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

Alexsandro Oliveira Alexandrino
Instituto de Computação da Universidade Estadual de Campinas
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas - SP, 13083-852
sandrooliveira1501@gmail.com

Críston Pereira de Souza
Universidade Federal do Ceará
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará 63902-580
criston@ufc.br

Lucas Ismaily Bezerra Freitas
Universidade Federal do Ceará
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará 63902-580
ismailybf@ufc.br

RESUMO

O cálculo da distância evolucionária entre espécies é um problema importante da área de biologia computacional. A abordagem chamada rearranjo de genomas consiste em determinar o número mínimo de reposicionamentos de fragmentos do genoma de uma espécie de modo a igualá-lo ao genoma de outra espécie. Neste trabalho, propomos um novo modelo de programação linear inteira para este problema, e realizamos uma comparação experimental com outros dois modelos disponíveis na literatura. Propomos e avaliamos também desigualdades válidas para os modelos, e aplicamos relaxação lagrangeana em um dos modelos na busca de melhores limites inferiores. Concluímos que o modelo definido por [Lancia et al., 2015] possui os melhores resultados, e que as desigualdades válidas propostas melhoram os outros dois modelos.

PALAVRAS CHAVE. Rearranjo de Genomas. Programação Linear Inteira. Desigualdades Válidas. Relaxação Lagrangeana.

ABSTRACT

Finding the evolutionary distance between species is an important problem in computational biology. The approach known as genome rearrangements consists of determining the minimun number of fragments repositions of a species's genome so that it matches the genome of another species. In this work, we propose a new integer programming model for this problem, and we make an experimental comparison with two other models available in the literature. We also propose and evaluate valid inequalities for the models, and apply lagrangean relaxation in one of the models in the search for better lower bounds. We conclude that the model defined by [Lancia et al., 2015] has the best results, and that the valid inequalities proposed improve the other two models.

KEYWORDS. Genome Rearrangements. Integer Linear Programming. Valid Inequalities. Lagrangian Relaxation.

1. Introdução

A evolução biológica é o conjunto de mudanças nas características hereditárias de uma população no decorrer do tempo. Computar a distância evolutiva entre duas espécies, em que são considerados apenas rearranjos, é um problema importante da biologia computacional. Entende-se por rearranjo de genoma qualquer evento que altere grandes trechos do genoma, sendo que esses eventos podem ser de tipos distintos, desde a alteração da ordem de uma parte do genoma até a troca de partes inteiras de uma posição para outra [Setubal e Meidanis, 1997].

Considerando que não existem repetições de genes, o genoma de uma espécie é representado por uma permutação de inteiros. Partindo do *Princípio da Máxima Parcimônia*, em que a natureza utiliza a melhor solução ou a solução mais simples, a distância evolucionária entre duas espécies A e B é definida como o menor número de rearranjos necessários para transformar o genoma de A no genoma de B [Dias, 2012].

Na literatura, os dois tipos de rearranjos 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. Este estudo está interessado apenas no estudo de transposições.

O problema envolvendo apenas transposições é NP-Difícil [Bulteau et al., 2012]. Esse problema possui algoritmo aproximativo com razão de 1,375 [Elias e Hartman, 2006]. Em [Dias, 2012] é apresentado um estudo detalhado com limites superiores, inferiores e heurísticas para uma nova versão do algoritmo de [Elias e Hartman, 2006], com melhores resultados práticos.

Em [Lancia et al., 2015] e [Dias e Souza, 2007] são definidos modelos de programação linear inteira para o problema em estudo. O modelo definido por [Lancia et al., 2015] permite 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 em como cada rearranjo altera um genoma.

Neste trabalho consideramos o problema de rearranjo de genomas por transposição. Propomos um novo modelo de programação linear inteira e uma heurística lagrangeana, e os comparamos com os modelos disponíveis na literatura. Também são propostas melhorias nos modelos atuais através da adição de desigualdades válidas, na tentativa de melhorar seus limites inferiores. Embora o foco seja em rearranjos por transposição, algumas ideias desenvolvidas neste trabalho podem ser adaptadas para outros tipos de rearranjo.

Este artigo esta organizado da seguinte forma: a Seção 2 descreve os conceitos sobre Rearranjo de Genomas; a Seção 3 apresenta a formulação de um novo modelo de programação inteira e resultados da comparação experimental entre os modelos; a Seção 4 descreve a aplicação da técnica de Relaxação Lagrangeana no modelo de [Lancia et al., 2015]; a Seção 5 apresenta desigualdades válidas para os modelos e os resultados experimentais das modificações; e a Seção 6 apresenta as considerações finais.

2. Rearranjo de Genomas

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. Isso se faz necessário devido à grande quantidade de genes no genoma. Depois de identificado os blocos, eles são numerados de 1 a n (número de blocos).

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 $i = j \Leftrightarrow \pi_i = \pi_j$. 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, $\rho_1\rho_2 \dots \rho_k\pi_1 = \pi_2$.

2.1. Transposição

Definimos uma *transposição* $\rho(a, b, c)$ em π (Figura 1), onde $1 \leq a < b < c \leq n + 1$, como $\rho\pi$ que resulta na permutação $(\pi_1 \ \dots \ \pi_{a-1} \ \pi_b \ \dots \ \pi_{c-1} \ \pi_a \ \dots \ \pi_{b-1} \ \pi_c \ \dots \ \pi_n)$.

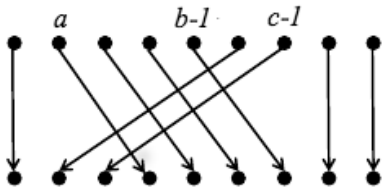


Figura 1: Transposição $\rho(a, b, c)$ agindo em uma permutação.

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].

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. Breakpoints

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*:

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

Chamamos de *strip* qualquer intervalo de elementos consecutivos sem *breakpoints*. Em (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].

3. Modelo Baseado em Emparelhamentos Perfeitos

Os conceitos de digrafo de camadas e emparelhamentos perfeitos são utilizados na proposta do novo modelo. Esses conceitos são apresentados a seguir.

Um *digrafo de camadas* $D = (V, A)$ com k camadas é um digrafo que pode ser particionado em k conjuntos de vértices, sendo que só existem arcos entre camadas consecutivas. Denotamos cada partição ou camada 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 $(b, a) \notin A$, e para todo arco $(a, b) \in A$ temos que $a \in L_i$ e $b \in L_{i+1}$, tal que $1 \leq i < k$ (Figura 2).

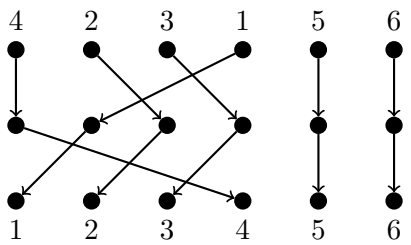


Figura 2: Exemplo de digrafo de camadas com $k = 3$.

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 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 $v \in V$.

Para o novo modelo, 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, denotado por M^*, é da forma $\{(1,1),\ (2,2),\ \ldots,\ (n,n)\}$; L é um limitante superior para o problema; a variável binária B_{ijk} denota se o arco (i,j) está presente no emparelhamento do nível k; a variável binária t_k denota se o emparelhamento de uma camada é distinto do emparelhamento identidade.

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:

$$\min \sum_{k=1}^{L} t_k \tag{2}$$

sujeito a:

$$\sum_{i \in V_{k-1}} B_{ijk} = 1, \forall j \in V_k, \forall k \in \{1, \ldots, L\} \tag{3}$$

$$\sum_{j \in V_{k-1}} B_{ijk} = 1, \forall i \in V_k, \forall k \in \{1, \ldots, L\} \tag{4}$$

$$B_{i\sigma_i L} = 1, \forall i \in V_L, \forall \sigma_i \in V_{L+1} \tag{5}$$

$$M_k \in T_k \cup M^*, \forall k \in \{1, \ldots, L\} \tag{6}$$

$$t_k = 1,\ M_k \in T_k, \forall k \in \{1, \ldots, L\} \tag{7}$$

$$t_k = 0,\ M_k = M^*, \forall k \in \{1, \ldots, L\} \tag{8}$$

$$t_k \leq t_{k+1}, \forall k \in \{1, \ldots, L-1\} \tag{9}$$

$$t_k \in \{0,1\}, B_{ijk} \in \{0,1\}, \forall k \in \{1, \ldots, L\} \tag{10}$$

Para facilidade de entendimento do modelo, as restrições (6), (7) e (8) apresentadas são não lineares. A transformação dessas restrições não lineares para restrições lineares foi feita com a utilização da biblioteca do *solver* CPLEX da IBM. Utilizamos (3) e (4) para garantir que os arcos da solução formem emparelhamentos perfeitos em cada camada de arcos. A restrição (5) força que a última camada de vértices possua a permutação σ. A (6) garante que sempre utilizamos operações válidas para cada camada, e (7) e (8) definem o valor da variável t_k. A restrição (9) agrupa os emparelhamentos identidade nas últimas camadas, reduzindo assim o número de soluções redundantes no espaço de soluções, e a restrição (10) define o tipo das variáveis.

Com a função objetivo (2), queremos minimizar o número de emparelhamentos utilizados que são distintos do emparelhamento identidade. Note que se $M_k \neq M^*$, então $t_k = 1$.

3.1. Comparação Experimental dos Modelos

Utilizando-se da linguagem C++ e da biblioteca do *solver* CPLEX 12.6.1 da IBM, foram implementados os modelos definidos por [Lancia et al., 2015] e [Dias e Souza, 2007], assim como o novo modelo apresentado na Seção 3. Para estimar o número de camadas do grafo (L), foram utilizadas as heurísticas apresentadas por [Dias, 2012].

Os testes foram realizados em um computador 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 obtidas com distribuição uniforme no intervalo $[1, n]$, repetindo o sorteio no caso de elementos repetidos. As instâncias πX e πY foram escolhidas por serem famílias de permutações consideradas na literatura [Dias e Souza, 2007]. Foi estipulado o tempo limite de duas horas para execução dos testes. Esse tempo limite reflete o tempo real decorrido desde o início até o fim da execução, já o tempo reportado é o valor da soma de tempo de uso de cada núcleo do processador calculado pelo CPLEX. Cinco instâncias aleatórias foram geradas para cada tamanho de entrada, e as médias de tempo de execução são apresentadas para cada tamanho. Os resultados são apresentados nas Tabelas 1, 2 e 3.

Tabela 1: Resultados da comparação experimental utilizando permutações πY.

Permutações $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$			
	Tempo de CPU (segundos)		
n	[Lancia et al., 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	476,29
9	23,26	521,60	1.215,85
10	4.558,68	21.939,10	timeout
11	24.627,00	timeout	timeout
12	timeout	timeout	timeout

Tabela 2: Resultados da comparação experimental utilizando permutações πX.

Permutações $\pi X = (n\ n-1\ \ldots\ 1)$			
	Tempo de CPU (segundos)		
n	[Lancia et al., 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	3.420,28
8	446,02	7.301,63	timeout
9	3.795,80	53.355,00	timeout
10	55.462,50	timeout	timeout
11	timeout	timeout	timeout

Tabela 3: Resultados da comparação experimental utilizando permutações aleatórias.

Permutações Aleatórias - Tempo Médio de Execução			
	Tempo de CPU (segundos)		
n	[Lancia et al., 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	0,13	0,14	0,13
6	0,68	0,45	3,38
7	9,32	7,57	114,47
8	31,31	287,29	394,09
9	706,97	6.391,27	1.085,95
10	5.286,78	timeout	timeout
11	39.348,43	timeout	timeout
12	timeout	timeout	timeout

A partir dos resultados apresentados e para as instâncias utilizadas nos testes, conclui-se que o modelo definido por [Lancia et al., 2015] tem melhor desempenho do que os outros dois modelos em todos os tipos de instâncias, pois ele possui melhor tempo de execução e resolve instâncias que os demais não resolvem dentro do limite de tempo estabelecido. Além disso, percebe-se que o novo modelo resolve menos instâncias que os demais modelos.

4. Aplicação de Relaxação Lagrangeana no Modelo de [Lancia et al., 2015]

Em otimização, resolver instâncias maiores de um problema difícil em tempo viável geralmente envolve a utilização de técnicas e criação de novos limitantes para o problema, sendo uma dessas técnicas a relaxação lagrangeana [Beasley, 1993]. Para um problema de minimização, 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.

A partir do modelo baseado em *multicommodity flow* de [Lancia et al., 2015], construímos uma nova formulação utilizando a relaxação lagrangeana. O modelo de [Lancia et al., 2015] é definido a partir de um digrafo de camadas com L camadas (limite superior). Os arcos utilizados de uma camada para outra estão mapeados no conjunto O, que representa o conjunto de rearranjos permitidos. O rearranjo nulo é representado por μ_0, sendo útil quando a permutação não é alterada de uma camada para outra. O modelo possui variáveis z^k_μ e x^{ki}_{ab}. A variável binária z^k_μ, para $k \in \{1,\ldots,L\}$ e $\mu \in O$, tem $z^k_\mu = 1$ se o rearranjo μ foi utilizado para ir da camada k para a $k+1$. Já a variável x^{ki}_{ab}, 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:

$$\min \sum_{k=1}^{L} \sum_{\mu \in O-\mu_0} z^k_\mu \tag{11}$$

Foram relaxadas as restrições (12) e (14), e excluída a restrição (13), que não impacta a qualidade da solução.

$$\sum_{i=1}^{n} x^{ki}_{ab} \leq \sum_{\mu \in O | (a,b) \in \mu} z^k_\mu, \forall a,b \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,L\} \tag{12}$$

$$z^k_{\mu_0} \leq z^{k+1}_{\mu_0}, \forall k \in \{1,\ldots,L-1\} \tag{13}$$

$$\sum_{\mu \in O} z^k_\mu = 1, \forall k \in \{1,\ldots,L\} \tag{14}$$

O modelo relaxado, com multiplicadores α e β, é definido como:

$$\min \sum_{k=1}^{L} \sum_{\mu \in O} (1 - \alpha_k - \sum_{a,b \in \mu} \beta^k_{ab}) z^k_\mu - \sum_{k=1}^{L} z^k_{\mu_0} + \sum_{k=1}^{L} \alpha_k + \sum_{k=1}^{L} \sum_{i=1}^{n} \sum_{a=1}^{n} \sum_{b=1}^{n} \beta^k_{ab} x^{ki}_{ab} \tag{15}$$

$$\alpha_k \text{ é livre}, \beta^k_{ab} \geq 0. \tag{16}$$

Seja $\overline{c}^k_\mu = 1 - \alpha_k - \sum_{a,b \in \mu} \beta^k_{ab}$. Definimos o valor de z da seguinte forma:

$$z^k_\mu = \begin{cases} 1, & \text{se } \overline{c}^k_\mu < 0 \\ 0, & \text{caso contrário.} \end{cases} \quad \text{Para } \mu \neq \mu_0$$

$$z^k_{\mu_0} = \begin{cases} 1, & \text{se } \bar{c}^k_\mu - 1 < 0 \\ 0, & \text{caso contrário.} \end{cases} \quad \text{Para } \mu = \mu_0$$

O modelo relaxado, adicionado da restrição redundante "capacidade unitária nos arcos" (17), é um *multicommodity flow* que determina os valores das variáveis x.

$$\sum_{i=1}^{k} x^{ki}_{ab} \leq 1, \forall a, b \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, L\}. \tag{17}$$

4.1. Escolha dos Multiplicadores de Lagrange com Subgradiente

Utilizamos o procedimento subgradiente para a escolha dos multiplicadores de Lagrange proposto por [Beasley, 1993], que segue os seguintes passos:

1. Definir ϵ como um parâmetro de valor $0 < \epsilon \leq 2$;
2. Inicializar Z_{UB} (limite superior) com alguma heurística do problema e λ com um conjunto de valores arbitrários;
3. Resolver o modelo relaxado com os valores de λ atuais e calcular o valor de Z_{LB}, onde Z_{LB} é o valor ótimo do modelo relaxado;
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 $T = \dfrac{\epsilon(Z_{UB} - Z_{LB})}{\sum_{i=1}^{m} G_i^2}$;
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.

Utilizando desse procedimento, definimos os subgradientes G_k para α e G^k_{ab} para β como:

$$G_k = 1 - \sum_{\mu \in O} z^k_\mu, \forall k \in \{1, \ldots, L\}; \tag{18}$$

$$G^k_{ab} = \sum_{i=1}^{n} x^{ki}_{ab} - \sum_{\mu \in O | ab \in \mu} z^k_\mu, \forall a, b \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, L\}. \tag{19}$$

O valor do passo T é calculado para cada subgradiente:

$$T_1 = \frac{\epsilon(Z_{UB} - Z_{LB})}{\sum_{i=1}^{n} G_i^2}, \text{para o subgradiente } G_k; \tag{20}$$

$$T_2 = \frac{\epsilon(Z_{UB} - Z_{LB})}{\sum_{k=1}^{L} \sum_{a=1}^{n} \sum_{b=1}^{n} (G^k_{ab})^2}, \text{para o subgradiente } G^k_{ab}. \tag{21}$$

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

$$\alpha_k \leftarrow \alpha_k + T_1 G_k, \forall k \in \{1, \ldots, L\}; \tag{22}$$

$$\beta^k_{ab} \leftarrow \max\{0, \beta^k_{ab} + T_2 G^k_{ab}\}, \forall a, b \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, L\}. \tag{23}$$

Z_{LB} corresponde ao valor ótimo do lagrangeano (limite inferior) 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, transformamos a solução do lagrangeano em uma solução viável. 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 $\epsilon = 2$, 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 $\epsilon \leq 0,005$. 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.

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 literatura.

5. Adição de Desigualdades Válidas

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. Em [Christie, 1998] é demonstrado que para toda permutação π e sua permutação mínima π_m, temos que $d(\pi) = d(\pi_m)$. Em [Dias, 2012] é apresentado 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.

Essa redução nos traz a possibilidade de inserirmos três desigualdades válidas nos modelos estudados: (i) fixar a *strip* inicial, caso ela exista; (ii) fixar a *strip* final, caso ela exista; (iii) 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 $z \in \{i, \ldots, j - 1\}$.

5.1. Modelo de [Lancia et al., 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, a partir da variável x_{ab}^{ki}, como:

$$\sum_{a=1}^{n} x_{a,1}^{k,ord[1]} \leq x_{1,1}^{k+1,ord[1]}, \forall k \in \{1, \ldots, L-1\} \tag{24}$$

$$\sum_{a=1}^{n} x_{a,n}^{k,ord[n]} \leq x_{n,n}^{k+1,ord[n]}, \forall k \in \{1, \ldots, L-1\} \tag{25}$$

$$\sum_{a=1}^{n} x_{a,b}^{k,ord[i]} + \sum_{a'=1}^{n} x_{a',b+1}^{k,ord[i+1]} + x_{b,b'}^{k+1,ord[i]} \leq x_{b+1,b'+1}^{k+1,ord[i+1]} + 2, \quad \forall b, b', i \in \{1, \ldots, n-1\}, \forall k \in \{1, \ldots, L-1\} \tag{26}$$

A restrição (24) garante que caso exista a *strip* inicial em determinada camada, a *strip* inicial não mudará de posição nas próximas camadas. Já a (25) garante que caso exista a *strip* final em determinada camada, a *strip* final não mudará de posição nas próximas camadas. Entretanto, as restrições (24) e (25) dependem da restrição (26) para funcionar adequadamente, pois elas duas

fixam apenas o primeiro elemento da *strip* inicial e o último elemento da *strip* final, respectivamente, e a (26) irá fixar as *strips* inteiras nas camadas seguintes. A restrição (26) garante que toda *strip* formada em uma determinada camada sempre fará movimentos adjacentes.

5.2. Modelo de [Dias e Souza, 2007]

No modelo de [Dias e Souza, 2007], podemos definir as três desigualdades válidas a partir da variável $B_{i,j,k}$. 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.

$$B_{1,1,k} \leq B_{1,1,k+1}, \forall k \in 1, \ldots, n-2 \tag{27}$$

$$B_{n,n,k} \leq B_{n,n,k+1}, \forall k \in 1, \ldots, n-2 \tag{28}$$

$$B_{i,j,k} + B_{i+1,j+1,k} + B_{j',j,k+1} \leq B_{j'+1,j+1,k+1} + 2, \quad \forall i, j, j' \in \{1, \ldots, n-1\}, \forall k \in \{0, \ldots, n-2\} \tag{29}$$

De modo similar às restrições adicionadas ao modelo de [Lancia et al., 2015], as restrições (27) e (28) fixam, caso existam, a *strip* inicial e a *strip* final, respectivamente. A (29) define movimentos adjacentes em *strips*. As restrições (27) e (28) dependem da restrição que define movimentos adjacentes.

5.3. 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:

$$1 - abs(V_k[1] - 1) \leq b_{1,1,k+1}, \forall k \in \{1, \ldots, L-1\} \tag{30}$$

$$1 - abs(V_k[n] - n) \leq b_{n,n,k+1}, \forall k \in \{1, \ldots, L-1\} \tag{31}$$

$$abs(V_k[i+1] - V_k[i] - 1) \geq b_{i,j,k+1} - b_{i+1,j+1,k+1}, \quad \forall i, j \in \{1, \ldots, n-1\}, \forall k \in \{1, \ldots, L-1\} \tag{32}$$

Como nos modelos anteriores, as restrições para fixação de *strip* inicial (30) e *strip* final (32) dependem da restrição de movimentos adjacentes (32).

5.4. Comparação Experimental dos Modelos com Desigualdades Válidas

Após a implementação das desigualdades válidas nos três modelos, foi realizada uma nova etapa de comparação experimental. Os resultados são apresentados nas Tabelas 4, 5 e 6. Para facilitar a visualização nas mudanças entre as versões dos modelos, foram criados os Gráficos 3a, 3b, 3c, 3e, 3d, 3f, 3h, 3g, 3i. O eixo y dos gráficos (Tempo de CPU em Segundos) está em escala logarítmica para melhor visualização da variação de valores.

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 et al., 2015], constatamos que na maioria dos casos as desigualdades válidas aumentam o tempo de execução. O modelo de [Lancia et al., 2015] continuou sendo o que resolve instâncias de maiores tamanhos, apesar da quantidade de instâncias resolvidas ter permanecido a mesma.

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

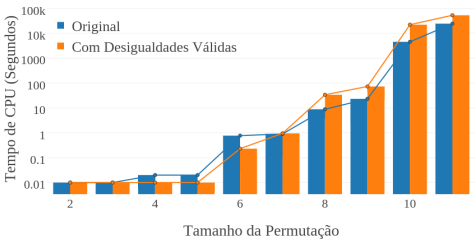
Permutações $\pi Y = (n\ n-2\ \ldots\ 2\ 1\ \ldots\ n-3\ n-1)$			
	Tempo de CPU (segundos)		
n	[Lancia et al., 2015]	[Dias e Souza, 2007]	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,01	0,01	0,03
5	0,01	0,01	0,01
6	0,23	0,32	0,59
7	0,95	0,38	1,12
8	33,79	12,44	89,19
9	73,16	17,09	1.033,07
10	22.269,20	10.291,00	50.683,50
11	53.414,50	timeout	timeout
12	timeout	timeout	timeout

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

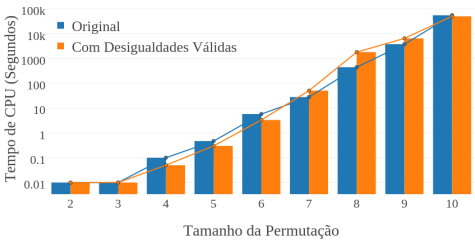
Permutações $\pi X = (n\ n-1\ \ldots\ 1)$			
	Tempo de CPU (segundos)		
n	[Lancia et al., 2015]	[Dias e Souza, 2007]	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,05	0,02	0,03
5	0,30	0,16	0,55
6	3,33	1,11	8,59
7	50,77	43,56	3.367,00
8	1.816,25	1.356,56	10.367,00
9	6.487,02	53.550,00	timeout
10	50.462,50	timeout	timeout
11	timeout	timeout	timeout

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

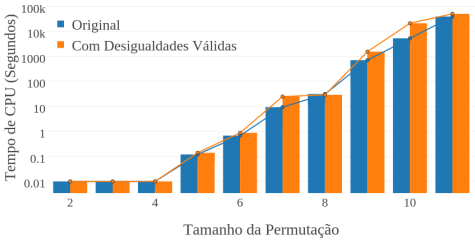
Permutações Aleatórias - Tempo Médio de Execução			
	Tempo de CPU (segundos)		
n	[Lancia et al., 2015]	[Dias e Souza, 2007]	Emparelhamentos Perfeitos
2	0,01	0,01	0,01
3	0,01	0,01	0,01
4	0,01	0,01	0,01
5	0,14	0,03	0,26
6	0,88	0,29	3,58
7	24,74	2,48	110,40
8	29,18	7,22	1.853,00
9	1.549,56	1.152,31	4.607,00
10	21.106,00	54.211,71	timeout
11	50.420,83	timeout	timeout
12	timeout	timeout	timeout



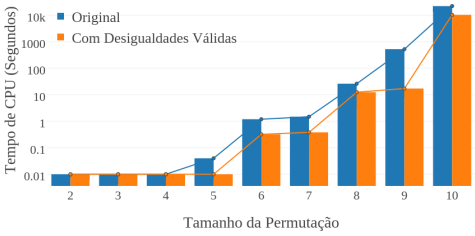
(a) [Lancia et al., 2015] com Instâncias πY.



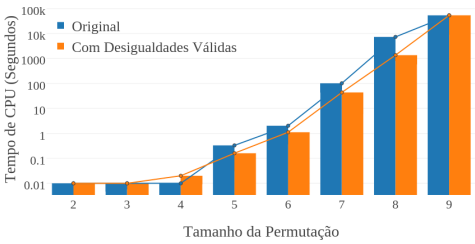
(b) [Lancia et al., 2015] com Instâncias πX.



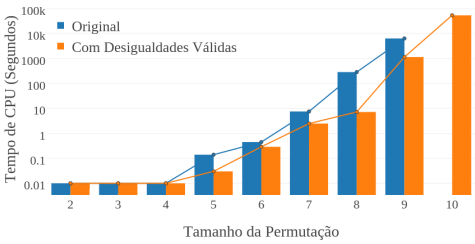
(c) [Lancia et al., 2015] com Instâncias Aleatórias.



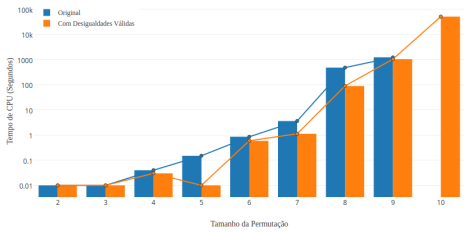
(d) [Dias e Souza, 2007] com Instâncias πY.



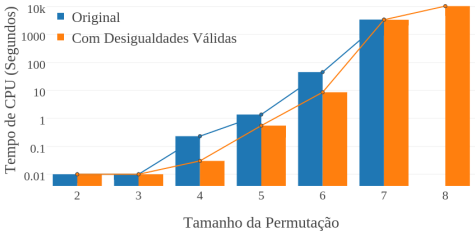
(e) [Dias e Souza, 2007] com Instâncias πX.



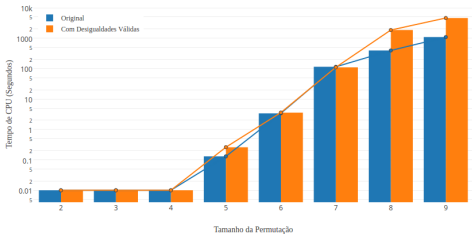
(f) [Dias e Souza, 2007] com Instâncias Aleatórias.



(g) Modelo Proposto com Instâncias πY.



(h) Modelo Proposto com Instâncias πX.



(i) Modelo Proposto com Instâncias Aleatórias.

Figura 3: Comparação Experimental entre os modelos com e sem as desigualdades válidas.

6. Conclusão e Considerações

Neste trabalho propomos um novo modelo de programação linear inteira para o problema de rearranjo de genomas utilizando operações de transposição. Este novo modelo é baseado em emparelhamentos perfeitos, e mais simples que os propostos na literatura. Porém, embora seu tempo de execução seja competitivo em algumas instâncias, de forma geral teve desempenho pior que os outros modelos. O modelo definido por [Lancia et al., 2015] apresentou os melhores resultados, porém nenhum dos modelos avaliados conseguiu encontrar solução viável dentro do tempo limite de duas horas para entradas de tamanho maior que 11, ou seja, nenhum deles é capaz de resolver instâncias reais do problema de rearranjo de genomas.

Propomos também novas desigualdades válidas baseadas nos resultados de [Christie, 1998]. Com estas desigualdades válidas foi possível reduzir o tempo de execução no modelo de [Dias e Souza, 2007] e no modelo proposto neste trabalho. Avaliamos também a aplicação da técnica de relaxação lagrangeana no modelo de [Lancia et al., 2015], mas não foi capaz de melhorar os limites propostos na literatura.

Referências

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

Bulteau, L., Fertin, G., e Rusu, I. (2012). Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics, 26(3), 1148-1180*.

Christie, D. A. (1998). *Genome rearrangement problems*. PhD thesis, University of Glasgow, Glasgow.

Dias, U. M. (2012). *Problemas de comparação de genomas*. PhD thesis, Instituto de Computação (IC), Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil.

Dias, Z. e Souza, C. C. (2007). Polynomial-sized ilp models for rearrangement distance problems. *Poster Proceedings of the 3rd Brazilian Symposium on Bioinformatics (BSB'2007), p. 74*.

Elias, I. e Hartman, T. (2006). A 1.375-approximation algorithm for sorting by transpositions. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 3(4):369–379.

Lancia, G., Rinaldi, F., e Serafini, P. (2015). A unified integer programming model for genome rearrangement problems. *Bioinformatics and Biomedical Engineering, Springer*, p. 491–502.

Setubal, J. C. e Meidanis, J. (1997). *Introduction to computational molecular biology*. PWS Pub.