UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

JEAN LIMA PIEROBOM

OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS APLICADA AO PROBLEMA DE ATRIBUIÇÃO DE TAREFAS DINÂMICO

DISSERTAÇÃO

CURITIBA

2012

JEAN LIMA PIEROBOM

OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS APLICADA AO PROBLEMA DE ATRIBUIÇÃO DE TAREFAS DINÂMICO

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Computação" – Área de Concentração: Sistemas de Informação.

Orientador: Celso Antonio Alves Kaestner

Co-orientadora: Myriam Regatteri Delgado

CURITIBA

2012

Dados Internacionais de Catalogação na Publicação

P619 Pierobom, Jean Lima
Otimização por nuvem de partículas aplicada ao problema de atribuição de tarefas dinâmico / Jean Lima Pierobom. — 2012.
84 f. : il. ; 30 cm

Orientador: Celso Antonio Alves Kaestner
Coorientadora: Myriam Regatteri Delgado.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2012.
Bibliografia: f. 69-73.

1. Inteligência coletiva. 2. Otimização combinatória. 3. Otimização matemática. 4. Inteligência artificial. 5. Algoritmos. 6. Simulação (Computadores). 7. Computação – Dissertações. I. Kaestner, Celso Antonio Alves, orient. II. Delgado, Myriam Regatteri, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD (22. ed.) 004

Biblioteca Central da UTFPR, Campus Curitiba

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Computação Aplicada – Mestrado Profissional

Título da Dissertação Nº: 01

" Otimização por Nuvem de Partículas Aplicada ao Problema de Atribuição de Tarefas Dinâmico "

por

Jean Lima Pierobom

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM COMPUTAÇÃO APLICADA – Área de Concentração: Engenharia de Sistemas Computacionais, pelo PPGCA - Programa de Pós-Graduação em Computação Aplicada – Mestrado Profissional – da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Curitiba, às 14h do dia 13 de fevereiro de 2012. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:

Prof. Celso Antonio Alves Kaestner, Dr.
(Orientador – UTFPR - CT)

Profª. Gina Maira Barbosa de Oliveira, Drª.
(UFU)

Prof. Julio Cesar Nievola, Dr.
(PUCPR)

Prof. Gustavo Alberto Giménez Lugo, Dr.
(UTFPR - CT)

Profª. Myriam Regatteri Delgado, Drª .
(UTFPR-CT)

Visto da coordenação:

Prof. João Alberto Fabro, Dr.
(Coordenador do PPGCA)\

Av. Sete de Setembro, 3165 - 80230-901 - Curitiba – PR.
Tel. +55 (41) 3310 4644 / e-mail: ppgca-ct@utfpr.edu.br / http://www.ppgca.ct.utfpr.edu.br

Dedico a minha esposa e a minha filha.

AGRADECIMENTOS

- A Deus por ter permitido que eu chegasse até aqui.
- A minha amada esposa Cristiane S. O. Pierobom e a minha filha Julia O. Pierobom, por todo o incentivo e também pela paciência que tiveram comigo nos momentos em que estive ausente.
- Ao Prof. Celso A. A. Kaestner e a Profa. Myriam R. Delgado, que me orientaram com muita dedicação e paciência. Me sinto privilegiado por tê-los como orientador e co-orientadora.
- Aos professores Cesar A. Tacla, Gustavo A. G. Lugo e João A. Fabro, pelas sugestões e correções que fizeram na ocasião dos seminários de acompanhamento do mestrado.
- Aos professores Gilda M. Friedlaender, Júlio C. Nievola, Laudelino C. Bastos, Luiz Nacamura Jr., Paulo C. Stadzisz e Tania M. Centeno, por todo o conhecimento que adquiri em suas disciplinas.
- Ao Prof. Alexandre Denes dos Santos, por ter me recomendado ao programa de mestrado e também pelas idéias transmitidas no início do curso.
- Aos amigos Anderson Augustinho, Axel Dietrichkeit, Eduardo Iwersen, Flávio L. de Oliveira, Hélio H. Sartorato, Hermano Pereira, José Roberto A. Jr., Oeslei T. Ribas, e Thalita Uba, por toda a ajuda.
- Ao Sr. Carsten Moeller pela boa vontade ao me ensinar a utilizar a biblioteca de roteamento utilizada neste trabalho.

RESUMO

PIEROBOM, Jean L. OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS APLICADA AO PROBLEMA DE ATRIBUIÇÃO DE TAREFAS DINÂMICO. 84 f. Dissertação – Programa de Pós-graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

A Inteligência de Enxame (*Swarm Intelligence*) é uma área de estudos que busca soluções para problemas de otimização utilizando-se de técnicas computacionais inpiradas no comportamento social emergente encontrado na biologia. A metaheurística *Particle Swarm Optimization* (PSO) é relativamente nova e foi inspirada no comportamento social de bandos de pássaros. PSO tem apresentado bons resultados em alguns trabalhos recentes de otimização discreta, apesar de ter sido concebido originalmente para a otimização de problemas contínuos. Este trabalho trata o Problema de Atribuição de Tarefas - *Task Assignment Problem* (TAP), e apresenta uma aplicação: o problema de alocação de táxis e clientes, cujo objetivo da otimização está em minimizar a distância percorrida pela frota. Primeiramente, o problema é resolvido em um cenário estático, com duas versões do PSO discreto: a primeira abordagem é baseada em codificação binária e a segunda utiliza permutações para codificar as soluções. Os resultados obtidos mostram que a segunda abordagem é superior à primeira em termos de qualidade das soluções e tempo computacional, e é capaz de encontrar as soluções ótimas para o problema nas instâncias para as quais os valores ótimos são conhecidos. A partir disto, o algoritmo é adaptado para a otimização do problema em um ambiente dinâmico, com a aplicação de diferentes estratégias de resposta às mudanças. Os novos resultados mostram que a combinação de algumas abordagens habilita o algoritmo PSO a obter boas soluções ao longo da ocorrência de mudanças nas variáveis de decisão problema, em todas as instâncias testadas, com diferentes tamanhos e escalas de mudança.

Palavras-chave: Inteligência de Enxame, Otimização por Nuvem de Partículas, Otimização Combinatória, Otimização Dinâmica, Problema de Atribuição de Tarefas

ABSTRACT

PIEROBOM, Jean L. PARTICLE SWARM OPTIMIZATION APPLIED TO THE DYNAMIC TASK ASSIGNMENT PROBLEM. 84 f. Dissertação – Programa de Pós-graduação em Computação Aplicada, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

Swarm Intelligence searches for solutions to optimization problems using computational techniques inspired in the emerging social behavior found in biology. The metaheuristic Particle Swarm Optimization (PSO) is relatively new and can be considered a metaphor of bird flocks. PSO has shown good results in some recent works of discrete optimization, despite it has been originally designed for continuous optimization problems. This paper deals with the Task Assignment Problem (TAP), and presents an application: the optimization problem of allocation of taxis and customers, whose goal is to minimize the distance traveled by the fleet. The problem is solved in a static scenario with two versions of the discrete PSO: the first approach that is based on a binary codification and the second one which uses permutations to encode the solution. The obtained results show that the second approach is superior than the first one in terms of quality of the solutions and computational time, and it is capable of achieving the known optimal values in the tested instances of the problem. From this, the algorithm is adapted for the optimization of the problem in a dynamic environment, with the application of different strategies to respond to changes. The new results show that some combination of approaches enables the PSO algorithm to achieve good solutions along the occurrence of changes in decision variables problem, in all instances tested, with different sizes and scales of change.

Keywords: Swarm Intelligence, Particle Swarm Optimization, Combinatorial Optimization, Dynamic Optimization, Task Assignment Problem

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACO	*Ant Colony Optimization*
AG	Algoritmo Genético
API	*Application Programming Interface*
BE	Busca Exaustiva
BT	Busca Tabu
COP	*Combinatorial Optimization Problem*
DPSO-P	PSO Dinâmico com Codificação de Sequências de Posições
FIFO	*First In First Out*
FSP	*Flowshop Scheduling Problem*
GPS	*Global Positioning System*
JSP	*Jobshop Scheduling Problem*
KP	*Knapsack Problem*
NQP	*N-Queens Problem*
OC	Otimização Combinatória
OSM	*Open Street Map*
OSP	*Openshop Scheduling Problem*
PATC	Problema de Alocação Táxi-Cliente
PSO	*Particle Swarm Optimization*
PSO-B	PSO com Codificação Binária
PSO-P	PSO com Codificação de Sequências de Posições
TAP	*Task Assignment Problem*
TS	Têmpera Simulada
TSP	*Travelling Salesman Problem*

SUMÁRIO

1 INTRODUÇÃO

A tarefa de otimização consiste em buscar soluções para um problema que conduzam a valores extremos de sua função objetivo em um determinado intervalo. A função objetivo é a definição de um critério de desempenho de soluções para o problema que está sendo otimizado, podendo ser de maximização ou minimização. No caso dos problemas de otimização combinatória, geralmente estas soluções são obtidas quando se identifica a combinação mais apropriada para as variáveis de decisão do problema. Considerando que, tipicamente, em problemas de otimização combinatória o domínio de soluções é finito, uma alternativa possível para isto seria enumerar todas as soluções factíveis e então identificar qual é a melhor delas. Entretanto, esta abordagem pode se tornar inviável computacionalmente dependendo do tamanho do espaço de busca, considerando que o número de soluções factíveis cresce exponencialmente conforme aumentam as opções de valores para as variáveis de decisão. Assim, a utilização de metaheurísticas se apresenta como uma boa opção para a resolução de problemas de otimização discreta.

Dentre os mais conhecidos problemas da área de Otimização Combinatória (OC), o Problema de Atribuição de Tarefas - *Task Assignment Problem* (TAP) pode servir de modelo para várias aplicações práticas, como por exemplo: programação de disciplinas em cursos universitários (SHIAU, 2011), alocação de espaço de armazenamento (DIAZ; FERNANDEZ, 2001), carregamento de caminhões (PIGATTI et al., 2005), entre outros. Segundo Salman (2002), não existe um algoritmo capaz de encontrar uma solução ótima em tempo polinomial para o TAP, portanto, é necessário que sejam desenvolvidos métodos de busca heurística para solucioná-lo. Neste trabalho será formalizado o Problema de Atribuição de Tarefas, bem como apresentada uma aplicação prática do problema envolvendo soluções para ambientes estáticos e dinâmicos.

O Problema prático considerado neste trabalho será o de Alocação Táxi-Cliente (PATC), que pode ser categorizado como um Problema de Atribuição de Tarefas. O PATC/TAP consiste em alocar N táxis (agentes de oferta de serviço) para o atendimento de N clientes (agentes de demanda de serviço), de modo que a distância total percorrida no trajeto dos táxis até os clientes

seja mínima.

Este trabalho foca o desenvolvimento e a aplicação do algoritmo de Otimização por Nuvem de Partículas - *Particle Swarm Optimization* (PSO) na resolução do Problema de Atribuição de Tarefas. PSO é uma técnica de otimização metaheurística desenvolvida em 1995 por Kennedy e Eberhart, com base na análise do comportamento inteligente de revoadas de pássaros. O algoritmo PSO faz parte de uma área de estudos denominada Inteligência de Enxame, que contempla uma série de algoritmos que simulam o comportamento social encontrado na natureza e objetiva otimizar problemas computacionais utilizando estas técnicas.

Nesta dissertação, o PSO é empregado para a resolução do problema em um experimento que aplica duas versões discretas do algoritmo: a primeira abordagem é baseada em codificação binária, e a segunda utiliza permutações de sequências de posições. Alguns experimentos são conduzidos para comparar as abordagens em uma simulação do problema em ambiente estático. Os resultados obtidos com ambas as versões do PSO são comparados com o valor ótimo encontrado com uma busca exaustiva, em algumas instâncias do problema em que isso é computacionalmente possível. No caso das outras instâncias, onde a aplicação da busca exaustiva é inviável, os resultados são comparados com as soluções de referência obtidas com um PSO robusto.

O melhor destes dois algoritmos é então adaptado para a resolução do problema em ambiente dinâmico. Novos experimentos são conduzidos para simular mudanças nas variáveis de decisão do problema, de modo a criar um ambiente dinâmico de otimização. Nestes experimentos, algumas abordagens de resposta às mudanças são combinadas entre si, e um comparativo é realizado para identificar as combinações que apresentam melhor desempenho.

1.1 MOTIVAÇÃO

A organização e o planejamento de sistemas de transporte são questões fundamentais para o funcionamento de grandes cidades, e também para o sucesso de eventos como a Copa do Mundo de Futebol e as Olimpíadas (COSTA; COSTA, 2010), eventos cujas próximas edições serão sediadas no Brasil em 2014 e 2016, respectivamente. Os táxis representam um importante meio de transporte urbano e a alocação de táxis para clientes de maneira otimizada pode trazer benefícios em termos da melhoria da mobilidade urbana e da redução das taxas de emissão de gases de efeito estufa, como uma consequência da diminuição do número de táxis por habitante.

Além disso, os Problemas de Otimização Combinatória - *Combinatorial Optimization Problems* (COP) possuem grande aplicabilidade prática e são dos mais conhecidos dentre os

problemas de otimização. Entretanto, ainda não é significativo o número de aplicações do PSO na resolução de problemas desta categoria. Segundo Poli (2008), apenas 3,5% das publicações sobre PSO encontradas no *IEEE Xplore Database* no período de 1995 à 2006 lidam com problemas de otimização combinatória. Isto se deve ao fato de que o PSO foi originalmente desenvolvido para problemas contínuos, e não discretos como é o caso dos problemas de Otimização Combinatória.

Desde a primeira versão do PSO para problemas discretos, desenvolvida por Kennedy e Eberhart (1997), outros trabalhos de otimização combinatória que utilizam o PSO foram elaborados. Em boa parte destes trabalhos, o PSO apresentou melhores resultados quando comparado com outros métodos de otimização bio-inspirados como Algoritmos Genéticos (AG's) (SALMAN, 2002; HU et al., 2003; SHA; HSU, 2006; LIAO et al., 2007; SHA; HSU, 2008; KUO et al., 2009; ROSENDO; POZO, 2010; SHA; LIN, 2010; LIU et al., 2011) e Otimização por Colônia de Formigas, do inglês *Ant Colony Optimization* (ACO) (JARBOUI et al., 2008; SHA; HSU, 2008), além de algumas outras metaheurísticas conhecidas, como Busca Tabu (BT) (SHA; HSU, 2006) e Têmpera Simuada (TS) (FANG et al., 2007). O PSO tem sido pouco utilizado na otimização de problemas de OC, apesar de os resultados se mostrarem promissores nos trabalhos onde isto foi realizado. O estado da arte da aplicação do PSO em COP é detalhado no Capítulo 4.

Além das questões relacionadas com OC, a investigação do problema motiva a realização de um comparativo das diferentes abordagens propostas para a utilização do PSO na otimização de problemas dinâmicos. Os trabalhos de Eberhart e Shi (2001), Hu e Eberhart (2002) e Esquivel e Coello (2004) propõem estratégias para adicionar ao PSO as habilidades necessárias para que o algoritmo seja capaz de reagir às mudanças ocorridas em ambientes dinâmicos de otimização. Os experimentos realizados nesta dissertação mostram que, em algumas situações, a combinação destas abordagens pode apresentar resultados melhores do que aqueles obtidos com a sua utilização individual.

Espera-se que este trabalho possa contribuir com a investigação do problema sob a perspectiva de uma aplicação prática, e com a proposta de uma abordagem eficiente baseada no algoritmo PSO para a sua resolução.

1.2 OBJETIVO GERAL

Este trabalho tem por objetivo solucionar um tipo específico de Problema de Atribuição de Tarefas dinâmico, no caso, o Problema de Alocação Táxi-Cliente, utilizando a metaheurística

PSO.

1.3 OBJETIVOS ESPECÍFICOS

Dentre os principais objetivos específicos, destacam-se:

- Propor um método de otimização para o problema, adaptando um PSO desenvolvido para otimização contínua para o problema considerado que é discreto;
- Propor uma estratégia de resposta às mudanças do problema apresentado, considerando os seus aspectos dinâmicos;
- Comparar os resultados obtidos com as diferentes versões do PSO discreto em ambiente estático;
- Comparar os resultados obtidos com as diferentes estratégias de resposta às mudanças do problema em ambiente dinâmico;
- Mostrar as vantagens e limitações do método proposto.

1.4 CONTRIBUIÇÕES DO TRABALHO

O presente trabalho apresenta contribuições relacionadas aos seguintes aspectos principais:

- O desenvolvimento do algoritmo PSO para a otimização de problemas dinâmicos de otimização combinatória, sendo que os resultados obtidos nesta dissertação caracterizam avanços em pesquisa sobre a aplicação do PSO em problemas desta natureza: em alguns casos, a combinação de abordagens conhecidas se mostrou mais eficiente do que estas mesmas abordagens utilizadas individualmente;
- A exploração de uma aplicação do problema que apresenta relevância prática, tendo como resultado um software piloto que pode ser utilizado futuramente por empresas deste ramo de negócios.

Os resultados obtidos na primeira parte dos experimentos foram publicados na forma de um artigo científico (PIEROBOM et al., 2011), o qual mostra um comparativo de duas versões do PSO discreto aplicadas ao problema em questão. Tais comparativos também são incluídos nesta dissertação de mestrado.

1.5 ESTRUTURA DO TRABALHO

O Capítulo 2 descreve e formaliza o problema abordado neste trabalho, apresentando uma aplicação prática. São descritos ainda os eventos que caracterizam o problema como dinâmico.

No Capítulo 3 são apresentados conceitos sobre Inteligência de Enxame, juntamente com uma revisão do algoritmo de Otimização por Nuvem de Partículas na sua versão original, voltada para a otimização de problemas contínuos.

O Capítulo 4 apresenta uma revisão bibliográfica sobre Otimização Combinatória, incluindo a descrição de algumas categorias de problemas desta natureza e, ainda, as adaptações que se fazem necessárias para a aplicação do PSO em problemas discretos.

O Capítulo 5 traz uma revisão sobre Otimização de Problemas Dinâmicos, onde são apresentadas as técnicas utilizadas para a detecção de mudanças, as estratégias de resposta às mudanças, e as métricas de avaliação de desempenho para algoritmos aplicados em problemas dinâmicos.

O Capítulo 6 aborda a metodologia adotada, e as etapas de desenvolvimento deste trabalho. São apresentados os algoritmos candidatos para otimização do problema proposto, que incluem: PSO discreto com representação binária de partículas e matriz de probabilidades de movimentação sobre o espaço de busca; e PSO discreto com representação de partículas como sequências de posições e movimentação com operações de troca de posições. A melhor das duas abordagens testadas é então empregada na otimização do problema em um ambiente dinâmico, com a combinação de algumas conhecidas estratégias de resposta às mudanças.

Os experimentos realizados neste trabalho são detalhados no Capítulo 7, juntamente com os resultados obtidos. Este capítulo descreve um processo de busca exaustiva por enumeração, o qual possibilita encontrar a solução ótima do problema e avaliar a qualidade das soluções encontradas com os dois algoritmos para instâncias de tamanho reduzido. Além disso, neste capítulo é descrito um PSO robusto, o qual é empregado na identificação de soluções de referência de instâncias do problema que não permitem o emprego da busca exaustiva.

Finalmente, as conclusões finais são apresentadas no Capítulo 8.

2 PROBLEMA DE ATRIBUIÇÃO DE TAREFAS

O Problema de Atribuição de Tarefas - *Task Assignment Problem* (TAP) foi proposto inicialmente por Tank e Hopfield (1987) para ilustrar o uso das redes de Hopfield na otimização de problemas combinatórios. No TAP, existem N tarefas que precisam ser executadas por um conjunto de M agentes, sendo que cada agente tem um desempenho melhor quando lida com certas tarefas, e desempenho pior quando lida com outras. O objetivo é alocar um (e somente um) agente para cada tarefa, de modo que o custo total de execução das tarefas seja o menor possível. Dado um conjunto A de agentes e um conjunto T de tarefas, o problema pode ser formalizado matematicamente como:

$$\begin{aligned} \text{Minimizar} \sum_{i \in A} \sum_{j \in T} x_{ij} c_{ij} \\ \text{sujeito a} \sum_{j \in T} x_{ij} = 1, i \in A \\ \sum_{i \in A} x_{ij} = 1, j \in T \\ x_{ij} \in \{0,1\}, i, j \in A, T, \end{aligned}$$

onde a variável x_{ij} representa a atribuição da tarefa j ao agente i, tomando o valor 1 caso a atribuição seja realizada, e 0 caso contrário; e a variável c_{ij} representa o custo de execução da tarefa j pelo agente i. Quando o problema é resolvido com o mesmo número N de agentes e tarefas, cada conjunto de atribuições é uma permutação de um conjunto de N inteiros. Assim, existem $N!$ maneiras distintas em que as tarefas podem ser atribuídas aos agentes, e para grandes valores de N as tentativas de examinar todas as possíveis permutações utilizando enumeração tornam-se inviáveis, já que o espaço de busca do problema aumenta exponencialmente conforme aumenta o número de tarefas e agentes. Por exemplo, para atribuir 10 tarefas a 10 agentes seria necessário avaliar um total de 10! (ou 3.628.800) permutações diferentes. No caso com 13 tarefas e 13 agentes, seria necessária a avaliação de 13! (ou 6.227.020.800) permutações. Assim, se faz necessário empregar algoritmos que aproximam a solução ótima do problema e que sejam viáveis em termos de tempo de processamento.

Outros problemas de Otimização Combinatória são descritos no Capítulo 4, e a aplicação prática do TAP considerada neste trabalho é apresentada nas próximas seções do presente capítulo.

2.1 APLICAÇÃO DO TAP: PROBLEMA DE ALOCAÇÃO TÁXI-CLIENTE

O Problema de Alocação Táxi-Cliente (PATC), categorizado neste trabalho como um Problema de Atribuição de Tarefas, consiste em alocar N táxis (agentes de oferta de serviço) para o atendimento de N clientes (agentes de demanda de serviço), de modo que a distância total percorrida no trajeto dos táxis até os clientes seja mínima. O espaço de busca S para este problema é o conjunto de diferentes combinações de alocação que podem ser formadas, com tamanho $|S| = N!$. Como no exemplo previamente apresentado, a avaliação de cada uma das soluções por enumeração é inviável para grandes valores de N, já que o espaço de busca do problema aumenta exponencialmente.

O PATC/TAP pode ser formalizado da seguinte maneira: seja A uma função de alocação que mapeia o conjunto de agentes de oferta V para o conjunto de agentes de demanda P, definida da seguinte forma:

$$A : V \rightarrow P, \tag{1}$$

onde $A(i) = j$ se o agente de oferta i estiver alocado ao agente de demanda j; neste trabalho está sendo considerado o caso onde $|V| = |P| = N$. Seja $C(A)$ uma função que calcula o custo de uma solução A:

$$C(A) = \sum_{i=1}^{N} distância(i, A(i)), \tag{2}$$

onde $distância(i, j)$ é a função que calcula o custo do caminho mínimo entre dois pontos na cidade, medido em quilômetros; neste caso, a distância entre os agentes i e j, e $j = A(i)$. Neste trabalho, o caminho mínimo é identificado com a aplicação do algoritmo de Dijkstra (DIJKSTRA, 1959) implementado no mecanismo de roteamento OSM2PO (MOELLER, 2011), e representa a rota com menor distância a ser percorrida por um táxi (agente de oferta de serviço) para atender a um cliente (agente de demanda de serviço). O objetivo é encontrar uma solução A^o com custo mínimo no conjunto de soluções S, ou seja:

$$A^o = argmin \;\; C(A) \qquad \forall A \in S \tag{3}$$

Em outras palavras, o objetivo é encontrar a solução de alocação que apresente a menor distância total a ser percorrida pelos táxis da frota, no trajeto até onde encontram-se os clientes

que estão aguardando. A Figura 1 ilustra uma instância do problema com valor de $N = 13$, ou seja, uma instância do problema com 13 táxis disponíveis para o atendimento de 13 clientes.

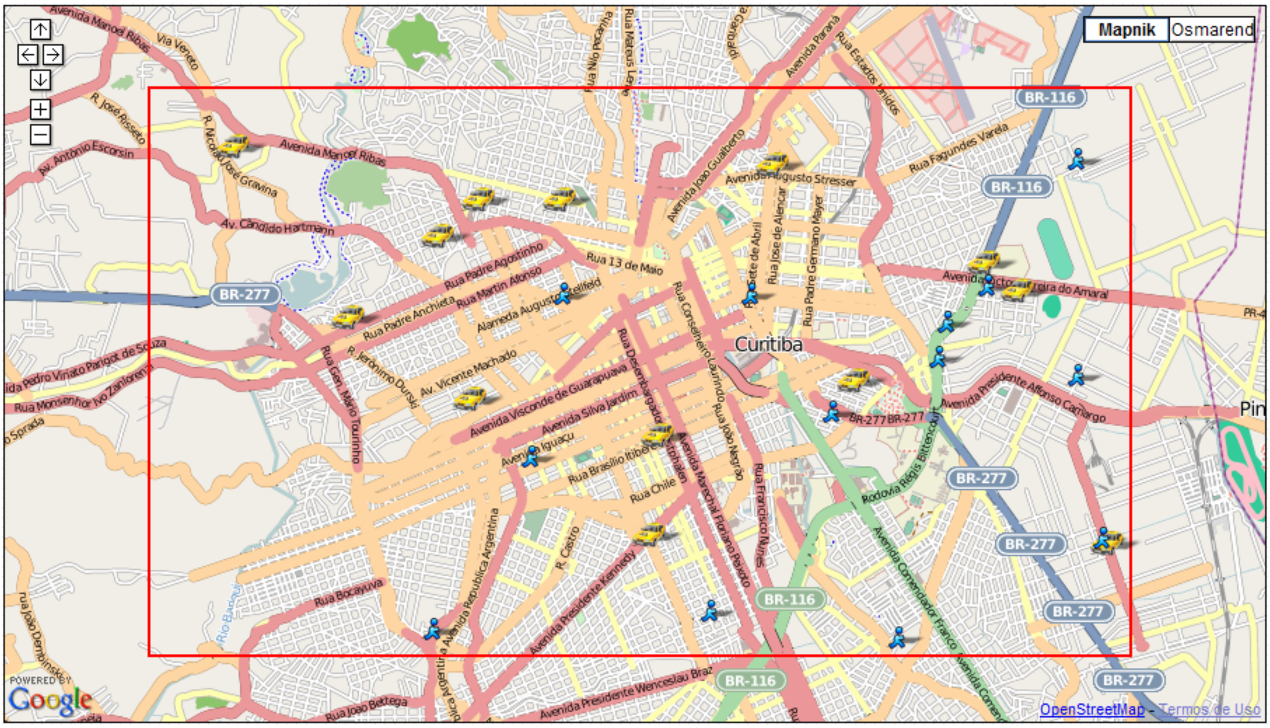

Figura 1: Instância do PATC/TAP com valor de $N = 13$

Fonte: Autoria própria com base em Google e Open Street Map

Ainda sobre a avaliação de custo das soluções: conforme é mostrado no Anexo B, o tempo médio para o cálculo da distância de uma única rota é de 130 milissegundos, em um computador com processador Intel Core 2 Duo 2.40GHz e 3GB de memória RAM. Para a composição da matriz de custo das soluções em uma instância do problema com tamanho $N = 13$, é necessário calcular a distância de $N^2 = 169$ rotas possíveis. No caso de instâncias com tamanho $N = 100$, este número aumenta para $N^2 = 10.000$. Isto acaba elevando o tempo de execução da função *distância*. Embora o custo de cada rota seja calculado apenas uma vez, e esta informação seja reutilizada para compor o custo das soluções que possuam a mesma alocação, o tempo computacional da função *distância* representa mais de 99% do tempo total de execução do algoritmo de melhor desempenho testado neste trabalho, como mostram os experimentos apresentados no Capítulo 7.

2.2 PROBLEMA DINÂMICO DE ALOCAÇÃO TÁXI-CLIENTE

Até aqui, o PATC/TAP foi descrito como um problema estático. Mas em um cenário real, algumas mudanças ocorrem e precisam ser consideradas (Figuras 2, 3, 4 e 5):

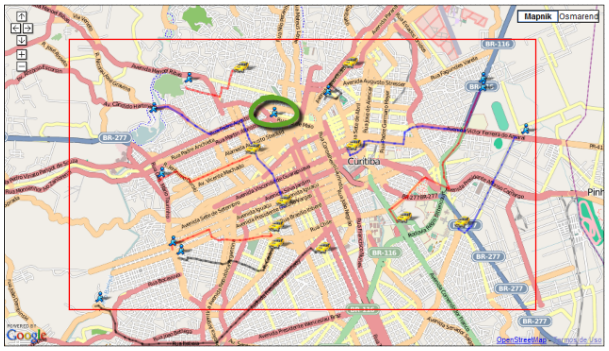

Figura 2: Mudança (a) - surgimento de um novo cliente

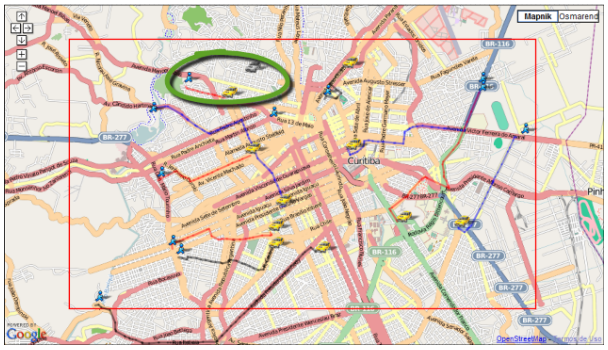

Figura 3: Mudança (b) - deslocamento do táxi

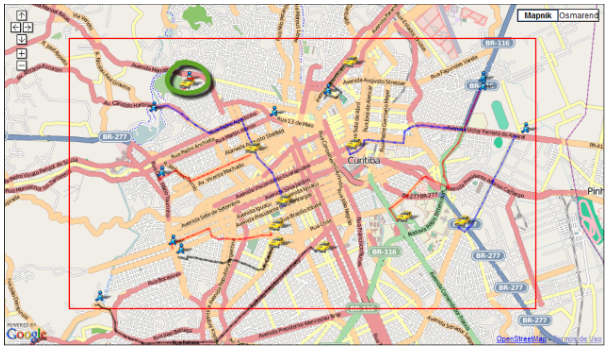

Figura 4: Mudança (c) - táxi chega até o cliente

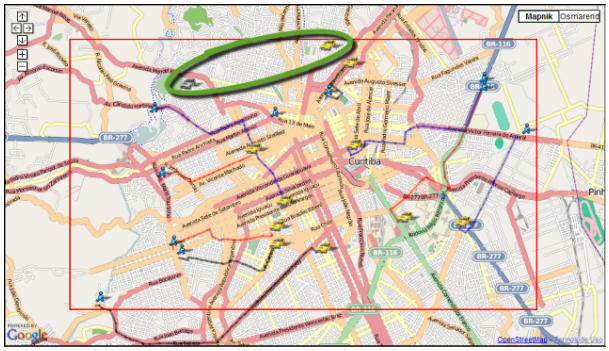

Figura 5: Mudança (d) - táxi fica livre novamente

- (a) um novo cliente surge (o cliente fica “aguardando”);
- (b) a posição do táxi muda devido ao seu deslocamento na direção do cliente para o qual foi alocado, podendo no caminho ficar mais próximo de outros clientes;
- (c) quando o táxi chega até a posição do cliente formando um par (táxi, cliente), e estes elementos precisam ser removidos do cenário de otimização (o táxi está “ocupado”);
- (d) quando o par (táxi, cliente) chega ao destino, o serviço termina, e um novo agente de oferta de serviço aparece no cenário (o táxi fica “livre”).

Estas mudanças caracterizam o PATC/TAP como um problema dinâmico de otimização, cujas avaliações de aptidão das soluções são sujeitas a variação temporal. Na otimização de problemas dinâmicos o objetivo deixa de ser apenas encontrar a solução ótima em um determinado instante. É preciso que o algoritmo empregado na resolução do problema seja capaz de rastrear a solução ótima enquanto ocorrem as mudanças, e para isto, é necessário que sejam adotadas estratégias de detecção e resposta para estas mudanças. No caso do problema em questão, a função de *fitness* não é exatamente dependente do tempo como define a Equação 10, mas sim,

as variáveis de decisão do problema ficam sujeitas a mudanças durante a otimização, fazendo com que o valor de fitness das soluções mude. Diagramas de Estado dos agentes de oferta e demanda são ilustrados nas Figuras 6 e 7, respectivamente.

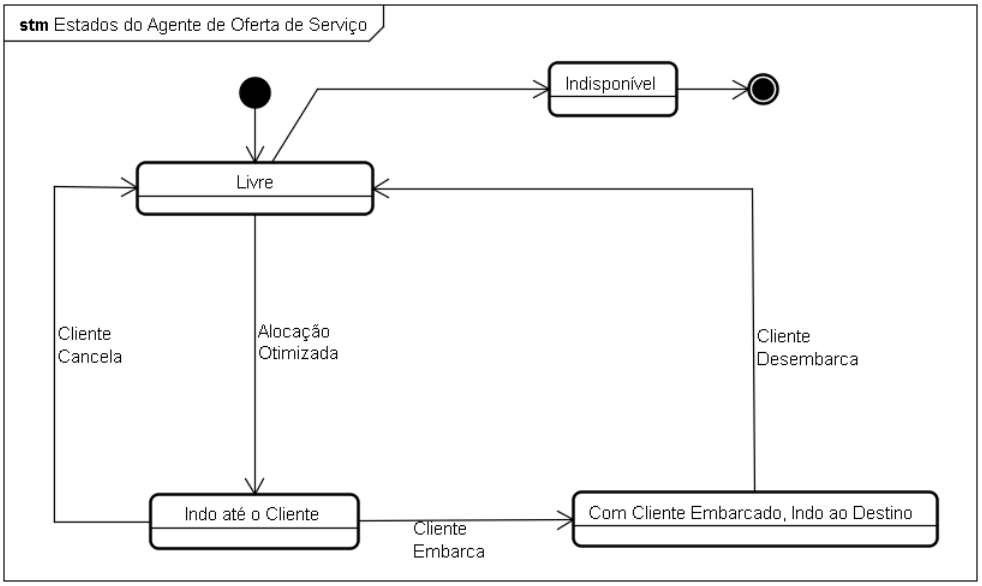

Figura 6: Diagrama de estados - agente de oferta de serviço

Fonte: Autoria própria

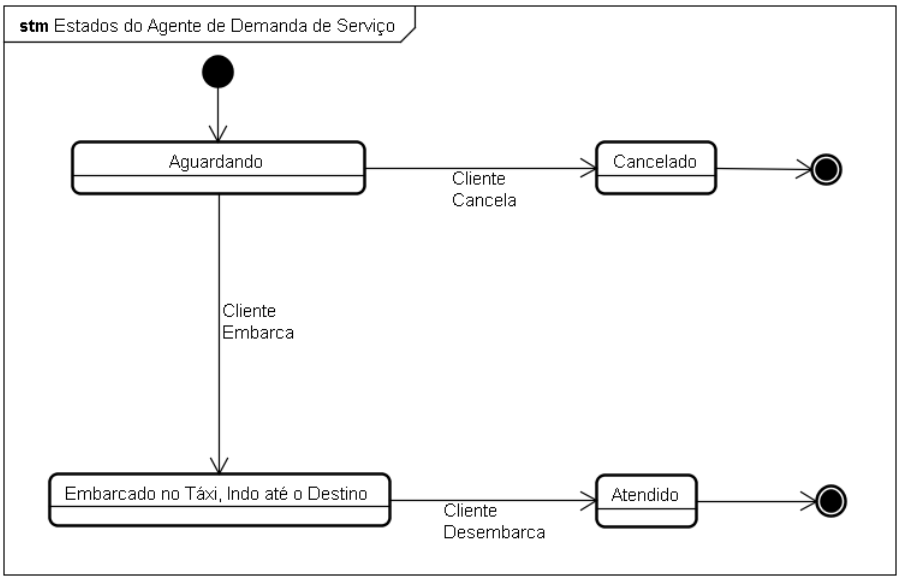

Figura 7: Diagrama de estados - agente de demanda de serviço

Fonte: Autoria própria

A dinâmica da PATC/TAP pode ser descrita da seguinte maneira: os agentes de oferta são incluídos no cenário do problema em um estado chamado “livre”. Da mesma forma, o

estado inicial dos agentes de demanda é “aguardando”. A partir do momento que ocorre a alocação, o estado do agente de oferta se torna “indo até o cliente”, sendo que neste momento se inicia o deslocamento geográfico do agente de oferta na direção da localização do agente de demanda. A alocação não elimina os agentes do cenário de otimização do problema, sendo que a alocação pode ser desfeita quando uma solução melhor for obtida pelo algoritmo.

Quando o agente de oferta chega até a mesma posição geográfica do agente de demanda, formando um par (táxi, cliente), o estado de ambos é modificado para “com o cliente embarcado, indo ao destino” e “embarcado no táxi, indo até o destino”, respectivamente, e ambos ficam indisponíveis no cenário de otimização, de modo que a alocação não seja desfeita. Cada agente de demanda possui um destino que é indicado por um segundo ponto geográfico, na direção do qual se inicia um novo deslocamento do par (táxi, cliente). O estado do agente de demanda é atualizado para “atendido” no momento em que o destino é alcançado, e o agente de oferta retorna para o estado “livre”.

Novamente, neste trabalho está sendo considerado o caso onde $|V| = |P| = N$, o que é viabilizado através da utilização de filas. Os novos agentes incluídos no cenário do problema são armazenados em filas que operam de forma tradicional FIFO (*First In First Out*), sendo uma fila O para os agentes de oferta e uma fila D para os agentes de demanda. Esta abordagem foi adotada para priorizar os agentes de oferta e demanda que estão aguardando alocação por mais tempo. Além disso, um cliente muito distante da frota de táxis poderia levar muito tempo para ser atendido (ou sequer ser atendido), já que grandes distâncias diminuiriam a qualidade da solução. Esta estratégia também permite que o trabalho seja distribuído entre os táxis de uma forma mais justa, fazendo com que todos tenham clientes para atender.

Dado que os agentes de oferta podem atender apenas um agente de demanda por vez, e um agente de demanda deve ser atendido por um (e apenas um) agente de oferta, uma quantidade N destes agentes é calculada (Equação 4), após a identificação de cada mudança ocorrida.

$$N = min\{|O_t|, |D_t|\} \tag{4}$$

Assim, em um dado momento t são selecionados os primeiros N agentes de oferta (Equação 5) e N agentes de demanda (Equação 6) conforme a ordem de inclusão nas respectivas filas, sendo que as soluções iniciais para o problema são criadas aleatoriamente a partir destes agentes. Desta forma, no contexto da otimização sempre existem tantos táxis quanto clientes.

$$V_t = O_t^{Nprimeiros} \tag{5}$$

$$P_t = D_t^{N primeiros} \tag{6}$$

É importante destacar que os diagramas de estado apresentados nas Figuras 6 e 7 são considerados apenas na etapa de simulação do problema, não sendo considerados na etapa de otimização.

No Capítulo 7 são descritos alguns experimentos que mostram mais detalhes sobre a dinâmica do problema, como a demonstração das escalas de mudança e a distância entre as soluções ótimas globais após mudanças no PATC/TAP dinâmico.

3 INTELIGÊNCIA DE ENXAMES

Com a busca por modelos inteligentes inspirados em processos da natureza, constatou-se que várias espécies se beneficiam da sociabilidade. Por mais que os indivíduos destas espécies não possuam inteligência individual, eles são capazes de executar tarefas complexas de maneira inteligente quando trabalham em grupo. A inspiração no comportamento social emergente encontrado na biologia levou a criação de uma área de estudos chamada Inteligência de Enxame.

Johnson (2002) define emergência como a criação de uma organização de alto nível em um grupo social, mesmo que não exista um controle centralizado, onde cada membro do grupo agindo individualmente acaba determinando o comportamento coletivo. O autor cita os exemplos das colônias de formigas, que lidam com o complexo gerenciamento de tarefas sem que exista uma formiga no comando; e dos bairros, que se formam de maneira organizada mesmo sem a existência de um planejador urbano. A emergência pode surgir em sistemas compostos de partes simples em termos cerebrais, quando estas partes interagem entre si fazendo com que seja criada uma estrutura maior (ou mais inteligente). Diante disso, a emergência é aquilo que ocorre quando o todo é mais inteligente do que a soma de suas partes.

Alguns modelos computacionais foram inspirados nestes processos naturais para se beneficiarem da inteligência coletiva. Por exemplo, as formigas agindo coletivamente são capazes de explorar o ambiente e encontrar o menor caminho entre o ninho e a fonte de alimentos. Os estudos das suas habilidades resultaram na criação do algoritmo de Otimização por Colônia de Formigas, que vêm sendo amplamente aplicado em problemas de otimização (ENGELBRECHT, 2007). Outra técnica que se beneficia da inteligência coletiva é a Otimização por Nuvem de Partículas, criada a partir dos estudos da coreografia graciosa e imprevisível dos bandos de pássaros.

Segundo Zuben e Attux (2008), a Inteligência de Enxame emerge de um sistema resultante das seguintes propriedades:

- Proximidade: os agentes devem ser capazes de interagir;

- Qualidade: os agentes devem ser capazes de avaliar seus comportamentos;
- Diversidade: permite ao sistema reagir diante de situações inesperadas;
- Estabilidade: nem todas as variações ambientais devem afetar o comportamento de um agente;
- Adaptabilidade: capacidade de adequação a variações no ambiente.

3.1 OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS

O algoritmo *Particle Swarm Optimization*, proposto por Kennedy e Eberhart em 1995, foi inspirado no comportamento social encontrado nos bandos de pássaros. A população no PSO, denominada nuvem (ou enxame), é composta por partículas que são soluções candidatas para o problema. Analogamente aos bandos de pássaros, cada partícula age como se fosse uma ave do bando em busca de comida, utilizando as informações das melhores posições encontradas no espaço do problema, por ela mesma e pelas demais partículas da nuvem.

Reeves (1983) já utilizava um sistema de partículas em seus trabalhos de animação gráfica na Lucasfilm. Ele criou um sistema estocástico que movia uma série de pontos para formar objetos difusos como explosões e nuvens. Posteriormente, Reynolds (1987) modificou o componente de movimentação das partículas adicionando orientação e comunicação entre os objetos do sistema. No trabalho que originou o PSO, Kennedy e Eberhart (1995) estenderam o modelo de Reynolds para incorporar comportamento social ao sistema.

Um sistema de nuvem de partículas inicia o processo de otimização com uma população de soluções aleatórias, e busca pela solução ótima atualizando as potenciais soluções através de iterações, assim como acontece com os Algoritmos Genéticos (EBERHART; SHI, 1998). Entretanto, o PSO não apresenta os operadores de mutação e *crossover* como em AG. Ao invés disso, as partículas "voam" sobre o espaço de busca procurando por melhores soluções (HU et al., 2003). Diferentemente dos Algoritmos Genéticos, cujas soluções atuam de maneira competitiva para perpetuarem suas características para a próxima geração, no PSO as soluções colaboram entre si em busca da solução ótima (BANKS et al., 2007).

PSO ganhou interesse mundial recentemente (LIU et al., 2011), e um dos motivos que atraem a sua utilização é o pequeno esforço demandado para parametrização, já que uma versão do algoritmo com poucos ajustes pode apresentar uma larga variedade de aplicações (HU et al., 2003).

3.1.1 TERMINOLOGIA

A seguir são listados os principais termos encontrados na literatura sobre Otimização por Nuvem de Partículas:

- **Nuvem**: é a população de soluções do algoritmo;
- **Função de *fitness***: é a função que avalia a adequação das soluções do problema, dadas as suas posições no espaço de busca;
- **Partícula**: é um membro individual da nuvem que representa uma solução potencial para o problema que está sendo otimizado. O que diferencia as partículas umas das outras é a sua posição sobre o espaço de busca, e consequentemente o valor calculado pela função de *fitness* sobre esta posição;
- **Topologia de vizinhança**: determina o conjunto de partículas que será utilizado como vizinhança de uma determinada partícula (COELLO; REYES-SIERRA, 2006);
- **Líderes**: são as melhores partículas da população dada uma determinada topologia de vizinhança (COELLO; REYES-SIERRA, 2006);
- **Velocidade**: determina a direção e a intensidade da movimentação da partícula sobre o espaço de busca, com o objetivo de melhorar a sua posição atual. A velocidade é atualizada durante as iterações, e esta atualização depende da estratégia que está sendo utilizada (COELLO; REYES-SIERRA, 2006);
- **Inércia (w)**: faz com que a partícula continue se movendo na mesma direção das iterações anteriores. Quanto mais alto o valor da inércia, mais improvável será a situação em que a partícula percorre posições já visitadas anteriormente, ou seja, o fator de inércia previne que a partícula se mova novamente para a posição atual (SHA; HSU, 2006). O fator de inércia permite que a velocidade da partícula seja reduzida dinamicamente, propiciando um melhor aproveitamento da região atualmente sendo explorada (BANKS et al., 2007);
- **Fator cognitivo individual (c1)**: determina qual a influência da autoconfiança da partícula no cálculo da velocidade, e consequente movimentação sobre o espaço de busca (BANKS et al., 2007);
- **Fator social (c2)**: determina a influência da experiência do grupo na movimentação da partícula (BANKS et al., 2007). É importante ressaltar que no enfoque do algoritmo, os termos “cognitivo” e “social” implicam apenas metáforas, o que é diferente da modelagem formal de tais conceitos;

- **pbest**: é a melhor posição do espaço de busca por onde já passou uma determinada partícula;
- **gbest**: é a melhor posição do espaço de busca por onde já passou qualquer partícula considerando toda a população.

3.1.2 O ALGORITMO PSO

Conforme é possível observar no Algoritmo 1, a nuvem é inicializada através da distribuição aleatória das partículas dentro do espaço de busca. Em seguida, inicia-se um processo iterativo onde a posição de cada partícula é alterada adicionando-se uma velocidade a sua posição corrente, fazendo com que a partícula se movimente sobre o espaço em busca de melhores soluções.

Algoritmo 1 Pseudo-código do algoritmo PSO para um problema de maximização

1: $t = 1$
2: **for** $k = 1$ to $NPartículas$ **do**
3: $\quad \mathbf{x}_k^1 \leftarrow$ uma solução aleatória
4: $\quad \mathbf{v}_k^1 \leftarrow$ uma velocidade aleatória $\in [V_{MIN}, V_{MAX}]$
5: $\quad \mathbf{pbest}_k \leftarrow \mathbf{x}_k^1$
6: **end for**
7: **gbest** $\leftarrow$ a melhor solução $\in [\mathbf{x}_1, \mathbf{x}_{NPartículas}]$
8: **while** não atingir condição de parada **do**
9: $\quad$ **for** $k = 1$ to $NPartículas$ **do**
10: $\quad\quad$ **if** $fitness(\mathbf{x}_k^t) > fitness(\mathbf{pbest}_k)$ **then**
11: $\quad\quad\quad \mathbf{pbest}_k \leftarrow \mathbf{x}_k^t$
12: $\quad\quad$ **end if**
13: $\quad\quad$ **if** $fitness(\mathbf{x}_k^t) > fitness(\mathbf{gbest})$ **then**
14: $\quad\quad\quad \mathbf{gbest} \leftarrow \mathbf{x}_k^t$
15: $\quad\quad$ **end if**
16: $\quad\quad \mathbf{v}_k^t = w\mathbf{v}_k^{t-1} + c_1 r_1(\mathbf{pbest}_k - \mathbf{x}_k^{t-1}) + c_2 r_2(\mathbf{gbest} - \mathbf{x}_k^{t-1})$
17: $\quad\quad \mathbf{v}_k^t \in [V_{MIN}, V_{MAX}]$
18: $\quad\quad \mathbf{x}_k^t \leftarrow \mathbf{x}_k^{t-1} + \mathbf{v}_k^t$
19: $\quad$ **end for**
20: $\quad t = t + 1$
21: **end while**

Uma função de avaliação (*fitness*) mede a qualidade de cada posição ocupada pela partícula sobre o espaço de busca. Esta função indica o quão boa é a posição em que a partícula se encontra em uma determinada iteração. Cada partícula k armazena na memória a sua melhor posição já alcançada ($\mathbf{pbest}_k$) e a melhor posição já alcançada entre todas as partículas da nuvem (**gbest**). Estes dois componentes em adição ao fator de inércia resultam na velocidade de movimentação (e direção) da partícula, como definido pela Equação 7:

$$\mathbf{v}_k^t = w.\mathbf{v}_k^{t-1} + c_1.r_1(\mathbf{pbest}_k - \mathbf{x}_k^{t-1}) + c_2.r_2(\mathbf{gbest} - \mathbf{x}_k^{t-1}), \tag{7}$$

onde w é o fator de inércia que força a partícula a mover-se na mesma direção da iteração anterior, $c1$ é o fator cognitivo que indica a autoconfiança da partícula, $c2$ é o fator social que força a partícula a seguir na direção da melhor partícula da nuvem, $r1$ e $r2$ são números aleatórios entre $[0,1]$ que ajudam a evitar que a nuvem fique presa a uma região de ótimo local, $\mathbf{pbest}_k$ é a posição com melhor *fitness* encontrado pela partícula k, e **gbest** é a posição do espaço de busca onde foi encontrado o melhor *fitness* entre todas as partículas da população. É importante observar que a posição e a velocidade da partícula são consideradas quantidades vetoriais no espaço de busca S.

Para evitar que a partícula se disperse demasiadamente do restante da nuvem, é possível aplicar um limitador para a velocidade, como mostra a Equação 8:

$$\mathbf{v}_k^t \in [V_{MIN}, V_{MAX}], \tag{8}$$

onde V_{MIN} e V_{MAX} são parâmetros do sistema.

A atualização da posição da partícula é definida pela Equação 9:

$$\mathbf{x}_k^t = \mathbf{x}_k^{t-1} + \mathbf{v}_k^t, \tag{9}$$

sendo que a movimentação da partícula consiste simplesmente em adicionar a velocidade a sua posição atual.

O processo é repetido até que uma condição de parada seja satisfeita, que pode ser encontrar um valor aceitável para a solução ótima, ou executar um número máximo de iterações.

Uma revisão geral do PSO pode ser encontrada nos trabalhos de BANKS et al. (BANKS et al., 2007, 2008), os quais reunem os conceitos desta metaheurística e as suas possíveis aplicações.

4 OTIMIZAÇÃO COMBINATÓRIA

Os problemas de otimização podem ser categorizados entre os que codificam as suas soluções com variáveis contínuas (otimização contínua), e os que o fazem com variáveis discretas (otimização combinatória) (PAPADIMITRIOU; STEIGLEITZ, 1982). Em problemas de otimização combinatória, os valores que podem ser assumidos pelas variáveis que compõem a solução são obtidos através de um conjunto finito (ou infinito enumerável).

A seguir são descritos alguns problemas de Otimização Combinatória que foram atacados por meio de técnicas de PSO pela comunidade da área, como apresentado na sequência.

4.1 PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA

O trabalho de Blum e Roli (2003) define um problema de Otimização Combinatória $P = (S, f)$ como:

- Um conjunto de variáveis discretas, $X = \{x_1, ..., x_n\}$, e os domínios das variáveis $D1, ..., Dn$;
- Restrições entre as variáveis;
- Uma função objetivo f a ser maximizada (ou minimizada), onde $f : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}^+$;

O conjunto das soluções viáveis é definido por: $S = \{s = \{(x_1, v_1), ..., (x_n, v_n)\} | v_i \in D_i\}$, em que s é uma solução candidata e satisfaz todas as restrições do problema. Dado um COP, o objetivo é encontrar uma solução ótima $s^* \in S$, cujo valor da função objetivo seja o maior possível (em um problema de maximização), ou seja, $f(s^*) \geq f(s) \forall s \in S$ (BLUM; ROLI, 2003).

O aumento no tamanho da dimensão deste tipo de problema pode gerar uma explosão combinatória, uma vez que a sua complexidade aumenta exponencialmente. Não existem algoritmos capazes de encontrar uma solução exata para problemas de OC em tempo polinomial, desta forma o desafio de desenvolver algoritmos de otimização para COP, aliada a sua grande

aplicabilidade prática, atraiu a atenção de pesquisadores para o tema nos últimos anos (BLUM; ROLI, 2003).

Nas próximas seções são apresentados alguns dos mais conhecidos problemas combinatórios existentes na literatura. Como acontece com o TAP, estes problemas também têm sido frequentemente investigados em trabalhos que propõem diferentes abordagens baseadas no PSO para a resolução de problemas combinatórios.

4.1.1 PROBLEMAS DE PROGRAMAÇÃO E AGENDAMENTO

A programação de processos produtivos é fundamental para manter empresas com alta competitividade no mercado, o que fez com que os Problemas de Programação de Tarefas tenham sido largamente estudados em trabalhos recentes da área, em busca do desenvolvimento de tecnologias e abordagens de programação avançadas para tal finalidade (LIU et al., 2008). Problemas deste tipo consistem em programar a execução de N tarefas para serem processadas em M máquinas.

Os Problemas de Programação de Tarefas geralmente são classificados conforme o fluxo de operação das tarefas nas máquinas. Segundo Nagano et al. (2004), as classificações mais difundidas são:

- *Openshop Scheduling Problem* (OSP): Não existem restrições de ordenação para o processamento das tarefas nas máquinas;
- *Jobshop Scheduling Problem* (JSP): Existe uma sequência pré-definida para o processamento de cada tarefa nas máquinas;
- *Flowshop Scheduling Problem* (FSP): Cada tarefa tem exatamente uma operação, em cada máquina. As tarefas seguem a mesma sequência de processamento nas máquinas.

O trabalho de (LIU et al., 2011) define o FSP da seguinte forma:

$$C_{i,j} = \begin{cases} p_{i,j} & i = j = 1 \\ p_{i,j} + C_{i-1,j} & j = 1, i > 1 \\ p_{i,j} + C_{i,j-1} & j > 1, i = 1 \\ p_{i,j} + max\{C_{i-1,j}, C_{i,j-1}\} & i > 1, j > 1 \end{cases}$$

$$C_{max} = C_{M,N},$$

sendo que $p_{i,j}$ e $C_{i,j}$ representam o tempo de processamento e o tempo de conclusão da tarefa j processando na máquina i, respectivamente, e C_{max} é o tempo de fluxo do sequenciamento

de tarefas. O objetivo ao se resolver este problema é encontrar o sequenciamento das tarefas que resulte no menor tempo de fluxo (LIU et al., 2011). Entre os problemas de programação e agendamento, o FSP é o que apresenta maior similaridade com o problema tratado neste trabalho, por este motivo foi apresentada a sua formalização nesta seção.

4.1.2 PROBLEMA DA MOCHILA

O Problema da Mochila - *Knapsack Problem* (KP) pode ser definido da seguinte maneira: existem N objetos, com diferentes pesos e valores, os quais se deseja armazenar em uma mochila. O objetivo é armazenar o maior valor de itens, de modo que a soma dos respectivos pesos não exceda a capacidade de armazenamento W da mochila. Matematicamente, o problema é formalizado por:

$$\begin{aligned} \text{Maximizar} \quad & \sum_{i \in N} x_i v_i \\ \text{sujeito a} \quad & \sum_{i \in N} x_i w_i \leq W, \\ & x_i \in \{0,1\}, i = 1,2,...,N, \end{aligned}$$

onde v_i é o valor do item i, w_i é o peso do item i, e a variável x_i toma o valor 1 caso o item i seja colocado na mochila, e 0 caso contrário.

4.1.3 PROBLEMA DAS N-RAINHAS

O Problema das N-Rainhas - *N-Queens Problem* (NQP), é um clássico de satisfação de restrições, que consiste em colocar N rainhas sobre um tabuleiro de xadrez de modo que uma rainha não possa ser atacada por outra, ou seja, em cada linha, coluna ou diagonal, deve existir apenas uma rainha (HU et al., 2003). Uma solução válida para o NQP é ilustrada na Figura 8.

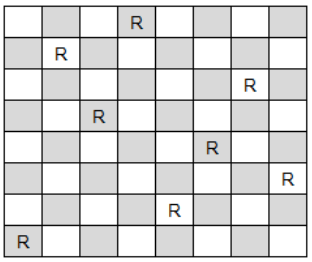

Figura 8: Solução válida para o NQP

Fonte: Autoria própria

Dado um tabuleiro de xadrez de tamanho N, o problema pode ser formalizado da se-

guinte maneira:

$$
\begin{aligned}
\text{Maximizar} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij} \\
\text{sujeito a} \quad & \sum_{i=1}^{N} d_{ij} \leq 1, j = 1,...,N \\
& \sum_{j=1}^{N} d_{ij} \leq 1, i = 1,...,N \\
& \sum_{i=1}^{N}\sum_{j=1}^{N}{}_{i+j=k} d_{ij} \leq 1, k = 2,...,2N \\
& \sum_{i=1}^{N}\sum_{j=1}^{N}{}_{i-j=k} d_{ij} \leq 1, k = 1-N,...,N-1 \\
& d_{ij} \in \{0,1\}, i, j = 1,...,N,
\end{aligned}
$$

onde $d_{ij} = 1$ se uma rainha estiver ocupando a posição (i, j); e caso contrário $d_{ij} = 0$.

4.1.4 PROBLEMA DO CAIXEIRO VIAJANTE

Certamente, o mais famoso e mais estudado problema de OC é o Problema do Caixeiro Viajante - *Traveling Salesman Problem* (TSP). Neste problema, uma série de cidades deve ser visitada pelo caixeiro viajante, e este deve procurar o caminho mais curto para visitar todas as cidades uma única vez e retornar à cidade inicial (VOUDOURIS, 1999). Embora o TSP já tenha sido objeto de estudo para o desenvolvimento de muitos algoritmos, este problema continua atraindo a atenção de pesquisadores devido a sua facilidade em representar outros problemas para os quas não existem algoritmos de tempo polinomial.

Christofides et al. (1979) formulam o TSP como um problema de programação 0-1 sobre um grafo $G = (N,A)$ onde N é o conjunto de nós (cidades que devem ser visitadas) e A é o conjunto de arcos (caminhos entre as cidades), como se segue:

$$
\begin{aligned}
\text{Minimizar} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij} \\
\text{sujeito a} \quad & \sum_{i=1}^{N} x_{ij} = 1, j = 1,...,N \\
& \sum_{j=1}^{N} x_{ij} = 1, i = 1,...,N \\
& x_{ij} \in \{0,1\}, i, j = 1,...,N,
\end{aligned}
$$

onde a variável c_{ij} indica o custo do deslocamento da cidade i até a cidade j. Se o arco $(i, j) \in A$

for escolhido para compor a solução, então $x_{ij} = 1$, e caso contrário $x_{ij} = 0$. .

4.2 APLICAÇÃO DO PSO EM PROBLEMAS COMBINATÓRIOS

Originalmente, o PSO foi concebido para ser aplicado em problemas de natureza contínua, com espaço de busca dado por números reais. Entretanto, muitos problemas práticos tratados na atualidade podem ser representados por problemas de otimização combinatória e suas variáveis de decisão precisam ser codificadas de maneira discreta. A primeira versão do PSO para problemas discretos foi desenvolvida no trabalho de Kennedy e Eberhart (1997). Desde então, outros trabalhos foram publicados com propostas de algoritmos baseados no PSO para a otimização de problemas combinatórios. Para que esta técnica possa ser aplicada em um COP, a sua abordagem clássica precisa passar por algumas adaptações, como redefinir a representação da partícula para um modelo discreto, e adaptar os operadores de velocidade (LIU et al., 2011). As equações originais do PSO previamente apresentadas são mantidas em alguns dos algoritmos propostos nestes trabalhos, mas isto não é uma regra geral.

Poli (2008) realizou um levantamento sobre os trabalhos de aplicação do PSO, considerando os artigos que foram publicados no *IEEE Xplore Database* entre os anos de 1995 e 2006. Este levantamento mostra que houve uma evolução considerável na quantidade de publicações. Entretanto, foi evidenciada a escassez de trabalhos que aplicam o PSO especificamente em problemas combinatórios, já que apenas 24 dos quase 650 artigos selecionados para citação são relacionados a CO. Dentre estes artigos, encontram-se trabalhos de otimização de problemas como o TSP, KP, NQP, entre outros.

No primeiro trabalho que propôs uma versão discreta do algoritmo, Kennedy e Eberhart (1997) codificaram a partícula k como uma matriz de valores binários $\mathbf{X}_k = (x_{k,11}, x_{k,12}, ..., x_{k,nn})$, $x_{k,ij} \in \{0,1\}$, e as velocidades e trajetórias das partículas foram definidas como uma matriz $\mathbf{V}_k = (v_{k,11}, v_{k,12}, ..., v_{k,nn}), v_{k,ij} \in \mathbb{R}$ de probabilidades de esses valores binários mudarem de 0 para 1. No espaço de busca binário, o deslocamento da partícula pode ser visto como a quantidade de bits alterados de uma iteração para outra. Uma partícula não se move quando nenhum bit da matriz é modificado, e se move o mais distante possível quando todos os seus bits são invertidos. A movimentação da partícula foi definida como a variação da probabilidade de uma posição tomar um estado ou outro, sendo que este espaço de estados é restrito ao intervalo $[0,1]$ com a aplicação da função sigmóide $S(v_{k,ij})$. Conforme o exemplo do autor, se $v_{k,ij} = 0,20$, então as chances de o bit $x_{k,ij}$ se tornar 1 são de 20%, e 80% de se tornar 0. A Equação 7 da versão original do PSO para problemas contínuos é mantida. O PSO discreto com matriz binária de representação das partículas é detalhado no Capítulo 6, onde é utilizado como base

em um dos algoritmos implementados neste trabalho para a otimização do PATC/TAP.

O trabalho de Liao et al. (2007) utilizou a mesma abordagem de codificação das partículas com matriz binária, adaptando o algoritmo para a otimização do FSP. Os experimentos realizados mostraram desempenho superior do PSO quando comparado com AG, inclusive na otimização do FSP com múltiplos objetivos.

Hu et al. (2003) desenvolveram um PSO discreto que codifica as partículas como sequências numéricas de posições, e define a movimentação das partículas como permutações nestas sequências das posições. O autor realizou um estudo de caso do Problema das N-Rainhas. No contexto da otimização combinatória, a velocidade das partículas foi definida como a chance de ocorrerem alterações na sequência numérica de posições. A Equação (7) do PSO clássico foi preservada neste algoritmo, entretanto, a velocidade é normalizada para o intervalo $[0,1]$ após o cálculo. A cada iteração do processo de busca heurística, as partículas sofrem trocas de posições de acordo com a probabilidade determinada pela velocidade. Se a troca de posições for determinada, a posição atual é trocada com a posição que armazena o mesmo valor em *gbest*. A equação de atualização da velocidade da partícula (7) faz com que a movimentação ocorra independentemente nos três componentes de movimentação, permitindo que duas ou mais posições obtenham o mesmo valor após a atualização da posição e assim gerando soluções inválidas para o problema, entretanto, o uso de permutações faz com que estes conflitos sejam eliminados. O algoritmo proposto por Hu et al. (2003) se mostrou competitivo conforme as comparações realizadas com AG, e também foi utilizado como base para a otimização do PATC/TAP no presente trabalho, sendo detalhado no Capítulo 6.

O trabalho de Liu et al. (2008) propõe um algoritmo híbrido baseado no PSO para otimizar o problema FSP. Neste trabalho, a discretização da partícula ocorre com a aplicação de uma regra de classificação de valores sobre a partícula $X_i = [x_{i1}, x_{i2}, ..., x_{in}]$ representada no espaço contínuo, para a obtenção da permutação $\pi = [\pi(1), \pi(2), ..., \pi(n)]$. Com esta regra de classificação, o índice 1 é atribuído a posição de menor valor, o índice 2 é atribuído a segunda posição de menor valor, e assim por diante. Como no exemplo dos autores, dada uma partícula $X_i = [0.06, 2.99, 1.86, 3.73, 2.13, 0.67]$ poderia ser obtida a permutação de tarefas $\pi = [1,5,3,6,4,2]$.

No trabalho de Sha e Hsu (2006), o problema JSP é otimizado com uma nova versão do PSO para problemas combinatórios. A partícula é representada por uma matriz binária X_{MN} de M máquinas por N tarefas e a sua discretização é realizada utilizando a heurística de Giffler e Thompson, em um processo de decodificação da posição no espaço contínuo para uma programação de tarefas. Assim como no trabalho de Kennedy e Eberhart (1997), $x_{ij} = 1$ indica

que a tarefa j será processada pela máquina i. A movimentação da partícula utiliza operação de troca de posições (*swap*), da seguinte maneira: quando $v_{ij} = 1$, a tarefa j de x_i será movida para a respectiva posição de $pbest_i$ com uma probabilidade c_1, e será movida para a respectiva posição de *gbest* com uma probabilidade c_2. O trabalho propõe ainda uma estratégia de diversificação da nuvem para que o processo de otimização consiga escapar de ótimos locais. O algoritmo proposto difere do algoritmo PSO original no sentido de que *pbest* e *gbest* não armazenam as melhores soluções obtidas, mas sim, as melhores programações de tarefas geradas pelo algoritmo Giffler e Thompson. Sha e Hsu (2006) mostraram que o algoritmo proposto foi melhor que as heurísticas de Shifting e Bottleneck e AG em um comparativo realizado sobre uma base de testes de problemas TAP.

A otimização do problema de programação de disciplinas em cursos universitários utilizando PSO foi proposta por Shiau (2011). Neste trabalho, o problema foi modelado com algumas restrições flexíveis, de modo que as preferências dos professores e alunos na criação das grades de horários pudessem ser consideradas. O autor mostrou que o PSO gerou um número menor de soluções inválidas para o problema quando comparado com AG, entretanto, um processo de reparação precisou ser incluído. PSO gerou melhores resultados na comparação com o AG para instâncias pequenas e médias do problema. Em grandes instâncias, AG se mostrou mais eficiente.

O trabalho de Rosendo e Pozo (2010) contribuiu com a criação de um meta-modelo baseado no PSO para a otimização de diferentes problemas discretos, como o TSP e o KP. Os autores mantiveram a equação clássica do PSO para cálculo da velocidade, e enfatizaram a utilização de busca local e *path relinking* com o objetivo de evitar a estagnação das soluções em regiões de ótimo local do espaço de busca. Goldbarg et al. (2006) otimizaram o TSP com uma versão do PSO com busca local e *path relinking* que aplica apenas um dos três operadores de movimentação da partícula por iteração, dada uma determinada probabilidade para cada componente. Ainda com o objetivo de evitar a convergência prematura da nuvem de partículas, Tang e Zhao (2010) implementaram busca local adaptativa com a inclusão de operadores de mutação ao algoritmo PSO.

5 OTIMIZAÇÃO DE PROBLEMAS DINÂMICOS

Neste capítulo são apresentadas informações a respeito das principais características presentes em problemas dinâmicos, as técnicas comumente utilizadas para a detecção das mudanças, as estratégias de resposta às mudanças em implementações do PSO, e as métricas que podem ser utilizadas para avaliar o desempenho de um algoritmo que se propõe a otimizar um problema dinâmico.

5.1 PROBLEMAS DINÂMICOS

Em muitos problemas de otimização do mundo real, uma ampla variedade de incertezas precisa ser considerada. Estas incertezas podem ser ocasionadas por mudanças nos objetivos do problema, nas prioridades e/ou na disponibilidade dos recursos. Existem diferentes maneiras em que os sistemas podem mudar ao longo do tempo: pode mudar a posição do valor ótimo no espaço de busca do problema; pode mudar o próprio valor ótimo do problema sem que a respectiva posição tenha sido alterada (EBERHART; SHI, 2001). Exemplificando no contexto do PATC/TAP: no primeiro caso, a alocação ótima $\pi = [1,3,2,4]$ muda, resultando na solução $\pi = [3,1,2,4]$; no segundo caso, a permutação $\pi = [1,3,2,4]$ permanece inalterada (nenhuma posição é trocada), mas pelo fato de o táxi 4 ter se deslocado geograficamente, por exemplo, o custo da solução muda. Em ambos os casos muda o *fitness* ótimo, para melhor ou para pior.

Jin e Branke (2005) categorizaram em quatro classes as incertezas encontradas em problemas de otimização: ruído, aproximação de *fitness*, avaliação de *fitness* com variação temporal, e robustez das soluções obtidas. Estas duas últimas incertezas aparecem principalmente em problemas dinâmicos, e por isso são descritas a seguir:

- Avaliação de *fitness* com variação temporal: esta incerteza ocorre quando a função de *fitness* é dependente do tempo, mesmo sendo determinística em qualquer ponto do tempo (Equação 10).

$$F(\mathbf{x}) = f(\mathbf{x},t) \tag{10}$$

Em problemas deste tipo, o algoritmo aplicado precisa ser capaz de acompanhar o valor ótimo que é modificado continuamente, ao invés de tratar cada "fotografia" do problema como uma nova instância e repetidamente reiniciar o processo de otimização. Isto requer que as informações de cenários anteriores do problema sejam reutilizadas após a ocorrência de mudanças.

- Robustez: a incerteza sobre a robustez das soluções acontece quando as variáveis de decisão do problema estão sujeitas à perturbações ou modificações ocorridas após ter sido determinada a solução ótima. Para que uma solução seja considerada robusta, ela deve ser satisfatória para o problema mesmo quando as variáveis de decisão passam por ligeiras modificações. A Equação 11 considera os distúrbios δ nas variáveis de decisão de uma solução $\mathbf{x}$.

$$F(\mathbf{x}) = f(\mathbf{x}+\delta) \tag{11}$$

Enquanto o ruído está presente nos valores de *fitness*, a incerteza sobre a robustez das soluções está relacionada com as variáveis de decisão do problema. O *fitness* esperado também é uma média de um conjunto de amostras (Equação 12):

$$F'(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}+\delta_i), \tag{12}$$

onde N é o número de amostras, e $F'(\mathbf{x})$ é uma estimativa de $F(\mathbf{x}) = f(\mathbf{x})$.

5.2 DETECÇÃO DE MUDANÇAS

O objetivo da otimização de problemas dinâmicos não é somente encontrar a solução ótima dado um determinado número de iterações do algoritmo, mas sim possibilitar o rastreamento da solução ótima do problema enquanto ocorrem mudanças nas suas variáveis de decisão (WEICKER, 2002). Em problemas reais, pode não existir uma maneira prática de determinar que o problema foi modificado, ou mesmo, os algoritmos podem ser incapazes de detectar as mudanças automaticamente. Portanto, para que o sistema de otimização possa reagir às alterações, primeiro é necessário que seja adotada uma técnica de detecção automática de mudanças.

Hu e Eberhart (2001) propuseram um método chamado "*changed-gbest-value*" para a detecção de mudanças em problemas de otimização. O método consiste em reavaliar a solução *gbest* a cada iteração do algoritmo. Se houver alguma mudança no valor de *fitness* retornado pela função sem que a localização da solução ótima tenha sida alterada, significa que o problema foi alterado. Esta conclusão é decorrente da suposição de que para uma mesma localização de

uma solução no espaço de busca, o valor de *fitness* não muda. Com base nisto, Carlisle e Dozier (2000) desenvolveram um método parecido, que monitora uma posição do espaço de busca escolhida de maneira aleatória para determinar a ocorrência de mudanças com base no valor retornado pela função de *fitness* sobre esta posição.

Em um algoritmo de otimização que não é capaz de acompanhar um sistema dinâmico, geralmente a solução *gbest* fica presa na região ótima anterior à ocorrência de uma mudança. A partir disto, Hu e Eberhart (2002) desenvolveram posteriormente o método "*fixed-gbest-value*", que monitora o valor de *gbest* durante um determinado número de iterações. Se a solução *gbest* não for modificada durante estas iterações, possivelmente a solução ótima do problema mudou. Para aumentar a acuracidade do método, é monitorado ainda um segundo melhor *gbest*. Um número de iterações muito pequeno permite que falsos alarmes sejam emitidos, e um número muito alto faz com o método identifique as mudanças com um certo atraso, logo, este número fixo de iterações é um parâmetro que deve ser ajustado empiricamente. Obviamente que o *gbest* também não muda quando o algoritmo converge para a solução ótima do problema, portanto, isto deve ser considerando quando da utilização do método proposto.

5.3 ESTRATÉGIAS DE RESPOSTAS ÀS MUDANÇAS

Um algoritmo capaz de otimizar problemas dinâmicos precisa fazer com que a nuvem busque pela nova solução ótima sempre que houver mudanças no problema, já que as posições e velocidades das partículas da nuvem presumivelmente teriam convergido para a solução ótima anterior (Figuras 9 e 10). Dependendo da complexidade da mudança, as partículas terão dificuldades para explorar outras regiões e assim identificar a nova solução ótima. Desta forma, é possível afirmar que o algoritmo PSO precisa ser adaptado para a sua aplicação em problemas dinâmicos.

Eberhart e Shi (2001) deram um primeiro passo na utilização do algoritmo PSO para o rastreamento de soluções ótimas em problemas dinâmicos, propondo a reinicialização da memória das partículas como uma forma de reiniciar a busca pela nova região ótima. Segundo o autor, esta abordagem é adequada quando o problema passa por modificações suaves, sendo que diante de alterações maiores o tratamento adequado seria reiniciar completamente a nuvem, de maneira aleatória. E uma terceira abordagem é a combinação das duas primeiras, ou seja, inicializar a nova população com metade dos membros da população anterior, e a outra metade da população sendo posicionada aleatoriamente sobre o espaço de busca do problema. Outras variações podem ser obtidas ao se decidir por preservar ou reinicializar *gbest*.

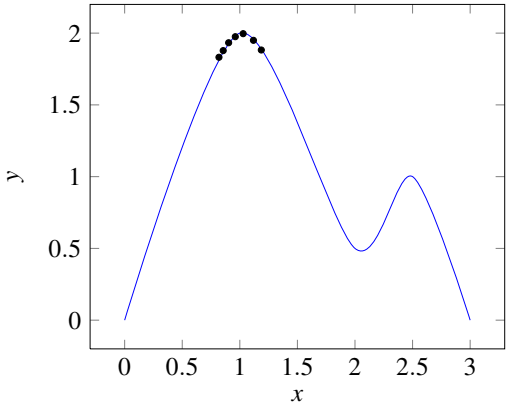

Figura 9: Antes de uma perturbação no problema: o enxame converge para a região da solução ótima no espaço de busca

Fonte: Autoria própria

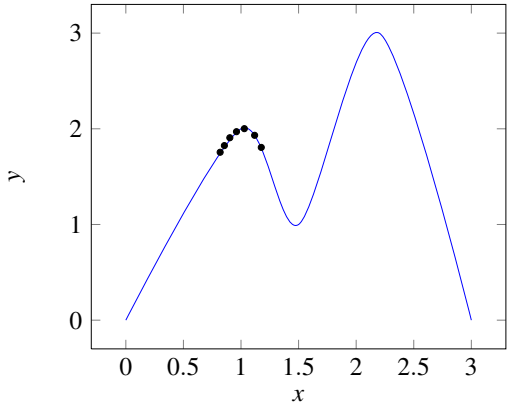

Figura 10: Após uma perturbação no problema: a solução ótima é alterada e a convergência do enxame não acompanha isto

Fonte: Autoria própria

O trabalho de Hu e Eberhart (2002) propõe uma nova estratégia de resposta às mudanças, já que reiniciar a memória das partículas (EBERHART; SHI, 2001) pode não funcionar quando a população inteira tiver convergido para uma pequena área do espaço de busca, tornando impossível para o algoritmo escapar desta área para acompanhar as mudanças. O autor explica que se as mudanças ocorridas fizerem com que o novo ótimo do problema ainda se encontre na pequena área de busca para onde convergiu a nuvem, o PSO irá detectar a nova solução ótima automaticamente, sem nenhuma modificação no algoritmo. Para as demais situações, é utilizado um método de realocação aleatória das partículas após a detecção de mudanças, sendo que entre as estratégias testadas, a melhor delas foi a de diversificar 10% da nuvem.

Esquivel e Coello (2004) introduziram um operador de mutação no algoritmo PSO, com o objetivo de manter a diversidade na nuvem de partículas. As taxas de probabilidade do operador de mutação utilizadas nos experimentos deste trabalho foram altas, o que foi necessário para manter uma diversidade mínima na nuvem. Esta mutação não interrompe o processo de busca graças ao uso de uma memória adicional que armazena as melhores soluções obtidas e não é destruída quando da ocorrência de mudanças. Esta memória só é atualizada quando uma partícula obtém um *fitness* melhor do que aquele armazenado em sua memória.

Yang et al. (2007) propuseram uma versão modificada do PSO, para promover a adaptação dinâmica do algoritmo e assim possibilitar respostas às situações de mudança ocorridas no ambiente de otimização. Neste algoritmo, é utilizada uma fórmula modificada para a atualização da velocidade das partículas, onde a aleatoriedade (componentes $r1$ e $r2$) é relativamente reduzida e a inércia de cada partícula é diferente. Além disso, o algoritmo introduz dois parâmetros que descrevem o estado evolutivo do processo de otimização: o fator velocidade de

evolução e o fator grau de agregação. A inércia é ajustada dinamicamente baseada na análise destes dois fatores. Segundo o autor, o algoritmo melhora a habilidade do PSO em fugir de regiões de ótimo local.

Kamosi et al. (2010) propuseram um algoritmo que utiliza múltiplas nuvens para abordar a manutenção da diversidade das partículas. Este algoritmo utiliza dois tipos de nuvem: uma nuvem "mãe" que explora o espaço de busca para encontrar regiões promissoras; e algumas nuvens "filhas" que realizam busca local nestas áreas encontradas pela nuvem "mãe". O algoritmo trata a sobreposição das nuvens, removendo a pior delas quando isto acontece. Para responder às mudanças, as partículas da nuvem "filha" realizam uma busca local em torno da melhor posição da sua própria nuvem sempre que uma mudança é detectada. Os resultados deste trabalho mostraram que o algoritmo é capaz de acompanhar a solução ótima mesmo após a ocorrência de mudanças no meio. Um trabalho parecido já havia sido proposto por Li e Yang (2008), que compartilha a idéia de utilizar uma nuvem "mãe" e nuvens "filhas".

5.4 AVALIAÇÃO DE DESEMPENHO

Um método de avaliação de desempenho deve possibilitar a comparação de resultados com significância estatística. Para que existam avanços em pesquisa, é necessário que os experimentos sejam repetíveis e que os resultados dos experimentos sejam relatados de maneira a facilitar a comparação de resultados. Em trabalhos de pesquisa sobre otimização de problemas dinâmicos, isto significa que além das extensões e modificações dos algoritmos, é necessário que sejam descritos o problema e o método de medição de desempenho que está sendo empregado. Apesar de existir grande interesse em algoritmos para a otimização de problemas dinâmicos, não há um acordo uniforme sobre o que constitui bom desempenho para estes algoritmos (MORRISON, 2003).

Para comparar diferentes abordagens em um ambiente dinâmico, não é suficiente a comparação da melhor solução encontrada ao final da execução do algoritmo, já que a solução ótima está constantemente sendo substituída por outra solução em decorrência das alterações do ambiente. Por isto, outros indicadores de desempenho tem sido sugeridos: segundo Weicker (2002), um algoritmo candidato a otimização de um problema dinâmico deve ser preciso, estável, e com grande capacidade de reação às mudanças. Estes e outros indicadores frequentemente utilizados em otimização de problemas dinâmicos são detalhados a seguir.

5.4.1 DESEMPENHO *OFFLINE*

A medida de desempenho *offline* é a média do valor de *fitness* das melhores soluções encontradas. Mais precisamente, o desempenho *offline* $F_{offline}$ é definido de seguinte forma:

$$F_{offline} = \frac{1}{T}\sum_{t=1}^{T} F(best_t), \tag{13}$$

onde T é o número de avaliações total, e $F(best_t)$ é o *fitness* da melhor solução encontrada desde a última mudança ocorrida no ambiente de otimização (JIN; BRANKE, 2005). Este indicador não requer o conhecimento prévio da solução ótima global do problema.

5.4.2 MÉDIA DE ERRO *OFFLINE*

Este indicador representa o erro da melhor solução encontrada pelo algoritmo após a última mudança ocorrida no ambiente de otimização, comparada com a solução ótima global do problema após esta mudança (BLACKWELL, 2007). Uma média de erro *offline* igual a zero indica um rastreamento perfeito da solução ótima. O indicador $E_{offline}$ é definido a seguir:

$$E_{offline} = \frac{1}{T}\sum_{t=1}^{T} (F(best_t) - F(best_t^*)), \tag{14}$$

onde T é o número de avaliações total, $F(best_t)$ é o *fitness* da melhor solução encontrada no instante t, e $F(best_t^*)$ é o *fitness* da solução ótima global para o problema no mesmo instante. Esta métrica requer conhecimento prévio da solução ótima para o problema em cada instante do tempo, o que pode inviabilizar a sua utilização em muitos problemas práticos.

5.4.3 PRECISÃO

Weicker (2002) definiu a precisão de um algoritmo de otimização para problemas dinâmicos da seguinte forma:

$$precisão = \frac{1}{T}\sum_{t=1}^{T} \frac{F(best_t) - F(worst_t)}{F(best_t^*) - F(worst_t)}, \tag{15}$$

onde, no momento t, $F(best_t)$ é o *fitness* da melhor solução da população, $F(best_t^*)$ é o *fitness* da solução ótima global, e $F(worst_t)$ é o *fitness* da pior solução encontrada no espaço de busca. Os valores possíveis para este indicador variam entre o intervalo $[0,1]$, sendo que a precisão 1 é a melhor possível. Assim como no caso da média de erro *offline*, o cálculo da precisão requer o conhecimento prévio da solução ótima, bem como da pior solução do espaço de busca.

5.4.4 ESTABILIDADE

Um algoritmo pode ser considerado estável se as mudanças no meio não afetam demasiadamente a precisão da otimização. Weicker (2002) definiu a estabilidade do algoritmo, em um momento t, da seguinte forma:

$$estabilidade = \frac{1}{T}\sum_{t=1}^{T} max\{0, precisão_{t-1} - precisão_t\}, \tag{16}$$

sendo que os possíveis valores deste indicador estão restritos ao intervalo $[0,1]$. Um valor próximo de 0 demonstra alto índice de estabilidade. Não é recomendada a utilização do indicador de estabilidade como o única medida de avaliação, já que ele não representa o nível de precisão do algoritmo, mas sim deve ser utilizado adicionalmente para determinar o quão afetado é o algoritmo em decorrência das mudanças do problema.

5.4.5 REATIVIDADE

Weicker (2002) definiu ainda uma fórmula de cálculo da habilidade do algoritmo para reagir às mudanças. A reatividade $reatividade_{t,\varepsilon}$ é definida em um momento t da seguinte forma:

$$reatividade_{t,\varepsilon} = min\{\{t' - t | t < t' \leq maxgen, t' \in \mathbb{N}, \frac{precisão_{t'}}{precisão_t} \geq (1-\varepsilon)\} \cup \{maxgen - t\}\}, \tag{17}$$

onde ε é a precisão mínima que deve ser atingida após uma mudança no meio, para que seja possível considerar que o algoritmo reagiu, e $maxgen$ é o número máximo de iterações. O indicador avalia quanto tempo leva o algoritmo para atingir um limiar de precisão desejada, e quanto menor for o valor da $reatividade_{t,\varepsilon}$, maior é o poder de reatividade do algoritmo.

Finalizada a revisão bibliográfica e o estado da arte, no próximo capítulo é apresentada a metodologia que será seguida no trabalho.

6 PROPOSTA DE APLICAÇÃO DE PSO DISCRETO AO PROBLEMA PATC/TAP DINÂMICO

Conforme foi apresentado no Capítulo 4, é possível observar que existem diferentes implementações e adaptações do PSO para a otimização de problemas cujo espaço de busca é discreto. Nota-se nestes trabalhos que são duas as abordagens mais comuns para a codificação das partículas:

- Codificação binária (KENNEDY; EBERHART, 1997; SHA; HSU, 2006; LIAO et al., 2007);
- Codificação com sequências de posições (HU et al., 2003; GOLDBARG et al., 2006; LIU et al., 2008; SHA; HSU, 2008; ROSENDO; POZO, 2010; LIU et al., 2011).

Nesta dissertação, dois destes trabalhos foram selecionados para servirem de base na implementação de dois algoritmos discretos baseados no PSO. As duas implementações seguem o algoritmo original do PSO (Algoritmo 1), sendo que a diferença entre elas está na estratégia de codificação das partículas. Neste trabalho, é realizado um comparativo conceitual e da qualidade das soluções obtidas com cada um dos algoritmos nos experimentos realizados. O objetivo da realização deste comparativo é auxiliar na escolha de uma abordagem eficiente que possa ser empregada na otimização do problema formulado.

6.1 PSO COM CODIFICAÇÃO BINÁRIA

A primeira versão do PSO abordada neste trabalho é baseada no algoritmo proposto por Kennedy e Eberhart (1997). Historicamente, o trabalho citado descreve a primeira aplicação do PSO em problemas de OC. O algoritmo utiliza codificação binária para a representação das partículas da nuvem no espaço de busca discreto, e será chamado PSO com Codificação Binária (PSO-B) neste trabalho.

Considerando que o espaço de busca do problema é discreto, a partícula foi definida como uma matriz bi-dimensional de valores binários, como mostra a Tabela 1. Uma das di-

mensões da matriz representa os agentes de oferta de serviço e a outra dimensão representa os agentes de demanda de serviço.

Tabela 1: Representação da partícula binária (4x4)

	Agente de Demanda ($j = A(i)$)			
Agente de Oferta (i)	1	2	3	4
1	0	0	1	0
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1

A Equação 18 define X_k^t como uma partícula composta por n^2 bits, a qual é considerada uma potencial solução para o problema na iteração t, onde k identifica a partícula na nuvem. Quando $x_{k,ij} = 1$, o *i-ésimo* agente de oferta será alocado ao *j-ésimo* agente de demanda. Caso contrário, $x_{k,ij} = 0$.

$$X_k^t = \begin{bmatrix} x_{k,11}^t & x_{k,12}^t & \cdots & x_{k,1n}^t \\ x_{k,21}^t & x_{k,22}^t & \cdots & x_{k,2n}^t \\ \cdots & & & \\ x_{k,n1}^t & x_{k,n2}^t & \cdots & x_{k,nn}^t \end{bmatrix} \tag{18}$$

Considerando a aplicação do algoritmo ao PATC/TAP, e conforme definido pela Equação 2, o *fitness* f de uma partícula X_k^t é calculado somando-se o custo do caminho mínimo entre os pares (táxi, cliente) alocados. A Equação 19 ajusta o cálculo do *fitness* para a codificação binária.

$$f(X_k^t) = 1/\sum_i \ distância(i, A(i)) \tag{19}$$

A velocidade da partícula, calculada com a Equação 7, é representada na forma de probabilidades (Tabela 2) de uma posição (ij) da matriz assumir o valor 1 nas próximas iterações. Quanto maior for a velocidade, maior será a probabilidade de que seja atribuído o valor 1 a variável binária. É importante destacar que os números aleatórios $r1$ e $r2$ podem ser diferentes para cada posição (ij) em $\mathbf{pbest}_k$ e **gbest**, respectivamente.

Tabela 2: Velocidade da partícula binária (4x4)

	Agente de Demanda ($j = A(i)$)			
Agente de Oferta (i)	1	2	3	4
1	0.10	0.20	0.50	0.73
2	0.99	0.13	0.67	0.41
3	0.72	0.99	0.12	0.55
4	0.13	0.02	0.83	0.17

Conforme proposto por Kennedy e Eberhart (1997), a Equação 20 é utilizada para normalizar a velocidade da partícula, mantendo-a limitada no intervalo $[0,1]$.

$$N(\mathbf{v}_k^t) = \frac{1}{1+\exp(-\mathbf{v}_k^t)} \tag{20}$$

Então, a posição da partícula é atualizada pela Equação 21, adicionando-se a velocidade normalizada a posição ocupada pela partícula na iteração anterior.

$$\mathbf{x}_k^t = \mathbf{x}_k^{t-1} + N(\mathbf{v}_k^t) \tag{21}$$

Neste trabalho, cada partícula constrói a sua matriz de alocação com base em suas probabilidades de troca de posições. A cada iteração do algoritmo, uma partícula $\mathbf{x}_k^t$ inicia com uma matriz preenchida com o valor 0 em todas as posições, e atribui o valor 1 aplicando as probabilidades às posições escolhidas em sorteio. Quanto maior for a velocidade relacionada a uma posição (i,j), maior é a probabilidade de a troca ocorrer em $x_{k,ij}^t$. As posições da matriz são escolhidas aleatoriamente para a realização do sorteio; e no caso de nenhuma posição de uma linha ou coluna ter assumido o valor 1, o sorteio recomeça nesta linha/coluna.

A matriz binária deve possuir apenas um valor 1 por linha e por coluna devido a uma restrição imposta pela codificação binária, logo, um cuidado especial é requerido: as linhas e colunas devem ser candidatas ao sorteio enquanto não possuam um valor 1. Isto é necessário para impedir a criação de soluções infactíveis para o problema, e por isso, um método inspirado em busca tabu é utilizado para armazenar os "estados tabu", ou seja, o método utiliza uma lista tabu que armazena as linhas e colunas que já foram "visitadas" em sorteios anteriores (posições que tiveram o seu valor alterado de 0 para 1). A construção da partícula é completada quando todas as linhas e colunas da matriz possuem uma (e somente uma) posição com valor 1. O processo é executado até que um determinado número de iterações seja alcançado.

6.2 PSO COM CODIFICAÇÃO DE SEQUÊNCIAS DE POSIÇÕES

A segunda versão do PSO discreto abordada neste trabalho utiliza representação das partículas como sequências de posições, e foi baseada na proposta de Hu et al. (2003). Nesta implementação do PSO com Codificação de Sequências de Posições (PSO-P), a posição da partícula é representada por um vetor $\mathbf{x}_k^t = (x_1, x_2, \ldots x_n)$ de números inteiros cujos índices identificam os agentes de oferta de serviço e os valores representam os agentes de demanda de serviço.

Uma partícula cuja posição i do vetor de posições $\mathbf{x}_k$ armazena o valor j, indica que o *i-ésimo* agente de oferta será alocado para o atendimento do *j-ésimo* agente de demanda. No exemplo da Tabela 3, o agente de oferta 1 seria alocado para o agente de demanda 4, o agente de oferta 2 seria alocado para o agente de demanda 3, e assim por diante.

Tabela 3: Representação da partícula como sequências de posições

Agente de Oferta (i)	1	2	3	4	5	...
Agente de Demanda ($j = x_{k,i}$)	4	3	7	9	2	...

Assim como no PSO-B, o *fitness* f da partícula $\mathbf{x}_k^t$ é calculado somando-se o custo do caminho mínimo entre os agentes de oferta e demanda (Equação 22).

$$f(\mathbf{x}_k^t) = 1/\sum_i distância(i, x_{k,i}^t) \tag{22}$$

A velocidade das partículas também é calculada com a Equação 7, e assim como no PSO-B, os números aleatórios $r1$ e $r2$ são diferentes para cada posição i dos vetores de alocação de $\mathbf{pbest}_k$ e **gbest**, respectivamente. O vetor de velocidade é então normalizado, dividindo-se os valores de todas as posições pelo maior valor deste vetor; assim, as velocidades das partículas são mantidas entre o intervalo $[0,1]$. A movimentação das partículas ocorre com a troca de posições dos valores do vetor $\mathbf{x}_k^t$, considerando que a velocidade indica a probabilidade de que a operação de "swap" ocorra em cada posição de $\mathbf{x}_k^t$. O sorteio ocorre de maneira independente em cada posição do vetor, e se for definido que a operação de troca deve ocorrer em uma determinada posição i de $\mathbf{x}_k^t$, esta posição será trocada com o valor da posição que armazena o valor correspondente em **gbest**. O processo é ilustrado na Figura 11.

						▼	
$\mathbf{v}_k^t$	...	0.15	0.22	0.63	0.94	0.51	...
						▼	
gbest	...	7	2	3	1	8	...
		▼				▼	
$\mathbf{x}_k^t$	...	8	1	3	4	2	...
$\mathbf{x}_k^{t+1} = \mathbf{x}_k^t + \mathbf{v}_k^t$	...	2	1	3	4	8	...

Figura 11: Movimentação da partícula codificada como sequências de posições

Conforme ilustrado na Figura 11, o componente **pbest** não é utilizado na movimentação da partícula, sendo utilizado apenas no cálculo da velocidade.

Diferentemente do PSO-B, no PSO-P não existe um processo de prevenção ou reparação de soluções infactíveis para o problema. Por si só, a movimentação das partículas com permutações de posições previne que um agente de oferta seja alocado para mais de um agente de demanda, ou o inverso.

Da mesma forma, o critério de parada do algoritmo é a execução de um número T de iterações.

6.3 AJUSTE DA INÉRCIA NOS ALGORITMOS PSO-B E PSO-P

O parâmetro V_{MAX} não é utilizado nos algoritmos PSO-B e PSO-P. Para compensar a sua ausência, o fator de inércia diminui gradativamente a cada iteração t do algoritmo (SHI; EBERHART, 1998). O decaimento da inércia é definido pela Equação 23:

$$w = \frac{(T-t) * (w_{final} - w_{inicial})}{T} + w_{inicial}, \quad t = 1,2,...,T, \tag{23}$$

onde w é o fator de inércia utilizado pelo algoritmo em cada iteração, T é o número máximo de iterações do algoritmo, e $w_{inicial}$ e w_{final} são parâmetros que definem os limites inicial e final de decaimento da inércia, respectivamente.

A utilização desta equação de ajuste se justifica pelo fato de que um peso de inércia maior no início da busca facilita a localização de boas regiões, as quais podem ser melhor aproveitadas com peso de inércia menor.

6.4 SOFTWARE PARA SIMULAÇÃO DO PATC/TAP

Para possibilitar a criação de instâncias do PATC/TAP e simular o problema em ambiente dinâmico, foi desenvolvido um software que dispõe agentes de oferta e demanda em pontos geográficos gerados aleatoriamente dentro de uma área central da Cidade de Curitiba/PR. As ilustrações de mapas e instâncias do problema apresentadas neste trabalho foram capturadas a partir deste software, que foi construído com o objetivo de viabilizar a realização dos experimentos desta dissertação, com a aplicação dos algoritmos implementados e a coleta dos resultados para a realização de comparativos.

Duas soluções válidas para uma instância do problema com tamanho $N = 13$ são apresentadas nas Listagens 6.1 e 6.2, conforme o log extraído do ambiente de simulação.

Listagem 6.1: Log do ambiente de simulação - partícula com sequência de posições

```
02:12:33 [PSO-P] Otimização por Nuvem de Partículas - Sequências de Posições
02:12:33 [PSO-P] Swarm inicializado com 100 partículas.
02:12:33 [PSO-P] Executar 100 iterações.
02:12:33 [PSO-P] Melhor fitness (gBest): 0,02181
            Melhor alocação encontrada:
            |9|6|10|3|4|13|11|7|12|5|8|2|1|
02:12:33 [PSO-P] Tempo de execução: 7288 milissegundos.
```

Listagem 6.2: Log do ambiente de simulação - partícula binária

```
22:12:52 [PSO-B] Otimização por Nuvem de Partículas - Codificação Binária
22:12:52 [PSO-B] Swarm inicializado com 100 partículas.
22:12:52 [PSO-B] Executar 100 iterações.
22:12:55 [PSO-B] Melhor fitness (gBest): 0,02003
            Melhor alocação encontrada:
            |0|0|0|0|0|0|0|0|1|0|0|0|0|
            |0|0|0|0|0|0|0|0|0|0|0|1|0|
            |0|0|1|0|0|0|0|0|0|0|0|0|0|
            |0|0|0|0|0|1|0|0|0|0|0|0|0|
            |1|0|0|0|0|0|0|0|0|0|0|0|0|
            |0|0|0|0|0|0|0|0|0|0|1|0|0|
            |0|0|0|0|0|0|0|0|0|0|0|0|1|
            |0|0|0|0|0|0|0|1|0|0|0|0|0|
            |0|1|0|0|0|0|0|0|0|0|0|0|0|
            |0|0|0|0|1|0|0|0|0|0|0|0|0|
            |0|0|0|1|0|0|0|0|0|0|0|0|0|
            |0|0|0|0|0|0|1|0|0|0|0|0|0|
            |0|0|0|0|0|0|0|0|0|1|0|0|0|
22:12:55 [PSO-B] Tempo de execução: 8538 milissegundos.
```

O mapa utilizado neste trabalho é o Open Street Map (OSM, 2011), cujos dados topológicos foram obtidos e transformados para uma estrutura de dados que viabiliza as operações de roteamento e identificação de caminho mínimo entre diferentes locais da cidade. Mais informações sobre o OSM são descritas no Anexo A. A conversão dos dados do mapa e a aplicação do algoritmo de roteamento se deu com o auxílio do software OSM2PO (Anexo B).

Adicionalmente, foi desenvolvido um software para ser utilizado como um dispositivo de rastreamento e propiciar um ambiente de otimização com táxis e clientes reais. Este software pode ser instalado em telefones celulares com algum dispositivo GPS conectado para coletar e enviar as informações de geoposicionamento através de *webservices* para o *website* que publica o mapa e aplica o algoritmo de otimização.

6.5 ALGORITMO DPSO-P PROPOSTO PARA O PATC/TAP DINÂMICO

Nesta seção, é descrita a proposta do algoritmo PSO Dinâmico com Codificação de Sequências de Posições (DPSO-P), o qual é baseado no PSO-P, para a otimização do problema PATC/TAP dinâmico. O algoritmo DPSO-P utiliza o mesmo esquema de codificação de partículas do PSO-P, e trata a dinâmica do problema com técnicas voltadas para a detecção e a resposta às mudanças do ambiente dinâmico (Algoritmo 2).

A condição de parada do algoritmo PSO-P precisou ser substituída, considerando que o objetivo da otimização de um problema dinâmico não é apenas encontrar a solução ótima dado um determinado número de iterações, mas sim acompanhar a solução ótima enquanto as variáveis de decisão do problema passam por modificações. No algoritmo DPSO-P, a condição de parada é o tempo total de simulação do problema.

O método utilizado para detectar as mudanças foi aquele sugerido por Carlisle e Dozier (2000), onde é escolhida uma posição aleatória do espaço de busca e a cada iteração do algoritmo é verificado se houve mudança no valor retornado pela função de *fitness* sobre esta posição. Constatou-se empiricamente que este método é capaz de detectar todas as mudanças modeladas para o problema em questão, mesmo aquelas consideradas suaves.

As abordagens utilizadas para prover resposta às mudanças são :

- Abordagem *A*: reiniciar aleatoriamente uma porção de partículas da nuvem, sempre que for detectada uma mudança (EBERHART; SHI, 2001; HU; EBERHART, 2002). As partículas escolhidas para serem reiniciadas são aquelas que apresentam o pior valor de *fitness*. Foram testadas diferentes taxas de reinicialização da nuvem, as quais são descritas no proximo capítulo;

- Abordagem *B*: reiniciar a memória *pbest* de todas as partículas da nuvem, sempre que for detectada uma mudança (EBERHART; SHI, 2001);

- Abordagem *C*: utilizar um operador de perturbação sobre a posição das partículas, com o objetivo de manter diversidade na nuvem (ESQUIVEL; COELLO, 2004). A cada iteração do algoritmo, existe uma chance de cada partícula sofrer uma perturbação. Foram testadas diferentes taxas de perturbação, as quais também são descritas no próximo capítulo.

Algoritmo 2 Pseudo-código do algoritmo DPSO-P para um problema de maximização

1: **for** $k = 1$ to $NPartículas$ **do**
2: $\mathbf{x}_k^1 \leftarrow$ uma solução aleatória
3: $\mathbf{v}_k^1 \leftarrow$ uma velocidade aleatória
4: $\mathbf{pbest}_k \leftarrow \mathbf{x}_k^1$
5: **end for**
6: **gbest** $\leftarrow$ a melhor solução $\in [\mathbf{x}_1, \mathbf{x}_{NPartículas}]$
7: **particulaMonitorada** $\leftarrow$ uma solução aleatória $\notin [\mathbf{x}_1, \mathbf{x}_{NPartículas}]$
8: $fitnessMonitorado \leftarrow fitness($**particulaMonitorada**$)$
9: **while** não atingir tempo de simulação **do**
10: **if** $fitness($**particulaMonitorada**$) <> fitnessMonitorado$ **then**
11: $fitnessMonitorado \leftarrow fitness($**particulaMonitorada**$)$
12: $t = 1$
13: **if** abordagem A **then**
14: $reiniciarPioresParticulas(\mathbf{x}^t)$
15: **end if**
16: **if** abordagem B **then**
17: $reiniciarMemoriaParticulas($**pbest**$_k)$
18: **end if**
19: **end if**
20: **for** $k = 1$ to $NPartículas$ **do**
21: **if** $fitness(\mathbf{x}_k^t) > fitness($**pbest**$_k)$ **then**
22: **pbest**$_k \leftarrow \mathbf{x}_k^t$
23: **end if**
24: **if** $fitness(\mathbf{x}_k^t) > fitness($**gbest**$)$ **then**
25: **gbest** $\leftarrow \mathbf{x}_k^t$
26: **end if**
27: $w = \frac{(T-t)*(w_{final}-w_{inicial})}{T} + w_{inicial}$
28: $\mathbf{v}_k^t = w\mathbf{v}_k^{t-1} + c_1 r_1(\mathbf{pbest}_k - \mathbf{x}_k^{t-1}) + c_2 r_2(\mathbf{gbest} - \mathbf{x}_k^{t-1})$
29: $\mathbf{x}_k^t \leftarrow \mathbf{x}_k^{t-1} + N(\mathbf{v}_k^t)$
30: **if** abordagem C **then**
31: $\mathbf{x}_k^t \leftarrow perturbação(\mathbf{x}_k^t)$
32: **end if**
33: **end for**
34: $t = t + 1$
35: **end while**

6.5.1 AJUSTE DA INÉRCIA NO ALGORITMO DPSO-P

No algoritmo DPSO-P a inércia também sofre um decaimento, seguindo o que é definido pela Equação 23. Entretanto, a diferença do caso estático para o caso dinâmico está na definição de T: se no caso estático T representa o número máximo de iterações do algoritmo, no caso dinâmico T representa o número de iterações do algoritmo até que seja detectada uma mudança, fazendo com que o decaimento da inércia seja formado por janelas temporais. O número médio de iterações do algoritmo entre a ocorrência de mudanças foi definido empiricamente.

No próximo capítulo são descritos os experimentos realizados neste trabalho para comparar o desempenho das implementações do PSO discreto na resolução do PATC/TAP. Além disso, são comparadas as abordagens que adicionam ao PSO a habilidade de lidar com cenários dinâmicos de otimização. Algumas estratégias de resposta às mudanças são testadas simultaneamente, com o objetivo de definir empiricamente uma combinação de abordagens que apresente os melhores resultados, de acordo com o tamanho das instâncias do problema e com a escala de mudanças.

7 EXPERIMENTOS E RESULTADOS

O primeiro experimento realizado neste trabalho consiste na comparação das duas implementações do PSO discreto que foram apresentadas no capítulo anterior. Ambos algoritmos são empregados na resolução do problema de otimização PATC/TAP estático, sobre determinadas instâncias do problema com diferentes tamanhos. Adicionalmente, a busca exaustiva é empregada para identificar as soluções ótimas destas instâncias do problema, e assim possibilitar uma avaliação de desempenho dos algoritmos.

Em seguida, o melhor dos algoritmos identificados no primeiro experimento é empregado na resolução do problema de otimização PATC/TAP dinâmico, com a adição e a combinação de algumas abordagens de resposta às mudanças em problemas dinâmicos.

A título de referência, o programa utilizado nos experimentos foi codificado em linguagem Java e executado em um computador com processador Intel Core 2 Duo 2.40GHz e 3GB de memória RAM.

7.1 PARÂMETROS DO ALGORITMO PSO

Os mesmos parâmetros de algoritmo foram utilizados nos experimentos do PSO-B (codificação binária) e PSO-P (codificação com permutações). Alguns destes parâmetros foram definidos empiricamente, e outros foram escolhidos com base em estudos específicos. Por exemplo, os fatores cognitivo $c1$ e social $c2$ foram definidos com o valor 1.49445, conforme proposto por Eberhart e Shi (2000). Este valor foi obtido pelo autor como resultado da multiplicação de 0.729 (um valor de inércia que apresentou bons resultados) por 2.05 (aproximadamente o dobro da inércia). Esta abordagem tem sido amplamente utilizada em trabalhos recentes de aplicação do PSO em problemas combinatórios.

De modo a possibilitar futuros comparativos, estes parâmetros são descritos na Tabela 4.

Tabela 4: Parâmetros do algoritmo PSO utilizados nos experimentos

Parâmetro	Valor (Problema Estático)	Valor (Problema Dinâmico)	Justificativa
$NPartículas$	100	100	Definido empiricamente
T	100	Proporcional ao tempo de simulação	Definido empiricamente
V_{MIN}	Não utilizado	Não utilizado	(SHI; EBERHART, 1998)
V_{MAX}	Não utilizado	Não utilizado	(SHI; EBERHART, 1998)
w	Equação 23 T = número máximo de iterações	Equação 23 T = número de janelas temporais	(SHI; EBERHART, 1998)
$w_{inicial}$	0.9	0.9	(EBERHART; SHI, 2000)
w_{final}	0.4	0.4	(EBERHART; SHI, 2000)
$c1$	1.49445	1.49445	(EBERHART; SHI, 2000)
$c2$	1.49445	1.49445	(EBERHART; SHI, 2000)
N	{10, 11, 12, 13, ... , 20, 25, 30, ... , 100}	{10, 100}	(EBERHART; SHI, 2000)
Abordagem A	Não se aplica	{10%, 50%, 100%}	Valores extremos
Abordagem B	Não se aplica	{Sim, Não}	Valores extremos
Abordagem C	Não se aplica	{10%, 50%, 100%}	Valores extremos

7.2 BUSCA EXAUSTIVA

Um processo de Busca Exaustiva (BE) também foi implementado, com o objetivo de identificar a solução ótima de algumas instâncias do problema e assim permitir uma avaliação qualitativa das soluções obtidas pelas implementações do PSO. A BE foi implementada utilizando um algoritmo de *backtracking*, que obtém todas as possíveis soluções para o problema, e as avalia com a mesma função "*distância*" utilizada nos algoritmos PSO para medir o *fitness* das soluções.

O fato da BE avaliar todas as soluções viáveis do problema faz com que o seu tempo de processamento seja relativamente alto, principalmente se comparado com o tempo de processamento dos algoritmos de busca heurística. Desta forma, apesar de ser possível a avaliação da viabilidade computacional da BE por meio do cálculo do tempo de processamento efetivo, o objetivo principal é obter as soluções ótimas das instâncias do problema em que isto é possível, para que possam ser utilizadas como uma abordagem de comparação.

A Figura 12 ilustra a solução ótima encontrada pela Busca Exaustiva para uma determinada instância do problema com tamanho $N = 13$. As linhas desenhadas sobre o mapa conectam os agentes de oferta aos agentes de demanda, simbolizando o caminho mínimo das alocações. As cores das linhas não tem significado especial.

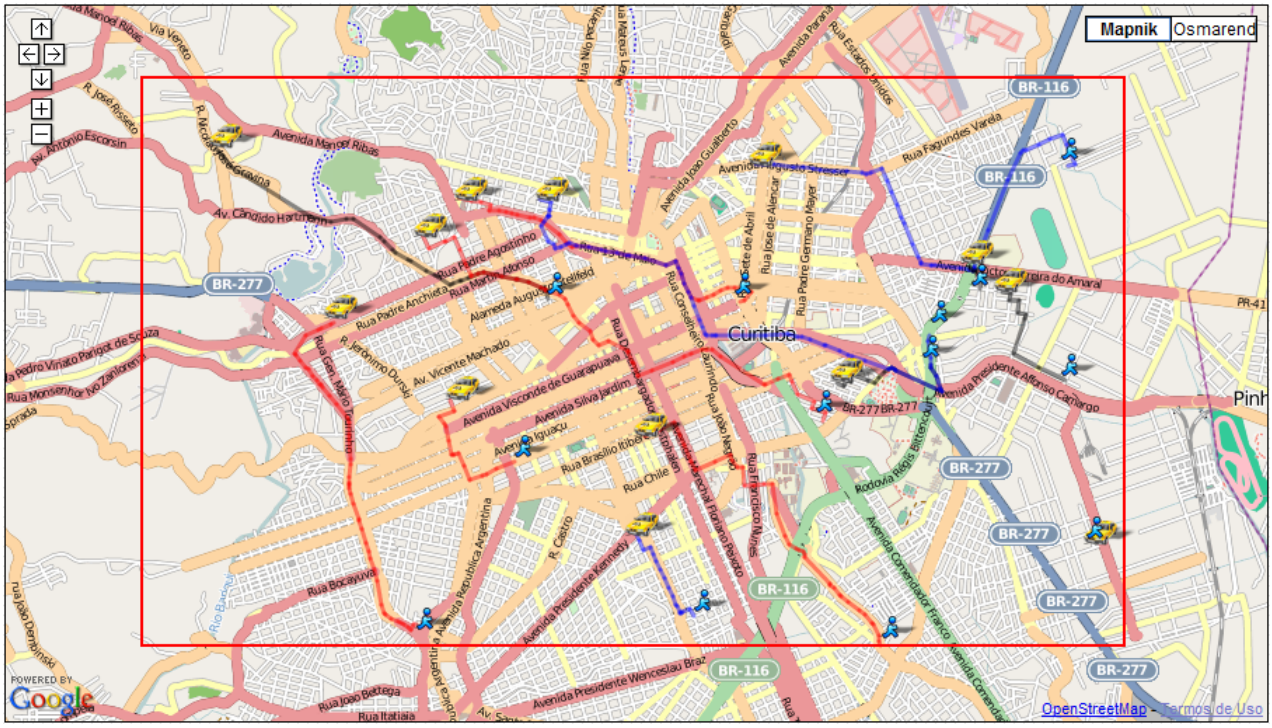

Figura 12: Solução ótima obtida com a busca exaustiva para uma instância com valor de $N = 13$

Fonte: Autoria própria com base em Google e Open Street Map

7.3 PSO ROBUSTO

No caso das instâncias do problema em que não é possível empregar a BE (devido ao seu alto custo computacional), foi utilizado um PSO robusto para identificar uma solução de referência de cada instância testada. Esta solução de referência pode ser utilizada na avaliação da qualidade das soluções obtidas com os algoritmos testados neste trabalho, quando executados sobre estas mesmas instâncias.

O PSO robusto se diferencia do PSO tradicional por ter uma nuvem de partículas maior do que normalmente é utilizado, além de executar durante um elevado número de iterações. O aumento no tamanho da nuvem indica que existem mais partículas procurando por soluções no espaço de busca, e assim a qualidade das soluções tende a melhorar conforme o tamanho da nuvem aumenta. Isto implica, porém, em um tempo de execução mais elevado devido ao maior número de avaliações necessárias na função de *fitness*. Desta forma, o PSO robusto é capaz de obter soluções melhores do que o PSO tradicional, embora apresente tempos de execução mais elevados.

Neste trabalho, o PSO robusto segue o mesmo esquema de codificação de partículas do algoritmo PSO-P, contudo, é executado durante 10.000 iterações e possui uma nuvem composta

por 10.000 partículas (números que representam 100 vezes os respectivos valores de parâmetros utilizados na execução do PSO-P), o que possibilita uma melhor exploração do espaço de busca do problema.

7.4 COMPARATIVO DAS ABORDAGENS DE CÁLCULO DA DISTÂNCIA ENTRE AGENTES DE OFERTA E DEMANDA

Nesta seção é realizado um breve comparativo das duas abordagens testadas no cálculo da distância entre os agentes de oferta e demanda. Foram avaliadas a distância do caminho mínimo identificado com o algoritmo de Dijkstra, e a distância euclidiana entre os pontos geográficos. A Tabela 5 mostra o *fitness* das melhores soluções obtidas, bem como o tempo de execução do algoritmo PSO-P e da busca exaustiva. Foram testada três instâncias do problema: uma instância pequena, uma média, e uma grande.

Tabela 5: Comparativo das abordagens de cálculo da distância entre agentes de oferta e demanda

N	PSO-P (Dijkstra)		PSO-P (Euclidiana)		BE (Dijkstra)		BE (Euclidiana)	
	Fitness	Tempo	Fitness	Tempo	Fitness	Tempo	Fitness	Tempo
10	0,03003	11,510	0,04184	7,205	0,03003	11,873	0,04184	6,626
50	0,00840	174,782	0,01240	35,570	-	-	-	-
100	0,00346	634,270	0,00488	80,022	-	-	-	-

Nota-se que o *fitness* obtido utilizando a distância euclidiana é superior (melhor) ao *fitness* obtido com o caminho mínimo de Dijkstra. Isto acontece porque calcular a distância geográfica entre dois pontos utilizando a primeira abordagem sempre resultará em um valor menor do que no caso da segunda abordagem, sobre os mesmos dois pontos, já que a abordagem da distância euclidiana é mais otimista.

Apesar de o tempo de execução do algoritmo e da busca exaustiva serem maiores utilizando o algoritmo de Dijkstra, esta abordagem foi escolhida para ser empregada neste trabalho por se mostrar mais aproximada do problema prático, já que considera as ruas no cálculo da distância, possibilitando que o sistema oferte a rota ao motorista.

7.5 RESOLUÇÃO DO PATC/TAP EM AMBIENTE ESTÁTICO

As duas implementações do PSO discreto foram testadas, em um ambiente de simulação que engloba uma série de táxis e clientes distribuídos aleatoriamente no mapa da Cidade de Curitiba/PR. Os algoritmos foram executados 30 vezes, conforme proposto por Shi e Eberhart (1998), sobre as mesmas instâncias do problema - com os mesmos agentes de oferta e demanda,

posicionados nas mesmas localizações geográficas - e em cada rodada uma semente diferente foi usada para a geração da população inicial.

Neste experimento, a busca exaustiva é executada sobre as instâncias do problema onde $10 \leq N \leq 13$, pois os tempos de execução são excessivamente altos para instâncias do problema com tamanhos maiores. No caso das instâncias onde $N > 13$, foi empregado o PSO robusto para ser utilizado como abordagem de comparação.

Os resultados obtidos neste experimento são apresentados na próxima seção.

7.5.1 RESULTADOS

A Tabela 6 mostra os resultados obtidos nos experimentos realizados com o PSO-B e PSO-P, além da solução ótima para o PATC/TAP obtida com a BE e a solução de referência obtida com o PSO robusto. A tabela apresenta o *fitness* da melhor solução encontrada dentre as 30 execuções, e a média do *fitness* obtido nas 30 execuções.

Tabela 6: Resultados da otimização do PATC/TAP em ambiente estático

N	PSO-B		PSO-P		BE	PSO Robusto
	Melhor Fitness	Fitness Médio	Melhor Fitness	Fitness Médio	Ótimo	Referência
10	0,02643	0,02574	**0,03003**	**0,02937**	**0,03003**	-
11	0,02259	0,02232	**0,02314**	**0,02308**	**0,02314**	-
12	0,02079	0,02038	**0,02215**	**0,02212**	**0,02215**	-
13	0,02003	0,01847	**0,02181**	**0,02152**	**0,02181**	-
15	0,02814	0,02513	**0,04069**	**0,03921**	-	**0,04069**
20	0,01842	0,01615	**0,02258**	**0,02190**	-	**0,02284**
25	0,01236	0,01214	**0,02221**	**0,02061**	-	**0,02430**
30	0,00938	0,00914	**0,01421**	**0,01383**	-	**0,01584**
35	0,00794	0,00779	**0,01708**	**0,01587**	-	**0,02059**
40	0,00548	0,00532	**0,00920**	**0,00858**	-	**0,00975**
45	0,00364	0,00343	**0,00560**	**0,00520**	-	**0,00600**
50	0,00442	0,00430	**0,00765**	**0,00724**	-	**0,00949**
55	0,00349	0,00337	**0,00519**	**0,00498**	-	**0,00591**
60	0,00388	0,00373	**0,00580**	**0,00551**	-	**0,00653**
65	0,00282	0,00270	**0,00442**	**0,00416**	-	**0,00521**
70	0,00357	0,00324	**0,00678**	**0,00603**	-	**0,00758**
75	0,00279	0,00273	**0,00511**	**0,00456**	-	**0,00709**
80	0,00262	0,00255	**0,00451**	**0,00426**	-	**0,00524**
85	0,00251	0,00243	**0,00474**	**0,00444**	-	**0,00646**
90	0,00216	0,00205	**0,00337**	**0,00322**	-	**0,00432**
95	0,00228	0,00223	**0,00438**	**0,00395**	-	**0,00509**
100	0,00222	0,00216	**0,00431**	**0,00398**	-	**0,00525**

Obviamente, a busca exaustiva encontra a solução ótima para o problema já que ela avalia todas as soluções viáveis. Entretanto, a sua aplicabilidade se torna inviável conforme o tamanho do problema aumenta, devido ao tempo de processamento; por este motivo, os experi-

mentos com BE só puderam ser realizados nos casos onde $N \leq 13$.

Os resultados apresentados mostram que o algoritmo PSO-P otimizou o problema em todas as instâncias para as quais o valor ótimo é conhecido, e encontrou boas soluções diante de instâncias maiores (comparação com as soluções de referência). O *fitness* médio das soluções obtidas nas 30 execuções se manteve próximo da melhor solução para todos os valores de N experimentados neste trabalho. Já o algoritmo PSO-B apresentou soluções relativamente boas nas instâncias do problema com $N \leq 13$. Mas em instâncias maiores, a melhor solução de PSO-B se distancia da melhor solução obtida com o algoritmo PSO-P.

Alguns comparativos também são ilustrados na forma de gráficos. A Figura 13 mostra um comparativo do *fitness* obtido em cada algoritmo com o aumento do tamanho do problema, e a solução ótima ou de referência do problema.

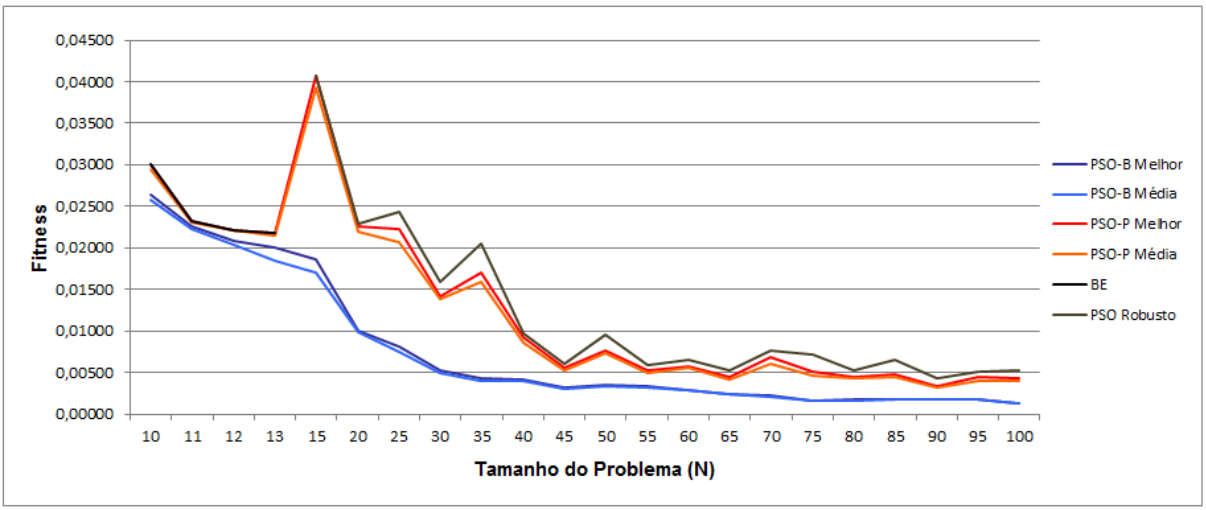

Figura 13: Evolução do *fitness* de acordo com o tamanho do problema

Fonte: Autoria própria

Adicionalmente, o gráfico ilustrado na Figura 14 mostra o *fitness* normalizado de acordo com o tamanho do problema.

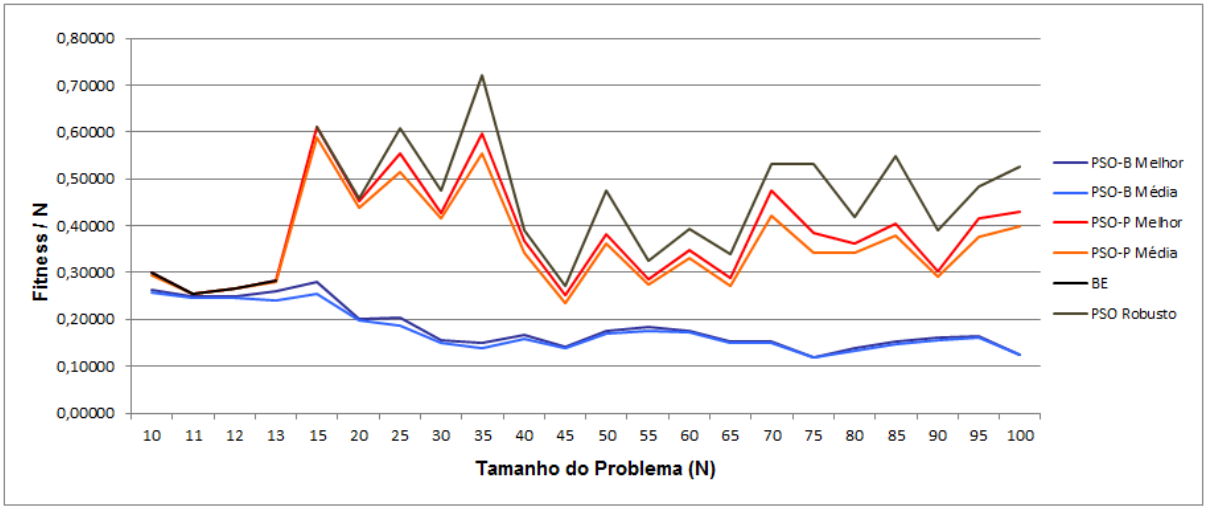

Figura 14: Evolução do *fitness* normalizado de acordo com o tamanho do problema

Fonte: Autoria própria

A Tabela 7 mostra a média do tempo total "*Total (TT)*" de execução de cada algoritmo nas 30 execuções, a média do tempo de execução da função distância "*F. Distância (TD)*" utilizada na avaliação de *fitness* das soluções, e a diferença entre "*TT*" e "*TD*". O tempo computacional da BE foi obtido em uma única execução, uma vez que a busca é determinística. O tempo computacional do PSO robusto também foi obtido em uma única execução, visto que o objetivo maior do experimento é apenas obter as soluções de referência que são utilizadas como abordagem de comparação em instâncias maiores do problema.

Tabela 7: Tempo de execução dos algoritmos PSO-B, PSO-P, BE e PSO Robusto, em segundos

N	Tempo de Execução do PSO-B (Média)			Tempo de Execução do PSO-P (Média)			BE	PSO
	Total (TT)	F. Distância (TD)	TT - TD	Total (TT)	F. Distância (TD)	TT - TD		Robusto
10	6,567	6,478	0,089	**5,506**	5,464	0,042	10,894	-
11	6,773	5,826	0,947	**5,867**	5,825	0,042	71,308	-
12	8,180	6,827	1,353	**6,542**	6,494	0,048	932,185	-
13	8,538	7,238	1,300	**7,288**	7,242	0,046	7.879,010	-
15	11,242	9,481	1,761	**9,560**	9,497	0,063	-	456,847
20	22,690	19,357	3,333	**18,813**	18,736	0,077	-	606,794
25	34,424	29,872	4,552	**29,268**	29,178	0,090	-	705,098
30	49,562	43,128	6,434	**42,976**	42,855	0,121	-	854,067
35	64,321	55,675	8,646	**56,269**	56,137	0,132	-	1.018,043
40	87,076	75,887	11,189	**75,701**	75,566	0,135	-	1.172,091
45	109,197	95,269	13,928	**96,088**	95,922	0,166	-	1.273,447
50	136,223	119,223	17,000	**118,741**	118,559	0,182	-	1.421,351
55	159,793	141,090	18,703	**140,149**	139,968	0,181	-	1.502,387
60	171,481	150,269	21,212	**155,532**	155,319	0,213	-	1.766,560
65	200,345	175,527	24,818	**173,899**	173,689	0,210	-	1.978,001
70	233,087	204,419	28,668	**203,651**	203,406	0,245	-	2.116,502
75	283,085	246,693	36,392	**245,399**	245,151	0,248	-	2.322,118
80	316,996	277,802	39,194	**275,167**	274,907	0,260	-	2.573,457
85	358,940	313,708	45,232	**309,192**	308,911	0,281	-	2.605,556
90	423,236	371,150	52,086	**347,792**	347,479	0,313	-	2.943,059
95	481,492	421,519	59,973	**414,600**	414,268	0,332	-	3.137,764
100	509,231	458,350	63,881	**454,602**	454,243	0,359	-	3.567,112

É possível observar que o tempo de execução dos algoritmos aumenta conforme o ta-

manho do problema: quanto maior é o tamanho da instância, mais operações de movimentação são necessárias para realizar o deslocamento das partículas sobre o espaço de busca. Entretanto, o principal fator que eleva o tempo de execução é o tempo computacional da função "distância", já que quanto maior o tamanho do problema, mais rotas de caminho mínimo precisam ser identificadas, elevando assim o tempo total de execução dos algoritmos. O tempo de execução da função "distância" é praticamente o mesmo para os algoritmos PSO-B e PSO-P. Mesmo assim, a diferença entre os tempos de execução total de PSO-B e PSO-P segue aumentando consideravelmente conforme aumenta o valor de N (Figura 15). Enquanto o tempo total de execução do algoritmo PSO-P aumenta linearmente, o tempo total de execução PSO-B tem crescimento quadrático.

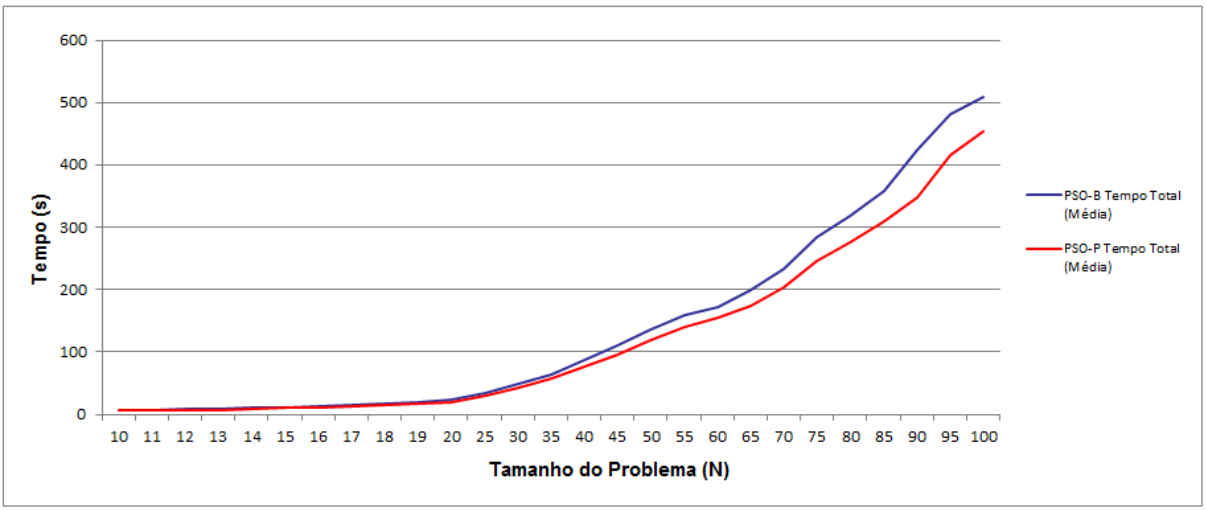

Figura 15: Tempo total de execução dos algoritmos

Fonte: Autoria própria

O PSO robusto apresenta tempos de execução mais elevados que os tempos dos algoritmos PSO-B e PSO-P, já que possui uma nuvem de partículas maior e é executado por mais iterações. Ainda assim, o PSO robusto é mais rápido que a busca exaustiva, e a sua execução sobre as maiores instâncias testadas neste trabalho é viável para a obtenção de soluções de referência.

A Figura 16 mostra um comparativo da diferença entre o tempo total de execução dos algoritmos e o tempo de execução da função "distância". No caso do algoritmo PSO-P, percebe-se que este tempo se mantém abaixo de um segundo em todos os casos testados, diferentemente do PSO-B, que apresenta tempos mais elevados, principalmete devido as operações adicionais relacionadas com o processo de reparação que se faz necessário para evitar a criação de soluções infactíveis para o problema.

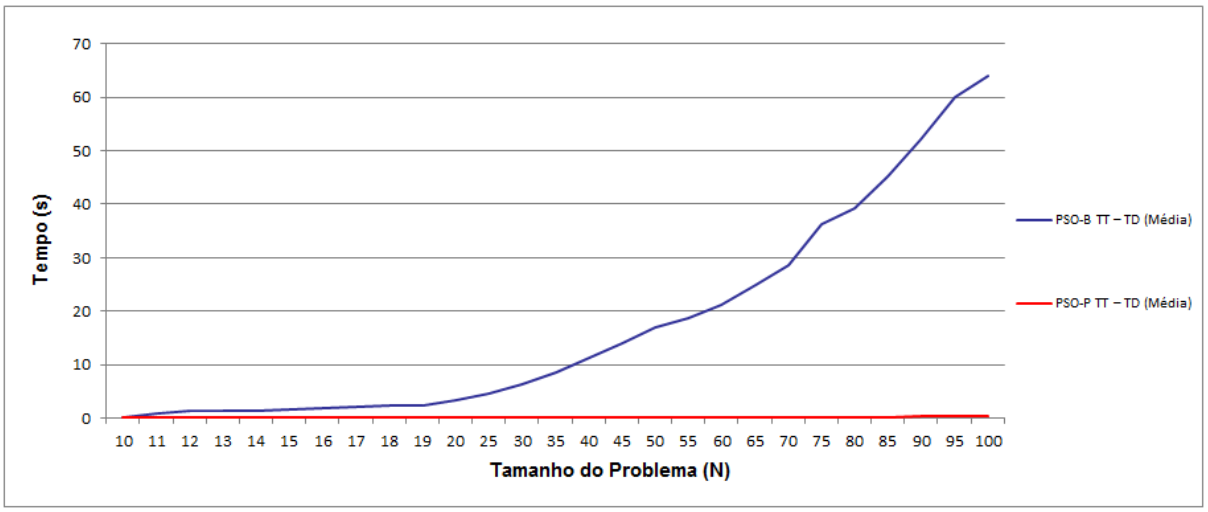

Figura 16: Diferença entre o tempo total de execução e o tempo da função distância
Fonte: Autoria própria

Diante dos resultados apresentados, o algoritmo PSO-P pode ser considerado melhor que o PSO-B, já que obtém soluções melhores em menos tempo de processamento. Sendo assim, na próxima seção o PSO-P é empregado em novos experimentos que abordam o PATC/TAP em um ambiente dinâmico de otimização.

7.6 RESOLUÇÃO DO PATC/TAP EM AMBIENTE DINÂMICO

Este experimento tem o objetivo de simular a dinâmica do PATC/TAP, e obter resultados da otimização realizada com o algoritmo DPSO-P neste ambiente dinâmico. A simulação ocorre com uma transição de episódios do problema, sendo que cada episódio do problema apresenta mudanças em relação ao episódio anterior.

7.6.1 CONJUNTOS DE TESTE DO PROBLEMA

Uma base de conjuntos de teste do problema (Tabela 8) foi criada através da aplicação de mudanças sucessivas, geradas artificalmente, sobre uma instância inicial. As mudanças aplicadas são aquelas ilustradas na Figuras 2, 3, 4 e 5 que constam no Capítulo 2.

Os conjuntos de teste criados possibilitam a análise da aplicação do algoritmo de otimização em diferentes situações, e se diferenciam entre si em termos de: tamanho N das instâncias; número de episódios sucessíveis da simulação; e escala M de mudanças ocorridas a cada transição.

Tabela 8: Conjuntos de teste do PATC/TAP dinâmico

Conjunto de Teste	N	Episódios Sucessíveis	Escala de Mudança (M)	Distância Média (de Hamming) entre Soluções Ótimas / de Referência
N10M1	10	100	1	3,0594
N10M3	10	100	3	4,5351
N10M5	10	100	5	6,0099
N100M10	100	10	10	-
N100M50	100	10	50	-

Neste experimento, foram testadas instâncias do problema com dois tamanhos: 10 e 100. Nos conjuntos de teste que possuem as instâncias menores ($N = 10$), foi criada uma série de 100 episódios do problema, e a solução ótima para cada episódio foi identificada previamente com a BE. Nos conjuntos maiores ($N = 100$) foi criada uma série de 10 episódios, e a solução de referência para cada episódio foi identificada com o PSO robusto; nestes casos a BE não foi empregada em decorrência do seu alto custo computacional.

A escala de mudança do problema foi definida neste trabalho como a quantidade de eventos de mudança que ocorrem em cada episódio de simulação, e influencia na distância de movimentação da solução ótima no espaço de busca. A tendência é que quanto maior seja a escala de mudança, para mais longe se mude a nova solução ótima. A Tabela 8 mostra ainda a distância média (de Hamming) entre as soluções ótimas / de referência do problema codificadas como sequências de posições, após a ocorrência de mudanças.

Além disso, foram testadas diferentes escalas de mudança do problema, o que permite que a habilidade de cada estratégia de resposta às mudanças seja observada de acordo com diferentes escalas de mudança.

7.6.2 SIMULAÇÃO DO PROBLEMA

O simulador desenvolvido aplica as mudanças à instância inicial de cada conjunto de teste do problema, enquanto o algoritmo DPSO-P é executado em segundo plano para reagir às mudanças e acompanhar as soluções ótimas. Considerando que a condição de parada do DPSO-P é o tempo total de simulação do problema, o algoritmo é executado durante a transição de toda a série de episódios de cada conjunto de testes.

É importante ressaltar que as mudanças são geradas artificialmente pelo simulador, sendo que as alocações de pares táxi-cliente resultantes do processo de otimização acabam produzindo novas mudanças. Além disso, é simulado o deslocamento geográfico dos táxis na direção dos clientes para os quais foram alocados. As operações de inclusão e remoção de agentes de oferta e demanda são realizadas conforme as respectivas filas, como definido no

Capítulo 2.

O tempo computacional necessário para a convergência da nuvem de partículas é considerado neste experimento, para determinar o intervalo entre a transição dos episódios de simulação. O experimento anterior (Seção 7.5) mostrou que o algoritmo PSO-P apresenta um tempo médio de execução total de $5,506$ segundos, no caso $N = 10$. Diante disso, a simulação ocorre com um intervalo de $5,506$ segundos entre os eventos de mudança aplicados pelo simulador. No caso $N = 100$, o intervalo entre mudanças é de $454,602$ segundos. Apesar de este intervalo de tempo ser suficiente para a convergência da nuvem de partículas, o que aparentemente reduz a relevância do experimento em questão, o objetivo é avaliar o comportamento do algoritmo na busca por uma nova solução ótima diante de situações onde a nuvem de partículas já convergiu para uma solução anterior.

Os experimentos de simulação do problema também são realizados com intervalo de mudança inferior (70%) ao tempo médio de convergência da nuvem de partículas no algoritmo PSO-P, com o objetivo de avaliar a aplicação do algoritmo em ambientes que apresentem mudanças mais frequentes. Nestes casos, as variáveis de decisão mudam antes mesmo de a nuvem de partículas convergir para a solução ótima, fazendo com que o algoritmo precise mudar a busca para a nova solução ótima.

Os parâmetros utilizados no experimento anterior são mantidos (Seção 7.5), salvo o número de iterações que deixou de ser uma condição de parada do algoritmo. Além disso, a equação da inércia foi modificada para que o decaimento seja reiniciado sempre que for detectada uma mudança, e para que a taxa de decaimento seja ajustada de acordo com o número médio de iterações entre a ocorrência de mudanças.

7.6.3 AVALIAÇÃO DE DESEMPENHO

Neste trabalho são utilizados os seguintes indicadores de desempenho, na simulação realizada com aqueles conjuntos de teste cuja solução ótima global é conhecida:

- Reações com sucesso (%): indica o percentual de sucesso obtido na tentativa de acompanhar o valor ótimo durante a ocorrência de mudanças. É considerada uma reação de "sucesso" aquela em que o algoritmo consegue encontrar a solução ótima (conhecida previamente), após a ocorrência de uma mudança. Este indicador está sendo proposto neste trabalho para complementar os comparativos das abordagens testadas;
- Erro prévio de *fitness*: indica a diferença média entre o *fitness* da melhor solução obtida no momento imediatamente anterior à mudança, e o valor ótimo previamente conhecido;

- Precisão média: representa a precisão média do algoritmo, considerando as amostras da precisão obtidas nos momentos imediatamente anteriores às mudanças;
- Estabilidade média: indica a estabilidade média do algoritmo, considerando os momentos de ocorrência de mudanças.

Diante das instâncias para as quais não se conhece a solução ótima, é utilizada uma solução de referência para calcular os indicadores apresentados.

7.6.4 RESULTADOS

As Tabelas 9 e 10 mostram os resultados obtidos no experimento realizado com o algoritmo DPSO-P, considerando-se os conjuntos de teste criados para a simulação do PATC/TAP em ambiente dinâmico. O algoritmo foi executado sobre os conjuntos de teste descritos na Tabela 8, uma vez para cada abordagem testada. Embora seja apenas uma execução, o algoritmo é executado durante o tempo de transição de um número considerável de episódios sucessíveis, e cada indicador de desempenho utilizado nos comparativos das abordagens representa uma média do seu respectivo valor, obtido em cada transição.

Nos conjuntos de teste das instâncias menores ($N = 10$), os indicadores foram calculados em relação a solução ótima, já que a informação da solução ótima de cada episódio do problema foi obtida previamente com a busca exaustiva. E nos conjuntos de teste das instâncias maiores ($N = 100$), a solução de referência , obtida com o PSO robusto, foi utilizada como referência no cálculo destes indicadores.

A abordagem $F2$ apresentou os melhores resultados nos testes realizados diante dos conjuntos de teste com $N = 10$, quando o intervalo de mudanças foi igual ao tempo médio necessário para a convergência do algoritmo. Independente da escala de mudança, com esta abordagem o algoritmo foi capaz de acompanhar a solução ótima do problema em todas as transições, e assim apresentar uma média de erro igual a 0. Ainda com intervalo de mudanças igual a 100% do tempo de convergência do algoritmo, mas agora diante das instâncias maiores ($N = 100$), a abordagem $C1$ apresentou os melhores resultados diante das mudanças suaves, e a abordagem $H1$ se mostrou melhor diante das mudanças abruptas.

Além disso, o DPSO-P foi testado em uma simulação que realiza a transição dos episódios sucessivos do problema com intervalo de 70% do tempo médio de convergência do algoritmo, ou seja, as mudanças ocorrem em um intervalo inferior ao tempo necessário para a convergência da nuvem de partículas. O objetivo é verificar a qualidade das soluções obtidas diante de situações com maior frequência de mudanças. Diante dos conjuntos de testes com

instâncias menores, a abordagem $C1$ apresentou os melhores resultados, independente de escala de mudança. No caso das instâncias maiores, a abordagem $D1$ apresentou os melhores resultados diante de mudanças suaves, e a abordagem $H1$ se mostrou melhor diante de mudanças abruptas.

Nesta seção foram apresentados apenas os resultados das melhores combinações de abordagens. Os resultados completos, obtidos com as demais combinações de abordagens, são apresentados no Apêndice A (Tabelas 11, 12, 13, 14 e 15).

Tabela 9: Resumo dos resultados da otimização do PATC/TAP dinâmico (N = 10)

Abordagem		Conjunto N10M1 - intervalo: 5,506 *seg* (100%)				intervalo: 3,854 *seg* (70%)		
		Sucesso (%)	**Erro**	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	3	0,00319	0,80608	0,05043	0,00712	0,56372	0,10910
A2	Diversificar 50% da nuvem	28	0,00134	0,91918	0,04403	0,00771	0,53510	0,07480
A3	Diversificar 100% da nuvem	22	0,00132	0,92475	0,03978	0,00791	0,53964	0,06537
B	Reinicializar memória das partículas	1	0,01127	0,77870	0,05412	0,01227	0,45231	0,10732
C1	Perturbação com chance de 10%	84	0,00010	0,99432	0,00374	**0,00367**	**0,77010**	**0,06032**
C2	Perturbação com chance de 50%	98	0,00000	0,99984	0,00016	0,00725	0,55851	0,08455
C3	Perturbação com chance de 100%	88	0,00005	0,99738	0,00215	0,00633	0,57071	0,08849
F1	A2 + C1	85	0,00010	0,99350	0,00213	0,00802	0,52094	0,08658
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00785	0,53981	0,05730
F3	A2 + C3	90	0,00011	0,99555	0,00433	0,00799	0,51934	0,09785
Abordagem		**Conjunto N10M3 - intervalo:** 5,506 *seg* (100%)				**intervalo:** 3,854 *seg* (70%)		
		Sucesso (%)	**Erro**	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	1	0,00306	0,79607	0,05337	0,00638	0,59544	0,10368
A2	Diversificar 50% da nuvem	20	0,00097	0,93707	0,02984	0,00663	0,58931	0,11675
A3	Diversificar 100% da nuvem	22	0,00156	0,89654	0,04906	0,00728	0,49814	0,08866
B	Reinicializar memória das partículas	0	0,01066	0,75215	0,05543	0,01082	0,47054	0,10519
C1	Perturbação com chance de 10%	65	0,00026	0,98481	0,00854	**0,00600**	**0,62262**	**0,10085**
C2	Perturbação com chance de 50%	82	0,00014	0,98710	0,00709	0,00741	0,52390	0,09280
C3	Perturbação com chance de 100%	68	0,00016	0,98913	0,00815	0,00691	0,53423	0,08025
F1	A2 + C1	71	0,00074	0,95054	0,03670	0,00653	0,60068	0,09349
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00689	0,55714	0,10197
F3	A2 + C3	77	0,00014	0,99088	0,00826	0,00723	0,55217	0,11190
Abordagem		**Conjunto N10M5 - intervalo:** 5,506 *seg* (100%)				**intervalo:** 3,854 *seg* (70%)		
		Sucesso (%)	**Erro**	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	3	0,00525	0,72710	0,08115	0,00998	0,51012	0,10057
A2	Diversificar 50% da nuvem	16	0,00235	0,88884	0,05603	0,00951	0,54057	0,10802
A3	Diversificar 100% da nuvem	17	0,00163	0,91721	0,04109	0,00986	0,51560	0,09484
B	Reinicializar memória das partículas	0	0,01547	0,65214	0,06298	0,01524	0,48719	0,10129
C1	Perturbação com chance de 10%	61	0,00042	0,97924	0,01430	**0,00888**	**0,57282**	**0,09117**
C2	Perturbação com chance de 50%	92	0,00006	0,99724	0,00260	0,00969	0,50859	0,10851
C3	Perturbação com chance de 100%	62	0,00031	0,98494	0,01163	0,00944	0,50865	0,11535
F1	A2 + C1	79	0,00012	0,99417	0,00484	0,00960	0,53096	0,10220
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00941	0,53975	0,11746
F3	A2 + C3	65	0,00161	0,92545	0,04667	0,00998	0,51500	0,09581

Tabela 10: Resumo dos resultados da otimização do PATC/TAP dinâmico (N = 100)

Abordagem		**Conjunto N100M10 - intervalo:** 454,602 *seg* (100%)			**intervalo:** 318,221 *seg* (70%)		
		Erro	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	0,00126	0,68396	0,37312	0,00955	0,42405	0,45591
A2	Diversificar 50% da nuvem	0,00115	0,71115	0,18243	0,00936	0,49433	0,47544
A3	Diversificar 100% da nuvem	0,00065	0,93601	0,12365	0,00917	0,58320	0,49686
B	Reinicializar memória das partículas	0,00146	0,67901	0,28780	0,00947	0,04381	0,48915
C1	Perturbação com chance de 10%	**0,00028**	**0,98554**	**0,06810**	0,00931	0,53724	0,45251
C2	Perturbação com chance de 50%	0,00063	0,72849	0,23962	0,00965	0,52166	0,27481
C3	Perturbação com chance de 100%	0,00041	0,98301	0,32591	0,00903	0,51432	0,36410
D1	A1 + C1	0,00047	0,98212	0,23212	**0,00887**	**0,59836**	**0,33503**
D2	A1 + C2	0,00030	0,98419	0,39136	0,00935	0,55025	0,28785
D3	A1 + C3	0,00069	0,93011	0,27521	0,00968	0,50056	0,20476
Abordagem		**Conjunto N100M50 - intervalo:** 454,602 *seg* (100%)			**intervalo:** 318,221 *seg* (70%)		
		Erro	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	0,00575	0,60718	0,57115	0,01522	0,46178	0,45762
A2	Diversificar 50% da nuvem	0,00506	0,76669	0,25571	0,01454	0,43445	0,42733
A3	Diversificar 100% da nuvem	0,00503	0,74323	0,40578	0,01721	0,41174	0,51451
B	Reinicializar memória das partículas	0,00704	0,48768	0,33978	0,01751	0,41223	0,69811
C1	Perturbação com chance de 10%	0,00431	0,96192	0,48399	0,01498	0,40900	0,49084
C2	Perturbação com chance de 50%	0,00504	0,67075	0,45440	0,01633	0,46625	0,41418
C3	Perturbação com chance de 100%	0,00591	0,87476	0,35422	0,01576	0,41212	0,45112
H1	A3 + C1	**0,00312**	**0,87650**	**0,23011**	**0,00945**	**0,56644**	**0,47991**
H2	A3 + C2	0,00481	0,71443	0,65983	0,01561	0,49887	0,49031
H3	A3 + C3	0,00412	0,78191	0,56731	0,01512	0,48455	0,51345

8 CONCLUSÕES E PERSPECTIVAS

Neste trabalho foi proposta a utilização do algoritmo PSO discreto para a otimização do Problema de Atribuição de Tarefas, em uma aplicação do Problema de Alocação Táxi-Cliente. Duas versões do PSO discreto foram consideradas: a primeira utiliza codificação binária e a segunda emprega permutação de sequências de posições. Os resultados obtidos por estes dois algoritmos foram comparados com a solução ótima obtida por um método de busca exaustiva, com o objetivo de avaliar qualidade e desempenho; as soluções ótimas foram obtidas em instâncias do problema com tamanho $N \in \{10, 11, 12, 13\}$.

Para $N > 13$ não foi possível identificar o valor ótimo, devido ao custo computacional da busca exaustiva (por exemplo: para $N = 13$, o tempo computacional foi superior a três horas, e este tempo aumenta exponencialmente conforme aumenta o tamanho do problema). Desta forma, as soluções encontradas pela BE são importantes para medir a qualidade das soluções obtidas com os algoritmos de busca heurística, embora o seu tempo de execução não possa ser tomado como referência em qualquer comparação.

No caso das instâncias maiores, com tamanho $N \in \{15, 20, 25, ..., 100\}$, as soluções obtidas pelos algoritmos desenvolvidos foram comparadas com soluções de referência do problema, obtidas com um PSO robusto que utiliza uma nuvem maior e é executado sobre um elevado número de iterações.

O algoritmo PSO-P foi capaz de otimizar o PATC/TAP em todas as instâncias do problema cujo valor ótimo era conhecido. Por outro lado, o PSO-B apenas se aproximou das soluções ótimas nas instâncias menores. Para ilustrar esta afirmação, o algoritmo PSO-B alcançou um *fitness* médio $14,10\%$ pior que o PSO-P quando $N = 10$, e chegou a ser $37,10\%$ pior no caso $N = 100$.

O algoritmo PSO-P apresentou menor custo computacional, o que se explica pela ausência de restrições na movimentação das partículas sobre o espaço de busca. O processo de prevenção de soluções inválidas no PSO-B acaba elevando o tempo computacional do algoritmo. Foi observado que a maior parte do custo computacional dos algoritmos PSO-B e PSO-P

é atribuída ao emprego da função "distância", que utiliza o algoritmo de Dijkstra sobre os dados topológicos do Open Street Map para encontrar as rotas mais curtas e calcular o custo das soluções.

Diante do exposto, os resultados obtidos com PSO-P são promissores: o algoritmo é capaz de encontrar as soluções ótimas em menor tempo, tornando-se um candidato para ser utilizado em aplicações *online*. Cabe observar que esta proposta pode ser aplicada em um sistema de alocação real, onde a posição dos agentes (táxi e cliente) poderia ser sinalizada para uma estação de controle central em tempo real através de dispositivos móveis, utilizando coordenadas do Sistema de Posicionamento Global - *Global Positioning System* (GPS).

Alguns dos aspectos dinâmicos do problema foram considerados, com a proposta do algoritmo DPSO-P, o qual utiliza o mesmo esquema de codificação de partículas do PSO-P e implementa algumas abordagens de resposta às mudanças em problema dinâmicos, a saber: (*a*) reinicialização de uma porção da nuvem quando detectada uma mudança nas variáveis de decisão do problema; (*b*) reinicialização da memória de todas as partículas da nuvem quando detectada uma mudança; e (*c*) utilização de um operador de perturbação aplicado sobre as partículas a cada iteração, com uma certa probabilidade.

Os resultados obtidos permitem concluir que, no caso de instâncias menores, é eficiente a reinicialização de metade da nuvem de partículas como uma resposta às mudanças, sejam elas mudanças suaves ou abruptas. O operador de perturbação com chance de 50% se mostrou preponderante para a manutenção da diversidade necessária na nuvem. A combinação destas abordagens resultou no rastreamento adequado da solução ótima do PATC/TAP dinâmico, em todos os conjuntos testados onde a solução ótima era conhecida e o intervalo de mudanças era igual ao tempo de convergência do algoritmo. Nos casos em que o intervalo de mudanças era inferior, a abordagem de perturbação com chance de 10% se mostrou mais eficiente.

No caso das instâncias maiores é preferível utilizar uma probabilidade menor para o operador de perturbação, independente da escala de mudanças, sendo que diante de mudanças abruptas também foi necessária a reinicialização completa da nuvem após a identificação de mudanças nas variáveis de decisão do problema.

Ainda que no caso prático do problema de alocação de táxis as mudanças ocorram em intervalos de tempo maiores do que os tempos de convergência apresentados pelo algoritmo, esta é uma questão que merece maior investigação, até mesmo para viabilizar a utilização do algoritmo nas aplicações onde o tempo de processamento é crítico. A função *"distância"* atualmente é responsável por elevar o custo computacional do algoritmo, representando mais de 99% do tempo total de processamento.

Em trabalhos futuros, os algoritmos e experimentos podem ser ampliados considerando o caso $|V| \neq |P|$ (número diferente de agentes de oferta e demanda). Esta extensão pode ser tratada internamente, deixando de utilizar matrizes quadradas.

Os resultados apresentados anteriormente no trabalho de Pierobom et al. (2011) mostram que o algoritmo PSO-P apresenta tempo de execução total inferior à 1 segundo, quando utilizando uma função de avaliação que calcula a distância Euclidiana entre dois pontos no mapa, sem considerar questões relacionadas com roteamento sobre as vias existentes. Sendo assim, fica evidente a necessidade de alternativas para a redução do custo computacional da função "distância", o que poderá viabilizar a obtenção das soluções ótimas do problema em ambientes de otimização que apresentem intervalos de mudança reduzidos. Uma abordagem possível é a de dividir o mapa em quadros e calcular a distância entre estes quadros; e num segundo passo, calcular a distância entre os agentes. Esta abordagem reduziria o número de execuções da função "distância", reduzindo o custo de execução do algoritmo.

Para todos os tamanhos de instâncias do problema que foram testadas neste trabalho, foi utilizado um mesmo número de partículas na nuvem do PSO. É interessante que o algoritmo seja modificado no sentido de aumentar o número de partículas conforme aumenta o tamanho das instâncias, já que o espaço de busca também aumenta.

Acerca do desenvolvimento do algoritmo, em trabalhos futuros é interessante que seja analisada a sua sensibilidade de acordo com os parâmetros do sistema, como o fator de inércia e os fatores de influência do grupo e autoconfiança da partícula no cálculo da velocidade.

Os resultados apresentados também encorajam a otimização multiobjetivo do problema (minimizar o tempo de espera dos agentes de oferta, minimizar o tempo de espera dos agentes de demanda), além da exploração das demais questões sobre tráfego urbano, como a previsão de congestionamento em vias.

Outras aplicações similares podem ser tratadas da mesma maneira, como por exemplo: otimização de tempos de ônibus e metrôs; alocação de viaturas policiais para o atendimento de ocorrências; alocação de técnicos de campo para o atendimento de chamados.

REFERÊNCIAS

BANKS, A.; VINCENT, J.; ANYAKOHA, C. A review of particle swarm optimization. part i: background and development. **Natural Computing**, Kluwer Academic Publishers, Hingham, MA, USA, v. 6, n. 4, p. 467–484, 2007.

BANKS, A.; VINCENT, J.; ANYAKOHA, C. A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. **Natural Computing**, Kluwer Academic Publishers, Hingham, MA, USA, v. 7, n. 1, p. 109–124, 2008.

BLACKWELL, T. Particle swarm optimization in dynamic environments. In: **Evolutionary Computation in Dynamic and Uncertain Environments**. Berlin, Heildelberg: Springer, 2007. cap. 2, p. 29–49.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys (CSUR)**, ACM, New York, NY, USA, v. 35, n. 3, p. 268–308, 2003.

CARLISLE, A.; DOZIER, G. Adapting particle swarm optimization to dynamic environments. In: **Proceedings of International Conference on Artificial Intelligence**. Las Vegas, Nevada, USA: , 2000. p. 429–434.

CHRISTOFIDES, N. et al. The travelling salesman problem. In: **Combinatorial Optimization**Wiley, Chichester, 1979. p. 131–149.

COELLO, C. A. C.; REYES-SIERRA, M. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. **International Journal of Computational Intelligence Research**, Citeseer, v. 2, n. 3, p. 287–308, 2006.

COSTA, B.; COSTA, F. O sistema de táxis: mobilidade urbana e redução nas emissões de gases de efeito estufa no rio de janeiro. In: **4º Congresso Luso-Brasileiro para o Planejamento Urbano, Regional, Integrado e Sustentável**. Faro, Portugal: , 2010.

DIAZ, J. A.; FERNANDEZ, E. A tabu search heuristic for the generalized assignment problem. **European Journal of Operational Research**, v. 132, n. 1, p. 22 – 38, 2001.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, Springer Berlin / Heidelberg, v. 1, n. 1, p. 269–271, 1959.

EBERHART, R. C.; SHI, Y. Comparison between genetic algorithms and particle swarm optimization. In: **Proceedings of the 7th International Conference on Evolutionary Programming VII**. London, UK: Springer-Verlag, 1998. (EP '98), p. 611–616.

EBERHART, R. C.; SHI, Y. Comparing inertia weights and constriction factors in particle swarm optimization. In: **Proceedings of the 2000 IEEE Congress on Evolutionary Computation**. Piscataway, NJ, USA: IEEE Press, 2000. v. 1, p. 84–88 vol.1.

EBERHART, R. C.; SHI, Y. Tracking and optimizing dynamic systems with particle swarms. In: **Evolutionary Computation, 2001. Proceedings of the 2001 Congress on**. Piscataway, NJ, USA: IEEE Service Center, 2001. v. 1, p. 94–100.

ENGELBRECHT, A. P. **Computational Intelligence: An Introduction**. 2nd. ed. Chichester, UK: Wiley Publishing, 2007.

ESQUIVEL, S. C.; COELLO, C. C. A. Particle swarm optimization in non-stationary environments. In: **Advances in Artificial Intelligence**. Heidelberg: Springer, 2004. v. 3315.

FANG, L.; CHEN, P.; LIU, S. Particle swarm optimization with simulated annealing for tsp. In: **Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases - Volume 6**. Stevens Point, WI, USA: World Scientific and Engineering Academy and Society (WSEAS), 2007. p. 206–210.

GOLDBARG, E. F.; GOLDBARG, M. F.; SOUZA, G. R. Particle swarm optimization algorithm for the traveling salesman problem. In: **Evolutionary Computation in Combinatorial Optimization, 6th European Conference, EvoCOP**. Budapest, Hungary: , 2006. p. 99–110.

GOOGLE. **Google Maps Javascript API Version 3 (http://www.google.com/apis/maps) - Acesso em 05/01/2011**. 2011.

HAKLAY, M. M.; WEBER, P. Openstreetmap: User-generated street maps. **IEEE Pervasive Computing**, IEEE Computer Society, Los Alamitos, CA, USA, v. 7, p. 12–18, 2008.

HU, X.; EBERHART, R. C. Tracking dynamic systems with pso: where's the cheese? In: **Proceedings of the workshop on particle swarm optimization.** Indianapolis, IN, USA: Purdue School of Engineering and Technology, 2001.

HU, X.; EBERHART, R. C. Adaptive particle swarm optimization: detection and response to dynamic systems. In: **Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02**. Washington, DC, USA: IEEE Computer Society, 2002. p. 1666–1670.

HU, X.; EBERHART, R. C.; SHI, Y. Swarm intelligence for permutation optimization: a case study on n-queens problem. In: **Proceedings of the IEEE Swarm Intelligence Symposium 2003**. Indianapolis, IN, USA: , 2003. p. 243–246.

JARBOUI, B. et al. A combinatorial particle swarm optimisation for solving permutation flowshop problems. **Computers & Industrial Engineering**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 54, p. 526–538, 2008.

JIN, Y.; BRANKE, J. Evolutionary optimization in uncertain environments-a survey. **Evolutionary Computation, IEEE Transactions on**, v. 9, n. 3, p. 303 – 317, 2005.

JOHNSON, S. **Emergence: The Connected Lives of Ants, Brains, Cities and Software**. 1st. ed.Scribner, 2002.

KAMOSI, M.; HASHEMI, A.; MEYBODI, M. A new particle swarm optimization algorithm for dynamic environments. In: **Swarm, Evolutionary, and Memetic Computing**. Berlin, Heidelberg: Springer, 2010. v. 6466, p. 129–138.

KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: **Proceeding of IEEE International Conference on Neural Networks**. Piscataway, NJ, USA: IEEE Computer Society, 1995. p. 1942–1948.

KENNEDY, J.; EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In: **Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on**. Piscataway, NJ, USA: IEEE Computer Society, 1997. v. 5, p. 4104–4108.

KUO, I.-H. et al. An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. **Expert Systems with Applications**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 36, n. 3, p. 7027–7032, 2009.

LI, C.; YANG, S. Fast multi-swarm optimization for dynamic optimization problems. In: **Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 07**. Washington, DC, USA: IEEE Computer Society, 2008. p. 624–628.

LIAO, C.-J.; TSENG, C.-T.; LUARN, P. A discrete version of particle swarm optimization for flowshop scheduling problems. **Computers & Operations Research**, Elsevier Science Ltd., Oxford, UK, v. 34, n. 10, p. 3099–3111, 2007.

LIU, B.; WANG, L.; JIN, Y.-H. An effective hybrid pso-based algorithm for flow shop scheduling with limited buffers. **Computers & Operations Research**, Elsevier Science Ltd., Oxford, UK, v. 35, n. 9, p. 2791–2806, 2008.

LIU, H.; GAO, L.; PAN, Q. A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. **Expert Systems with Applications**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 38, n. 4, p. 4348–4360, 2011.

MOELLER, C. **OSM2PO (http://www.osm2po.de) - Acesso em 05/10/2011**. 2011.

MORRISON, R. W. Performance measurement in dynamic environments. In: **Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference**. Chigaco, IL, USA: AAAI, 2003. p. 99–102.

NAGANO, M. S.; MOCCELLIN, J. V.; LORENA, L. A. N. Programação da produção flow shop permutacional com minimização do tempo médio de fluxo. In: **XXXVI Simpósio Brasileiro de Pesquisa Operacional, 2004**. São João Del-Rei - MG: Anais do XXXVI SBPO, 2004.

OPENLAYERS. **OpenLayers Javascript API (http://trac.osgeo.org/openlayers/) - Acesso em 17/10/2011**. 2011.

OSM. **Open Street Map (http://www.openstreetmap.org) - Acesso em 05/09/2011**. 2011.

PAPADIMITRIOU, C. H.; STEIGLEITZ, K. **Combinatorial optimization: algorithms and complexity**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

PGROUTING. **pgRouting open source routing library (http://www.pgrouting.org/) - Acesso em 06/10/2011**. 2011.

PIEROBOM, J. L.; DELGADO, M. R. B. S.; KAESTNER, C. A. A. Particle swarm optimization applied to task assigment problem. In: **X Congresso Brasileiro de Inteligência Computacional**. Fortaleza, CE: Anais do CBIC 2011, 2011.

PIGATTI, A.; POGGI; UCHOA, E. Stabilized branch-and-cut-and-price for the generalized assignment problem. In: **Electronic Notes in Discrete Mathematics**. 2005. v. 19, p. 389–395.

POLI, R. Analysis of the publications on the applications of particle swarm optimisation. **Journal of Artificial Evolution and Applications**, v. 2008, n. 1, p. 1–10, 2008.

POSTGIS. **PostGIS spatial database extension for PostgreSQL (http://postgis.refractions.net/)- Acesso em 02/10/2011**. 2011.

REEVES, W. T. Particle systems - a technique for modeling a class of fuzzy objects. **ACM Transactions on Graphics**, ACM, New York, NY, USA, v. 2, p. 91–108, 1983.

REYNOLDS, C. W. Flocks, herds and schools: A distributed behavioral model. **Computer Graphics**, ACM, New York, NY, USA, v. 21, n. 4, p. 25–34, 1987.

ROSENDO, M.; POZO, A. A hybrid particle swarm optimization algorithm for combinatorial optimization problems. In: **Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010**. Barcelona, Spain: IEEE Computer Society, 2010. p. 1–8.

SALMAN, A. Particle swarm optimization for task assignment problem. **Microprocessors and Microsystems**, v. 26, p. 363–371, 2002.

SHA, D. Y.; HSU, C.-Y. A hybrid particle swarm optimization for job shop scheduling problem. **Computers & Industrial Engineering**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 51, p. 791–808, 2006.

SHA, D. Y.; HSU, C.-Y. A new particle swarm optimization for the open shop scheduling problem. **Computers & Operations Research**, Elsevier Science Ltd., Oxford, UK, v. 35, p. 3243–3261, 2008.

SHA, D. Y.; LIN, H.-H. A multi-objective pso for job-shop scheduling problems. **Expert Systems with Applications**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 37, n. 2, p. 1065–1070, 2010.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. In: **Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on**. 1998. p. 69–73.

SHIAU, D.-F. A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences. **Expert Systems with Applications**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 38, n. 1, p. 235–248, 2011.

TANG, J.; ZHAO, X. A hybrid particle swarm optimization with adaptive local search. **Journal of Networks**, v. 5, n. 4, p. 411–418, 2010.

TANK, D. W.; HOPFIELD, J. J. Collective computation in neuronlike circuits. **Scientific American**, Scientific American, Inc., New York, NY, USA, v. 257, n. 6, p. 104–114, 1987.

VOUDOURIS, C. Guided local search and its application to the traveling salesman problem. **European Journal of Operational Research**, v. 113, n. 2, p. 469–499, 1999.

WEICKER, K. Performance measures for dynamic environments. In: **Proceedings of the 7th International Conference on Parallel Problem Solving from Nature**. London, UK: Springer-Verlag, 2002. p. 64–76.

YANG, X. et al. A modified particle swarm optimizer with dynamic adaptation. **Applied Mathematics and Computation**, v. 189, n. 2, p. 1205 – 1213, 2007.

ZUBEN, F. J. V.; ATTUX, R. R. F. Inteligência de enxame. **DCA/FEEC/Unicamp e DECOM/-FEEC/Unicamp**, 2008.

APÊNDICE A – RESULTADOS COMPLETOS

A.1 TABELAS COMPLETAS

Tabela 11: Resultados completos da otimização do PATC/TAP dinâmico (*N10M1*)

Abordagem		Conjunto N10M1 - intervalo: 5,506 *seg* (100%)				intervalo: 3,854 *seg* (70%)		
		Sucesso (%)	**Erro**	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	3	0,00319	0,80608	0,05043	0,00712	0,56372	0,10910
A2	Diversificar 50% da nuvem	28	0,00134	0,91918	0,04403	0,00771	0,53510	0,07480
A3	Diversificar 100% da nuvem	22	0,00132	0,92475	0,03978	0,00791	0,53964	0,06537
B	Reinicializar memória das partículas	1	0,01127	0,77870	0,05412	0,01227	0,45231	0,10732
C1	Perturbação com chance de 10%	84	0,00010	0,99432	0,00374	**0,00367**	**0,77010**	**0,06032**
C2	Perturbação com chance de 50%	98	0,00000	0,99984	0,00016	0,00725	0,55851	0,08455
C3	Perturbação com chance de 100%	88	0,00005	0,99738	0,00215	0,00633	0,57071	0,08849
D1	A1 + C1	89	0,00012	0,99503	0,00473	0,00743	0,54155	0,09737
D2	A1 + C2	98	0,00001	0,99976	0,00024	0,00800	0,51014	0,08336
D3	A1 + C3	91	0,00003	0,99834	0,00109	0,00656	0,59211	0,13381
E1	A1 + B + C1	83	0,00020	0,98963	0,00874	0,00646	0,59422	0,11812
E2	A1 + B + C2	100	0,00000	0,99993	0,00007	0,00754	0,54262	0,08063
E3	A1 + B + C3	81	0,00009	0,99425	0,00324	0,00665	0,57976	0,10280
F1	A2 + C1	85	0,00010	0,99350	0,00213	0,00802	0,52094	0,08658
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00785	0,53981	0,05730
F3	A2 + C3	90	0,00011	0,99555	0,00433	0,00799	0,51934	0,09785
G1	A2 + B + C1	83	0,00011	0,99282	0,00235	0,00731	0,56876	0,07779
G2	A2 + B + C2	99	0,00001	0,99987	0,00013	0,00790	0,52324	0,06109
G3	A2 + B + C3	81	0,00008	0,99536	0,00333	0,00783	0,53924	0,09628
H1	A3 + C1	79	0,00012	0,99269	0,00568	0,00852	0,49530	0,05917
H2	A3 + C2	97	0,00001	0,99965	0,00035	0,00834	0,51332	0,06301
H3	A3 + C3	80	0,00016	0,99052	0,00784	0,00800	0,52828	0,10041
I1	A3 + B + C1	79	0,00016	0,99073	0,00768	0,00772	0,54442	0,05902
I2	A3 + B + C2	99	0,00001	0,99983	0,00017	0,00702	0,59735	0,08244
I3	A3 + B + C3	73	0,00020	0,98865	0,00902	0,00772	0,55231	0,09974

Tabela 12: Resultados completos da otimização do PATC/TAP dinâmico (*N10M3*)

Abordagem		Conjunto N10M3 - intervalo: 5,506 *seg* (100%)				intervalo: 3,854 *seg* (70%)		
		Sucesso (%)	Erro	Precisão	Estabilidade	Erro	Precisão	Estabilidade
A1	Diversificar 10% da nuvem	1	0,00306	0,79607	0,05337	0,00638	0,59544	0,10368
A2	Diversificar 50% da nuvem	20	0,00097	0,93707	0,02984	0,00663	0,58931	0,11675
A3	Diversificar 100% da nuvem	22	0,00156	0,89654	0,04906	0,00728	0,49814	0,08866
B	Reinicializar memória das partículas	0	0,01066	0,75215	0,05543	0,01082	0,47054	0,10519
C1	Perturbação com chance de 10%	65	0,00026	0,98481	0,00854	**0,00600**	**0,62262**	**0,10085**
C2	Perturbação com chance de 50%	82	0,00014	0,98710	0,00709	0,00741	0,52390	0,09280
C3	Perturbação com chance de 100%	68	0,00016	0,98913	0,00815	0,00691	0,53423	0,08025
D1	A1 + C1	76	0,00033	0,98497	0,00971	0,00724	0,50409	0,11254
D2	A1 + C2	92	0,00002	0,99824	0,00173	0,00645	0,59700	0,10996
D3	A1 + C3	77	0,00022	0,98988	0,00646	0,00693	0,56139	0,10976
E1	A1 + B + C1	60	0,00043	0,97669	0,01483	0,00666	0,57097	0,10277
E2	A1 + B + C2	89	0,00006	0,99632	0,00335	0,00649	0,59038	0,11845
E3	A1 + B + C3	56	0,00058	0,96886	0,01552	0,00664	0,58474	0,09407
F1	A2 + C1	71	0,00074	0,95054	0,03670	0,00653	0,60068	0,09349
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00689	0,55714	0,10197
F3	A2 + C3	77	0,00014	0,99088	0,00826	0,00723	0,55217	0,11190
G1	A2 + B + C1	62	0,00025	0,98592	0,01150	0,00648	0,60128	0,09700
G2	A2 + B + C2	91	0,00003	0,99791	0,00206	0,00692	0,54764	0,09086
G3	A2 + B + C3	68	0,00031	0,98441	0,00960	0,00652	0,60055	0,12320
H1	A3 + C1	68	0,00034	0,98102	0,01574	0,00672	0,58147	0,08547
H2	A3 + C2	91	0,00009	0,99549	0,00451	0,00636	0,60914	0,09994
H3	A3 + C3	70	0,00021	0,98808	0,00886	0,00658	0,58262	0,09111
I1	A3 + B + C1	60	0,00032	0,98008	0,01383	0,00657	0,57567	0,08856
I2	A3 + B + C2	82	0,00112	0,94836	0,02609	0,00685	0,56814	0,07674
I3	A3 + B + C3	57	0,00101	0,94466	0,02948	0,00635	0,61796	0,13009

Tabela 13: Resultados completos da otimização do PATC/TAP dinâmico (*N10M5*)

Abordagem		Conjunto N10M5 - intervalo: 5,506 *seg* (100%)				intervalo: 3,854 *seg* (70%)		
		Sucesso (%)	**Erro**	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	3	0,00525	0,72710	0,08115	0,00998	0,51012	0,10057
A2	Diversificar 50% da nuvem	16	0,00235	0,88884	0,05603	0,00951	0,54057	0,10802
A3	Diversificar 100% da nuvem	17	0,00163	0,91721	0,04109	0,00986	0,51560	0,09484
B	Reinicializar memória das partículas	0	0,01547	0,65214	0,06298	0,01524	0,48719	0,10129
C1	Perturbação com chance de 10%	61	0,00042	0,97924	0,01430	**0,00888**	**0,57282**	**0,09117**
C2	Perturbação com chance de 50%	92	0,00006	0,99724	0,00260	0,00969	0,50859	0,10851
C3	Perturbação com chance de 100%	62	0,00031	0,98494	0,01163	0,00944	0,50865	0,11535
D1	A1 + C1	84	0,00021	0,99063	0,00813	0,01053	0,46910	0,10874
D2	A1 + C2	95	0,00004	0,99810	0,00190	0,00923	0,54766	0,11718
D3	A1 + C3	79	0,00016	0,99221	0,00662	0,00997	0,48684	0,09217
E1	A1 + B + C1	57	0,00046	0,97590	0,02057	0,00959	0,52219	0,11919
E2	A1 + B + C2	91	0,00007	0,99633	0,00359	0,00996	0,51370	0,12042
E3	A1 + B + C3	64	0,00043	0,97819	0,01745	0,00991	0,50303	0,10141
F1	A2 + C1	79	0,00012	0,99417	0,00484	0,00960	0,53096	0,10220
F2	A2 + C2	**100**	**0,00000**	**1,00000**	**0,00000**	0,00941	0,53975	0,11746
F3	A2 + C3	65	0,00161	0,92545	0,04667	0,00998	0,51500	0,09581
G1	A2 + B + C1	72	0,00030	0,98596	0,01248	0,01002	0,50609	0,09995
G2	A2 + B + C2	89	0,00055	0,97438	0,02217	0,00983	0,50123	0,08057
G3	A2 + B + C3	70	0,00061	0,96695	0,02233	0,00896	0,56055	0,09344
H1	A3 + C1	67	0,00049	0,97748	0,01903	0,00937	0,51308	0,11349
H2	A3 + C2	93	0,00003	0,99851	0,00149	0,00901	0,55958	0,10397
H3	A3 + C3	66	0,00036	0,98187	0,01453	0,01017	0,50264	0,09311
I1	A3 + B + C1	72	0,00027	0,98734	0,00938	0,01017	0,50264	0,09311
I2	A3 + B + C2	91	0,00008	0,99589	0,00361	0,00931	0,53723	0,09004
I3	A3 + B + C3	79	0,00016	0,99185	0,00638	0,00970	0,51380	0,09870

Tabela 14: Resultados completos da otimização do PATC/TAP dinâmico (*N100M10*)

Abordagem		Conjunto N100M10 - intervalo: 454,602 *seg* (100%)			intervalo: 318,221 *seg* (70%)		
		Erro	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	0,00126	0,68396	0,37312	0,00955	0,42405	0,45591
A2	Diversificar 50% da nuvem	0,00115	0,71115	0,18243	0,00936	0,49433	0,47544
A3	Diversificar 100% da nuvem	0,00065	0,93601	0,12365	0,00917	0,58320	0,49686
B	Reinicializar memória das partículas	0,00146	0,67901	0,28780	0,00947	0,04381	0,48915
C1	Perturbação com chance de 10%	**0,00028**	**0,98554**	**0,06810**	0,00931	0,53724	0,45251
C2	Perturbação com chance de 50%	0,00063	0,72849	0,23962	0,00965	0,52166	0,27481
C3	Perturbação com chance de 100%	0,00041	0,98301	0,32591	0,00903	0,51432	0,36410
D1	A1 + C1	0,00047	0,98212	0,23212	**0,00887**	**0,59836**	**0,33503**
D2	A1 + C2	0,00030	0,98419	0,39136	0,00935	0,55025	0,28785
D3	A1 + C3	0,00069	0,93011	0,27521	0,00968	0,50056	0,20476
E1	A1 + B + C1	0,00090	0,89115	0,22222	0,00992	0,40868	0,41498
E2	A1 + B + C2	0,00047	0,87658	0,16243	0,00913	0,56003	0,30474
E3	A1 + B + C3	0,00093	0,91030	0,24026	0,00982	0,46434	0,25863
F1	A2 + C1	0,00084	0,87247	0,34358	0,00990	0,50006	0,33542
F2	A2 + C2	0,00041	0,74696	0,29780	0,00916	0,53228	0,22331
F3	A2 + C3	0,00071	0,81239	0,14058	0,00982	0,56535	0,23576
G1	A2 + B + C1	0,00092	0,70673	0,26102	0,00986	0,58180	0,37591
G2	A2 + B + C2	0,00040	0,98414	0,25713	0,00912	0,45542	0,42131
G3	A2 + B + C3	0,00060	0,86768	0,22932	0,00960	0,56768	0,22938
H1	A3 + C1	0,00081	0,76341	0,36731	0,00933	0,54874	0,63877
H2	A3 + C2	0,00035	0,91317	0,21101	0,00927	0,58645	0,35948
H3	A3 + C3	0,00073	0,86124	0,32317	0,00934	0,55269	0,41377
I1	A3 + B + C1	0,00059	0,70981	0,33331	0,00928	0,52572	0,36533
I2	A3 + B + C2	0,00037	0,99021	0,17345	0,00927	0,58547	0,35299
I3	A3 + B + C3	0,00049	0,88936	0,29809	0,00931	0,56101	0,39188

Tabela 15: Resultados completos da otimização do PATC/TAP dinâmico (*N100M50*)

Abordagem		**Conjunto N100M50 - intervalo:** 454,602 *seg* (100%)			**intervalo:** 318,221 *seg* (70%)		
		Erro	**Precisão**	**Estabilidade**	**Erro**	**Precisão**	**Estabilidade**
A1	Diversificar 10% da nuvem	0,00575	0,60718	0,57115	0,01522	0,46178	0,45762
A2	Diversificar 50% da nuvem	0,00506	0,76669	0,25571	0,01454	0,43445	0,42733
A3	Diversificar 100% da nuvem	0,00503	0,74323	0,40578	0,01721	0,41174	0,51451
B	Reinicializar memória das partículas	0,00704	0,48768	0,33978	0,01751	0,41223	0,69811
C1	Perturbação com chance de 10%	0,00431	0,96192	0,48399	0,01498	0,40900	0,49084
C2	Perturbação com chance de 50%	0,00504	0,67075	0,45440	0,01633	0,46625	0,41418
C3	Perturbação com chance de 100%	0,00591	0,87476	0,35422	0,01576	0,41212	0,45112
D1	A1 + C1	0,00355	0,78590	0,58117	0,01609	0,40212	0,39011
D2	A1 + C2	0,00582	0,81087	0,42054	0,01578	0,49887	0,40892
D3	A1 + C3	0,00378	0,78168	0,45743	0,01565	0,56122	0,45437
E1	A1 + B + C1	0,00398	0,70469	0,88072	0,01477	0,47863	0,42472
E2	A1 + B + C2	0,00422	0,67298	0,51410	0,01576	0,47761	0,42681
E3	A1 + B + C3	0,00391	0,78211	0,79292	0,01501	0,48122	0,52119
F1	A2 + C1	0,00418	0,70728	0,60091	0,01633	0,47656	0,49800
F2	A2 + C2	0,00466	0,74923	0,57340	0,01554	0,45509	0,48221
F3	A2 + C3	0,00378	0,68629	0,64649	0,01671	0,44388	0,50911
G1	A2 + B + C1	0,00364	0,71428	0,62702	0,01590	0,48988	0,48902
G2	A2 + B + C2	0,00501	0,64633	0,33825	0,01621	0,42234	0,49556
G3	A2 + B + C3	0,00399	0,70864	0,51068	0,01588	0,40090	0,51549
H1	A3 + C1	**0,00312**	**0,87650**	**0,23011**	**0,00945**	**0,56644**	**0,47991**
H2	A3 + C2	0,00481	0,71443	0,65983	0,01561	0,49887	0,49031
H3	A3 + C3	0,00412	0,78191	0,56731	0,01512	0,48455	0,51345
I1	A3 + B + C1	0,00512	0,67325	0,47592	0,01610	0,50773	0,46882
I2	A3 + B + C2	0,00517	0,71342	0,56291	0,01569	0,51909	0,52015
I3	A3 + B + C3	0,00509	0,68987	0,49817	0,01592	0,49166	0,48986

ANEXO A – OPEN STREET MAP

Open Street Map é um projeto que tem como objetivo a criação e disponibilização de dados espaciais livres. OSM é baseado em uma filosofia de código aberto e segue o modelo que criou a Wikipedia, permitindo que usuários criem, editem, obtenham e usem os dados espaciais livremente em suas aplicações.

A Figura 17 mostra a página inicial do sítio do OSM.

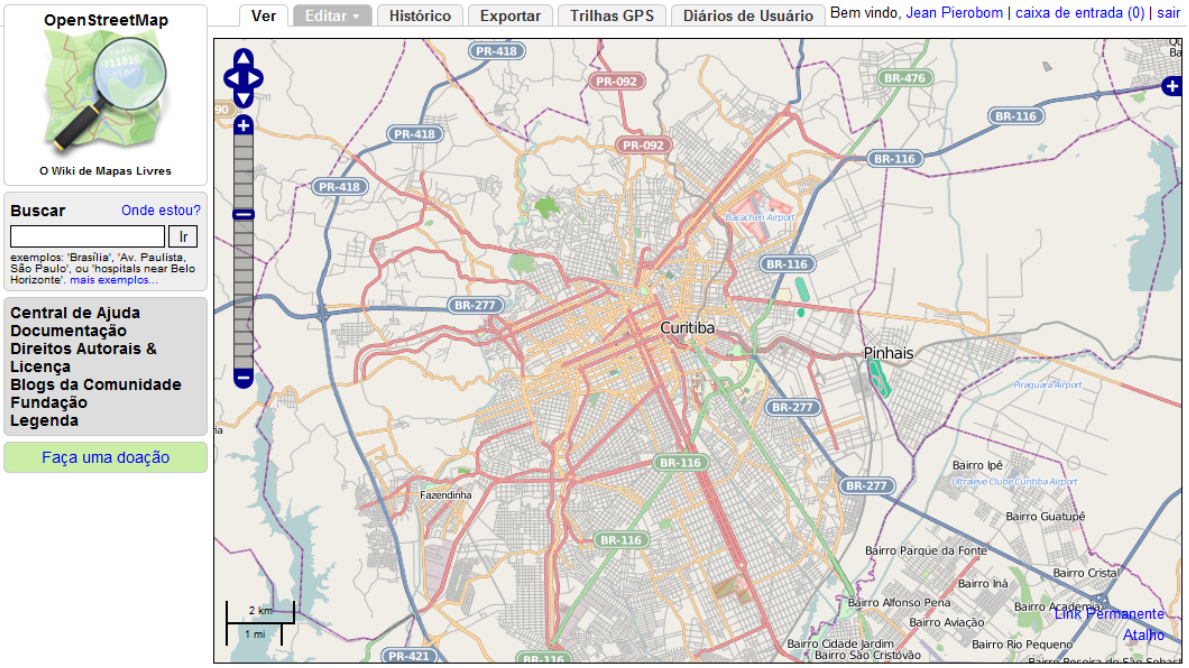

Figura 17: Página inicial do sítio do Open Street Map

Fonte: http://www.openstreetmap.org

O projeto foi fundado no ano de 2004 na University College London (UCL), e em Maio de 2008 já contava mais de 33.000 usuários registrados, sendo que destes, mais de 3.500 contribuem regularmente editando os dados geoespaciais de maneira colaborativa, com a utilização da infraestrutura técnica do OSM. Ao mesmo tempo, cerca de 40 voluntários dedicam o seu tempo para criar e melhorar a infraestrutura do OSM, o que inclui: manter o servidor, desenvolver o software que trata as requisições feitas para o servidor e responde com saídas de dados

cartográficos em diferentes formatos. Há também uma crescente comunidade de desenvolvedores de software que criam aplicações para tornar os dados do OSM disponíveis para uso em novos domínios de aplicação, plataformas de software, e dispositivos de hardware (HAKLAY; WEBER, 2008).

O OSM pode ser integrado com outras aplicações através da Google Maps Javascript API (GOOGLE, 2011). Outra possibilidade é a utilização da OpenLayers JavaScript API (OPENLAYERS, 2011). Além disso, os dados topológicos do OSM também podem ser obtidos para a manipulação local e importação para um sistema gerenciador de bancos de dados geográficos, como o PostGIS (POSTGIS, 2011).

ANEXO B – OSM2PO

O projeto OSM2PO foi criado por Moeller (2011), com o objetivo inicial de fazer um algoritmo de roteamento para o Open Street Map. A Figura 18 mostra a página inicial do sítio do OSM2PO, onde pode ser visualizado um diagrama da sua arquitetura.

Figura 18: Página inicial do sítio do OSM2PO

Fonte: http://osm2po.de/

Desenvolvido em Java, o OSM2PO é disponibilizado gratuitamente, e pode ser utilizado de duas maneiras:

- Software de conversão: pode ser utilizado para converter dados do OSM, sendo capaz de gerar scripts SQL para a criação de bases de dados PostGIS, com estrutura compatível com o pacote de roteamento pgRouting (PGROUTING, 2011);
- Biblioteca de programação: pode ser integrado em outras aplicações através de uma biblioteca de algoritmos de roteamento.

Nesta dissertação, foi utilizada a biblioteca de programação do OSM2PO na construção do software de simulação do PATC/TAP. Experimentos realizados para medir o desempenho do algoritmo de roteamento, em um computador com processador Intel Core 2 Duo 2.40GHZ e 3GB de memória RAM, indicam que o tempo médio de processamento para a identificação de uma rota entre pontos do centro da Cidade de Curitiba/PR, é de 130 milissegundos. A Listagem B.1 apresenta os segmentos do trajeto mais curto da empresa CELEPAR (Latitude: -25.394459 e Longitude: -49.272766) até a UTFPR (Latitude: -25.432827 e Longitude: -49.272835).

Listagem B.1: Segmentos da rota mais curta, partindo da CELEPAR até a UTFPR

```
1  INFO  Loading Graph from D:\Software\osm2po-4.2.11\hh
2  INFO  Loading Vertices from gv_fwd.2po - 719.663 Vertices loaded. - 249.914k
3  INFO  Loading Coordinates from gc_all .2po - 719.663 Coords loaded. - 244.480k
4  INFO  Loading Edges from ge_fwd.2po - 2.063.974 Edges loaded. - 202.118k
5  INFO  Graph is in memory - 202.264k free
6
7  Rota de CELEPAR {-25.394459, -49.272766} até UTFPR {-25.432827, -49.272835}
8  Segmento 0 - Distância (KM): 0,319370 - Seguir por Rua Mateus Leme
9  Segmento 1 - Distância (KM): 0,117987 - Seguir por Rua Mateus Leme
10 Segmento 2 - Distância (KM): 0,044159 - Seguir por Rua Mateus Leme
11 Segmento 3 - Distância (KM): 0,076819 - Seguir por Rua Mateus Leme
12 Segmento 4 - Distância (KM): 0,050496 - Seguir por Rua Mateus Leme
13 Segmento 5 - Distância (KM): 0,015367 - Seguir por Rua Mateus Leme
14 Segmento 6 - Distância (KM): 0,196643 - Seguir por Rua Mateus Leme
15 Segmento 7 - Distância (KM): 0,146026 - Seguir por Rua Mateus Leme
16 Segmento 8 - Distância (KM): 0,163022 - Seguir por Rua Mateus Leme
17 Segmento 9 - Distância (KM): 0,119717 - Seguir por Rua Mateus Leme
18 Segmento 10 - Distância (KM): 0,168912 - Seguir por Rua Mateus Leme
19 Segmento 11 - Distância (KM): 0,067810 - Seguir por Rua Mateus Leme
20 Segmento 12 - Distância (KM): 0,231479 - Seguir por Rua Mateus Leme
21 Segmento 13 - Distância (KM): 0,185171 - Seguir por Rua Mateus Leme
22 Segmento 14 - Distância (KM): 0,189318 - Seguir por Rua Mateus Leme
23 Segmento 15 - Distância (KM): 0,256175 - Seguir por Rua Mateus Leme
24 Segmento 16 - Distância (KM): 0,173997 - Seguir por Rua Mateus Leme
25 Segmento 17 - Distância (KM): 0,118420 - Seguir por Rua Mateus Leme
26 Segmento 18 - Distância (KM): 0,078969 - Seguir por Rua Doutor Roberto Barroso
27 Segmento 19 - Distância (KM): 0,083641 - Seguir por Rua Aristides Teixeira
28 Segmento 20 - Distância (KM): 0,096504 - Seguir por Rua Aristides Teixeira
29 Segmento 21 - Distância (KM): 0,174711 - Seguir por Avenida Candido de Abreu
30 Segmento 22 - Distância (KM): 0,146681 - Seguir por Avenida Candido de Abreu
31 Segmento 23 - Distância (KM): 0,111193 - Seguir por Avenida Candido de Abreu
32 Segmento 24 - Distância (KM): 0,184707 - Seguir por Avenida Candido de Abreu
33 Segmento 25 - Distância (KM): 0,009900 - Seguir por Avenida Candido de Abreu
34 Segmento 26 - Distância (KM): 0,122996 - Seguir por Rua Barão do Serro Azul
35 Segmento 27 - Distância (KM): 0,072028 - Seguir por Rua Barão do Serro Azul
36 Segmento 28 - Distância (KM): 0,105470 - Seguir por Rua Barão do Serro Azul
37 Segmento 29 - Distância (KM): 0,087520 - Seguir por Rua Barão do Serro Azul
38 Segmento 30 - Distância (KM): 0,035044 - Seguir por Travessa Nestor de Castro
39 Segmento 31 - Distância (KM): 0,201240 - Seguir por Travessa Nestor de Castro
40 Segmento 32 - Distância (KM): 0,007254 - Seguir por Travessa Nestor de Castro
41 Segmento 33 - Distância (KM): 0,095710 - Seguir por Travessa Nestor de Castro
42 Segmento 34 - Distância (KM): 0,059374 - Seguir por Alameda Doutor Muricy
43 Segmento 35 - Distância (KM): 0,054104 - Seguir por Alameda Doutor Muricy
44 Segmento 36 - Distância (KM): 0,094667 - Seguir por Alameda Doutor Muricy
45 Segmento 37 - Distância (KM): 0,059458 - Seguir por Alameda Doutor Muricy
46 Segmento 38 - Distância (KM): 0,172791 - Seguir por Alameda Doutor Muricy
47 Segmento 39 - Distância (KM): 0,075129 - Seguir por Rua Emiliano Perneta
48 Segmento 40 - Distância (KM): 0,213148 - Seguir por Rua Desembargador Westphalen
49 Segmento 41 - Distância (KM): 0,154848 - Seguir por Rua Desembargador Westphalen
50 Segmento 42 - Distância (KM): 0,198116 - Seguir por Rua Desembargador Westphalen
51 Segmento 43 - Distância (KM): 0,012396 - Seguir por Rua Desembargador Westphalen
52 Segmento 44 - Distância (KM): 0,142486 - Seguir por Rua Desembargador Westphalen
53 Segmento 45 - Distância (KM): 0,023449 - Seguir por Rua Desembargador Westphalen
54 Distância da Rota (KM): 5,514422
55 Tempo de Processamento: 134 milissegundos
```