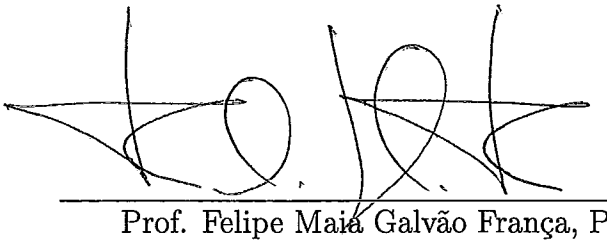
RESOLVENDO O PROBLEMA DE PLANEJAMENTO DA EXPANSÃO DA GERAÇÃO DE ENERGIA COM ENXAMES DE PARTÍCULAS BINÁRIOS

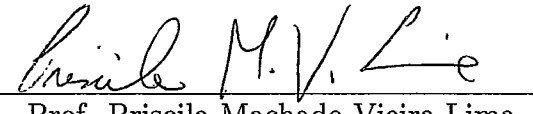
Ramon Diacovo

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

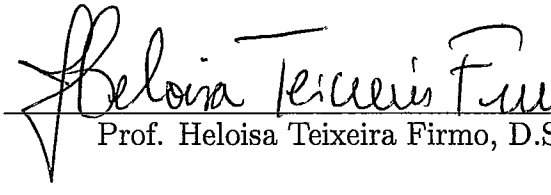
Aprovada por:



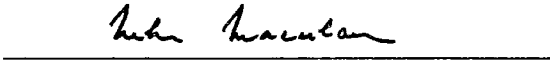
Prof. Felipe Maia Galvão França, Ph.D.



Prof. Priscila Machado Vieira Lima, Ph.D.



Prof. Heloisa Teixeira Firmo, D.Sc.



Prof. Nelson Maculan Filho, D.Habil.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2008

DIACOVO, RAMON

Resolvendo o Problema de Planejamento da Expansão da Geração de Energia com Enxames de Partículas Binários [Rio de Janeiro] 2008

XI, 118 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2008)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Enxames de partículas
2. GEP
3. SATyrus
4. Otimização combinatória

I. COPPE/UFRJ II. Título (série)

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RESOLVENDO O PROBLEMA DE PLANEJAMENTO DA EXPANSÃO DA GERAÇÃO DE ENERGIA COM ENXAMES DE PARTÍCULAS BINÁRIOS

Ramon Diacovo

Março/2008

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Programa: Engenharia de Sistemas e Computação

A importância da área de planejamento energético, particularmente o planejamento da expansão da geração de energia motiva o desenvolvimento de estudos sobre este problema. As opções para o enfoque são várias: para uma dada instância é possível buscar melhores soluções ou modelá-la de maneira a contemplar grandes quantidades de fatores relevantes. Outra possibilidade é a busca por novas ferramentas para trabalhar com o problema. O presente trabalho apresenta uma modelagem de uma instância do problema do planejamento da expansão da geração de energia utilizando a plataforma SATyrus. Esta plataforma consiste em um sintetizador de funções exatas, baseado nos princípios de minimização de energia e redes e Hopfield de alta ordem. Para a síntese das funções, o SATyrus recebe como entrada a descrição de um problema escrito em uma linguagem declarativa própria. Esta modelagem é, então, utilizada para realizar análises com o método de otimização por enxames de partículas binários (*binary Particle Swarm Optimization*). Este método é baseado no comportamento da movimentação de enxames, e toma proveito da interação entre os componentes dos mesmos para resolver problemas de otimização combinatória cujas dimensões são booleanas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SOLVING THE ENERGY GENERATION EXPANSION PLANNING PROBLEM WITH BINARY PARTICLE SWARMS

Ramon Diacovo

March/2008

Advisors: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Department: Systems Engineering and Computer Science

The importance of energy planning and the generation expansion planning, in particular, motivates the development of studies regarding this problem. There are plenty of options avaliable for focusing: for a given instance, it is possible to search for better solutions or to model it so that this instance contemplates large amounts of relevant factors. Another possibility is to find new tools to work with the problem. This work presents a modeling of a generation expansion problem instance, through the SATyrus platform. This platform consists of an exact function synthesizer, and it is based on the principles of energy minimization and high-order Hopfield networks. For the function synthesis, SATyrus takes as input the description of a problem, written on SATyrus' own declarative language. This modeling is then used to conduct an analysis of the binary Particle Swarm Optimization method, a paradigm based on bug swarms' behavior. It functions by taking advantage of the interaction between the swarm components in order to solve combinatorial optimization problems which have only boolean dimensions.

Sumário

Lista de Algoritmos

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

O planejamento energético é um setor de grande importância estratégica para o país, especialmente após o aumento da participação da iniciativa privada e da liberalização do setor. Esses fatores, embora permitam que as decisões sejam tomadas sob uma ótica não-centralizadora, aumentam a complexidade da modelagem do setor energético como um todo, afetando inclusive as abordagens matemáticas utilizadas.

Dentro deste campo, o planejamento da expansão da geração da energia (*generation expansion planning* ou GEP) é um dos problemas importantes, onde pesquisadores têm gasto recursos em busca de novos e eficientes métodos de resolução. Boas soluções para ele implicam em um melhor aproveitamento da verba utilizada na construção e operação de usinas produtoras de energia elétrica. Trata-se de um problema de otimização combinatória real — e difícil —, além de ser o primeiro alvo do projeto do Laboratório de Otimização Avançada (LOA) [4], que tem por objetivo explorar novas formulações para modelos de programação matemática orientados ao setor energético.

Por outro lado, outros problemas de difícil resolução motivaram a criação da plataforma SATyrus, que compila funções exatas, desenvolvida para auxiliar na modelagem e resolução de problemas de otimização combinatória. A plataforma emprega redes neurais sem peso para a tarefa da conversão em minimização de energia. O processo tem início a partir da modelagem do problema-alvo, escrita proposici-

onalmente na linguagem declarativa do SATyrus. A rede neural é então utilizada para gerar uma função de energia [13], a ser posteriormente minimizada pelo resolvedor. Observou-se que o SATyrus não contava com um resolvedor adequado, o que enquanto limita sua utilidade prática, motiva estudos em novos métodos de resolução.

1.2 Objetivos e contribuições

Este trabalho tem por objetivo contribuir para a diversificação das opções de ferramentas utilizáveis para a realização de estudos com o GEP. Apresentaremos uma primeira modelagem deste problema para a plataforma SATyrus. A partir dela, será mais simples modelar versões mais complexas do problema, que considerem outros fatores relevantes para o GEP.

Aproveitaremos a modelagem para realizar estudos com alguns resolvedores, mais especificamente com o *Particle Swarm Optimization* (PSO) binário. Verificaremos como este se comporta na tarefa de minimizar a função objetivo de GEP gerada pelo SATyrus, em ambientes de computação paralela. Ao mesmo tempo, realizaremos um extensivo conjunto de testes com nossa modelagem do GEP e o PSO binário, buscando agregar mais informações sobre este paradigma alternativo que, por ser menos difundido do que muitos outros, não conta com dados deste tipo suficientes.

Esperamos que a contribuição da nova modelagem do GEP seja útil para proporcionar alternativas para a resolução do problema. Como outra contribuição, o estudo realizado com os parâmetros do PSO binário proporcionará um maior nível de compreensão sobre seu comportamento.

1.3 Estrutura do texto

No Capítulo 2 apresentamos uma revisão bibliográfica sobre métodos de resolução de problemas de otimização combinatória relevantes para o presente trabalho. No Capítulo 3, o estudo de caso do trabalho, o GEP, é descrito, bem como algumas de suas possíveis formulações. Após a descrição do GEP temos, no Capítulo 4, uma

breve descrição da plataforma SATyrus, incluindo seu funcionamento e sua estrutura de arquivos de entrada.

A contribuição deste trabalho se inicia no Capítulo 5, onde é introduzida a modelagem do GEP para o SATyrus. O código-fonte do arquivo principal desta modelagem pode ser encontrado no Apêndice A. Seguindo, o Capítulo 6 contém a descrição, apresentação e análise dos experimentos realizados. Por questões de organização, os gráficos relativos a alguns dos experimentos não estão presentes no Capítulo 6, mas o leitor pode encontrá-los no Apêndice B. Finalmente, o Capítulo 7 contém as conclusões e comentários finais sobre o trabalho, assim como algumas propostas para trabalhos futuros.

Capítulo 2

Conhecimentos preliminares

Este capítulo apresenta alguns paradigmas e resolvedores para problemas de otimização combinatória. Todos foram estudados como possíveis abordagens para resolver o problema descrito no Capítulo 5, mas atenção especial foi dada ao resolvedor da Seção 2.4, pois há pouco estudo sobre seu emprego na situação apresentada.

2.1 Simulação Metropolis Monte Carlo

Esta seção apresenta conceitos que serão necessários para a compreensão do algoritmo da Seção 2.2, embora não descreva um método de resolução de problemas de otimização combinatória por si própria.

A simulação Metropolis Monte Carlo [23] utiliza movimentos aleatórios (perturbações) dentro do espaço de estados para explorá-lo, em busca do estado mais estável. A simulação é feita a uma temperatura T. A cada iteração, um novo estado candidato é apresentado. Caso este estado seja mais estável que o atual, ele é aceito. Caso contrário, é também aceito com uma probabilidade diretamente proporcional a T.

A probabilidade empregada para a aceitação de um novo estado garante que a média de qualquer propriedade do sistema, como a energia, possa ser calculada facilmente através da média de Boltzmann, para uma amostra suficientemente grande. A prova pode ser encontrada no artigo original [23].

É importante observar que o método utilizado para realizar as perturbações nos estados deve permitir que qualquer estado seja alcançado a partir de qualquer outro

Algoritmo 1 SimulaçãoMMC ($s_{inicial}$, T, $nMaxTentativas$)

$k \Leftarrow$ Constante de Boltzmann
$s_0 \Leftarrow s_{inicial}$
Para $i = 0$ **até** $nMaxTentativas$ **fazer**
  $e_0 \Leftarrow$ Energia calculada em s_0
  $s_1 \Leftarrow s_0 +$ perturbação
  $e_1 \Leftarrow$ Energia calculada em s_1
  Se $e_0 > e_1$ **então**
    $s_0 \Leftarrow s_1$
  Senão
    $r \Leftarrow$ número aleatório no intervalo [0,1]
    Se $e^{\frac{(e_1 - e_0)}{kT}} > r$ **então**
      $s_0 \Leftarrow s_1$

em um número de iterações menor ou igual a $nMaxTentativas$ (e, portanto, finito). Do contrário, alguns estados podem nunca ser alcançados, e a média de Boltzmann não vale.

No algoritmo acima, a média de Boltzmann com respeito a energia pode ser obtida se, ao final de cada iteração, for armazenada em um acumulador a energia do estado atual. Ao final da simulação, a média será o valor deste acumulador dividido pelo número de iterações.

2.2 Têmpera Simulada (*Simulated Annealing*)

Proposto em [20], este método de otimização utiliza uma simulação de Metropolis Monte Carlo para encontrar a configuração mais estável (com menor energia) de um sistema. A inspiração vem do processo de temperamento do vidro.

Inicialmente, o vidro é aquecido a uma temperatura muito elevada, de forma que se torne líquido e os átomos tenham maior mobilidade. A temperatura é reduzida gradualmente e os átomos do vidro buscam a configuração mais estável a cada temperatura. Se o resfriamento for conduzido de maneira lenta o suficiente, ao final do processo os átomos estarão na configuração mais estável.

Descrevemos agora o algoritmo da Têmpera Simulada. T_i e T_f são, respectivamente, as temperaturas inicial (alta) e final (baixa) do processo.

A taxa de resfriamento logarítmica utilizada não é a única possível. A distribuição de Cauchy também pode ser utilizada, mudando a fórmula da atualização da

Algoritmo 2 Têmpera Simulada (T_i, T_f, $nTentativasPasso$)

$nPassos \Leftarrow 0$
$s_0 \Leftarrow$ Estado aleatório
$T \Leftarrow T_i$
Enquanto $T > T_f$ **fazer**
  $s_0 \Leftarrow$ SimulacaoMMC(s_0, T, $nTentativasPasso$)
  $T \Leftarrow T_i/ln(nPassos)$
  $nPassos \Leftarrow nPassos + 1$

temperatura para a Equação 2.1.

$$T \Leftarrow \frac{T_i}{nPassos} \tag{2.1}$$

A determinação do parâmetro *nTentativasPasso* é uma das maiores dificuldades no uso da Têmpera Simulada. Embora a simulação Metropolis Monte Carlo tenha como objetivo reproduzir a distribuição de Boltzmann correta a uma dada temperatura, quando utilizada na Têmpera Simulada, ela só precisa rodar por tempo suficiente para explorar as regiões razoavelmente populadas do espaço de busca. Essa restrição de suficiência permite uma redução no número de iterações da simulação MMC. Entretanto, há uma relação entre este número e o tamanho máximo de cada passo (que depende do modo com que as perturbações nos estados são conduzidas) que pode ser difícil de ser otimizada, por ser muito dependente de características intrínsecas ao problema a ser resolvido.

2.3 Satisfabilidade

2.3.1 Definição do problema SAT

Uma fórmula da lógica proposicional é dita na forma normal conjuntiva (CNF) quando é uma conjunção ("e"s lógicos) de disjunções ("ou"s lógicos) de literais. Um literal é a ou $\neg a$, onde a é uma variável proposicional. A cada uma das disjunções damos o nome de cláusula.

Dada uma fórmula na CNF, resolver o problema SAT associado é encontrar uma valoração de suas variáveis booleanas tal que o valor da fórmula seja *verdadeiro*. Isto é equivalente a dizer que a valoração deve fazer com que cada cláusula da

fórmula possua pelo menos um literal com valor *verdadeiro*. Estas cláusulas são ditas satisfeitas; quando não existe uma valoração que satisfaça a todas as cláusulas, o problema é dito insatisfazível.

2.3.2 Definição do problema pseudo-booleano

Uma inequação pseudo-booleana é da forma $c_0x_0 + c_1x_1 + ... + c_{n-1}x_{n-1} \geq c_n$, onde para todo i, c_i é um coeficiente inteiro. x_i é da forma $p_j \times p_k \times ... \times p_l$, onde p_i é uma variável booleana ou sua negação. A inequação é dita linear se, para todo i, x_i for da forma p_i. Um literal *verdadeiro* é mapeado para o valor 1, ao passo que um literal $falso$ é interpretado como 0. A inequação pseudo-booleana é dita satisfeita sob uma determinada valoração se, e somente se, o primeiro membro for maior ou igual ao segundo. Uma função objetivo é uma expressão na mesma forma do primeiro membro de uma inequação pseudo-booleana. Dado um conjunto de inequações pseudo-booleanas Γ e um função objetivo f, o problema de otimização pseudo-booleana consiste em encontrar uma valoração para as variáveis de f que a minimize, enquanto satisfaz às inequações de Γ. O problema é dito linear se todas as suas inequações e função objetivo são lineares.

2.3.3 Minisat+

Para resolver problemas de otimização pseudo-booleanos, temos um conjunto de técnicas frequentes. Uma delas consiste no uso de um resolvedor SAT como método de decisão para um procedimento de mais alto nível com uma lógica mais rica. Outro baseia-se na extensão de resolvedores SAT para permitir que estes aceitem inequações e funções objetivo pseudo-booleanas.

O Minisat+ (introduzido em [11]) é um resolvedor de problemas pseudo-booleanos lineares que emprega uma técnica menos comum: transformar as inequações em cláusulas lógicas, e então aplicar um resolvedor SAT puro.

Antes da transformação, muitas normalizações são feitas para diminuir a diversidade das inequações e simplificar ao máximo o problema. Eis alguns exemplos:

- Inequações do tipo $\leq$ são transformadas em inequações do tipo $\geq$ através da negação de todas as suas constantes.

- Inequações satisfazíveis trivialmente, como "$p_1 + p_2 \geq 0$" são removidas.
- A presença de inequações trivialmente insatisfazíveis, como "$p_1 + p_2 \geq 9$" interrompe o pré-processamento e gera como resposta "insatisfazível".
- Em $c_0x_0 + ... + c_{n-1}x_{n-1} \geq c_n$, substitui-se todo $c_i \forall i < n$ por c_n, caso $c_i > c_n$.
- Valorações triviais são fixadas e propagadas para outras inequações. Por exemplo, a valoração $p_i = true$ é dita trivial se, ao assumir $p_i = false$, alguma inequação imediatamente se tornar insatisfazível.

Realizado o pré-processamento, as inequações são transformadas em circuitos de uma única saída através de um dos seguintes métodos: *binary decision diagram* (BDD), *network of adders* ou *network of sorters*. A preferência é para os BDDs, depois *sorters*, e por último *adders*, devido a um *tradeoff* entre as propriedades desejáveis do circuito resultante e a complexidade de tempo para gerá-lo.

Finalmente, é aplicada uma variação da transformação de Tseitin [29], que apresenta as cláusulas como saída. É importante notar que alguns dos passos dados até aqui podem introduzir novas variáveis ao problema, aumentando sua dimensão.

Dado um problema de otimização pseudo-booleano definido por inequações Γ e pela função objetivo $f(p)$, pode-se encontrar sua solução com chamadas iterativas ao resolvedor SAT. Primeiro, este é aplicado ao conjunto Γ' de cláusulas pré-processadas, e a valoração encontrada é utilizada para determinar $f(p) = k$. Adiciona-se $f(p) \leq k$ ao conjunto Γ e repete-se o processo até a obtenção de um conjunto insatisfazível. A sequência de passos está sintetizada no algoritmo 3.

Algoritmo 3 Otimizar Minisat+(Γ, $f(p)$)

$\Gamma' \Leftarrow$ preProcessar(Γ)
$v \Leftarrow$ resolverSAT(Γ')
$k \Leftarrow f(v)$
Enquanto $\Gamma \cup (f(p) \leq k)$ for satisfazível **fazer**
  $\Gamma' \Leftarrow$ preProcessar($\Gamma \cup (f(p) \leq k)$)
  $v \Leftarrow$ resolverSAT($\Gamma' \cup (f(p) \leq k)$)
  $k \Leftarrow f(v)$

2.3.4 Resolvedor bsolo

O resolvedor bsolo [22] utiliza uma variação da abordagem *branch and bound* [25] para solucionar problemas de otimização combinatória.

O processo utiliza uma árvore de busca, onde cada nó representa uma valoração para uma variável. Um caminho da raiz até uma folha qualquer determina os valores de tantas variáveis quantos forem os níveis da árvore. Caminhos não promissores são interrompidos (podados).

Os limites superiores para a função objetivo são identificados e os limites inferiores são estimados. Boas políticas para estimar os limites inferiores são de extrema importância para o bom funcionamento do algoritmo; as utilizadas pelo bsolo estão descritas em [10]. A árvore de busca é podada sempre que o valor para o limite superior se iguala ou se aproxima mais do valor ótimo do que a estimativa para o limite inferior. Eis os passos do *branch and bound* utilizado:

Algoritmo 4 Otimizar bsolo(Γ, $f(p)$)

```
lim_sup ⇐ ∞
Enquanto resposta não encontrada fazer
  Se v contém valorações para todas as variáveis então
    lim_sup ⇐ f(v)
    lançar um conflito[1] para garantir a poda
  Senão
    x ⇐ variável não-valorada de v
    v ⇐ v∪ valoração de x
    bifurcar a busca em x
  aplicar propagação booleana
  Se um conflito foi gerado então
    Γ ⇐ Γ∪ cláusulas relevantes
    Se backtracking é necessário então
      realizar backtracking
    Senão
      proceder com a busca
  lim_inf ⇐ nova estimativa
  Se lim_inf ≥ lim_sup então
    lançar um conflito
    realizar backtrack
```

A essa descrição simplificada, são acrescentadas otimizações provenientes de re-

[1] Conflitos ocorrem quando, durante a busca, um ramo cuja valoração correspondente torna alguma das restrições insatisfazível.

solvedores SAT, como *backtracking* não-cronológico e identificação de atribuições necessárias.

2.4 Particle Swarm Optimization (PSO)

O conceito de Particle Swarm Optimization (PSO) foi introduzido em [17], e corresponde à simulação do comportamento de um enxame em busca de comida. Ao invés de realizar a busca em um espaço real, com três dimensões, o espaço de busca do enxame é parametrizado por uma função objetivo, e terá tantas dimensões quantas forem as variáveis da função. Em cada dimensão, os intervalos válidos também são estabelecidos pela função objetivo, e não há nada no paradigma que os obrigue a serem iguais. Neste contexto, o melhor ponto para alimentação é aquele que minimiza o valor desta função, e a tarefa do enxame é encontrar esse ponto.

Os S elementos do enxame são chamados de partículas. Cada uma delas armazena apenas um pequeno conjunto de informações, a saber:

- sua velocidade vetorial $\vec{v}$;
- sua posição atual $\overrightarrow{pos}$;
- uma lista com as partículas que pertencem a sua vizinhança $\overrightarrow{neigh}$;
- o melhor ponto já alcançado por ela própria $\overrightarrow{pBest}$;
- o melhor ponto já alcançado por qualquer partícula na sua vizinhança $\overrightarrow{lBest}$.

No início do algoritmo, as partículas estão espalhadas aleatoriamente. Em todo passo, cada partícula se movimenta, seguindo as três seguintes tendências:

Inércia é a influência da velocidade atual da partícula sobre sua próxima posição. Faz com que ela continue seguindo na mesma direção no espaço de busca.

Individualismo é a influência do conhecimento local da partícula. Tende a trazer a partícula para o melhor ponto visitado por ela própria, por vezes contrariando a experiência conjunta do enxame.

Coletivismo é a influência do conhecimento do coletivo sobre a partícula. É uma tendência que move a partícula em direção ao ponto ótimo (até então) da vizinhança, em detrimento da exploração individualista.

Além da movimentação, as partículas também atualizam suas informações. Após um certo número de passos, espera-se que o enxame tenha convergido para um ponto, o *mínimo global* da função.

Em linhas gerais, este é o comportamento do algoritmo. Desde a sua criação, foram concebidas diversas variações, que têm por objetivo melhorar a performance do PSO em determinados casos. A seguir apresentamos o algoritmo original, e também algumas variantes interessantes, inclusive as utilizadas nos experimentos.

2.4.1 Tipos de vizinhança

Na Seção 2.4, este conceito foi mencionado superficialmente. Alterações na vizinhança podem mudar significativamente o comportamento do PSO. A topologia da vizinhança é facilmente visualizada se pensarmos nas partículas do PSO como nós em um grafo orientado, e nos relacionamentos de vizinhança como arestas nesse grafo.

A figura 2.1 mostra os dois tipos de vizinhança mais difundidos: a global, onde cada partícula é vizinha de todas as outras do enxame; e a em anel, onde cada partícula p_i é vizinha de outras duas p_{i-1} e p_{i+1}, $\forall i \in \mathbb{Z}_S$. Foi utilizada a notação de um grafo não-orientado porque, nestes exemplos, as vizinhanças são simétricas.

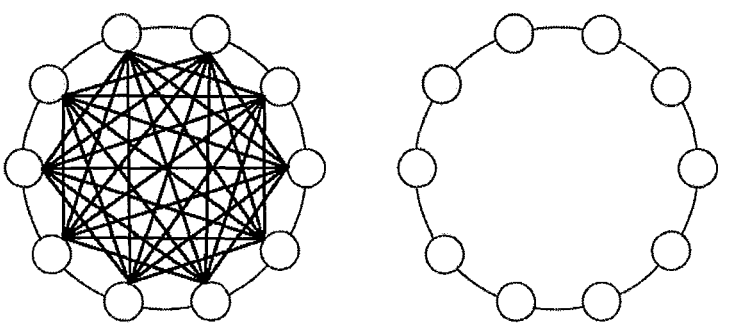


Figura 2.1: Vizinhanças comuns em PSO. À esquerda, global; à direita, anel.

Na vizinhança global, a informação sobre um novo ponto ótimo descoberto leva uma iteração para alcançar todas as partículas do enxame. Já na em anel, o número

de iterações necessárias para tal feito é $S/2$. Essa lentidão na propagação faz com que as partículas demorem mais para que sejam afetadas por um $\overrightarrow{lBest}$ recém-descoberto, permitindo a elas continuar a exploração com a informação atual por algum tempo. Isto tem a consequência teórica de fazer com que a convergência seja alcançada mais tardiamente, em troca de uma exploração mais completa do espaço de busca.

Existem inúmeras outras topologias para a definição das vizinhanças. Na verdade, não há restrição alguma para a formação destas, sendo possível até mesmo que variem durante a execução do algoritmo.

2.4.2 PSO clássico

Em [16], os criadores do PSO descrevem a primeira heurística para a movimentação das partículas do enxame, que serviu de base para todas as outras. Essa heurística consiste basicamente na implementação do comportamento descrito na Seção 2.4. O PSO clássico foi concebido para problemas cujas dimensões são números reais. Portanto, nesta seção, devemos considerar toda a informação numérica armazenada nas partículas ($\vec{v}$, $\overrightarrow{pos}$, $\overrightarrow{pBest}$ e $\overrightarrow{lBest}$) como valores em $\mathbb{R}^D$, onde D é o número de dimensões do problema.

A equação que rege a movimentação das partículas é simples, e dada pela 2.2.

$$\overrightarrow{pos} \Leftarrow \overrightarrow{pos} + \vec{v}. \tag{2.2}$$

Ou seja, a cada passo do algoritmo, as partículas movem-se seguindo suas velocidades atuais. A alma do PSO clássico está na atualização dessas velocidades, que é dada pela Equação 2.3.

$$\vec{v} \Leftarrow \vec{v} + \varphi_1 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{pBest}) + \varphi_2 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{lBest}), \tag{2.3}$$

onde $rand(0,1)$ é um número real aleatório entre 0 e 1; φ_1 e φ_2 são parâmetros do algoritmo. A correspondência entre as tendências da Seção 2.4 e as parcelas da Equação 2.3 é simples. *Inércia*, *individualismo* e *coletivismo* estão denotados pela primeira, segunda e terceira parcelas, respectivamente. Assim, podemos explicar a

atualização da velocidade como sendo a velocidade anterior perturbada pelas tendências do *individualismo* e *coletivismo*. A intensidade dessas perturbações varia aleatoriamente a cada atualização, respeitando um máximo para cada uma (φ_1 e φ_2).

Neste ponto temos todas as informações para explicitar o algoritmo.

Algoritmo 5 OtimizarSwarm()

```
inicializar cada partícula em posição e velocidade aleatórias
Enquanto critério de convergência não alcançado fazer
  Para toda partícula do enxame fazer
    avaliar a função objetivo na posição atual da partícula
    armazenar este valor no vetor temporário p⃗
    Se p⃗ < pBest⃗ então
      pBest⃗ ⇐ p⃗
      Se p⃗ < lBest⃗ então
        Para toda partícula em neigh fazer
          atualizar lBest;
    atualizar v⃗ segundo a Equação 2.3
    limitar cada componente de v⃗ ao intervalo [V_min, V_max]
    atualizar pos⃗ segundo a Equação 2.2
```

2.4.3 Os parâmetros do PSO

Uma característica até então não citada é a limitação da velocidade no intervalo $[V_{min}, V_{max}]$. O fato de as atualizações que ocorrem a cada passo nas velocidades serem estocásticas tem por consequência a possibilidade de que estas cresçam indefinidamente. Caso as velocidades não tivessem seu intervalo restrito, os altos valores que tomariam iriam impedir que as partículas convergissem para qualquer ponto. As constantes V_{min} e V_{max} são parâmetros do PSO. Excetuando-se raríssimos casos, $V_{min} = -V_{max}$, motivo pelo qual nos referiremos apenas a V_{max} no restante do texto. A determinação de um bom valor para este parâmetro é algo que depende de certos conhecimentos do problema. Por exemplo: caso seja configurado de forma que para escapar de um mínimo local seja necessário um passo maior do que V_{max}, então toda partícula que atingir este mínimo local ficará presa até que (ou a menos que) alguma partícula na vizinhança encontre um ponto melhor. A partir daí, a tendência do *coletivismo* atrairá a partícula presa para este novo ponto, possivelmente libertando-a do mínimo local.

Entretanto, não basta escolher um valor arbitrariamente alto para V_{max}. Embora isto contorne o problema descrito no parágrafo anterior, também prejudica a exploração de pontos próximos ao ótimo atual, onde há geralmente grande esperança de melhoria no melhor ponto encontrado.

É interessante notar o efeito que as constantes φ_1 e φ_2 têm sobre o comportamento do algoritmo. Quanto maior for a razão φ_1/φ_2, mais autônomas serão as partículas, o que fará com que a convergência ocorra mais tarde. Em contrapartida, quanto menor for esta razão, mais dependentes umas das outras as partículas serão, o que acarretará em uma convergência mais rápida. Essas constantes devem ser ajustadas empiricamente dependendo do tipo e da dificuldade do problema tratado, pois cada um tem sua velocidade de convergência ideal própria. Mesmo assim, há uma certo consenso na comunidade em aceitar $\varphi_1 = \varphi_2 = x$, e $\{0.9, 1, 2\}$ como bons valores para x.

Como visto em [15], uma partícula idealmente orbita um atrator formado por $\overrightarrow{pBest}$ e $\overrightarrow{lBest}$. Diferentes valores para os parâmetros φ_1, φ_2 e V_{max} alteram de forma definida o comportamento da partícula.

Uma análise do comportamento de uma partícula em condições especiais foi feito em [19], proporcionando uma boa visão da influência dos parâmetros na trajetória. Esta análise é apresentada[2] nesta seção, assim como nas seções 2.4.4 e 2.4.5, e considera um enxame de uma única partícula em um problema de uma única dimensão, onde $pBest$ contém o valor ótimo (0). Nestas condições, é possível simplificar a notação, assumindo $\varphi = \varphi_1 + \varphi_2$. A notação vetorial não é necessária, pois só há uma dimensão.

As Figuras 2.2, 2.3 e 2.4 ilustram as conclusões tiradas sobre o parâmetro V_{max}: quanto menor ele for, menor será o raio da órbita da partícula ao redor do atrator. Nas referidas figuras, $\varphi = 3.9$.

Nas Figuras 2.5, 2.6 e 2.7, o parâmetro V_{max} foi fixado em 2 para a análise do comportamento da trajetória em função de φ. Podemos observar que, à medida em que φ aumenta, a "força gravitacional" que atrai a partícula em direção ao ótimo aumenta. Como consequência, sua trajetória tem a amplitude reduzida, e apresenta cada vez menos passos entre uma passagem pelo ponto ótimo e outra. Podemos notar

[2]Os gráficos utilizados neste capítulo foram retirados de [19].

Figura 2.2: O fenômeno da explosão, que ocorre quando não limitamos a velocidade ($V_{max} = \infty$). Em apenas 150 iterações, a partícula foge da região de interesse.

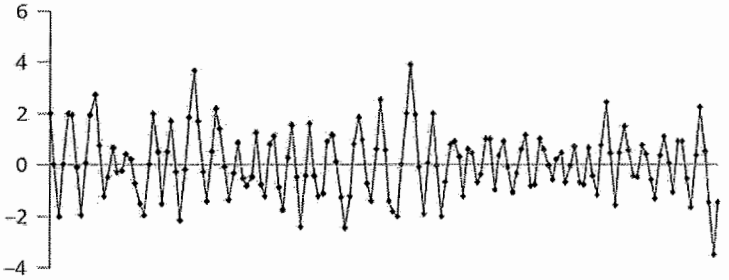


Figura 2.3: Quando $V_{max} = 2$, a partícula orbita o valor de $pBest$ com um raio médio entre 1 e 3.

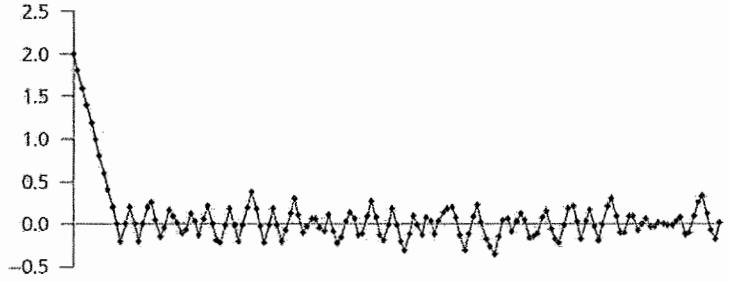


Figura 2.4: Configurando $V_{max} = 0.2$, o movimento da partícula fica restrito a uma região mais estreita ao redor do ótimo.

também, que um valor exageradamente alto para φ satura a trajetória, limitando a exploração a alguns pontos, pois os passos dados são quase sempre em sentidos alternados e de tamanho igual a V_{max}.

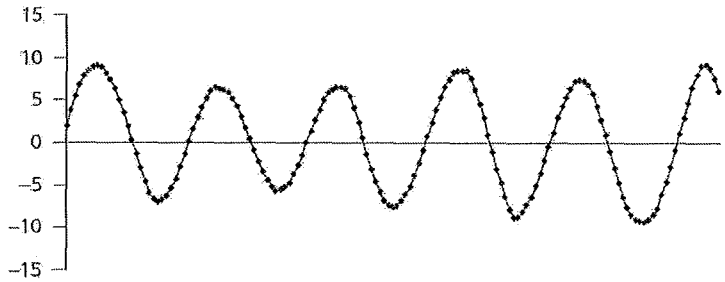


Figura 2.5: Quando $\varphi = 0.1$, a partícula se afasta bastante do ponto ótimo, até que a parcela $(pos - pBest)$ da equação da velocidade se torna grande o suficiente para puxá-la de volta.

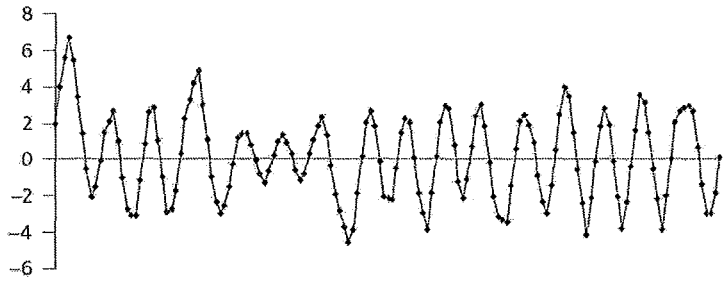


Figura 2.6: Com $\varphi = 1$, a trajetória apresenta o comportamento esperado, uma exploração irregular em torno do ótimo.

As próximas seções descrevem duas das principais variações do PSO clássico, a saber: PSO com inércia (*inertia constrained PSO*) e abordagem por fator de constrição (*constriction factor approach*). Ambas tem por objetivo fazer com que as partículas efetivamente convirjam para um ponto, ao invés de orbitar ao redor dele. A idéia por trás das modificações é aplicar coeficientes de tal forma que, com o passar das iterações, o movimento das partículas seja cada vez mais restrito.

2.4.4 PSO com inércia (*inertia constrained PSO*)

Esta variante, introduzida em [28], é provavelmente a mais difundida do PSO. Um coeficiente φ_0 é aplicado à primeira parcela da equação de atualização da velocidade, que passa a ser como a 2.4. O restante do algoritmo continua exatamente igual ao

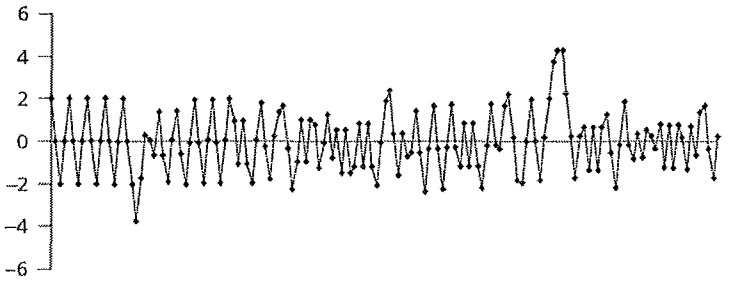


Figura 2.7: Quando $\varphi = 10$, a partícula explora ineficientemente os arredores do ponto ótimo. Dando passos de tamanho V_{max} na maior parte do tempo, ela passa repetidamente pelos mesmos pontos.

descrito na Seção 2.4.2.

$$\vec{v} = \varphi_0 \times \vec{v} + \varphi_1 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{pBest}) + \varphi_2 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{lBest}) \quad (2.4)$$

Assim como os outros parâmetros, a determinação do melhor valor para φ_0 depende do problema a ser resolvido. Geralmente são usados valores fixos menores que, porém próximos de 1, como 0.8, 0.9 ou 0.95. Outra possibilidade é fazer com que φ_0 varie ao longo das iterações. A variação é feita decrementando-se φ_0 linearmente de um valor inicial até um valor final, ambos definidos previamente. Os autores desta variante do PSO sugerem que φ_0 seja decrementado de 0.8 até 0.4.

O que se consegue com o uso do coeficiente de inércia é uma mudança progressiva no comportamento do algoritmo, passando da exploração do espaço de busca como um todo para a exploração de uma pequena região ao redor do(s) valor(es) ótimo(s) encontrado(s) até então. A explicação é simples: à medida em que o PSO progride, as velocidades tendem a diminuir, pois são multiplicadas por valores menores do que 1. Com isso, cada passo dado pelas partículas vai ficando menor, impedindo que as mesmas movam-se para muito longe de seus $\overrightarrow{pBest}$ e $\overrightarrow{lBest}$. Observemos na Figura 2.8 o comportamento desta variante em um experimento nas mesmas condições dos apresentados na Seção 2.4.3.

Podemos traçar um paralelo desta variante com a Têmpera Simulada (Seção 2.2): a desaceleração das partículas é equivalente ao resfriamento do vidro. Embora nem as partículas sejam reaceleradas, nem o vidro seja reaquecido, as chances de sucesso em achar o ponto ótimo aumentam quando diminui-se a velocidade do processo.

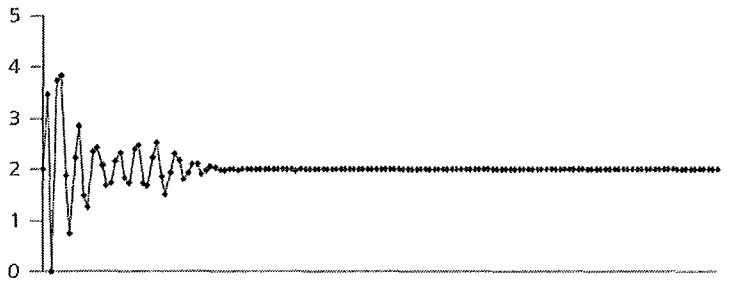


Figura 2.8: O PSO com inércia faz com que a partícula convirja para o atual ponto ótimo com o tempo.

2.4.5 Abordagem por fator de constrição (*constriction factor approach* ou CFA)

Esta variante é fruto de um estudo matemático do comportamento do Sistema de Equações 2.5 realizado em [9], que modela matematicamente o comportamento do PSO clássico.

$$\begin{cases} \vec{v}_{t+1} &= \vec{v}_t + \varphi\vec{y}_t \\ \vec{y}_{t+1} &= -\vec{v}_t + (1-\varphi)\vec{y}_t \end{cases} \tag{2.5}$$

A variável $\vec{y}_t$ é calculada através da Equação 2.6

$$\vec{y}_t = \frac{\varphi_1\overrightarrow{pBest} + \varphi_2\overrightarrow{lBest}}{\varphi_1 + \varphi_2} - \overrightarrow{pos_t} \tag{2.6}$$

A análise mostrou que é possível criar um sistema generalizado, onde a convergência e o fenômeno da explosão das velocidades podem ser controlados, e qua há infinitas maneiras de se fazer isso. Uma delas, denominada *constrição tipo 1"*, consiste na modificação da equação de atualização da velocidade para a Equação 2.7, respeitada a condição de que $\varphi = \varphi_1 + \varphi_2 > 4$.

$$\vec{v} = \chi\Big(\vec{v} + \varphi_1 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{pBest}) + \varphi_2 \times rand(0,1) \times (\overrightarrow{pos} - \overrightarrow{lBest})\Big) \tag{2.7}$$

A variável χ é calculada conforme a Equação 2.8.

$$\chi = \frac{2\kappa}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \tag{2.8}$$

onde $\kappa \in [0, 1]$ é um parâmetro que regula a intensidade da constrição. Quanto menor for, mais pronunciado será seu efeito.

O principal diferencial desta variante para o PSO com inércia está na capacidade do CFA de reacelerar as partículas, caso um ponto melhor do que o ótimo atual seja encontrado quando as partículas estão movimentando-se devagar, por estarem próximas da convergência. Este comportamento está exemplificado na Figura 2.9.

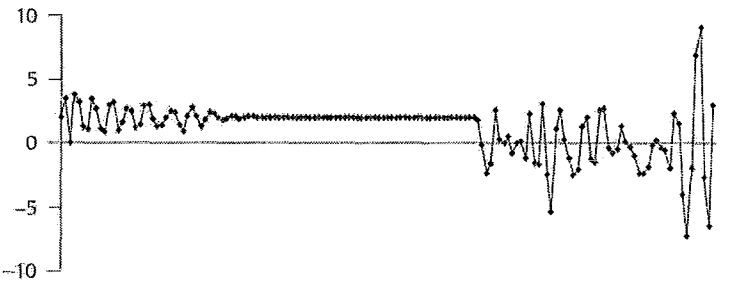


Figura 2.9: Com o CFA, mesmo após a convergência, se um novo ponto ótimo é inserido, a partícula volta a explorar o espaço entre os dois pontos ótimos.

2.4.6 PSO binário

O PSO clássico e suas variantes funcionam bem para problemas cujo espaço de busca é contínuo. Mas será possível aplicar esta técnica a outros grupos de problemas? Particularmente, será possível utilizá-la em situações onde o espaço de busca é discreto, ou até booleano? Em [18] e [8] são propostas alterações no algoritmo para tornar o PSO viável para espaços de busca não-contínuos. A seguir, descrevemos algumas delas. A primeira conserva, dentro dos limites do possível, maior semelhança com o PSO clássico, enquanto a segunda apresenta traços da heurística dos algoritmos genéticos.

2.4.6.1 Heurística 1

Por ser apenas uma discretização da heurística clássica, esta heurística não acrescenta muitos conceitos novos. A estrutura do algoritmo permanece rigorosamente a mesma. O que muda então? Embora a posição da partícula ($\vec{pos}$) seja agora um vetor de dimensões booleanas — onde *verdadeiro* = 1 e *falso* = 0 — $\vec{v}$ permanece contínuo. Logo, a única alteração necessária para que a heurística funcione no es-

paço binário deve ser feita na Equação 2.2. Isso é feito de forma que cada posição $\overrightarrow{pos}_d$ em $\overrightarrow{pos}$ seja atualizada conforme a Equação 2.9.

$$pos_d = \begin{cases} 1 & \text{se } \frac{1}{1+e^{-(pos_d+v_d)}} > rand(0,1); \\ 0 & \text{caso contrário.} \end{cases} \tag{2.9}$$

Embora as alterações do PSO clássico para esta versão sejam mínimas, a visão de certos componentes do algoritmo se modificas profundamente, o que requer explicações. O vetor $\vec{v}$ não representa mais a velocidade vetorial da partícula, pois tal conceito não faz muito sentido em um espaço no qual cada dimensão só tem duas posições possíveis. Nesse novo contexto, v_d, para cada dimensão d, representa a predisposição da partícula a tomar o valor 0 ou 1 na dimensão d. Quanto menor for v_d, maior a probabilidade de que pos_d, para esta partícula, seja igual a 0, e vice-versa. Valores de módulo extremamente alto para v_d, no PSO clássico, fazem com que a partícula se movimente vigorosamente, possivelmente cobrindo pontos distantes no espaço de busca em poucas iterações. No PSO binário, velocidades deste tipo implicam na indolência da partícula. Por exemplo, caso o módulo de v_d seja muito grande, e seu sinal negativo, a probabilidade de pos_d ser igual a 0 é tão grande que há pouquíssimas iterações do algoritmo nas quais a partícula se movimente de, ou para a posição 1.

O coeficiente de inércia da Seção 2.4.4 também teria um efeito bem diferente do pretendido, caso fosse aplicado a um enxame em um espaço de dimensões booleanas, pois a multiplicação por valores menores do que 1 diminui o módulo de v_d. No PSO clássico, a consequência disso é reduzir a movimentação da partícula até que, com $v_d = 0$, a partícula não se move na dimensão d. No PSO binário, $v_d = 0$ implica em pos_d ser igual a 0 com 50% de probabilidade. Ou seja, este valor para v_d é o que mais incita a movimentação.

2.4.6.2 Heurística 2

Esta heurística realiza modificações mais profundas na estrutura do algoritmo. Uma delas é que aqui não é levada em conta a velocidade da partícula. Para atualizar $\overrightarrow{pos}$, primeiro tomamos cópias de $\overrightarrow{lBest}$ e $\overrightarrow{pBest}$ com alguns bits invertidos; sejam