Algoritmos genéticos e *Branch and Bound* aplicados ao escalonamento de tarefas em multiprocessadores

Helder Raimundo Gouveia Linhares
Jaqueline Aparecida Jorge Papini
Paulo Moisés Vidica
Gina Maira Barbosa de Oliveira

Programa de Pós-Graduação em Ciência da Computação
Faculdade de Ciência da Computação
Universidade Federal de Uberlândia
gina@facom.ufu.br

Resumo — Este artigo apresenta a utilização e comparação de duas técnicas de busca na obtenção de um escalonamento ótimo de tarefas entre dois processadores em um ambiente multiprocessado. A primeira delas é a técnica de busca não-determinista conhecida por algoritmo genético, que é inspirada na teoria evolutiva de Darwin. A segunda é o método de busca exata conhecido por *Branch and Bound*, que se baseia na enumeração em árvore das diversas soluções possíveis de um problema, aliada a um método de poda inteligente. Inicialmente, uma comparação do desempenho das duas técnicas é apresentada através de um exemplo de programa paralelo simples, composto por sete tarefas. Posteriormente, essas duas técnicas foram aplicadas a um problema conhecido na literatura por Gauss18, no qual apenas o AG encontrou o escalonamento ótimo. Após um ajuste no algoritmo Branch and Bound, as técnicas foram aplicadas a quinze variações do Gauss18, para as quais o escalonamento ótimo não é conhecido.

I. INTRODUÇÃO

O problema proposto 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. Encontrar a solução ótima significa obter a melhor combinação para a execução das tarefas nos processadores, ou seja, a alocação mais rápida de ser executada.

O problema de decisão vinculado ao problema de escalonamento de tarefas em uma arquitetura paralela (mesmo no caso de apenas dois processadores) é NP-completo [1]. Por isso, constitui-se em um desafio para vários pesquisadores na área de computação paralela. A maioria dos algoritmos de escalonamento conhecidos são exatos e seqüenciais. Existem vários motivos que levam à procura de alternativas aos algoritmos exatos: eficiência, desempenho, rápida adaptação de soluções a novas circunstâncias, dentre outros.

Dentre os procedimentos de busca e otimização, 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. Devido a isso temos sua aplicabilidade a esse problema, uma vez que apresenta soluções ótimas ou sub-ótimas com um baixo custo de processamento.

Por outro lado, a técnica *Branch and Bound* tem sido uma das mais utilizadas para encontrar a solução ótima de problemas de otimização vinculados a problemas de decisão NP [3]. A principal motivação para o uso dessa técnica em nosso trabalho, reside na possibilidade de se encontrar o ótimo de um programa paralelo não conhecido na literatura. Dessa forma, teríamos uma referência para a avaliação do desempenho dos AGs mesmo em programas paralelos gerados aleatoriamente.

II. O PROBLEMA DO ESCALONAMENTO

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 decisão vinculado ao escalonamento de tarefas que compõem um programa paralelo em uma arquitetura multiprocessadora é conhecido por ser 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} do link (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 um grafo de programa composto por sete tarefas que chamamos de Paralelo7. Criamos esse grafo com o objetivo de avaliar as técnicas investigadas nesse trabalho em um grafo mais simples. Através de uma inspeção manual foi possível identificar que o escalonamento ótimo desse grafo em uma arquitetura composta por dois processadores idênticos retorna um tempo mínimo de 75 unidades de tempo.

A Figura 2 apresenta o grafo de precedência de tarefas conhecido como Gauss18 [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.

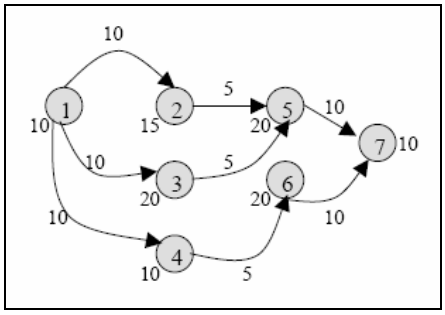

Figura 1: Grafo Paralelo7

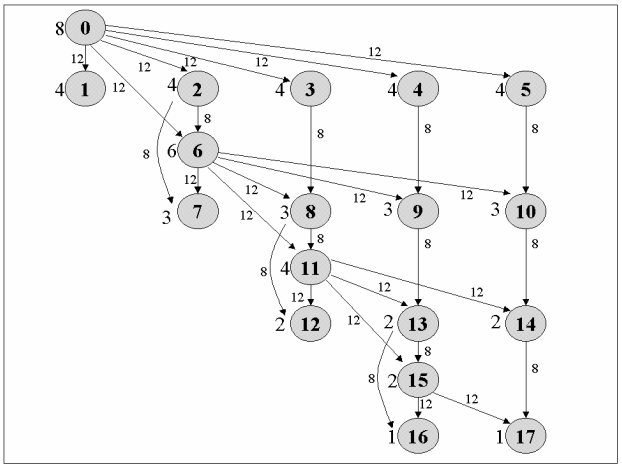

Figura 2: Grafo Gauss 18

Quinze variações do Gauss18 também foram utilizadas na avaliação das técnicas de busca empregadas no escalonamento. Essas variações também foram utilizadas em [5] e foram obtidas a partir de modificações aplicadas no grafo Gauss18 original. São elas: alteração no custo computacional de algumas tarefas, modificações no custo de comunicação entre determinadas tarefas e inclusão/exclusão de algumas arestas. O tempo mínimo retornado pelo escalonamento ótimo desses quinze grafos não é conhecido [5].

III. ALGORITMO GENÉTICO

O algoritmo genético implementado nesse trabalho é inspirado no Algoritmo Genético Padrão definido em [2]. A Figura 3 apresenta o seu fluxograma.

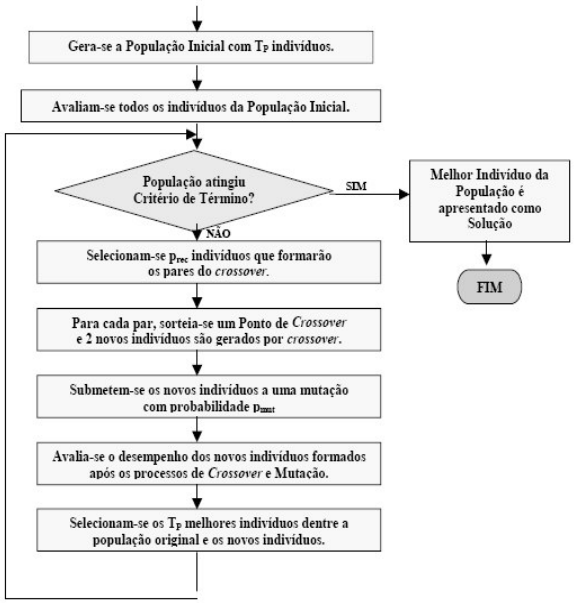

Figura 3: Fluxograma 1: Fluxograma do Algoritmo Genético

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 4 apresenta um exemplo de indivíduo que representa um possível escalonamento do grafo de programa Gauss18. 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.

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 4: Indivíduo do AG relativo ao Gauss 18

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 *Tp* determina o tamanho da população do AG. Diversos testes foram realizados com os seguintes valores: 50, 100 e 200 indivíduos.

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.

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 de execução). Tal indivíduo será o vencedor do torneio. Desta maneira, são selecionados os casais (pares) que darão origem a uma nova geração. Foram implementados torneios simples com *Tour* igual a 2, 3 e 4.

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. O ciclo de transferência do *crossover cíclico* é realizado somente entre as tarefas e os processadores alocados para as mesmas não são modificados. 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}.

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 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.

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 *Tp* 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 (*Nger*). 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.

III. ALGORITMO BRANCH AND BOUND

Os algoritmos *branch-and-bound* (B&B) são muito usados para resolver problemas combinatoriais. Estes algoritmos são baseados na idéia de enumerar todas as soluções *praticáveis* de um problema de forma direcionada e inteligente [6]. Uma solução é praticável se ela é uma solução completa "aceitável" para um determinado problema [3].

O B&B tem sido utilizado para encontrar soluções ótimas (ou sub-ótimas) para o problema do escalonamento de tarefas em sistemas multiprocessadores [3,6]. Ele é um eficiente algoritmo de busca 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.

Na aplicação do B&B ao problema do escalonamento de um grafo paralelo, a busca por uma solução é feita em uma *árvore de busca* que representa o espaço de soluções do problema, que são todas as possíveis permutações de atribuição tarefa-para-processador e ordem de execução das tarefas, isto é, a ordem de escalonamento. Cada vértice na árvore representa uma atribuição tarefa-para-processador, uma ordem de execução das tarefas nos processadores. O *nó raiz* da árvore representa um escalonamento vazio e cada um de seus descendentes (vértices filhos) representa o escalonamento de uma tarefa

específica em um processador específico. O filho de cada um destes vértices filhos representa o escalonamento de mais outra tarefa em um processador. A Figura 5 apresenta um grafo de precedência de tarefas composto por 4 tarefas e também apresenta um exemplo de árvore de busca para este grafo.

Um *vértice objetivo* na árvore de busca representa uma solução completa onde todas as tarefas foram alocadas nos processadores. O *nível* de um vértice é o número de tarefas que foram atribuídas para qualquer processador no escalonamento (nó da árvore) corrente. O *custo* de um vértice é a qualidade do escalonamento parcial representado pelo vértice e é dado pelo tempo total de execução para o escalonamento parcial representado pelo vértice.

Quando não há restrições de precedência entre as tarefas, considerando-se *n* tarefas e *m* processadores, o número de vértices objetivo na árvore de busca é $n!m^n$. Se as restrições de precedência são consideradas, o número de vértices objetivo na árvore de busca é reduzido para quase $k^n m^n$, sendo *k* o número máximo de vértices filhos de um vértice [3]. Devido ao crescimento exponencial do número de vértices na árvore de busca, vértices não são, normalmente, gerados até que o algoritmo B&B necessite explorá-los. Quando um novo vértice é gerado e ele pode levar a uma solução ótima, ele é chamado de *vértice ativo*. Geralmente, uma heurística é utilizada para definir a ordem em que os vértices serão explorados na árvore de busca. A estratégia B&B consiste em alternar operações de *branching* (ramificando) e *bounding* (avaliando) no conjunto de vértice ativos. *Branching* é o processo de gerar vértices filhos de um vértice ativo, e *bounding* é o processo de avaliar o custo de novos vértices filhos, para posteriormente decidir se eles também serão ramificados.

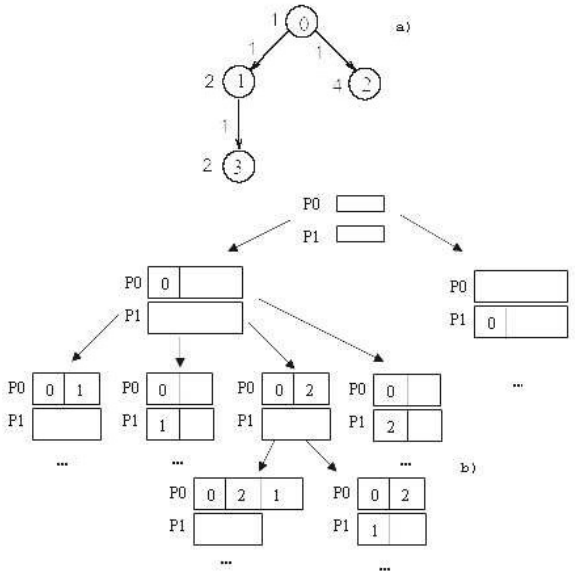

Figura 5: Exemplo de grafo de programa (a) e da árvore de busca (b) gerada pelo algoritmo B&B no escalonamento do grafo.

No B&B implementado nesse trabalho para o escalonamento de programas paralelos, a árvore de busca é iniciada com o escalonamento da tarefa 0 no processador 0. Em seguida, o próximo nível da árvore é formado por todos os vértices possíveis de serem criados, onde uma tarefa que dependa apenas da tarefa 0 é alocada em um dos dois processadores. De forma similar, cada um dos vértices poderá dar origem a um novo conjunto de vértices (expansão), alocando-se mais uma tarefa que dependa apenas das tarefas já alocadas.

A cada novo vértice expandido, é calculado o seu custo real que é dado pelo tempo total de execução das tarefas do escalonamento parcial que esse vértice representa.

A escolha do próximo nó a ser expandido é de extrema importância para a eficiência do algoritmo. Em nosso trabalho, o critério de escolha baseia-se na soma dada pelo custo real do escalonamento parcial das tarefas já alocadas adicionado a uma estimativa do custo das tarefas que ainda não foram alocadas. No nosso caso, adotamos como estimativa a alocação "ideal" das tarefas restantes, considerando-se apenas o tempo de execução das tarefas não alocadas, com um custo de comunicação igual a zero.

Todos os vértices não explorados são ordenados de acordo com o valor dessa soma (custo real das tarefas alocadas + estimativa das tarefas não alocadas) e colocados em uma fila de prioridade ascendente. A cada iteração, o vértice no início da fila é escolhido para expansão e removido da mesma.

O processo continua até que um *vértice objetivo* (solução completa) seja obtido e não exista nenhum nó não explorado com custo inferior ao melhor vértice objetivo. Quando essa situação é alcançada, o algoritmo é encerrado e o melhor vértice objetivo é retornado como solução ótima do escalonamento.

Ressalte-se que só é possível afirmar que a melhor solução encontrada é ótima porque a estimativa do custo utilizada é na verdade uma subestimativa. Ou seja, um vértice não explorado da fila de prioridade com custo superior ou igual ao custo real do vértice objetivo jamais gerará um descendente com custo mais baixo que o já alcançado.

O algoritmo descrito anteriormente encontra a solução ótima caso não existam nem limites de tempo de processamento nem de recurso de memória. Infelizmente, essa é uma situação ideal que só ocorre na prática para problemas de pequena dimensão (número baixo de tarefas). Para melhorar o desempenho do algoritmo, os vértices gerados podem ser eliminados por dois motivos: o mesmo constitui um individuo inválido ou foi encontrada uma solução (*vértice objetivo*) com um valor menor do que a estimativa de avaliação do mesmo.

Um indivíduo inválido caracteriza-se pelo fato de uma tarefa estar alocada antes de outra da qual é dependente em um mesmo processador. Os indivíduos inválidos são eliminados logo após a sua criação antes mesmo de serem avaliados. Quando um vértice ativo é encontrado, são eliminados da fila de prioridade todos os vértices que possuem custo (real + estimativa) superior ou igual ao custo real da solução completa.

IV. EXPERIMENTOS

- Experimentos iniciais

Foi implementado em JAVA um primeiro ambiente baseado no algoritmo genético descrito na seção II, que lê um grafo genérico e escalona seu conjunto de tarefas em dois processadores distintos. Testes preliminares foram realizados com os grafos Paralelo 7 (figura 1) e Gauss 18 (figura 2) para ajuste dos parâmetros do AG. A partir desses testes, chegou-se à seguinte configuração para o Gauss 18: T_P=100, p_{rec}=60%, p_{mut}=30%, Tour=2e Nger= 200. A configuração ajustada para o Paralelo 7 foi: T_P=50, p_{rec}=60%, p_{mut}=30%, Tour=2 e Nger= 50. A configuração do Gauss 18 leva, em média, 0,88 segundos de processamento em uma máquina AMD 2.00 GHz, 1GB de RAM, que foi a máquina utilizada em todos os experimentos desse trabalho. Na configuração do Paralelo7, o tempo médio de execução é de 0,05 segundos.

Posteriormente, foi implementado em JAVA um segundo ambiente baseado no algoritmo B&B descrito na seção III. Para avaliar o desempenho desse ambiente, o mesmo foi aplicado no grafo de programa Paralelo7 apresentado na figura 1. O algoritmo B&B foi capaz de encontrar uma solução ótima para o escalonamento (75 unidades de tempo) após 5 segundos de processamento. O tamanho máximo da fila de prioridade dos vértices não explorados, durante a execução, foi de 1406 elementos.

Para realizarmos nossa primeira análise comparativa, efetuamos um experimento com 20 execuções do AG na resolução do Paralelo 7 e em 100% das execuções uma solução ótima (75 unidades de tempo) foi encontrada. Assim, verificamos que para um grafo de programa pequeno, como o Paralelo 7, as duas técnicas foram capazes de encontrar o escalonamento ótimo com um tempo de processamento baixo. Porém, o AG é da ordem de 100 vezes mais rápido que o B&B.

Nossa segunda análise comparativa foi realizada utilizando-se o grafo Gauss18. O AG foi capaz de encontrar uma solução ótima (44 unidades de tempo) em 50% das vezes 20 execuções. Nas 10 execuções que não encontraram o ótimo, a média do tempo de escalonamento foi de 49 unidades de tempo e o pior caso foi de 51 unidades de tempo. Embora o tempo de processamento exigido pela configuração ajustada para o Gauss 18 tenha sido superior (0,88 segundos) em relação ao Paralelo7 (0,05 segundos) e a convergência para a solução ótima tenha caído para 50%, consideramos bom o resultado obtido pelo AG. Posteriormente, o B&B foi aplicado ao Gauss18. Após 48 horas de execução, abortamos a execução do programa. Assim, da forma original descrita na seção II, o algoritmo B&B não foi capaz de encontrar a solução para o programa paralelo Gauss18, composto pr 18 tarefas, em um tempo viável. Como esperado, o crescimento do tempo de processamento do B&B é exponencial em relação ao número de tarefas.

Foi necessária uma análise mais elaborada do B&B, para efetuarmos modificações no ambiente que serão descritas a seguir.

- Experimento com o B&B alterado

Após uma análise dos dados gerados durante a execução do B&B no escalonamento do Gauss18, verificamos que a fila de prioridade superou 700.000 elementos durante a execução e que nenhum vértice objetivo (solução completa) foi alcançado, de forma que a poda da fila não foi realizada em nenhum momento.

Fizemos duas alterações no algoritmo B&B descrito na seção II:

- Antes de iniciarmos a execução do algoritmo, uma amostra de soluções completas e viáveis foi gerada de forma aleatória, de forma similar à geração da população inicial do AG. Dessa amostra, a melhor solução é utilizada como o primeiro vértice objetivo na execução do B&B. Dessa forma, a execução já inicia com um valor de corte na soma do custo (parcial + subestimativa) que é aplicado a cada novo vértice expandido. Em nossos experimentos uma amostra de 200 soluções é gerada aleatoriamente.
- Foi estabelecido um tamanho máximo para a fila de prioridade ascendente. Dessa forma, se durante a execução do algoritmo, a fila atingir esse limite, a cada novo vértice gerado, o vértice com maior valor de soma do custo (parcial + subestimativa) é eliminado da fila.

A primeira alteração em nada modifica a garantia de se obter o ótimo, caso o algoritmo chegue a encontrar um vértice objetivo de custo inferior ao primeiro elemento da fila e sua execução seja encerrada. A segunda alteração, entretanto, faz com que o algoritmo perca a garantia de que o vértice objetivo é o ótimo global, no fim da execução do algoritmo.

De forma alternativa, armazenamos o maior valor de custo de um vértice cortado devido à segunda modificação (tamanho máximo da fila de prioridade excedido) durante a execução. Chamamos esse valor de *Piso* da solução ótima. Dessa forma, se ao final da execução encontrarmos um vértice de custo 30, mas o valor do piso é 25, podemos afirmar que a melhor solução encontrada é 30 e que, caso exista uma solução melhor (custo inferior a 30), com certeza esse custo é superior ou igual a 25.

Com essas modificações implementadas no ambiente do B&B, aplicamo-lo novamente ao grafo Gauss18. Com um tamanho máximo de fila igual a 10.000 elementos e uma amostra inicial com uma solução viável de 48 unidades de tempo, o B&B conseguiu alcançar uma solução ótima de escalonamento de 44 unidades de tempo. Nessa execução, o Piso encontrado foi de 39 unidades de tempo. Ou seja, se não soubéssemos *a priori* que a solução de 44 unidades trata-se de fato do ótimo global, poderíamos especular que poderia ainda existir uma solução melhor entre 39 e 43 unidades de tempo. O tempo de processamento do B&B nessa execução foi de aproximadamente 10,7 minutos.

Assim, vemos que mesmo com as modificações efetuadas, o B&B é bem mais lento que o AG (10,73 minutos versus 0,88 segundos) e não conseguimos garantir que o ótimo foi obtido.

Entretanto, de posse desses ambientes, conseguimos utilizar o B&B, com suas informações de melhor solução encontrada e piso para a solução ótima, na avaliação da convergência do AG em novos grafos para os quais não se conhece a solução ótima.

- Experimentos com as variações do Gauss18

Foram utilizadas quinze variações do Gauss18, retiradas de [5], para as quais não se conhece a solução ótima. Todos os quinze grafos possuem 18 tarefas e são muito similares ao Gauss 18 original. Por isso, a média do tempo de processamento é bem próximo ao obtido com o Gauss18: 0,92 segundos com o AG e 11,5 minutos com o B&B.

A Tabela 1 apresenta os resultados obtidos nas 15 variações.

Variação do Gauss18	**B&B**		**AG**
	Piso	**Melhor Solução**	
1	42	**48**	**47**
2	43	47	47
3	40	**47**	**46**
4	40	47	47
5	41.5	50	50
6	42	**49**	**47**
7	39	44	44
8	39	**45**	**44**
9	40	**47**	**46**
10	39	44	44
11	39	**47**	**46**
12	42	**48**	**47**
13	39	44	44
14	40	**47**	**46**
15	40	**47**	**46**

Tabela 1: Resultados obtidos com o B&B e o AG aplicados às quinze variações do Gauss18

Como pode ser observado na Tabela 1, nas quinze variações o AG conseguiu encontrar uma solução no mínimo tão boa quanto a melhor obtida pelo B&B. Além disso, em nove variações o AG superou o melhor resultado obtido pelo B&B. Pela informação dos pisos obtidos pelo B&B, não podemos afirmar em nenhuma das variações que o resultado obtido pelo AG seja de fato o ótimo global. Mas, podemos verificar que mesmo que esses valores não sejam ótimos, se encontram próximos deles. Assim, concluímos que o ambiente elaborado com o AG, embora não tenha a garantia da obtenção do ótimo, tem uma convergência satisfatória para soluções ótimas, ou pelo menos sub-ótimas, com um baixo tempo de processamento.

V. CONCLUSÕES

A maioria das instâncias do problema de se escalonar tarefas em máquinas paralelas são consideradas intratáveis computacionalmente. Assim, os algoritmos apresentados nesse trabalho se mostraram alternativas viáveis para encontrar uma solução sub-ótima, ou mesmo ótima, dependendo da dimensão do programa paralelo e da disponibilidade de equipamentos e tempo para sua execução.

O algoritmo genético estudado apresentou um melhor desempenho no problema comparado ao *Branch and Bound* adaptado, tanto em relação ao tempo de execução, quanto em relação ao menor custo obtido. Entretanto, o B&B serve como ferramenta para comprovar a boa convergência do AG em problemas para os quais a solução ótima não é conhecida.

Embora não possamos afirmar que os valores encontrados para as quinze variações do Gauss18 sejam de fato ótimos, temos boas evidências desse fato. Em [5], os valores obtidos nessas variações - utilizando-se uma outra técnica baseada em autômatos celulares e algoritmos genéticos, que também é heurística e não tem garantia de encontrar o ótimo - não superam os obtidos no presente trabalho. Realizamos outros experimentos elevando-se o tamanho da fila de prioridade para 50.000 no B&B. Embora o tempo de execução tenha se elevado para 4 horas, os valores de piso não garantem os resultados obtidos pelo AG como ótimos mas também as melhores soluções não superam os obtidos pelo AG.

VI. 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. Vidica & G. M. B. de Oliveira. Cellular automata-based scheduling: A new approach to improve generalization ability of evolved rules. In Brazilian Symposium on Artificial Neural Networks (SBRN). IEEE Press, 2006.

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