ADAPTAÇÃO DO ALGORITMO DE OTIMIZAÇÃO BASEADO EM ENXAME DE PARTÍCULAS PARA EXECUÇÃO EM CLUSTERS: UMA ABORDAGEM UTILIZANDO O MODELO ILHA

Jacques Brawerman

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. Luis Alfredo Vidal de Carvalho, D.Sc.

Profª. Myrian Christina de Aragão Costa, D.Sc.

Profª. Lúcia Maria de Assumpção Drummond, D.Sc.

Prof. Claudio Thomas Bornstein, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2006

BRAWERMAN, JACQUES

Adaptação do Algoritmo de Otimização Baseado em Enxame de Partículas para Execução em Clusters: Uma Abordagem Utilizando o Modelo Ilha [Rio de Janeiro] 2006.

VIII, 79p. 29,7cm em (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Otimização
2. Paralelismo
3. Clusters
4. Problema da Mochila

I. COPPE/UFRJ II. Título (série)

Aos meus pais Jamile e Ilitsz,
por estarem sempre ao meu lado
e me darem carinho, amor e força
em todos os momentos de minha vida.

Agradecimentos

Gostaria de agradecer primeiramente a Deus por ter permitido que este trabalho tenha se realizado.

Aos meus pais, Jamile e Ilitsz, por toda o carinho, amor, força e compreensão que me permitiram concluir esta dissertação.

Ao meu orientador professor Luis Alfredo Vidal de Carvalho por ter me conduzido durante a realização deste trabalho.

Ao programa de Inteligência Artificial da COPPE/SISTEMAS por ter permitido o meu ingresso, e as professoras Roseli Suzi Wedeman e Rosa Maria Esteves M. da Costa por me encaminharem ao meu orientador.

Aos meus professores da UERJ, que me deram a formação necessária para poder cursar o mestrado.

As professoras Myrian Christina de Aragão Costa e Beatriz de Souza Leite Pires Lima por todo ensinamento obtido em suas aulas.

Ao professor Basilio de Bragança Pereira pela ajuda que me prestou para a realização desta dissertação.

A todos os amigos que fiz durante o mestrado, em especial à Aline Marins Paes, Paula Fernanda Machado Vieira de Carvalho e Juliana Silva Bernardes pelos momentos de estudo que passamos juntos.

Ao meu amigo David Sotelo Pinheiro da Silva, por todo o auxílio que me prestou durante a execução deste trabalho.

À minha amiga Vanessa dos Reis de Souza por todo o apoio me dado durante a realização desta dissertação.

Ao meu gerente na PETROBRAS, Nelson Borges Alves Neto por ter permitido que eu concluísse mais esta fase em minha vida. Aos amigos de empresa Carlos Alberto Bayeux e Pedro Augusto de Araujo Pinto por todo o auxílio fornecido.

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

ADAPTAÇÃO DO ALGORITMO DE OTIMIZAÇÃO BASEADO EM ENXAME DE PARTÍCULAS PARA EXECUÇÃO EM CLUSTERS: UMA ABORDAGEM UTILIZANDO O MODELO ILHA

Jacques Brawerman

Março / 2006

Orientador: Luis Alfredo Vidal de Carvalho

Programa: Engenharia de Sistemas e Computação

Esta Dissertação apresenta uma adaptação do algoritmo de Otimização Baseado em Enxame de Partículas, para que este possa ser executado de forma paralela em diversas máquinas. O objetivo é verificar se há uma melhora nesta técnica avaliando se ela converge de forma mais rápida para um resultado ótimo.

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

CLUSTER ADAPTATION OF THE PARTICLE SWARM OPTIMIZATION ALGORITHM: AN ATTEMPT USING THE ISLAND MODEL

Jacques Brawerman

March / 2006

Advisor: Luis Alfredo Vidal de Carvalho

Department: Computer and Systems Engineering

This Dissertation presents an adaptation of the Particle Swarm Optimization, in order that this can be executed in the parallel form in several machines, using the Island Model. The objective is verify if there is an improvment in this technique, checking if it converges in less iterations and results in an optimum value.

1. Introdução

1.1. Apresentação do problema

Existem problemas que por maiores que sejam os recursos computacionais disponíveis, não podem ser resolvidos em um tempo aceitável ou requerem um espaço de armazenamento, seja em memória ou disco, que não pode ser fornecido. Uma classe deste tipo de problema são os chamados NP-Completo. Até agora não se provou que todas as instâncias destes possam ser resolvidos em um tempo razoável. Entretanto, alguns destes problemas são importantes aplicações para o campo industrial por exemplo.

Como a solução ótima para grandes instâncias deste tipo de problema se mostra inviável na prática, procura-se obter uma solução aproximada. Para isto são usadas heurísticas, que tentam obter uma solução aproximada para o problema com algumas informações a respeito deste.

Existem vários tipos de heurísticas para tentar resolver estes tipos de problemas. Uma heurística chamada otimização baseada em enxame de partículas – "Particle Swarm Optimization" (PSO) – é baseada no movimento de grupos de pássaros. Neste trabalho, esta heurística será adaptada para poder ser utilizada em um ambiente com várias máquinas interligadas agrupadas em um "cluster". Para testar esta adaptação no algoritmo, será utilizado um problema NP-Completo conhecido como problema multidimensional da mochila 0/1.

1.2. Descrição do trabalho desenvolvido

Para a resolução do problema da mochila com uma ou múltiplas restrições, a versão para problemas binários do PSO será adaptada para funcionar em cluster segundo o modelo ilha. As instâncias do problema da mochila com múltiplas restrições serão executadas de forma monoprocessada e multiprocessada. O objetivo é verificar se há alguma melhora em termos

de convergência ou de obtenção do valor ótimo, utilizando-se uma abordagem clusterizada para a execução do PSO, em relação à versão monoprocessada.

1.3. Apresentação dos capítulos

- No capítulo 1 é feita uma apresentação da tese bem como seu objetivo;
- No capítulo 2 é feita a apresentação do problema da mochila em suas versões com apenas 1 restrição e com múltiplas restrições. Além disso, são apresentadas algumas variantes de resolução utilizando outras técnicas e um histórico sobre o mesmo;
- No capítulo 3 é feita uma apresentação sobre clusterização. A importância desta técnica, os modelos de arquiteturas existentes, a utilização do MPI para a programação em ambientes deste tipo;
- No capítulo 4 é feita a apresentação do PSO contando a história do algoritmo bem como explicando a versão atual;
- No capítulo 5 é apresentada a metodologia usada para a realização dos experimentos bem como é explicado de forma mais detalhada o algoritmo adaptado para o modelo de ilhas;
- No capítulo 6 os resultados são apresentados;
- No Capítulo 7 a conclusão da tese é apresentada.

2. Problema da mochila

O problema da mochila consiste em tendo-se um conjunto de objetos, cada um com um volume e um valor monetário, e uma mochila com um volume específico, tentar colocar nesta a quantidade de produtos de forma a maximizar o valor transportado sem que a soma dos volumes dos produtos ultrapasse o volume da mochila.[10]

Este tipo de problema pode aparecer também em outros contextos. Suponha uma quantia em dinheiro para ser investida em aplicações financeiras, sendo que cada uma tem um retorno específico e um valor mínimo de capital para o investimento. Se este caso for tratado com uma instância do problema da mochila, a resposta será a melhor maneira de investir o capital disponível.[10]

Na indústria, este tipo de problema aparece em várias situações como: orçamento, carregamento de veículos para distribuição de produtos entre uma matriz e suas filiais e ajuste de produtos em estoque por exemplo.[10]

Além disso, outros fatos que levam ao intenso estudo deste problema são:

- pode ser considerado como uma forma mais simples de problemas de programação linear inteira;
- aparece como um subproblema em vários problemas complexos;
- pode ser adaptado a várias situações.[10]

2.1. Tipos de problema

Existem vários tipos de problema da mochila variando sobre a versão original. A principal distinção se faz entre os problemas da mochila que admitem partes fracionárias de produtos, e os que não admitem esta condição. Aqueles são chamados de problema da mochila fracionários e estes são chamados de problema da mochila 0/1.

2.1.1. Problema da mochila 0/1

Sejam n produtos p_n. Cada produto possui um custo associado c_n e um volume associado v_n. Os valores de p_n, c_n e v_n são considerados sempre positivos. O volume total que a mochila consegue transportar será indicado por V_{tot}. O objetivo do problema é [10]:

$$\text{Maximizar: } \sum_{j=1}^{n} c_j \cdot x_j \tag{2.1}$$

$$\text{Sujeito as restrições: } \sum_{j=1}^{n} v_j \cdot x_j \leq V_{tot} \tag{2.2}$$

$$x_j = 0 \text{ ou } 1 \text{ para } j = 1 \text{ a } n \tag{2.3}$$

Sendo que x_j será igual a 1 se o objeto estiver dentro da mochila e igual a 0 se ele não for posto nesta.

2.1.2. Problema fracionário da mochila

O problema da mochila fracionário é semelhante ao problema da mochila 0/1. A diferença entre os dois é que na versão fracionária, a restrição (2.3) seria substituída por [9,10]:

$$0 \leq x_j \leq 1 \text{ para } j = 1 \text{ a } n \tag{2.4}$$

Neste caso, seria permitido colocar frações de um produto dentro da mochila como, por exemplo, escolher entre algumas latas de refrigerante de um engradado. No item 2.1.1 não há esta possibilidade sendo o conjunto de latas tratado apenas como um item que não pode ser dividido na hora da venda.

2.1.3. Problema multidimensional da mochila 0/1

Este tipo de problema consiste na versão 0/1 do problema da mochila, apenas adicionando restrições a cada dimensão da mochila. Também é conhecido como problema da mochila m-dimensional (devido ao número de dimensões das restrições), problema da mochila com múltiplas restrições, problema da mochila múltiplo, sendo que ainda alguns autores colocam

0/1 no nome para indicar que este não é um problema da mochila do tipo fracionário. É matematicamente formulado da seguinte maneira:

$$\text{Maximizar: } \sum_{j=1}^{n} c_j \cdot x_j \tag{2.5}$$

$$\text{Sujeito as restrições: } \sum_{j=1}^{n} a_{ij} \cdot x_j \leq R_i, \ \forall \ i = 1 \ldots m \tag{2.6}$$

$$x_j = 0 \text{ ou } 1 \text{ para } j = 1 \text{ a } n \tag{2.7}$$

$$a_{ij} \geq 0 \text{ para } \forall \ i = 1 \ldots m, \ \forall \ j = 1 \ldots n \tag{2.8}$$

$$R_i \geq 0 \text{ para } \forall \ i = 1 \ldots m \tag{2.9}$$

Sendo n produtos p_n, cada produto possuindo várias características associadas como custo, representado por c_n, largura, altura, profundidade, volume, entre outros. Neste caso, há m restrições, sendo que cada uma é devida a uma característica do produto que é relevante para o preenchimento da mochila. A constante a_{ij} significa a restrição do produto j em relação a característica que está sendo observada na dimensão i. Se na dimensão 1 as restrições estivessem associadas a largura dos produtos, a_{21} por exemplo seria a largura do produto 2.[11]

2.2. Métodos de resolução

A seguir são apresentados alguns métodos de resolução do problema da mochila.

2.2.1. Força bruta

Esta técnica pode ser utilizada para a resolução de qualquer um dos tipos apresentados de problema da mochila. O método de resolução por força bruta consiste em se listar todas as combinações possíveis de produtos. Deste conjunto, eliminar aqueles em que a soma dos volumes dos produtos selecionados seja maior do que a capacidade da mochila. Entre as opções que sobrarem, verificar qual a que possui maior valor monetário. O número de soluções deste tipo a serem investigadas é 2^n. Se houvesse um computador capaz de examinar 10^6 vetores por segundo, cada um contendo uma suposta solução, para um problema de 60 produtos seriam necessários mais de 30 anos. Adicionando-se apenas mais um elemento seriam necessários 60 anos. Para 65 produtos este algoritmo seria executado em 10 séculos [10].

2.2.2. Programação dinâmica

Esta técnica pode ser utilizada apenas para a resolução da versão não fracionária do problema da mochila. A técnica de programação dinâmica começou a ganhar uma sólida base matemática a partir dos estudos de R. Bellman em 1955. Programação neste contexto se refere ao uso de métodos de solução que utilizavam tabelas[8].

A programação dinâmica consiste em resolver problemas e ir guardando as soluções intermediárias em uma tabela, por exemplo, para caso o valor computado seja necessário em futuras operações, este não tenha que ser recalculado. Quando os subproblemas utilizados para a resolução de um problema não são independentes a programação dinâmica representa uma grande vantagem sobre um método recursivo que não guarda os valores intermediários.

Um exemplo a ser citado é o cálculo do número de Fibonnacci. Este cálculo é feito baseado nas seguintes fórmulas:

$$Fib(0) = Fib(1) = 1 \quad (2.10)$$
$$Fib(n) = Fib(n-1) + Fib(n-2) \text{ para } n > 1 \quad (2.11)$$

Para o cálculo de Fib(4) por exemplo, Fib(3) é calculado 1 vez, Fib(2) é calculado 2 vezes, Fib(1) é calculado 3 vezes e Fib(0) é calculado 2 vezes. Com isso, percebe-se que há um gasto desnecessário de poder computacional. A solução baseada em programação dinâmica guarda os valores intermediários sendo que quando for necessário calcular Fib(2) no exemplo acima, este será calculado apenas 1 vez. Nas próximas vezes este valor será obtido de uma tabela já existente[9].

Para se tentar resolver um problema de otimização utilizando programação dinâmica são realizados os seguintes passos:

- estabelecer uma propriedade recursiva que produza uma solução ótima para cada instância do problema;
- realizar cálculos dos subproblemas e construir a solução ótima de forma "bottom-up".

Além disso, para que um problema possa ser resolvido utilizando-se programação dinâmica, este deve obedecer ao princípio da otimalidade, que diz que se S é uma solução ótima para um problema, então os componentes de S são soluções ótimas para os subproblemas[2,9].

2.2.2.1. Resolução do problema da mochila com programação dinâmica

Seja C[i][w] o valor monetário ótimo transportado quando escolhe-se produtos entre os primeiros i itens sob a restrição que o volume total não ultrapasse w, onde $i > 0$ e $w > 0$.

Tem-se então:

$$\text{Para } w_i < w\text{: } C[i][w] = \text{Max } (C[i-1][w], c_i + C[i-1][w - w_i]) \quad (2.12)$$
$$\text{Para } w_i > w\text{: } C[i][w] = C[i-1][w] \quad (2.13)$$

Em (2.12) o custo será o máximo entre o custo obtido com os produtos atuais e com a adição do i-ésimo produto caso o seu volume não ultrapasse o volume disponível na mochila.

Em (2.13) o volume do i-ésimo produto excederia o volume ainda restante na mochila, não devendo desta forma ser inserido na mochila.

Em [9] é apresentado um algoritmo para resolução do problema da mochila que utiliza estes conceitos. A entrada para o algoritmo é composta pelo número de elementos disponíveis n e pelo volume da mochila V. A complexidade deste algoritmo é O(nV). Entretanto, esta complexidade não é polinomial pois V não é dependente de n. Neste caso, se V for muito maior do que n, igual a n! por exemplo, têm-se que o algoritmo vai executar n.n! vezes. Se n for igual a 20 elementos, e com um computador capaz de executar 10^6 instruções por segundo, esta instância demoraria aproximadamente 1500000 anos para ser executada [9]. Neste caso, o algoritmo calcula todos os valores para a tabela da programação dinâmica. Se não for necessário verificar, para cada linha i da tabela, todos os valores de w entre 1 e W (volume máximo que se quer calcular), a performance pode ser melhorada. Para saber C[i][w] deve-se calcular apenas as entradas necessárias na (i-1) linha (C[i-1][w], C[i-1][$w - w_i$]). Assim, há no máximo 2^i computações a serem feitas na i-ésima linha. A partir deste fato se obtém que a soma do número de itens que deve ser computada em todas as linhas para se atingir o resultado será de $2^n - 1$. Com esta alteração o algoritmo passaria a ser $O(2^n)$ [8]. Combinando O(nV) com $O(2^n)$ têm-se que a complexidade deste algoritmo é O(Mín(nV, 2^n)) [9].

2.2.3. Algoritmo guloso

Esta técnica pode ser utilizada para a resolução da versão fracionária do problema da mochila. Com o algoritmo guloso, a solução é alcançada a partir de uma seqüência de escolhas sendo que cada uma é a melhor possível no momento. Está assim se buscando o

ótimo local a cada tentativa, com a esperança de que o ótimo global seja alcançado, o que nem sempre acontece [8].

2.2.3.1. Algoritmo guloso para o problema da mochila

Algumas estratégias podem ser adotadas na tentativa de se utilizar o algoritmo guloso para tentar resolver o problema da mochila.

1) Escolher sempre os produtos de maior valor monetário, de forma que o volume dos produtos não exceda o volume da mochila, conforme mostrado na tabela 2.1;
2) Escolher sempre os produtos de menor valor monetário, de forma que o volume dos produtos não exceda o volume da mochila, conforme mostrado na tabela 2.2;
3) Escolher os produtos que possuam a maior razão entre custo / volume de forma que o volume dos produtos não exceda o volume da mochila, conforme mostrado na tabela 2.3.

Tab. 2.1 - Exemplo para o caso 1 com uma mochila de volume máximo de 30 m^3:

Produto	Custo (R$)	Volume (m^3)
A	10	25
B	9	10
C	9	10

Neste caso, o algoritmo escolheria o produto A quando na verdade, uma escolha melhor seriam os produtos B e C.

Tab. 2.2 – Exemplo para o caso 2 com uma mochila com volume máximo de 30 m^3:

Produto	Custo (R$)	Volume (m^3)
A	8	5
B	7	20
C	9	7

Neste caso, o algoritmo escolheria os produtos A e B quando na verdade, uma escolha melhor seriam os produtos B e C.

Tab. 2.3 – Exemplo para o caso 3 com uma mochila com volume máximo de 30 m^3:

Produto	Custo (R$)	Volume (m^3)	Custo (R$) /Volume ($m^3$)
A	50	5	10
B	60	10	6
C	140	20	7

Neste caso, o algoritmo escolheria os produtos A e C quando na verdade, uma escolha melhor seriam os produtos B e C.

Entretanto, para o problema fracionário da mochila o algoritmo guloso sempre encontra a solução ótima [8].

2.2.4. "Backtracking"

É uma técnica utilizada para resolver problemas em que uma seqüência de objetos deve ser escolhida, a partir de um determinado conjunto, seguindo um critério específico. Estas escolhas são feitas percorrendo-se uma árvore (grafo acíclico) utilizando a busca em profundidade [9]. Esta técnica é utilizada para resolver qualquer tipo de problema da mochila.

2.2.4.1. "Backtracking" para o problema da mochila

No "backtracking", as possíveis soluções são dispostas como se fossem caminhos de uma árvore a serem percorridos. Enquanto que na força bruta todas as combinações são geradas, no "backtracking", a partir de um determinado momento em que se descobre que a seqüência de itens que se está percorrendo já resulta em volume de produtos maior do que o volume da mochila, este caminho é abandonado e volta-se ao nó pai do nó no qual o fato ocorreu. Este processo é chamado de poda. Após ter retornado ao nó pai do nó atual, percorre-se os outros filhos deste nó [8].

Tab. 2.4 – Exemplo para uma mochila com volume máximo de 6 m^3:

Produto	Custo (R$)	Volume (m^3)
A	10	2
B	20	4
C	8	5

A árvore gerada a partir dos dados da tabela 2.4 seria a mostrada na figura 2.1.

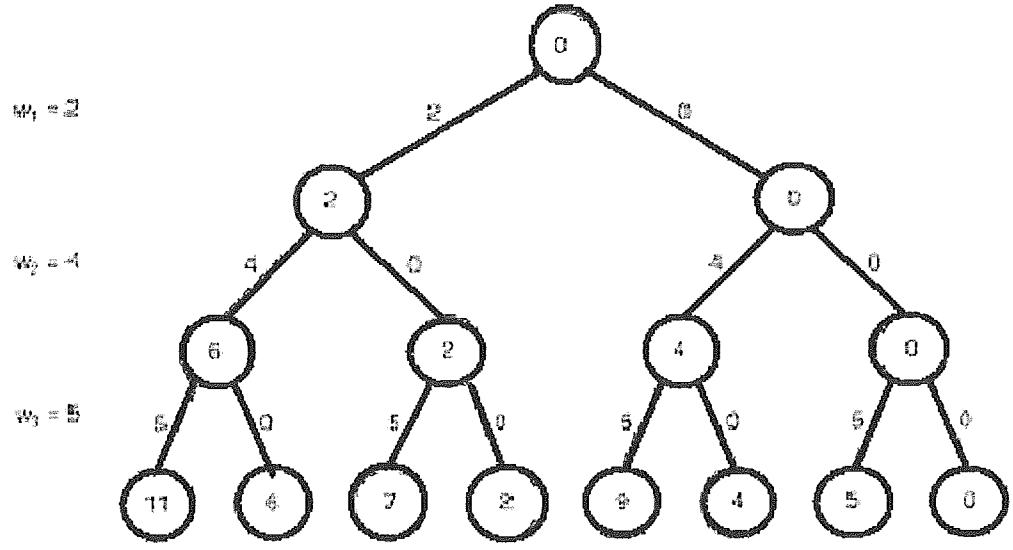

Fig 2.1 Árvore gerada para a execução de "backtracking" [9]

Sendo que os ramos da esquerda significam adicionar o elemento ao conjunto de itens e os ramos da direita significam não adicionar o item ao conjunto atual de itens. Em cada nível têm-se 2^i elementos, sendo i o nível da árvore. Caso a solução ótima estivesse no último nó do último nível e mais a direita na árvore, poderia acontecer uma situação em que ter-se-ia visitado $1 + 2 + ... 2^i$ itens sem que uma poda tivesse sido realizada. Este valor é igual a $2^{n+1} - 1$, e no pior caso esta técnica possui complexidade exponencial [9]. Na prática entretanto Horowitz & Sahni (1978) demonstraram que geralmente a técnica de "backtracking" é mais eficiente do que a de programação dinâmica [9,10].

2.2.5. "Branch and bound"

Esta é uma técnica parecida com o "backtracking". Também utiliza uma árvore como espaço de busca. As diferenças entre os dois métodos são as seguintes:

- enquanto no "backtracking" a árvore tem que ser percorrida com uma busca em profundidade, no "branch and bound" esta restrição não existe [9];
- esta técnica é utilizada apenas para problemas de otimização, enquanto o "backtracking" não (como por exemplo o problema das n-rainhas[1])[9].

Esta técnica serve para resolver qualquer tipo de problema da mochila.

2.2.5.1. "Branch and bound" para o problema da mochila

Não havendo a restrição da busca em profundidade do "backtracking", em [9] é reportado uma implementação de "branch and bound" para o problema 0/1 da mochila com uma busca em largura na árvore. Para cada nível, obtém-se o nó com maior valor financeiro até o momento e a partir daí os filhos deste nó serão explorados. Esta adaptação é chamada de "best-fit" com poda em "branch and bound". Mesmo assim no pior caso o algoritmo continua tendo uma complexidade assintótica exponencial [9].

2.3. *Histórico*

Nos anos 50, R. Bellman começa a prover a programação dinâmica de um maior formalismo matemático daí surgindo a teoria que produziria o primeiro algoritmo para resolver o problema da mochila 0/1. Em 1957, Dantzig desenvolveu um método eficiente para determinar a solução da versão fracionária do problema. Neste tipo de relaxação de restrição, o problema torna-se polinomial e pode ser resolvido em O(n) [10].

Nos anos 60, o problema da mochila foi estudado por Gilmore & Gomory [10]. Em 1965, Balas apresentou uma versão chamada 0/1 aditivo [11].

Na década de 70, Ingargiola e Korsh, em 1973 apresentaram um algoritmo de pré-processamento chamado de "reduction procedure" que diminuía significativamente o número de variáveis. Em 1974, Johnson apresentou a primeira aproximação de um resultado em tempo polinomial para o problema de soma de subconjuntos. Este resultado foi estendido para o problema 0/1 da mochila por Sahni. Ainda neste ano, Horowitz & Sahni, juntaram as técnicas de dividir para conquistar e programação dinâmica obtendo um algoritmo para resolver o problema 0/1 da mochila, que executava em $O(2^{n/2})$ no pior caso, tendo mostrado ainda que esta versão era mais eficiente do que a obtida utilizando o método de

"backtracking" [9]. A primeira aproximação completa de um resultado em tempo polinomial foi obtida em 1975 por Ibarra & Kim [10]. Em 1976 Kolesar fez uma primeira tentativa de se utilizar a técnica de "branch and bound" para resolver o problema, sendo que esta se mostrava para a época como sendo o único método para resolver instâncias com muitas variáveis[10]. Em 1979, Shih apresentou uma versão de problema da mochila múltiplo. Para instâncias de 5 restrições e 90 itens, seus resultados se mostraram melhores do que os obtidos por Balas em 1965 [11]. Neste mesmo ano, Loulou & Michaelides apresentaram um método baseado em algoritmo guloso e na "Primal heuristic" de Toyoda. Esta heurística foi testada em problemas de tamanhos variados, mostrando que o desvio ocorrido do ótimo conhecido para a solução encontrada ficou entre 0,26% e 1.08% para problemas pequenos e foi de 14% para problemas grandes [11].

Na década de 80 começam a surgir os resultados para instâncias grandes de problemas. Em 1984, Magazine & Oguz desenvolveram uma heurística que combinava as idéias da heurística dual de Senju & Toyoda (1968) com a técnica de generalização de multiplicadores de Lagrange apresentada por Evertt em 1963. Este algoritmo foi testado em problemas com o número de restrições variando entre 20 e 1000 assim como o número de objetos. Comparando os resultados obtidos com a técnica de "primal heuristic" e da heurística dual, a "primal heuristic" obteve melhores resultados do que a técnica deles sendo que a heurística dual ficou atrás destas duas. Em termos de tempo, a técnica das heurísticas combinadas e da heurística dual se saíram melhores do que a "primal heuristic". A complexidade assintótica da "primal heuristic", da heurística dual e da combinação de técnicas proposta por Magazine & Oguz foram $O(mn^2)$, $O(mn^2)$ e $O(mn^3)$ respectivamente, onde m representa o número de restrições e n o número de objetos[11]. Em 1985, Gavish e Pirkul apresentaram uma nova versão de "branch and bound" para o problema múltiplo da mochila. Eles testaram vários tipos de relaxamento de restrições para o problema. A que se mostrou melhor foi a relaxação

composta, apesar de requerer um maior poder computacional. Foram sugeridas regras para diminuição do tamanho do problema tendo estas se mostrado eficientes em testes computacionais, diminuindo em 1 ordem de magnitude os resultados encontrados por Shih em 1979 para problemas com 5 restrições e 200 objetos. Neste mesmo ano, Fox & Scudder apresentaram uma heurística onde todas as variáveis inicialmente começavam com o valor 0 e ao longo do processo iam sendo escolhidas variáveis para serem trocadas por 1. Resultados computacionais foram apresentados para um conjunto de problemas gerados de forma aleatória contendo até 100 restrições e 100 objetos. Ainda neste algoritmo cada produto possuía apenas 1 exemplar e a restrição poderia assumir os valores 0 ou 1, ou seja, aquele produto contentava ou não a restrição. Em 1988, Drexel apresentou uma solução baseada em "simulated annealing" obtendo resultados em 25 dos 57 problemas testados [11].

Na década de 90, Volgenant & Zoon trabalharam em cima dos resultados obtidos por Magazine & Oguz e obtiveram uma melhora, na média, acarretando uma pequena adição no tempo computacional. A complexidade do algoritmo desenvolvido ficou em O(n (n + m)) [11]. Em 1993, Dammeyer & Voss apresentaram uma heurística utilizando "tabu search" baseada em eliminação reversa. Conseguiram encontrar uma solução ótima em 41 dos problemas testados[11]. Em 1994, Khuri, Bäck & Heitkötter apresentaram um algoritmo genético para resolver o problema da mochila multidimensional, que permitia que soluções que violassem as restrições fizessem parte da população. Entretanto, nestes casos era aplicada uma penalidade à função de avaliação destes genes. Testes foram realizados em um pequeno número de problemas padrão, porém não foram obtidos resultados satisfatórios[11]. Neste mesmo ano, Thiel & Voss combinaram algoritmos genéticos com "tabu search". Os resultados obtidos não eram competitivos com outras heurísticas em termos computacionais[11]. Em 1995, Battiti & Tecchiolli apresentaram uma heurística baseada em "reactive tabu search" que é uma adaptação a técnica sendo que o tamanho do "tabu" (lista)

varia no decorrer da execução. Foram apresentados resultados para problemas com 500 objetos e 500 restrições[11]. Entre 1995-1996, Rudolph & Sprave adaptaram um algoritmo genético no qual a seleção seria restrita a uma vizinhança de soluções. Os indivíduos que violavam as restrições eram penalizados sendo que um limite de aceitação para os filhos provindo destes indivíduos foi introduzido. Resultados computacionais foram apresentados para um problema de 5 restrições e 50 objetos[11]. Em 1996, Glover & Kochenberger apresentam outra heurística baseada em "tabu search", conseguindo resultados em 57 dos 57 problemas testados, obtendo ainda resultados em instâncias com 25 restrições e 500 objetos[11]. Neste mesmo ano, Lokkentagen & Glover apresentaram uma heurística baseada em um "tabu search" probabilístico. Obtiveram resultados em 18 dos 57 problemas, sendo que estes possuíam tamanhos de até 30 restrições e 90 objetos[11]. Ainda em 1996, Hoff, Lokkentangen & Mittet adaptaram um algoritmo genético no qual todos os indivíduos deveriam ser soluções viáveis. Esta adaptação obteve resultados em 56 dos 57 problemas obtidos da literatura.[11]

3. "Cluster" computacional

Um "cluster" computacional, a partir de agora referido no presente trabalho apenas como "cluster", é um grupo de computadores que funcionam de forma coordenada e possuem 3 elementos básicos:

- computadores;
- rede para interconexão destes computadores;
- "software" que permita a estes computadores coordenarem-se e compartilharem informações e trabalho.

Para se construir "clusters", não há necessidade da utilização de computadores específicos. Máquinas normais podem ser utilizadas. O tipo mais conhecido de "cluster" chama-se Beowulf ("tendo o poder de muitos"), construído por Thomas Sterling e Don Becker para a NASA em 1994. Este cluster empregava computadores comuns e "software" livre [5].

Em "clusters" construídos com computadores comuns, geralmente os processadores são mais rápidos do que a rede subjacente. Além disso, cada computador pode ou não ser de uso exclusivo do cluster. Uma situação de exemplo seria um laboratório ou uma empresa, no qual após o término do expediente, determinadas pessoas poderiam usar os computadores que não estão sendo usados para montar um cluster. Este fato é facilitado pois existem pacotes de "software" que configuram um cluster a partir de um cd sem precisar formatar ou alterar o disco da máquina. Um destes exemplos é o "Bootable Cluster CD" (BCCD). Se os computadores forem utilizados apenas no cluster, então nem todos precisam de alguns periféricos como teclado, monitor e "mouse". Um computador principal, geralmente chamado de controlador ("head"), possuiria estes periféricos e comandaria os outros computadores, chamados de nó de processamento ("nodes") [5].

Em "clusters" comerciais – vendidos por uma empresa, e não simplesmente montado com computadores disponíveis – o "hardware" e o "software" são proprietários, sendo que o

"software" é feito para ter um desempenho otimizado no "hardware" específico. Em [5] é citado um exemplo de um cluster construído com 1100 processadores duais Macintosh G5. Este cluster chegara a velocidade de 10 teraflops. Foi construído em aproximadamente 1 mês e teria custado 5 milhões de dólares. Para esta velocidade ser obtida através de um supercomputador, deveriam ter sido gastos entre 100 e 250 milhões de dólares. Além disso, para este supercomputador ficar pronto demoraria alguns anos. Fica então evidenciado que o problema principal deste tipo de máquina é o seu alto custo.

Em relação ao "software" de "clusters", este pode ser fornecido diretamente pelo sistema operacional, ou o que é mais comum atualmente, estar executando em um nível acima deste. Vários sistemas operacionais incluem suporte a sistemas de processamento simétrico (SMP) e o suporte a arquitetura de memórias de acesso não linear (NUMA) já está sendo melhorado[5].

Outras características favoráveis à construção de um "cluster" são:

- alto desempenho;
- escalabilidade;
- tolerância a falhas;
- baixo custo;
- independência de fornecedores [16].

3.1. *Tipos de "cluster"*

3.1.1. "Clusters" de alta performance x "clusters" de alta disponibilidade

Os "clusters" de alta disponibilidade ("High Availability") possuem o objetivo de manter serviços disponíveis pelo maior período de tempo possível. Já os "clusters" de alta performance ("High Performance Computing") possuem o objetivo de fornecer um grande poder computacional que um processador apenas não poderia suprir.

Os "clusters" de alta disponibilidade são mais utilizados em empresas que oferecem serviços pela internet, por exemplo. Neste caso, o objetivo é que o serviço fique disponível de forma ininterrupta, pois caso contrário implicará em prejuízos financeiros.

Os "clusters" de alta performance são mais utilizados em laboratórios de pesquisa e simulação. Um exemplo seriam "clusters" utilizados para a realização de previsões metereológicas e para pesquisas na indústria de petróleo[16].

3.1.2. Quanto à estrutura física

3.1.2.1. "Clusters" simétricos

Neste tipo de arquitetura, os computadores são ligados através de uma rede ou adicionados a uma já existente. Como vantagem, apresenta-se a sua fácil configuração, bastando instalar os computadores, a rede e configurar os "softwares" necessários à rede e ao cluster. Como desvantagem apresentam o gerenciamento e a segurança – que deve ser feita em cada máquina separadamente. Além disso, há também o problema da distribuição e do balanceamento de carga [5]. Um esquema de cluster simétrico é mostrado na figura 3.1.

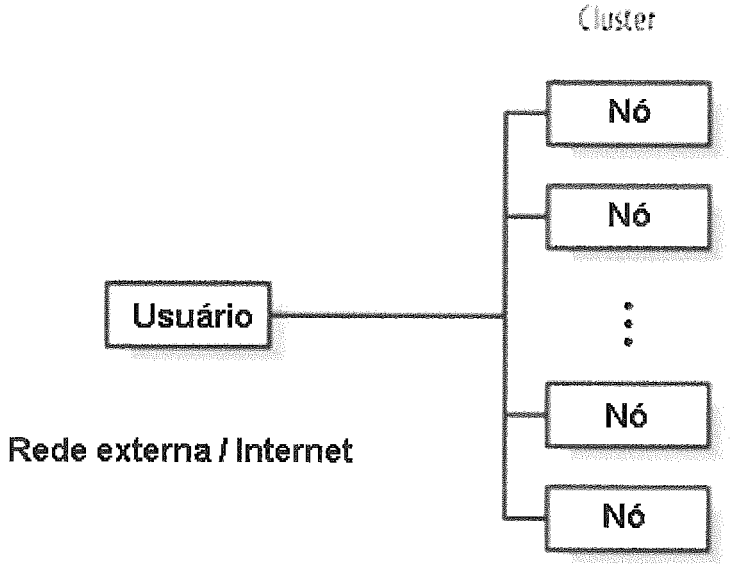

Fig 3.1 Cluster simétrico [5]

3.1.2.2. "Clusters" assimétricos

Neste tipo de arquitetura, um computador fica como controlador ("head") dos outros computadores ("nodes"). A parte do controlador que se comunica para fora da rede interna do cluster é chamada de interface pública, e a parte do controlador que se comunica internamente ao cluster é chamada de interface privada. Como vantagens, apresenta o fato de ser de mais fácil gerenciamento do que a arquitetura simétrica, pois a atualização e o controle das outras máquinas pode ser feita de forma centralizada através do computador controlador. Como desvantagem desta arquitetura, há a limitação de performance imposta pelo controlador principal [5]. Um exemplo de cluster assimétrico é o mostrado na figura 3.2.

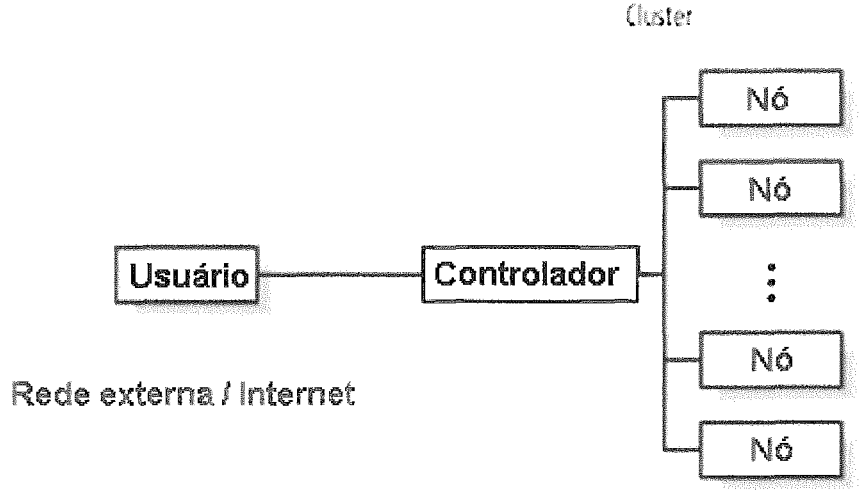

Fig 3.2 Cluster Assimétrico [5]

3.2. *Arquitetura de clusters*

Em 1966, Michael Flynn classificou os sistemas computacionais paralelos segundo o fluxo de instruções e dados. Esta classificação ficou conhecida como taxonomia de Flynn e era composta por 4 arquiteturas de máquinas que serão explicadas a seguir [6].

3.2.1. Instrução única, dados únicos – "Single Instruction, Single Data" (SISD)

Uma máquina deste tipo é a que segue a arquitetura de Von Neummann, conforme mostrada na figura 3.3. Neste, há uma entrada, uma saída de dados, uma memória onde os dados e as instruções são guardados de forma temporária antes de serem processados e uma memória permanente que pode guardar os arquivos (contendo programas ou dados) quando a máquina for desligada. Há ainda uma unidade de controle e outra de operações aritméticas.

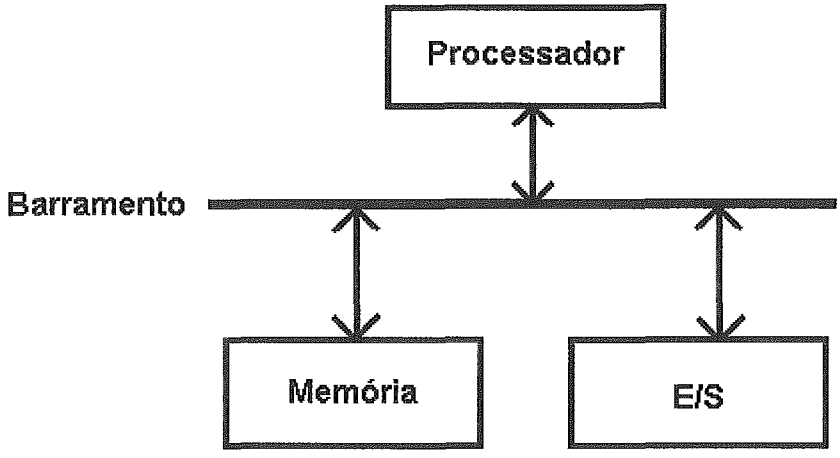

Fig 3.3 Esquema da máquina de Von Neumann [16]

Um programa é composto por dados e instruções, começando sempre por uma instrução. Quando tem que executar, caso encontre-se na memória permanente, é trazido para a memória temporária. A primeira instrução de um programa é obtida pela unidade controladora do processador e então decodificada. A partir daí, o programa pode ser composto por outras instruções ou dados. Uma vez que a instrução está na memória volátil, ela deve ser trazida para memórias mais rápidas chamadas de registradores a fim de poderem ser utilizadas pelos processadores [6].

Há ainda dois princípios a serem observados, o da localidade temporal e o da localidade espacial que dizem o seguinte:

- Princípio da localidade temporal: se um item é referenciado, este item então tende a ser referenciado novamente em um momento futuro próximo [14];
- Princípio da localidade espacial: se um item é referenciado, itens que estão próximos a este tendem a ser referenciados em um momento futuro próximo[14].

Sendo assim, foram adicionadas memórias intermediárias, chamadas de cache, entre os registradores e as memórias voláteis. Estas são mais rápidas do que as memórias voláteis e mais lentas do que os registradores. Neste tipo de memória são armazenadas as instruções e os dados que são mais utilizados. Isto se fez necessário para tentar resolver o seguinte problema encontrado nesta arquitetura: não adianta o processador ficar mais rápido se o tempo necessário para os dados e as instruções chegarem da memória volátil é grande. Com isto, aquele ficava ocioso grande parte do tempo [6].

3.2.2. Instrução única, dados múltiplos – "Single Instruction, Multiple Data" (SIMD)

Máquinas do tipo SIMD possuem uma única unidade de controle e várias unidades lógica aritméticas cada qual com sua própria memória. Em cada ciclo de operação a mesma instrução é passada para todas as unidades lógicas de processamento. Cada processador pode então executar sua instrução de execução de forma síncrona com todos os outros processadores, ou seja, em um determinado tempo, os processadores estão realizando a mesma instrução ou então inativos caso a instrução corrente não seja pertinente a eles (devido a instruções condicionais por exemplo). A desvantagem desta arquitetura são programas com muitas instruções condicionais que podem fazer com que em determinado tempo tenha-se processadores inativos[6; 31].

Alguns exemplos deste tipo de máquina são a CM-1 e a CM-2 "Connection Machines" da "Thinking Machines", e a MP-2 da "Maspar" [6]. Um exemplo de arquitetura SIMD pode ser vista na figura 3.4.

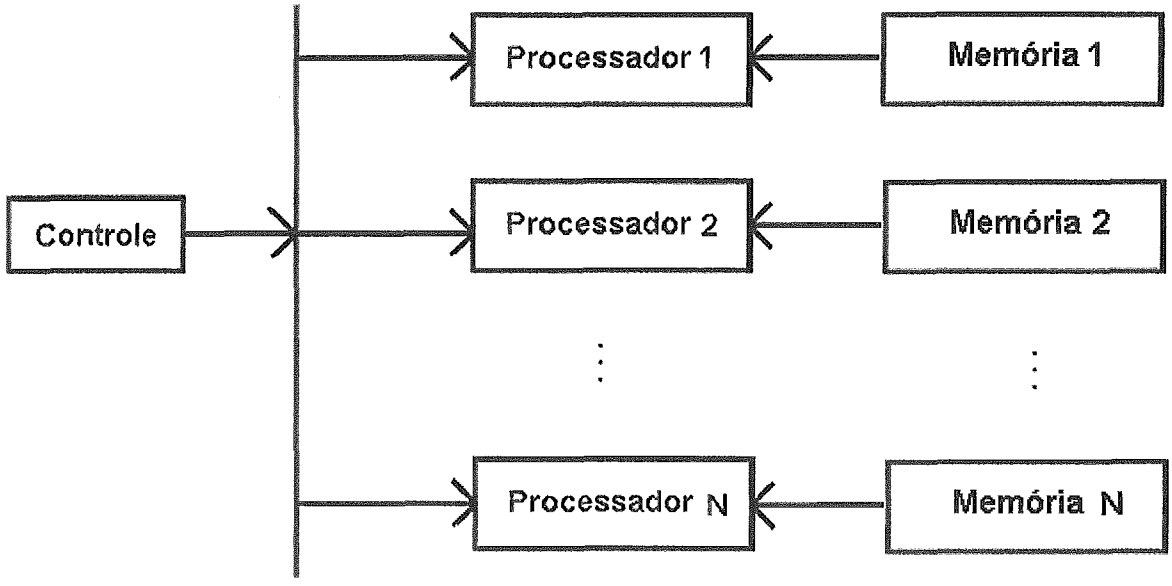

Fig 3.4 Esquema de uma máquina SIMD

3.2.3. Múltiplas instruções, dados únicos – "Multiple Instruction, Single Data" (MISD)

Este arquitetura é formada pelas máquinas vetoriais. Nestas, uma instrução pode ser executada sobre vários dados simultaneamente. Suponha dois arranjos de números, X e Y cada um deles contendo 50 inteiros. No caso das máquinas de Von Neumann, para cada soma de Z[i] = X[i] + Y[i], seria necessário decodificar a instrução, obter os valores da memória, realizar a operação e guardar estes valores na respectiva posição do arranjo Z. Em uma máquina vetorial, com apenas uma instrução, a soma poderia ser feita ao mesmo tempo em todos os elementos do arranjo. A grande vantagem deste tipo de máquina é que já possuem muitos estudos e compiladores bem projetados para tirar vantagem da arquitetura. Entretanto, este tipo de arquitetura não é indicado para resolução de problemas que utilizem estruturas

complexas ou muitos desvios condicionais, pois isso diminui a capacidade de utilização das instruções vetoriais [6].

Alguns exemplos de máquinas vetoriais são o CRAY C90 e o NEC SX4 [6].

3.2.4. Múltiplas instruções, múltiplos dados – "Multiple Instruction, Multiple Data" (MIMD)

Neste tipo de arquitetura, cada processador possui uma unidade de controle e uma unidade lógica aritmética, tornando-os assim máquinas independentes e podendo executar de forma assíncrona. Dividem-se em modelos de memória compartilhada (também chamados de multiprocessados) ou memória distribuída (também chamados de multicomputadores) [6].

3.2.4.1. MIMD de memória compartilhada

Consiste em um conjunto de processadores e módulos de memória conectados entre si por uma rede de computadores, conforme mostrado na figura 3.5. Devido a este fato, podem ser ainda subdivididos em modelo baseado em barramento e baseado em "switches" [6].

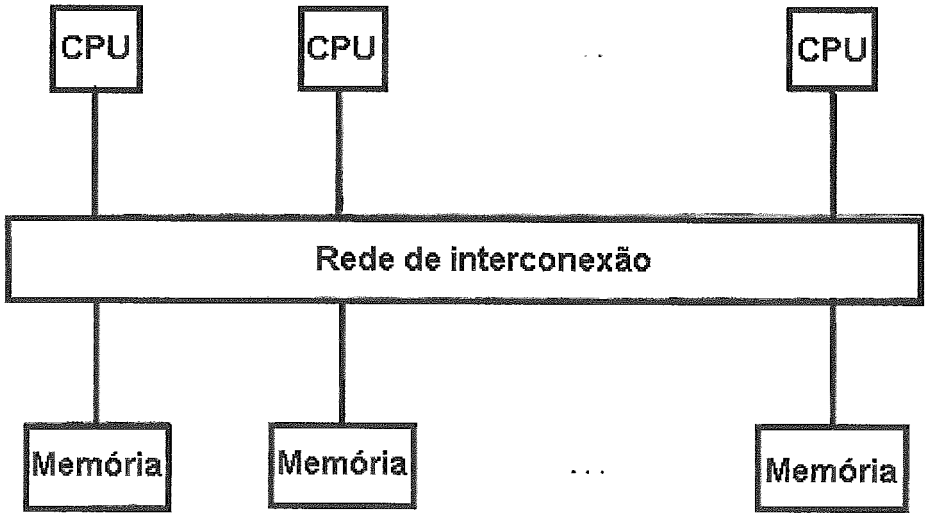

Fig 3.5 Arquitetura MIMD genérica de memória compartilhada [6]

3.2.4.1.1. Baseado em barramento – "bus based"

No modelo baseado em barramento, os processadores são interconectados por um barramento. Como todos os processadores devem usar o mesmo barramento, este pode saturar facilmente. Para tentar resolver este problema, cada processador tem geralmente acesso a uma quantidade razoável de memória cache. Devido a quantidade de dados que podem trafegar pelo barramento de forma simultânea, esta arquitetura não é facilmente escalonável. A maior configuração já montada possuía apenas 36 processadores [6]. Um exemplo de arquitetura MIMD de memória compartilhada baseada em barramento pode ser visto na figura 3.6.

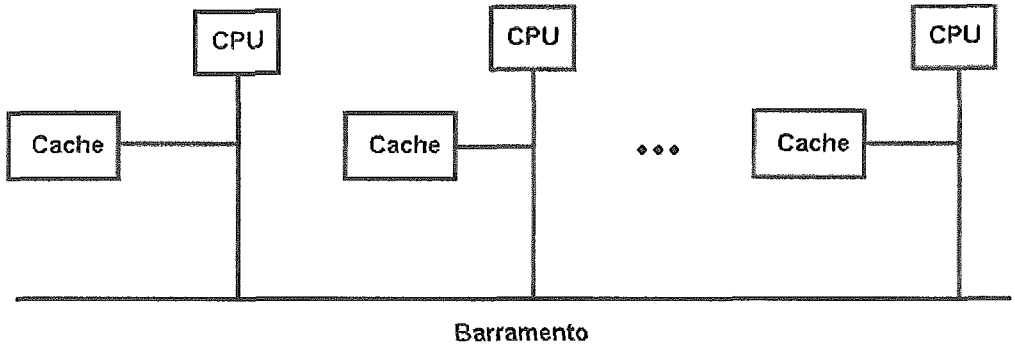

Fig 3.6 Arquitetura MIMD de memória compartilhada baseada em barramento [6]

3.2.4.1.2. Baseado em switches – "switch based"

Em uma arquitetura "switch based", os processadores e os bancos de memória estão interconectados entre si por uma malha formada por "switches" em cada interseção, conforme mostrado na figura 3.7.. Os sinais podem então caminhar tanto na direção horizontal quanto na vertical. Com isso, pode-se ter processadores diferentes acessando bancos de memória diferentes de forma simultânea. Sendo assim, este tipo de máquina não possui o problema de facilmente saturar como no modelo baseado em barramento. Entretanto, esta arquitetura torna-se cara devido a necessidade de colocar um "switch" em cada interseção da malha

formada pelos computadores e pelas memórias. Por isso, máquinas deste tipo geralmente são pequenas (para M processadores e N bancos de memória, seriam necessários M x N "switches"). Ainda neste tipo de arquitetura ocorre um fato que não há no modelo baseado em barramento. O tempo que um processador tem para acessar a memória nem sempre é constante. Memórias que se encontram fisicamente mais próximas do processador (em um mesmo barramento) são acessadas em menos tempo do que memórias que estão mais longe (em um barramento diferente). Devido a isto, este tipo de máquina também é conhecido como máquinas de acesso não linear a memória ("Non Uniform Memory Access" – NUMA) [6].

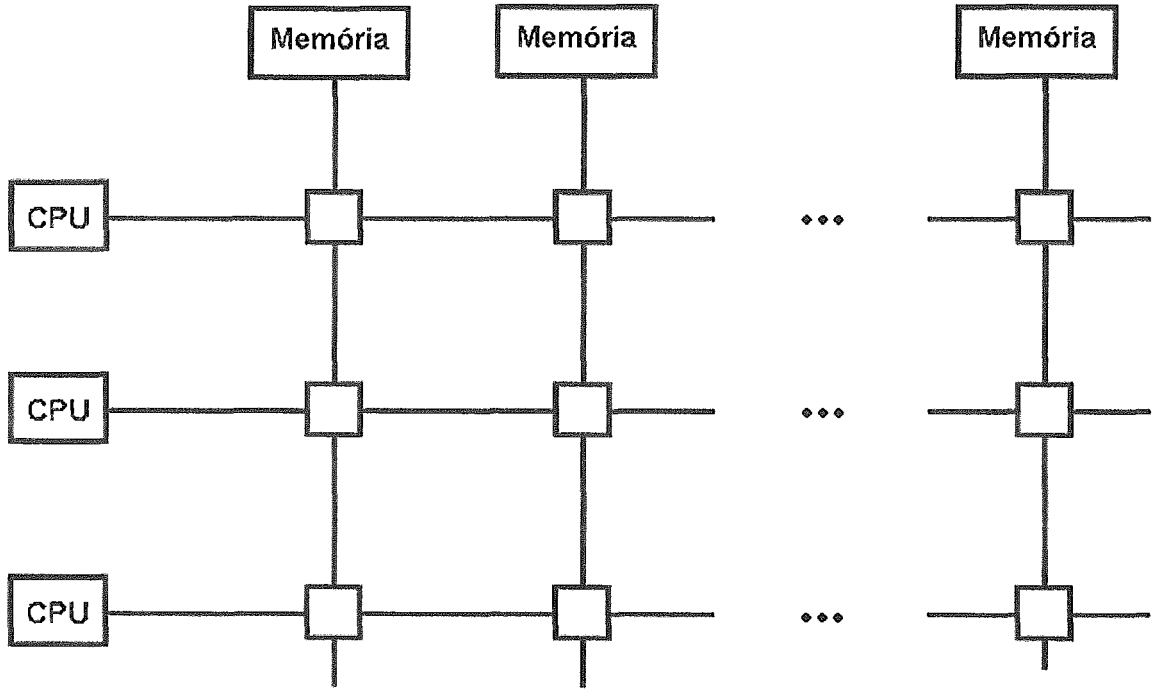

Fig 3.7 Arquitetura MIMD de memória compartilhada baseada em "switches" [6]

3.2.4.2. MIMD de memória distribuída

Em um sistema deste tipo, cada processador possui sua própria memória local. Neste tipo de arquitetura, a configuração que traria melhor performance seria a ligação de todos os processadores entre si. Assim, não haveria atraso de comunicação entre as máquinas pois cada uma poderia se comunicar diretamente com a outra e de forma independente (paralela). O problema para esta configuração é o custo de se montar uma rede com todas as máquinas

conectadas entre si. Esta arquitetura pode ser classificada ainda como de rede estática ou rede dinâmica [6]. Um exemplo de arquitetura MIMD de memória distribuída pode ser visto na figura 3.8.

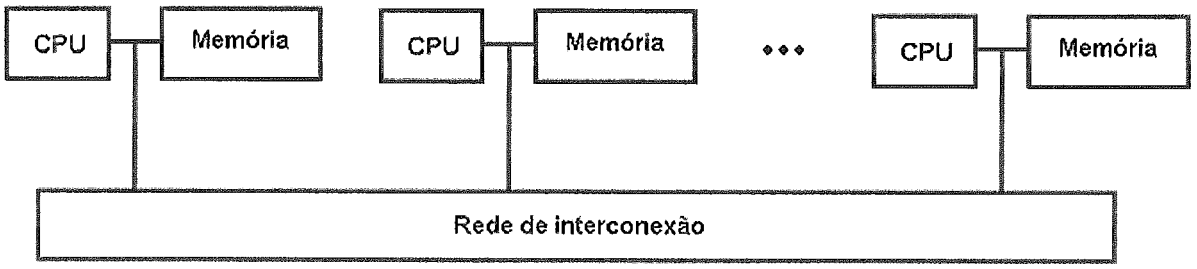

Fig 3.8 Arquitetura MIMD genérica de memória distribuída [6]

3.2.4.2.1. Com rede dinâmica

É uma configuração da arquitetura parecida com o modelo de memória compartilhada MIMD do tipo "switch based". A diferença é que em vez de se ter processadores ligados a bancos de memória através de "switches", há processadores ligados a outros processadores através de uma malha que possui "switches" em suas interseções. Assim, caso o processador 1 tente se comunicar com o processador 2 no tempo t_1, este não irão interferir com os outros processadores que queriam comunicar-se neste mesmo tempo. A desvantagem desta arquitetura é a mesma do modelo MIMD do tipo "switch based". Como cada interseção tem que ter um "switch", esta arquitetura torna-se inviável devido ao seu custo [6]. Esta arquitetura pode ser vista na figura 3.9, onde os quadrados indicam os "switches" e os círculos, o conjunto processador mais memória.

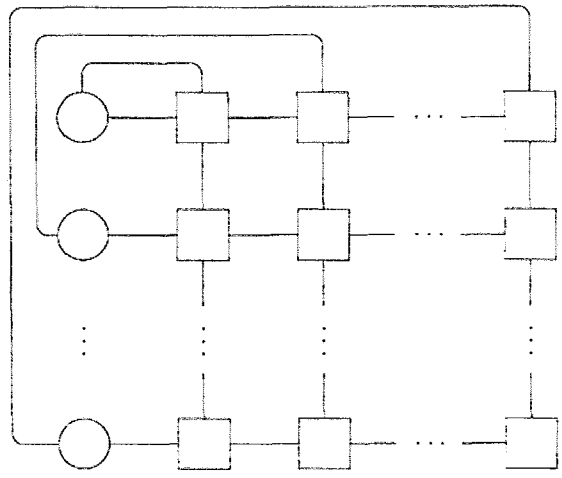

Fig 3.9 Arquitetura MIMD memória distribuída com rede dinâmica [6]

3.2.4.2.2. Com rede estática

Nesta arquitetura, cada dois processadores adjacentes são ligados entre si, conforme mostrado na figura 3.10. Um exemplo deste tipo de configuração é uma rede em anel, no qual o último processador é ligado ao primeiro. Entre as vantagens, pode-se citar o baixo custo (apenas o preço dos processadores e dos cabos de conexão) e a fácil escalabilidade (bastando apenas adicionar mais processadores e cabos). Como desvantagem há o fato que em um determinado momento, apenas 2 nodes podem estar se comunicando tendo que os outros nodes esperar pelo fim desta, não havendo então comunicação paralela [6].

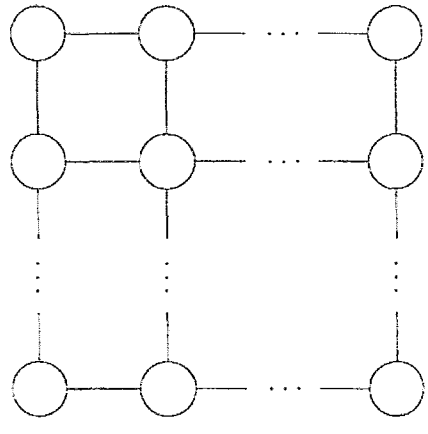

Fig 3.10 Arquitetura MIMD memória distribuída com rede estática[6]

3.3. Histórico

Em 1962, a Burroughs apresenta um multiprocessador simétrico MIMD. Era um cluster MIMD do tipo baseado em "switches", tendo 4 processadores e 16 bancos de memória. Em 1972 a NASA instala o ILLIAC-IV que possuía 64 processadores de 4MFlops e 500Mbits de velocidade de I/O. Em 1977, na Universidade Carnegie Mellon inicia-se um projeto que seria a base para futuras pesquisas em sistemas operacionais e linguagens paralelas na arquitetura MIMD de memória compartilhada e baseada em "switches". Neste cluster foram utilizados 16 minicomputadores PDP-11. Em 1980 a União Soviética construiu um cluster que possuía 64 processadores de 24 bits e chamava-se PS-20000. Em 1982, o Japão, envolvido em um projeto de inteligência artificial tenta construir máquinas paralelas, mas a tentativa foi frustrada. Em 1983, a NASA adquire um cluster de uma empresa chamada Goodyear Aerospace. O cluster se chamava Massively Parallel Processor e possuía 16000 processadores. Em 1983 surge o Connection Machine CM-1 desenvolvido pela Thinking Machine e que possuía 65536 processadores conectados por um hipercubo. Em 1987, a Intel criou o iPSC/2 que possuía 256 processadores da linha 80386/80387 ligados através de um hipercubo. Em 1989, a Cray Research lança o CRAY Y-MP com 8 processadores e poder de processamento de 2,1GFlops. Neste mesmo ano, a Fujitsu lança um processador vetorial, o VP-2600 com 4 GFlops de performance. Em 1993 a Fujitsu instala no Japan's National Aerospace Laboratory ("NAL") uma máquina com 140 processadores e com 124,5GFlops de performance [16]. Em 1994, a NASA necessitava de um equipamento que possuísse um processamento da ordem de um gigaflop. Este, custaria 1 milhão de dólares na época. Os pesquisadores Thomas Sterling e Donald J. Becker interligam 16 computadores pessoais, com processadores Intel 486, sistema operacional Linux e rede Ethernet. Com isso, construíram o cluster Beowulf, que possuía o poder de processamento de 70 megaflops [16]. Em 1994 e 1995, as melhores máquinas da época segundo [35;36] continuavam a serem produzidas pela

Fujitso, que lança o Numerical Wind Tunnel/140, possuindo um poder de processamento de 170,4GFlpos. Em 1996 [37], a melhor máquina era a CP-PACS/2048/2048 fabricada pela Hitachi. Possuía um poder computacional de 368,2GFlops. Em 1997 [38] a Intel ultrapassou a barreira de 1TFlop com o ASCI Red / 9152. Em 1998 [39] o ASCI Red da Intel, com os seus 1,388TFlpos continuava sendo a máquina com maior poder computacional. Em 1999 [40] a Intel lançou uma nova versão de sua máquina ASCI Red, a versão 9632 que atingia 2,379TFlops. Em 2000 [41], a IBM lançou o ASCI White, SP Power3 375 MHz / 8192. Esta máquina possui um poder computacional de 4,938TFlops. Em 2001 [42], a IBM fez com que o seu ASCI White, SP Power3 375Mhz / 8192 chegasse a 7,226TFlops. Em 2002 [43], o NEC Earth Simulator atingiu a velocidade de 35,86TFlops, utilizando 5.104 processadores. Este computador é utilizado no Japão para a realização de simulações a respeito de clima e terremotos. Em 2003 [44] o Earth Simulator continuava sendo a máquina mais potente. Em 2004 [45] a máquina com maior poder computacional foi a IBM Blue Gene/L DD2 beta-system que chegava a 70.720TFlops. Em 2005, novamente a IBM produziu a máquina mais potente, a BlueGene/L, que possuía 131.072 processadores e 280.6TFlops [34].

3.4. MPI

3.4.1. Histórico

Até 1992, várias implementações de bibliotecas para comunicação paralela existiam. Entretanto, elas geralmente eram voltadas para um "hardware" específico, não havendo portabilidade entre elas [7].

Neste ano, criou-se um fórum de discussão de interface de troca de mensagens ("Message Passing Interface Forum"), do qual participavam 80 pessoas e 40 empresas, além de laboratórios de pesquisa e universidades. Este fórum teve duração até 1994 e tinha como objetivo criar um padrão que pudesse ser utilizado em um grande número de máquinas

diferentes[7]. Em vez de criar uma nova linguagem para isso (como tinha sido feito com o Fortran, no qual extensões para a linguagem foram adicionadas a fim de permitir o processamento paralelo), este fórum preferiu criar uma biblioteca de funções que inicialmente foi pensada em termos das linguagens C e Fortran. Mais tarde foi adicionada a versão para a linguagem C++. A base para esta biblioteca foi um pequeno grupo de funções que poderiam ser utilizadas para obter o paralelismo necessário para a troca de mensagens. Após o esforço inicial a biblioteca recebeu um grande número de funções [6]. Os criadores da interface de passagem de mensagens resolveram adotar vários paradigmas que já tinham apresentado sucesso nas implementações proprietárias que existiam na época. Entre elas podem ser citadas [7]:

- portabilidade, sendo esta a maior vantagem da interface de passagem de mensagens;
- execução transparente em sistemas heterogêneos;
- performance, pois de nada adiantaria todas as vantagens oferecidas se os sistemas se tornassem demasiadamente lentos;
- tentativa de diminuir informações e procedimentos desnecessários ("overhead"), como por exemplo não ter codificações e decodificações complexas para cabeçalhos de mensagens;
- escalabilidade que pode ser atingida de várias maneiras pela biblioteca;
- definir um comportamento mínimo para a troca de mensagens, permitindo que se necessário, novas bibliotecas possam ser construídas tomando esta como base e facilitando ainda mais o trabalho dos desenvolvedores.

3.4.2. Modos de comunicação

3.4.2.1. Padrão

Neste modo, o MPI, dependendo de sua implementação, pode escolher se a mensagem que irá ser transmitida será buferizada. Se isto ocorrer, a operação de envio – realizada com a função MPI_Send – pode retornar antes que uma função de recebimento – MPI_Recv – tenha sido executada. Entretanto, há situações em que o MPI não irá buferizar o envio da mensagem. Isto pode acontecer pelo fato de não haver espaço disponível para buferização ou por o MPI determinar que para manter uma boa performance, não deve se comportar desta

maneira. Se estas condições ocorrerem, o MPI_Send será bloqueante, ou seja, a função só retornará para o código principal quando um MPI_Recv tiver sido executado no destino e os dados transferidos.

Sendo assim, uma operação de envio pode ser iniciada, neste modelo de comunicação, independente de uma operação de recebimento ter sido executada, sendo que pode até mesmo ser completada antes que uma operação de recebimento tenha sido comandada [7].

Como comando não-bloqueante do MPI pode ser citado MPI_Send [7].

3.4.2.2. Buferizado

O modo de operação buferizado é não bloqueante, ou seja, uma operação de envio pode ser executada independente de uma operação de recebimento ter ser sido comandada, e até mesmo retornar antes que uma operação de recebimento tenha sido executada no processo destino. Neste modo, a aplicação cuida da alocação dos buffers necessários, sendo que se não houver espaço suficiente para sua alocação uma mensagem de erro será emitida. A liberação do buffer ocorre quando a mensagem é recebida ou a comunicação é cancelada [7].

Como comandos buferizados do MPI podem ser citados: MPI_Bsend e MPI_Ibsend, sendo este último não bloqueante[7].

3.4.2.3. Imediato

Neste modo de funcionamento, uma operação de envio pode ser executada apenas se uma operação de recebimento já tiver sido executada na frente e estiver esperando este envio. Se isto não acontecer, ocorrerá um erro e o resultado da operação será indefinido [7].

Como exemplo de comando imediato, pode-se citar MPI_Rsend e MPI_Irsend, sendo este último não bloqueante [7].

3.4.2.4. Síncrono

Neste modo de funcionamento, uma operação de envio pode ser iniciada independente de uma operação de recepção ter sido iniciada. Entretanto, a finalização do envio só obterá sucesso se uma operação de recebimento tiver sido comandada para receber esta mensagem. Na finalização destas operações, o buffer poderá ser reutilizado[7].

Como exemplo de um comando síncrono, pode-se citar MPI_Ssend e MPI_Issend, sendo este último não bloqueante[7].

3.4.3. Tipos de comunicação

3.4.3.1. Ponto-a-ponto

Consiste na comunicação entre 2 processos de um mesmo comunicador [31].

3.4.3.2. Coletiva

Consiste na comunicação entre vários processos de um mesmo comunicador de forma simultânea entre si. Neste tipo de comunicação não há tags pois todos os processos que estiverem envolvidos recebem e/ou enviam os dados, além de todas estas instruções serem do tipo bloqueante[31]. Há três tipos de comunicação coletiva:

- Movimentação de dados → MPI_Bcast por exemplo;
- Computação coletiva → MPI_Reduce por exemplo;
- Sincronização → MPI_Barrier por exemplo[31].

3.5. Modelos de programação

3.5.1. Modelo Mestre/Escravo

Neste modelo de programação, a computação é dividida entre os processos. Escolhe-se um processo principal, geralmente o processo com posto 0 quando se utiliza o MPI, que irá particionar o problema em subproblemas a serem resolvidos pelos outros processos. Este será o chamado processo mestre. Quando os demais processos, chamados de processos escravos, acabarem de realizar suas computações, retornam os seus resultados ao processo mestre, que pode utilizar os dados retornados por exemplo. À medida que um processo escravo for acabando sua computação, se houver mais trabalho a ser realizado, o processo mestre pode delegar mais tarefas para este processo escravo [5]. Este fluxo continua até não haver mais computação a ser realizada pelos processos escravos. Um esquema de um modelo de programação Mestre/Escravo pode ser visto na figura 3.11.

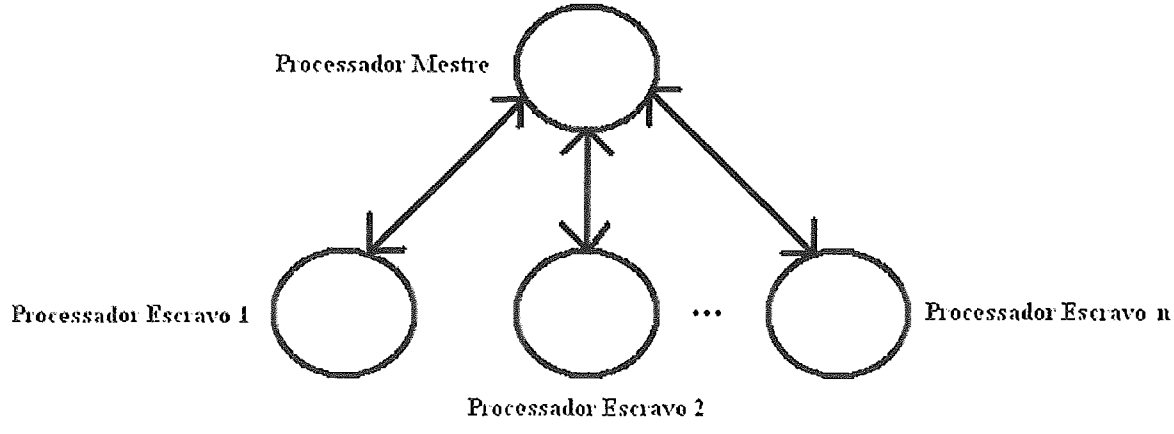

Fig 3.11 – Exemplo de modelo Mestre/Escravo de programação com n escravos

3.5.2. Modelo Ilha

Neste modelo, cada processo é comparado a uma ilha. Cada ilha possui uma versão do algoritmo sequencial. Em um determinado período, as ilhas comunicam-se trocando os melhores valores obtidos em cada uma[12]. Um exemplo de modelo ilha de programação pode ser visto na figura 3.12. Neste caso, o processo 1 poderia se comunicar diretamente com os processos 2 e 4 por exemplo.

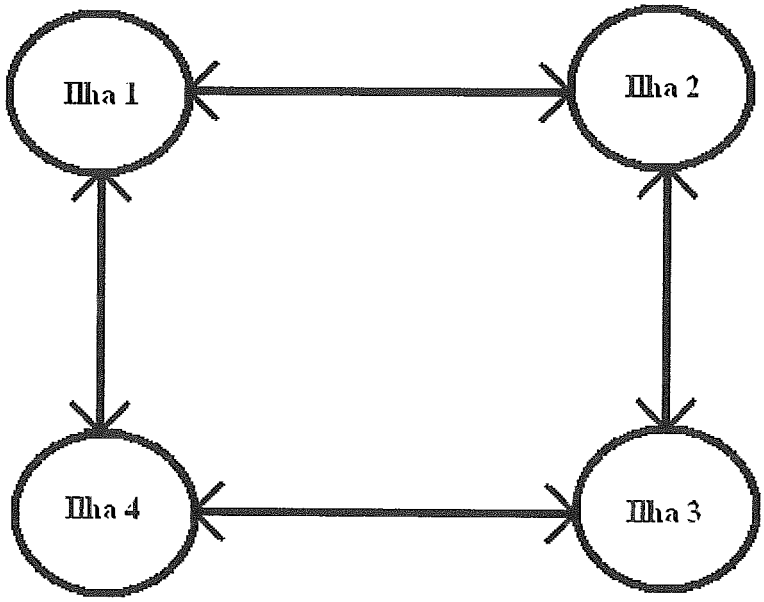

Fig 3.12 – Exemplo de modelo Ilha de programação, com 4 ilhas

4. PSO

4.1. Histórico

Um grande número de pesquisadores têm estudado o movimento dos pássaros, e tentado simulá-los de forma computacional. Entre estes, Reynolds em [19], e Heppner & Grenander [20] apresentaram resultados. As pesquisas de Reynolds direcionavam-se para a "coreografia" realizada pelos pássaros durante o vôo, enquanto Heppner & Grenander estavam interessados nas regras que regiam estes movimentos:

- Como os pássaros sabiam para onde ir?
- O que os fazia repentinamente mudar de direção?
- Como não se chocavam entre si durante o vôo?
- Como se separavam e se reagrupavam [20]?

Algumas das regras que governam os comportamentos de animais que vivem de forma coletiva como cardumes, bandos, colméias e colônias de formigas poderiam se aplicar ao comportamento social do ser humano. Em [21], Wilson escreveu:

"ao menos teoricamente, os indivíduos de um cardume podem ser beneficiados das descobertas e da experiência passada de todos os membros do cardume durante a busca por alimentos" [...]

Entretanto, a principal diferença entre pessoas e animais é que duas pessoas podem ter as mesmas crenças e convicções sem estarem fisicamente no mesmo local. Isto não ocorre com os animais pois estes não possuem crenças e nem convicções. Se fosse usado para simular o comportamento dos seres humanos, os indivíduos representados no algoritmo não estariam mais limitados ao espaço tridimensional. Cada dimensão poderia ser uma crença ou atitude e com isso o algoritmo seria utilizado para estudar o comportamento de pessoas[3].

O compartilhamento de informações entre os indivíduos da mesma espécie oferece uma vantagem evolucionária sendo este o principal destaque do PSO. Enquanto que em outras

heurísticas de computação evolucionária, como nos algoritmos genéticos, os indivíduos competem entre si podendo até mesmo não estar presente em uma próxima iteração através de mecanismos como a seleção, por exemplo[22], no PSO os indivíduos cooperam entre si para que todos possam se desenvolver de forma melhor em uma próxima iteração [3]. Há dois fatores bem caracterizados no algoritmo: o primeiro é a própria experiência anterior do indivíduo, enquanto que o segundo é a experiência de seus vizinhos na população. Fazendo uma analogia com seres humanos, o primeiro fator é a memória que cada pessoa possui sobre os fatos e acontecimentos pelos quais passou, chamada de "simple nostalgia" [3], enquanto que o segundo é a troca de informações com outras pessoas, como uma conversa por exemplo[4].

Em 1995, James Kennedy & Russel C. Eberhart publicam um artigo a respeito do desenvolvimento de uma nova heurística baseada no movimento dos pássaros chamada de otimização baseada em enxame de partículas ("Particle Swarm Optimization" – PSO). Neste artigo [3], eles explicam como o algoritmo surgiu e se modificou até chegar à fase atual. Além disso, neste trabalho explicam como adaptaram o algoritmo para a forma multidimensional e que este poderia ser utilizado em problemas de otimização. Um exemplo foi dado ao se treinar uma rede neural utilizando PSO ao invés de backpropagation.

Em 1997, Kennedy realiza experimentos com o algoritmo e propõe 3 adaptações tendo sido utilizados 4 modelos:

- Modelo clássico → neste tipo de modelo, ambos os componentes cognitivo e social estão presentes no cálculo da velocidade de cada dimensão da partícula atual;
- Modelo individualista ("cognitive only model") → neste modelo, apenas o componente cognitivo está presente no cálculo da velocidade de cada dimensão da partícula atual;
- Modelo coletivo ("social only model") → neste modelo, apenas o componente coletivo está presente no cálculo da velocidade de cada dimensão da partícula atual;
- Modelo "selfless" → neste modelo, a própria partícula não pode ser escolhida como partícula global de sua vizinhança, pois isso acarreta que ela seria influenciada por si

própria tanto na componente individual quanto na componente coletiva do cálculo da velocidade.

O problema estudado foi o treinamento de uma rede neural. Os resultados indicaram, que para este tipo de problema [17], a adaptação que obteve melhor resultado foram as seguintes nesta ordem de importância:

- Modelo coletivo;
- Modelo "selfless";
- Modelo clássico;
- Modelo individualista.

Kennedy começa a analisar a variável Vmax, utilizada para não permitir que as partículas possam ganhar velocidade ilimitada e extrapolar do espaço de soluções inicial. Se Vmax for muito grande, favorece uma exploração do espaço de soluções. Se for pequeno, a busca será feita minuciosamente em uma área menor do espaço de busca[17].

Em 1997, Kennedy & Eberhart publicaram [13] que descreve uma versão binária de PSO. A diferença desta versão para a que trabalha com números reais consiste apenas na atualização da posição de cada dimensão de cada partícula, e da inicialização das partículas. Na versão binária, as partículas só podem assumir o valor 0 ou 1 para as posições de cada dimensão. Quando a dimensão for atualizada, o valor obtido pela equação da velocidade será usado como parâmetro em uma função sigmóide que retorna um número entre 0 e 1[24]. Um arranjo é preenchido com números entre 0 e 1 fornecidos por um gerador de números pseudo-aleatórios com distribuição uniforme. Caso o valor resultante da função sigmóide seja menor do que o valor deste arranjo na posição correspondente àquela dimensão, a posição da partícula atual na dimensão receberá o valor 1. Se isto não acontecer, receberá o valor 0 [13]. Tendo sido testado nas funções F1 a F5 de DeJong, o algoritmo obteve bons resultados [13].

Em 1998 [25], Angeline faz uma adaptação ao PSO para que este utilize um mecanismo de algoritmos evolutivos: a seleção. Esta foi realizada da seguinte forma:

- escolhe-se um conjunto de partículas;
- executa-se um ranqueamento. O valor da função objetivo de cada indivíduo, baseada na sua posição atual na população é comparado ao valor da função objetivo de outros n indivíduos. O indivíduo que obtiver o pior valor para a função objetivo recebe uma pontuação. Quando todos os indivíduos estiverem pontuados, faz-se uma ordenação baseada no número do "rank";
- há agora um conjunto de partículas com "rank" bom e outro com "rank" ruim. Cada partícula que não possui um "rank" bom recebe a posição e a velocidade atual de uma partícula com "rank" bom. Entretanto, a posição desta partícula onde ela obteve o melhor valor da função de avaliação continua a mesma;
- o algoritmo continua sua execução normalmente [25].

Desta forma, em cada iteração, metade dos indivíduos serão conduzidos à posições que são melhores do que as posições em que estão atualmente. Angeline também estudou a forma de inicialização do algoritmo. Se previamente já se sabe que a resposta do problema está próxima do ponto 0,0 de um espaço de soluções – neste caso bidimensional por exemplo – caso as partículas sejam inicializadas de forma simétrica, isto pode influir para que o algoritmo convirja mais rápido. Muitos problemas, entretanto, não possuem esta característica de sua solução ótima estar próxima a um determinado ponto previamente conhecido. De forma a investigar se este fato realmente influenciava no algoritmo, Angeline fez um experimento com inicialização simétrica perto da origem e um outro experimento que limitava a população a uma localização restrita do espaço de buscas. Como resultado, a performance do modelo híbrido – PSO + seleção – foi afetada levemente pela inicialização assimétrica. A versão clássica também variou muito pouco independente do método de inicialização utilizado. O método híbrido também não se mostrou eficiente para todas as funções testadas, podendo citar a baixa performance para a resolução da função de Griewank [25].

No PSO, as partículas estão sempre alterando sua posição de acordo com sua velocidade. As mudanças nas velocidades de cada dimensão ocorrem de forma estocástica. Um efeito

indesejado deste fato é que a trajetória da partícula acabaria sem controle se expandindo para regiões muito distantes do espaço de busca, tendendo ao infinito. Para realizar uma busca com sucesso, algum controle deve ser feito no sentido de não deixar que este fato ocorra. A forma inicial de impedir que as partículas se afastassem demasiadamente pelo espaço de buscas, podendo até mesmo atingir o infinito, foi controlando a sua velocidade através das equações abaixo:

$$\text{Se } V_{id} > Vmax \text{ então } V_{id} = Vmax \quad (3.1)$$
$$\text{Senão se } V_{id} < -Vmax \text{ então } V_{id} = -Vmax \quad (3.2)$$

Onde Vid é a velocidade na dimensão d da partícula i, e Vmax é um limite imposto a velocidade. O efeito de Vmax é prevenir a explosão do sistema e escalonar a exploração das partículas. O problema deste parâmetro é que para ele ser bem ajustado, algum conhecimento prévio a respeito do problema sendo solucionado deve ser obtido. Se um valor baixo de Vmax for selecionado por exemplo, pode ser que a partícula nunca saia de uma faixa de buscas. Caso o valor ótimo procurado esteja fora desta faixa, então ele nunca será encontrado [4].

Em 1998, Kennedy [26] construiu gráficos da trajetória das partículas com intuito de entender como estas se comportavam no espaço de busca. Para chegar nestes gráficos, Kennedy realizou as seguintes considerações, apenas para efeito de estudo, pois na prática elas não ocorrem [4]:

- apenas 1 partícula;
- partícula com apenas 1 dimensão;

Com isso, a partícula poderia ser considerada:

$$p = (\varphi_1 . p_i + \varphi_2 . p_g) / (\varphi_1 + \varphi_2) \quad (3.3)$$
$$\varphi = \varphi_1 + \cdot 2 \quad (3.4)$$

fazendo p constante em vez de dinâmico, a trajetória da partícula poderia ser estudada. As fórmulas para o PSO para velocidade e distância respectivamente seriam:

$$v = v + \varphi . (p - x) \quad (3.5)$$
$$x = x + v \quad (3.6)$$

onde φ é um número randômico definido apenas por um limite superior e os outros valores são escalares. A figura 4.1 mostra como o sistema se comporta no caso de não haver Vmax.

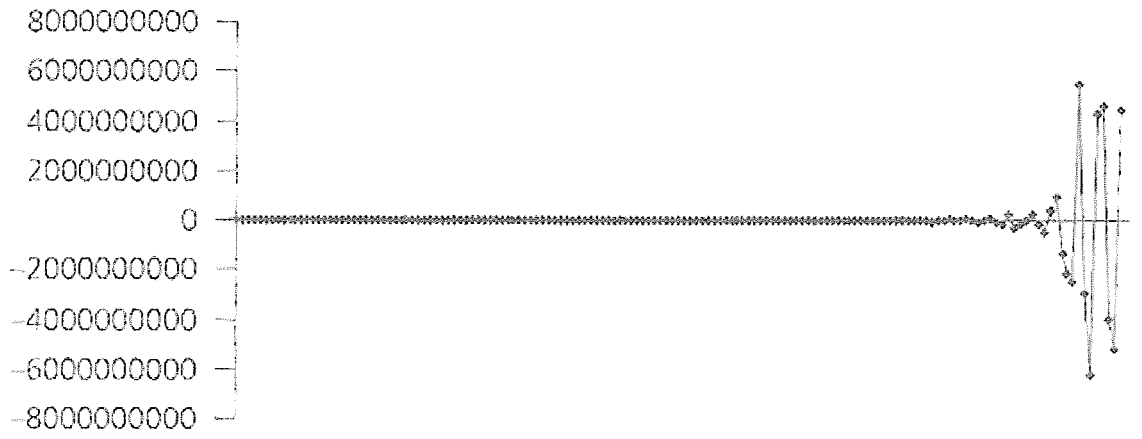

Fig 4.1 Execução do algoritmo sem o parâmetro Vmax. A velocidade rapidamente explode para além da região de interesse [4].

A figura 4.2 mostra que a explosão é evitada colocando-se uma variável Vmax no sistema.

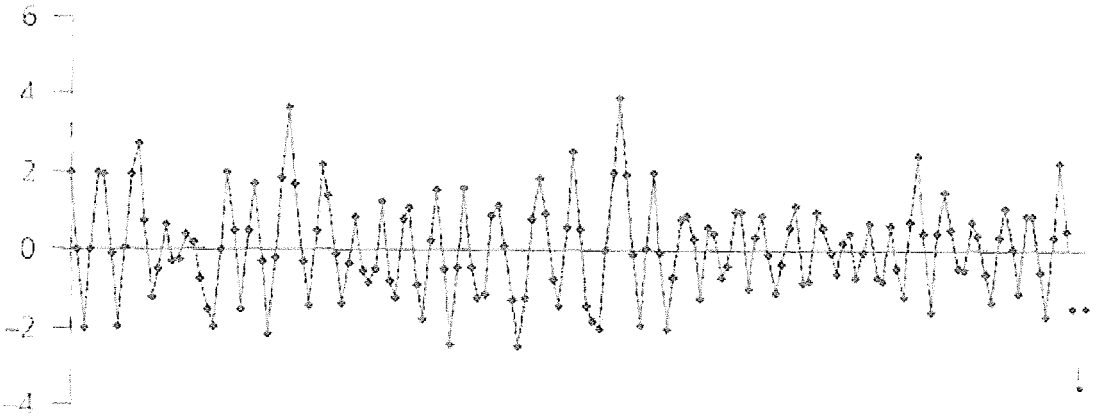

Fig 4.2 Execução do algoritmo com um valor de Vmax definido. A partícula limita seu espaço de busca, sem entretanto chegar no valor ótimo [4].

A figura 4.3 mostra que se Vmax for diminuído, o espaço de busca que a partícula poderá percorrer para tentar achar o ponto ótimo ficará mais estreito.

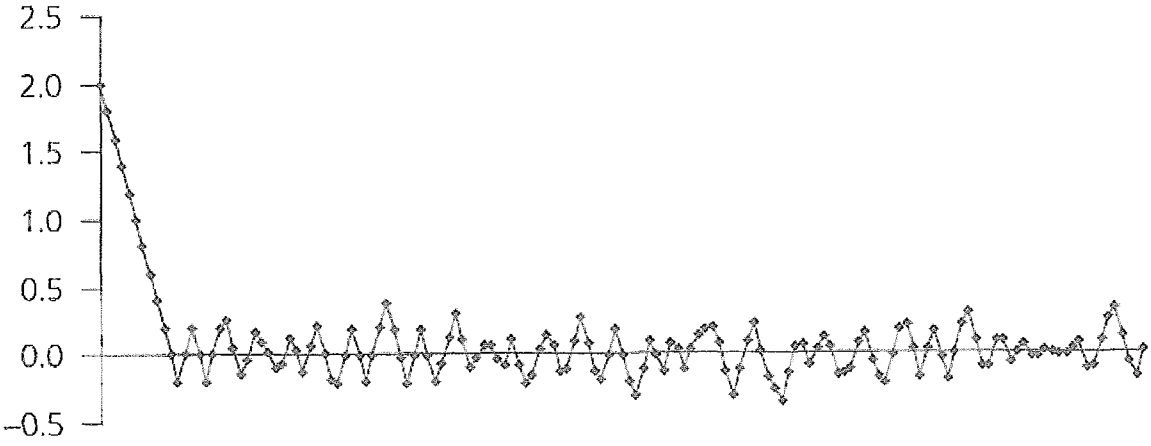

Fig 4.3 O valor de Vmax nesta figura é 1/10 do Vmax da figura 4.2. O espaço de busca que a partícula pode se movimentar fica mais estreito [4].

Em 1999, Ozcan & Mohan [27] estudam o comportamento de um sistema com apenas 1 partícula e 1 dimensão, parecido como estudado por Kennedy [26]. Clerc & Kennedy [28], obtiveram um resultado que ficou conhecido como coeficiente de constricção. Este possui a característica de impedir a explosão do sistema e auxiliar na convergência do mesmo. A fórmula de velocidade do PSO deve ser alterada para:

$$V_{id}(t) = K * [V_{id}(t-1) + \varphi_1 (P_{id} - X_{id}(t-1)) + \varphi_2 (P_{gd} - X_{id}(t-1))] \qquad (3.7)$$

onde K é dado por:

$$K = 2 / (| 2 - \varphi - \mathrm{sqrt}(\varphi^2 - 4\varphi) |) \qquad (3.8)$$

sendo que sqrt significa a raiz quadrada, e φ deve ser igual à soma de c1 e c2, e maior ou igual à 4.

Em 1999, Eberhart & Shi[29] criam o coeficiente de inércia. Este faz com que a fórmula da velocidade do PSO assuma a seguinte forma:

$$V_{id}(t) = (w * V_{id}(t-1)) + \varphi_1 (P_{id} - X_{id}(t-1)) + \varphi_2 (P_{gd} - X_{id}(t-1)) \qquad (3.9)$$

O coeficiente de inércia na fórmula (3.9) é representado pelo valor de w que multiplica a velocidade da partícula atual, na dimensão atual, no presente momento, dada por $V_{id}(t-1)$.

Este trabalho obteve resultados apenas para a função Schaffer F6 pois esta foi a única a ser utilizada. Concluiu-se que o coeficiente de inércia permite uma exploração inicial melhor do que a que se obtém caso se utilize apenas Vmax. Sugere ainda que Vmax não seja utilizado, pois é necessário algum conhecimento prévio do problema. Caso isto não ocorra, Vmax pode ser estipulado de forma a não permitir que as partículas alcancem o valor ótimo. Introduziu-se ainda um valor decrescente de coeficiente de inércia variando entre 0,9 e 0,4. Quando não houver informações sobre a função a ser otimizada, indicaram que o coeficiente de inércia deveria ser igual a 0.8 e Vmax igual a Xmax.

Em 2000, Eberhart & Shi [32] fizeram uma comparação entre os dois métodos que impedem a explosão do sistema. Foram utilizadas algumas funções para teste. Nas funções esférica, Rosenbrock e Schaffer F6, o coeficiente de constricção faz com que haja uma convergência mais rápida, porém há também uma maior variabilidade dos resultados. Na função de Rastringin, houve um caso em que não ocorreu convergência utilizando-se o coeficiente de constricção. Na função de Griewank, admitindo-se 0,05% de erro e utilizando-se o coeficiente de constricção, o sistema não convergiu em 3 dos 20 casos. Se o erro for elevado para 0,1%, não há convergência em 15% dos casos utilizando-se o coeficiente de constricção. Foi observado que apesar de convergir mais rápido quando isto ocorre, o coeficiente de constricção possui uma variação muito maior do que o coeficiente de inércia. O coeficiente de constricção possui uma área de exploração maior do que o coeficiente de inércia, por isso sua maior variabilidade nos resultados. A fim de tentar evitar este fato, adiciona-se ao sistema uma variável chamada Xmax que é a distância máxima que a partícula pode percorrer. Com isto, apesar de a partícula poder atingir grandes distâncias e afastar-se do ponto ótimo, ela irá se afastar de um valor máximo conhecido, ou seja, ficará nas

proximidades do valor ótimo. Para isto, é necessário um conhecimento prévio do problema, pois o valor ótimo deve estar na faixa compreendida entre –Xmax e +Xmax. Novos testes foram realizados com as funções e o algoritmo utilizando o coeficiente de constricção. Desta vez entretanto, o valor de Vmax foi igualado ao valor de Xmax. Os resultados em todos os casos apresentaram uma melhora.

Em 2003, Schutte et al. [33], desenvolveram uma versão de PSO que funcionava utilizando o modelo mestre-escravo para tentar resolver um problema de simulação de movimentação de tornozelo. Neste modelo, o cálculo da função objetivo foi passado para várias partículas, cada uma executando em um processador diferente, pois os cálculos eram independentes entre si, ou seja, admitiam paralelismo. Com isso, o tempo necessário para calcular todas as funções objetivo ficava limitado ao maior tempo gasto para calcular uma função objetivo (a mais lenta). Este fato fica evidenciado pois para o algoritmo funcionar, precisa do valor das funções objetivo de todas as partículas. O processador mestre ficou em estado de espera (através de uma função MPI_Barrier do MPI) até que todos os processadores tivessem devolvido a resposta do cálculo das funções objetivo.

Os resultados obtidos neste trabalho indicam que o tempo necessário para a resolução deste problema foi substancialmente diminuído [33].

4.2. *O algoritmo*

O algoritmo consiste em um conjunto de partículas andando em um espaço multidimensional. Cada posição da partícula em uma determinada dimensão é utilizada para computar uma função objetivo que indica o quanto a partícula está perto de atingir o seu objetivo. As partículas possuem velocidade para cada dimensão além de uma memória auxiliar chamada de "simple nostalgia" [3] que guarda as posições em cada dimensão da

localidade em que a partícula atingiu o melhor valor para a função objetivo até a função atual.

O movimento das partículas em cada iteração é feito baseado nas equações abaixo:

$$V_{id}(t) = V_{id}(t-1) + \varphi_1 (P_{id} - X_{id}(t-1)) + \varphi_2 (P_{gd} - X_{id}(t-1)) \qquad (4.1)$$

Onde:

- $V_{id}(t)$ é a velocidade na dimensão d da partícula i no tempo t;
- X_{id} é a posição atual na dimensão d da partícula i;
- P_{id} é a posição onde foi obtido o melhor valor para a função objetivo na dimensão d da partícula i;
- P_{gd} é a posição da melhor partícula da população – ou da vizinhança – para a dimensão d;
- φ_1 e φ_2 são componentes aleatórias chamadas respectivamente de coeficiente de aprendizagem individual e coeficiente de aprendizagem coletiva.

Para variar a posição das partículas, é utilizada a equação abaixo:

$$X_{id}(t) = X_{id}(t-1) + V_{id}(t) \qquad (4.2)$$

Onde:

- Vid(t) é a velocidade na dimensão d da partícula i no tempo t;
- Xid(t) é a posição atual na dimensão d da partícula i no tempo t.

Assim, cada partícula se movimenta no espaço multidimensional baseada na posição em que obteve a melhor avaliação da função objetivo e na posição da partícula global [4].

4.2.1. Topologia de vizinhança

A partícula global pode ser escolhida como sendo a partícula que possui o melhor resultado da função objetivo entre todas as partículas da população, ou pode ser escolhida como a partícula que possui o melhor resultado da função objetivo entre um grupo específico de partículas que compõem a vizinhança da partícula atual. Existem várias topologias para vizinhança, sendo que algumas delas são:

- anel → nesta topologia, vista na figura 4.4, cada partícula interage com os k vizinhos mais próximos. Admite algumas variações como vizinhos apenas à direita, apenas à esquerda ou uma parte de vizinhos a direita e outra a esquerda[4]. Ainda neste modelo, a própria partícula pode ou não participar da vizinhança na escolha da partícula global. Caso não participe, esta configuração será chamada de modo "selfless"[17];
- "wheel" → compara-se a algumas hierarquias governamentais pois todas as informações têm que ser passadas para apenas um responsável que irá

filtrá-las e repassar esta informação para as outras partículas[4]. Esta topologia está representada na figura 4.5.

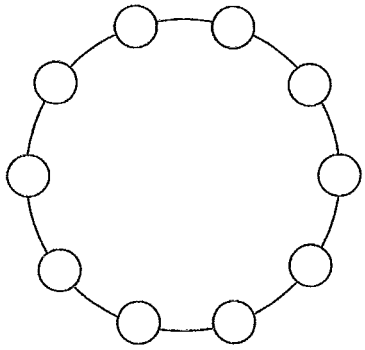

Fig 4.4 Topologia em anel [4]

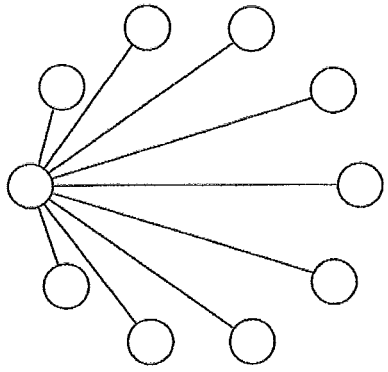

Fig 4.5 Topologia "wheel" [4]

Em [18] foi realizado um experimento com populações de 20 partículas seguindo estas duas topologias. Na topologia em anel, partes deste que estão distantes de outras partes são independentes. Sendo assim, se uma parte do anel ficar presa a um mínimo ou máximo local, outras partes poderão continuar a busca recebendo uma influência pequena deste fato. Entretanto, quando um máximo global é achado, ele é repassado a todo o anel.

A topologia em "wheel" isolava os indivíduos uns dos outros pois toda a informação era comunicada na população por apenas uma partícula. A partícula responsável pelas trocas de informação verificava os valores recebidos de cada partícula e dentre estes escolhia o melhor. Após isso, atualizava todas as partículas com este valor. Sendo assim, esta partícula fazia o papel de um filtro retornando apenas a melhor informação entre todas as recebidas. Como apenas uma partícula realiza este trabalho, há uma espera dos indivíduos até que esta

informação esteja disponível tornando o algoritmo mais lento. Entretanto, observou-se que este comportamento diminui a probabilidade de ocorrência de uma rápida convergência para um ótimo local[4]. Para as funções testadas em [18], a topologia em anel apresentou melhores resultados na maior parte dos casos.

4.2.2. Inicialização

Antes de o algoritmo poder ser executado, as posições e as velocidades de cada partícula devem ser inicializadas com valores aleatórios. Caso o problema a ser resolvido admita apenas respostas binárias, as partículas devem ser inicializadas apenas com valores pertencentes a {0,1} para as posições. No caso das velocidades, se houver uma velocidade máxima especificada, esta deverá ser obedecida na hora de se inicializar as partículas ($0 < Vid < Vmax$). Se o problema admitir respostas reais ou inteiras, então as partículas devem ser inicializadas de acordo com as restrições do problema. Como exemplo, suponha que cada dimensão de uma partícula represente determinado tipo de produto em um estoque, conforme mostrada na figura 4.6.

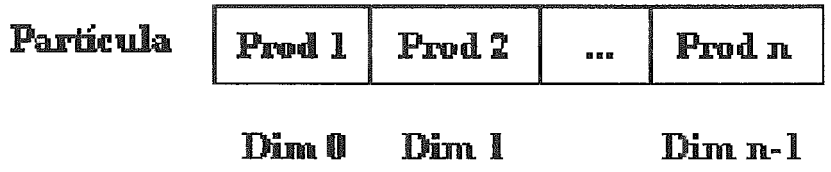

Fig 4.6 Codificação de uma partícula para o PSO

Se houver apenas 50 unidades do produto que estiver na dimensão 1, então esta deverá ser inicializada com um valor entre 0 e 50 (inclusive) [4].

4.2.2.1. Pseudocódigo

O pseudocódigo de inicialização é dado por:

(1) Para i = 1 até o número de partículas
(2) Para d = 1 até o número de dimensões para cada partícula
(3) $X_{id} = Rand(\beta)$
(4) $P_{id} = X_{id}$
(5) $V_{id} = Rand(Vmax)$
(6) Fim do para (2)
(7) Fim do para (1)

Onde X_{id} é o valor da posição na dimensão d da partícula i, P_{id} é a dimensão da partícula i onde foi obtido o melhor valor até o presente momento e V_{id} é a velocidade na dimensão d da partícula i.Rand é um algoritmo que gera números pseudo-aleatórios entre 0 e o valor de seu argumento utilizando uma distribuição uniforme.

Quando o algoritmo é inicializado, ainda não há melhor caminho já percorrido pela partícula. Neste caso, o valor do melhor caminho que a partícula já percorreu é o único que ela conhece, que é o valor com o qual ela foi inicializada. Por isso, na linha 4 o valor da posição onde a partícula obteve o melhor resultado recebe o valor da partícula na posição atual.

4.2.3. Cálculo da função objetivo

A função objetivo deve ser adaptada para cada problema a ser resolvido pelo PSO. Para o problema multidimensional da mochila 0/1, a função objetivo deverá levar em consideração dois fatos:

- O valor que resulta do transporte dos itens selecionados;
- Todas as restrições devem ser atendidas.

A função objetivo é dada pelo seguinte pseudocódigo:

```
(1)   Para i= 1 até o número de restrições
(2)     avaliacaoNaPosicaoAtual =0;
(3)     avaliacaoNaMelhorPosicao = 0;
(4)     Para j=1 até o número de dimensões da partícula atual
(5)       avaliacaoNaPosicaoAtual = (avaliacaoNaPosicaoAtual + (particula[j] *
                regra[i][j]))
(6)       avaliacaoNaMelhorPosicao = (avaliacaoNaMelhorPosicao + (particula[j] *
                regra[i][j]))
(7)     fim do para (4)
(8)     se avaliacaoNaPosicaoAtual <= restricoes[i] entao
(9)       restricoesObedecidasNaPosicaoAtual = (restricoesObedecidasNaPosicaoAtual + 1)
(10)    fim do se (8)
(11)    se avaliacaoNaMelhorPosicao <= restricoes[i] então
(12)      restricoesObedecidasNaMelhorPosicao =  (restricoesObedecidasNaMelhorPosicao +
                                                  1)
(13)    fim do se (11)
(14)  fim do para (1)
```

A função objetivo é calculada para a posição atual e ao mesmo tempo recalcula o valor da função de avaliação da partícula onde este obteve o melhor resultado até o momento. Deve ser verificado se a partícula obedece a todas as restrições. Isto é verificado nas linhas 8 e 9 para a posição atual e 10 e 11 para a melhor posição. Ao final deste código, a partícula que tiver obedecido ao maior número de restrições terá obtido o melhor valor da função de avaliação. Caso 2 partículas quaisquer obedeçam a todas as restrições por exemplo, a melhor partícula é dada pela que representar o maior valor monetário a ser transportado.

Suponha a seguinte instância, representada na figura 4.7, do problema multidimensional da mochila 0/1 retirada do sítio de Beasley[23]:

dimensão	1	2	3	4	5	6		
função objetivo	100	600	1200	2400	500	2000		
								restrição
regra 1	8	12	13	64	22	41	<=	80
regra 2	8	12	13	75	22	41	<=	96
regra 3	3	6	4	18	6	4	<=	20
regra 4	5	10	8	32	6	12	<=	36
regra 5	5	13	8	42	6	20	<=	44
regra 6	5	13	8	48	6	20	<=	48
regra 7	0	0	0	0	8	0	<=	10
regra 8	3	0	4	0	8	0	<=	18
regra 9	3	2	4	0	8	4	<=	22
regra 10	3	2	4	8	8	4	<=	24

Fig 4.7 Instância do problema 1 do sítio de Beasley

Se uma partícula tivesse os valores 0,1,0,1,0 e 0 para as dimensões de 1 a 6 respectivamente, então ela passaria na regra 1 (0x8 + 1x12 + 0x13 + 1x64 + 0x22 + 0x41 = 76 que é menor do que 80). O valor da função objetivo da partícula dada como exemplo seria 3000 (0x100 + 1x600 + 0x1200 + 1x2400 + 0x500 + 0x2000).

4.2.4. Versão binária

4.2.4.1. Pseudocódigo

O pseudocódigo da versão binária do algoritmo é dado por

(1) Inicialize as partículas
(2) Faça
(3) Para cada i de 1 até o número de partículas
(4) se $F(X_i) > F(P_i)$ então
(5) Para d de 1 até o número de dimensões da partícula atual
(6) $P_{id} = X_{id}$
(7) Fim do para (5)
(8) Fim do se (4)
(9) g = i
(10) obtém o conjunto dos vizinhos da partícula atual
(11) Para cada vizinho da partícula atual
(12) j = posição do vizinho
(13) se $F(P_j) > F(P_g)$ então
(14) g = j
(15) Fim do se (13)
(16) Fim do para (11)
(17) Para d = 1 até o número de dimensões da partícula atual
(18) $V_{id}(t) = V_{id}(t\text{-}1) + \varphi_1 (P_{id} - X_{id}(t\text{-}1)) + \varphi_2 (P_{gd} - X_{id}(t\text{-}1))$
(19) $V_{id} \in (-Vmax, Vmax)$
(20) Se $W_{id} < (Sigmóide(V_{id}))$ então
(21) $X_{id} = 1$
(22) senão
(23) $X_{id} = 0$
(24) Fim do se (20)
(25) Fim do para (17)
(26) Fim do para (3)
(27) até uma condição ser alcançada (2)

Sendo:

- F → Função objetivo;
- X_i → Posição atual da partícula atual;
- P_i → Posição onde a partícula atual obteve o melhor valor para a função objetivo;
- g → valor da partícula que possui melhor avaliação global da função objetivo. No início do algoritmo, como ainda não há dados para se saber qual partícula será a global, inicia-se esta variável com o valor da partícula atual;
- P_g → Partícula com a melhor avaliação da função objetivo entre todas as outras partículas. Se a vizinhança não fosse composta por todas as partículas mas por apenas um conjunto de partículas, como descrito no item 4.2.1, então este valor seria representado por Pl que indica a partícula com melhor avaliação da função de desempenho dentro de uma vizinhança definida;

- t → valor inteiro que representa o tempo. No algoritmo também representa o número da iteração corrente;
- V_{id} → Velocidade da partícula i na dimensão d;
- P_{id} → Posição da partícula i na dimensão d na posição em que obteve o melhor valor para a função objetivo;
- X_{id} → Posição atual da partícula i na dimensão d;
- P_{gd} → Posição da partícula global na dimensão d;
- φ_1 → Coeficiente de aprendizagem individual;
- φ_2 → Coeficiente de aprendizagem coletiva;
- V_{max} → Velocidade máxima permitida em qualquer dimensão. Esta velocidade é utilizada para impedir a explosão do sistema [4];
- W_{id} → vetor de números randômicos obtidos de uma distribuição uniforme variando entre 0 e 1;
- Sigmóide → função utilizada para levar um valor de entrada para 0 e 1 (ou –1 e 1). No caso, a função utilizada mapeia os parâmetros entre 0 e 1 [24].

Entre as linhas 2 e 30 estão os passos que o algoritmo executa em uma iteração. Sendo assim, o número de iterações é dado pela condição da linha 30. Entre as linhas 4 e 7 o valor da posição onde a partícula obteve a melhor avaliação da função objetivo é atualizado. Se o valor atual da função objetivo for melhor do que o melhor valor que a partícula tinha até o presente momento, então este valor é atualizado dentro do laço das linhas 5 e 6. Na linha 10, o índice da partícula atual passa a ser o índice da partícula global. Esta é apenas uma inicialização, pois a partícula global será escolhida mais a frente. Na linha 12, segundo a topologia da vizinhança sendo utilizada, os vizinhos da partícula atual são selecionados. Nas linhas 13 a 18, será selecionada entre os vizinhos da partícula atual, a que possuir o melhor valor para a função objetivo. Caso todas as partículas façam parte da vizinhança, esta partícula global será representada por P_g. Se apenas um conjunto de partículas fizer parte da vizinhança, esta partícula será a melhor global e representada por P_l. Nas linhas 20 a 29 a partícula é atualizada quanto as velocidades e suas posições em cada dimensão. Na linha 21 está sendo calculada a nova velocidade da partícula atual para a dimensão atual. Na linha 22 o valor obtido para a nova velocidade está sendo ajustado para ficar dentro dos limites superior e inferior de velocidade máxima. O bloco de operações que começa na linha 23 e termina na linha 27 atualiza as posições da partícula atual na dimensão atual. Na linha 23, o valor obtido

da velocidade é fornecido para uma função sigmóide que gera um resultado entre 0 e 1. No arranjo W , em cada posição encontra-se um número aleatório gerado por uma distribuição uniforme entre 0 e 1. Se o resultado da função sigmóide for menor do que este valor então a nova posição da partícula atual na dimensão atual assume o valor de 1. Caso contrário assume o valor de 0. Para cada iteração, todas as partículas passam por estes processo [4].

4.2.5. Versão para números reais

4.2.5.1. Pseudocódigo

O pseudocódigo do algoritmo para números reais consiste apenas em trocar no pseudocódigo binário o bloco entre as linhas 23 a 27 por:

$$X_{id}(t) = X_{id}(t-1) + V_{id}(t) \qquad (4.3)$$

Onde:

- $V_{id}(t)$ é a velocidade na dimensão d da partícula i no tempo t;
- $X_{id}(t)$ é a posição atual na dimensão d da partícula i no tempo t.

5. Metodologia

Nesta parte será explicada como foram realizadas as simulações. Os dados foram obtidos do sítio de J. E. Beasley [23]. Seja n o número de objetos e m o número de restrições. Neste se encontram os seguintes arquivos:

- mkanp1.txt → arquivo contendo 7 instâncias do problema da mochila 0/1 multidimensional sendo que estas possuem n variando de 6 até 50 e m variando de 5 até 10

A variável observacional será a primeira iteração na qual o algoritmo atingir o valor ótimo, no caso monoprocessado, ou que algum processo atingiu o valor ótimo, no caso multiprocessado. Para cada instância utilizada, serão criadas 30 posições iniciais. Cada posição inicial será fornecida à versão monoprocessada normal e as versões multiprocessada normal, multiprocessada utilizando o coeficiente de inércia e multiprocessada utilizando o coeficiente de constricção. Com isso, ambas as versões partem do mesmo ponto inicial no espaço de busca. Se houver realmente alguma diferença nos resultados, estes não serão devido a diferenças das posições iniciais. O seguinte procedimento será executado:

- para cada posição inicial, execute a versão monoprocessada normal, a multiprocessada normal, a multiprocessada com coeficiente de inércia e a multiprocessada com coeficiente de constricção do algoritmo partindo sempre do mesmo ponto para cada rodada;
- para cada rodada e cada versão do algoritmo guarda-se a iteração em que este obteve o valor ótimo ou o maior valor pela primeira vez.

5.1. *Adaptação do PSO para o modelo ilha*

No modelo ilha, cada processo possui um conjunto de partículas que realizam os cálculos entre si conforme explicado no item 4.2.4 do presente trabalho. Além disso, cada processo executando em um computador estará funcionando logicamente como um anel, ou seja, possuindo acesso direto apenas ao processo que está uma posição a sua frente no anel e uma

posição atrás no anel durante a execução do algoritmo. Para realizar a comunicação entre os processos, cada processo tem que saber calcular o número do vizinho posterior e anterior e saber enviar e receber os dados destes vizinhos.

5.1.1. Obtenção dos vizinhos no anel

Quando um programa MPI é posto em execução em um cluster, cada processador chama 3 funções inicialmente. São elas:

- MPI_Init
- MPI_Comm_rank
- MPI_Comm_size

MPI_Init é uma função que deve ser chamada antes que qualquer função de MPI possa ser chamada dentro de um programa. Permite ao sistema realizar qualquer inicialização necessária para que as bibliotecas de MPI possam ser usadas [6].

MPI_Comm_size reporta o número de processos que se encontram em um determinado comunicador.Um comunicador é um conjunto de processos que podem trocar mensagens entre si [6]

MPI_Comm_rank indica o número do processo atual dentro do comunicador, chamado de posto ("rank") dentro da terminologia do MPI [6].

Possuindo estas informações, podem ser utilizados dois esquemas de anel. Em um deles, todos os processos participam. No outro, o processo que recebe o posto 0 não participa das computações servindo apenas para aglutinar os resultados recebidos pelos outros processos. Com isso entretanto, um processo fica ocioso esperando apenas por dados ocasionando um desperdício em termos computacionais. Sendo assim, optou-se neste trabalho por utilizar o modelo em que o processo 0 além de ser responsável por aglutinar os dados no final da execução, realizará computações.

Para se calcular os vizinhos para quem deverá ser enviado e de quem deverá ser recebida as mensagens utiliza-se o seguinte algoritmo descrito em forma de pseudocódigo:

```
(1)   se ( tamanho_do_anel > 1) entao
(2)     se ( posto = 0 )  entao
(3)       proximo = 1
(4)       anterior = tamanho_do_anel – 1
(5)     senao se (posto = tamanho_do_anel – 1) entao
(6)       proximo = 0;
(7)       anterior = abs( módulo( (tamanho_do_anel – 2), tamanho_do_anel))
(8)     senao
(9)       proximo = posto + 1
(10)      anterior = posto – 1
(11)    fim do se (2)
(12)  fim do se (1)
```

Onde posto significa o número do processo recebido através da função MPI_Comm_rank, e tamanho do anel é o número de processos que estão participando da execução do programa, sendo retornado por MPI_Comm_size. O comando na linha 1, que pergunta sobre o tamanho do anel ser maior do que 1 está testando para ver se há mais do que um processo formando o anel. Caso haja apenas 1 processo, não há anel e não há comunicação. A função abs retorna o valor absoluto do parâmetro passado e a função módulo retorna o resto da divisão do primeiro parâmetro pelo segundo parâmetro.

5.1.2. Funções do MPI utilizadas para a comunicação entre os processos

A comunicação dos processos no MPI podem ocorrer de forma ponto-a-ponto quando apenas 2 processos se comunicam ou de forma coletiva, quando vários processos de um mesmo comunicador se comunicam entre si. As instruções utilizadas neste trabalho foram:

1) MPI_Send → é uma função bloqueante operando em modo padrão e utilizada em comunicação ponto-a-ponto para enviar dados. Possui a seguinte assinatura:

```
MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag,
          MPI_Comm comm)
```

onde buf é um arranjo com os dados a serem enviados, count é a quantidade de dados, datatype é o tipo dos dados que se encontram em buf, dest é o número do processo que

receberá os dados, tag é um identificador para os dados e comm é o comunicador onde este comando está sendo executado [7];

2) MPI_Recv → é uma função bloqueante operando no modo padrão e utilizada em comunicação ponto-a-ponto para recepção de dados. Possui a seguinte assinatura:

```
MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag,
                MPI_Comm comm,MPI_Status* status)
```

onde buf é um arranjo com os dados a serem recebidos, count é a quantidade de dados, datatype é o tipo dos dados que se encontram em buf, source é de que processo o processo atual espera receber a mensagem, tag é o tipo de mensagem que se espera receber, comm é o comunicador onde o comando está sendo executado e status é uma estrutura que contém informações do resultado da operação. Nesta estrutura de dados, encontram-se informações como: quem foi que enviou os dados, qual foi o tag recebido e qual foi o erro gerado. Estas informações são importantes para as rotinas de tratamento de erros [7].

3) MPI_Sendrecv → é uma função que combina, em apenas uma chamada, um envio de mensagem e um recebimento de outra mensagem. O processo destinatário e o processo de onde a informação virá podem ser o mesmo. Possui a seguinte assinatura:

```
MPI_Sendrecv(void* sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag,
                void* recbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag,
                MPI_Comm comm, MPI_Status* status)
```

onde sendbuf será um arranjo com os dados a serem enviados, sendcount será a quantidade de dados a serem enviados, sendtype será o tipo dos dados a serem enviados, dest será o processo destino para onde os dados irão, sendtag será um identificador para os dados, recbuf será um arranjo para os dados a serem recebidos, recvcount será a quantidade de dados a serem recebidos, recvtype será o tipo dos dados a serem recebidos, source será o processo de onde os dados estão vindo, recvtag será um identificador para os dados que estão sendo recebidos, comm será o comunicador onde este comando está sendo executado e status é uma estrutura que contém informações do resultado da operação. Nesta estrutura de dados, encontram-se

informações como: quem foi que enviou os dados, qual foi o tag recebido e qual foi o erro, caso algum tenha sido gerado. Estas informações são importantes para as rotinas de tratamento de erros [7].

4) MPI_Barrier → é uma função que faz com que todos os processos de um determinado comunicador cheguem a um mesmo ponto para que possam continuar a sua execução. Isto causa uma sincronização entre os processos. Com isso, garante-se que quando uma comunicação for feita, todos os processos estejam no mesmo ponto de execução. Possui a seguinte assinatura:

MPI_Barrier (MPI_Comm comunicador)

onde MPI_Comm é o comunicador onde o comando será executado.

5.1.3. Comunicação entre os processos

O pseudocódigo para esta comunicação é indicado abaixo:

```
(1)   para cada iteração
(2)   //executar o algoritmo do pso para todas as partículas
(3)     se (tamanho_do_anel > 1)
(4)       se (módulo (num_iteração, 100) = 0)
(5)         int melhor_particula = obter_posicao_melhor_particula()
(6)         int pior_particula = obter_posicao_pior_particula()
(7)         int vizinho_posterior = obter_vizinho_posterior()
(8)         int vizinho_anterior = obter_vizinho_anterior()
(9)         sincronização dos processo()
(10)        enviar(melhor_particula, vizinho_posterior)
(11)        receber(melhor_particula_vizinho_anterior, vizinho_anterior, pior_particula)
(12)      fim do se (4)
(13)    fim do se (3)
(14)  fim do para (1)
```

No pseudocódigo acima foram usadas 2 instruções – enviar e receber – apenas para facilitar o entendimento do algoritmo. Na implementação será utilizada a instrução MPI_Sendrecv que envia e recebe dados em uma única chamada. O comando da linha 3 do pseudocódigo é importante pois no caso de o programa executar apenas em 1 computador não há comunicação a ser feita.

A instrução de sincronização dos processos é realizada com a instrução MPI_Barrier que faz com que um determinado processo pare de executar até que todos os processos do comunicador selecionado tenham chegado neste ponto de execução. Com isso, garante-se que todos os processos estão no mesmo ponto e não haverá problemas de comunicação.

A comunicação entre os computadores é um procedimento custoso em termos de tempo. Sendo assim, a cada 100 iterações os computadores trocam informações entre si de modo a evitar uma excessiva perda de tempo. Isto é obtido com a função módulo. Se o número da iteração atual for múltiplo de 100 então neste momento haverá a comunicação entre os processos.

5.1.4. Aglutinação de resultados

A aglutinação dos dados é feita pelo processo com posto 0 no final da execução do algoritmo. Neste ponto, o processo 0 escreve em um arquivo o valor obtido por ele na execução do algoritmo, e recebe de cada processo um valor obtido por este para ser gravado no arquivo. A instrução utilizada para os processos enviarem seus dados para o processo de posto 0 é a MPI_Send, e a instrução utilizada pelo processo de posto 0 para receber os dados enviados pelos outros processos é a MPI_Recv.

5.1.5. Inicialização do algoritmo

Em vez de se gerar as partículas de forma randômica para cada rodada do algoritmo, as partículas serão fornecidas de um arquivo pré-gerado com um gerador de números pseudo-aleatórios da linguagem C. Isto atende ao fato de tanto a versão monoprocessada quanto a multiprocessada partirem do mesmo ponto no espaço de buscas. De outra forma, se cada versão partisse de um ponto inicial diferente, haveria mais um fator que poderia interferir com os resultados alcançados. Assim, o ponto inicial não irá interferir na performance do algoritmo.

5.2. *Versão final do algoritmo em pseudocódigo*

A seguir será mostrada a versão final do pseudocódigo do algoritmo a fim de facilitar o entendimento.

```
(1)   inicializa_o_MPI()
(2)   posto = obtem_posto()
(3)   tamanho_do_anel = obtem_tamanho_do_anel()
(4)   se tamanho_do_anel > 1 entao
(5)     se posto = 0 entao
(6)       proximo = 1
(7)       anterior = tamanho_do_anel - 1
(8)     senao se posto = tamanho_do_anel – 1
(9)       proximo = 0
(10)      anterior = abs( módulo( (tamanho_do_anel – 2), tamanho_do_anel))
(11)    senao
```

```
(12)     proximo = posto + 1
(13)     anterior = posto - 1
(14)    fim do se (5)
(15)   fim do se (4)
(16)   Inicialize as partículas
(17)   Faça
(18)    Para cada i de 1 até o número de partículas
(19)     se F(Xi) > F(Pi) então
(20)      Para d de 1 até o número de dimensões da partícula atual
(21)       Pid = Xid
(22)      Fim do para (20)
(23)     Fim do se (19)
(24)     g = i
(25)     obtém o conjunto dos vizinhos da partícula atual
(26)     Para cada vizinho da partícula atual
(27)      j = posição do vizinho
(28)      se F(Pj) > F(Pg) entao
(29)       g = j
(30)      Fim do se (28)
(31)     Fim do para (26)
(32)     Para d = 1 até o número de dimensões da partícula atual
(33)      Vid(t) = Vid(t-1) + φ1 ( Pid – Xid(t-1)) + φ2 (Pgd – Xid(t-1))
(34)      Vid ∈ (-Vmax, Vmax)
(35)      Se Wid < (Sigmóide(Vid)) então
(36)       Xid = 1
(37)      senão
(38)       Xid = 0
(39)      Fim do se (35)
(40)     Fim do para (32)
(41)    Fim do para (18)
(42)    se tamanho_do_anel > 1
(43)     melhor_particula = obter_pos_melhor_particula()
(44)     pior_particula = obter_pos_pior_particula()
(45)     enviar(particulas[melhor_particla], proximo)
(46)     receber(melhor_particula_vizinho_anterior,vizinho_anterior,
                  particulas[pior_particula])
(47)    fim do se (42)
(48)   até uma condição ser alcançada (17)
(49)   finalizar_MPI()
```

No código acima, foi utilizada uma instrução para finalizar o MPI que ainda não foi explicada. Esta função é a MPI_Finalize, utilizada na linha 49. Deve ser chamada apenas uma vez depois que não se for usar mais nenhuma função de MPI. Esta função serve para terminar o ambiente de execução do MPI – liberando memória que não vai mais ser utilizada pela biblioteca por exemplo.

5.3. Máquina utilizada

Foram utilizadas 4 máquinas virtuais. O cluster montado foi do tipo assimétrico sendo que uma das máquinas fez o papel de controlador ("head") enquanto as outras fizeram o papel de nós de processamento ("nodes"). Todas as máquinas possuíam um processador Intel Pentium IV com 2,4Ghz. A máquina controladora possuía 512Mb de memória RAM enquanto que as outras possuíam 256Mb deste tipo de memória. Todas as máquinas executavam o sistema operacional Linux versão Fedora Core 3. O software de cluster foi instalado a partir do pacote "Open Source Cluster Application Resources" – OSCAR. Este pacote vem com diversos programas necessários para a configuração de um cluster. Após instalado, todos os computadores do cluster possuem os softwares necessários e a configuração certa para executarem de forma correta. A versão do OSCAR utilizada foi a 4.2. Esta versão oferece 2 bibliotecas de MPI, a MPICH e a LAM. A biblioteca utilizada foi a LAM na versão 7.0.6. A linguagem de programação utilizada foi C++.

5.4. Parâmetros utilizados pelo algoritmo

Alguns dos parâmetros do algoritmo foram obtidos segundo [15]. Com isso, os valores utilizados foram:

- tamanho da população: 30 indivíduos [15] para problemas de até 30 dimensões. Com um número maior de dimensões, utilizou-se 3 partículas por dimensão;
- tamanho da vizinhança: vizinhança global abrangendo todos os indivíduos da população [15];
- coeficiente de aprendizagem individual igual a 2,1 [15];
- coeficiente de aprendizagem coletiva igual a 2,0 [15];
- a soma dos coeficiente de aprendizagem individual e coletiva ficou sendo 4.1 [15];
- Vmax será igual a 30;

6. Resultados

A seguir são apresentados os resultados da execução do programa sobre os problemas de teste. Nas colunas indicadas por iteração, está o número da iteração em que o algoritmo obteve o valor ótimo ou o melhor resultado pela primeira vez. A coluna monoprocessado normal indica o algoritmo executado em apenas um computador conforme descrito em [13]. A coluna multiprocessado normal indica o algoritmo descrito em [13] adaptado para executar em cluster no modelo ilha. A coluna multiprocessado inércia indica o algoritmo descrito em [13] adaptado para usar o coeficiente de inércia variando de forma linear entre 0,9 e 0,4 conforme descrito em [29] e ser executado em um cluster. A coluna multiprocessado constricção indica o algoritmo descrito em [13] adaptado para utilizar o coeficiente de constricção conforme descrito em [28] e ser executado em um cluster.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	5	0	1	0	3800	3800	3800	3800
1	5	1	0	0	3800	3800	3800	3800
2	3	0	0	1	3800	3800	3800	3800
3	4	0	1	0	3800	3800	3800	3800
4	3	0	0	0	3800	3800	3800	3800
5	2	0	1	0	3800	3800	3800	3800
6	3	0	0	0	3800	3800	3800	3800
7	1	0	1	0	3800	3800	3800	3800
8	1	0	0	0	3800	3800	3800	3800
9	0	0	0	0	3800	3800	3800	3800
10	0	3	1	0	3800	3800	3800	3800
11	1	0	0	0	3800	3800	3800	3800
12	1	2	1	0	3800	3800	3800	3800
13	1	1	0	0	3800	3800	3800	3800
14	1	1	0	0	3800	3800	3800	3800
15	1	1	0	1	3800	3800	3800	3800
16	1	0	0	0	3800	3800	3800	3800
17	7	1	0	0	3800	3800	3800	3800
18	3	1	0	0	3800	3800	3800	3800
19	6	0	0	2	3800	3800	3800	3800
20	0	4	0	0	3800	3800	3800	3800
21	1	0	0	0	3800	3800	3800	3800
22	1	2	0	0	3800	3800	3800	3800
23	2	5	1	0	3800	3800	3800	3800
24	0	2	0	1	3800	3800	3800	3800
25	5	0	1	0	3800	3800	3800	3800
26	0	0	0	0	3800	3800	3800	3800
27	1	1	0	0	3800	3800	3800	3800
28	5	1	0	0	3800	3800	3800	3800
29	0	0	0	0	3800	3800	3800	3800
max	7	5	1	2	3800	3800	3800	3800
min	0	0	0	0	3800	3800	3800	3800
media	2,13	0,87	0,27	0,17	3800	3800	3800	3800
desv. padrão	2,03	1,23	0,45	0,46	0	0	0	0

Tabela 6.1 Resultado para o problema 1.

O problema 1 possui 6 dimensões e 10 restrições. Os resultados apresentados na tabela 6.1 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 30 partículas, vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 3800 e foi alcançado em todas as rodadas do algoritmo. Este fato se deve ao espaço de buscas restrito a poucas dimensões.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	34	3	259	7	8706,1	8706,1	8706,1	8706,1
1	10	19	25	14	8706,1	8706,1	8706,1	8706,1
2	93	0	73	9	8706,1	8706,1	8706,1	8706,1
3	17	2	249	78	8706,1	8706,1	8706,1	8706,1
4	5	13	23	12	8706,1	8706,1	8706,1	8706,1
5	22	3	15	6	8706,1	8706,1	8706,1	8706,1
6	135	3	311	9	8706,1	8706,1	8706,1	8706,1
7	91	4	8	1	8706,1	8706,1	8706,1	8706,1
8	12	4	60	1	8706,1	8706,1	8706,1	8706,1
9	19	8	173	6	8706,1	8706,1	8706,1	8706,1
10	74	21	249	5	8706,1	8706,1	8706,1	8706,1
11	14	1	0	10	8706,1	8706,1	8706,1	8706,1
12	36	7	16	39	8706,1	8706,1	8706,1	8706,1
13	10	6	78	5	8706,1	8706,1	8706,1	8706,1
14	121	13	3	1	8706,1	8706,1	8706,1	8706,1
15	5	14	7	6	8706,1	8706,1	8706,1	8706,1
16	55	3	4	5	8706,1	8706,1	8706,1	8706,1
17	77	6	204	22	8706,1	8706,1	8706,1	8706,1
18	47	43	259	33	8706,1	8706,1	8706,1	8706,1
19	62	4	6	2	8706,1	8706,1	8706,1	8706,1
20	4	4	198	5	8706,1	8706,1	8706,1	8706,1
21	77	34	135	0	8706,1	8706,1	8706,1	8706,1
22	49	13	7	8	8706,1	8706,1	8706,1	8706,1
23	45	26	265	15	8706,1	8706,1	8706,1	8706,1
24	58	11	10	3	8706,1	8706,1	8706,1	8706,1
25	71	34	186	61	8706,1	8706,1	8706,1	8706,1
26	35	34	206	15	8706,1	8706,1	8706,1	8706,1
27	52	6	91	3	8706,1	8706,1	8706,1	8706,1
28	13	6	10	2	8706,1	8706,1	8706,1	8706,1
29	13	7	3	1	8706,1	8706,1	8706,1	8706,1
max	135	43	311	78	8706,1	8706,1	8706,1	8706,1
min	4	0	0	0	8706,1	8706,1	8706,1	8706,1
média	45,2	11,87	104,43	12,8	8706,1	8706,1	8706,1	8706,1
desv. padrão	35,33	11,54	106,56	17,99	0	0	0	0

Tabela 6.2 resultado para o problema 2

O problema 2 possui 10 dimensões e 10 restrições. Os resultados apresentados na tabela 6.2 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 30 partículas, vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 8706,1 e foi alcançado em todas as rodadas do algoritmo. O espaço de buscas começa a crescer no número de dimensões mas ainda assim não há diferenças entre as versões do algoritmo.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	75	150	6	4	**4015**	3985	**4015**	**4015**
1	278	265	7	23	4005	**4015**	**4015**	**4015**
2	414	34	6	76	3995	**4015**	**4015**	**4015**
3	17	407	11	17	3985	405	**4015**	**4015**
4	488	176	9	19	4005	4005	**4015**	**4015**
5	114	84	6	26	4005	3995	**4015**	**4015**
6	394	375	11	28	**4015**	4005	**4015**	**4015**
7	61	83	4	8	4005	**4015**	**4015**	**4015**
8	130	189	3	15	3945	**4015**	**4015**	**4015**
9	234	485	12	14	4005	3965	**4015**	**4015**
10	469	418	8	32	3985	**4015**	**4015**	**4015**
11	88	189	7	37	4005	**4015**	**4015**	**4015**
12	130	15	8	17	**4015**	4005	**4015**	**4015**
13	16	100	19	15	4005	4005	**4015**	**4015**
14	481	118	23	58	3955	4005	**4015**	**4015**
15	438	480	20	43	4005	3995	**4015**	**4015**
16	405	375	1	28	3915	**4015**	**4015**	**4015**
17	274	427	24	10	3965	**4015**	**4015**	**4015**
18	410	253	8	24	3945	3985	**4015**	**4015**
19	434	107	15	11	3955	4005	**4015**	**4015**
20	134	39	34	6	3935	4005	**4015**	**4015**
21	325	409	4	31	3995	4005	**4015**	**4015**
22	207	152	9	16	3940	4005	**4015**	**4015**
23	400	422	22	2	4005	**4015**	**4015**	**4015**
24	291	263	15	1	4005	**4015**	**4015**	**4015**
25	159	456	1	23	3995	**4015**	**4015**	**4015**
26	35	88	14	18	3950	3915	**4015**	**4015**
27	46	112	13	51	4005	4005	**4015**	**4015**
28	177	476	19	41	4005	**4015**	**4015**	**4015**
29	103	1	14	11	3950	**4015**	**4015**	4015
max	488	485	34	76	4015	4015	4015	4015
min	16	1	1	1	3915	405	4015	4015
média	240,9	238,27	11,767	26,5	3968,83	3883	4015	4015
desv. padrão	159,61	162,97	7,69	17,18	29,36	657,2	0	0

Tabela 6.3 Resultado para o problema 3

O problema 3 possui 15 dimensões e 10 restrições. Os resultados apresentados na tabela 6.3 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 30 partículas, vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 4015 e foi alcançado apenas 3 vezes no modo monoprocessado normal enquanto foi obtido em todas as execuções dos modos multiprocessados utilizando coeficiente de inércia e coeficiente de constricção, e 13 das 30 vezes no modo multiprocessado normal. A partir desta instância, o espaço de buscas do algoritmo cresceu e não houve mais convergência em todas as instâncias monoprocessadas. Nas instâncias multiprocessadas ocorreu um maior número de

convergências para o valor ótimo. Isto se deve ao fato de os coeficientes de inércia e de constricção permitirem uma exploração mais controlada do espaço de soluções.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	244	382	54	42	5980	5990	**6120**	**6120**
1	350	59	55	170	5825	5930	**6120**	**6120**
2	203	133	79	316	6040	5960	**6120**	**6120**
3	443	106	47	105	5775	5860	**6120**	**6120**
4	224	326	90	284	5880	5960	**6120**	**6120**
5	78	201	23	73	5930	5910	**6120**	6110
6	224	29	18	47	5915	6000	**6120**	**6120**
7	327	126	72	485	5900	5980	**6120**	**6120**
8	469	56	55	108	5820	6060	**6120**	**6120**
9	282	281	44	450	6000	5990	**6120**	**6120**
10	50	341	56	212	5910	6030	**6120**	**6120**
11	386	460	64	311	5830	5865	**6120**	**6120**
12	226	101	36	474	6040	6000	**6120**	**6120**
13	445	286	140	170	5940	5920	**6120**	6110
14	42	150	93	475	5980	5980	**6120**	**6120**
15	303	263	66	15	5900	5950	**6120**	**6120**
16	8	484	44	82	5930	6110	6110	**6120**
17	134	423	31	137	5750	6110	**6120**	**6120**
18	144	151	56	132	5870	5890	**6120**	**6120**
19	474	346	20	411	5880	5900	6110	**6120**
20	481	306	32	25	5925	6040	**6120**	**6120**
21	6	473	102	214	5800	5970	**6120**	**6120**
22	401	462	124	334	5820	6000	**6120**	**6120**
23	149	200	44	57	5870	5880	**6120**	**6120**
24	156	220	205	188	5950	5910	**6120**	**6120**
25	386	172	33	113	5880	5930	**6120**	**6120**
26	380	395	150	66	5960	6020	**6120**	**6120**
27	2	50	18	110	5940	6040	**6120**	**6120**
28	245	262	103	353	5840	5920	**6120**	**6120**
29	194	275	20	452	6050	6040	**6120**	**6120**
max	481	484	205	485	6050	6110	6120	6120
min	2	29	18	15	5750	5880	6110	6110
media	246,53	280,63	65,8	213,7	5914,33	5971,5	6119,3	6119,33
desv. padrão	151,82	138,73	44,26	155,00	77,47	67,02	2,54	2,54

Tabela 6.4 resultado para o problema 4

O problema 4 possui 20 dimensões e 10 restrições. Os resultados apresentados na tabela 6.4 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 30 partículas, vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 6120 e não foi alcançado nas versões monoprocessada e multiprocessada normais, tendo sido obtido nas versões multiprocessadas utilizando os coeficientes de inércia e de constricção. Estes dois

últimos permitem uma exploração mais controlada do espaço de buscas, o que não ocorre com as outras versões do algoritmo.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	418	178	62	353	11555	11850	12360	12380
1	478	273	69	296	11490	11940	12390	12360
2	247	34	82	403	11470	1560	12360	12340
3	140	345	105	166	11330	11610	12390	12290
4	128	240	151	272	12010	11670	12360	12300
5	8	203	81	42	11610	12010	12340	12270
6	330	1	62	407	12250	11840	12360	12290
7	112	323	51	60	11570	12160	12390	12320
8	421	105	82	343	1595	11890	12390	12240
9	84	99	126	496	11725	11770	12330	12260
10	254	167	49	146	11460	11740	12380	12280
11	89	181	14	112	12100	11930	12390	12300
12	84	19	69	338	11510	11750	12350	12300
13	79	78	77	418	11800	11860	12390	12300
14	28	271	126	307	11630	11980	12360	12280
15	383	156	66	111	11425	11750	12390	12350
16	100	12	97	410	11650	11450	**12400**	12320
17	52	371	169	138	11590	11945	**12400**	12340
18	100	470	132	199	11640	11990	12370	12310
19	49	266	167	255	12000	11860	**12400**	12320
20	38	490	66	386	11660	11805	12390	12290
21	472	10	11	270	11430	11930	12390	12300
22	62	62	79	258	11505	11770	12360	12300
23	6	423	60	217	11330	11750	**12400**	**12400**
24	246	43	312	184	11980	11515	12390	12310
25	487	235	223	193	11725	11640	12390	12340
26	7	90	45	402	11900	12210	12390	12310
27	269	61	25	309	11870	11950	12390	**12400**
28	443	184	83	472	11565	11750	12360	12280
29	298	76	214	316	11580	12140	12370	12250
max	487	490	312	496	12250	12210	12400	12400
min	6	1	11	42	1595	1560	12330	12240
media	197,07	182,2	98,5	276,97	11331,83	11501	12378	12311
desv. padrão	164,27	141,45	86,23	122,29	1853,29	1865,6	19,24	39,51

Tabela 6.5 resultado para o problema 5

O problema 5 possui 28 dimensões e 10 restrições. Os resultados apresentados na tabela 6.5 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 30 partículas, vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 12400 e não foi alcançado nas versões monoprocessada e multiprocessada normais, tendo sido obtido nas versões multiprocessadas utilizando os coeficientes de inércia e de constricção. Nesta

instância, mesmo utilizando os coeficientes de inércia e de constricção, não há uma convergência tão grande quanto a ocorrida nos problemas anteriores. Este fato se deve ao número de dimensões simuladas ser grande para um cluster montado a partir de máquinas virtuais.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	405	406	55	238	10357	10473	10526	10422
1	174	452	79	144	10366	10396	10548	10451
2	296	454	171	330	10435	10501	10540	10454
3	6	183	64	446	10376	10453	10560	10449
4	418	325	78	38	10398	10390	10585	10468
5	357	43	116	92	10363	10406	10512	10435
6	309	449	142	368	10391	10510	10547	10480
7	414	285	59	77	10422	10456	10547	10463
8	43	75	96	391	10391	10483	10555	10538
9	284	278	100	404	10447	10472	10537	10462
10	228	341	121	313	10372	10391	10553	10498
11	435	415	134	487	10413	10461	10579	10460
12	37	36	18	153	10373	10463	10540	10498
13	431	118	77	152	10427	10439	10523	10471
14	397	446	59	371	10366	10437	10559	10424
15	239	54	74	176	10384	10434	10573	10479
16	358	298	77	93	10360	10440	10536	10445
17	494	161	135	432	10352	10485	10557	10461
18	378	18	79	321	10449	10423	10538	10434
19	167	481	100	355	10480	10458	10551	10466
20	172	221	93	363	10430	10416	10584	10496
21	406	258	79	136	10452	10431	10564	10554
22	427	322	51	353	10400	10493	10575	10434
23	476	380	48	487	10362	10463	10582	10453
24	360	194	71	343	10527	10391	10543	10463
25	62	411	117	301	10453	10461	10541	10510
26	320	393	45	141	10453	10467	10521	10447
27	151	130	40	136	10426	10367	10494	10486
28	260	127	73	378	10483	10501	10582	10489
29	321	499	49	90	10449	10444	10559	10441
max	494	499	171	487	10527	10510	10585	10554
min	6	18	18	38	10352	10367	10494	10422
media	294,17	375,1	83,33	270,3	10411,9	10447	10550	10467,7
desv. padrão	138,48	152	34,62	137,08	44,75	37,05	23,40	31,13

Tabela 6.6 resultado para o problema 6

O problema 6 possui 39 dimensões e 5 restrições. Os resultados apresentados na tabela 6.6 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 117 partículas (3 partículas por dimensão), vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo

para este problema é 10618 e não foi alcançado por nenhuma versão dos algoritmos. O valor máximo alcançado foi de 10585, sendo que o desvio para alcançar o valor ótimo (33), equivale a apenas 0,31%. Entretanto, observa-se que na média o número de iterações necessárias para se chegar a um resultado nas versões multiprocessadas é menor do que na versão monoprocessada. Ainda pode-se notar que o valor monetário transportado nas versões multiprocessadas é maior do que o valor transportado na versão monoprocessada. Neste problema, apesar dos resultados das versões multiprocessadas serem melhores do que o da versão monoprocessada, o poder computacional do cluster de máquinas virtuais não foi suficiente para convergir o algoritmo para o valor ótimo conhecido.

	iteração				valor transportado			
	monoprocessado	multiprocessado			monoprocessado	multiprocessado		
ponto inicial	normal	normal	inércia	constricção	normal	normal	inércia	constricção
0	494	335	75	176	16226	16237	16383	16186
1	436	495	200	329	16095	16161	16317	16287
2	165	337	75	191	16096	16217	16362	16272
3	123	469	209	363	16112	16212	16324	16246
4	132	456	146	154	16182	16157	16332	16216
5	343	343	76	189	16164	16133	16316	16279
6	131	367	60	465	16115	16185	16391	16246
7	426	267	226	336	16171	16195	16336	16265
8	128	451	73	142	16068	16133	16391	16318
9	107	232	84	408	16101	16287	16340	16277
10	470	131	80	184	16086	16176	16328	16233
11	392	260	257	194	16275	16246	16343	16241
12	476	143	74	469	16232	16153	16334	16154
13	88	356	54	467	16147	16195	16394	16305
14	91	360	84	303	16172	16189	16300	16243
15	496	367	87	88	16208	16315	16334	16216
16	466	201	68	246	16161	16135	16284	16223
17	274	368	67	427	16200	16112	16337	16206
18	353	426	59	359	16258	16282	16336	16212
19	258	404	65	209	16166	16254	16396	16235
20	449	447	49	454	16242	16170	16341	16205
21	119	416	40	433	16172	16181	16348	16243
22	488	236	142	442	16034	16205	16319	16211
23	465	198	131	428	16133	16128	16323	16235
24	427	257	79	401	16175	16254	16362	16261
25	111	420	73	268	16183	16141	16396	16334
26	346	465	47	366	16138	16141	16344	16344
27	98	475	30	126	16218	16133	16372	16196
28	218	223	321	454	16120	16206	16349	16219
29	211	470	235	483	16221	16229	16342	16272
max	496	496	321	483	16275	16315	16396	16344
min	88	131	30	88	16034	16112	16284	16154
media	292,7	345,83	108,87	318,47	16162,37	16192	16345,8	16246
desv. padrão	155,43	106,83	74,30	125,74	58,69	53,07	29,09	43,72

Tabela 6.7 resultado para o problema 7

O problema 7 possui 50 dimensões e 5 restrições. Os resultados apresentados na tabela 6.7 foram obtidos utilizando coeficiente de aprendizagem individual 2,1 e coeficiente de aprendizagem coletiva igual a 2, 150 partículas (3 partículas por dimensão), vizinhança composta por todas as partículas, velocidade máxima de 30 e 500 iterações. O valor ótimo para este problema é 16537 e não foi alcançado por nenhuma versão do algoritmo. O valor máximo obtido foi de 16396, sendo que o desvio para alcançar o valor ótimo (141), equivale a apenas 0,85%. Neste caso, apenas a versão multiprocessada utilizando o coeficiente de inércia conseguiu seu valor ótimo em menos iterações do que a versão monoprocessada. Entretanto, os valores monetários transportados pelas versões multiprocessadas continuam sendo

superiores aos valores transportados pela versão monoprocessada. Neste problema, apesar de, na média, a convergência ser menor, o valor obtido por esta não superou nenhuma das versões multiprocessadas. Não se conseguiu chegar ao ótimo conhecido devido a falta de poder computacional que o cluster de máquinas virtuais oferece.

Os tempos de execução para a obtenção destes resultados foram:

	teste 1	teste 2	teste 3	teste 4	teste 5	teste 6	teste 7
tempo aproximado para executar 120 rodadas (em min.)	45	60	70	75	75	120	480
tempo aproximado para 1 execução (em s.)	22,5	30	35	37,5	37,5	60	240

Tabela 6.8 tempos de execução do algoritmo

Conforme mostrado na tabela 6.8, à medida que o número de dimensões aumenta, o tempo computacional também aumenta. Este fato se deve inicialmente a utilização de um cluster de máquinas virtuais executando todos sobre 1 máquina real. Caso houvesse um cluster de máquinas reais para ser utilizado, este tempo poderia ser diminuído. Entretanto, talvez não chegasse ao fator de diminuição correspondente ao número de máquinas, como por exemplo o tempo cair a um quarto caso se usassem 4 máquinas reais. Isto se deve a lei de Ahmdal que impõe um limite máximo sobre a melhora que se pode obter utilizando um cluster [6].

A seguir são mostrados os percentuais de obtenção do valor ótimo nos diversos testes realizados:

	% de obtenção do valor ótimo							
	monoprocessado	multiprocessado			número de			
	normal	normal	inércia	constricção	restrições	dimensões	partículas	iterações
teste 1 valor ótimo 3800	100%	100%	100%	100%	10	6	30	500
teste 2 valor ótimo 8706,1	100%	100%	100%	100%	10	10	30	500
teste 3 valor ótimo 4015	10%	43,30%	100%	100%	10	15	30	500
teste 4 valor ótimo 6120	0%	0%	93,30%	93,30%	10	20	30	500
teste 5 valor ótimo 12400	0%	0%	13,30%	6,60%	10	28	30	500
teste 6 valor ótimo 10618	0%	0%	0%	0%	5	39	117	500
teste 7 valor ótimo 16537	0%	0%	0%	0%	5	60	150	500

Tabela 6.9 percentual de obtenção do valor ótimo em cada problema

Na tabela 6.9, pode ser observado a influência do número de dimensões do problema sobre o percentual de convergência obtido para o valor ótimo. À medida que o número de dimensões vai aumentando, há uma superioridade das versões multiprocessadas em relação a versão monoprocessada, pois aquelas podem fazer uma exploração maior do espaço de

soluções, ou seja, cada nó computacional pode começar a sua pesquisa em uma posição diferente do espaço de soluções e trocar as melhores informações obtidas com os outros nós computacionais. Também pode ser observado que a partir de um determinado patamar relativo ao número de dimensões, o cluster de máquinas virtuais não possuiu o poder computacional necessário para a convergência do algoritmo para o valor ótimo conhecido.

A seguir é mostrado um quadro comparativo entre todos os testes realizados:

		iterações				valores financeiros transportados			
		monoprocessado	multiprocessado			monoprocessado	multiprocessado		
		normal	normal	inércia	constricção	normal	normal	inércia	constrilcção
problema 1	max	7	5	1	2	3800	3800	3800	3800
	min	0	0	0	0	3800	3800	3800	3800
	média	2,13	0,87	0,27	0,17	3800	3800	3800	3800
	desv. Padrão	2,03	1,28	0,45	0,46	0	0	0	0
problema 2	max	135	43	311	78	8706,1	8706	8706	8706,1
	min	4	0	0	0	8706,1	8706	8706	8706,1
	média	45,2	11,87	104,4	12,8	8706,1	8706	8706	8706,1
	desv. Padrão	35,33	11,54	106,6	17,99	0	0	0	0
problema 3	max	488	485	34	76	4015	4015	4015	4015
	min	16	1	1	1	3915	405	4015	4015
	média	240,9	238,3	11,77	23,5	3983,83	3883	4015	4015
	desv. Padrão	159,81	163	7,69	17,18	29,38	657,2	0	0
problema 4	max	481	484	205	485	6050	6110	6120	6120
	min	2	29	18	15	5750	5860	6110	6110
	média	248,53	250,6	65,8	213,7	5904,33	5972	6119	6119,33
	desv. Padrão	151,62	138,7	44,28	155,00	77,47	67,02	2,54	2,54
problema 5	max	487	490	312	496	12250	12210	12400	12400
	min	6	1	11	42	1595	1560	12330	12240
	média	197,07	182,2	98,5	275,97	11331,83	11501	12378	12311
	desv. Padrão	164,27	141,5	66,23	122,29	1853,29	1886	19,24	39,51
problema 6	max	494	499	171	487	10527	10510	10585	10554
	min	6	18	18	38	10352	10367	10494	10422
	média	294,17	275,1	83,33	270,3	10411,9	10447	10550	10467,7
	desv. Padrão	138,48	152	34,62	137,08	44,75	37,09	22,4	31,13
problema 7	max	496	495	321	483	16275	16315	16396	16344
	min	88	131	30	88	16034	16112	16284	16154
	media	292,7	345,8	108,9	318,47	16162,37	16192	16346	16246
	desv. Padrão	155,43	106,8	74,3	125,74	58,88	53,07	29,09	43,72

Tabela 6.10 Resumo dos valores obtidos nos testes.

Apesar de visto na tabela 6.9 que não houve poder computacional para convergir para o valor ótimo conhecido as instâncias do dos problemas 6 e 7, pode-se observar na tabela 6.10 que ainda assim as versões multiprocessadas obtiveram uma superioridade sob a versão monoprocessada. No problema 6 isto pode ser observado na iteração média para convergência e no valor médio obtido. No problema 7, o algoritmo monoprocessado converge na média de forma mais rápida que nas versões multiprocessadas normal e utilizando coeficiente de

constricção. Entretanto, em todas as versões multiprocessadas, o valor transportado foi maior do que na versão monoprocessada.

7. Conclusões e trabalhos futuros

A adaptação proposta para a heurística de otimização baseada em enxame de partículas se mostra funcional segundo as tabelas 6.8 e 6.9, onde são mostrados os tempos de execução e as taxas de convergência obtidas, respectivamente. Nos problemas 1 e 2, todas as versões, a monoprocessada e as multiprocessadas, do algoritmo conseguiram atingir o valor ótimo estabelecido. A partir do problema 3 podem ser observadas as vantagens da utilização das versões multiprocessadas. No problema 3, todas as versões multiprocessadas utilizando coeficientes, seja de inércia ou de constricção, obtiveram uma taxa de convergência em 100% das instâncias executadas. Algumas rodadas da versão multiprocessada normal começam a não obter o valor ótimo esperado devido a variável Vmax, que impede a explosão do sistema, mas limita seu espaço de buscas. Ainda assim, esta versão apresenta 43,3% de obtenção do valor ótimo. Neste problema, a verão monoprocessada obteve apenas 10% de êxito em chegar ao valor ótimo conhecido. No problema 4, Vmax novamente não permite que haja obtenção de valor ótimo para a versão multiprocessada normal. As versões utilizando os coeficientes, de inércia ou de constricção, conseguem ambas obter o valor ótimo conhecido em 93,3% das rodadas. A partir do problema 5, com o aumento do número de dimensões, o poder computacional das máquinas utilizadas não consegue obter uma grande taxa de convergência para o valor ótimo conhecido do problema. Nos problemas 6 e 7, o valor ótimo conhecido não foi obtido em nenhuma versão implementada. Entretanto, no problema 6, o melhor valor ótimo entre as rodadas possuía um desvio de 0,31% em relação ao ótimo conhecido, e no problema 7 este valor foi de 0,85%. De forma geral, as contribuições deste trabalho foram:

- Estudar o comportamento da heurística de otimização baseada em enxame de partículas quando adaptada para funcionar de forma clusterizada no modelo ilha;
- Estudar o comportamento dos coeficientes de inércia e de constricção em um ambiente clusterizado.

Como trabalhos futuros são sugeridos são sugeridos os seguintes experimentos a serem executados em um cluster com mais de uma máquina real:

- Manter as mesmas condições iniciais e o mesmo número de nós computacionais do trabalho atual para averiguar se há, e em caso afirmativo quais seriam, as diferenças em relação à convergência e ao tempo do algoritmo;
- Executar estes mesmos testes variando-se o número de nós computacionais para analisar o comportamento da convergência quando há mais ou menos poder de processamento, e tentar verificar o limite imposto pela lei de Ahmdal[6];
- Utilizar outras instâncias do problema da mochila 0/1 multidimensional, com maior número de restrições e / ou maior número de produtos para averiguar qual será o seu comportamento;
- Tentar outros tipos de problemas e fazer um estudo para avaliar se há alguma característica em um determinado tipo de problema que faça com que o algoritmo convirja em menos iterações ou execute em menos tempo;
- Tentar utilizar outras adaptações como os modelos individualista, coletivo e "selfless" [17].

8. Referências

[1] Brassard, G., Bratley, P., **Algorithms Theory and Practice**, Editora Prentice Hall, Inc, 1988

[2] Goodrich, M.T., Tamassia, R., **Estruturas de dados e algoritmos em java**, 2ª ed, Porto Alegre, Editora Bookman, 2002, tradução

[3] Kenney, J., Eberhart, R.C., **Particle Swarm Optimization**, Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ:IEEE press, pp 1942-1948, 1995

[4] Kennedy, J., Eberhart, R.C., Shi Y., **Swarm Intelligence**, Editora Morgan Kauffman Publishers, 2001

[5] Sloan, J. D., **High Performance Linux Clusters with OSCAR, Rocks, OpenMosix and MPI**, Editora O'Reilly, novembro de 2004

[6] Pacheco, P.S., **Parallel Programming with MPI**, Editora Morgan Kaufmann Publisers, Inc., 1997

[7] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J., **MPI-The Complete Reference Volume1, The MPI core**, 2ª ed., Editora The MIT Press, terceira impressão, 2000

[8] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., **Algoritmos – teoria e prática**, tradução da segunda edição americana, Editora campus, 2002

[9] Neapolitan, R., Naimipour, K., **Foundations of algorithms in C++ Pseudocode, Third Edition**, 3ª ed, Jones and Bartlett Publishers, 2004

[10] Martello, S., Toth, P., **Knapsack Problems Algorithms and Computer Implementations**, Biddles Ltd., Guildford, Grã-Bretanha, 1990

[11] Chu, P. C., Beasley, J. E., **A genetic algorithm for the multidimensional knapsack problem**, Journal of Heuristics, 4: 63-86 (1998), Kluwer Academic Publishers, 1998

[12] Silva, A.J.M., **Implementação de um Algoritmo Genético Utilizando o Modelo de Ilhas**, Dissertação de M.Sc.,COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2005

[13] Kennedy, J., Eberhart, R.C., **A discrete binary version of the particle swarm intelligence**, Proceedings of the 1997 Conference on Systems, Man and Cybernetics, 4104-4109. Piscataway, NJ:IEEE Service Center

[14] Patterson, D.A., Hennesy, J.L., **Computer Organization & Design**, 2ª edição,Editora Morgan Kaufmann, 1998

[15] Carlisle, A., Dozier, G., **An Off-The-Shelf PSO**, Proceedings of the workshop on Particle Swarm Optimization, Indianapolis, IN: Purdue School of Engeneering and Technology IUPUI, 2001

[16] Pitanga, M., **Construindo supercomputadores com linux**, 2ª edição, Editora Brasport, 2004

[17] Kennedy, J., **The Particle Swarm: Social Adaptation of Knowledge**, Proc. IEEE Intl. Conf. on Neural Networks (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ 303-308, 1997

[18] Kennedy, J., **Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance**, Proceedings of the 1999 Congress on Evolutionary Computation, 1931-1938. Piscataway, NJ:IEEE Service Center, 1999

[19] Reynolds, C.W., **Flocks, herds and schools: a distributed behavioral model**, Computer Graphics, 21(4): 25-34, 1987

[20] Heppner, F., Grenander,U., **A stochastic nonlinear model for coordinated bird flocks**, In S. Kranser Ed., The Ubiquity of Chaos, AAAS Publications, Washington DC, 1990

[21] Wilson,E.O., **Sociobiology: The new synthesis**, Belknap Press, Cambridge, MA, 1975

[22] Goldberg,D.E., **Genetic Algorithms in Search, Optimization and Machine Learning**, 23ª impressão, Editora Addison-Wesley, 2002
[23] Beasley, J.E., [**Sítio pessoal de Beasley com instâncias do Problema Multidimensional da Mochila 0/1**], Disponível em <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/>. Acesso em: 15 de outubro de 2005
[24] Haykin,S., **Redes Neurais Princípios e Práticas**, 2ª ed, tradução, Editora Bookman, Porto Alegre, reimpressão 2002
[25] Angeline,P.J., **Using Selection to Improve Particle Swarm Optimization**, Proceedings of the 1998 International Conference on Evolutionary Computation, 84-89, Piscataway, NJ:IEEE Press, 1998
[26] Kennedy, J., **The Behavior of Particles**, In V. W. Porto, N. Saravanan, D. Waagen, e A.E. Eiben (editores), Evolutionary Programming VII: Proceedings of the 7th Annual Conference on Evolutionary Programming, Berlin: Springer-Verlag, 1998
[27] Ozcan, E., Mohan,C.K., **Particle swarm optimization: particles surfing the waves**, Proceedings of the 1999 Congress of Evolutionary Computation, 1939-1944. Piscataway, NJ:IEEE Service Center, 1999
[28] Clerc, M., Kennedy, J., **The particle swarm: explosion, stability and convergence in a multidimensional and complex space**, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 1, Fevereiro 2002
[29] Shi, Y., Eberhart, R.C., **Parameter selection in particle swarm optimization**. Proceedings of the IEEE International Conference on Evolutionary Programming, San Diego, CA, 1998
[30] Shi, Y., Eberhart, R.C., **A modified particle swarm optimizer**, Proceedings of the IEEE International Conference on Evolutionary Computation, 69-73, Piscataway, NJ:IEEE Press, 1998
[31] Costa, M.C. de A., [**CPC 689 - Aula de processamento em cluster de pc**], COPPE/PEC, Junho a Agosto de 2005
[32] Eberhart, R.C., Shi, Y., **Comparing Inertia Weights and constriction factors in particle swarm optimization**, Proceedings of the 2000 Congress on Evolutionary Computation, 84-88, Piscataway, NJ:IEEE Service Center, 2000
[33] Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., George, A.D., **Parallel Global Optimization with Particle Swarm Algorithm**, Journal of numerical methods Engennering, John Wiley & Sons, 2003
[34] SC2005 Conference, Seattle, November, 2005, Disponível em <http://www.top500.org/lists/2005/11/top10_nov2005.pdf>, Acessado em: 20 de Dezembro de 2005
[35] SC94 Conference, 13-18, Washington, D.C, November, 1994, Disponível em <http://www.top500.org/lists/1994/11/>, Acessado em: 20 de Dezembro de 2005
[36] SC95 Conference, 3-8, San Diego, CA, December, 1995, Disponível em <http://www.top500.org/lists/1995/11/>, Acessado em: 20 de Dezembro de 2005
[37] SC96 Conference, 17-22, Pittsburgh, CA, November, 1996, Disponível em <http://www.top500.org/lists/1996/11/>, Acessado em: 20 de Dezembro de 2005.
[38] SC97 Conference, 15-21, San Jose, CA, November, 1997, Disponível em <http://www.top500.org/lists/1997/11/>, Acessado em: 20 de Dezembro de 2005
[39] SC98 Conference, 7-13, Orlando, FL, November, 1998, Disponível em <http://www.top500.org/lists/1998/11/>, Acessado em: 20 de Dezembro de 2005
[40] SC99 Conference, 13-19, Portalnd, Or, November, 1999, Disponível em <http://www.top500.org/lists/1999/11/>, Acessado em: 20 de Dezembro de 2005
[41] SC2000 Conference, 04-10, Dallas, TX, November, 2000, Disponível em <http://www.top500.org/lists/2000/11/>, Acessado em: 20 de Dezembro de 2005

[42] SC2001 Conference, 10-16, Denver, CO, November, 2001, Disponível em <http://www.top500.org/lists/2001/11/>, Acessado em: 20 de Dezembro de 2005
[43] SC2002 Conference, Baltimore, November, 2002, Disponível em <http://www.top500.org/lists/2002/11/>, Acessado em: 20 de Dezembro de 2005
[44] SC2003 Conference, Phoenix, AZ, November, 2003, Disponível em <http://www.top500.org/lists/2003/11/>, Acessado em: 20 de Dezembro de 2005
[45] SC2004 Conference, Pittsburgh, PA, November, 2004, Disponível em <http://www.top500.org/lists/2004/11/>, Acessado em: 20 de Dezembro de 2005