Daniel Zanin Delago

Renato Sanchez

Posicionamento rotacional de polígonos em recipientes fechados utilizando enxame de partículas

Monografia de Conclusão de Curso apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro

São Paulo
2008

Daniel Zanin Delago

Renato Sanchez

Posicionamento rotacional de polígonos em recipientes fechados utilizando enxame de partículas

Monografia de Conclusão de Curso apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Marcos de Sales Guerra Tsuzuki

São Paulo
2008

Ficha Catalográfica

Delago, Daniel Zanin
Sanchez, Renato
Posicionamento rotacional de polígonos em recipientes fechados utilizando enxame de partículas. São Paulo, 2008. 59 p.

Monografia de Conclusão de Curso (Graduação) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Mecatrônica e de Sistemas Mecânicos.

1. Otimização. 2. Polígonos. 3. Enxame de partículas. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Mecatrônica e de Sistemas Mecânicos. II.

a Deus

e o Céu.

Agradecimento

a Marcos de Sales Guerra Tsuzuki, por todo o apoio e conhecimento
a nossa família, pela compreensão e suporte,
aos amigos, pela companhia e incentivo.

Resumo

O posicionamento rotacional polígonos em recipientes fechados de modo a minimizar a quantidade de matéria prima desperdiçada é um problema NP-completo [7]. Ou seja, um problema sem solução computacional que seguramente determina o posicionamento ótimo. Utilizam-se então, métodos heurísticos probabilísticos para determinar soluções quase-ótimas. Os parâmetros que definem o posicionamento são paramêtros contínuos como ângulo de rotação e deslocamento. Entretanto, o valor da função objetivo que corresponde à área desperdiçada, é discreto - visto que pode-se posicionar apenas uma quantidade inteira de polígonos. Estas características dificultam a sensibilidade do algoritmo. Para melhorar a sensibilidade do método, determina-se um fator de escala que ao ser aplicado a um polígono não posicionado, permite o seu posicionamento. A heurística probabilística a ser utilizada é o enxame de partículas, que possui como base modelos sócio-cognitivos bem simples: avaliação (dos estímulos, caracterizados como positivos, negativos, atrativos ou repulsivos), comparação (entre indivíduos, estabelecendo padrões de referência na sociedade mediante comparação entre indivíduos) e imitação (encontrada apenas em alguns animais, imita apenas os que ele julga superiores).

Abstract

The rotational placement of polygons in closed containers to minimise the amount of wasted material is a NP-complete problem[3]. That consists in a computational problem without solution that certainly provides the optimal placement. It uses probabilistic heuristics to determine near-optimal solutions. The parameters that define the placement are continuous parameters such as angle of rotation and translation. Meanwhile, the value of the objective function that corresponds to the wasted area is discreet - since you can only put an integer number of polygons. These characteristics harms the sensitivity of the algorithm. To improve the sensitivity of the method, a factor of scale is applied to a non placed polygon allowing its placement. The probability heuristic used is the swarm particles, which is based on quite simple socio-cognitive models: evaluation (of stimuli, characterized as positive, negative, attractive or repulsive), comparison (between individuals, establishing standards of reference in society by comparison between individuals) and imitation (found only in some animals, mimics that he believes to be better)

Conteúdo

Lista de Figuras

1 Introdução

Um dos grandes problemas encontrados em diversas áreas atualmente é o desperdício presente em diversas etapas de um processo produtivo. Em função disto, podemos dizer que um processo pode se tornar viável economicamente pelo simples fato de ter menor desperdício.

Um exemplo para este fato pode ser o que é encontrado na indústria têxtil onde este desperdício pode ser determinado pela quantidade de matéria-prima descartada no processo de corte. Neste tipo de indústria o problema pode ser definido como posicionar o maior número de peças a serem cortadas na menor área possível e com isso minimizar o desperdício.

A idéia inicial deste trabalho é utilizar enxame de partículas para posicionar polígonos em um recipiente fechado para desta forma encontrar uma solução ótima para o problema citado anteriormente.

Para que fosse possível chegar neste objetivo proposto num primeiro momento foi feito um estudo de todos os problemas encontrados quando trabalhamos com posicionamento de polígonos que será abordado na parte inicial deste trabalho. Após esta etapa foi então desenvolvido um estudo baseado em diversos livros e trabalhos que tratavam deste mesmo problema de otimização de posicionamento de polígonos, sendo que o principal descrito é a aplicação do recozimento simulado como método para tentativa de busca de uma melhor solução deste problema.

Em função do método adotado não ser muito simples de ser compreendido, optou-se por desenvolver uma interface gráfica executada em VBA (*Visual Basic for Applications*) onde através desta ferramenta a idéia que existe por trás deste método é facilmente explicada para que assim fosse possível partir para a criação

do software que tem por objetivo a otimização descrita.

Este programa foi desenvolvido utilizando-se linguagem C++ através da utilização de diversas bibliotecas de funções gráficas para que fosse possível a visualização das soluções encontradas, além é claro, de outras classes que tem a função do tratamento do posicionamento de polígonos.

2 Problema de Posicionamento

O problema de otimização de posicionamento de polígonos pode-se inicialmente ser dividido em 2 grupos que são ilustrados na figura abaixo. O primeiro, que seria o problema de empacotamento que consiste no arranjo de um número fixo de polígonos em um recipiente variável que são denominados por problemas duais e o outro que consiste no posicionamento de um número variável de polígonos em um recipiente fixo que podem ser chamados de problemas primais.

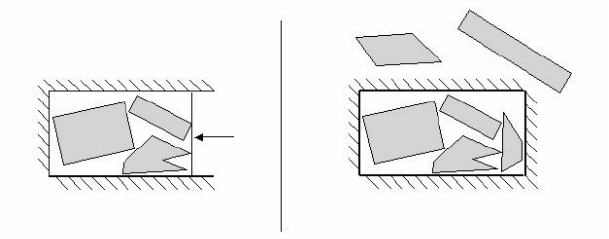


Figura 2.1: Características dos problemas de posicionamento.

Outra possível classificação para este problema pode ser descrita da seguinte forma como pode ser encontrada em [14]:

- Quanto à dimensão do problema. (1,2 ou 3)

- Objetivo: maximização de saída (nº de peças a serem produzidas) ; maximização de entrada (quantidade de matéria-prima utilizada em um processo).

- Recipientes (nº de recipientes ou recipiente com dimensões fixas ou variáveis).

- Conjunto de Objetos (Congruentes, fracamente heterogêneos ou fortemente heterogêneos).

- Restrições de forma nos objetos (formas simples como retângulos ou circunferências ou formas irregulares).

Este problema de otimização de posicionamento é um problema NP-completo. Com isso essas soluções que possuem um custo computacional crescente de forma exponencial acabam se tornando inviáveis para problemas mais complexos, servindo assim apenas para problemas mais simples e restritos.

No entanto a utilização desses algoritmos não é suficiente para a necessidade , por exemplo, da indústria têxtil em função de quando comparados com soluções geradas manualmente não representam vantagem. Em função disso tem-se utilizado abordagens heurísticas para este tipo de otimização.

Essa abordagem heurística consiste na obtenção de regras de posicionamento que muitas vezes simulam as atividades de um ser humano que desempenha a função de gerar o posicionamento desses polígonos. As regras para este posicionamento podem ser classificadas como determinísticas ou probabilísticas.

Um exemplo de regra determinística pode ser a chamada "bottom-left" , onde como o próprio nome diz ordena o posicionamento de uma série de polígonos sempre utilizando a posição mais abaixo e mais à esquerda disponível como pode ser observado na figura abaixo.

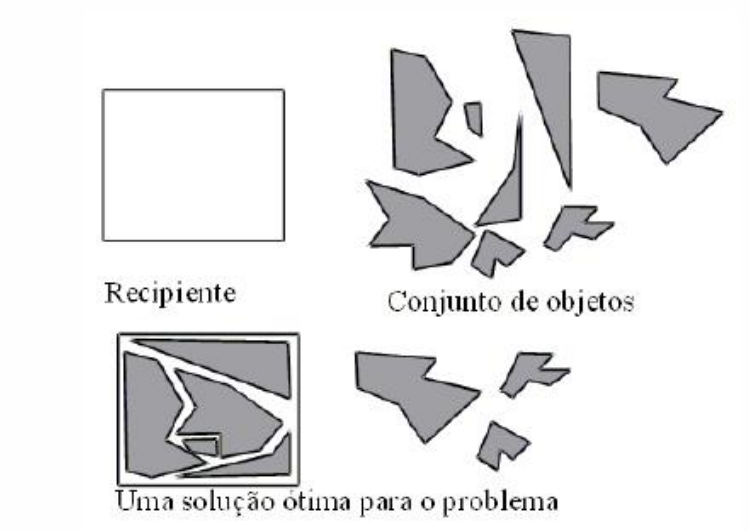


Figura 2.2: Otimização para problema de empacotamento.

No entanto esta regra embora pareça ser interessante para diversos problemas, a mesma não se torna eficiente quanto estamos lidando com problemas onde se torna necessário uma análise de, por exemplo, um posicionamento rotacional([8, 11]).

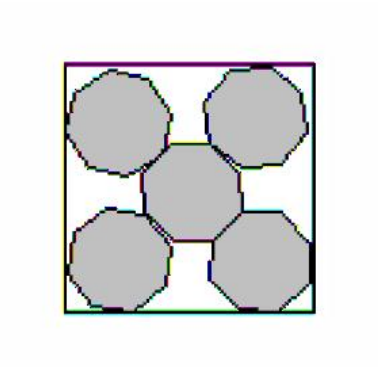


Figura 2.3: Problema com solução não alcançada apenas por regra "bottom-left".

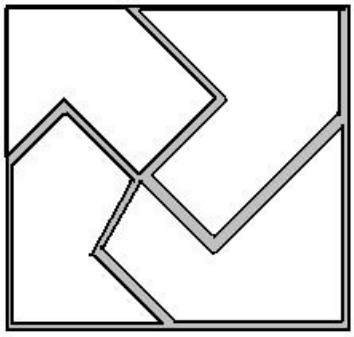


Figura 2.4: Posicionamento Rotacional.

Para a abordagem probabilística tem-se como o exemplo a utilização de recozimento simulado que consiste em simular o movimento dos polígonos a serem posicionados como os movimentos dos átomos de um metal sofrendo o processo de recozimento onde os mesmos se movimentam de forma a minimizar a energia do sistema final sendo que durante o processo possa passar por um posicionamento de maior energia.

2.1 Polígono de Obstrução

Neste trabalho, ao se tratar do problema de posicionamento, utilizamos o conceito do polígono de obstrução (no-fit polygon) introduzido por [1] que representa a área obstruída por uma forma no posicionamento de outra. Esse polígono é gerado a partir de um ponto de referência do polígono a ser posicionado e "des-

lizando" este em torno do polígono já posicionado, podemos verificar, na figura abaixo, um exemplo da obtenção do polígono de obstrução.

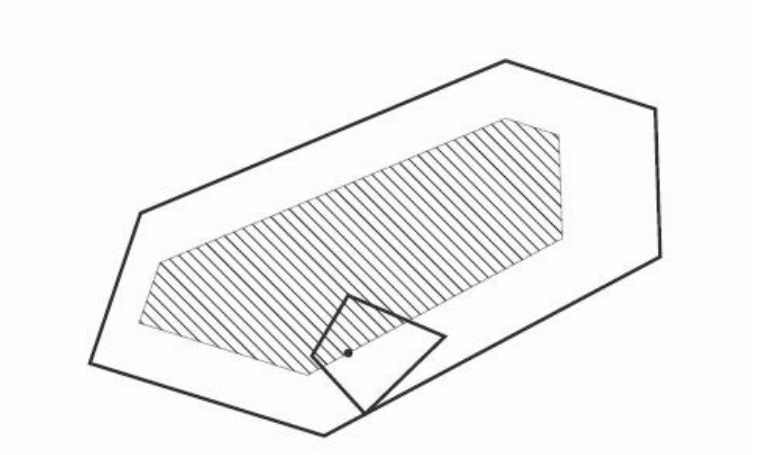


Figura 2.5: Posionamento de Poligono através de "deslizamento".

Assim teremos uma área limite, externa ao polígono já posicionado, que determina três regiões: Uma região de obstrução (para "dentro" da linha externa), uma região ótima ("em cima" da linha externa) e uma terceira região (para "fora" da linha externa).

É claro que analisando o problema dessa forma deve-se preocupar com a região para "dentro" que representa a região em que dois polígonos se interceptam. A figura abaixo ilustra o método de construção da área interceptada.

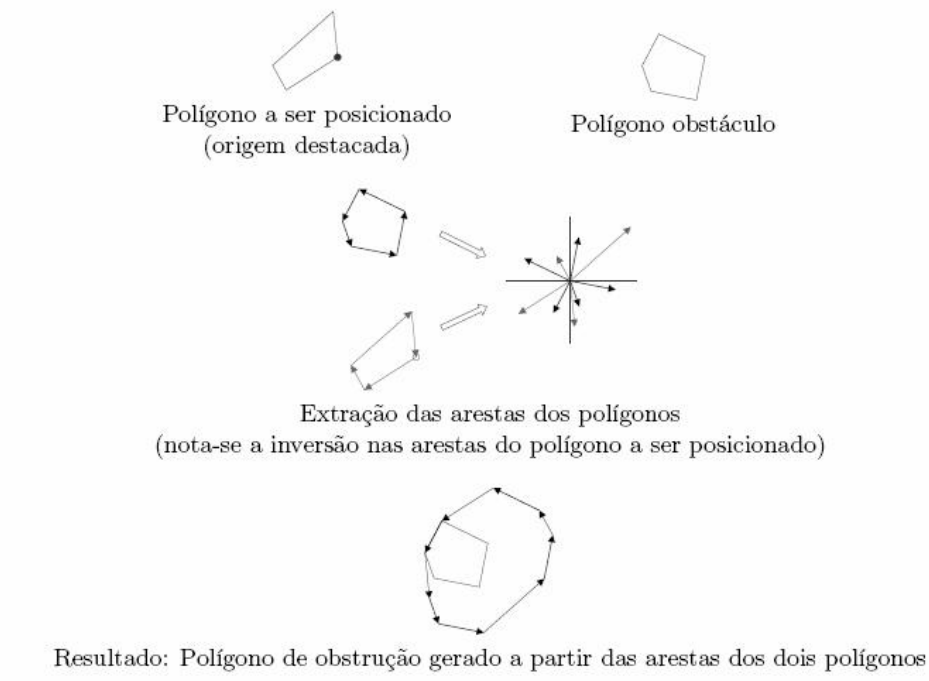


Figura 2.6: Polígono de Obstrução.

Muitos trabalhos que tratam do posicionamento dual forma desenvolvidos por

diversos autores, porém soluções puramente algorítmicas são eficientes apenas quando se trata de um número pequeno de formas a serem posicionadas. A não eficiência de abordagens algorítmicas vêm do fato, já mencionado, de que problemas de posicionamento tendem a ser NP-Difícil.

O custo computacional de tais problemas cresce exponencialmente com o tamanho dos problemas daí só são eficientes quando o problema trata de um pequeno número de entradas (formas).

Para "aliviar" problemas NP-Difícil como os expostos acima, utiliza-se a técnica de se adotar algoritmos de aproximação, ou seja, algoritmos que procuram soluções que ficam dentro de certos limites de desempenho quando comparadas às soluções ótimas.

Abordagens heurísticas envolvem regras de posicionamento empíricas que muitas vezes imitam o comportamento de especialistas humanos na geração de leiautes. Um tipo de abordagem heurística determinística é a "bottom-left" em que, normalmente, as formas são ordenadas de acordo com o tamanho ficando a maior forma abaixo no lado esquerdo do recipiente.

Abordagens heurísticas probabilísticas baseam-se, geralmente, em meta-heurísticas probabilísticas de otimização. As mais comuns são:

- Busca Tabu

- Algoritmos Genéticos

- Recozimento Simulado

2.2 Função Objetivo e Dualidade

Ao se tratar do posicionamento de formas, ou seja, encontrar as translações e rotações ao se posicionar formas em um recipiente, há uma diferença quando falamos de casos primais e casos duais.

Ao se analisar a função objetivo (área restante após posicionamento de formas) verificamos que no caso dual trata-se de uma função de variáveis contínuas

(Figura 2.7), porém no caso primal essa função admite valores discretos (Figura 2.8).

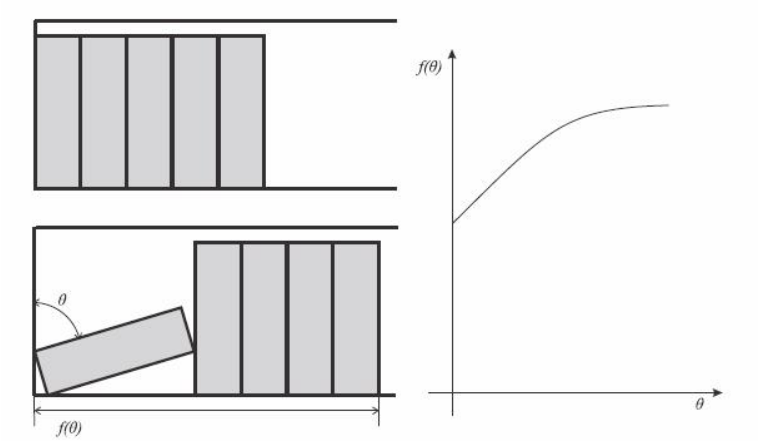


Figura 2.7: Função Objetivo para o problema do tipo dual.

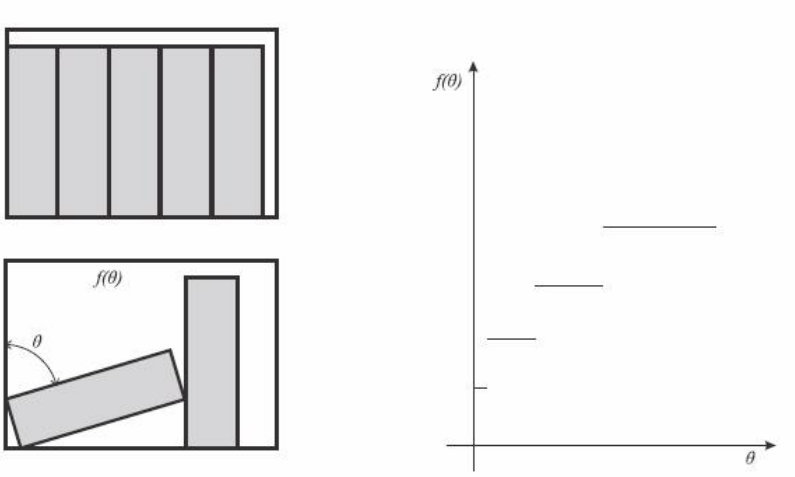


Figura 2.8: Função Objetivo para o problema do tipo primal.

2.3 Penalização Externa

Como foi dito anteriormente, adotam-se heurísticas probabilísticas ao se tratar de problemas de posicionamento já que estes são, normalmente, NP-Difícil e, então, não possuem soluções algorítmicas favoráveis.

Geralmente, essas heurísticas probabilísticas "relaxam" o problema original permitindo pontos fora do espaço de soluções válidas e aplicando penalizações à função objetivo de pontos externos [13].

Essa penalização externa, porém, pode fazer com que o processos de otimização produza soluções inválidas e a aplicação de penalizações pode não garantir que soluções ótimas do problema relaxado sejam soluções válidas do problema original.

Esse problema pode ser exemplificado pela Figura 2.9 onde temos como variável apenas o comprimento total x que corresponde ao tamanho do recipiente capaz de conter as duas formas da figura. A função objetivo para este problema pode ser obtida justamente por esta variável, ou seja, $f(x) = x$ quando temos uma condição de não sobreposição, ou seja, $x \geq b+h$. Sendo que para este caso podemos determinar a solução ótima que seria dada por $x_{otimo} = b + h$.

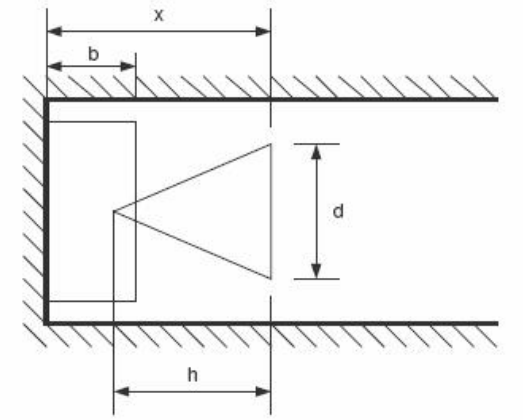


Figura 2.9: Posicionamento de duas formas com análise de sobreposição.

Mas quando utilizamos a penalização externa para o caso onde temos sobreposição das duas formas, a função objetivo para este exemplo pode ser dada pelas duas equações abaixo, onde k representa a quantidade de penalização aplicada em função desta sobreposição. Sendo que a Equação 2.1 é utilizada para o caso onde $x < b+h$ e a Equação 2.2 quando $x \geq b+h$. E com isso o formato desta função pode ser observado no gráfico da Figura 2.10.

$$f(x) = x + k * \frac{d}{2h}(b + h - x) * (b + h - x) \tag{2.1}$$

$$f(x) = x \tag{2.2}$$

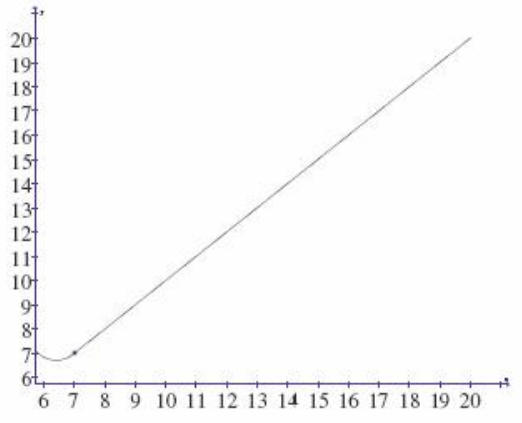


Figura 2.10: Função Objetivo com penalização.

3 Algoritmos de Otimização

3.1 Algoritmos Genéticos

Os algoritmos genéticos formam a parte da área dos sistemas inspirados na natureza; simulando os processos naturais e aplicando-os à solução de problemas reais. São métodos generalizados de busca e otimização que simulam os processos naturais de evolução, aplicando a idéia darwiniana de seleção. De acordo com a aptidão e a combinação com outros operadores genéticos, são produzidos métodos de grande robustez e aplicabilidade.

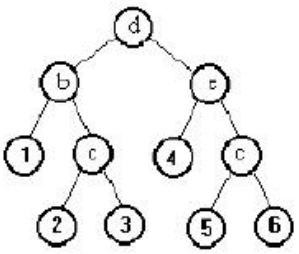


Figura 3.1: Árvore binária para solução utilizando algorítmo genético.

Estes algoritmos estão baseados nos processos genéticos dos organismos biológicos, codificando uma possível solução a um problema de "cromossomo" composto por cadeia de bits e caracteres. Estes cromossomos representam indivíduos que são levados ao longo de várias gerações, na forma similar aos problemas naturais, evoluindo de acordo com os princípios de seleção natural e sobrevivência dos mais aptos, descritos por [5]. Emulando estes processos, os algoritmos genéticos são capazes de "evoluir" soluções de problemas do mundo real. Na natureza os indivíduos competem entre si por recursos como comida, água e refúgio.

Adicionalmente, entre os animais de uma mesma espécie, aqueles que não obtêm êxito tendem provavelmente a ter um número reduzido de descendentes,

tendo portanto menor probabilidade de seus genes serem propagados ao longo de sucessivas gerações.

A combinação entre os genes dos indivíduos que perduram na espécie, podem produzir um novo indivíduo muito melhor adaptado às características de seu meio ambiente. Os algoritmos genéticos utilizam uma analogia direta deste fenômeno de evolução na natureza, onde cada indivíduo representa uma possível solução para um problema dado.

A cada indivíduo se atribui uma pontuação de adaptação, dependendo da resposta dada ao problema por este indivíduo. Aos mais adaptados é dada a oportunidade de reproduzir-se mediante cruzamentos com outros indivíduos da população, produzindo descendentes com características de ambas as partes. Se um algoritmo genético for desenvolvido corretamente, a população (conjunto de possíveis respostas) convergirá a uma solução ótima para o problema proposto. Os processos que mais contribuem para a evolução são o crossover e a adaptação baseada na seleção/reprodução. A mutação também tem um papel significativo, no entanto, seu grau de importância continua sendo assunto de debate.

O Algoritmo Genético pode convergir em uma busca de azar, porém sua utilização assegura que nenhum ponto do espaço de busca tem probabilidade zero de ser examinado. Toda tarefa de busca e otimização possui vários componentes, entre eles: o espaço de busca onde são consideradas todas as possibilidades de solução de um determinado problema, e a função de avaliação (ou função de custo), uma maneira de avaliar os membros do espaço de busca como descrito em [4].

As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado utilizando algumas heurísticas (estáticas) diretamente associadas ao problema a ser solucionado. Por outro lado, as técnicas de computação evolucionária operam sobre uma população de candidatos em paralelo.

Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões. Os algoritmos genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, principalmente em quatro aspectos:

1 - AGs trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros;

2 - AGs trabalham com uma população e não com um único ponto;

3 - AGs utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar;

4 - AGs utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas, em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais. Os pesquisadores referem-se a "algoritmos genéticos" ou a "um algoritmo genético" e não "ao algoritmo genético", pois AGs são uma classe de procedimentos com muitos passos separados, e cada um destes passos possui muitas variações possíveis. Os AGs não são a única técnica baseada em uma analogia da natureza.

3.2 Recozimento Simulado

O recozimento simulado é uma meta-heurística probabilística de exploração local, isto é, explora uma e somente uma solução do problema a cada iteração [6]. O custo dessa solução é comparado com o custo da solução anterior e, à partir de algumas regras, uma nova solução é gerada para a próxima iteração.

Considerando duas soluções para o problema x_i e x_i^*, onde x_i é a solução atual e x_i^* é a solução analisada, a solução atual será substituída pela solução analisada segundo as seguintes regras:

- $f(x_i^*) \leq f(x_i)$

a solução analisada na iteração tem custo menor ou igual ao da solução atual. Neste caso, a solução atual é imediatamente substituída pela solução analisada.

- $f(x_i^*) > f(x_i)$

a solução analisada na iteração tem custo superior ao da solução atual. Neste caso, a solução atual pode ser substituída pela analisada com probabilidade determinada pela diferença de custos e pela "temperatura" do processo.

A "temperatura" é um parâmetro que varia de acordo com o programa de resfriamento. Esse programa de resfriamento ditará as regras adotadas pelo processo para determinar quando trocar uma solução por uma nova, quando continuar trocando e quando deve parar na solução que, para o programa, será a solução ótima

(melhor solução).

Existem, no entanto, algumas diretrizes que devem ser levadas em conta sobre métodos de otimização de exploração local (que é o caso do recozimento simulado):

1 - Deve ser fácil gerar soluções válidas

2 - Para cada solução factível, deve existir um caminho ligando esta solução a uma solução ótima.

3 - A vizinhança de uma dada solução deve ser relativamente próxima a ela.

4 - A topologia do espaço induzida pela função objetivo no espaço de soluções factíveis não deve ser demasiado plana.

3.3 Inteligência de Enxame

Inteligência de Enxame é uma técnica de inteligência artificial baseada no estudo do comportamento colectivo em sistemas descentralizados e auto-organizados.

Os sistemas de informação são tipicamente constituídos por uma população de agentes simples interagindo localmente uns com os outros e com o seu ambiente. Apesar de não existir, normalmente, uma estrutura de controle central que dita como os agentes se devem comportar, as interacções locais entre os agentes levam a um comportamento global. Exemplos de sistemas como este podem ser encontrados na natureza, em colônias de formigas, bandos de pássaros (Figura 3.2) ou manadas de animais.

Figura 3.2: Representação da Inteligência de Enxame.

Existem vários algoritmos de inteligência de enxame, por exemplo: Otimiza-

ção por Colônia de Formigas e Otimização por Enxame de Partículas, que será o assunto tratado posteriormente neste trabalho

3.4 Enxame de Partículas

Otimização por Enxame de Partículas, normalmente identificado como PSO devido à denominação inglesa Particle Swarm Optimization (PSO) , é uma forma de Inteligência por Enxame. Foi desenvolvido em 1955 por Eberhart e Kennedy inspirado no comportamento social de um bando de pássaros ou um cardume de peixes [2, 3, 9, 12]. Trata-se de uma técnica de optimização estocástica.

Imaginando um bando de pássaros, ou um cardume de peixes, se um dos membros do grupo vê um caminho promissor a seguir, por exemplo, a busca de comida, segurança, o resto do grupo ou enxame é capaz de o seguir rapidamente, mesmo que se encontrem no lado oposto do enxame. Os exemplos mais comuns encontrados quando se fala em otimização por enxame de partículas são, o bando de pássaros e o cardume de peixes.

Normalmente para facilitar uma exploração adequada do espaço de procura, pretende-se que cada partícula que por analogia pode ser visto como um membro do enxame tenha um certo nível de aleatoriadade no movimento que executa, para que o movimento do enxame tenha capacidades exploratórias ao longo de todo o dominio de procura de posicionamento e rotação de um polígono para o caso desta abordagem. Os membros do enxame deverão ser influenciados pelo resto do enxame, mas deverão, também, até certo ponto, explorar o mesmo independentemente. Esta descrição é um exemplo do que acontece num problema básico de exploração.

3.4.1 Aplicação do Método

A otimização por Enxame de Partículas é apropriada para lidar com problemas em que a melhor solução pode ser representada como um ponto ou uma superfície num espaço n-dimensional.

Assim, este tipo de problemas é modelado por partículas que se encontram no espaço multidimensional e têm uma posição e velocidade. Estas partículas voam pelo espaço.

Com isso, esta abordagem tem se tornado muito eficiente para abordagens de problemas do tipo NP-completo como é o caso do problema de posicionamento de polígonos em recipientes fechados, onde não podemos obter uma solução exata para o processo, mas através de métodos de otimização buscar configurações que se mostrem melhores que outras de acordo com alguns critérios que serão exemplificados neste trabalho.

3.4.2 Estrutura do Algoritmo Proposto

Nos algoritmos de otimização por enxame de partículas, as partículas têm duas capacidades essenciais de raciocínio: a memória da sua melhor posição e o conhecimento da melhor do enxame. Melhor significa a posição com o menor valor para a função objetivo adotada de forma a ser minizada para encontrarmos neste caso o posicionamento ótimo. Os membros de um enxame comunicam as boas posições entre todos e ajustam a própria posição e rotação baseando-se nestas boas posições.

Existem duas formas principais para efetuar esta comunicação:

- Um posicionamento nomeado como sendo o melhor global até determinado momento é atualizado a todas as partículas do enxame assim que qualquer uma delas encontre outra que se torne mais eficiente que a anterior utilizada como melhor.

- Cada partícula porém pode se comunicar com um subconjunto menor de forma a obter uma configuração ótima local ou de vizinhança que também será atualizado assim que tivermos um membro do enxame com configuração superior a utilizada anteriormente.

Algoritmos do segundo tipo permitem uma melhor exploração do espaço de respostas e reduzem a probabilidade do algoritmo encontrar apenas um mínimo local, mas também em função da sua complexidade é de se esperar que a sua convergência tenha uma velocidade menor e portanto a melhor solução seria um estudo prévio do problema a ser otimizado e com isso a escolha do método que melhor se encaixa para ser aplicado ao mesmo.

No entanto devemos tomar cuidado para que esta comunicação entre vizinhanças não seja responsável por criarmos sub-enxames que provavelmente não irão convergir para a melhor solução e sim que esta abordagem terá como objetivo

apenas a proliferação de posicionamentos adotados como melhores para todo o enxame.

Podemos também acrescentar que a utilização de vizinhanças menores levam a uma menor velocidade de convergência enquanto que quando abordamos o problema com vizinhanças maiores temos teoricamente uma maior velocidade. Em função disto temos que a melhor solução seria adotarmos vizinhanças pequenas que a medida que forem encontrando bons posicionamentos locais, estas se juntem a outras até que uma vizinhança esteja representando todo o enxame estudado.

Para que esta metodologia seja entendida vamos utilizar o exemplo onde um grupo de aves procura aleatoriamente comida numa área e que nessa área exista apenas um pedaço de comida.

As aves não sabem a localização da comida, mas sabem a que distância se encontra em cada iteração. Com isso podemos dizer que a melhor estratégia para encontrar a comida seja seguir a ave que se encontra mais perto da comida, que neste caso poderia ser representada pela função objetivo proposta para o problema de posicionamento.

A PSO utiliza cenários deste tipo para resolver problemas de otimização, onde cada elemento que para este exemplo é denominado "ave" representa uma solução para o posicionamento de todos os polígonos. Com isso temos que cada "ave" ou solução será avaliada de forma a obtermos um melhor posicionamento global. Para este exemplo podemos dizer que as soluções "voam" seguindo as partículas que são avaliadas como melhores mas sempre podendo se tornar "aves" que serão seguidas por estarem mais próximas da solução ótima.

A PSO é inicializada com um grupo aleatório de partículas (soluções) que depois procura a sua solução ótima. Em todas as iterações, cada partícula é atualizada seguindo os dois melhores valores, o primeiro é a melhor solução da partícula até o momento e o segundo valor é o melhor obtido pelo enxame. Mas sempre vale ressaltar que embora esteja seguindo a melhor solução encontrada até o momento a mesma gera um movimento aleatório de forma a fazer uma melhor exploração do espaço. Abaixo citamos alguns passos de forma a termos uma estrutura do algorítmo a ser implementado para otimização por enxame de partículas:

- Criação da população inicial

- Avaliação da população

- Inicialização da melhor posição obtida pelas partículas para a posição inicial

- Geração das candidatas a novas soluções e armazenar em P_i a melhor posição obtida pelo indivíduo

- Escolher a partícula que exerce a influência social

- Armazenar em P_b a melhor posição obtida por esta partícula

- Determinar o ponto P, como a média estocástica de P_i e P_b

- Modificar a velocidade da partícula paraque esta se movimente em outra direcção

- Avaliação da posição

- Armazenamento da nova posição caso esta seja melhor do que P_i

- A não ser que seja atingida uma das condições de parada, retornar a geração de candidatas para novas soluções

4 Estudo do Método Utilizado

Para um melhor estudo do método utlizado para solução do problema proposto foi desenvolvido um programa inicial que tem por objetivo apenas uma melhor visualização das possíveis soluções encontradas. Sendo que para isso é necessário analizarmos apenas soluções de dois parâmetros cada para que esta mesma possa ser mostrada como um ponto em um plano onde fica fácil a visualização.

4.1 Algoritmo

O algoritmo proposto é desenvolvido de acordo com os princípios básicos da utilização do PSO onde cada solução é criada de forma aleatória, onde esta criação é dada como apenas a determinação de todos os parâmetros que integram a mesma que neste caso são dois: coordenadas X e Y. Após isso também temos a inicialização das velocidades de movimentação das partículas que também é dada como variáveis aleatórias. Terminada esta primeira etapa do processo, o algoritmo executa a análise das atuais soluções para determinar a melhor da atual iteração e armazenar também a melhor encontrada até o momento. E com isso as soluções são atualizadas de acordo com o equacionamento abaixo:

$$\vec{V_i}(t) = w * \vec{V_i}(t-1) + A * [\vec{P_i} - \vec{X_i}(t-1)] + B * [\vec{P_g} - \vec{X_i}(t-1)] \tag{4.1}$$

$$\vec{X_i}(t) = \vec{X_i}(t-1) + \vec{V_i}(t) \tag{4.2}$$

Onde V_i representa a velocidade de movimentação da partícula e os parâ-

metros A e B são variáveis aleatórias que se encontram dentro do intervalo de 0 a 1, armazenadas com 32 bits e geradas através do algorítmo conhecido como *Mersenne Twister* que consiste em uma rotina mais eficiente, ou seja, que gera uma sequência de números durante mais tempo sem repetição e de forma mais rápida. Existe outro parâmetro w que representa a inércia de cada solução que pode ser modificado de acordo com cada problema que será estudado futuramente para que possamos tentar obter uma convergência mais veloz. E em algumas implementações este parâmetro que representa o momento de inércia é descrescente a cada iteração para que assim possa ser melhor explorado o espaço possível para as soluções "voarem" a procura do melhor ponto. As posições P_i e P_g representam respectivamente as coordenadas da melhor solução encontrada na iteração atual e a melhor global encontrada até o momento que será sempre modificada quando for encontrada uma melhor em alguma iteração. Outro parâmetro que pode ser incluído no processo é dado por V_{max} que seria a máxima velocidade que cada partícula pode apresentar. Este parâmetro se torna importante dado que experimentalmente é notado que valores grandes para a velocidade máxima favorecem uma procura global maior mas com isso é de esperar que as partículas possam passar por pontos onde se tenha melhores soluções sem que estas sejam aproveitadas pelo método. Mas também valores muito pequenos para V_{max} favorecem uma busca mais refinada mas com isso podemos não ter todo o espaço explorado e uma convergência do método mais lenta.

4.2 Resultados

Os resultados obtidos pelo algorítmo é mostrado na Figura 7.1 que é a saída de um arquivo de *excel* onde podemos notar duas rotinas distintas que são dadas pelos botões "Gera Ptos" e "Otimização" que são reponsáveis por disparar respectivamente as rotinas onde serão geradas todas as partículas e depois o algorítmo PSO propriamente dito. Nesta mesma visualização podemos notar todas as coordenadas de cada partícula bem como o valor da função objetivo e as coordenadas utilizadas para a melhor solução global até o momento. Além disso podemos também visualizar todas as partículas no gráfico do lado esquerdo onde notamos que a partícula que representa a melhor solução global se encontra em vermelho. Neste exemplo proposto foi utilizado um decaimento linear no parâmetro w do equacionamento que representa o momento de inércia das partículas para que pudessemos ter uma convergência mais rápida. Além disso foram feitas 10 itera-

ções, sendo que cada uma é composta pela atualização as coordenadas de cada partícula de acordo com as equações propostas e os resultados são analisados e mostrados para o usuário na interface utilizada.

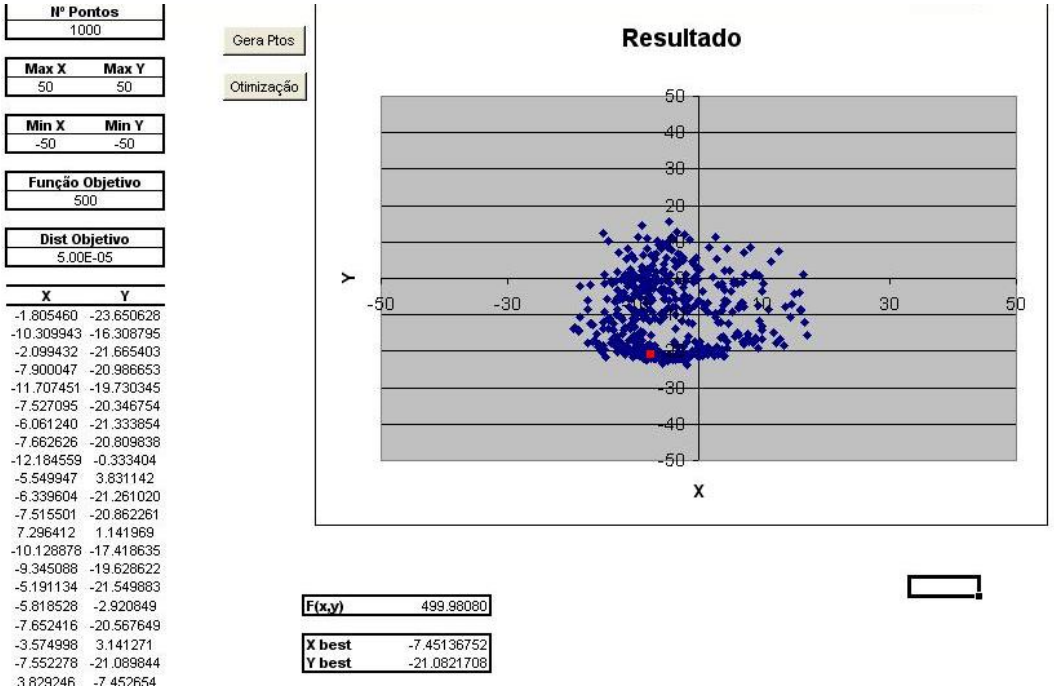


Figura 4.1: Interface do programa utilizado.

Para podermos testar o programa desenvolvido o mesmo foi testado para uma função objetivo dada por:

$$F(x, y) = x^2 + x^2 \tag{4.3}$$

Na Figura 4.2 podemos ver todas as soluções geradas aleatóriamente. E depois nas Figuras (4.2/4.3/4.4/4.5) o desenvolvimento do método onde podemos notar a aleatoriedade da busca por nova soluções somadas a tendência das partículas seguirem a melhor encontrada até o momento. Sendo que ao final do processo podemos notar, no menu mostrado, pela Figura 4.6 os resultados da melhor partícula encontrada onde temos as coordenadas utilizadas e a distância com relação à função objetivo procurada.

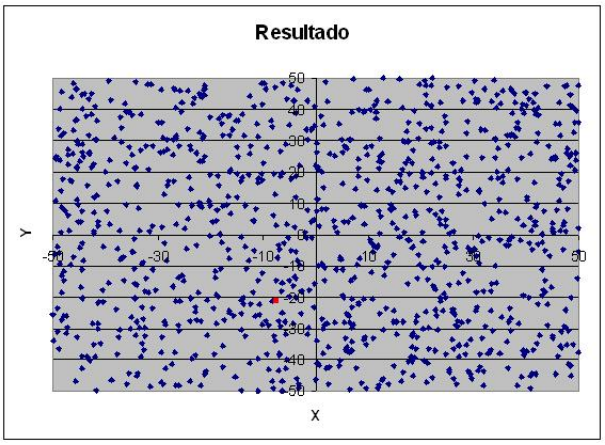


Figura 4.2: Ilustração da convergência do algorítmo (parte I).

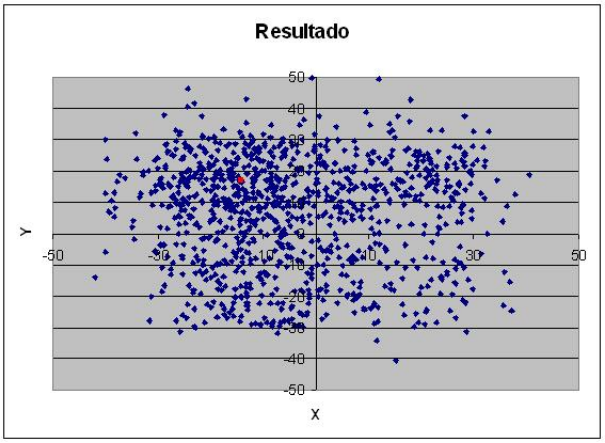


Figura 4.3: Ilustração da convergência do algorítmo (parte II).

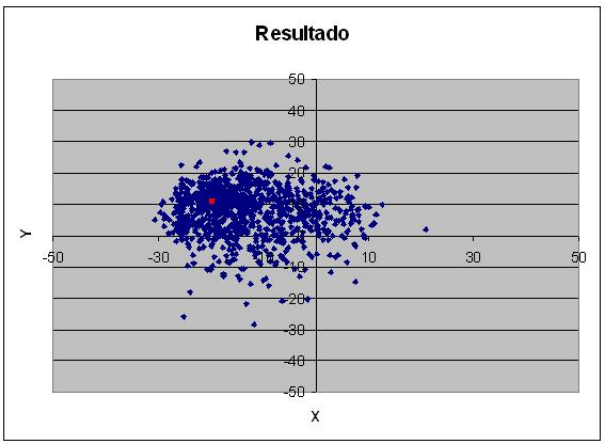


Figura 4.4: Ilustração da convergência do algorítmo (parte III).

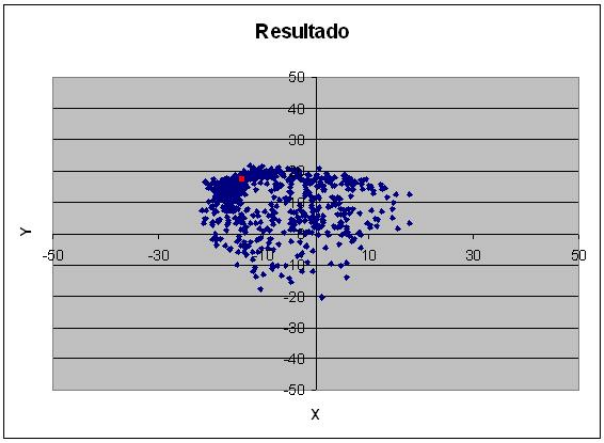


Figura 4.5: Ilustração da convergência do algorítmo (parte IV).

Figura 4.6: Resultados.

5 Resultados

5.1 Estrutura do Programa

Como resultado deste trabalho temos todos os estudos que foram feitos para que fosse possível total entendimento do problema a ser solucionado para isso foi necessário a total compreensão do problema de posicionamento já tratado anteriormente. Outro ponto muito importante foi o estudo das alternativas de solução para a otimização do posicionamento de polígonos dentre eles: *bottom-left*, *larger first* e recozimento simulado. Além disso, foi desenvolvido um programa mostrado no capítulo anterior para que tivéssemos um melhor entendimento do enxame de partículas que é o tema abordado nesta monografia. O objetivo final foi a criação de um software capaz de colocar em prática todos os temas abordados para que fosse possível chegarmos a visualização das soluções reais. Este programa foi desenvolvido através da utilização de diversas bibliotecas de funções e classes em função destas atenderem aos objetivos procurados e em função do tempo disponível para criação da solução não ser suficiente para desenvolver todo o tipo de código utilizado. O programa pode ser divido em três partes: a primeira concentra todas as funções responsáveis pela manipulação gráfica existente no programa; a segunda que representa todas as ferramentas de manipulações de dados e alocação de memória no computador; e uma terceira que representa a lógica presente no método utilizado através de todo o suporte feito pelo resto do programa. A idéia do programa é poder ser flexível para solucionar o maior numero de problemas possíveis, com isso existem diversos parâmetros que são fornecidos ao programa através de arquivos de texto que podem ser alterados a cada execução para que assim possa satisfazer a vontade do usuário para personalizar o problema proposto. Existem dois arquivos responsáveis pelo armazenamento dos parâmetros que podem ser visualizados em anexo a este trabalho: *puzzle.txt*

e *parameters.txt*. O arquivo *puzzle.txt* armazena todos os polígonos através das coordenadas dos vértices de cada um que é inserida neste arquivo, além também de armazenar a forma e tamanho do recipiente utilizado no problema. Já o outro arquivo chamado de *parameters.txt* apresenta os seguintes parâmetros:

- *vmax*
- *nParticulas*
- *niter*
- *Translational*
- *LargerFirst*
- *BottomLeft*

vmax representa a velocidade máxima que cada partícula presente no enxame poderá ter ao longo de todas as iterações. *nParticulas* armazena o número de partículas que serão utilizadas durante a busca pelas soluções. *niter* é o parâmetro responsável pelo critério de parada de execução do programada dado que representa o número de iterações que serão executadas. Já os próximos três parâmetros presentes neste arquivo tem o objetivo de determinar como serão posicionados os polígonos depois de determinado todos os parâmetros presentes em cada solução. *Translational* é a escolha se o método ira apenas levar em consideração a translação de cada polígono ou se a busca por novas soluções também irá alterar as rotações de cada um. Este parâmetro tem uma grande importância na velocidade de execução do programa dado que quando estamos buscando as rotações de cada polígono o problema se torna mais complexo e com um custo computacional mais elevado. O parâmetro *LargerFirst* é responsável por determinar se este método já descrito anteriormente será utilzado. Mesmo fato este acontece para o parâmetro *BottomLeft*. A estrutura básica de execução do programa pode ser observada na Figura 5.1. A primeira etapa é a leitura de todos os parâmetros de entrada descritos acima. O Segundo passo é a inicialização de cada solução, ou seja, se cada partícula, definindo assim, a posição e velocidade de cada uma. Após esta etapa o programa entra no processo de execução das iterações que começa com a avaliação das soluções que pode ser entendido como o ponto chave de ligação do método com todas as funções de tratamento gráfico e de dados presentes no programa. Isso em função desta etapa ser responsável pela tradução de diversos parâmetros que compõe uma solução para a representação gráfica desta e avaliação da área desperdiçada pela mesma. Depois da avaliação de todas as partículas atuais temos a atualização da melhor posição global do enxame até o momento além da melhor posição individual de cada partícula para

que possa ser utilizado nas próximas etapas que podem ser representadas pelas Equações 4.1 e 4.2. Com isso o programa atualiza o valor da velocidade e depois da posição de cada partícula e com isso o programa termina uma iteração que será repetida um número *niter* de vezes.

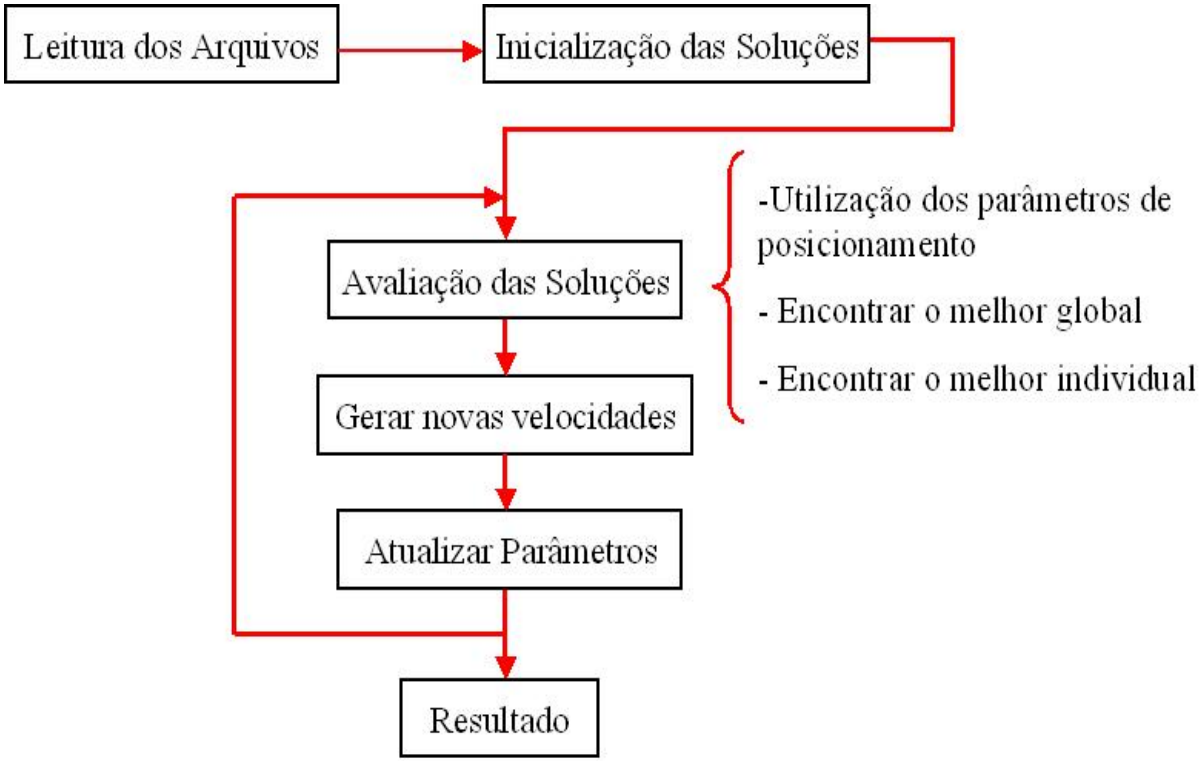


Figura 5.1: Estrutura do programa desenvolvido.

5.2 Testes

Para testar a eficiência do programa foram propostos alguns exemplos simples para notarmos a influência de cada parâmetro na obtenção da solução final para cada tipo de problema. Utilizando o parâmetro *Bottom left* existente no programa foi observado que existem alguns tipos de problemas simples em que não é possível obtermos uma boa solução usando este método como pode ser observado na Figura 5.2 que ilustra que em função de termos diversos polígonos com a mesma forma, tamanho e não estarmos utilizando o parâmetro de rotação, o arranjo obtido não é muito eficiente.

Nota-se que neste caso o componente rotacional era importante em função de quando utilizamos este parâmetro foi obtida uma solução onde podemos notar que para um mesmo espaço o número de polígonos posicionados foi bem maior, como pode ser observado na Figura 5.3 que mostra apenas uma configuração

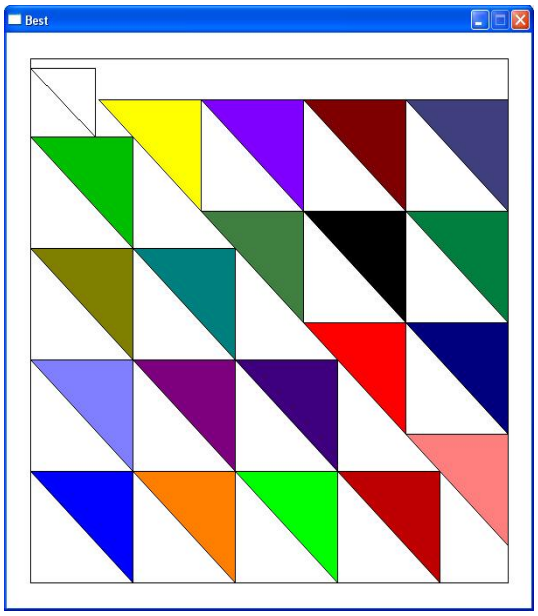


Figura 5.2: Problema cuja solução ótima não pode ser alcançada pelo método *Bottom left*.

intermediaria deste problema tratado de forma rotacional onde já temos uma melhor solução antes mesmo do final da execução do programa.

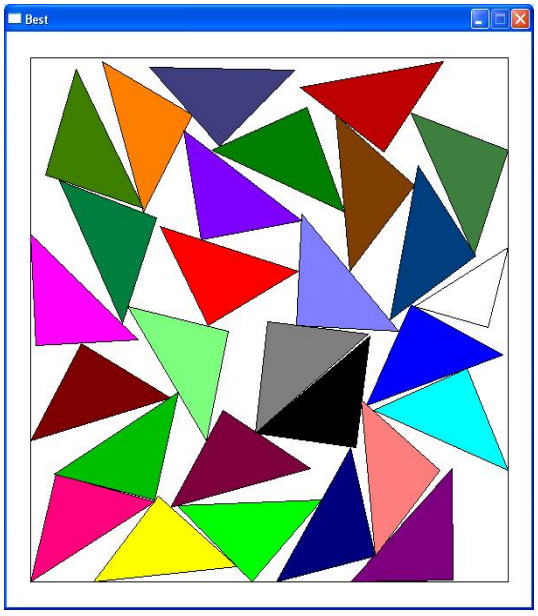


Figura 5.3: Arranjo rotacional mais eficiente que o utlizado pelo método *Bottom left*.

Outro problema proposto foi elaborado para mostrar a influência do parâmetro responsável pela utilização do método *larger first* onde a idéia é posicionar seqüencialmente os polígonos do maior para os menores. Este método nem sempre é eficiente como pode ser observado na Figura 5.4 onde o arranjo obtido não é o ótimo que era esperado. Quando deixamos de utilizar este método pode-se notar o arranjo ótimo mostrado na Figura 5.5 que ocupa todo o espaço do recipiente.

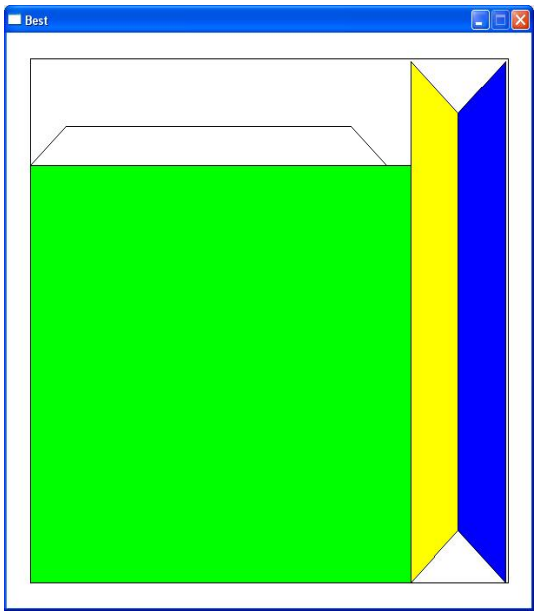


Figura 5.4: Problema cuja solução ótima não pode ser alcançada pelo método *Larger First*.

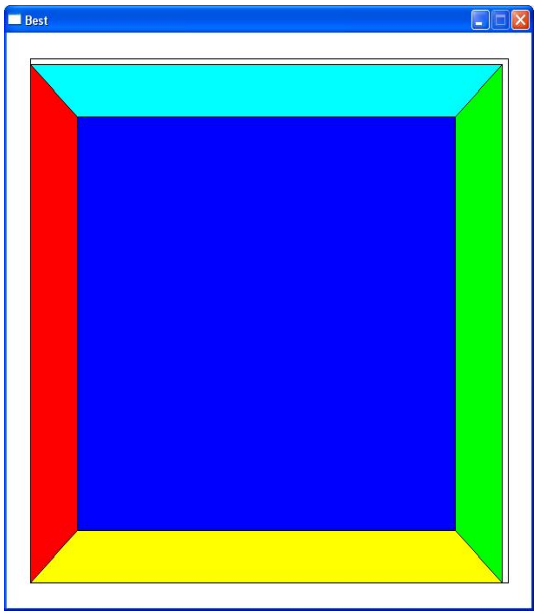


Figura 5.5: Solução ótima para o problema proposto sem a utilização do método *Larger First*.

O programa também possui a opção de se incluir obstáculos nos recipientes utilizados simulando um problema real onde a área disponível para posicionamento das formas não é contínua. Este tipo de problema pode ser observado na Figura 5.6 onde podemos observar o arranjo obtido com o obstáculo central proposto.

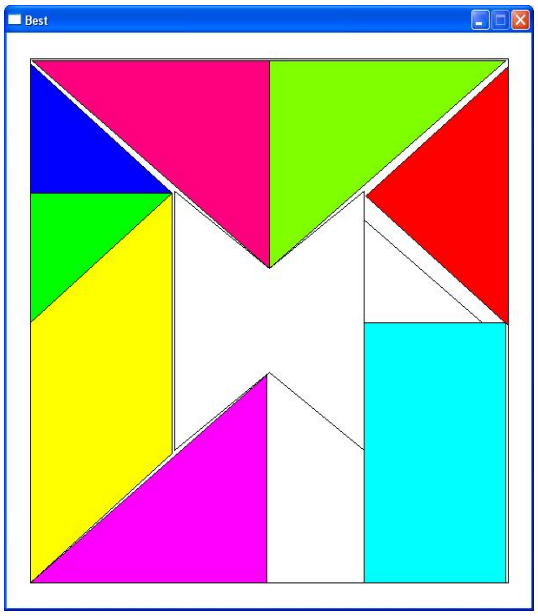


Figura 5.6: Solução para o problema com obstáculo.

O exemplo mostrado na Figura 5.7 mostra a solução ótima para o problema onde o recipiente é do tipo não-convexo sendo que esta solução foi obtida utilizando apenas o posicionamento translacional, desconsiderando o parâmetro de rotação.

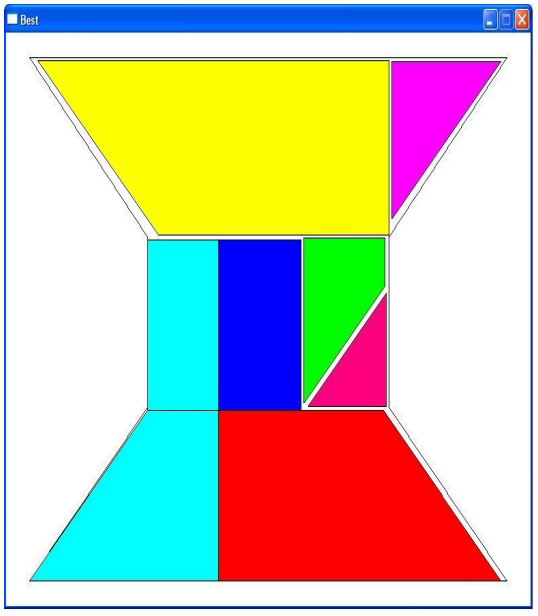


Figura 5.7: Solução para o problema com recipiente não-convexo.

Embora o algoritmo tenha se tornado eficiente para solução de diversos tipos de problemas utilizando diversos tipos de métodos de posicionamento existem alguns problemas que onde não foi possível encontrar uma solução ótima. Esse fato pode ser observado na Figura 5.8, embora a característica do problema proposto seja a mesma do problema da Figura 5.9 onde foi possível encontrar uma solução melhor.

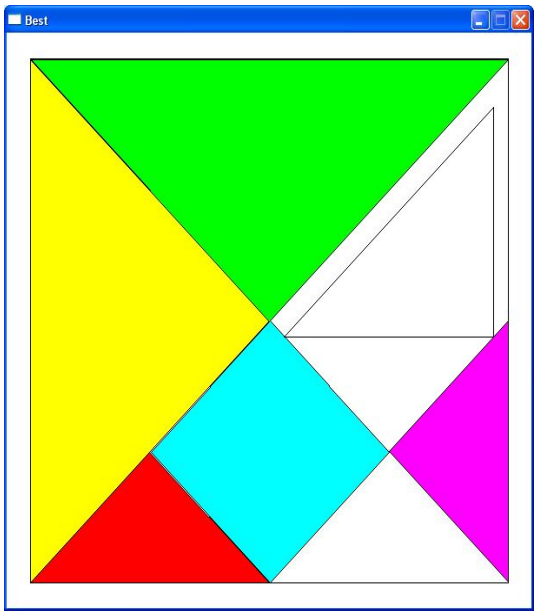


Figura 5.8: Problema cuja solução ótima não pode ser alcançada pelo método proposto.

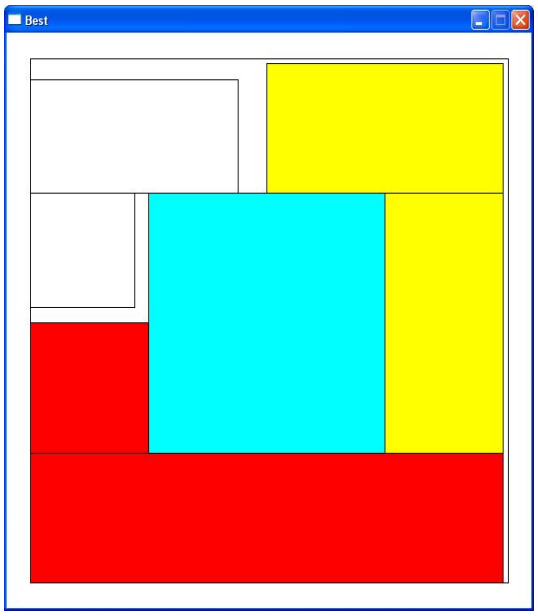


Figura 5.9: Solução para um problema genérico.

6 Conclusões

O método proposto para solucionar o problema de posicionamento tem se mostrado muito eficiente para solução de problemas do tipo NP-completo. Este fato pode ser notado na aplicação desenvolvida para melhor entendimento do PSO (*Particle Swarm Optimization*) que motiva futuros estudos sobre a integração deste método com outros.

Os teste realizados comprovaram a eficiência tanto do método de enxame de partículas proposto como o programa utilizado para obtenção do posicionamento dos polígonos. No entanto, embora o programa seja eficiente, a convergência para problemas onde seja necessário a utilização dos parâmetros de rotação é bastante demorada em função da complexidade adicional criada na solução.

De acordo com o que foi mostrado podemos dizer que foi alcançado o objetivo de se estudar a influência dos parâmetros utilizados na obtenção dos resultados, sendo que este tipo de teste motivará uma continuação no estudo do problema proposto para casos mais específicos integrados com outros métodos.

Referências

1 Adamowicz, M., Albano, A., Nesting two dimensional shapes in rectangular modules. Computer Aided Design, Vol. 8, pp. 27-33, 1976.

2 Castro, E. G., Tsuzuki, M. S. G., Swarm intelligence applied in synthesis of hunting strategies in a three-dimensional environment. Expert Systems with Applications, Vol. 34, pp.1995-2003, 2008.

3 Clerc, M., Particle Swarm Optimization, ISTE, ISBN 1-905209-04-5, 2006.

4 Coley, D., An Introduction to Genetic Algorithms for Scientists and Engineers. New Jersey, World Scientific Publishing Company, 1997. 227p.

5 Darwin, C., The Origin of Species: By Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life. New York: Bantam Classics, 1999. 512p.

6 Dowsland, K. A., Some experiments with simulated annealing techniques for paccking problems. European Journal of Operational Research, Vol. 68, pp. 389-399, 1993.

7 Garey, M. R., Johnson, D. S., Computers and Intractability - A Guide to the Theory of NP-Completeness. San Francisco, EUA Freeman, 1979.

8 Gomes, A. M., Oliveira, J. F., A 2-exchange heuristic for nesting problems. European Journal of Operational Research, Vol. 141, pp. 359-370, 2002.

9 Kennedy, J., Eberhart, R., Particle swarm optimization, in Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942-1948, 1995

10 Martins, T. C., Estudo do Recozimento Simulado e do Polígono de Obstrução Aplicados ao Problema de Empacotamento Rotacional de Polígonos Irregulares Não-Convexos em Recipientes Fechados. Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2007. 103p.

11 M'Hallah, R., Bouziri, A., Jilani, W., Layout of two dimensional irregular shapes using genetic algorithms. Lecture Notes In Computer Science, Vol. 2070, pp. 403-411, 2001.

12 Parsopoulos, K.E., Vrahatis, M. N., Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization, Natural Computing, Vol. 1, pp. 235-306, 2002.

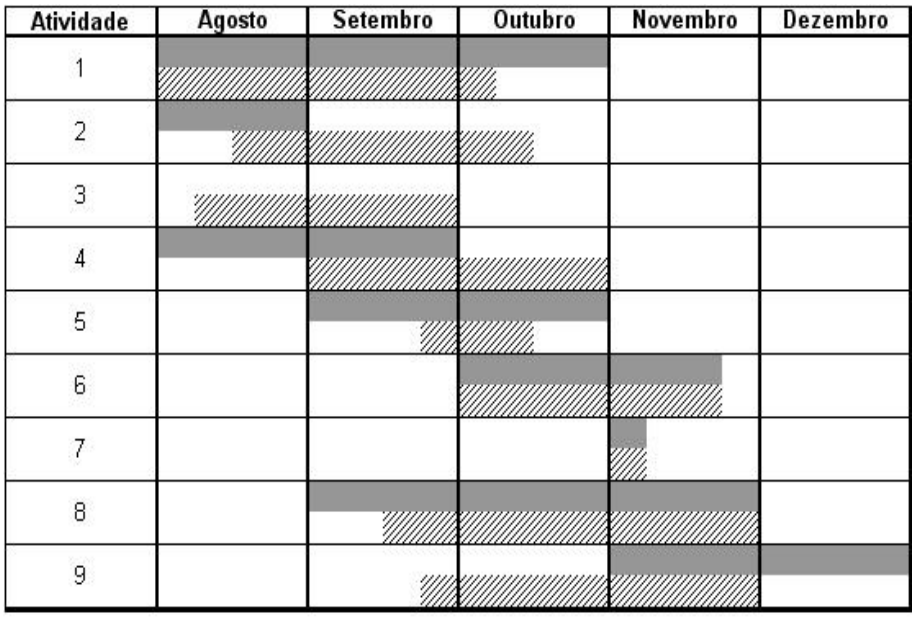
13 Vanderplaats, G. N., Numerical Optimization Techniques for Engineering Design: With Applications. New York: Mcgraw-Hill, 198.,

14 Wascher, G., Haussner, H., Schumann, H., An improved typology of cutting and packing problems. Faculty of Economics and Management Magdeburg - Working Paper Series, 2005.

Apêndice A – Trabalhos apresentados durante desenvolvimento desta monografia

◦ DELAGO, Daniel Zanin; SANCHEZ, Renato. Posicionamento rotacional de polígonos em recipientes fechados utilizando enxame de partículas. In: 16^o SIMPÓSIO INTERNACIONAL DE INICIAÇÃO CIENTÍFICA DA USP, 2008, São Paulo.

Apêndice B – Cronograma desenvolvido



Descrição das Atividades
1 - Esboço sobre como representar o problema de posicionamento com enxame de partículas
2 - Implementação do Método Enxame de Partículas em C++
3 - Utilização do VBA para ilustração do método utilizado
4 - Integração com o Pacote PolygonPacking
5 - Confecção de Artigo para o SIICUSP 2008
6 - Inclusão de Heurísticas Determinísticas (Larger First, Bottom Left, Translacional)
7 - Apresentação do Trabalho no SIICUSP 2008
8 - Testes e Resultados
9 - Confecção do relatório final

Figura B.1: Cronograma.

Apêndice C – Atividades Futuras

Após o desenvolvimento deste trabalho foram geradas novas idéias para serem propostas como próximos passos. Uma proposta seria um melhor estudo do desempenho da alteração de cada parâmetro utilizado no programa para que assim possam ser estabelecidas melhorias tanto para o método quanto para a lógica existente no programa. Além do método do enxame de partículas, existem outros que se mostraram eficientes também para a solução deste tipo de problema, com isso é natural de se pensar em desenvolver um método que possa englobar a lógica presente em diversos métodos para que assim seja criado talvez algum outro método que se mostre mais eficiente que cada um operando sozinho. Outra proposta seria a utilização e adaptação do enxame de partículas para solução de outros problemas deste mesmo tipo.

Apêndice D – Código-fonte do programa de estudo do método PSO

```
Option Explicit
Option Base 1
Function f_xy(x As Double, y As Double) As Double

    f_xy = x ^ 2 + y ^ 2

End Function
Function min_prox_f(x() As Double, y() As Double, tam As Long, f As Double) As Double
Dim min As Double
Dim i As Long

min = 5000000
For i = 1 To tam
    If Abs(f_xy(x(i), y(i)) - f) < min Then
        min = Abs(f_xy(x(i), y(i)) - f)
    End If
Next i

min_prox_f = min
End Function
Function min_prox_f_pto(x() As Double, y() As Double, tam As Long, f As Double) As Long
Dim min As Double
Dim i As Long
Dim x_pto As Long

min = 5000000
For i = 1 To tam
    If Abs(f_xy(x(i), y(i)) - f) < min Then
        min = Abs(f_xy(x(i), y(i)) - f)
        x_pto = i
    End If
Next i

min_prox_f_pto = x_pto
End Function
Function dist_f(xi As Double, yi As Double, f As Double) As Double

    dist_f = Abs(f_xy(xi, yi) - f)

End Function
Function dist(xi As Double, yi As Double, a As Double, b As Double) As Double

    dist = ((xi - a) ^ 2 + (yi - b) ^ 2) ^ (1 / 2)

End Function
Function min_dist(x() As Double, y() As Double, x_final As Double,
y_final As Double, tam As Long) As Double
Dim min As Double
Dim i As Long

min = 5000000
For i = 1 To tam
    If dist(x(i), y(i), x_final, y_final) < min Then
        min = dist(x(i), y(i), x_final, y_final)
```

```
    End If
Next i

min_dist = min
End Function
Function min_dist_pto(x() As Double, y() As Double, x_final As Double,
y_final As Double, tam As Long) As Long
Dim min As Double
Dim x_pto As Long
Dim i As Long

min = 5000000
For i = 1 To tam
    If dist(x(i), y(i), x_final, y_final) < min Then
        x_pto = i
        min = dist(x(i), y(i), x_final, y_final)
    End If
Next i

min_dist_pto = x_pto
End Function
Function sum_dist(x() As Double, y() As Double, x_final As Double,
y_final As Double, tam As Long) As Double
Dim soma As Double
Dim i As Long

soma = 0
For i = 1 To tam
    soma = soma + dist(x(i), y(i), x_final, y_final)
Next i

sum_dist = soma
End Function
Sub gera_coordenadas_aleatorias()
Dim linha_inicio As Long, col_x As Long, col_y As Long
Dim n_ptos As Long, x_max As Long, y_max As Long
Dim sht As Excel.Worksheet

    Set sht = ActiveWorkbook.ActiveSheet

    n_ptos = sht.Range("n_pontos").Value
    x_max = sht.Range("max_x").Value
    y_max = sht.Range("max_y").Value
    linha_inicio = sht.Range("x_inicio").Row
    col_x = sht.Range("x_inicio").Column
    col_y = sht.Range("y_inicio").Column

    sht.Range(Cells(linha_inicio, col_x), Cells(65536, col_x)).ClearContents
    sht.Range(Cells(linha_inicio, col_x), Cells(linha_inicio +
    n_ptos - 1, col_x)).FormulaR1C1 = "=RAND()*" & x_max
    sht.Range(Cells(linha_inicio, col_x), Cells(linha_inicio +
    n_ptos - 1, col_x)).Calculate
    sht.Range(Cells(linha_inicio, col_x), Cells(linha_inicio +
    n_ptos - 1, col_x)).Copy
    sht.Cells(linha_inicio, col_x).PasteSpecial Paste:=xlPasteValues,
    Operation:=xlNone, SkipBlanks _
            :=False, Transpose:=False
    Application.CutCopyMode = False

    sht.Range(Cells(linha_inicio, col_y), Cells(65536, col_y)).ClearContents
    sht.Range(Cells(linha_inicio, col_y), Cells(linha_inicio +
    n_ptos - 1, col_y)).FormulaR1C1 = "=RAND()*" & y_max
    sht.Range(Cells(linha_inicio, col_y), Cells(linha_inicio +
    n_ptos - 1, col_y)).Calculate
    sht.Range(Cells(linha_inicio, col_y), Cells(linha_inicio +
    n_ptos - 1, col_y)).Copy
    sht.Cells(linha_inicio, col_y).PasteSpecial Paste:=xlPasteValues,
    Operation:=xlNone, SkipBlanks _
            :=False, Transpose:=False
    Application.CutCopyMode = False

    sht.ChartObjects("Chart 2").Activate
    ActiveChart.Axes(xlValue).Select
    With ActiveChart.Axes(xlValue)
        .MinimumScale = 0
        .MaximumScale = y_max
    End With
```

```
    ActiveChart.Axes(xlCategory).Select
    With ActiveChart.Axes(xlCategory)
        .MinimumScale = 0
        .MaximumScale = x_max
    End With

    ActiveChart.SeriesCollection(1).XValues = "=" & sht.Name &
    "!R" & linha_inicio & "C" & col_x & ":R" & linha_inicio + n_ptos - 1 & "C" & col_x
    ActiveChart.SeriesCollection(1).Values = "=" & sht.Name &
    "!R" & linha_inicio & "C" & col_y & ":R" & linha_inicio + n_ptos - 1 & "C" & col_y

    Set sht = Nothing

End Sub
Sub gera_coordenadas_aleatorias_f()
Dim linha_inicio As Long, col_x As Long, col_y As Long, i As Long
Dim n_ptos As Long, x_max As Long, y_max As Long, x_min As Long, y_min As Long
Dim sht As Excel.Worksheet

    Set sht = ActiveWorkbook.ActiveSheet

    n_ptos = sht.Range("n_pontos_f").Value
    x_max = sht.Range("max_x_f").Value
    y_max = sht.Range("max_y_f").Value
    x_min = sht.Range("min_x_f").Value
    y_min = sht.Range("min_y_f").Value
    linha_inicio = sht.Range("x_inicio_f").Row
    col_x = sht.Range("x_inicio_f").Column
    col_y = sht.Range("y_inicio_f").Column

    sht.Range(Cells(linha_inicio, col_x), Cells(65536, col_x)).ClearContents
    For i = linha_inicio To linha_inicio + n_ptos - 1
        sht.Cells(i, col_x) = Rnd() * (x_max - x_min) + x_min
    Next i

    sht.Range(Cells(linha_inicio, col_y), Cells(65536, col_y)).ClearContents
    For i = linha_inicio To linha_inicio + n_ptos - 1
        sht.Cells(i, col_y) = Rnd() * (y_max - y_min) + y_min
    Next i

    sht.ChartObjects("Chart 2").Activate
    ActiveChart.Axes(xlValue).Select
    With ActiveChart.Axes(xlValue)
        .MinimumScale = y_min
        .MaximumScale = y_max
    End With
    ActiveChart.Axes(xlCategory).Select
    With ActiveChart.Axes(xlCategory)
        .MinimumScale = x_min
        .MaximumScale = x_max
    End With

    ActiveChart.SeriesCollection(1).XValues = "=" & sht.Name &
    "!R" & linha_inicio & "C" & col_x & ":R" & linha_inicio + n_ptos - 1 & "C" & col_x
    ActiveChart.SeriesCollection(1).Values = "=" & sht.Name &
    "!R" & linha_inicio & "C" & col_y & ":R" & linha_inicio + n_ptos - 1 & "C" & col_y

    Set sht = Nothing

End Sub
Sub otimizacao_f_vel()

Dim best_x As Double, best_y As Double, objetivo_f As Double
Dim x() As Double, y() As Double, x_best() As Double, y_best() As Double
Dim dist_total As Double
Dim tam As Long, i As Long
Dim linha_inicio As Long, col_x As Long, col_y As Long, n_ptos As Long
Dim sht As Excel.Worksheet
Dim x_max As Long, y_max As Long, x_min As Long, y_min As Long
Dim resp, teste
Dim vel_x() As Double, vel_y() As Double
Dim cont As Long


Set sht = ActiveWorkbook.ActiveSheet

Const vmax As Double = 10
```

```
tam = sht.Range("n_pontos_f").Value
objetivo_f = sht.Range("objetivo_f").Value
dist_total = sht.Range("dist_total_f").Value
x_max = sht.Range("max_x_f").Value
y_max = sht.Range("max_y_f").Value
x_min = sht.Range("min_x_f").Value
y_min = sht.Range("min_y_f").Value

ReDim x(tam) As Double
ReDim y(tam) As Double
ReDim vel_x(tam) As Double
ReDim vel_y(tam) As Double
ReDim x_best(tam) As Double
ReDim y_best(tam) As Double

n_ptos = sht.Range("n_pontos_f").Value
linha_inicio = sht.Range("x_inicio_f").Row
col_x = sht.Range("x_inicio_f").Column
col_y = sht.Range("y_inicio_f").Column

    For i = 1 To tam
        x(i) = sht.Cells(linha_inicio + i - 1, col_x).Value
        x_best(i) = sht.Cells(linha_inicio + i - 1, col_x).Value
        vel_x(i) = Rnd() * 2 * vmax - vmax
        y(i) = sht.Cells(linha_inicio + i - 1, col_y).Value
        y_best(i) = sht.Cells(linha_inicio + i - 1, col_y).Value
        vel_y(i) = Rnd() * 2 * vmax - vmax
    Next i

    cont = 0
    best_x = 100000
    best_y = 100000

    While cont < 10
    'While min_prox_f(x, y, tam, objetivo_f) > dist_total
        'verifica a melhor solucao ate o momento
        For i = 1 To tam
            If dist_f(best_x, best_y, objetivo_f) > dist_f(x(i), y(i), objetivo_f) Then
                best_x = x(i)
                best_y = y(i)
            End If
        Next i

        'executa a atualizacao das coordenadas dos pontos
        For i = 1 To tam
            'gera coordenadas apenas em funcao da melhor
            posicao da particula e melhor global
            vel_x(i) = vel_x(i) * (10 - cont) / 10 + 1 * Rnd() *
            (best_x - x(i)) + 1 * Rnd() * (x_best(i) - x(i))
            If vel_x(i) > vmax Then
                x(i) = x(i) + vmax
                vel_x(i) = vmax
            Else
                If vel_x(i) < -vmax Then
                    x(i) = x(i) - vmax
                    vel_x(i) = -vmax
                Else
                    x(i) = x(i) + vel_x(i)
                End If
            End If

            vel_y(i) = vel_y(i) * (10 - cont) / 10 + 1 * Rnd() *
            (best_y - y(i)) + 1 * Rnd() * (y_best(i) - y(i))
            If vel_y(i) > vmax Then
                y(i) = y(i) + vmax
                vel_y(i) = vmax
            Else
                If vel_y(i) < -vmax Then
                    y(i) = y(i) - vmax
                    vel_y(i) = -vmax
                Else
                    y(i) = y(i) + vel_y(i)
                End If
            End If
```

```
            If dist_f(x_best(i), y_best(i), objetivo_f) > dist_f(x(i),
            y(i), objetivo_f) Then
                x_best(i) = x(i)
                y_best(i) = y(i)
            End If
        Next i

        'atualiza dados para o grafico
        For i = 1 To tam
            sht.Cells(linha_inicio + i - 1, col_x).Value = x(i)
            sht.Cells(linha_inicio + i - 1, col_y).Value = y(i)
        Next i
        Calculate

        sht.Range("distancia").Value = f_xy(best_x, best_y)
        sht.Range("best_x").Value = best_x
        sht.Range("best_y").Value = best_y

        cont = cont + 1

        If sht.Range("parada_f") = "PARADA" Then
            MsgBox "PARADA"
        End If

    Wend

    'mostra resultados finais
    i = min_prox_f_pto(x, y, tam, objetivo_f)
    resp = MsgBox("Processo Finalizado !!" & Chr(13) & Chr(13) &
    "Distância : " & Format(dist_f(x(i), y(i), objetivo_f),
    "0.000000") & Chr(13) & Chr(13) & "Ponto escolhido" & Chr(13) &
    "X : " & x(i) & Chr(13) & "Y : " & y(i), vbExclamation, "Swarm Particle")

Set sht = Nothing

End Sub

}
```

Apêndice E – Código-fonte de parte do programa utlizado

```
//SwarmParticle.cpp : Defines the entry point for the console application.

#include <stdio.h>

#include "cObstructedRegionsCache.h"
#include "cSmartPointer.h"
#include "cShape.h"
#include "cObstacleList.h"
#include "cRectangularContainer.h"
#include "cShapeInlines.h"
#include "cShapeSetInlines.h"
#include "cBasicViewerPolygonTracer.h"
#include "cAnnealingPackingSolution.h"
#include "cAnnealingPackingSolutionInlines.h"
#include "problemInstance.h"
#include "cAnnealingLogger.h"
#include "mersennetwister/mt19937ar.h"

#include <fstream>
#include <iostream>
#include <string>
#include <strstream>

#include <crtdbg.h>
#include <time.h>

#define MAX_ARRAY_LENGTH 1000

using namespace std;

template<class C> bool getParam(std::ifstream *file, std::string param, C *val) {
char buff[256];
char separator;
bool ret = false;
file->seekg(0);
while(file->good()) {
file->getline(buff, sizeof(buff));
std::strstream parser(buff, sizeof(buff));
std::string aux;
parser >> aux;
if(aux==param) {
parser >> separator;
if(separator == '=') {
parser >> *val;
ret = true;
break;
}
}
}

return ret;
}

cTypedSmartPointer<cAnnealingPackingSolution> sol;
```

```
cTypedSmartPointer<cAnnealingPackingSolution> bestSolution;
double bestValue;
cBasicViewerPolygonTracer *viewer;
cBasicViewerPolygonTracer *bestViewer;

unsigned long iterations;

double evaluatePackingSolution(double *parameters, int numItems)
{
    double result;
    int i;

// --> set rotations
    for (i = 0 ; i < numItems ; i ++) {
        if (sol->parameters.shapes[i]->getParameters()->rot() != parameters[i])
            sol->parameters.shapes[i]->setRot(parameters[i]);
    }

// --> set translations
    for (i = 0 ; i < numItems ; i ++) {
        double f, t;
        f = 0;
        t = parameters[i];
        const cTypedSmartPointer<cShapeParameters> &param =
        sol->parameters.shapes[i]->getParameters();
if (param->t() != t || param->f() != f) {
            sol->parameters.shapes[i]->setFandT(f, t);
        }
    }
    result = sol->getValue();

if (result<bestValue){
        bestValue = result;
        bestSolution = sol;
        bestViewer->clearView();
        bestSolution->trace(bestViewer);
    }
    if (iterations ++ % 30 == 0) {
        printf("%d %f\n", iterations, result);
        viewer->clearView();
        sol->trace(viewer);
    }
    return result;
}

int best_particle(double *a, int n_particulas, int n_parametros)
{
double f_obj = evaluatePackingSolution(a, n_parametros);
int best = 0;
for(int i = 1 ; i < n_particulas ; i ++) {
a += n_parametros;
double f1 = evaluatePackingSolution(a, n_parametros);
if (f1 < f_obj) {
f_obj = f1;
best = i;
}
}
return best;
}

#ifndef SMALL_FLOAT
#define SMALL_FLOAT 1.0E-18
#endif

#ifndef EPS_DOUBLE
#define EPS_DOUBLE ((double) SMALL_FLOAT)
#endif

double getTime (clock_t time)
{
  static clock_t previousTime = -1;
  clock_t diffTime;
  double clocksPerSecF = CLOCKS_PER_SEC;
  double diffF;
  double ds;

  if (previousTime != -1) {
```

```
    diffTime = time - previousTime;
    diffF = diffTime;

    ds = diffF / clocksPerSecF;

return ds;
  } else {
    previousTime = time;
  }
  return 0;
}                               /* aux_print_time */

extern double obsArea;

int main(void)
{
// Initialise rng
init_genrand(GetTickCount());

// Read Parameters
cAnnealingPackingParameters params;

std::ifstream *paramFile = new std::ifstream("parameters.txt");
unsigned int scaleSteps;
if(!getParam(paramFile, "scaleSteps", &scaleSteps)) {
printf("Parameter error: expecting scale Steps\n"); return 2;
}

// PARAMETROS SERAO PASSADOS VIA ARQUIVO POSTERIORMENTE
//declaracao da velocidade maxima de cada particula
double v_max = 0.1;
if(!getParam(paramFile, "vmax", &v_max)) {
printf("Parameter error: expecting vmax\n"); return 2;
}

//declaracao do numero de particulas
int nParticulas = 500;
if(!getParam(paramFile, "nParticulas", &nParticulas)) {
printf("Parameter error: expecting nParticulas\n"); return 2;
}

//declaracao do numero de iteracoes
int iteracoes = 10;
if(!getParam(paramFile, "niter", &iteracoes)) {
printf("Parameter error: expecting numero de iteracoes\n"); return 2;
}

unsigned int auxInput;
if (!getParam(paramFile, "Translational", &auxInput)) {
printf("Parameter error: expecting Translational.\n"); return 2;
}
params.setTranslational(auxInput);
if (!getParam(paramFile, "LargerFirst", &auxInput)) {
printf("Parameter error: expecting LargerFirst.\n"); return 2;
}
params.setLargerFirst(auxInput);
if (!getParam(paramFile, "BottomLeft", &auxInput)) {
printf("Parameter error: expecting BottomLeft.\n"); return 2;
}
params.setBottomLeft(auxInput);
if (!getParam(paramFile, "RotationalIndex", &auxInput)) {
printf("Parameter error: expecting RotationalIndex.\n"); return 2;
}
params.setRotationalIndex(auxInput);
if (!getParam(paramFile, "ScaleIndex", &auxInput)) {
printf("Parameter error: expecting ScaleIndex.\n"); return 2;
}
params.setScaleIndex(auxInput);
if (!getParam(paramFile, "ScaleRotationalIndex", &auxInput)) {
printf("Parameter error: expecting ScaleRotationalIndex.\n"); return 2;
}
params.setScaleRotationalIndex(auxInput);
delete paramFile;

//declaracao da funcao objetivo procurada
const double value_objetivo = 500.0;
```

```
// Read problem instance
    FILE *f = fopen("puzzle.txt", "rt");
//fopen_s(&f, "puzzle.txt", "rt");
if(f==NULL) {
printf("Puzzle file not found.\n");
return 1;
}
cTypedSmartPointer<cAnnealingPackingSolution> solution =
readProblemInstance(f, &params);
fclose(f);

unsigned int bestIt = 0;

// A bit of a hack, works only for "puzzle" instances
double convergedValue = solution->obstList->getContainer()->getRefValue();
double containerArea = convergedValue;
for(unsigned int i = 0 ; i < solution->parameters.shapeCount ; i ++) {
convergedValue -= solution->parameters.shapes[i]->getValue();
}
printf("converge = %f\n", convergedValue);

cTypedSmartPointer<cAnnealingPackingSolution> auxSolution;

cTypedSmartPointer<cAnnealingPackingModifier> modifier;

solution->randomInit();
double v1 = solution->getValue(scaleSteps);
double bestValue = v1;

    getTime(clock ());

// --> Os parametros se referem apenas as posicoes de cada item,
cada particula possui o conjunto de posicoes
double *sol = (double *)malloc(sizeof(double) *
solution->parameters.shapeCount * nParticulas);

// --> Os parametros obtidos que definiram a melhor posicao para cada item
double *particula_best = (double *)malloc(sizeof(double) *
solution->parameters.shapeCount * nParticulas);

// --> Os parametros para a melhor particula
double *sol_best = (double *)malloc(sizeof(double) *
solution->parameters.shapeCount);

    // --> Velocidade associada a cada particula com os seus parametros
double *vel = (double *)malloc(sizeof(double) *
solution->parameters.shapeCount * nParticulas);

// --> inicializacao das velocidades
std::cout << "\n Velocidades \n";
for (unsigned int i = 0 ; i < solution->parameters.shapeCount *
nParticulas ; i ++) {
    vel[i] = genrand_real2() * v_max;
}

// --> inicializacao dos parametros
std::cout << "\n Parametros \n";
for (unsigned int i = 0 ; i < solution->parameters.shapeCount *
nParticulas ; i ++) {
        sol[i] = genrand_real2();
particula_best[i] = sol[i];
}

// --> atualiza valor da melhor particula encontrada
std::cout << "\n Best Particle \n";
int aux = best_particle(sol, nParticulas, solution->parameters.shapeCount);
std::cout << "\n Best Particle \n";
for (unsigned int j = 0 ; j < solution->parameters.shapeCount ; j ++) {
sol_best[j] = sol[aux * solution->parameters.shapeCount + j];
}
    bestValue = evaluatePackingSolution(sol + aux *
    solution->parameters.shapeCount, solution->parameters.shapeCount);

// --> executa as iteracoes
for (int cont = 1 ; cont <= iteracoes ; cont ++) {
for (int i = 0 ; i < nParticulas ; i ++) {
for (unsigned int j = 0 ; j < solution->parameters.shapeCount ; j ++) {
```

```
int index = i * solution->parameters.shapeCount + j;
vel[index] = vel[index] * (double)((iteracoes - cont) / iteracoes) +
         genrand_real1() * (sol_best[j] - sol[index]) +
 genrand_real1() * (particula_best[index] - sol[index]);
if ( -v_max < vel[index] && vel[index] < v_max ){
sol[index] = sol[index] + vel[index];
}
else{
if( vel[index] < 0 ){
sol[index] = sol[index] - v_max;
}
else{
sol[index] = sol[index] + v_max;
}
}
}
}

aux = best_particle(sol, nParticulas, solution->parameters.shapeCount);
double val = evaluatePackingSolution(sol + aux *
solution->parameters.shapeCount, solution->parameters.shapeCount);
if (val < bestValue) {
std::cout << "\nBest Particle at Iteration # " << cont;
for (unsigned int j = 0 ; j < solution->parameters.shapeCount ; j ++) {
sol_best[j] = sol[aux * solution->parameters.shapeCount + j];
}
}
}

   std::cout << "\n ITERACOES FINALIZADAS \n";
return 0;
}
```

Apêndice F – Exemplo de arquivo do tipo *puzzle.txt*

```
//Generated by SEParser

POLYGON
80,160
120,120
160,120
200,160
200,200
160,240
120,240
80,200

POLYGON
240,160
280,120
320,120
360,160
360,200
320,240
280,240
240,200

POLYGON
80,320
120,280
160,280
```

```
200,320
200,360
160,400
120,400
80,360

POLYGON
240,320
280,280
320,280
360,320
360,360
320,400
280,400
240,360

POLYGON
160,240
200,200
240,200
280,240
280,280
240,320
200,320
160,280

CONTAINER
80,120
360,120
360,405
80,405

}
```

G – Exemplo de arquivo do tipo *parameters.txt*

```
vmax = 0.1
nParticulas = 100
niter = 10
Translational = 1
LargerFirst = 1
BottomLeft = 0

}
```