ESCALONAMENTO DE TAREFAS EM MULTIPROCESSADORES BASEADO EM ALGORITMOS GENÉTICOS

JAQUELINE A. J. PAPINI, HELDER R. G. LINHARES, GINA M. B. DE OLIVEIRA

Faculdade de Ciência da Computação, Universidade Federal de Uberlândia
Av. João Naves de Ávila, 2121- Campus Santa Mônica, Bloco B, sala 1B60
E-mail: jaque_comp@yahoo.com.br, heldercomp@yahoo.com.br, gina@facom.ufu.br

Resumo. Este artigo apresenta a aplicação de algoritmos genéticos (AGs) na obtenção de um escalonamento ótimo de tarefas entre dois processadores idênticos em uma arquitetura multiprocessadora. O AG é uma técnica de busca não-determinista inspirada na teoria evolutiva de Darwin. O ambiente evolutivo foi aplicado inicialmente a um problema conhecido na literatura por Gauss 18, representado por um grafo de programa composto por dezoito tarefas, no qual o AG foi capaz de encontrar o escalonamento ótimo. Esse problema foi analisado intensivamente para ajuste dos parâmetros do AG. Posteriormente, dez grafos de programa foram elaborados de forma aleatória, compostos de 11 tarefas, com características estruturais bastante variadas. Para se conhecer o escalonamento ótimo de cada grafo, um algoritmo exato foi aplicado aos mesmos. Posteriormente, o AG foi aplicado nos dez grafos e foram encontrados bons resultados de convergência para o ótimo, com um baixo tempo de processamento.

Palavras-chave— Algoritmo Genético, Escalonamento de Tarefas.

1 Introdução

Nos últimos anos, a principal solução proposta para o aumento de desempenho dos computadores tem sido o paralelismo. Diferentes máquinas paralelas baseadas em novas arquiteturas têm surgido. No entanto, em relação às máquinas seqüenciais, ainda há uma enorme dificuldade de programação e gerenciamento destas máquinas paralelas.

O problema de escalonamento estudado nesse trabalho consiste na alocação de diversas tarefas que compõem um programa paralelo e que devem ser executadas em dois processadores idênticos, em uma estrutura de *hardware* multiprocessadora. O problema de decisão vinculado à busca do escalonamento ótimo é sabido ser NP-completo [1].

A maioria dos algoritmos de escalonamento conhecidos são exatos e seqüenciais. Para descobrirmos o escalonamento ótimo de grafos desconhecidos, utilizamos o método de busca exata conhecido por *Branch and Bound*. Existem vários motivos que levam à procura de alternativas aos algoritmos exatos: eficiência, desempenho, rápida adaptação de soluções a novas circunstâncias, dentre outros.

Os algoritmos genéticos [2] realizam uma busca não determinista baseada na abstração do conceito de evolução e são especialmente úteis na solução de problemas em que não se conheça um algoritmo convencional eficaz. No escalonamento de tarefas, o AG se mostrou uma alternativa válida, uma vez que apresenta soluções ótimas ou sub-ótimas com um baixo custo de processamento.

Na próxima seção, o problema do escalonamento é apresentado, bem como os grafos de programa utilizados nesse trabalho. A seção 3 apresenta o ambiente evolutivo elaborado para esse problema, apresentando detalhes como o tipo de *crossover* e mutação. Em seguida os resultados obtidos são apresentados. Primeiramente, são discutidos os resultados da aplicação do AG em um escalonamento de um grafo muito conhecido na literatura, o Gauss 18. Os experimentos conduzidos com o Gauss 18 permitiram um ajuste dos parâmetros do AG. Uma vez ajustado, o AG foi aplicado a dez grafos de programa gerados aleatoriamente, mas com características estruturais bastante diversificadas. Os resultados mostram que o AG elaborado possui uma boa convergência no escalonamento desses grafos.

2 Escalonamento de tarefas

O problema de decisão vinculado ao escalonamento de tarefas que compõem um programa paralelo em uma arquitetura multiprocessadora é NP-completo [1]. Por isso, constitui-se em um desafio para vários pesquisadores na área de computação paralela.

Um programa paralelo pode ser representado por um grafo acíclico, direcionado e com pesos $G = (V, E)$ conhecido como grafo de precedência das tarefas ou grafo de programa. V é o conjunto de N nós que representam as N tarefas do programa paralelo. Para simplificação, é assumido que cada tarefa é uma unidade computacionalmente indivisível. Há uma relação de restrição de precedência entre as tarefas k e l em um grafo de programa se o resultado produzido pela tarefa k deve ser enviado para a tarefa l, antes que a última possa ser executada. Um grafo de programa tem dois atributos: os pesos b_k e a_{kl}. O peso b_k do nó k representa o custo computacional deste nó, ou seja, o tempo necessário para execução da tarefa k em um dado processador de um sistema multiprocessador. E é o conjunto de arestas do grafo de programa que descreve os padrões de comunicação entre as tarefas. O peso a_{kl} da aresta (k, l) representa o custo (tempo) de comunicação entre os pares de tarefas k e l quando as mesmas estão alocadas em processadores vizinhos. Se as tarefas k e l estão alocadas no mesmo processador, o custo de comunicação é considerado nulo.

O propósito do escalonamento é distribuir as tarefas entre os processadores de maneira que as restrições de precedência sejam preservadas, e o tempo total de execução T seja minimizado.

A Figura 1 apresenta o grafo de precedência de tarefas conhecido como Gauss 18 [4]. Ele representa o algoritmo de eliminação Gaussiana consistindo de 18 tarefas e foi extensivamente investigado na literatura [4]. Sabe-se que o escalonamento ótimo desse grafo em uma arquitetura composta por dois processadores idênticos retorna um tempo mínimo de 44 unidades de tempo.

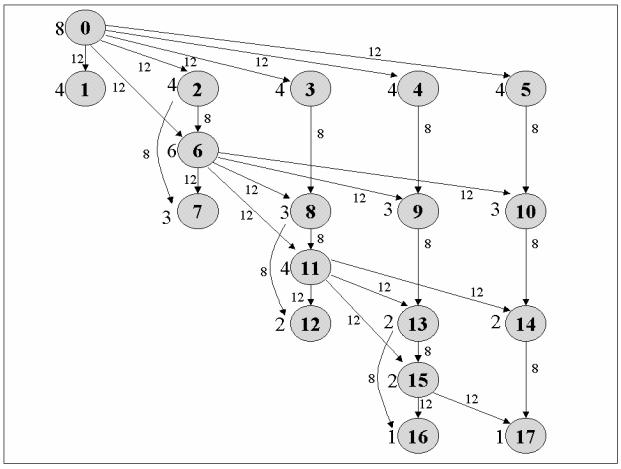

Figura 1: Grafo Gauss 18

Nos resultados que apresentaremos na seção 4, além do Gauss 18, utilizaremos dez grafos que foram gerados aleatoriamente, cada um composto por 11 tarefas. Dois desses grafos são ilustrados nas figuras 2 e 3, chamados de P11C e P11D. Como é possível perceber pelas figuras, os grafos foram geradas de forma que houvesse uma grande variação entre eles, indo desde grafos com uma estrutura mais regular como o P11C, até estruturas mais irregulares como o P11D.

Podemos representar a estrutura dos grafos de programa, através da notação usual de grafos e de seus conjuntos de vértices (V) e arestas (E), sendo que cada elemento (k,b_k) de V representa a tarefa k e seu custo computacional b_k e cada elemento (k,l,a_{kl}) de E representa que a tarefa k precede a tarefa l e o custo de comunicação entre elas é a_{kl}. Por exemplo, o Gauss18 é representado por $G=(V,E)$, sendo V={(0,8), (1,4),..., (17,1)} e E={(0,1,12), (0,2,12),..., (15,17,12)}. A Tabela 1 apresenta os conjuntos de vértices (V) e arestas (E) dos 10 grafos de programa compostos de 11 tarefas, utilizados nesse trabalho.

Para conhecermos o tempo do escalonamento ótimo de cada um dos grafos da Tabela 1, um algoritmo *branch-and-bound* (B&B) foi implementado para o problema do escalonamento. O algoritmo B&B é baseado na idéia de enumerar todas as soluções *praticáveis* de um problema de forma direcionada e inteligente [5]. O B&B tem sido utilizado para encontrar soluções ótimas para o problema do escalonamento de tarefas em sistemas multiprocessadores [3]. Ele é um algoritmo de busca enumerativa não exaustiva no espaço de soluções para o problema de escalonamento. Este espaço de soluções é freqüentemente representado por uma estrutura de árvore, onde cada vértice da árvore representa uma solução completa ou parcial para o problema. Utilizando-se regras inteligentes para a seleção de vértices a explorar/expandir e também para a exclusão de vértices (poda) que não conduzam a uma solução ótima, a complexidade da busca pode ser reduzida quando comparada a métodos de busca enumerativa exaustiva. O resultado da aplicação do algoritmo B&B na obtenção do escalonamento ótimo (tempo mínimo) para todos os 10 grafos representados na Tabela 1 em uma arquitetura composta por dois processadores idênticos é apresentado na Tabela 4.

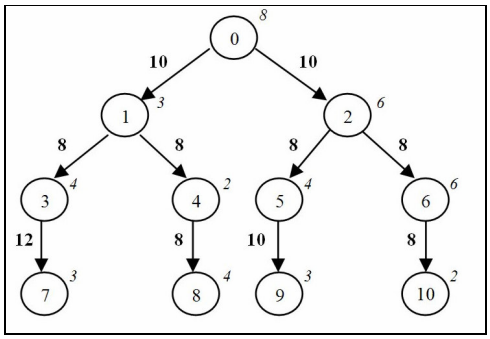

Figura 2: Grafo P11C

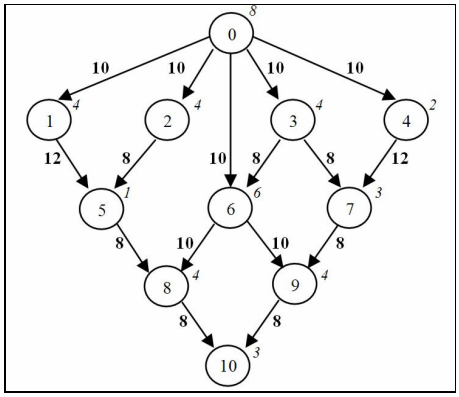

Figura 3: Grafo P11D

3 Algoritmo Genético

O algoritmo genético implementado nesse trabalho é inspirado no algoritmo genético padrão definido em [2]. A Figura 4 apresenta o seu fluxograma.

A primeira etapa do algoritmo genético é a geração da população inicial. Cada indivíduo dessa população é um escalonamento em n processadores de todas as tarefas existentes em um grafo de programa, ou seja, é uma possível solução para o problema. A Figura 5 apresenta um exemplo de indivíduo que representa um possível escalonamento do grafo de programa Gauss 18.

Considerando-se uma arquitetura com dois processadores - P0 e P1-, o primeiro indivíduo da figura representa o escalonamento das tarefas 0, 2, 6, 3, 8, 5, 9, 13, 15, 16, 14 e 17 no processador P0 e das tarefas 1, 11, 10, 4, 7 e 12 no processador P1, sendo que a ordem de execução das tarefas dentro de cada processador é dada pela ordenação das mesmas no indivíduo.

Tabela 1. Descrição dos grafos de 11 tarefas.

Grafo	*V*	*E*
P11A	{(0,8),(1,4),(2,4),(3,4),(4,4),(5,4),(6,6),(7,3),(8,3),(9,3),(10,3)}	{(0,1,8),(0,2,12),(1,3,12),(1,4,12),(2,5,8),(2,6,12),(3,7,8),(4,7,8),(4,8,8),(5,8,8),(5,9,8),(6,9,8),(7,10,12),(8,10,8),(9,10,12)}
P11B	{(0,8),(1,4),(2,4),(3,6),(4,4),(5,4),(6,3),(7,4),(8,3),(9,2),(10,3)}	{(0,1,8),(0,2,8),(0,3,8),(1,4,12),(2,4,8),(2,5,12),(2,6,8),(3,6,12),(4,7,8),(4,8,8),(5,8,8),(5,9,8),(6,10,12)}
P11C:	{(0,8),(1,3),(2,6),(3,4),(4,2),(5,4),(6,6),(7,3),(8,4),(9,3),(10,2)}	{(0,1,10),(0,2,10),(1,3,8),(1,4,8),(2,5,8),(2,6,8),(3,7,12),(4,8,8),(5,9,10),(6,10,8)}
P11D:	{(0,8),(1,4),(2,4),(3,4),(4,2),(5,1),(6,6),(7,3),(8,4),(9,4),(10,3)}	{(0,1,10),(0,2,10),(0,3,10),(0,4,10),(0,6,10),(1,5,12),(2,5,8),(3,6,8),(3,7,8),(4,7,12),(5,8,8),(6,8,10),(6,9,10), (7,9,8),(8,10,8),(9,10,8)}
P11E	{(0,8),(1,2),(2,3),(3,4),(4,2),(5,4),(6,6),(7,3),(8,4),(9,2),(10,3)}	{(0,1,12),(0,2,12),(0,3,12),(0,4,12),(1,3,8),(1,6,8),(2,4,8),(2,7,8),(3,5,10),(4,5,10),(4,7,10),(5,6,8),(5,8,8),(5,9,8)(6,8,10),(7,9,10), (8,10,4),(9,10,4)}
P11F	{(0,8),(1,4),(2,3),(3,4),(4,2),(5,4),(6,6),(7,1),(8,3),(9,2),(10,3)}	{(0,1,12),(0,2,12),(0,3,12),(0,4,12),(1,5,10),(2,5,4),(2,6,4),(2,7,4),(3,7,10),(3,8,10),(3,9,10),(4,9,8),(4,10,8)}
P11G	{(0,8),(1,4),(2,4),(3,4),(4,4),(5,4),(6,6),(7,3),(8,3),(9,3),(10,3)}	{(0,1,12),(0,2,12),(0,3,12),(1,4,8),(2,4,10),(2,5,10),(2,7,10),(3,6,8),(4,7,4),(5,8,4),(5,9,4),(6,8,4),(6,9,4),(7,10,4),(8,10,4),(9,10,4)}
P11H	{(0,8),(1,4),(2,4),(3,4),(4,2),(5,2),(6,3),(7,3),(8,6),(9,4),(10,4)}	{(0,1,10),(0,2,10),(0,3,10),(0,4,10),(1,5,8),(1,8,8),(2,5,12),(2,6,12),(3,6,10),(3,7,10),(4,7,12),(4,10,12),(5,8,4),(5,9,4),(6,9,4),(7,9,4),(7,10,4)}
P11I	{(0,8),(1,6),(2,4),(3,3),(4,4),(5,2),(6,4),(7,4),(8,6),(9,5),(10,2)}	{(0,1,8),(0,2,8),(0,3,8),(0,4,8),(0,5,8),(1,6,6),(2,6,6),(2,7,6),(3,7,6),(3,8,6),(4,8,6),(4,9,6),(5,9,6),(6,10,10),(9,10,10)}
P11J	{(0,8),(1,3),(2,4),(3,3),(4,1),(5,2),(6,4),(7,4),(8,6),(9,2),(10,3)}	{(0,1,10),(0,2,10),(0,3,10),(0,5,10),(0,7,10),(1,4,8),(2,5,8),(2,6,8),(3,7,8),(4,8,10),(4,9,10),(5,9,4),(6,9,4),(6,10,4),(7,10,12)}

A população inicial é gerada de forma aleatória, garantindo-se que todos os indivíduos gerados são válidos. Um indivíduo inválido caracteriza-se pelo fato de uma tarefa estar alocada antes de outra da qual é dependente, em um mesmo processador. Cada indivíduo gerado aleatoriamente, se necessário, é corrigido de forma a se tornar válido. Essa correção consiste em uma reorganização mínima das tarefas e é realizada assim que o indivíduo é gerado. Através de testes experimentais, foi possível constatar que mesmo empregando as operações de correção, a população inicial gerada possui aleatoriedade e diversidade. O parâmetro *Tpop* determina o tamanho da população do AG. Por fim, tem-se uma população inicial válida que é submetida ao processo de avaliação.

O método de avaliação de um algoritmo genético consiste em verificar, por meio de uma função, o quão bem adaptado está cada indivíduo da população. A adaptação deve ser maior quanto melhor for a solução que o indivíduo representa. No problema do escalonamento, a função contabiliza quantas unidades de tempo são necessárias para que todas as tarefas sejam executadas, de acordo com a ordem e alocação das mesmas, e das condições de precedência. É importante ressaltar que algumas tarefas necessitam de outras para serem realizadas, e, portanto, se tais tarefas forem executadas em processadores distintos a função terá que contabilizar um tempo extra de comunicação. Quanto menor for o tempo total de execução, melhor é o indivíduo da população. Um indivíduo é considerado ótimo se sua avaliação resultar no valor mais baixo possível para o grafo de programa que será escalonado.

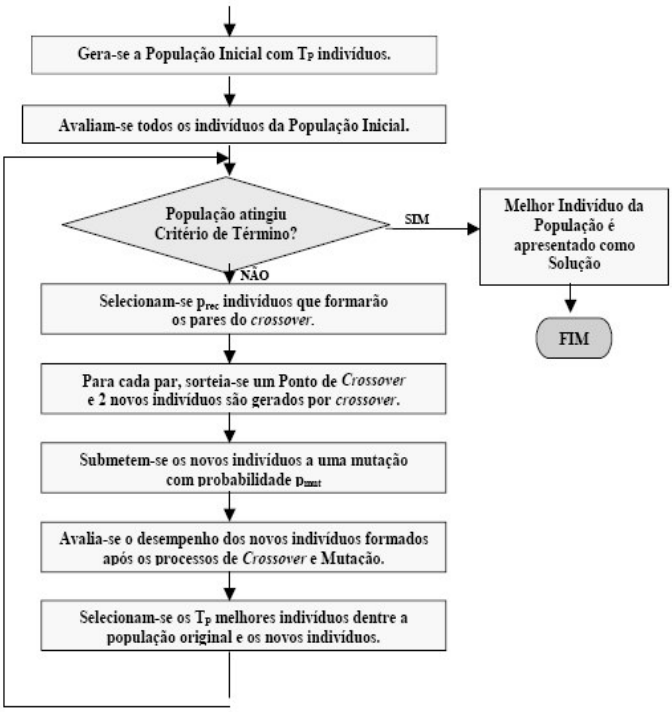

Figura 4: Fluxograma do Algoritmo Genético

Tarefa	0	2	6	3	8	5	9	13	15	16	14	17	1	11	10	4	7	12
Processador	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figura 5: Indivíduo do AG relativo ao Gauss 18

Após a avaliação da população, é feita a seleção dos indivíduos da população que participarão da reprodução. O método adotado para selecionar os indivíduos que passarão seu material genético às gerações futuras é o torneio simples [2]. Nesse método, sorteiam-se *Tour* indivíduos aleatoriamente e é feita uma verificação de qual possui a melhor adaptação (o menor tempo). Tal indivíduo será o vencedor do torneio. Desta maneira, são selecionados os casais (pares) que darão origem a uma nova geração.

Um casal de pais - cada pai vencedor de seu respectivo torneio - dará origem a dois filhos através do *crossover*. O método utilizado é o *Crossover* Cíclico [2], que evita que o indivíduo filho gerado possua tarefas repetidas. Esse método consiste no seguinte processo: após a seleção de dois indivíduos (os pais), sorteia-se uma posição aleatória no indivíduo chamado de Pai 1 para passar a sua respectiva tarefa para a mesma posição no novo indivíduo, o Filho 2. Para garantir que não haja repetição de tarefas nesses novos indivíduos, identifica-se no Pai 2, a tarefa que foi selecionada

no Pai 1, e transfere-se essa tarefa para o Filho 1, na mesma posição em que foi encontrada, formando-se assim um ciclo, onde todos os genes dentro desse ciclo são transferidos da mesma maneira. As posições são trocadas até que a última posição da troca seja a primeira do ciclo. Os genes que permanecem fora desse ciclo, são transferidos do Pai 1 e Pai 2, sem alterações, para as mesmas posições do Filho 1 e Filho 2, respectivamente. O ciclo de transferência é realizado somente entre as tarefas, uma vez que os processadores alocados para as mesmas não são trocados. A Figura 6 exemplifica a aplicação do *Crossover* Cíclico. Na figura, cada indivíduo pai é uma permutação dos dígitos de 1 a 8. Vemos que como resultado são gerados dois novos filhos que também são permutações válidas de 1 a 8, sem que haja repetição dos dígitos. O *crossover* é aplicado em diversos pares de pais gerados a partir da população corrente. O parâmetro que define a quantidade de pares a cada geração é a taxa de *crossover* (p_{rec}).

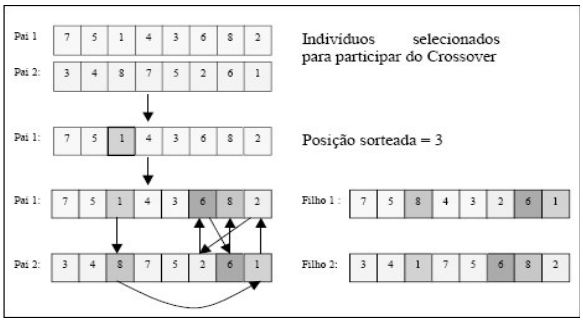

Figura 6: *Crossover* Cíclico

Após a aplicação do *crossover*, alguns filhos são submetidos à mutação. O método utilizado para a mutação é a Permutação Simples, que também evita que tarefas repetidas componham o cromossomo. Esse método consiste em sortear duas posições aleatórias de um indivíduo e trocar as respectivas tarefas. Os processadores não são permutados. A Figura 7 exemplifica a aplicação da Permutação Simples em um indivíduo. A mutação é importante, uma vez que impede a perda de diversidade genética. Por isso, um percentual dos filhos gerados pelo *crossover*, definido pela taxa de mutação (p_{mut}), é selecionado de maneira aleatória para sofrer mutação.

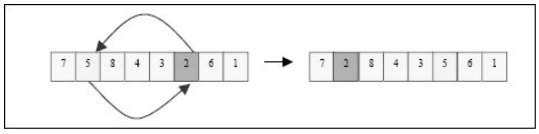

Figura 7: Permutação Simples

Nessa fase, após os processos de *crossover* e mutação, uma verificação dos novos indivíduos é feita, e os que não são válidos, são corrigidos. Isso evita que existam novos indivíduos inválidos. O processo de correção verifica se existe alguma precedência entre as tarefas que não esteja sendo atendida e altera os genes até que todas as posições sejam corrigidas, sendo que, no final do processo, todos os indivíduos são válidos.

Na etapa final da iteração do AG, é feita uma ordenação de toda a população. Essa ordenação feita de maneira que os primeiros indivíduos são aqueles que possuem maior adaptação, ou seja, menor tempo de execução do conjunto de tarefas. Os T_{pop} melhores indivíduos ao final de cada iteração são selecionados para formarem uma nova população que será utilizada na próxima iteração, que realizará novamente o processo de seleção, *crossover*, mutação e ordenação.

Cada iteração do AG é chamada de geração. Em nosso AG, foi utilizado como critério de parada um número máximo de gerações (N_{ger}). Uma vez que o número de gerações atinge esse patamar a execução do AG é interrompida e o melhor indivíduo da última geração é retornado como solução da execução.

Por sua natureza probabilística, normalmente é feita uma série de execuções de um AG para avaliar sua convergência para boas soluções.

4 Experimentos

Foi implementado em JAVA um primeiro ambiente baseado no algoritmo genético descrito na seção 3, que lê um grafo genérico e escalona seu conjunto de tarefas em dois processadores distintos. Os primeiros experimentos foram realizados com o grafo Gauss 18 (Figura 1) para ajuste dos parâmetros do AG. Como dito anteriormente, o tempo ótimo no escalonamento do Gauss 18 é 44 unidades de tempo.

Com o objetivo de avaliar diversas configurações dos principais parâmetros do AG, foram realizadas diferentes simulações. Em todas as simulações os parâmetros p_{rec} e p_{mut} não foram alterados e utilizamos p_{rec}=60%, p_{mut}=30%. Os parâmetros que foram alterados nesses testes foram:

- tamanho da população (T_{pop}): 50, 100 e 200
- número de gerações (N_{ger}): 200, 300 e 500
- tamanho do torneio (*Tour*): 2, 3 e 4

No total, 27 combinações desses parâmetros foram avaliadas. Executou-se o programa 20 vezes para cada uma das combinações, totalizando-se 540 execuções do AG. Em 188 execuções foi possível encontrar uma solução ótima. Portanto, no geral, a convergência para o ótimo foi de 34,8%. O pior caso encontrado foi de 55 unidades de tempo e a média geral (nas 540 execuções) foi de 47,2 unidades de tempo. Portanto, a média geral está 7,3% acima do valor ótimo e o pior caso está 25% acima. Entretanto, em algumas combinações, a convergência para o ótimo foi bem maior e, portanto, tivemos uma média mais baixa e o pior caso mais próximo do ótimo. Os resultados detalhados para cada combinação de parâmetros são apresentados na Tabela 3. Essa tabela apresenta na coluna *Convergência* o número de execuções do AG (no total de 20) que conseguiram encontrar uma solução ótima com *T*=44. A coluna *Média* apresenta a média dos valores obtidos nas 20 execuções de cada combinação e a coluna Pior apresenta o maior valor de tempo de escalonamento obtido nas 20 execuções.

Se analisarmos apenas o número de execuções que convergiram para o ótimo, as melhores combinações foram a 4 e a 11, nas quais 11 das 20 execuções convergiram para o ótimo (55% de convergência). Entretanto, essas combinações

apresentaram um valor de pior caso mais alto (53 e 55, respectivamente). Por outro lado, as combinações 10 e 19 tiveram uma convergência de 50% (10 em 20 execuções) quase tão boa quantos as anteriores e apresentaram um pior caso mais baixo: 51 e 49, respectivamente. A combinação 19 apresenta o menor valor de pior caso (49), a menor média (45,75) e o segundo melhor resultado de convergência (50%). Assim, nessa primeira análise destacamos as seguintes configurações:

- Combinação 4: *Tpop*=50, *Nger*= 300 e *Tour*=2;
- Combinação 10: *Tpop*=100, *Nger*= 200 e *Tour*=2;
- Combinação 11:*Tpop*=100, *Nger*= 200 e *Tour*=3;
- Combinação 19: *Tpop*=200, *Nger*= 200 e *Tour*=2.

Com o intuito de aprofundar a busca da melhor configuração, foi realizada uma nova análise dos resultados, agrupando-se as configurações pelos parâmetros *Tpop, Nger* e *Tour,* conforme apresentado na Tabela 3. Foi utilizado como critério para comparação dos resultados, a convergência para o ótimo (o total de soluções ótimas obtidas em 20 execuções). A melhor configuração é: *Tpop*=100, *Nger*= 200 e *Tour*=2, equivalente à combinação 10, já destacada anteriormente. Para finalmente escolhermos a melhor combinação dos parâmetros, consideramos um critério ainda não mencionado, que é o tempo de processamento. Vemos que a única diferença entre a combinação 10 e a 19 é que a segunda utiliza o dobro de população. Como o tempo de processamento do AG é diretamente proporcional ao tamanho da população temos que a configuração 19 leva duas vezes mais tempo que a configuração 10. Como o desempenho das duas foi similar, escolhemos a configuração 10 como o melhor ajuste dos parâmetros, dente os 27 avaliados. Assim, chegou-se à seguinte configuração para o AG, a partir dos experimentos com o grafo Gauss18: T_{pop}=100, *Tour*=2, N_{ger}= 200 p_{rec}=60% e p_{mut}=30%,. Essa configuração do Gauss 18 leva, em média, 0,92 segundos de processamento em uma máquina AMD 2.00 GHz, 1GB de RAM, que foi a máquina utilizada em todos os experimentos desse trabalho.

O AG ajustado com o Gauss 18 foi utilizado no escalonamento dos dez grafos de 11 tarefas apresentados na Tabela 1. Cabe ressaltar que nosso objetivo inicial era utilizar grafos de tamanho compatível com o Gauss 18 (número de tarefas em torno de 18). Entretanto, o algoritmo B&B implementado não foi capaz de encontrar o tempo ótimo para grafos dessa magnitude em um tempo viável. O próprio Gauss 18 foi executado no B&B e, após 2 dias de processamento, interrompemos a execução do mesmo.

Para se ter uma idéia prática da diferença entre os grafos, o número de vértices explorados pelo B&B nos grafos de 11 tarefas variou aproximadamente de 6500 (P11E) a 234130 (P11D), enquanto que o B&B, ao ser encerrado no Gauss 18, já havia explorado mais de 1.500.000 de vértices. Nosso objetivo com a realização desses novos experimentos com o AG foi o de investigar a convergência do AG em um conjunto de grafos com características estruturais bastante diversificadas, gerados manualmente mas de forma aleatória, e para os quais pudéssemos encontrar o ótimo com o algoritmo B&B. Assim, a utilização dos grafos de 11 tarefas atendeu ao nosso objetivo, com um tempo de processamento viável do B&B, variando de 1 segundo de processamento no melhor caso (P11E) a quatro minutos de execução no pior caso (P11D). Essa variação de tempo na utilização do B&B ressalta a diferença estrutural entre os grafos.

Tabela 2: Resultados AG: Gauss 18

	Tpop	*Nger*	*Tour*	*Converg*	*Média*	*Pior*
1	50	200	2	2	49,10	53
2	50	200	3	6	48,10	51
3	50	200	4	7	48,70	53
4	**50**	**300**	**2**	**11**	**46,45**	**53**
5	50	300	3	5	48,50	52
6	50	300	4	10	46,90	54
7	50	500	2	9	46,35	50
8	50	500	3	5	48,65	53
9	50	500	4	3	49,60	54
10	**100**	**200**	**2**	**10**	**46,90**	**51**
11	**100**	**200**	**3**	**11**	**46,40**	**55**
12	100	200	4	8	47,65	53
13	100	300	2	7	46,85	51
14	100	300	3	8	46,20	52
15	100	300	4	7	47,05	53
16	100	500	2	6	47,65	52
17	100	500	3	4	48,25	53
18	100	500	4	6	47,20	52
19	**200**	**200**	**2**	**10**	**45,75**	**49**
20	200	200	3	9	45,95	50
21	200	200	4	6	46,80	52
22	200	300	2	4	47,25	51
23	200	300	3	5	46,95	50
24	200	300	4	5	46,65	49
25	200	500	2	9	46,00	51
26	200	500	3	9	45,85	49
27	200	500	4	6	46,75	53

Tabela 3: Resultados agrupados: Gauss 18

Valor Parâmetro	*Convergência*
T_{pop}	
50	58
100	**67**
200	63
N_{ger}	
200	**69**
300	62
500	57
Tour	
2	**68**
3	62
4	58

Os dez grafos não sofrem grande oscilação no tempo de processamento ao serem escalonados pelo AG, pois os parâmetros permaneceram fixos e em todos os casos o tamanho do indivíduo foi de 11 genes. O tempo médio de execução do AG foi de 0,04 segundos, na mesma máquina descrita anteriormente. Os resultados de 20 execuções do AG para cada grafo são apresentados na Tabela 4. A tabela apresenta para cada grafo avaliado: o tempo ótimo obtido pelo B&B ($T_{Ótimo}$); o número de execuções em 20 que

encontraram uma solução ótima (*Convergência*); a média do valores obtidos nas 20 execuções e o aumento percentual que essa média representa em relação ao ótimo (*Média*), o pior valor obtido nas 20 execuções e o aumento percentual em relação ao ótimo. Como pode ser observado, nos 10 grafos o AG conseguiu encontrar uma solução ótima em várias execuções. Em dois grafos, P11A e P11J, o AG convergiu para o ótimo em todas as vinte execuções. Em cinco grafos (P11B, P11C, P11D, P11E e P11G), a convergência embora não tenha sido total, foi excelente: acima de 85%. No grafo P11H, a convergência foi um pouco mais baixa, 70%, mas ainda um valor muito bom. Em dois dos dez grafos analisados, P11F e P11I, o resultado de convergência para o ótimo foi baixo (15% e 20% das execuções). Entretanto, devemos destacar que, mesmo nesse pior caso, a convergência para soluções sub-ótimas foi muito bom, pois a média está próxima do ótimo (3,9% acima do ótimo no P11F e 2,5% no P11I) e mesmo o pior caso pode ser considerado sub-ótimo (7,14% acima do ótimo para o P11F e 3,25% para o P11I). Nos demais casos, apenas uma única execução do P11E superou 7% em relação ao ótimo, atingindo 10%. Em todos os outros oito grafos, o pior caso manteve-se abaixo de 6,5% em relação ao ótimo.

Tabela 4: Resultados do AG nos grafos de 11 tarefas

Grafo	$T_{Ótimo}$	*Convergência*	*Média* ($\%T_{Ótimo}$)	*Pior* ($\%T_{Ótimo}$)
P11A	40	20	40,0 (0%)	40 (0%)
P11B	31	18	31,4 (1,29%)	33 (6,45%)
P11C	32	17	32,5 (1,56%)	34 (6,25%)
P11D	38	19	38,1 (0,26%)	39 (2,63%)
P11E	40	18	40,6 (1,5%)	44 (**10%**)
P11F	28	**3**	**29,1** (**3,9**%)	30 (**7,14**%)
P11G	37	17	37,5 (1,35%)	39 (5,40%)
P11H	33	14	**33,8** (**2,42**%)	35 (6,06%)
P11I	32	**4**	32,8 (2,5%)	33 (3,25%)
P11J	30	20	30,0 (0%)	30 (0%)

5 Conclusões

A maioria das instâncias do problema de se escalonar tarefas em máquinas paralelas são consideradas intratáveis computacionalmente. Assim, o algoritmo genético implementado nesse trabalho se apresentou como uma alternativa viável para encontrar uma solução ótima, ou pelo menos uma solução sub-ótima, no escalonamento de grafos de programas paralelos, em um tempo de processamento muito bom (abaixo de 1 segundo para o Gauss 18 e abaixo de 0,1 segundo para os grafos de 11 tarefas).

No escalonamento do Gauss 18, com a configuração de parâmetros ajustada para o próprio grafo, foi obtida uma convergência de 50%, com uma média 6,59% acima do ótimo e o pior caso 15,9% acima do ótimo. No escalonamento de grafos de dimensão menor do que aquela para o qual o AG foi ajustado, o desempenho do AG foi ainda melhor: 75% de convergência para o ótimo, em média. Nas execuções que não atingiram o ótimo, a média dos tempos obtidos não excedeu 4% em relação ao ótimo e o pior caso não excedeu 10% em relação ao ótimo.

Não estamos dizendo que a configuração ajustada para o AG seria uma boa solução para qualquer instância de grafo de programa que se deseje escalonar. Mas, a sistemática desenvolvida nesse trabalho poderia ser utilizada em aplicações reais. Ou seja, o ajuste do AG pode ser realizado em instâncias conhecidas e de dimensão maior do que aquelas para os quais o AG irá efetivamente trabalhar.

Como continuidade imediata desse trabalho, iremos ampliar o número de testes com a configuração ajustada nesse trabalho, utilizando-se grafos de dimensão menor que o Gauss 18 (com um fator de ramificação similar aos utilizados nesse trabalho) e para os quais o algoritmo B&B seja capaz de encontrar o escalonamento ótimo. No momento, visamos grafos de 10, 11, 12 e 13 tarefas. A partir de 14 tarefas, não acreditamos que o B&B seja viável e abaixo de 9 tarefas o problema se torna muito trivial (tanto para o B&B, quanto para o AG).

Como trabalho futuro, pretendemos investigar a elaboração de um AG híbrido (ou memético) que incorpore algumas características do B&B em buscas locais, na tentativa de aumentar a convergência para o ótimo, dentro ainda de um tempo viável.

6 Referências

[1] J. Blazewicz. Scheduling in Computer and Manufacturing Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[2] D. E. Goldberg. Genetic Algorithms in Search, Optimization, andMachine Learning. Addison-Wesley, 1989.

[3] J. Jonsson & K. G. Shin. A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks in a multiprocessor system. In ICPP '97: Proceedings of the international Conference on Parallel Processing, pg. 158–165, Washington, DC, USA, 1997. IEEE Computer Society.

[4] F. Seredynski. Evolving cellular automata-based algorithms for multiprocessor scheduling. In S. Olariu A. Zomaya, F. Ercal, editor, Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences,Wiley Series on Parallel and Distributed Computing, pg. 179–207, New York, 2001. Wiley

[5] P. Brucker. Scheduling Algorithms. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.