Algoritmos Genéticos para Escalonamento de Processadores

Jan Mendonça Corrêa e Alba Cristina Melo

Departamento de Ciência da Computação, Universidade de Brasília, Campus Universitário, Asa Norte, Brasília, Brasil
{jan,albamm}@cic.unb.br

Abstract—

In its generic formulation, processor scheduling is a NP-Complete problem. Many different approaches have been proposed to the scheduling problem and one of them is the use of Genetic Algorithms (GA). These algorithms work using an analogy with living beings, mimicking their ability to adapt to the environment. The aim of this paper is to present a tentative of classification of GA-based processor schedulers and describe some of the results achieved and some aspects of GA that can interfere with the quality of the results.

Keywords— **Processor scheduling, genetic algorithms**

I. INTRODUÇÃO

O problema do escalonamento de processadores consiste em, dado um conjunto **p** de processos e um conjunto **P** de processadores, determinar como será feita a divisão dos elementos de **p** entre os elementos de **P**, levando-se em conta as restrições de precedência, procurando otimizar a divisão de acordo com algum critério.

Alguns métodos propostos para o escalonamento de processadores tem mostrado que seu desempenho pode ter uma grande dependência do número de processadores, dos seus parâmetros internos e dos processos que irão executar sobre eles [Kre92], [Sev94], [Zho86]. Uma possível explicação para isto é que o problema do agendamento[1] de processadores no caso genérico é um problema NP-Completo [Gar79]. Sendo assim, não é possível fazer uma heurística que dê sempre resultados igualmente bons para qualquer espaço de soluções do problema [Wol96]. Para se tentar evitar a dependência destes parâmetros no espaço de solução pode-se utilizar uma heurística que procure se adaptar às especificidades do problema. Uma forma de se fazer isto é através dos Algoritmos Genéticos [Gol89]. Os algoritmos genéticos são uma meta-heurística onde possíveis soluções para um dado problema são codificadas na forma de indivíduos que interagem com outros na mesma população, procurando através das operações genéticas (reprodução, mutação, entre outros) gerar indivíduos mais aptos, que são codificações de soluções melhores.

Vários trabalhos publicados até o momento [Gra99], [Bau95] e [Sun97] apontam vantagens na utilização dos algoritmos genéticos para o escalonamento de processadores. Até o presente momento várias abordagens foram usadas para o problema de escalonamento utilizando os AGs, mas infelizmente é difícil fazer uma comparação sobre quais tipos de abordagem são mais eficientes. O principais motivos para isto são os objetivos de otimização diferentes. Estes algoritmos genéticos com diferentes abordagens possuem modelagens diferentes como a representação direta e indireta. Alguns utilizam a paralelização dos algoritmos genéticos e outros não. Além das diferenças de implementação também existe a escolha de diferentes conjuntos de processos e parâmetros que são avaliados nos testes de desempenho dos AGs. Apesar disto foram encontrados resultados positivos com a utilização dos algoritmos genéticos.

O objetivo deste trabalho é mostrar alguns dos resultados obtidos até o momento para o escalonamento de processadores utilizando algoritmos genéticos e algumas de suas características importantes. Baseado nestas características propomos e discutimos uma classificação para escalonadores de processadores baseados em algoritmos genéticos.

O resto deste artigo está dividido da seguinte forma. A seção II descreve a representação do problema. A seção III descreve como os AGs são executados. A seção IV mostra a classificação dos AGs para escalonamento. A seção V descreve os resultados obtidos e, finalmente, a seção VI apresenta as conclusões.

II. ESCALONAMENTO DE PROCESSADORES

Para se fazer um escalonamento de processadores escolhendo quais processos irão executar em quais processadores é necessário, primeiramente, obter informações sobre os processos a serem executados no sistema através de uma representação do relacionamento entre os processos. Uma das formas mais comuns de se fazer esta representação é através de um GDA (grafo direcionado acíclico) [Kwo97], [Gra99], [Bau95], [Yan91], [Kal98]. Neste tipo de representação temos o grafo G = (V, A) onde V é o conjunto dos vértices direcionados do grafo e A é o seu conjunto de arestas. Os vértices do grafo representam os

[1] Neste trabalho, utilizaremos indistintamente os termos agendamento e escalonamento, sendo que eles se referem ao mesmo problema.

processos que irão ser executados, sendo que cada um deles tem um peso que representará o custo computacional do processo. As arestas do grafo representam os custos das comunicações que ocorrem entre os processos e as relações de precedência entre os processos. Os custos podem ser estimados antes da execução no caso do agendamento estático, ou medidos diretamente no sistema no caso dinâmico.

A figura 1 exemplifica um grafo de processos (GDA).

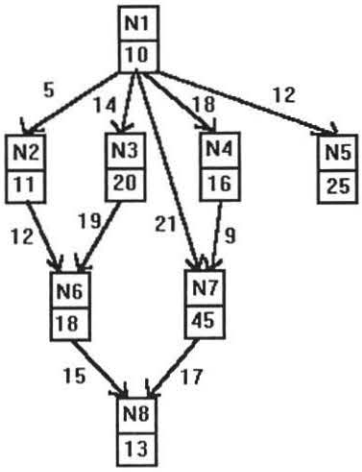

Figura 1 Grafo Direcionado Acíclico com 8 processos

Na figura 1 temos 8 processos, cada um com seu tempo que é dado na parte de baixo do vértice. As arestas indicam o tempo de comunicação entre os processos que é dado pelo peso da aresta se os processos estiverem em processadores diferentes. Este custo de comunicação é considerado zero se os processos estiverem no mesmo processador. Temos que o vértice que recebe uma aresta é chamado de vértice filho e aquele vértice de onde a aresta provém é chamado de pai. Na figura 1 N2 e N3 são pais de N6. Um vértice que não tem pai é chamado de vértice de entrada como por exemplo o N1. O vértice que apenas não tem filhos é chamado vértice de saída com por exemplo o N8. De acordo com a ordem de precedência dada no grafo, nenhum vértice (processo) pode ser iniciado até que tenha recebido todas as sua arestas (comunicações) de seus pais. O GDA é geralmente utilizado em problemas de escalonamento onde não existe comunicação entre vértices sem relação pai-filho. Nos casos em que os vértices se comunicam sem a relação pai-filho pode se utilizar um grafo cíclico como em [San97].

III. Algoritmos Genéticos

Um algoritmo genético consiste basicamente de um laço de repetição dentro do qual são executadas as operações genéticas até que um determinado critério de parada, como o número máximo de gerações, seja atingido [Gol89] e [Mic96].

Um exemplo de um algoritmo genético simples seria o seguinte.

```
Algoritmo Genético

Inicializa População
Enquanto o critério de parada não for atingido faz
  Para n =1 até Parte_População faz :
    Seleciona randomicamente 2 indivíduos para
    crossing-over
    Seleciona randomicamente 1 indivíduos para
    mutação
  Fim do Para
  Avalia indivíduos e faz seleção
Fim do Enquanto
Escolhe melhor indivíduo como solução
```

A seguir descreveremos as principais operações do AG.

O primeiro passo na execução de um AG consiste em determinar como a população inicial será gerada. Ela pode ser inicializada com indivíduos gerados aleatoriamente ou de acordo com alguma heurística.

O próximo passo no AG é decidir qual parte da população irá sofrer a atuação dos operadores genéticos. Cada tipo de representação escolhida de indivíduo tem um tipo de operação genética mais adequada [Jon75]. Uma das operações genéticas é o *crossing-over*, cujo objetivo é gerar novos indivíduos a partir de dois indivíduos já existentes na população. Alguns métodos de *crossing-over* amplamente divulgados na literatura [Gol89], [Mic96] são os *crossing-over* de ordem, parcialmente mapeado (*partially matched*) PMX e o cíclico.

Uma outra operação genética é a mutação. O objetivo de mutação é proporcionar novos genes para a população. Assim como os outros operadores, a mutação deve procurar, na medida do possível, gerar resultados válidos.

A seguir é feita a avaliação dos indivíduos da população, que no caso do agendamento de processadores, corresponde a escolher os indivíduos que são os melhores agendamentos de acordo com os critérios escolhidos. Para se fazer a avaliação das soluções é necessária a escolha de uma função de avaliação. Existem várias maneiras possíveis de se avaliar um agendamento de processadores. Por exemplo minimizar o tempo de início [Kwo97], minimizar o tempo execução total [Sun97], melhorar o balanceamento de carga [Bau95] e minimizar o tempo de término de execução de cada processo [Gra99].

Existem vários parâmetros que tem influência na velocidade de convergência e qualidade das soluções encontradas pelos algoritmos genéticos. Alguns destes parâmetros são a taxa de mutação, a taxa de *crossing-over*, tamanho da população e taxa de renovação da população. Encontrar valores ótimos para esta taxas é uma tarefa complexa [Gol89]. Uma saída para este problema é tentar fazer com que as taxas se adaptem ao problema específico [Sri94] e [Spe95]. Esta abordagem é utilizada em [Kwo97] para resolver o problema da convergência prematura para o escalonamento de processadores.

IV. Classificação dos Algoritmos Genéticos para Escalonamento de Processadores

Uma possível classificação para os algoritmos genéticos para escalonamento de processadores quanto a paralelização [Can99] é a apresentada na figura 2.

Figura 2 Subdivisões dos algoritmos genéticos para escalonamento de processadores

A. *AGs Seqüenciais e Paralelos*

Os AGs seqüenciais são aqueles que executam sobre uma única população em um único processador. Os algoritmos genéticos são considerados facilmente paralelizáveis [Can99], [Gre91] desta forma se torna útil paralelizar os AGs que fazem o escalonamento de processadores [Kwo97]. Existem várias formas de se paralelizar os algoritmos genéticos [Can99]. Uma maneira simples de paralelização é o modelo de população distribuída onde existe apenas uma única população e esta está dividida entre vários processadores que se comunicam. Uma outra forma de paralelização seria a criação de várias populações que trocam indivíduos entre si com uma nova operação genética chamada migração. Um exemplo comum deste tipo de paralelização é o modelo ilha. Neste modelo existem vários parâmetros que influenciam a velocidade de obtenção e qualidade da solução obtida [Can99] e [Tan89]. Desta forma encontrar parâmetros que maximizem o desempenho se constitui um problema a parte.

Independente da forma de distribuição do algoritmo de escalonamento com AG, existem várias formas destes serem implementados. Uma possível classificação destas implementações para o escalonamento de processadores é a dada na figura 3.

Figura 3 Formas de implementação dos Algoritmos genéticos para escalonamento de processadores.

B. *Otimizador de Escalonamentos*

O que caracteriza este tipo de escalonamento utilizando AG é que cada um dos indivíduos representa a codificação de uma solução para o escalonamento. Esta solução geralmente está baseada no agendamento em lista. Nesta representação os indivíduos são seqüências de vértices (processos) representados em forma de lista [Ada74], [Yan93]. A validade ou não da ordem dos vértices na lista é dada pelo GDA que diz quais processos podem ser colocados em execução. Existem várias ordens possíveis de execução obedecendo as restrições do GDA. O número de possibilidades possíveis aumenta exponencialmente com o número de processos e a escolha da melhor ordem é geralmente um problema NP-Completo [Gar79].

Existem várias formas de se definir qual será a ordem dos vértices no agendamento em lista [Kwo97].Uma das formas mais comuns é atribuir prioridades para cada vértice de forma que eles possam ser ordenados obedecendo a um determinado critério de prioridade. Alguns critérios comuns para atribuir prioridades são o T-Level (*top level*) e B-Level (*botton level*). Para um determinado vértice v1, o seu T-Level é calculado como sendo o maior caminho desde o vértice de entrada até o v1, excluindo-se o v1. O B-Level de um vértice v1 é calculado como sendo o maior caminho desde v1 até um vértice de saída. Estes caminhos são geralmente calculados levando em consideração os custos de tempo de execução de cada processo e seus custos de comunicação. Um outro atributo comum para se calcular a prioridade é o ALAP (*As Late As Possible*) para o tempo de início da execução dos processos. Neste caso se calcula qual o maior tempo possível que um processo pode demorar a executar sem atrasar a execução dos processos que são dependentes dele pela comunicação.

Quanto à representação dos escalonamentos pelos AGs classificados como Otimizadores de Escalonamentos existem dois tipos de representação [Reb98] : representação direta e indireta.

B.1. Representação Direta

Na representação direta, os agendamentos de processadores estão diretamente representados nos genes dos indivíduos. Na implementação de [Kwo97], [Sch94], [Dus98] e [Alb94], após a obtenção da lista ordenada de vértices é necessário fazer um mapeamento destes vértices (processos) para os processadores. Um critério comum para fazer este mapeamento é o critério de menor tempo de início. Por este critério cada processo é atribuído ao processador que permita que ele possa começar o mais cedo possível, geralmente levando em conta os tempos de comunicação. De acordo com [Kwo97] este método de mapeamento não garante que se encontre sempre uma solução ótima uma vez que, sabendo de antemão qual é a solução ótima, nem sempre é possível encontrar uma lista de vértices que resulte nela através da minimização do tempo de início.

Uma outra forma de representação direta seria por exemplo a utilizada por [Sun97]. Nesta representação os

indivíduos são várias listas cada uma correspondendo aos processos que serão atribuídos a cada processador, observando-se as restrições de precedência.

B.2. Representação Indireta

Na representação indireta, os indivíduos codificados nos algoritmos genéticos são decisões a respeito de como serão feitos os escalonamentos. Algumas abordagens [Gra99] e [San97] para escalonamento em lista de processadores com algoritmos genéticos se utilizam da representação indireta. Grajcar [Gra99], por exemplo, representa indivíduos como vetores de prioridades para os processos e para os recursos (processadores). Os processos devem ser atribuídos aos processadores de acordo com as suas prioridades, obedecendo a ordem de precedência no GDA (grafo direcionado acíclico). Os processadores tem prioridades onde os de maior prioridade oferecem o menor tempo de execução para o processo. Estas prioridades são calculadas dinamicamente a cada iteração e cada indivíduo é avaliado levando em conta o tempo que demora para executar toda a árvore de processos (*makespan*).

C. *Sistema Classificador*

Uma outra abordagem para o balanceamento de carga utilizando algoritmos genéticos seria o uso de um sistema classificador (*classifier system*) [Gol89] e [Boe94]. Este sistema classificador é composto de várias partes :

1 Condições de entrada
2 Ações de saída
3 Uma base de regras
4 Mecanismo de mapeamento
5 Mecanismo de seleção
6 Sistema de recompensa
7 Sistema de descobrimento de regras

O sistema classificador é composto de regras. Comportamentos complexos exigem que mais de uma regra atue em conjunto de forma a obter o resultado desejado. O relacionamento entre as regras é ditado pelo sistema de recompensas que utiliza os dados do ambiente para avaliar através da seleção quais são as boas soluções. Para encontrar melhores soluções são realizadas operações genéticas de mutação e *crossing-over* sobre as regras existentes.

Baumgartner em [Bau95] implementou o algoritmo de balanceamento de carga CBLB (*Classifier Based Load Balancer*) onde um sistema classificador utiliza conjuntos de regras para obter melhores decisões de balanceamento.

D. *Otimizador de Parâmetros*

Nesta abordagem cada conjunto de parâmetros de um escalonador já existente representa um indivíduo. Desta forma, o AG procura obter melhores conjuntos de parâmetros para o escalonador através das operações genéticas. Não é do nosso conhecimento a existência de AGs deste tipo, apesar dos AGs serem utilizados em problemas complexos como por exemplo otimização de redes neurais [Dod90].

V. RESULTADOS

Kwok em [Kwo97] implementou um algoritmo genético paralelo de várias populações em uma configuração do tipo ilha em uma máquina Intel Paragon. Para fazer os seus testes ele introduziu uma razão entre o tempo gasto com comunicações e o tempo gasto em computação (CCR *Comunication to Computation Ratio*). Com isto ele verificou que quanto maior a quantidade de comunicação realizada pelos processos em relação a sua quantidade de computação, menor será a probabilidade de se encontrar um solução ótima. Foi observado também que um aumento do número máximo de gerações que o algoritmo genético executa tem um impacto significativamente maior na qualidade das soluções obtidas do que um aumento do número de indivíduos da população. Utilizando o escalonador com algoritmo genético obteve-se resultados significativamente melhores do que com o método DCP (Dynamic Critical-Path) [Kwo96] para executar algoritmos distribuídos de eliminação de Gauss, fatoração LU e equação de Laplace.

Em [Alb94] foi proposto um AG seqüencial de representação direta, sendo observado que a alocação de tarefas para vários processadores compensou o *overhead* gasto para esta divisão. Mais ainda, os pesos dados para as comunicações entre processos e o desbalanceamento de carga tem grande influência para a obtenção de bons resultados pelo AG.

Grajcar [Gra99] implementou um AG seqüencial de representação indireta. Utilizando uma Sparc 20 ele fez comparações medindo quanto tempo era necessário para que seu escalonador encontrasse soluções tão boas quanto as encontradas por [Dho95] e [Pra92]. Verificou-se que seu algoritmo era significativamente mais rápido. A seguir foi feita uma comparação do tempo para encontrar a solução para várias instâncias do problema de escalonamento de [Ben96]. Grajcar em [Gra99] verificou que o tempo gasto pelo seu algoritmo foi menor para quase todas as instâncias sendo que em alguns casos chegava a ser dezenas de vezes menor.

Sandnes [San97] obteve com seu AG seqüencial de representação indireta para reestruturação de grafos cíclicos de processos resultados bem melhores que o método *Critical-path / Immediate Successor First* [Hir85].

Baumgartner [Bau95] implementou um AG para balanceamento de carga CBLB (*Classifier Based Load Balancer*) utilizando um sistema classificador. O CBLB foi comparado com os métodos Centex e No Balance em uma máquina Connection Machine 5 e Intel Hypercube. Foi verificado que o método Centex tem um desempenho relativamente constante enquanto que o CBLB consegue adaptar os seus parâmetros, melhorando com o passar do tempo e conseguindo resultados bem melhores que o Centex.

Sung-Ho [Sun97] implementou seu escalonador utilizando AG seqüencial de representação direta em uma

Sparc workstation obteve resultados para escalonamentos em sistemas distribuídos homogêneos e heterogêneos. No ambiente homogêneo o escalonador com AG foi comparado com um algoritmo de escalonamento em lista para algoritmos distribuídos de eliminação de Gauss e transformada rápida de Fourier. O algoritmo de Sung-Ho teve uma performance melhor em todos os casos. No ambiente heterogêneo o AG foi comparado com o escalonador em lista OLROG sendo que o AG teve uma performance significativamente superior.

A seguir temos a tabela 1 que faz resumo comparativo das características de cada implementação levando em consideração a classificação dada na seção IV :

Implementação	Pop. Distri.	Várias Pop.	Seqüencial	Otim. Para.	Sist. Class.	Otim. Esc.
[Alb94]	Não	Não	Sim	Não	Não	Rep. Dir.
[Bau95]	Não	Não	Sim	Não	Sim	Não
[Dus98]	Não	Não	Sim	Não	Não	Rep. Dir.
[Gra99]	Não	Não	Sim	Não	Não	Rep. Ind.
[Kwo97]	Não	Sim	Não	Não	Não	Rep. Dir.
[San97]	Não	Não	Sim	Não	Não	Rep. Ind.
[Sch94]	Não	Sim	Não	Não	Não	Rep. Dir.
[Sun97]	Não	Não	Sim	Não	Não	Rep. Dir.

Tabela 1 Quadro comparativo das implementações

Observando a tabela 1 podemos verificar que embora a utilização de uma população distribuída seja uma abordagem comum em AGs [Can99] não foi utilizada por nenhum dos artigos avaliados. Uma possível explicação para isto é que a utilização de várias populações geralmente oferece melhores resultados. Podemos ver também que comparativamente poucos artigos utilizam a paralelização dos AGs, embora a paralelização possibilite um ganho de desempenho. Os artigos listados na tabela 1 têm implementações diferentes e foram utilizadas em instâncias diferentes, sendo difícil indicar qual a melhor abordagem.

VI. Conclusão

Existem várias abordagens para o problema do escalonamento de processadores utilizando AGs. Estas abordagens geralmente utilizam algum método de escalonamento já existente como por exemplo o escalonamento em lista e procuram otimizá-lo através das operações genéticas. A comparação entre estas abordagens é bastante difícil uma vez que as instâncias e as representações dos problemas utilizadas são diferentes. Cada tipo de representação implica em tipos diferentes de operações genéticas que serão efetuadas sobre os indivíduos. Um dos grandes problemas em relação a estas operações é o aparecimento de indivíduos inválidos, que podem alterar a qualidade e velocidade da convergência do AGs. Em vários artigos pode-se ver que, em geral, os algoritmos genéticos para escalonamento de processadores apresentam resultados satisfatórios quando comparados com outros algoritmos de escalonamento conhecidos. Além de apresentar diversos algoritmos para escalonamento de processos, o presente artigo propôs e discutiu uma classificação de tais AGs. Nós consideramos que a utilização de uma classificação única é fundamental tanto para o melhor entendimento de cada AG como para a comparação entre eles. Ainda existem vários problemas a serem solucionados através da pesquisa em algoritmos genéticos para escalonamento de processadores tais como identificar qual é a representação mais adequada nos AGs, quais são os melhores parâmetros, qual deve ser a função de avaliação das soluções e como estes fatores se relacionam com problemas específicos de escalonamento.

Referências

[Ada74] ADAM, Thomas; CHANDY, K.M.; DICKSON, J. R. *A comparison of list schedules for parallel processing Systems*, CACM, 17:12 (1974), 685-690.

[Alb94] ALBA, Enrique; ALDANA, J.F.; TROYA, José .M. Load Balancing and Query Optimization in Dataflow Parallel Evaluation of Datalog Programs, *Proceedings of the 1994 International Conference on Parallel and Distributed Systems, Lionel M. Ni (ed.)*, IEEE Computer Society Press, pp. 682-688, 1994

[Bau94] BAUMGARTNER, Joey; COOK, Diane, J. A. genetic algorithm for load balancing in parallel computers. In *Proceedings of the Seventh International Confernece on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 619--628, 1994.

[Bau95] BAUMGARTNER, Joey; COOK, Diane, J. A.; SHIRAZI, Behrooz, Genetic solutions to the load balancing problem. *Proc of ICPP 95 Workshop*, pp 72-78.

[Boe94] BOER, Bart de, *Classifier System : A Useful Approach to Machine Learning ?*, Master's Thesis, Leiden University, 1994

[Can99] CANTÚ-PAZ, Erick *Designing Efficient and Accurate Parallel Genetic Algorithms* IlliGal Report No. 99017 July 1999 Illinois Genetic Algorithms Laboratory

[Cor99] CORRÊA, Ricardo C.; FERREIRA, Afonso; REBREYEND, Pascal Scheduling Multipocessors Tasks with Genetic Algorithms, *IEEE Transaction On Parallel and Distribute Systems*, Vol 10, No.8, pp 825-837 August 1999

[Dho95] DHODDI, Muhammad K.; AHMAD, Imtiaz; STORER, Robert. Shemus: synthesis of heterogeneous multiprocessor systems. *Microprocessors and Mircosystems*, 19(6):pp 311--319, 1995

[Dod90] DODD, N. Optimisation of network structure using genetic techniques. In *Proceedings of the International Joint Conference on Neural Networks*, pages 965--970, 1990

[Dus98] DUSSA-ZIEGER, Klaudia; SCHWEHM, Markus Scheduling of Parallel Programs on Configurable Multiprocessors by Genetic Algorithms, *Journal of Approximate Reasoning*, Special Issue on 'Approximative Methods in Scheduling', Vol 19 (1-2) 23-38, 1998

[Gar79] GAREY, Michael R.; JOHNSON, David S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

[Gol89] GOLDBERG, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. AddisonWesley Publishing Company, 1989.

[Gra99] GRAJCAR, Martin Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System; *Proceedings of the 36th Design Automation Conference (DAC)*, p. 280-285, New Orleans, 1999, ISBN 0-7803-5560-1

[Gre91] GREFENSTETTE, John J. Conditions for implicit parallelism. In *Foundations of Genetic Algorithms*, G. J. E. Rawlins (ed.), Bloomington, IN: Morgan Kaufmann. (1991a)

[Hir85] HIRONORI Kasahara; SEINOSUKE Narita. Parallel processing of robot-arm control computation ona multiprocessor system. *IEEE Journal of Robotics and Automation*, RA-1(2):pp104--113, 1985.

[Jon89] DE JONG, Kenneth A.; SPEARS, Willian M. Using genetic algorithms to solve NP-complete problems, Proc. 3rd Int'nl Conf. Genet. Algorithms, Morgan Kaufmann,1989, 124-132.

[Kre92] KREMIEN, O.; KRAMER, J. Methodical analysis of adaptive load sharing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):747--760, November 1992.

[Kwo97] KWOK, Yu-Kwong; AHMAD, Ishfaq, Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using A Parallel Genetic Algorithm, *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 58-77, November 1997.

[Kwo96] KWOK, Yu-Kwong, AHMAD, Ishfaq, Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors, *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.

[Mac87] MACDOUGALL, M. H. *Simulating Computer Systems : Techniques and Tools*. The MIT Press, 1987

[Mic96] MICHALEWICZ, Zbigniew, *Genetic Algorithms+Data Structures=Evolution Programs*, 3rd edition. Springer, 1996.

[Pra92] PRAKASH, S.; PARKER, A. C. SOS: synthesis of application-specific heterogeneous multiprocessor systems, *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 38--51, 1992

[Reb98] REBREYEND, P. SANDNES, F. E.; MEGSON, G. M. *Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations*, Laboratoire de l'Informatique du Parallelisme ' Ecole Normale Sup'erieure de Lyon, Research Report N. 98-25, 1998

[San97] SANDNES, F.E.; MEGSON, G.M. Improved Static Multiprocessor Scheduling Using Cyclic Task Graphs - A Genetic Approach. *IPPS'97 Workshop on Randomised Parallel Computing*, 1997.

[Sch94] SCHWEHM, Markus; WALTER, Thomas (1994): Mapping and Scheduling by Genetic Algorithms, In: Buchberger, B. and Volkert, J. (Eds.): *Parallel Processing: CONPAR 94 - VAPP VI'Third Joint Int. Conf. Vector and Parallel Processing in Linz*, Austria, September 1994,Springer Verlag, LNCS 854, pp. 832-841

[Spe95] SPEARS, Willian M. Adapting Crossover in Evolutionary Algorithms, in *Proceedings of 4th Annual Conf. on Evolutionary Computing*, ed Fogel, 1995, MIT Press.

[Sri94] SRINIVAS, M.; PATNAIK, L.M. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms, *IEEE Trans. Sys., Man and Cybernetics*, vol. 24, no. 4, Apr. 1994, pp. 656-667.

[Sun97] SUNG-HO, Woo; SUNG-BONG Yang; SHIN-DUG, Kim; Tack-Don Han, Task scheduling in distributed computing systems with a genetic algorithm , *Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia '97*, 1997

[Tan89] TANESE, Reiko. *Distributed Genetic Algorithm for Function Optimization*. PhD. dissertation. Department of Electrical Engineering and Computer Science. University of Michigan, 1989

[Wol96] WOLPERT, David H.; MACREADY, Willian G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, April 1996.

[Yan91] YANG, Tao; GERASOULIS, Apostolos, "A Fast Static Scheduling Algorithm for DAGs on an Unbounded Number of Processors", *Proceedings of Supercomputing* 91,pages 633-642, Albuquerque, 1991.

[Zho86] ZHOU, Songnian. *A trace-driven simulation study of dynamic load balancing*. Technical Report UCB/CSD 87/305, University of California, Berkely, 1986.