O Problema da Ordenação por Reversões Ponderadas

Thiago da Silva Arruda
Zanoni Dias

Instituto de Computação - Universidade Estadual de Campinas

28 de Abril de 2016

Agenda

- Introdução
- Algoritmo guloso paramétrico: Permutações sem sinal
- Meta-Heurística GRASP: Permutações com e sem sinal

Parte I

Introdução

Rearranjo de Genomas: biologia

- Evolução:
  - Mudança das características hereditárias.
  - Resultado de mutações ocorridas no material genético.
- Mutações pontuais:
  - Afetam bases individuais do DNA.
  - Ocorrem com maior frequência.
- Rearranjo de genomas:
  - Mutações que afetam grandes trechos de DNA

Representação de Genomas: computação

- Podemos construir um modelo computacional para rearranjo de genomas.
- Um genoma com n genes é representado como uma **n-tupla** de números inteiros.
- Caso não haja repetição, esta *n-tupla* pode ser representada como uma **permutação** $\pi = (\pi_1\, \pi_2\, \pi_3 \ldots \pi_n)$, $1 \leq |\pi_i| \leq n$, $|\pi_i| \neq |\pi_j| \leftrightarrow i \neq j$.

Permutações com e sem sinais

- Para alguns modelos de rearranjo, a **orientação** dos genes é considerada, o que é representado pela atribuição de **sinais** aos elementos da permutação.
- Exemplos:
  - Sem sinal: $(1 \quad 3 \quad 7 \quad 6 \quad 2 \quad 5 \quad 4)$
  - Com sinal: $(-3 \ +2 \ -5 \ -4 \ +1 \ +7 \ -6)$

Distância de Rearranjo

- Um modelo de rearranjo de genomas é composto por um conjunto de **operações**.
- A distância de rearranjo entre duas permutações π e σ é o número **mínimo** t de operações ρ_i necessárias para transformar π em σ:

$$\sigma = (((\pi \cdot \rho_1) \cdot \rho_2) \dots) \cdot \rho_t$$

$$d(\pi, \sigma) = t$$

Ordenação de Genomas

- A permutação identidade ι é definida como $(1\,2\,3\,\ldots\,n)$.
- A **ordenação** de uma permutação π consiste no processo de transformar π na permutação identidade.
- A **distância** de ordenação de π é denotada por $d(\pi, \iota) = d(\pi)$.

Reversão Genomas

- Reversão é uma operação que consiste em inverter a **ordem** e os **sinais** (para permutações com sinais) de elementos de um segmento da permutação.
- Exemplos:
  - Sem sinais:

$$\pi \cdot \rho = (\pi_1 \ldots \pi_{i-1} \underline{\pi_j \ \pi_{j-1} \ldots \pi_{i+1} \ \pi_i} \ \pi_{j+1} \ldots \pi_n)$$

  - Com sinais:

$$\pi \cdot \rho = (\pi_1 \ldots \pi_{i-1} \underline{-\pi_j \ -\pi_{j-1} \ldots \ -\pi_{i+1} \ -\pi_i} \ \pi_{j+1} \ldots \pi_n)$$

Reversões: Estado da arte

- **Com sinais:** Algoritmo exato de complexidade sub-quadrática [10].
- **Sem sinais:** NP-Difícil [6], melhor algoritmo conhecido possui fator de aproximação 1,375 [5].

Reversões ponderadas

Definição

No modelo de ordenação por reversões ponderadas, o custo de uma reversão ρ é uma função polinomial: $f(\rho) = |\rho|^{\alpha}$, onde $|\rho| = j - i + 1$, para uma reversão $\rho = (i, j)$.

Nota

Em nosso trabalho abordamos o caso em que $\alpha = 1$.

Reversões ponderadas

Exemplo

$\pi = (\ 1\ \ 7\ \ 6\ \ 5\ \ 2\ \ 3\ \ 4\ \ 8\ \ 9\ )$

$(\ 1\ \ 7\ \ 6\ \ 5\ \boxed{2\ \ 3\ \ 4}\ \ 8\ \ 9\ ) \rightarrow \rho(5,7),\ f(\rho) = 7 - 5 + 1 = 3$

$(\ 1\ \ 7\ \ 6\ \ 5\ \ 4\ \ 3\ \ 2\ \ 8\ \ 9\ )$

$(\ 1\ \boxed{7\ \ 6\ \ 5\ \ 4\ \ 3\ \ 2}\ \ 8\ \ 9\ ) \rightarrow \rho(2,7),\ f(\rho) = 7 - 2 + 1 = 6$

$(\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ ) = \iota$

Breakpoints e strips

Definição (Breakpoints para permutações sem sinal)

$$b(\pi) = \sum_{i=0}^{|\pi|} \begin{cases} 1, & \text{se } |\pi_{i+1} - \pi_i| \neq 1. \\ 0, & \text{caso contrário.} \end{cases}$$

Definição (Breakpoints para permutações com sinal)

$$b(\pi) = \sum_{i=0}^{|\pi|} \begin{cases} 1, & \text{se } \pi_{i+1} - \pi_i \neq 1. \\ 0, & \text{caso contrário.} \end{cases}$$

Definição (Strip)

Seja π uma permutação sem sinal, uma strip de π é um intervalo maximal $[\pi_i, \ldots \pi_j]$ sem nenhum breakpoint interno tal que (π_{i-1}, π_i) e (π_j, π_{j+1}) são breakpoints.

Estado da Arte

- O estado da arte compreende:
  - Complexidade do problema
  - Limitantes
  - Algoritmos

Complexidade do problema

- Não há provas de complexidade conhecida para o problema de ordenação por reversões ponderadas.

Limitantes

- Limitantes segundo Pinter e Skiena [9] e Bender *et al.* [4]:
  - Limite inferior: $O(n\lg(n))$
  - Limite superior: $O(n\lg(n)^2)$

Algoritmos

- Pinter e Skiena [9] e Bender *et al.* [4] mostram algoritmos aproximados para o problema.
- Para ambas as versões do problema, o melhor fator de aproximação conhecido é $O(n \lg n)$.

Parte II

Heurística paramétrica para permutações sem sinal

Heurística paramétrica

- Desenvolvemos um algoritimo heurístico guloso para a versão do problema para **permutações sem sinal**.
- A função objetivo é composta de 3 métricas:
  - Entropia
  - *Breakpoint*
  - Número de inversões

Entropia

Definição

Entropia para permutações **sem sinal** *é definida pela formulação abaixo, onde* π^{-1} *é a permutação inversa:*

$$ent(\pi) = \sum_{i=1}^{|\pi|} |i - |\pi_i^{-1}||$$

Número de inversões

Definição

O número de inversões é definido pela formulação abaixo:

$$inv(\pi) = \sum_{i=1}^{|\pi|} \sum_{j=i+1}^{|\pi|} \begin{cases} 1, & \text{se } |\pi_i| > |\pi_j|. \\ 0, & \text{caso contrário.} \end{cases}$$

Função Heurística

Definição

Função heurística para estimar o custo de ordenação de uma permutação π é mostrada abaixo, onde A, B e C são parâmetros de ponderação tais que $A, B, C \in \mathbb{R}$ e $A \geq 0$, $B \geq 0$ e $C \geq 0$.

$$hc(\pi) = A \times ent(\pi) + B \times b(\pi) + C \times inv(\pi)$$

Função Heurística: Exemplo

$A = 2 \quad B = 1 \quad C = 3$	
$\pi = (8\ 2\ 3\ 4\ 5\ 6\ 7\ 1)$	$\sigma = (1\ 7\ 6\ 5\ 4\ 3\ 2\ 8)$
$hc(\pi) = 71 \begin{cases} 2 \times ent(\pi) = 2 \times 14 \\ 1 \times b(\pi) = 1 \times 4 \\ 3 \times inv(\pi) = 3 \times 13 \end{cases}$	$hc(\sigma) = 83 \begin{cases} 2 \times ent(\sigma) = 2 \times 18 \\ 1 \times b(\sigma) = 1 \times 2 \\ 3 \times inv(\sigma) = 3 \times 15 \end{cases}$

Função Heurística

Intuição

Intuitivamente esta heurística pode ser vista como uma uma estimativa para o custo de ordenação de uma permutação π.

Benefício

Definição

Seja ρ uma reversão aplicável à permutação π, o benefício de ρ, denotado por $ben_{hc}(\pi, \rho)$, é definido pela seguinte fórmula:

$$ben_{hc}(\pi, \rho) = \frac{hc(\pi) - hc(\pi \cdot \rho)}{|\rho|}$$

Algoritmo guloso

Algoritmo 1 greedyHeuristic

1: **Entrada:** π, A, B, C
2: $\varrho \leftarrow ()$
3: **enquanto** $\pi \neq \iota$ **faça**
4: $\quad \rho \leftarrow$ **chooseReversalMaxBenefit**(π, A, B, C)
5: $\quad$ **se** $\rho \neq \emptyset$ **então**
6: $\quad\quad \varrho \leftarrow \varrho \parallel \rho$
7: $\quad\quad \pi \leftarrow \pi \cdot \rho$
8: $\quad$ **else**
9: $\quad\quad nbk \leftarrow b(\pi)$
10: $\quad\quad$ **enquanto** $b(\pi) = nbk$ **faça**
11: $\quad\quad\quad \rho \leftarrow$ **sortByReversionsTwoAprox**(π)
12: $\quad\quad\quad \varrho \leftarrow \varrho \parallel \rho$
13: $\quad\quad\quad \pi \leftarrow \pi \cdot \rho$
14: $\quad\quad$ **fim enquanto**
15: $\quad$ **fim se**
16: **fim enquanto**
17: **return** ϱ

Resultados

- Utilizamos algoritmos existentes na literatura para avaliar o algoritmo proposto.
- Foram realizados experimentos em dois cenários:
  - Todas as permutações pequenas: $2 \leq n \leq 10$.
  - Amostras de permutações grandes: $10 \leq n \leq 100$.

Permutações pequenas: Métodos comparados

- **TEST**$_1$: $A \in [0, 5]$, $B \in [0, 5]$ e $C \in [0, 5]$.
- **TEST**$_2$: $A \in [0, 10]$, $B \in [0, 10]$ e $C = 0$.
- **OPT**: solução exata de ordenação por reversões ponderadas.
- **REV**: solução exata para reversões com custos unitários.
- **Bender**: algoritmo de aproximação $O(\lg n)$ de Bender et al [4].
- **2-aprox**: O Algoritmo 2-aproximado de *Kececioglu e Sankoff* [8] para a ordenação por reversões com custos unitários.

Métricas de comparação

- $\mathbf{C_{avg}}$: Custo médio de ordenação.
- $\mathbf{C_{max}}$: Custo máximo de ordenação.
- $\mathbf{R_{avg}}$: Razão de aproximação média em relação às soluções exatas.
- $\mathbf{R_{max}}$: Razão de aproximação máxima em relação às soluções exatas.
- **%EX**: Porcentagem de soluções exatas.
- $\mathbf{NR_{avg}}$: Número médio de reversões.
- $\mathbf{NR_{max}}$: Número máximo de reversões.

Comparação entre os métodos

n	Método	C_{avg}	C_{max}	R_{avg}	R_{max}	%EX	NR_{avg}	NR_{max}
	2-Approx	28.78	54	1.44	3.77	3.94	6.83	18
	Bender	26.60	46	1.33	2.91	5.01	8.23	15
10	REV	25.58	54	1.28	2.83	13.33	5.76	9
	TEST$_1$	20.76	33	1.04	1.85	58.75	6.22	12
	TEST$_2$	20.78	35	1.04	1.85	58.62	6.21	13
	OPT	19.95	27	1.00	1.00	100.00	6.02	9

Tabela : Resultados para todas as permutações de tamanho 10.

Combinação de parâmetros

- Realizamos experimentos utilizando a combinação de resultados de configurações de parâmetro.
- As configurações de parâmetros são escolhidas de forma aleatória.
- Considerando o $\mathbf{TEST}_2$: $A \in [0, 10]$, $B \in [0, 10]$ e $C = 0$.

Resultados para permutações pequenas

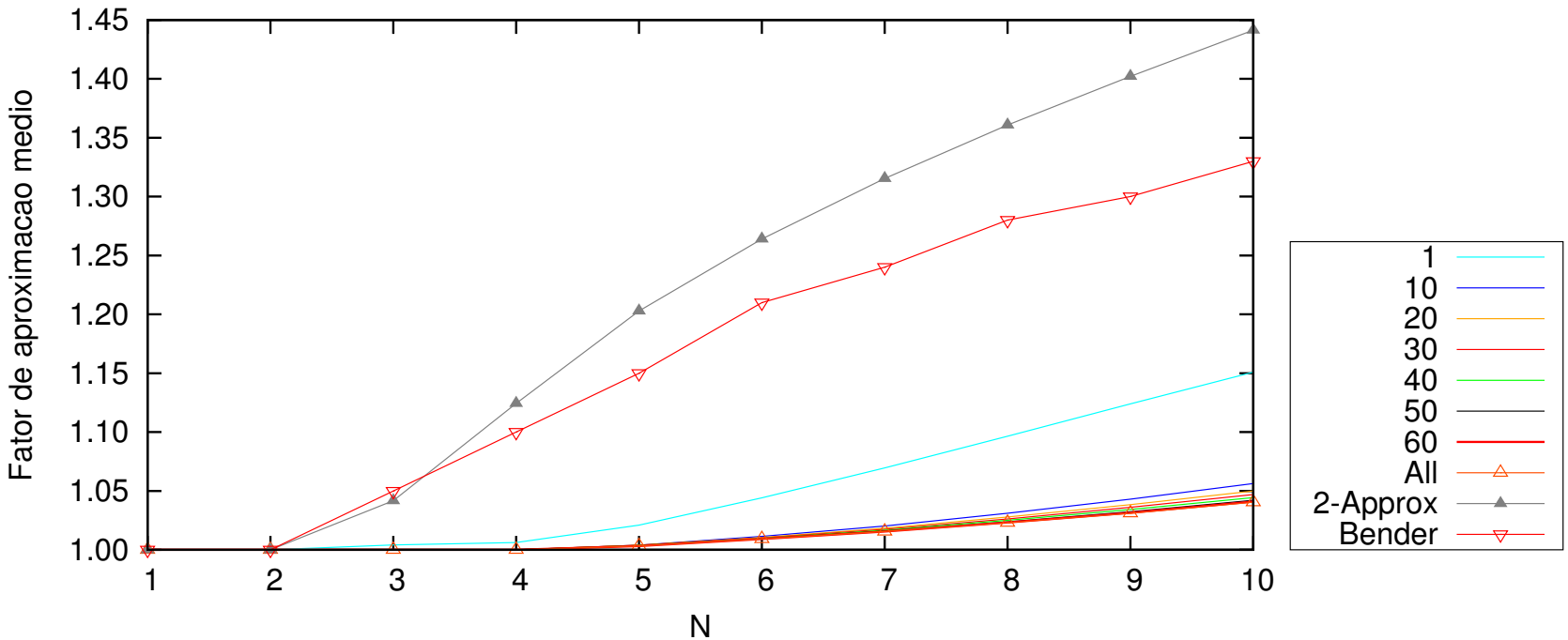

Figura : Fator de aproximação médio para permutações pequenas.

Resultados para permutações grandes

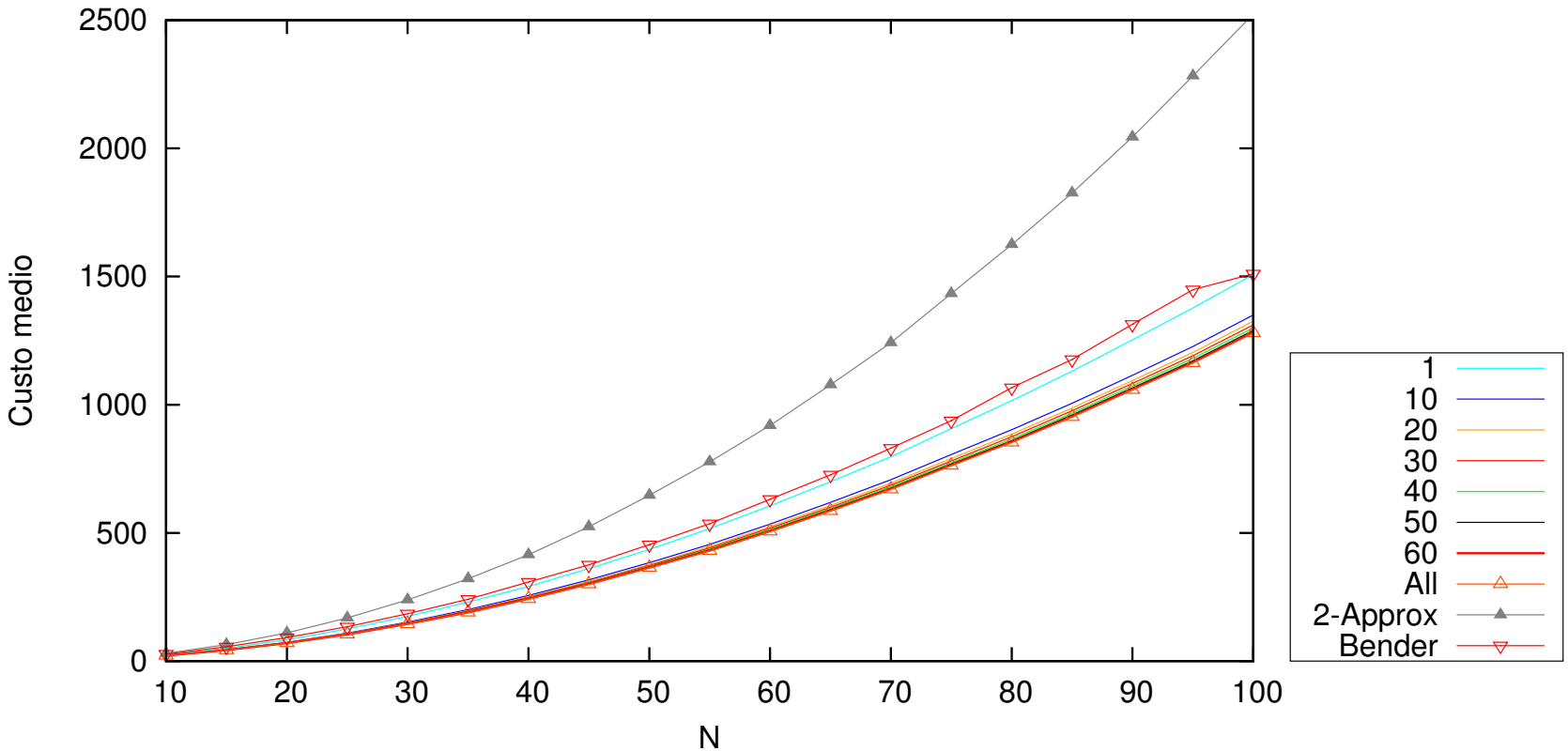

Figura : Custo médio para permutações grandes.

Método paramétrico: Conclusões

- Nosso método apresenta soluções melhores do que os métodos previamente existentes na literatura, tanto para permutações pequenas quanto permutações grandes.
- Para se obter bons resultados é necessário executar apenas um pequeno número de configurações de parâmetros, escolhidos aleatoriamente.

Método paramétrico: Publicações

Os resultados foram apresentados na conferência **ACM BCB'2013** (Conference on Bioinformatics, Computational Biology and Biomedical Informatics), realizada em Washington (EUA), em setembro de 2013, sob o título "Heuristics for the Sorting by Length-Weighted Inversion Problem" (Thiago Silva Arruda, Ulisses Dias e Zanoni Dias) [1].

Parte III

Meta-Heurística GRASP

Introdução ao GRASP

- Nesta seção nós apresentamos uma meta-heurística para as versões com e e sem sinal do problema de ordenação por reversões ponderadas.

Introdução ao GRASP

- *Greedy Randomized Adaptive Search Procedure* (GRASP) foi introduzido por Feo e Resende [7].
- *GRASP* é um modelo de meta-heurística iterativa que combina uma estratégia gulosa com aleatoriedade.

Introdução ao GRASP

Algoritmo 2 ClassicalGRASP

1: **Entrada:** *Max_Iteracoes*, *Semente*
2: *Melhor_Solucao* $\leftarrow +\infty$
3: **para** $k = 1, ..., Max\_Iteracoes$ **faça**
4: $s \leftarrow Construcao\_Gulosa\_Randomizada(Semente)$
5: **se** s não é viável **então**
6: $s \leftarrow Repair(s)$
7: **fim se**
8: $s \leftarrow Busca\_Local(s)$
9: $Melhor\_Solucao \leftarrow Atualizar\_Solucoes(s, Melhor\_Solucao)$
10: $Semente \leftarrow s$
11: **fim para**
12: **return** *Melhor_Solucao*

Introdução ao GRASP

Algoritmo 3 *Construcao_Gulosa_Randomizada*

1: **Entrada:** *Semente*
2: $s \leftarrow \emptyset$
3: Inicializar o conjunto de elementos candidatos
4: Avaliar o custo incremental dos elementos candidatos
5: **enquanto** existir pelo menos um elemento na lista de candidatos **faça**
6: Construir a lista restrita de elementos candidatos (RCL)
7: Selecionar um elemento e de RCL aleatoriamente
8: $s \leftarrow s \cup \{e\}$
9: Atualizar o conjunto de elementos candidatos
10: Reavaliar os custos incrementais
11: **fim enquanto**
12: **return** s

Algoritmo 4 *Busca_Local*

1: **Entrada:** s
2: **enquanto** s não é localmente ótima **faça**
3: Procure $s^* \in N(s)$ tal que $f(s^*) < f(s)$
4: $s \leftarrow s^*$
5: **fim enquanto**
6: **return** s

Introdução ao GRASP: características

- **Guloso**: constrói a lista *RCL* utilizando os melhores elementos.
- **Aleatório**: escolhe aleatoriamente um elemento da *RCL*.
- **Adaptativo**: realiza a construção randomizada a cada iteração.

Meta-Heurística GRASP

Elementos do método

- Solução inicial
- Vizinhança
- Busca Local

Meta-Heurística GRASP

Solução inicial

- Uma solução é uma sequência de permutações $s =< s_0, s_1, \ldots, s_m >$ tal que s_k é diferente de s_{k-1} por uma reversão, $1 \leq k \leq m$, $s_0 = \pi$ e $s_m = \iota$.
- Exemplo:
  $s =< (-5\ +2\ -3\ +1\ +4), (-5\ +2\ -1\ +3\ +4), (-4\ -3\ +1\ -2\ +5), (+2\ -1\ +3\ +4\ +5), (+1\ +2\ +3\ +4\ +5) >$.
- Utilizamos métotos distintos para gerar solução inicial para cada variação do problema.

Meta-Heurística GRASP: Vizinhança

$$
\begin{aligned}
s = < & (+2\ +1\ -5\ \cdots\ -3\ +16\ +17),\\
& (+2\ +3\ +4\ \cdots\ -1\ +16\ +17),\\
& (-5\ -4\ -3\ \cdots\ -1\ +16\ +17),\\
& \boxed{x \text{ permutations}}\\
& (-5\ -4\ -3\ \ldots\ +15\ +16\ +17),\\
& (+1\ +2\ +3\ \cdots\ +15\ +16\ +17)>
\end{aligned}
$$

$$
\begin{aligned}
s' = < & (+2\ +1\ -5\ \cdots\ -3\ +16\ +17),\\
& (+2\ +3\ +4\ \cdots\ -1\ +16\ +17),\\
& (-5\ -4\ -3\ \cdots\ -1\ +16\ +17),\\
& \boxed{y \text{ permutations}}\\
& (-5\ -4\ -3\ \ldots\ +15\ +16\ +17),\\
& (+1\ +2\ +3\ \cdots\ +15\ +16\ +17)>
\end{aligned}
$$

Figura : Solução s' na vizinhança de s.

Meta-Heurística GRASP

Vizinhança: escolha de janelas

- A cada iteração, utilizamos o algoritmo *roulette wheel* para escolher uma janela para aplicar busca local.
- A probabilida de uma janela ser escolhida é proporcional ao respectivo tamanho.

Meta-Heurística GRASP

Busca Local

Para construir uma nova janela que começa com α e termina com β executamos iterativamente os seguintes passos:

1. Nós construímos uma lista de reversões que diminuem o número de *breakpoints*.
2. Selecionamos uma quantidade de reversões com maior benefício (parâmetro `numero_reversoes`).
3. Utilizamos o algoritmo *roulette wheel selection* para escolher uma reversão: probabilidade de escolha proporcional ao benefício.

Meta-Heurística GRASP

Busca Local

- Utilizamos o algoritmo *GRIMM*, de Tesler e Glenn [11], para gerar o conjunto de reversões.
- *GRIMM* fornece soluções para o problema de ordenação por reversões de custo unitário.

Meta-Heurística GRASP

Definição (Busca Local: Benefício de uma reversão)

Seja π uma permutação (com sinal ou sem sinal) e seja ρ uma reversão, definimos o benefício de aplicar ρ em π como: $ben(\pi, \rho) = \frac{ent(\pi) - ent(\pi \cdot \rho)}{cost(\rho)}$.

Meta-Heurística GRASP: Resultados

- Executamos testes experimentais para avaliar o nosso método.
- Foram utilizadas 1000 amostras de permutações para cada tamanho n, $n \in \{10, 20, 30...100\}$.

Meta-Heurística GRASP: Resultados

Permutações com sinal: melhoria da solução inicial

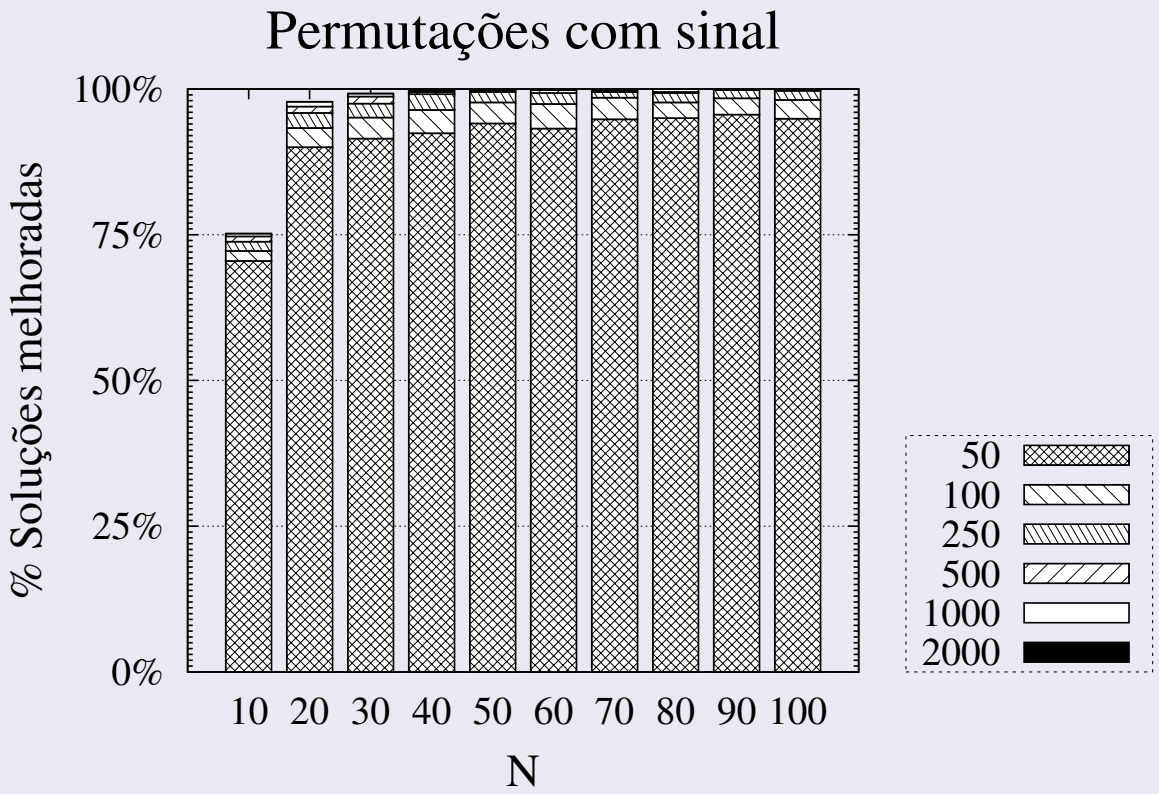

Figura : Porcentagem de vezes que o nosso método melhorou a solução inicial.

Meta-Heurística GRASP: Resultados

Permutações com sinal: melhoria da solução inicial

- Em geral, a solução inicial foi melhorada em 97.1% dos casos.
- Para $n > 50$, nós observamos que 99.9% das soluções iniciais foram melhoradas.
- Observamos também que 91.2% dos melhoramentos ocorreram nas primeiras 50 iterações.

Meta-Heurística GRASP: Resultados

Permutações sem sinal: melhoria da solução inicial

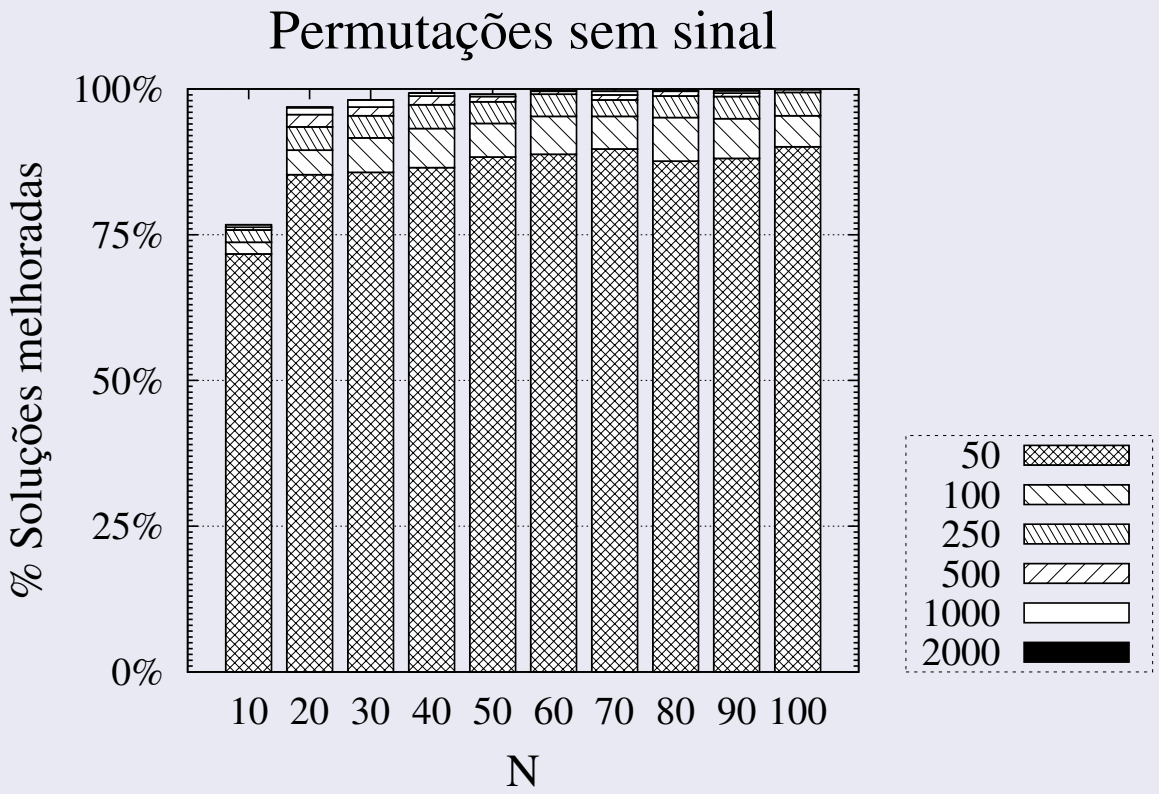

Figura : Porcentagem de vezes que a nossa heurística melhorou a solução inicial.

Meta-Heurística GRASP: Resultados

Permutações sem sinal: melhoria da solução inicial

- Em geral, a solução inicial foi melhorada em 96.8% dos casos.
- Para $n > 50$, nós observamos que 99.7% das soluções iniciais foram melhoradas.
- Observamos também que 86.2% dos melhoramentos ocorrem nas primeiras 50 iterações.

Meta-Heurística GRASP: Resultados

Permutações com sinal: porcentagem de melhoria

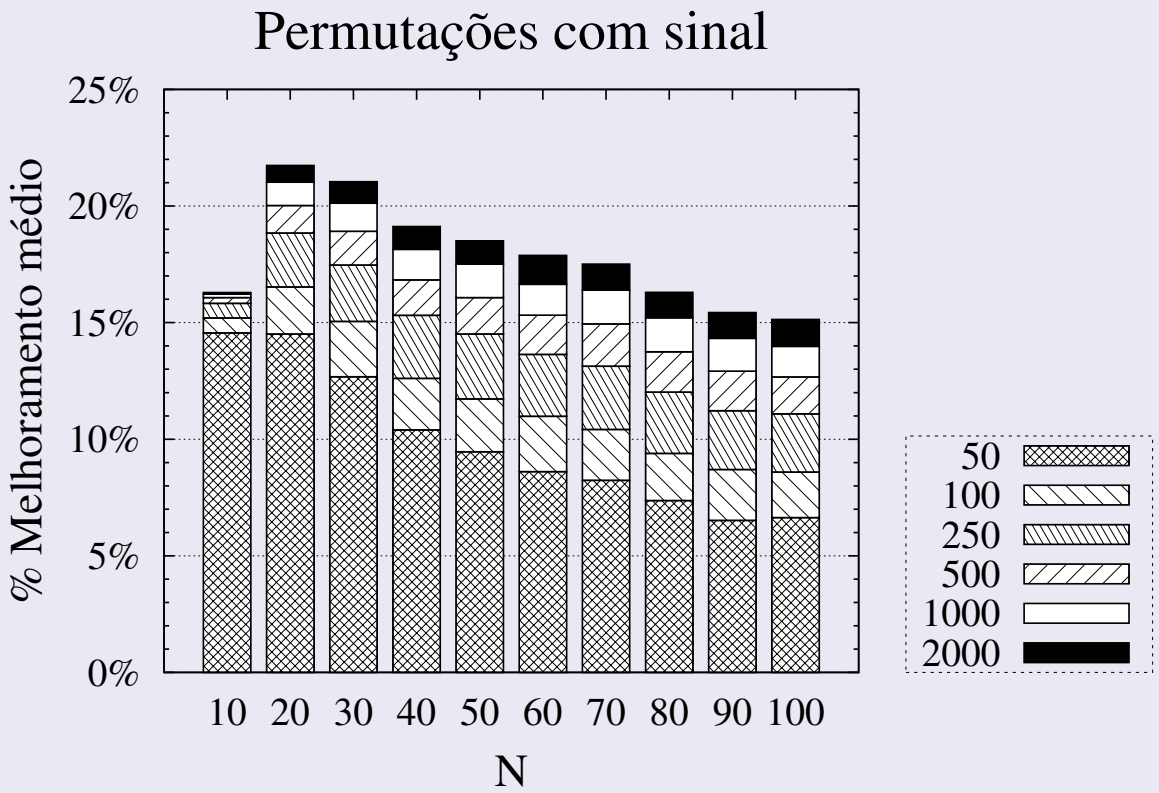

Figura : Porcentagem de melhoria obtida pelo nosso método em permutações com sinal.

Meta-Heurística GRASP: Resultados

Permutações com sinal: melhoria da solução inicial

- Nós observamos que nossas soluções custam por volta de 17.9% menos do que a solução inicial.
- Nós observamos uma tendência de que mais iterações devem ser executadas conforme se aumenta o tamanho das permutações.

Meta-Heurística GRASP: Resultados

Permutações sem sinal: porcentagem de melhoria

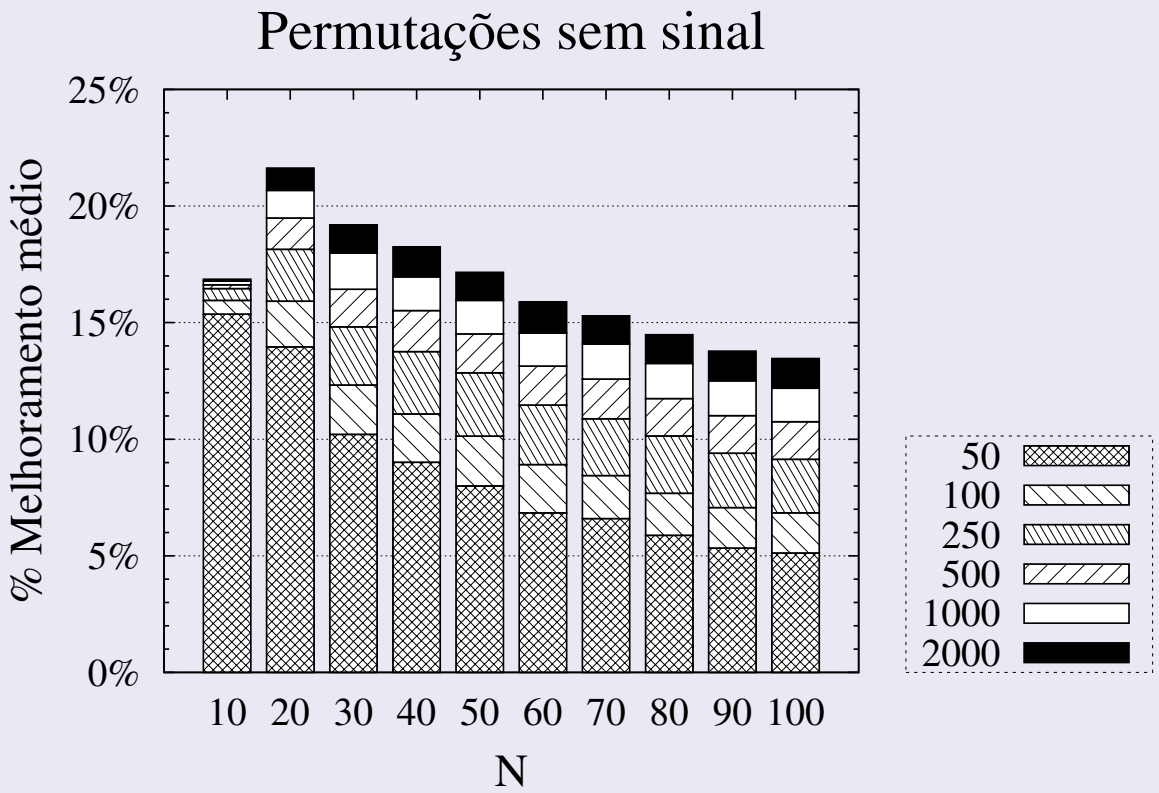

Figura : Porcentagem de melhoramento pelo nosso método em permutações sem sinal.

Meta-Heurística GRASP: Resultados

Permutações sem sinal: melhoria da solução inicial

- Soluções fornecidas pelo nosso método custam 16.6% menos do que a solução inicial.
- Em geral o comportamento mostrado aqui é similar ao caso com sinal.

Meta-Heurística GRASP: Resultados

Permutações com sinal: análise comparativa

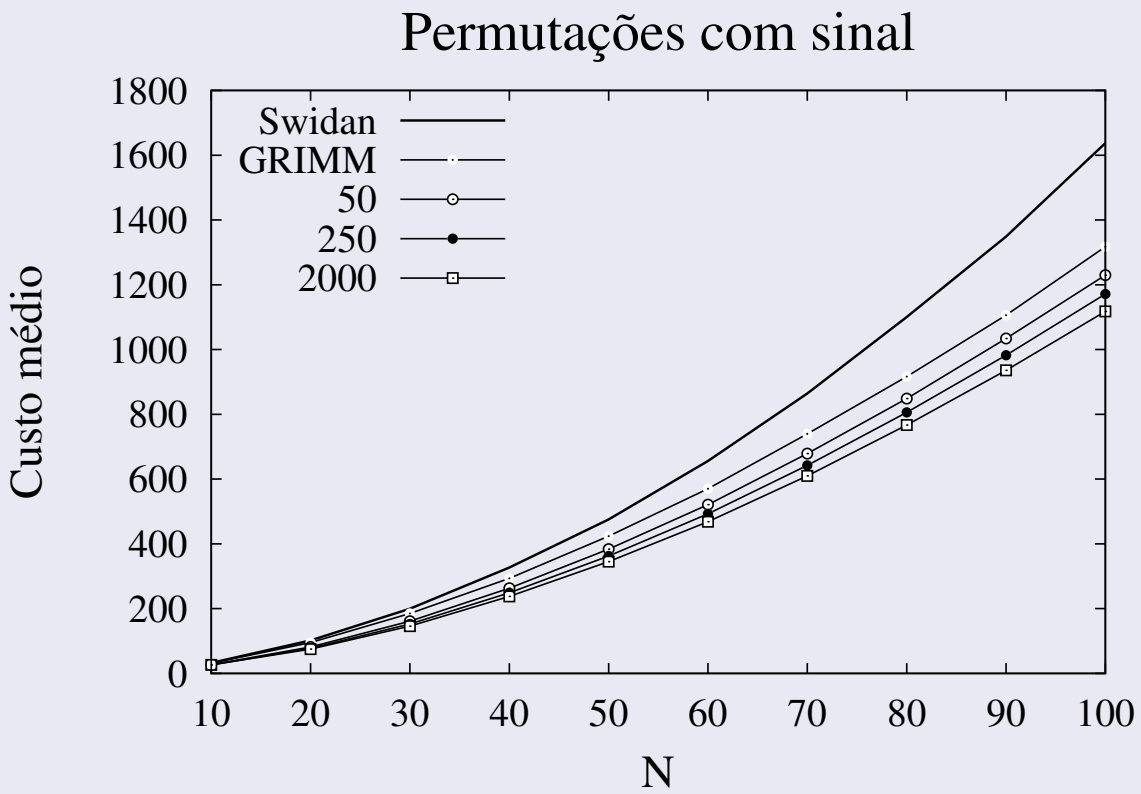

Figura : Análise comparativa entre três algoritmos para permutações com sinal.

Meta-Heurística GRASP: Resultados

Permutações sem sinal: análise comparativa

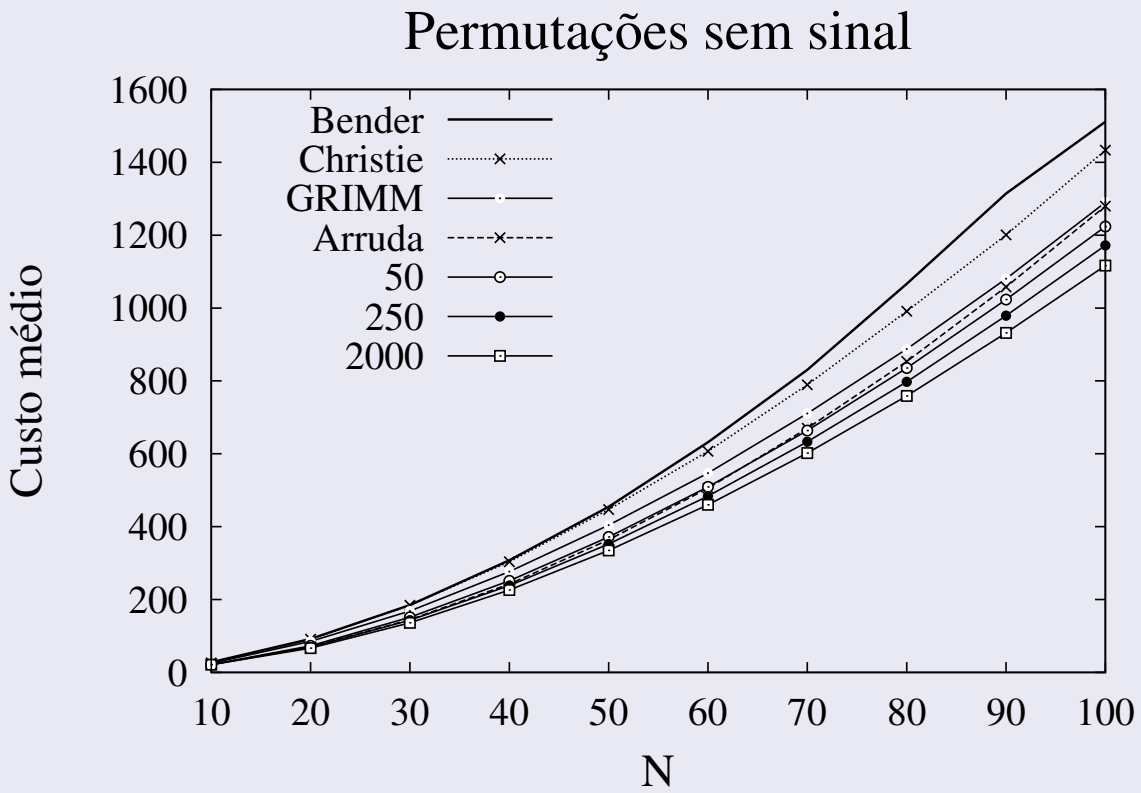

Figura : Análise comparativa entre cinco algoritmos para permutações sem sinal.

Meta-Heurística GRASP: Conclusões

- Os parâmetros podem ser configurados facilmente e impactam o tempo de processamento e a qualidade das soluções.
- Conseguimos melhorar as soluções iniciais com uma margem significativa: em mais de 96% dos casos.
- Nossa solução custa 17.9% e 16.6% menos do que a solução inicial para permutações com sinal e sem sinal, respectivamente.
- Quanto maior o tamanho da permutação, mais fácil se torna melhorar a solução inicial.
- Nós conseguimos obter melhores resultados do que outras abordagens previamente disponíveis na literatura.

Meta-Heurística GRASP: Publicações

- **1st International Conference on Algorithms for Computational Biology** (AlCoB'2014), realizada em Tarragona (Espanha), em julho de 2014, com o título "Heuristics for the Sorting by Length-Weighted Inversions Problem on Signed Permutations"(Thiago Arruda, Ulisses Dias e Zanoni Dias) [2].
- **IEEE/ACM Transactions on Computational Biology and Bioinformatics** com o título "A GRASP-based Heuristic for the Sorting by Length-Weighted Inversions Problem" (Thiago Arruda, Ulisses Dias e Zanoni Dias) [3].

O Problema da Ordenação por Reversões Ponderadas

Thiago da Silva Arruda
Zanoni Dias

Instituto de Computação - Universidade Estadual de Campinas

28 de Abril de 2016

Obrigado!

[1] T. S. Arruda, U. Dias, and Z. Dias. Heuristics for the sorting by length-weighted inversion problem. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, pages 498–507. ACM, 2013.

[2] T. S. Arruda, U. Dias, and Z. Dias. Heuristics for the sorting by length-weighted inversions problem on signed permutations. In *Algorithms for Computational Biology -1st International Conference on Algorithms for Computational Biology, AlCoB'2014, Tarragona, Spain*, pages 59–70, 2014.

[3] T. S. Arruda, U. Dias, and Z. Dias. A GRASP-based heuristic for the sorting by length-weighted inversions problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PP(99):1–1, 2015.

[4] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting by length-weighted reversals. *Journal of Computer and System Sciences*, 74(5):744 – 774, 2008.

[5] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA'2002, pages 200–210, London, UK, 2002. Springer-Verlag.

[6] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the first annual international conference on Computational molecular biology*, RECOMB'1997, pages 75–83, New York, NY, USA, 1997. ACM.

[7] T. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[8] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement, 1995.

[9] R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:2002, 2002.

[10] E. Tannier, A. Bergeron, and M-F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, April 2007.

[11] G. Tesler. GRIMM: genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.