

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Well Dantas Moraes

**Modelos Evolutivos aplicados ao Escalonamento
de Tarefas em Sistemas Multiprocessados:
Algoritmo Genético Serial e Multipopulação**

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Bruno Well Dantas Moraes

**Modelos Evolutivos aplicados ao Escalonamento de
Tarefas em Sistemas Multiprocessados:
Algoritmo Genético Serial e Multipopulação**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Ciência da Computação.

Orientador: Gina Maira Barbosa de Oliveira

Coorientador: Tiago Ismaier de Carvalho

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2017

Resumo

Neste trabalho foi desenvolvido um algoritmo genético multipopulação para o problema de escalonamento de tarefas com custos de comunicação, com objetivo de comparar seu desempenho com o algoritmo genético serial. Para isto, um conjunto de instâncias do problema foi montado e abordagens de operações genéticas foram comparadas. Experimentos foram conduzidos com variação de parâmetros de número de populações. Foram avaliados a qualidade das soluções produzidas e o tempo de execução, e concluiu-se que o AGMP bem parametrizado geralmente obtém soluções melhores e em menos tempo.

Palavras-chave: algoritmo genético multipopulação, escalonamento, paralelismo

Lista de ilustrações

Figura 1	– (A): grafo laplace9 com custos de comunicação omitidos, que são iguais a 40 para todas as arestas. (B): diagrama de Gantt que representa um possível escalonamento para 4 processadores com makespan de 540 unidades de tempo.	11
Figura 2	– Grafos gauss27 e gauss35. Arestas cheias representam custo de comunicação igual a 12 e arestas pontilhadas, custo igual a 8. Fonte: elaborada pelo autor.	20
Figura 3	– Grafos laplace9 e laplace16. Todos os custos de comunicação nesta família são iguais a 40. Fonte: elaborada pelo autor.	20
Figura 4	– Grafos fft15 e fft39. Nesta família, os vértices possuem pesos iguais a 60 e os custos de comunicação são iguais a 80. Fonte: elaborada pelo autor.	21
Figura 5	– (A): grafo laplace9 com custos de comunicação omitidos, que são iguais a 40 para todas as arestas. (B): exemplo de indivíduo para o grafo laplace9 e 4 processadores. (C): diagrama de Gantt que representa o escalonamento correspondente ao indivíduo de (B), com <i>makespan</i> de 540 unidades de tempo.	22
Figura 6	– Exemplos de <i>crossover</i> sobre vetores de tarefa para o grafo laplace9, omitindo vetores de processadores (P) dos indivíduos. À esquerda, as tarefas dos indivíduos pais. À direita, tarefas de filhos gerados por crossovers 1-point ($F1$), 2-point ($F2$), uniforme ($F3$) e OX ($F4$). . . .	24
Figura 7	– Resultados agregados dos testes de configurações do AG. Os melhores resultados estão em destaque (vermelho).	28
Figura 8	– Resultados agregados dos experimentos variando número de populações do AGMP. Os melhores resultados estão em destaque (vermelho). . . .	29
Figura 9	– Distribuição de fitness encontrados em 10^5 execuções do AG e do AGMP de 8 populações, respectivamente.	30
Figura 10	– Boxplot das distribuições de soluções do AG e do AGMP, respectivamente.	30
Figura 11	– <i>Speedup</i> por número de populações.	31

Lista de tabelas

Tabela 1	–	Configurações de AG	25
Tabela 2	–	<i>Fitness</i> ótimos estimados para 4 processadores.	27
Tabela 3	–	Melhor <i>fitness</i> encontrado pelo AG e pelo AGMP com número de populações indicado.	29
Tabela 4	–	Estatísticas dos <i>fitness</i> encontrados em 10^5 execuções do AG e do AGMP de 8 populações.	29
Tabela 5	–	Tempo de execução médio dos experimentos, em milissegundos.	31

Lista de abreviaturas e siglas

AG	Algoritmo genético
AGMP	Algoritmo genético multipopulação
N_{pop}	Tamanho da população
N_{ger}	Número de gerações
T_{cross}	Taxa de <i>crossover</i>
T_{mut}	Taxa de mutação
F_{mig}	Frequência de migração
T_{mig}	Taxa de migração

Sumário

1	INTRODUÇÃO	8
1.1	Motivação	8
1.2	Método e objetivo	8
1.3	Estrutura do texto	9
2	REVISÃO BIBLIOGRÁFICA	10
2.1	Escalonamento de tarefas	10
2.2	Algoritmo Genético	11
2.2.1	Representação genética	12
2.2.2	<i>Fitness</i>	12
2.2.3	População inicial	13
2.2.4	Seleção	13
2.2.5	<i>Crossover</i>	14
2.2.6	Mutação	14
2.2.7	Reinserção	14
2.3	Algoritmo Genético Multipopulação	15
2.3.1	Topologia de comunicação	16
2.3.2	Frequência de migração	16
2.3.3	Seleção de migrantes	17
3	DESENVOLVIMENTO	18
3.1	Grafos para benchmark	18
3.1.1	Grafos de outros trabalhos	18
3.1.2	Famílias de grafos	18
3.2	Algoritmo Genético	19
3.3	Algoritmo Genético Multipopulação	23
4	EXPERIMENTOS E RESULTADOS	25
4.1	Definição de parâmetros	25
4.1.1	Modelo de Experimentos	26
4.1.2	Análise das configurações	26
4.2	Comparações	27
4.2.1	Análise da qualidade das soluções	27
4.2.2	Distribuição de soluções	28
4.2.3	Tempo de execução	29

5	CONCLUSÕES E TRABALHOS FUTUROS	32
	REFERÊNCIAS	34

1 Introdução

1.1 Motivação

A crescente utilização de sistemas multiprocessados e o desenvolvimento de programas paralelos e distribuídos implicam no aumento da relevância de métodos que buscam a eficiência em sua execução. Entre eles, o problema de escalonamento envolve alocar as subtarefas de um programa paralelo em cada processador de uma arquitetura multiprocessada de modo a reduzir o tempo de processamento total. Porém, o espaço de busca relacionado ao problema do escalonamento geralmente apresenta uma alta cardinalidade, dificultando a obtenção da solução ótima. Assim tem sido utilizadas heurísticas e aproximações para obter soluções aceitáveis para esse problema. (HOU; ANSARI; REN, 1994)

Algoritmo genético é uma técnica bio-inspirada e baseada em processos estocásticos que é utilizada para busca e otimização em problemas que envolvem espaços de busca de alta cardinalidade, incluindo o escalonamento (GOLDBERG, 1989). O AG mantém um conjunto de soluções que são evoluídas iterativamente a fim de explorar melhor o espaço de busca. Isso é feito a partir de operações que geram novas soluções a partir das soluções já exploradas, e as integram no conjunto de forma semelhante ao processo da seleção natural. No AG serial, estas operações são executadas sequencialmente.

Neste trabalho foi investigado um algoritmo genético multipopulação para o problema de escalonamento estático de tarefas com custos de comunicação. Tal técnica estende o modelo do AG serial mantendo subconjuntos de soluções que são evoluídos de forma semi-independente. Essa abordagem é interessante por explorar o paralelismo de sistemas multiprocessados, obtendo maior desempenho sem que haja degradação das soluções encontradas. (ALBA; TROYA, 1999)

1.2 Método e objetivo

O método empregado consiste na composição de um repositório de grafos de tarefas para testes, na implementação de um AG e sua extensão como AGMP, cujos resultados são comparados. Tais experimentos foram repetidos 100 vezes a fim de obter dados estatisticamente significativos do comportamento dos algoritmos, uma vez que eles operam com aleatoriedade. É esperado que ambos produzam soluções próximas às ótimas e que o AGMP apresente um ganho de desempenho em relação ao AG simples de acordo com o número de processos e processadores utilizados.

O objetivo deste trabalho é comparar os resultados dos dois algoritmos em termos

de qualidade das soluções obtidas e do tempo de execução.

1.3 Estrutura do texto

Este texto é organizado da seguinte forma:

- O capítulo 2 apresenta uma revisão de conceitos de escalonamento, AG, AGMP, e abordagens exploradas em trabalhos correlatos.
- O capítulo 3 contém descrições do conjunto de grafos de benchmark e do desenvolvimento e implementação do AG e do AGMP.
- O capítulo 4 descreve os parâmetros utilizados, as configurações avaliadas, o modelo de experimentos, e apresenta as comparações feitas entre o AG e o AGMP.
- O capítulo 5 descreve as conclusões e trabalhos futuros.

2 Revisão Bibliográfica

2.1 Escalonamento de tarefas

Escalonamento de tarefas é um problema clássico de otimização combinatória considerado computacionalmente intratável (HOU; ANSARI; REN, 1994). Ele consiste no mapeamento de um conjunto de tarefas a um conjunto de processadores de forma a satisfazer um critério de avaliação. Assim, algoritmos baseados em heurísticas, aproximações e meta-heurísticas foram desenvolvidos para abordar o problema (KWOK; AHMAD, 1999).

O problema tratado neste trabalho assume as seguintes propriedades (HOU; ANSARI; REN, 1994).

- Escalonamento estático e determinístico: os tempos de execução, comunicação e as relações de precedência entre tarefas são conhecidos e não se alteram.
- Processadores não-preemptivos: um processador executa apenas uma tarefa por vez.
- Os processadores são idênticos para fins de eficiência relativa entre processadores. Ou seja, todo processador leva x unidades de tempo para executar tarefas de custo x .

Uma instância do problema de escalonamento é representada por uma tupla (G, P) , em que P é o conjunto de processadores que executarão as tarefas; e $G = \{V, E\}$ é um grafo direcionado acíclico cujos vértices de V representam um conjunto de tarefas, e arestas de E representam relações de precedência entre as tarefas. Para cada tarefa $T_i \in V$ existe um peso w_i que indica o tempo de processamento necessário para executar a tarefa correspondente. Para cada aresta $e_{ij} \in E$ existe um custo c_{ij} que indica o tempo necessário de comunicação entre o término da execução de T_i e o começo da execução de T_j , no caso dessas tarefas serem executadas em processadores diferentes. Caso sejam executadas no mesmo processador, não há custo de comunicação. Na Figura 1A é apresentado o grafo de tarefas laplace9 com os custos de comunicação omitidos, uma vez que são iguais a 40 para todas as arestas.

Uma tarefa T_i só pode ser escalonada por um processador P_j após o término da execução de todas as tarefas que precedem T_i , levando em conta os custos de comunicação para predecessoras executadas em processadores diferentes de P_j .

Dado um conjunto de p processadores, um escalonamento pode ser representado por p sequências de tarefas, cada uma associada a um processador. Cada sequência é par-

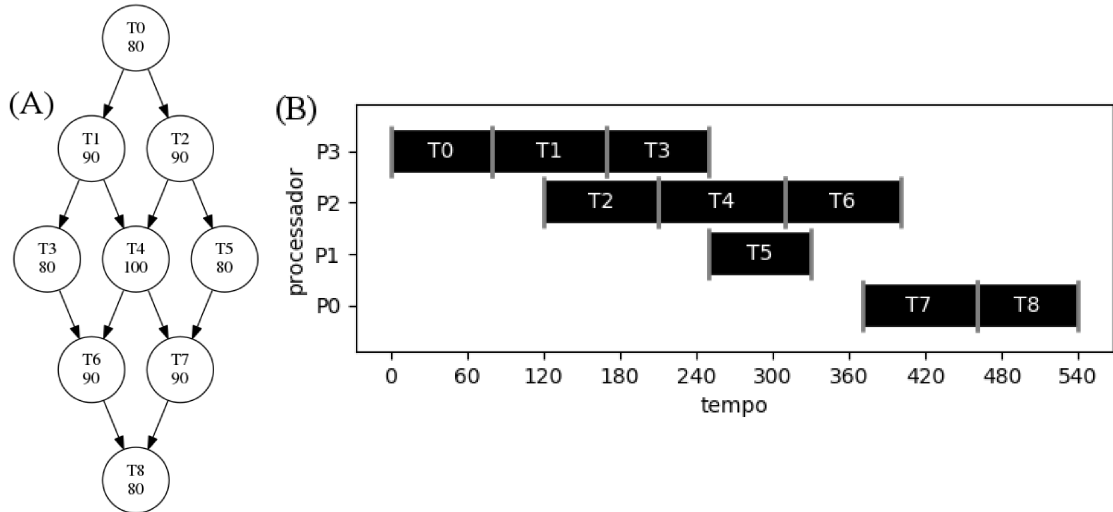


Figura 1 – (A): grafo laplace9 com custos de comunicação omitidos, que são iguais a 40 para todas as arestas. (B): diagrama de Gantt que representa um possível escalonamento para 4 processadores com makespan de 540 unidades de tempo.

cialmente ordenada, obedecendo as precedências do grafo. A sequência de tarefas alocada a um processador indica as tarefas que ele executará e a ordem.

Um processador fica ocioso caso a tarefa a ser executada dependa de um resultado de outra tarefa que ainda não foi terminada ou comunicada por outro processador.

Dada a representação de um escalonamento, é possível calcular uma linha do tempo de execução, com pontos de início e término de cada tarefa. Essa linha do tempo pode ser ilustrada por um diagrama de Gantt. A Figura 1B apresenta o diagrama de Gantt para uma possível solução de escalonamento do grafo laplace9 em uma arquitetura de 4 processadores, que demanda um tempo total de execução igual a 540.

Nesta variação do problema, supõe-se que um escalonamento não possui duplicação da execução de uma tarefa, i.e., cada tarefa é executada apenas uma vez por um único processador.

O tempo do escalonamento ou *makespan* corresponde ao ponto da linha do tempo em que a execução de todas as tarefas foi terminada. Isto corresponde à duração total do escalonamento. Um escalonamento ótimo é aquele que apresenta um *makespan* mínimo.

2.2 Algoritmo Genético

Algoritmo genético é uma meta-heurística proposta por John Holland que se inspira na evolução das espécies e tem sido usada para tratar diversos tipos de problemas relacionados a busca e otimização, incluindo o problema de escalonamento (GOLDBERG, 1989).

Um AG opera de forma estocástica para evoluir um conjunto de soluções com

objetivo de encontrar uma solução globalmente satisfatória no espaço de busca de um problema. Para isso, deve ser definida uma forma de representação de uma solução e uma função para avaliação de soluções. Depois é escolhida uma forma de inicializar uma população de soluções como ponto inicial da busca, e os operadores que modificarão tal população de forma iterativa. Tais operações são detalhadas nas próximas seções, e são apresentadas algumas das abordagens escolhidas em trabalhos correlatos.

Além do problema de entrada, um AG possui uma série de parâmetros numéricos para controlar seu comportamento.

2.2.1 Representação genética

Dado um problema a ser resolvido pelo algoritmo genético, deve ser definida uma estrutura de dados que codifique uma solução válida pertencente ao espaço de busca do problema. Uma instância dessa estrutura de dados constitui um indivíduo (ou “cromossomo”) que fará parte da população do AG (GOLDBERG, 1989).

A maior parte dos trabalhos consultados formulam suas representações considerando a ordenação topológica do grafo direcionado acíclico dado pelo problema de escalonamento, que permite a disposição dos vértices em uma lista ordenada pela precedência. Assim, para toda aresta e_{ij} , a posição da tarefa T_i na lista é anterior à posição de T_j .

A representação genética de Hou, Ansari e Ren (1994), Correa, Ferreira e Rebreyend (1999), e Kaur et al. (1999) mantém, para cada processador, uma lista de tarefas ordenada pela precedência. Wang et al. (1997), Omara e Arafa (2010), Chitra, Rajaram e Venkatesh (2011), e Morady e Dal (2016) empregam dois vetores de inteiros: um representa uma ordenação topológica de tarefas e o outro representa o processador correspondente a cada tarefa.

Kwok e Ahmad (1997) utilizam uma única lista de tarefas ordenada por precedência, cujos processadores são atribuídos durante a computação do *fitness*. Hwang, Gen e Katayama (2008) codificam a sequência de tarefas através de uma lista de prioridades numéricas, na qual os processadores associados correspondem ao valor da prioridade módulo o número de processadores. Xu et al. (2014) fazem uso de uma fila de prioridade que indica a ordem de execução das tarefas.

2.2.2 Fitness

Fitness ou aptidão é uma função escolhida para a avaliação da qualidade de um indivíduo. Em geral, todo indivíduo explorado pelo AG é avaliado.

Hou, Ansari e Ren (1994), Wang et al. (1997), Correa, Ferreira e Rebreyend (1999), Hwang, Gen e Katayama (2008), Omara e Arafa (2010), Morady e Dal (2016) utilizam o *makespan* como *fitness*. Kwok e Ahmad (1997) usam o *makespan* normalizado entre 0 e 1.

Kaur et al. (1999) consideram o *makespan* e o “*flowtime*”, que foi definido como soma dos tempos de término de cada tarefa. Chitra, Rajaram e Venkatesh (2011) fazem uma soma ponderada entre *makespan* e confiabilidade, obtida ao contabilizar possíveis cenários de falha de processadores. Xu et al. (2014) calculam o *fitness* subtraindo o *makespan* de um indivíduo do maior *makespan* presente na população.

2.2.3 População inicial

Gerar uma população inicial de indivíduos constitui a etapa de inicialização do AG. Dado um parâmetro N_{pop} que indica o tamanho da população, são gerados N_{pop} indivíduos, de forma estocástica, que representam soluções válidas e compõem a população a ser evoluída pelo algoritmo. Geralmente, a população de um AG é uma lista de indivíduos ordenada pelo *fitness*.

As populações iniciais de Hou, Ansari e Ren (1994), Correa, Ferreira e Rebreyend (1999), Hwang, Gen e Katayama (2008), Chitra, Rajaram e Venkatesh (2011), Morady e Dal (2016) são formadas por indivíduos gerados aleatoriamente. Wang et al. (1997), Kwok e Ahmad (1997), Kaur et al. (1999), Omara e Arafa (2010), Xu et al. (2014) geram uma parte da população aleatoriamente e obtêm a outra por heurísticas de ranqueamento de prioridade das tarefas.

2.2.4 Seleção

Ao começo de uma iteração (ou geração) do AG, pares de indivíduos são selecionados da população para reproduzir, gerando indivíduos filhos. Em geral, indivíduos são selecionados de acordo com seu *fitness*, tal que indivíduos melhores terão maior probabilidade de se reproduzir. O número de pares a serem selecionados é obtido a partir da taxa de *crossover*, que é dada como parâmetro do AG. (GOLDBERG, 1989)

Na literatura existem vários métodos para seleção. Hou, Ansari e Ren (1994), Wang et al. (1997), Hwang, Gen e Katayama (2008), Kaur et al. (1999), Chitra, Rajaram e Venkatesh (2011), Xu et al. (2014) utilizam o método da roleta, que associa a cada indivíduo uma probabilidade de ser selecionado diretamente relacionada a seu *fitness*. Kwok e Ahmad (1997) avaliam o *fitness* de um indivíduo em relação ao *fitness* médio da população para definir o número de vezes que cada indivíduo se reproduzirá. Omara e Arafa (2010) utilizam o método do torneio binário, em que o melhor entre dois indivíduos selecionados aleatoriamente é escolhido. Morady e Dal (2016) fazem parte da seleção com torneio e parte com a escolha direta de uma porção dos melhores indivíduos da população.

2.2.5 Crossover

O operador de *Crossover* é o método de gerar novos indivíduos a partir de um par de indivíduos pais. O objetivo do *crossover* é que os filhos possuam características de ambos os pais combinando suas representações. *Crossover* de um ou mais pontos consiste em dividir dois indivíduos (representados por vetores) em várias secções delimitadas por pontos que são alternadamente permutadas para gerar dois filhos. (GOLDBERG, 1989)

Wang et al. (1997), Kwok e Ahmad (1997), Chitra, Rajaram e Venkatesh (2011), Kaur et al. (1999), Omara e Arafa (2010), Xu et al. (2014) se baseiam em *crossover* de um ponto, com estratégias para reordenar ou recombinar metade do cromossomo conforme necessário. Hou, Ansari e Ren (1994) escolhem pontos de *crossover* durante iterações sobre as listas de tarefas de cada processador. Correa, Ferreira e Rebreyend (1999) aplicam *crossover* uniforme: mantém-se as tarefas que coincidem nos pais e sorteia-se uma das alternativas caso contrário. Hwang, Gen e Katayama (2008) desenvolvem um método baseado em *crossover* de dois pontos. Morady e Dal (2016) combina *crossover* de dois pontos e PMX, um algoritmo utilizado para vetores que envolvem permutação, que mantém uma subsequência do vetor inalterada e elimina elementos repetidos.

2.2.6 Mutação

Após o processo de *crossover*, alguns dos indivíduos filhos podem sofrer uma modificação pelo operador de mutação. A probabilidade de haver mutação em um filho definida pela taxa de mutação, a qual é dada como parâmetro do AG. Quando necessário, correções e adaptações são feitas para que o novo indivíduo represente uma solução válida. (GOLDBERG, 1989)

Hou, Ansari e Ren (1994), Wang et al. (1997), Kwok e Ahmad (1997), Hwang, Gen e Katayama (2008), Chitra, Rajaram e Venkatesh (2011), e Xu et al. (2014) utilizam *swap*: troca-se duas tarefas de lugar na sequência de execução, podendo implicar uma mudança de processador. Omara e Arafa (2010) mudam o processador em que uma tarefa é executada. Correa, Ferreira e Rebreyend (1999) fazem a remoção de uma tarefa e sua reinserção num ponto diferente. Kaur et al. (1999), Morady e Dal (2016) usam duas operações: *swap* sem mudança de processador ou apenas mudança de processador.

2.2.7 Reinserção

Dada uma lista de indivíduos pais e filhos, a operação de reinserção produz uma lista de indivíduos que constituirá a população na iteração seguinte combinando a população atual e a lista de filhos. (GOLDBERG, 1989)

Hou, Ansari e Ren (1994), Wang et al. (1997), Kwok e Ahmad (1997), Correa, Ferreira e Rebreyend (1999), Hwang, Gen e Katayama (2008), Kaur et al. (1999), Chitra,

Rajaram e Venkatesh (2011), Xu et al. (2014), Morady e Dal (2016) fazem reinserção por elitismo: uma porção dos melhores indivíduos permanece na população e o restante é preenchido pelos filhos gerados.

2.3 Algoritmo Genético Multipopulação

Um algoritmo genético paralelo é um modelo de AG para execução em sistemas multiprocessados visando obter um tempo de processamento menor em relação a execução serial. Segundo Cantú-Paz (1998), os vários tipos de AGs paralelos podem ser categorizados em:

- Paralelização global ou mestre-escravo: utiliza uma população única controlada por um programa mestre enquanto avaliações e/ou operadores genéticos são feitas por programas escravos.
- Paralelização de fina granularidade: voltada para sistemas massivamente paralelos. Utiliza uma população única cujos indivíduos competem e reproduzem apenas com uma vizinhança limitada.
- Paralelização multipopulação: utiliza várias subpopulações semi-isoladas que se comunicam com uma certa frequência.

Alba e Troya (1999) consideram AGs paralelos como uma classe distinta de meta-heurísticas por apresentarem a característica da interação local em subgrupos da população (com exceção da abordagem de paralelismo global), o que causa um impacto na qualidade das soluções. São citados casos em que AGs paralelos encontram soluções melhores do que as de AGs seriais. Os autores argumentam que a abordagem paralela é benéfica mesmo em sistemas que não possuam múltiplos processadores.

Algoritmo genético multipopulação é a forma mais popular de AG paralelo (CANTÚ-PAZ, 1998). Um AGMP pode ser encarado como uma extensão da implementação de AG, em que vários AGs independentes são executados, trocando alguns indivíduos entre si ocasionalmente. Assim, é relativamente simples converter um AG serial em AGMP, e é possível executá-lo em vários tipos de sistemas diferentes, seja uma máquina com um número qualquer de processadores ou uma rede de computadores (CANTÚ-PAZ, 1998).

AGMP é também conhecido como: AG distribuído, por sua abordagem relacionada a paralelismo MIMD (múltiplas instruções, múltiplos dados); AG paralelo de grossa granularidade, por sua alta taxa de computação por comunicação; e AG baseado em ilhas, devido a sua semelhança com o modelo de ilhas em genética populacional. O AGMP se caracteriza por usar poucas subpopulações de tamanho relativamente grande, em contraste

com as pequenas e numerosas vizinhanças mantidas na abordagem de fina granularidade (CANTÚ-PAZ, 1998).

O funcionamento do AGMP requer parâmetros adicionais como número de populações, frequência de migração e taxa de migração. Abordagens para a comunicação entre populações de AGMP são apresentadas a seguir.

2.3.1 Topologia de comunicação

A topologia de comunicação estabelece as ligações entre populações, definindo as populações de destino a serem consideradas por cada população durante migrações. Topologias são geralmente especificadas *a priori* e imutáveis ao longo da execução. (CANTÚ-PAZ, 1998)

As populações de Kwok e Ahmad (1997), Qi, Burns e Harrison (2000), Srinivasa, Venugopal e Patnaik (2007) são totalmente conectadas. Gehring e Bortfeldt (2002), Morady e Dal (2016) usam topologia em anel, na qual cada população possui uma única população de destino, descrevendo um anel conectado. Mühlenbein, Schomisch e Born (1991) dispõem uma estrutura de “escada circular”, com múltiplas conexões em cada população. Han et al. (2001) seguem uma estrutura de agrupamento voltada para a redução da carga de comunicação entre processadores. Yao, Kharm e Grogono (2010) utilizam uma topologia dinâmica em que populações podem se unir ou dividir durante a execução.

2.3.2 Frequência de migração

A frequência de migração define a periodicidade de trocas de indivíduos entre subpopulações, entre as quais cada população evolui isoladamente. Migrações podem ocorrer de forma síncrona num intervalo predeterminado, ou assíncrona sob satisfação de uma condição. (CANTÚ-PAZ, 1998)

Para o caso especial em que a frequência de migração é igual a zero, o AGMP se torna uma sequência de execuções independentes de AG serial com população reduzida. Tal abordagem é desencorajada em (CANTÚ-PAZ; GOLDBERG, 2003) por, geralmente, produzir soluções de baixa qualidade.

Mühlenbein, Schomisch e Born (1991), Qi, Burns e Harrison (2000), Han et al. (2001), Gehring e Bortfeldt (2002), Srinivasa, Venugopal e Patnaik (2007), Morady e Dal (2016) usam uma frequência de migração fixa. Kwok e Ahmad (1997) usam uma frequência exponencialmente crescente ao longo da execução.

2.3.3 Seleção de migrantes

Seleção de migrantes é o critério de escolha de quais indivíduos serão removidos de sua população de origem para serem inseridos em outra. Em geral, imigrantes substituem indivíduos emigrantes ou os de pior *fitness*. (CANTÚ-PAZ, 1998)

Em Mühlenbein, Schomisch e Born (1991), Kwok e Ahmad (1997), Qi, Burns e Harrison (2000), Han et al. (2001), Gehring e Bortfeldt (2002), Srinivasa, Venugopal e Patnaik (2007), e Morady e Dal (2016), os melhores indivíduos são selecionados para migração. Yao, Kharma e Grogono (2010) determinam migrações por análise de agrupamento de indivíduos.

O capítulo seguinte trata do desenvolvimento do AG e do AGMP para o problema do escalonamento de tarefas.

3 Desenvolvimento

O trabalho desenvolvido consiste em quatro etapas:

1. Selecionar um conjunto de grafos para benchmark, a ser utilizado nas avaliações comparativas dos algoritmos implementados.
2. Implementar e comparar configurações de AG, selecionando as melhores configurações.
3. Implementar e comparar configurações de AGMP, selecionando as melhores configurações.
4. Comparar as melhores configurações do AG e do AGMP.

3.1 Grafos para benchmark

Para fins de comparação, foi elaborado um conjunto de grafos referentes ao problema de escalonamento com custos de comunicação em algoritmos reais, uma vez que não foi encontrada uma base de dados padronizada como benchmark.

Compondo esse conjunto, foram utilizados grafos obtidos diretamente de outros trabalhos e também famílias de grafos baseadas em instâncias usadas na literatura.

3.1.1 Grafos de outros trabalhos

Em [Morady e Dal \(2016\)](#) foram disponibilizados onze grafos com número de vértices entre 4 e 18, obtidos de diversos trabalhos da literatura. Aqui, eles são referenciados com prefixo “TG”. Os autores não mencionam correspondência entre esses grafos e algoritmos paralelos reais.

Em [CARNEIRO \(2012\)](#) foi utilizada uma ferramenta para criação de três grafos aleatórios com 30, 40 e 50 vértices, identificados pelo prefixo “random”.

3.1.2 Famílias de grafos

Com objetivo de incrementar o conjunto com grafos relevantes e com espaços de busca de alta cardinalidade, foram implementados três scripts em linguagem Python que geram grafos pertencentes a “famílias” correspondentes a algoritmos paralelos mencionados na literatura: eliminação gaussiana, algoritmo de Laplace e transformada rápida de

Fourier (OLTEANU; MARIN, 2011). Assim, um grafo pertencente a uma dessas famílias corresponde à execução do algoritmo para um dado tamanho de entrada.

Os scripts implementados geram representações de grafos em arquivos de texto e também suas visualizações através da ferramenta Graphviz. Quatro grafos de cada família foram selecionados para compor o conjunto de benchmark, totalizando $11 + 3 + 12 = 26$ grafos.

A família “gauss” é definida pelo algoritmo da Decomposição LU, uma forma matricial do método da eliminação gaussiana. Esse um algoritmo que realiza fatoração de matrizes quadradas em matrizes triangulares, o que se reflete na estrutura desses grafos. Seu número de tarefas depende diretamente do tamanho da matriz de entrada, e seus pesos e custos de comunicação obedecem um padrão regular (JIANG; SHAO; GUO, 2014). A Figura 2 mostra duas instâncias desta família, nas quais as arestas contínuas representam custo de comunicação igual a 12 e as arestas pontilhadas, custo igual a 8 (COSNARD et al., 1988).

A família “laplace” é dada pelo algoritmo para resolução de equações de Laplace, composta por uma estrutura de tarefas que reflete diretamente o seu tamanho de entrada, que é uma matriz quadrada. Nesta família, ilustrada na Figura 3, as dependências entre tarefas são dadas pela forma em que a matriz é percorrida no algoritmo (WU; GAJSKI, 1990). Nesta família, os pesos das tarefas obedecem um padrão regular e todos os custos de comunicação são iguais a 40.

A família “fft” é baseada no algoritmo de Cooley–Tukey para a transformada rápida de Fourier. Esta família de grafos assume como entrada uma lista de tamanho igual a uma potência de 2. Seus grafos assumem uma estrutura com uma etapa recursiva e uma etapa de operação “borboleta” característica do algoritmo (XU; LI; HU, 2014). Duas instâncias são mostradas na Figura 4. Os pesos e custos de comunicação nesta família são iguais a 60 e 80, respectivamente.

3.2 Algoritmo Genético

Inicialmente, foi construído um AG serial para o problema do escalonamento de tarefas. Nessa etapa, algumas configurações relevantes de AG foram definidas e comparadas a fim de obter uma boa opção de operadores e parâmetros.

O propósito deste trabalho foi explorar abordagens de AG “puro”, em contraste com a tendência recente de desenvolver AGs híbridos incorporando diversas técnicas e heurísticas específicas do problema.

A representação do indivíduo utilizada é a mesma de Wang et al. (1997), Omara e Arafa (2010), Chitra, Rajaram e Venkatesh (2011), e Morady e Dal (2016): dois vetores de

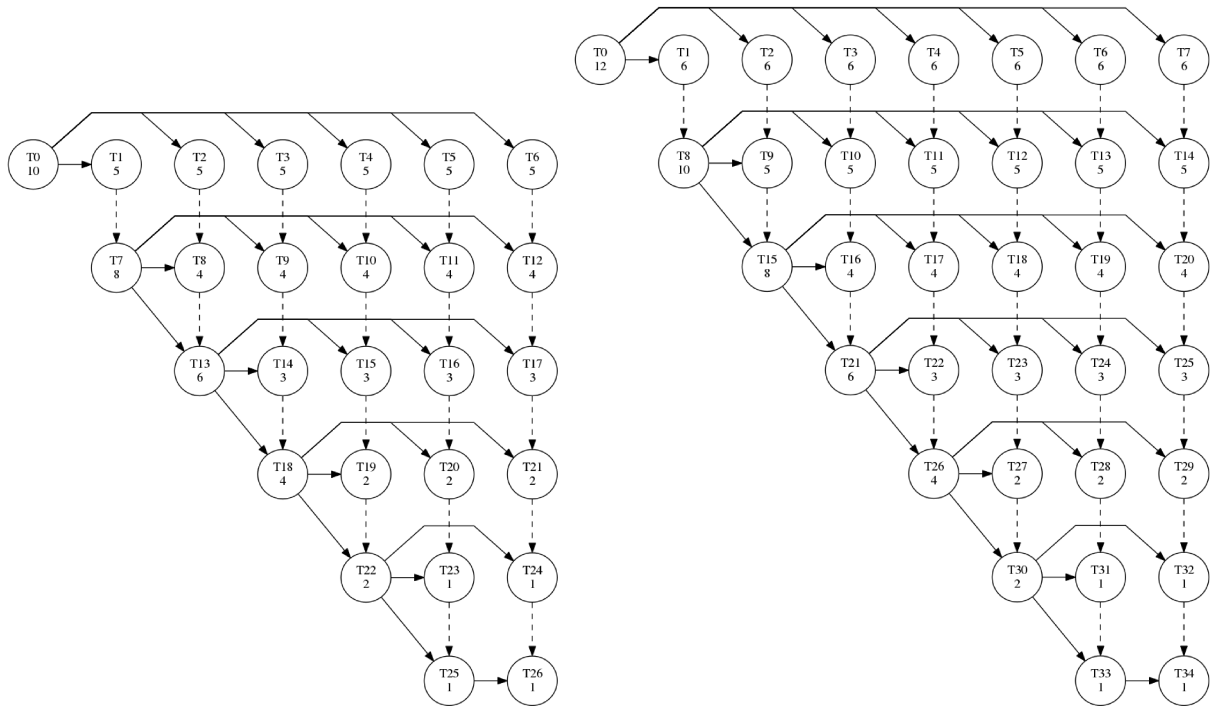


Figura 2 – Grafos gauss27 e gauss35. Arestas cheias representam custo de comunicação igual a 12 e arestas pontilhadas, custo igual a 8. Fonte: elaborada pelo autor.

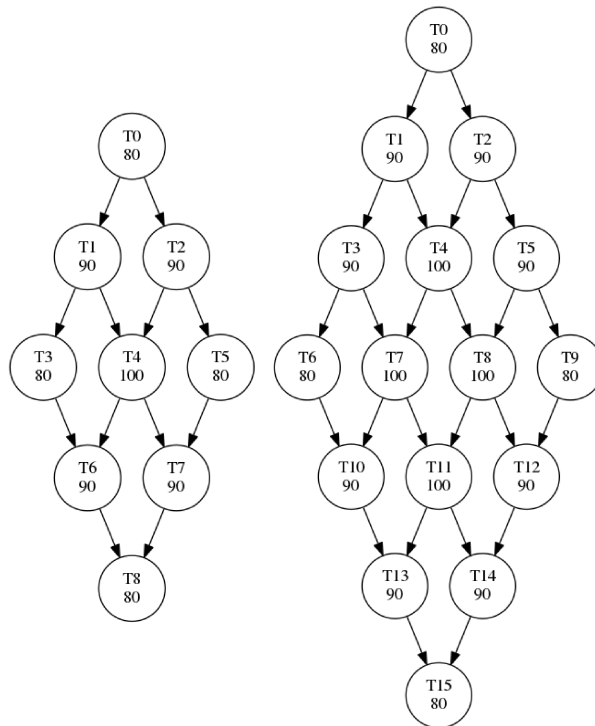


Figura 3 – Grafos laplace9 e laplace16. Todos os custos de comunicação nesta família são iguais a 40. Fonte: elaborada pelo autor.

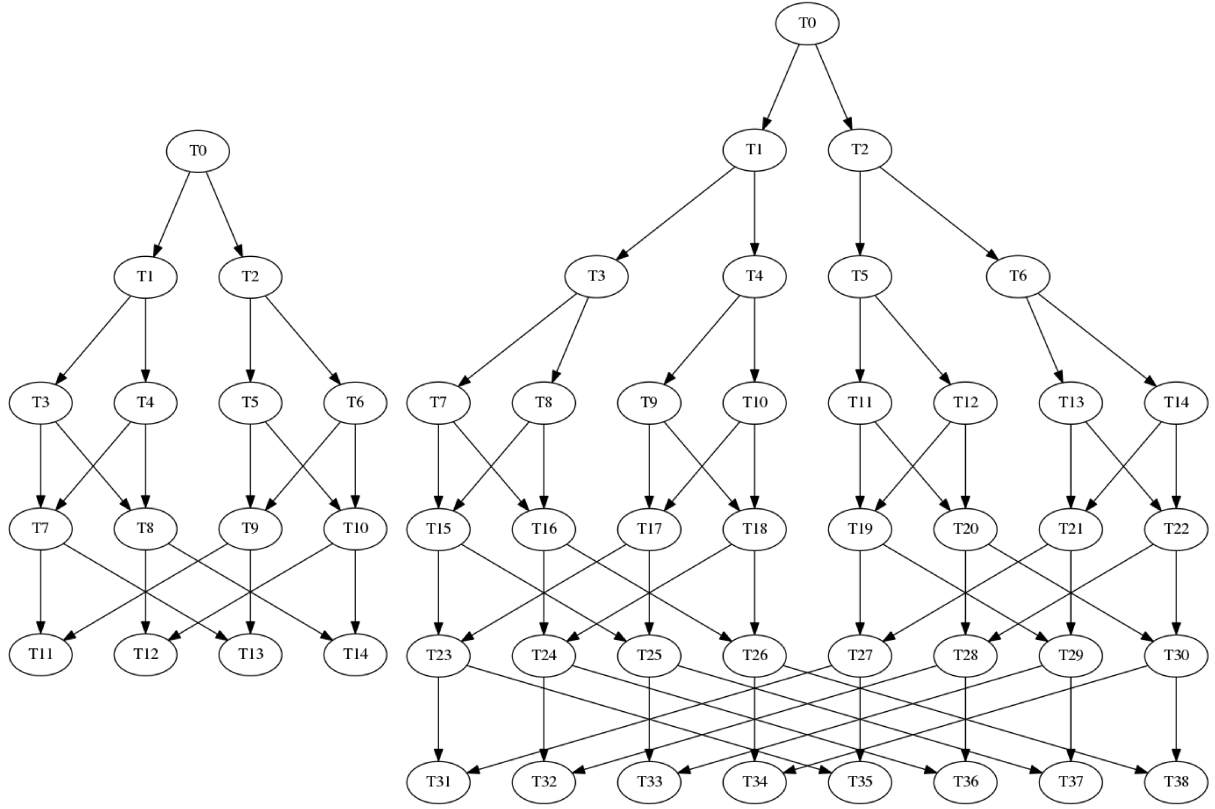


Figura 4 – Grafos *fft15* e *fft39*. Nesta família, os vértices possuem pesos iguais a 60 e os custos de comunicação são iguais a 80. Fonte: elaborada pelo autor.

inteiros S, P tal que S é uma sequência de tarefas que obedece as restrições de ordenação topológica do grafo dado, e P é um vetor que mapeia os vínculos tarefa/processador de forma que $P[tarefa] = processador$.

A Figura 5B ilustra um indivíduo válido para o grafo *laplace9* (Figura 5A), considerando-se alocação para 4 processadores. Para representar seu escalonamento correspondente, ilustrado na Figura 5C, o vetor $S = [0, 2, 5, 1, 3, 4, 6, 7, 8]$ é particionado em uma sequência para cada processador, obedecendo as associações dadas pelo vetor $P = [3, 3, 2, 3, 2, 1, 2, 0, 0]$. Por exemplo, as tarefas executadas pelo processador P3 são T0, T1 e T3, pois $P[0] = P[1] = P[3] = 3$. O processamento de cada tarefa é iniciado após a conclusão de todas as suas predecessoras, somado ao custo de comunicação em caso de serem executadas em processadores diferentes. Por exemplo, a tarefa T5 é iniciada após a conclusão e comunicação de T2. Com isso, T5 é executada após o início de T4, embora T5 seja anterior a T4 no vetor S .

A função de *fitness* utilizada é dada pelo *makespan* associado ao escalonamento. Por exemplo, o indivíduo representado na Figura 5B tem *fitness* igual a 540. A população inicial é gerada aleatoriamente com indivíduos válidos. A seleção dos pais para o *crossover* é feita com torneio binário. A reinserção da população no final de cada geração se dá por elitismo. A mutação é uma operação pontual em um dos dois vetores do indivíduo,

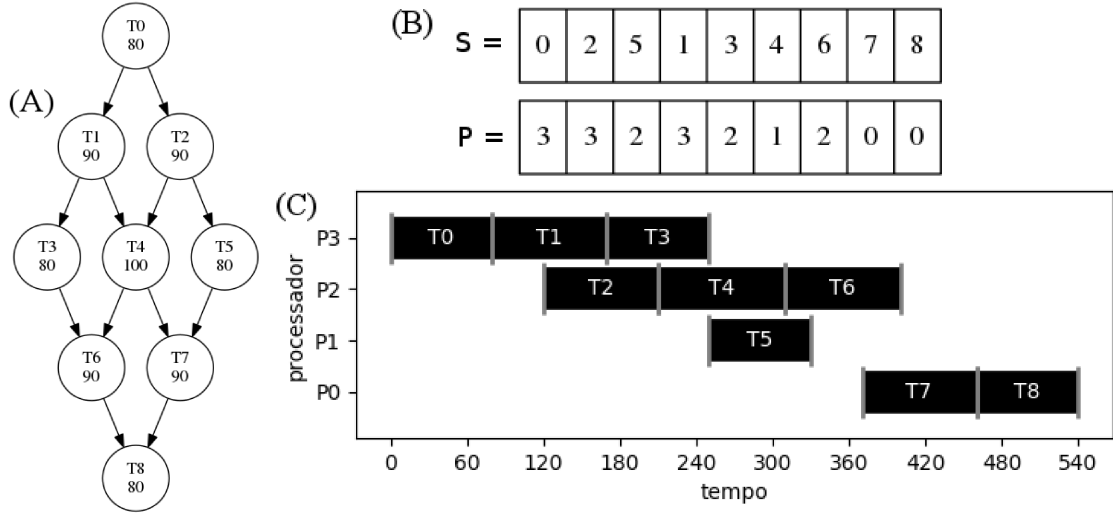


Figura 5 – (A): grafo laplace9 com custos de comunicação omitidos, que são iguais a 40 para todas as arestas. (B): exemplo de indivíduo para o grafo laplace9 e 4 processadores. (C): diagrama de Gantt que representa o escalonamento correspondente ao indivíduo de (B), com *makespan* de 540 unidades de tempo.

mudando o processador de uma tarefa ou trocando duas tarefas de lugar na sequência caso não haja dependência entre elas. Essas duas operações de mutação são referidas aqui como mutação “proc” e mutação “seq”, respectivamente.

Como a representação adotada é formada por duas partes distintas (S e P), existe uma variedade de opções diferentes para a aplicação do operador de *crossover*. Três abordagens foram observadas na bibliografia, aqui nomeadas como *carry*, *choose* e *combine*:

- *Carry*: o *crossover* é aplicado no vetor de tarefas “carregando” o vínculo de cada tarefa com o processador definido no pai correspondente. Assim, tarefas do indivíduo filho vindas do pai1 possuem o mesmo processador do pai1. Da mesma forma, tarefas vindas do pai2 são executadas nos processadores definidos no pai2. Esta abordagem é análoga a métodos usados em trabalhos com representação de listas de tarefas por processador (HOU; ANSARI; REN, 1994), (CORREA; FERREIRA; REBREYEND, 1999), (KAUR et al., 1999).
- *Choose*: é definido uma operação de *crossover* para o vetor de tarefas e uma para o vetor de processadores. Durante a reprodução de indivíduos, um dos dois tipos de *crossover* é sorteado e aplicado. O outro vetor é copiado do primeiro pai para o primeiro filho e vice-versa (OMARA; ARAFA, 2010).
- *Combine*: é definido um *crossover* para o vetor de tarefas e um para o vetor de processadores. Durante a reprodução de indivíduos, ambos *crossovers* são aplicados (MORADY; DAL, 2016).

Para avaliar estes métodos, foram implementados os seguintes algoritmos clássicos de crossover, que foram adaptados para evitar duplicação de tarefas e obedecer as restrições do vetor de tarefas (S):

- 1-ponto: sorteia-se um ponto de corte na sequência. Copia-se a primeira parte do pai1 e a segunda parte é preenchida, sem repetição, com tarefas seguindo a ordem em que se elas estão dispostas no pai2 (OMARA; ARAFA, 2010).
- 2-pontos: análogo ao anterior; a sequência é dividida em 3 partes. A primeira é uma cópia do pai1, a segunda é preenchida com tarefas na ordem do pai2, e a terceira é preenchida com tarefas restantes do pai1, também obedecendo sua ordem.
- Uniforme: a sequência filha é composta percorrendo os pais e escolhendo aleatoriamente de qual pai será herdada a tarefa seguinte, ignorando repetições.
- Cíclico: uma posição p é sorteada e os p -ésimos elementos das sequências são trocados. Enquanto haja uma duplicata na primeira sequência, o elemento repetido é trocado pelo elemento na mesma posição na outra sequência. É necessária validação posterior para garantir as restrições de precedência.
- PMX: mantém-se uma subsequência do pai1, troca-se o restante com elementos do pai2 e faz trocas até eliminar duplicatas. Também é necessária validação do filho.
- OX: mantém-se uma subsequência do pai1 e preenche-se o restante seguindo a ordem do pai2. A versão implementada gera um filho válido por garantir que todas as tarefas predecessoras da subsequência estejam à esquerda dela.

Alguns desses métodos de *crossover* são exemplificados para o grafo laplace9 na Figura 6, em que filhos $F1$, $F2$, $F3$, $F4$ foram produzidos respectivamente por *crossover* 1-ponto, 2-pontos, uniforme, OX, a partir dos pais $P1$ e $P2$. Além deles, foram implementados crossovers 1-ponto, 2-pontos e uniforme para o vetor de processadores, que são análogos aos descritos e sem restrição sobre repetição e ordenação. A implementação foi feita em linguagem C e compilada e otimizada com GCC.

3.3 Algoritmo Genético Multipopulação

O AGMP desenvolvido nesse trabalho emprega topologia em anel unidirecional para comunicação entre as populações, que é um modelo de simples implementação e que produziu resultados de boa qualidade em trabalhos correlatos (CANTÚ-PAZ, 1998). No anel unidirecional, toda população tem uma vizinha para onde envia alguns indivíduos, e uma segunda vizinha da qual recebe outros indivíduos em um processo conhecido por migração. Os melhores indivíduos da população são os selecionados para a migração. Os

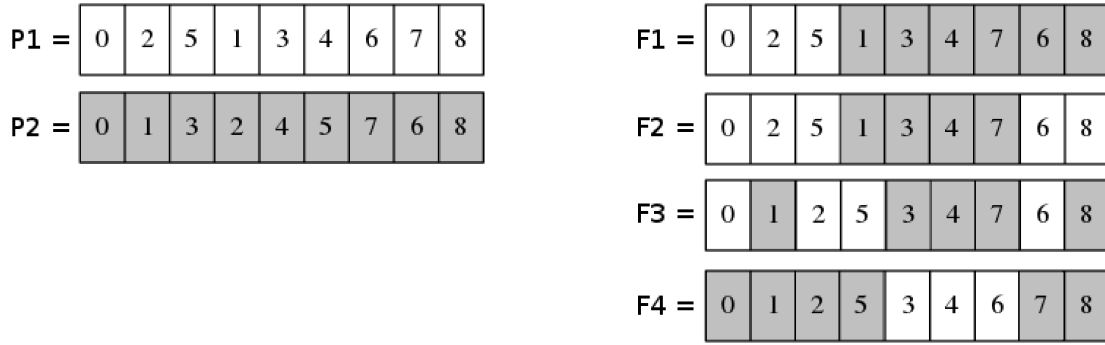


Figura 6 – Exemplos de *crossover* sobre vetores de tarefa para o grafo laplace9, omitindo vetores de processadores (P) dos indivíduos. À esquerda, as tarefas dos indivíduos pais. À direita, tarefas de filhos gerados por crossovers 1-point ($F1$), 2-point ($F2$), uniforme ($F3$) e OX ($F4$).

parâmetros de migração foram definidos experimentalmente: $T_{mig} = 10\%$ e $F_{mig} = 5$. Demais parâmetros são dados pela *config6*, escolhida para o AG serial.

Note que a quantidade de avaliações de *fitness* realizadas pelo AGMP é igual à do AG, pois o número total de indivíduos é distribuído igualmente entre cada população. Por exemplo, para 400 indivíduos e 4 populações, o tamanho de cada uma é igual a 100.

A taxa de migração é relativa ao tamanho das populações. No exemplo de 4 populações com 100 indivíduos cada, os parâmetros de migração ditam que os 10 melhores indivíduos (10%) de cada população migrarão.

A implementação foi feita estendendo o AG com procedimentos de inicialização das populações e migração, que foram feitos em memória distribuída com a biblioteca OpenMPI, que implementa o padrão *Message Passing Interface* para execução paralela de programas através de múltiplos processos. Dessa forma, cada população é executada em um processo diferente que se comunica com populações vizinhas de acordo com a frequência de migração. Por fim, os melhores indivíduos de cada população são enviados para uma mesma população, e o melhor deles é retornada pelo programa. Uma execução do AG serial é equivalente a executar o AGMP com número de populações igual a 1.

4 Experimentos e Resultados

Neste capítulo são definidos os parâmetros a serem usados pelos algoritmos e o modelo de experimentos para as comparações. Em seguida, o AG e o AGMP são comparados em termos de qualidade das soluções produzidas e tempo de execução.

4.1 Definição de parâmetros

Para escolha da forma de aplicação de *crossover*, foram elaboradas execuções do AG serial com as configurações apresentadas na Tabela 1, que contém duplas de operações de *crossover* para vetor de tarefas e de processadores em casos do tipo *choose* ou *combine* e apenas um tipo de *crossover* para o vetor de tarefas no caso *carry*.

Config0 é uma configuração sugerida nas fases iniciais deste trabalho a partir de experiências anteriores do grupo com o problema. *Config1* é baseada na descrição de (OMARA; ARAFA, 2010). *Config2* é igual à anterior, com a adição de mutação no vetor de tarefas (mutação “seq”). *Config3* é baseada em (CORREA; FERREIRA; REBREYEND, 1999), com a representação genética de dois vetores aqui descrita em vez de listas de tarefas para cada processador. *Config4* é igual à anterior com seleção via torneio em vez de roleta. *Config12* segue o trabalho de Morady e Dal (2016). As restantes são algumas variações das configurações anteriores.

O número de avaliações de *fitness* feitas em uma execução do AG representa o

Tabela 1 – Configurações de AG

config	seleção	método	crossover(s)	mutação
0	torneio	<i>carry</i>	cíclico	seq/proc
1	torneio	<i>choose</i>	1-point, 1-point	proc
2	torneio	<i>choose</i>	1-point, 1-point	seq/proc
3	roleta	<i>carry</i>	uniforme	seq/proc
4	torneio	<i>carry</i>	uniforme	seq/proc
5	torneio	<i>choose</i>	2-point, 2-point	seq/proc
6	torneio	<i>combine</i>	1-point, 1-point	seq/proc
7	torneio	<i>combine</i>	uniforme, uniforme	seq/proc
8	torneio	<i>combine</i>	2-point, 2-point	seq/proc
9	torneio	<i>choose</i>	uniforme, uniforme	seq/proc
10	torneio	<i>carry</i>	pmx	seq/proc
11	torneio	<i>carry</i>	ox	seq/proc
12	torneio	<i>combine</i>	pmx, 2-point	seq/proc
13	torneio	<i>combine</i>	cíclico, 2-point	seq/proc
14	torneio	<i>combine</i>	ox, 2-point	seq/proc

número de pontos explorados no espaço de busca. Esse número decorre dos parâmetros adotados no AG. Os parâmetros usados por Morady e Dal (2016) foram tomados como ponto de partida neste trabalho: $N_{pop} = 200$, $N_{ger} = 400$, e $T_{cross} = 65\%$. Dessa forma, os parâmetros iniciais totalizam 52200 avaliações de *fitness*, de acordo com o cálculo:

$$\begin{aligned} \text{avaliações} &= N_{pop} + N_{ger} * N_{pop} * T_{cross} \\ \text{avaliações} &= 200 + 400 * 200 * 0.65 = 52200 \end{aligned}$$

Porém, foi observado que as configurações implementadas produziam resultados melhores se a população fosse maior em relação ao número de gerações. Então, os parâmetros adotados foram: $N_{pop} = 400$, $N_{ger} = 198$, $T_{cross} = 65\%$ e $T_{mut} = 35\%$, totalizando 51880 avaliações de *fitness*, ou seja, aproximadamente o mesmo número de avaliações adotado em (MORADY; DAL, 2016).

4.1.1 Modelo de Experimentos

Para avaliar a qualidade das soluções de cada configuração e levando-se em conta o comportamento estocástico dos AGs, os experimentos consistem em 100 execuções para cada grafo do conjunto de benchmark em cada uma das configurações da Tabela 1. O número de processadores foi fixado em 4 para todos os experimentos.

As informações obtidas nesses experimentos incluem: convergência (número de vezes em que foi encontrada uma solução ótima); melhor *fitness* obtido; *fitness* médio; pior *fitness*.

As soluções ótimas não são conhecidas para a maioria dos grafos do conjunto, que foram então estimadas utilizando-se 100 execuções do AG implementado com população de 1200 indivíduos e 800 gerações. Os melhores resultados são apresentados na Tabela 2. Esses resultados foram considerados como “ótimos” no presente trabalho.

Para agregar e comparar os resultados das configurações, foram usadas quatro métricas: média das convergências e média dos desvios do melhor, médio e pior *fitness* em relação ao *fitness* ótimo, tal que o desvio de um *fitness* x é dado por:

$$\text{desvio}(x) = \frac{x - \text{ótimo}}{\text{ótimo}}$$

Assim, uma boa configuração é a que obtém alta convergência e baixos desvios, o que significa que suas soluções são próximas da solução ótima.

4.1.2 Análise das configurações

A Figura 7 apresenta os gráficos comparando as métricas de cada configuração. É possível verificar que *config6* obteve o melhor resultado em 3 das 4 métricas. Essa

Tabela 2 – *Fitness* ótimos estimados para 4 processadores.

grafo	fitness
TG4	34
TG9	21
TG10	83
TG11	20
TG12	27
TG14	35
TG17	37
TG18	440
TGb9	16
TGb11	49
TGc11	12
gauss27	67
gauss35	91
gauss44	116
gauss65	173
laplace9	520
laplace16	760
laplace25	1000
laplace36	1260
fft15	560
fft39	820
fft95	1600
fft223	3520
random30	751
random40	550
random50	496

configuração utiliza torneio binário na seleção dos pais, método *combine* no *crossover*, com *crossover* 1-ponto em ambos os vetores, ambas mutações “seq” e “proc” e reinserção por elitismo. Com isso, conclui-se que *config6* produziu boas soluções de forma mais consistente que as outras no caso geral. Assim, essa configuração é utilizada na próxima seção, para elaboração e comparação com o AGMP.

4.2 Comparações

Para comparar o desempenho do AG e do AGMP, foram feitos experimentos que envolvem a qualidade das soluções, a distribuição de *fitness* e o tempo de execução.

4.2.1 Análise da qualidade das soluções

Seguindo o modelo de experimentos definido para o AG serial, foram feitos testes comparativos variando-se o número de populações do AGMP entre 1 e 10, tal que as

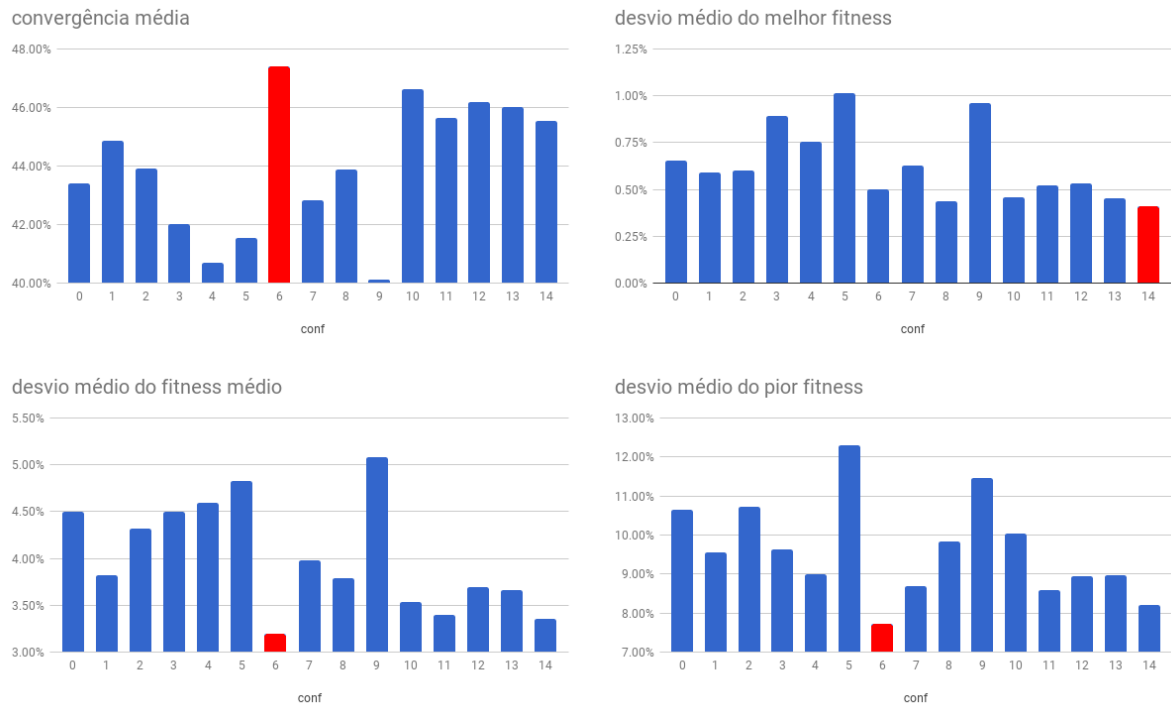


Figura 7 – Resultados agregados dos testes de configurações do AG. Os melhores resultados estão em destaque (vermelho).

execuções com uma população são equivalentes ao AG serial.

Os resultados da Figura 8 mostram que a qualidade das soluções do AG serial foi superada em todas as métricas por pelo menos uma configuração do AGMP. Em alguns dos experimentos, também nota-se que o AGMP encontrou soluções melhores do que as do AG com a configuração 6, como nos grafos *fft39*, *fft95*, *laplace36* e *random40*, que são apresentados na Tabela 3. Entretanto, esses valores não superaram os resultados considerados como ótimos (Tabela 2), obtidos por execuções do AG com parâmetros mais robustos.

A configuração do AGMP com 7 populações foi a melhor em convergência e obteve o desvio mais baixo de *fitness* médio, mas com um alto desvio de melhor *fitness*. Por outro lado, o AGMP com 8 populações obteve valores baixos e consistentes de desvio, possuindo o menor desvio de pior *fitness*, e alta convergência (a segunda melhor). Por isso, o AGMP com 8 populações foi escolhido como a melhor configuração, e adotada como padrão nas seções seguintes.

4.2.2 Distribuição de soluções

Para observar a distribuição da qualidade de soluções, foram feitos 10^5 experimentos com AG e AGMP para o grafo *gauss44* na arquitetura de 4 processadores, resultando nos histogramas da Figura 9 e nas estatísticas da Tabela 4.

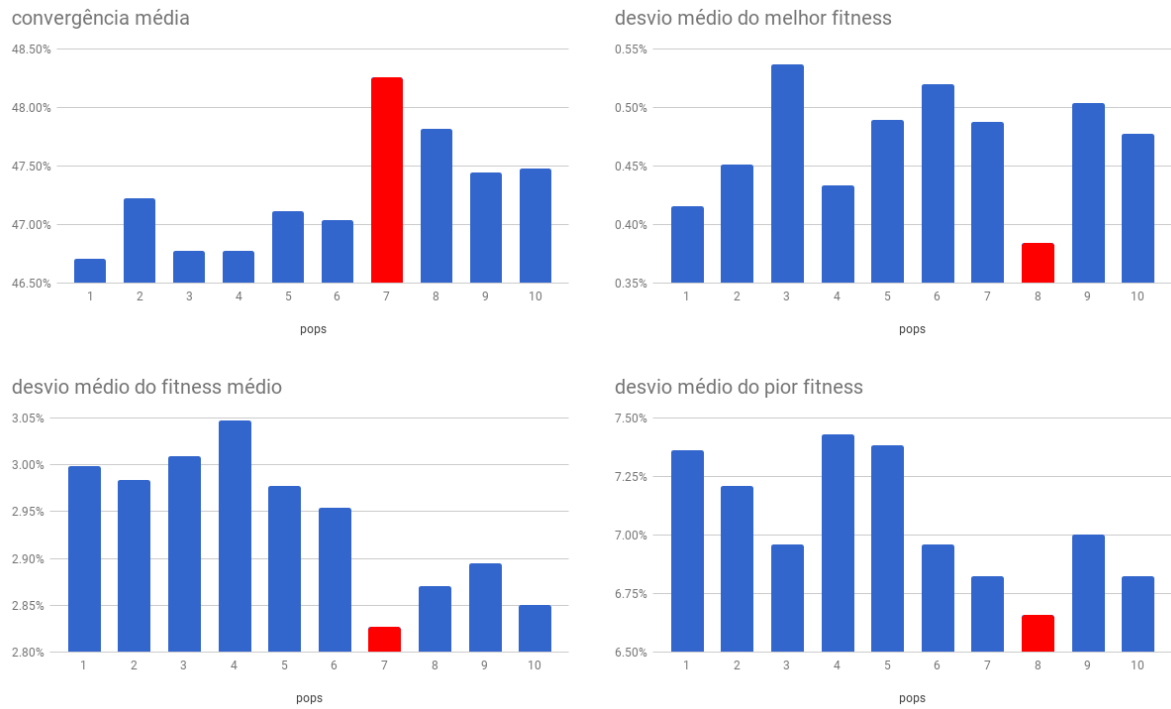


Figura 8 – Resultados agregados dos experimentos variando número de populações do AGMP. Os melhores resultados estão em destaque (vermelho).

Tabela 3 – Melhor *fitness* encontrado pelo AG e pelo AGMP com número de populações indicado.

grafo	AG	AGMP	populações
fft39	840	820	8
fft95	1620	1600	4
laplace36	1300	1270	8
random40	555	551	6

Tabela 4 – Estatísticas dos fitness encontrados em 10^5 execuções do AG e do AGMP de 8 populações.

	convergência	fitness médio	desvio padrão	pior fitness
AG	698	123.51	3.19	140
AGMP	1074	122.41	2.73	135

O AGMP apresentou um aumento significativo na convergência. O desvio padrão menor indica mais ocorrências de soluções em torno da média. Neste caso, o AGMP também superou o AG. Por fim, o AGMP obteve uma quantidade menor de *outliers*, de acordo com o boxplot da Figura 10.

4.2.3 Tempo de execução

Os experimentos foram executados num laptop com sistema operacional Ubuntu 16.04 e processador Intel Core i7-6500U, que possui 4 núcleos e clock de 2.50GHz. Os

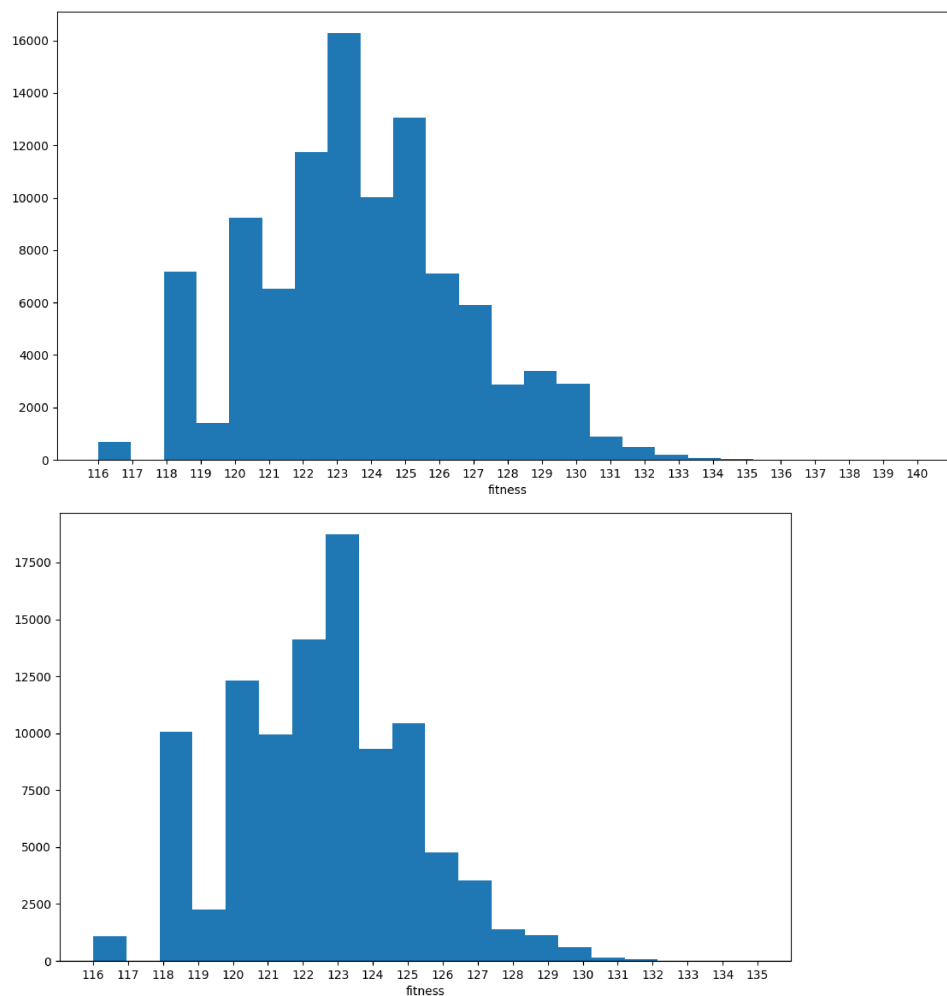


Figura 9 – Distribuição de fitness encontrados em 10^5 execuções do AG e do AGMP de 8 populações, respectivamente.

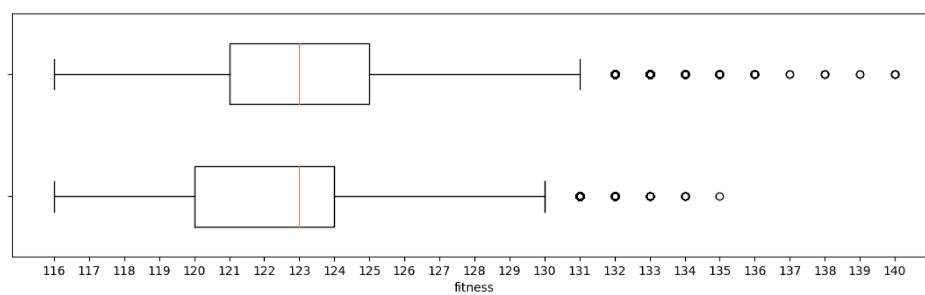


Figura 10 – Boxplot das distribuições de soluções do AG e do AGMP, respectivamente.

tempos de execução médios dos experimentos são dados na Tabela 5.

Assim, foi calculado o *Speedup* médio das configurações de AGMP (Figura 11), que tem pico em 4 populações e degrada em seguida, como previsto pela lei de Amdahl.

Tabela 5 – Tempo de execução médio dos experimentos, em milissegundos.

populações	1	2	3	4	5	6	7	8	9	10
tempo (ms)	40.33	20.52	19.93	15.59	23.30	21.93	23.67	23.00	22.89	23.26

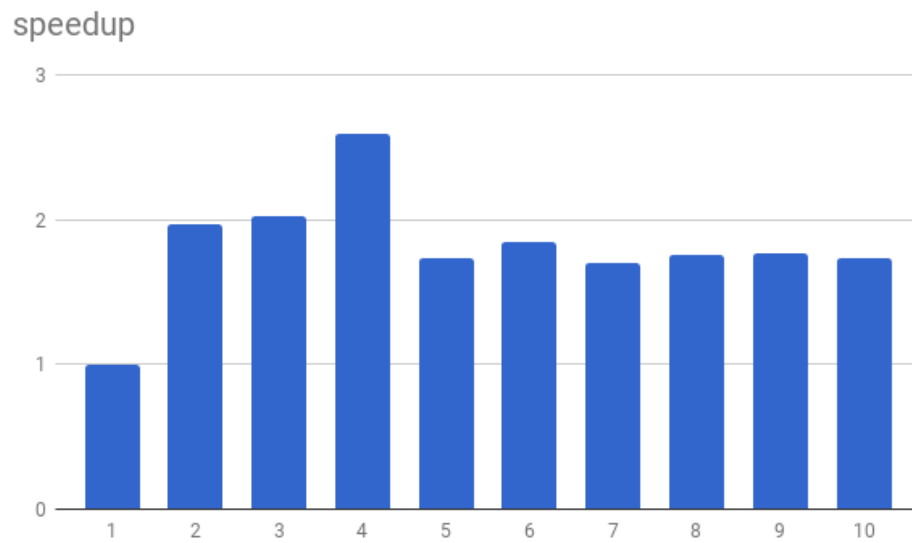


Figura 11 – *Speedup* por número de populações.

5 Conclusões e Trabalhos Futuros

Neste trabalho foi desenvolvido um AGMP para o problema de escalonamento de tarefas com a representação genética de dois vetores, que tem sido utilizada em trabalhos recentes. Foram implementados diversos métodos de *crossover* adequados para cada vetor e, para a forma de sua aplicação, foram identificadas na literatura três abordagens diferentes, que foram implementadas e comparadas.

O desempenho do AGMP foi medido variando o número de populações e comparado com o desempenho do AG serial em termos de qualidade agregada das soluções e tempo de execução.

Foi composto um conjunto de grafos para benchmark dos algoritmos, dividido entre grafos retirados da literatura e famílias de grafos baseadas em algoritmos reais. Para estas famílias, foram implementados scripts que geram grafos correspondentes à execução do algoritmo para diferentes tamanhos de entrada.

Os resultados suportam a hipótese de que AGMP é uma técnica que, dados parâmetros adequados, faz um melhor aproveitamento dos recursos computacionais disponíveis em comparação com o AG serial. Isto se dá de duas formas:

- Produção mais consistente de soluções quase ótimas para uma dada quantidade de avaliações de *fitness*.
- Melhoria de desempenho através da execução nos múltiplos núcleos dos processadores atuais.

Trabalhos futuros incluem:

- Realizar experimentos complementares, envolvendo escalonamento com alocação para arquiteturas com número de processadores diferentes, visto que o atual trabalho se limitou a arquiteturas de quatro processadores; comparar os resultados do AGMP com resultados da literatura; ampliar o repositório de benchmark com base em algoritmos paralelos de relevância; avaliar um número maior de configurações para AG e AGMP, incluindo as combinações possíveis de métodos de *crossover*, além de outros métodos para seleção e reinserção; avaliar *Speedup* em máquinas com maior número de núcleos de processamento; analisar a convergência dos indivíduos ao longo das gerações.
- Refinar o AGMP, investigando diferentes taxas de migração, frequências de migração, e topologias de comunicação. Também avaliar a dependência entre ajustes dessa

parametrização e subconjuntos dos grafos do benchmark, como famílias específicas ou grafos com número de vértices semelhante.

- Investigar novas formas de implementação e modelos de paralelização de AG. Por exemplo, experimentos em cluster, AGs paralelos de fina granularidade baseados em vizinhança, e implementação para unidades de processamento gráfico (GPU).
- Analisar a representatividade das formas de representação genética em relação às soluções alcançáveis no espaço de busca.
- Implementar variações do escalonamento de tarefas, e.g. sistemas com processadores heterogêneos, com processadores em redes sem conexão total entre nós.

Referências

- ALBA, E.; TROYA, J. M. A survey of parallel distributed genetic algorithms. *Complex.*, John Wiley & Sons, Inc., New York, NY, USA, v. 4, n. 4, p. 31–52, mar. 1999. ISSN 1076-2787. Citado 2 vezes nas páginas 8 e 15.
- CANTÚ-PAZ, E. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, v. 10, n. 2, p. 141–171, 1998. Citado 4 vezes nas páginas 15, 16, 17 e 23.
- CANTÚ-PAZ, E.; GOLDBERG, D. E. Are multiple runs of genetic algorithms better than one? In: CANTÚ-PAZ, E. et al. (Ed.). *Genetic and Evolutionary Computation — GECCO 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 801–812. ISBN 978-3-540-45105-1. Citado na página 16.
- CARNEIRO, M. G. *Abordagens baseadas em autômatos celulares síncronos para o escalonamento estático de tarefas em multiprocessadores*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, 2012. Citado na página 18.
- CHITRA, P.; RAJARAM, R.; VENKATESH, P. Application and comparison of hybrid evolutionary multiobjective optimization algorithms for solving task scheduling problem on heterogeneous systems. *Applied Soft Computing*, v. 11, n. 2, p. 2725 – 2734, 2011. ISSN 1568-4946. The Impact of Soft Computing for the Progress of Artificial Intelligence. Citado 5 vezes nas páginas 12, 13, 14, 15 e 19.
- CORREA, R. C.; FERREIRA, A.; REBREYEND, P. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, v. 10, n. 8, p. 825–837, Aug 1999. ISSN 1045-9219. Citado 5 vezes nas páginas 12, 13, 14, 22 e 25.
- COSNARD, M. et al. Parallel gaussian elimination on an mimd computer. *Parallel Computing*, v. 6, n. 3, p. 275 – 296, 1988. ISSN 0167-8191. Citado na página 19.
- GEHRING, H.; BORTFELDT, A. A parallel genetic algorithm for solving the container loading problem. v. 9, p. 497 – 511, 07 2002. Citado 2 vezes nas páginas 16 e 17.
- GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675. Citado 5 vezes nas páginas 8, 11, 12, 13 e 14.
- HAN, K.-H. et al. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. [S.l.: s.n.], 2001. v. 2, p. 1422–1429 vol. 2. Citado 2 vezes nas páginas 16 e 17.
- HOU, E. S. H.; ANSARI, N.; REN, H. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, v. 5, n. 2, p. 113–120, Feb 1994. ISSN 1045-9219. Citado 6 vezes nas páginas 8, 10, 12, 13, 14 e 22.

- HWANG, R.; GEN, M.; KATAYAMA, H. A comparison of multiprocessor task scheduling algorithms with communication costs. *Computers & Operations Research*, v. 35, n. 3, p. 976 – 993, 2008. ISSN 0305-0548. Part Special Issue: New Trends in Locational Analysis. Citado 3 vezes nas páginas 12, 13 e 14.
- JIANG, Y.; SHAO, Z.; GUO, Y. A dag scheduling scheme on heterogeneous computing systems using tuple-based chemical reaction optimization. v. 2014, p. 404375, 06 2014. Citado na página 19.
- KAUR, K. et al. Heuristics based genetic algorithm for scheduling static tasks in homogeneous parallel system. *International Journal of Computer Science and Security (IJCSS)*, v. 4, n. 2, 1999. Citado 4 vezes nas páginas 12, 13, 14 e 22.
- KWOK, Y.-K.; AHMAD, I. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 47, n. 1, p. 58–77, nov. 1997. ISSN 0743-7315. Citado 5 vezes nas páginas 12, 13, 14, 16 e 17.
- KWOK, Y.-K.; AHMAD, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 31, n. 4, p. 406–471, dez. 1999. ISSN 0360-0300. Citado na página 10.
- MORADY, R.; DAL, D. A multi-population based parallel genetic algorithm for multiprocessor task scheduling with communication costs. In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.: s.n.], 2016. p. 766–772. Citado 11 vezes nas páginas 12, 13, 14, 15, 16, 17, 18, 19, 22, 25 e 26.
- MÜHLENBEIN, H.; SCHOMISCH, M.; BORN, J. The parallel genetic algorithm as function optimizer. *Parallel Comput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 17, n. 6-7, p. 619–632, set. 1991. ISSN 0167-8191. Citado 2 vezes nas páginas 16 e 17.
- OLTEANU, A.; MARIN, A. Generation and evaluation of scheduling dags: How to provide similar evaluation conditions. *Computer Science Master Research*, v. 1, n. 1, 2011. ISSN 2247-5575. Citado na página 19.
- OMARA, F. A.; ARAFA, M. M. Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*, v. 70, n. 1, p. 13 – 22, 2010. ISSN 0743-7315. Citado 7 vezes nas páginas 12, 13, 14, 19, 22, 23 e 25.
- QI, J. G.; BURNS, G. R.; HARRISON, D. K. The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, v. 16, n. 8, p. 609–615, Jul 2000. ISSN 1433-3015. Citado 2 vezes nas páginas 16 e 17.
- SRINIVASA, K. G.; VENUGOPAL, K. R.; PATNAIK, L. M. A self-adaptive migration model genetic algorithm for data mining applications. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 177, n. 20, p. 4295–4313, out. 2007. ISSN 0020-0255. Citado 2 vezes nas páginas 16 e 17.
- WANG, L. et al. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, v. 47, n. 1, p. 8 – 22, 1997. ISSN 0743-7315. Citado 4 vezes nas páginas 12, 13, 14 e 19.

WU, M. Y.; GAJSKI, D. D. Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, v. 1, n. 3, p. 330–343, Jul 1990. ISSN 1045-9219. Citado na página [19](#).

XU, Y.; LI, K.; HU, J. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. v. 270, p. 255–257, 06 2014. Citado na página [19](#).

XU, Y. et al. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, v. 270, p. 255 – 287, 2014. ISSN 0020-0255. Citado 4 vezes nas páginas [12](#), [13](#), [14](#) e [15](#).

YAO, J.; KHARMA, N.; GROGONO, P. Bi-objective multipopulation genetic algorithm for multimodal function optimization. *IEEE Transactions on Evolutionary Computation*, v. 14, n. 1, p. 80–102, Feb 2010. ISSN 1089-778X. Citado 2 vezes nas páginas [16](#) e [17](#).