UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Thiago Fernandes Soares

Algoritmos evolutivos e modelo TIG para escalonamento de processos em ambientes distribuídos

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Thiago Fernandes Soares

Algoritmos evolutivos e modelo TIG para escalonamento de processos em ambientes distribuídos

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: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2018

Agradecimentos

Agradeço primeiramente a Deus por ter me dado forças, aos meus pais por terem me ajudado ao longo de todo o meu caminho de formação e a minha irmã por sempre ter me apoiado ao longo desses anos. Também agradeço ao Marcos Paulo Vilela por ter me ajudado e ensinado durante toda a minha graduação. Agradeço ao Arthur Moreira por todas as conversas e companheirismo dos últimos meses. Agradeço também ao Vitor Carvalho pelas dicas e ensinamentos durante minha graduação. Por fim agradeço ao Professor Paulo Henrique Ribeiro Gabriel por ter aceitado ter sido meu orientador e por todo o conhecimento e paciência para lidar com os problemas que encontramos.

Resumo

O escalonamento de processos se tornou uma tarefa fundamental para o melhor desempenho e utilização dos sistemas computacionais distribuídos. Muitas soluções têm sido propostas para tratar esse problema, dentre as quais se destacam algoritmos baseados em heurísticos. Os algoritmos genéticos (AGs) são um ramo dos algoritmos heurísticos que adotam conceitos biológicos para otimizar funções. No caso do problema de escalonamento, têm sido consideradas métricas de desempenho como *makespan* (custo máximo) e *flowtime* (custo total). Este trabalho propõe um AG para minimização do custo total, tendo como base um modelo de interação entre processos (do inglês, *Task Interaction Graph*, TIG). A solução encontrada pelo AG foi comparada com valores ótimos dados pelo solver IBM ILOG CPLEX. Além do custo total, foi analisado o valor do *makespan*, de modo a observar como esse se comporta com a minimização deste *flowtime*. Resultados mostram uma boa qualidade das soluções encontras pelo AG quando comparado aos valores ótimos.

Palavras-chave: Escalonamento de processos, Sistemas distribuídos, Algoritmos genéticos, Modelo TIG.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Sistemas computacionais distribuídos ganharam grande espaço na área de computação nas últimas décadas, sendo que grandes empresas multinacionais, como Google, IBM e Microsoft, vêm trazendo grandes avanços tecnológicos, principalmente pela necessidade de armazenamento de dados em locais distribuídos pelo mundo (DONG; AKL, 2006; STOCKINGER, 2007).

Os sistemas distribuídos sofreram grandes avanços com o passar dos anos e, com isso, se tornaram complexos, resultando em diversos problemas como: *i*) necessidade de adaptar um novo paradigma de desenvolvimento de software; *ii*) consistência dos dados; *iii*) segurança; *iv*) disponibilidade do sistema; *v*) escalonamento de tarefas. Como foco do trabalho temos o problema de escalonamento de processos (DONG; AKL, 2006).

Dentre os problemas descritos anteriormente, destaca-se o problema de escalonamento de processos (ou tarefas computacionais). Esse problema consiste em determinar uma alocação de tarefas sobre os processadores que estão no sistema distribuído (DONG; AKL, 2006; CASAVANT; KUHL, 1988). As soluções de escalonamento tomadas influenciam diretamente no desempenho de um sistema distribuído procurando satisfazer um conjunto de objetivos. Entre esses objetivos podemos citar a redução de tempo de máximo execução (ou *makespan*) de aplicações, a maximização do uso de recursos computacionais, dentre outros (DONG; AKL, 2006).

Na literatura, o problema de escalonamento de processos é caracterizado como pertencente a classe de problemas intratáveis (NP-Completo), ou seja, a busca por uma solução ótima em tempo polinomial é inviável (GAREY; JOHNSON, 1979). Pode-se demonstrar que o espaço de soluções do problema cresce exponencialmente em relação ao número de processos e recursos. Considerando x o total de computadores e y o total de processos, o total de soluções para o problema de escalonamento é da ordem de x^y (GABRIEL, 2013).

Neste trabalho, os processos a serem escalonados são modelados utilizando um *Task Interaction Graph* (TIG) com pesos. Em um TIG, tem-se um grafo onde os vértices correspondem às tarefas e as arestas correspondem ao volume de dados transferido entre as tarefas. Nesse modelo, não há precedência entre as tarefas, ou seja, as comunicações entre as tarefas podem ocorrer a qualquer momento ou intermitentemente ao longo das execuções das respectivas tarefas (UÇAR *et al.*, 2006; KAYA; UÇAR, 2009).

Devido a complexidade do problema de escalonamento de processos, várias propostas têm sido apresentadas, tendo como base diferentes heurísticas ou métodos aproximados (GRAHAM R., 1979; DONG; AKL, 2006; XHAFA; ABRAHAM, 2010). Os algo-

ritmos aproximados tentam encontrar uma solução próxima à da solução ótima, porém na grande maioria dos casos esses algoritmos tendem a ser de difícil projeto, demandando o desenvolvimento de modelos computacionais mais elaborados e também provas de aproximação (WILLIAMSON; SHMOYS, 2011).

Os algoritmos heurísticos buscam soluções satisfatórias, porém não apresentam qualquer fator de qualidade quanto a solução ótima. Contudo os algoritmos baseados em heurísticas têm menor dificuldade em seu desenvolvimento, podendo ser reutilizados em outros problemas (BLUM; ROLI, 2003). Existem diversos algoritmos heurísticos que obtém bons escalonamentos em sistemas distribuídos (CHAPIN, 1996). Os algoritmos genéticos (AGs) têm recebido destaque na literatura dentre essas heurísticas (ABRAHAM; BUYYA; NATH, 2000; BRAUN *et al.*, 2001; ARROYO; VIANNA, 2007; CARRETERO; XHAFA; ABRAHAM, 2007; XHAFA *et al.*, 2011).

Os algoritmos genéticos utilizam diversos conceitos oriundos das ciências biológicas como seleção natural, sobrevivência do melhor indivíduo, mutação, entre outros conceitos, assim melhorando a busca de uma melhor solução (LACERDA; CARVALHO; LUDERMIR, 2002). Os AGs desenvolvidos para solucionar o problema de escalonamento de tarefas geralmente são projetados com o objetivo de minimizar o tempo total entre o início e o fim do escalonamento (métrica chamada de *makespan*). Outro objetivo bastante usado é a minimização dos tempos totais de conclusões de cada tarefa (métrica chamada de *flowtime*) (XHAFA; ABRAHAM, 2010).

1.1 Objetivos

O objetivo deste trabalho é utilizar o modelo TIG para modelar o problema de escalonamento de processos em sistemas distribuídos e implementar um AG para resolver esse problema, avaliar as métricas bastante usadas na literatura: *makespan* e *flowTime*. Para isso, são realizados experimentos com diferentes cenários computacionais.

As duas métricas, *makespan* e *flowtime*, têm sido bastante utilizadas na literatura, porém poucos trabalhos analisam as duas ao mesmo tempo tentando alcançar um equilíbrio entre as duas métricas, a fim de, por exemplo, descobrir se realmente vale a pena usar várias máquinas ou se é melhor usar uma única máquina.

1.2 Resultados

O AG foi então executado com diferentes instâncias, geradas aleatoriamente. O objetivo adotado foi a minimização do custo total, visto que essa métrica é bastante utilizada quando se refere à otimização com modelos TIG. Além disso, os valores ótimos de custo total já eram conhecidos para as instâncias utilizadas.

A análise dos resultados permite concluir que o AG conseguiu bons resultados em comparação com o resultado ótimo obtido por Gabriel (2013), porém com um tempo de execução pior. Também foi analisada a relação entre o *makespan* e o custo total, sendo que a minimização do custo total teve um impacto negativo no *makespan*.

1.3 Organização do Trabalho

O restante desta monografia está organizada como descrito a seguir. O Capítulo 2 apresenta os principais conceitos envolvidos neste trabalho, ou seja, o escalonamento de processos e os algoritmos genéticos. O Capítulo 3 apresenta trabalhos relacionados a esta pesquisa. No Capítulo 4 é descrito o desenvolvimento do AG, discutindo seus operadores e o modelo matemático adotado na avaliação das soluções de escalonamento. Os resultados são apresentados e discutidos no Capítulo 5. No Capítulo 6, finalmente, são apresentadas as conclusões e descritos possíveis trabalhos futuros.

2 Referencial Teórico

Neste capítulo são apresentados e discutidos conceitos importantes para o entendimento deste trabalho. Na Seção 2.1 é formalizado o problema de escalonamento de processos. Já na Seção 2.2, explica-se mais a fundo os conceitos de algoritmos genéticos, que serão utilizados para se obter soluções neste trabalho.

2.1 Escalonamento de Processos

Escalonamento de processos é uma atribuição de um conjunto de *processos* sobre um conjunto de *elementos de processamento* (CHAPIN, 1996). Essa atribuição deve ser guiada por meio de algum critério de otimização que avalia se o escalonamento feito é uma solução adequada.

Para o processo de escalonar tarefas, Casavant e Kuhl (1988) definem três componentes básicos: *i*) os *consumidores* que correspondem aos processos que utilizam dos meios computacionais; *ii*) os *recursos* computacionais, normalmente processadores, dispositivos de armazenamentos de arquivos, memória e outras propriedades computacionais; e *iii*) a *política* ou *algoritmo* de *escalonamento*. Os consumidores consomem os recursos computacionais e os algoritmos determinam quais recursos os consumidores irão consumir de acordo com a disponibilidade dos recursos.

Diversos algoritmos estão sendo propostos na literatura (DONG; AKL, 2006), muitos desses algoritmos simplificam o problema diminuindo o espaço de busca para cenários específicos. Esse problema já foi abortado de diversas maneiras na literatura.

2.1.1 Formalização do Problema de Escalonamento de Processos

Neste trabalho é utilizada a formalização do problema de escalonamento descrita por (GABRIEL, 2013). Primeiramente, seja um conjunto A de aplicações paralelas e uma função $tam(a_k)$ que computa o número de processos contidos em uma aplicação $a_k \in A$. Cada aplicação contém diversos processos. Considera-se um conjunto P que contém todos os processos de todas as aplicações, como sequência, o número de elementos de P é dado por $|P| = \sum_{k=1}^{|A|} tam(a_k)$.

Cada processo $p_i \in P$ possui demandas específicas em relação aos recursos, como a utilização de CPU, entrada e saída em discos rígidos, operações em redes e acesso à memória. Os processos em P são alocados em um conjunto de diferentes computadores, denotado por V, sendo que em cada $v_j \in V$ pode conter diferentes capacidades como ta-

manho e tempo de acesso a memória principal, capacidade, velocidade de processamento, entre outras características.

Os computadores em V são conectados por diferentes canais de comunicação. Podemos utilizar um grafo não-dirigido $G = (V,E)$ para representar esse canal utilizado, em que cada vértice V representa um computador e o conjunto E de arestas contém os canais de comunicações entre pares de vértices, ou seja, $(v_i,v_j) \in E$. Esses canais de comunicação têm propriedades específicas como largura de banda e latência de rede.

Garey e Johnson (1979) formalizaram esse problema e demonstraram sua complexidade. Esses autores afirmam que achar a solução ótima para o problema de escalonamento de processos é um problema NP-Difícil, mesmo considerando dois computadores sem comunicação.

2.1.2 Métricas de Desempenho

Para avaliar as soluções do problema de escalonamento de processos temos métricas a serem utilizadas. Xhafa e Abraham (2010) descreve métricas para esse problema como o *makespan*, o *flowtime*, a utilização de recursos (*utilization*), o balanceamento de carga (*load balancing*), entre outras.

As duas primeiras, e mais utilizadas, métricas podem ser definidas em função do tempo que cada computador permanece ocupado para executar os processos atribuídos a ele. Esse tempo é denotado por C_j, para $j = 1,2,\ldots,|V|$.

O *makespan* é o intervalo de tempo total entre o início e o término da execução de um processo $p_i \in P$ (DONG; AKL, 2006). Esse tempo pode ser obtido considerando o maior valor C_j, ou seja, o *makespan* é definido por:

$$M = \max_{1 \leq j \leq |v|} C_j \tag{2.1}$$

Já a métrica *flowtime* é definida por Carretero, Xhafa e Abraham (2007) como sendo a soma de todos os tempos gastos para executar todos os processos $p_i \in P$, sendo definido como:

$$F = \sum_{j=1}^{|V|} C_j \tag{2.2}$$

É possível observar que essas duas métricas são de minimização, ou seja, deseja-se obter um escalonamento que reduza o valor de *makespan* e/ou *flowtime*. Seus valores são dados em segundos de execução. Devido à complexidade do problema, tem-se explorado algoritmos baseados em heurísticas, como os algoritmos genéticos.

2.2 Algoritmo Genético

Os algoritmos genéticos (AGs) são métodos que visam achar a solução ótima de um problema inspirado na evolução dos seres vivos. Os AGs foram primeiramente apresentados por John Holland (alguns trabalhos anteriores já sugeriam essa ideia), porém só foram popularizados por David Goldberg. Os AGs são de uma classe particular de algoritmos chamados algoritmos evolutivos (AEs) que utilizam a biologia como modelo. Segundo Ridley (2006) "*Quanto mais apto o indivíduo está, maior será a probabilidade de que o mesmo sobreviva*".

Os AGs utilizam conceitos da biologia como: genes, cromossomos, hereditariedade, seleção natural, reprodução e mutação. A maioria desses conceitos foram inspirados nos estudos do inglês Charles Darwin e estendidos por outro trabalhos ao longo do Século XX (RIDLEY, 2006).

Nas seções a seguir, esses conceitos serão explicados, bem como sua importância para o desenvolvimento do AG. Em seguida, será mostrado um AG típico detalhando cada etapa.

2.2.1 Genes

Em Biologia, os genes são parte fundamental da hereditariedade (SETUBAL; MEIDANIS, 1997). Os genes são formados por um sequência específica de biomoléculas que contém a informação genética de cada indivíduo. Os genes modelados na computação são uma parte do indivíduo onde está a informação de acordo com a representação do problema apresentado. Essa representação pode ser representada por números inteiros (o mais comum), ponto flutuante ou binário (GOLDBERG, 1989; LACERDA; CARVALHO; LUDERMIR, 2002).

2.2.2 Cromossomos

Os cromossomos, nos seres vivos, são longas sequência de DNA que contém diversos genes com funções de cada indivíduo (SETUBAL; MEIDANIS, 1997). Na área de computação evolutiva, os cromossomos são representados por uma estrutura de dados que pode ser um vetor, uma cadeia de bits ou mesmo árvores, sendo que cada cromossomo representa uma solução diferente do problema (LACERDA; CARVALHO; LUDERMIR, 2002). Na Figura 1, é possível observar um exemplo mostrando um cromossomo e seus genes.

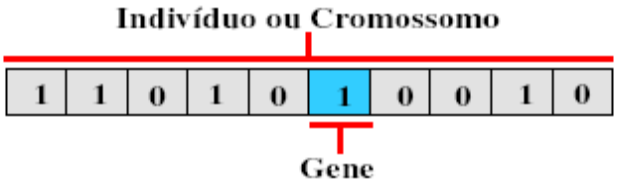

Figura 1 – Exemplo de Cromossomo

2.2.3 Hereditariedade

A hereditariedade é um conceito o qual um ser vivo transmite os seus genes (suas informações genéticas) para seus descendentes (filhos) por meio da reprodução (RIDLEY, 2006). Na computação evolutiva, a hereditariedade é representada pela troca de genes entre cromossomos de dois indivíduos diferentes, para então formar novos indivíduos, ou seja, novas soluções (GOLDBERG, 1989).

2.2.4 Seleção

Na natureza, a seleção ocorre de maneira natural, sendo que os seres mais adaptados a um ambiente sobrevivem e são selecionados para o cruzamento que então transmitem seus genes para seus descendentes (SETUBAL; MEIDANIS, 1997; RIDLEY, 2006). Nos AGs, existem métodos para fazer essa seleção no meio da *população* (todos os cromossomos disponíveis). Lacerda, Carvalho e Ludermir (2002) listam alguns desses métodos:

Roleta: Neste método, os indivíduos são escolhidos de maneira aleatória, sendo que os melhores indivíduos tem uma probabilidade maior de serem escolhidos.

Torneio: São escolhidos n indivíduos da população de maneira aleatória e então e feito um "torneio", onde os melhores indivíduos são escolhidos para o cruzamento.

Torneio Estocástico: Este método é uma combinação do método da roleta e do método do torneio; primeiramente, são sorteados n indivíduos aleatórios pelo método da roleta e, então, faz-se um torneio com esses indivíduos, sendo que escolhe-se os melhores para cruzamento.

Existem outros métodos de seleção que podem ser utilizados nos AGs. A seleção é bastante importante nos AGs, pois, a partir de um método escolhido, pode facilmente ignorar completamente os indivíduos avaliados como piores e gerando uma população única (repetindo várias vezes os melhores indivíduos) sem conseguir chegar em uma solução ótima porque os indivíduos são todos iguais (GOLDBERG, 1989).

2.2.5 Reprodução

A reprodução é como geramos novos indivíduos. A reprodução começa na seleção dos indivíduos e então há a troca de genes e no final os indivíduos gerados podem sofrer uma mutação, a seleção já foi explicada na Seção 2.2.4.

A recombinação de genes, também conhecido como *crossover*, tem a função de misturar os genes de dois indivíduos que foram selecionados e, então, gerar descendentes. Para o *crossover*, existem também diferentes métodos para tipos diferentes de dados como cadeias binárias, inteiros e ponto flutuante.

2.2.6 Mutação

Como parte final da reprodução, os indivíduos gerados poderão sofrer mutação de acordo com uma probabilidade. A mutação é a transformação de um gene (ou uma parte dos genes) do indivíduo. Por exemplo, em uma cadeia binária, um gene com a informação 1 pode vir a ser substituído por 0 e vice-versa. A aplicação da mutação é feita para não permitir a estagnação da busca pela melhor solução sem, contudo, tornar o processo altamente aleatório.

2.2.7 Função de avaliação

A função de avaliação, ou aptidão, é utilizada para avaliar um indivíduo, classificando-o como adequado ou não (LACERDA; CARVALHO; LUDERMIR, 2002). As funções de avaliação podem ser de maximização (os indivíduos com maiores valores são os melhores indivíduos) ou de minimização (os indivíduos com menores valores são os melhores indivíduos).

A função de avaliação pode ser adaptada para cada tipo de problema. Toda a população inicial é avaliada e os filhos gerados dessa população também são avaliados pela mesma função de avaliação, para então decidir quem irá continuar na população para a próxima geração e então continuar este ciclo até alcançar o critério de parada.

2.2.8 Estrutura de um algoritmo genético

Com os conceitos apresentados anteriormente permitem a construção de um AG, com alguns passos: *i*) Criação da população inicial; *ii*) cálculo das aptidões; *iii*) seleção de pais; *iv*) cruzamento; *v*) mutação; *vi*) critério de parada. Cada um desses passos são explicados a seguir:

1. Criação da população inicial:
   A população inicial, na maioria dos casos, e feita de forma aleatória onde temos um

número pré-definido de indivíduos que serão o que chamamos de população, sendo que essa população não poderá ser maior ou menor do definido.

2. Cálculo das aptidões:
   Após termos a população inicial criada temos que avaliar essa população para que possamos fazer a seleção. Avaliamos tanto a população inicial quanto os filhos gerados das populações seguintes, lembrando que os filhos gerados também serão avaliados e então será verificado quais indivíduos tanto dos pais quantos dos filhos irão para a próxima geração, pois não podemos ter mais indivíduos na população que na população inicial.

3. Seleção de pais:
   Com a população criada e avaliada podemos selecionar os pais para começar os cruzamentos. Na seleção de pais podemos usar os métodos de seleção apresentados anteriormente, selecionamos indivíduos até termos o número de filhos que queremos criar.

4. Cruzamento:
   Após selecionar os pais para o cruzamento, eles são cruzados por algum método de cruzamento para gerar seus filhos, que são novas soluções do problema, para que possamos passar para as mutações.

5. Mutação:
   De acordo com a taxa de mutação alguns indivíduos irão sofrer a mutação de modo aleatório, ou seja, não se sabe quais indivíduos irão sofrer a mutação. No final dessa fase, teremos todos os filhos gerados. Basta, agora, avaliá-los com função de avaliação e analisar a nova população para verificar se paramos com o AG ou se continuamos começando o ciclo novamente.

6. Critério de parada:
   Para verificarmos se vamos ou não finalizar o AG, precisamos ter um ou mais critérios de paradas. Geralmente o critério de parada e quanto atingimos um número x de épocas, sendo que cada ciclo do AG corresponde a uma época.

Temos um exemplo de AG que segue esses procedimentos na Figura 2, na qual pode-se observar os passos descritos anteriormente.

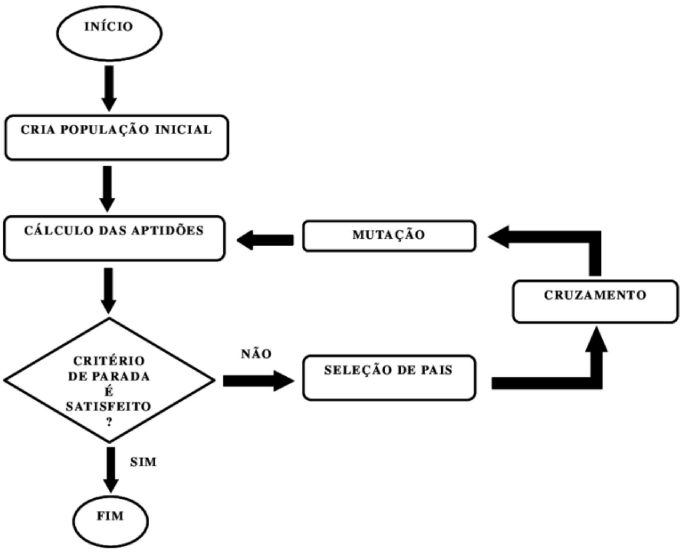

Figura 2 – Exemplo de AG.

3 Trabalhos Correlatos

Neste capítulo são apresentados alguns trabalhos mais conhecidos e estudados da literatura no problema de escalonamento de processos. Para se otimizar o problema de escalonamento, deve-se, inicialmente, adotar um modelo matemático que permita uma boa cobertura do espaço de busca. Nesta monografia, esse problema é modelado com base nas seguintes características: processos são modelados considerando-se um grafo de interações ponderado (do inglês, *task interaction graph*, TIG).

Em um TIG, os vértices correspondem aos processo e as arestas correspondem à troca de mensagens entre tais processos. Além disso, não há precedência entre os processos, ou seja, arestas representam eventos de comunicação que podem ocorrer a qualquer momento ou de maneira intermitente ao longo de sua execução. (UÇAR *et al.*, 2006; KAYA; UÇAR, 2009). Assim, um processo passa o controle para outro e espera resposta (NORMAN; THANISCH, 1993; UÇAR *et al.*, 2006).

O modelo TIG foi, originalmente, descrito por Stone (1977) de modo a representar a execução de processos persistentes (KAYA; UÇAR, 2009). Nesse modelo, a qualquer momento, um processo é executado em um único processador, podendo trocar mensagens a qualquer momento. Posteriormente, Bokhari (1981) demonstrou que o escalonamento de TIG é NP-completo para dois ou mais elementos de processamento.

Considerando o modelo TIG, Kafil e Ahmad (1997) projetou uma heurística A^* para um escalonamento ótimo. Porém, este modelo era limitado para redes homogêneas, nas quais o canal de comunicação é idêntico entre todos os computadores. Além disso, A^* proposto também não lidava com mais que 20 processos. Tom e Murthy (1999) descreveu um modelo mais geral, sugerindo a heterogeneidade da rede. Ele também considerou uma heurística A^*, mas apresentou resultados para ambientes muito pequenos (de 3 a 5 computadores e de 4 8 processos). Os autores também consideraram que os tempos de execução e comunicação de todas as tarefas em todos os processadores já eram pré-conhecidos; consequentemente, o escalonamento requer um passo de extra de pré-processamento.

Mais tarde, Ma, Chen e Chung (2004) propôs um método *branch-and-bound* para achar a solução ótima do problema de escalonamento. Foi modelada uma rede heterogênea considerando o atraso nas comunicações entre processos; porém, assim como Kafil e Ahmad (1997), foram considerados apenas para computadores homogêneos.

Kaya e Uçar (2009) também projetou um algoritmo A^* para um escalonamento exato. Essa proposta teve um modelo mais geral do que os apresentados anteriormente, considerando redes e computadores heterogêneos. Os autores adotaram uma matriz de tempo de execução estimado (do inglês *Estimated Time of Completion*, ETC) que des-

creve o custo envolvido na execução de processos. Além disso, o custo da comunicação é modelado considerando a chamada "distância" da rede, ou seja, dados dois computadores, a distância é definida como o inverso da largura de banda da rede que liga os dois computadores. O modelo proposto é quadrático em termos de custos de comunicação. Kaya e Uçar (2009) também considerou instâncias relativamente pequenas, com no máximo 307 processos para 8 computadores.

Outros métodos de solução também foram considerados na literatura. Uçar *et al.* (2006) revisou várias heurísticas, e as organizou em três grupos: *i*) meta-heurística; *ii*) teoria dos grafos; e *iii*) busca estado-espaço. Também propuseram duas novas heurísticas, baseadas em agrupamento, ordenação de tarefas e refinamento (ARAFEH; DAY; TOUZENE, 2008). Foram obtidos bons escalonamentos (em alguns casos, a solução ótima foi encontrada), porém a estratégia era limitada a redes homogêneas. Também merece destaque o trabalho de Arafeh, Day e Touzene (2008), que propôs um esquema de partição de gráfico multinível, considerando grandes instâncias e ambientes heterogêneos. Como limitação, a heurística proposta não garante soluções ótimas.

Finalmente, Gabriel (2013) projetou um modelo mais genérico, que considera separadamente as seguintes características do problema de escalonamento: *i*) carga de trabalho de processos, ou seja, o número de instruções requeridas pelas aplicações; *ii*) volume de comunicação, ou seja, o total de *bytes* trocados entre os processos; *iii*) capacidade dos computadores, i.e., o número de instruções que cada elemento de processamento por executar, por segundo; e *iv*) capacidade da rede, ou seja, o total de *bytes* que os canais de rede podem transmitir em um período de tempo. Ao separar esses componentes, o modelo torna-se mais flexível que outros; de fato, cada característica pode ser alterada individualmente, de modo a analisar seu impacto sobre os demais componentes do modelo. Posteriormente, Gabriel *et al.* (2018) geraram um conjunto de instâncias para o problema e implementaram o modelo proposto utilizando o solver IBM ILOG CPLEX (BLIEK; BONAMI; LODI, 2014), de modo a encontrar a solução ótima para tais instâncias.

4 Desenvolvimento

Este trabalho de conclusão de curso tem como principal objetivo a implementação de AG para o problema de escalonamento de processos, comparando os resultados com os encontrados pelo solver ILOG CPLEX. O AG proposto tem como meta a minimização do custo total, ou *flowtime*. Além disso, como objetivo secundário, estuda-se a correlação desse custo com o tempo máximo de execução, denominado *makespan*.

Neste capítulo, o modelo proposto por (GABRIEL, 2013) é descrito em mais detalhes. Posteriormente, o AG desenvolvido é descrito em detalhes, apresentando-se suas principais características.

4.1 Modelo Matemático

O modelo de otimização adotado neste trabalho foi o de Gabriel (2013), apresentado anteriormente. Este modelo leva em consideração as seguintes variáveis a seguir. Seja w uma vetor na forma:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{|P|} \end{bmatrix} \tag{4.1}$$

onde $w_1, \ldots, w_{|P|}$ são os totais de instruções de processamento (ou seja, a carga de trabalho de cada processo) que cada processo $p_i \in P$ possui para ser executado em todo seu ciclo de vida.

Seja um c um vetor na forma:

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{|V|} \end{bmatrix} \tag{4.2}$$

em que $c_1, \ldots, c_{|V|}$ é a capacidade de processamento que cada computador $v_j \in V$ possui, expressos em milhões de instruções por segundo (MIPS).

Tem-se, também, a matriz Γ que representa o canal de comunicação entre os com-

putadores e é representada no seguinte formato:

$$\Gamma = \begin{bmatrix} \Gamma_{1,1} & \dots & \Gamma_{1,|V|} \\ \Gamma_{2,1} & \dots & \Gamma_{2,|V|} \\ \vdots & \vdots & \vdots \\ \Gamma_{|V|,1} & \dots & \Gamma_{|V|,|V|} \end{bmatrix} \tag{4.3}$$

Nessa matriz, cada $\Gamma_{i,j}$ representa a capacidade ou largura de banda em *megabytes* por segundo, do canal de comunicação representado pela aresta $(v_i,v_j) \in E$. Essa matriz possui duas propriedades: *i*) Γ é uma matriz simétrica, ou seja, a rede que conecta v_i e v_j é a mesma que v_j e v_i; e *ii*) a comunicação de um computador com eles mesmo é representado por um valor alto já que a comunicação é feita através de um barramento interno do próprio computador, ou seja, a diagonal principal da matriz Γ é preenchida com valores altos.

A matriz L representa a comunicação entre os processos, uma matriz $|P| \times |P|$ na forma:

$$L = \begin{bmatrix} L_{1,1} & \dots & L_{1,|P|} \\ L_{2,1} & \dots & L_{2,|P|} \\ \vdots & \vdots & \vdots \\ L_{|P|,1} & \dots & L_{|P|,|P|} \end{bmatrix} \tag{4.4}$$

em que $L_{i,j}$ é o total de *bytes* a ser transferido do processo p_i para o processo p_j. Se o processo p_i não se comunica com o processo p_j então o valor que representa a comunição entre os processos será $L_{i,j} = 0$. A matriz não é simétrica, ou seja, se o processo p_i se comunica com o processo p_j, não significa que p_j se comunica com p_i.

Por fim, seja uma matriz S de escalonamento $|P| \times |V|$ que representa uma solução de escalonamento, mapeando os processos P sobre os computadores V, da seguinte maneira:

$$S = \begin{bmatrix} S_{1,1} & \dots & L_{1,|V|} \\ S_{2,1} & \dots & L_{2,|V|} \\ S_{3,1} & \dots & L_{3,|V|} \\ \vdots & \vdots & \vdots \\ S_{|P|,1} & \dots & L_{|P|,|V|} \end{bmatrix} \tag{4.5}$$

cada elemento dessa matriz S pode assumir apenas dois valores $\{0,1\}$. Nesta matriz cada linha terá apenas um elemento com o valor 1 e os demais 0, caso $S_{i,j} = 1$ então o processo p_i será processado pelo computador v_j, caso contrário se $S_{i,j} = 0$, então o processo p_i não será processado pelo computador v_j.

O objetivo deste trabalho é encontrar a melhor matriz de escalonamento S através de um AG, sem repetições de processos, ou seja, um processo será alocado para somente uma máquina. Para encontrar a matriz S, é utilizado a definição de atraso de execuções. Este atraso corresponde ao tempo em que cada recurso computacional permanece ocupado, a partir de uma solução de escalonamento, apresentado pela equação (4.6). Nessa equação temos dois componentes: $\rho^{CPU}(S)$ que representa os atrasos de processamento gerado por computadores em V, e $\rho^{comun}(S)$ que define os atrasos impostos pela comunicação entre os processos.

$$\rho(S) = \rho^{CPU}(S) + \rho^{comun}(S) \tag{4.6}$$

A partir desta equação, tem-se os atrasos, ou custos, envolvidos na execução, impostos por operações de processamento de CPU e comunicação entre processos.

O primeiro componente da equação (4.6) $\rho^{CPU}(S)$ é definido pela equação (4.7), nela consideramos a transposta de S com o produto do vetor w, na forma $S^t * w$, assim é obtido o total de instruções de processamento atribuídas a todos os computadores em V. E então ocorre uma divisão termo a termo pelo vetor de capacidade de cada computador, representado pelo vetor c, resultando no custo de execução dos processos atribuídos a cada computador, em segundos.

$$\rho^{CPU}(S) = \frac{S^t * w}{c} \tag{4.7}$$

O segundo componente da equação (4.6) $\rho^{Comun}(S)$ é definido pela equação (4.8), em que o produto da matriz transposta de S^t pela matriz L com o produto pela matriz S, na forma $S^t * L * S$, assim se obtém o total de bytes a ser transferido entre todos os processos em P. A divisão termo a termo pela matriz Γ resulta em outra matriz que representa o tempo consumido em todas as ocorrências de comunicação entre processos. Após calculado o produto pelo vetor unitário de tamanho $|V|$, denotado por $\mathbf{1}$, se obtém a soma de todos os atrasos de comunicação por computador.

$$\rho^{Comun}(S) = \frac{S^t * L * S}{\Gamma} * 1 \tag{4.8}$$

Juntando as equações anteriores (4.7,4.8), temos o vetor $\rho(S)$ representado pela equação (4.9) com todos os atrasos envolvidos na solução de escalonamento representado pela matriz S.

$$\rho(S) = \frac{S^t * w}{c} + \frac{S^t * L * S}{\Gamma} * 1 \tag{4.9}$$

Com essas definições descrevemos o algoritmo genético na seção a seguir.

4.2 Algoritmo Genético Proposto

Nesta seção, é descrito o algoritmo genético implementado neste trabalho, levando-se em consideração cada um de seus componentes.

4.2.1 Representação do indivíduo

Neste trabalho, cada indivíduo de um AG representa uma solução de escalonamento, ou seja, indica o mapeamento dos processos nos computadores. Foi adotado então a representação por inteiros, ou seja, cada solução consiste em um valor de inteiros. Assim, cada indivíduo foi representado por um vetor de tamanho $|P|$ em que cada gene tem valores de 0 até $|V-1|$. Isso significa que cada gene representa em qual computador $v_i \in V$ o processo $p_i \in P$ está escalonado. Essa representação tem como vantagem evitar que um mesmo processo seja escalonado sobre dois computadores diferentes; assim, tem-se apenas soluções de escalonamento factíveis (GABRIEL, 2013).

Figura 3 – Exemplo de solução representado por um indivíduo. Nesse caso, temos que: $(p_1 \rightarrow v_3), (p_2 \rightarrow v_2), (p_3 \rightarrow v_1), (p_4 \rightarrow v_4), (p_5 \rightarrow v_4), (p_6 \rightarrow v_1)$.

Analisando a representação da Figura 3 temos então que o processo P_1 está escalonado na máquina v_3, o processo p_2 está escalonado na máquina v_2, o processo p_3 está escalonado na máquina v_1, e assim por diante. Com essa representação é possível realizar todas as etapas do algoritmo genético, pois qualquer mudança de gene no indivíduo o mesmo continua sendo valido.

A população foi representada sendo um conjunto de indivíduos na forma vetorial, gerada de maneira aleatória. Com a representação da população e dos indivíduo definidos, o algoritmo genético possue seis etapas como descrito na seção 2.2.8: Inicialização, Avaliação, Seleção, Reprodução, Mutação e Reinserção. As próximas secções detalham como essas fases foram implementadas.

4.2.2 Inicialização

A inicialização do algoritmo consiste em criar todas as estruturas de dados necessárias para o funcionamento do AG. Como descrito na seção 4.1 dependemos de quatro componentes do modelo. Assim, o algoritmo inicializa as seguintes variáveis: dois vetores c e w e duas matrizes Γ e L com os valores contidos nos dados de entrada.

Apos a inicialização dessas variáveis, o algoritmo gera a população inicial de indivíduos feito por um vetor de estrutura, no qual será modificado no percorrer do processo evolutivo.

4.2.3 Avaliação

Após a definição da variáveis e da geração da população inicial, avaliamos esta população para termos sua aptidão. A avaliação realizada por meio de uma função objetivo consiste em quantificar o quão boa é cada solução (ou indivíduo). Para realizar essa operação é preciso criar uma matriz S, que representa o escalonamento de cada indivíduo. Considerando a Figura 3, a matriz S é gerada da seguinte maneira:

$$S = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.10}$$

Então com a matriz S é realizado o cálculo da função de avaliação (ou *fitness*), apresentado na equação 4.9. Assim, no final da avaliação, obtém-se o vetor ρ que representa os custos de execução de cada computador para cada indivíduo da população.

Após termos o vetor ρ, calculamos a aptidão de cada indivíduo, que na verdade é a soma dos elementos do vetor de ρ. Neste trabalho temos como objetivo minimizar essa aptidão,ou seja, encontrar um indivíduo com a menor aptidão.

4.2.4 Seleção

Com todos os indivíduos classificados, a seleção é realizada para escolher indivíduos para o cruzamento. Neste algoritmo genético foi adotado o método torneio de 3. Nesse método são sorteado três indivíduos aleatoriamente (x_1, x_2 e x_3), sendo que todos os indivíduos tem a mesma chance de serem selecionados.

Após a seleção desses três indivíduos olhamos para as aptidões dos mesmos, ou seja, olhamos as aptidões dos indivíduos x_1, x_2 e x_3 e as comparamos para escolher a menor entre as três (problema de minimização), então esse indivíduo de menor aptidão é o qual o método do torneio de 3 seleciona.

Então ao final do torneio teremos um indivíduo selecionado para o cruzamento. Para o cruzamento usamos o cruzamento paternal, portanto é necessário ter dois indivíduos para o cruzamento. Então fazemos o torneio duas vezes para gerar dois pais distintos.

4.2.5 Cruzamento

Após selecionarmos os dois pais (indivíduos) na seleção realizamos o cruzamento entre os indivíduos selecionados. O cruzamento tem objetivo de preservar os cromossomos dos pais para o novo indivíduo gerado. Neste algoritmo, foi realizado o cruzamento utilizando *crossover* uniforme, nele sorteamos uma máscara do mesmo tamanho do indivíduo onde seus valores serão atribuídos como 0 ou 1.

Este método gera dois filhos de acordo com a máscara sorteada, onde o primeiro filho irá seguir de acordo com a máscara, ou seja, o gene que estiver o valor 0 ele irá herdar o valor do primeiro pai para o seu gene naquela posição, caso o valor seja 1, então ele irá herdar o gene do segundo pai. O segundo filho irá considerar o valores trocados da máscara, ou seja, onde estiver 0 ele irá herdar o gene do segundo pai e onde estiver 1, ele irá herdar o gene do primeiro pai. Podemos ver um exemplo de dois pais para o cruzamento na figura 4 e de uma máscara aleatória na figura 5

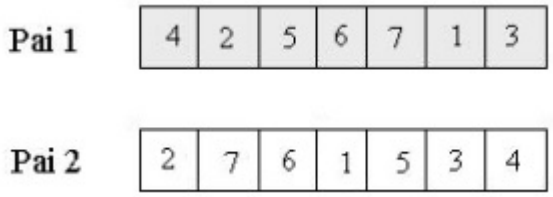

Figura 4 – Exemplo de pais para o cruzamento

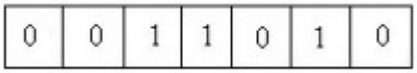

Figura 5 – Exemplo de máscara para o cruzamento

De acordo com a figura 4 e a figura 5, teremos dois filhos que herdarão os genes dos pais de acordo com a máscara sorteada, podemos então ver o resultado na figura 6.

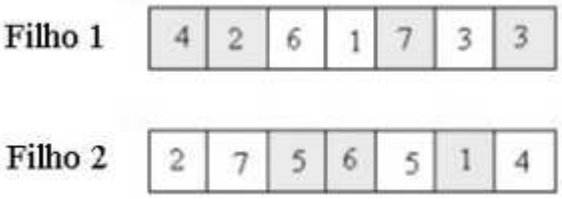

Figura 6 – Exemplo de Filhos gerados pelo cruzamento

Após gerado os dos filhos através do cruzamento eles poderão passar por uma mutação para então serem inseridos na nova população da próxima geração.

4.2.6 Mutação

A mutação é aplicada com uma probabilidade de 1% para que alguns filhos gerados no cruzamento tenha algum gene afetado. Neste algoritmo todos os genes do filho gerado tem a probabilidade de 1%, para que naquele gene tenha uma alteração, ou seja, para aquele processo p_i que estava escalonado na máquina $v_j \in V$ agora será sorteado aleatoriamente outra máquina $v_k \in V$ em que o processo p_i estará escalonado.

Na figura 7 podemos observar um exemplo de mutação em um indivíduo, onde temos a mudança de um gene.

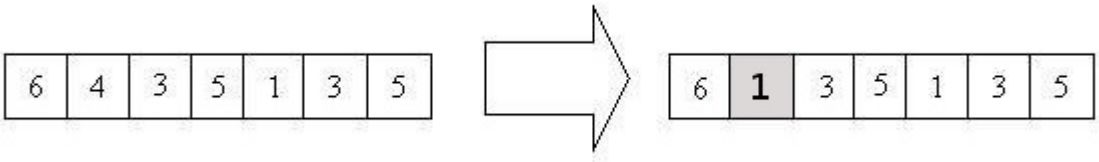

Figura 7 – Exemplo de Mutação

Após a Mutação teremos que decidir quem passará para a próxima geração, sendo que agora temos todos os pais (antiga população) e todos os filhos gerados, teremos que escolher quem sobreviverá para a próxima geração e fará parte da nova população.

4.2.7 Reinserção

A reinserção é o último passo de um algoritmo genético para completar uma geração, nela iremos escolher quais indivíduos (pais ou filhos) que irão para a próxima geração. Neste trabalho simplesmente ordenamos toda nossa população de pais e filhos através do algoritmo *quicksort* (método de ordenação), onde na próxima geração iremos considerar como nova população apenas os indivíduos que estão entre o tamanho da população inicial, enquanto os outros serão descartados.

Novamente nessa nova população iremos executar as fases de seleção, cruzamento, mutação e reinserção, até que tenhamos atingido um número de geração pré-estabelecido.

5 Resultados

Neste capítulo, são descritos os experimentos desenvolvidos a fim de cumprir com o objetivo deste trabalho. Primeiramente, são descritas as instâncias geradas como entrada para o AG, e quais tecnologias foram utilizadas para implementação do mesmo.

5.1 Experimentos Iniciais

Para a execução dos experimentos, foi desenvolvido um AG na linguagem de programação *C*. Os experimentos foram executados em uma máquina com um processador Intel(R) Core(TM) i7-4500U 1.80GHz, com 8,0GB de memória RAM e sistema operacional de 64 bits.

Inicialmente, considerou-se instâncias menores. Foram geradas aleatoriamente valores para as matrizes L e Γ e vetores c e w (seção 4.1). Os valores da matriz L foram gerados considerando as propriedades da matriz descrita na seção 4.1, com valores entre 0 e 50. A matriz Γ também foi gerada considerando suas propriedades, seus valores foram gerados entre 1 e 5. O vetor c teve valores entre 100 e 200, e o vetor w entre 1000 e 5000.

Foram considerados também a quantidade de máquinas e processos para as instâncias, foram um total de nove instâncias: 3 máquinas e 50 tarefas, 3 máquinas e 100 tarefas, 3 máquinas e 200 tarefas, 5 máquinas e 50 tarefas, 5 máquinas e 100 tarefas, 5 máquinas e 200 tarefas, 8 máquinas e 50 tarefas, 8 máquinas e 100 tarefas, 8 máquinas e 200 tarefas.

Estes experimentos foram projetados para observar o comportamento do AG, o qual considera separadamente os termos da função apresentado na equação 4.9, ou seja, ρ^{CPU} e ρ^{Comun}. Para os parâmetros do AG, foi considerada uma população de tamanho 100 e a cada geração foram criados 50 filhos, em um total de 100 gerações. O AG foi executado 50 vezes para cada instância e com o objetivo de minimizar o ρ^{CPU} e o ρ^{Comun} separadamente. Os resultados podem ser vistos na Tabela 1.

A fim de melhorar esses valores, o número de gerações foi aumentado para 1000, e então o AG novamente foi executado com as mesmas instâncias descritas anteriormente. Os resultados podem ser vistos na Tabela 2.

Nos experimentos iniciais, verificamos uma melhora significativa nos resultados das duas execuções ao se aumentar o número de gerações do algoritmo genético, ou seja, o algoritmo genético precisa de mais tempo para a convergência de um bom resultado. Com essa observação, foram então feitos os últimos experimentos, levando em consideração 1000 gerações.

Tabela 1 – Resultados Iniciais

Máquinas	Tarefas	ρ^{CPU}	ρ^{Comun}
3	50	440,2203	3.530,0568
	100	926,2542	23.021,165
	200	1.974,3753	111.791,79
5	50	442,6791	5.380,8006
	100	909,9914	28.773,76
	200	1.958,2343	133.304,98
8	50	450,7834	6.800,3693
	100	905,5373	34.226,065
	200	1.837,0014	153.286,95

Tabela 2 – Resultados Iniciais com 1000 gerações

Máquinas	Tarefas	ρ^{CPU}	ρ^{Comun}
3	50	426,4637	860,7865
	100	895,8768	11.849,9187
	200	1.796,9896	75.878,545
5	50	427,6791	1.388,9096
	100	853,2376	17.992,59
	200	1.772,1098	102.084,69
8	50	400,6744	2.631,1628
	100	842,7062	21.978,8875
	200	1.753,2018	120.305,77

5.2 Experimentos Finais

Com base nas avaliações descritas na seção 5.1, foram executados novos experimentos, porém, dessa vez, foi analisada a soma do ρ^{CPU} com ρ^{Comun}, ou seja, a Equação 4.9. As instâncias usadas no experimento foram obtidas da *Harwell-Boeing matrix collection* via *MatrixMarket*. Foram consideradas sete matrizes DWT para representar as TIGs; segundo Uçar *et al.* (2006) essa matrizes são ricas em *non-zero patterns*, ou seja, as TIGs geradas apresentam muitas arestas. Os custos de comunicação das arestas foram gerados com inteiros aleatórios entre [10,100]. Na Tabela 3 é apresentado detalhes sobre os TIGs.

Tabela 3 – Propriedades das matrizes DTW e modelos TIG

Nome da Matriz	Número de Tarefas	Número de Arestas
DWT59	59	104
DWT66	66	127
DWT72	72	75
DWT209	209	767
DWT221	221	704
DWT234	234	300

Foram gerados três conjunto de máquinas, com $|P| = \{4,8,16\}$. A topologia da rede é um grafo completo, ou seja, cada máquina é conectada a todas as outras. Também

foram atribuídos números aleatórios para a largura de banda, no intervalo [1,10]. Para o AG foi considerado os mesmos parâmetros descritos na Seção 5.1, usando 1000 geração e fazendo 30 execuções de cada instância. Na Tabela 4 são apresentados os resultados.

Tabela 4 – Resultados Finais

Tarefas	Máquinas	Média	Desvio Padrão	Melhor Indivíduo
59	4	1.455.679,5435	29.910,922	1.453.287,0939
	8	1.374.202,597	95.933,231	1.352.765,7012
	16	1.187.383,5539	44.929,4423	1.182.165,509
66	4	1.625.461,9286	47.141,8366	1.621.920,5973
	8	1.542.380,9644	231.370,4494	1.526.422,569
	16	1.331.177,8836	77.555,0701	1.317.439,6319
72	4	1.784.452,776	16.773,9105	1.779.338,91
	8	1.704.999,7721	142.004,8195	1.672.217,3433
	16	1.466.388,3975	197.814,5688	1.452.324,3167
209	4	5.170.030,1838	57.947,6378	5.163.245,8352
	8	5.218.611,3433	619.249,9165	5.184.302,8949
	16	4.438.314,2854	231.057,8389	4.404.761,3663
221	4	5.548.424,2205	102.161,5162	5.540.520,4514
	8	5.610.292,7517	527.624,0084	5.580.541,7844
	16	4.770.969,0776	297.584,5463	4.737.737,96
234	4	5.851.318,1	135.649,5112	5.842.712,1015
	8	5.931.760,5349	573.770,5349	5.889.542,5686
	16	5.038.228,5504	394.522,4531	5.009.383,1543
592	4	15.015.338,9189	202.433,8564	15.004.210,4569
	8	15.734.080,9877	1.534.499,1802	15.664.846,9506
	16	13.303.821,6242	824.966,1186	13.264.403,7695

Na Tabela 4, temos a média do resultado das 30 execuções do AG, com seu desvio padrão e seu melhor indivíduo após as execuções, onde podemos ver uma convergência das execuções para um mesmo valor olhando o desvio padrão.

Gabriel *et al.* (2018) apresentou resultados com essas instâncias utilizando o solver IBM ILOG CPLEX. Na Tabela 5 temos uma comparação do melhor indivíduo obtido nos experimentos com o resultado exato de Gabriel *et al.* (2018), além de uma comparação do tempo gasto para chegar nesses resultados.

Na Tabela 5, observamos a comparação de resultados com o CPLEX que garante o resultado ótimo, notamos que o algoritmo genético se aproximou dos resultados ótimo do CPLEX, porém com um tempo muito mais alto. Isso pode ser explicado pelo solver ser paralelizado (*multithread*), ou seja, ele utiliza múltiplos processadores para chegar em uma solução enquanto o AG foi implementado de maneira sequencial.

Na Tabela 5 também é possível notar que o AG foi pior para as instâncias maiores em comparação com o CPLEX, sendo que, para as menores instâncias, foram obtidos valores mais próximos do resultado ótimo.

Tabela 5 – Comparação de Resultados

Tarefas	Máquinas	Melhor Indivíduo	Tempo	CPLEX	Tempo CPLEX
59	4	1.453.287,0939	275,13	1.452.530	0,107
	8	1.352.765,7012	507,66	1.330.779	0,219
	16	1.182.165,509	725,01	1.181.343	0,802
66	4	1.621.920,5973	208,02	1.615.439	0,094
	8	1.526.422,569	404,07	1.480.033	0,255
	16	1.317.177,8836	696,24	1.313.836	0,974
72	4	1.779.338,91	188,52	1.767.322	0,092
	8	1.672.217,3433	399,51	1.619.186	0,254
	16	1.452.324,3167	672,75	1.437.363	0,905
209	4	5.163.245,8352	1.317,72	5.095.793	0,544
	8	5.184.302,8949	2.286,3	4.668.665	1,997
	16	4.404.761,3663	5.042,94	4.873.331	8,446
221	4	5.540.520,4514	1.472,127	5.466.392	0,570
	8	5.580.541,7844	2.803,8	5.008.201	2,101
	16	4.737.737,96	5.259,81	4.445.817	8,892
234	4	5.842.712,1015	1.466,43	5.762.149	0,565
	8	5.889.542,5686	2.796,84	5.279.168	1,958
	16	5.009.383,1543	5.983,77	4.686.356	7,902
592	4	15.004.210,4569	9.062,76	14.660.860	3,405
	8	15.664.846,9506	19.461,09	13.431.990	13,406
	16	13.264.403,7695	37.254,27	10.955.730	55,791

Após essa comparação, foram analisados o custo total e o *makespan* para três instâncias em particulares, a menor delas (59 tarefas), a mediana (209 tarefas) e a maior (592 tarefas), com o número de máquinas mediano, ou seja, 8 máquinas. Essas instâncias foram analisadas pelo número de gerações de uma execução do AG, e foram construídos os gráficos das Figuras 8, 9 e 10.

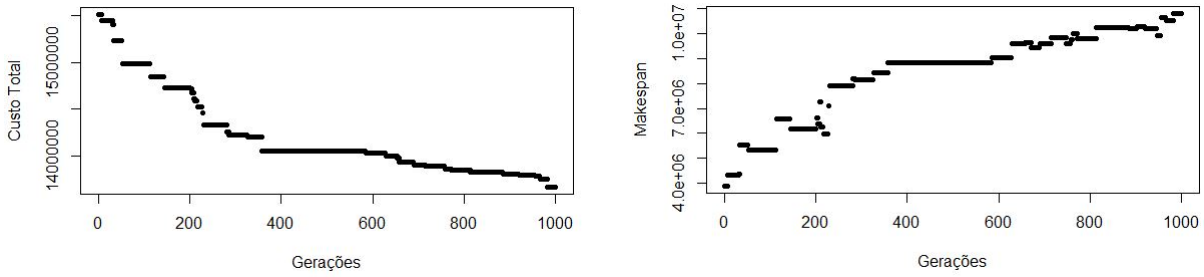

Figura 8 – Custo total e *makespan* por 1000 gerações com 59 tarefas e 8 máquinas

A partir das Figuras 8, 9 e 10, temos comparações do custo total e do *makespan* de três instâncias diferentes. Para o custo total, temos os gráficos semelhantes, em que os valores diminuem rapidamente no começo e no final começam a convergir para um valor. Já o *makespan* tem um comportamento mais irregular com subidas e leves baixas, porém os gráficos sempre representam um aumento no *makespan*. Isso se dá ao fato que o custo

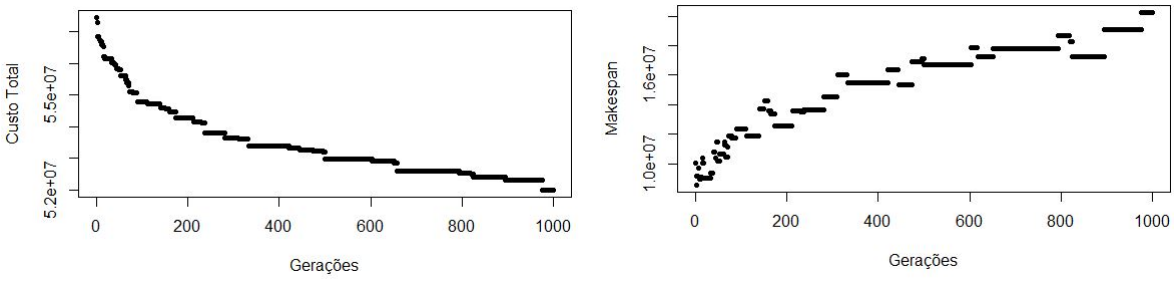

Figura 9 – Custo total e *makespan* para 1000 gerações com 209 tarefas e 8 máquinas

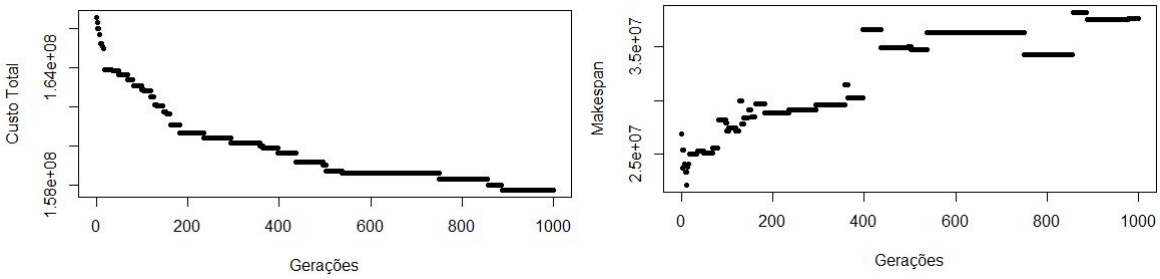

Figura 10 – Custo total e *makespan* para 1000 gerações com 592 tarefas e 8 máquinas

total não faz uma boa distribuição de carga de tarefas entre as máquinas, diferente do *makespan*. Isso justifica novos estudos buscando uma melhor otimização entre essas duas métricas de desempenho.

6 Conclusão

O problema de escalonamento de processos tem se destacado entre os atuais problemas computacionais, devido ao constante uso de computadores, microcomputadores e outras tecnologias que estão presentes cada vez mais no cotidiano das pessoas. Obter um escalonamento de todas as tarefas de maneira eficiente é essencial para balancear os usos de recursos, alcançando, assim, um bom paralelismo e uma melhor experiência e utilização desses sistemas ao usuário final.

Vários algoritmos heurísticos são propostos com o objetivo de tratar esse problema. Dentre eles, estão os algoritmos genéticos (AGs) que tentam otimizar o problema do escalonamento de processos.

Para que essa otimização seja realizada, é necessário adotar como objetivos algumas métricas de desempenho, das quais são bastante usadas na literatura. Este trabalho de conclusão de curso teve como finalidade o desenvolvimento de um AG a fim de otimizar o custo total de execução. Os experimentos foram conduzidos para mostrar a comparação do resultado e tempo do AG com um método exato.

Observou-se, nesses experimentos, a piora do *Makespan* devido a má distribuição de cargas do AG visando em melhorar o custo total de trabalho, ou seja, melhorar o custo carga total piorou o *makespan*.

6.1 Trabalhos Futuros

Foi observado, neste trabalho, que o algoritmo implementado pode ainda ter seu desempenho melhorado por mudanças de parâmetros do AG ou mudanças nos diferentes operadores do AG. Também é possível aplicar um algoritmo multi-objetivo no método de avaliação a fim de comparar com os resultados.

Finalmente, pode-se otimizar o tempo das execuções do AG, paralelizando a função de avaliação e mesmo implementando um AG paralelo. Assim, é possível conduzir uma busca mais sistemática por novas soluções de escalonamento superando, inclusive, outros trabalhos da literatura.

Referências

ABRAHAM, A.; BUYYA, R.; NATH, B. Nature's heuristics for scheduling jobs on computational grids. In: . [S.l.: s.n.], 2000. p. 45–52. Citado na página 9.

ARAFEH, B.; DAY, K.; TOUZENE, A. A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems. *Journal of Systems Architecture*, v. 54, n. 5, p. 530–548, maio 2008. Citado na página 19.

ARROYO, J. E. C.; VIANNA, D. S. Algoritmos genéticos para o problema de programação de tarefas em máquinas paralelas idênticas com dois critérios. In: . [S.l.: s.n.], 2007. p. 1538–1547. Citado na página 9.

BLIEK, C.; BONAMI, P.; LODI, A. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In: HOSEI UNIVERSITY. *Proceedings of the Twenty-Sixth RAMP Symposium*. Tokyo, 2014. p. 171–180. Citado na página 19.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308, 2003. Citado na página 9.

BOKHARI, S. H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Transactions on Software Engineering*, v. 7, n. 6, p. 583–589, nov. 1981. Citado na página 18.

BRAUN, T. D.; SIEGEL, H. J.; BECK, N.; BÖLÖNI, L. L.; MAHESWARAN, M.; REUTHER, A. I.; ROBERTSON, J. P.; THEYS, M. D.; YAO, B.; HENSGEN, D.; FREUNDK, R. F. *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*. [S.l.: s.n.], 2001. 810–837 p. Citado na página 9.

CARRETERO, J.; XHAFA, F.; ABRAHAM, A. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control*, p. 1–19, 2007. Citado 2 vezes nas páginas 9 e 12.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 2, p. 141–154, 1988. Citado 2 vezes nas páginas 8 e 11.

CHAPIN, S. J. Distributed and multiprocessor scheduling. *ACM Computing Surveys*, v. 28, n. 1, p. 233–235, mar. 1996. Citado 2 vezes nas páginas 9 e 11.

DONG, F.; AKL, S. G. *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*. Kingston, Ontario, 2006. 55 p. Citado 3 vezes nas páginas 8, 11 e 12.

GABRIEL, P. H. R. *Uma abordagem orientada a sistemas para otimização de escalonamento de processos em grades computacionais*. Tese (Doutorado) — Universidade de São Paulo, São Carlos, 2013. Citado 6 vezes nas páginas 8, 10, 11, 19, 20 e 23.

GABRIEL, P. H. R.; ALBERTINI, M. K.; CASTELO, A.; MELLO, R. F. de. *Mixed-integer quadratic programming model to address the task scheduling problem*. 2018. Submetido a Discrete Applied Mathematics. Citado 2 vezes nas páginas 19 e 29.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. [S.l.]: W. H. Freeman and Co., 1979. 338 p. Citado 2 vezes nas páginas 8 e 12.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. [S.l.]: Addison-Wesley Professional, 1989. 432 p. Citado 2 vezes nas páginas 13 e 14.

GRAHAM R., L. E. L. J. K. A. R. *Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics*. [S.l.: s.n.], 1979. 287-326 p. Citado na página 8.

KAFIL, M.; AHMAD, I. Optimal task assignment in heterogeneous computing systems. In: HENSGEN, D. (Ed.). *Heterogeneous Computing Workshop*. Geneva: The IEEE Computer Society, 1997. p. 135–146. Citado na página 18.

KAYA, K.; UÇAR, B. Exact algorithms for a task assignment problem. *Parallel Processing Letters*, v. 19, n. 3, p. 451–465, 2009. Citado 3 vezes nas páginas 8, 18 e 19.

LACERDA, E. G. M. d.; CARVALHO, A. d.; LUDERMIR, T. B. Um tutorial sobre algoritmos genéticos. *Revista de Informática Teórica e Aplicada*, v. 9, p. 7–39, 2002. Citado 4 vezes nas páginas 9, 13, 14 e 15.

MA, Y.-C.; CHEN, T.-F.; CHUNG, C.-P. Branch-and-bound task allocation with task clustering-based pruning. *Journal of Parallel and Distributed Computing*, v. 64, n. 11, p. 1223–1240, nov. 2004. Citado na página 18.

NORMAN, M. G.; THANISCH, P. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, v. 25, n. 3, p. 263–302, set. 1993. Citado na página 18.

RIDLEY, M. *Evolution*. 3. ed. [S.l.]: Wiley-Blackwell, 2006. 784 p. Citado 2 vezes nas páginas 13 e 14.

SETUBAL, J.; MEIDANIS, J. *Introduction to computational molecular biology*. [S.l.]: PWS Publishing, 1997. 296 p. Citado 2 vezes nas páginas 13 e 14.

STOCKINGER, H. Defining the grid: a snapshot on the current view. *Journal of Supercomputing*, Springer-Verlag, v. 42, n. 1, p. 3–17, 2007. Citado na página 8.

STONE, H. S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, v. 3, n. 1, p. 85–93, jan. 1977. Citado na página 18.

TOM, P. A.; MURTHY, C. S. R. Optimal task allocation in distributed systems by graph matching and state space search. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 46, n. 1, p. 59–75, abr. 1999. Citado na página 18.

UÇAR, B.; AYKANAT, C.; KAYA, K.; IKINCI, M. Task assignment in heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, v. 66, n. 1, p. 32–46, jan. 2006. Citado 4 vezes nas páginas 8, 18, 19 e 28.

WILLIAMSON, D. P.; SHMOYS, D. B. *The Design of Approximation Algorithm*. [S.l.: s.n.], 2011. 518 p. Citado na página 9.

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, v. 26, n. 4, p. 608–621, 2010. Citado 3 vezes nas páginas 8, 9 e 12.

XHAFA, F.; KOODZIEJ, J.; DURAN, B.; BOGDANSKI, M.; BAROLLI, L. A comparison study on the performance of population-based meta-heuristics for independent batch scheduling in grid systems. *International Conference on Complex, Intelligent and Software Intensive Systems*, p. 123–130, 2011. Citado na página 9.