UNIVERSIDADE FEDERAL DE SANTA CATARINA

PARALELIZAÇÃO DO ALGORITMO GENÉTICO COM SIMULATED ANNEALING, APLICADO SOBRE O PROBLEMA DO CAIXEIRO VIAJANTE.

Tiago Steinmetz Soares

Florianópolis

Outubro de 2009

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

PARALELIZAÇÃO DO ALGORITMO GENÉTICO COM SIMULATED ANNEALING, APLICADO SOBRE O PROBLEMA DO CAIXEIRO VIAJANTE.

Trabalho de conclusão de curso como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Tiago Steinmetz Soares

Orientador:

Profº Dr. José Mazzuco Jr.

Florianópolis

Outubro de 2009

Titulo: *Paralelização do Algoritmo Genético com Simulated Annealing, aplicado sobre o problema do caixeiro viajante.*

Autor: *Tiago Steinmetz Soares*

Banca Examinadora:

_____

Profº Dr. José Mazzuco Jr.
Universidade Federal de Santa Catarina - UFSC

_____

Profº Dr. Frank Augusto Siqueira
Universidade Federal de Santa Catarina - UFSC

_____

Profº Dr. Mário Antônio Ribeiro Dantas
Universidade Federal de Santa Catarina - UFSC

AGRADECIMENTOS

Gostaria muito de agradecer primeiramente ao professor José Mazzuco Jr, pela sua atenção e dedicação a esse projeto. À minha família, que me apoiaram esses anos na graduação. Por fim aos meus colegas de curso, por me acompanharem durante essa fase da minha vida.

“*Tudo o que começa, uma hora termina”*

LISTA DE FIGURAS

LISTA DE TABELAS

.

SUMÁRIO

LISTA DE ABREVIAÇÕES E SIGLAS

AG	Algoritmo Genético
SA	Simulated Annealing
PMX	Partially Mapped Crossover
GPS	Global Positioning System
VSA	Vetor de Soma Acumulada

RESUMO

Este trabalho tem como objetivo principal explorar a computação paralela através da utilização de máquinas simples, mono processadas, interligadas por uma rede. Para tanto, será desenvolvida uma abordagem paralela para algoritmos de otimização combinatória conhecidos como algoritmos genéticos.

A arquitetura modelada está baseada nos conceitos de programação orientada a objetos, com utilização de threads e monitores segundo a conceituação apregoar pela linguagem de programação Java. O algoritmo genético paralelo está modelado segundo o conceito de populações cooperantes. Nesse modelo, mestre-escravos, cada máquina individualmente (o escravo) é responsável por processar uma população com N indivíduos e a troca de informações se realiza através da migração de indivíduos para uma máquina central (o mestre) que coordena todo o processamento. Nessa máquina central existe, para cada máquina escrava, uma thread que faz o trabalho de comunicação entre elas, essas threads são sincronizadas através de monitores oferecidos pela linguagem Java. Os escravos foram modelados como unidades autônomas com a responsabilidade da aplicação dos operadores genéticos (crossover e mutação) e pela determinação da aptidão de cada indivíduo.

Algoritmo Genético (AG) é uma técnica de otimização estocástica inspirada no processo evolutivo e na genética (Goldberg, 1989). Nesta técnica, parte-se de uma população inicial de soluções candidatas geradas aleatoriamente que são avaliadas de forma a se determinar a qualidade dessas soluções. Baseado nesta avaliação, um processo de seleção separa um sub-conjunto desta população para servir de base para a geração de uma nova população de soluções candidatas. Esta nova população é obtida a partir da aplicação de operadores genéticos sobre o sub-conjunto selecionado. Assim, o processo segue interagindo e ao longo das iterações, a população evolui até que se chegue a soluções consideradas satisfatórias.

Assim sendo, em uma modelagem utilizando AG, o espaço de soluções possíveis de um determinado problema é dividido em partes chamadas de população e quanto mais difícil for esse problema, maior deve ser suas populações. Nessa abordagem paralela, uma grande população é dividida em pequenas populações cooperantes que são

executadas em paralelo. Para avaliar o desempenho desta nova abordagem, foram realizados testes utilizando um laboratório de ensino de programação contendo máquinas simples interligadas através de rede. Nos testes do modelo proposto foram utilizadas instâncias de um dos mais conhecidos benchmarks na área de otimização combinatória que é o problema do caixeiro viajante. Os resultados obtidos estão descritos neste trabalho.

ABSTRACT

This paper's main objective is to explore parallel computing through the use of simple machines, mono processed, linked by a network. Therefore, it has a parallel approach for combinatorial optimization algorithms known as genetic algorithms.

The architecture modeled is based on the concepts of object-oriented programming, using threads and monitors based on the concept proclaim the Java programming language. The parallel genetic algorithm is modeled after the concept of cooperating populations. In this model, master-slave, each individual machine (slave) is responsible for processing a population with N individuals and exchange of information takes place through the migration of individuals to a central machine (the master) which coordinates all the processing. This machine is central to each slave machine, a thread that does the work of communication between them, these threads are synchronized with monitors offered by Java. The slaves were modeled as autonomous units with the responsibility of the application of genetic operators (crossover and mutation) and by determining the fitness of each individual.

Genetic Algorithm (GA) is a stochastic optimization technique inspired by the evolutionary process and genetics (Goldberg, 1989). In this technique, is part of an initial population of randomly generated candidate solutions are evaluated to determine the quality of these solutions. Based on this assessment, a screening process separates a subset of this population as the basis for the generation of a new population of candidate solutions. This new population is obtained from the application of genetic operators on the subset selected. Thus, the proceedings and iterate over the iterations, the population evolves until it reaches solutions satisfactory.

Thus, in a model with GA, the space of possible solutions to a problem is divided into parts called the population and the more difficult for this problem, the greater should be its people. In this parallel approach, a large population is divided into small cooperative populations that are performed in parallel. To evaluate the performance of this new approach, tests were performed using a laboratory teaching program containing simple machines connected through network. In testing the proposed model were used instances of one of the best known benchmarks in the field of combinatorial optimization is the traveling salesman problem. The results obtained are described in this work.

1 INTRODUÇÃO

A pesquisa por algoritmos evolutivos aplicados em problemas de otimização combinatória tem se intensificado nos últimos tempos como métodos alternativos aos modelos matemáticos que soluciona de forma exata esse tipo de problema. A inconveniência do emprego de um modelo matemático na solução de um determinado problema consiste na sua grande dependência ao particular problema tratado, assim como, em muitos casos, apresenta tempo de execução extremamente proibitivo. Nesse sentido, novas técnicas da computação evolutiva, tais como, *Simulated Annealing*, *Tabu Search* e Algoritmos Genéticos, têm se apresentado como alternativas bastante otimistas, mesmo considerando que seus resultados são aproximados, não exatos.

A computação paralela consiste, basicamente, na utilização de elementos de processamento, que cooperam e comunicam-se entre si para solucionarem problemas complexos, de maneira mais rápida que a computação seqüencial.

A escolha do AG na construção do modelo proposto justifica-se pelo fato de que o mesmo apresenta-se como intrinsecamente paralelo. A geração e avaliação dos novos indivíduos durante o processo iterativo de evolução para a solução ótima podem ser executadas em paralelo. Dessa forma pode ser possível reduzir o tempo de execução adicionando mais processadores. Se por um lado a adição de mais processadores aumenta o poder computacional, sem aumentar o tempo de execução, por outro, com a execução paralela torna-se possível avaliar e cruzar vários indivíduos simultaneamente. O presente trabalho enfoca o uso da computação paralela como forma de redução do tempo de execução, aceleração da convergência e melhoria nas soluções encontradas através de métodos baseados em algoritmos genéticos.

1.1 OBJETIVOS GERAIS

Este trabalho tem como objetivo principal explorar a computação paralela através da utilização de um cluster de máquinas simples, mono processadas. Essa exploração se dará através da transformação de um algoritmo genético seqüencial em um paralelo e da instigando do seu comportamento através de comparações de resultados obtidos.

1.2 OBJETIVOS ESPECÍFICOS

- Introdução aos algoritmos usados para o estudo
- Apresentação de um modelo simples dos algoritmos AG e SA
- Desenvolvimento do algoritmo genético com simulated annealing
- Comparação dos resultados do AG e SA com o algoritmo genético simples
- Paralelização do AG e AS
- Apresentação das classes responsáveis por cada função do algoritmo
- Comparação dos testes de desempenho e qualidade dos resultados

1.3 METODOLOGIA

A metodologia empregada neste trabalho foi:

- Inicialmente uma pesquisa bibliográfica sobre os algoritmos em questão e sobre o problema do caixeiro viajante.
- Desenvolvimento dos algoritmos na linguagem JAVA
- Pesquisa sobre a paralelização do Algoritmo Genético
- Desenvolver a paralelização utilizando “sockets” de JAVA
- Testes sobre o problema.

Esta dissertação está organizada em 7 capítulos. O capítulo 2 aborda o Algoritmo genético através de uma introdução teórica, e em seguida, fornecendo uma visão geral de sua elaboração. No capítulo 3 trata do algoritmo Simulated Annealing, abordando da mesma maneira do capitulo 2, com uma introdução e elaboração do algoritmo. As propostas dos algoritmos AG e SA são descritos nos capítulos 4 e 5, respectivamente. No capítulo 6 são demonstrados os resultados experimentais, e por fim, no capítulo 7, a conclusão e trabalho futuros.

1.4 LINGUAGENS DE PROGRAMAÇÃO

A linguagem utilizada para este trabalho foi a linguagem Java que oferece ferramentas simples tanto para a programação concorrente entre suas threads como também ferramentas para programação distribuída através de comunicação entre máquinas.

1.5 MOTIVAÇÕES

Este trabalho foi motivado pelo interesse em explorar a potencialidade em termos de capacidade de processamento numérico que uma simples infra-estrutura com alguns computadores comuns conectados via rede, cabeada ou não, pode apresentar. Foi escolhido o algoritmo genético como um meio de se explorar o poder do paralelismo dessas máquinas.

2 ALGORITMO GENÉTICO

2.1 INTRODUÇÃO

O algoritmo genético é uma forma de computação evolutiva, baseados nos mecanismos de evolução natural, na genética, na hereditariedade e preservações de características de indivíduos pela “sobrevivência” dos membros mais aptos ao ambiente. Uma população de possíveis soluções para o problema em questão evolui de acordo com operadores probabilísticos concebidos a partir de uma função, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua.

O AG é classificado como um método fraco de resolução de problemas, que são concebidos para resolverem problemas genéricos em um mundo genérico. São operados em mundos não-lineares e não-estacionários, não garantem eficiência total na obtenção da solução, mas geralmente garantem uma boa aproximação. Faz parte de uma família de modelos computacionais inspirados na evolução, que incorporam uma solução potencial para um problema específico numa estrutura semelhante à de um cromossomo e aplicam operadores de seleção e "crossover" a essas estruturas de forma a preservar informações críticas relativas à solução do problema.

Uma das vantagens de um algoritmo genético é a simplificação que eles permitem na formulação e solução de problemas de otimização. São numericamente robustos, ou seja, não são sensíveis a erros de arredondamento no que se refere aos seus resultados finais. [10]

2.2 ALGORITMO

AG opera sobre uma população que consiste em um subconjunto de soluções do espaço de soluções do problema que está sendo tratado. Normalmente essa população é gerada aleatoriamente. Para cada indivíduo da população, o AG necessita conhecer a qualidade do mesmo, ou seja, o quanto aquela solução se aproxima da solução do problema. Esse valor, conhecido como fitness do individuo, é determinado por uma função chamada função objetivo. Quanto maior for o seu fitness maior será a chance daquele indivíduo sobreviver e reproduzir, os indivíduos com fitness ruins também têm chances de reproduzir. Após a determinação do fitness de cada indivíduo, são emulados os processos de seleção natural. A seleção natural se processa através de algum método probabilístico normalmente baseado na média das qualidades dos indivíduos daquela população.

Como já foi dito, indivíduos adequados, ou seja, com maior fitness terão grandes chances de sobreviverem, porém, como é uma função probabilística, membros fracos daquela população também permanecem uma vez que estes podem conter algum componente essencial para a solução. Concluído o processo de seleção, o operador genético de cruzamento é utilizado. É um processo onde ocorre a troca de fragmentos entre pares de cromossomos (indivíduo). Estes fragmentos são determinados por pontos de divisão, que podem ser determinados aleatoriamente.

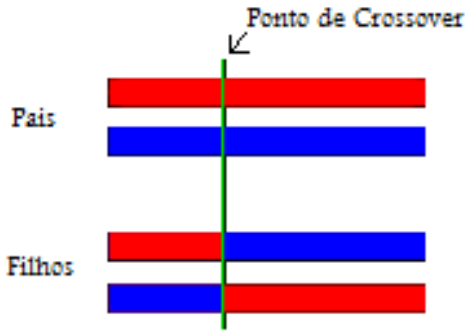

Figura 1 - Crossover

A operação de crossover é também um processo probabilístico, ou seja, não é utilizada sobre todos os indivíduos daquela população. Após o processo de cruzamento, ainda

pode ser utilizado outro operador genético, outra vez, probabilístico, que é a mutação. Nessa última operação, envolvendo um único individuo, seleciona-se uma posição aleatoriamente do cromossomo e se altera o valor do gene correspondente.

Figura 2 - Mutação

O mecanismo de mutação é um processo importante, pois a reprodução após uma série de iterações tende a estacionar quando se aproxima de um valor ótimo local, a operação de crossover começa a não mais ter efeito uma vez que os cromossomos se tornam muito semelhantes.

O encerramento do processamento de um algoritmo genético pode ocorrer de diversas formas. Na prática determina-se um número máximo de gerações ou um tempo limite de processamento. Outro critério é a idéia de estagnação, o algoritmo tende a parar quando não se observa melhoria na população após um número determinado de iterações.

3 SIMULATED ANNEALING

3.1 INTRODUÇÃO

O nome recozimento (*annealing*) é dado ao processo de aquecimento de um sólido até o seu ponto de fusão, seguido de um resfriamento gradual e vagaroso, até que se alcance novamente o seu enrijecimento. Nesse processo, o resfriamento vagaroso é essencial para se manter um equilíbrio térmico no qual os átomos encontrarão tempo suficiente para se organizarem em uma estrutura uniforme com energia mínima. Se o sólido é resfriado bruscamente, seus átomos formarão uma estrutura irregular e fraca, com alta energia em conseqüência do esforço interno gasto.

Computacionalmente, o recozimento pode ser visto como um processo estocástico de determinação de uma organização dos átomos de um sólido, que apresente energia mínima. Em temperatura alta, os átomos se movem livremente e, com grande probabilidade, podem mover-se para posições que incrementarão a energia total do sistema. Quando se baixa a temperatura, os átomos gradualmente se movem em direção à uma estrutura regular e, somente com pequena probabilidade, incrementarão suas energias.

Segundo Metropolis et al. (1953), quando os átomos se encontram em equilíbrio, em uma temperatura *T*, a probabilidade de que a energia do sistema seja *E*, é proporcional à $e^{-E/kT}$, onde *k* é conhecida como constante de Bolzmann.

Desta forma, a probabilidade de que a energia de um sistema seja (*E* + *dE*) pode ser expressa por:

$$prob\,(E + dE) = prob\,(E)\;prob\,(dE) = prob\,(E)\;e^{-dE/kT}$$

Em outras palavras, a probabilidade de que a energia de um sistema passe de *E* para *(E + dE)* é dada por $e^{-dE/kT}$. Na expressão $e^{-dE/kT}$, como *k* é uma constante, observa-se que a medida que *T* diminui, a probabilidade da energia do sistema se alterar é cada vez menor. Nota-se, também, que, nessas condições, quanto menor o valor de *dE*, maior a probabilidade da mudança ocorrer.

O método computacional que imita esse processo de recozimento de um sólido é chamado de *simulated annealing* [5]

3.2 ALGORITMO

Simulated Annealing é considerado um tipo de algoritmo conhecido como de busca local. Ele se constitui em um método de obtenção de boas soluções para problemas de otimização combinatória de difíceis resoluções. Desde a sua introdução vem sendo vastamente utilizado em diversas áreas, tais como projeto de circuitos integrados auxiliado por computador, processamento de imagem, redes neuronais etc.

A sua semelhança com o método original no qual foi inspirado é muito grande. Na sua apresentação, nos trabalhos independentes de Kirkpatrick et al. (1983), e Cerny (1985), é mostrado como um modelo de simulação de recozimento de sólidos, como proposto em Metropolis et al. (1953), pode ser utilizado em problemas de otimização, onde a função objetivo, a ser minimizada, corresponde à energia dos estados do sólido.

Seja S o espaço total de soluções de um problema de otimização combinatória que se pretende resolver. Ou seja, S é o conjunto finito que contém todas as combinações possíveis que representam as soluções viáveis para o problema. Seja f uma função de valores reais definida sobre S, $f : S \rightarrow R$. O problema se constitui em encontrar uma solução (ou estado) $i \in S$, tal que $f(i)$ seja mínimo.

Uma das formas mais simples e imediata de tentar resolver o problema, utilizando busca local em S**,** conhecida como algoritmo descendente, é iniciar o processo de busca por uma solução, normalmente, tomada de forma aleatória. Outra solução j é então gerada, na vizinhança desta, por meio de um mecanismo apropriado e dependente do problema. Caso uma redução do custo dessa nova solução se verifique, ou seja, $f(j) < f(i)$, a mesma passa a ser considerada a solução corrente e o processo se repete. Caso contrário, a nova solução é rejeitada e uma outra gerada. Esse processo se repete até que nenhum melhoramento possa ser obtido na vizinhança da solução corrente, após um número determinado de insistências. O algoritmo retorna, então, o valor da última solução corrente, considerada uma solução de mínimo local.

O grande problema desse método, muito simples e rápido, é que o mínimo local encontrado pode estar longe de ser um mínimo global, o que se traduziria em uma solução inaceitável para o problema. Uma estratégia muito simples de aprimorar a solução obtida através desse tipo de algoritmo seria escolher a menor solução, de um conjunto de soluções obtidas de várias execuções sucessivas, realizadas a partir de diferentes soluções iniciais.

O algoritmo *Simulated annealing* não utiliza essa estratégia. Esse método tenta evitar a convergência para um mínimo local, aceitando, às vezes, uma nova solução gerada, mesmo que essa incremente o valor de *f.* O aceite ou a rejeição de uma nova solução, que causará um incremento de δ em *f,* em uma temperatura *T*, é determinado por um critério probabilístico, através de uma função *g* conhecida por função de aceite. Normalmente, essa função é expressa por

$$g(\delta, T) = e^{-\delta / T}$$

Caso $\delta = f(j) - f(i)$, for menor que zero, a solução *j* será aceita como a nova solução corrente. Caso contrário, a nova solução somente será aceita se

$$g(\delta, T) > random(0, 1)$$

A semelhança com o método original de simulação de recozimento na termodinâmica, como já foi salientado, é grande, pois o parâmetro δ, corresponde à variação da energia de um estado para outro *(dE)* e o parâmetro de controle *T*, corresponde à temperatura. Uma vez que, agora *T* é imaginário, a constante *K,* que aparecia na expressão original multiplicando *T*, é considerada igual a *1*.

Da mesma forma que no processo físico, a função $g(\delta, T)$ implica em:

- a probabilidade de aceite de uma nova solução é inversamente proporcional ao incremento de δ e

- quando *T* é alto, a maioria dos movimentos (de um estado para outro ou de uma solução para outra) é aceita, entretanto, a medida que *T* se aproxima de zero, a grande maioria das soluções são rejeitadas.

Procurando evitar uma convergência precoce para um mínimo local, o algoritmo inicia com um valor de *T* relativamente alto. Esse parâmetro é gradualmente diminuído e, para cada um dos seus valores, são realizadas várias tentativas de se alcançar uma melhor solução, nas vizinhanças da solução corrente [5]

4 IMPLEMENTAÇÃO DO ALGORITMO GENÉTICO

4.1 POPULAÇÃO DO AG

Para a resolução dos algoritmos do AG é necessário um vetor população, que é representado por um conjunto de soluções. Esta população foi adquirida de uma base de dados da Gatech, onde este disponibiliza 25 países, sendo que cada país possui as coordenadas cartesianas de suas respectivas cidades. Esta base de dados tem como objetivo testar algoritmos para problemas do caixeiro viajante. Para cada país, é exibida também qual seria a melhor solução para o problema.

Inicialmente, para cada indivíduo da população, o programa inicia aleatoriamente uma cidade como início do problema, e aplica o algoritmo da busca gulosa, a fim de encontrar a cidade mais próxima, e assim sucessivamente, até a última cidade da busca. Assim cada cromossomo do AG é uma representação de um possível caminho, ou uma possível solução, e o gene do cromossomo é a representação de uma cidade.

Algoritmos genéticos necessitam da informação do valor de uma função objetivo para cada membro da população, que deve ser um valor não negativo. A função objetivo fornece, para cada indivíduo, uma medida de quão bem adaptado ao ambiente ele está. Neste caso e no SA, a função objetivo de cada indivíduo é o custo do caminho, ou seja, a soma das distâncias das cidades na ordem em que se encontra no cromossomo. Por exemplo, um cromossomo com as cidades na seguinte ordem: 4-2-1-3, a função objetivo é o somatório das distâncias de 4 e 2, 2 e 1, 1 e 3, 3 e 4.

Logo após calculado o fitness de cada indivíduo, a população é lidada com o processo de seleção dos indivíduos.

4.2 SELEÇÃO DE INDIVÍDUOS

O método utilizado para a seleção de indivíduos que vão compor a próxima geração é de certa forma inédito e apresentou bons resultado. Ele consiste nos seguintes passos:

- Ordenar a população em ordem decrescente, fundamentado no fitness de cada indivíduo, ou seja, o pior indivíduo será o primeiro da população.

- Construir um Vetor de Soma Acumulada (VSA), ilustrada na figura 3, que representa um vetor com as somas dos seus índices com os índices antecessores. Por exemplo, o índice 4 do vetor conterá a somatória dos índices 4,3,2,1.

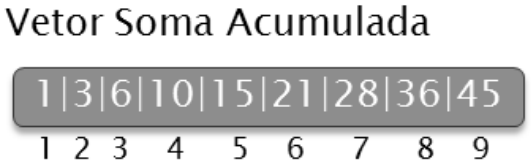

Figura 3 – VSA

- Sorteia-se um número, no intervalo fechado de 1 até o ultimo valor do vetor. No caso do exemplo ilustrado na figura 3, o número aleatório estaria contido no intervalo de 1 a 45.

- Verifica-se em que intervalo formado por um par de elementos sucessivos do vetor VSA esse número aleatório estará contido. Por exemplo, considerando que os intervalos são fechados a direita, na figura 3, existiriam então os intervalos: [1], [2,3], [4,5,6], [7,8,9,10], [11,12,13,14,15], e assim por diante. Supondo ainda que um número sorteando tenha sido o 13, o indivíduo escolhido para participar da nova geração ou do cruzamento seria o de numero 5.

É importante observar que a probabilidade no sorteio do número é teoricamente igual para todo o intervalo de 1 a 45. Entretanto, os indivíduos com melhores fitness ocupam os lugares dos maiores índices do vetor VSA e conseqüentemente, seus intervalos correspondentes são maiores.

4.3 CROSSOVER UTILIZANDO PMX

Foram testados diferentes tipos de operados crossovers e o que apresentou melhores resultados em diferentes situações foi o PMX – Partially-Matched Crossover. Seu funcionamento consiste no seguinte: selecionar aleatoriamente dois cromossomos pais, p1 e p2, selecionar também aleatoriamente dois pontos de corte; dentro do corte formado pelos pontos de corte cada gene situado no pai p1 é comparado com o seu correspondente no pai p2, se esses genes forem diferentes, é procurado no pai p1 o gene igual ao do pai p2 e feito a troca. Esse processo se encerra quando todo o corte for varrido, tanto para o pai p1, como para o pai p2. Na figura 4 é demonstrado um exemplo desta técnica de crossover para duas amostras.

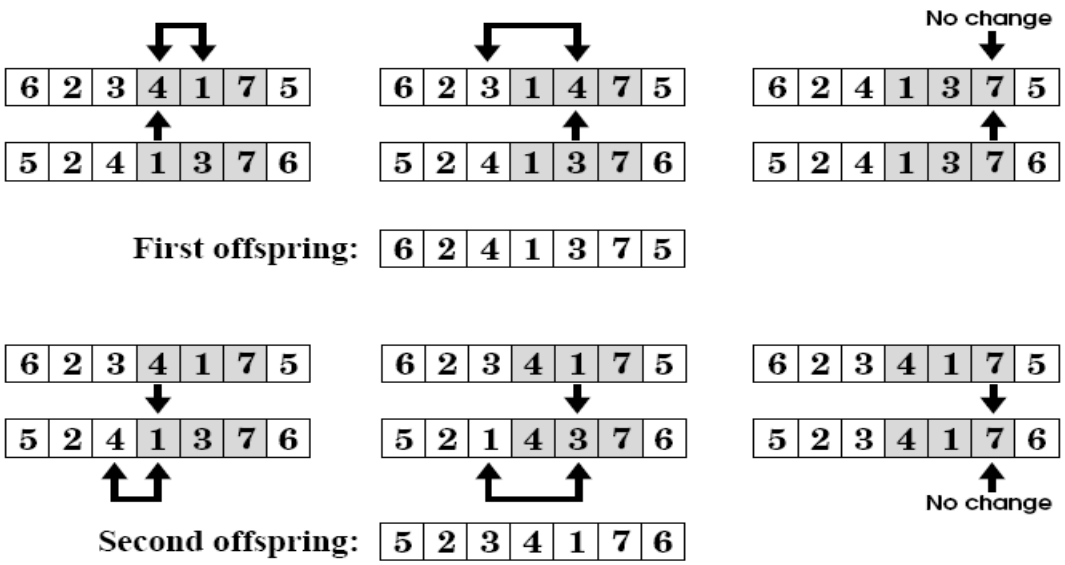

Figura 4 - Crossover PMX (ÜÇOLUK, 2007)

Encerrando o processo de cruzamento genético, passa-se ao outro operador que é a mutação. A probabilidade de um indivíduo sofrer essa tipo de operação depende muito do problema tratado, mas normalmente é baixa.

Nesse trabalho, também de uma forma relativamente inédita, foi utilizado o algoritmo Simulated Annealing na realizar da operação de mutação. O objetivo desta hibridização é tornar a mutação controlada, ou seja, com uma maior probabilidade, uma operação de mutação resultará em um individuo de melhor qualidade.

Ao termino da aplicação da operação de mutação uma nova população é então gerada. Caso não sejam satisfeitas as condições de término, o processo se repete, aplicando novamente as operações genéticas sobra à nova população. Neste trabalho, para o termino do AG utilizou-se um número fixo de iterações que podendo ser alterado, de acordo com o tamanho da população.

5 PARALELISMO DO AG

O algoritmo genético é intrinsecamente paralelo, suas populações podem, perfeitamente, serem processadas paralelamente e de forma totalmente independentes, assim sendo, existem diversos modelos para paralelisar um algoritmo genético na literatura. Nesse trabalho foram investigados dois modelos que serão descritos a seguir.

5.1 MODELO 1

Nesse modelo, a estrutura do algoritmo apresenta uma máquina mestra ou supervisor e várias máquinas escravas (quantas estiverem disponíveis na infra-estrutura). O supervisor cria e distribui parcelas iguais da população total, definida para aquele determinado problema, entre as máquinas escravas. Cada máquina escrava processa a evolução de sua particular população, executando o algoritmo genético, aplicando os operadores de seleção, *crossover* e mutação e então submete ao supervisor os melhores indivíduos escolhidos probabilisticamente. O supervisor então, após a distribuição das populações passa a intermediar a migração de indivíduos entre as populações das máquinas escravas. Na realidade ele propicia um banco de indivíduos migrantes que permanecem na sua área de memória, até serem requisitados por uma outra máquina que não a de origem, ou até serem substituídos por novos migrantes.

5.2 MODELO 2

Nesse modelo, a estrutura do algoritmo é praticamente idêntica ao anterior: apresenta uma máquina supervisor e várias máquinas escravas. Entretanto, nesse segundo modelo, o supervisor cria na sua área a população total, definida para aquele determinado problema. Cada máquina escrava recebe do supervisor um naco da população gerada. Diferentemente do modelo anterior, o supervisor mantém com ele a população total que vai se renovando repetidamente. Cada máquina escrava processa a evolução de sua particular população, executando o algoritmo genético igualmente como no modelo anterior. Entretanto, nesse segundo modelo as escravas enviam ao supervisor, a cada geração, a sua nova população integralmente. O supervisor então, após receber de volta todas as populações, formando assim uma nova população total, repete o ciclo, enviando novamente os novos pedaços de população às máquinas escravas. A distribuição dos fragmentos da população total é feita de forma totalmente aleatória.

Nos dois modelos, a máquina mestra faz uso de *threads* para se comunicar com as máquinas escravas, uma *thread* para cada máquina. Toda a parte de sincronização na programação concorrente entre as *threads,* no supervisor, foi implementada utilizado as ferramentas fornecidas pela linguagem Java. A comunicação entre as máquinas foi implementada utilizando *Sockets.*

As mensagens trocadas entre a máquina supervisor, através das suas *threads*, e as máquinas escravas, são objetos serializados.

6 RESULTADOS EXPERIMENTAIS

Inicialmente foram realizados testes com o algoritmo genético simples e com o algoritmo genético com simulated annealing, os dois seqüencias. Dentre as populações disponibilizadas pela GATHEC, foram feitos testes com o país de Qatar(194 cidades). De acordo com o site, a melhor solução para o caixeiro viajante é de um caminho com custo de 9352.

6.1 RESULTADO DO AG SEQUENCIAL

Os teste apresentado a seguir, foram feitos em um computador dual core 2GHz, com 3 Gigabytes de memória. As configurações do algoritmo estão representadas na tabela 1, na qual a população de soluções é de 5000 mil indivíduos, com 12 gerações e 194 cidades. A princípio foram feitos testes do AG com mais de 12 gerações, porém a probabilidade de o AG apresentar soluções melhores a partir da 10ª geração é mínima. Na figura 5, observa-se que não ocorre muita variação nos resultados obtidos, sempre mantendo um custo de caminho mínimo de aproximadamente 11330. Essa estagnação de resultado ocorre devido ao algoritmo permanecer em mínimos locais, diminuindo a qualidade dos resultados.

Tabela 1 Configuração AG seqüencial

Nome	Valores
AG	
tamanho população	5000
número de gerações	12
número de cidades	194
Tempo	Milissegundos

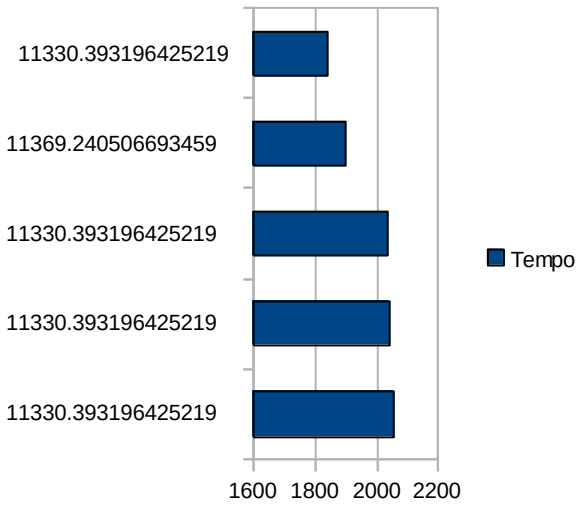

Figura 5 - Resultados AG seqüencial

6.2 RESULTADOS DO AG COM SA SEQÜENCIAL

Novamente para os testes do AG com SA, foi utilizado o mesmo computador descrito no AG simples. As configurações iniciais dos algoritmos estão na tabela 2. Pode se perceber que as configurações do AG permaneceram as mesmas, para fins de intensificar as diferenças que o SA suscita, contribuindo ao AG a resultados mais próximos à solução ótima

Tabela 2 Configuração AG com SA

Nome	Valores
AG	
tamanho população	5000
número de gerações	12
número de cidades	194
SA	
Interações	91,51
Total trocas	194
Total tentativas	194
Tempo	Milissegundos

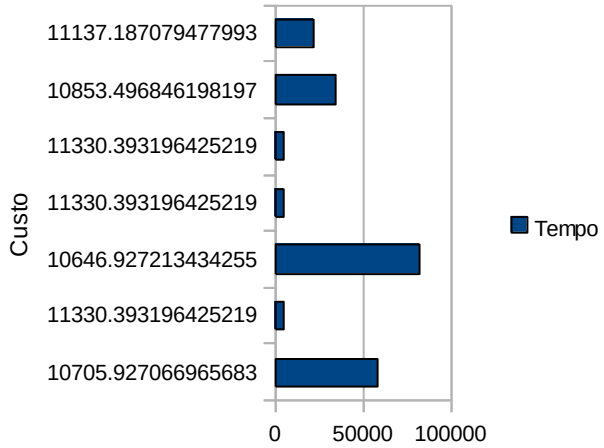

Figura 6 Resultados do AG com AS seqüencial

Para este caso foram realizados mais testes devido à grande variação nos resultados obtidos, uma vez que o SA permitindo que o AG alterne os pontos de ótimos locais. Dentre as 7 amostras apresentadas na figura 6, apenas 40% dos resultados mantiveram-se iguais ao AG simples, e os que não mantiveram, apresentaram soluções sempre melhores. Observa-se que o teste que apresentou a melhor solução (aproximadamente 10647), foi claramente o que apresentou o maior número de mutações nas gerações do AG. A atuação do SA pode deixar o AG até 10 vezes mais lento, entretanto, a sua influência na qualidade das soluções encontradas é indiscutível.

6.3 RESULTADOS DO AG COM SA PARALELO

Como já foi dito anteriormente, neste trabalho foram propostos e testados dois modelos de paralelismo. Como o segundo modelo foi o que apresentou melhores soluções, serão abordados somente resultados do mesmo neste item.

Primeiramente foram feitos testes com o algoritmo paralelizado utilizando apenas duas máquinas. Na máquina servidora foi executado também um cliente e o outro cliente em uma máquina separada. De acordo com os resultados da figura 7, diminuíram os picos do gráfico, apresentando um ganho em tempo de execução. Ocorreu pouco rendimento em relação à qualidade dos resultados, diminuindo o custo mínimo de 10647 para apenas 10063.

Tabela 3 Configuração do AG com SA paralelo com 2 clientes

Nome	Valores
AG – Paralelo	2 clientes
tamanho população	5000
número de gerações	12
número de cidades	194
SA	
Interações	91,51
Total trocas	194
Total tentativas	388
Tempo	Milissegundos

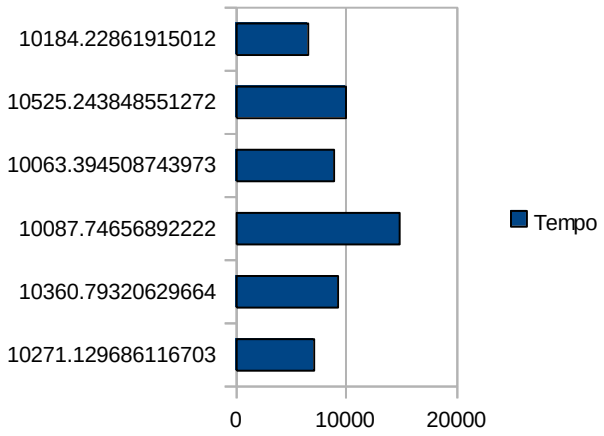

Figura 7 Resultados do AG com SA paralelo com 2 clientes

Numa outra simulação o modelo foi testado utilizando 10 máquinas. Aproveito-se da quantidade de maquinas em execução, foi duplicado o tamanho da população passando de 5 mil para 10 mil.

Tabela 4 Resultados do AG com SA paralelo com 10 clientes

Nome	Valores
AG – Paralelo	10 clientes
tamanho população	10000
número de gerações	12
número de cidades	194
SA	
Interações	91,51
Total trocas	194
Total tentativas	388
Tempo	Milissegundos

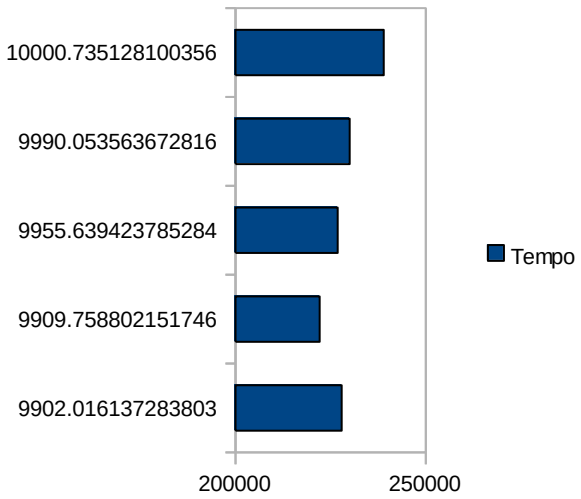

Figura 8 Resultados do AG com SA paralelo com 10 clientes

Observa-se, através da figura 8, que os resultados baixaram de 10647 para 9902. Entretanto, ocorreu um aumentou em cem por cento no tempo de processamento. Esse fato é decorrente do modelo de paralelização implementado que utiliza uma barreira para sincronizar todas as máquinas escravas. Essa barreira faz com que todas as máquinas aguardem pela devolução, ao mestre, de todas as novas populações geradas. Muitas dessas máquinas, que processaram suas populações com taxa bai xa de mutação, ficam aguardando a liberação, sem realiza processamento algum.

7 CONCLUSÕES E TRABALHOS FUTUROS

Embora não tenham sido realizados testes exaustivos, e mesmo não tendo sido feito um estudo para identificar os melhores operadores de *crossover* e mutação. Os resultados obtidos sugerem que a implementação de AGs paralelos segundo a estrutura proposta neste trabalho, tem boas perspectivas. Os resultados obtidos com a simulação do paralelismo indicam que a troca periódica de indivíduos não acelera tanto o processo de convergência, entretanto, regulando o intervalo de migração se obtém maior diversidade de indivíduos, o que possibilita alcançar melhores resultados antes da estagnação.

Os resultados obtidos do algoritmo genético são valores aproximados do valor ótmo, uma vez que este tipo de resultado é uma característica tanto do AG como do SA, sempre procurando encontrar um melhor valor aproximado. Como esperado, quando utilizado o AG paralelo, como o SA realizando mutação, obteve-se valores mais constantes e significativamente mais próximos ao valor ótimo procurado.

Este trabalho pode ser complementado futuramente com a otimização do Simulated Annealing, uma vez que este necessita de alto processamento, uma sugestão razoável seria eleger algumas máquinas escrevas para processar exclusivamente o SA.

A descoberta de novos modelos para a paralelização de AG´s também seria um grande e importante desafio nessa área.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]. *GOLDBERG*, D.E..*Genetic Algorithms in Search, Optimization, and Machine Learning*. EUA:Addison-Wesley, 1989.

[2]. *KOZA*, J.R.. *Genetic Programming*: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

[3]. *RICCARDO*, P., WILLIAN, B. L., NICHOLAS, F.M.. *A Field Guide to Genetic Programming*. ISBN 978-1-4092-0073-4, 2008.

[4]. *BECKER*, D., STERLING, T., SAVARESE, D., DORBAND, J., RANAWAK, U., PACKER, C.. In *Proceedings of the International Conference on Parallel Processing (New York: Institute of Electrical and Electronics Engineers),* 1995.

[5]. *MAZZUCCO*, José JR. *Uma abordagem híbrida do problema da programação da produção através dos algoritmos simulated annealing e genético.* UNIVERSIDADE, FEDERAL, DE, SANTA, CATARINA, - FLORIANÓPOLIS, - 1999,

[6]. *PIMENTA*, Ana Isabel; CASTANHEIRA, Susana. *Complementos de Investigação Operacional "Simulated Annealing"(Meta-Heurísticas)* .

[7]. *KIRKPATRICK*, S; SCHNEIDER, Johannes J. *The Traveling Salesman Problem,* in: Stochastic Optimization. Disponível em: <www.springerlink.com>, p. 211-231, 2006 (Scientific Computation).

[8]. *KIRKPATRICK*, S; SCHNEIDER, Johannes J. *Application of Simulated Annealing to TSP,* in: Stochastic Optimization. Disponível em: <www.springerlink.com>, p. 299-314, 2006 (Scientific Computation).

[9]. *ÜÇOLUK*, Göktürk. *Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation,* DEPARTMENT OF COMPUTER ENGENEERING MIDDLE EAST TECHNICAL UNIVERSITY 06531 ANKARA, TURKEY.

[10]. ROISENBERG, Mauro. *Computação Evolucionário.* Florianópolis: UNIVERSIDADE FEDERAL DE SANTA CATARINA, 2007. 48 slides: color. Slides gerados a partir do Adobe Reader.

[11]. *GATHEC,* Tsp. Disponível em: <http://www.tsp.gatech.edu/world/count**>** Acesso em: outubro. 2009

[12]. *SIMULATED annealing*. 12 maio. 2009. Disponível em: <http://pt.wikipedia.org/wiki/Simulated_annealing>. Acesso em: 17 agosto. 2009.