# Uma Abordagem Paralela Baseada em Colônia de Formigas para o Problema do Caixeiro Viajante

*Euzébio de O. A. da Silva[§] , Cristiana Bentes[δ], Laura Bahiense[γ] e Maria Clicia Stelling de Castro[§]*

*Instituto de Matemática e Estatística*
*CTC - UERJ* [§]
*clicia@ime.uerj.br*
*Brasil*

*Faculdade de Engenharia*
*FEN - UERJ* [δ]
*cristianabentes@gmail.br*
*Brasil*

*Empresa de Pesquisa Energética*
*MME*[γ]
*bahiense_l@yahoo.com*
*Brasil*

## Resumo

*Neste artigo propomos uma versão paralela para um problema clássico de otimização, denominado caixeiro viajante ou Travelling Salesman Problem (TSP). O TSP é um problema NP-completo de difícil solução. Um dos problemas principais no TSP é o alto tempo de computção necessário para encontrar uma solução. Uma das abordagens para resolver esse problema consiste em encontrar uma solução aproximada, através de alguma heurística conhecida, que pode ser encontrada na literatura. Por exemplo, soluções que utilizam algoritmos genéticos e GRASP.*

*O objetivo deste trabalho é descrever e avaliar uma versão paralela do TSP, utilizando como heurística a simulação de colônia de formigas. Nossos resultados foram obtidos em um sistema SP2, com 2, 4 e 6 processadores, e demonstram que obtivemos bons speedups, e soluções bem próximas do valor ótimo.*

## 1. Introdução

A constante demanda de poder computacional, exigida por diversas classes de aplicação, vem gerando a necessidade de redução do tempo de processamento. Os projetistas têm buscado soluções para proporcionar maior desempenho e ampliar o conjunto de aplicações que podem ser resolvidos de maneira eficiente.

A computação de alto desempenho é utilizada na programação de aplicações científicas, de multimídia, de gerenciamento de grandes volumes de dados entre outras. Os sistemas de alto desempenho podem ser construídos por máquinas com múltiplos processadores ou, ainda, pela utilização de dois ou mais computadores conectados em rede. Nestes sistemas é necessário o uso de um modelo de programação que expresse o paralelismo.
atividades. Esta característica é apropriada para o desenvolvimento de um algoritmo distribuído. Isso porque, cada indivíduo pode ser representado por um processo distinto e independente. Mesmo com uma

O caixeiro viajante (*Travelling Salesman Problem* - TSP) [5,9,13] é uma aplicação que demonstra ser adequada para os sistemas de alto desempenho. Ele é um problema clássico de otimização e definido como um problema NP-completo[8], de difícil solução. Inúmeras aplicações práticas podem ser representadas como um TSP. Por exemplo, aplicações que controlam a perfuração de placas de circuito[13]; o seqüenciamento de tarefas; o roteamento de veículos entre outras [3].

Neste trabalho descrevemos e avaliamos uma versão paralela para o problema do caixeiro viajante (TSP), utilizando a heurística baseada em colônia de formigas. A versão seqüencial inicial utilizada foi proposta em [4]. Na paralelização do problema utilizamos o modelo de programação de passagem de mensagem MPI[7,14]. Mostramos que a implementação paralela proposta pode resolver mais rapidamente o TSP e com valores próximos ao ótimo.

Este trabalho está organizado da seguinte forma. Na próxima seção descrevemos o comportamento e a modelagem matemática de uma colônia de formigas. Na Seção 3 e 4, descrevemos a versão seqüencial do algoritmo TSP baseado na heurística AS e a versão paralela implementada, respectivamente. A Seção 5 contém a análise dos resultados obtidos. Finalmente, na Seção 6 ressaltamos as nossas conclusões e sugerimos trabalhos futuros.

## 2. Colônia de Formigas

Pesquisadores, baseados no estudo do comportamento de uma colônia de formigas, propuseram o algoritmo *Ant System* (AS) [6]. O objetivo desse algoritmo é encontrar o caminho mínimo entre dois ou mais pontos.

Em uma colônia de formigas, cada uma delas é um indivíduo independente agindo sem a presença de uma autoridade ou outro indivíduo que coordene suas

anarquia aparente, pela ausência de um controle centralizado, a colônia de formigas possui uma organização, como se existisse de um governo central.

Os indivíduos (operários) de uma colônia de formigas[6], normalmente, só realizam um conjunto de tarefas definido de acordo com determinadas condições. Podemos citar, como exemplo entre as várias condições, a morfologia e a idade do indivíduo. É de conhecimento geral, em organização e métodos, que uma linha de produção industrial apropriada, é eficiente. Podemos dizer que é mais eficiente a divisão de trabalho entre os indivíduos da colônia, onde diferentes atividades são realizadas, simultaneamente, por grupos de indivíduos especializados, do que se as mesmas tarefas fossem realizadas por indivíduos não especializados e de forma seqüencial.

A capacidade de organização e de especialização encontrada nas colônias de formigas soluciona vários dos seus problemas diários. Esses problemas incluem basicamente encontrar comida, alimentar a população, dividir as tarefas eficientemente entre os indivíduos, além de responder a mudanças externas entre outros. Muitos desses problemas são semelhantes aos encontrados nas Engenharias ou na Ciência da Computação. Uma das características mais importantes destes indivíduos é capacidade de poder solucionar seus problemas de forma flexível e robusta.

Para uma colônia de formigas cumprir seu objetivo é necessário que exista interação entre os indivíduos. Uma das formas de comunicação é denominada estigmergia, e está abordada a seguir.

### 2.1 Estigmergia

Existem duas formas de interação entre indivíduos de uma colônia de formigas: as interações diretas e as indiretas. As interações diretas ocorrem com o contato: das antenas; das mandíbulas; através da visão; meio químico (odor emitido pelos indivíduos) entre outros. As formas de interações indiretas ocorrem quando um indivíduo da colônia modifica de alguma forma o ambiente e, algum tempo depois, um outro indivíduo responde a essa modificação. A interação indireta é denominada estigmergia. As formigas respondem a estímulos que ativam reações encadeadas. A produção de um novo estímulo, como conseqüência das reações vindas de um outro estímulo, determina a coordenação das atividades. É dessa forma que os indivíduos se comunicam indiretamente. Essa comunicação pode ser observada durante a construção de uma colônia.

A fase inicial da construção de uma colônia possui atividades aleatórias e não coordenadas, nos depósitos de pequenas quantidades de terra. Quando a quantidade de terra atinge uma determinada densidade em uma área restrita, ela se torna um estímulo significativo e novo. Então, mais formigas adicionam terra nesta área, construindo assim, os pilares da colônia. Os arcos são, então, montados sobre os pilares e finalmente são feitas as paredes nestes espaços terminando a construção de toda a colônia. A Figura 1 mostra, algumas fases, da construção de uma nova colônia.

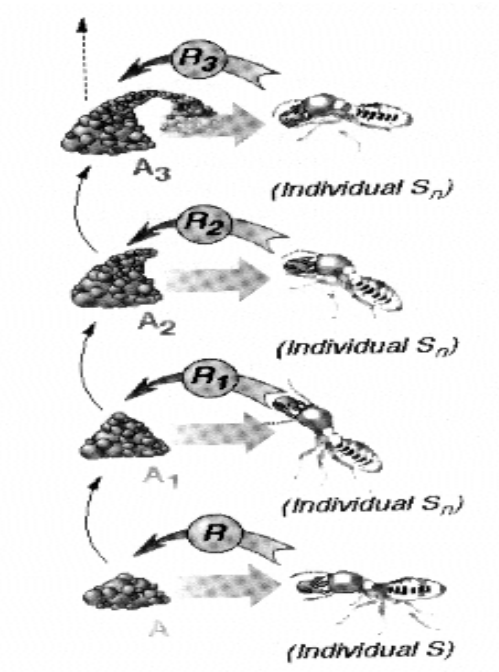


**Figura 1. Fases de construção de uma nova colônia**

A Figura 1 mostra as iterações indiretas entre as formigas na construção de uma colônia de formigas. A área *A* estimula uma resposta *R* da formiga *S* que deposita uma pequena quantidade de terra, resultando em um novo estímulo $A_1$. Esse novo estímulo resulta em uma resposta $R_1$ que pode ser atendido pelo indivíduo *S* ou por qualquer operário $S_n$. Essas ações são repetidas sucessivamente até que a construção termine [4].

A conduta individual de uma formiga modifica o ambiente, o que provoca estímulos a outros indivíduos da colônia. A comunicação entre indivíduos é apenas local, ou seja, somente os indivíduos que visitam o local modificado é que podem ser estimulados.

### 2.2 A busca do menor caminho

A estigmergia, que proporciona a comunicação entre as formigas, permite que seja encontrado, sem utilizar a percepção visual, o menor caminho entre a colônia e uma fonte de alimentos. O processo para encontrar a fonte de alimentos ocorre da seguinte forma.

Enquanto as formigas caminham da colônia para a fonte de alimentos, e vice-versa, elas depositam no solo uma substância química denominada feromônio. Em cada caminho é formada uma trilha de feromônio. As formigas podem sentir seu odor. Quando elas escolhem um caminho, a tendência é escolher o caminho que tiver

uma forte concentração de feromônio. Essa trilha permite que as formigas caminhem para a fonte de alimentos ou de volta para a colônia. Outras formigas, também, podem utilizar uma trilha para encontrar a localização desejada.

Para exemplificar o processo de estigmergia nos baseamos nas ilustrações das Figuras 2 e 3.

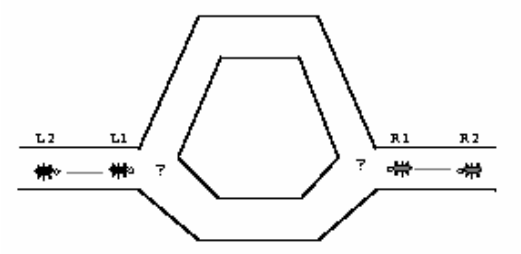


**Figura 2. Ponte binária**

A Figura 2 mostra um caminho hipotético (ponte binária) entre uma colônia e a fonte de alimentos. Este caminho possui uma bifurcação, criando assim dois caminhos distintos, um maior que o outro, até a fonte de alimentos.

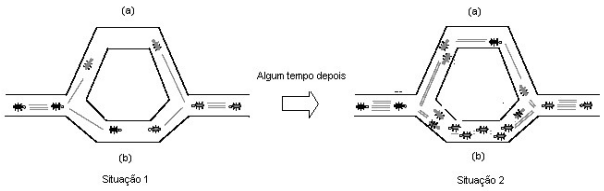


**Figura 3. Diferentes caminhos escolhidos pelas formigas, durante um determinado intervalo de tempo**

Inicialmente, um grupo de formigas busca por alimentos. No entanto, elas não sabem qual é o menor caminho. Então, as formigas escolhem aleatoriamente um caminho. Elas derramam feromônio pelo caminho enquanto caminham.

A Figura 3 mostra os caminhos da formigas durante um certo intervalo de tempo. As formigas que escolhem o caminho (b) chegam à fonte de alimentos e retornam para a colônia mais rápido do que as formigas que escolhem o caminho (a). Dessa forma, o caminho (b) acumula uma concentração maior de feromônio. A situação 2 da Figura 3 mostra que a concentração maior de feromônio no caminho (b) influencia outras formigas a seguirem este caminho. Porém, algumas formigas continuarão a escolher o caminho mais longo. Este fato ocorre porque existe um fator aleatório que influencia, também, na decisão da formiga. Entretanto, depois de um certo tempo, a quantidade de formigas que escolhe o caminho mais longo diminui significativamente. A conseqüência deste fato é que a concentração de feromônio torna-se o parâmetro principal na decisão da escolha do caminho.

Para todos os caminhos entre dois pontos que são formados, por cada formiga, a colônia deve ser capaz de explorar cada uma das trilhas de feromônio e descobrir o menor caminho entre estes dois pontos.

O processo de busca pela fonte de alimentos pode ser modelado matematicamente e está descrito a seguir.

### 2.3 Modelagem Matemática

A modelagem matemática, do algoritmo que simula o comportamento das formigas, pode ser elaborada considerando o caminho das formigas entre a colônia e uma fonte de alimentos. Este caminho pode ser representado (Figura 4) como uma ponte binária.

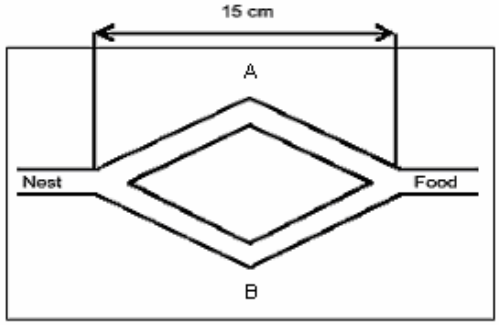


**Figura 4. Caminho entre uma colônia e uma fonte de alimentos, representado de forma binária**

A ponte binária é formada por dois caminhos de mesmo comprimento. Inicialmente não existe nenhuma quantidade de feromônio.

Denominamos o caminho superior de caminho $A$ e o inferior de caminho $B$. Quando as formigas caminham durante um certo intervalo de tempo, observa-se que as formigas escolhem o caminho de uma forma aleatória. Porém, em um dado momento um grupo de formigas escolhe aleatoriamente mais vezes um mesmo caminho. Esse comportamento provoca na trilha um aumento da concentração de feromônio. A partir deste momento mais formigas seguem este caminho. E assim, apenas um dos lados da ponte binária passa a ser utilizado pelas formigas da colônia.

A quantidade de feromônio é proporcional ao número de formigas. Logo o modelo probabilístico construído depende do número total de formigas que caminham por uma trilha até num determinado instante de tempo. Definimos como $A_i$ e $B_i$ o número de formigas que caminham pelo lado $A$ e lado $B$, respetivamente. Denominamos, da mesma forma, $P_A$ e $P_B$ como a probabilidade de uma outra formiga escolher o caminho $A$ ou $B$. Dessa forma, o modelo matemático pode ser definido da seguinte forma.

$$P_A = \frac{(k + A_i)^n}{(k + A_i)^n + (k + B_i)^n} = 1 - P_B \qquad (1)$$

$$P_B = \frac{(k + B_i)^n}{(k + B_i)^n + (k + A_i)^n} = 1 - P_A \qquad (2)$$

A quantidade de feromônio é proporcional ao número de formigas (*Ai* e *Bi*). A concentração de feromônio em um determinado caminho aumenta, e portanto, aumenta, também, a probabilidade da escolha do caminho. Os parâmetros *n* e *k* ajustam no modelo as observações do caminho das formigas. O *n* representa o grau de linearidade da função e o parâmetro *k* representa o grau de atração de um lado sem feromônio. Os valores dos parâmetros *k* e *n* que dão o melhor ajuste para as medidas experimentais, segundo citado no trabalho realizado por Campos [4], são $n \approx 2$ e $n \approx 20$. Se $Ai >> Bi$ e $Ai >> 20$, $P_A \approx 1$; *se* $Ai >> Bi$, mas $Ai < 20$, $P_A \approx 0{,}5$. O mesmo ocorre com a probabilidade $P_B$.

A dinâmica da escolha oriunda da equação (1) é dado por:

$$A_{i+1} = \begin{cases} A_i + 1 & \text{se } \delta \le P_A \\ A_i & \text{se } \delta > P_A \end{cases} \qquad (3)$$

$$B_{i+1} = \begin{cases} B_i + 1 & \text{se } \delta \le P_B \\ B_i & \text{se } \delta > P_B \end{cases} \qquad (4)$$

$$A_i + B_i = i$$

onde δ é uma variável aleatória sorteada em um intervalo ]0,1[.

Sem perder a generalidade, esta mesma modelagem é válida para pontes com mais de dois caminhos e com comprimentos diferentes [6].

## 3. TSP Baseado no Algoritmo AS

Para encontrar uma solução aproximada para o TSP podemos utilizar como heurística o algoritmo AS. A idéia é modelar a construção de uma trilha de feromônio produzida pelas formigas. No algoritmo AS é utilizado o mecanismo de comunicação indireta (estigmergia).

### 3.1 O Problema do Caixeiro Viajante (TSP)

O TSP pode ser definido da seguinte forma. Dado um determinado conjunto de cidades, como encontrar a rota de menor comprimento, onde todas as cidades devem ser visitadas, somente uma única vez.

Os grafos utilizados na representação do TSP são completamente conectados. Um grafo é denominado completamente conectado, se existe uma aresta ligando cada nó do grafo a todos os outros nós do conjunto [6].

Podemos destacar algumas razões que justificam a implementação do problema do caixeiro viajante baseado no algoritmo AS: i) é um problema de caminho mínimo, no qual a heurística da colônia de formigas pode ser facilmente adaptada; ii) é um problema clássico e importante que reflete vários problemas reais, tais como de perfuração de placas de circuito impresso [13] e seqüenciamento de tarefas; iii) é um problema de fácil entendimento e modificação, seus procedimentos podem ser facilmente adaptados com diferentes técnicas.

### 3.2 Formigas Artificiais

As formigas artificiais são agentes que representam uma abstração do comportamento observado nas colônias de formigas reais. Porém, elas podem ser implementadas com algumas habilidades, que não são encontradas em formigas reais, para tornar o sistema mais eficiente. Algumas das características que podemos citar são que: i) uma colônia de formigas é composta por indivíduos cooperativos; ii) as formigas preferem caminhos com alta concentração de feromônio; iii) existe uma taxa alta de aumento da concentração de feromônio nos caminhos menores (soluções de menor custo); iv) a comunicação indireta realizada entre as formigas pode ser medida pela trilha de feromônio (estigmergia); v) os critérios de decisão para encontrar o caminho de menor custo podem utilizar informações locais para determinar a direção de deslocamento dentro do espaço de busca.

As formigas artificiais podem receber habilidades extras, não encontradas nas formigas reais, para melhorar a eficiência do sistema AS. Por exemplo, podemos adicionar uma memória, que ao ser verificada e utilizado um procedimento, impeça que uma formiga passe mais de uma vez por uma cidade. Outra habilidade diferente das formigas reais é que as formigas artificiais podem apresentar a capacidade de derramar feromônio na trilha depois de gerar uma rota e não enquanto caminha.

### 3.3 Comportamento das formigas no algoritmo AS

As formigas artificiais se comportam de forma semelhante aos caixeiros viajantes. Elas se movem entre as cidades do grafo que representa o TSP. Cada formiga escolhe a próxima cidade a ser visitada baseada numa função probabilística (Seção 2.3). Essa função envolve o valor do comprimento e o feromônio acumulado nas rotas (arestas). As formigas artificiais preferem as cidades mais próximas que possuem uma grande concentração de feromônio.

Inicialmente, as formigas artificiais são colocadas em cidades selecionadas aleatoriamente. Enquanto elas caminham entre as cidades formando suas rotas, as formigas modificam a quantidade de feromônio. Quando todas as formigas completam seus caminhos, a formiga que percorreu a menor rota adiciona uma quantidade de feromônio nessa rota. Assim, é acumulada uma quantidade maior de feromônio nos caminhos (arestas) pertencentes a menor rota.

As formigas artificiais podem determinar a distância entre as cidades. Além disso, são dotadas de uma

memória ($M_k$), que armazena as cidades ainda não visitadas. A memória $M_k$ é representada por uma estrutura de dados bidimensional. Ela é atualizada a cada passo, retirando dela a cidade visitada. A formiga que estiver em uma cidade *i* escolhe visitar a cidade *j*, desde que essa cidade pertença a sua memória $M_k$.

### 3.4 Modelando o algoritmo AS

As soluções para o TSP são formadas pelas formigas, que caminham entre as cidades representadas no grafo do problema, até completar uma rota. Durante uma iteração do algoritmo AS cada formiga $k[1,..., m]$ constrói uma rota executando $n = N$ passos, onde $n$ representa o número da cidade e $N$ o total de cidades. As iterações são representadas por $t$, onde $t$ pertence ao intervalo entre 1 e o número máximo de iterações, $t_{max}$, definido pelo usuário.

Para cada formiga $k$, a transição da cidade $i$ para a cidade $j$ na iteração $t$ do algoritmo depende de três fatores:

i) se a cidade *j* já foi visitada pela formiga *k*. Existe uma memória para cada formiga, que é representada pelo conjunto $J_i^k$. Esse conjunto contém todas as cidades que a *k*-ésima formiga, estando na cidade *i*, ainda tem de visitar. Inicialmente, $J_i^k$ contém todas as cidades, exceto a cidade de onde a *k*-ésima formiga inicia sua rota. Quando uma cidade *i* é visitada pela formiga *k*, ela é retirada do conjunto $J_i^k$ para evitar uma nova visita;

ii) do parâmetro visibilidade, definido como o inverso da distância $\eta_{ij} = I/d_{ij}$. Este parâmetro é baseado em informação local e representa a conveniência heurística de se escolher a cidade *j,* a partir da cidade *i*. A visibilidade é estática, ou seja, não se modifica durante a execução da aplicação;

ii) da trilha de feromônio virtual $\tau_{ij}(t)$ presente nos caminhos (arestas) que conectam as cidades. A trilha de feromônio é atualizada a cada iteração. Ela pode ser vista como uma métrica da aprendizagem ao se escolher a cidade *j,* a partir da cidade *i*. A trilha de feromônio é uma informação global, modificada dinamicamente, para refletir as experiências adquiridas pelas formigas durante a construção da solução.

A regra de transição, isto é, a probabilidade da formiga *k* ir da cidade *i* para a cidade *j,* enquanto constrói a sua *t*-ésima rota, é dada pela seguinte fórmula:

$$p_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, & \text{se } j \in J_i^k \\ 0, & \text{se } j \notin J_i^k \end{cases} \qquad (5)$$

onde $\alpha$ e $\beta$ são dois parâmetros ajustáveis que controlam o peso relativo da intensidade da trilha, $\tau_{ij}(t)$, e da visibilidade, $\eta_{ij}$. Se o parâmetro $\alpha$ for nulo, as cidades próximas são mais prováveis de serem selecionadas. Este comportamento corresponde a um algoritmo guloso clássico (com múltiplos pontos de inicialização, pois as formigas são aleatoriamente distribuídas entre as cidades no início do processo). Porém, se $\beta$ for nulo é considerado somente a quantidade de feromônio. Esse método pode conduzir à seleção rápida de rotas. Entretanto, essas rotas podem não ser ótimas, isto é, as rotas mínimas. É importante notar que embora a fórmula da equação (5) permaneça constante durante uma iteração, o valor da probabilidade $p_{ij}^k(t)$ pode ser diferente para duas formigas em uma mesma cidade *i.* Isso porque a probabilidade $p_{ij}^k(t)$ é uma função de $J_i^k$. Ou seja, ela depende da solução parcial construída pela *k*-ésima formiga.

Na prática, quando a formiga *k* caminha da cidade *i* para a cidade *j*, ela adiciona a aresta *(i, j)* em sua rota. Este passo é repetido até que a formiga complete a sua rota. Ao completar a rota, a *k*-ésima formiga derrama uma quantidade de feromônio $\Delta\tau_{ij}(t)$ nas arestas *(i, j)* utilizadas. Este fato torna as rotas mais atraentes às futuras formigas na construção de novas rotas. Na iteração *t*, a formiga *k* derrama feromônio na aresta *(i, j)* pertencente a sua rota de acordo com a equação (6):

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q / L^k(t) & \text{se (i, j)} \in T^k(t), \\ 0 & \text{se (i, j)} \notin T^k(t), \end{cases} \qquad (6)$$

onde $T^k(t)$ é a rota percorrida pela formiga *k* na iteração *t*, e $L^k(t)$ é o seu comprimento. $Q$ é um parâmetro ajustável usado para determinar a taxa de feromônio que é derramado nas rotas.

Este método pode produzir um mau desempenho, se não houver a redução da quantidade de feromônio. O aumento ilimitado da quantidade de feromônio pode conduzir a pouca amplificação das flutuações aleatórias iniciais, o que provavelmente não seria ótimo para assegurar a exploração eficiente do espaço de busca de solução do problema. A intensidade da trilha deve permitir a redução do feromônio. Caso contrário, todas as formigas poderiam terminar percorrendo a mesma rota (estagnação). A estagnação é uma decorrência do aumento da intensidade da trilha. A probabilidade de transição entre as cidades deve ser dominada pelo termo feromônio. A redução da quantidade de feromônio na trilha é implementada pela introdução do coeficiente de redução. Ele é denominado por $\rho$, e seus valores estão no intervalo [0,1]. A regra da atualização da quantidade de feromônio resultante aplicada em todas as arestas é dada pela seguinte fórmula:

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t), \qquad (7)$$

onde $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t)$, e $m$ representa o número de formigas.

A quantidade inicial de feromônio nas arestas é dada por uma pequena constante positiva $\tau_o$. Desta forma,

existe uma distribuição homogênea de feromônio na primeira iteração onde $t = 0$.

É importante manter o número total de formigas $m$ constante durante o tempo de execução da aplicação. Se houver um grande número de formigas, as trilhas subótimas podem ter rapidamente incrementada a quantidade de feromônio. Este fato pode conduzir a uma convergência prematura de soluções ruins. Por outro lado, poucas formigas podem não produzir os efeitos esperados de cooperação, por causa do processo de declínio de feromônio.

Dorigo *et al* [7] sugerem que um bom compromisso é atingido definindo o número de formigas igual ao número de cidades do problema ($m = n$). No início de cada rota, as formigas são alocadas aleatoriamente as cidades (representadas pelos nós do grafo).

Para melhorar o desempenho do algoritmo AS Dorigo *et al* [7], também, introduziram o elitismo nas formigas (similar ao elitismo utilizado nos algoritmos genéticos). Uma formiga elitista, representada por $e$, é uma formiga que reforça as arestas pertencentes a $T^+$ (a melhor rota encontrada), com a taxa de feromônio igual a $Q/L^+$, onde $L^+$ é o comprimento de $T^+$ e $Q$ é um parâmetro variável para ajustar a taxa de feromônio derramada.

A cada iteração, $e$ formigas elitistas são adicionadas às formigas normais de maneira que as arestas pertencentes a $T^+$ obtenham um reforço extra $e.Q/L^+$. A idéia é que a trilha de feromônio de $T^+$, então reforçada, direcione a busca de todas as outras formigas para uma solução composta de algumas arestas da melhor rota.

O TSP baseado no algoritmo AS proposto utiliza os valores dos parâmetros adotados por Dorigo *et al* [7] em seus experimentos (8). A complexidade de AS é $O(t.n^2.m)$, onde $t$ é o número de iterações realizadas. Se $m = n$, ou seja, se o número de formigas for igual ao número de cidades, a complexidade torna-se $O(t.n^3)$.

$$\alpha = 1, \beta = 5, \rho = 0.5, m = n, Q = 100, \tau_0 = 10^{-6}, e = 5 \quad (8)$$

A Figura 5 descreve em alto nível (pseudocódigo) o o algoritmo AS.

/* Inicialização */
Para cada aresta *(i, j)* faça $\tau_{ij}(0) = \tau_0$
/* Distribuição das formigas */
Distribua todas as $k$ formigas numa cidade escolhida aleatoriamente
Seja $T^+$ a menor rota encontrada inicialmente e $L^+$ o seu tamanho
/* Loop principal */
Para todas as $t$ iterações e todas as $k$ formigas
  Construa a rota $T^k(t)$ aplicando $n - 1$ vezes o passo:
  Escolha a próxima cidade $j$ de acordo com a probabilidade

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta},$$

  onde $i$ = cidade atual
  Para todas as $k$ formigas compute o tamanho $L^k(t)$ da sua rota $T(t)$
  Se uma rota melhor for encontrada então atualize $T^+$ e $L$
  Para cada formiga atualize a aresta *(i, j)* segundo a regra

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) + e \cdot \Delta\tau_{ij}^e(t) \text{ onde}$$

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q/L^k(t) & \text{se (i, j)} \in T^k(t); \\ 0 & \text{caso contrário,} \end{cases} \quad \text{e}$$

$$\Delta\tau_{ij}^e(t) = \begin{cases} Q/L^+ & \text{se (i, j)} \in T^+; \\ 0 & \text{caso contrário,} \end{cases}$$

Para cada aresta *(i, j)* faça $\tau_{ij}(t+1) = \tau_{ij}(t)$

Imprima a menor rota $T^+$ e o se tamanho $L^+$

**Figura 5. Pseudocódigo do algoritmo AS**

## 4. TSP Paralelo Baseado no Algoritmo AS

Esta seção discute a paralelização da aplicação TSP utilizando a heurística AS. Nossa aplicação é inspirada na versão seqüencial apresentada em [4], implementada utilizando a linguagem Pascal. Para obter melhor desempenho e utilizar a biblioteca MPI, a aplicação foi desenvolvida na linguagem C[10]. Procuramos manter a estrutura lógica da versão inicial, porém, algumas estruturas de dados foram modificadas devido à existência de características distintas entre as duas linguagens. Os principais pontos que formam a aplicação paralela estão abordados nas sessões a seguir.

### 4.1 Versão do Algoritmo PTSP_AS Paralelo

O algoritmo AS executa um *loop* principal de acordo com o número de iterações determinadas pelo usuário (Seção 3.4). A cada iteração um determinado número de formigas são distribuídas entre as cidades, uma de cada vez. No final do caminho realizado por todas as formigas, na iteração corrente, é selecionada a melhor rota e atualizada a taxa de feromônio desse caminho. Este processo repete-se em todas as iterações. Em uma colônia real de formigas podemos notar que não apenas uma, mas várias formigas participam simultaneamente na solução do problema de encontrar o menor caminho. A idéia consiste, então, em adaptar este modelo para distribuir mais de uma formiga para percorrer as cidades simultaneamente. Denominamos esta versão PTSP_AS (*Parallel Travelling Salesman Problem Based on Ant System*).

#### Arquitetura e Lógica da Aplicação PTSP_AS

A aplicação PTSP_AS foi desenvolvida no modelo de programação de memória distribuída e para ser executada em uma arquitetura MIMD[11]. Ela possui um processo mestre coordenando vários processos escravos.

No sistema paralelo implementado no nível do usuário, através da biblioteca MPI, é necessário que o ambiente seja iniciado com os parâmetros adequados. No PTSP_AS o ambiente é iniciado na função `main`.

O sincronismo entre os processos é realizado utilizando um número de identificação, obtido durante a

execução da aplicação. Então, cada processo possui uma identificação que é armazenada numa variável global, local a cada processo. O número de identificação é um inteiro no intervalo de [0..*N*-1], onde *N* é igual ao número total de processos.

O processo mestre é identificado pelo número 0. Na versão PTSP_AS definimos uma macro, denominada `QUEEN_ANT`, através de um `define`, para o processo mestre. Este `define` é importante porque controla quais seguimentos de código são executados somente pelo processo mestre. Estes trechos de código são realizados seqüencialmente.

A aplicação possui cinco fases, correspondentes a: inicialização, distribuição de formigas, percurso das cidades pelas formigas, determinação do caminho mínimo e atualização do caminho com feromônio. As duas últimas fases são atualizadas a cada iteração. Discutimos, a seguir, em termos gerais cada uma das fases da aplicação PTSP_AS.

Na fase de inicialização, o processo mestre verifica quantos processos foram disparados (parâmetro definido pelo usuário) e realiza a leitura dos dados de entrada.

A distribuição das formigas é realizada uma única vez no inicio da aplicação. Isto é, o processo mestre calcula apenas uma vez a quantidade de formigas que é atribuída aos processos escravos. O total de formigas, igual ao número de cidades (Seção 3.4), é constante durante todo o tempo de execução da aplicação paralela. Assim, na segunda fase, o processo mestre verifica a quantidade de processos e tenta dividir igualmente o total de formigas entre os processos escravos. Porém, se o total de formigas não for múltiplo do número de processos é feita uma distribuição das formigas restantes resultante da divisão, uma a uma, para cada processo até todas tenham sido distribuídas.

O processo mestre (`QUEEN_ANT`) executa o trecho do código responsável pela distribuição das formigas. Caso contrário, o processo é escravo e executa o trecho do código de recepção das formigas enviadas.

Cada formiga é identificada por um número inteiro armazenado num vetor em cada processo escravo. O processo mestre envia aos processos escravos quais formigas eles devem armazenar em seu vetor local. A função de envio de dados é síncrona. Desta forma, a execução do processo mestre permanece parada até que os processos escravos enviem uma mensagem de confirmação de recebimento. A função de recepção de dados também é síncrona. Assim, ela pára a execução no processo escravo até que ele receba a informação enviada pelo mestre. Ao receber uma mensagem o processo escravo envia a confirmação de recebimento ao processo mestre.

Na terceira fase, cada processo escravo, após receber suas formigas, inicia a determinação das rotas naquela iteração. Para a escolha dos caminhos é utilizado o modelo probabilístico apresentado na Seção 2.3. Cada processo possui armazenado o valor da melhor rota encontrada em uma iteração anterior. Se a melhor rota encontrada pelas formigas na iteração corrente for melhor do que armazenada anteriormente, o processo escravo envia o tamanho da rota e o seu percurso correspondente ao processo mestre. Caso contrário, as formigas deste processo não conseguiram encontrar uma rota melhor. Quando ocorre este fato o processo escravo envia um valor indicando ao processo mestre que não foi encontrado um caminho melhor na iteração corrente.

O processo mestre recebe, então, uma mensagem de cada processo escravo contendo: ou um valor da rota menor do que o valor global corrente junto com a respectiva rota, ou um valor nulo informando que a rota encontrada não é menor.

Na quarta fase, a partir destas informações, o processo mestre verifica qual é a menor rota dentre todas as computadas. E, então, informa aos processos escravos as informações da menor rota.

Um sinal, representado por –1, é enviado pelo processo mestre aos processos escravos, quando a rota encontrada na iteração corrente não é menor que a encontrada em uma iteração anterior. Dessa forma, o valor da menor rota, armazenado em cada processo escravo deve ser preservado.

Após o processo mestre enviar o valor da nova rota aos processos escravos, na quinta fase, ele calcula e atualiza, com feromônio, a estrutura de dados que armazena todas as trilhas. A taxa de feromônio é calculada usando o comprimento e a rota de todas as formigas (Seção 3.4).

Cada processo possui, em sua memória local, uma cópia da estrutura de dados que armazena: as cidades, os caminhos entre as cidades e a taxa de feromônio de cada caminho. Então, o processo mestre deve enviar as informações aos processos escravos para que todas as suas estruturas de dados sejam atualizadas.

Só após esta atualização, o sistema paralelo inicia uma nova iteração. A aplicação termina após $t_n$ iterações definidas pelo usuário.

## 5. Resultados Preliminares

Nesta seção descrevemos os experimentos executados para a avaliação de desempenho da versão da aplicação paralela PTSP_PAS. Estes experimentos mostram o comportamento do desempenho desta versão em relação à versão seqüencial da aplicação. Para isso utilizamos uma máquina SP2[1]. Ela é formada por 6 processadores, tecnologia IBM RISC System/6000[2], independentes e interligados por *High Speed Switch*[2]. A aplicação

1 Sistema com arquitetura MIMD com memória distribuída construído pela IBM.

PTSP_AS foi executada em 1, 2, 4 e 6 processadores. Utilizamos sempre a mesma massa de dados de entrada em nossos experimentos. As informações fornecidas, utilizadas na análise, são: o tamanho e o percurso de um possível caminho mínimo, e o tempo de processamento utilizado para encontrar este caminho. As informações disponibilizadas são utilizadas no cálculo e na análise do *speedup*[15].

### 5.1Ambiente Experimental

O sistema SP2 usado neste trabalho, de propriedade da Universidade do Estado do Rio de Janeiro, é composto por 6 processadores. Esse é um sistema híbrido, onde 2 processadores possuem tecnologia *RISC POWER2 Wide*[2]; com desempenho de pico de 308 *MFLOPS*, uma memória RAM de 512 *Mbytes* em cada processador e um disco rígido de 9 *Gbytes*. Os outros 4 processadores possuem tecnologia *RISC Power2 Thin:* com desempenho de pico de 266 *MFLOPS,* uma memória RAM de 256 *Mbytes* em cada processador e um disco rígido de 2,2 *Gbytes*. Todos os processadores estão interligados por um *High Performance Switch*[2]. O sistema operacional utilizado foi o AIX[3] versão 4. O sistema possui, ainda, suporte para as linguagens C/C++ e Fotran e para as bibliotecas PVM [12] e MPI[14] .

A versão PTSP_AS foi desenvolvida na linguagem C e para dar o suporte ao paralelismo foi utilizada a biblioteca MPI.

Para executar uma aplicação paralela usando a biblioteca MPI, no sistema SP2, é necessário ativar uma variável de ambiente, denominada `MP_HOSTFILE`, com o nome do arquivo que contém a identificação dos processadores usados durante a execução da aplicação. Além disso, é necessário usar a opção `-procs` seguido do número de processos a serem criados.

Os dados de entrada utilizados são lidos de um arquivo do tipo texto padrão. Este arquivo contém as coordenadas das cidades, e os parâmetros constantes utilizados por Dorigo *et al*. [7], os quais apresentaram os melhores resultados do trabalho. As coordenadas são formadas por pares de pontos de um plano cartesiano bidimensional. A semente, para geração dos números aleatórios, é obtida por um número dentro do intervalo ]0,1[. Assim como em Campos [4] (em sua versão seqüencial), nos experimentos realizados foi utilizado um conjunto com 27 entradas diferentes. Cada uma dessas entradas possui sementes distintas, que são usadas para a geração de números aleatórios.

O conjunto de dados foi obtido na página TSPLIB. TSPLIB é uma biblioteca de problemas TSP de diversas fontes e de diversos tipos, que podem ser encontradas em [1]. Usamos um exemplo com 101 cidades fornecido por esta biblioteca, onde cada cidade é representada por um par de coordenadas cartesianas. Com este exemplo foram criados 27 arquivos com as coordenadas e com uma semente distinta.

O número aleatório interfere na escolha do caminho, da seguinte forma. A formiga escolhe um caminho comparando o número aleatório com a probabilidade de escolha daquele caminho. Se a probabilidade de um determinado caminho for maior que o número aleatório, então, a formiga escolhe este caminho. Caso contrário, a formiga descarta este caminho. Depois, verifica a probabilidade dos outros caminhos até encontrar um, cuja probabilidade seja maior que o número aleatório.

Após a execução da versão da aplicação PTSP_AS obtivemos um arquivo de saída com as informações necessárias à análise, como mostrada na Figura 6.

```
Semente: 0.007000
Melhor Rota: 85 15 60 84 90 99 36 97 92 98
95 58 91 96 94 93 5 88 100 52 57 39 20 72
71 73 21 40 74 55 38 22 66 24 54 3 53 23
28 67 79 11 25 27 26 68 0 49 75 76 2 78 77
33 80 32 50 8 70 34 64 65 19 29 69 30 87 6
81 47 46 35 48 63 10 62 89 31 9 61 18 45 7
44 16 83 4 59 82 17 51 12 86 1 56 14 42 41
13 43 37
Tamanho da Menor Rota: 664.97519
Número da Iteração da Menor Rota: 2188
Insucessos: 811
Tempo de processamento: 11711
```

**Figura 6. Exemplo de um arquivo de saída**

### 5.2 Análise de Resultados

Inicialmente, a aplicação PTSP_AS foi executada em apenas um processador. A aplicação foi submetida ao sistema de forma seqüencial em um dos processadores do sistema SP2. Para este primeiro teste, foi escolhido o processador com desempenho de pico de 308 *Mflops*. Este é um dos 2 processadores mais rápidos do sistema SP2. Isto foi feito para determinar se a versão paralela da aplicação executada no sistema com mais de um processador, é mais rápida do que a execução em apenas um único processador. Escolhemos o processador mais rápido porque desta forma garantimos que nossos resultados têm maior potencial em obter *speedups*.

Os resultados obtidos, mostrados na Tabela 1, são usados para gerar os valores de *speedup*. A primeira coluna da tabela mostra as sementes utilizadas nos experimentos. Na segunda coluna está o menor caminho encontrado com a semente da coluna anterior. A terceira coluna mostra o tempo em horas, minutos e segundos para encontrar o menor caminho. A quarta coluna mostra

2 O POWER2 foi a segunda geração do projeto da arquitetura POWER, e caracterizou-se por ser um projeto inovador para a época. Seu desempenho foi significativamente superior e mantinha a compatibilidade com a arquitetura POWER RS1.

3 Sistema operacional UNIX *like* desenvolvido pela IBM.

a porcentagem da diferença entre o menor caminho encontrado e o valor do caminho ótimo.

| Semente | Comprimento | Tempo | % Ótimo |
|---|---|---|---|
| 0,0009 | 683,17707 | 2:48:48 | 8,61321 |
| 0,0070 | 664,97518 | 2:46:56 | 5,71942 |
| 0,0120 | 683,18082 | 2:48:01 | 8,61380 |
| 0,0210 | 668,13851 | 2:46:50 | 6,22234 |
| **0,0290** | **663,66383** | **2:46:45** | **5,51094** |
| 0,0380 | 680,09888 | 2:48:40 | 8,12383 |
| 0,0510 | 687,22132 | 2:48:09 | 9,25617 |
| 0,0740 | 692,04696 | 2:47:39 | 10,02336 |
| 0,0830 | 668,97857 | 2:47:27 | 6,35589 |
| 0,0890 | 687,44713 | 2:48:55 | 9,29207 |
| 0,1020 | 697,32683 | 2:49:51 | 10,86277 |
| 0,1230 | 683,02266 | 2:47:34 | 8,588659 |
| 0,1520 | 689,87209 | 2:48:55 | 9,677598 |
| 0,1670 | 682,38520 | 2:48:23 | 8,487312 |
| 0,1760 | 682,98646 | 2:48:38 | 8,582903 |
| 0,1900 | 678,97124 | 2:48:59 | 7,944553 |
| 0,2410 | 678,70606 | 2:49:42 | 7,902395 |
| **0,3150** | 670,70301 | **2:46:24** | 6,630050 |
| 0,3660 | 702,07533 | 2:49:10 | 11,617699 |
| 0,3960 | 682,61005 | 2:48:45 | 8,523060 |
| 0,4850 | 674,50842 | 2:50:04 | 7,235043 |
| 0,5860 | 671,93819 | 2:47:07 | 6,826421 |
| 0,6060 | 692,86203 | 2:49:08 | 10,152946 |
| 0,7080 | 681,79398 | 2:48:59 | 8,393320 |
| 0,8590 | 687,07925 | 2:48:08 | 9,233584 |
| 0,9200 | 687,49986 | 2:48:50 | 9,300455 |
| 0,9870 | 681,55429 | 2:49:38 | 8,355213 |

**Tabela 1. Resultados obtidos para 1 processador**

Verificamos na Tabela 1 que o menor caminho foi encontrado com a semente 0,0290, cujo caminho teve comprimento de aproximadamente 663,66. O melhor caminho para esta instância [1] tem 629, o que nos dá uma distância de 5,5% de diferença em relação ao valor ótimo. Observamos, também, que a semente que gerou o melhor tempo de execução foi igual a 0,3150.

Para os experimentos da versão paralela PTSP_AS utilizamos 2, 4 e 6 processadores. Os resultados obtidos estão mostrados nas Tabelas 2, 3 e 4. Nessas tabelas, primeira e segunda colunas mostram as sementes e o tamanho do menor caminho encontrado com essas sementes. A terceira coluna mostra o tempo de execução. A quarta e quinta colunas mostram a porcentagem do desvio médio do tempo de 3 execuções e da diferença entre o menor caminho encontrado e o valor do caminho ótimo.

Todos os teste foram realizados com o sistema SP2 dedicado à execução da aplicação PTSP_AS.

As execuções da aplicação PTSP_AS em 1, 2, 4, e 6 processadores gerou diferentes tamanhos de percursos. Este fato ocorre porque na versão paralela existe um conjunto distinto de formigas percorrendo as cidades simultaneamente. Além disso, na execução com mais de um processador, cada processo gera números aleatórios diferentes daqueles gerados pela versão seqüencial.

| Semente | Comprimento | Tempo | Desvio % | % Ótimo |
|---|---|---|---|---|
| 0,0009 | 677,47070 | 1:33:43 | 0,3 | 7,705994 |
| 0,0070 | 676,28496 | 1:34:00 | 0,3 | 7,517482 |
| 0,0120 | 677,50690 | 1:33:18 | 0,3 | 7,711750 |
| **0,0210** | **657,95304** | **1:32:52** | **0,4** | **4,603027** |
| **0,0290** | **657,95304** | **1:33:36** | **0,3** | **4,603027** |
| 0,0380 | 689,74350 | 1:32:14 | 0,3 | 9,657154 |
| 0,0510 | 689,17995 | 1:33:23 | 0,1 | 9,567559 |
| 0,0740 | 683,12010 | 1:33:52 | 0,1 | 8,604151 |
| 0,0830 | 689,42195 | 1:32:36 | 0,0 | 9,606034 |
| 0,0890 | 674,63683 | 1:32:57 | 0,3 | 7,255458 |
| 0,1020 | 667,42882 | 1:34:01 | 0,1 | 6,109511 |
| 0,1230 | 677,74500 | 1:32:44 | 0,3 | 7,749603 |
| 0,1520 | 678,76433 | 1:33:11 | 0,3 | 7,911659 |
| 0,1670 | 684,48814 | 1:32:44 | 0,4 | 8,821645 |
| 0,1760 | 682,10669 | 1:33:02 | 0,3 | 8,443034 |
| 0,1900 | 682,38520 | 1:32:56 | 0,0 | 8,487312 |
| 0,2410 | 684,22976 | 1:33:46 | 0,1 | 8,780567 |
| 0,3150 | 677,50690 | 1:33:09 | 0,3 | 7,711750 |
| 0,3660 | 685,69375 | 1:33:07 | 0,3 | 9,013315 |
| 0,3960 | 683,97734 | 1:38:48 | 0,2 | 8,740436 |
| 0,4850 | 675,70338 | 1:33:24 | 0,1 | 7,425021 |
| 0,5860 | 685,06601 | 1:43:03 | 0,2 | 8,913514 |
| 0,6060 | 674,15869 | 1:32:53 | 0,2 | 7,179441 |
| 0,7080 | 693,04689 | 1:33:09 | 0,3 | 10,18234 |
| **0,8590** | 670,28285 | **1:31:58** | 0,0 | 6,563251 |
| 0,9200 | 678,76433 | 1:33:30 | 0,2 | 7,911659 |
| 0,9870 | 675,63763 | 1:32:42 | 0,3 | 7,414567 |

**Tabela 2. Resultados obtidos para 2 processadores**

A Tabela 2 mostra as informações relativas à execução da aplicação em 2 processadores. O menor caminho foi encontrado com as sementes 0,0210 e 0,0290, sendo que a semente 0,0210 teve um tempo de execução ligeiramente menor. A semente 0,8590 foi a que gerou o melhor tempo de execução.

Resumindo, a melhor solução encontrada foi gerada a partir das sementes 0,0210 e 0,0290. Ambas geraram resultados semelhantes, com tamanho de percurso igual a 657,95 e cujo valor é o mais próximo do ótimo, aproximadamente 4,6%.

Note que com 2 processadores o tempo de execução é significativamente menor se comparado com a versão seqüencial, sendo aproximadamente 1:14h mais rápido, com a melhor solução encontrada.

| Semente | Comprimento | Tempo | Desvio % | % Ótimo |
|---|---|---|---|---|
| **0,0009** | **664,9752** | **1:04:39** | **0,5** | **5,719424** |
| 0,0070 | 690,2311 | 1:05:24 | 0,5 | 9,734678 |
| 0,0120 | 682,4252 | 1:04:25 | 0,4 | 8,493666 |
| 0,0210 | 679,2355 | 1:05:16 | 0,6 | 7,986569 |
| 0,0290 | 684,1886 | 1:04:47 | 0,3 | 8,774017 |
| 0,0380 | 692,9071 | 1:05:40 | 0,4 | 10,160117 |
| 0,0510 | 668,4063 | 1:05:11 | 0,1 | 6,264916 |
| 0,0740 | 691,5979 | 1:05:22 | 0,1 | 9,951963 |
| 0,0830 | 674,0360 | 1:04:39 | 0,3 | 7,159938 |
| 0,0890 | 681,7483 | 1:05:13 | 0,4 | 8,386060 |
| **0,1020** | 666,1830 | **1:04:18** | 0,2 | 5,911453 |
| 0,1230 | 677,5069 | 1:04:53 | 0,5 | 7,711750 |
| 0,1520 | 699,7701 | 1:05:01 | 0,4 | 11,251206 |
| 0,1670 | 683,0227 | 1:05:17 | 0,5 | 8,588659 |
| 0,1760 | 681,8392 | 1:05:46 | 0,4 | 8,400500 |
| 0,1900 | 679,8810 | 1:05:59 | 0,3 | 8,089193 |
| 0,2410 | 677,0164 | 1:05:18 | 0,2 | 7,633762 |
| 0,3150 | 677,5069 | 1:05:10 | 0,3 | 7,711750 |
| 0,3660 | 675,2937 | 1:04:36 | 0,4 | 7,359890 |
| 0,3960 | 679,1833 | 1:05:25 | 0,3 | 7,978268 |
| 0,4850 | 674,9032 | 1:04:53 | 0,3 | 7,297812 |
| 0,5860 | 674,6971 | 1:05:04 | 0,2 | 7,265043 |
| 0,6060 | 682,7033 | 1:05:01 | 0,4 | 8,537886 |
| 0,7080 | 689,3508 | 1:07:54 | 0,3 | 9,594722 |
| 0,8590 | 684,1645 | 1:05:15 | 0,1 | 8,770189 |
| 0,9200 | 679,4274 | 1:05:33 | 0,3 | 8,017078 |
| 0,9870 | 689,1432 | 1:05:32 | 0,5 | 9,561720 |

**Tabela 3. Resultados obtidos para 4 processadores**

A Tabela 3 mostra os resultados obtidos na execução da aplicação em 4 processadores. O menor caminho foi encontrado com a semente 0,0009, com tamanho de percurso igual a 664,97, cujo valor é o mais próximo do ótimo. A semente que proporcionou o melhor tempo de execução foi a de 0,1020.

Observe que com 4 processadores os tempos de execução são menores se comparados com a versão seqüencial e executada em 2 processadores. Porém, a melhor solução encontrada com 4 processadores é pior do que as melhores soluções da versão seqüencial ou da executada em 2 processadores.

Este fato demonstra que o componente aleatório da heurística pode proporcionar resultados não satisfatórios.

| Semente | Comprimento | Tempo | Desvio % | % Ótimo |
|---|---|---|---|---|
| **0,0009** | 677,47070 | **0:57:17** | 0,6 | 7,705994 |
| 0,0070 | 676,2850 | 0:57:36 | 0,5 | 7,517482 |
| 0,0120 | 677,5069 | 0:57:39 | 0,5 | 7,711750 |
| **0,0210** | **657,9530** | **0:57:56** | **0,5** | **4,603027** |
| 0,0290 | 689,7435 | 0:58:12 | 0,5 | 9,657154 |
| 0,0380 | 689,1800 | 0:58:07 | 0,1 | 9,567559 |
| 0,0510 | 683,1201 | 0:58:01 | 0,1 | 8,604151 |
| 0,0740 | 689,4220 | 0:58:05 | 0,1 | 9,606034 |
| 0,0830 | 674,6368 | 0:57:53 | 0,4 | 7,255458 |
| 0,0890 | 667,4288 | 0:57:53 | 0,4 | 6,109511 |
| 0,1020 | 677,7450 | 0:58:34 | 0,5 | 7,749603 |
| 0,1230 | 678,7643 | 0:57:29 | 0,5 | 7,911659 |
| 0,1520 | 684,4881 | 0:57:55 | 0,5 | 8,821645 |
| 0,1670 | 682,1067 | 0:58:26 | 0,6 | 8,443034 |
| 0,1760 | 682,3852 | 0:58:03 | 0,0 | 8,487312 |
| 0,1900 | 684,2298 | 0:57:57 | 0,6 | 8,780567 |
| 0,2410 | 677,5069 | 0:58:19 | 0,6 | 7,711750 |
| 0,3150 | 685,6938 | 0:57:51 | 0,6 | 9,013315 |
| 0,3660 | 683,9773 | 0:57:59 | 0,1 | 8,740436 |
| 0,3960 | 675,7034 | 0:57:49 | 0,2 | 7,425021 |
| 0,4850 | 685,0660 | 0:58:30 | 0,3 | 8,913514 |
| 0,5860 | 674,1587 | 0:57:44 | 0,4 | 7,179441 |
| 0,6060 | 693,0469 | 0:57:51 | 0,4 | 10,182334 |
| 0,7080 | 670,2829 | 0:58:01 | 0,1 | 6,563251 |
| 0,8590 | 678,7643 | 0:58:13 | 0,3 | 7,911659 |
| 0,9200 | 675,6376 | 0:57:54 | 0,5 | 7,414567 |
| 0,9870 | 683,8049 | 0:58:36 | 0,5 | 8,713018 |

**Tabela 4. Resultados obtidos para 6 processadores**

A Tabela 4 mostra os resultados obtidos com a execução em 6 processadores. O menor caminho foi encontrado com a semente 0,0210, com tamanho de percurso igual a 657,95 e cujo valor é o mais próximo do ótimo (4,6%). A semente 0,0009 foi a que gerou o melhor tempo de execução.

Entre todos os testes realizados, o que demonstrou obter o melhor resultado, isto é, o menor caminho com o melhor tempo de execução, foi o realizado com 6 processadores

### 5.3 Discussão

**Qualidade da solução aproximada obtida**

Como citado anteriormente, para a instância TSBLIB de 101 cidades o menor caminho encontrado é igual a 629. Em nossa implementação, o sistema encontrou o menor caminho com aproximadamente 657,95. Este valor é inferior a 5% de distância do valor ótimo, e pode ser considerado uma boa aproximação.

***Speedup***

Para o cálculo do *speedup* criamos a Tabela 5, a partir das informações das Tabelas 1, 2, 3 e 4. Esta tabela contém, para cada semente, os tempos médios de execução em segundos e o *speedup* para cada execução das aplicações em 2, 4 e 6 processadores.

A primeira coluna da Tabela 5 mostra as sementes usadas em cada experimento. A segunda, terceira, quarta e a quinta colunas mostram os tempos médios de execução da aplicação para 1, 2, 4 e 6 processadores. A sétima, oitava e nona colunas mostram o *speedup* de 2, 4 e 6 processadores, respectivamente.

A versão paralela da aplicação PTSP_AS gerou comprimentos de caminhos diferentes para 2, 4 e 6 processadores. Este fato ocorre porque várias formigas caminham entre as cidades, simultaneamente, e cada processo gera números aleatórios distintos.

O algoritmo baseado nas colônias de formigas encontra, em geral, valores aproximados. Em nossos experimentos encontramos valores que podemos considerar muito satisfatório. Com a heurística baseada na colônia de formigas, eventualmente, o algoritmo pode encontrar o menor caminho igual ao do valor ótimo.

De acordo com os experimentos realizados, a aplicação PTSP_AS utilizando 6 processadores foi a que gerou os melhores resultados (menor caminho, melhor tempo) com um *speedup* de 2,9 em relação á versão seqüencial. Porém, o sistema se mostrou mais eficiente (90%) nas execuções com 2 processadores, melhor relação entre *speedup* e quantidade de processadores.

Com o aumento do número de processadores, o *speedup* não aumenta proporcionalmente. Quanto maior o número de processadores, mais informações devem ser trocadas entre os processos. O tempo gasto com a comunicação pode se tornar maior do que o tempo de computação útil para processar os dados em cada processador. Isso ocorre porque a massa de dados utilizada nos experimentos não possui paralelismo suficiente.

Para aumentar a eficiência obtida em 6 processadores é necessário utilizar um problema com maior número de cidades.

| **Semente** | **Tempo por Processador** | | | | ***Speedup*** | | |
|---|---|---|---|---|---|---|---|
| | **1** | **2** | **4** | **6** | **2** | **4** | **6** |
| 0,0009 | 10128 | 5623 | 3879 | 3437 | 1,80 | 2,61 | 2,95 |
| 0,0070 | 10016 | 5640 | 3924 | 3456 | 1,78 | 2,55 | 2,90 |
| 0,0120 | 10081 | 5598 | 3865 | 3459 | 1,80 | 2,61 | 2,91 |
| **0,0210** | **10010** | **5572** | **3916** | **3476** | **1,80** | **2,56** | **2,88** |
| 0,0290 | 10005 | 5616 | 3887 | 3492 | 1,78 | 2,57 | 2,87 |
| 0,0380 | 10120 | 5534 | 3940 | 3487 | 1,83 | 2,57 | 2,90 |
| 0,0510 | 10089 | 5603 | 3911 | 3481 | 1,80 | 2,58 | 2,90 |
| 0,0740 | 10059 | 5632 | 3922 | 3485 | 1,7 | 2,57 | 2,89 |
| 0,0830 | 10047 | 5556 | 3879 | 3473 | 1,81 | 2,59 | 2,89 |
| 0,0890 | 10135 | 5577 | 3913 | 3473 | 1,82 | 2,59 | 2,92 |
| 0,1020 | 10191 | 5641 | 3858 | 3514 | 1,81 | 2,64 | 2,90 |
| 0,1230 | 10054 | 5564 | 3893 | 3449 | 1,81 | 2,58 | 2,92 |
| 0,1520 | 10135 | 5591 | 3901 | 3475 | 1,81 | 2,60 | 2,92 |
| 0,1670 | 10103 | 5564 | 3917 | 3506 | 1,82 | 2,58 | 2,88 |
| 0,1760 | 10118 | 5582 | 3946 | 3483 | 1,81 | 2,56 | 2,91 |
| 0,1900 | 10139 | 5576 | 3899 | 3477 | 1,82 | 2,60 | 2,92 |
| 0,2410 | 10182 | 5626 | 3918 | 3499 | 1,81 | 2,60 | 2,91 |
| 0,3150 | 9984 | 5589 | 3910 | 3471 | 1,79 | 2,56 | 2,88 |
| 0,3660 | 10150 | 5587 | 3876 | 3479 | 1,82 | 2,62 | 2,92 |
| 0,3960 | 10125 | 5928 | 3925 | 3469 | 1,71 | 2,58 | 2,92 |
| 0,4850 | 10204 | 5604 | 3893 | 3510 | 1,82 | 2,62 | 2,91 |
| 0,5860 | 10027 | 6183 | 3904 | 3464 | 1,62 | 2,57 | 2,90 |
| 0,6060 | 10148 | 5573 | 3901 | 3471 | 1,82 | 2,60 | 2,92 |
| 0,7080 | 10139 | 5589 | 4074 | 3481 | 1,81 | 2,49 | 2,91 |
| 0,8590 | 10088 | 5518 | 3915 | 3493 | 1,83 | 2,58 | 2,89 |
| 0,9200 | 10130 | 5610 | 3933 | 3474 | 1,81 | 2,58 | 2,92 |
| 0,9870 | 10178 | 5562 | 3932 | 3516 | 1,83 | 2,59 | 2,90 |
| ***Speedup* Médio** | | | | | **1,80** | **2,58** | **2,90** |
| **Desvio Médio Percentual** | | | | | **1,35** | **0,79** | **0,48** |
| **Eficiência (%)** | | | | | **90,0** | **64,5** | **48,3** |

**Tabela 5. *Speedup* para 2, 4 e 6 processadores**

## 6. Conclusões e Trabalhos Futuros

O uso de diferentes heurísticas para encontrar a solução aproximada de problemas difíceis de otimização combinatória tem sido exaustivamente investigado. Os resultados obtidos têm se mostrado eficazes. Muitos deles utilizam algoritmos polinomiais que encontram boas soluções, próximas ao valor ótimo. A paralelização de aplicações tem demonstrado ser uma solução viável para melhorar o desempenho dessas heurísticas, tanto em

tempo computacional quanto na obtenção de uma solução aproximada eficaz.

Neste trabalho desenvolvemos uma aplicação paralela, PTSP_AS, utilizando o problema do caixeiro viajante baseado na heurística *Ant System*. O algoritmo AS é uma das técnicas, propostas na literatura, para encontrar soluções aproximadas de otimização discreta, como por exemplo o caminho mínimo e controle de distribuição. Esta técnica, proposta por Marco Dorigo, é inspirada no comportamento de colônias de formigas reais e consiste em criar uma colônia virtual de formigas. O algoritmo AS possui complexidade de tempo de execução polinomial. Porém, o seu desempenho pode ser significativamente reduzido se for estabelecido um número muito grande de cidades e que a paralelização demonstre ser uma solução viável.

Utilizamos a biblioteca MPI, para a paralelização do algoritmo. Esta biblioteca possui implementações para várias arquiteturas e um padrão bem definido. Por essa razão as aplicações são portáteis entre diversas arquiteturas, sem a necessidade de adaptação do código fonte. O desenvolvimento do código fonte de uma aplicação paralela baseado no modelo de programação de passagem de mensagem exige um esforço extra do programador. O programador passa a ser responsável pelos procedimentos de sincronismo e distribuição de carga de trabalho da aplicação. Estes procedimentos não são necessários em uma versão seqüencial. Além disso, o programador, também, é responsável por obter um equilíbrio na aplicação com relação à computação útil e o suporte a paralelização. Quanto menor a quantidade de troca de mensagens melhor será o desempenho da aplicação. Se a troca de mensagens entre os processos consumir um tempo de execução maior do que o tempo de processamento útil, isso inviabiliza a utilização da aplicação paralela.

A versão da PTSP_AS foi projetada com o intuito de diminuir o máximo a troca de mensagens entre os processos. Cada processo da aplicação executa sua tarefa de forma independente, trocando poucas mensagens por iteração, o que possibilitou, no geral, um ganho de desempenho bem expressivo.

A versão paralela da aplicação AS mostrou resultados diferentes em relação à versão seqüencial. Este fato ocorre devido à geração independente dos números aleatórios em cada processo. Na versão paralela cada processo gera seus próprios números aleatórios. A mesma formiga da versão seqüencial pode ter um comportamento distinto, isso porque ela é afetada por um número aleatório diferente em cada processo. A aplicação PTSP_AS mantém os mesmos conceitos heurísticos de sua versão seqüencial. Ela não só obteve um melhor tempo de execução em 2, 4 e 6 processadores, como, também, encontrou um caminho menor do que a versão seqüencial, quando foi executado em 6 processadores,.

As técnicas utilizadas na paralelização da versão seqüencial forneceram bons resultados nos experimentos realizados. Os *speedups* médios obtidos são iguais a 1,8, 2,6 e 2,9 para 2, 4 e 6 processadores, respectivamente.

Como sugestões para futuros trabalhos consideramos interessante complementar a análise dos resultados executando a aplicação num *cluster* de PCs e, também, avaliar o tempo ocioso de cada processo. Dessa forma, poderíamos descobrir uma possível fonte de desbalanceamento de carga e otimizar a aplicação paralela reduzindo ainda mais seu tempo total de execução. Por exemplo, poderíamos implementar uma outra forma de distribuição de carga entre os processos. A distribuição de carga de trabalho na versão PTSP_AS atual é estática. Este pode ser um fator contribua para a ociosidade nos processos.

Uma outra sugestão é avaliar a aplicação com um número maior de cidades.

## Referências

[1]A Biblioteca TSPLIB, acesso 2004. http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/

[2] Bennett, M. *Inside the IBM RISC System/6000*. McGraw-Hill, 1994.

[3]Bodin, L., Golden, B., Assad, A. e Ball, M., "Routing and Scheduling of Vehicles and Crews: the State of the Art". *Special Issue*. England: Pergamon Press, 1983.

[4]Campos, L., Otimização Combinatorial Inspirada em Colônias de Formigas Aplicada ao Problema do Caixeiro Viajante, Monografia de Final de Curso, Bacharelado em Ciência da Computação, Juiz de Fora, 2000. 74p.

[5] Cormen, T., *et al*, *Algoritmos Teoria e Prática,* 2ª ed., Rio de Janeiro: Campus, 2001, 810p.

[6]Dorigo, M. Optimization, Lerning and Natuaral Algorithms. Ph.D Thesis, Politécnico de Milario, Italy, 1992.

[7]Dorigo, M., Di Caro, G., Gambardella, L.M., "Ant Algorithms for Discrete Optimization", *Artificial Life*, IRIDIA/98-10, Vol.5, No.3, pp. 137-172, 1999.

[8]Garey, M. R. e Johnson, D. S., *Computers and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[9]Lawler, E. L. e Lenstra, J. K., *The Traveling Salesman Problem*, John Wiley, 1985.

[10] Loudon, K., *Dominando Algoritmos com C*, 1ª ed., Rio de Janeiro, 1999.

[11]Patterson, D., Hennessy, J. Organização e Projeto de Computadores: A Interface Hardware/Software, 2ª ed. Rio de Janeiro: LTC, 2000, 551p.

[12]PVM: Parallel Virtual Machine *home page*, acesso 2005, http://www.csm.ornl.gov/pvm/pvm_home.html

[13]Reinelt, G., *The Traveling Salesman: Computational Solutions for TSP Applications*, Springer-Verlag, 1994.

[14]Snir, M, Otto, S.M., Huss-Lederman, S., Dongarra, J., MPI: The Complete Reference, The MIT Press, 1996.

[15]Wilkinson, B., Allen, M. Parallel Programming Techniques and Applications using Networked Workstations and Parallel Computers, 1ª ed, New Jersey, 1999, 430p.