Aplicando Algoritmos Genéticos na Alocação de Tarefas em Sistemas Distribuídos de Tempo Real

André C. Nácul[+], Maurício Lima Pilla[+], Rômulo Silva de Oliveira[*]
{anacul,pilla}@inf.ufrgs.br, romulo@lcmi.ufsc.br

[+] Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064, CEP 91501-970,
Porto Alegre - RS, Brasil

[*] Universidade Federal de Santa Catarina
Dep. de Automação e Sistemas
Caixa Postal 476, CEP 88040-900,
Florianópolis - SC, Brasil

Resumo

O objetivo deste trabalho é analisar o comportamento dos algoritmos genéticos quando aplicados na alocação de tarefas de tempo real em sistemas distribuídos. O sistema distribuído de tempo real formalizado inclui tarefas periódicas e esporádicas as quais devem ser alocadas em tempo de projeto e escalonadas segundo uma política de prioridades fixas. O objetivo primário da alocação é garantir os *deadlines* das tarefas, enquanto o objetivo secundário é minimizar a ocupação dos meios de comunicação. O estudo descrito neste artigo visa caracterizar a taxa de sucesso do algoritmo genético quando usado para alocar aplicações sabidamente viáveis.

Abstract

This work aims at analyzing the behavior of genetic algorithms on distributed real time systems scheduling. The system consists of periodic and sporadic tasks that must be scheduled in design time and according to a fixed priority policy. The main goal of the scheduling it to guarantee that all deadlines are met, while minimizing the use of the interconnection bus. The work described in this paper shows the success rate of genetic algorithms when used to schedule feasible applications.

1 Introdução

Em função do baixo custo dos processadores, dos avanços na área de redes de computadores e da necessidade física de algumas aplicações, soluções distribuídas são cada vez mais empregadas em sistemas de tempo real. Nestes sistemas, as tarefas da aplicação encontram-se distribuídas por diversos processadores, os quais são conectados entre si através de uma rede de comunicação.

No contexto de sistemas distribuídos, o escalonamento tempo real é geralmente resolvido em duas etapas. Na primeira etapa é feita a alocação das tarefas aos processadores. Na segunda etapa é feito o escalonamento local de cada processador, tomado individualmente. Este escalonamento local considera como carga as tarefas alocadas na primeira etapa. Como cada tarefa é alocada estaticamente a um processador, aumenta a importância de uma alocação adequada.

No contexto dos sistemas de tempo real críticos, o objetivo primário da alocação é garantir que todas as tarefas sempre poderão concluir suas respectivas execuções antes do *deadline*. Para isto o algoritmo de alocação tenta diversas soluções de alocação. Com respeito ao objetivo primário, qualquer solução de alocação que forneça garantia para os *deadlines* é igualmente satisfatória. Muitas vezes o algoritmo de alocação pode possuir também objetivos secundários. Por exemplo, alocar preferencialmente determinadas tarefas em determinados processadores.

Problemas de alocação apresentam alto grau de complexidade, e isso ocorre por dois motivos principais. Em primeiro lugar, encontrar o escalonamento ótimo é um problema NP-completo. Além disso, cada problema de escalonamento contém detalhes particulares, que implicam em uma mudança do algoritmo de busca das soluções.

Desta forma, torna-se necessário o uso de algum tipo de heurística para limitar o espaço de busca. Uma alternativa é o uso de algoritmos genéticos, os quais inspiram-se nos mecanismos utilizados pela Seleção Natural para o desenvolvimento de indivíduos melhores.

É possível encontrar na literatura um grande conjunto de problemas aos quais foram aplicados algoritmos genéticos [DAV91]. Trabalhos recentes sobre a aplicação de algoritmos genéticos no escalonamento de computadores com arquiteturas paralelas podem ser encontrados em [ZOM99] e [COR99]. Em [ZOM99] é analisado como variações nos parâmetros do algoritmo afetam o seu desempenho, tanto o seu tempo de execução quanto a qualidade dos resultados obtidos. Em [COR99] é usada uma solução que combina algoritmo genético com heurísticas baseadas em lista. O resultado são soluções melhores porém com um tempo de execução maior.

O emprego de algoritmos genéticos em problemas de tempo real é muito menos freqüente. O uso desta técnica em tempo de execução (*on-line*) é inviável em função do seu tempo de execução. Mesmo sua aplicação em tempo de projeto (*off-line*) não é relatada. Os autores desconhecem trabalhos que utilizem algoritmos genéticos em problemas de alocação semelhantes ao descrito na seção 2.

O objetivo deste trabalho é analisar o comportamento dos algoritmos genéticos quando aplicados na alocação de tarefas de tempo real em sistemas distribuídos. O sistema distribuído de tempo real em questão é formalizado na seção 2 e inclui tarefas periódicas e esporádicas, as quais devem ser alocadas em tempo de projeto (*off-line*) e escalonadas segundo uma política de prioridades fixas.

Embora o tempo de execução do algoritmo de alocação seja obviamente importante, o estudo descrito neste artigo visa caracterizar a taxa de sucesso do algoritmo genético quando usado para alocar aplicações sabidamente viáveis. Neste sentido, é usado uma solução clássica do tipo *branch-and-bound* [PIN95] para identificar a alocação ótima, a qual é usada para avaliar a qualidade da alocação gerada pelo algoritmo genético.

A seção 2 formaliza o problema de alocação de tarefas tempo real atacado neste trabalho. A seção 3 faz uma rápida revisão dos algoritmos genéticos, apresentando os pontos mais importantes deste tipo de algoritmo e mostrando como a técnica de algoritmos genéticos foi aplicada no problema em questão. A seção 4 descreve os resultados obtidos com esta técnica. Por fim, na seção 5 são tecidas algumas conclusões e propostas outras experiências.

2 Descrição do Problema

Neste artigo é suposto que uma aplicação tempo real **A** executa em um sistema distribuído composto por **m** processadores idênticos, interconectados através de uma rede de comunicação no formato de barramento.

A aplicação é definida por um conjunto de **n** tarefas periódicas ou esporádicas. Cada tarefa T_i é caracterizada pelo intervalo mínimo de tempo P_i entre duas ativações sucessivas. No caso de tarefas periódicas, P_i corresponde ao período da tarefa. É suposto que a primeira ativação de todas as tarefas ocorre no instante zero. Não existe relação de precedência ou exclusão mútua entre as tarefas.

Cada tarefa T_i também é caracterizada por um *jitter* de liberação máximo J_i (*maximum release jitter*), um tempo máximo de execução C_i e um *deadline* D_i relativo ao instante de sua ativação. Para todas as tarefas T_i, temos $D_i \leq P_i$. Para que sucessivas execuções de uma mesma tarefa possam compartilhar informações de estado, cada tarefa deve ser alocada a um processador específico e todas as suas ativações deverão executar neste mesmo processador. Uma tarefa não pode migrar durante a execução da aplicação. A função H(i) será usada para denotar o processador onde T_i executa. O escalonamento é preemptivo, ou seja, tarefas podem ser suspensas e retomadas mais tarde.

As tarefas alocadas a cada processador em particular recebem uma prioridade fixa segundo a política *Deadline* Monotônico, isto é, tarefas com *deadline* relativo menor recebem prioridades mais altas [LEU82]. Em tempo de execução, um disparador baseado em prioridades fixas é utilizado para determinar a ordem de execução das tarefas. O símbolo R_i denota o tempo máximo de resposta da tarefa T_i. Considerando o modelo de tarefas em questão, R_i pode ser calculado, por exemplo, através do teste de escalonabilidade apresentado em [AUD93] e [BUR96].

Cada tarefa T_i pode enviar mensagens para outras tarefas da aplicação. O envio de mensagens não afeta a liberação da tarefa destinatária, isto é, não estabelece uma relação de precedência. Entretanto, o envio de mensagens aumenta a carga na rede de comunicação sempre que as tarefas remetente e destinatária forem alocadas a processadores diferentes. Para cada tarefa T_i existe um conjunto de mensagens $M_i = \{M_{ia}, M_{ib}, ...\}$ onde a, b, ... são as tarefas destinatárias. Cada mensagem possui um tamanho fixo conhecido em tempo de projeto. Cada mensagem M_{ij} representa para a rede de comunicação uma carga B_{ij} dada por

$B_{ij} = | M_{ij} | \div P_i$,

onde $|M_{ij}|$ representa o tamanho da mensagem de T_i para T_j e P_i representa o período da tarefa remetente T_i. A carga total **B** na rede de comunicação é dada por:

$$B = \sum_{T_i \in A} \left(\sum_{M_{ij} \in M_i \wedge H(i) \neq H(j)} \left(\left| M_{ij} \right| \div P_i \right) \right).$$

Dada uma aplicação do tipo descrito acima, é necessário determinar em qual processador cada tarefa deve executar de tal forma que todos os *deadlines* sejam satisfeitos ao mesmo tempo em que a carga na rede de comunicação é minimizada. Em outras palavras, determinar os valores da função H(i) que resultem no menor valor possível para **B**, desde que satisfeita a condição $\forall T_i, R_i \leq D_i$.

3 Algoritmos Genéticos

Algoritmos genéticos são heurísticas que procuram imitar o processo de evolução observado na natureza, com o intuito de encontrar soluções próximas do ótimo para problemas de elevada complexidade [DAV91]. Consistem de uma representação das soluções sob a forma de genes, um conjunto de operadores genéticos para gerar novos indivíduos (ou soluções) e uma função de avaliação para determinar quais os indivíduos mais capazes, ou seja, as soluções mais próximas do ótimo. Através de sucessivas gerações, os resultados são selecionados e, de uma forma geral, convergem para a solução ótima. Pode-se, desta forma, dizer que os algoritmos genéticos utilizam a idéia de sobrevivência do mais capaz para selecionar os melhores resultados e os operadores genéticos para explorar novas regiões do conjunto de soluções.

Segundo Hou, Ren e Ansari [HOU92], um algoritmo genético consiste das seguintes etapas:

1. Inicialização: é a geração aleatória da população inicial;
2. Aplicação da Função de Avaliação: determinação dos indivíduos mais capazes (melhores soluções);
3. Operações Genéticas: a partir dos indivíduos mais capazes, é calculada a próxima geração através da aplicação dos operadores genéticos, os quais representam a mutação, a reprodução e o *crossover*;
4. Repetir os passos 2 e 3 até chegar a uma solução convergente.

3.1 Representação das soluções

A representação de uma das soluções do problema de alocação em termos de genes consiste em transformar a distribuição de tarefas entre os processadores em uma seqüência de símbolos que representa a cadeia genética das soluções.

Dependendo da representação escolhida, podem existir seqüências genéticas que não representam soluções válidas. Nestes casos, estas seqüências devem ser eliminadas. Tipicamente, a representação dos genes é feita na forma de *strings* binários, porém é possível utilizar outras representações, como listas, árvores ou outra estrutura de dados que seja julgada conveniente [HOU92].

No problema estudado, a representação de uma solução de escalonamento foi feita através de um vetor de inteiros de *n* posições, onde cada gene *i* assume um valor de 1 a *m*, representando em qual processador a tarefa *i* vai ser executada. A figura 3.1 ilustra uma possível solução para um problema com 4 processadores e 8 tarefas.

1	4	3	2	1	1	2	4

Figura 3.1 – Representação de uma solução.

Esta representação foi escolhida pois com ela é possível construir uma função de mapeamento bijetora entre o universo de soluções de escalonamento e o conjunto de indivíduos da população. Apesar de não gerar nenhuma representação inválida e cobrir todas as soluções, eliminando assim o passo de verificação de cada nova solução gerada na aplicação dos operadores genéticos, a representação pode gerar escalonamentos que não cumpram todos os *deadlines*. Estes deverão ser eliminados ao longo da execução do algoritmo, com a aplicação do mecanismo de seleção.

3.2 Operadores genéticos

A função dos operadores genéticos é criar novas soluções para o problema, possivelmente melhores, baseada na população atual de soluções. Assim, ao combinar duas soluções de qualidade, é aceitável imaginar que a solução gerada será de melhor qualidade, mais próxima da solução ótima.

A representação genética adotada é bastante simples, e isso facilita o desenvolvimento dos operadores genéticos. Foram implementados os três operadores genéticos mais tradicionais: reprodução, mutação e seleção.

A *reprodução* simplesmente copia os indivíduos mais aptos, não havendo nenhuma modificação nos mesmos. Representa a Seleção Natural.

O *cruzamento* sorteia uma posição *p* aleatória e dois indivíduos, i_1 e i_2. A partir destes, gera mais dois indivíduos trocando os genes de i_1 e i_2 a partir da posição sorteada *p*. Após a aplicação da reprodução, a população dobra de tamanho. A figura 3.2 mostra este mecanismo, com dois genes de 8 posições sendo cruzados exatamente no meio. A probabilidade de cruzamento utilizada no algoritmo genético foi de 0,8, valor determinado experimentalmente.

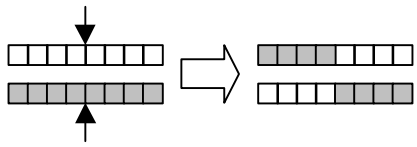

Figura 3.2 – Operação de cruzamento.

A *mutação* é aleatória. São sorteados um indivíduo e um gene, e um novo indivíduo é gerado a partir da troca do valor do gene sorteado, conforme a figura 3.3. A mutação não aumenta a população. A probabilidade de ocorrência de mutação utilizada foi de 0,2.

Figura 3.3 – Mutação de um gene

Por fim, a seleção é feita ordenando-se os indivíduos pelo valor da função de avaliação. Os melhores indivíduos comporão a população na próxima geração. A seleção reduz novamente a população ao seu tamanho normal, após ela ter sido dobrada na etapa de reprodução.

3.3 Função de avaliação

A função de avaliação das soluções é essencialmente a função objetivo que deseja-se minimizar. É ela que controla os mecanismos de reprodução e permite identificar a aptidão de cada indivíduo/solução. Ela é completamente dependente do problema em estudo.

Ponto chave de um algoritmo genético por ser responsável pela avaliação de cada solução gerada, a função de avaliação é baseada no cálculo de B, conforme mostrado na seção 3, e no número de deadlines perdidos, já que o objetivo do escalonamento é encontrar uma solução na qual nenhum deadline é perdido e o uso do barramento é minimizado. As melhores soluções são as que apresentam menor valor na função de avaliação.

De forma a eliminar rapidamente as soluções que perdem deadlines, foi dado um peso a cada um dos parâmetros da função de avaliação, determinados empiricamente por experimentos preliminares. Sendo G a função de avaliação e LD o número de deadlines perdidos, podemos expressar G pela seguinte fórmula:

$$G = 25 * B + 500 * LD$$

Com isso, as soluções que perdem deadlines são eliminadas nas primeiras gerações do algoritmo. Nas últimas iterações do algoritmos, temos praticamente apenas escalonamentos que cumprem os deadlines, e a seleção fina da solução será feita com base no uso do barramento de comunicação.

4 Experiências

Esta seção apresenta os resultados das experiências realizadas com o algoritmo genético descrito na seção anterior. Inicialmente são descritas as condições dos testes. Em seguida são apresentados os resultados obtidos, incluindo variações no perfil dos conjuntos de tarefas considerados. O objetivo das experiências é analisar empiricamente a capacidade do algoritmo descrito neste artigo de encontrar soluções para o problema de alocação considerado.

4.1 Condições das experiências

O algoritmo descrito na seção 3 foi aplicado sobre conjuntos de tarefas (aplicações) com parâmetros aleatórios (período, tempo máximo de computação, deadline, *jitter* de liberação máximo, etc) porém seguindo determinados padrões de carga (ocupação média de processadores), número de tarefas e número de processadores. Cada aplicação gerada foi analisada por um algoritmo ótimo baseado em *branch-and-bound* [PIN95]. Apenas os conjuntos de tarefas escalonáveis foram analisados.

Os testes foram executados com 500 gerações e com um tamanho de população fixo em 3000 indivíduos, visto que experimentos preliminares obtiveram melhores resultados com estes parâmetros.

Os dados de entrada consistiram de um conjunto de problemas de alocação, variando o número de processadores, o número de tarefas a serem escalonadas e a carga do sistema.

Foram executadas 11 baterias de testes, cada uma consistindo de 10 aplicações diferentes de escalonamento. Cada bateria analisa o efeito resultante da variação de um dos parâmetros de entrada: carga média nos processadores, número de processadores e número de tarefas. Cada resultado foi obtido com apenas uma execução, ou seja, caso não fosse encontrada uma solução viável para o problema, o algoritmo genético não era executado novamente. Cada execução do algoritmo genético não demorou mais que alguns segundos em um computador pessoal usando processador Intel Pentium.

4.2 Resultado das experiências

O conjunto de resultados foi dividido em três grandes grupos, variando cada um dos parâmetros: número de processadores, número de tarefas e carga dos processadores.

Para cada conjunto de resultados, são apresentados dois gráficos. O primeiro mostra o total de soluções encontradas pelo algoritmo (objetivo primário), enquanto o segundo mostra a diferença média na ocupação do meio de comunicação das soluções encontradas para a solução ótima (objetivo secundário).

Em termos gerais, foram submetidas 110 aplicações, todas escalonáveis e com solução ótima conhecida. Destas, o algoritmo genético conseguiu encontrar 98, o que representa 89% das aplicações.

Além de ter conseguido encontrar grande parte das soluções, o algoritmo também conseguiu se aproximar bastante das soluções ótimas. Em 26 casos, a solução ótima foi encontrada, e em outros tantos ficou dentro de uma margem aceitável de 15% acima do valor ótimo para ocupação do barramento, que era o parâmetro a ser minimizado.

4.2.1 Variando o número de processadores

Neste primeiro conjunto de resultados, foi variado o número de processadores, enquanto o número de tarefas (20) e a carga (0.8) eram mantidas constantes. Como pode ser observado pelos gráficos, pelo menos 60% das soluções foram encontradas, e na maior parte dos casos chegando a 80% de sucesso no escalonamento (figura 5.1). Uma pequena variação em relação ao ótimo, que no pior caso chegou a 35%, pode ser observada na figura 5.2.

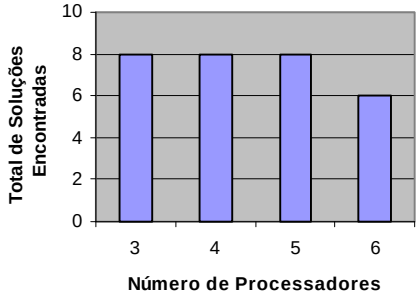

Figura 5.1 – Total de soluções encontradas com variação no número de processadores.

Figura 5.2 – Diferença média da solução encontrada para a solução ótima.

A queda de rendimento do algoritmo no caso de 6 processadores, observada na figura 5.1, pode ser justificada pelo aumento do tamanho das tarefas. Como a carga é constante, um aumento do número de processadores implica em um aumento do tamanho de cada tarefa, tornando-as mais difíceis de serem escalonadas.

4.2.2 Variando o número de tarefas

Nestes resultados, variou-se o número de tarefas, enquanto o número de processadores (4) e a carga (0.8) eram mantidos constantes. Os resultados obtidos foram muito bons, conseguindo-se escalonar grande parte dos problemas. Com 14, 16 e 18 tarefas, todas as aplicações da bateria foram escalonadas, enquanto com 20 e 22 tarefas, 80% delas tiveram sucesso na tentativa de escalonamento (figura 5.3).

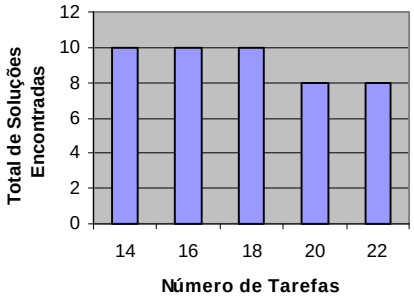

Figura 5.3 – Total de soluções encontradas com variação no número de tarefas.

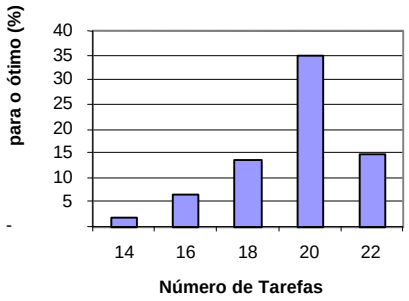

Figura 5.4 – Diferença média da solução encontrada para a solução ótima.

Como pode ser observado na figura 5.4, a variação da solução genética em relação à ótima também foi pequena, com uma variação média máxima de 35%. No caso de 14 tarefas, a variação obtida se aproxima dos 2%, e a solução encontrada pelo algoritmo genético coincidiu com a ótima em grande parte das aplicações.

4.2.3 Variando a carga dos processadores

Neste conjunto de soluções, a carga dos processadores foi variada de 50% a 80%, sendo mantidos fixos o número de processadores (4) e o número de tarefas (20). Aqui, os resultados foram novamente satisfatórios. Com baixa carga, até 70% de ocupação do processador, o algoritmo se comportou muito bem, resolvendo 100% dos problemas (figura 5.5). Com carga de 80%, dois casos não foram resolvidos.

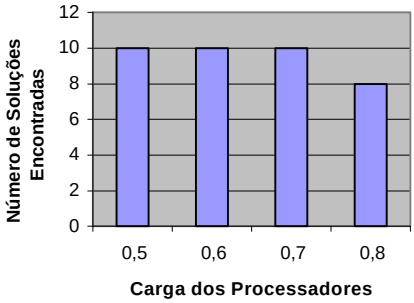

Figura 5.5 – Total de soluções encontradas com variação na carga dos processadores.

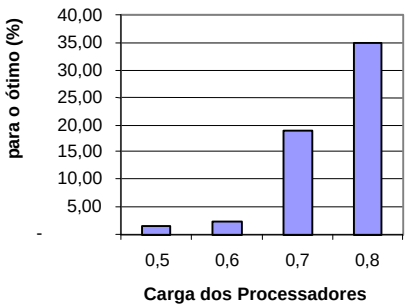

Figura 5.6 – Diferença média da solução encontrada para a solução ótima.

A variação em relação ao ótimo também foi pequena e crescente com o aumento da carga (figura 5.6), um comportamento já esperado, visto que com o aumento da carga há também um aumento no tamanho das tarefas e uma maior dificuldade de encaixá-las nos processadores.

5 Conclusões

Neste artigo, foi apresentada uma aplicação de algoritmos genéticos para a solução de um problema de alocação de tarefas com restrições de tempo real em ambiente distribuído. As experiências mostraram que, embora o algoritmo genético não obtenha sucesso sempre, ele apresenta um comportamento que o torna interessante para este tipo de problema. O algoritmo genético encontrou solução para cerca de 90% das aplicações consideradas. Além disto, em mais de 20% dos casos ele encontrou a solução ótima.

Provavelmente, um aumento no número de gerações ou no tamanho da população tivesse resultado em um maior número de soluções. Entretanto, isto também teria aumentado o tempo de execução. Estes parâmetros foram calibrados para que cada execução não demorasse mais que alguns segundos em um microcomputador de mesa típico.

É importante destacar que as aplicações atacadas pelo algoritmo genético tinham seu tamanho limitado pela solução baseada em *branch-and-bound*. O objetivo das experiências era determinar a taxa de sucesso do algoritmo genético. Logo, era necessário saber se cada aplicação era realmente viável ou não. Para isto foi usada uma solução ótima com tempo de execução elevado. Embora o algoritmo genético seja perfeitamente capaz de atacar grandes instâncias de problema, tais instâncias exigiriam anos de processamento para o algoritmo ótimo.

Além da questão sobre o comportamento do algoritmo em aplicações com muitas tarefas, diversas outras questões permanecem em aberto. Entre elas podemos citar uma análise do tempo de execução em função dos parâmetros do algoritmo genético e o seu efeito sobre os resultados.

Referências

[AUD93] AUDSLEY, N. C.; BURNS, A.; RICHARDSON, M. F.; TINDELL, K.; WELLINGS, A. J. **Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling.** Software Engineering Journal, V. 8, N. 5, 1993. pp.284-292.

[BUR96] BURNS, A.; WELLINGS, A. **Real-Time Systems and Programming Languages**. Addison-Wesley, 1996.

[COR99] CORRÊA, R. C. ; FERREIRA, A. ; REBREYEND, P. **Scheduling Multiprocessor Tasks with Genetic Algorithms**. IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 8, p. 825-837, august 1999.

[DAV91] DAVIS, L. **Handbook of Genetic Algorithms**. New York: Van Nostrand Reinhold, 1991. 385 p.

[HOU92] HOU, E. S. H.; REN, H.; ANSARI, N. Efficient Multiprocessor Scheduling Based on Genetic Algorithms. IN: SOUCEK, B. et al**. Dynamic, Genetic, and Chaotic Programming**. New Jersey: John Wiley & Sons, 1992. p. 339-352.

[LEU82] LEUNG, J. Y. T.; Whitehead, J. **On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks**. Performance Evaluation, 2 (4), p. 237-250, december 1982.

[PIN95] PINEDO, M. **Scheduling.** Englewood Cliffs: Prentice Hall, 1995. 378 p.

[ZOM99] ZOMAYA, A. Y. ; WARD, C. ; MACEY, B. **Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues**. IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 8, p. 795-812, august 1999.