UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

ALEXSANDRO OLIVEIRA ALEXANDRINO

MODELOS DE PROGRAMAÇÃO LINEAR INTEIRA PARA O PROBLEMA DE REARRANJO DE GENOMAS POR TRANSPOSIÇÃO

QUIXADÁ, CEARÁ

2016

ALEXSANDRO OLIVEIRA ALEXANDRINO

MODELOS DE PROGRAMAÇÃO LINEAR INTEIRA PARA O PROBLEMA DE REARRANJO DE GENOMAS POR TRANSPOSIÇÃO

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientador: Prof. Dr. Críston Pereira de Souza

Co-Orientador: Prof. Msc. Lucas Ismaily B. Freitas

QUIXADÁ, CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A371m Alexandrino, Alexsandro Oliveira.
Modelos de Programação Linear Inteira para o Problema de Rearranjo de Genomas por Transposição / Alexsandro Oliveira Alexandrino. – 2016.
54 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2016.
Orientação: Prof. Dr. Críston Pereira de Souza.
Coorientação: Prof. Me. Lucas Ismaily B. Freitas.

1. Programação linear. 2. Programação inteira. 3. Biologia computacional. 4. Otimização combinatória. I. Título.

CDD 005.1

ALEXSANDRO OLIVEIRA ALEXANDRINO

MODELOS DE PROGRAMAÇÃO LINEAR INTEIRA PARA O PROBLEMA DE REARRANJO DE GENOMAS POR TRANSPOSIÇÃO

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovada em: 15 de Setembro de 2016

BANCA EXAMINADORA

Prof. Dr. Críston Pereira de Souza (Orientador)
Campus Quixadá
Universidade Federal do Ceará – UFC

Prof. Msc. Lucas Ismaily B. Freitas (Co-Orientador)
Campus Quixadá
Universidade Federal do Ceará - UFC

Prof. Msc. Arthur Rodrigues Araruna
Campus Quixadá
Universidade Federal do Ceará - UFC

Prof. Msc. Paulo Henrique Macedo de Araujo
Campus Quixadá
Universidade Federal do Ceará - UFC

Aos meus pais, Pedro e Marileide.

Ao meu irmão, Alex.

AGRADECIMENTOS

Agradeço aos meus pais, Pedro e Marileide, por terem me dado a educação e apoio necessário para minha formação, e ao meu irmão, Alex Alexandrino, pela amizade e companheirismo que sempre tivemos.

Agradeço à Ana Paula dos Santos Dantas pelo apoio e companheirismo em todos os momentos.

Agradeço aos professores Críston Souza e Lucas Ismaily pela orientação neste trabalho e por terem contribuído na minha formação acadêmica.

Agradeço ao professor Samy Soares, que foi ao longo dos últimos anos meu orientador na bolsa do PET Tecnologia da Informação.

Agradeço aos professores Wladimir, Arthur Araruna, Diana, Carla, e Paulyne.

Agradeço aos meus amigos Wellington, Katyeudo, Yago, Anderson, Italos, Cayk, Luan, Tiago, Daiane, Micaele, Raul, Júlio, Jordy, Guilherme, Adeilson, Araújo, Leonel Junior, Wanrly, Sávio, e todos que conviveram comigo ao longo da minha formação.

E a todos que direta ou indiretamente fizeram parte da minha formação.

“Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth.”

(Sir Arthur Conan Doyle)

RESUMO

A evolução biológica é o conjunto de mudanças nas características hereditárias de uma população no decorrer do tempo. Um problema na área de biologia computacional é calcular a distância evolucionária entre espécies, sendo que na distância evolucionária consideramos apenas conjuntos de mutações que alteram pedaços do genoma, ao qual chamamos de rearranjos de genomas (SETUBAL; MEIDANIS, 1997). Na literatura estudada, os dois tipos de rearranjo mais observados são os de reversão e transposição. Este trabalho tem o objetivo de criar um novo modelo de programação linear inteira para o problema de rearranjo de genomas por transposição, e o de comparar com os modelos existentes na literatura. Também são propostas desigualdades válidas com o objetivo de melhorar a eficiência dos modelos, e aplicada a técnica de relaxação lagrangeana na tentativa de melhorar os limites inferiores. Concluímos que o modelo definido por Lancia, Rinaldi e Serafini (2015) possui melhores resultados práticos. As propostas de melhorias não trazem melhoras significativas aos modelos.

Palavras-chave: Biologia Computacional. Rearranjo de Genomas. Programação Linear. Programação Inteira. Desigualdades Válidas. Relaxação Lagrangeana.

ABSTRACT

The biological evolution is the set of changes in the heritable characteristics in a population over time. A problem in computational biology is to calculate the evolutionary distance among species, being that the evolutionary distance considers only sets of mutations that updates pieces of the genome, which we call genome rearrangements (SETUBAL; MEIDANIS, 1997). In the studied literature, the two most observed kinds of genome rearrangements are reversal and transposition. This study aims to create a new integer linear programming model for the genome rearrangement problem, and to compare it with the current models in literature. It is also proposed the addition of valid inequalities to improve the efficiency of the models, and it is applied the lagrangian relaxation to improve the lower bounds. We conclude that the model defined by Lancia, Rinaldi and Serafini (2015) has better practical results. The proposed improvements do not bring significant improvements to the models.

Keywords: Computational Biology. Genome Rearrangement. Linear Programming. Integer Programming. Valid Inequalities. Lagrangian Relaxation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

A evolução biológica é o conjunto de mudanças nas características hereditárias de uma população no decorrer do tempo. Estudos em rearranjo de genomas estão interessados em descobrir a sequência de rearranjos que transformam um genoma em outro, esse problema também é conhecido como distância evolutiva entre espécies (DIAS, 2012). Entende-se por rearranjo qualquer evento que altere a ordem dos genes, sendo que esses eventos podem ser de vários tipos distintos, desde a alteração da ordem de uma parte do genoma até a troca de partes inteiras de uma posição para outra (SETUBAL; MEIDANIS, 1997).

Ainda não é possível provar se uma determinada sequência de rearranjos reflete a real evolução entre duas espécies. Entretanto, utilizando-se do princípio da parcimônia, que a natureza utilizaria sempre a melhor solução ou solução mais simples, podemos determinar a distância evolucionária entre duas espécies A e B como o número mínimo de rearranjos necessários para transformar o genoma de A no de B. Tal distância nos dá uma boa aproximação do resultado do problema (DIAS, 2012).

Na literatura estudada, os dois tipos de rearranjo mais observados são os de reversão e transposição. Na reversão uma sequência de genes tem sua ordem invertida, já na transposição uma sequência de genes é destacada e inserida em outra posição.

Para a reversão, existe um algoritmo exato com tempo polinomial para um tipo de instância do problema em que os blocos de genes são orientados (HANNENHALLI; PEVZNER, 1999). Para os outros tipos foi provado que o problema é NP-Difícil (CAPRARA, 1999). O problema envolvendo apenas transposições também é NP-Difícil (BULTEAU; FERTIN; RUSU, 2012) e possui menos trabalhos na literatura que o problema de reversão. Para os dois tipos de rearranjo, o problema possui algoritmo aproximativo com razão de 1,375 (BERMAN; HANNENHALLI; KARPINSKI, 2002; ELIAS; HARTMAN, 2006). Dias (2012) apresenta um estudo detalhado com limites superiores, limites inferiores e heurísticas para uma nova versão do algoritmo de Elias e Hartman (2006) com melhores resultados práticos.

A programação linear é um método bastante utilizado para resolver problemas de otimização. Nela estamos interessados em conseguir o melhor resultado para um problema de minimização ou maximação de uma função linear, sujeito a uma série de restrições lineares (CORMEN, 2009). Quando as variáveis do modelo de programação linear devem ter valores inteiros, definimos esse modelo como de programação linear inteira (WOLSEY, 1998). Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007) definem modelos de programação linear inteira

para o problema. Esses modelos tem grande contribuição teórica para o assunto e são definidos para os principais tipos de rearranjos.

O modelo definido por Lancia, Rinaldi e Serafini (2015) é genérico para qualquer tipo de rearranjo e introduz o conceito de digrafo de camadas. Já Dias e Souza (2007) definem modelos específicos para cada tipo de rearranjo, sendo que esses modelos possuem restrições em comum que se distinguem apenas nas restrições que definem como cada rearranjo altera um genoma.

Para o melhor do nosso conhecimento, os únicos algoritmos exatos para o problema de rearranjo de genomas, que servem para todos os tipos de genomas, conseguem resolver apenas instâncias pequenas do problema, sendo seu uso na prática inviável para a comparação de genomas reais.

O objetivo deste trabalho é criar um novo modelo de programação linear inteira para o problema de rearranjo de genomas por transposição, e o comparar com os atuais existentes na literatura. Também é proposto o desenvolvimento de melhorias nos modelos atuais através da adição de desigualdades válidas, e aplicada a técnica de relaxação lagrangeana na tentativa de melhorar os limites inferiores. O foco em rearranjos de transposição se deve ao fato desta operação ter menos resultados na literatura. Apesar disso, algumas ideias desenvolvidas neste trabalho podem ser adaptadas para outros tipos de rearranjo.

Como etapa inicial do desenvolvimento do trabalho, foi realizada a implementação dos modelos definidos por Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007). Após isso, foi definido e implementado um novo modelo de programação inteira para o problema de rearranjos por transposição, sendo esse modelo baseado nos conceitos de emparelhamentos perfeitos e digrafo de camadas. Com a implementação dos três modelos, foi feita a execução de testes e análise de resultados a fim de comparar os três modelos em relação à quantidade de instâncias resolvidas. Ao modelo com melhores resultados foi aplicada a técnica de Relaxação Lagrangeana. Por fim, foram adicionadas desigualdades válidas nos modelos a fim de melhorar o tempo de execução.

Este trabalho está organizado da seguinte forma: no Capítulo 2, são descritos os conceitos utilizados neste trabalho; no Capítulo 3, são apresentados os trabalhos de Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007); no Capítulo 4, são descritos os procedimentos metodológicos deste trabalho; no Capítulo 5, o desenvolvimento e os resultados são descritos e analisados; por fim, no Capítulo 6, são apresentados os objetivos alcançados, conclusão e possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo está organizado da seguinte forma. A Seção 2.1 introduz conceitos de grafos que serão utilizados para a apresentação dos limitantes superiores e inferiores do problema (digrafo de ciclos), na apresentação das formulações de programação linear inteira existentes na literatura (digrafo de camadas), e na formulação de um novo programa linear inteiro para o problema estudado (digrafo de camadas e emparelhamentos perfeitos). A Seção 2.2 introduz o problema de rearranjo de genomas por transposição e apresenta toda a sua terminologia, assim como introduz resultados de trabalhos anteriores. Na Seção 2.3 introduzimos o conceito de programação linear inteira, que está presente nos trabalhos relacionados e também será utilizado na formulação proposta neste trabalho para a resolução do problema. Por fim, é apresentada a relaxação lagrangeana (Seção 2.4), que será utilizada neste trabalho com o intuito de conseguir melhores limitantes para o problema.

2.1 Grafos

Um **grafo** G é um par ordenado (V,E), sendo V um conjunto finito de elementos e E um conjunto de pares não ordenados de elementos pertencentes a V. Chamamos de **vértices** os elementos que pertencem a V, e de **arestas** os elementos pertencentes a E. Denotamos o número de vértices por $|V|$, e o número de arestas por $|E|$. De modo similar, definimos um digrafo D como um par ordenado (V,A), sendo V o conjunto de vértices e A o conjunto de arcos formado por pares ordenados de vértices. Um digrafo também pode ser definido como $G = (V,E)$ ou $G = (V,A)$ (PAPADIMITRIOU; STEIGLITZ, 1982; GOLDBARG, 2012).

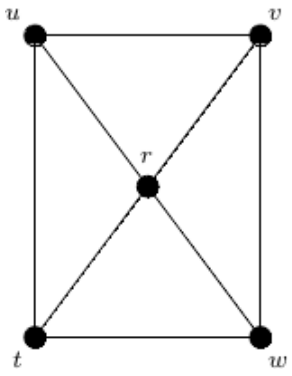
Em um grafo, caso exista uma aresta $e = uv$, tal que $e \in E$, dizemos que ela é incidente a u e a v e que os vértices u e v são adjacentes. Denotamos o **grau** de um vértice v como o número de arestas que incidem em v. **Arestas adjacentes** são aquelas que possuem pelo menos um dos extremos em comum, por exemplo, as arestas ab e bc são adjacentes por possuírem o extremo b em comum. Um **caminho** em um grafo G é definido como uma sequência de vértices distintos $P = < v_1, \ldots, v_n >$, tal que para todo $v_i \in V$, v_i é **adjacente** a v_{i+1}, com $1 \leq i \leq n-1$. Um **ciclo** é uma sequência de três ou mais vértices $C = < v_1, \ldots, v_n, v_1 >$, tal que vértices consecutivos na sequência são adjacentes, sendo v_1 o único vértice que se repete. Um grafo $G = (V,E)$ é **bipartido** se V pode ser particionado em dois conjuntos X e Y, tal que para toda aresta $uv \in E$ temos $u \in X$ e $v \in Y$.

Já em um digrafo, caso exista um arco $a = (u,v)$, tal que $a \in A$, dizemos que ele é

incidente a u e **emergente** a v. A principal diferença de um arco para uma aresta é que um arco é direcionado de um vértice a outro. Para um arco $a = (u,v) \in A$, em um digrafo, dizemos que a é um arco de **entrada** em v e de **saída** em u. A noção de grau para digrafos é dividida em grau de entrada e grau de saída. O **grau de entrada** de um vértice v é a quantidade de arcos de entrada em v. De modo similar, o **grau de saída** de um vértice v é a quantidade de arcos de saída em v. Um caminho orientado é uma sequência de vértices distintos $P = < v_1, \ldots, v_n >$ de D, tal que $(v_i, v_{i+1}) \in A$ para todo $i \in \{1, \ldots, n-1\}$. Um **ciclo orientado** é uma sequência de dois ou mais vértices $C =< v_1, \ldots, v_n, v_1 >$, onde todo arco é formado pelo par (v_i, v_{i+1}), com $1 \leq i \leq n$ e $v_{n+1} = v_1$. É importante notar que, em um ciclo orientado, a ordem dos vértices nos arcos é relevante. O conceito de digrafo bipartido é semelhante ao de grafo bipartido. Na Figura 2, $C_1 =< r,v,t,r >$ é um ciclo orientado.

Representamos graficamente um grafo como um conjunto de pontos (vértices) ligados por linhas (arestas). É possível utilizar rótulos, nos vértices e arestas, para facilidade de identificação. Na Figura 1 é apresentado um grafo com vértices rotulados.

Figura 1 – Exemplo de um grafo $G = (V,E)$ com o conjunto de vértices $V = \{u,v,r,w,t\}$ e arestas $E = \{uv, vw, tw, ut, ur, vr, rw, rt\}$.



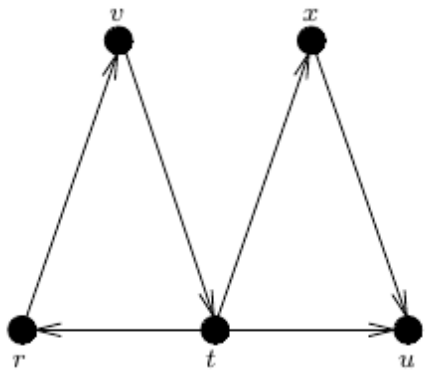
Fonte: Freitas (2014).

A única diferença para a representação de um digrafo é que, ao invés de linhas, utilizamos setas para representar arcos, sendo que a ponta da seta é direcionada ao vértice de entrada (Figura 2).

2.1.1 Emparelhamentos

Dado um grafo $G = (V,E)$, um **emparelhamento** M é um conjunto de arestas, em que todo par $a,b \in M$ é não adjacente. Com relação a M, um vértice é **saturado** se ele é um extremo de alguma aresta de M, caso contrário ele é chamado de **insaturado**. Dizemos que um

Figura 2 – Exemplo de um digrafo.



Fonte: Freitas (2014).

emparelhamento é **perfeito** se ele satura todos os vértices de G.

Para um digrafo $D = (V, A)$, um emparelhamento M é um subconjunto de A, tal que a soma dos graus de entrada e saída é menor ou igual a um, para todo vértice v no digrafo induzido por M. Um emparelhamento perfeito é aquele em que a soma dos graus de entrada e saída é sempre igual a um, para todo vértice v no digrafo induzido por M.

2.1.2 Digrafo de Ciclos

Os digrafos de ciclos foram introduzidos por Bafna e Pevzner (1998). Eles são utilizados nos algoritmos aproximativos e nos melhores limitantes conhecidos para o problema de Rearranjo de Genomas. Como será apresentado na Seção 2.2, representaremos um genoma como uma permutação de inteiros $\pi = (\pi_1 \pi_2 \dots \pi_n)$ de tamanho n.

Um digrafo $G(\pi)$, com arcos pretos e cinzas, é um **digrafo de ciclos** de π se $G(\pi)$ possui $n+2$ vértices rotulados de 0 a $n+1$, um conjunto de $n+1$ arcos pretos e um conjunto de $n+1$ arcos cinzas. Os arcos são definidos da seguinte forma:

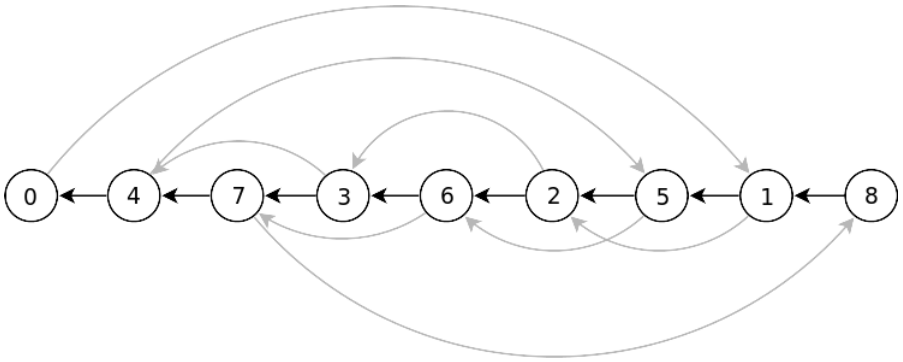
(i) Arcos cinzas vão do vértice $i-1$ ao vértice i, para todo $i \in \{1, \dots, n+1\}$;

(ii) Arcos pretos vão do vértice $\pi(i)$ ao vértice $\pi(i-1)$, para todo $i \in \{1, \dots, n+1\}$.

Portanto, podemos falar que os arcos pretos conectam vértices segundo a permutação π, e os arcos cinzas conectam vértices adjacentes segundo a permutação identidade (Figura 3).

Um **ciclo alternado** de $G(\pi)$ é um ciclo direcionado com arcos de cores alternadas. Existe uma decomposição única de ciclos alternados para os arcos de $G(\pi)$, pois para cada vértice em $G(\pi)$, todos os arcos de entrada são pareadas com um arco de saída de diferente cor (BAFNA; PEVZNER, 1998). Definimos um ciclo alternado com k arcos pretos como um ***k*-ciclo**, sendo k o **tamanho do ciclo**. Um ciclo pode ser classificado como par ou ímpar, dependendo do

Figura 3 – Digrafo de ciclos para a permutação $\pi = (4\ 7\ 3\ 6\ 2\ 5\ 1)$.



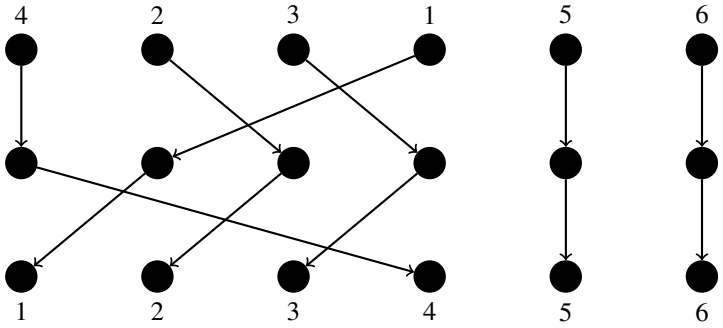
Fonte: Iizuka (2012).

tamanho do ciclo. Usamos a notação $c(G(\pi))$ para representar o número de ciclos em $G(\pi)$, e de modo similar $c_{impar}(G(\pi))$ representa o número de ciclos ímpares.

2.1.3 Digrafo de camadas

Um **digrafo de camadas** $D = (V,A)$ com k camadas, é um digrafo que pode ser particionado em k conjuntos de vértices. Denotamos cada partição como L_i, tal que $1 \leq i \leq k$ e i representa o número da partição. Assim, temos que para todos os vértices $a \in L_i$ e $b \in L_i$ não existe arco que conecte os dois, ou seja, $(a,b) \notin A$ e $(b,a) \notin A$ (Figura 4). Em especial, um grafo bipartido é um digrafo de camadas com $k = 2$. Ao longo do texto, chamaremos uma partição de camada.

Figura 4 – Exemplo de digrafo de camadas com $k = 3$.



Fonte: Elaborada pelo autor.

2.2 Rearranjo de Genomas

Espécies evoluíram ao longo do tempo a partir de mutações em suas populações. Enquanto muitas dessas mutações debilitam o indivíduo, o qual não consegue sobreviver, outras

mutações favorecem o indivíduo em relação a outros da sua espécie, proporcionando vantagens que o fazem sobreviver e propagar seus novos genes para as próximas gerações. A partir do avanço tecnológico e de algoritmos que permitiram o mapeamento do genoma de espécies, foi observado que espécies com ancestral comum possuem praticamente os mesmos genes, mas com ordenação diferente na sequência do genoma (JONES; PEVZNER, 2004). Isso permitiu a possibilidade de calcular quais eventos evolucionários ocorreram entre espécies distintas. Dado duas espécies s_1 e s_2 que partilham genes semelhantes, e por isso pressupõe-se que possuem um ancestral comum s_a, podemos traçar um caminho evolucionário de s_1 a s_a e de s_2 a s_a, permitindo assim a comparação de quais mudanças cada espécie teve em relação a um ancestral comum entre elas. O tamanho desse caminho é denominado distância evolucionária (SETUBAL; MEIDANIS, 1997).

Em vez de calcular mutações isoladas que afetam apenas nucleotídeos, estamos interessados em grupos de mutações que afetam trechos do genoma. Esses grupos são chamados de eventos de rearranjo de genomas, eventos evolucionários, **rearranjos de genomas** ou apenas rearranjos. Representamos um genoma como uma permutação de inteiros $\pi = (\pi_1 \pi_2 \dots \pi_n)$, onde $\pi_i \in \mathbb{N}, 0 < \pi_i \leq n$ e $i = j \Leftrightarrow \pi_i = \pi_j$.

Encontrar o caminho evolucionário exato ainda não é possível na prática, porém podemos nos utilizar do princípio da máxima parcimônia, de que a natureza sempre optaria pela melhor solução ou solução mais simples, e então esse caminho seria composto pelo menor número possível de rearranjos (DIAS, 2012).

Portanto, sejam π_1 e π_2 genomas de espécies distintas, o problema de rearranjo de genomas está interessado em calcular o número mínimo de rearranjos $\rho = \{\rho_1, \rho_2, \dots, \rho_k\}$, tal que a aplicação desses rearranjos a π_1 transforma-o em π_2, ou seja, $\rho_1 \rho_2 \dots \rho_k \pi_1 = \pi_2$.

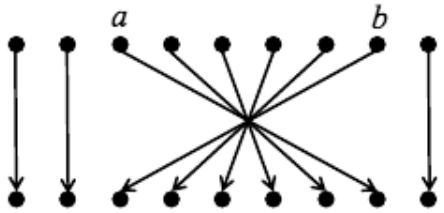
Os principais rearranjos de genomas são: reversões, transposições, translocações, fusões e fisões (DIAS, 2012). Os rearranjos mais estudados e que consequentemente possuem o maior número de resultados são:

- **Reversões**: um segmento do genoma tem sua ordem invertida (Figura 5a) (BAFNA; PEVZNER, 1996).
- **Transposições**: um segmento do genoma é destacado e inserido em uma posição diferente sem trocar a ordem desse segmento (Figura 5b) (BAFNA; PEVZNER, 1998).

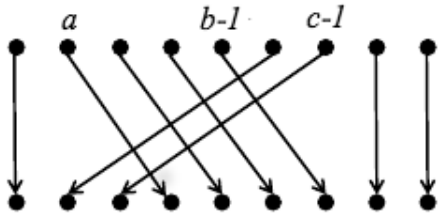
A maioria dos trabalhos encontrados definem algoritmos, heurísticas e limitantes, supondo que apenas um tipo de rearranjo ocorreu no genoma. Outros trabalhos, como o de Dias

Figura 5 – Exemplos de rearranjos em um digrafo de camadas.

(a) Exemplo de uma reversão $\rho(a,b)$ em um digrafo de camadas.



(b) Exemplo de uma transposição $\rho(a,b,c)$ em um digrafo de camadas.



Fonte: Elaborada pelo autor.

e Souza (2007), estudam a possibilidade de acontecerem múltiplos rearranjos ou combinações de rearranjos. Nesses casos, um rearranjo que possua características de reversão e transposição ao mesmo tempo pode ser tratado ou como duas operações distintas (uma reversão seguida por uma transposição, por exemplo) ou uma única operação que tenha características dos dois tipos de rearranjo.

Ao comparar dois genomas A e B de espécies distintas, é necessário realizar um mapeamento dos genomas com o intuito de encontrar blocos de genes comuns. Definimos como **bloco** um segmento contíguo que aparece nos genomas de A e B. O agrupamento em blocos se faz necessário devido à grande quantidade de genes no genoma, o que tornaria o trabalho de calcular a distância evolucionária computacionalmente inviável. Depois de mapeados, os blocos são numerados de 1 a n (quantidade de blocos). Esses blocos de genomas sempre possuem uma orientação, porém muitas vezes o mapeamento do genoma não é preciso o suficiente para que a informação de orientação seja armazenada, o que torna necessário o estudo do problema para blocos orientados e blocos não orientados (SETUBAL; MEIDANIS, 1997).

Em particular, considerando-se apenas blocos orientados e reversão, existe um algoritmo polinomial desenvolvido por Hannenhalli e Pevzner (1999). Para os demais tipos de entrada e uso exclusivo de reversão ou transposição, o problema é NP-Difícil (CAPRARA, 1999; BULTEAU; FERTIN; RUSU, 2012).

Neste trabalho, estamos interessados no estudo de rearranjo de genomas apenas por transposições. Foi escolhida a transposição depois do estudo bibliográfico, pois concluímos que ela possui menos resultados na literatura.

A Subseção 2.2.1 detalha os rearranjos de transposição e a Subseção 2.2.2 apresenta limitantes para o problema em que apenas tranposições são permitidas.

2.2.1 Transposição

Formalmente, definimos uma **transposição** $\rho(a,b,c)$, onde $1 \leq a < b < c \leq n+1$, em $\pi = (\pi_1 \ \pi_2 \ \ldots \ \pi_{n-1} \ \pi_n)$ como a operação $\rho\pi$ que resulta na permutação $(\pi_1 \ \ldots \ \pi_{a-1} \ \pi_b \ \ldots \ \pi_{c-1} \ \pi_a \ \ldots \ \pi_{b-1} \ \pi_c \ \ldots \ \pi_n)$ (Figura 5b).

Denotamos a **permutação identidade** como $\iota = (1\ 2\ 3\ \ldots\ n)$. Sendo a distância de tranposição $d(\pi,\sigma)$ o número mínimo de transposições $\rho_1, \rho_2 \ldots, \rho_t$ tal que $\rho_1\rho_2 \ \ldots \rho_t\pi = \sigma$, podemos reduzir o problema de distância por transposição entre π e σ ao problema de ordenação por transposição entre $\sigma^{-1}\pi$ e ι, onde $\sigma^{-1}\pi$ é gerado ao listar cada elemento de π na sua respectiva posição em σ. Assim, podemos tratar o problema como uma ordenação por transposição (DIAS, 2012).

Bafna e Pevzner (1998) apresentaram o primeiro algoritmo aproximativo que tem fator de 1,5. Desde esse trabalho, o fator de aproximação foi melhorado apenas por Elias e Hartman (2006), que definiram um algoritmo com razão de 1,375, sendo o melhor fator conhecido. Bafna e Pevzner (1998) introduziram conceitos importantes, como o digrafo de ciclos, que foram utilizados posteriormente no algoritmo de razão 1,375, e são utilizados neste trabalho para a geração de limites superiores e inferiores.

Chamamos o problema de distância evolucionária que envolve apenas transposições como **distância de transposição**, e a definimos como $d(\pi,\sigma)$, onde π é a permutação inicial e σ é a permutação final. A distância entre uma permutação π e a permutação identidade ι é denotada apenas por $d(\pi)$.

2.2.2 Limitantes para Distância de Transposição

Os limitantes conhecidos se baseiam em dois conceitos principais: *breakpoints* e digrafos de ciclo.

Um ***breakpoint*** é um par na permutação π que não está ordenado em relação a ι, ou seja, é um par (π_i, π_{i+1}) tal que $\pi_{i+1} \neq \pi_i + 1$. O número de *breakpoints* em uma permutação π é representado por $b(\pi)$. Por exemplo, a permutação $\pi = (1\ 2\ 5\ 3\ 4\ 6\ 7\ 9\ 8\ 10\ 11)$ possui seis *breakpoints*:

$$\pi = (1\ 2 \bullet 5 \bullet 3\ 4 \bullet 6\ 7 \bullet 9 \bullet 8 \bullet 10\ 11) \tag{2.1}$$

Chamamos de ***strip*** qualquer intervalo de elementos consecutivos sem *breakpoints*. Em (2.1) o intervalo $(6\ 7)$ é uma *strip*.

Uma transposição ρ afeta três posições de π. Portanto, ρ pode inserir ou remover até três *breakpoints*. Para qualquer π, sempre podemos encontrar uma transposição que retire pelo menos um *breakpoint* (DIAS, 2012). Assim, podemos definir os seguintes limitantes para π:

Limitante 2.2.1 $d(\pi) \leq b(\pi)$.

Limitante 2.2.2 $d(\pi) \geq \dfrac{b(\pi)}{3}$.

Utilizando-se digrafos de ciclos, Bafna e Pevzner (1998) apresentam dois limitantes para o problema:

Limitante 2.2.3 $\frac{n+1-c_{impar}(\pi)}{2} \leq d(\pi) \leq \frac{3(n+1-c_{impar}(\pi))}{4}$.

Limitante 2.2.4 $d(\pi) \geq \frac{n+1-c(\pi)}{2}$.

Os limitantes são utilizados para a otimização dos algoritmos, sendo que podemos descartar soluções que não estão entre os limites superior e inferior de uma instância do problema.

2.3 Programação Linear Inteira

A base da programação linear é determinar o valor ótimo de uma função linear sujeito a uma série de restrições lineares. Um problema dessa forma surge em uma grande variedade de situações reais, e a programação linear é um método que vem sendo utilizado em larga escala tanto teoricamente quanto na prática (THIE; KEOUGH, 2011). Também podemos definir programação linear como a otimização de alocação de recursos limitados entre atividades concorrentes, sobre um conjunto de restrições impostas pela natureza do problema (BRADLEY; HAX; MAGNANTI, 1977).

Considere um conjunto $A \subset \mathbb{R}$, um número $b \in \mathbb{R}$ e um conjunto de váriaveis x, sendo $n = |A| = |x|$. Portanto, podemos definir uma função linear e uma restrição linear como $f(x) = \sum_{i=1}^{n} a_i x_i$ e $\sum_{i=1}^{n} a_i x_i \square b$, $\square \in \{=, \leq, \geq\}$, respectivamente.

Definimos um modelo de programação linear, nesse caso um modelo é equivalente a um programa, da seguinte forma:

$$\max \text{ ou } \min cx \tag{2.2}$$

$$\text{sujeito a: } Ax \square b, \square \in \{=, \leq, \geq\} \tag{2.3}$$

$$x \geq 0 \tag{2.4}$$

Sejam A uma matriz $m \times n$, $b = (b_1\ b_2\ \ldots\ b_m)$ um vetor de tamanho m, $c = (c_1\ c_2\ \ldots\ c_n)$ um vetor de tamanho n de valores reais, e $x = (x_1\ x_2\ \ldots\ x_n)$ um vetor de variáveis. Generalizando a terminologia, chamamos (2.2) de **função objetivo**, (2.3) e (2.4) de **restrições**, e (2.4) de restrições de **não negatividade** (CORMEN, 2009).

Um conjunto X que satisfaz todas as restrições do modelo é chamado de **solução viável**, caso contrário é chamado de **solução inviável**. O valor da função objetivo é calculado a partir dos valores de X (2.2). Se estamos falando de um problema de minimização, a **solução ótima** x_o é o conjunto de valores que é uma solução viável e tem o menor valor dentre todas as soluções viáveis. Caso o problema seja de maximização, seguimos a mesma regra, com a exceção que o valor objetivo deve ser o maior entre todas as soluções viáveis. Um programa linear pode ainda não possuir um valor finito para a sua solução ótima, nesse caso dizemos que ele é **ilimitado** (CORMEN, 2009).

Em programação linear inteira, definimos como **desigualdades válidas** as restrições que diminuem o espaço de solução de um problema sem afetar o seu resultado. Assim, ao adicionar desigualdades válidas, espera-se que o modelo gerado seja resolvido de forma mais rápida. Em alguns casos, mesmo diminuindo o espaço de solução do problema, as desigualdades válidas adicionadas deixam o tamanho do modelo muito grande, tomando muito tempo para ser resolvido (WOLSEY, 1998).

2.3.1 Multicommodity Flow

Definimos uma **rede de fluxo** como um digrafo $D = (V,A)$ em que cada arco possui uma capacidade c não negativa. Consideramos que se $(u,v) \in A$, então $(v,u) \notin A$, e se $(u,v) \notin A$, então $c(u,v) = 0$, ou seja, não existem arcos em direção oposta a um arco existente, e arcos que não estão no digrafo terão capacidade zero. Em uma rede de fluxo existirá um vértice s chamado de **fonte** e outro vértice t chamado de **terminal**. Uma fonte s não possui arcos de entrada, e um terminal t não possui arcos de saída.

Denotamos um **fluxo** como uma função $f : V \text{ x } V \rightarrow \mathbb{R}$ que segue as seguintes propriedades (CORMEN, 2009):

- Restrição de Capacidade: O fluxo em cada arco deve ser menor ou igual a capacidade do arco. Para todo $u,v \in V$, temos que $0 \leq f(u,v) \leq c(u,v)$;
- Conservação de Fluxo: Para todo vértice que não seja fonte ou terminal, o seu

fluxo de entrada é igual ao fluxo de saída.

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u), \forall u \in V - \{s,t\} \tag{2.5}$$

No problema *multicommodity flow* temos uma rede de fluxo $D = (V,A)$ e existem k *commodities*. Uma ***commodity*** K_i é uma tripla (s_i,t_i,d_i), sendo s_i a fonte, t_i o vértice terminal, e d_i é um valor de demanda para o fluxo. Esse valor de demanda deve passar da fonte para o terminal ao decorrer do fluxo.

Cormen (2009) define a seguinte formulação de programação linear para o problema, em que a variável $f_{i,u,v}$ representa a quantidade de fluxo da *commodity i* que passa pelo arco (u,v) :

$$\min\ 0 \tag{2.6}$$

sujeito a:

$$\sum_{i=1}^{k} f_{i,u,v} \leq c(u,v), \forall u,v, \in V \tag{2.7}$$

$$\sum_{v \in V} f_{i,u,v} - \sum_{v \in V} f_{i,v,u} = 0, \forall i \in \{1,2,\ldots,k\} \text{ e } \forall u \in V - \{s_i,t_i\} \tag{2.8}$$

$$\sum_{v \in V} f_{i,s_i,v} - \sum_{v \in V} f_{i,v,s_i} = d_i, \forall i \in \{1,2,\ldots,k\} \tag{2.9}$$

$$f_{i,u,v} \geq 0, \forall i \in \{1,2,\ldots,k\} \text{ e } \forall u,v \in V \tag{2.10}$$

A (2.7) é uma restrição de capacidade, (2.8) é uma restrição de conservação de fluxo, (2.9) faz com que o fluxo seja igual à demanda de cada *commodity*, e (2.10) força que o fluxo seja sempre não negativo. Observe que a função objetivo (2.6) é nula, isso é devido ao fato de essa formulação estar interessada apenas em achar uma solução viável para o problema.

Como será apresentado posteriormente, uma das formulações de Lancia, Rinaldi e Serafini (2015) é definida tomando como base um *multicommodity flow*.

2.3.2 Programação Inteira

Para a programação linear, existem algoritmos de tempo polinomial em relação ao tamanho do modelo (número de variáveis e restrições), como por exemplo o *ellipsoid* e o método de ponto interior. Além desses, existe o simplex que, apesar de não ser polinomial no pior caso, geralmente possui melhores resultados na prática, sendo o mais comumente utilizado (CORMEN, 2009).

Um tipo especial de modelo de programação linear são os que possuem uma restrição adicional de que as suas variáveis possuem valores inteiros. A esse tipo de problema damos o nome de **programação linear inteira**, ou simplesmente programação inteira. A adição dessa restrição torna o problema NP-Difícil (CORMEN, 2009). Podemos ainda ter modelos em que apenas algumas variáveis possuem restrições de integralidade (**programação inteira mista**), e outros em que as variáveis possuem valores binários $\{0,1\}$ (**programação inteira binária**) (WOLSEY, 1998).

2.4 Relaxação Lagrangeana

Em otimização combinatória, encontrar boas soluções para problemas difíceis consiste em encontrar limitantes superiores e inferiores que possuam valor o mais próximo possível do valor ótimo. Considerando problemas de minimização, heurísticas podem ser utilizadas para encontrar bons limitantes superiores. Já para os limitantes inferiores, existem técnicas de relaxação, como a relaxação linear (BEASLEY, 1993).

Técnicas de relaxação consistem em retirar restrições que impedem que o problema seja resolvido de forma rápida, e como essas restrições são retiradas, o resultado do modelo relaxado é um limite inferior para o problema. Convém observar que estamos considerando problemas de minimização, caso o problema seja de maximação a relaxação nos dará um limite superior do problema.

A relaxação linear elimina as restrições de integralidade do modelo, transformando-o em um problema de programação linear. Outra técnica de relaxação é a relaxação lagrangeana. Beasley (1993) demonstra que, mesmo no pior caso, a relaxação lagrangeana gera um limite tão bom quanto o da relaxação linear, sendo que em muitos casos esse limite é melhor.

Considere a seguinte formulação de programação linear inteira:

$$\min\ cx \tag{2.11}$$

sujeito a:

$$Ax \geq b \tag{2.12}$$

$$Bx \geq d \tag{2.13}$$

$$x \in \{0,1\} \tag{2.14}$$

A relaxação lagrangeana consiste em escolher um conjunto de restrições a serem retiradas da formulação e colocadas na função objetivo com pesos definidos pelos multiplicadores

de lagrange. A adição desses multiplicadores tem o intuito de penalizar as soluções do modelo relaxado que desobedecem a restrição relaxada, isso é feito aumentando o valor da função objetivo através do multiplicador, de forma a tornar as soluções inviáveis no modelo original indesejáveis no modelo relaxado. Considere λ o conjunto de multiplicadores de lagrange e $Ax \geq b$ as restrições escolhidas, sendo assim:

$$\min \ cx + \lambda(b - Ax) \tag{2.15}$$

sujeito a:

$$Bx \geq d \tag{2.16}$$

$$x \in \{0,1\}, \lambda \geq 0 \tag{2.17}$$

Essa formulação é chamada de LLBP ("*Lagrangean Lower Bound Problem*", "Problema de Limite Inferior Lagrangeano" em tradução livre). Podemos então dividir a geração do modelo relaxado em duas partes:

- Questão estratégica: consiste em definir qual conjunto de restrições relaxar, pode ser definida a partir de tentativas ou por experiência com outros casos silimares.
- Questão tática: achar valores para os multiplicadores de lagrange que geram o melhor limite possível para o modelo relaxado.

Agora basta resolver o modelo relaxado. O valor ótimo do LLBP é um limite inferior para o problema original.

2.4.1 Dual Lagrangeano

Cada restrição relaxada R possui um λ_R associado. Se a restrição relaxada for uma igualdade, então λ_R é livre, caso contrário $\lambda_R \geq 0$. Do ponto de vista tático, temos que encontrar os valores dos multiplicadores de lagrange que nos deem o melhor limite inferior para o problema. Esses multiplicadores são encarados como constantes no LLBP, e para cada conjunto de multiplicadores nós temos um LLBP distinto. Então, estamos interessados em encontrar:

$$\max_{\lambda \geq 0} \left\{ \begin{array}{ll} \min & cx + \lambda(b - Ax) \\ \text{sujeito a:} & Bx \geq d \\ & x \in (0,1) \end{array} \right\} \tag{2.18}$$

Esse modelo é chamado de **dual lagrangeano**. No melhor caso, o valor ótimo do dual lagrangeano é igual ao valor ótimo do problema original. Para isso acontecer é necessário

que o resultado X do dual seja: viável no problema original e que $cX = [cX + \lambda(b - AX)]$ (BEASLEY, 1993).

2.4.2 Heurística Lagrangeana

A solução do lagrangeano pode não ser uma solução viável do problema original, portanto, devemos aplicar heurísticas para a transformação da solução do lagrangeano em uma solução viável. Após a aplicação da heurística, a solução se transforma em um limite superior do problema. Chamamos essa heurística de **heurística lagrangeana**.

Projetar uma heurística lagrangeana é uma arte, variando de problema para problema e não possuindo um passo a passo definido (BEASLEY, 1993). Para o nosso problema, podemos utilizar as heurísticas apresentadas por Dias (2012) que se baseiam no algoritmo apresentado por Elias e Hartman (2006), utilizando-se do conceito de digrafo de ciclos.

2.4.3 Método Subgradiente

"O método subgradiente é um procedimento iterativo o qual, a partir de um conjunto de multiplicadores de lagrange, gera novos multiplicadores de uma maneira sistemática" (BEASLEY, 1993, p. 267). Utilizaremos Z_{UB} e Z_{LB} como os valores do limite superior e limite inferior, respectivamente. Beasley (1993) divide esse procedimento nos seguintes passos:

1. Definir ε como um parâmetro de valor $0 < \varepsilon \leq 2$;
2. Inicializar Z_{UB} com alguma heurística do problema e λ com um conjunto de valores arbitrários;
3. Resolver o LLBP com os valores de λ atuais e calcular o valor de Z_{LB};
4. Definir subgradientes G_i para as restrições relaxadas, sendo que $G_i = b_i - \sum_{j=1}^{n} a_{ij}X_j; i \in \{1,\ldots,m\}$;
5. Definir o tamanho do passo $T = \dfrac{\varepsilon(Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} G_i^2}$;
6. Atualizar λ_i com $\lambda_i = \max(0, \lambda_i + TG_i), i = \{1,\ldots,m\}$;
7. Voltar ao passo 3. Caso as iterações comecem a repetir os valores do conjunto λ, deve-se diminuir o valor do parâmetro ε e definir um limite inferior para ε como condição de parada.

2.4.4 Considerações

Beasley (1993) apresenta dois motivos para a ampla utilização dessa abordagem: muitos problemas de otimização consistem em um problema fácil que é dificultado por uma séries de restrições extras, e então, retirar essas restrições extras gera uma formulação que pode ser resolvida com algum algoritmo conhecido; e a experiência prática do autor indica que os limites dados são bons, considerando a computação necessária para resolvê-los.

3 TRABALHOS RELACIONADOS

Neste Capítulo, apresentamos os trabalhos de Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007). Ambos os trabalhos propõem modelos de programação linear inteira para o problema.

3.1 Polynomial-sized ILP Models for Rearrangement Distance Problems

Dias e Souza (2007) formulam três modelos de programação inteira para os problemas de rearranjo de genomas, correspondendo às operações de transposições, de reversões, e a uma variante do problema em que ambas as operações são permitidas. Uma característica dessa formulação é que ela possui número polinomial de variáveis e restrições em relação ao tamanho da permutação de entrada.

A formulação está dividida em duas partes. Na primeira são definidas variáveis e restrições genéricas para todos os modelos, e que dizem respeito a geração de permutações válidas. A segunda parte apresenta restrições específicas para cada tipo de rearranjo de genomas.

Estamos interessados apenas na formulação que é específica para transposições. A seguir esse modelo é apresentado.

3.1.1 Definição das Variáveis e Função Objetivo do Modelo

Sejam π a permutação inicial e σ a permutação final. A variável B_{ijk}, para todo $1 \leq i, j \leq n$ e todo $0 \leq k < n$, indica se a i-ésima posição de π possui o valor j após o k-ésimo rearranjo. Ou seja,

$$B_{ijk} = \begin{cases} 1, & \text{se } \pi_i = j \text{ depois do } k\text{-ésimo rearranjo,} \\ 0, & \text{caso contrário.} \end{cases}$$

A variável t_{abck} indica qual transposição foi utilizada na k-ésima operação, para todo $1 \leq a < b < c \leq n+1$ e todo $1 \leq k < n$. Cada transposição é indicada por $\rho(a,b,c)$.

$$t_{abck} = \begin{cases} 1, & \text{se } \rho_k = \rho(a,b,c), \\ 0, & \text{caso contrário.} \end{cases}$$

Como há a possibilidade de que o k-ésimo rearranjo use a operação identidade, operação que não altera a posição dos elementos na permutação, é necessário definir a variável

t_k, para todo $1 \leq k < n$. Ela é útil para decidir se a k-ésima transposição alterou a permutação ou não.

$$t_k = \begin{cases} 1, & \text{se } \rho_k = \rho(x,y,z) \text{ e } \rho_k\rho_{k-1}\cdots\rho_1\pi \neq \rho_{k-1}\cdots\rho_1\pi, \\ 0, & \text{caso contrário.} \end{cases}$$

A função objetivo do modelo (3.1) tenta minimizar o número de operações para transformar π em σ.

$$\min \sum_{k=1}^{n-1} t_k \tag{3.1}$$

3.1.2 Definição das Restrições do Modelo

As restrições (3.2) e (3.3) fixam a permutação inicial e a permutação final.

$$B_{i,\pi_i,0} = 1, \text{ para todo } 1 \leq i \leq n. \tag{3.2}$$

$$B_{i,\sigma_i,n-1} = 1, \text{ para todo } 1 \leq i \leq n. \tag{3.3}$$

Para garantir que um valor aparece exatamente uma vez em cada camada são utilizadas as restrições (3.4) e (3.5), respectivamente.

$$\sum_{j=1}^{n} B_{ijk} = 1, \text{ para todo } 1 \leq i \leq n, 0 \leq k < n. \tag{3.4}$$

$$\sum_{i=1}^{n} B_{ijk} = 1, \text{ para todo } 1 \leq j \leq n, 0 \leq k < n. \tag{3.5}$$

Essas restrições são gerais para os três modelos. A seguir estão todas as restrições que especificam as operações de transposição. A restrição (3.6) garante que se a k-ésima transposição não alterar a permutação, então nenhuma das transposições seguintes poderão alterá-la. Para garantir que apenas uma transposição seja usada a cada iteração, é utilizada a restrição (3.7).

$$t_k \leq t_{k-1}, \text{ para todo } 1 \leq k < n. \tag{3.6}$$

$$\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \sum_{c=b+1}^{n+1} t_{abck} \leq t_k, \text{ para todo } 1 \leq k < n. \tag{3.7}$$

As restrições (3.8), (3.9) e (3.10) lidam com as modificações na permutação causadas por uma transposição $\rho(a,b,c)$. A análise é dividida em três casos:

1. $i < a$ ou $i \geq c$: posições são inalteradas por $\rho(a,b,c)$.

$$\sum_{a=i+1}^{n-1}\sum_{b=a+1}^{n}\sum_{c=b+1}^{n+1} t_{abck} + \sum_{a=1}^{n-1}\sum_{b=a+1}^{n}\sum_{c=b+1}^{i} t_{abck} + (1-t_k) + B_{i,j,k-1} - B_{ijk} \leq 1, \quad \text{para todo } 1 \leq i,j \leq n \text{ e todo } 1 \leq k < n. \tag{3.8}$$

2. $a \leq i < a+c-b$: após $\rho(a,b,c)$, estas posições serão ocupadas pelos elementos de índices b a $c-1$.

$$t_{abck} + B_{b-a+i,j,k-1} - B_{ijk} \leq 1, \quad 1 \leq a < b < c \leq n+1, a \leq i < a+c-b, 1 \leq j \leq n, 1 \leq k < n. \tag{3.9}$$

3. $a+c-b \leq i < c$: após $\rho(a,b,c)$, estas posições serão ocupadas pelos elementos de índices a a $b-1$.

$$t_{abck} + B_{b-c+i,j,k-1} - B_{ijk} \leq 1, \quad 1 \leq a < b < c \leq n+1, a+c-b \leq i < c, 1 \leq j \leq n, 1 \leq k < n. \tag{3.10}$$

3.2 A Unified Integer Programming Model for Genome Rearrangement Problems

Lancia, Rinaldi e Serafini (2015) propõem dois modelos, sendo um de tamanho exponencial. Ao contrário do que foi proposto por Dias e Souza (2007), cada um dos dois modelos é genérico para o uso de qualquer tipo de rearranjo.

A seguir serão apresentados conceitos em comum aos dois modelos.

3.2.1 Conceitos Básicos

Os dois modelos são definidos a partir de um digrafo de camadas. Entre duas camadas de vértices existem n^2 arcos, n corresponde ao número de elementos da permutação π, ou seja, duas camadas consecutivas induzem um grafo bipartido completo, com cada camada contendo n vértices. Cada arco conectando dois vértices i e j de camadas consecutivas representa que o i-ésimo elemento de π passou para a j-ésima posição em π. Assim, podemos dizer que os arcos conectando os vértices da camada k à camada $k+1$ representam as novas posições dos vértices após o k-ésimo rearranjo.

O grafo possui L camadas de arcos e $L+1$ camadas de vértices, sendo L um limite superior do problema e podendo ser calculado a partir de heurísticas presentes na literatura.

Observe que a k-ésima camada de arcos representa o k-ésimo rearranjo que define as novas posições dos elementos da camada de vértices $k+1$. A primeira camada de vértices do grafo representa a permutação inicial π, e a última camada representa a permutação identidade. Como L é um limite superior para o problema, a permutação identidade pode surgir antes da última camada.

Lancia, Rinaldi e Serafini (2015) definiram um conjunto de rearranjos O a fim de manter o modelo geral para qualquer tipo de rearranjo. Cada elemento $\mu \in O$ representa um rearranjo que pode ser realizado para sair da camada k para a $k+1$. Esses rearranjos são definidos como conjuntos de arcos, sendo que cada arco $(a,b) \in \mu$ representa que o vértice na posição a da camada k passará para a posição b na camada $k+1$. Consideramos que $O = \{\mu_0, \mu_1, \ldots, \mu_m\}$, tal que μ_0 é o rearranjo nulo, ou seja, $\mu_0 = \{(1,1),(2,2),\ldots,(n,n)\}$. O rearranjo nulo é necessário devido ao fato de que a permutação ordenada pode aparecer antes da última camada.

3.2.2 Modelo Básico

Sejam π uma permutação com n elementos, $D = (V,A)$ o digrafo de camadas com camada inicial definida a partir de π, e O o conjunto de rearranjos mapeados a partir da permutação π. O modelo possui variáveis associadas a caminhos de D, que o torna um modelo exponencial, e variáveis associadas ao uso dos rearranjos em O.

O conjunto de caminhos $P(i)$ em D iniciam no vértice i da primeira camada e terminam no vértice π_i da última camada. Sendo assim, atribuímos a cada caminho em $P(i)$ uma variável binária x_P, que indica se o caminho P está ou não na solução. Note que o caráter exponencial do modelo está no fato de que existem n^L caminhos.

A variável binária z_μ^k, para $k \in \{1,\cdots,L\}$ e $\mu \in O$, representa se o rearranjo μ foi utilizado na camada k de D. Podemos interpretar z_μ^k como a ativação do conjunto de arcos μ na camada k.

A função objetivo do modelo pretende minimizar o número de operações não nulas utilizadas na solução (3.11). O modelo completo é descrito como:

$$\min \sum_{k=1}^{L} \sum_{\mu \in O - \{\mu_0\}} z_\mu^k \tag{3.11}$$

sujeito a:

$$\sum_{\mu \in O} z_{\mu}^{k} = 1, \forall k \in \{1, \cdots, L\} \tag{3.12}$$

$$\sum_{P \in P(i)} x_P = 1, \forall i \in \{1, \cdots, n\} \tag{3.13}$$

$$\sum_{i=1}^{n} \sum_{P \in P(i) | a \in P} x_P \leq \sum_{\mu \in O | (u,v) \in \mu} z_{\mu}^{k}, \forall a = (u^k, v^{k+1}) \in A \tag{3.14}$$

$$z_{\mu_0}^{k} \leq z_{\mu_0}^{k+1}, \forall k \in \{1, \cdots, L-1\} \tag{3.15}$$

$$z_{\mu}^{k} \in \{0,1\}, x_P \geq 0, \forall k \in \{1, \cdots, L\}, \forall \mu \in O, \forall P \in \cup_i P(i) \tag{3.16}$$

A restrição (3.12) exige que apenas um rearranjo seja usado por camada, já a (3.13) assegura que apenas um caminho partirá de cada vértice da primeira camada. Os caminhos utilizados em determinada camada devem pertencer apenas aos arcos ativadas naquela camada, isso é definido em (3.14), sendo que A é o conjunto de todos os arcos possíveis no digrafo de camadas que conectam um vértice da camada k a camada $k+1$. A restrição (3.15) agrupa as operações identidade nas últimas camadas, reduzindo assim o número de soluções redundantes no espaço de soluções. A última restrição (3.16) serve para definir a variável z como binária, e para que x tenha valores não negativos.

3.2.3 Modelo Compacto Baseado em Multicommodity Flow

São definidos dois tipos de variáveis: z e x. A variável binária z_{μ}^{k}, para $k \in \{1, \ldots, L\}$ e $\mu \in O$, tem $z_{\mu}^{k} = 1$ se o rearranjo μ foi utilizado para ir da camada k para a $k+1$, assim como no modelo básico. Já a variável x_{ab}^{ki}, para cada $k \in \{1, \ldots, L\}$, cada fonte $i \in \{1, \ldots, n\}$, e cada (a,b) representando um arco, tal que a pertence à camada k e b pertence à camada $k+1$, representa, por sua vez, a quantidade de fluxo da *commodity* i que passa pelo arco (a,b). A função objetivo tem como intuito minimizar o número de rearranjos não nulos, ou seja:

$$\min \sum_{k=1}^{L} \sum_{\mu \in O - \mu_0} z_{\mu}^{k} \tag{3.17}$$

As restrições do modelo são :

- Restrições de saída de fluxo:

$$\sum_{j=1}^{n} x_{ij}^{1i} = 1, \forall i \in \{1, \ldots, n\} \tag{3.18}$$

- Restrições de entrada de fluxo:

$$\sum_{j=1}^{n} x_{j\pi_i}^{Li} = 1, \forall i \in \{1, \ldots, n\} \tag{3.19}$$

- Restrições de conservação de fluxo:

$$\sum_{j=1}^{n} x_{aj}^{ki} - \sum_{j=1}^{n} x_{ja}^{k-1,i} = 0, \forall i, a \in \{1,\ldots,n\}, \forall k \in \{2,\ldots,L\} \tag{3.20}$$

Além dessas, temos as restrições (3.21), (3.22) e (3.23). Em (3.21) o objetivo é garantir que apenas arcos que estão no rearranjo μ, utilizado em determinada camada k, possam ser utilizados. Já (3.22) é utilizada para obrigar que os rearranjos nulos sempre fiquem nas últimas camadas. E (3.23) define que cada camada possui apenas um rearranjo μ utilizado.

$$\sum_{i=1}^{n} x_{ab}^{ki} \leq \sum_{\mu \in O | (a,b) \in \mu} z_{\mu}^{k}, \forall a, b \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,L\} \tag{3.21}$$

$$z_{\mu_0}^{k} \leq z_{\mu_0}^{k+1}, \forall k \in \{1,\ldots,L-1\} \tag{3.22}$$

$$\sum_{\mu \in O} z_{\mu}^{k} = 1, \forall k \in \{1,\ldots,L\} \tag{3.23}$$

$$z_{\mu}^{k} \in \{0,1\}, x_{ab}^{ki} \geq 0, \forall k \in \{1,\ldots,L\}, \forall \mu \in O, \forall i, a, b \in \{1,\ldots,n\} \tag{3.24}$$

4 PROCEDIMENTOS METODOLÓGICOS

O trabalho foi planejado e desenvolvido em quatro macro etapas (Figura 6): implementação dos modelos de Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007) (Etapa 1); criação de um novo modelo de programação linear inteira com o uso de digrafo de camadas e emparelhamentos perfeitos (Etapa 2); comparação experimental dos modelos (Etapa 3); propostas de melhorias nos modelos (Etapa 4). É importante notar que as Etapas 3 e 4 aconteceram de forma iterativa, sendo que cada proposta de melhoria é acompanhada por outra fase de realização de testes e análise de resultados.

Na Figura 6 cada subconjunto de passos está em uma etapa. As Etapas 3 e 4 aconteceram várias vezes no decorrer do desenvolvimento do trabalho. Como todos os testes usam as mesmas instâncias, o passo de preparação de instâncias (Etapa 3) ocorreu apenas uma vez.

4.1 Implementação dos Modelos de Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007)

Inicialmente, foi implementado o modelo compacto baseado em *Multicommodity Flow* definido por Lancia, Rinaldi e Serafini (2015) e apresentado na Seção 3.2. Esse modelo foi escolhido por ter um número polinomial de variáveis e restrições em relação à entrada do modelo. Em seguida, foi implementado o modelo específico para transposições apresentado na Seção 3.1 e definido por Dias e Souza (2007).

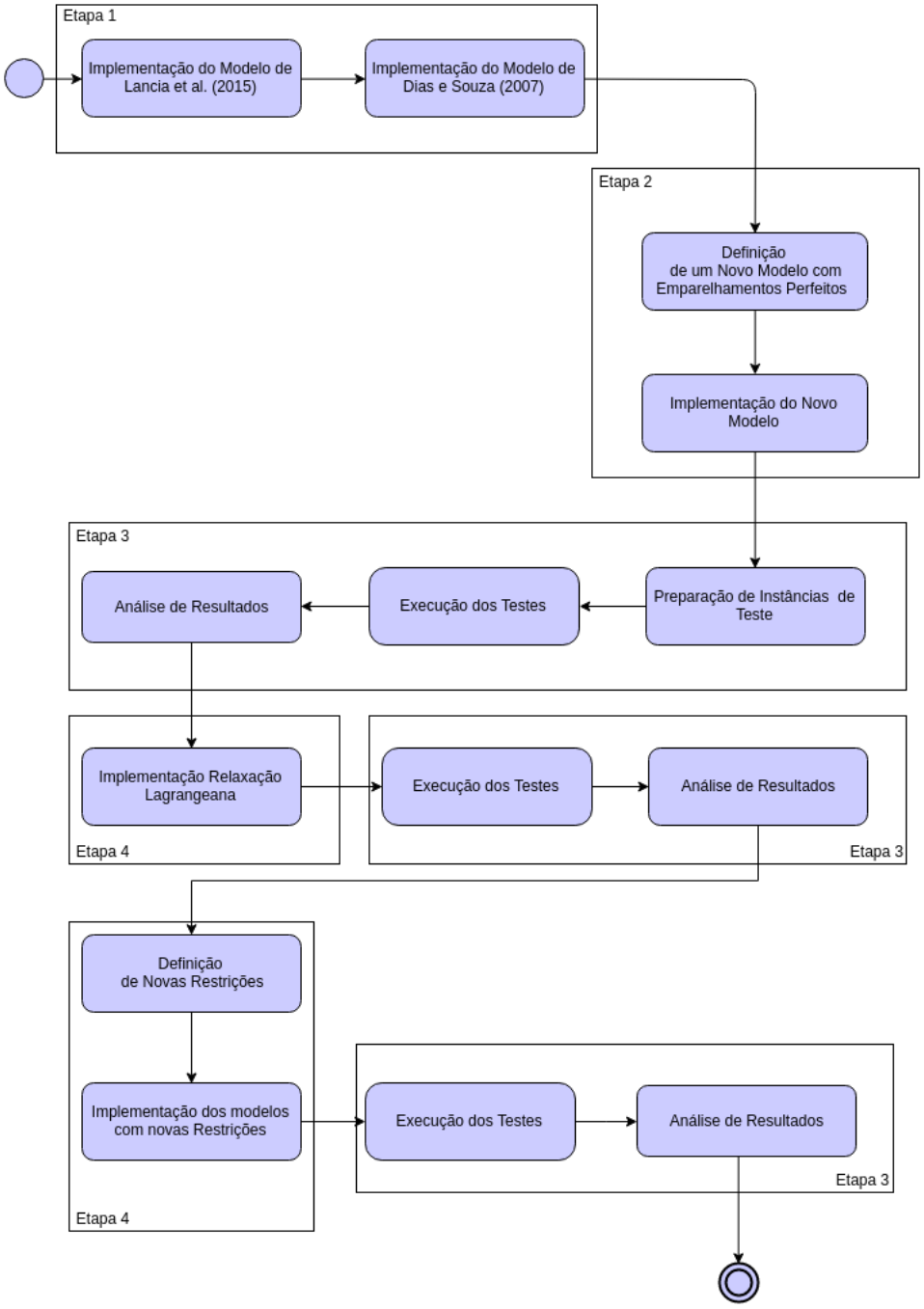
4.2 Criação de um Novo Modelo com Emparelhamentos Perfeitos

A ideia principal dessa etapa foi utilizar-se dos conceitos de digrafo de camadas e emparelhamentos perfeitos para a criação de um novo modelo de programação linear inteira. A definição do modelo, com suas variáveis e restrições, é apresentada na Seção 5.1.

4.3 Comparação Experimental dos Modelos

Para a etapa de comparação, as instâncias de teste foram divididas em três grupos, sendo que os dois primeiros grupos seguiram os seguintes padrões: $\pi X = (n\ n-1\ n-2\ \ldots\ 2\ 1)$; e $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$. Para o último grupo, foram criadas instâncias aleatórias para cada tamanho.

Figura 6 – Fluxograma dos Procedimentos Metodológicos.



Fonte: Elaborada pelo autor.

Nessa fase, também houve a preparação do servidor em que os testes foram executados e definição do tempo limite de duas horas para execução de cada instância de teste. Após a preparação do servidor, todos os três modelos foram submetidos a testes com o intuito de compará-los com relação ao tempo de execução e quantidade de instâncias resolvidas. Então, os resultados dos testes foram analisados a fim de descobrir qual modelo consegue resolver instâncias com o maior tamanho de entrada.

4.4 Propostas de Melhorias nos Modelos

Ao modelo com melhores resultados da etapa anterior, foi aplicada a técnica de Relaxação Lagrangeana apresentada na Seção 2.4. Além disso, foram propostas melhorias a partir da adição de desigualdades válidas aos modelos. Ao final de cada um dos dois passos, foram realizados novos testes para comparação entre os modelos com e sem as desigualdades válidas.

5 DESENVOLVIMENTO E RESULTADOS

A primeira etapa deste trabalho envolveu a implementação dos modelos apresentados nos trabalhos relacionados. Detalhes da implementação serão apresentados na Seção 5.2, em que serão apresentados os testes desses modelos e do novo modelo criado.

A seguir são apresentados detalhes sobre o desenvolvimento deste trabalho. A Seção 5.1 apresenta a definição do novo modelo. A Seção 5.2 descreve a realização dos testes de comparação dos modelos. A Seção 5.3 descreve os passos para a aplicação da técnica de relaxação lagrangeana. Por último, a Seção 5.4 apresenta a definição e adição de desigualdades válidas aos modelos, e reexecução dos testes.

5.1 Proposta de Modelo Baseado em Emparelhamentos Perfeitos

A partir dos conceitos de digrafo de camadas (Subseção 2.1.3) e emparelhamentos perfeitos (Subseção 2.1.1), foi criado um novo modelo que será apresentado a seguir.

Considere a seguinte notação:

- T_k é um conjunto com as transposições possíveis no nível k;
- V_k é um vetor que contém os vértices no nível k;
- M_k é um emparelhamento perfeito que representa os arcos da solução no nível k;
- O emparelhamento identidade é denotado por M^*;
- L é um limitante superior para o problema;
- A variável binária B_{ijk} denota se o arco ij está presente no emparelhamento do nível k:

$$B_{ijk} = \begin{cases} 1, & \text{se } (i,j) \in M_k, \\ 0, & \text{caso contrário.} \end{cases}$$

- A variável binária t_k denota se o emparelhamento de uma camada é distinto do emparelhamento identidade:

$$t_k = \begin{cases} 1, & \text{se } M_k \neq M*, \\ 0, & \text{caso contrário.} \end{cases}$$

A formulação consiste em transformar a permutação π na permutação σ, onde σ pode ser interpretado como a permutação identidade na implementação do modelo. Assim, consideramos um digrafo de camadas, em que a primeira camada de vértices possui a permutação π e a última camada de vértices corresponde à permutação σ. Usaremos o limite superior L como

limitante da quantidade de camadas de arcos, pois cada camada representa uma transposição. Desse modo, temos um digrafo de camadas com $L+1$ camadas de vértices e L camadas de arcos.

O modelo é definido como:

$$\min \sum_{k=1}^{L} t_k \tag{5.1}$$

sujeito a:

$$\sum_{i \in V_{k-1}} B_{ijk} = 1, \forall j \in V_k, \forall k \in \{1,\ldots,L\} \tag{5.2}$$

$$\sum_{j \in V_{k-1}} B_{ijk} = 1, \forall i \in V_k, \forall k \in \{1,\ldots,L\} \tag{5.3}$$

$$B_{i\sigma_i L} = 1, \forall i \in V_L, \forall \sigma_i \in V_{L+1} \tag{5.4}$$

$$M_k \in T_k \cup M^*, \forall k \in \{1,\ldots,L\} \tag{5.5}$$

$$t_k = 1, M_k \in T_k, \forall k \in \{1,\ldots,L\} \tag{5.6}$$

$$t_k = 0, M_k = M^*, \forall k \in \{1,\ldots,L\} \tag{5.7}$$

$$t_k \leq t_{k+1}, \forall k \in \{1,\ldots,L-1\} \tag{5.8}$$

$$t_k \in \{0,1\}, B_{ijk} \in \{0,1\}, \forall k \in \{1,\ldots,L\} \tag{5.9}$$

Para facilidade de entendimento do modelo, algumas restrições apresentadas não são lineares. A transformação dessas restrições não lineares para restrições lineares foi feita com a utilização da biblioteca do solver CPLEX da IBM (CPLEX, 2009). Para garantir que os arcos da solução formem emparelhamento perfeitos em cada camada de arcos, usamos (5.2) e (5.3). A restrição (5.4) força que a última camada de vértices possua a permutação σ. A (5.5) garante que sempre utilizamos operações válidas para cada camada, e (5.6) e (5.7) definem o valor da variável t_k. A última restrição (5.8) agrupa as operações identidade nas últimas camadas, reduzindo assim o número de soluções redundantes no espaço de soluções.

Como função objetivo, queremos minimizar o número de operações não nulas utilizadas (5.1). Note que se $M_k \neq M^*$, então $t_k = 1$.

5.2 Comparação Experimental dos Modelos

Utilizando-se da linguagem C++ e da biblioteca do *solver* CPLEX da IBM (CPLEX, 2009), foram implementados os modelos definidos por Lancia, Rinaldi e Serafini (2015) e Dias e Souza (2007), assim como o novo modelo apresentado na Seção 5.1. Para estimar o número de camadas do grafo (L), foram utilizadas as heurísticas apresentadas por Dias e Souza (2007).

Os testes foram realizados em um servidor com um processador octacore de 3,0GHz, 8GB de RAM e executando o sistema operacional Ubuntu 14.04.3. As instâncias de teste utilizadas eram de três formas: i) πX, tal que $\pi X = (n\ n-1\ \ldots\ 1)$; ii) πY, tal que $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$; iii) permutações aleatórias. As instâncias πX e πY foram escolhidas por serem famílias de permutações conhecidas (DIAS; SOUZA, 2007). Para todo $\pi \in \pi X$, π possui $n-1$ *breakpoints*. Já para todo $\pi \in \pi Y$ de tamanho n, se n é par, então π possui $n-1$ *breakpoints*, caso contrário π possui $n-2$ *breakpoints*. Foi estipulado o tempo limite de duas horas para execução dos testes, sendo abortadas as instâncias que não conseguissem executar nesse tempo. Para os testes com instâncias aleatórias, foram geradas cinco instâncias para cada tamanho de entrada, sendo que são apresentados o tempo médio para cada tamanho de entrada. Os testes posteriores utilizaram o mesmo servidor e as mesmas instâncias de teste. Os resultados são mostrados nas Tabelas 1, 2 e 3.

É importante notar que a biblioteca do CPLEX calcula a soma do tempo de uso de cada núcleo do processador (CPLEX, 2009). Como o servidor possui oito núcleos, notamos que na última instância (Tabela 2), por exemplo, o modelo de Lancia, Rinaldi e Serafini (2015) apresenta tempo de 55462,5 segundos, o que excede o tempo limite de duas horas, porém esse valor reflete a soma do tempo de uso de cada núcleo e o tempo limite reflete o tempo real decorrido desde o início até o fim da execução.

Tabela 1 – Resultados dos testes utilizando permutacões πY para os modelos de Lancia, Rinaldi e Serafini (2015), Dias e Souza (2007) e a formulação com emparelhamentos perfeitos.

Permutações $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,02	0,01	0,04
5	0,02	0,04	0,15
6	0,77	1,19	0,85
7	0,90	1,48	3,60
8	8,84	25,99	1315,85
9	23,26	521,6	446,29
10	4558,68	21939,10	timeout
11	24627,00	timeout	timeout
12	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

Tabela 2 – Resultados dos testes utilizando permutacões πX para os modelos de Lancia, Rinaldi e Serafini (2015), Dias e Souza (2007) e a formulação com emparelhamentos perfeitos.

Permutações $\pi X = (n\ n-1\ \ldots\ 1)$			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,10	0,01	0,23
5	0,47	0,33	1,37
6	5,77	2,02	45,20
7	27,97	101,19	3420,28
8	446,02	7301,63	timeout
9	3795,80	53355,00	timeout
10	55462,50	timeout	timeout
11	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

Tabela 3 – Resultados dos testes utilizando permutações aleatórias para os modelos de Lancia, Rinaldi e Serafini (2015), Dias e Souza (2007) e a formulação com emparelhamentos perfeitos.

Permutações Aleatórias - Tempo Médio de Execução			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,01	0,01	0,01
5	0,125	0,14	0,13
6	0,68	0,45	3,38
7	9,315	7,57	114,47
8	31,31	287,29	1075,95
9	706,97	6391,27	384,09
10	5286,78	timeout	timeout
11	39348,43	timeout	timeout
12	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

A partir dos resultados, conclui-se que, para as instâncias utilizadas nos testes, o modelo definido por Lancia, Rinaldi e Serafini (2015) tem melhores resultados do que os outros dois modelos em todos os tipos de instâncias, pois ele possui menores tempos de execução, e resolve instâncias que os demais não resolvem dentro do limite de tempo estabelecido.

5.3 Relaxação Lagrangeana no modelo de (LANCIA; RINALDI; SERAFINI, 2015)

A partir do modelo baseado em *multicommodity flow* de Lancia, Rinaldi e Serafini (2015), definido na Subseção 3.2.3, construímos uma nova formulação utilizando a relaxação lagrangeana.

Nossa decisão estratégica foi a exclusão da restrição (3.22) e relaxação das restrições (3.21) e (3.23), colocando-as na função objetivo com multiplicadores β e α para (3.21) e (3.23), respectivamente:

$$\min \sum_{k=1}^{L} \sum_{\mu \in O | \mu \neq \mu_0} z_\mu^k + \sum_{k=1}^{L} \alpha_k (1 - \sum_{\mu \in O} z_\mu^k) + \sum_{k=1}^{L} \sum_{a=1}^{n} \sum_{b=1}^{n} \beta_{ab}^k (\sum_{i=1}^{n} x_{ab}^{ki} - \sum_{\mu \in O | ab \in \mu} z_\mu^k) \tag{5.10}$$

$$= \min \sum_{k=1}^{L} \sum_{\mu \in O} (z_\mu^k - \alpha_k z_\mu^k - \sum_{a,b \in \mu} \beta_{ab}^k z_\mu^k) - \sum_{k=1}^{L} z_{\mu_0}^k + \sum_{k=1}^{L} \alpha_k + \sum_{k=1}^{L} \sum_{i=1}^{n} \sum_{a=1}^{n} \sum_{b=1}^{n} \beta_{ab}^k x_{ab}^{ki} \tag{5.11}$$

$$= \min \sum_{k=1}^{L} \sum_{\mu \in O} (1 - \alpha_k - \sum_{a,b \in \mu} \beta_{ab}^k) z_\mu^k - \sum_{k=1}^{L} z_{\mu_0}^k + \sum_{k=1}^{L} \alpha_k + \sum_{k=1}^{L} \sum_{i=1}^{n} \sum_{a=1}^{n} \sum_{b=1}^{n} \beta_{ab}^k x_{ab}^{ki} \tag{5.12}$$

$$\alpha_k \text{ é livre}, \beta_{ab}^k \geq 0. \tag{5.13}$$

Seja $\overline{c}_\mu^k = 1 - \alpha_k - \sum_{a,b \in \mu} \beta_{ab}^k$, definimos o valor de z da seguinte forma:

- Para $\mu \neq \mu_0$,

$$z_\mu^k = \begin{cases} 1, & \text{se } \overline{c}_\mu^k < 0 \\ 0, & \text{caso contrário.} \end{cases}$$

- Para $\mu = \mu_0$,

$$z_{\mu_0}^k = \begin{cases} 1, & \text{se } \overline{c}_\mu^k - 1 < 0 \\ 0, & \text{caso contrário.} \end{cases}$$

Podemos determinar as variáveis x incluindo a restrição reduntante "capacidade unitária nos arcos":

$$\sum_{i=1}^{k} x_{ab}^{ki} \leq 1, \forall a,b \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,L\}. \tag{5.14}$$

Formulações do lagrangeano com a propriedade de integralidade (poliedro com vértices inteiros) fazem com que a relaxação lagrangeana forneça o mesmo limite da relaxação linear (remoção das restrições de integralidade das variáveis do modelo original) (BEASLEY, 1993). Para o *multicommodity flow*, não temos a propriedade de integralidade. Apesar de ainda ser um problema NP-Completo (EVEN; ITAI; SHAMIR, 1975), ele pode ser resolvido para instâncias maiores do que as já conseguidas nas outras três formulações.

5.3.1 Subgradiente

Nossa escolha tática é o uso do subgradiente para a escolha dos multiplicadores de Lagrange. Definimos os subgradientes G_k para α e G^k_{ab} para β como:

$$G_k = 1 - \sum_{\mu \in O} z^k_\mu, \forall k \in \{1,\ldots,L\}; \tag{5.15}$$

$$G^k_{ab} = \sum_{i=1}^{n} x^{ki}_{ab} - \sum_{\mu \in O | ab \in \mu} z^k_\mu, \forall a,b \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,L\}. \tag{5.16}$$

O valor do passo T é calculado para cada subgradiente:

$$T_1 = \frac{\varepsilon(Z_{UB} - Z_{LB})}{\sum_{i=1}^{n} G_i^2}, \text{para o subgradiente } G_k; \tag{5.17}$$

$$T_2 = \frac{\varepsilon(Z_{UB} - Z_{LB})}{\sum_{k=1}^{L}\sum_{a=1}^{n}\sum_{b=1}^{n}(G^k_{ab})^2}, \text{para o subgradiente } G^k_{ab}. \tag{5.18}$$

A partir dos valores dos subgradientes, atualizamos os valores dos multiplicadores de lagrange, a cada iteração, da seguinte forma:

$$\alpha_k \leftarrow \alpha_k + T_1 G_k, \forall k \in \{1,\ldots,L\}; \tag{5.19}$$

$$\beta^k_{ab} \leftarrow \max\{0, \beta^k_{ab} + T_2 G^k_{ab}\}, \forall a,b \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,L\}. \tag{5.20}$$

Z_{LB} corresponde ao valor ótimo do lagrangeano e Z_{UB} ao valor de limitante superior. Utilizamos o algoritmo aproximativo de Dias (2012) para calcular o valor de Z_{UB}.

Após cada iteração, precisamos transformar a solução do lagrangeano em uma solução viável (heurística lagrangeana). Para isso, a cada nível $k \in \{1,\ldots,L-1\}$, escolhemos uma transposição μ tal que μ possua a maior quantidade de arcos da solução daquela camada. Após transformar a solução do lagrangeano em solução viavél, atualizamos o valor de Z_{UB} caso o novo limitante seja melhor.

Iniciamos com o parâmetro $\varepsilon = 2$, e adicionamos o parâmetro $N = 30$ para o limite de iterações sem alterações no valor de Z_{LB}. A cada N iterações sem alteração de Z_{LB}, o contador é reiniciado e o valor de ε é divido por 2. O fim do algoritmo se dá com $Z_{LB} = Z_{UB}$ ou $\varepsilon \leq 0,005$. Podemos realizar testes diferentes alterando o valor inicial de ε, o valor da condição de parada, e o parâmetro N. Observe que se $Z_{LB} = Z_{UB}$ e a solução do lagrangeano for viável no problema original, então ela também é ótima no problema original.

5.3.2 Implementação e Testes

A formulação do problema relaxado foi implementada como um programa linear e solucionado com o *solver* CPLEX. Para as instâncias de teste e parâmetros que o lagrangeano foi submetido, seus resultados não conseguiram melhorar os limitantes inferiores apresentados na Subseção 2.2.2. Portanto, novos testes foram realizados, em que novos valores foram utilizados para os parâmetros do subgradiente. Porém, mesmo com os ajustes, o lagrangeano não conseguiu melhorar os limites inferiores fornecidos na literatura. Dessa forma, os resultados não foram reportados.

5.4 Inclusão de Desigualdades Válidas

Nesta Seção, são apresentadas três desigualdades válidas para o problema, como elas são inseridas em cada modelo estudado neste trabalho, e apresentação e discussão dos resultados experimentais comparados aos dos modelos originais.

Para todo π, definimos a sua **permutação mínima**, π_m, como a permutação gerada a partir da substituição de todas as *strips* de π por um único elemento, e excluindo qualquer *strip* inicial que inicie com 1 e qualquer *strip* final que termine com n (CHRISTIE, 1998). Christie (1998) demonstra que para toda permutação π e sua permutação mínima π_m, temos que $d(\pi) = d(\pi_m)$. Dias (2012) apresenta o seguinte procedimento para a geração de uma permutação mínima de π:

1. Caso a primeira *strip* inicie com 1, remover a primeira *strip*;
2. Caso a última *strip* termine com n, remover a última *strip*;
3. Para cada *strip*, substituir a strip pelo seu elemento de menor valor;
4. Mapear a permutação gerada em uma permutação válida.

Por exemplo, considerando $\pi = (4\ 5\ 6\ 3\ 1\ 2\ 7)$, geramos $\pi_m = (3\ 2\ 1)$ com os seguintes passos:

1. A permutação π possui 3 *strips*:

$$\pi_m = (4\ 5\ 6 \bullet 3 \bullet 1\ 2 \bullet 7)$$

2. A primeira *strip* não pode ser removida, pois não começa com 1. Podemos remover apenas a última *strip* que termina com $n = 7$:

$$\pi_m = (4\ 5\ 6 \bullet 3 \bullet 1\ 2)$$

3. Para as três *strips* restantes, substituímos cada uma pelo seu menor elemento:

$$\pi_m = (4 \bullet 3 \bullet 1)$$

4. Mapeamos a permutação do passo anterior para gerar uma permutação válida:

$$\pi_m = (3\ 2\ 1)$$

Essa redução nos traz a possibilidade de inserirmos três restrições (desigualdades válidas) nos modelos estudados:

1. Fixar a *strip* inicial, caso ela exista;
2. Fixar a *strip* final, caso ela exista;
3. Definir movimentos adjacentes para todas as *strips*.

Podemos definir um movimento adjacente da seguinte forma: para qualquer *strip* $(i\ \ldots\ j)$ que possua mais de um elemento, onde $1 \leq i < j \leq n$, se o elemento z passar para a posição a após um rearranjo, então o elemento $z+1$ passará para a posição $a+1$, para todo $z \in \{i, \ldots, j-1\}$.

5.4.1 Modelo de Lancia, Rinaldi e Serafini (2015)

Dada uma permutação π, considere o vetor *ord* de tamanho n, onde $ord[\pi_i] = i$ para todo $i \in \{1, \ldots, n\}$. O vetor *ord* é responsável por guardar os índices da permutação π de forma que os valores de π naqueles índices estejam ordenados de forma crescente, ou seja, $\pi_{ord[i]} < \pi_{ord[i+1]}$ para todo $i \in \{1, \ldots, n-1\}$. Esse vetor é útil para determinar se duas *commodities* estão em uma *strip*. Sendo assim, podemos definir as desigualdades válidas como:

$$\sum_{a=1}^{n} x_{a,1}^{k,ord[1]} \leq x_{1,1}^{k+1,ord[1]}, \forall k \in \{1, \ldots, L-1\} \tag{5.21}$$

$$\sum_{a=1}^{n} x_{a,n}^{k,ord[n]} \leq x_{n,n}^{k+1,ord[n]}, \forall k \in \{1, \ldots, L-1\} \tag{5.22}$$

$$\sum_{a=1}^{n} x_{a,b}^{k,ord[i]} + \sum_{a'=1}^{n} x_{a',b+1}^{k,ord[i+1]} + x_{b,b'}^{k+1,ord[i]} \leq x_{b+1,b'+1}^{k+1,ord[i+1]} + 2, \quad \forall b, b', i \in \{1, \ldots, n-1\}, \forall k \in \{1, \ldots, L-1\} \tag{5.23}$$

A restrição (5.21) garante que caso exista a *strip* inicial em determinada camada, a *strip* inicial não mudará de posição nas próximas camadas. Já a (5.22) garante que caso exista a *strip* final em determinada camada, a *strip* final não mudará de posição nas próximas

camadas. Entretanto, as restrições (5.21) e (5.22) dependem da restrição (5.23) para funcionar adequadamente, pois elas duas fixam apenas o primeiro elemento da *strip* inicial e o último elemento da *strip* final, respectivamente, e a (5.23) irá fixar as *strips* inteiras nas camadas seguintes. A restrição (5.23) garante que toda *strip* formada em uma determinada camada sempre fará movimentos adjacentes.

5.4.2 Modelo de Dias e Souza (2007)

Para o modelo de Dias e Souza (2007), podemos definir as três desigualdades válidas como:

$$B_{1,1,k} \leq B_{1,1,k+1}, \forall k \in 1,\ldots,n-2 \tag{5.24}$$

$$B_{n,n,k} \leq B_{n,n,k+1}, \forall k \in 1,\ldots,n-2 \tag{5.25}$$

$$\begin{gathered} B_{i,j,k} + B_{i+1,j+1,k} + B_{j',j,k+1} \leq B_{j'+1,j+1,k+1} + 2, \\ \forall i,j,j' \in \{1,\ldots,n-1\}, \forall k \in \{0,\ldots,n-2\} \end{gathered} \tag{5.26}$$

De modo similar às restrições adicionadas ao modelo de Lancia, Rinaldi e Serafini (2015), as restrições (5.24) e (5.25) fixam, caso existam, a *strip* inicial e a *strip* final, respectivamente. A (5.26) define movimentos adjacentes em *strips*. As restrições (5.24) e (5.25) dependem da restrição que define movimentos adjacentes.

5.4.3 Modelo com Emparelhamentos Perfeitos

Considere *abs* como uma função que retorna o valor absoluto de uma expressão. Para facilitar a compreensão, utilizaremos a função *abs* para representação das desigualdades válidas. A conversão dessa função para uma restrição linear pode ser feita com a adição de variáveis extras. Como V_k representa o vetor contendo a permutação da camada k, utilizaremos a notação $V_k[a]$ para representar o elemento na posição a da camada k. Sendo assim, temos:

$$1 - abs(V_k[1] - 1) \leq b_{1,1,k+1}, \forall k \in \{1,\ldots,L-1\} \tag{5.27}$$

$$1 - abs(V_k[n] - n) \leq b_{n,n,k+1}, \forall k \in \{1,\ldots,L-1\} \tag{5.28}$$

$$\begin{gathered} abs(V_k[i+1] - V_k[i] - 1) \geq b_{i,j,k+1} - b_{i+1,j+1,k+1}, \\ \forall i,j \in \{1,\ldots,n-1\}, \forall k \in \{1,\ldots,L-1\} \end{gathered} \tag{5.29}$$

Como nos modelos anteriores, as restrições para fixação de *strip* inicial (5.27) e *strip* final (5.29) dependem da restrição de movimentos adjacentes (5.29).

5.4.4 Implementação e Testes com as Desigualdades Válidas

Após a implementação das desigualdades válidas nos três modelos, foi realizada uma nova etapa de testes. Os resultados são apresentados nas Tabelas 4, 5 e 6.

Tabela 4 – Resultados dos testes utilizando permutações πY para os modelos com desigualdades válidas.

Permutações $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,01	0,01	0,03
5	0,01	0,01	0,01
6	0,23	0,32	0,59
7	0,95	0,38	1,12
8	33,79	12,44	89,19
9	73,16	17,09	1033,07
10	22269,20	10291,00	50683,50
11	53414,50	timeout	timeout
12	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

Tabela 5 – Resultados dos testes utilizando permutações πX para os modelos com desigualdades válidas.

Permutações $\pi X = (n\ n-1\ \ldots\ 1)$			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,05	0,02	0,03
5	0,30	0,16	0,55
6	3,33	1,11	8,59
7	50,77	43,56	3367,00
8	1816,25	1356,56	10367,00
9	6487,02	53550,00	timeout
10	50462,50	timeout	timeout
11	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

Tabela 6 – Resultados dos testes utilizando permutações aleatórias para os modelos com desigualdades válidas.

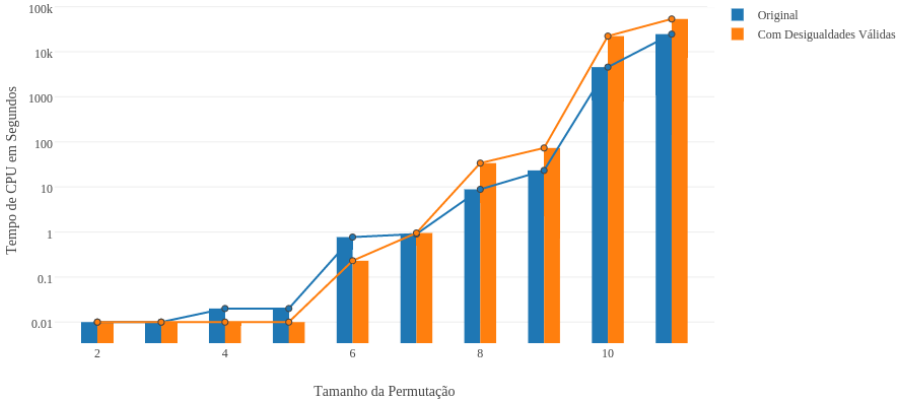
Permutações Aleatórias - Tempo Médio de Execução			
	Tempo de CPU (segundos)		
n	Lancia, Rinaldi e Serafini (2015)	Dias e Souza (2007)	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,01	0,01	0,01
5	0,14	0,03	0,26
6	0,88	0,29	3,58
7	24,74	2,48	110,40
8	29,18	7,22	4657,00
9	1549,56	1152,31	1849,00
10	21106,00	54211,71	timeout
11	50420,83	timeout	timeout
12	timeout	timeout	timeout

Fonte: Elaborada pelo autor.

Ao comparar os resultados entre as implementações originais e as implementações com desigualdades válidas, percebemos melhoras no modelo de Dias e Souza (2007), para todas as instâncias testadas. No modelo baseado em emparelhamentos perfeitos, nota-se melhoria nas instâncias πX e πY, mas um aumento no tempo de execução para instâncias aleatórias. No modelo de Lancia, Rinaldi e Serafini (2015), constatamos que na maioria dos casos as desigualdades válidas aumentam o tempo de execução. O modelo de Lancia, Rinaldi e Serafini (2015) continou sendo o que resolve instâncias de maiores tamanhos, apesar que a quantidade de instâncias resolvidas tenha permanecido a mesma.

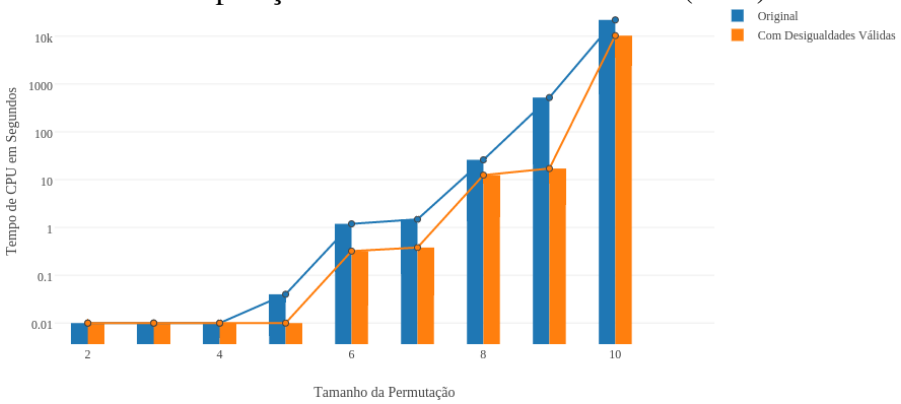
Para facilitar a visualização nas mudanças entre as versões dos modelos, foram criados os Gráficos 1, 2, 3, 4, 5, 6, 7, 8, 9. O eixo y dos gráficos (Tempo de CPU em Segundos) está em escala logarítmica para melhor visualização da variação de valores.

Gráfico 1 – Gráfico de Comparação do Modelo de Lancia, Rinaldi e Serafini (2015) com Instâncias πY.



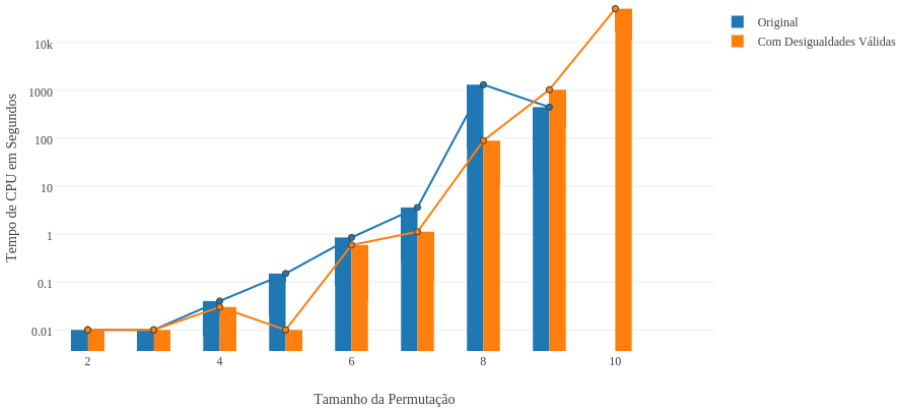
Fonte: Elaborado pelo autor.

Gráfico 2 – Gráfico de Comparação do Modelo de Dias e Souza (2007) com Instâncias πY.



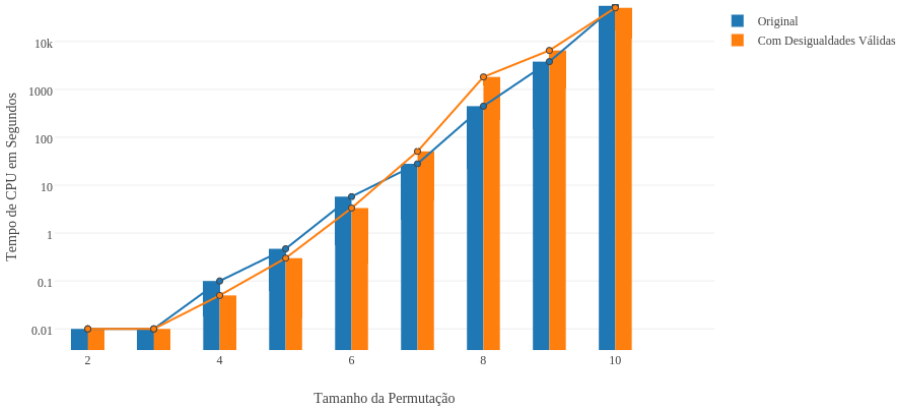
Fonte: Elaborado pelo autor.

Gráfico 3 – Gráfico de Comparação do Modelo Baseado em Emparelhamentos Perfeitos com Instâncias πY.



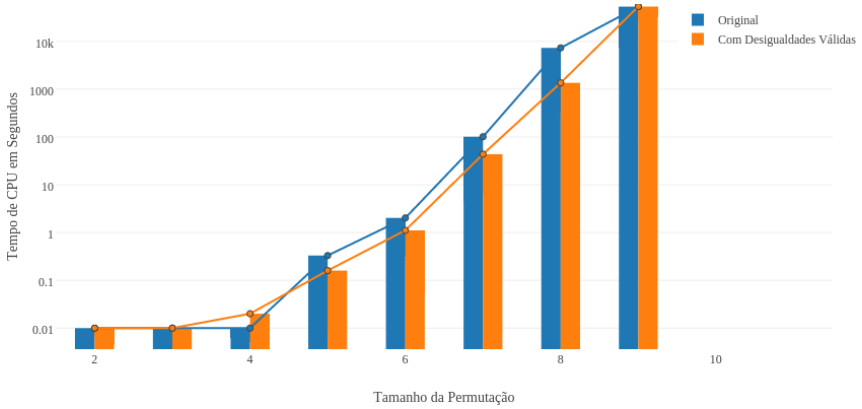
Fonte: Elaborado pelo autor.

Gráfico 4 – Gráfico de Comparação do Modelo de Lancia, Rinaldi e Serafini (2015) com Instâncias πX.



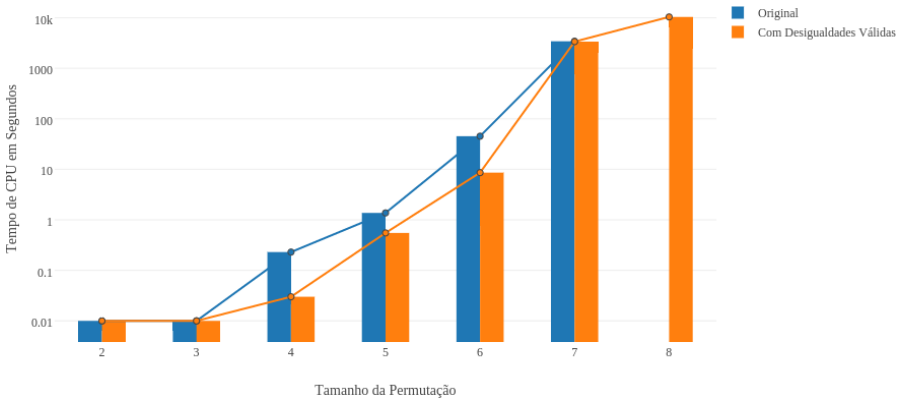
Fonte: Elaborado pelo autor.

Gráfico 5 – Gráfico de Comparação do Modelo de Dias e Souza (2007) com Instâncias πX.



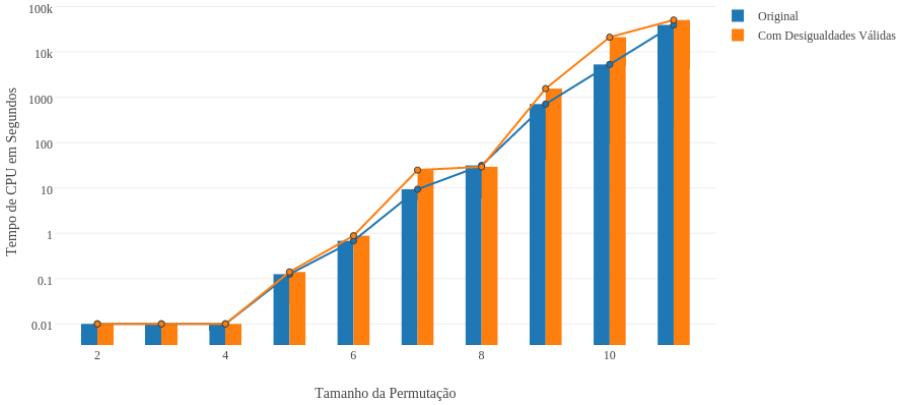
Fonte: Elaborado pelo autor.

Gráfico 6 – Gráfico de Comparação do Modelo Baseado em Emparelhamentos Perfeitos com Instâncias πX.



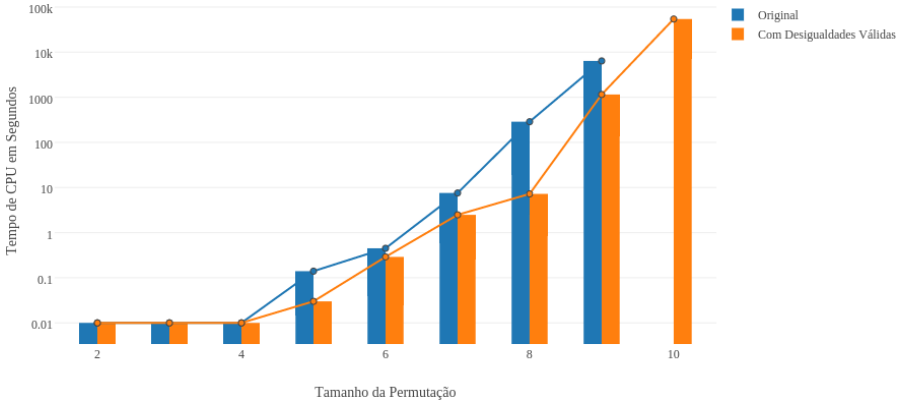
Fonte: Elaborado pelo autor.

Gráfico 7 – Gráfico de Comparação do Modelo de Lancia, Rinaldi e Serafini (2015) com Instâncias Aleatórias.



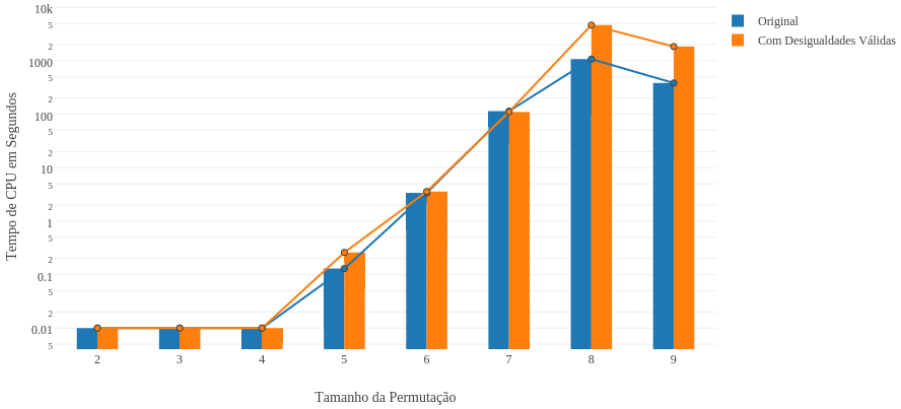
Fonte: Elaborado pelo autor.

Gráfico 8 – Gráfico de Comparação do Modelo de Dias e Souza (2007) com Instâncias Aleatórias.



Fonte: Elaborado pelo autor.

Gráfico 9 – Gráfico de Comparação do Modelo Baseado em Emparelhamentos Perfeitos com Instâncias Aleatórias.



Fonte: Elaborado pelo autor.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve o objetivo de criar um novo modelo de programação inteira para o problema de rearranjo de genomas, assim como comparar o novo modelo com os existentes na literatura e propor melhorias aos modelos através da adição de desigualdades válidas. Conseguimos alcançar parcialmente o objetivo do trabalho, sendo que um novo modelo foi criado, porém esse modelo não possui melhores resultados dos que os já existentes. Além disso, constatamos que o modelo definido por Lancia, Rinaldi e Serafini (2015) tem melhores resultados dentre os três modelos que consideramos neste trabalho.

Como subprodutos, conseguimos criar desigualdades válidas para os modelos a partir dos resultados de Christie (1998), atualizar os limitantes utilizados nos modelos com os resultados de Dias (2012), e aplicar a técnica de relaxação lagrangeana no modelo de Lancia, Rinaldi e Serafini (2015), mesmo sendo ineficaz na prática.

Como não se conseguiu melhores resultados utilizando programação linear inteira, uma possibilidade interessante é a continuação desse trabalho utilizando programação por restrições. Em Dias e Dias (2009) foram realizadas implementações de modelos de programação por restrição, e ao final foram executados testes comparando essa abordagem com a de programação linear inteira utilizando o modelo de Dias e Souza (2007). Como conclusões, Dias e Dias (2009) apresentaram que a nova abordagem conseguiu melhores resultados que o modelo de programação linear inteira. Uma possível direção de trabalho é a implementação de modelos de programação por restrição baseados no digrafo de camadas, e a comparação com os modelos de programação linear inteira. Outra direção de trabalhos futuros é a modelagem do problema de rearranjo de genomas envolvendo outros tipos de rearranjos.

REFERÊNCIAS

BAFNA, V.; PEVZNER, P. A. **Genome rearrangements and sorting by reversals**. SIAM Journal on Computing, San Franciso, USA, v. 25, n. 2, p. 272–289, 1996.

BAFNA, V.; PEVZNER, P. A. **Sorting by transpositions**. SIAM Journal on Discrete Mathematics, 11(2), 224-240, San Franciso, USA, 1998.

BEASLEY, J. E. Lagrangian relaxation. In: **Modern heuristic techniques for combinatorial problems**. New York: John Wiley & Sons, Inc., 1993. p. 243–303.

BERMAN, P.; HANNENHALLI, S.; KARPINSKI, M. **1.375-approximation algorithm for sorting by reversals**. Springer, Rome, Italy, 2002.

BRADLEY, S.; HAX, A.; MAGNANTI, T. **Applied mathematical programming**. Addison Wesley, Boston, USA, 1977.

BULTEAU, L.; FERTIN, G.; RUSU, I. **Sorting by transpositions is difficult**. SIAM Journal on Discrete Mathematics, 26(3), 1148-1180, Germany, 2012.

CAPRARA, A. **Sorting permutations by reversals and Eulerian cycle decompositions**. SIAM journal on discrete mathematics, v. 12, n. 1, p. 91–110, 1999.

CHRISTIE, D. A. **Genome rearrangement problems**. Tese (Doutorado) — University of Glasgow, Glasgow, 1998.

CORMEN, T. H. **Introduction to algorithms**. Cambridge, USA: MIT press, 2009.

CPLEX, I. I. **12.1 User's Manual for CPLEX**. In: ILOG INC. [S.l.]: International Business Machines Corporation, 46(53), 157, 2009.

DIAS, U.; DIAS, Z. Constraint programming models for transposition distance problem. In: SPRINGER. **Brazilian Symposium on Bioinformatics**. [S.l.], 2009. p. 13–23.

DIAS, U. M. **Problemas de comparação de genomas**. Tese (Doutorado) — Instituto de Computação (IC), Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil, 2012.

DIAS, Z.; SOUZA, C. C. **Polynomial-sized ILP Models for Rearrangement Distance Problems**. Poster Proceedings of the 3rd Brazilian Symposium on Bioinformatics (BSB'2007), p. 74, Rio de Janeiro, Brasil, 2007.

ELIAS, I.; HARTMAN, T. **A 1.375-approximation algorithm for sorting by transpositions**. Computational Biology and Bioinformatics, IEEE/ACM Transactions on, v. 3, n. 4, p. 369–379, 2006.

EVEN, S.; ITAI, A.; SHAMIR, A. **On the complexity of time table and multi-commodity flow problems**. Foundations of Computer Science, 1975., 16th Annual Symposium on, IEEE, Berkeley, USA, p. 184–193, 1975.

FREITAS, L. I. B. **A conjectura de Tuza sobre triângulos em grafos**. Dissertação (Mestrado) — Instituto de Computação (IC), Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil, 2014.

GOLDBARG, M. **Grafos: Conceitos, algoritmos e aplicações**. [S.l.]: Elsevier Brasil, 2012.

HANNENHALLI, S.; PEVZNER, P. A. **Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals**. Journal of the ACM (JACM), v. 46, n. 1, p. 1–27, 1999.

IIZUKA, V. de A. **Programação por Restrições aplicada a Problemas de Rearranjo de Genomas**. Dissertação (Mestrado) — Instituto de Computação (IC), Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil, 2012.

JONES, N. C.; PEVZNER, P. **An introduction to bioinformatics algorithms**. Cambridge, USA: MIT press, 2004.

LANCIA, G.; RINALDI, F.; SERAFINI, P. **A Unified Integer Programming Model for Genome Rearrangement Problems**. Bioinformatics and Biomedical Engineering, Springer, Granada, Spain, p. 491–502, 2015.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. [S.l.]: Courier Corporation, 1982.

SETUBAL, J. C.; MEIDANIS, J. **Introduction to computational molecular biology**. [S.l.]: PWS Pub., 1997.

THIE, P. R.; KEOUGH, G. E. **An introduction to linear programming and game theory**. New York: John Wiley & Sons, 2011.

WOLSEY, L. A. **Integer programming**. New York: John Wiley & Sons, Inc., 1998. v. 42.