5
Estudo de caso: Sistema de transporte automatizado de cargas

Este capítulo apresenta o estudo de caso realizado para validar a aplicabilidade do método de verificação proposto. Ao contrário do exemplo prático do capítulo anterior, cujo objetivo era facilitar a compreensão do método por meio de exemplos práticos simplificados, este estudo de caso aborda um problema um pouco mais complexo – um sistema de transporte automatizado de cargas – e tem a principal finalidade de prover resultados que:

1. Comprovem a viabilidade de se projetar uma solução multiagente para um domínio de problema real através da estrutura proposta para o subsistema central;

2. Forneçam informações quantitativas que indiquem a presença de auto–organização no sistema proposto;

3. Demonstrem a eficácia do método para a verificação experimental de sistemas multiagentes, no que diz respeito à capacidade de validar o sistema e, ao final, fornecer uma sequência finita de ações que leve a simulação à um estado desejado.

5.1
Definição

Um sistema de transporte automatizado de cargas é composto por veículos automatizados (AGVs[1]), que têm como objetivo o transporte automatizado de cargas dentro de fábricas, depósitos ou ambientes similares.

Neste problema, três entidades principais estão envolvidas no processo de transporte de cargas:

– **Estação de origem (*Pickup station:*)** Uma estação de origem representa o ponto inicial de uma tarefa de transporte. Como resultado de uma ação externa, estações de origem recebem novas cargas a serem

[1] *Automated Guided Vehicles* , ou "veículos guiados de forma automatizada", em tradução livre

transportadas. Então, geram requisições de transporte que são enviadas à um ou mais veículos. Para fins de simplificação, todas as cargas possuem a mesma prioridade. Na prática, isso significa que um veículo que esteja disponível sempre transportará a carga mais próxima de sua localização atual;

- **Estação de destino (*Drop station*):** Uma estação de destino representa o ponto final de uma tarefa de transporte. Estações de destino estão constantemente aguardando que novas cargas transportadas pelos veículos sejam entregues. Para fins de simplificação, todas as estações de destino são iguais, isto é, todas as cargas podem ser entregues em qualquer uma das estações de destino disponíveis;
- **Veículo (*Vehicle*):** Um veículo automatizado (AGV) é capaz de, ao receber uma requisição de transporte, se locomover para a estação de origem, receber a carga a ser transportada, se dirigir à uma estação de destino e, então, entregar a carga.

Enfim, uma solução viável para este cenário deve contemplar as tarefas de expedição e roteamento, tratando, evitando ou contornando as características e dificuldades enumeradas acima. Portanto, o sistema deve ser **flexível** para conseguir tratar alterações normais ocorridas no ambiente e também **robusto** para ser capaz de contornar possíveis falhas mais graves.

Como num ambiente real, todas essas entidades estão sujeitas à falhas. Por exemplo: estações de origem ou destino podem ser interrompidas por falhas mecânicas nas esteiras, enquanto veículos de transporte podem ficar imobilizados caso suas fontes de energia se esgotem.

Esses fatores, combinados às características altamente dinâmicas e mutáveis de uma fábrica, obrigam que os veículos sejam capazes de detectar a presença de outros veículos (muitas vezes em estado de falha) ou obstáculos a fim de evitarem possíveis colisões e/ou congestionamentos.

Portanto, uma modelagem viável desse sistema deve ser capaz de completar, de forma autonômica, duas tarefas principais:

- **Expedição de cargas:** Toda nova carga recebida por uma estação de origem deve gerar uma requisição de transporte que será enviada ao ambiente de tal forma que um ou mais veículos a recebam. Uma carga não é atribuída à um veículo de forma definitiva até que ele de fato a receba, dando início ao seu transporte. Essa característica implica na escolha do veículo mais apto para cada tarefa de transporte. Então, após a atribuição definitiva de uma carga à um veículo, caso essa mesma estação não possua mais nenhuma carga a ser transportada, uma mensagem de cancelamento

de requisição de transporte deve ser enviada para que todos os outros outros veículos que se dirigiam à essa estação abortem o processo.

– **Roteamento:** Os veículos devem ser roteados pelo ambiente para que consigam atingir as estações de origem (para receber cargas) e as estações de destino (para entregar cargas). Esse processo deve ser adaptativo, uma vez que a configuração da fábrica pode ser constantemente alterada por obstáculos, falhas em entidades ou interrupções em zonas específicas.

Em (Weyns, 2006a, Weyns, 2006b), o autor propõe a utilização de campos de gradientes para completar ambas as tarefas. Na física, os campos vetoriais são comumente utilizados para descrever o comprimento e a direção de uma força (ou de uma combinação de várias forças) em um espaço. A associação desses vetores à cada ponto do espaço possibilita que, a partir de qualquer posição do ambiente, seja possível identificar a direção do ponto de origem dessas forças.

Nesse sentido, a emissão de campos de gradiente pelas entidades de um sistema de transporte automatizado pode, de fato, ser utilizada para a cumprir as tarefas de expedição de cargas e roteamento adaptativo. Entretanto, será necessário estabelecer três tipos de gradiente diferentes, cada qual emitido por uma entidade diferente e com um propósito distinto:

1. **Gradiente de origem (*Pickup Gradient*):** Os gradientes de origem são campos de gradiente gerados pelas estações de origem para notificar os veículos em um determinado raio de que existem uma ou mais cargas disponíveis para transporte. Como os campos de gradiente possuem uma característica de dispersão, quanto mais distante da estação estiver o veículo, mais fraco será o gradiente recebido por ele. Então, cada veículo deve calcular o gradiente resultante em sua posição e decidir para qual direção irá se mover. Portanto, um veículo acabará por se locomover para a estação de origem mais próxima de sua localização atual;

2. **Gradiente de destino (*Drop Gradient*):** Os gradientes de destino possuem a mesma essência dos gradientes de origem, mas são emitidos pelas estações de destino de forma constante e ininterrupta. Neste caso, a sua função é rotear um veículo até a estação de destino mais próxima;

3. **Gradiente de veículo (*Vehicle Gradient*):** Os gradientes de veículo são um tipo específico de gradiente repulsivo emitido por todos os veículos, em todas as situações. Eles se diferem dos outros gradientes em um aspecto importante: ao invés de atrair um veículo para a origem da força (no caso, um outro veículo), eles fazem o inverso, pois possuem

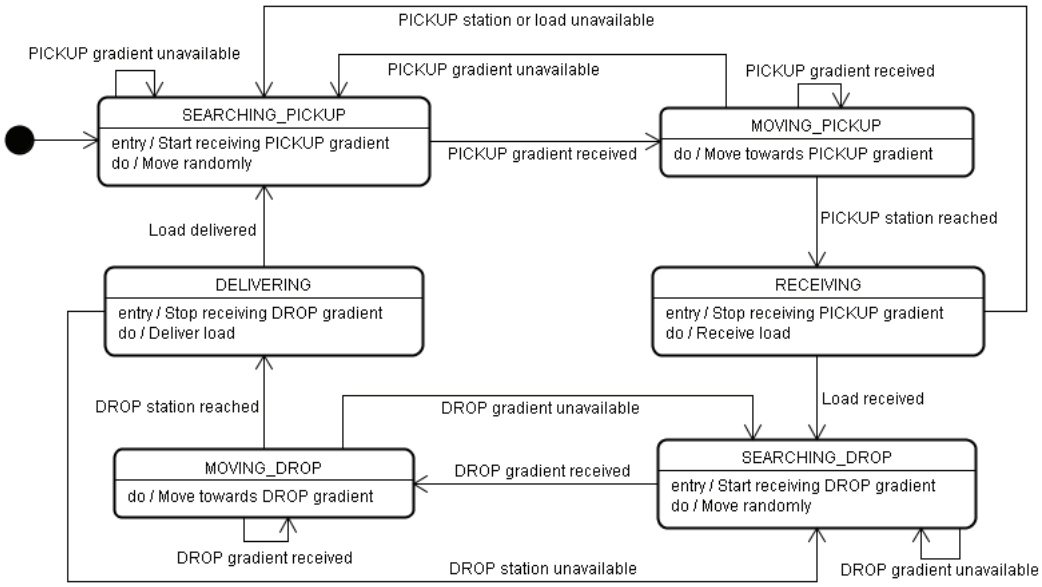

Figura 5.1: Diagrama de estados: Veículo de transporte automatizado

magnitude negativa. Ou seja, têm a finalidade de evitar que um veículo se locomova em direção à outro, prevenindo colisões e congestionamentos.

A figura 5.1 descreve os estados de um veículo automatizado que utiliza os campos de gradientes enumerados acima para orientar os veículos no ambiente:

- ***SEARCHING_PICKUP*:** Inicialmente, o veículo está aguardando uma requisição de transporte e se locomovendo de forma aleatória. Ele está constantemente monitorando o seu entorno em busca de campos de gradiente do tipo *PICKUP* que indiquem a existência de cargas a serem transportada. Assim que um gradiente deste tipo é percebido, o veículo executa o seu próximo estado;
- ***MOVING_PICKUP*:** Quando o veículo está dentro do alcance de um ou mais campos de gradiente do tipo *PICKUP*, ele interrompe a sua movimentação aleatória e, através do cálculo da direção do gradiente *PICKUP* resultante[2] na sua localização, o veículo decide para onde se locomover. Se, em um dado momento, nenhum gradiente *PICKUP* for capturado, o veículo retorna ao estado *SEARCHING_PICKUP*. Senão, assim que o veículo atingir uma estação de origem de transporte ativa (com uma ou mais cargas disponíveis), ele passa a executar o estado seguinte;
- ***RECEIVING*:** Aqui o veículo já se encontra em uma estação de origem ativa, pronto para receber a carga a ser transportada. Entretanto, se

[2]O cálculo do gradiente resultante em um ponto é realizado através da soma simples de todas as forças que afetam este mesmo ponto.

por algum motivo essa carga não estiver disponível, o veículo considera que a requisição de transporte foi cancelada e retorna ao estado *SEARCHING_PICKUP*. Senão, ele recebe a carga e continua a sua execução normal;

- ***SEARCHING_DROP*:** Neste estado, o veículo está carregado, se locomovendo aleatoriamente e monitorando o ambiente em busca dos gradientes do tipo *DROP* emitidos pelas estações de destino. Assim que um gradiente desse tipo é percebido, o próximo estado é executado;
- ***MOVING_DROP*:** Estando dentro do alcance de um ou mais campos de gradiente do tipo *DROP* (que indicam a presença de estações de destino nas proximidades), o veículo interrompe a sua movimentação aleatória e começa a se locomover na direção do gradiente *DROP* resultante. Se, por algum acaso, o veículo sair do campo de abrangência dos gradientes *DROP*, ele retorna ao seu estado anterior. Senão, o veículo continua nesse estado até que uma estação de destino seja encontrada, quando o próximo estágio é iniciado;
- ***DELIVERING*:** Finalmente, o veículo carregado se encontra pronto para entregar a carga em uma estação de destino. Existe, ainda, a possibilidade da estação encontrada estar indisponível. Neste caso, o veículo deve procurar uma nova estação, retornando ao estado *SEARCHING_DROP*. Se a estação de destino estiver ativa, o veículo entrega a carga e reinicia o seu ciclo de execução, buscando novas cargas a serem transportadas.

5.2 Modelagem

O primeiro passo para a aplicação do método de verificação proposto neste trabalho foi a modelagem da solução apresentada para o transporte automatizado de cargas através da definição e implementação dos agentes, entidades e eventos que atuam sobre o ambiente.

A figura 5.2 detalha essa modelagem por meio de um diagrama de classes que instancia o subsistema central do *framework* de verificação apresentado na seção 4.2.1. Esse diagrama foi posteriormente implementado, para tornar possível a validação da modelagem proposta através da instanciação do subsistema de verificação (seção 4.2.2).

1. **Agente sujeito à falhas (classe abstrata *FailableAgent*):** Como foi dito na seção anterior, todos os agentes de um sistema de transporte automatizado estão sujeitos à falhas. Por esse motivo, optou–se por

criar uma classe base denominada *FailableAgent*, derivada de *Agent* (subsistema central), que implementa essa característica. Na verdade, a diferença desse novo agente é que, à cada passo da simulação, ele analisa uma chance probabilística de falha (definida na sua criação). Caso uma falha seja gerada, ele interrompe seu funcionamento normal e passa a analisar, à cada novo passo, uma chance probabilística de recuperação (também definida na sua criação). Assim que a recuperação é realizada, ele volta a executar normalmente. Esse comportamento simula a probabilidade real de um agente falhar em virtude do mau funcionamento de um de seus componentes;

2. **Estação de carga (classe abstrata *LoadStation*):** Tanto as estações de origem, quanto as de destino, possuem o mesmo funcionamento básico: uma esteira mecânica que entrega e recebe cargas. Por esse motivo, foi criada uma classe base para estações de carga, chamada de *LoadStation*, derivada de *FailableAgent*. Na prática, essa classe apenas facilita o processo de enfileiramento e desenfileiramento de cargas na esteira da estação;

3. **Estação de origem (classe *PickupStation*):** Uma estação de origem (classe *PickupStation*) é uma estação na qual o enfileiramento de cargas constitui uma ação externa – executada por um humano, que adiciona novas cargas à serem transportadas pelo sistema – e o desenfileiramento de cargas uma ação interna – executada por um veículo que retira uma carga para transportá-la. Além disso, uma estação de origem ativa (que não se encontra em estado de falha e que possui cargas a serem transportadas) está constantemente emitindo gradientes do tipo *PICKUP*, para cumprir a tarefa de expedição de cargas, conforme explicado na seção anterior;

4. **Estação de destino (classe *DropStation*):** Uma estação de destino (classe *DropStation*) é uma estação na qual o enfileiramento de cargas é uma ação interna – executada por um veículo que entrega um carga – e o desenfileiramento de cargas uma ação externa – executada por um humano que retira as cargas da estação, levando–as ao seu destino final (um caminhão, uma prateleira, etc.). Além disso, uma estação de destino ativa (que não se encontra em estado de falha) está constantemente emitindo gradientes do tipo *DROP*, para direcionar os veículos em transporte até a estação;

5. **Veículo (classe *Vehicle*):** Os veículos de transporte automatizado foram modelados através da classe *Vehicle*, que estende *FailableAgent*. Essa classe foi implementada seguindo o diagrama de estados detalhado na seção anterior. Portanto, a movimentação de um veículo no ambiente (depósito) é coordenada pelos gradientes que ele recebe, de acordo com o estado em que se encontra (vazio ou carregado). Além disso, um veículo também emite um gradiente repulsivo que tem o propósito de evitar a sua colisão com outros veículos;

6. **Depósito (classes *Warehouse* e *WarehouseLocation*):** A classe *Warehouse* representa o ambiente (depósito ou fábrica) onde esse sistema de transporte automatizado é implementado. Por isso, ela herda a classe *Environment* (subsistema central) – que define um ambiente no contexto de sistemas multiagentes – e adiciona algumas características particulares, tal como a divisão do espaço em um *grid* discreto, com $largura$ e $altura$ pré-definidas. A classe *WarehouseLocation*, por sua vez, implementa a interface *Location* (subsistema central) e indica uma localização dentro desse *grid*, dada pelas coordenadas (x, y), onde $0 \leq x < largura$ e $0 \leq y < altura$. Ela é, ainda, capaz de calcular a distância entre duas localizações, através da fórmula $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, onde (x_1, y_1) e (x_2, y_2) são as coordenadas das localizações. Uma localização (x, y) também conhece todos os seus vizinhos, isto é, as localizações com as quais ela faz fronteira: $(x-1, y-1)$, $(x, y-1)$, $(x+1, y-1)$, $(x-1, y)$, $(x+1, y)$, $(x-1, y+1)$, $(x, y+1)$ e $(x+1, y+1)$.

7. **Gradientes (classe *Gradient* e enumeração *Gradient.Type*):** Os gradientes (classe *Gradient*) emitidos pelas estações e pelos veículos foram implementados como eventos (derivados da classe *Event* do subsistema central) que são propagados no ambiente em um raio pré-determinado, à partir da localização do agente emissor. Um gradiente possui um tipo (*PICKUP*, *DROP* ou *VEHICLE*, definidos na enumeração *Gradient.Type*), uma magnitude (um número real com sinal), representando a força desse gradiente, e um ângulo (em graus) que aponta para a localização do agente emissor no ambiente. Um veículo calcula o gradiente resultante em sua localização através da soma vetorial de todos os gradientes percebidos em sua localização. Por exemplo: um veículo no estado *MOVING_PICKUP* irá se locomover de acordo com o gradiente resultante de todos os gradientes do tipo *PICKUP* (para encontrar uma estação de origem) e *VEHICLE* (para evitar colisão com outros veículos próximos) recebidos.

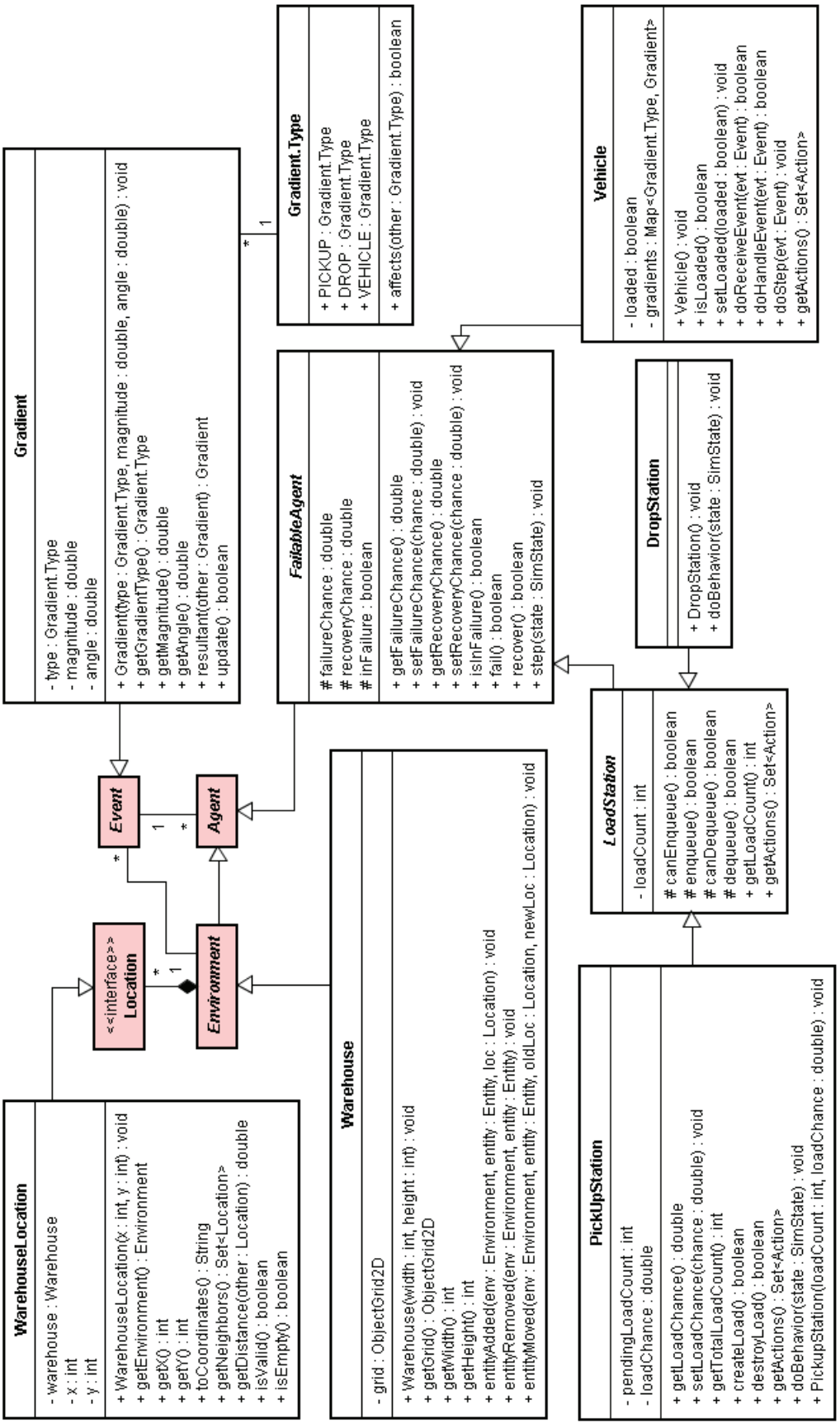

Figura 5.2: Diagrama de classes: Sistema de transporte automatizado de cargas

5.3 Verificação

Para garantir que a modelagem proposta na seção anterior é válida para o domínio de problema analisado – sistema de transporte automatizado de cargas –, aplicou–se o método de verificação experimental proposto neste trabalho. O processo de verificação, quando completado, garante que o sistema é viável, ou seja, é capaz de cumprir os objetivos definidos. Além disso, após a verificação é possível ter a certeza de que o sistema não apresentará um comportamento emergente indesejável, o que o levaria à um estado caótico (de total desorganização).

A aplicação do método de verificação ocorre em quatro etapas, que são realizadas através da instanciação do subsistema de verificação apresentado na seção 4.2.2, conforme exemplificado na seção 4.3:

1. **Definir ações internas e externas (e suas reversas):**

Considerando as entidades modeladas na seção anterior, as seguintes ações podem ser identificadas, para cada agente:

- **Agente sujeito à falhas (*FailableAgent*):** Todo e qualquer agente sujeito à falhas possui duas ações externas básicas que podem ser executadas de acordo com o seu estado (normal ou falha):

  (a) **Falha (classe *Fail*):** Quando o agente estiver em seu estado normal, isto é, em perfeito funcionamento, fatores externos podem causar uma falha em seus componentes mecânicos. No sistema modelado, isso é simulado através da execução de uma ação externa que, de acordo com uma probabilidade matemática pré-definida, faz com que um veículo ou estação passe ao estado de falha;

  (b) **Recuperação (classe *Recover*):** Da mesma forma, fatores externos ao sistema podem recuperar estações e veículos avariados (ex: um reparo mecânico). Portanto, a modelagem proposta também simula esse comportamento através de uma ação externa probabilística que faz com que agentes em estado de falha voltem ao estado normal.

- **Estação de origem (*PickupStation*):** Além das ações inerentes aos agentes sujeitos à falhas, as estações de origem podem sofrer a execução de uma ação externa adicional:

  (c) **Criar carga (classe *CreateLoad*):** No cenário real do problema de transporte automatizado de cargas, um ator externo ao sistema é

responsável por adicionar novas cargas nas estações de origem para que elas sejam transportadas até as estações de destino. Na modelagem proposta, essa característica é simulada por meio de uma ação externa que é executada de acordo com uma probabilidade matemática pré-definida, adicionando uma nova carga à uma estação de origem.

– **Veículo (classe *Vehicle*):** Todo veículo possui, inicialmente, as ações externas de falha e recuperação, uma vez que também são agentes sujeitos à falhas. Adicionalmente, os veículos implementam algumas ações internas importantes:

(d) **Mover (classe *Move*):** Como todo agente móvel, um veículo tem a capacidade de se locomover pelo ambiente (depósito). Portanto, esse comportamento deve ser implementado na forma de uma ação, pois, indiscutivelmente, a locomoção dos veículos altera o estado do ambiente;

(e) **Receber carga (classe *ReceiveLoad*):** Essa classe implementa a ação de recebimento de uma carga, que pode ser executada quando um veículo no estado *MOVING_PICKUP*, descarregado, localizado nos arredores de uma estação de origem ativa, decide receber uma carga a ser transportada;

(f) **Entregar carga (classe *DeliverLoad*):** Assim que um veículo carregado, no estado *MOVING_DROP*, encontra uma estação de destino, ele pode optar por entregar a carga transportada. Esse comportamento é implementado pela ação definida na classe *DeliverLoad*;

Com relação às ações reversas, é fácil perceber que as ações *falha* (a) e *recuperação* (b) são opostas, ou seja, uma ação é a reversa da outra e vice–versa. O mesmo ocorre com as ações *receber carga* (e) e *entregar carga* (f). Portanto, é necessário definir ações reversas explícitas apenas para as ações *criar carga* (c) e *mover* (d).

Para a primeira, a ação reversa foi implementada internamente (classe interna), portanto não é visível no diagrama de classes. O seu funcionamento é bastante simples: para reverter uma ação de criação de carga, basta destruir a carga recém criada, retirando-a da estação de origem onde foi enfileirada.

Já para segunda, a ação reversa constitui uma variação da própria ação: para reverter um movimento ou locomoção de um veículo, basta fazer com que ele retorne à sua posição original. Portanto, considerando uma ação que mova

um veículo da localização $(0, 0)$ para a localização $(0, 1)$, a ação reversa é dada pelo movimento contrário, ou seja, de $(0, 1)$ para $(0, 0)$.

2. **Definir as variáveis de estado:**

Para cada agente do ambiente, existe um conjunto de variáveis à serem observadas. A união dessas variáveis para todos os agentes do ambiente forma o identificador do estado da simulação:

- **Agente sujeito à falhas (*FailableAgent*):** Para todo e qualquer agente sujeito à falhas, uma variável importante deve ser observada:
  (a) **Estado de funcionamento:** O estado de funcionamento de um agente implementado através da classe *FailableAgent* pode assumir dois valores: *normal* ou *falha*. O primeiro indica que o agente está em perfeito funcionamento, enquanto o segundo indica que ele está em falha.
- **Estação de carga (*LoadStation*):** Tanto para as estações de origem quanto para as de destino, além da indicação do seu estado de funcionamento, uma outra variável deve ser monitorada:
  (b) **Número de cargas:** O número de cargas atualmente enfileiradas define o estado atual de uma estação de carga.
- **Veículo (*Vehicle*):** Finalmente, duas variáveis adicionais definem o estado dos únicos agentes móveis do ambiente:
  (c) **Localização:** A localização dos veículos no ambiente é um fator importantíssimo para estabelecer uma estimativa do custo até o estado objetivo da simulação, uma vez que os veículos são os agentes responsáveis pelo transporte de cargas;
  (d) **Estado de carregamento:** Outro fator importante a ser observado é o estado de carregamento de veículo. Essa variável pode assumir dois valores distintos: *carregado* – que indica que o veículo possui um carga –, ou *descarregado* – indicando que ele está vazio.

3. **Definir os objetivos:**

Um sistema de transporte automatizado de cargas, considerando–se a versão adotada para este estudo de caso (seção 5.1), possui um único objetivo: transportar todas as cargas que chegam nas estações de origem até as estações de destino. Na prática, isso significa que:

(a) Todas as estações de origem devem estar vazias, ou seja, $b_i = 0 \; \forall \; 0 \leq i < n$, onde b reflete a variável *número de cargas* (b) das estações de carga e n é a quantidade de estações de origem existentes no ambiente;

(b) Todos os veículos devem estar descarregados. Isso implica que $d_j = 0 \; \forall \; 0 \leq j < m$, onde d representa a variável *estado de carregamento* dos veículos ($d = 0$ indicando veículo descarregado e $d = 1$ veículo carregado) e m é o número total de veículos no ambiente.

É interessante observar que, através da definição de uma boa função heurística para a avaliação dos estados da simulação, o verificador de sistemas multiagentes acaba por se tornar também um otimizador. Neste caso em específico, a função heurística proposta fornece uma estimativa que considera que, quanto mais cargas tiverem sido entregues, menor será o custo para se atingir o estado objetivo. Então, como o verificador sempre escolherá os caminhos que possuem um custo estimado menor para atingir o objetivo, ele acabará escolhendo os estados que contém mais cargas entregues.

Portanto, considerando–se o *throughput*[3] do sistema como sendo a taxa que relaciona a entrega de cargas com os passos da simulação, o verificador também funcionará como um otimizador que busca maximizar esse *throughput*.

4. **Definir a função de avaliação de estados:**

Sabe–se que uma boa função de estimativa heurística para avaliação de estados deve sempre buscar uma aproximação do limite inferior do melhor caso, ou seja, ela nunca deve ser mais otimista que o melhor caso, mas também não deve se distanciar muito dele (Sadikov, 2008).

Para o problema em questão, a função de avaliação de estados deve fornecer uma estimativa heurística que indique o custo para se atingir o estado objetivo (definido anteriormente) a partir de um determinado estado da simulação. Portanto, deve–se analisar principalmente três variáveis macroscópicas:

(a) o número total de cargas enfileiradas aguardando transporte;

(b) o número total de cargas pendentes, ou seja, cargas que ainda não foram geradas e introduzidas no sistema; e

(c) o número total de cargas sendo transportadas pelos veículos.

[3]Na computação, o termo *throughput* é comumente utilizado para denotar a taxa de transferência de dados em um sistema. Entretanto, também é comum o seu uso para indicar o desempenho de um sistema qualquer, onde se analisa o cumprimento de uma tarefa com relação ao tempo gasto (ex: velocidade de processamento, tempo de resposta, etc.).

Como as estações de origem e destino estão localizadas em extremidades opostas do *grid* que representa o ambiente (depósito), no sentido longitudinal, e considerando que no melhor caso sempre haverá um veículo ao lado de uma estação de origem, cada carga enfileirada implicará em um custo mínimo de transporte c_1 dado pela equação $c_1 = c_{rl} + (w \cdot c_m) + c_{dl}$, onde w é a largura do *grid* e c_{rl}, c_m e c_{dl} são os custos das ações *receber carga*, *mover* e *entregar carga*, respectivamente.

Já o custo total para o transporte de uma carga pendente c_2 pode ser obtido através da fórmula $c_2 = c_{cl} + c_1$, onde c_{cl} é o custo da ação *criar carga*.

Finalmente, considerando as cargas em transporte (veículos atualmente carregados) e assumindo que no melhor caso os veículos carregados encontram–se em localizações vizinhas às posições das estações de destino, o custo c_3 para completar a entrega de cada uma dessas cargas em transporte pode ser dado pela equação $c_3 = c_{dl}$, pois nenhuma ação além de *entregar carga* será necessária.

Portanto, considerando n_1, n_2 e n_3 como sendo o número total de cargas enfileiradas aguardando transporte, o número total de cargas pendentes e o número total de cargas atualmente em processo de transporte, respectivamente, o custo estimado c para completar o transporte de todas as cargas do sistema pode ser dado pela equação $c = (n_1 \cdot c_1) + (n_2 \cdot c_2) + (n_3 \cdot c_3)$.

5.3.1 Estatégia de busca

O último passo para a instanciação do verificador autonômico é opcional – e, por isso, apresentado em uma subseção separada: a definição de uma estratégia de busca para ser utilizada no processo de verificação. Apesar de ser um ponto flexível do *framework*, a definição algoritmos personalizados que apresentem uma convergência melhor para domínios de problema específicos também é permitida.

Entretanto, para o problema abordado neste estudo de caso, optou–se por utilizar um algoritmo de busca heurística já implementado no subsistema de verificação: LRTA* ("*Learning Real-Time A**") (Sadikov, 2008). O seu funcionamento é definido por uma função denominada `lookahead` (figura 5.3). Essa função recebe um estado da simulação e atualiza a sua estimativa de custo com base na menor estimativa dentre os estados vizinhos somada ao custo da ação relacionadas ao caminho entre estes estados.

Através de uma adaptação simples nessa função, é possível fazê-la, além de atualizar a estimativa do estado corrente, verificar se uma determinada ação é a melhor alternativa para este estado. Essa adaptação, então, implementa

```
1  lookahead(curr_state) : void
2    FOR ALL path IN get_paths(curr_state) DO
3      IF NOT visited(path) THEN
4        state = get_state(path)
5        action = get_action(path)
6        estimate = get_estimate(state) + get_cost(action)
7        IF estimate < best_estimate THEN
8          best_estimate = estimate
9        END
10     END
11   END
12   IF best_estimate < get_estimate(curr_state) THEN
13     set_estimate(curr_state, best_estimate)
14   END
15 END
```

Figura 5.3: Algoritmo de busca heurística LRTA* (Sadikov, 2008)

a função `is_best_choice` definida pelas estratégias de busca, tornando o algoritmo LRTA* apto para funcionar como parte do processo de verificação proposto.

5.4 Experimentos

Para validar a abordagem multiagente proposta para o sistema de transporte automatizado de cargas nas seções anteriores deste capítulo, e, principalmente, avaliar o desempenho do verificador proposto no capítulo 4, várias simulações foram realizadas.

A seção 5.4.1 descreve as diferentes configurações utilizadas nas simulações, assim como a preparação dos dados obtidos. Então, na seção 5.4.2 os resultados são discutidos com base nos objetivos estabelecidos e no comportamento esperado.

5.4.1 Preparação

Nessas simulações, utilizou–se uma implementação para este problema desenvolvida em Java, seguindo a modelagem proposta na seção 5.2. A instanciação do verificador, também implementado em Java, baseou–se nos parâmetros e discussões da seção 5.3, onde foram cumpridas todas as etapas teóricas necessárias para a aplicação do método de verificação descrito.

Com relação à simulação propriamente dita, estabeleceu–se um tamanho padrão para o *grid* do ambiente: *11 x 11* (11 posições de largura por 11 posições de altura). As estações de origem e destino foram sempre posicionadas em lados opostos do *grid*, no sentido longitudinal. Adotou–se, também, um espaçamento constante entre as estações do mesmo tipo, variando de acordo com a quantidade de estações. Foram adotadas 250 configurações diferentes para esse mesmo ambiente, de tal forma que fosse possível observar o comportamento do sistema frente à variação de três parâmetros de entrada importantes:

1. número de estações de origem (de 1 à 5);
2. número de estações de destino (de 1 à 5);
3. número de veículos (de 1 à 10).

Como a modelagem proposta para o sistema inclui um certo fator de aleatoriedade (causado pelas ações externas baseadas em probabilidade), cada uma das configurações foi simulada 30 vezes, obtendo como saída a quantidade de passos necessários para completar a simulação, ou seja, entregar todas as cargas nas respectivas estações de destino.

Entretanto, verificou–se que, dentro de uma mesma configuração, alguns dos resultados obtidos eram completamente discrepantes da grande maioria. Isso pode ser explicado pelo fato de algumas ações externas de falha terem ocorrido mais vezes em algumas simulações. Isso poderia ser considerado normal, entretanto, como se sabe, a API de geração de números randômicos em Java não é completamente aleatória, e sim pseudo–aleatória, pois é baseada em um algoritmo que utiliza uma "semente" para geração dos números randômicos. Como essa "semente" é obtida através de informações de data e hora do computador, a geração dos números pode se tornar um pouco tendenciosa.

Esses resultados anômalos poderiam afetar de forma significativa a média e comprometer a confiabilidade da medida. Portanto, por segurança, optou–se por eliminar os resultados discrepantes de cada configuração. Para isso, utilizou–se o critério de Chauvenet (MSCP, 2009), de grande aplicação no âmbito da metodologia para o tratamento de dados amostrais associados à experimentos de medição, eliminando dados duvidosos.

Considerando–se uma sequência de n medições que estatisticamente seguem o comportamento comum da distribuição normal, o critério de Chauvenet afirma que os resultados cujas probabilidades sejam menores que $\frac{1}{2 \cdot n}$ podem ser eliminados. Na prática, isso significa que apenas os resultados cujas probabilidades estejam dentro da faixa $1 - \frac{1}{2 \cdot n}$ são considerados, conforme gráfico da figura 5.4.

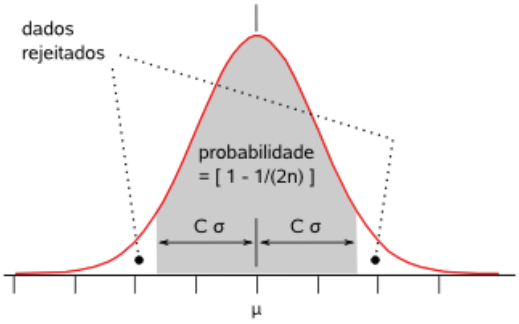

Figura 5.4: Faixa de confiabilidade de valores do critério de Chauvenet (MSCP, 2009)

Para aplicar o critério de Chauvenet a fim de se eliminar os valores duvidosos de uma amostragem, em primeiro lugar calcula–se a média $\overline{x}$ e o desvio padrão x_s desse conjunto de medições $\{x_1, x_2, x_3, ..., x_{n-1}, x_n\}$ através das fórmulas $\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n}(x_i)$ e $x_s = \sqrt{\frac{1}{n+1} \cdot \sum_{i=1}^{n}(x_i - \overline{x})^2}$, respectivamente. Então, obtém–se o coeficiente de Chauvenet pela equação $C = 0,9969 + 0,4040 \cdot ln(n)$. Finalmente, a faixa de confiabilidade dos valores é dada pelo intervalo $[\overline{x} - (C \cdot x_s), \overline{x} + (C \cdot x_s)]$.

Então, após a eliminação dos resultados não confiáveis, calculou–se uma nova média para os resultados das simulações de cada uma das 250 configurações utilizadas. Essa nova média foi o valor considerado para a plotagem dos gráficos e análise dos resultados.

5.4.2 Discussão

Um dos objetivos dos experimentos realizados foi comprovar que a modelagem proposta para o sistema de transporte automatizado de cargas era realmente capaz de cumprir o seu objetivo, ou seja, transportar todas as cargas das estações de origem até as estações de destino.

Demonstrar que esse objetivo foi atingido com a interferência direta do verificador na simulação – isto é, revertendo ações e induzindo a simulação para estados específicos –, implica em atingir o principal objetivo deste trabalho: constatar a importância do papel do verificador proposto em uma simulação de um sistema multiagente.

Quando o verificador é acoplado à uma simulação, ela não termina até que o objetivo avaliado seja cumprido. À medida que as ações monitoradas vão sendo executadas e o verificador analisa os impactos de cada uma delas sobre o estado do ambiente, um relatório de verificação vai sendo formado. Este relatório nada mais é do que um plano de ações que, ao final do processo de

```
...
[EXECUTED] [10 ] CreateLoad: PickupStation #1
[ACCEPTED] [10 ] CreateLoad: PickupStation #1
[EXECUTED] [10 ] Move: Vehicle #15 from (1,2) to (2,2)
[REVERTED] [10 ] Move: Vehicle #15 from (1,2) to (2,2)
[EXECUTED] [10 ] Vehicle #14 from (1,5) to (1,4)
[REVERTED] [10 ] Vehicle #14 from (1,5) to (1,4)
[EXECUTED] [10 ] Vehicle #13 from (2,5) to (3,5)
[REVERTED] [10 ] Move: Vehicle #13 from (2,5) to (3,5)
[EXECUTED] [10 ] Move: Vehicle #12 from (1,6) to (1,7)
[REVERTED] [10 ] Move: Vehicle #12 from (1,6) to (1,7)
[EXECUTED] [10 ] ReceiveLoad: Vehicle #11 from PickupStation #1
[ACCEPTED] [10 ] ReceiveLoad: Vehicle #11 from PickupStation #1
[EXECUTED] [11 ] Move: Vehicle #15 from (1,2) to (2,2)
[ACCEPTED] [11 ] Move: Vehicle #15 from (1,2) to (2,2)
...
[SUCCESS ] [297] Verification succeeded!
```

Figura 5.5: Trecho de um relatório de verificação

verificação, conterá todo o caminho percorrido pela simulação, com o auxílio do verificador, até o seu estado final (objetivo).

É interessante observar que, à partir desse relatório, é possível identificar se o verificador está se comportando da maneira esperada, ou seja, revertendo as ações que não contribuem para o alcance do objetivo, quando da existência de outras ações mais adequadas. Essa decisão, conforme dito anteriormente, cabe à estratégia de busca definida no momento da instanciação do verificador, que, por sua vez, utiliza a função heurística elaborada especificamente para o domínio de problema em estudo.

A figura 5.5 apresenta um trecho do relatório obtido em uma das simulações realizadas que evidencia esse comportamento. Neste exemplo, pode–se perceber que o verificador reverteu uma série de ações de movimento consecutivas que, de acordo com a estimativa heurística utilizada (seção 5.3, etapa 4), não contribuíam para o alcance do objetivo. Isso ocorreu porque, no mesmo momento, um outro veículo estava pronto para receber uma carga enfileirada na estação de origem #1. Portanto, essa ação era prioritária para o verificador, pois reduziria a estimativa de custo para o alcance do objetivo. Então, assim que foi executada, o verificador aceitou a ação *receber carga* sobre a estação de origem #1, pois, conforme o esperado, ela contribuía significativamente para a redução da estimativa do custo total para o alcance do estado objetivo da simulação. Em seguida, no próximo passo, o verificador aceita as ações de movimentação revertidas anteriormente, uma vez que não existe outra ação possível que gere uma estimativa de custo menor.

Os gráficos apresentados na figura 5.7 exibem a frequência de reversão de ações com relação ao número de veículos colocados no ambiente, considerando a melhor e a pior das configurações, que neste caso são: cinco estações de origem e cinco de destino; e apenas uma estação de origem e uma de destino, respectivamente. É perfeitamente visível, em ambos os casos, que, logicamente, a quantidade de ações total cresce linearmente à medida que se aumenta o número de veículos. Entretanto, é interessante observar que o número de ações revertidas pelo verificador também possui uma tendência de crescimento linear, inclusive um pouco maior do que a tendência do crescimento das ações aceitas pelo verificador.

Uma explicação plausível para esse comportamento é o fato de que a função heurística utilizada não considera a posição dos veículos como sendo um fator que influencia o cálculo da estimativa de custo. Portanto, as ações de locomoção acabaram em um segundo plano, sendo os principais alvos das reversões de estado executadas por parte do verificador. Como possível solução para amenizar este problema, poderia se alterar a função de estimativa heurística de tal forma que ela passasse à considerar a proximidade real de cada veículo com relação às estações de carga.

Entretanto, isso não traria uma melhoria significativa para o desempenho do sistema com relação ao seu objetivo; apenas contribuiria para que a simulação sofresse menos reversões nas ações executadas, acelerando o processo de verificação e simplificando o relatório (plano de ações) gerado.

Como o aumento das reversões ocorre principalmente em um mesmo passo da simulação, o *throughput* do sistema acaba não sendo prejudicado. Os gráficos da figura 5.6 evidenciam esse comportamento. Ambos traçam curvas relacionando o desempenho do sistema (*throughput*) – dado pela taxa de entrega de cargas por etapa – com a quantidade de veículos, considerando–se, mais uma vez, a melhor e a pior configuração. Essas curvas apresentam uma tendência de crescimento e estabilização do desempenho, à medida que o número de veículos cresce.

Entretanto, é possível perceber que o aumento da quantidade de veículos no ambiente não melhora o desempenho do sistema na mesma proporção, principalmente considerando–se as configurações com o número de estações de origem e destino menor que o número de veículos. Isso deve–se ao fato de que o gargalo do sistema não está apenas na quantidade de veículos, mas também no número de estações de origem e destino disponíveis; os três fatores são igualmente importantes.

De fato, um cenário onde muitos veículos acessam uma mesma estação apresenta vários congestionamentos, o que reduz o desempenho do sistema.

Isso é perfeitamente observável a partir da análise das tendências das curvas que relacionam o *throughput* ao número de veículos, que já evidenciam uma tendência de estabilização do *throughput* com o número máximo de veículos utilizados. Caso fossem feitas outras simulações com o mesmo número de estações e quantidades ainda maiores de veículos, seria possível observar o início da queda do *throughput* do sistema. Esse comportamento, inclusive, já havia sido observado anteriormente (De Wolf, 2007).

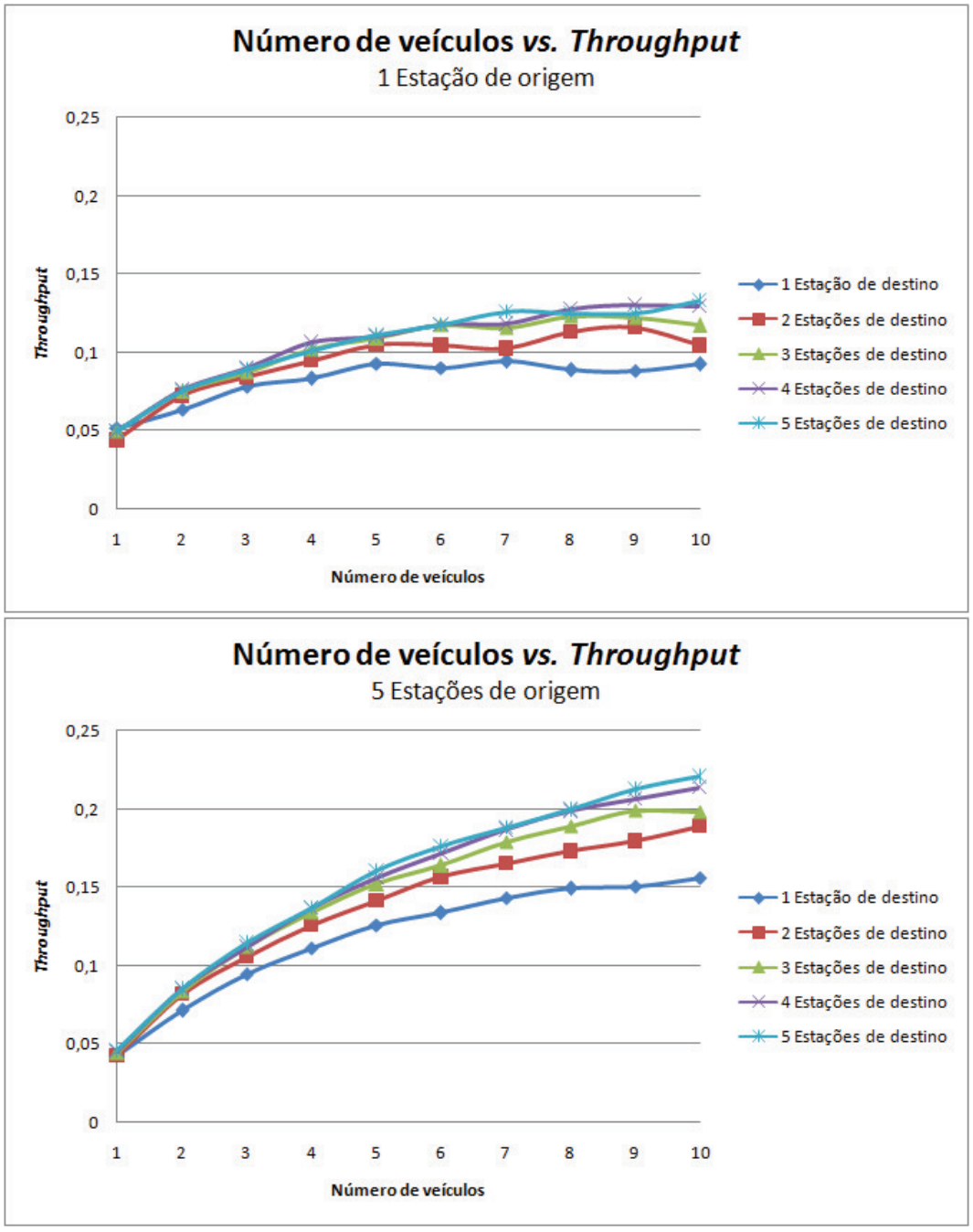

Figura 5.6: Gráfico: Número de veículos vs. *Throughput*

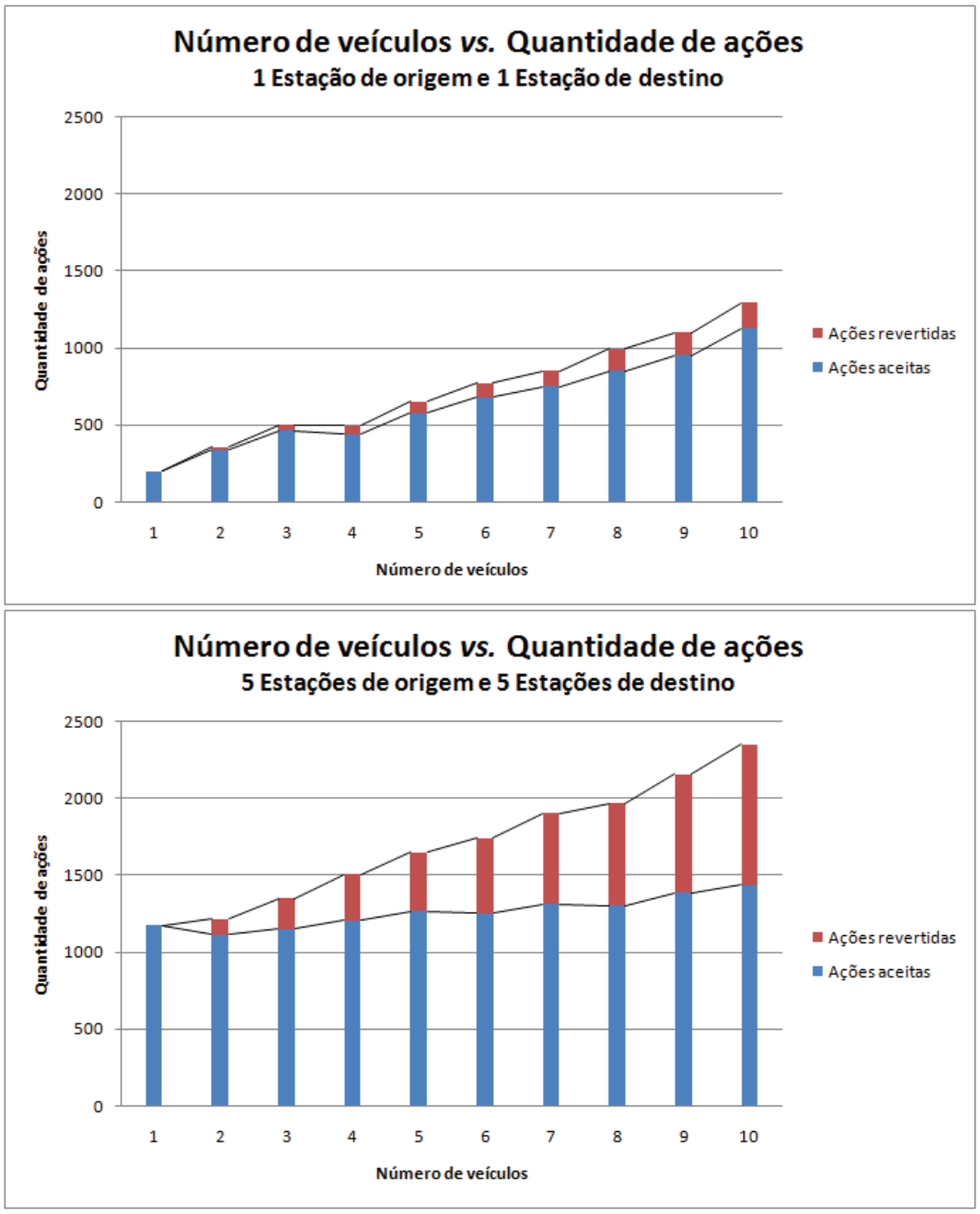

Figura 5.7: Gráfico: Número de veículos vs. Quantidade de ações executadas