Um algoritmo genético híbrido para otimização do escalonamento de tarefas independentes em máquinas heterogêneas

José Junio Ribeiro de Sousa

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2022

José Junio Ribeiro de Sousa

Um algoritmo genético híbrido para otimização do escalonamento de tarefas independentes em máquinas heterogêneas

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Paulo Henrique Ribeiro Gabriel

Uberlândia
2022

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

S725 Sousa, José Junio Ribeiro de, 1993-
2022 Um algoritmo genético híbrido para otimização do escalonamento de tarefas independentes em máquinas heterogêneas [recurso eletrônico] / José Junio Ribeiro de Sousa. - 2022.

Orientador: Paulo Henrique Ribeiro Gabriel.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2022.582
Inclui bibliografia.

1. Computação. I. Gabriel, Paulo Henrique Ribeiro , 1984-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação de Mestrado 14/2022, PPGCO				
Data:	27 de setembro de 2022	Hora de início:	09:00	Hora de encerramento:	11:00
Matrícula do Discente:	11922CCP007				
Nome do Discente:	José Junio Ribeiro de Sousa				
Título do Trabalho:	Um algoritmo genético híbrido para otimização do escalonamento de tarefas independentes em máquinas heterogêneas				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Inteligência Artificial				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Prof.ª Dr.ª Márcia Aparecida Fernandes - FACOM/UFU, Prof.ª Dr.ª Priscila Cristina Berbert Rampazzo - UNICAMP e Prof. Dr. Paulo Henrique Ribeiro Gabriel - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Priscila Cristina Berbert Rampazzo - Limeira/SP; Márcia Aparecida Fernandes e Paulo Henrique Ribeiro Gabriel - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Paulo Henrique Ribeiro Gabriel, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Paulo Henrique Ribeiro Gabriel**, **Professor(a) do Magistério Superior**, em 28/09/2022, às 15:48, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Priscila Cristina Berbert Rampazzo**, **Usuário Externo**, em 28/09/2022, às 19:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Márcia Aparecida Fernandes**, **Professor(a) do Magistério Superior**, em 28/09/2022, às 19:59, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3954444** e o código CRC **36F9A16F**.

Agradecimentos

Agradeço à minha mãe Miraci e meu pai José Ribeiro, pelo o apoio a enfrentar as dificuldades da vida e por sempre incentivar nos estudos. Agradeço as minha irmãs, Júlia e Jordana, pela visão de perspectivas de vidas diferentes e pelas inúmeras dificuldades que passamos nos últimos anos. Agradeço à minha namorada Jéssica, pelo incentivo a vida acadêmica e profissional sempre apoiando a evoluir.

A esse trabalho, agradeço meu orientador Paulo Henrique Ribeiro Gabriel, pelo incentivo na pesquisa e paciência e por acreditar no meu trabalho.

Resumo

Nos últimos anos com o crescente poder de processamento das máquinas e a comunicação entre aplicações distribuídas cada vez mais rápidas devido a alta velocidade das redes proporcionaram ainda mais o uso de computação distribuída para solucionar problemas de escalonamento. Diversos algoritmos buscam soluções ótimas para problemas de escalonamento, baseado em diversas funções objetivo. O critério mais abordado na literatura é a minimização do makespan. Motivado por essas características, este trabalho propõe a aplicação de um algoritmo genético (AG) híbrido para o problema de escalonamento de tarefas independentes. O algoritmo possui duas fases: na primeira, utiliza-se um modelo de otimização relaxado (linear) para gerar um conjunto de soluções válidas (inteiras), correspondentes à primeira geração do AG. Em seguida, o algoritmo evolui essa população. O processo evolutivo é aperfeiçoado por meio de um algoritmo de busca local. Esse algoritmo busca reduzir a carga de trabalho dos processadores sobrecarregados, migrando tarefas para os processadores menos ocupados. Caso a migração não encontre melhores resultados, o algoritmo troca tarefas entre esses dois processadores (consequentemente, a busca local tenta balancear a carga de trabalho). O algoritmo genético híbrido proposto aqui foi comparado com outros algoritmos bastante conhecidos da literatura, superando-os em diversas instâncias. Isso indica que o método proposto é uma abordagem promissora a ser considerada para instâncias maiores.

Palavras-chave: Escalonamento de tarefas. Algoritmo genético. Busca local. Modelo ETC.

Abstract

In recent years with the increasing processing power of machines and the increasingly faster communication between distributed applications due to the high speed of networks have provided even more use of distributed computing to solve scheduling problems. Several algorithms seek optimal solutions to scheduling problems, based on several objective functions. The criterion most often addressed in the literature is the minimization of makespan. Motivated by these characteristics, this work proposes the application of a hybrid genetic algorithm (GA) to the problem of scheduling independent tasks. The algorithm has two phases: in the first phase, a relaxed (linear) optimization model is used to generate a set of valid (integer) solutions, corresponding to the first generation of the GA. Then, the algorithm evolves this population. The evolutionary process is refined by means of a local search algorithm. This algorithm seeks to reduce the workload of overloaded processors by migrating tasks to the less busy processors. If the migration does not find better results, the algorithm switches tasks between these two processors (consequently, the local search tries to balance the workload). The hybrid genetic algorithm proposed here was compared with other well-known algorithms in the literature, outperforming them in several instances. This indicates that the proposed method is a promising approach to consider for larger instances.

Keywords: Task Scheduling. Genetic Algorithm. Local Search. ETC Model.

Lista de ilustrações

Lista de tabelas

Lista de siglas

HUX Half Uniform Crossover

LJFR Longest Job to Fastest Resource

MET Minimum Execution Time

MCT Minimum Completion Time

OLB Opportunistic Load Balancing

SA Simulated Annealing

SJFR Shortest Job to Fastest Resource

TS Tabu Search

VNS Variable Neighborhood Search

Sumário

Capítulo 1

Introdução

Problemas relacionados a escalonamento de tarefas aparecem em diversas aplicações no mundo real (XHAFA; ABRAHAM, 2008c; XHAFA; ABRAHAM, 2008b). A área industrial é uma das principais áreas de aplicabilidade, pois se deseja que as máquinas fiquem o mínimo de tempo ociosas a fim de atingir a maior produtividade possível. Esses problemas também são bastante frequentes no domínio da informática (DONG; AKL, 2006; XHAFA; ABRAHAM, 2010; SILVA; GABRIEL, 2020). Por essa razão, esse problema tem sido bastante estudado na literatura buscando diferentes objetivos como, por exemplo: a alocação eficiente dos recursos nas máquinas, a minimização no tempo de ociosidade de cada máquina ou a minimização no tempo de espera das tarefas na fila para serem processadas (FLÓREZ; BARRIOS; PECERO, 2015; CHHABRA; SINGH; KAHLON, 2020).

Problemas de escalonamento de tarefas são computacionalmente complexos (GAREY; JOHNSON, 1979; DONG; AKL, 2006; GUPTA et al., 2012). Por esse motivo, métodos exatos são válidos apenas para resolver instâncias do problema de tamanho reduzido. Como consequência, para ambientes de larga escala, heurísticas e meta-heurísticas têm sido constantemente abordadas (FLÓREZ; BARRIOS; PECERO, 2015; IBRAHIM et al., 2020; HOUSSEIN et al., 2021).

Este trabalho considera o problema de escalonamento em ambientes computacionais heterogêneos, ou seja, o tempo de execução de uma tarefa varia de um processador para outro. Além disso, foca-se em tarefas independentes; portanto, não há eventos de comunicação nem restrições de precedência. O objetivo é reduzir o tempo necessário para completar todas as tarefas, ou seja, deseja-se minimizar o *makespan* (BRAUN et al., 2001; NESMACHNOW; CANCELA; ALBA, 2012; NESMACHNOW; LUNA; ALBA, 2012). É possível formular esse problema como um modelo de programação inteira (PI) (GOGOS et al., 2016a; GOGOS et al., 2016b); entretanto, esse programa não pode ser resolvido em tempo polinomial devido à natureza NP-difícil do problema. Por outro lado, modelos de programação linear (PL) são resolvidos de maneira eficiente (WILLIAMSON; SHMOYS, 2011); assim, é possível relaxar um modelo de PI em um PL, encontrando um limitante

inferior para o *makespan* (WILLIAMSON; SHMOYS, 2011).

Os experimentos realizados neste trabalho consideraram dois cenários para a geração da população inicial: o primeiro cenário, a população é gerada utilizando o algoritmo Min-Min e o segundo cenário, a geração é realizada com nosso método proposto baseado em programação linear. Essa comparação mostrou que o algoritmo proposto obteve resultados superiores em instâncias consistentes com relação à resultados da literatura.

O restante desta dissertação é organizada da seguinte maneira: no Capítulo 2 são apresentados os conceitos necessários para compreensão da abordagem proposta. No Capítulo 3, são discutidos os principais trabalhos relacionados ao tema de pesquisa abordado aqui. O Capítulo 4 detalha o algoritmo genético híbrido proposto neste trabalho, descrevendo suas etapas. Resultados experimentais são apresentados no Capítulo 5 e o Capítulo 6 conclui esta dissertação.

Capítulo 2

Fundamentação Teórica

2.1 Escalonamento de Tarefas Independentes

Esta dissertação tem como objeto de estudo o problema de escalonamento de tarefas independentes em sistemas distribuídos heterogêneos. Mais especificamente, busca-se tratar o cenário em que o tempo de execução de uma tarefa específica varia em relação ao processador ao qual ela foi atribuída. Além disso, não há eventos de comunicação, nem ao menos uma ordem de precedência na execução das tarefas.

Embora seja uma versão simplificada do problema de escalonamento mais geral, que leva em consideração as dependências de tarefas (SILVA; GABRIEL, 2020), o escalonamento de tarefa independente é considerado importante dentro de ambientes distribuídos, como centros de supercomputação e infraestruturas de grade computacionais (BERMAN; FOX; HEY, 2003; NESMACHNOW, 2010). De fato, esse problema aparece com frequência em cenários compostos por múltiplos processadores autônomos executando o mesmo programa com diferentes entradas de dados (do inglês, *single-program multiple-data*, SPMD), comuns em processamento multimídia e em mineração de dados (NESMACHNOW, 2010). Além disso, tais características não tornam o problema menos desafiador. De fato, o escalonamento de tarefas é um problema NP-difícil, pois é uma reformulação do *Multiprocessor Scheduling*, descrito por Garey e Johnson (1979).

Formalmente, seja T um conjunto composto por n tarefas e P um conjuntos de m processadores (ou máquinas). Tipicamente, $n > m$, ou seja, há mais tarefas que processadores. Ainda, seja E uma matriz de dimensões $n \times m$, em que $e_{ij} \in E$ o tempo esperado para se executar a tarefa t_i no processador p_j. Essa matriz é comumente referenciada pela sigla ETC (do inglês, *expected-time-to-compute*). Nesse cenário, HCSP consiste em encontrar uma atribuição (ou mapeamento) $A : T \to P$ tal que cada tarefa é executada por um único processador (ou seja, não há preempção). O principal objetivo desse problema é minimizar o *makespan* definido na Equação (1):

$$C_{\max} = \max_{i=1,\ldots,n} C_i, \tag{1}$$

em que C_i representa o instante em que a tarefa t_i tem sua execução concluída. Para ilustrar esses conceitos, considere a seguinte matriz ETC, para um cenário com três tarefas e dois processadores:

$$E = \begin{pmatrix} 10 & 5 \\ 2 & 7 \\ 4 & 11 \end{pmatrix}$$

Nesse exemplo, a tarefa t_1, t_2 e t_3 demandam, respectivamente 10, 2 e 4 unidades de tempo para serem executada no processador p_1 e 5, 7 e 11 unidades em p_2. Uma possível solução para esse problema é dada pelo mapeamento mostrado a seguir:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Para esse exemplo, $C_1 = 5$ e $C_2 = 2 + 4 = 6$, ou seja, o makespan é dado por $C_{\max} = 6$ unidades de tempo (nesse exemplo, é o makespan ótimo). É importante observar que apenas um valor por linha da matriz A será igual a 1; além disso, todas as tarefas têm que ser mapeadas, ou seja, nenhuma linha de A pode ser composta apenas por valores iguais a 0.

Como é possível observar, a matriz ETC traz todas as informações necessárias para se obter uma solução do HCSP. Esse modelo, também denominado Modelo ETC, foi projetado originalmente por Ali et al. (2000), que descreveu três propriedades empregadas para gerar diferentes matrizes: heterogeneidade dos processadores, heterogeneidade das tarefas e consistência. A primeira propriedade avalia a variação dos tempos de execução de uma determinada tarefa nos recursos computacionais, enquanto a segunda representa a variação dos tempos de execução das tarefas para uma determinada máquina. Finalmente, a consistência é classificada em três categorias:

1. **Matriz consistente:** Se o processador p_j executa qualquer tarefa t_i em menos tempo que o processador p_k, então p_j executará todas as demais tarefas de maneira mais rápida que p_k.

2. **Matriz inconsistente:** Diferente do cenário anterior, um processador p_j pode executar uma tarefa t_i em menos tempo que p_k, mas pode ser mais lento que p_k para a tarefa t_ℓ.

3. **Matriz semi-consistente:** Nesse último caso, pode haver submatrizes consistentes dentro de uma matriz ETC inconsistente.

Conforme mencionado, o escalonamento de tarefas independentes é um problema computacionalmente complexo (GAREY; JOHNSON, 1979). Por essa razão, diversos algoritmos têm sido explorados ao longo dos anos. Entre esses algoritmos, merecem destaque os algoritmos evolutivos (AEs), descritos na Seção 2.2.

2.2 Algoritmos Genéticos

Algoritmos genéticos (GOLDBERG, 1989; DE JONG, 2006; EIBEN; SMITH, 2015), ou AGs, são métodos de busca e otimização inspirados nos mecanismos de evolução dos seres vivos. Esses algoritmos procuram imitar os princípios de seleção natural e sobrevivência do mais apto de acordo com a Teoria Neodarwinista. Os AGs utilizam a ideia de ênfase das soluções mais adequadas, as quais são obtidas pela troca de informação presentes em outras soluções. Esse processo busca simular os mecanismos de seleção, reprodução, hereditariedade e dinâmica das populações encontradas na natureza.

Um AG age basicamente da seguinte forma: inicialmente gera-se uma população aleatória de indivíduos (possíveis soluções obtidas aleatoriamente). Tais soluções são, em geral, representadas por *strings* de caracteres (geralmente 0s e 1s). Durante o processo evolutivo, os indivíduos são avaliados por meio de uma função (*fitness*) e recebem um valor (*score*) que reflete sua capacidade de adaptação. Os indivíduos mais aptos tendem a sobreviver gerando uma nova população. Cada ciclo de processamento que produz uma nova população é chamando de *geração*. A cada geração, tende a ocorrer uma evolução (melhora das soluções). Para a obtenção de novos indivíduos para formar uma nova população, são aplicados os chamados operadores de reprodução, denominados *crossover* e *mutação* (respectivamente a combinação de trechos de duas *strings* e a alteração aleatória de alguns caracteres da *string*) sobre os indivíduos sobreviventes da geração anterior (EIBEN; SMITH, 2015). A Figura 1 mostra o pseudocódigo de um AG típico.

Figura 1 – Pseudocódigo de um AG típico.

```
   // Seja P(t) uma população na geração t.
 1 t ← 0;
 2 inicializar P(t);
 3 avaliar P(t);
 4 enquanto condição de parada não for satisfeita faça
 5 |   t ← t + 1;
 6 |   selecionar P(t) a partir de P(t − 1);
 7 |   aplicar crossover sobre P(t);
 8 |   aplicar mutação sobre P(t);
 9 |   avaliar P(t);
10 fim
```

Fonte: Adaptado de Lacerda, Carvalho e Ludermir (1999).

A vantagem dos AGs em relação a outras técnicas de otimização está na possibilidade de modelar problemas pela simples descrição de uma potencial solução do mesmo (GABRIEL, 2013). Isso possibilita que essa técnica possa ser facilmente adaptada para uma grande diversidade de problemas complexos.

Além disso, a utilização de operadores de mutação e recombinação equilibra dois objetivos aparentemente conflitantes: o *aproveitamento das melhores soluções* e a *exploração do espaço de busca*. O processo de busca é, portanto, multidimensional, preservando soluções candidatas e promovendo a troca de informação entre as soluções exploradas (EIBEN; SMITH, 2015).

Por padrão, os AGs usam como mecanismo de seleção um algoritmo conhecido como *método da roleta* (GOLDBERG, 1989; LACERDA; CARVALHO; LUDERMIR, 1999). Esse método calcula a soma do *fitness* de todos os indivíduos ($Total$) e gera um número aleatório r no intervalo $[0, Total]$; em seguida, calcula o *fitness* acumulado para cada indivíduo, selecionando aquele cujo valor acumulado é maior que r. O pseudocódigo do método da roleta é mostrado na Figura 2, como pseudocódigo.

Figura 2 – Pseudocódigo do método da roleta.

```
  // f_i é o fitness do indivíduo i.
1 Total ← Σ_{i=0}^{n} f_i;
2 r ← rand(0, Total);
3 Parcial ← 0;
4 i ← 0;
5 repita
6 |   i ← i + 1;
7 |   Parcial ← Parcial + f_i;
8 até Parcial ≥ r;
9 retorna Indivíduo i
```

Fonte: Adaptado de Lacerda, Carvalho e Ludermir (1999).

O desempenho de AGs pode, em muitos casos, ser melhorado forçando a escolha do melhor indivíduo encontrado em todas as gerações do algoritmo. Outra opção é simplesmente manter sempre o melhor indivíduo da geração atual na geração seguinte, estratégia essa conhecida como *seleção elitista* ou *elitismo* (EIBEN; SMITH, 2015). Outro exemplo de mecanismo de seleção é a seleção baseada em *ranking*, que utiliza as posições dos indivíduos quando ordenados de acordo com seu *fitness* para determinar a probabilidade de seleção (EIBEN; SMITH, 2015). Nesse caso, podem ser utilizados mapeamentos lineares ou não lineares para determinar a probabilidade de seleção. Uma forma de implementação desse mecanismo é simplesmente manter os n melhores indivíduos para a próxima geração.

Existe ainda a *seleção por torneio*, segundo qual um subconjunto da população com k indivíduos é sorteado e o melhor indivíduo desse grupo é selecionado para se reproduzir. Em geral, utiliza-se o torneio de dois, ou seja, $k = 2$; assim, dois indivíduos (obtidos aleatoriamente da população) competem entre si e o vencedor (o de melhor *fitness*) torna-se um dos pais. Em seguida, repete-se o torneio com mais dois indivíduos, selecionando-

-se o segundo pai. Uma importante propriedade da seleção por torneio que esta não depende de um conhecimento global da população. Além disso, essa seleção não leva em consideração a posição que o indivíduo ocupa na população, permitindo uma seleção menos enviesada (LACERDA; CARVALHO; LUDERMIR, 1999; EIBEN; SMITH, 2015).

Com base no potencial apresentado pelos AGs, esses algoritmos têm sido investigados para encontrar soluções de escalonamento de tarefas em diversos domínios (BRAUN et al., 2001; CARRETERO; XHAFA; ABRAHAM, 2007; ABRAHAM et al., 2008; YOUNIS; YANG, 2017), com o objetivo de diminuir o custo computacional de tal processo. Mais recentemente, AGs passaram a ser explorados em conjunto com outras técnicas, de modo a criar abordagens híbridas. Dentre essas técnicas, destacam-se métodos de *busca local* (XHAFA; BAROLLO; DURRESI, 2007; NESMACHNOW; CANCELA; ALBA, 2012; YOUNIS, 2018).

2.3 Algoritmos de Busca Local

Busca local (do inglês, *local search* – LS) é um processo iterativo que examina um conjunto de pontos na *vizinhança* de uma solução factível, substituindo tal solução pela melhor vizinha, caso exista (EIBEN; SMITH, 2015). Esse procedimento é ilustrado no pseudocódigo da Figura 3.

Figura 3 – Pseudocódigo de uma busca local típica.

```
  // Dada uma solução inicial I e uma função de vizinhança η.
 1 melhor ← I;
 2 it ← 0;
 3 repita
 4 |   cont ← 0;
 5 |   repita
 6 |   |   gerar vizinho J ∈ η(I);
 7 |   |   cont ← cont + 1;
 8 |   |   se f(J) for melhor que f(melhor) então
 9 |   |   |   melhor ← J;
10 |   |   fim
11 |   até melhor solução ser encontrada;
12 |   I ← melhor;
13 |   it ← it + 1;
14 até condição de parada for satisfeita;
```

Fonte: Adaptado de Eiben e Smith (2015).

No processo de geração de vizinhos, podem ser consideradas diversas heurísticas distintas como, por exemplo, *hill climbing*, *tabu search* e *simulated annealing* (XHAFA; BAROLLO; DURRESI, 2007). Quando aplicadas em conjunto com AGs, métodos de

busca local, geralmente, são empregados após os operadores de reprodução. Assim, para uma nova solução (gerada via *crossover* e mutação), aplica-se a LS. No contexto desta dissertação, diversos algoritmos de busca local foram propostos (NESMACHNOW; LUNA; ALBA, 2012; YOUNIS, 2018), sendo aplicados individualmente ou em conjunto com AGs e outras heurísticas.

Capítulo 3

Trabalhos correlatos

Conforme mencionado, o problema de escalonamento de tarefas independentes é computacionalmente intratável por métodos exatos (GOGOS et al., 2016a). Por outro lado, algoritmos heurísticos e meta-heurísticas têm sido recorrentemente empregados para lidar com esse problema (ABRAHAM et al., 2008; YOUNIS, 2018). Embora não forneçam, necessariamente, uma solução ótima, esse algoritmos têm sido empregados por serem implementados de maneira relativamente simples.

Além disso, diversos trabalhos utilizam heurísticas determinísticas para gerar soluções que serão utilizadas para auxiliar na geração da população inicial de algoritmos genéticos e outras meta-heurísticas populacionais. Esse processo é conhecido como *semeadura* e diz-se que a heurística semeou a população inicial. Dessa maneira, as meta-heurísticas tendem a demandar menos tempo para encontrar soluções satisfatórias, uma vez que partem de um ponto relativamente bom dentro do espaço de estados do problema (NESMACHNOW; CANCELA; ALBA, 2012).

3.1 Heurísticas para escalonamento de tarefas independentes

Diversas heurísticas determinísticas têm sido projetadas para lidar com diferentes problemas de escalonamento de tarefas. No contexto de tarefas independentes em ambientes heterogêneos, Braun et al. (2001) descreveram onze heurísticas, sendo dez determinísticas e uma baseada em AGs. Mais recentemente, Rafsanjani e Bardsiri (2012) estenderam esse estudo, propondo novas heurísticas e considerando outras funções objetivo. Nesta seção, são apresentadas algumas dessas heurísticas, todas empregadas com o objetivo de minimizar o makespan.

Opportunistic Load Balancing

A heurística Opportunistic Load Balancing (OLB) trabalha de forma arbitrária, atribuindo cada tarefa t_i para o próximo processador p_j que esteja disponível, independentemente do tempo de execução esperado daquele processador (RAFSANJANI; BARDSIRI, 2012). A expectativa do algoritmo OLB é manter todos os processadores o mais ocupados possível. A vantagem do algoritmo é a simplicidade, porém não considera os tempos de execução de tarefas; assim, tende a gerar soluções com makespan consideravelmente alto.

Minimum Execution Time

Ao contrário do algoritmo OLB, o algoritmo Minimum Execution Time (MET) atribui cada tarefa, em ordem arbitrária, ao processador com o melhor tempo de execução esperado para aquela tarefa, independentemente da disponibilidade desse processador. A motivação por trás do MET é entregar cada tarefa à sua melhor máquina, o que pode, entretanto, causar grande desequilíbrio de carga de trabalho entre as máquinas (RAFSANJANI; BARDSIRI, 2012).

Minimum Completion Time

O algoritmo Minimum Completion Time (MCT) atribui cada tarefa, em ordem arbitrária, ao processador com o tempo mínimo de conclusão esperado para essa tarefa. Isso faz com que algumas tarefas sejam atribuídas a processadores que não possuem o tempo mínimo de execução para elas Braun et al. (2001).

Max-min e Min-min

As heurísticas Max-min e Min-min foram propostas por Ibarra e Kim (1977) e, ainda hoje, têm apresentado bom desempenho para o problema de escalonamento de tarefas independentes. Ambos algoritmos possuem um comportamento similar.

A heurística Min-min usa o MCT como base, o que significa que a tarefa que pode ser concluída mais cedo tem prioridade. Essa heurística começa com o conjunto U de todas as tarefas não mapeadas. Então o conjunto de tempos mínimos de conclusão (M), é encontrado, com base na matriz ETC. Esse conjunto consiste em uma entrada para cada tarefa não mapeada. Em seguida, a tarefa com o tempo de conclusão **mínimo** geral de M é selecionada e atribuída ao processador correspondente e a carga de trabalho do processador selecionado é atualizado. Finalmente, a tarefa selecionada é removida de U e o processo se repete até que todas as tarefas sejam (IBARRA; KIM, 1977; RAFSANJANI; BARDSIRI, 2012). A Figura 4 ilustra esses passos como pseudocódigo.

Figura 4 – Pseudocódigo da heurística Min-min.

```
1  t[][] ← etc[][];
2  visitados[tasks] ← false;
3  numTarefas ← tarefas;
4  enquanto numTarefas > 0 faça
5      valorMinimo ← −∞;
6      tarefa ← −1;
7      maquina ← −1;
8      para i = 0; i < tarefas; i++ faça
9          se visitados[i] == true então
10             break;
11         fim
12         para j = 0; j < maquinas; j++ faça
13             se t[i][j] < valorMinimo então
14                 maquina ← j;
15                 tarefa ← i;
16                 valorMinimo ← t[i][j];
17             fim
18         fim
19     fim
20     solucao ← valorMinimo;
21     visitados[tarefa] ← true;
22     para i = 0; i < tarefas; i++ faça
23         t[i][maquina] ← etc[i][maquina] + solucao.mapeamento[maquina];
24     fim
25 fim
   Resultado: solucao
```

Fonte: Adaptado de Rafsanjani e Bardsiri (2012).

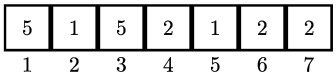
Já a heurística Max-min é muito semelhante à Min-min, também utilizando como base o MCT e começa com os mesmos conjuntos U e M já descritos. Como diferença, essa heurística seleciona a tarefa com o tempo de conclusão **máximo** geral de M, atribuindo-a atribuída ao processador correspondente. Em seguida, a carga de trabalho do processador é atualizada e a tarefa é removida de U, repetindo o processo até que U esteja vazio (IBARRA; KIM, 1977; RAFSANJANI; BARDSIRI, 2012). Este comportamento produz um mapeamento de carga balanceado proporcionando um melhor makespan.

Empiricamente (BRAUN et al., 2001; RAFSANJANI; BARDSIRI, 2012; YOUNIS, 2018), o Min-min tem apresentado resultados melhores em termos de makespan que as demais heurísticas, inclusive quando comparado ao Max-min. Por essa razão, esse algoritmo tem sido utilizado como base de comparação para diversos outros trabalhos no contexto desta dissertação (NESMACHNOW; CANCELA; ALBA, 2010; YOUNIS; YANG, 2018).

3.2 Meta-heurísticas para escalonamento de tarefas independentes

Conforme mencionado na Seção 3.1, Braun et al. (2001) desenvolveram um estudo comparando onze diferentes algoritmos, sendo um deles um AG. Nesse estudo, os autores observaram que o algoritmo genético apresentou melhores resultados, em termos de makespan, quando comparado às demais heurísticas. O AG proposto evoluía uma população de 200 indivíduos; desses, 199 foram gerados aleatoriamente e um foi construído por meio da heurística Min-min (IBARRA; KIM, 1977), em um processo conhecido como semeadura. Os autores propuseram também um esquema de codificação de indivíduos (cromossomo) que passou a ser empregado em outros trabalhos. Nesse modelo, o cromossomo é um vetor de números inteiros de tamanho n, onde n é o número de tarefas e cada gene pode assumir um valor inteiro no intervalo $[1, m]$, ou m é o número de processadores. Esse cromossomo garante a geração de soluções factíveis, pois cada tarefa é atribuída a apenas um processador. A Figura 5 ilustra um cromossomo em um cenário de sete tarefas e cinco processadores. No exemplo, a tarefa t_1 foi atribuída a p_5, t_2 a p_1 e assim por diante.

Figura 5 – Exemplo de cromossomo



Fonte: Gabriel (2013).

Os estudos conduzidos por Braun et al. (2001) também definiram um padrão na geração de instâncias do problema. Em seu artigo, os autores criaram 12 instâncias, cada uma com 512 tarefas e 16 processadores, a partir do algoritmo proposto por Ali et al. (2000). Adotaram, também, a abreviação a seguir, que passou a ser utilizada para identificar o tipo de matriz ETC, $D - T - JHRH.0$, onde (BRAUN et al., 2001; YOUNIS; YANG, 2018):

- D denota o tipo de distribuição de probabilidade (geralmente uniforme – u);
- T denota o tipo de consistência, com as seguintes siglas: c para consistente, i para inconsistente e s para semi-consistente (ver Seção 2.1);
- TH denota a heterogeneidade das tarefas, com duas possibilidades: hi para alto ou lo para baixo;
- RH denota a heterogeneidade dos processadores, com duas possibilidades: hi para alto ou lo para baixo.

O trabalho empírico de Braun et al. (2001) motivou o desenvolvimento de diversas outras pesquisas focadas em meta-heurísticas populacionais, tanto algoritmos evolutivos, quanto métodos baseados em enxames (RITCHIE; LEVINE, 2004; ZAMFIRACHE; FRÎNCU; ZAHARIE, 2011; ZHOU; ZHANG; WANG, 2014; ATIEWI; YUSSOF; EZANEE, 2015; YOUNIS; YANG; PASSOW, 2017; IBRAHIM et al., 2020). Em uma série de artigos, Xhafa e seus colaboradores desenvolveram diversos estudos empíricos sobre algoritmos evolutivos em escalonamento, comparando diferentes operadores genéticos e métodos de inicialização da população inicial (XHAFA, 2007; XHAFA; BAROLLO; DURRESI, 2007; ABRAHAM et al., 2008; XHAFA; ABRAHAM, 2008a; XHAFA et al., 2009; XHAFA; ABRAHAM, 2010). Xhafa, Barollo e Durresi (2007) e Abraham et al. (2008) buscaram identificar as melhores combinações de operadores genéticos para tratar diferentes instâncias do problema. Os autores exploraram diferentes métodos para semear a população inicial, obtendo os melhores resultados com as heurísticas MCT e Min-min. Além disso, empregaram operadores clássicos de crossover e mutação. Outros trabalhos do mesmo grupo de pesquisa (XHAFA, 2007; XHAFA et al., 2009; XHAFA et al., 2008) focaram no desenvolvimento de algoritmos híbridos, empregando técnicas de busca local.

Xhafa et al. (2009) apresentaram um algoritmo genético híbrido com uma Tabu Search (TS), denominado GA(TS). Esse algoritmo tem como objetivo beneficiar a exploração de uma solução dentro da população de indivíduos, usando para isso a exploração de soluções da busca tabu. O GA(TS) encontra a solução inicial utilizando o algoritmo Min-min. Com a solução inicial já definida, o algoritmo trabalha com duas memórias: recente e histórica. A memória recente o status da busca e a memória histórica utiliza uma tabela de espalhamento para filtrar as soluções. A exploração da vizinhança é feita utilizando duas modificações na solução: *transferência*, que move uma tarefa de um o processador para outro, e *troca*, quando duas tarefas em dois processadores diferentes são trocadas. Resultados experimentais apontaram que o algoritmo GA(TS) superou os algoritmos GA e TS indivualmente para minimização de makespan em pequenas e médias instâncias.

Outro trabalho é o GA híbrido com Simulated Annealing (SA), proposto por Xhafa e Abraham (2010). O algoritmo SA explora o conceito de como o metal esfria e congela em cristal com a energia mínima (processo de *recozimento*) e a busca no sistema geral. A execução do algoritmo inicia-se estimando a temperatura inicial e, através dessa estimativa, gera a solução inicial. A partir da solução inicial, avalia-se a solução; se for atendida, é adicionada na lista de melhores soluções e posteriormente é ajustada a temperatura para uma nova execução. A principal vantagem do algoritmo SA sobre outras heurísticas é habilidade de evitar ficar preso nos mínimos locais. Sua junção com o GA trouxe como vantagem a sua capacidade de convergência alinhado a paralelização proporcionado do GA Xhafa e Abraham (2010). O algoritmo GA-SA pode ser formulado conforme a seguir:

1. Gera uma população inicial $\mathbf{P}$ e inicializa de $i = 1$ até P, o vetor $T_h(i)$ com a energia gerado na posição i. Para cada escalonamento ($i = 1$ até P) primeiro é alocado as

tarefas para os recursos disponíveis utilizando a heurística Longest Job to Fastest Resource (LJFR) e depois que uma máquina é desocupado (quando se completa a execução), a tarefa é alocado se baseando na heurística Shortest Job to Fastest Resource (SJFR).

2. Etapa de resfriamento

   a) O vetor de energia (VE) é setada para zero e de $i = 1$ até N é realizado mutação no vetor de escalonamentos.

   b) Calcula a energia (E) do vetor de escalonamentos

   c) Se E > $T_h(i)$, a configuração antiga é restaurada.

   d) Se E < $T_h(i)$, a diferença da energia (E - $T_h(i)$) é incrementada no VE.

3. Etapa de reaquecimento

   a) Calcula o incremento de reaquecimento durante o vetor VE, para $i = 1$ até N.

   b) Adiciona o valor incrementado para cada posição do vetor T_h.

4. Repete as etapas 2 e 3 até o escalonamento ótimo ser encontrado.

Os algoritmos GA(TS) e GA(SA) são mencionados no trabalho de Sheikh e Nagaraju (2017), que compara técnicas de escalonamento de tarefas e balanceamento de carga utilizando a matriz ETC. Os resultados mostram que o GA(SA) tem melhor convergência que o algoritmo GA(TS).

A ideia de utilizar algoritmos híbridos também foi bastante explorada por Nesmachnow (2010), Nesmachnow, Cancela e Alba (2012). Os autores propuseram um algoritmo de busca local baseado no método *Problem Aware Local Search* (PALS). O PALS é um algoritmo determinístico projetado para o problema de montagem de fragmentos de DNA e tem sido adaptado para outros problemas de otimização combinatória (MINETTI; LUQUE; ALBA, 2017). Na versão de Nesmachnow, Cancela e Alba (2012), algoritmo rPALS começa a execução com um escalonador inicial utilizando a heurística MCT; em seguida, seleciona um processador p com alta probabilidade da seleção do maior makespan local para focar na melhoria desse makespan e introduzir essa melhoria nas demais máquinas. Após a seleção, é feita a iteração sobre as tarefas escalonadas na máquina p, a partir dessa iteração é feito o cálculo da variação do makespan da troca das tarefas. Se o resultado for melhor, o makespan é atualizado, esse processo é aplicado até melhorar o makespan global (NESMACHNOW; CANCELA; ALBA, 2012). A Figura 6 mostra mais detalhes deste algoritmo na forma de pseudocódigo.

Experimentos mostraram que o algoritmo rPALS encontrou resultados até 30% melhores que o Min-min para as instâncias de Braun et al. (2001). Por conta disso, Nesmachnow, Cancela e Alba (2010), Nesmachnow, Luna e Alba (2012) incorporaram esse algoritmo

Figura 6 – Pseudocódigo do algoritmo rPALS.

```
1  s ← MCT
2  enquanto MAX_ETAPAS não for satisfeita faça
3      m ← Escolhe uma máquina aleatória
4      operador ← Escolhe operador troca ou transferencia
5      se operador = troca então
6          enquanto RAND_MAX_TAREFAS não for satisfeita faça
7              t ← Escolhe tarefa aleatória na máquina m
8              m_troca ← Escolhe máquina aletória (m ≠ m_troca)
9              t_troca ← Escolhe RAND_MAX_TASKS tarefas no m_troca e escolhe o
                que tem o melhor makespan para realizar a troca da tarefa t na m e t_i
                na m_swap(t^m ⟷ t^m_itroca)
10         fim
11     fim
12     senão se operador = transferencia então
13         m_move ← Escolhe tarefa aleatória (m ≠ m_move)
14         t_move ← Escolhe RAND_MAX_TASKS tarefas no m_move e escolhe o que
            tem o melhor makespan para transferir a tarefa t_i na m_move para m
            (t^m_imove → t^m_i)
15     fim
16     Implanta o melhor entre a troca (t^m ⟷ t^m_iswap) ou a transferência
        (t^m_imove → t^m_i) da s se o makespan global foi reduzido
17 fim
   Resultado: s
```

Fonte: Adaptado de Nesmachnow (2010).

em uma meta-heurística evolutiva denominada CHC (*Cross-generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation*), uma variação do algoritmo genético que trabalha com populações pequenas e altas taxas de mutação (ESHELMAN, 1991; WHITLEY et al., 1996). Bons resultados foram obtidos, tanto para as instâncias de Braun et al. (2001), quanto para novas instâncias maiores (NESMACHNOW; LUNA; ALBA, 2012).

Mais recentemente, Younis e Yang (2017) apresentaram duas variações de operadores de mutação: uma baseada no conceito de troca e outra no conceito de transferência. O primeiro operador é denominado como melhor mutação de troca, na qual consiste na alteração da solução ao encontrar uma troca melhor entre uma das tarefas escalonada em uma máquina p_j e todas as outras tarefas para minimizar o makespan. Já o segundo operador, denominado mutação de transferência, transfere uma tarefa da máquina com o maior tempo de execução para a máquina com o menor tempo de execução. As figuras 7 e 8 ilustram esses dois operadores, respectivamente, em pseudocódigo.

Os resultados encontrados mostram que o algoritmo genético proposto com os novos operadores, encontraram melhores valores de makespan quando comparados a outras

Figura 7 – Operador de mutação de melhor troca.

busca máquina m com o maior makespan local
busca tarefas escalonadas t_{list} na máquina m
para $tl \in t_{list}$ **faça**
    para $te \in tarefas$ && $te \neq tl$ **faça**
        $nova\_solucao \leftarrow$ troca a máquina escalonada da tarefa tl com a máquina escalonada da tarefa te
        calcula a função fitness
        adiciona a $nova\_solucao$ na $lista\_solucoes$
    fim
fim
Resultado: menorMakespan($lista\_solucoes$)

Fonte: Adaptado de Younis e Yang (2017).

Figura 8 – Operador de mutação de melhor transferência.

busca máquina m com o maior makespan local
busca tarefas escalonadas t_{list} na máquina m
busca tarefa j na t_{list} com o maior tempo de execução esperado
escalona a tarefa j na máquina r com o menor tempo de processamento para a tarefa j

Fonte: Adaptado de Younis e Yang (2017).

abordagens não híbridas da literatura. Finalmente, Younis (2018) apresentam uma abordagem híbrida de algoritmo genético com Variable Neighborhood Search (VNS), denominado GA(VNS). Esse algoritmo híbrido apresenta alta qualidade nos escalonamentos, inclusive para instâncias maiores do problema. A população inicial do algoritmo é gerada com um indivíduo utilizando, novamente, a heurística Min-min e, para introduzir maior diversidade na população, os demais indivíduos são gerados aleatoriamente. A função objetivo focal abordado é a minimização da makespan.

No processo de crossover, o autor aborda três tipos utilizados no algoritmo: one-point, two-point e Half Uniform Crossover (HUX). Os crossovers one-point e two-point foram abordados anteriormente pelo autor (XHAFA, 2007). O crossover HUX realiza a permuta da metade dos genes não correspondentes entre os dois pais. Essa quantidade de genes não correspondentes é obtida pela aplicação da distância de Hamming que é o número de recursos que são diferentes entre as duas soluções. O processo de mutação abordado é o mais utilizado pela literatura, onde os indivíduos são randomicamente alterados para manter e introduzir diversidade nas próximas gerações.

A etapa final do algoritmo GA-VNS é a aplicação do VNS; a partir de uma solução inicial, começa a exploração das vizinhanças e se encontrada uma melhor solução é tro-

cada. Esta fase de exploração é definida por uma busca local para encontrar um ótimo local. O algoritmo GA-VNS é descrito, como pseudocódigo, na Figura 9.

Figura 9 – Algoritmo GA-VNS.

```
1  t ← 0
2  Gera a geração inicial Gen(t) com k indivíduos, onde Gen(t)[0] = min_Min() e
     os demais indivíduos são gerados aleatoriamente.
3  Avalia a função fitness(Gen(t))
4  enquanto criterio de parada não satisfeito faça
5  |  t ← t + 1
6  |  Seleciona Parent(t) da Gen(t − 1)
7  |  Aplica a probabilidade p_c, recombina os indivíduos do Parent(t) produzindo
   |   Offspr1(t)
8  |  Aplica a probabilidade p_m, mutação dos indivíduos do Offspr1(t)
   |   produzindo Offspr2(t) usando o algoritmo VNS
9  |  Avalia a função fitness(Offspr2(t))
10 |  Troca Gen(t) da Offspr2(t) por Gen(t − 1)
11 fim
   Resultado: melhorSolucao
```

Fonte: Adaptado de Younis (2018).

Os testes realizados pelo autor utilizaram as 12 instâncias de Braun et al. (2001). Os cenários dos testes envolveram três populações com tamanhos distintos: 10, 20 e 30 indivíduos, respectivamente. O melhor resultado atingido foi com a população com 20 indivíduos; quando aumentado para 30 indivíduos, mostrou-se uma taxa de melhora muito lenta e um tempo computacional maior para encontrar um bom escalonamento. Esses experimentos sugeriram que para populações maiores o GA-VNS não é benéfico.

Capítulo 4

Algoritmo genético com programação linear e busca local

Pela análise dos trabalhos discutidos no Capítulo 3, é possível observar duas características comuns ao projeto de algoritmos genéticos para escalonamento de tarefas: *i*) a semeadura da população inicial, com uma solução gerada por uma heurística; e *ii*) a execução de uma busca local para refinar os indivíduos gerados pelo GA. Tais características motivaram o desenvolvimento do algoritmo proposto neste trabalho. Por um lado, para gerar um conjunto de indivíduos iniciais (etapa de semeadura) utilizou-se um método baseado em programação linear e arredondamento; por outro lado, para refinar os indivíduos, propôs-se um método de busca local que combina busca em vizinhança com uma lista de soluções já visitadas. Ambas as estratégias, bem como detalhes do algoritmo genético em si, são descritas nas próximas seções.

4.1 Geração da solução inicial

O objetivo central do problema de escalonamento é encontrar uma designação $A : T \to P$ que minimiza o makespan, i.e., o valor de $C_{\max}$ (ver detalhes na Seção 2.1). Esse problema pode ser representado pelo modelo de programação inteira (PI) representado a seguir:

$$\text{minimize} \quad C_{\max} \tag{2a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_{ij} e_{ij} \leq C_{\max}, 1 \leq j \leq m, \tag{2b}$$

$$\sum_{j=1}^{m} a_{ij} = 1, \qquad 1 \leq i \leq n, \tag{2c}$$

$$a_{ij} \in \{0, 1\}, \forall\, i, j \tag{2d}$$

Nesse modelo, se atribuirmos uma tarefa t_i ao processador p_j, então $a_{ij} = 1$; caso contrário, o valor de a_{ij} será 0. O conjunto de restrições (2b) garante que a soma dos custos de execução de todas as tarefas atribuídas ao processador p_j é, no máximo, $C_{\max}$. Além disso, as restrições (2c) certificam que a tarefa t_i é atribuída a apenas um processador. A funções objetivo (2a) busca a minimização do valor de $C_{\max}$, ou seja, o makespan.

Como é possível observar pela condição (2d), a variável de decisão a_{ij} é inteira. Devido a essa característica, torna-se impraticável encontrar o makespan ótimo, utilizando métodos exatos, em um tempo de execução aceitável para instâncias relativamente grandes (GAREY; JOHNSON, 1979; WILLIAMSON; SHMOYS, 2011). Entretanto, se as constantes (2d) forem substituídas por $a_{ij} \geq 0$, então obtém-se um modelo de programação linear (PL) que é uma *relaxação* do modelo de PI original. De acordo com Williamson e Shmoys (2011), toda solução ótima para o modelo de PI é viável para o modelo de PL. Além disso, se $Z^{\star}_{LP}$ e $Z^{\star}_{IP}$ denotam, respectivamente, o valor ótimo dos modelos PL e PI, então pode-se demonstrar que

$$Z^{\star}_{LP} \leq Z^{\star}_{IP}.$$

Como consequência, $Z^{\star}_{LP}$ é um *limitante inferior* (do inglês, *lower bound*) para o problema original (2a)—(2d). A fim de ilustrar os efeitos dessa relaxação, considere a seguinte matriz ETC, com duas tarefas e três máquinas:

$$E = \begin{pmatrix} 2 & 3 \\ 5 & 7 \\ 11 & 13 \end{pmatrix}$$

Seguindo o modelo de PI (2a)–(2d), o valor do makespan ótimo é $C_{\max} = 11$, que consiste na seguinte alocação:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Entretanto, caso se resolva o modelo PL, obtém-se a seguinte alocação:

$$A' = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0.25 & 0.75 \end{pmatrix}$$

cujo makespan é $C'_{\max} = 9.75$, ou seja, um valor inferior ao da solução ótima A. Neste trabalho, propõe-se um algoritmo para, a partir do limitante inferior, gerar uma solução viável factível. Trata-se de um método de *aproximação não-determinística* que, a partir de A' gera um conjunto de soluções por meio de arredondamento dos valores não-inteiros de A'. Assim, para o exemplo mostrado, pode-se gerar duas soluções arredondadas:

$$A'_1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad A'_2 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Deve-se observar que A'_1 é um *arredondamento* direto de A', ou seja, os valores não-inteiros foram arredondados para os inteiros mais próximos. Já a solução A'_2 é uma *permutação* de A'_1. Além disso, o makespan para A'_1 é igual a 13 enquanto que, para A'_2, é 18. Ambos os valores são maiores que o ótimo (lembrando que $C_{\max} = 11$), porém representam soluções válidas e que podem ser utilizadas como um palpite inicial para se buscar soluções melhores.

Inicialmente, utilizou-se um método de força-bruta para gerar todas as possíveis permutações das linhas não-inteiras da solução relaxada; porém, tal estratégia é ineficiente, tornando-se proibitiva na prática. Ao mesmo tempo, foi possível observar que a solução arredondada (no exemplo, A'_1) acabava sendo a melhor, ou seja, a de menor makespan. Isso motivou uma estratégia similar porém que, em vez de gerar todas as permutações, gera um número k definido pelo usuário. Em todo caso, a solução arredondada sempre será mantida e as $k - 1$ restantes são permutações dessa.

4.2 Algoritmo de busca local

Nesta seção, é descrito o algoritmo de busca local (LS) proposto neste trabalho. A fim de simplificar a notação, considere uma solução de escalonamento válida A cujo makepan é C. Seja p o processador com a maior carga de trabalho, ou seja, p é responsável pelo valor de makespan. Além disso, seja q_i o i-ésimo processador com menor carga de trabalho. Inicialmente, $i = 1$, i.e., q_i é o processador menos carregado. O objetivo é reduzir a carga do processador p sem sobrecarregar outros processadores, reduzindo o makespan. Em outras palavras, deseja-se melhorar o *balanceamento de carga* da solução.

O algoritmo de LS proposto encontra o conjunto de todas as tarefas atribuídas ao processador p e, a partir desse conjunto, seleciona a tarefa t_k que possui o *menor* valor de ETC no processador q_i. Então, o algoritmo atribui t_k a q_i, gerando uma nova solução A'. Nesta dissertação, esse processo é denominado *migração*. Se o makepan de A', denotado por C', for menor que C, então atualiza-se A e o processo é repetido. Entretando, se o algoritmo não observa uma melhoria, ele desfaz a migração e tenta executar um processo denominado *troca*. Nesse passo, encontra uma tarefa t_ℓ atribuída a q_i com o menor valor de ETC no processador p; em seguida, o algoritmo troca ambos os processadores, i.e., atribui t_ℓ a p e t_k a q_i, computando o novo makespan.

Durante a busca, o algoritmo armazena todas as soluções em uma lista $\mathcal{L}$. Se a nova solução A' (gerada por migração ou por troca) já se encontra na lista, incrementa-se o valor de i em uma unidade. Assim, q_2 representa a segunda máquina menos carrega, q_3 a terceira, e assim por diante. Quando q_i for igual a p, não há mais candidatos a migração e o algoritmo devolve a melhor solução contida em $\mathcal{L}$. A Figura 10 ilustra a busca local proposta na forma de um pseudocódigo.

A Figura 11 ilustra uma sequência de valores de makespan obtidos durante a execução

Figura 10 – Pseudocódigo do operador de busca local proposto.

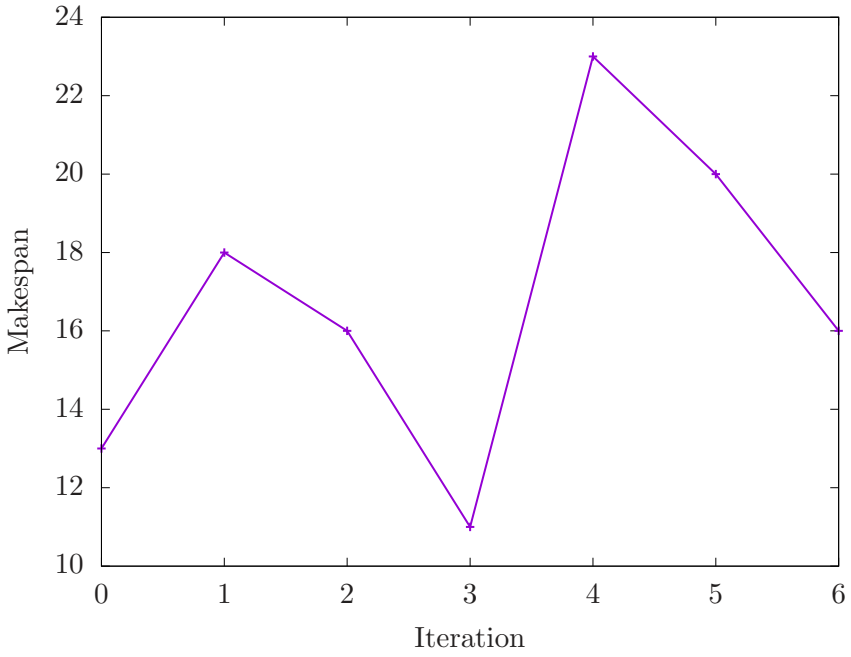
Entrada: Uma solução de escalonamento A.
Saída: Uma nova solução de escalonamento potencialmente melhor.

```
1  seja C o makespan A;
2  L ← ∅;
3  i ← 1;
4  repita
5  |  encontre o processador com maior carga de trabalho p;
6  |  encontre o i-ésimo processador menos carregado q_i;
7  |  se q_i = p então
8  |  |  retorna a melhor solução em L;
9  |  fim
   |  // Fase de migração.
10 |  encontre a tarefa t_k em p que tenha menor valor de ETC em q_i;
11 |  migre a tarefa t_k para q_i, gerando A';
12 |  compute o makespan C' de A';
13 |  se C' < C então
14 |  |  L ← L ∪ A';
15 |  |  A ← A';
16 |  fim
17 |  senão
   |  |  // Fase de troca.
18 |  |  encontre a tarefa t_ℓ de q_i que possua menor valor de ETC em p;
19 |  |  atribua t_ℓ a p e t_k a q_i, gerando A'';
20 |  |  compute o makespan C'' de A'';
21 |  |  se C'' < C então
22 |  |  |  L ← L ∪ A'';
23 |  |  |  A ← A'';
24 |  |  fim
25 |  |  senão
26 |  |  |  i ← i + 1;
27 |  |  fim
28 |  fim
29 até para sempre;
30 retorna melhor solução em L;
```

Fonte: autoria própria.

do algoritmo de busca local. Como entrada, considerou-se a designação A'_1 mostrada como exemplo na Seção 4.1. Para esse exemplo específico, o algoritmo proposto gerou um conjunto de seis soluções válidas, uma por iteração, e avaliou cada uma. O melhor makespan foi encontrado na terceira iteração. Nesse caso, consistiu no valor da solução ótima.

Figura 11 – Exemplo de execução do algoritmo de busca local. O algoritmo executou seis iterações e devolveu a melhor solução, encontrada na terceira iteração.



Fonte: autoria própria.

4.3 Algoritmo genético proposto

O principal objetivo do algoritmo proposto neste trabalho é a minimização do makespan. Como entrada, o algoritmo utiliza a matriz ETC (Seção~2.1) de diversos tamanhos de tarefas e máquinas. A partir da matriz, utiliza-se o processo de inicialização descrito na Seção 4.1.

Após o processo de inicialização, o algoritmo genético é executado utilizando a população inicial, com a sequência de operadores: seleção, cruzamento (crossover) e mutação. Após essas etapas, é aplicado o operador de busca local sobre uma porcentagem dos novos indivíduos gerados.

O processo de seleção encontra os melhores indivíduos da população da geração corrente, a partir dela somente os indivíduos que mais se adaptaram permanecem na população. O processo de cruzamento (crossover) é o processo mais impactante durante a execução do algoritmo, pois é a etapa de criação de novos indivíduos a partir das indivíduos

pais selecionados. A proposta do algoritmo utiliza as seguintes derivações de operadores de crossover: Uniforme, 1-point, K-point e Average, respectivamente nas figuras a seguir.

Figura 12 – Crossover Uniforme

```
1  solucao ← newSolucao();
2  cruzamento ← newint[];
3  para i = 0; i < tarefas; i++ faça
4  |  cruzamento[i] ← aleatorio(0, 1);
5  fim
6  para i = 0; i < tarefas; i++ faça
7  |  se cruzamento[i] == 1 então
8  |  |  solucao.mapeamento[i] ← pai1.mapeamento[i];
9  |  senão
10 |  |  solucao.mapeamento[i] ← pai2.mapeamento[i];
11 |  fim
12 fim
   Resultado: solution
```

Figura 13 – Crossover 1-Point

```
1  solucao ← newSolucao();
2  corteInicial ← aleatorio(0, tarefas);
3  para i = 0; i < corteInicial; i++ faça
4  |  solucao.mapeamento[i] ← pai1.mapeamento[i];
5  fim
6  para i = corteInicial; i < tarefas; i++ faça
7  |  solucao.mapeamento[i] ← pai2.mapeamento[i];
8  fim
   Resultado: solucao
```

Após a fase de cruzamento, os indivíduos sofrem o processo de mutação, que consiste na geração aleatória de tarefa para ser processada em um máquina aleatória do indivíduo atual, confome mostrado na Figura abaixo.

O resultado do processamento do AG é uma população pré-processada com os melhores indivíduos gerados durante o processo de seleção natural. O processamento da população vai ser finalizado com a aplicação da busca local, que verifica se o escalonamento pré-definido pelo AG pode ser melhorado.

Figura 14 – Crossover K-Point

```
1  solucao ← newSolucao();
2  corteInicial ← aleatorio(0, tarefas);
3  corteFinal ← aleatorio(corteInicial, tarefas);
4  para i = 0; i < corteInicial; i++ faça
5  |  solucao.mapeamento[i] ← pai1.mapeamento[i];
6  fim
7  para i = corteInicial; i < corteFinal; i++ faça
8  |  solucao.mapeamento[i] ← pai2.mapeamento[i];
9  fim
10 para i = corteFinal; i < tarefas; i++ faça
11 |  solucao.mapeamento[i] ← pai1.mapeamento[i];
12 fim
   Resultado: solucao
```

Figura 15 – Crossover Average

```
1  solucao ← newSolucao();
2  para i = 0; i < tarefas; i++ faça
3  |  media ← (pai1.mapeamento[i] + pai2.mapeamento[i])/2;
4  |  solucao.mapeamento[i] ← media;
5  fim
   Resultado: solucao
```

Figura 16 – Operador Mutation

```
1  tarefa ← aleatorio(0, tarefas);
2  maquina ← aleatorio(0, maquinas);
3  solucao.mapeamento[maquina] ← tarefa;
   Resultado: solucao
```

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo apresenta os resultados obtidos com o algoritmo proposto no Capítulo 4 utilizando as instâncias HSCP.

Para a avaliação dos resultados, foram utilizados dois conjuntos de instâncias[1]. O primeiro, proposto por Braun et al. (2001), possui 512 tarefas e 16 processadores; o segundo, proposto por Nesmachnow, Cancela e Alba (2010), tem dimensão 1024×32.

Os experimentos foram realizados utilizando duas configurações: *i*) o algoritmo proposto com o primeiro indivíduo sendo gerado pelo algoritmo Min-min e os demais aleatoriamente; e *ii*) e a população inicial gerada pelo método baseado em programação linear (Seção 4.1). Todos os testes foram executados em três cenários distintos: 100, 150 e 200 indivíduos com 1000 iterações em todos os cenários. Nas seções a seguir, são apresentados os resultados.

5.1 Instâncias de 512 tarefas e 16 processadores

Os resultados utilizando a instância 512×16 foram separados pelos operadores de crossover utilizados em cada teste, conforme as seções seguintes.

Crossover Uniforme

As tabelas 1 à 3 mostram os resultados obtidos usando o indivíduo inicial gerado pelo algoritmo Min-min.

As demais tabelas (4–6) mostram os resultados obtidos usando o método proposto de geração da população inicial apresentado na Seção 4.1.

A grande diferença entre os resultados das tabelas 1 a 6 consiste no método de geração da população inicial, quanto utilizamos o método proposto descrito na Seção 4.1, a população inicial é gerada com melhores indivíduos comparados com o mesmo processo

[1] Todas essas instâncias estão disponíveis no link a seguir: <https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/>.

Tabela 1 – Crossover Uniforme + Algoritmo Min-Min: 100 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio Instância
u_c_hihi	7925750.05	7982165.44	0.40%
u_c_hilo	156285.28	156586.91	0.08%
u_c_lohi	254684.54	256285.42	0.40%
u_c_lolo	5282.86	5292.31	0.09%
u_i_hihi	3147702.24	3170406.45	0.56%
u_i_hilo	75775.72	76074.66	0.24%
u_i_lohi	108935.44	110243.50	0.49%
u_i_lolo	2636.18	2644.98	0.21%
u_s_hihi	4428236.3	4498554.55	0.65%
u_s_hilo	99497.46	99687.49	0.17%
u_s_lohi	129212.63	129943.15	0.30%
u_s_lolo	3550.63	3561.22	0.14%

Tabela 2 – Crossover Uniforme + Algoritmo Min-Min: 150 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio padrão
u_c_hihi	7493492.63	7797448.0	2.95%
u_c_hilo	156022.65	156471.54	0.11%
u_c_lohi	254572.98	256248.23	0.44%
u_c_lolo	5280,32	5291,07	0.11%
u_i_hihi	3138534.86	3175534,72	0.64%
u_i_hilo	75966.56	76214.58	0.28%
u_i_lohi	108772.1	110086.87	0.71%
u_i_lolo	2621.93	2641.65	0.32%
u_s_hihi	4427158.84	4491548.57	0.67%
u_s_hilo	99414,93	99732.01	0.21%
u_s_lohi	129224.4	130010.32	0.32%
u_s_lolo	3548.01	3554.90	0.12%

Tabela 3 – Crossover Uniforme + Algoritmo Min-Min: 200 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio padrão
u_c_hihi	7912285.61	7967124.75	0.41%
u_c_hilo	156665.84	156817.67	0.08%
u_c_lohi	256247.24	257742.92	0.24%
u_c_lolo	5292.45	5296.84	0.06%
u_i_hihi	3193473.98	3227012.42	0.46%
u_i_hilo	76010.12	76397.27	0.24%
u_i_lohi	110152.88	111288.75	0.69%
u_i_lolo	2637.51	2652.03	0.36%
u_s_hihi	4494672.15	4542983.65	0.43%
u_s_hilo	99054	99749.23	0.28%
u_s_lohi	129267.77	131122.94	0.62%
u_s_lolo	3548.14	3563.56	0.24%

Tabela 4 – Crossover Uniforme + Método proposto: 100 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio padrão
u_c_hihi	7377382	7382059	0.02%
u_c_hilo	152964.2	152992.1	0.01%
u_c_lohi	239006.2	239158.9	0.02%
u_c_lolo	5141.53	5142.36	0.01%
u_i_hihi	2968409	2971776	0.05%
u_i_hilo	73393.65	73403.1	0.01%
u_i_lohi	102482.1	102576.4	0.07%
u_i_lolo	2539.12	2539.55	0.01%
u_s_hihi	4116024	4119614	0.04%
u_s_hilo	95872.97	95896.91	0.01%
u_s_lohi	121779.7	121800.1	0.01%
u_s_lolo	3427.89	3428.74	0.01%

Tabela 5 – Crossover Uniforme + Método proposto: 150 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio padrão
u_c_hihi	7494229.68	7706937.41	3%
u_c_hilo	154321.5	154767.05	0.19%
u_c_lohi	244747.66	245890.10	0.22%
u_c_lolo	5172	5178.78	0.09%
u_i_hihi	3086740.56	3127033.13	0.67%
u_i_hilo	75372.3	75918.45	0.30%
u_i_lohi	109803.57	110710.34	0.67%
u_i_lolo	2590.55	2607.25	0.34%
u_s_hihi	4294860.81	4330463.35	0.50%
u_s_hilo	97356.84	97569.44	0.17%
u_s_lohi	126631.51	127837.33	0.49%
u_s_lolo	3484,05	3492.75	0.12%

Tabela 6 – Crossover Uniforme + Método proposto: 200 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio padrão
u_c_hihi	7475524.18	7500031.21	0.19%
u_c_hilo	154394.21	154693.47	0.14%
u_c_lohi	243424.43	245060.41	0.37%
u_c_lolo	5168.67	5185.09	0.16%
u_i_hihi	3085528.32	3128322.88	0.67%
u_i_hilo	75454.43	75803.46	0.28%
u_i_lohi	109090.26	110430.33	0.85%
u_i_lolo	2591.83	2607.78	0.28%
u_s_hihi	4274905.04	4327099.07	0.60%
u_s_hilo	97173.73	97516.04	0.19%
u_s_lohi	126198.15	127811.64	0.51%
u_s_lolo	3485.07	3494.36	0.19%

utilizando o algoritmo Min-Min. A partir do método proposto de geração, obteve-se melhores resultados em todos os cenários de: 100, 150 e 200 indivíduos das tabelas 4 à 6.

Demais Operadores de Crossover

Os demais operadores de crossover utilizados nos experimentos foram: 1-Point, K-Point e Average e seus resultados estão disponíveis no link[2].

Dezenas de testes foram executados utilizando o algoritmo proposto variando os operadores de crossover apresentados na Seção 4.3. Dentre todos os testes, os melhores resultados obtidos estão descritos na tabela 4. Estes resultados foram comparados com alguns trabalhos da literatura ((NESMACHNOW; CANCELA; ALBA, 2012) e (ZAMFIRACHE; FRÎNCU; ZAHARIE, 2011)), conforme as tabelas 7, 8 e 9.

Tabela 7 – Comparação dos experimentos com os resultados do autor Braun et al. (2001)

	Resultados Experimentos	Resultados de Braun et al.
u_c_hihi	**7377382**	7381570.0
u_c_hilo	**152964.2**	153105.4
u_c_lohi	**239006.2**	239260.0
u_c_lolo	**5141.53**	5147.9
u_i_hihi	2968409	**2938380.8**
u_i_hilo	73393.65	**73378.0**
u_i_lohi	102482.1	**102050.6**
u_i_lolo	**2539.12**	2541.4
u_s_hihi	4116024	**4103500.3**
u_s_hilo	95872.97	**95787.4**
u_s_lohi	**121779.7**	122083.3
u_s_lolo	**3427.89**	3433.5

Tabela 8 – Comparação dos makespan dos experimentos e demais meta-heurísticas (NESMACHNOW; CANCELA; ALBA, 2012)

Instâncias 512x16	GA	MA+TS	cMA	ACO+TS	pμCHC	GA Proposed
u_c_hihi	8050844.5	7530020.2	7700929.8	7497200.9	7381570.0	**7377382**
u_c_hilo	156249.2	154917.2	155334.8	154234.6	153105.4	**152964.2**
u_c_lohi	258756.8	245288.9	251360.2	244097.3	239260.0	**239006.2**
u_c_lolo	5272.3	5173.7	5218.2	5178.4	5147.9	**5141.53**
u_i_hihi	3104762.5	3058474.9	3186664.7	2947754.1	**2938380.8**	2968409
u_i_hilo	75816.1	75108.5	75856.6	73776.2	**73378.0**	73393.65
u_i_lohi	107500.7	105808.6	110620.8	102445.8	**102050.6**	102482.1
u_i_lolo	2614.4	2596.6	2624.2	2553.5	2541.4	**2539.12**
u_s_hihi	4566206	4321015.4	4424540.9	4162547.9	**4103500.3**	4116024
u_s_hilo	98519.4	97177.3	98283.7	96762	**95787.4**	95872.97
u_s_lohi	130616.5	127633	130014.5	123922	122083.3	**121779.7**
u_s_lolo	3583.4	3484.1	3522.1	3455.2	3433.5	**3427.89**

As comparações dos resultados dos experimentos das tabelas 7 e 8 (destacados em **negrito**), mostram que o algoritmo proposto encontrou soluções melhores que o da lite-

[2] Todos os resultados das instâncias utilizando os crossovers: 1-Point, K-Point e Average estão disponíveis no hyperlink a seguir: <https://github.com/JoseJunio/two-phase-scheduling/tree/main/logs/512x16>.

ratura nas instâncias: u_c_*, u_i_lolo, u_s_lohi e u_s_lolo. Para as demais instâncias, alcançou resultados bastante satisfatórios.

Tabela 9 – Comparação dos makespan dos experimentos e demais heurísticas (ZAMFIRACHE; FRÎNCU; ZAHARIE, 2011)

Instâncias 1024x32	GreedyMove	GreedySwap	Hybrid	MA+TS	GA Proposed
u_c_hihi	7684852.40	7689131.76	7609663.13	7530020.18	**7377382**
u_c_hilo	155248.33	155495.10	154979.43	153917.17	**152964.2**
u_c_lohi	251445.60	250558.63	248903.70	245288.94	**239006.2**
u_c_lolo	5255.06	5258.4	5235.00	5173.72	**5141.53**
u_i_hihi	3072453.70	3019756	3014083.63	3058474.90	**2968409**
u_i_hilo	75222.90	74684.43	74553.20	75108.49	**73393.65**
u_i_lohi	106309.56	105261.20	105013.60	105808.58	**102482.1**
u_i_lolo	2617.26	2590.83	2585.70	2596.57	**2539.12**
u_s_hihi	4382845.80	4352017.96	4316556.23	4321015.44	**4116024**
u_s_hilo	98036.16	98302.36	97964.86	97177.29	**95872.97**
u_s_lohi	127565.00	127026.63	126763.23	127633.02	**121779.7**
u_s_lolo	3538.96	3526.53	3520.80	3484.08	**3427.89**

A tabela 9 mostrou que os experimentos geraram resultados melhores em todas as instâncias em relação ao trabalho do autor (ZAMFIRACHE; FRÎNCU; ZAHARIE, 2011) que sugere um algoritmo genético com operador de pertubação na população combinado com estratégias de escalonamento. Na seção seguinte, são apresentados os resultados dos experimentos com instâncias 1024×32.

5.2 Instâncias de 1024 tarefas 32 processadores

A outra instância HSCP utilizada nos experimentos foi de 1024 tarefas e 32 máquinas. Na seção seguinte, contêm os resultados produzidos com o algoritmo proposto utilizando o operador de crossover uniforme. Os cenários ficaram limitados a 100 e 150 indivíduos, devido a limitação do processamento da máquina utilizada nos testes.

Crossover Uniforme

As tabelas 10 e 11 mostram os resultados obtidos a partir da utilização do algoritmo Min-Min como geração inicial da população.

Os experimentos a seguir utilizaram o método proposto de programação linear para geração da população inicial, conforme as tabelas 12 e 13.

Dentre todos os experimentos realizados com as instâncias 1024×32 (tabelas 10 à 13), selecionamos os melhores indivíduos encontrados e comparamos com os resultados obtidos por Braun et al. (2001) com o algoritmo pμ-CHC (NESMACHNOW; LUNA; ALBA, 2012), conforme a tabela 14.

Tabela 10 – Crossover Uniforme + Algoritmo Min-Min: 100 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio Instância
u_c_hihi	20996051.84	21091362.42	0.25%
u_c_hilo	2116751.63	2121713.72	0.15%
u_c_lohi	2015.1	2023.83	0.25%
u_c_lolo	212.91	213.95	0.25%
u_i_hihi	5893252.96	6063355.7	1.22%
u_i_hilo	553972.21	568640.68	1.05%
u_i_lohi	588.96	604.03	1.11%
u_i_lolo	57.49	58.50	0.63%
u_s_hihi	13171788.41	13273758.06	0.47%
u_s_hilo	1248532.16	1255948.36	0.31%
u_s_lohi	1273.24	1282.20	0.38%
u_s_lolo	130.55	131.65	0.36%

Tabela 11 – Crossover Uniforme + Algoritmo Min-Min: 150 indivíduos e 1000 iterações

Instance	Melhor indivíduo	Média	Desvio Instância
u_c_hihi	20923311.53	21042180.27	0.22%
u_c_hilo	2110995.48	2118329,32	0.17%
u_c_lohi	2016.88	2021.67	0.13%
u_c_lolo	212.74	213.59	0.20%
u_i_hihi	5912622.94	6039287.36	1.39%
u_i_hilo	558598.63	566507.41	0.93%
u_i_lohi	592.08	605.22	1.35%
u_i_lolo	58.33	58.73	0.46%
u_s_hihi	13065842.93	13257057.69	0.70%
u_s_hilo	1250646.79	1255166.99	0.26%
u_s_lohi	1274.17	1280.95	0.29%
u_s_lolo	131.05	131.65	0.30%

Tabela 12 – Crossover Uniforme + Método proposto: 100 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio Instância
u_c_hihi	19879842.63	19977236.97	0.29%
u_c_hilo	1997941.83	2005180.49	0.23%
u_c_lohi	1910.79	1917.75	0.33%
u_c_lolo	204.47	205.57	0.33%
u_i_hihi	5810980.26	5974382.22	1.22%
u_i_hilo	557688.31	564631.47	0.81%
u_i_lohi	579.94	590.44	1.04%
u_i_lolo	54.99	56.49	1.40%
u_s_hihi	12761406.79	12865656.3	0.60%
u_s_hilo	1228223.66	1243060.32	0.65%
u_s_lohi	1240.94	1257.21	0.74%
u_s_lolo	127.55	129.53	0.97%

Tabela 13 – Crossover Uniforme + Método proposto: 150 indivíduos e 1000 iterações

Instâncias	Melhor indivíduo	Média	Desvio Instância
u_c_hihi	19892587.39	20002808.61	0.28%
u_c_hilo	2000616.3	2009419.03	0.27%
u_c_lohi	1906.59	1919.99	0.47%
u_c_lolo	205.32	206.01	0.12%
u_i_hihi	5868884.58	5983841.06	0.91%
u_i_hilo	556319.42	563807.87	1.16%
u_i_lohi	579.25	595.43	1.18%
u_i_lolo	55.93	56.62	0.70%
u_s_hihi	12706658.92	12886574.38	0.70%
u_s_hilo	1236836.11	1246068.85	0.52%
u_s_lohi	1243.45	1257.34	0.73%
u_s_lolo	128.94	129.65	0.42%

Tabela 14 – Cenário 1024X32: Comparação dos experimentos com os resultado do autor Braun et al. (2001)

Instâncias 1024x32	MCT	Min-Min	Sufferage	pμ-CHC	GA Proposed
u_c_hihi	32832740.0	22508064.0	26063096.0	19677858.0	19879842.63
u_c_hilo	3245777.0	2255966.3	2694595.0	1969980.0	1997941.83
u_c_lohi	3058.7	2155.0	2537.5	1885.3	1906.59
u_c_lolo	323.9	225.9	261.0	201.2	204.47
u_i_hihi	7567147.0	6367767.5	5601367.0	5126273.0	5810980.26
u_i_hilo	713132.4	641428.4	533545.2	485189.8	553972.21
u_i_lohi	754.1	664.7	551.7	513.8	579.25
u_i_lolo	73.4	63.7	55.4	50.2	54.99
u_s_hihi	19008366.0	14125880.0	14481880.0	11837170.0	12706658.92
u_s_hilo	1825499.9	1319050.5	1379341.3	1148940.7	1228223.66
u_s_lohi	1822.0	1380.5	1417.8	1152.5	1240.94
u_s_lolo	194.2	138.7	141.0	118.9	127.55

Os resultados encontrados foram melhores em comparação aos algoritmos MCT, Min-Min e Sufferage para as instâncias: u_c_**, u_s_** e u_i_lolo. Em comparação com o algoritmo pμ-CHC, os resultados foram próximos ao da literatura.

O algoritmo proposto nesta dissertação mostrou-se bastante eficiente, conseguindo melhores resultados em 60% das instâncias, conforme as tabelas 8 e 9 para as instâncias 512×16, principalmente em instâncias consistentes. Para as instâncias 1024×32, os resultados se mostraram melhores comparados com demais heurísticas, conforme visto anteriormente. Este processo deve-se ao fato do método proposto proporcionar indivíduos iniciais melhores que o algoritmo Min-Min devido a utilização de programação linear juntamente com a hibridização do algoritmo proposto, mesclando o algoritmo genético com diversos tipos de crossover com o algoritmo de busca local.

Capítulo 6

Conclusões

Este trabalho abordou problemas de escalonamento de tarefas e propôs a implementação de algoritmo genético híbrido com algoritmo de busca local, para otimizar o tempo total de execução das tarefas. O algoritmo proposto, foi dividido em duas fases: a primeira fase consistiu na geração da população inicial e o processamento da população selecionando os melhores indivíduos, através dos operadores de crossover e mutação; a segunda fase do algoritmo utilizou a população processada pela primeira fase, para buscar ótimos locais através de permutas das máquinas mais carregadas para as menos carregadas por indivíduo.

Vários experimentos foram realizados utilizando diversos cenários para as instâncias: 512×16 e 1024×32, a fim de validarmos que os resultados produzidos fossem expressivos comparados com a literatura e identificar as vantagens e limitações. Os experimentos mostram que o algoritmo proposto, gerou e selecionou melhores indivíduos que demais meta-heurísticas da literatura para alguns casos, e piores para outros, através do critério de minimização do makespan. Com tudo se mostrou bastante eficaz em gerar soluções ótimas. As limitações encontradas foram durante os experimentos, as máquinas utilizadas para realizar os testes não possuem processadores e CPUs com capacidade necessária para realizar os testes em instâncias maiores: 2048×64 e 4096×128.

6.1 Trabalhos Futuros

Para trabalhos futuros, indica-se:

1. Execução do AG proposto para instâncias maiores: 2048×64 e 4096×128 (NESMACHNOW, 2010).
2. Inclusão de novas funções objetivo, como, por exemplo, minimização do *flowtime* e minimização de consumo energético (CHHABRA; SINGH; KAHLON, 2020).
3. Utilização de novos operadores de crossover e mutação (EIBEN; SMITH, 2015).

Referências

ABRAHAM, A.; LIU, H.; GROSAN, C.; XHAFA, F. Nature inspired meta-heuristics for grid scheduling: Single and multi-objective optimization approaches. In: XHAFA, F.; ABRAHAM, A. (Ed.). *Metaheuristics for Scheduling in Distributed Computing Environments*. Heidelberg: Springer Berlin, 2008, (Studies in Computational Intelligence, v. 146). p. 247–272.

ALI, S.; SIEGEL, H. J.; MAHESWARAN, M.; HENSGEN, D.; ALI, S. Task execution time modeling for heterogeneous computing systems. In: RAGHAVENDRA, C. (Ed.). *Proceedings 9th Heterogeneous Computing Workshop*. Cancun, Mexico: IEEE Computer Society, 2000. p. 185–199.

ATIEWI, S.; YUSSOF, S.; EZANEE, M. A comparative analysis of task scheduling algorithms of virtual machines in cloud environment. *Journal of Computer Science*, Science Publications, v. 11, n. 6, p. 804–812, jun 2015.

BERMAN, F.; FOX, G.; HEY, A. J. G. (Ed.). *Grid Computing*: Making the global infrastructure a reality. EUA: John Wiley & Sons, Inc., 2003. 1080 p. (Wiley Series on Communications Networking & Distributed Systems). ISBN 9780470853191.

BRAUN, T. D.; SIEGEL, H. J.; BECK, N.; BÖÖNI, L. L.; MAHESWARAN, M.; REUTHER, A. I.; ROBERTSON, J. P.; THEYS, M. D.; YAO, B.; HENSGEN, D.; FREUND, R. F. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, Elsevier BV, v. 61, n. 6, p. 810–837, jun. 2001.

CARRETERO, J.; XHAFA, F.; ABRAHAM, A. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative*, v. 3, n. 5, p. 1–19, out. 2007.

CHHABRA, A.; SINGH, G.; KAHLON, K. S. Performance-aware energy-efficient parallel job scheduling in HPC grid using nature-inspired hybrid meta-heuristics. *Journal of Ambient Intelligence and Humanized Computing*, Springer Science and Business Media LLC, v. 12, n. 2, p. 1801–1835, jun 2020.

DE JONG, K. A. *Evolutionary computation*: A unified approach. Cambridge, MA: MIT Press, 2006. 256 p. ISBN 0262041944.

DONG, F.; AKL, S. G. *Scheduling Algorithms for Grid Computing*: State of the art and open problems. Kingston, Ontario, 2006.

EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing*. 2. ed. UK: Springer London, Limited, 2015. 287 p. (Natural Computing Series). ISBN 9783662448748.

ESHELMAN, L. J. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In: *Foundations of Genetic Algorithms*. [S.l.]: Elsevier, 1991. p. 265–283.

FLÓREZ, E.; BARRIOS, C. J.; PECERO, J. E. Methods for job scheduling on computational grids: Review and comparison. In: OSTHOFF, C.; NAVAUX, P. O. A.; HERNANDEZ, C. J. B.; DIAS, P. L. S. (Ed.). *High Performance Computing*. Cham: Springer International Publishing, 2015, (Communications in Computer and Information Science, v. 565). p. 19–33.

GABRIEL, P. H. R. *Uma abordagem orientada a sistemas para otimização de escalonamento de processos em grades computacionais*. 98 p. Tese (Doutorado) — Universidade de São Paulo, São Carlos, SP, abr. 2013.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability*: a guide to the theory of NP-completeness. EUA: W. H. Freeman, 1979. 338 p. (A Series of Books in the Mathematical Sciences). ISBN 0716710447.

GOGOS, C.; VALOUXIS, C.; ALEFRAGIS, P.; GOULAS, G.; VOROS, N.; HOUSOS, E. Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing. *Future Generation Computer Systems*, Elsevier BV, v. 60, p. 48–66, jul. 2016.

GOGOS, C.; VALOUXIS, C.; ALEFRAGIS, P.; XANTHOPOULOS, I.; HOUSOS, E. Scheduling independent tasks on heterogeneous systems by optimizing various objectives. In: BURKE, E. K.; DI GASPERO, L.; ÖZCAN, E.; MCCOLLUM, B.; SCHAERF, A. (Ed.). *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*. Udine, Italy: EWG, 2016. p. 149–161.

GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. Michigan: Addison-Wesley Pub. Co., 1989. 412 p. ISBN 0201157675.

GUPTA, A.; IM, S.; KRISHNASWAMY, R.; MOSELEY, B.; PRUHS, K. Scheduling heterogeneous processors isn't as easy as you think. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. USA: Society for Industrial and Applied Mathematics, 2012. p. 1242–1253.

HOUSSEIN, E. H.; GAD, A. G.; WAZERY, Y. M.; SUGANTHAN, P. N. Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation*, Elsevier BV, v. 62, p. 100841, abr. 2021.

IBARRA, O. H.; KIM, C. E. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, Association for Computing Machinery (ACM), v. 24, n. 2, p. 280–289, abr. 1977.

IBRAHIM, M.; NABI, S.; HUSSAIN, R.; RAZA, M. S.; IMRAN, M.; KAZMI, S. M. A.; ORACEVIC, A.; HUSSAIN, F. A comparative analysis of task scheduling approaches in cloud computing. In: LEFEVRE, L.; VARELA, C. A.; PALLIS, G.; TOOSI, A. N.; RANA, O.; BUYYA, R. (Ed.). *20th IEEE/ACM International Symposium on Cluster,*

Cloud and Internet Computing. Melbourne, Australia: IEEE Computer Society, 2020. p. 681–684.

LACERDA, E. G. M. de; CARVALHO, A. C. P. L. F. de; LUDERMIR, T. B. Um tutorial sobre algoritmos genéticos. *Revista de Informática Teórica e Aplicada*, v. 4, n. 2, p. 109–139, jul. 1999.

MINETTI, G. F.; LUQUE, G.; ALBA, E. The problem aware local search algorithm: an efficient technique for permutation-based problems. *Soft Computing*, Springer Science and Business Media LLC, v. 21, n. 18, p. 5193–5206, fev. 2017.

NESMACHNOW, S. *Parallel evolutionary algorithms forscheduling on heterogeneous computingand grid environments*. 143 p. Tese (Doutorado) — Universidad de la República, Montevideo, Uruguay, abr. 2010.

NESMACHNOW, S.; CANCELA, H.; ALBA, E. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, Springer Science and Business Media LLC, v. 15, n. 4, p. 685–701, mar. 2010.

_____. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing*, Elsevier BV, v. 12, n. 2, p. 626–639, fev. 2012.

NESMACHNOW, S.; LUNA, F.; ALBA, E. An efficient stochastic local search for heterogeneous computing scheduling. In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum*. Shangai, China: IEEE, 2012. p. 593–600.

RAFSANJANI, M. K.; BARDSIRI, A. K. A new heuristic approach for scheduling independent tasks on heterogeneous computing systems. *International Journal of Machine Learning and Computing*, EJournal Publishing, v. 2, n. 4, p. 371–376, ago. 2012.

RITCHIE, G.; LEVINE, J. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*. Cork, Ireland: Computer And Information Sciences, 2004. p. 1–7.

SHEIKH, S.; NAGARAJU, A. A comparative study of task scheduling and load balancing techniques with MCT using ETC on computational grids. *Indian Journal of Science and Technology*, Indian Society for Education and Environment, v. 10, n. 32, p. 1–14, fev. 2017.

SILVA, E. C. da; GABRIEL, P. H. R. A comprehensive review of evolutionary algorithms for multiprocessor DAG scheduling. *Computation*, MDPI AG, v. 8, n. 2, p. 26, abr. 2020.

WHITLEY, D.; RANA, S.; DZUBERA, J.; MATHIAS, K. E. Evaluating evolutionary algorithms. *Artificial Intelligence*, Elsevier BV, v. 85, n. 1-2, p. 245–276, aug 1996.

WILLIAMSON, D. P.; SHMOYS, D. B. *The design of approximation algorithms*. Cambridge, MA: Cambridge University Press, 2011. 504 p. ISBN 9780521195270.

XHAFA, F. A hybrid evolutionary heuristic for job scheduling on computational grids. In: ABRAHAM, A.; GROSAN, C.; ISHIBUCHI, H. (Ed.). *Hybrid Evolutionary Algorithms*. Heidelberg: Springer Berlin, 2007, (Studies in Computational Intelligence, v. 75). p. 269–311.

XHAFA, F.; ABRAHAM, A. Meta-heuristics for grid scheduling problems. In: XHAFA, F.; ABRAHAM, A. (Ed.). *Studies in Computational Intelligence*. Heidelberg: Springer Berlin, 2008, (Studies in Computational Intelligence, v. 146). p. 1–37.

_____. *Metaheuristics for Scheduling in Distributed Computing Environments*. Berlin: Springer London, Limited, 2008. v. 146. 364 p. (Studies in Computational Intelligence, v. 146). ISBN 9783540692775.

XHAFA, F.; ABRAHAM, A. (Ed.). *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. Berlin: Springer-Verlag Berlin Heidelberg, 2008. v. 128. 346 p. (Studies in Computational Intelligence, v. 128). ISBN 9783540789840.

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, Elsevier BV, v. 26, n. 4, p. 608–621, abr. 2010.

XHAFA, F.; ALBA, E.; DORRONSORO, B.; DURAN, B.; ABRAHAM, A. Efficient batch job scheduling in grids using cellular memetic algorithms. In: XHAFA, F.; ABRAHAM, A. (Ed.). *Studies in Computational Intelligence*. Heidelberg: Springer Berlin, 2008, (Studies in Computational Intelligence, v. 146). p. 273–299.

XHAFA, F.; BAROLLO, L.; DURRESI, A. An experimental study on genetic algorithms for resource allocation on grid systems. *Journal of Interconnection Networks*, World Scientific Pub Co Pte Lt, v. 08, n. 04, p. 427–443, dez. 2007.

XHAFA, F.; GONZALEZ, J. A.; DAHAL, K. P.; ABRAHAM, A. A GA(TS) hybrid algorithm for scheduling in computational grids. In: CORCHADO, E.; WU, X.; OJA, E.; HERRERO, Á.; BARUQUE, B. (Ed.). *Hybrid Artificial Intelligence Systems*. Heidelberg: Springer Berlin, 2009, (Lecture Notes in Computer Science, v. 5572). p. 285–292. ISBN 978-3-642-02319-4.

YOUNIS, M. T. *Hybrid Meta-heuristic Algorithms for Static and Dynamic Job Scheduling in Grid Computing*. 146 p. Tese (Doutorado) — De Montfort University, Leicester, UK, set. 2018.

YOUNIS, M. T.; YANG, S. Genetic algorithm for independent job scheduling in grid computing. *MENDEL*, Brno University of Technology, v. 23, n. 1, p. 65–72, jun. 2017.

_____. Hybrid meta-heuristic algorithms for independent job scheduling in grid computing. *Applied Soft Computing*, Elsevier BV, v. 72, p. 498–517, nov. 2018.

YOUNIS, M. T.; YANG, S.; PASSOW, B. Meta-heuristically seeded genetic algorithm for independent job scheduling in grid computing. In: SQUILLERO, G.; SIM, K. (Ed.). *Applications of Evolutionary Computation*. Cham: Springer International Publishing, 2017, (Lecture Notes in Computer Science, v. 10199). p. 177–189.

ZAMFIRACHE, F.; FRÎNCU, M.; ZAHARIE, D. Population-based metaheuristics for tasks scheduling in heterogeneous distributed systems. In: DIMOV, I.; DIMOVA, S.; KOLKOVSKA, N. (Ed.). *Numerical Methods and Applications*. Heidelberg: Springer Berlin, 2011, (Lecture Notes in Computer Science, v. 6046). p. 321–328.

ZHOU, Y.; ZHANG, J.; WANG, Y. Performance analysis of the $(1 + 1)$ evolutionary algorithm for the multiprocessor scheduling problem. *Algorithmica*, Springer Science and Business Media LLC, v. 73, n. 1, p. 21–41, jun. 2014.