Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Engenharia de Produção

Uma Ferramenta para Ensino da Metaheurística *Particle Swarm Optimization*

Carlos Ubialli Neto

TCC-EP-14-2013

Maringá - Paraná
Brasil

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Engenharia de Produção

Uma Ferramenta para Ensino da Metaheurística *Particle Swarm Optimization*

Carlos Ubialli Neto

TCC-EP-14-2013

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Produção, do Centro de Tecnologia, da Universidade Estadual de Maringá.
Orientador(a): Prof(ª). MSC. Gislaine Camila Lapasini Leal

Maringá - Paraná
2013

DEDICATÓRIA

Dedico à minha família que sempre me deu o suporte necessário, aos professores que me instruíram no decorrer destes anos e à minha namorada que me ajudou, apoiou e incentivou em todos os momentos.

EPÍGRAFE

[...] O mundo vai girando cada vez mais veloz, a gente espera do mundo e o mundo espera de nós, um pouco mais de paciência [...]

(Lenine)

AGRADECIMENTOS

A Deus, que me criou, abençoa e ilumina os meus passos.

Aos meus pais, pela educação, carinho, amor, compreensão, apoio e por fazerem de mim o que sou.

A minha irmã, quem eu amo incondicionalmente.

Aos meus amigos que sempre me divertiram, ajudaram e distraíram.

A minha orientadora, por seus conselhos, sugestões, correções e orientações.

E a minha namorada, que vem trazendo felicidade, amor, companheirismo, tranquilidade, suporte, cumplicidade e amizade a minha vida dia após dia.

RESUMO

O presente trabalho apresenta as características da metaheurística *Particle Swarm Optimization* (PSO) e sua aplicação em problemas clássicos da área de Engenharia de Produção, como o Problema do Caixeiro Viajante (PCV) e problemas de Programação Linear (PL). Foi desenvolvido um software chamado *Sagacious Bird* (SB), uma ferramenta didática que resolve problemas de PCV e PL por meio do PSO. Esta ferramenta é de fácil utilização e possui uma interface gráfica autoexplicativa, é flexível e parametrizável, e tem por objetivo o auxílio no ensino da metaheurística PSO. Por fim, o SB foi testado e comparado com o Algoritmo Genético (AG), comprovando ser eficiente na resolução dos problemas para que foi implementado.

Palavras-chave: Metaheurística. *Particle Swarm Optimization*. Problema do Caixeiro Viajante.

ABSTRACT

This paper presents the characteristics of metaheuristic Particle Swarm Optimization (PSO) and its application to classical problems of Production Engineering's area, as the Travelling Salesman Problem (TSP) and Linear Programming problems (LP). A software called Sagacious Bird (SB) was developed, a didactic tool that solves TSP's problems and LP's problems by PSO. This tool is easy to use and has a graphical interface self-explicative, it is flexible and configurable, and it has by purpose, assistance in the teaching of metaheuristic PSO. Lastly, the SB was tested and compared with the Genetic Algorithm (GA), proving to be effective in solving the problems for which it was implemented.

Key-words: Metaheuristic. Particle Swarm Optimization. Travelling Salesman Problem.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
GLS	*Guided Local Search*
GRASP	*Greedy Randomized Adaptive Search Procedure*
PCV	Problema do Caixeiro Viajante
PL	Programação Linear
PSO	*Particle Swarm Optimization*
AS	*Simulated Annealing*
SB	*Sagacious Bird*

1. INTRODUÇÃO

Acompanhando a evolução da humanidade nos mais diversos aspectos, os problemas encontrados por praticamente todos os profissionais também evoluíram, tendo sua complexidade aumentada. Desta maneira tornou-se necessário o desenvolvimento de técnicas mais avançadas para a resolução destes problemas.

As técnicas de otimização evoluíram, deixando de trabalhar somente com base em cálculos matemáticos e partindo para as buscas aleatórias direcionadas, como os métodos heurísticos e metaheurísticos.

A engenharia, mais especificamente a de produção, é uma área que possui diversos problemas complexos que necessitam ser resolvidos diariamente, e os profissionais da área necessitam de ferramentas para apoiar a tomada de decisão. Muitos destes problemas podem ser classificados quanto a sua complexidade como NP-Completos ou NP-Difíceis, não existindo algoritmo de tempo polinomial capaz de resolvê-los. Nestes casos utilizam-se as técnicas heurísticas, e metaheurísticas, a fim de encontrar uma solução muito próxima da melhor solução com recursos e tempo viáveis.

Diversos métodos heurísticos e metaheurísticos vêm sendo utilizados, dentre eles: Algoritmos Genéticos (AG), *Simulated Annealing* (AS), *Particle Swarm Optimization* (PSO), Busca Tabu, Algoritmos Meméticos, Otimização por Colônia de Formigas e Redes Neurais.

O foco deste trabalho é a aplicação do método *Particle Swarm Optimization*, uma metaheurística baseada no comportamento de aprendizado compartilhado entre os membros de um bando de pássaros, na resolução de problemas didáticos da área de Engenharia de Produção. O software desenvolvido neste trabalho tem o objetivo de auxiliar didaticamente no processo de ensino da disciplina de Metaheurísticas, especificamente em relação ao funcionamento e aplicação do PSO.

1.1 Justificativa

A área da Engenharia de Produção está repleta de problemas nos quais é necessário otimizar o uso dos recursos disponíveis (mão de obra, recursos financeiros, materiais, tecnologia, informação, energia e tempo), geralmente escassos, tais como: programação da produção, dimensionamento e sequenciamento de lotes de produção, custo de transporte e distribuição-armazenagem, parâmetros de corte, mistura de ingredientes, uso de recursos ambientais, planejamento mestre da produção, controle da qualidade de produtos e processos, entre outros. Grande parte desses problemas possui alta complexidade, sendo que, em diversas ocasiões, encontrar a melhor solução pelos métodos tradicionais é inviável. Em contrapartida, existem métodos não-exatos que apresentam uma solução muito próxima da ótima em tempo e custo viáveis.

Como o método *Particle Swarm Optimization* tem demonstrado bons resultados na otimização de problemas de alta complexidade, servindo como apoio à tomada de decisão aos profissionais da área, foi realizado um estudo do mesmo e suas aplicações em problemas de otimização da área de Engenharia de Produção.

A construção deste trabalho se justifica no caráter didático voltado ao ensino de metaheurísticas, mais especificamente o PSO, uma vez que essa área possui uma grande deficiência quanto ao referencial teórico e ferramentas de apoio.

1.2 Definição e delimitação do problema

Os profissionais da área de Engenharia de Produção se deparam repetidamente com diversos problemas categorizados como NP-Difícil ou NP-Completo.

Estes problemas são de solução inviável pelos métodos que utilizam Algoritmos Exatos na resolução, como os vistos na Programação Matemática. Nessas situações pode-se utilizar metaheurísticas para encontrar a solução, ou seja, chegar a uma solução ótima para o problema, em tempo e custo viáveis.

Assim, a apresentação da metaheurística *Particle Swarm*, sua implementação, sua aplicação na otimização de alguns problemas da área de Engenharia de Produção, verificação da

efetividade do método analisando seus resultados, e o auxílio no ensino de metaheurísticas, como ferramenta, é foco deste trabalho.

1.3 Objetivos

1.3.1 Objetivo geral

Elaborar uma ferramenta didática que implementa a metaheurística *Particle Swarm Optimization*.

1.3.2 Objetivos específicos

Como objetivos específicos, têm-se:

- Estudar o método *Particle Swarm*, suas características e aplicações;
- Selecionar problemas da área de Engenharia de Produção que possam ser resolvidos por meio do método;
- Implementar o método utilizando a linguagem pascal;
- Analisar os resultados obtidos, a fim de mostrar a viabilidade do método na otimização dos problemas da área de Engenharia de Produção.
- Apoiar o ensino da disciplina metaheurísticas.

1.4 Metodologia

Este trabalho caracteriza-se por uma pesquisa aplicada a respeito dos assuntos: otimização, metaheurísticas, o método *Particle Swarm Optimization* e suas áreas de aplicação.

Conforme pode ser visto em Gil (2002), a classificação da pesquisa com base nos procedimentos técnicos é uma pesquisa bibliográfica a partir de livros, artigos, teses, e monografias já publicadas com conteúdo referente ao deste trabalho.

Dos pontos de vista da forma da abordagem e dos objetivos, a pesquisa é qualitativa e exploratória, respectivamente.

Problemas da área de engenharia de produção que podem ser resolvidos pelo método foram elencados dentre os encontrados em pesquisa realizada em artigos.

Dentre os problemas elencados, alguns serão selecionados e resolvidos pelo método. A implementação será realizada em ambiente Delphi.

Por fim, os resultados foram analisados e comparados com os de outros métodos encontrados na literatura, a fim de mostrar o desempenho e a viabilidade do método *Particle Swarm Optimization.*

1.5 Estrutura do trabalho

O primeiro capítulo apresenta ao leitor a introdução, justificativa, definição e delimitação do problema, objetivos e metodologia do presente trabalho.

O segundo capítulo contém a revisão da literatura, expondo os conceitos de otimização, heurística, metaheurística, *Particle Swarm Optimization*, e o problema do caixeiro viajante.

O terceiro capítulo descreve o desenvolvimento do trabalho, do software e a comparação de seus resultados com os encontrados na literatura.

Por fim, as considerações finais, contribuições, dificuldades e limitações, e trabalhos futuros são apresentados no quarto capítulo.

2. REVISÃO DE LITERATURA

2.1 Otimização

Otimizar é encontrar os mínimos ou máximos de uma função de várias variáveis, cujos valores estão restritos em uma região do espaço multidimensional, como apresentam Martínez e Santos (1995).

Otimização é o processo de buscar a melhor solução dentre as várias, ou infinitas, existentes em um problema. Segundo Leal (2007), otimização também é a técnica utilizada para maximizar ou minimizar uma função, levando em conta suas restrições e variáveis.

Conforme Prado e Saramago (2005), as técnicas clássicas de otimização são conhecidas há bem mais de um século, sendo utilizadas na física e geometria. Nos últimos anos, com o aumento da complexidade dos problemas e o grande avanço dos recursos computacionais, as técnicas de otimização evoluíram muito.

Leal (2007) expõe que as técnicas de otimização podem ser agrupadas em: técnicas baseadas em cálculos matemáticos, técnicas baseadas em programação matemática, e técnicas baseadas em busca aleatória direcionada.

As duas primeiras são muito conhecidas, confiáveis e muito aplicadas em diversos campos da física, engenharia e outras ciências. Porém apresentam dificuldades na resolução de problemas que apresentam descontinuidade de funções, funções não-convexas, multimodalidade, ruídos nas funções, e existência de mínimos ou máximos locais, como descreve Leal (2007). Nestes casos, as técnicas baseadas em busca aleatória direcionada possuem um desempenho muito bom.

Para aplicar as técnicas de otimização é necessária a realização da modelagem do problema real. Esta modelagem consiste na formalização das variáveis, restrições e objetivos que melhor definem e representam o problema.

O ponto de máximo ou de mínimo de uma função, ou seja, a melhor solução, é chamado de solução ótima. Um algoritmo que obtenha essa solução ótima é chamado de algoritmo exato.

Há diversos problemas que são muito grandes e/ou muito complexos de forma que os algoritmos exatos não são capazes de resolvê-los, ou o tempo necessário para a resolução é inviável. Quando é necessário obter soluções satisfatórias, com eficiência, rapidez e agilidade os algoritmos heurísticos e metaheurísticos são muito utilizados.

2.2 Heurísticas

Métodos heurísticos são procedimentos que, na maioria das vezes encontrarão uma excelente solução viável para o problema a ser avaliado. Não é possível garantir a qualidade da solução obtida, porém, quando são bem elaborados, retornam soluções bem próximas à ótima conforme afirma Hillier e Lieberman (2006).

As heurísticas são eficientes para lidarem com problemas muito grandes e complexos. São normalmente algoritmos iterativos que buscam a cada iteração um novo resultado, e que retornam a melhor solução encontrada. Além disso, esses algoritmos utilizam conhecimento prévio sobre o problema em questão para direcionar os esforços no caminho mais provável em que os melhores resultados se encontram.

Conselheiro (1999) apresenta os seguintes fatores que justificam o uso das heurísticas, e as tornam interessantes:

- Inexistência de um método exato de resolução ou este requer muito esforço computacional.
- Limitação do tempo, obrigando o uso de métodos de resposta rápida à custa da precisão.
- Como passo intermediário de outro algoritmo para geração de uma solução inicial para métodos iterativos.
- Quando os dados são pouco confiáveis ou ainda o modelo não descreve bem a realidade, abre-se mão de soluções exatas já que estas são uma aproximação da realidade.

- Quando não se necessita da solução ótima, se os valores adquiridos pela função objetivo são relativamente pequenos pode não valer a pena o esforço por uma solução ótima. Pode ser suficiente a apresentação de uma solução melhor do que a atual.

As heurísticas podem ser divididas em construtivas, melhoramento e metaheurísticas, conforme afirma Cordenonsi (2008). Uma heurística construtiva somente considera o próximo passo, partindo de uma solução vazia, portanto seu critério de escolha é basicamente local. As heurísticas de melhoramento partem de uma solução que satisfaz todas as restrições do problema e a cada passo busca melhorar a solução atual.

Estes dois tipos de heurísticas são muito específicos ao problema, obrigando que a cada novo problema um algoritmo que se adeque deva ser desenvolvido. Outro fator negativo encontrado nestes tipos de heurísticas é que em diversas ocasiões o algoritmo fica preso em uma solução que é um ótimo local, não conseguindo chegar mais próximo do ótimo global, como apresenta Trigueros (2011).

O terceiro tipo, metaheurística, é descrito detalhadamente a seguir.

2.3 Metaheurísticas

“Uma metaheurística é um método de resolução geral que fornece tanto uma estrutura quanto diretrizes de estratégia gerais para desenvolver um método heurístico específico que se ajuste a um tipo de problema particular.” (HILLIER e LIEBERMAN, 2006). Diferente dos outros tipos de heurísticas, elas são desenvolvidas para um tipo de problema, precisando de alterações bem simples para se adaptar aos diferentes problemas daquele tipo.

As metaheurísticas possuem a capacidade de explorar o espaço de busca de forma mais ampla, podendo chegar a uma solução pior que a última encontrada. Assim não ficam aprisionadas em ótimos locais, possuindo uma chance maior de alcançar o ótimo global, ou soluções mais próximas deste valor.

Outra característica interessante é a velocidade em que o algoritmo se move na direção de soluções muito boas, sendo eficaz na resolução de problemas grandes e complicados. Porém,

não há a garantia de que a melhor solução encontrada será a solução ótima ou nem mesmo uma solução próxima da ótima, como expõem Hillier e Lieberman (2006).

São encontradas na literatura diferentes classificações das metaheurísticas, sendo uma delas a de Melián et al (2003):

- Metaheurísticas de Métodos de Relaxação: Em problemas cujo cálculo da função objetivo consome grande tempo de execução, torna-se importante uma modificação da função objetivo, de forma a diminuir o tempo consumido por este cálculo. Este tipo de metaheurística é um método que consiste em modificar a modelagem do problema original, geralmente alterando a função objetivo e o conjunto de restrições, resolver este problema "relaxado" e usar a solução encontrada como guia para encontrar a solução do problema original. Exemplo: Relaxação Lagrangeana.
- Metaheurísticas para Processos Construtivos: Metaheurísticas Construtivas consistem em estabelecer uma estratégia para a construção de soluções viáveis de forma gradativa partindo de uma solução inicial vazia. A solução parcial deve ser incrementada gradativamente, de forma a não se tornar inviável, até que o critério de parada seja atingido. Exemplo: GRASP (*Greedy Randomized Adaptive Search Procedure*).
- Metaheurísticas de Busca por Entornos: As metaheurísticas de busca por entornos podem ser definidas como algoritmos que percorrem espaços de buscas formados por soluções, sendo que a cada iteração é considerada a vizinhança obtida na iteração anterior. Estes algoritmos necessitam de uma solução prévia, comumente chamada de solução inicial, que pode ser gerada utilizando-se estratégias de aleatoriedade, trivialidade ou por meio de métodos construtivos. Exemplos: GLS (*Guided Local Search*), *Simulated Annealing*, Busca Tabu e Busca Reativa.
- Metaheurísticas para Métodos Evolutivos: As metaheurísticas evolutivas podem ser definidas como algoritmos baseados na teoria evolutiva Darwiniana, que busca, a cada geração, promover a evolução da população por meio da reprodução de seus indivíduos, a fim de encontrar a solução ótima para o problema. Exemplos: Algoritmos genéticos, Algoritmos meméticos, Estimação de distribuição, Busca dispersa e *Path relinking*.

- Metaheurísticas Híbridas: Algumas metaheurísticas apresentam uma abordagem completamente diferente, não pertencendo a nenhuma das classes vistas anteriormente, ou utilizam mais de uma das descritas acima, sendo assim denominadas metaheurísticas híbridas. Exemplos: Metaheurísticas de decomposição, *Ant colony optimization*, Otimização extrema, *Iterated local search* e *Particle swarm optimization*.

Segundo Zayatz (2012), as técnicas metaheurísticas Algoritmos Genéticos, *Simulated Annealing* e *Particle Swarm* têm sido bastante utilizadas nas áreas da Engenharia de Produção.

2.4 *Particle Swarm Optimization*

Particle Swarm Optimization (PSO), Otimização por Enxame de Partículas em português, é uma metaheurística desenvolvida por James Kennedy e Russell Eberhart em 1995. Este método de otimização de funções contínuas foi motivado pela simulação do comportamento social de interação de indivíduos de um grupo (SERAPIÃO, 2009).

Vários cientistas, incluindo zoólogos, se interessaram pelas regras que gerenciam o comportamento dos indivíduos de um grupo, como a sincronia do voo dos pássaros, Figura 1, e nado de cardume de peixes, suas mudanças bruscas de direção, dispersão e reagrupamento (LUZ, 2008).

Figura 1 - Sincronia do voo de um bando de pássaros

(LUZ, 2008)

O método foi proposto com base no comportamento de pássaros quando estão em bando. Os pássaros compartilham informações referentes à comida, predadores, busca de parceiros, melhoria de parâmetros ambientais como a temperatura, entre outras. Percebeu-se que o grupo era influenciado por três fatores: a experiência individual acumulada, a experiência acumulada e compartilhada pelo grupo, e fatores aleatórios (KENNEDY e EBERHART, 1995).

O PSO possui um conceito muito simples, pode ser implementado em poucas linhas de código, requer somente operações matemáticas primitivas, é computacionalmente barato em termos de uso de memória e velocidade, e independe de conhecimento prévio sobre o problema.

Analogamente aos pássaros, as partículas “voam” pelo espaço multidimensional buscando possíveis soluções para o problema. Cada partícula se baseia em seu próprio conhecimento e no conhecimento compartilhado pelo grupo para guiar sua busca pela melhor solução. Para conseguir fugir dos ótimos locais há um terceiro fator que direciona as partículas: aleatoriedade.

A cada iteração do algoritmo as partículas se movem no espaço multidimensional de busca atualizando a sua posição por um parâmetro chamado de velocidade, de forma que, a posição atual da partícula somada à velocidade calculada gera sua nova posição.

A cada posição que a partícula “ocupa” é verificado se a posição é uma solução válida para o problema. Caso seja, compara-se esta solução com a melhor solução da partícula e a melhor solução global. Se for uma solução melhor, tanto a posição quanto o valor da solução são armazenados para usos futuros. Desta forma, cada partícula vai se movimentando e atualizando suas informações, compartilhando seus resultados com as outras partículas da nuvem, até que a condição de parada seja atendida, como se pode visualizar na Figura 2.

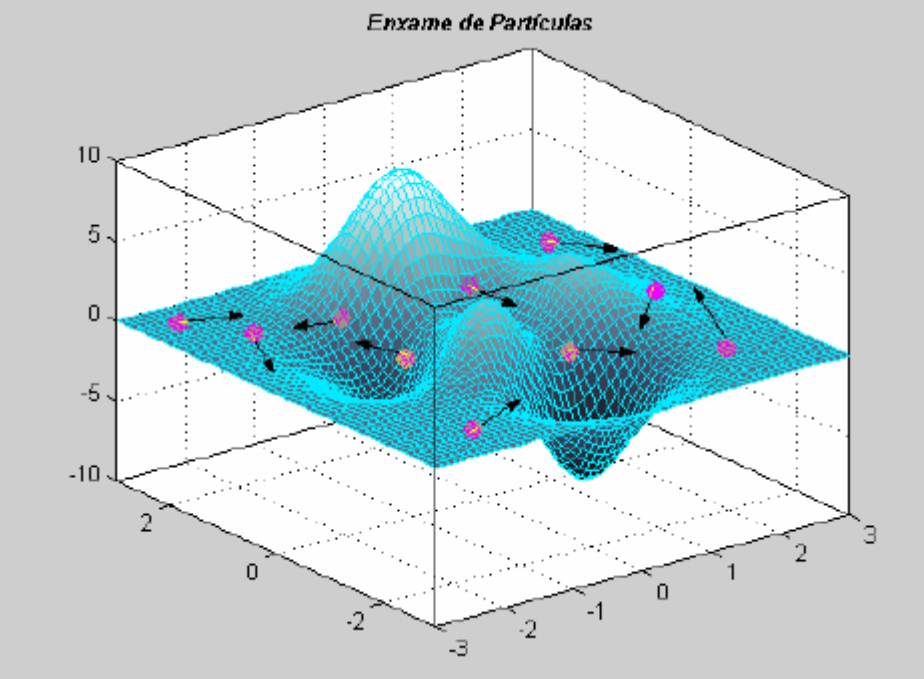


Figura 2 - Enxame de partículas

(BRANDINI, 2007)

Mais detalhadamente, têm-se:

- Partícula: cada indivíduo que se deslocará no espaço n-dimensional buscando a melhor solução para o problema em questão. O número de dimensões será definido pela quantidade de variáveis a serem encontradas para solucionar o problema;
- Posição: local no espaço ocupado pela partícula. Caso seja necessária a determinação de dois valores para a solução do problema, o valor da posição da partícula compreenderá um valor para a dimensão X e outro valor para a dimensão Y. Cada posição da partícula é uma possível solução para o problema;
- Função objetivo: função que caracteriza o problema e servirá de teste na determinação da solução do mesmo;
- *Fitness*: valor encontrado ao utilizar a posição da partícula na função objetivo;
- *MelhorPosiçãoPartícula*: valor da posição que retornou o melhor *fitness* da partícula;
- *MelhorFitnessPartícula*: valor do melhor *fitness* da partícula;
- *MelhorPosiçãoGlobal*: valor da posição que retornou o melhor *fitness* dentre todas as partículas da nuvem;
- *MelhorFitnessGlobal*: valor do melhor *fitness* dentre todas as partículas da nuvem;
- Velocidade: é o quanto a partícula se deslocará no espaço. A velocidade é calculada levando em consideração a velocidade atual da partícula, a posição atual da partícula,

a *MelhorPosiçãoPartícula*, a *MelhorPosiçãoGlobal*, números aleatórios e três constantes de aprendizado, como pode ser visto na Equação (1):

$$NovaVelocidade = (\mathbf{w} * VelocidadeAtual) + (\mathbf{c1} * \mathbf{r1} * (MelhorPosiçãoPartícula - PosiçãoAtualPartícula)) + (\mathbf{c2} * \mathbf{r2} * (MelhorPosiçãoGlobal - PosiçãoAtualPartícula)), \quad (1)$$

na qual **r1** e **r2** são números aleatórios uniformemente distribuídos entre 0 e 1, **c1** é a constante de aceleração cognitiva, **c2** é a constante de aceleração social e **w** é o fator inercial. As constantes **c1** e **c2** determinam o quanto os valores individuais da partícula e os valores do enxame, respectivamente, contribuirão na nova velocidade. Por sua vez, **w** mantém o refinamento da busca mesmo após a região do ótimo ter sido encontrada. Valores elevados de **w** geram uma busca global, enquanto valores baixos provocam uma busca local, como apresenta Waintraub (2009).

A nova posição da partícula pode ser definida como sua posição atual somada à sua nova velocidade, como representado na Equação (2):

$$NovaPosição = PosiçãoAtual + NovaVelocidade. \quad (2)$$

Alguns parâmetros possuem uma grande relevância no desempenho do algoritmo, como descreve Brandini (2007):

- As constantes **c1** e **c2** são geralmente números entre 0 e 2, ajustados por tentativa e erro. Elas determinam se as partículas se moverão em direção ao melhor resultado da partícula, em direção ao melhor resultado global, ou um valor misto;
- A quantidade de partículas determina se o algoritmo conseguirá solucionar o problema em tempo hábil;
- O fator inercial **w** é responsável por fazer a partícula continuar se movendo na mesma direção que estava seguindo na iteração anterior. Valores altos de **w** tornam a busca mais global, enquanto valores baixos tornam a busca mais local. Resultados experimentais mostram que o melhor a se fazer é iniciar **w** com um valor alto, tendo uma exploração global do espaço de busca, e gradualmente diminui-lo a fim de refinar a solução. A atualização do fator inicial é dada pela Equação (3):

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} * iter, \quad (3)$$

sendo **iter$_{max}$** o número máximo de iterações e **iter** o numero da iteração corrente;

- Um parâmetro que pode ser implementado e que aprimora bastante o desempenho do método é a velocidade máxima da partícula. A velocidade é o fator determinante na evolução da partícula e uma atualização indesejada pode tornar a trajetória da partícula incontrolável. Limitar o valor da velocidade da partícula torna a trajetória bem mais próxima da região de convergência do problema. Caso o valor calculado para a velocidade da partícula seja maior que a velocidade máxima – ou menor que o valor oposto da velocidade máxima – a velocidade assumirá o valor da velocidade máxima, como é possível ver na Equação (4):

$$\begin{aligned} &se\ NovaVelocidade > Vmax \text{ então } NovaVelocidade = \mathrm{V}max \\ &senão\ se\ NovaVelocidade < -Vmax \text{ então } NovaVelocidade = -\mathrm{V}max \end{aligned} \quad (4)$$

Basicamente, o algoritmo PSO pode ser definido pelos seguintes passos:

1) Definir o número de partículas e os valores para as constantes **c1**, **c2** e **w**;
2) Inicializar de forma aleatória os valores da posição e velocidade de cada partícula;
3) Calcular o valor do *fitness* de cada partícula atualizando os valores *MelhorPosiçãoPartícula, MelhorFitnessPartícula, MelhorPosiçãoGlobal e MelhorFitnessGlobal;*;
4) Executar um *loop* contendo os próximos passos, até a condição de parada ser atingida, para cada partícula;
5) Calcular a *NovaVelocidade* e a *NovaPosição;*
6) Calcular o valor do *fitness*;
7) Atualizar os valores *MelhorPosiçãoPartícula, MelhorFitnessPartícula, MelhorPosiçãoGlobal e MelhorFitnessGlobal.*

O Quadro 1 mostra o pseudocódigo do PSO.

```
Atribuir valores aos parâmetros numPartículas, c1, c2 e w;

para i = 0 até numPartículas faça
    Inicializa a posição de forma aleatória;
    Inicializa a velocidade de forma aleatória;
    Calcular fitness;
    Atualizar valores MelhorPosiçãoPartícula, MelhorFitnessPartícula,
                      MelhorPosiçãoGlobal e MelhorFitnessGlobal;
fim;

enquanto não atingir condição de parada faça
    para i = 0 até numPartículas faça
        Calcular NovaVelocidade;
        Calcular NovaPosição;
        Calcular fitness;
        Atualizar valores MelhorPosiçãoPartícula, MelhorFitnessPartícula,
                          MelhorPosiçãoGlobal e MelhorFitnessGlobal;
    fim;
fim;
```

Quadro 1 - Pseudocódigo do PSO

A Figura 3 ilustra o fluxo de controle do PSO.

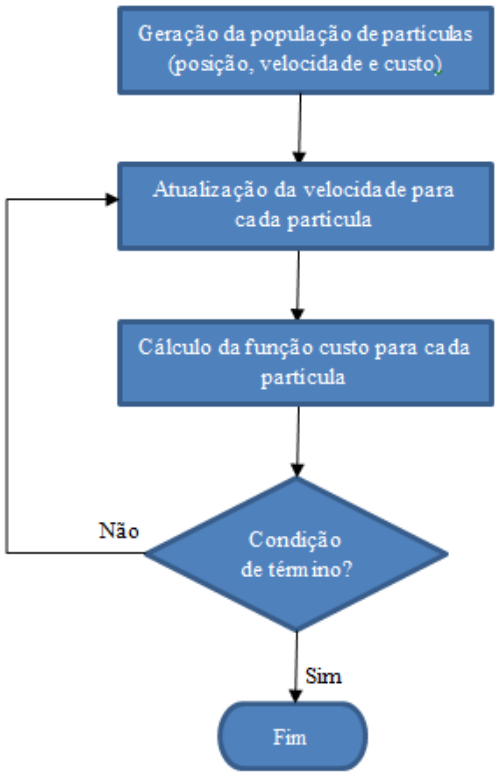


Figura 3 - Fluxo de controle do PSO

(LUZ, 2008)

O método PSO sofreu diversas alterações e adaptações. Uma destas alterações foi quanto ao modo como é realizada a ligação entre as partículas, ou seja, como foi implementada a influência entre as partículas.

Brandini (2007) apresenta algumas topologias de implementação, sendo elas:

- Estrela: Todas as partículas estão conectadas entre si, conforme Figura 4. Esta forma de topologia proporciona uma convergência mais rápida, porém, o risco de uma convergência para um ótimo local é a maior dentre as demais.

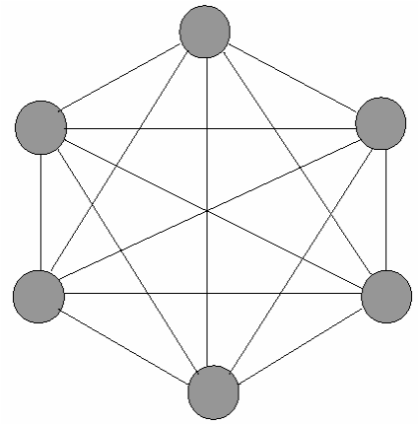


Figura 4 - Topologia em estrela

(BRANDINI, 2007)

- Roda: Uma partícula central é conectada a todas as outras, enquanto as outras não se comunicam entre si, como pode ser visto na Figura 5. Desta forma, a partícula central recebe os resultados e influencia as demais. A convergência é um pouco mais lenta que a da topologia estrela, porém, a possibilidade de convergência para um ótimo local é menor.

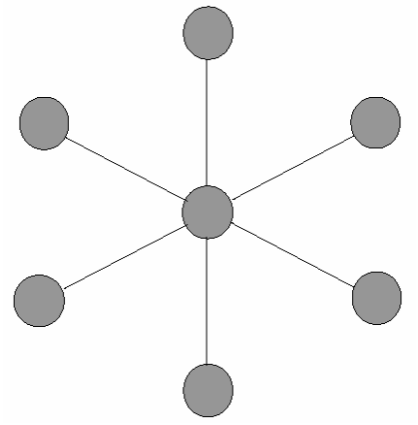


Figura 5 - Topologia em roda

(BRANDINI, 2007)

- Círculo: Cada partícula é conectada somente a duas outras, ilustrado na Figura 6. Desta maneira, a influência das partículas mais distantes é muito pequena. Por essa razão, esta topologia é a mais eficaz em não convergir para ótimos locais.

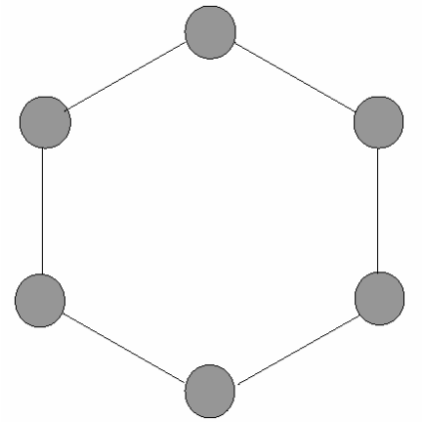


Figura 6 - Topologia em círculo

(BRANDINI, 2007)

- Randômica: As partículas são conectadas a outras, ou não, de forma aleatória. A Figura 7 é um exemplo desta topologia.

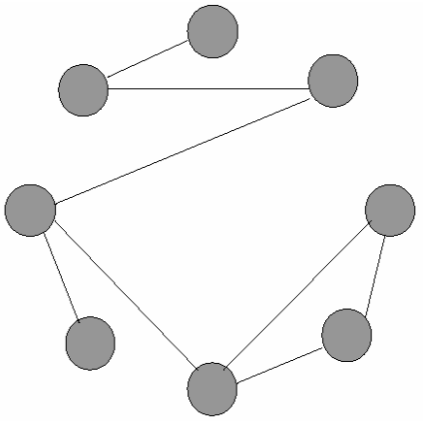


Figura 7 - Topologia randômica

(BRANDINI, 2007)

Zayatz (2012) apresenta que o PSO foi aplicado nas seguintes áreas da Engenharia de Produção: Engenharia de Operações e Processos de Produção, Logística, e Engenharia da Qualidade.

O PSO, desenvolvido por Kennedy e Eberhart, permite a resolução de problemas com variáveis contínuas com grande facilidade e tem sido aplicado com sucesso. Estas características levaram ao desenvolvimento de diversas adaptações desta metaheurística para

tratar dos problemas cujas variáveis são contínuas, e também de problemas com variáveis discretas.

A primeira versão do PSO para problemas discretos foi desenvolvida por Kennedy e Eberhart (1997). Este novo método mantém a essência do PSO original, porém o problema é tratado em um espaço binário, no qual a velocidade da partícula foi definida como uma matriz de probabilidades de mudança de 0 para 1 e vice-versa.

Diversas outras versões foram criadas, com novas abordagens, técnicas inovadoras, e em muitas delas foi realizada uma hibridização com outros métodos como *Path Relinking*, Algoritmos Genéticos, e busca local.

Como descreve Fuchs, Delgado e Lüders (2012), dentre as versões do PSO discreto encontradas na literatura, as que possuem maior destaque são as apresentadas por: Kennedy e Eberhart em 1997, Hu em 2003, Clerc em 2004, Goldbarg et al. em 2006 e Rosendo em 2010.

A versão do PSO discreto apresentada por Clerc mantém o princípio do PSO original, permanecendo inalteradas as equações de atualização da posição e da velocidade. Já os operadores utilizados no espaço de busca e nas soluções encontradas foram modificados, bem como a posição e a velocidade da partícula.

A posição da partícula é representada por um vetor de valores discretos e a velocidade passou a ser um vetor contendo uma lista de transposições a serem executadas no vetor posição das partículas, Maia (2009).

A velocidade e a posição da partícula continuam sendo calculadas por meio da Equação (1) e Equação (2), respectivamente, mas como os valores são discretos, algumas relações devem ser compreendidas:

- Cálculo da nova posição (P') por meio da "soma" da posição atual e a velocidade (P + v): "Somar" **P** e **v** é transpor os valores da posição **P** de acordo com os valores contidos em **v**. Sendo **P** = [2,3,4,5,1] e **v** = [(1,2), (2,4)], aplica-se a transposição (1,2) em **P** obtendo a posição $\mathbf{P_1}$ = [3,2,4,5,1], e depois aplica-se a troca (2,4) em $\mathbf{P_1}$ obtendo a posição **P'** = [3,5,4,2,1].

- Obtenção de velocidade (v) por meio da "subtração" de duas posições ($P_1 - P_2$): A "subtração" de duas posições ($\mathbf{P_1} - \mathbf{P_2}$) resulta em uma velocidade **v** de forma que $\mathbf{P_2} = \mathbf{P_1} + \mathbf{v}$. Sendo $\mathbf{P_1}$ = [**1**,2,3,4,5] e $\mathbf{P_2}$ = [2,3,**1**,5,4], nota-se que $\mathbf{P_1}[1] = \mathbf{P_2}[3] = 1$, portanto, a primeira troca será (1,3) e tem-se $\mathbf{P_3} = \mathbf{P_2} + (1,3)$ = [1,3,2,5,4]. Na sequência, $\mathbf{P_1}[2] = \mathbf{P_3}[3] = 2$, assim a segunda troca será (2,3) e $\mathbf{P_4}$ = [1,2,3,5,4]. Para concluir, percebe-se que $\mathbf{P_1}[4] = \mathbf{P_4}[5] = 4$, a terceira troca será (4,5) e $\mathbf{P_5} = \mathbf{P_1}$ = [1,2,3,4,5]. Portanto a velocidade será igual a **v** = [(1,3), (2,3), (4,5)].
- Obtenção de velocidade (v) por meio da soma entre duas velocidades ($v_1 + v_2$): A adição entre duas velocidades é a concatenação das duas velocidades. Se $\mathbf{v_1}$ = (1,2) e $\mathbf{v_2}$ = (3,5), a soma das duas velocidades será $\mathbf{v} = \mathbf{v_1} + \mathbf{v_2}$ = [(1,2), (3,5)].
- Obtenção de velocidade (v) por meio da multiplicação entre um coeficiente real e uma velocidade ($c * v_1$): A multiplicação entre um coeficiente real e uma velocidade apresenta três casos, dependendo do valor do coeficiente:
  i. Se **c** < 0, **c** * **v** não é definido;
  ii. Se **c** = 0, **c** * **v** = Ø, ou seja, a velocidade será nula, independentemente do valor da velocidade;
  iii. Se **c** > 0, trunca-se o valor de **v** com o tamanho calculado por **c** * |**v**|, na qual |**v**| é o tamanho do vetor velocidade. Se **c** = 0,6 e **v** = [(1,2), (4,5)], tem-se |**v**| = 2 e **c** * |**v**| = 1,2. Truncando este valor, obtém-se o valor 1 que corresponderá ao tamanho da nova velocidade. Deste modo **v'** = (1,2).

2.5 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um dos mais estudados na literatura, dentre os problemas de otimização combinatória. Dado um conjunto de N cidades e a matriz de distâncias entre elas, o problema consiste em encontrar o caminho com menor custo, partindo de uma cidade, passando somente uma vez em cada cidade, e retornando à cidade inicial. Esta trajetória em que as cidades somente são visitadas uma vez é conhecida como Ciclo Hamiltoniano (CORDENONSI, 2008).

É importante observar que o que foi chamado de "cidades" são os nós de um grafo e não precisam necessariamente ser cidades, podendo ser postos de trabalhos, distribuidores, clientes, fornecedores, ou o que for coerente ao caso analisado. Seguindo o mesmo raciocínio,

as distâncias podem ser realmente distâncias como podem ser custos, dificuldade de travessia, tempo, entre outros fatores.

Problemas matemáticos relacionados ao PCV foram estudados no século XIX pelo matemático irlandês Sir William Rowan Hamilton e pelo matemático britânico Thomas Penyngton Kirkman. Por volta de 1930 o matemático e economista Karl Menger estudou o método em Viena e em Harvard. Já em 1940 o PCV foi estudado pelos estatísticos Mahalanobis, Jessen, Gosh, e Marks relacionando-o com aplicações agrícolas. Os métodos de solução do PCV começaram a aparecer em artigos em meados de 1950 (MAREDIA, 2010).

A Tabela 1 apresenta a evolução das resoluções do PCV a partir de 1954 até 2004 quando Applegate, Bixby, Chvátal, Cook e Helsgaun solucionaram uma instância com 24.978 cidades e provaram não existir um percurso menor. Atualmente, esta é a maior instância já resolvida.

Tabela 1 - Evolução das resoluções do PCV

Ano	Quantidade de Cidades	Pesquisadores
1954	49	Dantzig, Fulkerson e Johnson
1962	64	Held e Karp
1974	67	Camerini, Fratta e Maffioli
1980	120	Grötschel
1980	318	Crowder e Padberg
1987	532	Padberg e Rinaldi
1991	666	Grötschel e Holland
1991	2392	Padberg e Rinaldi
1995	7.397	Applegate, Bixby, Chvátal e Cook
1998	13.509	Applegate, Bixby, Chvátal e Cook
2001	15.112	Applegate, Bixby, Chvátal e Cook
2004	24.978	Applegate, Bixby, Chvátal, Cook e Helsgaun

Fonte: Prestes, 2006, adaptado

A Figura 8 mostra a instância de 24.978 pontos sobre o mapa da Suécia e o percurso ótimo encontrado.

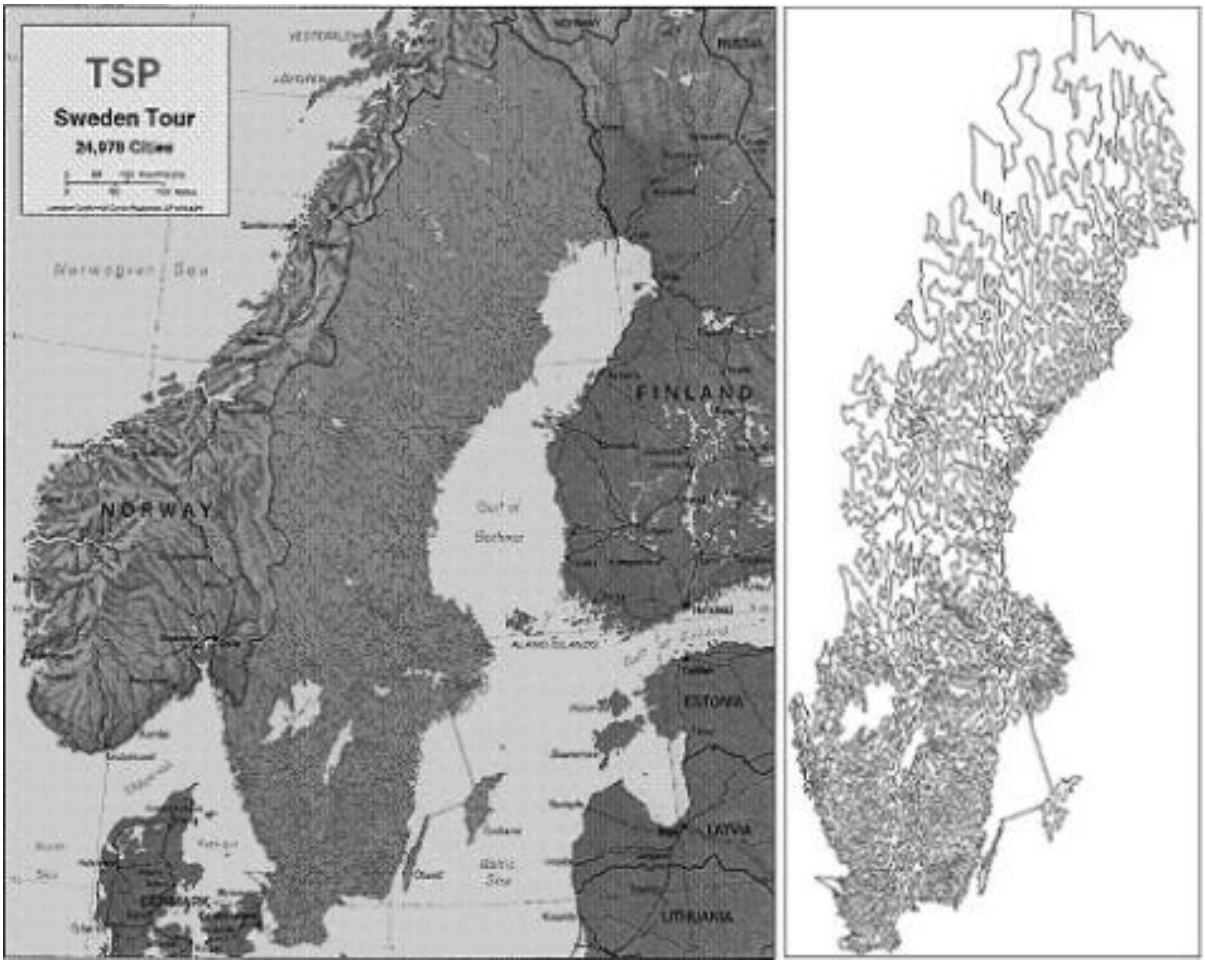


Figura 8 - Mapa da Suécia à esquerda e o melhor percurso à direita

(PRESTES, 2006)

A Figura 9 ilustra um exemplo do PCV com seis cidades, as distâncias entre elas, e uma possível solução indicada pelo traço de cor preta.

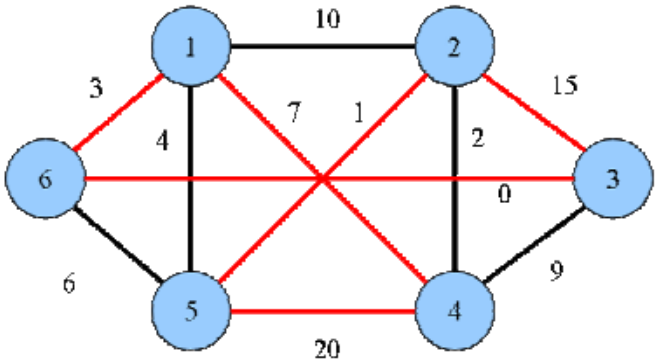


Figura 9 - Exemplo do PCV

(MAIA, 2009)

O PCV pode ser dividido em 2 grupos, os problemas simétricos e os assimétricos. Os PCVs simétricos são os que a distância para ir da cidade i para a cidade j é a mesma para ir da cidade j para a cidade i, ou seja, a distância de ida e de volta entre as cidades é igual. Já os PCVs assimétricos são os que a distância de ida e de volta entre as cidades é diferente.

O PCV é um problema pertencente à classe NP-Difícil, segundo Melo (2006). O número total de combinações de resolução cresce em proporção fatorial a cada cidade adicionada. Caso a cidade inicial seja fixada, o número total de possibilidades é dado pela Equação (5):

$$R(n) = (n - 1)! \tag{5}$$

Quando a primeira cidade não é fixada, podendo o caixeiro partir de qualquer cidade, o número de soluções é dado pela Equação (6):

$$R(n) = n! \tag{6}$$

Apesar de toda a evolução em termos de hardware e software, números altos de cidades geram quantidades muito elevadas de possibilidades, não sendo possível encontrar a melhor solução listando todas as opções (ZAYATZ, 2012). Por este motivo, os algoritmos heurísticos e metaheurísticos são muito utilizados para resolver o PCV.

Além da aplicação direta do PCV em situações de construção de rotas, há também aplicações em fabricação de placas de circuitos eletrônicos, roteamento de veículos, sequenciamento de tarefas em uma determinada máquina, programação de transporte entre células de manufatura, otimização do movimento de ferramentas de corte, trabalhos administrativos, entre outros (CORDENONSI, 2008).

3. SAGACIOUS BIRD (SB)

O *Sagacious Bird* (SB) foi desenvolvido na linguagem pascal e no ambiente de desenvolvimento Delphi. É um programa didático com o objetivo de auxiliar nas atividades que envolvem o ensino de metaheurísticas por meio da resolução do PCV, usando o PSO para variáveis discretas, e da resolução de problemas de Programação Linear (PL), usando o PSO para variáveis contínuas. A Figura 10 apresenta a tela de menu do SB.

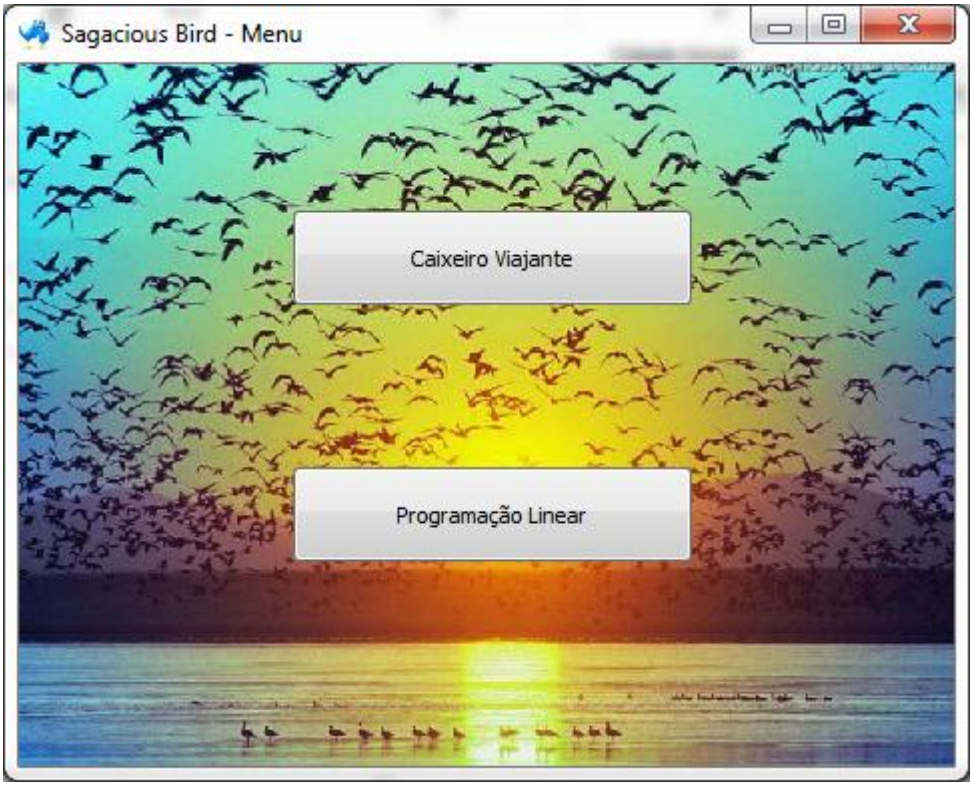


Figura 10 – *Sagacious Bird* - Menu

Conforme aumenta o número de cidades de um PCV ou aumenta o número de variáveis e restrições de um problema de PL, a complexidade do problema fica mais alta. Nestes casos, resolver pelos métodos clássicos não é viável. O SB proporciona um grande auxílio no aprendizado de metaheurísticas, pois é capaz de resolver diversos tamanhos de problemas, proporcionando ao usuário a condição de avaliar o funcionamento do método em problemas conhecidos pelo usuário.

Além de ser didático, o SB é um software projetado para ser de fácil utilização, possuindo termos autoexplicativos. É flexível, uma vez que resolve tanto PCV quanto problemas de PL.

Como é possível resolver problemas de tamanhos diferentes, pode-se definir o programa como escalável. É também parametrizável, visto que o usuário pode definir todos os parâmetros que influenciam na performance da resolução do problema.

Independentemente de qual problema se está resolvendo, foi definida uma classe contendo as informações da posição, *fitness* e velocidade atuais da partícula, e também a melhor posição e melhor *fitness* já encontrado pela partícula. Além disso, foram criadas duas variáveis para guardar o melhor *fitness* e a melhor posição entre todas as partículas. São definidas as quantidades de partículas e de iterações de modo a determinar qual o tamanho da nuvem e quantas vezes o programa irá rodar, procurando a solução.

As partículas são inicializadas aleatoriamente, e a partir daí, o programa começa a rodar procurando a melhor solução para o problema. Para cada partícula, a posição é usada para verificar o *fitness*; caso o novo *fitness* seja melhor que o melhor *fitness* da partícula e/ou global, as atualizações pertinentes são realizadas; a nova velocidade e a nova posição são calculadas; até o número de iterações ter sido atingido.

3.1 *Sagacious Bird* – PSO Discreto – Caixeiro Viajante

O SB soluciona PCVs que possuem de 4 a 26 cidades. Os problemas podem ser simétricos ou assimétricos, ou seja, as distâncias de ida e volta entre as cidades podem ser iguais ou não. Ainda é possível fixar a cidade inicial ou não, ou seja, a solução deverá partir de uma cidade específica ou pode partir de qualquer uma das cidades.

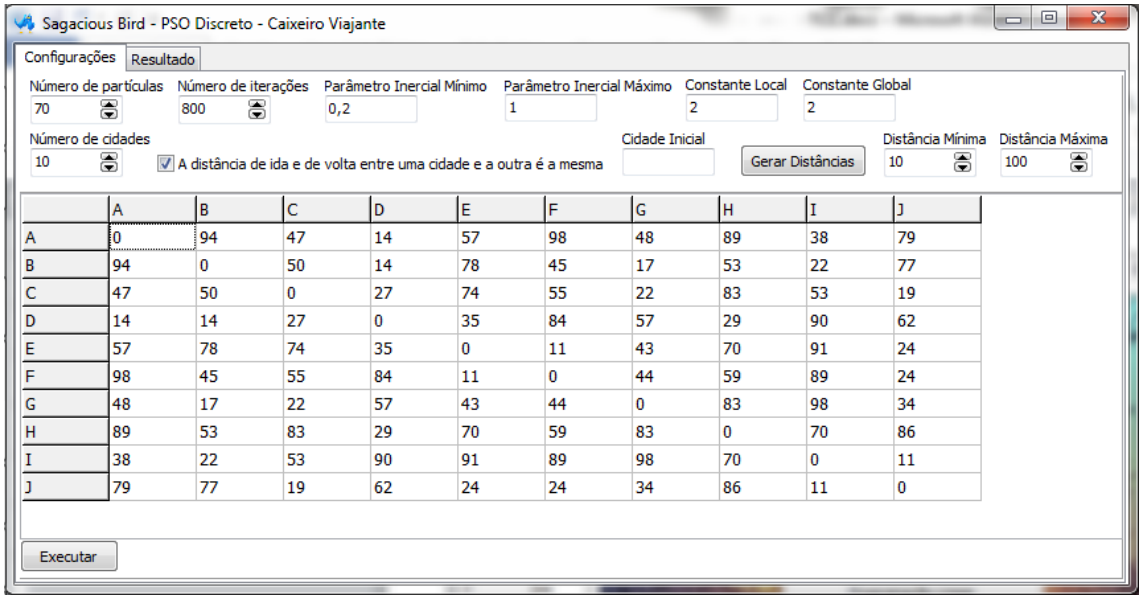
Na resolução do PCV:

- A posição da partícula é uma possível solução, ou seja, uma lista com a sequência de cidades a serem visitadas. Por exemplo: (Maringá, Londrina, Apucarana, Mandaguari, Sarandi).
- A velocidade da partícula é o fator que fará a partícula mudar sua posição, alterando a possível solução, ou seja, uma lista de trocas na lista da sequência de cidades. Por exemplo: [(1,2), (3,5)] resultando na nova posição (Londrina, Maringá, Sarandi, Mandaguari, Apucarana).
- O *fitness* da partícula é o valor da soma das distâncias das cidades listadas na posição atual da partícula.

A Tabela 2 apresenta a relação dos termos do PSO com os termos do PCV.

Tabela 2 - Termos do PSO X Termos do PCV

PSO	Problema
Partícula	"Indivíduo" que busca as possíveis soluções para o problema
Posição da Partícula	Possível sequência de cidades a serem visitadas
Velocidade da Partícula	Operador de troca que gera uma nova sequência de cidades a serem visitadas
Fitness	Distância percorrida pelo caixeiro viajante ao percorrer a sequência de cidades



	A	B	C	D	E	F	G	H	I	J
A	0	94	47	14	57	98	48	89	38	79
B	94	0	50	14	78	45	17	53	22	77
C	47	50	0	27	74	55	22	83	53	19
D	14	14	27	0	35	84	57	29	90	62
E	57	78	74	35	0	11	43	70	91	24
F	98	45	55	84	11	0	44	59	89	24
G	48	17	22	57	43	44	0	83	98	34
H	89	53	83	29	70	59	83	0	70	86
I	38	22	53	90	91	89	98	70	0	11
J	79	77	19	62	24	24	34	86	11	0

Figura 11 - PSO Discreto - Caixeiro Viajante

A Figura 11 apresenta a tela de configuração do PSO Discreto resolvendo o PCV, na qual é possível definir:

- O número de partículas da nuvem. Define quantos "indivíduos" buscarão possíveis soluções a cada iteração;
- O número de iterações. Define quantas vezes as possíveis soluções serão atualizadas e avaliadas;
- O parâmetro inercial máximo, o parâmetro inercial mínimo, a constante local, e a constante global. Os parâmetros inerciais definem se a busca terá um comportamento

mais local ou global. Valores altos geram uma busca global enquanto valores baixos geram uma busca local. Já os valores das constantes local e global determinam o quanto os valores individuais da partícula e os valores do enxame influenciarão no resultado, respectivamente;

- O número de cidades. É a quantidade de locais que o Caixeiro Viajante deve visitar;
- Se a distância de ida e de volta entre uma cidade e a outra é a mesma. Caso esta opção esteja marcada, quando a distância entre a cidade A e B for atribuída, o mesmo valor é atribuído para a distância entre a cidade B e A;
- A cidade inicial. Esta opção serve para definir a cidade inicial do caixeiro viajante. Se não estiver definida, qualquer cidade poderá ser a inicial;
- Gerar distâncias. Esta opção serve para atribuir as distâncias entre as cidades de forma aleatória limitadas aos valores definidos nas opções Distância Máxima e Distância Mínima;
- Por fim, tem-se a matriz de distâncias entre as cidades. Nela deve-se informar a distância entre as cidades, informando zero (0) para a distância entre a cidade e ela mesma, e informando "um negativo" (-1) para a distância entre cidades que não possuem um caminho que as liguem.

A Figura 12 mostra o início do resultado apresentado na resolução do PCV, exibindo as informações referentes ao melhor resultado da inicialização, e o resultado da melhor partícula nas três primeiras iterações.

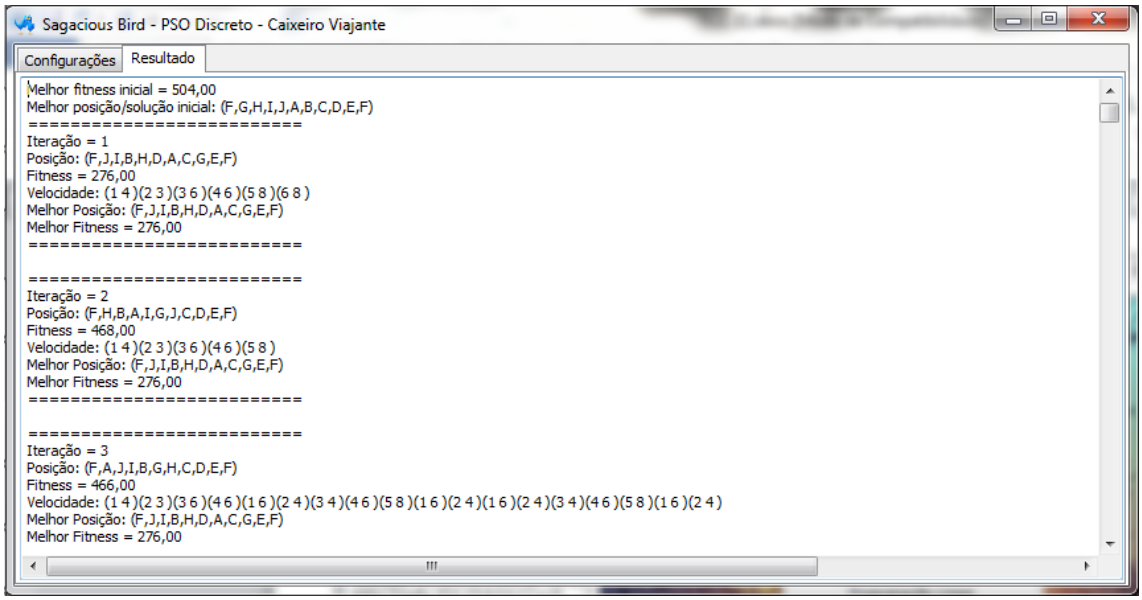


Figura 12 - PSO Discreto - Caixeiro Viajante - Início do Resultado

A Figura 13 apresenta o fim do resultado apresentado na resolução do PCV, contendo as informações referentes ao resultado da melhor partícula nas duas últimas iterações, e o resultado final, além do tempo de execução do algoritmo.

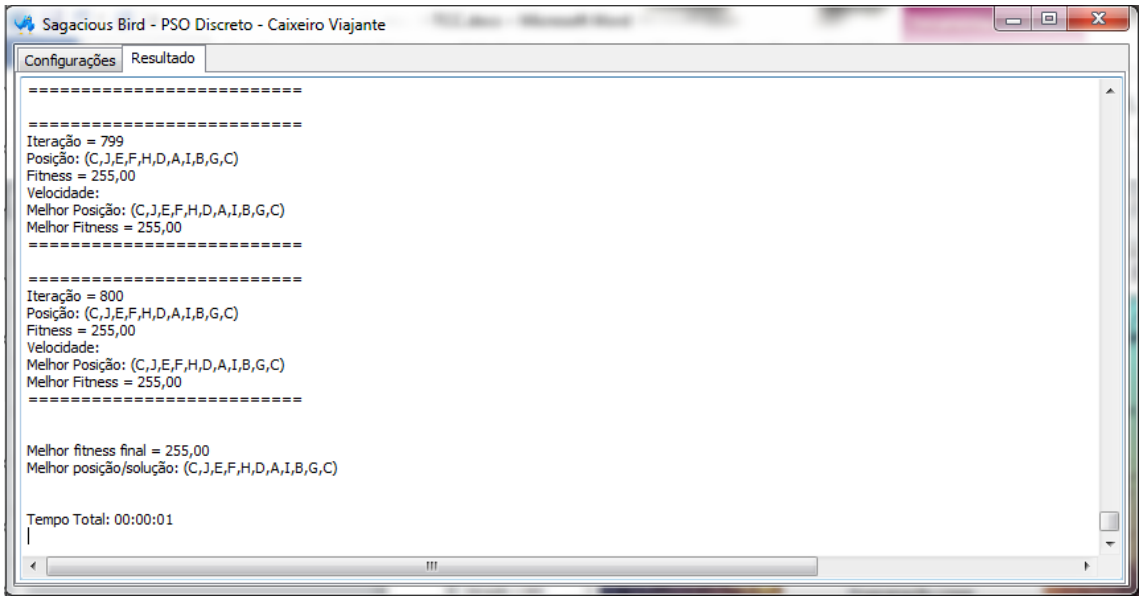


Figura 13 - PSO Discreto - Caixeiro Viajante - Fim do Resultado

3.2 *Sagacious Bird* – PSO Contínuo – Programação Linear

O SB resolve tanto os problemas de PL de maximizar quanto os de minimizar. As restrições do problema podem ser de menor (<), maior (>), igual (=), menor ou igual (<=) e maior ou igual (>=). Além disso, as restrições de não negatividade estão implícitas no programa, não sendo necessário adicioná-las manualmente. Portanto, pode-se notar a flexibilidade e facilidade ao utilizar o SB para resolução de problemas de PL.

Na resolução dos problemas de PL:

- A posição da partícula é uma possível solução, ou seja, um valor real para cada variável envolvida no problema.
- A velocidade da partícula é o fator que fará a partícula mudar sua posição, alterando a possível solução, ou seja, um valor diferente a ser somado em cada variável contida na posição da partícula.
- O *fitness* da partícula é o valor da função objetivo ao substituir os valores da posição às respectivas variáveis da função objetivo.

A Tabela 3 apresenta a relação dos termos do PSO com os termos de PL.

Tabela 3 - Termos do PSO X Termos de PL

PSO	Problema
Partícula	"Indivíduo" que busca as possíveis soluções para o problema
Posição da Partícula	Valores das variáveis do problema
Velocidade da Partícula	Termo que altera o valor das variáveis do problema
Fitness	Valor da função objetivo calculado com os valores da posição da partícula

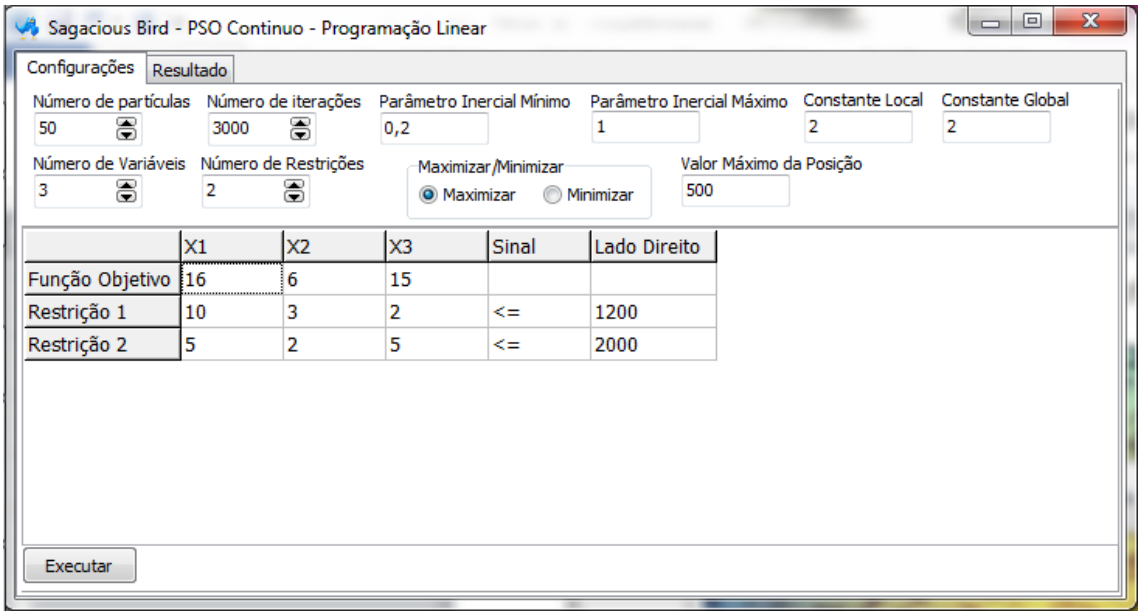


Figura 14 - PSO Contínuo - Programação Linear

A Figura 14 apresenta a tela de configuração do PSO Contínuo resolvendo problemas de PL, na qual é possível definir:

- O número de partículas da nuvem. Define quantos "indivíduos" buscarão possíveis soluções a cada iteração;
- O número de iterações. Define quantas vezes as possíveis soluções serão atualizadas e avaliadas;
- O parâmetro inercial máximo, o parâmetro inercial mínimo, a constante local, e a constante global Os parâmetros inerciais definem se a busca terá um comportamento mais local ou global. Valores altos geram uma busca global enquanto valores baixos

geram uma busca local. Já os valores das constantes local e global determinam o quanto os valores individuais da partícula e os valores do enxame influenciarão no resultado, respectivamente;

- O número de variáveis. Corresponde à quantidade de variáveis necessárias para solucionar o problema;
- O número de restrições. Não é preciso adicionar as restrições de não negatividade, pois estas já estão implícitas no desenvolvimento;
- Se o problema é de maximização ou minimização;
- Valor máximo da posição. Este valor é uma limitação à posição da partícula. Utiliza-se esta opção para que a busca pelo resultado não fique muito ampla, quando há o conhecimento prévio dos máximos valores que a solução pode assumir. Caso o valor seja zero (0) ou não seja informado, o limite será igual a um quadrilhão (1.000.000.000.000.000);
- Por fim, tem-se a matriz na qual são informados os valores referentes à função objetivo e às restrições do problema.

A Figura 15 mostra o começo do resultado apresentado na resolução do problema de PL, exibindo as informações referentes ao melhor resultado da inicialização, e o resultado da melhor partícula nas duas primeiras iterações.

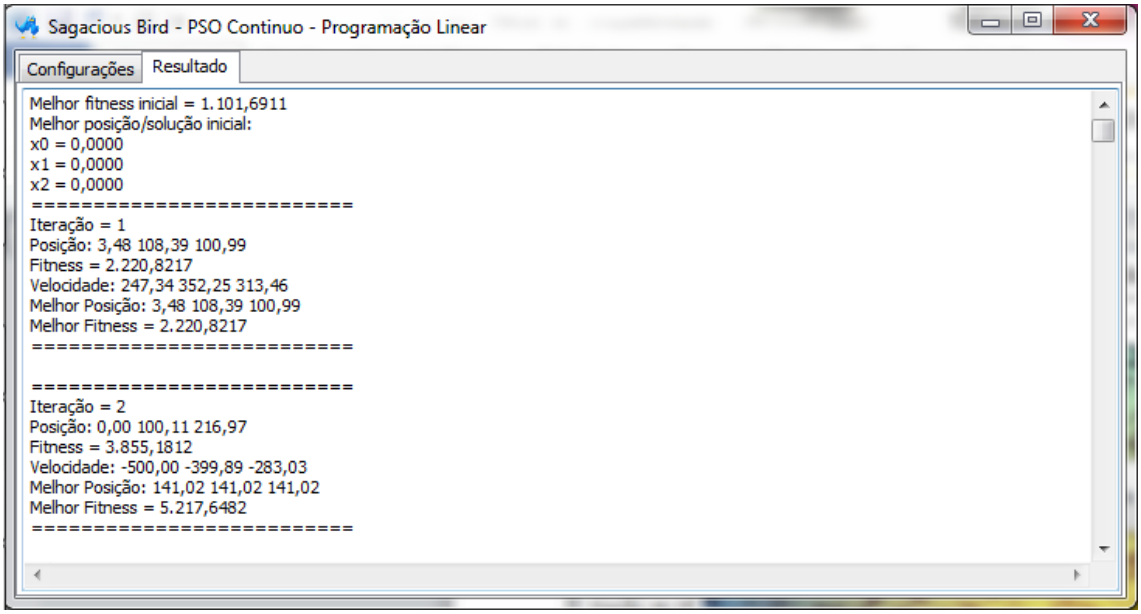


Figura 15 - PSO Contínuo - Programação Linear - Início do Resultado

A Figura 16 apresenta o fim do resultado apresentado na resolução do problema de PL, contendo as informações referentes ao resultado da melhor partícula na última iteração, e o resultado final, além do tempo de execução do algoritmo.

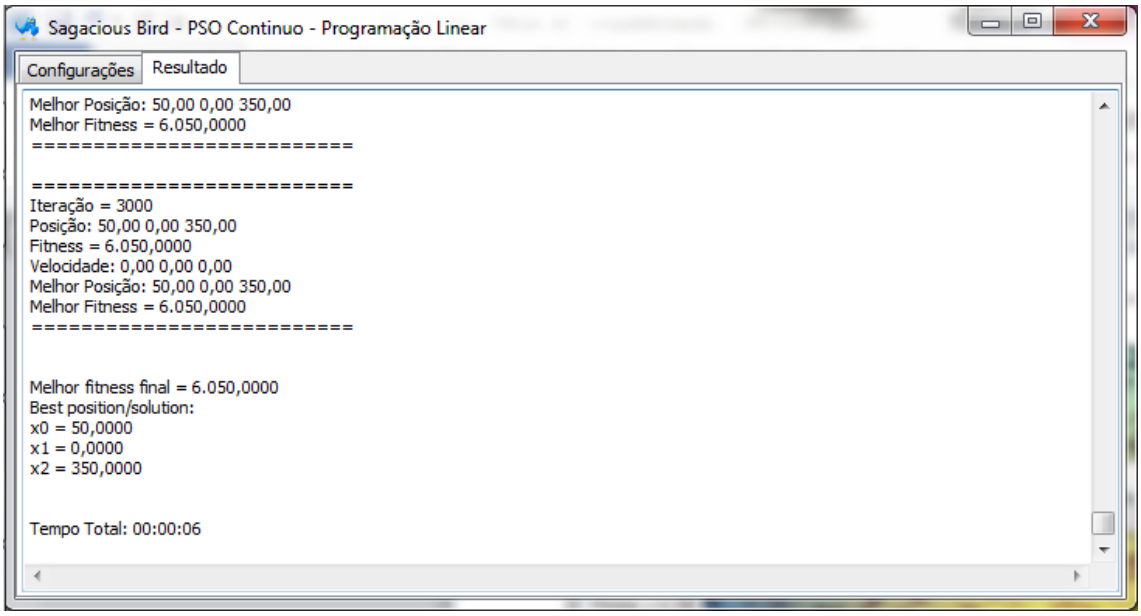


Figura 16 - PSO Contínuo - Programação Linear - Fim do Resultado

3.3 Aplicação do *Sagacious Bird* em um Caso do PCV

Zayatz (2012) apresenta uma aplicação do Algoritmo Genético na resolução de um caso do PCV com nove cidades. As distâncias entre as cidades se encontra na Tabela 4.

Tabela 4 - Matriz de distâncias entre as cidades

	A	B	C	D	E	F	G	H	I
A	0	42	61	30	17	82	31	11	80
B	42	0	14	87	28	70	19	33	67
C	61	14	0	20	81	21	8	29	56
D	30	87	20	0	34	33	91	10	34
E	17	28	81	34	0	41	34	82	57
F	82	70	21	33	41	0	19	32	65
G	31	19	8	91	34	19	0	59	31
H	11	33	29	10	82	32	59	0	43
I	80	67	56	34	57	65	31	43	0

Fonte: Zayatz, 2012, adaptado

Utilizando uma população de 20 cromossomos e 3000 iterações, o autor encontrou a melhor solução, que é igual a 185, em menos de 1 segundo.

Aplicando o mesmo problema no *Sagacious Bird* encontrou-se o mesmo resultado, 185, em menos de 1 segundo, porém utilizando somente 15 partículas e 1500 iterações. Portanto, pode-se perceber que o SB chegou à solução ótima utilizando menos da metade dos recursos utilizados no AG.

O código foi rodado no ambiente de desenvolvimento Delphi 2007, em computador equipado com processador Intel® Core™ I5-M460, 2,53 GHz e 4 GB de memória. O sistema operacional utilizado foi o Windows 7, da Microsoft Corporation.

O trabalho realizado por Zayatz (2012) também teve cunho didático, proporcionando a possibilidade da comparação entre os metaheurísticas AG e PSO aplicadas no mesmo problema, o PCV.

4. CONSIDERAÇÕES FINAIS

4.1 Contribuições

O referencial teórico sobre o PSO foi apresentado neste trabalho de forma simples e objetiva, com exemplos e ilustrações, facilitando o aprendizado desta metaheurística.

A ferramenta desenvolvida, o *Sagacious Bird*, utiliza o método do PSO para resolver problemas de PCV e de PL. É uma ferramenta projetada para ser muito fácil de ser utilizada, contando com uma interface gráfica autoexplicativa desenvolvida para que mesmo usuários com pouco conhecimento na área possam utilizar.

O SB permite a resolução de diversos problemas de PCV e de PL, com diversas características e tamanhos diferentes. Além disso, todos os parâmetros que interferem na performance do PSO podem ser alterados. Esta flexibilidade e o fato de ser parametrizável faz do *Sagacious Bird* uma ferramenta didática muito útil aos professores e alunos da disciplina de metaheurísticas, pois proporciona a visualização real da aplicação do método e a experimentação e constatação de como os parâmetros influenciam na agilidade e desempenho do PSO.

Considerando que existem poucas ferramentas com o objetivo do SB, e que o material encontrado na literatura raramente é voltado para área da Engenharia de Produção, este trabalho servirá como texto complementar do material didático, e o software como ferramenta para melhor compreensão do assunto.

O principal objetivo do trabalho consistiu em desenvolver um software didático com o intuito de auxiliar os professores e alunos na disciplina de metaheurísticas. Este objetivo foi alcançado, além de mostrar que o PSO possui uma boa performance na resolução de problemas discretos, como o PCV.

4.2 Dificuldades e Limitações

As dificuldades encontradas referentes à revisão de literatura se deram por conta de não haver muitos materiais específicos da aplicação do PSO na resolução de problemas específicos da área de Engenharia de Produção. Além disso, a maior parte do material encontrado possui uma linguagem muito específica da área de Informática.

Quanto à implementação do software, a maior dificuldade foi encontrada no desenvolvimento do PSO Contínuo para resolução de problemas de PL. Esporadicamente o programa apresenta erro de memória e precisa ser fechado. A solução para este problema foi procurada exaustivamente, porém sem sucesso.

4.3 Trabalhos Futuros

A partir deste trabalho pode-se vislumbrar como trabalhos futuros a incorporação de outras metaheurísticas, realizando comparações entre as mesmas, ao aplicá-las nos problemas de Engenharia de Produção. Além de adicionar novas metaheurísticas, seria interessante buscar resolver outros tipos de problemas da área de produção. Por fim, implementar novas metaheurísticas no programa desenvolvido neste trabalho, o tornaria uma ferramenta mais completa e didática.

REFERÊNCIAS

ALOISE, D.; OLIVEIRA, M.; SILVA, T.. Otimização discreta por nuvem de partículas aplicada ao problema do caixeiro viajante. **GEPROS,** Ano 1, n. 2. p. 87-95. abr 2006.

BRANDINI, P.. **Metaheurística *Particle Swarm* Utilizada para Alocação Ótima de Bancos de Capacitores em Sistemas de Distribuição Radial.** 2007. 128 f. Tese (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia – UNESP, Ilha Solteira.

CHAVES, A.. Modelagens Exata e Heurística para Resolução do Problema do Caixeiro Viajante com Coleta de Prêmios. **Trabalho de Conclusão de Curso.** Departamento de Ciência da Computação – Universidade Federal de Ouro Preto, 2003.

CONSELHEIRO, F. **Estudo da Aplicação da Estratégia de Simulated Annealing aos Problemas de Programação da Produção em Unidades Batelada**. Dissertação (Mestrado em Engenharia Química) – Universidade Estadual de Campinas. 1999.

CORDENONSI, A. **Ambientes, Objetos e Dialogicidade: Uma Estratégia de Ensino Superior em Heurísticas e Metaheurísticas**. 2008. 228 f. Tese (Doutorado em Informática na Educação) – Programa de Pós-Graduação em Informática na Educação – UFRGS, Porto Alegre.

FUCHS, S.; DELGADO, M.; LÜDERS, R.. PSO para Otimização Combinatória: Uma Análise da Equação de Atualização da Velocidade das Partículas. **CLAIO SBPO.** Rio de Janeiro, setembro de 2012.

GARCIA, V.. **Metaheurísticas multiobjetivo para o problema de restauração do serviço em redes de distribuição de energia elétrica**. 2005. 197 f. Tese (Doutorado em Engenharia Elétrica) **–** Faculdade de Engenharia Elétrica e de Computação – UNICAMP, Campinas.

GIL, A.. **Como Elaborar Projetos de Pesquisa**. 4ª ed. São Paulo: Editora Atlas, 2002.

HILLIER, F.; LIEBERMAN, G.. **Introdução à Pesquisa Operacional.** 8ª ed. São Paulo: McGraw-Hill, 2006.

KENNEDY, J.; EBERHART, R.. Particle Swarm Optimization. **Proceedings of IEEE International Conference on Neural Networks**, v. 4, p. 1942-1948. nov/dez 1995.

KENNEDY, J.; EBERHART, R.. A Discrete Binary Version of the Particle Swarm Algorithm. **IEEE International Conference on Systems, Man, and Cybernetics**, v. 5, p. 4104–4108. 1997.

LEAL, G.. Extensão de um Algoritmo Cultural para problemas de despacho de Energia Elétrica. **Trabalho de Conclusão de Curso**. DEP-UEM, 2007.

LUZ, E.. **Estimação de Fonte de Poluição Atmosférica Usando Otimização por Enxame de Partículas.** 2008. 84 f. Tese (Mestrado em Computação Aplicada) – Pós Graduação em Computação Aplicada – INPE, São José dos Campos.

MAIA, P.. Uma Metaheurística Híbrida Paralela para o Problema do Caixeiro Viajante. **Trabalho de Conclusão de Curso.** Departamento de Computação e Automação – Universidade Federal do Rio Grande do Norte, 2009.

MAREDIA, A.. History, Analysis, and Implementation of Traveling Salesman Problem (TSP) and Related Problems. **Trabalho de Conclusão de Curso.** Curso de Matemática – University of Houston – Downtown, 2010.

MARTINEZ, J.; SANTOS, S.. Métodos Computacionais De Otimização, Impa, 1995.

MELIÁN, B.; PÉREZ, J.; VEJA, J.. Metaheuristics: A Global View. **Revista Iberoamericana de Inteligencia Artificial** (Asociación Española de Inteligencia Artificial), No. 19, p. 7-28. 2003.

MELO, E.. Implementação de um Algoritmo Genético para uma Aplicação do Problema do Caixeiro Viajante. **Trabalho de Conclusão de Curso**. DEP-UEM, 2006.

PRADO, J.; SARAMAGO, S.. Otimização por Colônia de Partículas. **Revista Cientifica Eletrônica da Faculdade de Matemática**, Universidade Federal de Uberlândia, n. 4. p. 87-103. abr 2005.

PRESTES, A.. **Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante.** 2006. 74 f. Tese (Mestrado em Sistemas e Computação) – Universidade Federal do Rio Grande do Norte, Natal.

SERAPIÃO, A.. Fundamentos de Otimização por Inteligência de Enxames: Uma Visão Geral. **Revista Controle & Automação**, UNESP, n.3. p. 271-304. ago/set 2009.

TRIGUEIROS, D.. **Reutilização de água em processos industriais: uma abordagem metaheurística**. 2011. 190 f. Tese (Doutorado em Engenharia Química) – Programa de Pós-Graduação em Engenharia Química – UEM, Maringá.

WAINTRAUB, M.. **Algoritmos Paralelos de Otimização por Enxame de Partículas em Problemas Nucleares.** 2009. 97 f. Tese (Doutorado em Engenharia Nuclear) – Programa de Pós-Graduação em Engenharia Nuclear – Universidade Federal do Rio de Janeiro, Rio de Janeiro.

ZAYATZ, J.. Aplicação de Meta-Heurísticas à Engenharia de Produção. **Trabalho de Conclusão de Curso.** DEP – UEM, 2012.

Universidade Estadual de Maringá
Departamento de Engenharia de Produção
Av. Colombo 5790, Maringá-PR CEP 87020-900
Tel: (044) 3011-4196/3011-5833 Fax: (044) 3011-4196