UNIVERSIDADE FEDERAL DO PARANÁ

MATHEUS ROSENDO

UM ALGORITMO DE OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS PARA RESOLUÇÃO DE PROBLEMAS COMBINATÓRIOS

CURITIBA
2010

MATHEUS ROSENDO

UM ALGORITMO DE OTIMIZAÇÃO POR NUVEM DE PARTÍCULAS PARA RESOLUÇÃO DE PROBLEMAS COMBINATÓRIOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná, como requisito parcial à obtenção do grau de Mestre.

Orientadora: Profa. Dra. Aurora T. Ramirez Pozo

CURITIBA
2010

Dedico este trabalho a todas as pessoas que me acompanharam de perto durante todo o meu mestrado, em especial, Marlene Maria Rosendo, minha mãe.

Agradeço aos meus pais pelo apoio em todos os sentidos, por me ouvirem em todos os momentos. A minha mãe pelo apoio espiritual e psicológico. Ao meu pai por prestigiar resultados de experimentos mesmo nos horários mais inoportunos. Ao Tiago Buriol, por todos os conselhos e apoio técnico-científico. A Márcia Ferreira (Marcinha) pelo incentivo e ajuda com a carta de intenção para entrar no mestrado. Ao prof. Luis Allan por ter me aceito no programa. A profa. Aurora Pozo por ter me aceito em sua linha de pesquisa, ao prof. Sérgio Scheer por ter me integrado ao grupo de visualização do CESEC, onde trabalhei como bolsista durante o mestrado. E finalmente, aos membros da banca examinadora, profa. Myriam Delgado e prof. Eduardo Spinosa, pelas sugestões que tanto contribuiram para a elaboração da versão final deste trabalho.

Coragem, coragem,
se o que você quer
é aquilo que pensa e faz.
Coragem, coragem,
que eu sei que você pode mais.

Raul Seixas

Resumo

O *Particle Swarm Optimization* (PSO) pertence a uma classe de algoritmos inspirados em comportamentos sociais naturais inteligentes, chamada *Swarm Intelligence* (SI). O algoritmo PSO tem sido aplicado com sucesso na resolução de problemas de otimização contínua, no entanto, o seu potencial em problemas discretos não foi suficientemente explorado. Trabalhos recentes têm proposto a implementação de PSO usando algoritmos de busca local e *Path relinking* com resultados promissores. Este trabalho tem como objetivo apresentar um algoritmo PSO como um meta-modelo que utiliza internamente busca local e *Path relinking*, mas diferentemente das abordagens anteriores, o algoritmo proposto mantém o conceito principal de PSO para a atualização da velocidade da partícula. O trabalho descreve o algoritmo proposto como uma plataforma geral para problemas combinatórios. Tal proposta é validada em duas implementações: uma aplicada ao Problema do Caixeiro Viajante e outra ao Problema da Mochila. As peculiaridades e uma série de experimentos de calibragem de ambos os algoritmos são relatados. Finalmente, a qualidade do algoritmo proposto é testada na comparação com outros PSO discretos da literatura recente e também com outro conhecido algoritmo de metaheurística: o *Ant Colony Optimization* (ACO). Os resultados são encorajadores e reforçam a idéia de que o algoritmo PSO também pode ser competitivo em espaço de busca discreto, assim como levam a crer que a utilização de métodos dependentes do problema pode ser uma excelente alternativa na aplicação de PSO a este tipo de problema.

Abstract

Particle Swarm Optimization (PSO) belongs to a class of algorithms inspired by natural social intelligent behaviors, called Swarm Intelligence (SI). PSO has been successfully applied to solve continuous optimization problems, however, its potential in discrete problems has not been sufficiently explored. Recent works have proposed PSO implementations using local search and Path relinking algorithms with promising results. This paper aims to present a PSO algorithm as a meta-model that uses local search and Path relinking, but differently to the previous approaches, the proposed algorithm maintains the main PSO concept for the update of the velocity of the particle. The work describes the proposed algorithm as a general platform for combinatorial problems. Such proposal is evaluated by two implementations: one applied to the Traveling Salesman Problem and another to the Multidimensional Knapsack Problem. The peculiarities and a set of experiments of tuning for both algorithms are reported. Finally, the quality of the proposed algorithm is tested in comparison to others discrete PSO of recent literature and also with other known metaheuristic algorithm: Ant Colony Optimization (ACO). The results are encouraging and reinforce the idea that the PSO algorithm can also be competitive in discrete search space, and suggest that the use of problem dependent methods can be an excellent alternative in the application of PSO to this type of problem.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ACO	- *Ant Colony Optimization*, 18
CB	- Custo benefício, 61
CO	- *Combinatorial Optimization*, 12
COP	- Combinatorial Optimization Problems, 12
DP	- Desvio Padrão, 53
DPSO	- *Discrete Particle Swarm Optimization*, 74
ENS	- *Expanding Neighborhood Search*, 35
FIPA	- Função para Índices Pseudo Aleatórios, 62
FLS	- *Fast local search*, 76
GA	- *Genetic Algorithm*, 76
GCC	- Grafo completamente conectado, 25
GRASP	- *Greedy Randomized Adaptive Search Procedure*, 35
INV	- *Inversion Procedure*, 35
LK	- *Lin-Kernighan*, 35
LRP	- *Location-Routing Problem*, 35
LS	- *Local Search*, 61
PL	- Programação Linear, 82
POO	- Programação Orientada a Objeto, 73
PSO	- *Particle Swarm Optimization*, 12
PSOPR	- *Particle Swarm Optimization with Path Relinking*, 37
S-PSO	- *Set-Based PSO*, 35
SI	- *Swarm Intelligence*, 12
TSPLIB	- *Traveling Salesman Problem Library*, 52
VO	- Valor Ótimo, 53

Sumário

Capítulo 1

Introdução

A presente dissertação de mestrado tem como foco o estudo da aplicação de um algoritmo de Inteligência de Enxames denominado Otimização por Nuvem de Partículas - *Particle Swarm Optimization* (PSO), uma técnica de otimização desenvolvida nos anos 90 (KENNEDY; EBERHART, 1995) baseada na análise do comportamento inteligente de revoadas de pássaros.

Inteligência de enxames - *Swarm Intelligence* (SI) - é uma área de estudo bastante abrangente que engloba uma série de algoritmos que simulam comportamentos sociais inteligentes existentes na natureza com o objetivo de encontrar soluções otimizadas para problemas computacionalmente complexos.

Um dos tipos mais conhecidos de problemas complexos são os de Otimização Combinatória - *Combinatorial Optimization* (CO), foco de estudo deste trabalho. Estes problemas têm sido amplamente estudados por diversos pesquisadores de todo o mundo, utilizando algoritmos dos mais variados tipos incluindo SI. Apesar disso, existem poucas aplicações de algoritmos PSO a problemas de otimização combinatória, visto que este tipo de algoritmo foi projetado inicialmente para problemas cujo espaço de busca é contínuo e não discreto como é o caso dos problemas de otimização combinatória (COP).

Em 1997, os criadores da PSO, Kennedy e Eberhart, desenvolveram a primeira versão deste algoritmo para problemas CO. Posteriormente vários trabalhos relacionados

foram desenvolvidos, tais como Hu, Eberhart e Shi (2003), Clerc (2004), Goldbarg, Souza e Goldbarg (2006) e Liao, Tseng e Luarn (2007). Na grande maioria destes trabalhos, apenas um problema foi utilizado para implementação do algoritmo proposto. Em alguns, o algoritmo não foi capaz de gerar resultados competitivos em relação a outros algoritmos para otimização combinatória. Além disso, em vários destes trabalhos apenas um dos três componentes de velocidade é aplicado à partícula a cada iteração, diferentemente do algoritmo original proposto por Kennedy e Eberhart.

O objetivo deste trabalho é desenvolver um meta-modelo baseado no PSO aplicável a Problemas Combinatórios de uma forma geral mantendo as características originais do PSO, descritas por Kennedy e Eberhart, no que se refere à aplicação da velocidade das partículas. Tal modelo prevê a utilização de técnicas de busca local e *Path relinking* internamente.

Como forma de validação do modelo proposto, o trabalho contempla duas implementações do algoritmo, uma para o Problema do Caixeiro Viajante e outra para o Problema da Mochila. A sensibilidade de cada uma destas versões em relação aos coeficientes/parâmetros é relatada, juntamente com uma série de comparações com trabalhos recentes da literatura.

Apesar de propor um modelo geral para COPs, o trabalho encoraja a utilização de técnicas dependentes do problema como forma de explorar melhor as peculiaridades de cada problema e assim chegar a melhores resultados. Tal afirmação é baseada nos resultados obtidos a partir dos experimentos de comparação de ambas as versões do algoritmo com trabalhos relacionados.

1.1 Organização do trabalho

O capítulo 2 relata uma breve introdução sobre Inteligência de Enxames e apresenta uma revisão bibliográfica a respeito do algoritmo tratado neste trabalho: Otimização por Nuvem de Partículas. Ainda neste capítulo, são apresentados os conceitos, a origem, as

terminologias e também as variações e peculiaridades deste algoritmo.

No capítulo 3 é apresentada uma visão geral sobre Otimização Combinatória, alguns conceitos primordiais da área como Ótimos Locais e Globais, além de uma descrição de alguns dos principais problemas de otimização combinatória existentes no meio acadêmico.

No capítulo 4 são apresentados alguns trabalhos relacionados da literatura. O objetivo deste capítulo é mostrar as características específicas provenientes da adaptação de algoritmos PSO a este tipo de problema.

O capítulo 5 trata das características gerais do modelo proposto como um algoritmo PSO discreto padrão para problemas combinatórios. Seguido pelos capítulos 6 e 7, nos quais são relatadas as características específicas da implementação do modelo para dois problemas combinatórios distintos. Também é apresentada uma análise de sensibilidade de cada implementação do algoritmo.

Finalmente, no capítulo 8, a qualidade do modelo proposto é avaliada através de vários experimentos de comparação com trabalhos relacionados da literatura (CLERC, 2004; MACHADO; LOPES, 2005; CHEN et al., 2009; GOLDBARG; SOUZA; GOLDBARG, 2006; KONG; TIAN, 2006; DORIGO; GAMBARDELLA, 1997; HEMBECKER; LOPES; GODOY, 2007), seguido pelas conclusões finais no capítulo 9.

Capítulo 2

Inteligência de Enxames

Na natureza várias espécies se beneficiam da sociabilidade, pois a vida em grupos sociais aumenta a probabilidade de acasalamento, facilita a caça e coleta de alimentos, reduz a probabilidade de ataque por predadores, permite a divisão de trabalho, etc (ZUBEN; ATTUX, 2008). O termo "enxame" (ou coletivo) é utilizado de forma genérica para se referir a qualquer coleção estruturada de agentes capazes de interagir. O exemplo clássico é um enxame de abelhas. Entretanto, a metáfora de enxame pode ser estendida a outros sistemas com uma arquitetura similar como uma colônia de formigas, uma revoada de pássaros ou até mesmo um engarrafamento, no qual os agentes são carros; uma multidão é um enxame de pessoas, um sistema imunológico é um enxame de células e moléculas, e uma economia é um enxame de agentes econômicos (ZUBEN; ATTUX, 2008).

Baseados nas vantagens que certos indivíduos possuem ao viver coletivamente no mundo real, pesquisadores desenvolveram ferramentas computacionais para a solução de problemas e estratégias de coordenação e controle de robôs (ZUBEN; ATTUX, 2008). Surge assim o termo "*Swarm Intelligence*" (SI), proposto por Beni e Wang no fim da década de 1980, quando se referia a sistemas robóticos compostos por uma coleção de agentes simples em um ambiente interagindo de acordo com regras locais. De acordo com White e Pagurek (1998), inteligência de enxames ou inteligência coletiva é uma propriedade de sistemas compostos por agentes não (ou pouco) inteligentes e com capacidade

individual limitada, capazes de apresentar comportamentos coletivos inteligentes.

Inteligência de Enxames é uma técnica de Inteligência Computacional que estuda o comportamento coletivo de agentes descentralizados. Tais agentes, fazendo uma analogia à natureza, seriam indivíduos integrantes de uma população que interagem localmente uns com os outros e também com o seu meio ambiente. Segundo Bonabeau, Dorigo e Theraulaz (1999), a inteligência de enxames inclui qualquer tentativa de projetar algoritmos ou dispositivos distribuídos de solução de problemas inspirados no comportamento coletivo de insetos sociais e outras sociedades animais.

Em vários enxames naturais, como as já citadas abelhas e formigas, existe um indivíduo chamado rainha que é de suma importância para o enxame como um todo devido a sua função reprodutiva. Mas apesar do nome, que remete a soberania, a rainha não exerce qualquer autoridade sobre o grupo, não havendo assim, uma estrutura hierárquica no enxame.

Embora normalmente não haja controle centralizado ditando o comportamento destes indivíduos, interações locais entre eles muitas vezes causam um padrão global de comportamento que surge de baixo para cima, este padrão de comportamento é chamado Emergência - "*Emergence*" e pode ser encontrado em vários sistemas de organismos vivos, além de cidades, furacões e até mesmo nos meios de comunicação, de acordo com Steven Johnson em seu livro Emergence (JOHNSON, 2001). Para Johnson Emergência é o que acontece quando várias entidades independentes de baixo nível conseguem criar uma organização de alto nível sem ter estratégia ou autoridade centralizada. O padrão de comportamento destas entidades é chamado "*bottom up*" (de baixo para cima), pois cada indivíduo ou entidade através de suas atividades acabam inconscientemente, determinando o comportamento do coletivo formado por ele e seus semelhantes, sem que haja um líder ou uma entidade centralizadora.

Ainda citando as formigas como exemplo, a complexidade da vida social destes insetos é maximizada pela existência de várias tarefas pré-definidas que são de suma importância para o grupo como coleta, produção de alimentos, proteção, etc. Tais tarefas

precisam ser executadas de forma sincronizada e muitas vezes ininterrupta em prol da sobrevivência do enxame. E mesmo sem um comando, a sociedade se mantém organizada e funcional.

Baseado nestes sistemas naturais de comportamento emergente, vários sistemas artificiais de otimização têm sido desenvolvidos e aprimorados com o passar do tempo. Eles se enquadram dentro de um conjunto mais amplo de técnicas de busca computacionais chamado metaheurística. Segundo Stützle (1998) metaheurísticas são tipicamente estratégias de alto nível que orientam uma heurística base, e mais específica em relação ao problema, para aumentar seu desempenho. Analisando a etimologia da palavra, o prefixo "*meta*", do grego, significa "além" ou "um nível mais elevado" e heurística, da palavra grega heuriskein, significa "encontrar".

Ao estudar classes de algoritmos de metaheurística, como é o caso dos algoritmos de SI, dois importantes conceitos são amplamente utilizados: intensificação (*exploitation*) e diversificação (*exploration*). A intensificação é a procura mais exaustiva de uma região do espaço de busca já conhecida na qual se espera encontrar boas soluções, ao passo que a diversificação é a mudança do foco da busca para regiões ainda não exploradas (BLUM; ROLI, 2003). Uma boa metaheurística deve obter um equilíbrio, comumente chamado "*trade off*", entre intensificação e diversificação.

A próxima seção descreve o comportamento do algoritmo de Otimização por Nuvem de Partículas, além de suas principais características e terminologias a respeito.

2.1 Otimização por Nuvem de Partículas

Desenvolvido pelo psicólogo social James Kennedy e o engenheiro eletricista Russel Eberhart em 1995 (KENNEDY; EBERHART, 1995), PSO é uma técnica de otimização que surgiu a partir da análise do comportamento social de revoadas de pássaros. No algoritmo cada indivíduo é chamado de partícula e se comporta como um pássaro do bando a procura de alimento ou do local de seu ninho. Para atingir seu objetivo a partícula utiliza o apren-

dizado adquirido por suas próprias experiências e também o aprendizado do bando (ou enxame). Por ter um comportamento emergente, em uma nuvem de partículas não existe um controle central. Cada partícula atua e toma decisões com base em informações locais e globais, como nas demais técnicas de Vida Artificial (VESTERSTRØM; RIGET, 2002). Mais abstratamente, conforme definido por Vesterstrøm:

> Uma partícula seria um estado de pensamento, representando nossas crenças e atitudes. Uma mudança de pensamento, dessa forma, corresponderia a um movimento da partícula. Ajustamos nossas crenças com base nos outros, avaliamos os estímulos do ambiente, comparamos com as nossas crenças, e imitamos o estímulo.

Assim, avaliação, comparação e imitação são importantes propriedades do comportamento social humano e, por isso, são a base para a nuvem de partículas, que utiliza esses conceitos na adaptação a mudanças no ambiente e na resolução de problemas complexos (KENNEDY; EBERHART, 2001).

O PSO é considerado um sistema multiagente para resolver problemas de otimização complexos. Uma característica marcante deste algoritmo é a forma como estes agentes, ou partículas, se relacionam entre si. Eles apresentam um comportamento colaborativo, simulando um compartilhamento de experiências e cooperando entre si em busca das melhores soluções. Assim como PSO, outro algoritmo de SI também possui esta característica: a Otimização por Colônia de Formigas - *Ant Colony Optimization* (ACO), utilizado como comparação com o algoritmo deste trabalho no capítulo 8.1. Mas nem todos os algoritmos de metaheurística possuem esta característica no relacionamento entre os agentes da população. No caso dos algoritmos de computação evolutiva, por exemplo, o comportamento regente é justamente ao contrário, os agentes competem entre si para perpetuar as suas próprias características para a próxima iteração, ou geração, simulando assim a teoria da evolução das espécies de Darwin (1872).

No algoritmo PSO as possíveis soluções, chamadas partículas, percorrem o espaço do problema, seguindo as melhores posições encontradas até o momento pelas partículas do bando. Estas melhores posições, as quais as partículas procuram seguir, podem ser classificadas em três grupos distintos: a melhor posição encontrada por ela mesma até

o momento, chamada P_{best}; a melhor posição encontrada pelas partículas vizinhas a ela, chamada L_{best}; e a melhor posição encontrada por toda a população levando em conta todas as partículas, chamada G_{best} ou posição de melhor valor global. Como em todo problema de otimização baseada em populações, ao final da execução a melhor ou as melhores soluções, de acordo com uma função objetivo, são apresentadas como resultado.

Conforme definido por Carvalho (2008) no PSO cada partícula representa uma possível solução e é representada como uma posição no espaço de estados. A ideia da PSO é executar um conjunto de operadores e movimentar cada partícula para regiões promissoras no espaço de busca. A cada iteração as partículas são alteradas como se fossem movimentadas num espaço n-dimensional e, assim, um novo conjunto de soluções é obtido. Isso ocorre através da aplicação de suas respectivas velocidades. A cada iteração a velocidade de cada partícula é ajustada. O cálculo da velocidade é baseado na melhor posição encontrada pela vizinhança da partícula e pela melhor posição encontrada pela própria partícula.

2.1.1 Terminologias em algoritmos PSO

Tendo como base as definições contidas em Reyes-Sierra e Coello (2006), a seguir são descritos os principais termos utilizados no PSO:

- **Swarm ou Enxame**: População/Nuvem do algoritmo;
- **Partícula**: Indivíduo da população ou enxame. Cada partícula possui uma determinada posição no espaço de busca do problema e esta posição representa uma solução em potencial para o problema tratado;
- **Velocidade (Vetor)**: Responsável por comandar o processo de otimização, a velocidade de uma partícula determina a direção na qual ela se movimentará, com objetivo de melhorar sua posição atual. Ela é atualizada, durante as iterações do algoritmo, de acordo com as melhores posições, sendo que esta atualização varia de acordo com a estratégia utilizada;
- P_{best}: Melhor posição já alcançada pela partícula;
- L_{best}: Melhor posição já alcançada por uma partícula pertencente à vizinhança de uma determinada partícula;
- G_{best}: Melhor posição já alcançada por uma partícula em toda população.
- **Líderes**: Partículas da população que possuem os melhores valores da função ob-

jetivo para o problema;

- **Coeficiente de inércia** (w): Usado para controlar a influência dos valores anteriores da velocidade no cálculo da velocidade atual;
- **Fator de individualidade** ($c1$): Influencia na atração que a partícula tem em direção à melhor posição já encontrada por ela mesma (P_{best});
- **Fator de sociabilidade** ($c2$): Influencia na atração que a partícula tem em direção à melhor posição já encontrada por qualquer partícula vizinha a ela (L_{best});
- **Topologia de Vizinhança**: Determina o conjunto de partículas usado como vizinhança de uma determinada partícula.

2.1.2 O Algoritmo PSO

Em PSO a população é chamada nuvem. Uma nuvem é um número de partículas que se movem em um espaço n-dimensional, dentro de um subespaço de busca S (VESTERSTRØM; RIGET, 2002).

Cada partícula p, numa dada iteração t, tem uma posição em R^n, $\vec{X}(t)$ e uma velocidade de deslocamento nesse espaço, $\vec{V}(t)$. Possui também uma memória contendo sua melhor posição já alcançada, P_{best}, e a melhor posição já alcançada pelas partículas vizinhas a p, L_{best}, que é determinada pela topologia de vizinhança implementada. É importante ressaltar que $\vec{X}(t)$, $\vec{V}(t)$, P_{best}, L_{best}, são vetores n-dimensionais, sendo n determinado pelo problema que está sendo atacado pelo algoritmo. Se este fosse, por exemplo, o conhecido problema do Caixeiro Viajante, n seria o número total de cidades que devem ser visitadas como apresentado em Wang et al. (2003).

A posição de cada partícula representa uma solução potencial para o problema. O objetivo do algoritmo é movimentar essas partículas a fim de fazer com que elas se tornem soluções ótimas para o problema (TORÁCIO, 2008). Para essa movimentação, uma partícula tem três opções chamadas neste trabalho de M1, M2, M3:

- Seguir seu próprio caminho (M1);
- Seguir em direção a sua melhor posição já encontrada (M2);
- Seguir em direção à melhor posição da vizinhança (M3).

Toda a movimentação das partículas ocorre de acordo com a qualidade das mesmas,

como a posição de uma partícula equivale a uma solução em potencial para o problema, o objetivo do algoritmo é guiar a partícula pelo espaço de busca rumo às posições de melhor qualidade já conhecidas em busca de soluções consideradas ótimas para o problema. Esta qualidade, ou nível de aptidão, é calculada através de uma função de avaliação (*fitness*), responsável por informar o quão boa é uma determinada posição de uma partícula num dado instante.

Algorithm 1 Pseudocódigo do algoritmo de metaheurística PSO

```
1: Atribui parâmetros
2: for i = 0 até tamanhoEnxame do
3:    Inicia X⃗^i com uma solução aleatória para o problema
4:    Inicia V⃗^i com uma velocidade aleatória < V_MAX
5:    P⃗^i_best ← L⃗^i_best ← X⃗^i
6: end for
7: while Não atingir condição de parada do
8:    for i = 0 até tamanhoEnxame do
9:       atualiza V⃗^i (equação 2.2)
10:      X⃗^i = X⃗^i + V⃗^i
11:      P⃗^i_best ← melhor entre X⃗^i e P⃗^i_best
12:      L⃗^i_best ← melhor entre P⃗^i_best e L⃗^i_best
13:   end for
14: end while
15: return melhor L⃗_best
```

Conforme se pode observar no algoritmo 1, pseudocódigo do PSO, a nuvem é iniciada no tempo $t = 0$, espalhando-se as partículas aleatoriamente no espaço S. Para cada partícula, suas posições são iniciadas ($\vec{X}(0) = P_{best}(0) = L_{best}(0)$) juntamente com suas velocidades $\vec{V}(0)$. Feito isso, inicia-se o processo iterativo. A posição de cada partícula é alterada num determinado tempo t adicionando a velocidade à posição atual da partícula de acordo com a equação:

$$\vec{X}(t+1) = \vec{X}(t) + \vec{V}(t+1) \tag{2.1}$$

A velocidade da partícula numa determinada iteração t é baseada na melhor posição já alcançada pela partícula ($P_{best}(t)$) e pela melhor posição alcançada pelos vizinhos. Como apresentado anteriormente, a definição da melhor solução da vizinhança ($L_{best}(t)$)

depende da topologia de vizinhança implementada, assunto da próxima subseção.

A função de atualização da velocidade, no tempo $t+1$, é definida por:

$$\vec{V}(t+1) = w * \vec{V}(t) + c1 * \phi1 * \left(\vec{P}_{best}(t) - \vec{X}(t)\right) + c2 * \phi2 * \left(\vec{L}_{best}(t) - \vec{X}(t)\right) \quad (2.2)$$

Onde w é um coeficiente chamado na literatura de inércia da partícula. Ela determina o quanto a velocidade anterior influencia na velocidade atual, equivalendo a autoconfiança da partícula (TORÁCIO, 2008). Um alto valor para w faz com que a partícula procure seguir mais o seu próprio caminho ao invés de optar por seguir as melhores posições já alcançadas por ela mesma e por suas vizinhas.

Já os coeficientes $c1$, $c2$, $\phi1$ e $\phi2$ determinam a influencia de $\vec{P}_{best}(t)$ e $\vec{L}_{best}(t)$ sobre a partícula p. Onde $\phi1$ e $\phi2$ são coeficientes aleatórios que variam de 0 a 1, enquanto $c1$ e $c2$ são pré-configurados e influenciam o quanto a partícula vai tender a seguir cada uma das duas opções. Caso $c1$ seja significativamente maior que os coeficientes $c2$ e w, a partícula tenderá a ir de encontro à posição do $\vec{P}_{best}(t)$, por outro lado, caso $c2$ seja o coeficiente de maior valor, ela tenderá à posição do $\vec{L}_{best}(t)$.

Para que a velocidade não fique muito alta fazendo com que a partícula se disperse demasiadamente do restante da nuvem, pode ser definido um parâmetro de velocidade máxima (V_{MAX}) que limita o valor da velocidade da partícula. Após a atualização da velocidade e da posição de todas as partículas o processo é repetido nas próximas iterações ate o final da execução do algoritmo.

2.1.3 Topologias de Vizinhança

Como apresentado anteriormente, no algoritmo PSO um conjunto de possíveis soluções, representadas através de partículas, movimenta-se pelo espaço de busca. Esse movimento é influenciado pela própria história da partícula e por sua vizinhança. Para cada partícula

p, sua vizinhança ($N(p)$) consiste no conjunto de todas as partículas vizinhas da partícula p. Dada uma partícula i, se $i \in N(p)$, então p conhece i e pode tomar decisões com base nas posições da partícula i (VESTERSTRØM; RIGET, 2002).

Existem dois tipos distintos de vizinhança em PSO na literatura, a física e a social (SOUZA, 2006):

- A vizinhança física (ou geográfica) leva em conta as distâncias entre as soluções a partir de uma função de cálculo de distância pré-definida. As distâncias são computadas a cada passo e tomadas as k arestas mais próximas como vizinhas.
- A vizinhança social leva em conta os relacionamentos. Para cada partícula, sua vizinhança é definida com uma lista de partículas. Portanto, não é necessário calcular distâncias, sendo uma grande vantagem para alguns casos, particularmente para espaços discretos. Pode notar também que, se o processo converge, uma vizinhança social tende a se tornar uma vizinhança física.

Como o trabalho aborda problemas de espaço de busca discreto, nesta seção são apresentadas, baseado em Reyes-Sierra e Coello (2006) e Carvalho (2008), algumas das principais topologias sociais de vizinhança da literatura. Em todas as figuras que seguem para exemplificação das topologias (2.1, 2.2, 2.3, 2.4), cada nó do grafo representa uma partícula e cada aresta representa um relacionamento de vizinhança entre duas partículas.

Grafo vazio: Nesta vizinhança a partícula está conectada somente a ela mesma. Assim, não há influência das demais partículas no seu movimento.

Melhor local: Nesta vizinhança a partícula está conectada a k partículas. Assim, o líder é definido como sendo a melhor partícula da vizinhança k, L_{best}. Equivalente a topologia de anel se $k = 2$, na qual cada partícula é influenciada somente por seus dois vizinhos imediatos (figura 2.1), e equivalente ao grafo completamente conectado (figura 2.4) caso k seja igual ao número total de partículas. Efetua poucas operações para o cálculo do movimento das partículas, porem esse movimento é influenciado por um número pequeno de partículas.

Estrela: Nesta topologia todas as partículas estão conectadas somente a uma partícula, conhecida por partícula focal (figura 2.2). A partícula focal compara o desempenho de todas as partículas da população e efetua o seu movimento de acordo com a

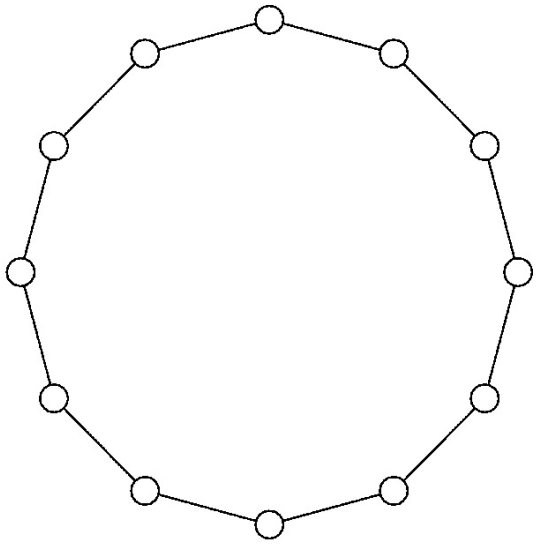

Figura 2.1: Topologia de vizinhança em forma de anel
Fonte: http://aimotion.blogspot.com

melhor partícula. As demais partículas se movimentam de acordo com a partícula focal, que propaga a escolha da melhor solução para as demais partículas. Esta topologia consegue produzir um movimento com a influência da melhor partícula da população sem executar muitas operações, porem a propagação da melhor posição depende da posição da partícula focal.

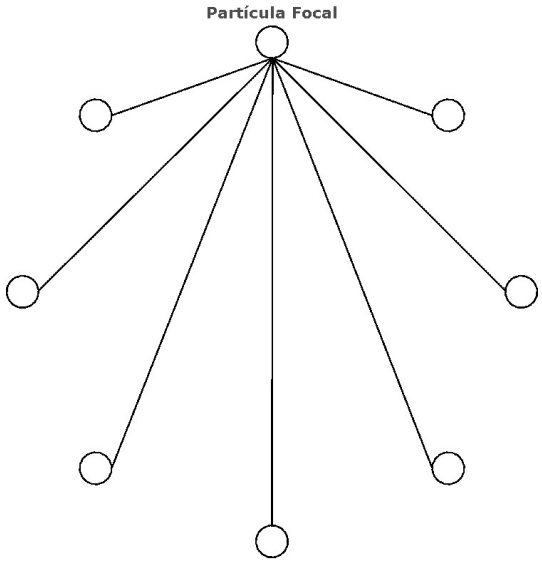

Figura 2.2: Topologia de vizinhança em forma de estrela
Fonte: O autor (2010)

Árvore: Nesta topologia todas as partículas são organizadas no formato de árvore e cada nó desta árvore possui apenas uma partícula, figura 2.3. Cada partícula é

influenciada por sua melhor posição até o momento (P_{best}) e pela melhor posição da partícula que está logo acima dela na árvore (Partícula Pai - *Parent*). Se uma partícula em um nó filho encontrar uma melhor posição no espaço de estados que seu pai, é feita uma troca de posição entre essas duas partículas. Esta topologia propicia um acesso rápido às melhores soluções do problema, porém a estrutura dinâmica da árvore torna a busca mais complexa.

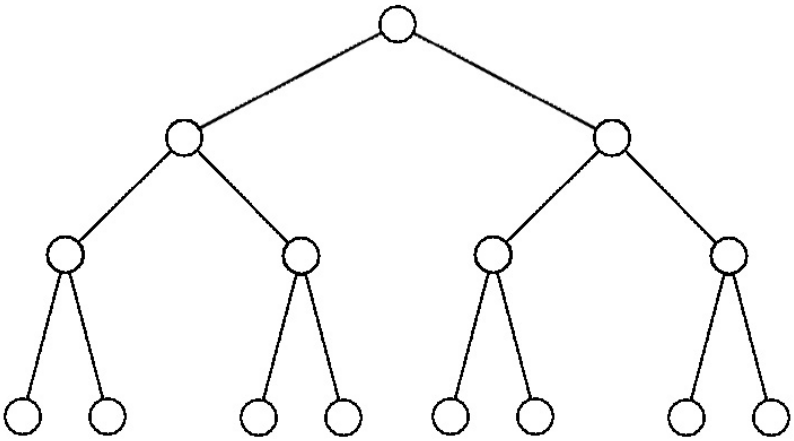

Figura 2.3: Topologia de vizinhança em forma de árvore
Fonte: O autor (2010)

Grafo completamente conectado (GCC): Esta topologia é o oposto do gráfico vazio. Ela conecta todos os membros do enxame uns aos outros. Cada partícula usa o seu histórico de experiências para seguir a sua melhor posição até o momento (P_{best}), mas, além disso, ela utiliza a melhor posição adquirida até o momento levando em conta todas as partículas do enxame (G_{best}). Nesta topologia toda a população é considerada vizinha de cada uma das partículas, de forma que cada uma seja influenciada pela partícula de melhor posição até o momento, dessa forma, $L_{best} = G_{best}$ na equação 2.2.

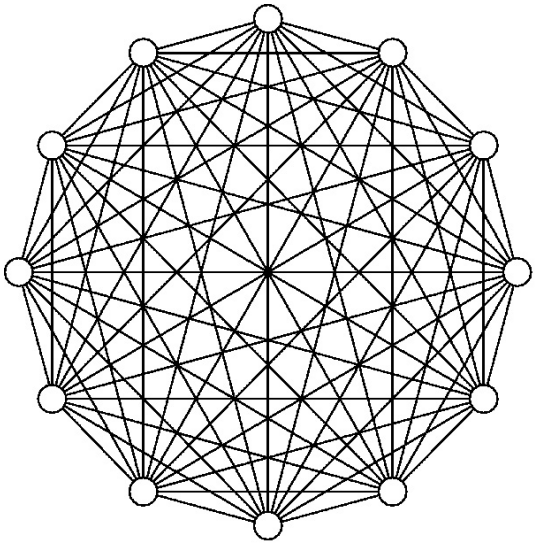

Figura 2.4: Topologia de vizinhança em forma de grafo completamente conectado
Fonte: http://aimotion.blogspot.com

Capítulo 3

Otimização Combinatória

Muitos problemas de otimização, de importância tanto prática quanto teórica, consistem na busca de uma "melhor" configuração de um conjunto de variáveis para alcançar determinados objetivos. Eles se dividem naturalmente em duas categorias: aqueles em que as soluções são codificadas com variáveis de valores reais (otimização contínua), e aqueles em que as soluções são codificadas com variáveis discretas (BLUM; ROLI, 2003). Dentre a segunda categoria citada, encontra-se uma classe de problemas chamados de Otimização Combinatória - *Combinatorial Optimization* (CO).

Em todo COP existe uma função objetivo a ser otimizada. Esta função pode ser considerada de maximização ou de minimização. Os conceitos que se aplicam a funções objetivo de minimização e maximização são os mesmos. Por exemplo, se num determinado problema, o objetivo é diminuição de custos, a função objetivo será de minimização, ou seja, será buscada a solução com o "Menor" custo possível. Caso contrário, por exemplo, num problema cujo objetivo seja encontrar a solução com o "Maior" ganho possível, a função objetivo será de maximização.

Papadimitriou e Steiglitz (1982) se referem à solução do problema de CO como um objeto que é geralmente um número inteiro, um subconjunto, uma permutação, ou a estrutura de um grafo. Este objeto é comumente chamado de solução do problema e pode ser considerado um *ótimo global* ou *ótimo local* dependendo da posição que essa

solução se encontre no espaço de busca. Na figura 3.1 é apresentada uma ilustração tridimensional do espaço de busca, na qual cada ponto preto no gráfico representa uma solução, aquelas localizadas nas áreas vermelhas seriam as melhores soluções do espaço de busca do problema, assim, aquelas pertencentes à área A são as ótimas globais do problema exemplo e aquelas da área B, ótimas locais.

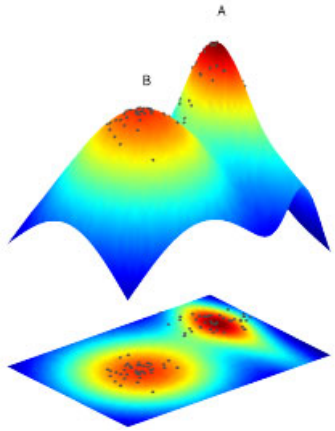

Figura 3.1: Representação gráfica tridimensional de soluções ótimas globais e locais
Fonte: http://www.esteco.com/robust_design.jsp

De acordo com Blum e Roli (2003) um problema de CO $P = (S, f)$ pode ser definido por:

- Um conjunto de variáveis $X = \{x_1, \ldots, x_n\}$;
- Os domínio das variáveis $D_1, \ldots, D_n$;
- Restrições entre as variáveis;
- Uma função objetivo f a ser minimizada, onde $f : D_1 \times \cdots \times D_n \rightarrow \Re^+$.

O conjunto de todas as possíveis atribuições viáveis é definido por:

$$S = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \,|v_i \in D_i\} \tag{3.1}$$

Sendo que s satisfaz todas as restrições. S é normalmente chamado de espaço de busca, de modo que cada elemento do conjunto é considerado uma solução candidata. Para resolver um problema de otimização combinatória é necessário encontrar uma solução $s^* \in S$, tal que o valor da função objetivo para s^* seja o menor possível, isto é, $f(s^*) \leq$

$f(s) \forall s \in S$. A solução s^* é chamada de ótimo global de (S, f) e o conjunto $S^* \subseteq S$ é chamado de conjunto de soluções ótimas globais (BLUM; ROLI, 2003).

Tendo como base as definições acima é possível concluir que uma das principais características que diferem os problemas de CO dos demais problemas de otimização é em relação ao domínio das variáveis. O domínio de uma variável representa o conjunto de possíveis valores discretos que podem ser atribuídos a esta variável. Além disso, pode haver restrições em relação às variáveis. No caso de problemas de permutação, como o problema do caixeiro viajante por exemplo, nenhum valor pode se repetir, ou seja, neste tipo de problema um mesmo valor não pode ser atribuído a duas ou mais variáveis de uma mesma solução.

Além disso, pode haver regras mais específicas como, por exemplo: num determinado problema nenhuma solução pode ter o valor z atribuído a uma variável precedente de outra variável com o valor y, assim, toda solução que tivesse a estrutura $s = \{\ldots, (x_i, z), (x_{i+1}, y), \ldots\}$ seria considerada uma solução não viável para este problema.

3.1 Problemas Combinatórios

Esta seção descreve alguns problemas conhecidos de otimização combinatória. A definição destes problemas será a base para o capítulo 4 no qual são apresentados trabalhos relacionados que os utilizaram em suas abordagens.

Em complexidade computacional, existem classificações para problemas dos mais variados tipos. A ideia desta área de estudo é analisar e categorizar os problemas de acordo com a sua complexidade para ser resolvido, ou seja, a quantidade de recursos computacionais necessários, como tempo e memória, para se alcançar a solução esperada. Os problemas citados a seguir pertencem a uma categoria cuja complexidade aumenta exponencialmente de acordo com o tamanho do problema. Isso porque um simples aumento na instância do problema pode gerar uma explosão combinatória tornando o problema

intratável, ou seja, impossível de gerar todas as possíveis soluções para depois compará-las e então chegar à solução exata para o problema. Para este tipo de problema não existem algoritmos eficientes a ponto de encontrar uma solução exata para o problema dentro de um tempo aceitável, ou polinomial. A seguir serão apresentados alguns exemplos deste tipo de problema.

O Problema de Chão de Fábrica é do tipo escalonamento. Proveniente da indústria, consiste em uma simulação de processos concorrentes que ocorrem nas fabricas. De uma forma geral: uma série de tarefas devem ser executadas numa determinada ordem de forma a minimizar os custos envolvidos no processo.

O Problema das N-Rainhas consiste basicamente em encontrar todas as combinações possíveis de N rainhas num tabuleiro de xadrez de dimensão N por N tal que nenhuma das rainhas esteja em posição de ataque sobre qualquer outra.

Sem dúvida um dos problemas de otimização combinatória mais conhecidos e estudados na literatura é o Problema do Caixeiro Viajante - *Traveling Salesman Problem* (TSP). Neste problema uma série de cidades deve ser visitada pelo caixeiro viajante e este, por sua vez, deve procurar executar o trajeto mais curto de forma a visitar todas as cidades. Este problema ainda se divide em dois tipos distintos: simétrico e assimétrico. No primeiro tipo, a distância, ou custo, entre as cidades independe da direção, enquanto no segundo tipo este custo é variável.

O problema é definido por Chen et al. (2009) da seguinte forma: dado um grafo completo ponderado $G = (N, A)$, onde N é o conjunto de nós e A é o conjunto de arcos. $n = |N|$ é o número de nós. Para cada arco $(j, k) \in A$ é atribuído um comprimento d_{jk}. No TSP simétrico, (j, k) é o mesmo que (k, j) e, portanto, $d_{jk} = d_{jk}$ se mantém para todos os arcos. Já no TSP assimétrico, d_{jk} depende da direção do arco (j, k) e, assim, d_{jk} pode não ser igual a d_{kj}. O objetivo do TSP é encontrar um comprimento mínimo de circuito hamiltoniano. Tal circuito, ou ciclo, é definido como um caminho em um grafo onde cada nó do grafo é visitado uma única vez, terminando no nó de início.

Assim um problema com n cidades tem seu espaço de busca definido pela per-

mutação:

$$S = (n-1)!/2$$

Onde S é o número de possíveis rotas que podem ser percorridas, o que resulta em um número formidável mesmo para poucas cidades. Para $n = 30$, por exemplo, há um total de 4,42 X 10^{30} rotas distintas. Com um computador capaz de analisar um milhão de rotas por segundo, a busca exaustiva levaria o equivalente a 10^7 vezes a idade do universo (TANOMARU, 1995).

Christofides et al. (1979) formularam o TSP como um problema de programação 0-1 sobre um grafo $G(N,A)$, como se segue:

$$\begin{array}{l} \text{maximizar} \quad f(x) = \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \\ \text{tal que} \quad \sum_{i=1}^{n} x_{ij} = 1; \;\; \sum_{j=1}^{n} x_{ij} = 1 \;\; e \;\; \sum_{i=1}^{n} x_{ij} \leq |S| - 1 \\ \text{onde} \quad x_{ij} = \begin{cases} 1, & \text{se arco } (i,j) \in A \text{ escolhido para integrar a solução} \\ 0, & \text{caso contrário} \end{cases} \end{array} \tag{3.2}$$

Tal que S é um subgrafo de G, em que $|S|$ representa o número de vértices deste subgrafo. Nesta formulação está implícito que não existe x_{ii} e tem-se $n(n-1)$ variáveis inteiras 0-1.

Outro COP bastante difundido é o Problema da Mochila. Basicamente neste problema a situação dada é a seguinte: existem vários objetos com valores e pesos diferentes os quais se deseja colocar numa mochila. O objetivo é escolher um conjunto de itens com o maior valor possível de forma a não exceder a capacidade da mochila.

Existem algumas variações para este problema. Uma das mais difundidas (CHU; BEASLEY, 1998) e escolhida para ser abordada neste trabalho é o *Multidimensional Knapsack Problem* (MKP), também conhecido como *Multiconstraint Knapsack Problem*, *Multi-Knapsack Problem*, *Multiple Knapsack Problem* ou *0/1 Multidimensional Knapsack Problem* (KHURI; BÄCK; HEITKÖTTER, 1994).

Este problema pode ser definido da seguinte maneira: sendo n o número de itens e m o número de mochilas. O item i possui um valor v_i e pesa r_{ij} unidades na j^{ezima} mochila. Tal mochila possui um limite l_j de peso que pode carregar. Assim, o objetivo é encontrar um subconjunto dos n itens, tal que o valor total deste subconjunto seja maximizado sem ultrapassar nenhum dos limites das mochilas. Ou seja, para que um subconjunto s seja considerado uma solução válida para o problema, s deve caber em cada uma das mochilas individualmente. Tal problema é matematicamente representado da seguinte forma:

$$\begin{aligned} &\text{maximizar} \quad f(x) = \textstyle\sum_{i=1}^{n} v_i x_i \\ &\text{tal que} \quad \textstyle\sum_{i=1}^{n} r_{ij} x_i \leq l_j, \quad j = 1,2,...,m \\ &\text{onde} \quad x_i = \begin{cases} 1, & \text{se item selecionado} \\ 0, & \text{caso contrário} \end{cases}, \quad i = 1,2,...,n \end{aligned} \tag{3.3}$$

Além destes problemas, outros exemplos provenientes do meio acadêmico e da indústria podem ser citados (KNOWLES; CORNE; DEB, 2008), como:

- Encontrar boas estratégias para jogos;
- Definir uma taxonomia de genes procariontes, através de suas atividades funcionais;
- Projetar uma ponte pênsil.

Capítulo 4

PSO em Problemas Discretos

Neste capítulo serão apresentados trabalhos relacionados que abordaram problemas combinatórios utilizando o algoritmo de metaheurística PSO. Aspectos relacionados às peculiaridades e adaptações deste algoritmo em relação a este tipo específico de problemas serão mostrados.

No algoritmo PSO clássico a velocidade é o operador responsável pela movimentação da partícula. Ao analisar a equação de atualização da velocidade (2.2) percebe-se que este operador engloba, em uma única operação, os três movimentos possíveis, e quando aplicado à partícula, faz com que a mesma seja influenciada pelos três caminhos de uma só vez proporcionalmente aos coeficientes w, $c1$ e $c2$. Entretanto quando se lida com problemas discretos pode não haver um único operador capaz de englobar os três movimentos.

Além disso, na PSO tradicional a partícula é codificada como um conjunto de variáveis reais que representam a sua localização num espaço multidimensional. E, de acordo com Hu, Eberhart e Shi (2003), todas as dimensões normalmente são independentes umas das outras, de forma que a aplicação da velocidade à posição da partícula é realizada independentemente em cada dimensão. Esta é uma das principais características do algoritmo PSO. Entretanto, desta forma o algoritmo PSO tradicional não é aplicável a problemas combinatórios, pois assim, conforme apresentado no capítulo anterior, o algoritmo violaria regras de restrição gerando soluções inválidas para o problema

tratado.

Portanto, é possível afirmar que o algoritmo PSO foi inicialmente projetado para a resolução de problemas de otimização contínua. Desse modo, tal algoritmo deve passar por uma série de adaptações para se tornar capaz de resolver problemas discretos. Várias abordagens têm sido adotadas na literatura, a seguir serão relatados alguns destes trabalhos e como foram feitas tais adaptações.

O primeiro documento que propõe uma versão discreta do algoritmo PSO foi apresentado por Kennedy e Eberhart (1997). Neste trabalho partículas foram codificadas como sequências binárias e a velocidade (v) como um conjunto de probabilidades. Dessa forma, uma partícula s se move em um espaço de estados restrito a zero e um em cada dimensão, onde cada v_i representa a probabilidade do bit s_i assumir o valor 1. Esta representação binária para solução de problemas discretos proposta por Kennedy e Eberhart serviu de base para vários outros trabalhos, alguns deles são citados nesta seção.

Hu, Eberhart e Shi (2003) implementaram um algoritmo de Nuvem de Partículas para o problema das N-Rainhas. Neste trabalho a velocidade também foi implementada como um vetor de probabilidades, mas com algumas peculiaridades: a velocidade é normalizada através da divisão de cada valor do vetor velocidade pelo maior valor do vetor. Em seguida, cada dimensão da partícula é escolhida aleatoriamente, então é determinado se haverá uma troca (*swap*) naquela posição. Quanto maior o valor da velocidade associado à dimensão x, maior a probabilidade da troca ocorrer em x.

Em Clerc (2004) foi desenvolvido um algoritmo PSO discreto aplicado a uma instância do TSP de 17 cidades. A velocidade foi definida como uma lista de pares (i, j), onde i e j são os índices dos elementos (cidades) da partícula que serão trocados. Clerc (2004) chamou estes pares de transposições e eles são aplicados à partícula de forma unitária. Por exemplo, para somar a velocidade v = ((1,2),(2,3),...) a uma posição p = (1,2,3,4,5,1) de uma partícula qualquer, primeiramente é aplicada a primeira transposição (1,2) e então é obtida a posição p = (2,1,3,4,5,2), depois a segunda transposição (2,3) é aplicada sobre o resultado da primeira e então é obtida a posição p = (3,1,2,4,5,3) e assim

sucessivamente enquanto houver transposições a serem feitas.

Neste trabalho a subtração entre duas posições $(x2 - x1)$ foi definida como uma velocidade v, tal que v é encontrada através de um algoritmo que garante que a operação inversa também possa ser feita, ou seja, com $v + x1$ obtém - se $x2$. Além das operações, Clerc (2004) relata características específicas a respeito de como aplicar ou multiplicar os coeficientes à velocidade dependendo do valor do coeficiente.

Dentre as abordagens híbridas da literatura recente encontra - se o trabalho de Marinakis e Marinaki (2008) que utilizou *Expanding neighborhood search*(ENS), *Greedy randomized adaptive search procedure* (GRASP) e *Path relinking* para compor um PSO híbrido aplicado ao location-routing problem (LRP) - problema de roteamento de veículos. Neste trabalho o *Path relinking* também foi utilizado para representar o movimento M2 e M3, mas apenas um destes movimentos é aplicado à partícula, através da velocidade, a cada iteração.

Outra estratégia semelhante foi adotada por Goldbarg, Souza e Goldbarg (2006). Nessa proposta, o Lin-Kernighan (LK) e outro algoritmo de busca local chamado *Inversion Procedure*(INV) foram usados junto com o *Path relinking* para formar duas versões de PSO para o TSP chamadas pelo autor PSO-LK e PSO-INV. Nesse trabalho, os coeficientes foram implementados como a probabilidade de escolha entre os três movimentos possíveis. Além disso, essas probabilidades foram estabelecidas de forma a privilegiar a escolha da busca local (LK ou INV), ou seja, o movimento M1, em relação ao *Path relinking* (outros dois movimentos).

Um dos trabalhos mais recentes da área envolvendo PSO discreto foi publicado por Chen et al. (2009). Neste trabalho foi desenvolvido um algoritmo baseado na teoria dos conjuntos e possibilidades. A principal característica do algoritmo de Chen et al. (2009), chamado por ele "Set-based PSO" (S-PSO) - PSO baseado em conjuntos - é a forma como foi implementada a velocidade, definida como um conjunto de possibilidades: sendo E uma sequência binária que representa uma solução para o problema. Um conjunto de possibilidades V definido em E é dado por

$$V = \{e/p(e) \mid e \in E\}$$

tal que cada elemento $e \in E$ tem uma possibilidade $p(e) \in [0, 1]$ em V.

A atualização de velocidade foi feita seguindo a tradicional equação de algoritmos PSO (2.2), mas utilizando a teoria de conjuntos para implementar a subtração entre duas posições. Além disso, regras específicas foram usadas para fazer a multiplicação entre os coeficientes e as posições das partículas. Este algoritmo foi um dos escolhidos para comparação neste trabalho.

Tabela 4.1: Comparação entre PSO discretos

Trabalho	Problema	Híbrido	*Path relinking*	Busca local	Bin
Kennedy et. al (1997)	TSP	não	não	não	sim
Hu et. al (2003)	N-rainhas	não	não	não	não
Clerc (2004)	TSP	não	não	não	não
Marinakis et. al (2008)	LRP	sim	sim	ENS	não
Goldbarg et. al (2006)	TSP	não	sim	LK / INV	não
Chen et. al (2009)	TSP / MKP	não	não	não	não
PSOPR	TSP / MKP	não	sim	LK / PSOPR-LS	não

A tabela 4.1 mostra uma comparação entre os trabalhos citados neste capítulo juntamente com o algoritmo desenvolvido neste trabalho (PSOPR). Para tal comparação foram extraídas algumas características gerais comumente utilizadas para descrever os algoritmos, são elas: o(s) problema(s) tratado(s), se a abordagem é híbrida, se foi utilizado *Path relinking*, a busca local utilizada e se a solução foi codificada como uma sequência binária (Bin).

Dada a revisão bibliográfica, o próximo capítulo trata do algoritmo desenvolvido, técnicas relacionadas e a abordagem em relação aos coeficientes do algoritmo PSO.

Capítulo 5

O algoritmo PSOPR

O algoritmo desenvolvido neste trabalho foi denominado PSOPR como uma alusão às técnicas utilizadas para o seu desenvolvimento: Otimização por Nuvem de Partículas (PSO) com *Path relinking* (PR).

O objetivo deste capítulo é apresentar o PSOPR como uma plataforma geral para otimização combinatória. Assim, aqui serão apresentados os conceitos e as peculiaridades da plataforma sem mencionar nenhum problema especificamente. Tanto a busca local quanto o *Path relinking* possuem características genéricas que são relatadas neste capítulo. Entretanto, ambas são técnicas dependentes do problema e, por isso, os detalhes da implementação referente aos problemas são relatados nos capítulos seguintes.

No algoritmo PSO, uma partícula possui três componentes de movimento: seguir seu próprio caminho (M1), seguir sua melhor posição anterior (M2), seguir a melhor posição dentre as suas vizinhas (M3). E os coeficientes w, $c1$, $c2$ são usados para limitar tais movimentos. No PSOPR o movimento M1 é implementado como uma busca local. Outro operador de velocidade é considerado quando uma partícula tem que se mover de sua posição atual para outra ($\vec{P}_{best}$ ou $\vec{L}_{best}$, M2 ou M3). Uma maneira natural de realizar essa tarefa é executar o *Path relinking* entre as duas soluções.

A principal diferença entre o presente trabalho e as demais abordagens que descrevem PSO discretos utilizando busca local e *Path reliking* diz respeito aos operadores de

velocidade. Pois no PSOPR todas as operações, ou movimentos, são aplicados à partícula em cada iteração.

$$\vec{V}(t+1) = \overbrace{w * \vec{V}(t)}^{\textbf{Local Search}} + \overbrace{c1 * \Phi 1 * \left(\underbrace{\vec{P}_{best}(t) - \vec{x}(t)}_{\textbf{M2}}\right) + c2 * \Phi 2 * \left(\underbrace{\vec{L}_{best}(t) - \vec{x}(t)}_{\textbf{M3}}\right)}^{\textbf{Path relinking}}$$
$$\underbrace{\qquad\qquad}_{\textbf{M1}}$$

Figura 5.1: A velocidade no algoritmo PSOPR
Fonte: O autor (2010)

De acordo com a equação 2.2, no PSO clássico, a velocidade é um operador que cobre os três movimentos em uma única operação. Assim, no algoritmo PSOPR (ver figura 5.1) a velocidade também foi definida como o conjunto de três operações que são aplicadas a cada partícula.

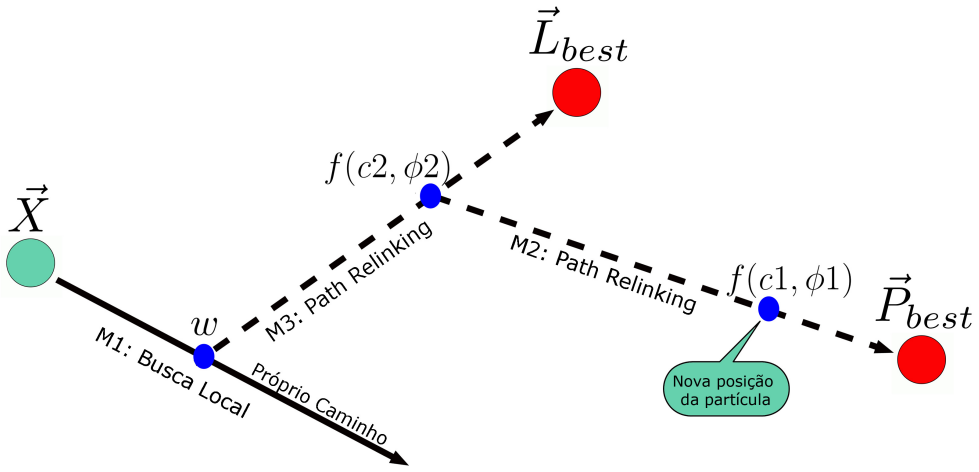

Figura 5.2: Representação gráfica do algoritmo PSOPR
Fonte: O autor (2010)

Como pode ser observado na figura 5.2, primeiro a busca local é realizada até o limite estipulado por w, então o *Path relinking* para o movimento M3 é aplicado até o limite estabelecido por $c2$ e $\phi 2$ através da função f que pode ser implementada de diferentes formas dependendo do problema tratado. Finalmente, o mesmo é feito para o movimento M2 cujo limite é estabelecido por $c1$ e $\phi 1$. A decisão de executar o movimento M3 antes de M2 foi tomada de forma empírica após vários experimentos. O pseudocódigo do PSOPR é apresentado no algoritmo 2.

Quanto a M1, como apresentado na seção 2.1, o mesmo envolve um conceito

Algorithm 2 Pseudocódigo of the PSOPR algorithm

1: Atribui parâmetros
2: **for** $i = 0$ até $tamanhoEnxame$ **do**
3: Inicia $\vec{X}^i$ com valores aleatórios
4: $\vec{P}^i_{best} \leftarrow \vec{L}^i_{best} \leftarrow \vec{X}^i$
5: **end for**
6: **while** Não atingir condição de parada **do**
7: **for** $i = 0$ até $tamanhoEnxame$ **do**
8: $\vec{X}^i \leftarrow localSearch(\vec{X}^i)$ (M1)
9: $\vec{X}^i \leftarrow pathRelinking(\vec{X}^i, \vec{L}^i_{best})$ (M3)
10: $\vec{X}^i \leftarrow pathRelinking(\vec{X}^i, \vec{P}^i_{best})$ (M2)
11: $\vec{P}^i_{best} \leftarrow$ melhor entre $\vec{X}^i$ e $\vec{P}^i_{best}$
12: $\vec{L}^i_{best} \leftarrow$ melhor entre $\vec{P}_{best}$ e $\vec{L}^i_{best}$
13: **end for**
14: **end while**
15: **return** melhor $\vec{L}_{best}$

bastante amplo: "Seguir o próprio caminho". Talvez por isso existam tantas abordagens diferentes em relação a este componente, principalmente no que diz respeito à PSO híbridos, tais como Yin et al. (2006), Machado e Lopes (2005) e Chen, Yang e Wu (2006).

No algoritmo PSOPR este conceito foi implementado como sendo uma busca local específica ou adaptada para o problema em questão. Portanto o presente trabalho descreve a implementação e experimentação de uma metaheurística focada para problemas combinatórios na qual se utiliza, além do *Path relinking*, um algoritmo de busca local internamente. Na próxima seção 5.1 serão apresentados alguns conceitos gerais sobre este tipo de algoritmo.

As demais características do PSOPR em relação ao algoritmo *Path relinking* e a implementação dos coeficientes($c1$, $c2$, $\phi 1$, $\phi 2$) são apresentadas, respectivamente, nas seções 5.2 e 5.3.

5.1 Busca Local

Segundo West (2000) a busca local é uma técnica de resolução de problemas de otimização através de sucessivas pequenas mudanças em uma possível solução.

Algoritmos de busca local são utilizados em problemas de otimização computacionalmente complexos. Este tipo de algoritmo, geralmente, atua sobre uma solução pré-existente (s) fazendo consecutivas alterações na tentativa de obter uma solução melhor para o problema em questão.

Neste tipo de algoritmo, a solução s, chamada solução candidata, possui uma vizinhança $N(s)$ que é um subconjunto do espaço de busca do problema ($N(s) \subset S$). Esta vizinhança é definida de acordo com as características específicas de cada algoritmo e do problema tratado. Em resumo, o algoritmo inicia com uma solução candidata e iterativamente se move para uma solução vizinha v, se $F(v) < F(s)$ (para um objetivo de minimização), sendo F a função de avaliação das soluções do problema em questão. A partir disso é definida a nova vizinhança de v que se move para uma solução pertencente a $N(v)$ e assim sucessivamente até que não seja mais possível ocorrer melhoras na solução atual, que é então retornada como a solução otimizada para o problema.

Um dos algoritmos mais conhecidos e utilizados desta categoria é o *Hill Climbing*. Esta técnica de busca consiste em um ciclo contínuo que altera a solução buscando sempre um maior valor correspondente a função objetivo do problema, ou seja, a solução é alterada uma vez, caso aja melhora decorrente desta alteração, altera-se a solução novamente e assim continuamente. Esse processo se repete até que, invariavelmente, o algoritmo atinge um ponto no qual não são feitos mais progressos nesta solução. Quando isso ocorre, o algoritmo pode ser iniciado novamente a partir de um ponto de partida diferente (RUSSELL; NORVIG, 2002). Uma ilustração simples do comportamento deste algoritmo pode ser vista na figura 5.3.

O método *Hill Climbing* por si só aplicado a problemas complexos de otimização pode não ser considerado uma técnica eficiente. Pois seu comportamento é demasiadamente inflexível em relação à melhora das soluções, visto que não permite que a solução recue a um ponto menos qualificado no espaço de busca, visando à melhora a cada iteração, sem exceções. Esse comportamento torna o algoritmo passível de gerar soluções consideradas ótimos locais, pois quando a solução se encontra em uma dessas regiões do

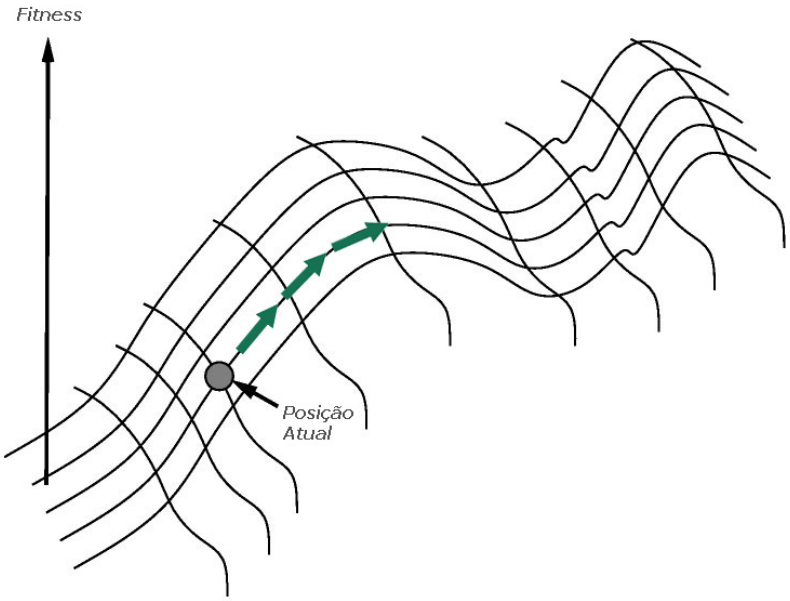

Figura 5.3: Representação gráfica do algoritmo de busca local *Hill Climbing*
Fonte: Adaptado de http://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html

espaço de busca, uma das formas de ainda obter alguma melhora é permitir que alterações sejam feitas mesmo sem um aumento imediato, ou até mesmo uma queda, na qualidade da solução.

Devido a esta característica o *Hill Climbing* é comumente aplicado juntamente com uma metaheurística como em Lim et al. (2006) e Al-kazemi e Mohan (2000), servindo como um complemento importante para o algoritmo principal, visto que aumenta a capacidade de exploração do algoritmo como um todo e, consequentemente, auxilia na descoberta de melhores soluções.

5.2 *Path Relinking*

O *Path relinking* foi sugerido por Glover e Laguna (1997), como uma abordagem para integrar estratégias de intensificação e diversificação. Vários trabalhos posteriores descrevem a aplicação desta técnica em conjunto com outras metaheurísticas como GRASP (LAGUNA; MARTI, 1999; RESENDE; RIBEIRO, 2005).

O algoritmo consiste na geração de um caminho de soluções intermediárias entre

uma solução inicial x' e uma solução objetivo x''. Tal solução (x'') deve pertencer à vizinhança de x' e é chamada na literatura solução guia (*guiding solution*). Para gerar tal caminho, é necessário apenas que sejam aplicados progressivamente movimentos a x' de forma a reduzir a distância entre esta e a solução objetivo x''. Assim, o caminho é formado a partir da sequência de soluções $x' = x(l), x(2), ..., x(r) = x''$, tal que a solução $x(i+1)$ é criada a partir de $x(i)$ a cada passo, escolhendo um movimento que deixe um número reduzido de movimentos restantes para atingir x'' (GLOVER; LAGUNA, 2000).

Geralmente $F(x'') < F(x')$, onde F é a função de avaliação de um problema de minimização. Mas existem variantes nas quais a solução inicial é melhor que a solução objetivo. Outra abordagem apresentada por Glover e Laguna (2000), é o *Tunneling*, no qual o algoritmo começa com ambas as soluções x' e x'' como soluções objetivo produzindo duas sequências de soluções intermediárias simultaneamente. Outra variação do algoritmo, o *Extrapolated Relinking*, é considerada uma extensão do *Path relinking* tradicional, pois vai além do ponto final do caminho utilizado como solução objetivo. Utilizada por Beausoleil, Baldoquin e Montejo (2007), nesta estratégia os movimentos feitos no caminho entre x' e x'' são analisados de forma a extrair um padrão que é aplicado à solução depois dela atingir a solução objetivo e, assim, continuar gerando novas soluções.

Como citado anteriormente, o *Path relinking* foi utilizado na implementação dos movimentos M2 e M3. A versão utilizada se baseia na proposta original de Glover e Laguna (1997) na qual $F(x'') < F(x')$. Quanto à adaptação ao PSO, naturalmente x' foi implementado como a posição atual da partícula (P_{ini}) e x'' como a posição da partícula objetivo (P_{obj}). Assim, nesta seção P_{obj} representa uma das duas posições:

$$P_{obj} = \begin{cases} P_{best} & \text{se movimento} = M2 \\ L_{best} & \text{se movimento} = M3 \end{cases}$$

Antes de executar qualquer movimento, ou passo, na P_{ini} em direção a P_{obj}, a distância entre as duas soluções deve ser calculada. Existem várias funções que podem ser utilizadas para o cálculo de distância entre soluções de problemas discretos. A função

utilizada nesse trabalho foi a função de distância exata ($distExata$), *exact match distance*, descrita em Ronald (1998). Este é um método considerado computacionalmente barato no qual a distância é definida, basicamente, como o total de dimensões cujos valores são diferentes entre si. Por exemplo:

$$\vec{P}_{ini} = \{1, 7, \mathbf{5}, 6, \mathbf{2}, 8, \mathbf{4}, \mathbf{3}, 1\}$$
$$\vec{P}_{obj} = \{1, 7, \mathbf{2}, 6, \mathbf{4}, 8, \mathbf{3}, \mathbf{2}, 1\}$$
$$distExata(\vec{P}_{ini}, \vec{P}_{obj}) = 4$$

O movimento ocorre a partir de uma alteração em uma dada dimensão de P_{ini} e esta alteração é aplicada de acordo com o problema em questão. Apesar disso, o método de cálculo da quantidade de movimentos que serão aplicados não difere, este método é chamado $calculaPassos$ e sua descrição, juntamente com os parâmetros c e ϕ, é apresentada na próxima seção (5.3).

Durante a execução do *Path relinking*, a cada alteração na solução atual, uma nova solução é criada. Mas isso não significa que toda solução intermediária precisa ser avaliada. Assim, o parâmetro $c_{pr} = (0, 1]$ foi criado. Este coeficiente limita o número de soluções intermediárias que são avaliadas durante o *Path relinking*. Portanto, quanto maior o valor de c_{pr}, mais soluções avaliadas haverá entre P_{ini} e P_{obj}. A idéia é tornar o PSOPR mais eficiente em termos computacionais, já que a reavaliação completa da solução a cada alteração pode representar um custo computacional significativo dependendo do problema que está sendo abordado.

O pseudocódigo do algoritmo 3 mostra como o *Path relinking* foi definido neste trabalho. Primeiramente, a distância, o número de passos que serão executados e o intervalo entre as avaliações (de acordo com c_{pr}) são calculados. Logo após começam as iterações, a cada passo, uma alteração é aplicada a P_{ini} (linha 7). Na sequência, o algoritmo verifica se o intervalo de passos sem avaliar a solução foi atingido (linha 8), em caso positivo P_{ini} é avaliada, então é verificado se esta nova posição é melhor que P_{best} (linha 10). Em caso positivo a posição atual de P_{ini} é guardada como P_{best}

Algorithm 3 Pseudocódigo do algoritmo PSOPR-Path-Relinking

1: $distancia = distExata(P_{ini}, P_{obj})$
2: $totalPassos = calculaPassos(distancia, c, \phi)$
3: $avalicoes = 0$
4: $intervalo = 1/c_{pr}$
5: **for** $passo = 1$ até $totalPassos$ **do**
6: escolhe dimensão d
7: aplica alteração em P^d_{ini}
8: **if** $passo >= intervalo * avaliacoes$ **then**
9: avalia P_{ini}
10: **if** P_{ini} melhor que P_{best} **then**
11: $P_{best} = P_{ini}$
12: **end if**
13: $avaliacoes++$
14: **end if**
15: **end for**
16: **return** P_{ini}

e, consequentemente, utilizada posteriormente como P_{obj} em um novo processamento do método *Path relinking*, depois disso o contador de avaliações é incrementado na linha 13. Após todas as iterações, a posição alcançada ao final da última troca é retornada como a nova posição da partícula.

Esta seção apresentou o *Path relinking* e como ele é utilizado no PSOPR de uma forma geral. Ao implementar tal algoritmo a algum COP específico, algumas peculiaridades do problema devem ser levadas em consideração. Por isso a implementação deste algoritmo para o MKP é ligeiramente diferente da versão para o TSP. Características específicas a respeito de cada versão, tal como os algoritmos de busca local utilizados, são apresentados nos dois capítulos seguintes.

5.3 Os Coeficientes $c1$, $c2$, $\phi1$ e $\phi2$

Conforme apresentado no capítulo 2, de acordo com Kennedy e Eberhart (1995), todo algoritmo PSO deve conter o coeficiente $c1$ (fator de individualidade) implementado como uma variável real que influencia a partícula a seguir a sua melhor posição já encontrada, e $c2$ (fator de sociabilidade) que influencia a partícula em direção a melhor posição já

encontrada pela vizinhança.

Além disso, a cada iteração a velocidade da partícula deve ser atualizada levando em conta os três coeficientes (w, $c1$ e $c2$) e, então, aplicada à partícula. Ou seja, mesmo com um dos coeficientes tendo um valor muito pequeno em relação aos demais, ainda assim, haverá a sua influência a cada iteração. Por exemplo, supondo que os coeficientes estejam configurados da seguinte forma: $w = 0.02$, $c1 = 1.5$ e $c2 = 1.5$. Neste caso o algoritmo faria com que suas partículas pouco procurassem por regiões inexploradas do espaço de busca tendendo a seguir as melhores posições já encontradas até o momento pelo algoritmo. Mas mesmo com um valor consideravelmente menor que os demais coeficientes o coeficiente de inércia w continuaria influenciando a partícula, ainda que pouco, a seguir seu próprio caminho.

Na tentativa de manter as características e conceitos originais em relação a algoritmos PSO os coeficientes foram implementados como limitadores dos três possíveis caminhos a serem seguidos. E a cada iteração todas as operações, ou movimentos, são aplicados à partícula.

Assim o fator de individualidade é responsável por determinar até que ponto a partícula seguirá M2 e o fator de sociabilidade determina o mesmo em relação a M3. Da mesma forma, o coeficiente de inércia (w) é utilizado para limitar o movimento M1. Entretanto este foi implementado de forma bastante distinta para cada problema abordado devido às peculiaridades de cada busca local utilizada. Portanto w é relatado nos capítulos 6 e 7, nos quais as implementações do PSOPR são apresentadas.

Na literatura, geralmente o coeficiente de individualidade é representado por $c1$ e o de sociabilidade por $c2$. Neste trabalho o método de aplicação de ambos os coeficientes é o mesmo e, por isso, nesta seção ambos serão representados pela variável c, já que todas as operações são realizadas da mesma forma independentemente de qual dos dois coeficientes esteja sendo utilizado. A mesma regra vale para os coeficientes aleatórios, aqui representados por ϕ, assim: se $P_{obj} = P_{best}$, então $c = c1$ e $\phi = \phi1$; Se $P_{obj} = L_{best}$, então $c = c2$ e $\phi = \phi2$.

No presente trabalho os coeficientes $c1$ e $c2$ são valores reais predefinidos no início do algoritmo, sendo $c = (0, 1)$, já os coeficientes $\phi 1$ e $\phi 2$ são gerados aleatoriamente a cada execução do *Path relinking* ($\phi = [0, 1)$).

Conforme apresentado anteriormente, os coeficientes atuam dentro do algoritmo *Path relinking* através da função $calculaPassos$ que é representada pela seguinte equação:

$$return \quad distExata(P_{ini}, P_{obj}) * f(c, \phi) \tag{5.1}$$

A função f utiliza o coeficiente c e o coefienciente aleatório ϕ para determinar o número de passos que serão percorridos entre a posição de origem (P_{ini}) e destino (P_{obj}). Tal função pode ser implementada de várias formas diferentes como por exemplo, simplesmente multiplicar um coeficiente pelo outro, conforme é mostrado na equação 2.2. Porém esta implementação tenderia a números muito baixos e como o resultado representa a porcentagem da distância a ser percorrida entre as posições de origem e destino, todos as posições finais tenderiam a ser muito mais parecidas com a posição de origem, o que não seria uma característica positiva para o algoritmo de uma forma geral. Assim, nas duas implementações do algoritmo PSOPR, que serão relatadas a seguir, optou-se por extrair a média aritmética dos dois valores:

$$f(c, \phi) = \frac{c + \phi}{2}$$

Capítulo 6

Uma Implementação Para o TSP

Este capítulo descreve a implementação do algoritmo PSOPR para o TSP. Primeiramente, a técnica de busca local utilizada nesta versão é detalhada, seguida pela descrição do comportamento da inércia em relação ao algoritmo de busca local. Na sequência, a seção 6.3 relata as peculiaridades do algoritmo *Path reliking* para o problema em questão. Finalmente a configuração final do algoritmo é apresentada depois de uma série de experimentos visando à análise de sensibilidade do algoritmo em relação aos seus coeficientes.

Em relação às topologias de vizinhança, a utilizada nesta versão do algoritmo foi a GCC, descrita anteriormente na subseção 2.1.3.

6.1 Busca Local para o TSP

Um dos métodos mais simples de busca local para o TSP é o 2-opt. Neste algoritmo, a vizinhança $N(s)$ da solução s é definida pelo conjunto de soluções que podem ser alcançadas através da permutação de duas arestas não-adjacentes em s (HELSGAUN, 2000). Este movimento é chamado *2-interchange* ou movimento 2-opt.

A figura 6.1 ilustra de forma simples como duas arestas do grafo (a) são trocadas de posição, através do movimento 2-interchange, resultando no grafo (b).

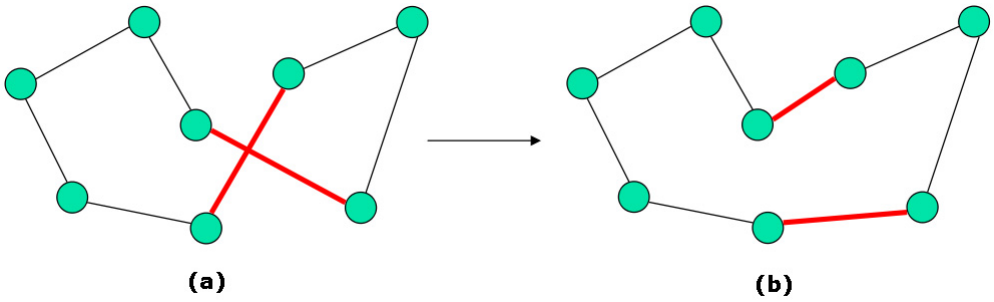

Figura 6.1: Um caminho (a) antes e (b) depois do movimento *2-interchange*.
Fonte: Michalewicz e Fogel (2004).

Outro método encontrado na literatura é o k-opt. Ele é considerado a generalização do anterior, ou seja, neste método a vizinhança $N(s)$ da solução s é definida pelo conjunto de soluções obtidas através da permutação de até k arestas. Os movimentos executados através deste método são chamados *k-interchange* ou movimento k-opt.

Dentre os algoritmos de busca local desenvolvidos especificamente para o TSP, um dos mais conhecidos é o Lin-Kernighan (LK). Desenvolvido por Lin e Kernighan (1973), o núcleo do algoritmo é um método de busca que tenta realizar uma sequência (possivelmente longa) de movimentos k-opt de tal forma que cada subsequência inicial pareça ter uma chance de levar a uma solução melhor ou caminho de menor custo no caso do TSP.

Embora qualquer movimento k-opt possa ser realizado como uma sequência de movimentos 2-opt, algumas das soluções intermediárias podem ter custo maior que o da solução inicial, mesmo em um movimento k-opt que gere alguma melhora, esse é um dos diferenciais deste algoritmo. Se a busca for bem-sucedida em encontrar uma solução melhor, então a sequência de movimentos k-opt é feita e uma nova busca é iniciada. Caso contrário, a sequência de movimentos não é executada, mas sim, guardada pelo algoritmo para garantir que a mesma sequência sem êxito não seja gerada novamente. O processo termina depois que mais nenhum ponto de partida do algoritmo se mostrar capaz de gerar qualquer melhora (APPLEGATE et al., 1999).

Em resumo, o algoritmo busca a sequência de arestas $\{x1, x2, ..., xk\}$ que, quando trocadas pelas arestas $\{y1, y2, ...,yk\}$, retornam um caminho factível e de menor custo.

O LK inicia construindo um caminho aleatório e sorteando um nó base pelo qual começarão as trocas. Em seguida, é escolhida uma aresta contendo o nó inicial $x1 = (base, next(base))$ e uma aresta de troca que parta de $next(base)$ e que gere um ganho positivo $y1 = (next(base), a)$. O vértice a é chamado vizinho promissor de $next(base)$ e o objetivo do algoritmo é encontrar um vértice a de forma a maximizar

$$c(prev(a), a) - c(next(base), a) \tag{6.1}$$

Ou seja, faz com que o custo (c) formado pela nova aresta $c(next(base), a)$ seja menor do que a aresta anterior formada por $c(prev(a), a)$. Existem varias formas de definir uma vizinhança para um vértice, a mais comum é considerar os k vértices mais próximos ao vértice em questão.

Escolhida a primeira aresta de troca, o processo iterativo é iniciado, no qual a cada passo uma aresta é escolhida contendo o último nó escolhido na iteração anterior (MICHALEWICZ; FOGEL, 2004).

Terminada a construção do conjunto de arestas originais (x) e o conjunto de arestas de troca (y), o novo caminho é construído e um novo vértice ainda não escolhido é eleito como ponto de partida para a execução do algoritmo novamente. Na versão de Applegate et al. (1999) a cada vez que isto ocorre a variável *steps* é incrementada, totalizando assim, ao final da execução completa do algoritmo, n *steps*, sendo n o número de cidades do problema.

Uma característica interessante do algoritmo LK é que o mesmo utiliza uma variável Δ que guarda o ganho obtido por toda a sequência de movimentos. Dessa forma em qualquer momento da execução do algoritmo é possível saber o custo da solução atual s através de $c(s) - \Delta$, assim não é necessário recalcular o peso de cada aresta a cada iteração o que torna o algoritmo mais rápido.

A literatura contém relatos de muitas implementações diferentes do LK com grande variação de comportamento (JOHNSON; MCGEOCH, 2001). A versão do LK

utilizada neste trabalho foi extraída do *software Concorde TSP Solver* cujo código fonte foi disponibilizado para *download* [1]

6.2 Inércia (w)

Um dos principais objetivos deste trabalho consiste em desenvolver um algoritmo de metaheurística mantendo as principais características do PSO original. Para isso, procurou-se nivelar o custo de processamento do LK em relação ao *Path relinking*. Como o LK funciona através de execuções iniciadas por um nó ainda não escolhido como o nó inicial, foi decidido que não mais de 10% dos nós do problema seriam utilizados como base para tais execuções. Além disso, o número de execuções deveria ser variável, portanto, $w = (0,1]$ foi implementado como um limite para o algoritmo LK. Toda vez que o LK é chamado, a seguinte equação é usada para definir o número e de movimentos do algoritmo:

$$e = (int)\ (n/10) * w$$

Onde n é o número de cidades do problema, consequentemente, o número de possíveis vértices para início das execuções do LK.

6.3 *Path Relinking* para o TSP

Como o TSP é um problema modelado em forma de grafo, é impossível modificar apenas uma dimensão da partícula por alteração mantendo a viabilidade da solução. Assim, para executar cada alteração a fim de movimentar P_{ini} em direção a P_{obj}, foi utilizado o operador de troca (*swap*).

O procedimento de aplicação deste operador é simples. Conforme ilustrado pela figura 6.2, num determinado passo/iteração i da execução do *Path relinking*, uma dimen-

[1] http://www.tsp.gatech.edu/concorde/downloads/downloads.htm

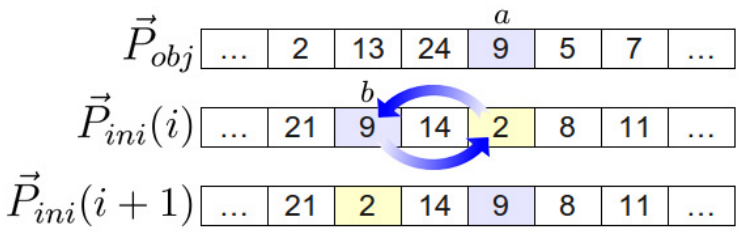

Figura 6.2: Operação de troca (*swap*) dentro do *Path relinking*
Fonte: Autor (2010)

são a é escolhida tal que $P^a_{obj} \neq P^a_{ini}$, depois é encontrada a dimensão b tal que $P^b_{ini} = P^a_{obj}$ e então são trocados os valores contidos em P^a_{ini} e P^b_{ini}.

Em relação ao cálculo de passos do algoritmo, no caso do TSP, a função de distância exata não é capaz de fornecer o número de passos de distância entre a solução inicial e final, visto que, em uma única troca duas dimensões de P_{ini} podem assumir o mesmo valor de P_{obj}, ainda citando a figura 6.2, isso é o que ocorreria com a operação demonstrada em caso de $P^b_{obj} = 2$. E a probabilidade disto ocorrer aumenta de acordo com o número de operações já realizadas, pois quanto mais trocas aplicadas à P_{ini}, mais esta será semelhante à P_{obj}.

Portanto, na implementação para o TSP a distância calculada pela função de distância exata é uma estimativa do número de posições intermediárias que podem ser geradas entre as duas soluções, em outras palavras, a distância é igual ao máximo de trocas consecutivas necessário para que P_{ini} se transforme em P_{obj}.

O fato do algoritmo não calcular previamente com exatidão o número de trocas que uma solução se encontra da outra não acarreta problemas para geração das soluções, visto que a própria equação que calcula o número de passos leva em conta o coeficiente aleatório ϕ, cuja função é justamente variar a quantidade de trocas aplicadas à partícula a cada iteração.

6.4 Sensibilidade do Algoritmo

A seguir, são relatados três experimentos distintos que consistem na análise e avaliação da sensibilidade do algoritmo aos coeficientes em questão. Tais experimentos são necessários pois ao implementar um novo algoritmo baseado em um metamodelo, não se sabe exatamente o grau de influência que cada coeficiente exerce sobre o algoritmo como um todo para o problema tratado, ou mesmo se existe alguma influência. Dessa forma os experimentos a seguir não têm como objetivo o ajuste fino do algoritmo e sim a avaliação da influência e um ajuste superficial dos parâmetros/coeficientes do algoritmo.

O primeiro experimento diz respeito ao coeficiente c_{pr}, o segundo se refere ao coeficiente de inércia(w) e o terceiro aos coeficientes de individualidade($c1$) e sociabilidade($c2$). Ao final da bateria de experimentos de sensibilidade as configurações que apresentaram os melhores resultados foram definidas como a configuração padrão do algoritmo PSOPR-TSP. Esta configuração foi utilizada para comparação com outros algoritmos que será descrita no capítulo 8.

Para os experimentos a seguir foram escolhidas 12 bases do TSPLIB[2] sendo essas variando de 76 a 5915 cidades. Para cada base foram feitas 30 execuções de cada configuração do algoritmo. A média e melhor resultado das execuções são apresentados e os resultados analisados.

Devido à grande quantidade de parâmetros para ajustar, o tamanho do enxame e número de iterações foram configurados baseados na literatura. De acordo com Chen et al. (2009), os algoritmos de PSO aplicados ao TSP tendem a gerar boas soluções com um número máximo de partículas variando entre 20 e 40. Quanto ao número de iterações, para os experimentos de sensibilidade, o valor foi definido baseado em Kennedy e Eberhart (1995).

A tabela 6.1 mostra os valores que foram testados nos experimentos de sensibilidade do PSOPR-TSP (descritos a seguir neste capítulo) e os valores em negrito repre-

[2]http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/

sentam a configuração padrão inicial do algoritmo para tais experimentos.

Tabela 6.1: PSOPR-TSP - Parâmetros testados nos experimentos de sensibilidade

Parâmetro	Valores testados
iterações	30
tamanho do enxame	30
c_{pr}	**0,2**; 1,0
w	0,2; **0,5**; 0,65; 0,9...0,4
$c1$	0,2; **0,5**; 0,8
$c2$	0,2; **0,5**; 0,8
topologia	**GCC**

Os experimentos relatados a seguir, juntamente com os demais experimentos do trabalho, foram executados em um computador Intel Core 2 Quad 2.4 GHZ com sistema operacional Ubuntu 9.10.

6.4.1 Variação do coeficiente c_{pr}

Em relação à sensibilidade do algoritmo ao coeficiente c_{pr}, foram testadas duas configurações diferentes: $c_{pr} = 0{,}2$ e $c_{pr} = 1$, chamadas respectivamente de PSOPR-c02 e PSOPR-c1. Assim, de acordo com os conceitos apresentados na seção anterior, na primeira configuração o algoritmo avalia uma solução intermediária a cada cinco trocas realizadas pelo *Path relinking* e na segunda, a solução intermediária é avaliada a cada troca.

A tabela 6.2 mostra a média (Med), melhor solução obtida (Min), desvio padrão (DP) e tempo de execução (T) de cada versão do algoritmo para cada base. Onde Med representa a média aritmética de todos os Min das 30 execuções, enquanto os valores de Min são medidos pela porcentagem da diferença de qualidade entre a solução obtida pelo algoritmo e a melhor solução conhecida para a base em questão. Portanto o valor 0,0000 significa que o algoritmo alcançou uma solução cuja qualidade é igual à melhor solução já encontrada até o momento para o problema na literatura, e a qualidade desta solução é chamada valor ótimo (VO).

Tabela 6.2: Comparação entre valores do coeficiente c_{pr}

	$c_{pr} = 0{,}2$				$c_{pr} = 1{,}0$			
Bases	Med(%)	Min(%)	DP(%)	T(s)	Med(%)	Min(%)	DP(%)	T(s)
pr76	0,0000	0,0000	0,0000	6,55	0,0000	0,0000	0,0000	**6,42**
rat195	0,0000	0,0000	0,0000	**29,45**	0,0000	0,0000	0,0000	36,17
kroA200	0,0000	0,0000	0,0000	13,64	0,0000	0,0000	0,0000	**12,52**
pr299	**0,0000**	0,0000	0,0000	**44,73**	0,0004	0,0000	0,0000	59,70
pr439	**0,0414**	0,0000	0,0362	**423,53**	0,0416	0,0000	**0,0360**	503,83
d657	0,2524	0,1349	**0,0738**	**564,17**	**0,2440**	**0,0654**	0,0867	727,01
pr1002	0,7584	0,3579	**0,1415**	**957,16**	**0,7047**	**0,3358**	0,1900	1409,12
nrw1379	**0,5878**	0,3778	**0,0766**	**1216,72**	0,5970	**0,3107**	0,1142	2036,63
d2103	4,7576	**2,3356**	**0,7439**	**3110,59**	**4,5183**	2,3493	0,8736	4710,66
pcb3038	**0,9714**	0,6681	**0,1555**	**4109,76**	1,0079	**0,5490**	0,1567	8345,45
fnl4461	**0,9697**	**0,6173**	0,1125	**7549,46**	0,9813	0,7970	**0,0813**	16318,60
rl5915	5,9289	2,8902	**0,9944**	**11952,60**	**5,6421**	**2,5412**	1,0697	27759,80

De acordo com os resultados mostrados na tabela 6.2, o algoritmo PSOPR-c1 foi capaz de gerar a melhor solução para um maior número de bases. Apesar disso, o PSOPR-c02 foi melhor no quesito média e mostrou ser significativamente mais rápido, chegando até a 43,06% do tempo de processamento do concorrente para a maior base utilizada (rl5915).

Dessa forma foi escolhida a configuração $c_{pr} = 0{,}2$ para os demais experimentos da dissertação. Visto que a diferença entre a média e o melhor resultado foi mínima e ao mesmo tempo as diferenças entre tempo de processamento chegou a ser mais que o dobro para as maiores bases do experimento.

6.4.2 Variação de inércia

Para os experimentos em relação ao coeficiente de inércia, foram testadas três configurações diferentes: duas configurações estáticas com $w = 0{,}2$ e $w = 0{,}65$, e uma configuração dinâmica com w decrescendo linearmente de 0,9 até 0,4.

O objetivo da primeira configuração (PSOPR-W02) foi analisar o comportamento do algoritmo com um valor de inércia significativamente menor que as outras duas configurações e também menor que em outros trabalhos relacionados.

Em algoritmos PSO cada coeficiente pode ser estático, com apenas um valor atribuído que se mantém desde o começo até o fim da execução ou então dinâmico com seu valor atualizado a cada iteração. Assim, o objetivo da segunda configuração (PSOPR-W065) foi fazer uma comparação direta entre as duas abordagens, dinâmica e estática, por isso $w = 0{,}65$ foi definido como a média entre o valor inicial e final da abordagem dinâmica (PSOPR-WD).

Dentre as configurações de inércia dinâmica, uma em especial foi utilizada em vários trabalhos relacionados: w variando linearmente de 0,9 até 0,4. Esta configuração foi proposta em (SHI; EBERHART, 1998) e é definida pela equação 6.2.

$$w = \frac{(iMAX - i) * (w_{final} - w_{inicial})}{iMAX} + w_{inicial}, \quad i = 1, 2, ..., iMAX \tag{6.2}$$

Onde $iMAX$ é o número máximo de iterações por execução e i representa a iteração corrente.

A tabela 6.3 mostra os resultados do experimento. O PSOPR-W02 foi visivelmente inferior aos demais visto que conseguiu uma melhor solução que os outros em apenas uma das bases (pcb3038). Já quanto à comparação entre a inércia estática (PSOPR-W065) e a dinâmica não foi possível afirmar com certeza qual a melhor configuração apenas através da média e melhor solução. O PSOPR-WD apresentou como resultado uma melhor solução a mais e uma melhor média a menos que o concorrente ao mesmo tempo em que mostrou ser um pouco mais lento para bases maiores.

A partir destes resultados foi necessário utilizar um método estatístico na tentativa de definir qual a melhor dentre as duas configurações. Dessa forma foi utilizado o teste não paramétrico de Wilcoxon (WILCOXON, 1945), U-Test, com 5% de confiança. O teste de Wilcoxon foi aplicado utilizando as 24 séries de dados (2 algoritmos e 12 bases) com as melhores soluções das 30 execuções. Como resultado, de acordo com o U-Test, não houve diferença significativa entre as configurações para nenhuma das 12 bases.

Dessa forma foi decidido por manter a configuração dinâmica de inércia por ter gerado maior quantidade de melhores execuções.

6.4.3 Variação dos fatores de sociabilidade e individualidade

Analisando aplicações e versões diferentes de PSO, é possível afirmar que vários pesquisadores alcançaram resultados conclusivos e positivos no que diz respeito à utilização de ambos os coeficientes, $c1$ e $c2$, com o mesmo valor atribuído (EBERHART; SHI, 2000). Devido a este fato, e também como forma de limitar a gama de possibilidades de experimentos de sensibilidade, neste trabalho esta característica foi mantida.

Assim, para avaliar o nível de influência dos coeficientes de sociabilidade e individualidade nos resultados do algoritmo foram avaliadas três configurações diferentes: $c1 = c2 = 0{,}2$ (PSOPR-c02), $c1 = c2 = 0{,}5$ (PSOPR-c05) e $c1 = c2 = 0{,}8$ (PSOPR-c08).

O resultado do experimento foi expressivo (tabela 6.4). O PSOPR-c08 gerou todas as melhores soluções e médias em todas as bases. Em relação ao custo de processamento, esta configuração resultou nos menores tempos para quatro das cinco menores bases utilizadas no experimento. Isso prova que esta configuração foi capaz de guiar as soluções para o ótimo global de forma mais rápida que as outras.

Além disso, a diferença no tempo de processamento para as bases maiores é mínima. Todos estes fatores facilitaram a escolha da melhor configuração para os coeficientes $c1$ e $c2$. Dessa forma foi atribuído o valor 0,8 aos coeficientes em questão para os experimentos subsequentes com outros algoritmos.

6.4.4 Configuração final padrão

Após todos os experimentos de sensibilidade aos coeficientes e análises dos resultados, a configuração capaz de gerar os melhores resultados para o PSOPR-TSP é mostrada abaixo. Esta configuração foi definida para comparação com outros algoritmos de trabalhos relacionados no capítulo 8.

- tamanho do enxame = 30
- c_{pr} = 0,2
- w = 0,9...0,4
- $c1$ = 0,8
- $c2$ = 0,8
- topologia = GCC

Quanto ao número máximo de iterações, é importante ressaltar que este não foi definido acima pelo fato de variar de experimento para experimento de acordo com os critérios utilizados no trabalho no qual o algoritmo está sendo comparado. Mais detalhes a respeito serão apresentados no capítulo 8 (Resultados Finais).

Tabela 6.3: Comparação entre valores do coeficiente de inércia

	w = 0,2				**w = 0,65**				**w = 0,9...0,4**			
Bases	Med(%)	Min(%)	DP(%)	T(s)	Med(%)	Min(%)	DP(%)	T(s)	Med(%)	Min(%)	DP(%)	T(s)
pr76	0,0000	0,0000	0,0000	**3,99**	0,0000	0,0000	0,0000	7,85	0,0000	0,0000	0,0000	10,73
rat195	0,0100	0,0000	0,0215	44,15	0,0000	0,0000	**0,0000**	32,31	0,0000	0,0000	**0,0000**	**20,09**
kroA200	0,0000	0,0000	0,0000	16,55	0,0000	0,0000	0,0000	13,43	0,0000	0,0000	0,0000	**12,70**
pr299	0,0016	0,0000	0,0041	77,50	0,0004	0,0000	0,0023	51,04	**0,0000**	0,0000	**0,0000**	**38,53**
pr439	0,0780	0,0065	0,0589	**357,88**	0,0365	0,0000	0,0369	458,05	**0,0260**	0,0000	**0,0223**	478,63
d657	0,3187	0,1288	0,0900	**497,39**	0,2389	**0,0797**	**0,0644**	597,25	**0,2203**	0,1043	0,0708	611,63
pr1002	0,8560	0,5065	0,1657	**885,02**	**0,6822**	0,2154	0,1681	991,31	0,7171	**0,1780**	**0,1635**	1006,69
nrw1379	0,6240	0,3867	0,0977	**1142,73**	**0,5695**	**0,3125**	**0,0866**	1230,34	0,5767	0,3796	0,0925	1865,61
d2103	4,7966	3,7290	**0,6433**	**2974,38**	**4,7245**	3,0839	0,6813	3204,03	4,7790	**2,7582**	0,7066	3204,94
pcb3038	1,0279	**0,6849**	0,1308	**4053,72**	**1,0168**	0,7371	**0,1105**	4140,85	1,0325	0,6994	0,1266	6235,17
fnl4461	0,9915	0,8101	**0,0726**	7923,67	**0,9073**	0,6436	0,1269	**7584,19**	0,9169	**0,5850**	0,1374	7607,90
rl5915	5,8392	3,4653	**0,8215**	**11881,60**	5,7641	**3,2764**	0,8453	14627,40	**5,4553**	3,3383	0,9818	18082,20

Tabela 6.4: Comparação entre valores dos coeficientes c1 e c2

	c1 = c2 = 0,2				**c1 = c2 = 0,5**				**c1 = c2 = 0,8**			
Bases	Med(%)	Min(%)	DP(%)	T(s)	Med(%)	Min(%)	DP(%)	T(s)	Med(%)	Min(%)	DP(%)	T(s)
pr76	0,0000	0,0000	0,0000	**6,58**	0,0000	0,0000	0,0000	6,85	0,0000	0,0000	0,0000	6,68
rat195	0,0057	0,0000	0,0220	34,70	0,0029	0,0000	0,0108	43,71	**0,0000**	0,0000	**0,0000**	**26,81**
kroA200	0,0000	0,0000	0,0000	15,55	0,0000	0,0000	0,0000	11,81	0,0000	0,0000	0,0000	**11,44**
pr299	0,0000	0,0000	0,0000	71,66	0,0000	0,0000	0,0000	48,28	0,0000	0,0000	0,0000	**46,64**
pr439	0,0605	0,0056	0,0441	438,86	0,0435	0,0000	0,0360	427,87	**0,0219**	0,0000	**0,0306**	**404,82**
d657	0,2881	0,1431	0,0747	**548,63**	0,2432	0,0593	0,0772	565,48	**0,1690**	**0,0552**	**0,0613**	570,68
pr1002	0,8179	0,5235	**0,1331**	**907,14**	0,7185	0,4370	0,1436	955,04	**0,4713**	**0,1606**	0,1431	986,43
nrw1379	0,6735	0,5862	**0,0508**	**1108,66**	0,6128	0,4220	0,0818	1227,70	**0,4685**	**0,2860**	0,1032	1283,40
d2103	4,7999	3,1299	0,7548	3611,93	4,8090	3,3524	**0,6832**	**3129,56**	**4,3316**	**1,9602**	1,0219	3307,01
pcb3038	1,0601	0,8236	**0,0970**	**3610,60**	0,9893	0,6965	0,1336	4114,33	**0,8102**	**0,5440**	0,1503	4583,49
fnl4461	1,0196	0,7586	**0,0803**	**6466,88**	0,9698	0,6934	0,0991	7492,66	**0,8498**	**0,6436**	0,1105	8521,46
rl5915	5,6625	3,0980	**0,9004**	**10051,00**	5,6494	3,0817	0,9616	11968,90	**5,0785**	**2,0220**	1,2088	13809,90

Capítulo 7

Uma Implementação Para o MKP

Tanto o TSP quanto o MKP são problemas combinatórios de otimização. Entretanto, eles possuem características bem distintas. No TSP, por exemplo, a solução pode ser expressa em forma de grafo (6.1), visto que a solução para o problema retrata uma sequência ordenada de cidades a serem visitadas, sendo cada cidade representada por um nó do grafo. Já no MKP, as soluções geralmente são modeladas através de um conjunto de variáveis binárias, já que cada item do problema pode ter dois estados dentro da solução: contido ou não contido.

Assim, os algoritmos de busca local e *Path relinking* que se aplicam ao TSP, não podem ser implementados da mesma forma para o MKP, já que ambas são técnicas dependentes do problema.

Dessa forma, para o MKP, optou-se por desenvolver um algoritmo próprio de busca local ao invés de utilizar um método pronto da literatura, esse método é relatado a seguir na seção 7.2. E a implementação e adaptação do *Path reliking* para este problema é apresentada na seção 7.3.

7.1 Inércia (w)

No algoritmo PSOPR-MKP o coeficiente w possui função dupla. Ele foi utilizado para determinar o número máximo de tentativas de melhora da solução pelo algoritmo de busca local, e também o número de operações executadas pela busca local a cada tentativa.

Em relação ao número de tentativas, dada uma iteração qualquer, o algoritmo de busca local implementado, que será relatado na próxima seção, é executado repetidamente até que uma melhora ocorra na solução ou até que o número máximo de t tentativas sem melhora tenha se esgotado, tal que t é obtido por:

$$t = (int)\ w\ *\ n$$

Onde n é a quantidade de partículas do enxame.

Em relação ao total de operações feitas pelo algoritmo de busca local. O coeficiente w, que possui um valor atribuído entre 0 e 1, é multiplicado pelo número de itens contidos na solução em questão, resultando no número de operações que serão realizadas (mais detalhes na próxima seção).

7.2 PSOPR Local Search

O algoritmo de busca local desenvolvido neste trabalho foi chamado simplesmente PSOPR *Local Search* (PSOPR-LS) . Como toda busca local o algoritmo começa a partir de uma solução s, na qual são aplicadas várias pequenas alterações na tentativa de melhorar a qualidade de s, que por sua vez é medida através de $F(s)$.

O PSOPR-LS trabalha com um conceito de custo-benefício (CB). O CB de cada item é calculado apenas uma vez no início do algoritmo PSOPR, pois este não se altera ao longo da execução. Como se pode observar na equação 7.1, tal valor é obtido através da divisão do valor do item pela soma dos seus pesos em cada recurso.

$$cb_i = \frac{v_i}{\sum_{j=1}^{m} r_{ij}}, \quad i = 1, 2, ..., n \tag{7.1}$$

Para cada execução do algoritmo, são criadas duas listas ordenadas por CB, uma contendo todos os itens que estão contidos em s, chamada $\vec{L}^c$, e outra com todos os itens que não estão, chamada $\vec{L}^{nc}$.

O objetivo desta busca local é procurar inserir itens com um bom CB e retirar itens com um baixo CB. Para isso, neste trabalho foi criada uma Função para geração de Índices Pseudo Aleatórios (FIPA). Esta função recebe como parâmetro de entrada o coeficiente c_{exp}.

A FIPA gera um número inteiro $d = [0, n)$ a ser utilizado como índice, tal que d tenda a ser um valor mais próximo de 0 e mais distante de n:

$$d = (int)\ x^{c_{exp}} * n \ \mid x = rand() = [0, 1)$$

Isso ocorre porque ao elevar um número real x qualquer entre 0 e 1 a um expoente maior que 1, o resultado será sempre menor que o próprio x. Dessa forma atribuindo um valor aleatório inicialmente a x, e multiplicando o resultado de x elevado a c_{exp} pelo valor máximo que deseja ser gerado (n), a função irá tender a retornar um número mais baixo. Tal tendência é diretamente proporcional ao valor atribuído a c_{exp}.

O funcionamento da função FIPA é ilustrado pelo gráfico da Figura 7.1. Neste gráfico, foram definidos 3 valores para c_{exp}: 1, 2 e 2,5. Estes valores foram utilizados para o experimento de sensibilidade deste coeficiente, relatado a seguir na seção 7.4. Como se pode observar no gráfico, quanto maior o valor de c_{exp}, mais baixos são os valores de d correspondentes. Enquanto $c_{exp} = 1$ resulta em uma função linear o que torna a geração de índices completamente aleatória, pois, com esta configuração, valores altos de d passam a ter a mesma probabilidade de serem gerados que valores baixos.

Como as listas são ordenadas em ordem decrescente por CB, d é utilizado dire-

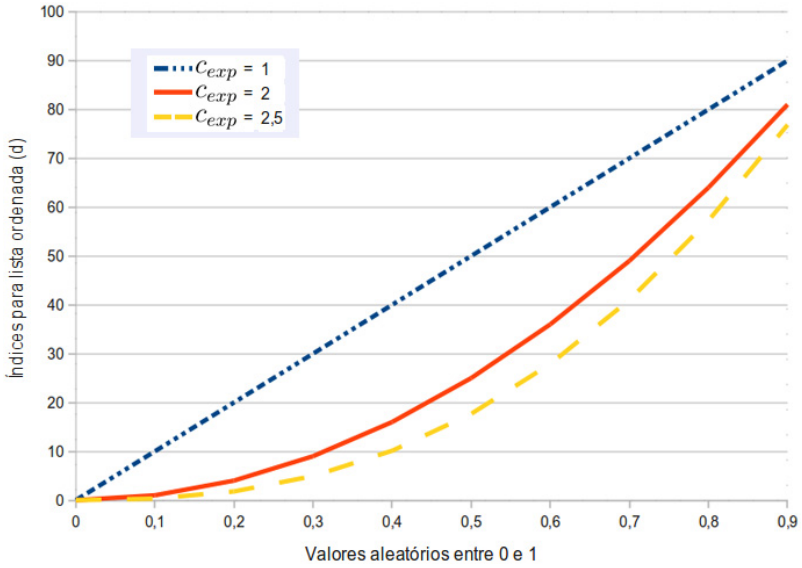

Figura 7.1: Gráfico da função FIPA para três valores de c_{exp}
Fonte: Autor (2010)

tamente como índice para a lista $\vec{L}^{nc}$, assim os primeiros itens da lista, ou seja, os de melhor CB, são escolhidos com mais frequência para serem inseridos em s. Mas isso sem excluir a possibilidade dos últimos itens também serem escolhidos. De forma semelhante é desejável retirar os itens contidos em s com os piores CB, assim $(n-1)-d$ é calculado e utilizado como índice para a lista $\vec{L}^{c}$.

Funções semelhantes à FIPA são utilizadas em algoritmos genéticos para escolha dos indivíduos que irão gerar descendentes para a próxima geração.

O cálculo do custo da solução é um procedimento relativamente caro computacionalmente, pois o peso de cada um dos itens tem que ser somado para cada uma das mochilas do problema. Para evitar que esse cálculo seja feito a cada vez que um item é inserido ou retirado da solução s, foi criado o vetor Δp. Este vetor é responsável por guardar os valores que representam a quantidade de peso que cada mochila ainda pode carregar além de todos os itens contidos em s. Assim no início da execução da busca local o Δp_j^s é preenchido calculando a diferença entre a capacidade l_j da mochila e o custo total de s para j. Este cálculo é feito de acordo com a equação 7.2.

$$\Delta p_j^s = l_j - \sum_{i=1}^{n} r_{ij} s_i, \quad j = 1,2,...,m \tag{7.2}$$

Da mesma forma que para os pesos, também foi criada uma variável Δv para os valores. Esta variável guarda a diferença de qualidade de s atual em relação ao s inicial. Portanto no início da execução do PSOPR-LS, Δv é iniciado com o valor 0 até que a primeira alteração em s seja feita. Depois Δv é atualizado através da soma dos valores dos itens inseridos e da subtração dos valores dos itens retirados de s.

Utilizando as variáveis Δ, durante a execução do algoritmo, uma grande quantidade de cálculos é poupada, visto que a cada alteração feita em s, ao invés de recalcular o custo e a qualidade da solução inteira, o algoritmo atualiza apenas os valores de Δp e Δv. Quando, por exemplo, um item i é incluso em s, os cálculos realizados consistem apenas na adição de v_i a Δv e os valores de r_i ao vetor Δp_i. E a operação inversa ocorre quando um item é retirado. Esta técnica foi desenvolvida baseada no trabalho de Applegate et al. (1999).

Algorithm 4 Pseudocódigo do algoritmo PSOPR-LS

1: ordena $\vec{L}^c$, $\vec{L}^{nc}$
2: inicia Δv e Δp
3: $operacoes = (int)\ (w * totalItens(\vec{L}^c)$
4: **for** $var = 1$ to operacoes **do**
5: **if** $\forall\ j\ \ \Delta p_j > 0$ **then**
6: $d = FIPA(c_{exp})$
7: inclui $\vec{L}^{nc}_d$ em s
8: **else**
9: $d = (totalItens(\vec{L}^c) - 1) - FIPA(c_{exp})$
10: exlui $\vec{L}^c_d$ de s
11: **end if**
12: atualiza Δp e Δv
13: **if** $\Delta v + F(s_{ini}) > F(P_{best})$ **then**
14: $P_{best} = s$
15: **end if**
16: **end for**
17: repara s
18: **if** $\Delta v < 0$ **then**
19: desfaz alterações em s
20: **end if**
21: **return** s

O pseudocódigo do PSOPR-LS (4) mostra o que ocorre a cada tentativa de melhora da solução s. Primeiramente as listas são ordenadas, as variáveis Δ iniciadas, e o

total de operações que serão realizadas é calculado através da multiplicação do coeficiente w pelo número de itens contidos em s. Então para cada operação o algoritmo verifica se a solução s está dentro do limite das mochilas (linha 5) e, em caso positivo, adiciona um item da lista $\vec{L}^{nc}$, caso contrário, retira um item da lista $\vec{L}^{c}$. Logo após as variáveis Δ são atualizadas. Então, na linha 13 é verificado se a soma de Δv com a $fitness$ de s antes das alterações ($F(s_{ini})$) é maior que a $fitness$ de P_{best}, ou seja, se as alterações realizadas em s até então foram capazes de levar a partícula à sua melhor posição já alcançada. Em caso positivo, P_{best} é atualizada.

Ao inserir o último item em s uma solução infactível pode ser gerada, nesse caso o algoritmo repara a solução (linha 17) retirando itens até tornar a solução válida novamente da mesma forma como ocorre na linha 9 e 10.

Caso todas as operações realizadas em s na tentativa vigente não tenha gerado nenhuma melhora, as operações são desfeitas e uma nova tentativa de melhora é executada conforme apresentado na seção anterior. Finalmente, o algoritmo parte para a execução dos componentes M2 e M3 através do algoritmo *Path relinking* relatado a seguir.

7.3 *Path Relinking* para o MKP

Diferentemente do TSP, no MKP as soluções são modeladas como sequências de variáveis binárias. Dessa forma, para o MKP, a alteração que ocorre a cada passo do algoritmo é feita através da mudança de estado em uma dada dimensão d.

Esta dimensão é escolhida aleatoriamente tal que $P^d_{ini} \neq P^d_{obj}$, ou seja, se o item d está contido em P_{ini}, ele é retirado, caso contrário, ele é colocado. Tal operação, chamada de inversão (*flip*), é mostrada na figura 7.2.

O algoritmo começa preenchendo duas listas, uma com todos os itens contidos em P_{obj} e que não estão contidos em P_{ini}, chamada $\vec{L}^{cs_t}$, e outra com todos os itens não contidos em P_{obj} e que estão contidos em P_{ini}, chamada $\vec{L}^{ncs_t}$.

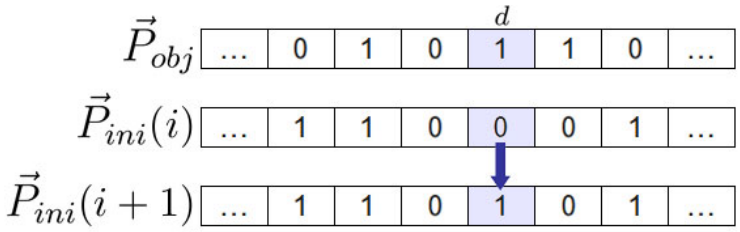

Figura 7.2: Operação de inversão (*flip*) dentro do *Path relinking*
Fonte: Autor (2010)

A quantidade de passos do algoritmo, nesse caso, é calculada da mesma forma que para o TSP, com a diferença de que para o MKP, a função de distância exata retorna exatamente o número de passos entre uma solução e outra e não somente uma estimativa. Isso ocorre porque cada alteração é realizada somente em uma dimensão de P_{ini}. Assim:

$$distExata(P_{ini}, P_{obj}) = totalItens(\vec{L}^{cs_t}) + totalItens(\vec{L}^{ncs_t})$$

Quanto à viabilidade das soluções, a cada vez que um item é adicionado a P_{ini}, a solução pode se tornar inválida transpondo a restrição do problema referente aos limites das mochilas. Dessa forma, a cada inversão, assim como na busca local, é verificado se nenhum limite de peso foi excedido, em caso positivo é feita a inclusão de item: um item é retirado aleatoriamente de $\vec{L}^{cs_t}$ e então utilizado para fazer a inversão em P_{ini}, caso contrário é feita a exclusão de item: o item utilizado para fazer a inversão em P_{ini} é escolhido a partir de $\vec{L}^{ncs_t}$.

O procedimento se repete $totalPassos$ vezes, mas pode continuar depois disso caso a última inversão tenha tornado P_{ini} uma solução inválida para o problema ultrapassando o limite de uma ou mais mochilas, neste caso a iteração continua com itens de $\vec{L}^{ncs_t}$ sendo escolhidos para a inversão, até tornar P_{ini} uma solução válida novamente.

7.4 Sensibilidade do Algoritmo

Da mesma forma como relatado anteriormente em 6.4, uma nova série de experimentos de sensibilidade foram realizados para o PSOPR-MKP. Nesta seção dois experimentos

distintos são relatados: o primeiro diz respeito à comparação entre duas topologias de vizinhança e o segundo relata as diferenças de desempenho entre configurações do coeficiente c_{exp}.

Em ambos os experimentos foram realizadas 30 execuções em 10 bases diferentes extraídas da ORLIB[1], mais precisamente dos arquivos mknapcb1.txt e mknapcb4.txt, dos quais foram utilizadas somente as cinco primeiras bases de cada arquivo.

Para tais execuções foram estipuladas 500 iterações e o tamanho do enxame foi configurado para ser igual ao número de itens (n) do problema. Tal configuração foi definida baseada no primeiro experimento do trabalho de Kong e Tian (2006). E o computador utilizado foi o mesmo para os experimentos do capítulo anterior.

Quanto aos parâmetros, após alguns testes preliminares, a inércia foi definida como 0,9, o coeficiente c_{pr} definido como 1, ou seja, todas as soluções geradas são avaliadas e os demais coeficientes ($c1$ e $c2$) foram mantidos com o valor 0,5, que no algoritmo *Path relinking* faz com que o ponto final tenda a ser a metade do caminho entre posição inicial e posição objetivo. A tabela 7.1 mostra a configuração padrão inicial (negrito) para os experimentos de sensibilidade do PSOPR-MKP em relação as topologias de vizinhança e ao coeficiente c_{exp}.

Tabela 7.1: PSOPR-MKP - Parâmetros testados nos experimentos de sensibilidade

Parâmetro	Valores testados
iterações	500
tamanho do enxame	n
c_{pr}	1,0
w	0,9
$c1$	0,5
$c2$	0,5
c_{exp}	1,0; **1,5**; 2,0; 2,5
topologia	**GCC**; Anel

[1]http://people.brunel.ac.uk/ mastjjb/jeb/info.html

7.4.1 Topologias de Vizinhança

A primeira versão do algoritmo PSOPR-MKP foi implementada com a topologia de vizinhança GCC. Ao analisar o comportamento deste algoritmo percebeu-se que a melhor solução da execução estava sendo encontrada logo nas primeiras iterações. Tal solução na maioria das vezes era um ótimo local. Ou seja, o algoritmo estava convergindo rapidamente para um ponto no qual não se conseguia mais melhoras.

Aparentemente, para o algoritmo em questão, este problema deve-se ao fato de que esta topologia propicia tornar todas as partículas demasiadamente parecidas entre si, pois a cada iteração todo o enxame é influenciado pela partícula G_{best}. Analogamente, é como se a cada momento houvesse apenas um líder (G_{best}) ditando o comportamento da população como um todo.

Como apresentado no capítulo 2, tal característica é oposta a de sistemas emergentes, assim optou-se por experimentar outra topologia a qual o comportamento geral do enxame fosse mais condizente com o modelo *bottom up*. Dessa forma, foi implementada a topologia em forma de anel, nesta topologia cada partícula é uma líder em potencial dependendo da posição atual das suas duas vizinhas diretas, então, a cada iteração, o algoritmo passa a ter várias líderes que influenciam e são influenciadas mutuamente e assim, o controle geral do enxame passa a ser descentralizado, uma característica de sistemas emergentes. Como resultado deste comportamento as partículas se mantém mais distantes umas das outras no espaço de busca gerando maior variabilidade de soluções.

Para analisar o comportamento das duas topologias durante as execuções, foi gerado o gráfico de convergência das duas versões do algoritmo ao longo das 500 iterações para uma das bases do experimento (10.100.03). Este gráfico (figura 7.3) mostra duas linhas que demonstram a média das melhores partículas geradas até o momento em todas as execuções de cada versão do algoritmo. Dessa forma é possível acompanhar a evolução da melhor solução encontrada e em que momento ocorre a estagnação do algoritmo. De acordo com o gráfico a topologia de anel fez com que as partículas convergissem mais suavemente até a melhor solução encontrada pelo enxame, tendo, assim, mais chances de

encontrar uma solução ótima global.

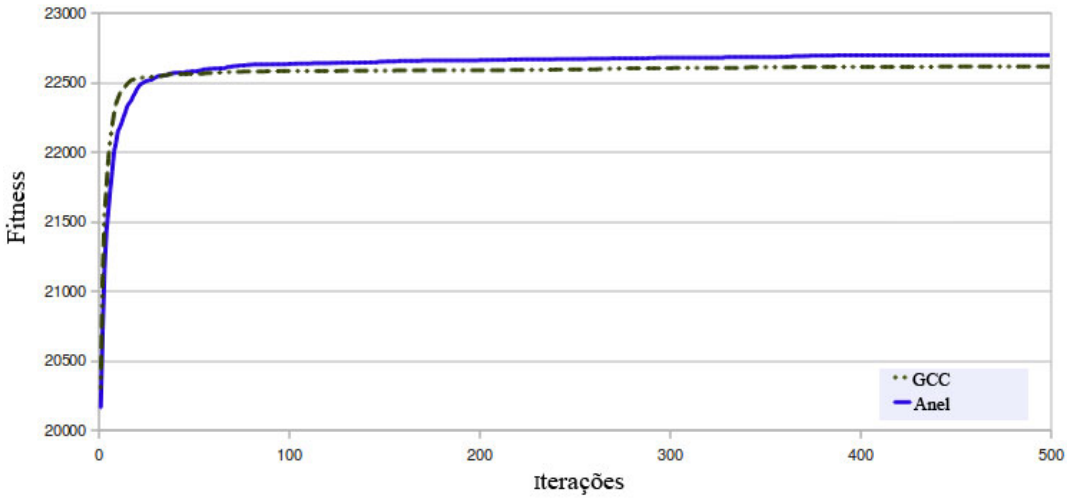

Figura 7.3: Gráfico de convergência comparativo entre as duas topologias
Fonte: Autor (2010)

O padrão apresentado pelo gráfico da figura 7.3 se repetiu para todas as outras bases do experimento, sendo que em algumas bases a diferença entre a qualidade das melhores médias das duas versões foi mais discrepante e em outras menos. Entretanto, de qualquer modo, para as 10 bases testadas todas as melhores médias foram alcançadas pela PSOPR-Anel.

Os dados deste experimento podem ser observados na tabela 7.2 que mostra o valor ótimo (VO) de cada base, a média aritmética (Med) dentre as melhores soluções encontradas em cada execução e o valor da melhor solução encontrada (Max) dentre todas as execuções. Os resultados mostram que além de a versão com topologia de anel ter gerado melhores médias, o PSOPR-GCC alcançou o valor ótimo em apenas seis bases, enquanto o PSOPR-Anel obteve em nove bases. Esta versão também se mostrou significativamente mais estável com valores de desvio padrão variando de 1,5 a 4,2 vezes mais baixos que a outra versão.

7.4.2 Coeficiente c_{exp}

Conforme apresentado na seção 7.2, o coeficiente c_{exp} foi criado para variar a intensidade da função FIPA em relação à geração de índices baixos para as listas ordenadas do PSOPR-

Tabela 7.2: Comparação entre as topologias de vizinhança

		Anel			GCC		
Bases	VO	Med	Max	DP	Med	Max	DP
5.100.00	24381	**24361,30**	**24381**	**22,86**	24330,10	**24381**	35,57
5.100.01	24274	**24272,00**	**24274**	**8,45**	24250,90	**24274**	36,02
5.100.02	23551	**23535,33**	**23551**	**6,17**	23532,63	**23551**	10,03
5.100.03	23534	**23474,53**	**23534**	**21,58**	23440,10	23527	47,62
5.100.04	23991	**23966,40**	**23991**	**8,93**	23951,70	**23991**	23,04
10.100.00	23064	**23050,43**	**23064**	**12,55**	23022,57	**23064**	41,00
10.100.01	22801	**22737,30**	**22801**	**31,93**	22683,70	22753	51,43
10.100.02	22131	**22089,53**	**22131**	**26,34**	22035,33	**22131**	57,26
10.100.03	22772	**22697,47**	**22772**	**43,54**	22615,90	22763	76,76
10.100.04	22751	**22642,50**	**22697**	**23,66**	22601,73	22654	54,53

LS.

Para o experimento de sensibilidade deste coeficiente, várias versões foram testadas, dentre elas os resultados mais relevantes foram obtidos através de três configurações diferentes cujos dados são mostrados na tabela 7.3.

Tabela 7.3: Comparação entre valores de c_{exp}

	$c_{exp} = 1.0$		$c_{exp} = 2.0$		$c_{exp} = 2.5$	
Bases	Med	DP	Med	DP	Med	DP
5.100.00	24327,10	26,52	24367,83	**18,67**	**24369,73**	19,18
5.100.01	24244,93	44,71	**24274,00**	**0,00**	**24274,00**	**0,00**
5.100.02	23523,33	14,47	23538,10	4,53	**23538,87**	**3,30**
5.100.03	23446,17	42,92	**23488,63**	17,86	23486,73	**14,89**
5.100.04	23962,70	11,20	**23967,93**	9,62	23965,70	**9,21**
TP	02:46:38		**02:40:32**		04:44:49	
10.100.00	23032,30	37,04	23050,80	16,28	**23051,70**	**15,61**
10.100.01	22663,07	60,47	**22751,90**	**22,51**	22749,10	28,71
10.100.02	22048,13	51,10	**22095,17**	28,43	22086,77	**23,51**
10.100.03	22611,77	70,38	**22702,73**	**47,46**	22702,53	52,71
10.100.04	22616,63	42,89	**22662,27**	39,69	22650,40	**32,65**
TP	03:43:03		**03:42:17**		04:06:59	

Na primeira configuração ($c_{exp} = 1$), o objetivo foi verificar qual o grau de influência deste parâmetro para os resultados e também se o método de escolha de dimensões para realizar a inversão efetivamente funciona. Isso porque com $c_{exp} = 1$, conforme apresentado em 7.2 através do gráfico da Figura 7.1, a FIPA (equação 7.2) retorna um índice

uniformemente aleatório, ao invés de um índice pseudo aleatório. Nesse sentido o objetivo foi atingido, visto que, analisando os dados da tabela 7.3 pode se constatar que os resultados dessa configuração foram visivelmente inferiores aos demais em relação à média. Além disso, a aleatoriedade desta configuração levou o algoritmo a gerar resultados bastante diferentes em termos de qualidade para cada execução de uma mesma base, isso se reflete nos altos valores de desvio padrão em relação às outras configurações.

As outras duas configurações foram consideradas as mais relevantes pelo fato de terem conseguido atingir o valor ótimo para todas as bases do experimento em pelo menos uma das execuções. Dentre estas a configuração de busca local mais intensiva ($c_{exp} = 2.5$) se mostrou mais estável com sete valores de desvio padrão inferiores às demais configurações, apesar disso, a configuração 2.0 apresentou as melhores médias em sete das dez bases do experimento. Além disso, a configuração $c_{exp} = 2.0$ se mostrou significativamente mais rápida, com 56% do tempo da $c_{exp} = 2.5$ para execução das cinco primeiras bases, e 90% do tempo de processamento para as 5 últimas (maiores bases).

Outras configurações com valores maiores que 2.5 foram testadas e, de forma geral, os resultados apresentaram baixos valores de desvio padrão e algumas médias ainda maiores que as de 2.0 e 2.5, porém em várias bases, geralmente as maiores, tais configurações não foram capazes de atingir o valor ótimo com a mesma quantidade de execuções. Provavelmente, um valor muito alto para este coeficiente propicia a rápida convergência e, consequentemente, a geração de soluções provenientes de ótimos locais.

Como se pode observar na tabela 7.3, a configuração $c_{exp} = 2.5$ conseguiu melhores médias para 3 das menores bases (100.5.0/1/2) enquanto a configuração $c_{exp} = 2.0$ atingiu 4 melhores médias dentre as maiores bases (100.10.1/2/3/4). Analisando tal comportamento e também baseado nos resultados obtidos a partir de outras configurações, é possível supor que valores mais altos para c_{exp} no geral são melhores para bases menores enquanto valores mais baixos (desde que $c_{exp} > 1$) fazem com que o algoritmo se adapte melhor a bases maiores as quais o espaço de busca é mais extenso e, portanto, uma convergência mais lenta pode propiciar maiores chances de se atingir o ótimo global.

7.4.3 Configuração final padrão

Após análise dos resultados dos experimentos de sensibilidade realizados até então, percebeu-se que o algoritmo já estava gerando resultados competitivos, não havendo necessidade de reajustar os coeficientes $c1$, $c2$ e w. Assim a configuração padrão dos parâmetros do algoritmo PSOPR-MKP foi definida de acordo com a lista abaixo:

- tamanho do enxame = n;
- c_{pr} = 1,0;
- w = 0.9;
- $c1$ = 0.5;
- $c2$ = 0.5;
- c_{exp} = 2.0;
- topologia = anel.

Onde n é o total de itens do problema. Esta configuração foi utilizada para os experimentos de comparação relatados em 8.2.

Capítulo 8

Resultados

Ambas as versões do algoritmo descrito neste trabalho foram implementadas em C++ devido ao fato desta linguagem suportar Programação Orientação a Objeto (POO), provendo reusabilidade e organização de código. Isso facilitou a implementação do algoritmo para os dois problemas citados que, apesar de pertencerem a uma mesma categoria de problemas (CO), possuem características bastante distintas. Além disso, o C++ é capaz de prover um bom desempenho em termos de tempo de processamento.

Esta seção relata os experimentos realizados para comparar cada uma das duas versões do PSOPR com outros trabalhos da literatura. Ao todo, foram realizados seis experimentos, quatro para o PSOPR-TSP e dois para o PSOPR-MKP. E os resultados foram comparados com sete trabalhos relacionados.

8.1 Experimentos com o PSOPR-TSP

Alguns trabalhos relacionados executam experimentos contando o número de soluções completas geradas durante a execução ao invés de apenas informar o número de partículas da população e o número de iterações. Isso faz sentido na medida em que uma única iteração pode fazer diversas alterações em cada uma das suas partículas, assim como pode fazer poucas alterações.

Conforme relatado anteriormente uma simples alteração em qualquer uma das dimensões da posição da partícula gera uma nova solução para o problema. Assim, o número de soluções geradas durante toda a execução poderia variar muito entre duas abordagens de algoritmos PSO mesmo que ambas tivessem a mesma configuração em relação ao número de partículas e iterações.

Um bom exemplo disso é o que acontece no caso do algoritmo *Path relinking* do PSOPR-TSP. Durante a execução do algoritmo diversas soluções intermediárias são criadas e avaliadas dependendo do coeficiente c_{pr}. Por isso para comparar de forma justa os resultados do PSOPR com outros trabalhos relacionados na área foi implementado um contador que é incrementado a cada vez que uma nova solução é avaliada. Assim o critério de parada foi definido como sendo o limite pré-estipulado de soluções geradas ou, simplesmente, encontrar a melhor solução conhecida para o problema.

Nesta seção serão relatados alguns experimentos nos quais serão feitas comparações com outros algoritmos PSO discretos e também com o algoritmo de Otimização por Colônia de Formigas.

8.1.1 Clerc (2004)

A primeira comparação foi feita com o trabalho de Clerc (2004). Conforme apresentado no capítulo 3, Clerc (2004) propôs um algoritmo PSO discreto baseado em listas de transposições. Esse algoritmo foi chamado *Discrete Particle Swarm Optimization* (DPSO) e está disponível para *download* [1].

Todas as configurações do DPSO foram mantidas exatamente de acordo com o algoritmo padrão, exceto pela modificação de dois parâmetros: O Nmax (máximo de nós do grafo) e o Max_size (máximo de partículas da população). O Primeiro teve o valor aumentado de 51 para 150 para que o algoritmo pudesse suportar bases TSP maiores. Enquanto o segundo teve seu valor aumentado de 601 pra 800 porque neste algoritmo, o cálculo do tamanho da população é dinâmico variando de acordo com a instância e dessa

[1] http://clerc.maurice.free.fr/pso/

forma o algoritmo pode gerar populações maiores para bases maiores sem prejudicar a qualidade das soluções por conta de uma possível disparidade entre tamanho da instância do problema e o tamanho da população.

Concluídos os ajustes do algoritmo necessários para início dos experimentos, foi necessário definir o parâmetro responsável por guardar um número máximo de avaliações para cada execução do sistema. A ideia era configurar um número elevado de avaliações para que os resultados do DPSO não fossem prejudicados por conta disso. Por isso este parâmetro foi definido como $2500n$, onde n é o número de cidades do problema. Este valor foi utilizado por Li et al. (2006) e pode ser considerado uma quantidade razoavelmente grande de avaliações por execução comparada com experimentos realizados em outros trabalhos relacionados como Goldbarg, Souza e Goldbarg (2006), Machado e Lopes (2005) e Wang et al. (2003).

Além das configurações padrão o algoritmo DPSO possui ainda quatro variantes em relação à forma como as partículas se movimentam. Neste trabalho elas foram chamadas DPSO(0), DPSO(1), DPSO(2), DPSO(3). Para o experimento foram utilizadas oito bases diferentes da TSPLIB e cada algoritmo foi executado 10 vezes.

Tabela 8.1: Comparação com DPSO

		PSOPR		DPSO(0,1,2,3)		DPSO0	DPSO1	DPSO2	DPSO3
Bases	VO	Med	T(s)	Min	Med	T(s)	T(s)	T(s)	T(s)
bays29	2020	**2020**	**1**	2035	2035	804	525	566	791
att48	10628	**10628**	**4**	11316	11316	1618	2219	1439	1512
brazil58	25395	**25395**	**8**	26781	26781	3183	3190	2072	2434
eil76	538	**538**	**2**	580	580	3767	7734	3712	5426
gr96	55209	**55209**	**8**	60349	60710,6	8473	8316	5655	5757
lin105	14379	**14379**	**3**	15523	15523	7803	7117	6694	6777
pr124	59030	**59030**	**17**	63339	63339	10040	9902	9316	9902
pr144	58537	**58537**	**112**	60758	60758	19533	12595	19769	13181

O resultado do experimento é apresentado na tabela 8.1. Para cada base, a melhor solução conhecida (VO), a média e o tempo de processamento (T) do PSOPR, a média do DPSO (mesmo resultado para as quatro versões) e o melhor tempo de processamento entre as quatro versões. Apenas as médias foram apresentadas porque os resultados foram

os mesmos para todas as execuções, resultando em desvio padrão igual a 0 para todas as bases, exceto para a base gr96 cujo desvio padrão foi igual a 190,58 em todas as versões do DPSO. Obviamente, a disparidade em relação ao tempo de processamento dos dois algoritmos é devido ao fato do PSOPR ter sido capaz de chegar rapidamente a melhor solução conhecida, não necessitando assim das $2500n$ avaliações.

Os resultados do algoritmo original de Clerc (2004) não se mostraram competitivos. Porém, tal constatação não foi surpreendente, uma vez que o próprio autor caracteriza os resultados obtidos pelo seu algoritmo como sendo "nem particularmente bons nem ruins". Apesar disso, cientificamente, a importância desse trabalho é inquestionável, pois tem sido utilizado como base e inspirado diversos outros trabalhos na área, tais como Li et al. (2006), Shi et al. (2007), Liang et al. (2006) Reyes-Sierra e Coello (2006).

8.1.2 Machado e Lopes (2005)

Os resultados do primeiro experimento foram motivantes, mas como apresentado na seção anterior, o objetivo do trabalho de Clerc (2004) não era um algoritmo competitivo e sim a apresentação de uma nova abordagem não hibrida de PSO para problemas discretos. Assim, tornou-se necessário a realização de um novo experimento cujo objetivo fosse a comparação com um algoritmo PSO que já tivesse apresentado resultados competitivos e que, de preferência, assim como o PSOPR, utilizasse alguma técnica de busca local internamente.

Dessa forma, foi escolhido o trabalho de Machado e Lopes (2005) para a segunda bateria de experimentos de comparação. Neste trabalho Machado e Lopes (2005) implementaram um algoritmo híbrido baseado em PSO, algoritmos genéticos - *Genetic Algorithm* (GA) e *Fast local search* (FLS), assim no presente trabalho o algoritmo de Machado e Lopes (2005) foi chamado PSOGAFLS.

O GA é uma técnica de computação evolutiva proposta por John Holland em meados dos anos 60 (HOLLAND, 1975). Possivelmente a mais estudada e difundida das metaheurísticas, foi desenvolvida baseada na teoria da evolução das espécies de Darwin

(1872).

O FLS foi proposto por Voudouris e Tsang (1999), e é também conhecida como *Fast Hill-Climbing* pelo fato de apresentar resultados semelhantes ao *Hill-Climbing* mas, ao mesmo tempo, ser mais eficiente, levando em conta o tempo de processamento.

No experimento relatado por Machado e Lopes (2005) foram feitas 10 execuções e o número máximo de iterações por execução foi definido como 1200. O tamanho do enxame foi estipulado como 20 partículas. Tais configurações foram aplicadas ao PSOPR para a devida comparação e todos os dados provenientes do algoritmo PSOGAFLS foram extraídos diretamente da publicação.

Tabela 8.2: Comparação com PSOGAFLS

	PSOPR		PSOGAFLS	
Base	Med(%)	Min(%)	Med(%)	Min(%)
pr76	**0,000**	**0,000**	**0,000**	**0,000**
ratl95	**0,000**	**0,000**	1,148	0,810
pr299	**0,000**	**0,000**	0,620	0,120
pr439	**0,000**	**0,000**	0,500	0,280
d657	**0,006**	**0,002**	3,193	2,114
prl002	**0,065**	**0,000**	4,715	3,569
rll304	**0,211**	**0,004**	2,498	1,454
d2103	**1,000**	**0,000**	4,524	3,433

Os resultados do experimento apresentados na tabela 8.2 foram bastante satisfatórios. O PSOPR alcançou o melhor resultado conhecido em 6 de 8 bases, sendo que em quatro delas este resultado foi alcançado em todas as execuções, enquanto o PSOGAFLS conseguiu atingir o valor ótimo apenas na menor base (pr76). Comparando a percentagem das médias obtidas por um algoritmo em relação ao outro, é possível observar que a menor diferença entre os resultados foi obtida na base d2104, e mesmo nesta base, a diferença pode ser considerada grande, com o PSOPR alcançando 22,1% do valor obtido pelo PSOGAFLS.

8.1.3 Chen et al. (2009) e Dorigo e Gambardella (1997)

Outro algoritmo usado para comparação é o S-PSO de Chen et al. (2009), apresentado no capítulo 4. Nesse trabalho também existem algumas variações do algoritmo principal, como o S-CLPSO, baseado no algoritmo CLPSO, proposto por Liang et al. (2006). Essa versão tem dois parâmetros adicionais: a probabilidade de aprendizagem Pc e outro parâmetro chamado *refreshing gap* (rg), utilizadas de acordo com Liang et al. (2006) para garantir que a partícula aprenda com os bons exemplares (posições alcançadas) e para minimizar o tempo desperdiçado em direções ruins.

Outra característica interessante desta versão do algoritmo de Chen é que a inércia também foi implementada como um coeficiente dinâmico que varia linearmente de 0,9 a 0,4. Outro motivo que levou a escolha deste algoritmo para efetuar experimentos de comparação foi o fato de ele ter sido comparado diretamente com outro algoritmo de metaheurística, o ACO (Otimização por Colônia de Formigas), cuja comparação com o PSOPR já estava prevista na proposta desta dissertação. Mais especificamente o algoritmo utilizado por Chen para a comparação foi o Ant Colony System (ACS) de Dorigo e Gambardella (1997). O ACS é uma das principais variantes do ACO juntamente com o Ant System (AS) (o primeiro algoritmo proposto do gênero) (DORIGO, 1992) e o MAX-MIN Ant System (MMAS) (STÜTZLE; HOOS, 1997).

O primeiro algoritmo de Otimização por Colônia de Formigas foi apresentado à comunidade científica em 1992, por Dorigo, em sua tese de doutorado (DORIGO, 1992). Originalmente projetado baseado no comportamento de formigas reais, o ACO consiste em um algoritmo de inteligência computacional cooperativo, no qual formigas visam encontrar o menor caminho entre o ninho e a comida (STÜTZLE; HOOS, 1997) e cada caminho representa uma solução completa para o problema tratado.

O escopo do presente trabalho não contempla detalhes sobre algoritmos ACO, mas é importante ressaltar, que conforme formalizado por Dorigo e Caro (1999) e Dorigo, Caro e Gambardella (1999), esta é uma metaheurística de otimização combinatória, inicialmente projetada para o TSP. Vários pesquisadores já comprovaram a sua eficiência para o TSP

e outros problemas discretos. Este fato motivou a comparação com o PSOPR, pois, conforme apresentado no capítulo 4, ao contrário do ACO, o PSO foi desenvolvido para problemas cujo espaço de busca é contínuo e precisa ser adaptado para lidar com problemas discretos.

Para o experimento, foram definidos os mesmos critérios utilizados em Chen et al. (2009), com 50 execuções para cada base. O limite estipulado para cada execução foi um número máximo de soluções geradas. Este número varia de base para base sendo proporcional a n (tamanho da base).

A tabela 8.3 apresenta o resultado do experimento contendo o número máximo de soluções por base (Max Soluções), melhor solução conhecida (VO) e médias de cada algoritmo após as 50 execuções.

Tabela 8.3: Comparação com S-CLPSO e ACO (média de execuções)

Bases	Max Soluções	VO	PSOPR	S-CLPSO	ACO
d493	246500	35002	**35002,0**	37028,83	37120,97
d657	328500	48912	**48913,0**	51799,53	52098,87
u724	362000	41910	**41910,0**	43373,3	43405,6
d1291	645500	50801	**50918,63**	52353,1	53122,3
fl1400	700000	20127	**20127,0**	22025,2	21727,3
fl1577	788500	22249	**22585,3**	23038,4	23107,5

É válido observar que os resultados dos algoritmos S-CLPSO e ACO mostrados na tabela 8.3 foram extraídos diretamente de (CHEN et al., 2009). Como se pode observar os resultados foram satisfatórios novamente, o PSOPR foi capaz de atingir médias melhores para todas as seis bases testadas, sendo que na metade delas, o algoritmo gerou o melhor resultado conhecido em todas as execuções.

8.1.4 Goldbarg, Souza e Goldbarg (2006)

No quarto e último experimento de comparação do PSOPR-TSP foi utilizado o trabalho de Goldbarg, Souza e Goldbarg (2006) (dados extraídos da publicação). Conforme relatado anteriormente no capítulo 4, o trabalho compara duas versões do algoritmo PSO

com diferentes buscas locais: o PSO-INV com *Inversion Procedure* e o PSO-LK com o algoritmo LK. No experimento relatado em Goldbarg, Souza e Goldbarg (2006) foram utilizadas 11 bases da TSPLIB com tamanho variando de 51 a 2103 cidades e realizadas 20 execuções independentes com os seguintes critérios de parada: encontrar o valor ótimo, encontrar um número máximo de iterações definido como 2000 ou um máximo de 20 iterações sem melhora da melhor solução encontrada até o momento.

Para comparação com o PSOPR, os mesmos critérios e parâmetros citados foram adotados. Os resultados são apresentados na tabela 8.4. O PSO-INV apresentou médias significativamente inferiores em todas as bases em relação aos outros algoritmos. Quanto a comparação entre os dois algoritmos que utilizam LK, nas 5 maiores bases (últimas) o PSO-LK apresentou resultados superiores, tanto de médias quanto de mínimos, em relação ao PSOPR e nas demais bases conseguiu atingir o VO em todas execuções.

Tabela 8.4: Comparação com Goldbarg, Souza e Goldbarg (2006)

	PSO-INV		PSO-LK		PSOPR	
Base	Med(%)	Min(%)	Med(%)	Min(%)	Med(%)	Min(%)
eil51	1,9836	0,2347	**0,0000**	**0,0000**	**0,0000**	**0,0000**
berlin52	2,0041	**0,0000**	**0,0000**	**0,0000**	**0,0000**	**0,0000**
eil76	4,5167	2,4164	**0,0000**	**0,0000**	**0,0000**	**0,0000**
rat195	8,7581	5,8114	**0,0000**	**0,0000**	0,0043	**0,0000**
pr299	7,9952	5,8476	**0,0000**	**0,0000**	**0,0000**	**0,0000**
pr439	8,0111	4,4200	**0,0000**	**0,0000**	0,0183	**0,0000**
d657	9,6157	6,9656	**0,0000**	**0,0000**	0,1512	0,0184
pr1002	11,1900	9,8574	**0,0000**	**0,0000**	0,4048	0,1575
d1291	15,5505	13,2104	**0,0113**	**0,0000**	8,5682	3,7972
rl1304	11,9942	10,4432	**0,0000**	**0,0000**	1,4802	0,3443
d2103	18,4180	16,7383	**0,0267**	**0,0087**	4,5242	2,3145

A principal diferença entre os dois algoritmos é justamente a forma de utilização da busca local. Ambos utilizam a mesma versão do LK (Concorde TSP Solver) porém o PSO-LK utiliza muito mais a busca local LK do que o PSOPR, visto que o primeiro algoritmo tende a escolher a busca local a cada iteração com maior frequência que os outros dois caminhos. Além disso no PSO-LK a busca local é efetuada por completo enquanto no PSOPR apenas uma pequena porcentagem de cidades, determinada pelo coeficiente w, é utilizada como base das iterações do LK.

No experimento a respeito da sensibilidade do coeficiente de inércia (6.4), mostrado pela tabela 6.3, a versão com o maior valor de w, ou seja, que utiliza mais o LK que as outras, gerou os melhores resultados. Tal constatação, juntamente com a superioridade apresentada neste experimento pelo PSO-LK e a discrepância entre os resultados comparado a outras técnicas levam a crer que o algoritmo LK é comprovadamente uma das mais eficientes formas de abordar o TSP simétrico.

8.2 Experimentos com o PSOPR-MKP

Nesta seção são relatados dois experimentos de comparação com algoritmos PSO aplicados ao MKP. O primeiro relata a comparação com Hembecker, Lopes e Godoy (2007). E o segundo apresenta a comparação direta com dois outros algoritmos da literatura recente, Chen et al. (2009) e Kong e Tian (2006).

8.2.1 Hembecker, Lopes e Godoy (2007)

Hembecker, Lopes e Godoy (2007) relatam a aplicação de um algoritmo PSO (aqui chamado PSOH) a 10 bases do MKP. Tais bases possuem três configurações diferentes: 30 mochilas com 60 itens, 2 mochilas com 28 itens e 2 mochilas com 105 itens. Foram feitas 100 execuções para cada base com 300 iterações por execução. E o melhor resultado obtido em cada base foi extraído para o experimento.

Entretanto, o tamanho da população, um dos parâmetros referentes ao PSOH, foi omitido em Hembecker, Lopes e Godoy (2007). Assim, esse parâmetro foi definido como o número de itens do problema, o mesmo critério utilizado por Chen et al. (2009) e Kong e Tian (2006). Visto que as bases utilizadas neste experimento são relativamente pequenas, o PSOPR não levaria vantagem adicional devido a esta configuração. Além disso, para garantir que não houvesse tal vantagem o número de iterações do PSOPR foi definido para 20% do valor estipulado no PSOH.

Dessa forma um número máximo de apenas 60 iterações foi definido para o

PSOPR. Ainda assim, como mostra a tabela 8.5 (valores de PSOH extraídos da publicação), este valor foi suficiente para o PSOPR alcançar o valor ótimo para todas as bases e assim superar todos os melhores resultados do PSOH cujos valores apresentam uma diferença média de 4,38% em relação aos resultados do PSOPR. Além dos melhores resultados, a tabela 8.5 mostra ainda, para cada base, o número de mochilas (m), o número de itens (n) e também o melhor valor conhecido (VO).

Tabela 8.5: Comparação com PSOH (melhores resultados)

Bases	m	n	VO	PSOPR	PSOH
sento1	30	60	7772	**7772**	7725
sento2	30	60	8722	**8722**	8716
weing1	2	28	141278	**141278**	138927
weing2	2	28	130883	**130883**	125453
weing3	2	28	95677	**95677**	92297
weing4	2	28	119337	**119337**	116622
weing5	2	28	98796	**98796**	93678
weing6	2	28	130623	**130623**	128093
weing7	2	105	1095445	**1095445**	1059560
weing8	2	105	624319	**624319**	492347

8.2.2 Kong e Tian (2006) e Chen et al. (2009)

Ao lidar com problemas combinatórios modelados como uma sequência binária, como o MKP, a alteração de apenas um bit pode tornar toda a solução inválida para o problema tratado. Existem várias técnicas para contornar este problema de forma a manter a viabilidade das soluções. Uma delas é a função de penalização, nesta técnica não há restrição quanto à viabilidade das soluções durante as iterações, as restrições das mochilas são inteiramente manifestadas pela função de penalização que torna as partículas em posições inválidas demasiadamente desqualificadas e assim com pouca chance de influenciar outras partículas em sua direção.

Outra técnica consiste na utilização de um operador de reparação, nesta técnica quando uma solução se torna inviável, um operador de reparo desta solução é utilizado para torná-la viável novamente. Esta técnica desenvolvida baseada em Chu e Beasley

(1998), utiliza Programação Linear (PL) para definir uma lista u com pesos para cada uma das dimensões. O operador então começa retirando os itens com menor valor de u até atingir uma solução plausível e, depois que isso ocorre, o processo é repetido inversamente, adicionando os itens com maior valor para u.

Kong e Tian (2006) foi o primeiro trabalho a relatar aplicação do PSO ao MKP e um dos principais objetivos deste trabalho foi comparar as duas técnicas relatadas. Assim, em Kong e Tian (2006) foram desenvolvidos dois algoritmos, um PSO utilizando a função de penalização (PSO-P) e outro utilizando o operador de reparação (PSO-R).

Conforme explicado no capítulo 7 o problema da viabilidade das soluções foi resolvido utilizando listas ordenadas por ordem de custo-benefício da mesma forma como u é utilizada pelo operador de reparação no algoritmo PSO-R. Mas diferentemente da forma como é aplicada a técnica de operador de reparação, no PSOPR não foi utilizada PL. Além disso, a seleção dos itens a serem retirados da solução para torná-la viável não ocorre de forma determinística como em Kong e Tian (2006) e sim através de um método pseudo aleatório (equação 7.2).

Chen et al. (2009), citado anteriormente na comparação com o PSOPR-TSP, também implementou seu algoritmo ao MKP. Ele utilizou o trabalho de Kong e Tian (2006) para comparação, desenvolvendo, da mesma forma, duas versões do seu algoritmo, cada uma utilizando uma das técnicas de viabilidade utilizadas por Kong e Tian (2006), assim a versão que utiliza a função de penalização foi chamada S-CLPSO-V1 e a versão com o operador de reparação foi chamada S-CLPSO-V2. No geral em ambos os trabalhos a versão com o operador de reparação gerou melhores resultados.

Para comparação de ambos os algoritmos com o PSOPR, foi gerado um experimento com as mesmas bases e configurações utilizadas por Kong e Tian (2006) e Chen et al. (2009). Tais configurações se caracterizam pelas mesmas configurações utilizadas nos experimentos de sensibilidade do PSOPR-MKP com a exceção do número de iterações de cada execução que neste caso foi configurado como 2000 iterações.

Para permitir a comparação direta dos resultados obtidos pelas quatro versões,

os melhores valores dentre as duas versões de cada algoritmo foram extraídos. A tabela 8.6 mostra o resultado deste experimento. A coluna PSO-K mostra os melhores valores obtidos pelas duas versões do algoritmo de Kong e Tian (2006) e o S-CLPSO representa o mesmo em relação a Chen et al. (2009).

Tabela 8.6: Comparação entre melhores valores e médias dos algoritmos de Kong e Tian (2006) e Chen et al. (2009)

		Kong		Chen		PSOPR	
Bases	VO	Med	Max	Med	Max	Med	Max
5.100.00	24381	24356	**24381**	24356	**24381**	**24364**	**24381**
5.100.01	24274	24036	24258	24213	**24274**	**24274**	**24274**
5.100.02	23551	23523	**23551**	23530	**23551**	**23540**	**23551**
5.100.03	23534	23481	23527	23478	23527	**23490**	**23534**
5.100.04	23991	23966	23966	23963	**23991**	**23967**	**23991**
10.100.00	23064	23050	23057	23051	23057	**23057**	**23064**
10.100.01	22801	22668	22781	22725	**22801**	**22758**	**22801**
10.100.02	22131	22029	**22131**	22073	**22131**	**22093**	**22131**
10.100.03	22772	22733	**22772**	**22741**	**22772**	22723	22763
10.100.04	22751	22632	**22751**	22605	22697	**22677**	**22751**

Conforme relatado na tabela 8.6 os resultados deste experimento foram bastante favoráveis para o PSOPR. Em nove das dez bases do experimento, o PSOPR conseguiu todos os valores máximos e médias melhores que os outros algoritmos. Inclusive, as melhores soluções se igualaram ao valor ótimo em tais bases.

Capítulo 9

Conclusão

Neste trabalho, um modelo de PSO discreto foi proposto. Nesta abordagem as posições das partículas são atualizadas através da aplicação da sua velocidade de acordo com a equação de velocidade proposta no primeiro algoritmo PSO, ou seja, todos os componentes de movimentação são aplicados à partícula a cada iteração, não havendo escolha entre um caminho ou outro. Tal proposta considera busca local para movimentação de acordo com sua própria experiência e *Path relinking* para movimentação de acordo com a experiência de outras partículas.

O objetivo principal do trabalho consistiu na criação de um meta-modelo capaz de gerar resultados satisfatórios ao abordar diferentes COPs. Para tal, foi necessário distinguir entre a parte geral da proposta da parte específica que se aplica somente ao problema abordado.

Assim, no capítulo 5 o modelo foi apresentado como uma plataforma genérica para problemas combinatórios sem citar nenhum COP especificamente. E para validação e experimentação do modelo proposto, dois conhecidos COPs da literatura foram escolhidos para implementação do algoritmo: o TSP e o MKP. Os detalhes e características específicas das duas implementações, juntamente com uma série de experimentos de sensibilidade dos algoritmos em relação aos coeficientes, foram descritos nos capítulos 6 e 7.

Conforme abordado na seção 5.1, uma característica marcante dos algoritmos de busca local consiste na obrigatoriedade em relação à melhora das soluções a cada iteração. Entretanto tal característica pode propiciar a rápida convergência e consequentemente a estagnação da solução em um ótimo local do espaço de busca. Por outro lado, metaheurísticas baseadas em comportamentos sociais naturais como GA, PSO, ACO não possuem um critério rigoroso em relação à melhora da solução o que propicia uma exploração mais diversificada do espaço de busca, mas ao mesmo tempo, pode guiar a solução por caminhos que não virão a gerar nenhuma melhora. Justamente devido a tais características opostas que se completam, a união destas duas categorias de algoritmos pode propiciar uma excelente alternativa ao lidar com problemas complexos.

Ainda em relação aos algoritmos de busca local, para o TSP, foi utilizado o algoritmo LK, já para o MKP, uma nova busca local foi desenvolvida neste trabalho especificamente para este problema. Este algoritmo foi chamado PSOPR-LS e se baseia na escolha pseudo aleatória de itens a partir de uma lista ordenada por custo-benefício.

Em um dos experimentos de sensibilidade do PSOPR-MKP, duas topologias de vizinhança foram testadas através de duas versões diferentes do algoritmo: uma implementada com a topologia GCC e a outra com a topologia Anel. Conforme apresentado na seção 2.1 a GCC simula uma liderança contínua, pois, como todas as partículas são vizinhas entre si, a cada momento uma única partícula influencia toda a população. Enquanto a topologia em forma de Anel possui um caráter mais emergente, visto que cada partícula é vizinha somente de outras duas partículas e, assim, a liderança é fragmentada dentre os membros da população. Neste experimento um gráfico com a curva de convergência de cada versão do algoritmo foi gerado a partir dos dados extraídos das execuções em uma das bases do MKP. Quanto ao resultado, este foi visivelmente favorável à versão com a topologia em forma de Anel, o que leva a crer que o caráter emergente foi um ponto relevante e positivo para o algoritmo PSOPR.

A fim de validar o modelo proposto, o algoritmo PSOPR foi submetido a uma série de experimentos de comparação com trabalhos publicados recentemente. A primeira

versão, aplicada ao TSP, foi comparada com outras três abordagens de PSO discreto, além de um algoritmo ACO. Os resultados foram encorajadores, uma vez que o algoritmo foi capaz de atingir, ou melhor média que outras abordagens, ou o valor ótimo em todas as bases dos três experimentos.

Os bons resultados obtidos com a primeira versão do algoritmo, que serviram de base para o trabalho de Rosendo e Pozo (2010), motivaram a aplicação do modelo em outro COP. Assim, na segunda versão, aplicada ao MKP, outros três trabalhos relacionados foram utilizados para comparação dividida em dois experimentos distintos. No primeiro, o PSOPR conseguiu gerar soluções melhores que o outro algoritmo em todas as bases do experimento. No segundo, comparando com outras duas abordagens para o mesmo problema, foram utilizadas dez bases diferentes, nas quais o PSOPR gerou resultados melhores para nove bases.

Todo problema de otimização, apesar de possuir vários pontos em comum entre eles, possui características únicas. Tais peculiaridades podem ser de primordial importância para o algoritmo ao abordar o problema. Propondo um modelo completamente geral para problemas combinatórios, as características únicas de cada problema são inevitavelmente desconsideradas, o que pode significar uma desvantagem para o algoritmo. Porém um meta-modelo que utiliza técnicas dependentes do problema internamente permite que parte do algoritmo seja implementada explorando as peculiaridades de cada COP, reforçando o elo entre o algoritmo e o problema. Entretanto, ao aplicar técnicas menos gerais e mais específicas ao problema, a implementação do algoritmo a cada novo problema se torna mais complexa.

Desta forma neste trabalho, ao definir o modelo PSOPR procurou-se obter o melhor custo-benefício possível entre manter o algoritmo geral e aplicável a COPs e, ao mesmo tempo, procurar abordar as características únicas de cada problema em busca de melhores resultados.

Várias funcionalidades e experimentos não puderam ser contemplados neste trabalho. Um destes possíveis trabalhos futuros diz respeito às topologias de vizinhança.

Um novo experimento comparando diferentes topologias poderia ser feito também para a versão do algoritmo aplicada ao TSP. Visto que, a comparação entre duas topologias para o PSOPR-MKP gerou resultados bastante significativos.

Quanto à sensibilidade aos coeficientes, outros experimentos poderiam ser feitos variando o valor dos fatores de individualidade e sociabilidade entre si. Além de um ajuste fino do algoritmo com experimentos que poderiam tomar como base as configurações que se mostraram melhores neste trabalho.

Ainda como trabalho futuro, o PSOPR poderia ser adaptado a problemas com mais de uma função objetivo a ser otimizada como o próprio problema da mochila em versão multiobjetivo - *Multiobjective Knapsack Problem*.

Outro experimento interessante poderia ser desempenhado em relação ao método de escolha de um item da mochila a ser removido ou adicionado. No PSOPR-MKP (capítulo 7) foi implementado um método próprio de seleção descrito em 7.2. Métodos semelhantes são utilizados em algoritmos genéticos para a tarefa de seleção dos indivíduos para reprodução. Uma destas técnicas, como, por exemplo, o método roleta, poderia ser implementada para comparação com o método atual.

Referências Bibliográficas

AL-KAZEMI, B.; MOHAN, C. K. Multi-phase discrete particle swarm optimization. In: **in FEA 2000: Fourth International Workshop on Frontiers in Evolutionary Algorithms**. Atlantic City, NJ, USA, 2000.

APPLEGATE, D.; BIXBY, R.; CHVTAL, V.; COOK, W. Finding tours in the tsp. In: **Institute for Discrete Mathematics, Universitat Bonn**. Bonn, Germany, 1999.

BEAUSOLEIL, R. P.; BALDOQUIN, G.; MONTEJO, R. A. Multi-start and path relinking methods to deal with multiobjective knapsack problems. **Annals of Operations Research**, v. 157, p. 105–133, 2007.

BENI, G.; WANG, J. Swarm intelligence. In: **Proc. 7th Ann. Meeting of the Robotics Society of Japan**. RSJ Press, 1989. p. 425–428.

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. ISSN 0360-0300.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. **Swarm Intelligence: From Natural to Artificial Systems**. Oxford University Press, New York, NJ, 1999.

CARVALHO, A. B. de. **Otimização por nuvem de partículas multiobjetivo no aprendizado indutivo de regras: extensões e aplicacações**. Dissertação (Mestrado) — Universidade Federal do Paraná, 2008.

CHEN, A. ling; YANG, G. ke; WU, Z. ming. Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. **Journal of Zhejiang University - Science A**, 2006.

CHEN, W.-N. et al. A novel set-based particle swarm optimization method for discrete optimization problems. **Evolutionary Computation, IEEE Transactions on**, v. 14, n. 2, p. 278–300, april 2009. ISSN 1089-778X.

CHRISTOFIDES, N.; MINGOZZI, M.; P.TOTH; SANDI, C. The travelling salesman problem. In: **Combinatorial Optimization**. Wiley, Chichester, 1979. p. 131–149.

CHU, P. C.; BEASLEY, J. E. A genetic algorithm for the multidimensional knapsack problem. **Journal of Heuristics**, v. 4, n. 1, p. 63–86, June 1998. Disponível em: <http://dx.doi.org/10.1023/A:1009642405419>.

CLERC, M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: **New Optimization Techniques in Engineering, Springer**. Heidelberg, Germany: Springer Berlin / Heidelberg, 2004. p. 219–239.

DARWIN, C. **The origin of species by means of natural selection :**. New York :Hurst, 1872. 532 p.

DORIGO, M. **Optimization, Learning, and Natural Algorithms**. Tese (Doutorado) — Politecnico di Milano, 1992.

DORIGO, M.; CARO, G. D. The ant colony optimization meta-heuristic. McGraw-Hill Ltd., UK, Maidenhead, UK, England, p. 11–32, 1999.

DORIGO, M.; CARO, G. D.; GAMBARDELLA, L. M. Ant algorithms for discrete optimization. **Artificial Life**, v. 5, p. 137–172, 1999.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. **Evolutionary Computation, IEEE Transactions on**, v. 1, n. 1, p. 53–66, 1997. Disponível em: <http://dx.doi.org/10.1109/4235-.585892>.

EBERHART, R. C.; SHI, Y. Comparing inertia weights and constriction factors in particle swarm optimization. **Evolutionary Computation, 2000. Proceedings of the 2000 Congress on**, v. 1, p. 84–88 vol.1, 2000. Disponível em: <http://dx.doi.org/10.1109-/CEC.2000.870279>.

GLOVER, F.; LAGUNA, M. **Tabu search**. Boston, MA: Kluwer Academic Publishers. xix, 1997. 382 p.

GLOVER, F.; LAGUNA, M. Fundamentals of scatter search and path relinking. **Control and Cybernetics**, v. 29, n. 3, p. 653–684, 2000.

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

HELSGAUN, K. An effective implementation of the lin-kernighan traveling salesman heuristic. **European Journal of Operational Research**, v. 126, p. 106–130, 2000.

HEMBECKER, F.; LOPES, H. S.; GODOY, W. Particle swarm optimization for the multidimensional knapsack problem. In: **Adaptive and Natural Computing Algorithms**. Springer Berlin / Heidelberg, 2007. p. 358–365.

HOLLAND, J. H. Adaptation in natural and artificial systems. In: **MIT Press, Cambridge, second edition**. 1975.

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**. 2003. p. 243–246.

JOHNSON, D. S.; MCGEOCH, L. A. Experimental analysis of heuristics for the stsp. In: **Local Search in Combinatorial Optimization**. Wiley & Sons, 2001.

JOHNSON, S. **Emergence: The Connected Lives of Ants, Brains, Cities, and Software**. Scribner, 2001. 288 p.

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

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

KENNEDY, J.; EBERHART, R. C. **Swarm Intelligence**. Academic Press, 2001. 512 p.

KHURI, S.; BÄCK, T.; HEITKÖTTER, J. The zero/one multiple knapsack problem and genetic algorithms. In: **SAC '94: Proceedings of the 1994 ACM symposium on Applied computing**. New York, NY, USA: ACM, 1994. p. 188–193. ISBN 0-89791-647-6.

KNOWLES, J.; CORNE, D.; DEB, K. **Multiobjective Problem Solving from Nature: From Concepts to Applications**. Springer, 2008.

KONG, M.; TIAN, P. Apply the particle swarm optimization to the multidimensional knapsack problem. In: **Artificial Intelligence and Soft Computing – ICAISC 2006**. Springer Berlin / Heidelberg, 2006. p. 1140–1149.

LAGUNA, M.; MARTI, R. Grasp and path relinking for 2-layer straight line crossing minimization. **INFORMS J. on Computing**, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 11, n. 1, p. 44–52, 1999. ISSN 1526-5528.

LI, X.; TIAN, P.; HUA, J.; ZHONG, N. A hybrid discrete particle swarm optimization for the traveling salesman problem. In: **Simulated Evolution and Learning**. 2006. v. 4247, p. 181–188.

LIANG, J.; QIN, A.; SUGANTHAN, P.; BASKAR, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. **Evolutionary Computation, IEEE Transactions on**, v. 10, n. 3, p. 281–295, June 2006. ISSN 1089-778X.

LIAO, C.-J.; TSENG, C.-T.; LUARN, P. A discrete version of particle swarm optimization for flowshop scheduling problems. **Computers and Operations Research**, v. 34, n. 10, p. 3099 – 3111, 2007. ISSN 0305-0548.

LIM, A.; LIN, J.; RODRIGUES, B.; XIAO, F. Ant colony optimization with hill climbing for the bandwidth minimization problem. **Applied Soft Computing**, v. 6, n. 2, p. 180 – 188, 2006. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science-/article/B6W86-4FJGW2K-1/2/6384a41414164f676a3ff26120d6ed81>.

LIN, S.; KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem. **Operations Research**, v. 21, n. 2, p. 498–516, 1973.

MACHADO, T.; LOPES, H. A hybrid particle swarm optimization model for the traveling salesman problem. In: **Adaptive and Natural Computing Algorithms**. Springer Vienna, 2005. p. 255–258.

MARINAKIS, Y.; MARINAKI, M. A particle swarm optimization algorithm with path relinking for the location routing problem. **Journal of Mathematical Modelling and Algorithms**, 2008.

MICHALEWICZ, Z.; FOGEL, D. B. **How to Solve It: Modern Heuristics**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004. ISBN 3540224947.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982. ISBN 0-13-152462-3.

RESENDE, M. G. C.; RIBEIRO, C. C. Grasp with path-relinking: Recent advances and applications. In: **Metaheuristics: Progress as Real Problem Solvers**. Springer, 2005. p. 29–63.

REYES-SIERRA, M.; COELLO, C. A. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. **International Journal of Computational Intelligence Research**, 2006.

RONALD, S. More distance functions for order-based encodings. In: **Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on**. 1998. p. 558–563.

ROSENDO, M.; POZO, A. A hybrid particle swarm optimization algorithm for combinatorial optimization problems. In: **IEEE Congress on Evolutionary Computation (CEC 2010)**. Barcelona, Spain, 2010.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Prentice Hall, 2002. ISBN 0137903952.

SHI, X. H.; LIANG, Y. C.; LEE, H. P.; LU, C.; WANG, Q. X. Particle swarm optimization-based algorithms for tsp and generalized tsp. **Inf. Process. Lett.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, v. 103, n. 5, p. 169–176, 2007. ISSN 0020-0190.

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**. Anchorage, Alaska, USA, 1998. p. 69–73.

SOUZA, G. R. **Uma abordagem por nuvem de partículas para problemas de otimização combinatória**. 75 p. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2006.

STÜTZLE, T. **Local Search Algorithms for Combinatorial Problems - Analysis, Improvements, and New Applications**. Tese (Doutorado) — Department of Computer Science, Darmstadt University of Technology, 1998.

STÜTZLE, T.; HOOS, H. Max-min ant system and local search for the traveling salesman problem. In: **IEEE Conference on Evolutionary Computation (ICEC'97)**. 1997.

TANOMARU, J. Motivação, fundamentos e aplicações de algoritmos genéticos. **Anais do II Congresso Brasileiro de Redes Neurais**, Curitiba, Brasil, 1995.

TORÁCIO, A. **Aprendizado de regras de classificação com otimização por nuvem de partículas multiobjetivo**. Dissertação (Mestrado) — Universidade Federal do Paraná, 2008.

VESTERSTRØM, J. S.; RIGET, J. **Particle Swarms: Extensions for improved local, multi-modal, dynamic search in numerical optimization**. 203 p. Dissertação (Mestrado) — Faculty of Science, Aarhus Universitet, 2002.

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

WANG, K.-P.; HUANG, L.; ZHOU, C.-G.; PANG, W. Particle swarm optimization for traveling salesman problem. In: **Proceedings of the 2003 International Conference on Machine Learning and Cybernetics**. 2003. p. 1583–1585.

WEST, D. B. **Introduction to Graph Theory (2nd Edition)**. Prentice Hall, 2000. Hardcover.

WHITE, T.; PAGUREK, B. Towards multi-swarm problem solving in networks. In: **In Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98**. IEEE Computer Society, 1998. p. 333–340.

WILCOXON, F. Individual comparisons by ranking methods. **Biometrics Bulletin**, International Biometric Society, v. 1, n. 6, p. 80–83, 1945. ISSN 00994987. Disponível em: <http://dx.doi.org/10.2307/3001968>.

YIN, P.-Y.; YU, S.-S.; WANG, P.-P.; WANG, Y.-T. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. **Computer Standards & Interfaces**, v. 28, n. 4, p. 441 – 450, 2006. ISSN 0920-5489.

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