os algoritmos apresentam melhor desempenho quando o ambiente está mais congestionado por agentes realizando tarefas. Portanto, sistemas com ambientes vazios e com poucas tarefas nele entrando, tendem a ter CRs de melhor qualidade, tornando a heurística da simples proximidade do agente à tarefa suficiente e ou mais eficiente.

Algoritmo	Eq. Objetivos	Tempo(s)
GPPT	0,8691	1,811
GPPTP	0,8742	2,185
GAT(COLL)	0,8754	178,373
GAT(CHECK)	0,8781	29,953
GATP(COLL)	0,8823	170,757
GATP(CHECK)	0,8845	19,754
GAT(TASK)	0,8846	44,340
GATP(TASK)	0,8859	35,666
GAT(COUNT)	0,8928	41,520
GATP(COUNT)	0,8973	30,077

Figura 5.4: Resultado das Abordagens Meta-Heurísticas (Ambiente 21×35).

5.4.2 Abordagens Meta-Heurísticas

Para as abordagens meta-heurísticas, foram realizadas 28 execuções com diferentes sementes geradoras combinando atribuição de tarefas feitas pelo *NSGA-II* e os planejadores de caminho *GAT* e *GATP*. Tendo em vista o elevado custo computacional da função objetivo *real*, os experimentos a envolvendo foram realizados apenas com instâncias reduzidas construídas para este fim.

A figura 5.4 resume o desempenho das abordagens meta-heurísticas propostas aplicadas sobre o ambiente 21×35 em relação às médias dos objetivos de otimização e tempo de execução das instâncias. Os valores que representam um equilíbrio entre os objetivos de otimização foram obtidos pela média das médias dos resultados de consumo energético e *makespan* dos experimentos, normalizados dentro de cada grupo de experimento. Um grupo de experimento é representado por uma instância de ambiente e uma de instância de tarefa. Os valores de tempo são valores médios dos tempos de execução dos experimentos em cada grupo. Na tabela, encontram-se para fins de comparação de resultados os algoritmos planejadores de caminho *GAT* e *GATP*, ambos fazendo uso do *NSGA-II* com as funções objetivo aproximadas propostas e o uso também da função construtora gulosa, a mesma usada para a formação da população inicial do mencionado algoritmo genético. Nesse último caso, com a aplicação da função gulosa, as abordagens tornam-se puramente heurísticas e são nomeadas de *GPPT* e *GPPTP*.

O algoritmo *GAT* obteve melhor desempenho em termos de equilíbrio de objetivos, enquanto que o algoritmo *GATP* obteve melhor desempenho termos de tempo de execução. O melhor desempenho em termos de equilíbrio de objetivos se deve ao fato de que o algoritmo *GAT* permite a substituição de tarefas atribuídas e, dessa forma, apro-

Frenq.	Agentes	GPPT	GPPTP	GPPT	GPPTP
0,2	10	0,9783	0,9567	0,9783	0,9567
	20	0,9380	0,9286	0,9380	0,9286
	30	0,9236	0,9162	0,9236	0,9162
	40	0,9288	0,9181	0,9288	0,9181
	50	0,9289	0,9303	0,9289	0,9303
0,5	10	0,8975	0,8970	0,8975	0,8970
	20	0,9602	0,9290	0,9602	0,9290
	30	0,9020	0,8963	0,9020	0,8963
	40	0,9035	0,8870	0,9035	0,8870
	50	0,8944	0,8904	0,8944	0,8904
1	10	0,9139	0,9080	0,9139	0,9080
	20	0,8815	0,8509	0,8815	0,8509
	30	0,8245	0,8082	0,8245	0,8082
	40	0,8540	0,8477	0,8540	0,8477
	50	0,8088	0,7970	0,8088	0,7970
2	10	0,8653	0,8933	0,8653	0,8933
	20	0,8350	0,8669	0,8350	0,8669
	30	0,8508	0,8703	0,8508	0,8703
	40	0,8355	0,8130	0,8355	0,8130
	50	0,7708	0,7536	0,7708	0,7536
5	10	0,9343	0,9659	0,9343	0,9659
	20	0,8782	0,8996	0,8782	0,8996
	30	0,8080	0,8349	0,8080	0,8349
	40	0,8373	0,8254	0,8373	0,8254
	50	0,7026	0,6927	0,7026	0,6927
10	10	0,9308	0,9557	0,9308	0,9557
	20	0,8224	0,9369	0,8224	0,9369
	30	0,8970	0,9270	0,8970	0,9270
	40	0,8531	0,8246	0,8531	0,8246
	50	0,7129	0,8035	0,7129	0,8035

Figura 5.5: Resultado dos algoritmos planejadores usados na abordagem meta-heurística (Ambiente 21×35).

veita melhor as mudanças no conjunto de tarefas pendentes, redirecionando os agentes para tarefas mais próximas. O melhor desempenho em termos de tempo de execução do algoritmo *GATP* se deve ao fato de que esse algoritmo não permite a substituição de tarefas atribuídas, e, dessa forma, são realizados menos cálculos de caminhos de coleta e caminhos para enviar agentes para *endpoints* de espera.

Na tabela 5.4, pode-se verificar também que as funções aproximadas cuja heurística busca qualificar os CEs estimados (*coll* e *check*), obtiveram melhor desempenho do que as funções aproximadas cuja heurística busca quantificar os CEs (*task* e *count*). Ao comparar o desempenho das funções aproximadas com relação à função construtora gulosa, verifica-se que, em média, todas elas não foram capazes de permitir ao algoritmo genético convergir para soluções melhores, além de consumirem bem mais recursos computacionais.

A tabela 5.5 mostra desempenho dos planejadores de caminho *GAT* e *GATP* combinados à função construtora gulosa aplicados sobre o ambiente 21×35 em relação aos objetivos de otimização por grupos de experimentos. Os valores representam o equilíbrio entre os objetivos de otimização normalizados dentro de cada grupo de experimento. Os algoritmos, nessa tabela, foram analisados sob dois aspectos relacionados às peculiaridades

dos grupos de experimentos. Do lado esquerdo, as cores, que indicam uma gradação (de verde à vermelha) dos resultados, sinalizam que os algoritmos planejadores de caminho tendem a ter melhor desempenho quando o número de agentes no ambiente é maior. Do lado direito da tabela, as cores indicam que os algoritmos tendem a ter melhor desempenho com frequências de entrada de tarefas maiores. Como tendência geral, os algoritmos apresentam melhor desempenho quando o ambiente está mais congestionado por agentes realizando tarefas.

5.4.3 Abordagens Heurísticas e Meta-Heurísticas

A figura 5.6 resume o desempenho das abordagens heurísticas e meta-heurísticas propostas aplicadas sobre o ambiente 5×9 em relação às médias dos objetivos de otimização e tempo de execução das instâncias. Os valores que representam um equilíbrio entre os objetivos de otimização foram obtidos pela média das médias dos resultados de consumo energético e *makespan* dos experimentos, normalizados dentro de cada grupo de experimento. Um grupo de experimento é representado por uma instância de ambiente e uma de instância de tarefa. Os valores de tempo são valores médios dos tempos de execução dos experimentos em cada grupo. Na tabela, encontram-se para fins de comparação de resultados, os algoritmos correspondentes às abordagens heurísticas e meta-heurísticas, inclusive, utilizando-se, nas meta-heurísticas, a função objetivo *real*.

Verifica-se que as abordagens meta-heurísticas obtiveram melhor desempenho em termos de equilíbrio de objetivos que as demais quando fizeram uso da função objetivo *real*. Em contrapartida, os tempos de execução dessas abordagens foram relativamente muito maiores.

A tabela 5.7 traz uma análise comparativa dos resultados das abordagens propostas neste trabalho com os resultados de algoritmos presentes na literatura. Apesar de não ser um objetivo principal, haja vista que o problema estudado não se preocupa somente com o recurso de tempo para a execução das tarefas, é interessante saber o quanto os algoritmos propostos estão afastados em termos de *makespan* do desempenho desses outros algoritmos considerados como estado da arte.

Algoritmo	Eq. Objetivos	Tempo(s)
GATP(REAL)	0,8074	3,285
GAT(REAL)	0,8174	1,946
BTT(0.3:0.8)	0,8343	0,001
BTT(0.8:0.8)	0,8343	0,001
TTP(0.8:0.8)	0,8355	0,001
BTT(1:1)	0,8386	0,001
TTP(0.8:0.3)	0,8399	0,001
TTP(1:1)	0,8406	0,001
TTP(0.5:0.5)	0,8414	0,001
GAT(CHECK)	0,8415	0,145
GAT(COLL)	0,8418	0,150
GAT(TASK)	0,8432	0,159
TTP(0:0)	0,8434	0,001
BTT(0:0)	0,8434	0,001
TTP(0.3:0.8)	0,8434	0,001
BTT(0.8:0.3)	0,8435	0,001
BTT(0.3:0.3)	0,8435	0,001
BTT(0.5:0.5)	0,8441	0,001
TTP(0.3:0.3)	0,8456	0,001
GAT(COUNT)	0,8522	0,144
GATP(CHECK)	0,8679	0,101
GATP(COLL)	0,8704	0,122
GATP(TASK)	0,8744	0,125
GATP(COUNT)	0,8756	0,105

Figura 5.6: Resultado das Abordagens Heurísticas e Meta-Heurísticas (Ambiente 5×9).

5.4.4 Algoritmos da Literatura

Uma análise comparativa dos resultados alcançados neste trabalho com os resultados de algoritmos presentes na literatura não é um objetivo principal, haja vista que o problema estudado não se preocupa somente com o recurso de tempo para a execução das tarefas. Porém, é interessante saber o quanto os algoritmos propostos estão afastados em termos de *makespan* do desempenho desses outros algoritmos considerados como estado da arte. Na tabela 5.7, se encontram lançados os resultados (somatórios de resultados de todos os grupos de experimentos) das abordagens heurísticas propostas, bem como os do *TP* e do *TPTS*, que serviram de referência para este trabalho. Os Algoritmos determinísticos *TTP* e *BTT* atingiram, em algumas parametrizações, resultados melhores que o *TP* em ambos os objetivos buscados, ficando abaixo do *TPTS* em termos de *makespan*. A versão implementada do *tp* obteve melhor desempenho em tempo de execução.

Algoritmo	Makespan	Energia	Tempo(s)
TPTS	30792		
TTP(0.8:0.8)	30865	2887074	14,451
TTP(0.8:0.3)	31017	2902276	11,976
TTP(0.3:0.8)	31129	2902804	10,367
TTP(0.3:0.3)	31269	2914686	9,637
TTP(1:1)	31332	2917670	38,443
TTP(0.5:0.5)	31368	2915464	10,726
BTT(0:0)	31537	2927850	7,895
TTP(0:0)	31537	2927850	7,898
BTT(0.3:0.3)	31560	2926444	7,965
BTT(0.8:0.3)	31560	2926444	7,885
TP	31581	2927850	7,838
BTT(0.5:0.5)	31661	2929178	8,054
BTT(0.3:0.8)	31916	2950360	8,711
BTT(0.8:0.8)	31916	2950360	8,851
BTT(1:1)	33401	3047308	10,989

Figura 5.7: Resultado das Abordagens Heurísticas e Algoritmos da Literatura TP e $TPTS$(Ambiente 5×9).

6 Conclusões e Trabalhos Futuros

Nos dias hoje, muito se discute sobre questão energética envolvendo escassez de recursos naturais, impacto ambiental decorrente da obtenção de fontes energéticas e os custos cada vez mais altos para os sistemas produtivos. Nesse contexto, surge o desafio de projetar sistemas que minimizem custos com recursos energéticos e, dessa forma, impactem menos o meio ambiente e revertam mais benefícios para a sociedade como um todo. Os responsáveis por sistemas autônomos de armazéns, instalações industriais, portos, aeroportos e outro ambientes que utilizam frotas de robôs devem incluir em seus objetivos reduzir ao máximo o consumo de energia decorrente da operação com essas máquinas. Uma forma inteligente de fazê-lo, aproveitando os meios já disponíveis, é aprimorar os algoritmos de otimização dos sistemas que as controlam.

Este trabalho, alicerçado sobre o tema do problema *MAPD*, buscou construir modelos que incluem, além do tempo de execução das tarefas, o consumo de energia como mais objetivo de otimização. Nele, foi definido, para fins de medição do consumo energético, um Regime de Consumo Energético (RCE) simplificado. O regime permitiu desvincular o tempo de execução das tarefas do consumo energético dos agentes, aumentando a complexidade do problema. O objetivo imposto às abordagens apresentadas foi de resolver o problema de forma que a solução se equilibre entre a agilidade para cumprir as tarefas e o menor custo energético.

Foram apresentados quatro resolvedores sequenciais para o problema, sendo duas abordagens heurísticas e duas meta-heurísticas. Esses algoritmos foram parametrizados de forma empírica, pelo que os resultados não podem ser considerados limítrofes. O *BTT*, uma das heurísticas, muda a definição do problema *MAPD* ao incluir a possibilidade de fracionamento da etapa de transporte da tarefa, o que não deve ser um problema para muitos sistemas reais. No entanto, esse fracionamento das tarefas, nas instâncias utilizadas nos experimentos, mostrou ser uma ação prejudicial na busca de melhores aos resultados. À medida que mais fracionamentos foram realizados, piores soluções advieram.

Dentre as abordagens heurísticas, o algoritmo *TTP* mostrou melhor desempenho

geral que o *BTT* e, dentre as abordagens meta-heurísticas, o panejador de caminhos *GAT* superou o *GATP*. Os experimentos mostraram que todas as abordagens propostas obtiveram melhor desempenho em ambientes mais congestionados por agentes, evidenciando o viés das heurísticss para lidarem com situações desse tipo.

Às abordagens meta-heurísticas, foi incluído o *NSGA-II*, um algoritmo genético multi-objetivo com o intuito de buscar soluções de escalonamento das tarefas. Foram testadas cinco funções objetivo para avaliação das soluções provenientes do algoritmo genético, sendo quatro pensadas para serem de custo computacional baixo e uma para ser mais precisa nesta avaliação.

As funções objetivo aproximadas apresentaram baixa capacidade de estimar o *makespan* e o consumo energético das soluções, impedindo, assim, o algoritmo genético convergir para melhores soluções durante sua busca. A função objetivo *real*, por outro lado, possibilitou a convergência para melhores soluções, mas como foi desenvolvida, apresentou um elevado custo computacional. Ao simular a execução do planejamento de caminhos para avaliar cada solução gerada pelo algoritmo genético e, assim, obter valores reais de consumo energético e *makespan*, a função realiza cópias significativas de estruturas de dados na memória. O tempo para se realizar essas cópias representou quase que a totalidade do tempo total de processamento da função.

Como trabalhos futuros, pretende-se aprimorar as heurísticas, dotando-os da capacidade realizar de trocas de tarefas entre os agentes como se faz no *TPTS*, assim como torná-los reativos às variações de fluxo de entrada de novas tarefas no sistema ou mudanças de características do ambiente.

Outra linha de aprimoramento a ser seguida é a paralelização das meta-heurísticas, que pode incluir o paralelismo das funções-objetivo ou do própria meta-heurística como um todo. As funções aproximadas devem, também, ser objeto de pesquisa no sentido de torná-las mais eficazes, por exemplo, conferindo à ordem dos agentes no vetor de agentes alguma importância nas estimativas do CE e penalizações, e, também, estudar seu comportamento em relação aos seus parâmetros de penalização. Uma estratégia, por exemplo, seria usar a função *real* para avaliar as funções aproximadas a cada geração do algoritmo genético, e, assim, usar essa informação para, dinamicamente, parametrizá-las.

Bibliografia

BOYARSKI, E. et al. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. [S.l.: s.n.], 2015. p. 740–746.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm:nsga-ii. In: . [S.l.]: IEEE Transactions on Evolucionay Computation, 2002. v. 6, p. 182–197.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. In: *IEEE Transactions on Systems Science and Cybernetics 4*. [S.l.: s.n.], 1968. p. 100–107.

HOLTE, R. C. et al. Hierarchical a*: Searching abstraction hierarchies efficiently. In: *AAAI*. [S.l.: s.n.], 1996. v. 1, p. 530–535.

JVD, B.; W, Z. Models for warehouse management: Classiőcation and examples. In: *International Journal of Production Economics*. [S.l.]: Science Direct, 1999. v. 59, p. 519–528.

LIU, M. et al. Task and path planning for multi-agent pickup and delivery. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*. [S.l.]: AAMAS, 2019.

MA, H. et al. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In: . [S.l.]: AAAI-19, 2019.

MA, H. et al. Lifelong multi-agent path finding for online pickup and delivery tasks. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. [S.l.]: AAMAS, 2017. p. 837–845.

MORRIS, R. et al. Planning, scheduling and monitoring for airport surface operations. In: *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, 13 February 2016*. [S.l.]: AAAI, 2016. WS-16-12, p. 608–614.

OLIVEIRA, G. S. et al. Efficient multi-constrained task allocation in smart warehouses. In: . [S.l.: s.n.], 2022.

PHILLIPS, M.; LIKHACHEV, M. Sipp: Safe interval path planning for dynamic environments. In: . [S.l.]: ICRA, 2011. p. 5628–5635.

QUEIROZ, A. C. L. C. et al. Solving multi-agent pickup and delivery problems using a genetic algorithm. In: *Intelligent Systems 9th Brazilian Conference, BRACIS 2020 Rio Grande, Brazil, October 20–23, 2020 Proceedings, Part II*. [S.l.]: Springer, 2020. p. 140–153.

SALZMAN, O.; STERN, R. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*. [S.l.]: AAMAS, 2020. p. 1711–1715.

SHARON, G. et al. Conflict-based search for optimal multi-agent pathfinding. In: *Artificial Intelligence*. [S.l.]: ELSEVIER, 2015. v. 219, p. 40–66.

SILVER, D. Cooperative pathfinding. In: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment*. [S.l.]: AIIDE, 2005. p. 117–122.

STANDLEY, T. S. Finding optimal solutions to cooperative pathfinding problems. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. [S.l.]: AAAI, 2010.

STERN, R. et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Proceedings of the Twelfth International Symposium on Combinatorial Search (SoCS 2019)*. [S.l.: s.n.], 2019. p. 151–159.

SVANCARA, J. et al. Online multi-agent pathfinding. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. [S.l.]: AAAI, 2019. p. 7732–7739.

VELOSO, M. et al. Cobots: Robust symbiotic autonomous mobile service robots. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. [S.l.]: IJCAI, 2015. p. 4423.