Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Estruturas de índices sucintas para grafos de subsequencias

Proposta de Trabalho de Graduação

Aluno: Vitor Travassos Castelo Branco
(vtcb@cin.ufpe.br)
Orientador: Paulo Gustavo Soares da Fonseca
(paguso@cin.ufpe.br)
Área: Teoria da Computação / Algoritmos

Recife, 04 de Abril de 2018

Resumo

Boa parte das ferramentas para o sequenciamento do DNA baseado nas plataformas de alto desempenho ditas de nova geração, especificamente as destinadas à montagem dos fragmentos sequenciados, utilizam estruturas de dados para grafos de subsequencias. Devido ao enorme volume de dados, um dos principais gargalos dessas representações é espaço em memória exigido. Várias representações compactas vêm sendo propostas nos últimos anos. Entretanto, existe uma relação inversa entre o espaço de memória e a eficiência/flexibilidade das operações suportadas. Mais recentemente, têm-se explorado a conexão entre os grafos de subsequencias e os índices de texto. Neste projeto, pretendemos produzir uma implementação dos Grafos de de Bruijn baseada numa representação sucinta da árvore de sufixos. Essa implementação será analisada e comparada a alternativas disponíveis na literatura para aferir a sua viabilidade em termos de espaço e tempo.

Introdução

Atualmente, o processo de sequenciamento de DNA é efetuado principalmente utilizando-se as plataformas de sequenciamento de alto desempenho ditas de "nova geração" (*Next-Generation Sequencing—NGS*) [1]. Essas tecnologias produzem um enorme volume de fragmentos curtos (comprimento abaixo das centenas) que precisam ser *montados*, i.e., alinhados e combinados, para reconstruir sequências originais de bilhões de letras.

As ferramentas para montagem de fragmentos NGS são majoritariamente baseadas nos chamados *Grafos de de Bruijn* (GDB) [2]. No GDB de ordem k construído a partir do conjunto de fragmentos $\mathcal{S}$, $G(\mathcal{S})$, os nós correspondem às subsequencias de comprimento k (k-mers) das cadeias em $\mathcal{S}$, e dois k-mers (nós) são unidos por uma aresta desde que haja uma sobreposição de tamanho $k-1$, de forma que as arestas correspondem aos $k+1$-mers de $\mathcal{S}$. Efetuar a montagem de fragmentos usando GDB envolve problemas como o de encontrar *Caminhos Eulerianos*, que admite solução em tempo polinomial. Entretanto, um dos principais limitadores quanto ao emprego dessas técnicas é o espaço de memória exigido pelos GDB que, se representado explicitamente, pode requerer centenas de GigaBytes [3]. Diante disto, diversos esforços vêm sendo empreendidos para desenvolver estruturas de dados eficientes do ponto de vista de espaço, permitindo todavia operações sobre o GDB em tempo comparável a uma representação tradicional.

A representação e manipulação de GDB envolve técnicas algorítmicas específicas que visam a minimisar o uso de memória principal. Essas técnicas incluem o desenho de algoritmos otimizados para memória externa/cache, o desenvolvimento de *estruturas de dados sucintas* [4], e o uso de compressão de dados. Os métodos descritos na literatura procuram representar ou o conjunto de vértices (k-mers) ou arestas ($K+1$-mers) com estruturas de dados específicas de forma a permitir que as operações de navegação como 'quais são os sucessores/predecessores de um nó v?' sejam respondidas indiretamente através de operações primitivas de baixo nível.

Num dos primeiros esforços, Conway e Bromage [3] propuseram-se a representar $G(\mathcal{S})$ a partir do seu conjunto de arestas E como um conjunto de $k+1$-mers ordenados lexicograficamente. Cada uma das possíveis arestas pode ser identificada com uma posição de um bit array comprimido [5]. Operações de navegação como são realizadas através de consultas ao bit array. Os autores obtiveram assim cerca de 28 bits por aresta para o GDB do genoma de um humano.

Pell et al [6] propuseram uma representação mais econômica que consiste em armazenar o conjunto dos vértices V (k-mers) de $G(\mathcal{S})$ num *Filtro de Bloom (FB)* [7]. Apesar de consultas ao FB poderem retornar falsos-positivos, Pell et al sustentam que a representação mantém propriedades adequadas para tarefas como a decomposição do GDB em componentes para a posterior montagem de trechos contíguos (*contigs*) da sequência original. Chikhi e Rizk [8] estenderam a representação de Pell ara representar explicitamente um conjunto de *falsos-positivos críticos* (FPc). Com isso, obteve-se uma representação exata que permitiu que o GDB do genoma humano (HapMap: NA18507) fosse armazenado em cerca de 6 GigaBytes. Posteriormente, essa estrutura foi refinada por Salikhov et al [9] com a utilização em cascata de FB para representar o conjunto de FPc (ao invés de hash tables), obtendo uma representação cerca de 30–40% mais eficiente em termos de memória, e 18–30% em termos de tempo de percurso do GDB.

A representação apresentada por Bowe et al [10] foi inspirada pela *Transformada de Burrows-Wheeler* (BWT) [11]. A BWT é uma permutação de um texto que permite a compressão e a busca eficiente por padrões sobre ele. Bowe et al propuseram representar o GDB de uma cadeia S de comprimento n como uma permutação W similar à sua BWT. Essa cadeia, juntamente com um bit

array de tamanho n e mais as contagens de cada caractere em S, permitem a navegação sobre o GDB. No total, essa representação do GDB requer cerca de 2.5 GigaBytes para o mesmo data set do genoma humano (HapMap: NA18507) mencionado acima. Boucher et al [12] estenderam a representação de [10] para permitir operações de navegação com o k variável e escolhido *on-the-fly*. Belazzougui et al [13] refinaram ainda mais a representação de [12], introduzindo, uma estrutura bidirecional [14]. Essa modificação tem um custo acrescido de $O(n \lg k)$ bits de espaço (n =número de vértices) sobre a representação base de [10], permitindo todavia navegar sobre o GDB nos dois sentidos e com k variável em tempo $O(\lg k)$ por aresta.

Outras propostas de representação são baseadas *FM-index* [15], que é uma extensão sucinta da BWT. Rødland [16] propôs uma variação do FM-index, chamado *k*FM-index, para representar diretamente o conjunto de $k+1$-mers (arestas) de um conjunto de sequências em ordem lexicográfica. Essa representação requer cerca de 5 bits por nó, o que é suficiente para permitir a navegação no GDB no sentido reverso das arestas. Chikhi et al [17] também propuseram uma representação chamada DBGFM que consiste em representar através de um FM-index a cadeia correspondente à concatenação dos rótulos das arestas de todos os *caminhos simples maximais* (i.e. contendo apenas nós sem bifurcação) de $G(\mathcal{S})$. A implementação de [17] usa ainda a codificação de Huffman para comprimir a BWT, demandando apenas $\approx$ 1.5 GigaBytes para o data set NA18507 do genoma humano referido anteriormente.

Embora os métodos anteriores já evidenciem a relação entre os GDB e as estruturas de índice, Cazaux et al [18] propuseram-se a investigar mais aprofundadamente essa conexão, em particular com as árvores de sufixos, que é um índice de referência muito estudado [19]. Nesse estudo teórico, os autores apresentaram algoritmos lineares para a construção de GDB a partir de árvores de sufixos, árvores de sufixos truncadas [20], e arrays de sufixos [21]. Essa estratégia pode tirar proveito da construção prévia de um desses índices para uma outra tarefa, como a correção de erros de sequenciamento, por exemplo.

Objetivos

Conforme visto acima, o tema deste projeto é um tópico ativo de pesquisa com muitas contribuições recentes. Apesar da literatura relativamente extensa, os desenvolvimentos ainda não estão consolidados. Em particular, observa-se uma grande disparidade metodológica nas abordagens, com diversas estruturas sendo adotadas. Essas estruturas refletem as necessidades e objetivos particulares dos estudos e das ferramentas para as quais foram desenvolvidas, e implementam portanto um conjunto não-uniforme de funcionalidades. Além disso, as estruturas apresentam níveis muito distintos de maturidade, algumas delas já bem integradas em ferramentas mais estáveis e.g. [8, 9], outras ainda em fase de protótipo, e.g. [10, 16], e outras para as quais não conhecemos implementações, e.g.[18].

O principal objetivo deste projeto é explorar a conexão teórica entre os GDB e as estruturas de índices, e sua realização prática. Em particular, o recente trabalho de Cazaux et al [18] aponta para as conexões entre as árvores de sufixos e os GDB. Neste trabalho iremos desenvolver uma implementação sucinta baseada em Árvores de Sufixos Comprimidas [22]. Esta implementação será avaliada do ponto de vista teórico e prático quanto ao (i) tempo e memória de construção da estrutura, (ii) espaço de memória final da estrutura, (iii) tempo das operações de navegação através da simulação de percursos no grafo. Os resultados serão comparados a outras implementações mencionadas acima.

O software desenvolvido no projeto será acrescentado a uma biblioteca básica em escrita em C, atualmente em desenvolvimento, para que possa ser útil à comunidade de processamento de texto e Biologia Computacional.

Metodologia

O projeto será organizado nas atividades descritas abaixo e desenvolvidas conforme o cronograma a seguir.

T0. Preparação da proposta

Nesta fase inicial, aluno e orientador farão uma série de reuniões para definição do problema e escopo do projeto. O orientador apresentará a bibliografia básica sobre o tema e os possíveis pontos a serem abordados, e ambos decidirão sobre aqueles a serem desenvolvidos com base no interesse mútuo.

T1. Revisão e acompanhamento bibliográfico

Essa tarefa será executada de maneira mais acentuada no início do projeto. A atividade consiste num estudo dirigido centrado no material bibliográfico mais diretamente relacionado às estruturas de dados e algoritmos a serem implementados no projeto. Ao final dessa fase inicial, espera-se que o aluno possa manter-se atualizado e aprofundar-se em pontos específicos de maneira mais autônoma.

T2. Implementação das estruturas

Esta tarefa corresponde à principal componente de desenvolvimento do projeto. O aluno deverá, em interação com o orientador, estudar detalhadamente algoritmos e estruturas de dados relativos aos índices e grafos de subsequências, e desenvolver uma implementação de referência em nível de produção para os mesmos, com base em uma biblioteca em C atualmente em desenvolvimento.

T3. Realização dos Testes

Nesta tarefa, o aluno deverá fazer uma análise experimental comparativa do desempenho dos métodos propostos e implementados relativamente a outras alternativas públicas identificadas num levantamento inicial na tarefa T1. Pode ser necessário obter dados reais e produzir dados sintéticos que simulem uma situação limite de estresse para os métodos.

T4. Redação e revisão da monografia

A monografia produzida deverá conter: (1) uma breve revisão bibliográfica do estado da arte, fruto da tarefa T1, (2) descrição detalhada dos seus desenvolvimentos, incluindo a análise teórica dos algoritmos e estruturas propostas em termos de tempo e espaço, (3) uma análise crítica com base em resultados experimentais da tarefa T3, e (4) uma discussão com conclusões gerais sobre o projeto.

T5. Preparação da Apresentação

Finalmente, o aluno preparará a apresentação da defesa do TG com o resumo dos desenvolvimentos e resultados obtidos.

O aluno já possui alguma familiaridade com a área, tendo inclusive cursado uma disciplina eletiva oferecida pelo orientador, pelo que poderá por-se rapidamente em desenvolvimento. O acompanhamento será feito pessoalmente através de reuniões semanais. Todo material desenvolvido é compartilhado entre orientador e aluno através de um repositório privado na plataforma GitHub.

Cronograma de atividades

	Mar		Abr				Mai				Jun				Jul	
Preparação da proposta	●	●														
Revisão bibliográfica†	●	●	○	○	○	○	○	○								
Implementação das estruturas			●	●	●	●	●	●	●	●	●					
Realização dos testes‡				○	○	○	○	○	○	●	●	●				
Redação e revisão da monografia											●	●	●	●		
Preparação da apresentação															●	●

(†) ● = levantamento inicial, ○ = aprofundamento

(‡) ● = experimentos formais , ○ = testes *ad hoc* unitários

Referências

[1] Mihai Pop and Steven L Salzberg. Bioinformatics challenges of new sequencing technology. *Trends in Genetics*, 24(3):142–149, 2008.

[2] Phillip E C Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, 2011.

[3] Thomas C Conway and Andrew J Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, feb 2011.

[4] Guy Jacobson. *Succinct static data structures*. PhD thesis, Carnegie Mellon University, 1989.

[5] Daisuke Okanohara and Kunihiko Sadakane. Practical Entropy-Compressed Rank / Select Dictionary. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments - ALENEX 2007*, pages 60–70, New Orleans, 2007.

[6] Jason Pell, Arend Hintze, Rosangela Canino-Koning, et al. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proceedings of the National Academy of Sciences*, 109(33):13272–13277, 2012.

[7] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[8] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom Filter. *Algorithms for Molecular Biology*, 8:22, 2013.

[9] Kamil Salikhov, Gustavo Sacomoto, and Gregory Kucherov. Using cascading Bloom filters to improve the memory usage for de Brujin graphs. *Algorithms for Molecular Biology*, 9:2, feb 2014.

[10] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *Proceedings of the Workshop on Algorithms in Bioinformatics - WABI 2012*, pages 225–235, Ljubljana, 2012. Springer, Berlin, Heidelberg.

[11] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Systems Research*, R(124):24, 1994.

[12] Christina Boucher, Alex Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-Order de Bruijn Graphs. In *Proceedings of the 2015 Data Compression Conference - DCC 2015*, pages 383–392, Snowbird, 2015.

[13] Djamal Belazzougui, Travis Gagie, Veli Mäkinen, Marco Previtali, and Simon J Puglisi. Bidirectional variable-order de Bruijn Graphs. In *LATIN 2016: Theoretical Informatics, 12th Latin American Symposium*, pages 164–178, 2016.

[14] Thomas Schnattinger, Enno Ohlebusch, and Simon Gog. Bidirectional search in a string with wavelet trees and bidirectional matching statistics. *Information and Computation*, 213:13–22, 2012.

[15] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398, Redondo Beach, 2000.

[16] Einar Andreas Rødland. Compact representation of k-mer de Bruijn graphs for genome read assembly. *BMC bioinformatics*, 14(1):313, 2013.

[17] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T Simpson, and Paul Medvedev. On the Representation of De Bruijn Graphs. *Journal of Computational Biology*, 22(5):336–352, may 2015.

[18] Bastien Cazaux, Thierry Lecroq, and Eric Rivals. Linking indexing data structures to de Bruijn graphs: Construction and update. *Journal of Computer and System Sciences*, In press, jul 2016.

[19] A Apostolico. The myriad virtues of subword trees. In *Combinatorial Algorithms on Words*, volume 12, pages 85–96, 1985.

[20] Joong Chae Na, Alberto Apostolico, Costas S. Iliopoulos, and Kunsoo Park. Truncated suffix trees and their application to data compression. *Theoretical Computer Science*, 304(1-3):87–101, 2003.

[21] Udi Manber and Gene Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[22] Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.

Possíveis Avaliadores

1. Profa. Katia Guimarães
2. Prof. Pedro Manhães

Assinaturas

Recife, 04 de Abril de 2018

Aluno: Vitor Travassos Castelo Branco

Orientador: Paulo Gustavo Soares da Fonseca