HENRIQUE SOARES DE SOUZA

ESCALONAMENTO DE TAREFAS ATRAVÉS DE ALGORITMO GENÉTICO

Monografia apresentada à Universidade de São Paulo - USP, como requisito parcial para a obtenção da certificação do curso de MBA em Tecnologia da Informação - TI.

Área de Concentração:
Algoritmo Genético e
Escalonamento de Tarefas

Orientador:
Prof. Alexandre Ricardo Nardi

São Paulo
2007

FICHA CATALOGRÁFICA

Souza, Henrique Soares de
Escalonamento de tarefas através de algoritmo genético / H.S. de Souza. -- São Paulo, 2007.
62 p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1.Inteligência artificial 2.Sistemas distribuídos I.Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II.t.

A meus pais José e Maria Cristina, que sempre me apoiaram para realização de todo esse trabalho. À Daniela, uma homenagem como agradecimento de toda paciência, ajuda e incentivo.

AGRADECIMENTOS

Agradeço a todos que me incentivaram a iniciar essa caminhada e realizar esse projeto em minha vida.

Em especial aos meus pais, José e Maria Cristina que sempre me incentivaram, torceram e me ajudaram em todos os sentidos e foram extremamente pacientes nos momentos mais difíceis.

A minha noiva Daniela pela torcida, pela paciência e enorme ajuda em todos os aspectos do trabalho.

Ao meu orientador Alexandre Ricardo Nardi, que me acompanhou nessa jornada sempre com grandes contribuições para realização desse trabalho.

Aos amigos da Convergys, que sempre me apoiaram e torceram por mim nessa caminhada.

A todos os Alunos e Professores do PECE, os quais dividiram suas experiências e conhecimentos comigo.

RESUMO

Não são raras situações no mundo computacional, onde encontram-se agregados de computadores que possuam a necessidade de um balanceamento de carga, capaz de garantir um bom desempenho com o menor número de recursos possíveis. Este trabalho tem justamente o objetivo de explorar o Algoritmo Genético como uma alternativa para tais problemas. O pleno entendimento das etapas do AG, juntamente com as variações de cada uma delas, traz para o analista uma visão mais próxima do que é realmente possível aplicar e também quais as reais condições necessárias para a utilização de um algoritmo. Espera-se que após a leitura o analista tenha uma visão do que é o AG e quais são as vertentes de pesquisa para uma possível aplicação.

Palavras-Chave: Algoritmo Genético, Escalonamento de Tarefas, NP-Completo

ABSTRACT

It is easy to find situations in the computer world, where there are computer clusters that need to balance overflow. Balanced overflow provides the guarantee of good performance with the smallest number of resources. This work explores the Genetic Algorithm (GA) as an alternative for such problems. The full understanding of all GA stages and the variations of each one bring the analyst a clearer vision of which algorithm to apply in different real-world conditions. Finally, we hope the analyst has a vision of what GA is, and which possibility would be most appropriate for a given application.

Key-Words: Genetic Algorithm, Scheduler Task, NP-Complete

Sumário

Lista de Figuras

Lista de Tabelas

1. INTRODUÇÃO

O crescente avanço das tecnologias de hardware e software, juntamente com a necessidade de desempenho computacional cada vez maior das organizações, tem impulsionado o uso de sistemas paralelos e distribuídos. Uma solução normalmente utilizada é os agregados de computadores, também conhecidos como *clusters* computacionais.

Um dos grandes problemas em tais sistemas é o desenvolvimento de técnicas efetivas de distribuição de processos entre os nós, pois desta forma existe uma grande probabilidade de um nó ficar sobrecarregado, enquanto outros ficam ociosos. Tal problema, conhecido como desbalanceamento de carga, degrada o desempenho do sistema como um todo. Uma vez que uma distribuição de processos mais efetiva diminui o tempo de resposta computacional dos processos.

Para tentar minimizar tal problema, pode ser utilizado o escalonamento de tarefas juntamente com um algoritmo capaz de agilizar a tomada de decisão, de qual processador ou computador que será utilizado para o processamento daquela tarefa.

A análise do problema do escalonamento de tarefas normalmente recai nos problemas NP completos, o qual freqüentemente requer o uso de aproximações, já que a obtenção da solução ótima é, em geral, computacionalmente inviável. Entretanto, aplicando-se técnicas de inteligência artificial em conjunto com os já conhecidos métodos heurísticos, é possível proporcionar um modelo computacional mais flexível, capaz de produzir resultados de boa qualidade em um tempo satisfatório.

Este trabalho descreve os passos do escalonamento de tarefas através do algoritmo genético, explicando cada um deles através de um problema de escalonamento sugerido, com o objetivo de minimizar o tempo requerido para se executar um conjunto de tarefas, mostrando também como um algoritmo genético pode ser utilizado para determinar as prioridades relativas entre as tarefas que competem por um mesmo recurso de *hardware*, permitindo assim, selecionar em que instante de tempo cada tarefa será executada em um determinado recurso.

Como todo o trabalho é baseado em dois principais tópicos, Escalonamento de Tarefas e Algoritmos Genéticos, iremos abordar os conceitos de ambos para

conseguir desenvolver melhor o assunto de escalonamento de tarefas através do Algoritmo Genético.

É importante também ressaltar que antes da escolha de qualquer solução para o problema proposto, deve-se realizar primeiramente a análise do projeto conforme descrito no item 2, a fim de entender melhor a complexidade envolvida. Isso pode ajudar na escolha de qual algoritmo pode ser empregado para resolver o problema.

Para entender a importância do tema escolhido, verificamos que a utilização de Algoritmo Genético como ferramenta para soluções de escalonamento de tarefas com restrição de recursos, pode ser aplicado em diversas áreas do conhecimento. Na área de redes, por exemplo, há a necessidade de escolha de rotas de transmissão de dados, onde os custos e desempenho da rede são considerados para a escolha de uma solução com boa qualidade em um tempo satisfatório, ou mais adaptada para o problema, entre outros problemas similares.

2. ANÁLISE DO PROJETO

A análise do projeto juntamente com o algoritmo é uma metodologia que permite viabilizar a escolha do melhor algoritmo para um problema. Podemos considerar alguns critérios sobre a complexidade no desempenho de algoritmos, como [TOSCANI, 2002]:

- Quantidade de trabalho requerido (Complexidade do Algoritmo);
- Quantidade de espaço requerido;
- Simplicidade;
- Exatidão de resposta;
- Otimalidade;

2.1. A complexidade no desempenho de Algoritmos

Quando um algoritmo qualquer, rodando em uma máquina com memória e capacidade de processamento suficientes, produz uma resposta correta, para qualquer entrada, esse algoritmo não é necessariamente aceitável na prática, pois os recursos de memória e capacidade de processamento podem ser escassos nos casos práticos.

O crescente avanço tecnológico permite a criação de máquinas computacionais cada vez mais rápidas. Isso pode distorcer a importância da escolha de um algoritmo mais adaptado para o problema em questão. Para um algoritmo rápido, qualquer melhoria na velocidade de processamento da máquina é sentida imediatamente. Essa melhoria fica ainda mais evidente no caso de algoritmos menos eficientes [TOSCANI, 2002].

2.1.1. Medidas de Complexidade

A complexidade do algoritmo não pode ser representada simplesmente por um número, pois geralmente o número de operações básicas efetuadas não é o mesmo para qualquer entrada.

Mesmo em casos com entradas do mesmo tamanho, pode haver entradas especiais que exijam um processamento maior. Para ordenar uma lista quase

classificada, pode não ser necessário o mesmo esforço comparado com uma lista de mesmo tamanho, mas com os elementos em maior desordem.

Essa complexidade pode variar para um mesmo problema e uma mesma entrada aplicada em algoritmos distintos. Isso pode ser de grande utilidade em casos de problemas pré-fixados com finitas entradas.

Se a complexidade é tomada como a máxima para qualquer entrada de um dado "tamanho", a complexidade é chamada complexidade no pior caso ou simplesmente complexidade. Se, entretanto, é levada em conta a probabilidade de ocorrência de cada entrada de um mesmo "tamanho", a complexidade é chamada de complexidade esperada ou complexidade média.

Complexidade de um algoritmo é o esforço (quantidade de trabalho) de um algoritmo. As principais medidas de complexidade são tempo e espaço, relacionadas à velocidade e quantidade de memória, respectivamente. A complexidade depende da entrada em particular: os principais critérios são pior caso e caso médio.

2.1.1.1. Complexidade de Tempo

Uma das medidas de complexidade mais importante é a complexidade de tempo. Para a sua medida pode-se utilizar diversos critérios como a utilização de um computador específico para medir o tempo de execução de um algoritmo. Entretanto uma medida como essa é fortemente dependente do ponto de execução do algoritmo, sendo assim qualquer alteração no código pode alterar consideravelmente o desempenho do programa.

2.1.1.2. Complexidade de Espaço

A memória usada por um programa, bem como o tempo requerido para execução de um programa, dependem da implementação em particular. Um programa requer uma área para guardar suas instruções, suas constantes, suas variáveis e os dados, mas pode também utilizar uma área de trabalho para manipular os dados e guardar informações para levar adiante a computação.

A forma de representação dos dados devem ser considerados, pois isso pode exigir maior ou menor espaço de memória.

2.1.2. Critérios de Complexidade

Para medir a quantidade de trabalho realizado por um algoritmo, é escolhida uma operação, chamada operação fundamental, e então é contado o número de execuções dessa operação na execução do algoritmo. A operação escolhida como fundamental deve ser tal que a contagem do número de vezes que ela é executada expressa a quantidade de trabalho do algoritmo, dispensando outras medidas. Às vezes, se faz necessário mais de uma operação fundamental, com pesos diferentes.

Por exemplo, para um algoritmo de ordenação, uma operação fundamental natural é a comparação entre elementos quanto à ordem.

2.1.2.1. Complexidade de algoritmo

O tempo requerido por um algoritmo em dada entrada pode ser medido através de suas seqüências de execução, ou seja, a complexidade pode ser determinada com base em operações fundamentais e no tamanho da entrada.

Seja E o conjunto de todas as seqüências de execuções fundamentais.

- exec: A x D ? E;

  exec(**a**, d) := seqüência de execuções de operações fundamentais efetuadas na execução do algoritmo a, com entrada d.

- custo: E ? IN (Conjunto dos Números Naturais);

  custo(s) := comprimento da seqüência s, definido conforme o peso estabelecido para as operações fundamentais.

- desemp(d):=custo (exec(A,D)).

O desempenho do algoritmo **A** com a entrada D pode ser dado por custo (exec (A,D)). No contexto de um algoritmo **A** fixado, podemos referir-nos ao desempenho em D.

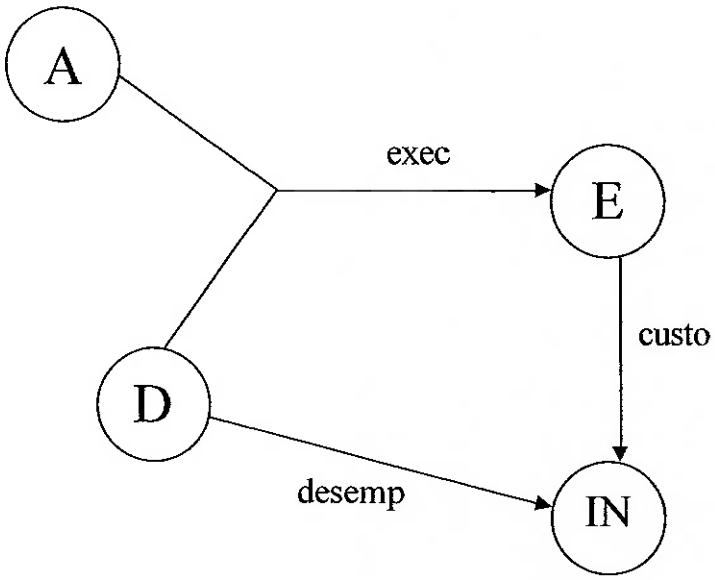

Figura 1: Desempenho do algoritmo

Note que desemp : D ? IN dá informação sobre cada entrada específica. Em geral, essa informação é demasiadamente detalhada e a análise de tal função é bastante difícil.

Por exemplo, considere um algoritmo de busca (seqüencial) em uma tabela com n elementos. Em certos casos, encontra-se a chave após uma comparação; em outros, são necessárias várias, até mesmo n comparações.

Deseja-se condensar a informação dada pela função desempenho em termos das entradas de um tamanho dado.

Para tanto, seleciona-se uma função:

- tam : D ? IN;

tam (D) := tamanho da entrada D.

De posse de tal função tam, pode-se considerar a informação dada pela função desemp:

D ? IN em uma função aval : IN ? IN tal que aval(n)=desemp(D) para tam (D) = n.

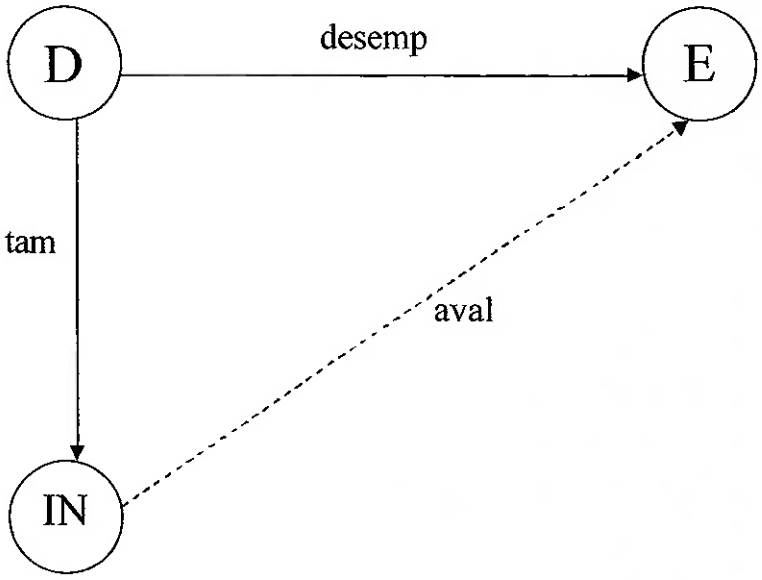

Figura 2: Avaliação de algoritmo

2.2. A complexidade do problema e a intratabilidade

A complexidade também pode ser vista como uma propriedade do problema, o que significa dar uma medida independente do tratamento dado ao problema, independente do caminho percorrido na busca da solução, portanto independente do algoritmo.

Alguns problemas são "bem comportados" e permitem chegar a limites de complexidade, mas outros são tão difíceis de serem resolvidos que parecem intratáveis. Nesse contexto, surgem os problemas NP-Completos, que, segundo Cook [TOSCANI, 2002], podem ser considerados como um limite de complexidade. Entretanto, esses problemas existem, são usados no dia-a-dia e, portanto, precisam ser tratados, e o que é mais importante é que precisam ser identificados.

2.2.1. Complexidade do Problema

O mais importante questionamento sobre um problema é a sua computabilidade, isto é, se ele pode ser resolvido mecanicamente (por um algoritmo). Para identificar e definir questionamento como esse, surgiram vários

formalismos e máquinas abstratas. Entre esses, o mais conhecido é a Máquina de Turing.

A máquina de Turing é um dispositivo teórico, conhecido como máquina universal, que foi concebido pelo matemático britânico Alan Turing (1912-1954), muitos anos antes de existirem os modernos computadores digitais. É um modelo abstrato de um computador, que se restringe apenas aos aspectos lógicos do seu funcionamento (memória, estados e transições) e não à sua implementação física, sendo possível em uma máquina de Turing modelar qualquer computador digital. Uma máquina de Turing consiste em [DIVERIO, 2003]:

- Uma fita que é divida em células, uma adjacente à outra. Cada célula contém um símbolo de algum alfabeto finito. O alfabeto contém um símbolo especial branco e um ou mais outros símbolos. Assume-se que a fita é arbitrariamente extensível para a esquerda e para a direita, isto é, a máquina de Turing possui tanta fita quanto é necessário para a computação. Assume-se também que células que ainda não foram escritas estão preenchidas com o símbolo branco;
- Um cabeçote, que pode ler e escrever símbolos na fita e mover-se para a esquerda e para a direita;
- Um registrador de estados, que armazena o estado da máquina de Turing. O número de estados diferentes é sempre finito e há um estado especial denominado estado inicial com o qual o registrador de estado é inicializado;
- Uma tabela de ação (ou função de transição) que diz à máquina que símbolo escrever, como mover o cabeçote ('E' para esquerda e 'D' para direita) e qual será seu novo estado, dados o símbolo que ele acabou de ler na fita e o estado em que se encontra. Se não houver nenhuma entrada na tabela para a combinação atual de símbolo e estado então a máquina pára.

O modelo formal de uma Máquina de Turing é uma 8-upla, como segue abaixo:

$M = (\Sigma, Q, \Pi, qo, F, V, ß, °)$

Onde:

Σ = alfabeto de símbolos de entrada;

Q = conjunto de estados possíveis da máquina, o qual é finito;

Π = programa ou função de transição

qo = estado inicial da máquina tal que qo é elemento de Q;

F = conjunto de estados finais tal que F está contido em Q;

V = alfabeto auxiliar;

ß = símbolo especial branco;

° = símbolo especial marcador de início ou símbolo de início da fita

A Máquina de Turing tornou-se a mais utilizada definição de algoritmo e assim deu-se à definição de "Computável": um problema é computável se é resolvido em uma Máquina de Turing [DIVERIO, 2003].

Quando se acrescenta à computabilidade uma propriedade de eficiência, além de um algoritmo que resolve o problema, quer-se um algoritmo eficiente (uma definição aceitável para "eficiente" é "cuja complexidade seja polinomial"), têm-se também duas classes de problemas: P, a classe de problemas para os quais existe algoritmo (Máquina de Turing) de ordem polinomial, que resolve o problema; NP, a classe de problemas para os quais existe algoritmo (Máquina de Turing) de ordem polinomial, que, dada uma entrada para o problema, verifica (faz a certificação) se tem resposta SIM (determinístico) ou NÃO (não determinístico) [DIVERIO, 2003].

2.2.2. Intratabilidade

Historicamente, a expressão "algoritmo não eficiente" é associada à condição de algoritmo de complexidade não polinomial, ou simplesmente algoritmo não polinomial, como é mais comumente dito. Essa associação de problema intratável, algoritmo não eficiente e algoritmo não polinomial justifica-se, porque um algoritmo com complexidade, por exemplo, 2^n tem um tempo de execução que cresce tão rapidamente com n que, para um n razoavelmente grande, torna-se proibitivo, e o problema que tem esse algoritmo como melhor opção de solução torna-se intratável para entradas razoavelmente grandes.

Os algoritmos podem então ser particionados em duas classes: os razoáveis (polinomiais) e os não razoáveis (exponenciais) [TOSCANI, 2002].

Um problema é dito tratável se o limite superior de complexidade é polinomial, isto é, conhece-se um algoritmo razoável que o resolve. Um problema é dito intratável se o limite superior de complexidade é exponencial. Há problemas cujas respostas são tão longas, que se precisa de um tempo exponencial para descrevê-la. Mas, nesse caso, costuma-se dizer que o problema não está bem posto, isto é, não está definido realisticamente.

Os chamados "NP-Completos" são problemas que têm a questão da complexidade ou tratabilidade não resolvida, pois não se sabe se existe ou não algoritmo polinomial que os resolva.

3. CONCEITOS DE ESCALONAMENTO

O termo escalonamento ("*scheduling*") identifica o procedimento de ordenar tarefas. Uma escala de execução ("*schedule*") é uma ordenação ou lista que indica a ordem de ocupação do processador por um conjunto de tarefas disponíveis.

Os primeiros sistemas computacionais não permitiam que mais de um programa compartilhasse o tempo de processamento disponível. Com a evolução dos sistemas veio a possibilidade de carregar múltiplos programas para a memória e executá-los concorrentemente, com isso surgiu à necessidade de um controle mais rígido, resultando na noção de processo como um programa em execução [FARINES,2000].

Um programa pode ser encarado como uma série de instruções que são executadas seqüencialmente por um processador. Cada processo contém duas partes: a primeira é a tarefa que contém as instruções que devem ser executadas, e a segunda guarda informações relativas à execução da tarefa. Quando se executa múltiplos processos, observa-se uma pseudo-paralelização da execução, conseguida com o rápido escalonamento desses processos em execução.

Os escalonadores ("*scheduler*") podem ser encontrados em diversos níveis do sistema, desde o núcleo dos sistemas operacionais modernos, controlando acesso ao processador, ou seja, é responsável pela gestão do processador, até o nível das aplicações com escalonamento para dividir a carga entre as diferentes atividades de uma aplicação [MELLO,2005]. É o escalonador também que implementa uma política de escalonamento ao ordenar para execução sobre o processador um conjunto de tarefas.

Essas políticas de escalonamento definem critérios ou regras para a ordenação das tarefas. Utilizando-se dessas políticas os escalonadores produzem escalas, e se essas escalas forem "realizáveis" garantem o cumprimento das restrições temporais impostas às tarefas. Uma escala é dita ótima se a ordenação do conjunto de tarefas de acordo com os critérios pré-estabelecidos é a melhor possível no atendimento das restrições temporais.

O escalonador é parte integrante do *kernel,* ou seja, do núcleo do sistema operacional. Podemos dividir os algoritmos de escalonamento em duas categorias: preemptivos e cooperativos.

O escalonamento preemptivo define qual tarefa deve possuir a CPU e até quando. Dessa forma, a tarefa em execução pode ser interrompida pelo escalonador e outra colocada em seu lugar. Existem vários algoritmos preemptivos que definem quais são as regras de preempção das tarefas.

No escalonamento cooperativo as tarefas cedem o seu tempo de processamento quando não podem prosseguir na execução. Este algoritmo é mais simples de ser implementado, mas as interações entre as tarefas devem ser bem elaboradas para não causar latência na execução e um possível *crash* do sistema. Os algoritmos descritos fazem uso de um escalonador para controlar as interações de escolha e troca das tarefas. Em um sistema microcontrolado nem sempre é possível ter espaço de código o suficiente para codificar um escalonador, mesmo que simples, pode custar alguns recursos importantes como *timers* e memória [FARINES,2000].

Um problema de escalonamento, na sua forma geral, envolve um conjunto de processadores, recursos compartilhados e um conjunto de tarefas especificadas através de restrições de tempo, precedência e exclusão. O escalonamento de tempo real, na sua forma geral, é dito como um problema intratável (NP-Completo). Os algoritmos existentes representam uma solução polinomial para um problema de escalonamento particular, onde em um conjunto de hipóteses podem surgir simplificações no conjunto de tarefas, com o intuito de diminuir a complexidade do problema. Um algoritmo é identificado como ótimo se minimiza algum custo ou alguma métrica definida na sua classe de problema. No entanto, quando nenhum custo ou métrica é definida, a única preocupação é encontrar alguma escala "realizável", sendo assim, o algoritmo é dito ótimo quando encontra essa escala.

3.1. Tarefas

O conceito de tarefa faz parte do que chamamos de um problema de escalonamento. Tarefas tornam as unidades de processamento seqüencial que concorrem sobre um ou mais recursos computacionais de um sistema. Uma simples

aplicação é constituída de várias tarefas, onde uma tarefa deve satisfazer seus prazos e restrições temporais [FARINES,2000].

As restrições temporais, as relações de precedência e de exclusão usualmente são impostas sobre tarefas que são determinantes na definição de um modelo de tarefas, que é parte integrante de um problema de escalonamento.

Todas as tarefas possuem um prazo, ou seja, seus "*deadlines*", que normalmente devem ser concluídas antes de seu "*deadline*". A conseqüência de uma tarefa ser concluída após o seu "*deadline*" define dois tipos de tarefas [FARINES,2000]:

- Tarefas Críticas ("*hard*"): Uma tarefa é chamada crítica, quando ela é completada após o seu prazo, causando falhas catastróficas no sistema e no ambiente, podendo apresentar prejuízos para as empresas;

- Tarefas Brandas ou Não-Críticas ("*soft*"): As tarefas são chamadas brandas quando também são terminadas após o prazo, mas causam apenas uma diminuição no desempenho do sistema, geralmente sem causar prejuízos graves.

Outra característica das tarefas é a regularidade de suas ativações. Os modelos de tarefas comportam dois tipos de tarefas:

- Tarefas Periódicas: Ocorrem em uma seqüência infinita e regular, possuem uma previsibilidade, sendo ativadas por período formando um conjunto de diferentes instâncias da tarefa. Normalmente, são associadas a "*deadline hard*", ou seja, são tarefas críticas;

- Tarefas Aperiódicas: São tarefas que correspondem a eventos internos ou externos definindo uma característica aleatória na ativação. Normalmente, são associadas a "*deadline soft*" nas suas execuções.

Outras restrições são importantes na definição do comportamento temporal de uma tarefa:

- Tempo de computação: é o tempo necessário para execução completa da tarefa;

- Tempo de início: corresponde ao instante de início do processamento da tarefa em uma ativação;

- Tempo de término: corresponde ao tempo em que se completa a execução da tarefa;

- Tempo de chegada: corresponde ao instante em que o escalonador recebe a ativação dessa tarefa. Nas tarefas periódicas, o tempo de chegada coincide com o início do período de ativação. As tarefas aperiódicas apresentam o tempo de chegada coincidindo com o tempo da requisição do processamento;

- Tempo de liberação: corresponde com o instante de sua inclusão na fila de tarefas prontas para execução.

3.1.1. Relações de Precedência e de Exclusão

Na maioria das vezes, os processamentos não podem executar tarefas em ordem arbitrária, por isso são definidas as relações de precedência entre as tarefas da aplicação, determinando ordens entre as mesmas. Uma tarefa T_i é precedida por uma outra T_k (T_k ? T_i), se T_i pode iniciar sua execução somente após o término da execução de T_k..

Relações de precedência expressam a dependência entre as tarefas. Em alguns casos, as relações são representadas na forma de um grafo acíclico orientado, onde os nós correspondem às tarefas do conjunto e os arcos descrevem as relações de precedência existentes entre as tarefas.

Outra forma de relações entre tarefas são as relações de exclusão. Uma tarefa T_i exclui T_k quando a execução de uma seção crítica de T_k que manipula o recurso compartilhado não pode executar porque T_i já ocupa o recurso [FARINES,2000].

4. CONCEITOS DE ALGORITMO GENÉTICO

Neste trabalho abordamos os conceitos do Algoritmo Genético, bem como sua estrutura e características. Mas existem outros tipos de algoritmos genéticos com características específicas para resolverem outros tipos de problemas: esses algoritmos são detalhados no Apêndice A.

A primeira teoria sobre a origem da variabilidade das formas de vida foi desenvolvida por Jean Baptiste Lamark, que sugeriu uma teoria evolucionária no "uso e desuso" de órgãos; e de Thomas Robert Malthus, que propôs que fatores ambientais tais como doenças e carência de alimentos, limitavam o crescimento de uma população.

Algumas décadas depois, Charles Darwin apresentou em 1858 sua teoria de evolução através de seleção natural, simultaneamente com outro naturalista inglês Alfred Russel Wallace. No ano seguinte, Darwin publica a sua grande obra: *On the Origin of Species by Means of Natural Selection* com a sua teoria completa.

Darwin não conseguiu explicar de onde vinham às variações das espécies e como eram herdadas. Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética, foi redescoberto pelos cientistas e teve grande influência sobre os futuros trabalhos relacionados à evolução. A moderna teoria da evolução combina a genética e as idéias de Darwin e Wallace sobre a seleção natural, criando o princípio básico de Genética Populacional: a variabilidade entre indivíduos em uma população de organismos que se reproduzem sexualmente é produzida pela mutação e pela recombinação genética [DAMM, 2005].

Este princípio foi desenvolvido durante os anos 30 e 40, por biólogos e matemáticos de importantes centros de pesquisa. Nos anos 50 e 60, muitos biólogos começaram a desenvolver simulações computacionais de sistemas genéticos. Entretanto, foi o professor John Holland da Universidade de Michigan a partir de experiências com implementações de redes neurais artificiais, percebeu um nítido elo entre a biologia e a computação, ou seja, as máquinas podiam ser levadas a adaptar-se ao meio ambiente, tais como os seres vivos. Publicados inicialmente em 1975 (*Adaptation in Natural and Artificial Systems)* os Algoritmos Genéticos são

algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética, procurando solucionar problemas complexos de otimização e aprendizado computacional, sem exigir muito conhecimento prévio destes. Trata-se de um método de resolução de propósitos gerais, cujo princípio é independente da natureza do problema específico [HOLLAND, 1975].

A Teoria da Evolução supõe que a evolução das espécies está diretamente ligada à capacidade dos indivíduos se adaptarem ao seu habitat, onde apenas os mais aptos sobrevivem e deixam descendentes. Nesse processo seletivo são gerados indivíduos com algumas características salientes, que se destacam dos demais de sua espécie. Com isso aumentam sua probabilidade de sobrevivência e geram descendentes ainda melhores, evoluindo a espécie quanto à sua adaptação ao meio ambiente. Por outro lado, indivíduos que não se destacam tendem a não sobreviverem e a não gerar descendentes, sendo gradativamente eliminados. Os algoritmos genéticos procuram imitar, de uma forma computacional, algumas etapas desse processo evolutivo das espécies. Recebem esse nome porque assumem como referência a codificação de DNA, encontrada no núcleo da célula de cada indivíduo. Esse código genético é representado por cromossomos, que são cadeias de genes [HOLLAND, 1975].

Toda tarefa de busca e otimização possui vários componentes, entre eles: o espaço de busca, onde são consideradas todas as possibilidades de uma solução de um problema e a função de avaliação (ou função de custo), uma maneira de avaliar os membros do espaço de busca. As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado associado ao problema a ser solucionado. Geralmente, a simulação em computadores pode ser muito complexa. No entanto, as técnicas de computação evolucionária operam sobre uma população de candidatos em paralelo. Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões. Portanto, os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, principalmente em quatro aspectos [HOLLAND, 1975]:

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

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

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

-AGs utilizam regras de transição probabilísticas e não determinísticas;

- As modificações de um AG a fim de modelar variações do problema original são muito fáceis de ser implementadas, diferentemente de muitos outros sistemas de otimização.

O principal benefício dos Algoritmos Genéticos é que são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais, como exemplo, o detalhamento dos parâmetros para iniciar a busca.

Outra vantagem é a simplificação que eles permitem na formulação e solução de problemas de otimização, os algoritmos simples normalmente trabalham com descrições de entradas formadas por cadeias de bits de tamanho fixo, outros tipos podem trabalhar com cadeias de bits de tamanho variável, como exemplo, AG's usados para Programação Genética. AG's possuem um paralelismo implícito decorrente da avaliação independente de cada uma dessas cadeias de bits, ou seja, pode-se avaliar a viabilidade de um conjunto de parâmetros para a solução do problema de otimização em questão. O AG é indicado para a solução de problemas de otimização complexos, NP-Completos, como exemplo, o problema do "caixeiro viajante", que envolvem um grande número de variáveis e, conseqüentemente, espaços de soluções de dimensões elevadas.

O problema do caixeiro viajante é um caso típico de otimização combinatória, onde o caixeiro viajante deseja visitar N cidades (vértices) de uma certa localização sendo que, entre alguns pares de cidades existem rotas (arcos ou arestas), através das quais o mesmo pode viajar a partir de uma cidade para outra. Cada rota tem um número associado que pode representar a distância ou o custo necessário para percorrê-la. Assim, o caixeiro viajante deseja encontrar um caminho que passe por

cada uma das N cidades apenas uma vez e, além disso, que tenha um custo menor que certo valor; onde o custo do caminho é a soma dos custos das rotas percorridas [CUNHA,2000].

Além disso, em muitos casos onde outras estratégias de otimização falham na busca de uma solução, os AGs convergem, sendo numericamente robustos, ou seja, não são sensíveis a erros de arredondamento no que se refere aos seus resultados finais [MIRANDA,2004].

4.1. Terminologia dos Algoritmos Genéticos

A terminologia utilizada pelos Algoritmos Genéticos consiste basicamente de analogias extraídas da Biologia. Os termos mais utilizados são [MITCHELL,1996]:

Aptidão (*Fitness*)

Probabilidade que um organismo possui para reproduzir ou uma função do número de descendentes (fertilidade) que este organismo possui;

Cromossomo ou Indivíduo

Uma estrutura de solução, uma cadeia de símbolos candidata para o problema;

População

Conjunto dos cromossomos ou indivíduos que compõe cada geração;

Gene

Divisão conceitual ou bloco funcional de um cromossomo, capaz de codificar uma característica;

Posição

Posição em que um gene se localiza no cromossomo;

Genótipo

Refere-se ao jogo particular de genes contido em um genoma (coleção completa de material genético de um organismo, envolvendo todos os cromossomos);

Cruzamento

Troca de partes entre dois cromossomos;

Mutação

Mudança ou troca de uma ou mais posição de um cromossomo;

4.2. Estrutura do Algoritmo Genético

O Algoritmo Genético inicia-se com uma estrutura onde se replicam os mecanismos da natureza e espera-se emergir um comportamento espontaneamente, sendo um procedimento iterativo que mantém sempre uma população de estruturas que são candidatas à solução do problema.

Inicialmente, é gerada uma população formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é avaliada: para cada indivíduo é dado uma nota, ou índice, refletindo sua habilidade de adaptação a determinado ambiente. Uma porcentagem dos mais adaptados é mantida, enquanto os outros são descartados (darwinismo). Os membros mantidos pela seleção podem sofrer modificações em suas características fundamentais através de mutações e cruzamento (crossover) ou recombinação genética gerando descendentes para a próxima geração. Este processo, chamado de reprodução, é repetido até que uma solução satisfatória seja encontrada.

Embora possam parecer simplistas do ponto de vista biológico, estes algoritmos são suficientemente complexos para fornecer mecanismo de busca adaptativo poderoso e robusto.

Abaixo segue uma seqüência de passos ilustrando o ciclo do algoritmo genético:

- Definição do problema a ser tratado e uma possível solução para o mesmo;
- Criação da representação cromossômica;
- Construção da população inicial;
- Definição de um mecanismo de avaliação e seleção dos cromossomos ou indivíduos;
- Definição dos operadores genéticos de crossover (reprodução) e mutação;
- Seleção de parâmetros para uma boa convergência de evolução: tamanho da população, seleção dos cromossomos reprodutores e descarte dos menos aptos, critério de sobrevivência dos cromossomos e critério de parada do algoritmo genético [AZAMBUJA,2001].

Abaixo na Figura 3, segue o ciclo de execução do Algoritmo Genético, mostrando todas as etapas do processo de evolução:

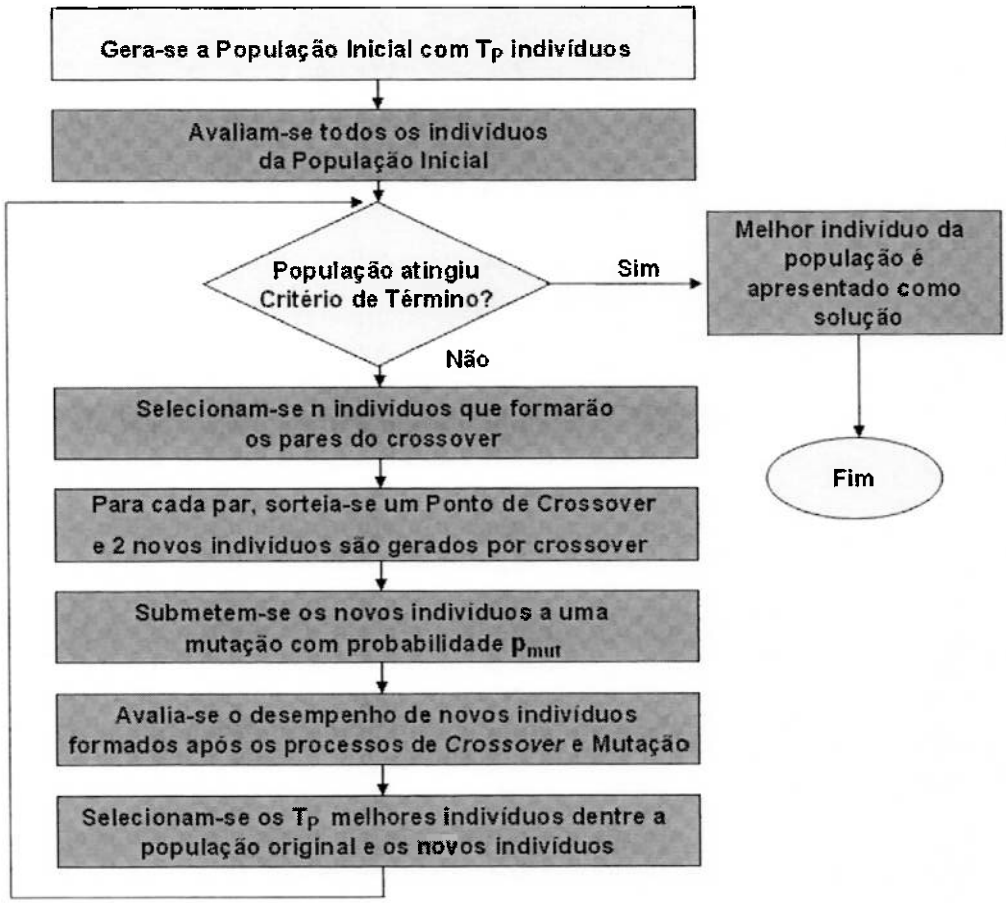

Figura 3: Ciclo do Algoritmo Genético [AZAMBUJA,2001]

4.2.1. Operadores Genéticos

O cromossomo ou indivíduo é formado por uma cadeia de símbolos que representa uma possível solução para o problema a ser resolvido. Na maioria das aplicações são representados por um vetor ou um conjunto de vetores, cujos elementos podem ser: binários, inteiros ou reais de acordo com o tipo de problema. A essa representação define-se o alfabeto do AG, a representação de cada parâmetro de acordo com o alfabeto adotado é chamada de gene. Define-se a população como um conjunto de pontos no espaço de busca utilizado, representada por um conjunto de indivíduos.

O AG inicia-se com uma definição aleatória da população inicial com n indivíduos que representam possíveis soluções para o problema. Para cada indivíduo calcula-se a aptidão através da função objetivo, que mensura a adaptação do indivíduo como solução do problema. Verifica-se então se os critérios de parada foram atingidos, sendo que estes critérios geralmente são a aptidão atingida pelo melhor indivíduo, em conjunto com o número de gerações. A geração representa uma iteração completa do algoritmo genético, gerando uma nova população [MIRANDA,2003]. O processo de seleção é composto basicamente por dois métodos: o crossover e a mutação.

O Crossover, também chamado de Recombinação Cromossômica consiste na troca de genes entre dois pais para gerar um ou mais filhos, formando uma nova população com filhos provenientes de recombinações [DAMM, 2005]. Quando se utiliza a codificação binária, a recombinação consiste na troca de partes dos cromossomos dos pais para gerar os filhos, essa troca pode ser definida por um ou mais pontos sorteados aleatoriamente, segue exemplo na Figura 4 utilizando-se um único ponto de crossover:

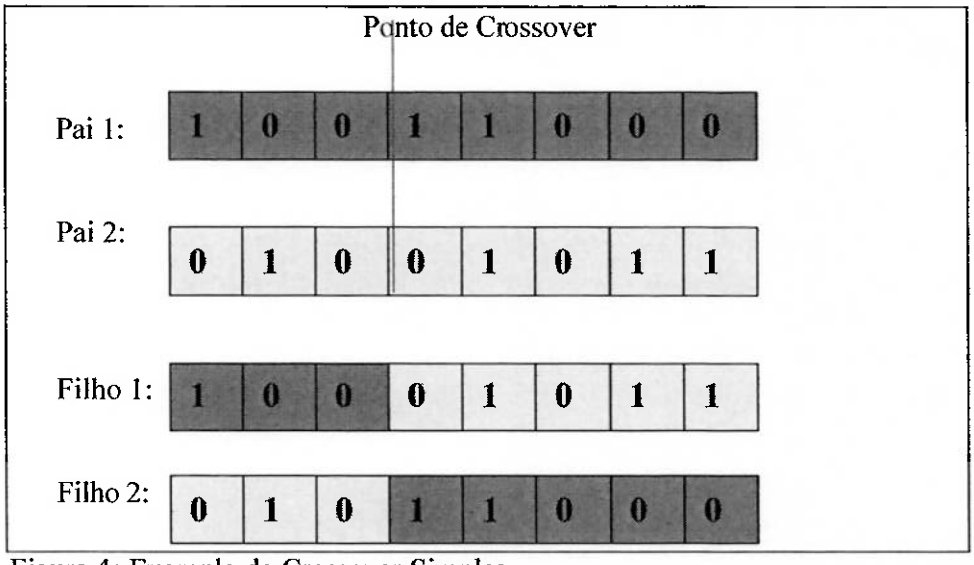

Figura 4: Exemplo de Crossover Simples

Os métodos de vetores binários podem ser aplicados para vetores reais, entretanto o nível de recombinação seria inferior, pois a troca não alteraria os valores reais. Por isso, existem alguns métodos específicos, como: Recombinação Discreta, Recombinação Intermediária e Linear [DAMM, 2005].

A Recombinação Discreta cria uma máscara de cada filho que é sorteada independentemente, segue abaixo o exemplo [DAMM, 2005]:

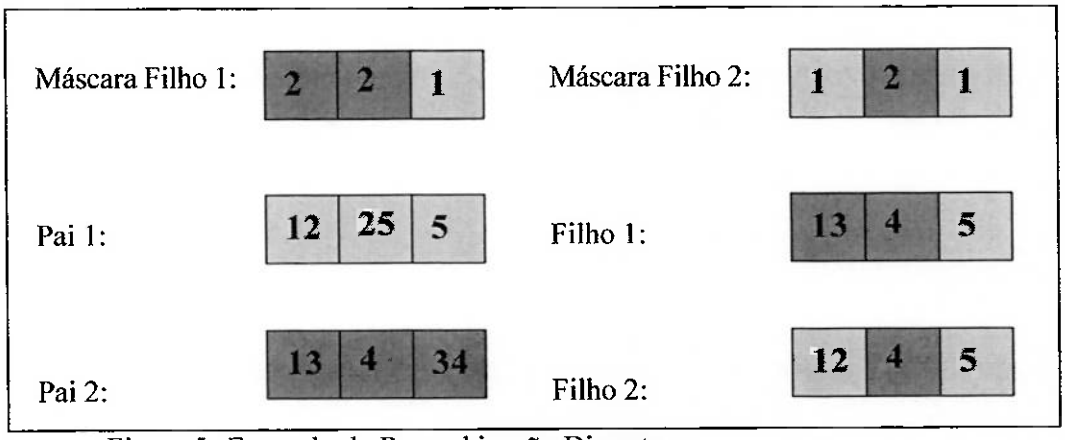

Figura 5: Exemplo de Recombinação Discreta

No exemplo, a formação do filho 1 segue a máscara Filho 1 definida, onde o filho número 1 é formado pelas duas primeiras posições do Pai número 2 e a última posição do Pai número 1 [DAMM, 2005].

Na Recombinação Intermediária, as máscaras são sorteadas para definirem os pesos a da equação, por exemplo:

Filho = Pai1 + a* (pai2-pai1), para o Filho 1 utiliza-se o a1 e sucessivamente.

a1 = 0,5 / 1,1 / -0,1

a2 = 0,1 / 0,8 / 0,5

A Recombinação Linear é bem parecida com a Recombinação Intermediária, a única diferença é que ao invés da máscara é sorteado apenas um valor escalar por filho, para definirem os pesos a da equação [DAMM, 2005]:

Filho = Pai1 + a* (pai2-pai1)

a1 = 0,5

a2 = - 0,1

Para vetores inteiros (problemas de permutação) é utilizado o Crossover Cíclico ou o PMX (Partially Matched Crossover). Permutação é simplesmente uma seqüência ordenada sem duas ou mais repetições de cada elemento retirado de um conjunto fixo de símbolos, e com comprimento máximo, ou seja, a ordem é relevante [MIRANDA,2003].

O Crossover cíclico é iniciado a partir da troca de uma única posição, verificando até que gene será alterado formando um ciclo, segue abaixo exemplo do Crossover Cíclico. Neste exemplo, a posição de número 3 foi escolhida e o gene foi alterado, o número 8 do Pai 2 foi para a terceira posição do Pai 1, mas como já existia um número 8 na posição 7, este número foi para o Pai 2 e o número 6 foi para o Pai 1 e assim sucessivamente, até não repetir nenhum número e encerrar o ciclo.

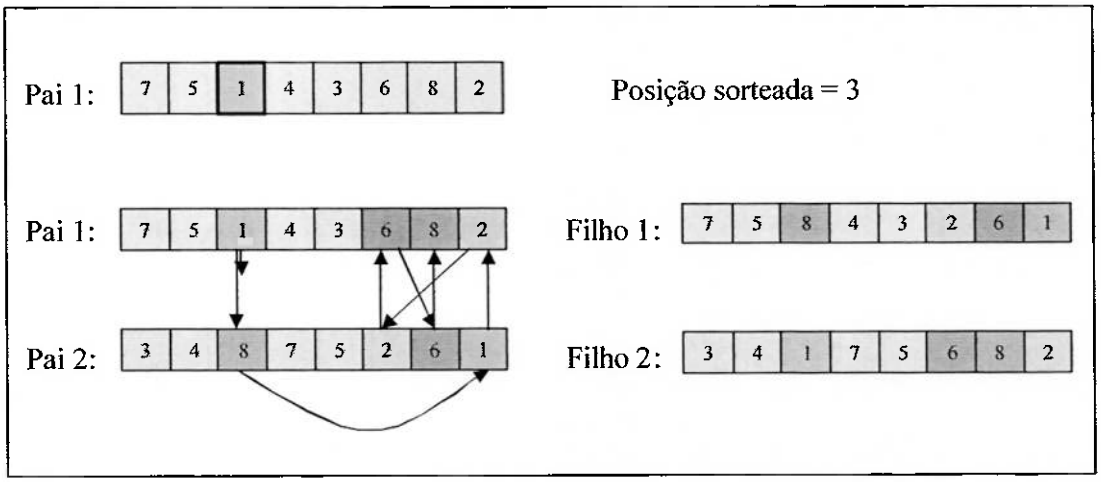

Figura 6: Exemplo de Crossover Cíclico

O crossover PMX (Partially Matched Crossover) é realizado através da seleção de uma seção no cromossomo (através de dois pontos de Crossover), onde o material genético dos 2 pais será integralmente trocado. Depois, são feitas mudanças de posição a posição para não existirem repetições nos filhos que serão gerados [MIRANDA,2003].

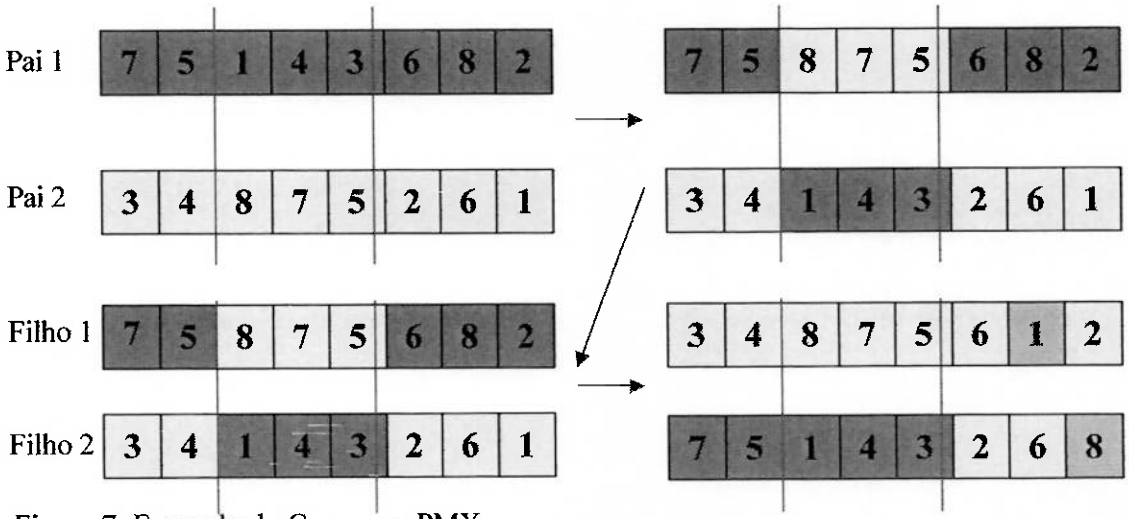

Figura 7: Exemplo do Crossover PMX

A mutação é um operador genético que consiste em substituir genes em um mesmo cromossomo. Existem vários tipos de mutação, como exemplo, podemos citar a mutação para vetor binário, vetor real e inteiro [MIRANDA,2003].

No vetor binário altera-se um ou mais bits do cromossomo pelo seu complemento binário, por exemplo, onde é 0 altera-se para 1.

Nos vetores reais altera-se um gene do cromossomo através de um sorteio de incremento ou decremento, sendo que este valor deverá ser pequeno, segue exemplo:

Incremento = 0,1 ou Decremento = 0,7

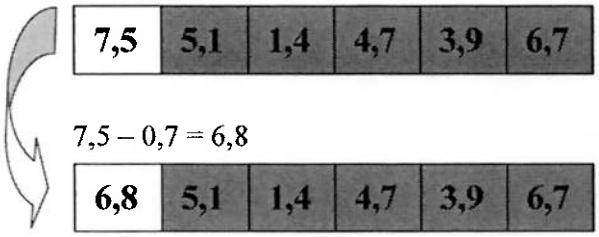

Figura 8: Exemplo Mutação com Vetor Real

A posição (1) sorteada possui o valor 7,5, e o decremento sorteado possui o valor 0,7, então o valor 7,5 será trocado por 6,8, ou seja, 7,5 – 0,7 = 6,8 novo valor da posição (1).

Para valores inteiros utilizamos o método da permutação ocorrendo à troca de dois ou mais genes, como exemplo:

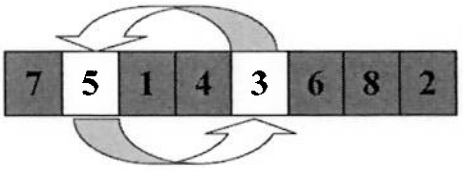

Figura 9: Exemplo de Mutação com Vetor Inteiro

4.2.2. Seleção para o Crossover

Após a geração da população inicial e avaliação dos indivíduos é realizada a seleção dos indivíduos que farão parte do Crossover, ou seja, quais indivíduos serão pares para o cruzamento. Existem vários métodos de seleção dos indivíduos com

maior capacidade de sobrevivência para a escolha do crossover (processo de reprodução), são eles: Truncamento, Ranking, Roleta, Amostragem Universal Estocástica e Torneio.

No método de Truncamento ordenam-se todos os indivíduos de acordo com a aptidão. Selecionam-se os N primeiros indivíduos para o Crossover que são alocados em pares aleatoriamente, o valor de N depende da taxa utilizada para o Crossover. Exemplo: Tamanho da População Inicial é 100 e a taxa utilizada é de 40%, sendo assim, os 40 melhores indivíduos formarão os pares para o Crossover [DAMM, 2005].

No método Ranking (*Rank-Based Fitness Assignment)* ordena-se todos os indivíduos de acordo com a aptidão, ou seja, a probabilidade de um indivíduo ser sorteado depende unicamente da sua aptidão.

O método da Roleta consiste em somar as avaliações dos indivíduos selecionados para a reprodução e dividir a roleta entre os indivíduos selecionados de acordo com a sua avaliação. A probabilidade de um indivíduo ser sorteado para o cruzamento é o valor da sua avaliação, pois se o valor da sua avaliação é alto, o indivíduo pode ter uma porcentagem maior na roleta, sendo assim, ao "girar" a roleta a fim de obter aleatoriamente um indivíduo a parte maior será do indivíduo que possui uma avaliação melhor [MIRANDA,2003].

Segue abaixo figura 10 ilustrando o método da Roleta.

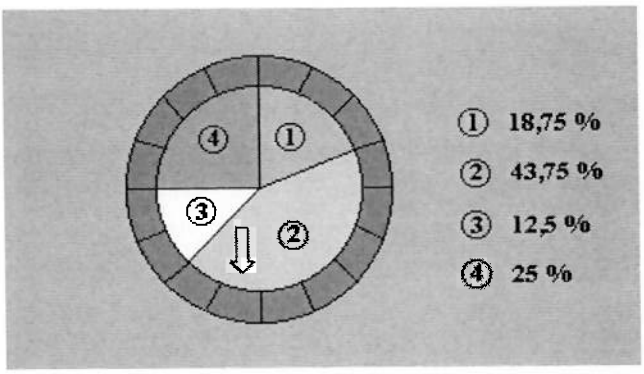

Figura 10: Exemplo do Método da Roleta [SILVA,2003]

O método Amostragem Universal Estocástica é semelhante ao método da Roleta, a distribuição e divisão dos indivíduos é a mesma, a forma de selecionar é diferente. Na roleta temos uma única "agulha" (cursor) que ao girar a roleta seleciona um único indivíduo, já na Amostragem Universal Estocástica utiliza-se n "agulha" igualmente espaçada girando-as em conjunto uma só vez. Segue abaixo, figura 11 ilustrando o método de Amostragem Universal Estocástica.

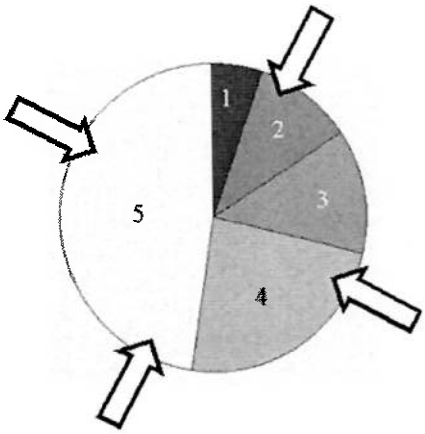

Figura 11: Exemplo de Amostragem Universal Estocástica [LOPES,2003]

No método do Torneio utilizam-se sucessivas disputas para realizar a seleção, existe o *Tour* que é uma variável que indica o número de indivíduos que serão envolvidos em cada torneio. A cada torneio um sub-grupo de *tour* indivíduos é

selecionado da população, o melhor indivíduo do sub-grupo é selecionado como resultado do torneio. Por exemplo, se é definido o *tour* = 3, significa que serão sorteados três indivíduos verificando qual será o melhor entre eles, o melhor é armazenado e o sorteio é realizado novamente até que se obtenha o número desejado de indivíduos [LOPES,2003].

4.2.3. Seleção dos indivíduos para a próxima geração (Reinserção)

O Operador de Seleção dos Algoritmos Genéticos é uma analogia com a seleção biológica natural de Darwin. Realizar uma seleção significa escolher dentre os indivíduos pertencentes à população, aqueles que servirão para criar descendentes para a próxima geração e quantos descendentes cada um deverá gerar. Uma seleção pode ser denominada "muito fraca", se resultar em evolução muito baixa e "muito forte" se resultar em populações de indivíduos mais fortes (o termo "forte" aqui utilizado refere-se aos indivíduos mais adaptados ao meio ambiente; esta adaptação é que direciona o processo evolutivo). Isto significa que os indivíduos mais adequados vivem mais e geram mais descendentes, os quais herdam suas qualidades. Ficando evidente, que o mecanismo da seleção tem por finalidade enfatizar um filtrador de indivíduos na população, para aumentar a perspectiva de gerar indivíduos melhores segundo algum critério. Os critérios utilizados são: Reinserção Pura, Uniforme, Baseado na Aptidão e Elitismo [ICHIHARA,1998].

A Reinserção Pura consiste na troca de toda população (substituição da população antiga pelos filhos), a cada ciclo N novos indivíduos são criados substituindo a população anterior. Uma vez que a Reinserção Uniforme selecionam-se (a partir de qualquer um dos métodos vistos na seleção para o Crossover) os melhores indivíduos da população total (Filhos + Pais).

No entanto, o critério Baseado na aptidão seleciona os melhores indivíduos da população, sendo que a população total é ordenada. No Elitismo uma parte da população (os melhores pais-Elite) é mantida para a próxima geração [PACHECO,2006].

4.3. Aplicações do Algoritmo Genético

Os AGs possuem uma larga aplicação em muitas áreas científicas, seguem algumas delas [MIRANDA,2004],[PACHECO,2006]:

- Alocação e Escalonamento de Tarefas;

- Data Mining (Mineração dos Dados): Classificação de Clientes;

- Aprendizagem de Máquina, problemas com muitas variáveis e espaços das dimensões elevadas: Gerenciamento de Carteiras de Fundos de Investimento;

- Problemas de Otimização Complexos: Otimização de Funções Matemáticas, Otimização Combinatorial, Otimização de Planejamento, Otimização de Distribuição e de Negócios;

- Seleção de rotas: Otimização de Rota de Veículos e Problema do Caixeiro Viajante;

- Programação Genética: gera a listagem de um programa, em uma determinada linguagem para que um determinado conjunto de dados de entrada forneça uma saída desejada;

- Computação Evolutiva: gera programas que se adaptam a mudanças no sistema ao longo do tempo;

- Otimização Evolutiva Multi-Critério: otimiza funções com múltiplos objetivos que sejam conflitantes;

- Ciências Biológicas: modela processos biológicos para o entendimento do comportamento de estruturas genéticas;

- Autômatos autoprogramáveis;

- Síntese de Circuitos Analógicos: para uma certa entrada e uma saída desejada, tensão, o AG gera a topologia, o tipo e o valor dos componentes do circuito; Síntese de Circuitos Eletrônicos e Otimização de Layout de Circuitos;

- Síntese de Protocolos: determina quais funções do protocolo devem ser implementadas em hardware e quais devem ser implementadas em software para que um certo desempenho seja alcançado;

- Gerenciamento de redes: supervisiona o tráfego nos *links* e nas filas dos "*buffers*" de roteadores para descobrir rotas ótimas e para reconfigurar as rotas existentes no caso de falha de algum link.

5. ESCALONAMENTO DE TAREFAS ATRAVÉS DO AG

O problema de escalonamento de tarefas consiste em encontrar um ordenamento de tarefas ao longo do tempo, obedecendo a restrições de precedências (tarefas) e recursos (processadores) [HOLLAND, 1975]. É um problema de otimização combinatória bem conhecido pertencente à classe de problemas intratáveis (NP completos) [AZAMBUJA,2003].

O problema de escalonamento de tarefas é formulado como um problema de minimização do tempo requerido para se executar um conjunto de tarefas com restrições [AZAMBUJA,2003].

5.1. Motivação para utilizar o AG no Escalonamento

Não se conhecem algoritmos com complexidade polinomial que garantam sempre a obtenção de uma solução ótima global para o problema de escalonamento de tarefas. Os algoritmos exatos conhecidos têm complexidade exponencial. Por isso, a utilização de algoritmos exatos restringe-se a pequenas instâncias do problema, pois do contrário, o tempo computacional necessário seria inviável.

As diferentes heurísticas utilizadas em métodos encontrados na literatura [AZAMBUJA,2001], embora tenham se mostrado eficientes, para várias instâncias do problema de escalonamento há situações onde a solução encontrada não é de boa qualidade. A deficiência dessas abordagens é que uma única solução é gerada, não existindo a possibilidade de se explorar soluções alternativas que, eventualmente, poderiam levar a um menor custo [AZAMBUJA,2001].

Comparando-se os algoritmos exatos com os algoritmos genéticos percebe-se que o algoritmo genético proporciona diversas soluções para o mesmo problema, podendo existir a solução ótima ou não, e também os AGs trabalham com uma população e não com um único ponto como no caso dos algoritmos exatos.

Para um melhor entendimento da diferença entre os algoritmos exatos e o AG, vemos uma demonstração do algoritmo genético da figura 12, só que adaptado para demonstrar uma cituação que pode ocorrer em um escalonamento, esta figura contém as tarefas (1, 2, 3 e 7) e mais uma precedência entre a tarefa (3 -> 7) e (1 -> 3) com custo 0 para simplificar o problema.

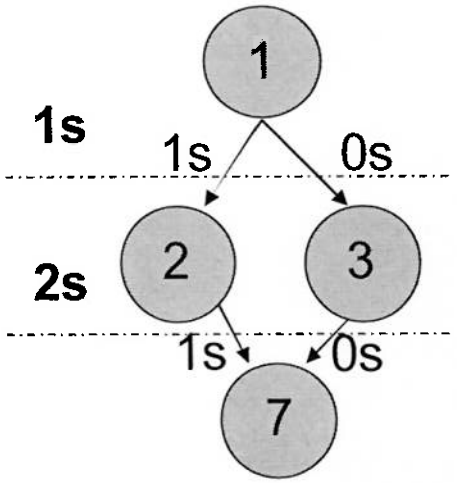

Figura 12: Exemplo de grafo para um escalonador de tarefas simplificado

Os grafos das figuras 12 e 13 são representados da seguinte forma: os círculos (nós) representam as tarefas, ao lado esquerdo do grafo temos o tempo de execução de cada tarefa, o qual corresponde a todas as tarefas daquele mesmo alinhamento. Os links entre os nós representam as precedências, por exemplo, a tarefa 9 só pode ser iniciada após as execuções das tarefas 3 e 7.

Os números ao lado de cada link apresentam o custo de comunicação entre as tarefas, caso estejam alocadas em processadores diferentes. Assim, se a tarefa 7 estiver alocada no mesmo processador da tarefa 9, esta última pode ser iniciada imediatamente após a realização de 7. Porém, se estiverem em processadores distintos, à tarefa 9 só poderá ser realizada 5 unidades de tempo após o término da tarefa 7.

Seguindo as regras de precedência ilustradas na figura 12 do problema proposto, vemos que para as quatro tarefas em questão, temos somente duas possibilidades. Conforme tabela abaixo:

Tabela 1: Tarefas possíveis para o problema da figura 12

	Tarefas			
Possibilidade 1	1	2	3	7
Possibilidade 2	1	3	2	7

Dentre as possibilidades vistas na tabela acima, existem variações dos dois processadores para cada tarefa, podendo somente executar uma tarefa por vez ou executar a comunicação entre eles. Observe também que não há concorrência de processamento com qualquer atividade externa, sendo assim, os processadores estão totalmente dedicados à execução das quatro tarefas em questão.

Calculando as possíveis combinações, obtemos um total de 16 indivíduos para cada possibilidade de tarefas, ou seja, para as duas possibilidades temos um total de 32 individuos.

A avaliação dos indivíduos nesse escalonador torna-se viável, pois as tarefas envolvidas foram reduzidas, ficando apenas quatro tarefas. Isso permite a visualização de quais são as soluções ótimas, sendo que no caso da execução de um algoritmo exato teríamos necessariamente somente uma das soluções ótimas.

A capacidade do AG em mostrar uma solução ótima global, torna-o interessante porque não buscamos somente a melhor solução, mas um conjunto de boas soluções. Veja a ilustração da tabela 2, onde foram encontradas todas as soluções possíveis para o problema, incluindo a melhor solução com custo de 7, destacadas em negrito.

Tabela 2: Avaliação do custo para o problema da figura 12

	Tarefas				Avaliação do Custo	
Indivíduos Possibilidade 1	1	2	3	7		
	Processadores				Tempo Comunicação	Custo Total
1	1	1	1	1	0	8
2	1	1	1	2	2	10
3	1	1	2	1	1	10
4	1	1	2	2	3	7
5	1	2	1	1	3	9
6	1	2	1	2	1	7
7	1	2	2	1	4	12
8	1	2	2	2	2	10
9	2	1	1	1	2	10
10	2	1	1	2	4	12
11	2	1	2	1	1	7
12	2	1	2	2	3	8
13	2	2	1	1	3	11
14	2	2	1	2	1	7
15	2	2	2	1	2	10
16	2	2	2	2	0	8
	Tarefas				Avaliação do Custo	
Indivíduos Possibilidade 2	1	3	2	7		
	Processadores				Tempo Comunicação	Custo Total
17	1	1	1	1	0	8
18	1	1	1	2	2	10
19	1	1	2	1	3	10
20	1	1	2	2	1	7
21	1	2	1	1	1	9
22	1	2	1	2	3	7
23	1	2	2	1	4	12
24	1	2	2	2	2	10
25	2	1	1	1	2	10
26	2	1	1	2	4	12
27	2	1	2	1	3	7
28	2	1	2	2	1	8
29	2	2	1	1	1	11
30	2	2	1	2	3	7
31	2	2	2	1	2	10
32	2	2	2	2	0	8

Para esse problema temos um total de quatro indivíduos para cada possibilidade da tabela 1, com o menor custo possível que iremos analisar em especial de dois indivíduos, o número 4 e o 6, na tabela 3.

Nessa demonstação vamos supor uma hipotética situação, em que o algoritmo genético escolha justamente o indivíduo 6 para ser executado, mas enquanto as tarefas são executadas ocorre uma interrupção da comunicação entre os processadores 1 e 2, somente no segundo instante de tempo, marcado em vermelho na tabela 3. Esse erro de comunicação levaria o custo de execução subir de 7 para 8. Observe que esse atraso poderia ser evitado caso a segunda opção encontrada pelo AG fosse justamente o indivíduo número 4, pois no segundo instante de tempo não há comunicação entre os processadores e já que no indivíduo número 6 também foi executado a tarefa número 1 no processador 1, então, não há qualquer problema na troca do indivíduo 6 para o 4.

Tabela 3: Avaliação dos indivíduos 4 e 6 com problema de comunicação

Indivíduo 6

Processador 1								
Tarefas	Tempos de execução							
	1s	2s	3s	4s	5s	6s	7s	8s
Tarefa 1	-							
Comunicação		-						
Comunicação			-					
Tarefa 3			-	-				
Tarefa 3				-	-			

Processador 2								
Tarefas	Tempos de execução							
	1s	2s	3s	4s	5s	6s	7s	8s
Comunicação		-						
Comunicação			-					
Tarefa 2			-	-				
Tarefa 2				-	-			
Tarefa 7					-	-	-	
Tarefa 7						-	-	-

Indivíduo 4

Processador 1								
Tarefas	Tempos de execução							
	1s	2s	3s	4s	5s	6s	7s	8s
Tarefa 1	-							
Tarefa 2		-	-					
Comunicação				-				
Tarefa 7					-	-	-	

Processador 2								
Tarefas	Tempos de execução							
	1s	2s	3s	4s	5s	6s	7s	8s
Comunicação				-				
Tarefa 3		-	-					
Tarefa 7					-	-	-	

5.2. Estrutura para a construção de um escalonador através do AG

Este trabalho demonstra a estrutura para a construção de um escalonador através do Algoritmo Genético, com o objetivo de escolher o melhor caminho (menor tempo) para executar determinadas tarefas, seguindo determinadas regras (precedência). Essa estrutura será utilizada para determinar as prioridades relativas entre as tarefas que competem por um mesmo recurso de *hardware*, permitindo assim, selecionar em que instante de tempo cada tarefa será executada por um determinado recurso.

O problema proposto aqui é um escalonamento de 12 tarefas entre dois processadores distintos de estrutura paralela sem restrição ou balanceamento de carga. A Figura 13 apresenta um grafo do problema.

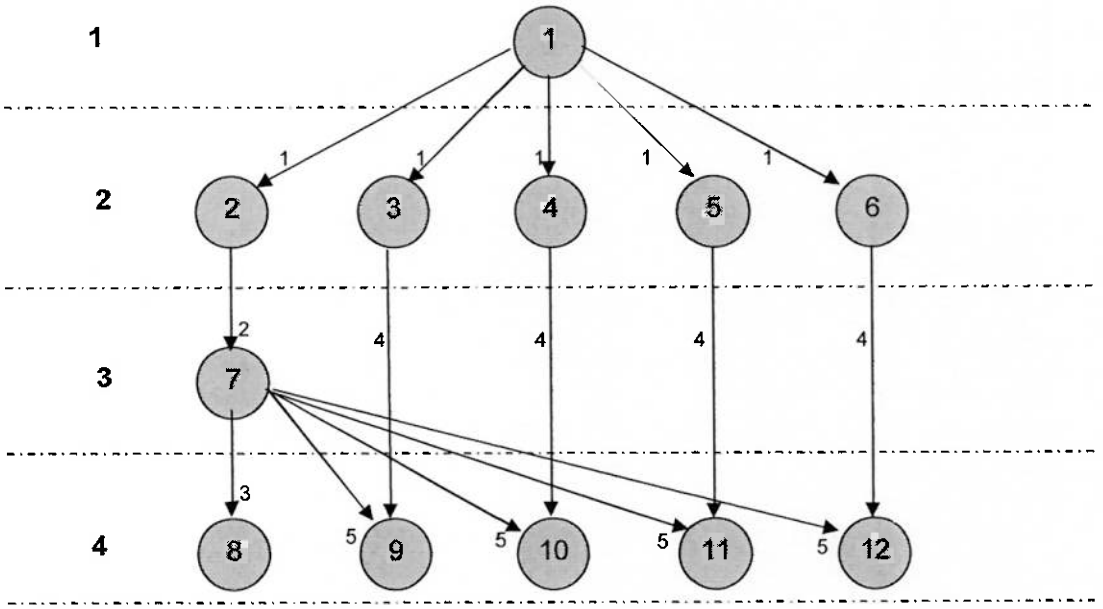

Figura 13: Exemplo de grafo para um escalonador de tarefas

Existem trabalhos com soluções diferentes para esse problema, sendo que a maioria das técnicas encontradas para resolvê-los estão baseadas em métodos heurísticos, ou seja, o algoritmo garante encontrar uma solução próxima da ótima em menos tempo que o exponencial [AZAMBUJA,2003].

A técnica de algoritmo genético para resolver esses problemas consiste em gerar possíveis soluções codificadas na forma de indivíduos, que interagem com outros na mesma população, procurando através de operações genéticas, gerar indivíduos mais aptos, ou seja, obter a solução ótima [AZAMBUJA,2001].

Os métodos propostos que utilizam algoritmo genético mostram que seu desempenho pode ter grande dependência do número de processadores e tarefas que serão executadas, bem como, dos parâmetros internos [AZAMBUJA,2003].

5.2.1. A escolha da Representação Cromossômica

A forma de representação do cromossomo pode variar conforme o problema em questão. No caso do escalonamento de tarefas através do Algoritmo Genético entre processadores, adotaremos as seguintes formas de representação:

Representação 1: Os indivíduos são representados através de uma matriz dupla contendo as tarefas e os respectivos processadores, são 12 tarefas (1-12) que podem ser alternadas entre 2 processadores (1 ou 2).

Representação 2: Os indivíduos são representados através de uma matriz Sij, onde para cada tarefa i existe um processador j associado. O cromossomo é estático, ou seja, com um tamanho conhecido. Nesse caso, o número de tarefas será de 12 que poderão ser executadas no processador 1 ou 2. Os indivíduos ou cromossomos, em ambas as abordagens, são representados através de uma matriz, onde para cada tarefa existe um processador associado. Essa matriz é composta por 12 tarefas variando de 1 a 12, sem repetições, que podem ser executadas tanto no processador 1 quanto no processador 2.

Como o problema é bem conhecido e limitado a 12 tarefas, opta-se por um cromossomo estático, ou seja, com um tamanho conhecido e pré-determinado. A Tabela 4 mostra um exemplo da representação adotada.

Tabela 4: Representação 1 - Exemplo da representação do indivíduo

Tarefas	12	2	4	6	10	7	3	8	11	5	9	1
Processadores	1	1	2	1	2	2	1	1	1	1	1	2

Tabela 5: Representação 2 - Exemplo da representação de n indivíduos

Tarefas												
Indivíduo 1	12	2	4	6	10	7	3	8	11	5	9	1
Indivíduo 2	1	2	3	4	5	6	7	8	9	10	11	12
Indivíduo 3	12	11	10	9	8	7	6	5	4	3	2	1
Indivíduo n	1	2	3	4	5	6	12	11	9	8	7	10

Processadores												
Indivíduo 1	1	1	2	1	2	2	1	1	1	1	1	2
Indivíduo 2	2	2	2	1	2	2	1	1	1	2	1	2
Indivíduo 3	2	1	2	1	2	2	1	1	2	1	1	2
Indivíduo n	1	2	2	1	2	2	1	2	1	2	1	2

Ao implementar uma determinada aplicação em um software, deve-se levar em consideração a escolha da representação do cromossomo. Uma vez que em algumas linguagens há restrições do tipo de variável a ser utilizado, obrigando-nos a escolher outras formas de representação.

5.2.2. População Inicial

Para a criação da população inicial pode-se criar uma rotina especifica, a fim de validar as possíveis tarefas para cada posição da matriz. Imaginando-se que a posição seja como no exemplo de 1 a 12 da matriz do indivíduo, isso representa justamente a seqüência a ser seguida na execução das tarefas.

Conforme podemos ver na tabela abaixo a seqüência das tarefas são válidas seguindo o grafo proposto da figura 13, ou seja, a execução das tarefas atende as restrições de precedência do escalonador de tarefas.

Tabela 6: Indivíduo para 12 tarefas e suas respectivas posições

Posição	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Tarefas	1	2	3	4	5	6	7	8	9	10	11	12

Para gerar uma população inicial seguindo os critérios de precedência, pode-se gerar inicialmente uma variável que guardará às possíveis das tarefas a serem executadas naquele instante de tempo, sem repeti-las. Com essas opções de tarefas geradas, podemos fazer o sorteio da primeira à última posição, até que acabem todas as opções das tarefas. Segue a tabela 7 com um dos possíveis indivíduos da população inicial:

Tabela 7: Opções de tarefas por ciclo

1° Ciclo	Opções de Tarefas
Posição 1	1

2° Ciclo	Opções de Tarefas				
Posição 2	2	3	4	5	6
Posição 3		3	4	5	6
Posição 4			4	5	6
Posição 5				5	6
Posição 6					6

3° Ciclo	Opções de Tarefas
Tarefas	7

4° Ciclo	Opções de Tarefas				
Posição 8	8	9	10	11	12
Posição 9		9	10	11	12
Posição 10			10	11	12
Posição 10				11	12
Posição 12					12

Outra forma de gerar a população inicial é sorteando de 1 a 12 e garantindo que os números não sejam repetidos. Nesse caso a validação pode ser realizada posteriormente em uma segunda etapa, descartando assim os indivíduos inválidos.

Como na geração da população inicial dos processadores não há qualquer restrição referente ao escalonamento proposto da figura 13, logo não é necessário qualquer tipo de validação. Mas poderíamos considerar alguma restrição para a escolha do processador a ser validado, como por exemplo, o processador 1 deve executar sempre a segunda tarefa, assim já na população inicial poderíamos seguir essa regra. Observe que isso não garantirá que a solução final obedeça tal critério, pois para isso faz-se necessário uma validação final.

Adotaremos aqui para exemplificar o problema do AG do escalonador da figura 13 uma população inicial de 100 indivíduos.

5.2.3. Avaliação da População

Depois de gerar a população inicial deve-se fazer a avaliação da população, com o objetivo de medir a sua performance dentro do problema a ser tratado. No caso do problema do escalonador de tarefas os indivíduos são avaliados de acordo

com o custo da tarefa e o tempo de comunicação entre elas, se houver mudança de processador e se as tarefas tiverem precedências.

O melhor indivíduo é aquele que possui o menor tempo para distribuir as tarefas entre dois processadores. A avaliação é realizada verificando se existe precedência entre as tarefas, se existir é verificado se houve alteração de processador e qual o tempo de comunicação entre as tarefas, ou seja, é necessário aguardar o tempo de comunicação para iniciar a próxima execução. Se não houver precedência à tarefa é executada imediatamente.

Veja na tabela 8 um exemplo de um indivíduo a ser avaliado utilizando o escalonador de tarefas da figura 13.

Tabela 8: Avaliação de um indivíduo

Tarefas	1	2	3	4	5	6	7	8	9	10	11	12
Processadores	1	2	1	1	2	2	1	1	2	1	1	2
Tempo de cada Tarefa	1	2	2	2	2	2	3	4	4	4	4	4
Tempo de comunicação	0	1	0	0	1	1	1	0	1	0	0	1

Na tabela 9 temos os tempos de execução de cada tarefa escalonados no tempo, observe que cada quadrado representa uma unidade do custo da tarefa ou de comunicação entre os processadores. Baseando-se na divisão das tarefas entre os processadores 1 e 2 , temos que o custo total desse indivíduo é 27, pois somente quando o processador 1 terminar a tarefa número 11 é que as atividades realmente serão concluídas.

Tabela 9: Avaliação de um indivíduo

Processador 1																				
Tarefas	**Tempos de execução**																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Tarefa 1	-																				
Tarefa 3		-	-																		
Tarefa 4				-	-																
Tarefa 7						-	-	-													
Tempo de Comunicação									-												
Tarefa 8										-	-	-	-								
Tarefa 10														-	-	-	-				
Tarefa 11																		-	-	-	-

Processador 2																					
Tarefas	**Tempos de execução**																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Tarefa 2		-	-																		
Tempo de Comunicação				-																	
Tarefa 5					-	-															
Tempo de Comunicação							-														
Tarefa 6								-	-												
Tempo de Comunicação										-											
Tarefa 9											-	-	-	-							
Tempo de Comunicação															-						
Tarefa 12																-	-	-	-		
Tempo de Comunicação																				-	

Para o problema proposto iremos chamar o tempo total de custo, como uma forma de simplificação da notação. Uma vez que a idéia de avaliação será bem empregada nesse trabalho.

5.2.4. Critério de Término

É preciso definir um número de gerações que deverão ser pesquisadas pelo algoritmo evolucionário. Inicialmente deverão ser definidos valores para testar a qualidade dos resultados. Normalmente é utilizada entre 20 e 50 gerações, podendo este número variar conforme o número de vértices do grafo a ser escalonado, bem como os recursos computacionais utilizados.

Torna-se muito importante que o algoritmo seja implementado, observando a evolução das soluções encontradas, podendo ser encerrado após algumas gerações em que os resultados apresentam-se estagnados.

O critério de término é a etapa que define quando é finalizado o algoritmo com o objetivo de fornecer uma solução ótima. Esse critério é geralmente definido através de um parâmetro pré-configurado pelo programador com as quantidades de gerações necessárias para obter um resultado satisfatório (convergência).

Esse parâmetro além de afetar diretamente na qualidade do resultado obtido também afetará o tempo de convergência do algoritmo. O que em alguns casos pode ser decisivo no critério da escolha do parâmetro.

Como no problema proposto o resultado final é a minimização do tempo de execução das tarefas por dois processadores. A escolha da configuração de um bom critério de término pode determinar o quão próximo o resultado estará da melhor solução, ou seja, o melhor indivíduo da população.

Adotaremos como exemplo um critério de término de 200 gerações para a finalização do algoritmo, mas esse valor tanto pode ser alterado quanto se tornar uma variável de sistema.

5.2.5. Método de Seleção

Pode-se empregar diversas soluções para o método de seleção, sendo que neste trabalho foi escolhido como exemplo o método conhecido como "Torneio". Esse método foi escolhido por apresentar uma baixa pressão seletiva e, assim, evitar uma convergência prematura já que os critérios para geração da população inicial são

elitistas. Além disso, é um método mais simples para implementação e mais fácil para alterações, caso necessário.

Uma das formas de implementar tal solução é utilizar o método Torneio Simples com *tour* de três. Para realização desse processo é feito um sorteio de três indivíduos verificando qual é o melhor entre eles, o melhor é armazenado, e então é realizado o sorteio novamente até que se obtenha os números desejados de indivíduos. Os indivíduos sorteados são escolhidos para a seleção podendo ser sorteados e escolhidos mais de uma vez, pois a cada sorteio o melhor indivíduo de uma rodada pode não ser o melhor indivíduo da próxima.

Depois dessa etapa os indivíduos escolhidos que formarão os pares estão prontos para o crossover, que é a próxima etapa do algoritmo genético.

Na tabela 10 temos um exemplo prático de três indivíduos escolhidos aleatoriamente na população para realizar o torneio de três. Para os três indivíduos da figura, o torneio irá escolher o indivíduo número 1 o qual possui o menor custo (21) entre eles.

Tabela 10: Exemplo do método Torneio com *tour* de 3

Tarefas – Indivíduo 1	1	2	3	4	5	6	7	8	9	10	11	12	**Total do Tempo**
Tempo de cada Tarefa	1	2	2	2	2	2	3	4	4	4	4	4	
Processadores	1	2	1	1	2	2	1	1	2	1	1	2	**21**
Tempo de comunicação	0	1	0	0	1	1	1	0	1	0	0	1	
Tarefas – Indivíduo 2	1	3	4	2	5	6	7	8	9	10	11	12	**Total do Tempo**
Tempo de cada Tarefa	1	2	2	2	2	2	3	4	4	4	4	4	
Processadores	1	1	1	1	1	1	1	1	2	1	1	2	**28**
Tempo de comunicação	0	0	0	0	0	0	0	0	1	0	0	1	
Tarefas – Indivíduo 3	1	6	5	4	3	2	7	12	11	10	9	8	**Total do Tempo**
Tempo de cada Tarefa	1	2	2	2	2	2	3	4	4	4	4	4	
Processadores	2	2	2	2	2	2	2	2	2	2	2	2	**34**
Tempo de comunicação	0	0	0	0	0	1	0	0	0	0	0	0	

5.2.6. Crossover

Entres os métodos demonstrados no item de introdução do algoritmo genético, vimos diversas opções de crossover, mas para o problema sugerido iremos escolher o método do Crossover Cíclico.

Nesse método é escolhida uma posição através de um sorteio e essa posição é alterada com o outro par formando um ciclo, todos os genes dentro do ciclo serão trocados e os que permanecerem fora do ciclo não serão trocados. Foi adotado neste trabalho que os processadores dos indivíduos também iram acompanhar as tarefas, mas isso é opcional. Podemos considerar alternativas como manter os processadores na mesma seqüência.

Seguindo a seqüência de exemplos dos indivíduos para o problema de escalonamento de tarefas sugerido, temos na tabela 11 o indivíduo 3 escolhido no método de seleção com o seu respectivo crossover com outro indivíduo escolhido em outro torneio de 3.

Para o crossover cíclico temos como exemplo que a posição sorteada seja a número 4. Segue abaixo o resultado do crossover cíclico na tabela 11.

Tabela 11: Exemplo do Crossover Cíclico

Tarefas – Indivíduo 3 -Pai	1	6	5	4	3	2	7	12	11	10	9	8
Processadores	1	1	1	1	1	2	2	2	2	2	2	2
Tarefas – Indivíduo X -Pai	1	6	4	3	5	2	7	9	11	10	12	8
Processadores	2	2	2	2	1	1	2	1	2	2	2	1

Tarefas – Indivíduo 3-Filho	1	6	4	3	5	2	7	12	11	10	9	8
Processadores	1	1	2	2	1	2	2	2	2	2	2	2
Tarefas – Indivíduo X-Filho	1	6	5	4	3	2	7	9	11	10	12	8
Processadores	2	2	1	1	1	1	2	1	2	2	2	1

5.2.7. Mutação

Para a realização da mutação no caso do problema do escalonamento de tarefas é selecionado um indivíduo e uma posição através de um sorteio. O indivíduo escolhido tem sua posição alterada com a próxima.

Após esse processo, podem existir problemas de novos indivíduos inválidos, então uma maneira é verificar e corrigir, ou descartar esses indivíduos inválidos. Esse processo de correção verifica se existe alguma precedência entre as tarefas e se existir verifica a ordem, até que todas as posições sejam corrigidas, sendo assim é garantido que no final, todos os indivíduos sejam válidos.

Assim como no processo de crossover fica a critério do desenvolvedor verificar se existe a necessidade de fazer a mutação da tarefa juntamente com o processador ou não. No nosso exemplo da tabela 12 temos um indivíduo válido e veremos um indivíduo sofrer o processo de mutação descrito na tabela 12, juntamente com o processador. Após a escolha aleatória da posição da mutação, é gerado o indivíduo válido da tabela 13 ou um inválido da tabela 14.

Na tabela 15 temos o mesmo indivíduo da tabela 14 só que depois de passar pelo processo de verificação e correção do mesmo.

Tabela 12: Exemplo de um indivíduo válido

Tarefas	1	2	3	4	5	6	7	8	9	10	11	12
Processadores	1	1	2	1	2	2	1	2	1	1	1	2

Tabela 13: Exemplo de um indivíduo válido depois da mutação

Tarefas	1	2	4	3	5	6	7	8	9	10	11	12
Processadores	1	1	1	2	2	2	1	2	1	1	1	2

Tabela 14: Exemplo de um indivíduo inválido depois da mutação

Tarefas	1	2	3	4	5	6	8	7	9	10	11	12
Processadores	1	1	2	1	2	2	2	1	1	1	1	2

Tabela 15: Exemplo de um indivíduo corrigido após a mutação

Tarefas	1	2	3	4	5	6	7	8	9	10	11	12
Processadores	1	1	2	1	2	2	1	2	1	1	1	2

Observe que a idéia da mutação é justamente a troca de dois ou mais elementos em posições próximas, como nos exemplos acima onde foi sorteada uma posição e trocada com a posição mais próxima. Podemos também utilizar posições distintas, ou seja, trocar um indivíduo por outro em uma posição também sorteada.

5.3. Desempenho dos algoritmos genéticos

A fim de identificar o desempenho sugerido no item 2.1.2.1. – Complexidade do Algoritmo, o qual vimos que o desempenho de um algoritmo para uma determinada entrada está diretamente relacionado com os custos versus a execução do algoritmo para essa determinada entrada, conforme a seguir:

E = Entrada= Variavél=parâmetros
AG = Algoritmo Genético

Desempenho (E):=custo (execuções(**AG**,E))

Onde, o custo é o peso estabelecido para as operações fundamentais, ou seja, qual é o impacto dessa operação no tempo total de execução. Sendo que o exec(AG,E) representa a seqüência de execuções de operações fundamentais efetuadas pelo algoritmo genético para uma entrada E. Em termos práticos, podemos através da tabela 16 identificar as operações e variáveis envolvidas no algoritmo genético. Este exemplo segue o escalonador de tarefas demonstrado no item 5, onde foram definidos os operadores e parâmetros para o exemplo da figura 13.

Tabela 16: Operações envolvidas no Desempenho do Algoritmo Genético

OPERAÇÕES ENVOLVIDAS NO DESEMPENHO DO ALGORITMO GENÉTICO		
OPERAÇÃO	**DESCRIÇÃO**	**ENTRADA**
GERAÇÃO DA POPULAÇÃO INICIAL	Sorteio das tarefas e processadores, respeitando restrições de precedência.	Número de tarefas; Número de processadores; Tarefa sorteada; Processador sorteado; Tamanho da população; Precedência;
AVALIAÇÃO	Atribui o tempo de execução do indivíduo com o seu respectivo processador.	Tamanho da população; Número de tarefas; Número de processadores; Quantidade de troca de processadores;
SELEÇÃO DO TORNEIO DE 3	Método de escolha do melhor indivíduo entre três.	Tamanho da População Inicial; Resultado da Avaliação;
CROSSOVER CICLÍCO	Operação de alteração de tarefas até que todas as tarefas sejam diferentes para cada indivíduo.	Indivíduos da Seleção do Torneio de 3; Porcentagem dos indivíduos para o crossover; Tamanho da população; Número de tarefas; Sorteio da posição de inicio do crossover;
MUTAÇÃO DE 1 TAREFA E 1 PROCESSADOR	Alteração da posição de uma tarefa e de um processador.	Porcentagem dos indivíduos para mutação; Tamanho da População;
CONDIÇÃO TÉRMINO	Número de gerações a ser realizado.	Quantidade de geração para o término; Critério de reinserção;
ORDENAÇÃO DA AVALIAÇÃO	Ordena os indivíduos conforme a sua avaliação do melhor para o pior.	Tamanho da População;
SORTEIO	Gera números aleatórios para a geração da população inicial, posição do torneio e posição do crossover.	Universo das Tarefas; Universo das Posições; Universo do Torneio; Tipo de Sorteio;
AVALIAÇÃO DA PRECEDÊNCIA	Avalia entre os números sorteados conforme a precedência.	Precedência;

Vimos também que a entrada do algoritmo influencia o desempenho do algoritmo pelo seu tamanho, valor e característica em uma determinada operação fundamental. Veremos na tabela 17 as entradas escolhidas, a descrição e a quantidade como exemplo de uma solução para o problema sugerido.

Tabela 17: Entradas envolvidas no Desempenho do Algoritmo Genético

ENTRADAS ENVOLVIDAS NO DESEMPENHO DO ALGORITMO GENÉTICO		
ENTRADAS	**DESCRIÇÃO ENTRADA**	**EX. REFERENTE À FIGURA 13 QUANTIDADE/VALOR ADOTADO**
NÚMERO DE TAREFAS	Quantidade de tarefas a serem executadas pelo escalonador.	12
NÚMERO DE PROCESSADORES	Quantidade de processadores existentes no sistema.	2
TAREFA SORTEADA	Possíveis tarefas para gerar a população inicial.	Depende da precedência
PROCESSADOR SORTEADO	Processador sorteado para executar uma determinada tarefa.	Processador 1 ou Processador 2
PRECEDÊNCIA	Indica a relação de dependência entre as tarefas.	0 a 3 precedências conforme figura 12
QUANTIDADE DE TROCA DE PROCESSADORES	Número de vezes que o processador é alterado no mesmo indivíduo.	0 a 12
RESULTADO DA AVALIAÇÃO	É o valor atribuído ao indivíduo, ou seja, o custo do mesmo.	Valor de cada tarefa do indivíduo + tempo de comunicação.
INDIVÍDUOS DA SELEÇÃO DO TORNEIO DE 3	Indivíduos selecionados para participar do crossover.	*****
PORCENTAGEM DOS INDIVÍDUOS PARA O CROSSOVER	Indica o número de indivíduos que serão selecionados para participar do crossover.	Taxa de crossover =50% da população.
SORTEIO DA POSIÇÃO DE INICIO DO CROSSOVER	Indica a posição inicial da execução do crossover.	1 a 12 posições
PORCENTAGEM DOS INDIVÍDUOS PARA MUTAÇÃO	Número de indivíduos que sofreram a mutação.	Taxa de mutação= 10% da população
TAMANHO DA POPULAÇÃO	Número de indivíduos existentes no AG.	100
QUANTIDADE DE GERAÇÃO PARA O TÉRMINO	Número de vezes que o algoritmo é executado.	200 gerações
CRITÉRIO DE REINSERÇÃO	Escolha do tipo de reinserção do AG.	Os melhores indivíduos da população total.
UNIVERSO DAS TAREFAS	Tipo de representação numérica das tarefas ou número de tarefas.	Número inteiro entre 1 e 12.
UNIVERSO DAS POSIÇÕES	Número de posições do indivíduo.	12
UNIVERSO DO TORNEIO	Número de indivíduos pertencentes à população	100
TIPO DE SORTEIO	Número de participantes do torneio.	Torneio de 3

6. TRABALHOS FUTUROS

Devido a limitações de tempo e escopo do trabalho, vemos que nem tudo pode ser abordado ou desenvolvido. Sendo assim gostaria de colocar alguns pontos que considero importante ou relevante para um trabalho futuro.

Para um melhor entendimento das diferentes formas e construção de um algoritmo genético, pode-se desenvolver em uma linguagem de programação, um sistema de AG flexível o suficiente para obter resultados comparativos entre os diversos operadores genéticos como: definição de diferentes taxas de crossover e mutação, escolha do tipo de crossover e mutação (Método da roleta, Torneio, Ranking, etc), escolha do tipo de seleção para reinserção de novos indivíduos, definição da população inicial e de seu tamanho.

Uma boa avaliação de um algoritmo deve sempre ser atrelada a um comparativo teórico e prático, com objetivo de mostrar o quanto o AG é realmente melhor ou pior que os outros algoritmos.

O item de Análise do Projeto tratado neste trabalho pode ser melhor desenvolvido, com o objetivo de tornar mais prática a escolha do tipo de solução abordada para cada tipo de problema. Pode-se também fazer essa análise considerando o fator desempenho e precisão necessária para cada aplicação.

7. CONCLUSÃO

Para qualquer problema proposto sendo ou não um escalonador de tarefas é sempre importante conhecer muito bem o problema, quanto à sua complexidade por exemplo. Isso nos leva a entender a necessidade de se considerar o Escalonador de Tarefas ou o Algoritmo Genético como parte da solução do problema conforme ilustrado nesse trabalho.

As etapas do AG aplicado ao escalonamento de tarefas nos ajudam a obter um melhor entendimento do funcionamento do sistema como um todo, e quais são as diferenças básicas entre os algoritmos exatos e os AGs, motivando a escolha do AG para esse tipo de problema.

Portanto, através deste trabalho verificamos que o Algoritmo Genético é uma solução eficaz, pois proporciona diversas soluções, podendo existir a melhor solução durante as inúmeras gerações executadas no processo de evolução dos indivíduos.

8. REFERÊNCIAS BIBLIOGRÁFICAS

AZAMBUJA, Rogério Xavier; KLEIN, Felipe Vieira; SANTOS, Luiz C. V,. *"Uma Abordagem Orientada à Exploração automática de Soluções para o Problema de Escalonamento com Restrição de Recursos"*. I Congresso Brasileiro de Computação – CBComp, 2003.

AZAMBUJA, Rogério X.; SANTOS, Luiz Claudio V.; BORGES, Paulo Sérgio S., *"Escalonamento com restrição de recursos utilizando algoritmo genético"*, I Congresso Brasileiro de Computação, 2001.

CUNHA, Cláudio Barbieri. Artigo *"Aspectos Práticos da Aplicação de Modelos de Roteirização de Veículos a Problemas Reais"*. Revista Transportes da ANPET – Associação Nacional de Pesquisa e Ensino em Transportes - Novembro/2000.

DAMM, Ricardo de Brito. *"Um Estudo dos Parâmetros de Controle de um Algoritmo Genético"*. Dissertação (Proposta de Mestrado em Matemática Aplicada) – Instituto de Matemática e Estatística (IME), Universidade de São Paulo – USP, São Paulo, 2005.

DIVERIO, Tiarajú Asmuz; MENEZES, Paulo Blauth. Livro *"Teoria da Computação"*. Porto Alegre: Instituto de Informática UFRGS. Ed. Sagra Luzzatto, 2003.

FARINES, Jean Marie; Fraga, J.S.; Oliveira, R.S. Artigo *"Sistemas em Tempo Real"*. Universidade Federal de Santa Catarina – Florianópolis, Julho/2000.

FERREIRA, Cândida. *"Entrevista sobre Programação de Expressão Genética"*. Entrevista elaborada por Ricardo Carvalho – Revista Alagamares em 23/09/2006. Link acessado em 30/10/2006: http://www.alagamares.net/artigo227.html.

HOLLAND, John H., Livro *"Adaptation in natural and artificial systems"*, University of Michigan Press, Bradford Books edition, 1975.

ICHIHARA, Jorge de Araújo. *"Um método de Solução Heurística para a Programação de Edifícios dotados de Múltiplos Pavimentos-Tipo."* Tese submetida á Universidade Federal de Santa Catarina para a obtenção do título de Doutor em Engenharia de Produção. Florianópolis, 1998.

LOPES, Luis R. M. Artigo *"Fundamentos de Algoritmo Genético."* Programa de Engenharia de Sistemas e Computação (PESC). Universidade Federal do Rio de Janeiro - UFRJ. Junho, 2003.

MELLO, Reinaldo Xavier. "*Um Modelo de Escalonamento Colaborativo de Eventos Baseado em Corrotinas*". Tese submetida á Pontifícia Universidade Católica do Rio de Janeiro para a obtenção do título de Mestre - Rio de Janeiro, Setembro / 2005.

MIRANDA, Clay R. dos Santos; OLIVEIRA, Gina M. Barbosa; SANTOS, Jamilson Bispo. "*Algoritmos Genéticos aplicados em Data Mining para obtenção de Regras Simples e Precisas*". VI Simpósio Brasileiro de Automação Inteligente, São Paulo - Bauru, 2003.

MIRANDA, Marcio Nunes. Artigo "*Algoritmos Genéticos: Fundamentos e Aplicações*". GTA – Grupo de Teleinformática e Automação. UFRJ – Universidade Federal do Rio de Janeiro, 2004.

MITCHELL, M. Livro "*An Introduction to Genetic Algorithms*". MIT, Cambridge: 1996.

PACHECO, Marco Aurélio Cavalcanti. Artigo "*Algoritmos Genéticos: Princípios e Aplicações*". Pontifica Universidade Católica do Rio de Janeiro, PUC. Link acessado em 28/10/2006: http://www.ica.ele.puc-rio.br/inteligencia_computacional.

SILVA, Alexandre Tadeu Rossini; NAZARENO, Gustavo Setúbal; SCHNEIDER, André Marcelo. "*Resolvendo o Problema do Cavalo do Xadrez Utilizando o Algoritmo Genético*". Anais do Congresso V Encontro de Estudantes de Informática do Tocantins. Palmas, TO. Outubro, 2003.

TOSCANI, Laira Vieira; VELOSO, Paulo A. S. Livro "*Complexidade de Algoritmos*". Porto Alegre: Instituto de Informática UFRGS. Ed. Sagra Luzzatto, 2002.

9. APÊNDICE A – OUTROS TIPOS DE ALGORITMOS

Todos os fundamentos apresentados até o momento basearam-se nos algoritmos genéticos simples. Existem outros tipos de algoritmos genéticos que foram desenvolvidos para problemas mais específicos, como exemplo, podemos citar: Genitor, CHC e Algoritmos Híbridos. Podemos citar também um algoritmo inovador: Programação de Expressão Genética (PEG) [FERREIRA,2006].

O Genitor é um algoritmo cujos melhores pontos encontrados são preservados na população, este procedimento é denominado *elitismo*. Na prática isto resulta em uma busca mais agressiva, geralmente bastante efetiva podendo existir uma convergência prematura. Cada indivíduo selecionado e cruzado com seu parceiro é colocado no lugar do pior indivíduo da população anterior, a aptidão é atribuída de acordo com um "*ranking*", ou seja, a aptidão de cada indivíduo assume valores discretos.

Outro AG que coleciona os melhores indivíduos da população atual é o CHC (*Cross Generational Elitist Selection, Heterogeneous Recombination and Cataclysmic Mutation*). Após o cruzamento, feito aleatoriamente, os N melhores indivíduos são coletados levando-se em consideração a população atual e a população gerada após o cruzamento, retirando-se os indivíduos duplicados. Este método impõe uma busca mais agressiva, assim como no Genitor. O ponto de crossover utilizado é sempre a metade do cromossomo. Para solucionar o problema de convergência prematura é utilizada uma alta taxa de mutação, sempre preservando o melhor indivíduo da população.

No entanto, os AG's nem sempre são a melhor solução para problemas de otimização específico. Desta forma, os algoritmos híbridos utilizam os AG's como ponto de partida juntamente com as técnicas tradicionais de otimização. Essa mistura de técnicas tradicionais com os AG's introduzem uma espécie de aprendizado no AG, pois os cromossomos utilizados foram resultado da técnica denominada "*hill-climbing*", utilizada nos métodos de otimização tradicionais, que utilizam derivadas resultando em uma melhor solução para problemas mais específicos [MIRANDA,2004].

A Programação de Expressão Genética (PEG) foi criada pela Dra. Cândida Ferreira no ano de 2000 [FERREIRA,2006] é também um algoritmo genético, pois utiliza o mesmo princípio darwiniano da seleção natural para evoluir modelos computacionais ou, de forma mais geral, descobrir soluções para problemas. Assim, todos os algoritmos genéticos (Algoritmo Genético, Programação Genética, Programação Expressão Genética, e outros) têm pelo menos três coisas em comum: (1) utilizam populações de indivíduos (soluções para o problema); (2) introduzem variação genética na população usando um ou mais operadores genéticos como, por exemplo, a mutação ou a recombinação; e (3) selecionam, de acordo com a aptidão, os indivíduos que depois se reproduzem para criar a nova geração [FERREIRA,2006].

Quanto às diferenças, existem várias, mas a PEG, por ser um algoritmo tão inovador e tão versátil, oferece algumas vantagens não só relativamente à Programação Genética (o algoritmo mais próximo da PEG), mas também aos Algoritmos Genéticos. Por exemplo, os Algoritmos Genéticos são essencialmente usados em problemas de otimização de parâmetros, com os parâmetros representados de uma forma muito simples em um genoma linear. Este sistema funciona bastante bem e até melhor que a maior parte dos métodos matemáticos que existem para resolver o mesmo tipo de problema, mas mesmo assim, por vezes a evolução do AG em um determinado ponto não consegue encontrar melhores soluções. A PEG, devido ao fato de evoluir estruturas em árvore e por ser um sistema multigénico (isto é, podem evoluir múltiplas árvores), pode também ser utilizada para otimização de parâmetros (a PG não pode, pois só consegue evoluir sistemas com uma árvore só).

A PEG é uma nova técnica da computação evolutiva para criação de programas de computador que utiliza cromossomos lineares compostos por genes organizados estruturalmente numa cabeça e numa cauda. Os cromossomos funcionam como genoma e estão sujeitos a modificações por meio de mutação, transposição, inversão e recombinação. Os cromossomos codificam árvores de expressão, sendo estas o objeto da seleção. A criação destas duas entidades distintas (cromossomos e árvores de expressão) com funções diferentes permite que o algoritmo tenha um grande desempenho, superando a Programação Genética entre 100 a 60.000 vezes [FERREIRA,2006].

No sistema genótipo/fenótipo da PEG, o cromossomo e as árvores correspondem respectivamente ao DNA e às proteínas. Os cromossomos da PEG são também estruturas lineares compostas por genes que depois são expressos em árvores. Estas árvores exibem um grau de complexidade muito superior às estruturas lineares que lhe deram origem e, similarmente às proteínas, podem desempenhar um número virtualmente infinito de funções. Assim, no sistema da PEG temos programas de computador extremamente sofisticados codificados em sistemas simples lineares. E como na natureza, é mutando os cromossomos que na PEG se consegue uma grande diversidade de programas. Mas tanto na Programação Genética como nos Algoritmos Genéticos não existe um sistema genótipo/fenótipo definido e as estruturas básicas destes algoritmos (cromossomos nos AG e árvores na PG) são aquilo que o Richard Dawkins chama replicadores simples, isto é, estruturas que funcionam ao mesmo tempo como genótipo e fenótipo. As estruturas usadas nos AG - os cromossomos - são semelhantes ao DNA na medida em que são estruturas lineares extremamente simples. E dada a sua simplicidade, a diversidade de funções que estas estruturas podem desempenhar é bastante limitada.

Na Programação Genética, os replicadores (as árvores) são consideravelmente mais complexos, mas como estas estruturas não são codificadas num genoma linear, elas só podem ser transformadas através de modificações nelas aplicadas diretamente. Isto faz lembrar um pouco o que se pensa ter acontecido nos primórdios da vida na terra, onde replicadores simples do tipo do RNA acumulavam tanto a função de genótipo como fenótipo. No entanto, a explosão da vida na Terra, em toda a sua diversidade e esplendor, só foi possível quando estes replicadores simples evoluíram para sistemas mais complexos com um genótipo e um fenótipo perfeitamente definidos e autônomos. A escala dos algoritmos genéticos artificiais (a Vida é obviamente o algoritmo genético supremo), foi mais ou menos isto que aconteceu com a invenção da PEG: os replicadores rudimentares dos AG e da PG foram suplantados pelos replicadores sofisticados da PEG com genótipo e fenótipo definidos [FERREIRA,2006].