Relatório Algoritmos para Bioinformática

Mestrado em Bioinformática e Biologia Computacional, FCUP
Porto, 18 de Junho de 2019
Catarina Lopes Alves

Introdução

O presente trabalho consiste na implementação de uma pipeline bioinformática destinada a estudar a relação entre um conjunto de sequências, partindo de uma sequência de referência. Uma pipeline consiste na implementação de um conjunto de tarefas interdependentes, executadas de forma consecutiva e automática. Neste caso, a pipeline deve incluir as seguintes tarefas: importação das sequências; blast; alinhamento múltiplo; construção de uma árvore filogenética; construção de um grafo.

Descrição e Estratégias de Implementação

O trabalho solicitado foi executado com recurso a um conjunto de classes desenvolvidas ao longo do ano, completadas com novos métodos ou adaptadas sempre que relevante, e ainda à implementação de novos métodos que resultam na construção da pipeline propriamente dita constantes no script run_me.py. O trabalho em questão foi realizado com sequências proteicas no entanto poderia ser aplicado a outro tipo de sequências biológicas. De seguida será feita uma breve descrição das estratégias de implementação aplicadas em cada passo:

- **Importação/Exportação das sequências em formato fasta**
  Os métodos necessários para a execução desta tarefa, e que vêm sido utilizados nos restantes trabalhos, encontram-se implementados no módulo Sequence_import.py, brevemente descritos na Tabela 1.

Tabela 1. Descrição dos métodos implementados na classe Sequence_import.

Sequence_import	
Método	**Descrição**
read_fasta(file)	Importação de múltiplas sequências a partir de um ficheiro em formato fasta. Retorna a lista das sequências importadas.
read_fasta_titles(file)	Importe dos títulos das sequencias a partir de um ficheiro em formato fasta. Retorna uma lista dos títulos importados.
read_fasta_specie(file)	Adaptação da função read_fast_title(file) para realizar o importe da espécie de cada sequência. Neste caso em particular, essa informação era fornecida no titulo entre parêntesis retos, tendo-se aplicado expressões regulares para a obter
export_fasta(file, seqs, seqsID)	Exportação de um conjunto de sequências e respetivas identificações para um ficheiro em formato fasta.

- **Blast**
  Esta tarefa consiste em a partir de um conjunto de sequências que servirão de biblioteca encontrar as 10 sequências mais semelhantes a uma sequência de referência. Neste caso em particular deveríamos ainda ter em atenção a que não fossem selecionadas espécies repetidas. Este procedimento é executado através da implementação de uma versão simplificada do blast na classe MyBlast.py. O Blast é um algoritmo de procura de sequências semelhantes numa base de dados baseado numa abordagem heurística de k-indexing. De uma forma simplificada, as sequências são divididas em pequenos fragmentos de tamanho k e é feita uma posterior análise da presença de

fragmentos em comum entre a sequência de referência e a base de dados, seguido da respetiva extensão, para inferir a semelhança entre elas. Um objeto do tipo MyBlast é composto por uma base de dados à qual podem ser adicionadas ou removidas (método implementado adicionalmente) sequências, um valor de w que define o tamanho dos fragmentos a considerar e ainda um mapa que corresponde a um dicionário onde são guardados os fragmentos de tamanho w que compõe a sequência de referência e a(s) respetiva(s) posição(ões). Na Tabela 2 é feita uma breve descrição dos métodos implementados nesta classe.

Tabela 2. Descrição dos métodos implementados na classe MyBlast.

MyBlast	
Método	**Descrição**
readDatabase(filename)	Importação da base de dados a parti de um ficheiro em formato fasta. A base de dados é representada por uma lista.
addSequenceDB(seq)	Adição de uma sequência à base de dados
removeSequenceDB(index)	Remove a sequência da base de dados presente em determinado índice.
buildMap(query)	Construção do mapa da sequência de referência.
getHits(seq, query)	Retorna a lista de tuplos correspondentes às posições onde são detetados fragmentos comuns entre sequência de referência e uma sequencia da base de dados
extendsHits(seq, hit, query)	Para determinado hit entre a sequencia de referencia e uma sequência a base de dados, realiza a extensão do tamanho do fragmento em ambas as direções enquanto for vantajoso, isto é, enquanto promover a semelhança entre as duas sequências
hitsBestScore(seq, query)	Para um conjunto de hits entre duas sequências e respetivas extensões identifica e retorna o melhor score encontrado
bestAlignment(query)	Com a aplicação dos métodos descritos a cima, identifica o melhor alinhamento entre a sequência de referencia e uma sequência da base de dados, retornando o índice correspondente à sequência da base de dados, entre outras informações.

È importante ressalvar dois aspetos: o método removeSequenceDB foi adicionado à classe e será utilizado na pipeline; o método readDatabase é idêntico ao método read_fasta implementado no modulo Sequence_import e poderia ser substituído pelo mesmo em vez de se realizar a implementação de dois método redundantes.

- **Alinhamento Múltiplo**
  Este passo consiste no alinhamento múltiplo das 11 sequências resultantes da etapa anterior utilizando a classe MultipleAlignment. Um objeto MultipleAlignment possui como atributos a lista das sequências envolvidas e um objeto da classe PairwiseAlignment descrita em trabalhos anteriores. Este objeto será utilizado no alinhamento múltiplo das sequências uma vez que este será feito de forma gradual entre cada dois pares de sequências utilizando para isso o algoritmo de Needlman Wunsch. Isto é, primeiro é realizado o alinhamento pareado das duas primeiras sequências resultando uma sequência consensus. De seguida, será feito o alinhamento da próxima sequência com o consensus obtido anteriormente, o que se irá repetir de forma consecutiva para todas as sequências. Em cada iteração, o alinhamento é recuperado com recurso à matriz trace-back (T) do alinhamento, atendendo à penalização de gap (neste caso fio utilizado um valor de -8) e armazenado numa lista que será retornada no final. De notar que as sequências devem ser do tipo MySeq e ainda que o resultado final será do tipo MyAlign. Estes objetos advém da aplicação de classes implementadas durante o ano que visam representar de forma completa e correta sequências

biológicas e alinhamentos, respetivamente. Na Tabela 3 é feita uma breve descrição dos métodos implementados na classe MultipleAlignment.

Tabela 3. Descrição dos métodos implementados na classe MultipleAlignment.

MultipleAlignment	
Método	**Descrição**
num_seqs(self)	Retorna o número de sequências envolvidas no alinhamento.
add_seq_alignment(alignment, seq)	Adiciona uma nova sequência (seq) a um alinhamento múltiplo em curso, alignment, que deve ser um objeto do tipo MyAlign. A nova sequência será alinhada ao consensus obtido no passo anterior segundo a abordagem Needleman Wunsch
align_consensus()	Realiza o alinhamento das primeiras duas sequências a considerara no alinhamento múltiplo, obtendo a sequência consensus entre elas. Realiza ainda a chamada iterativa da função descrita a cima sobre cada uma das restantes sequências a alinhar. Retorna um objeto do tipo MyAlign.
printMat(mat)	Realiza o preatty printing de uma matriz passada como argumento

- **Construção da Árvore Filogenética**

Esta passo consiste na construção de uma árvore filogenética a partir das 11 sequências selecionadas pelo método Blast. Esta tarefa é relativamente complexa e requer a execução de vários passos intermédios. È completada com recurso à classe UPGMA (*Unweighted Pair Group Method Using Arithmetic Averages*) que se baseia num algoritmo heurístico de clustering hierárquico aglomerativo. Em primeiro lugar, com recurso à classe PairwiseAlignment é feito o alinhamento pareado de todos os possíveis pares de sequências, de acordo com o algoritmo de Needleman Wunsch. Para cada alinhamento é contabilizado o número de posições que diferem entre as duas sequências e é completada a matriz de distâncias. De seguida, com recurso à classe HierarchicalClustering as sequências são agrupadas hierarquicamente, utilizando como medida de semelhança a distância inferida anteriormente. Durante este processo, dá-se a construção da árvore com recurso à classe BinaryTree que será retornada no final. Nas Tabelas 4,5 e 6 estão descritos os métodos implementados em cada uma das classes mencionadas.

Tabela 4. Descrição dos métodos implementados na classe UPGMA.

UPGMA	
Método	**Descrição**
create_mat_dist()	Função que realiza a construção da matriz de distâncias entre um conjunto de sequências a partir do alinhamento de cada par
ru()	Método que chama a classe HierarchicalClustering e que realiza o agrupamento hierárquico das sequencias e consequente construção da árvore filogenética

Tabela 5. Descrição dos métodos implementados na classe HierarchicalClustering.

HierarchicalClustering	
Método	**Descrição**
execute_catering()	Agrupa as sequencias hierarquicamente com base na matriz de distâncias construida anteriormente. Para isso, é identificado em cada iteração os elementos (sequencias ou clister) mais próximos. Simultaneamente, dá-se a construção da árvore filogenética com recurso à chamada da classe BinaryTree.

Tabela 5. Descrição dos métodos implementados na classe BinaryTree.

BinaryTree	
Método	**Descrição**
get_cluster()	Retorna uma lista com os valores associados às folhas da árvore.
print_tree(seqsID=None)	Realiza a chamada da função recursiva print_tree_rec que será responsável pelo preatty printing da árvore
print_tree_rec(seqsID=None)	Realiza a impressão da árvore em profundidade para a consola. Recebe como argumento opcional os identificadores das folhas que serão impressos se fornecidos.
size()	Retorna o numero de nós e de folhas presentes na árvore.
exists_leaf(leafnum)	Verifica se determinada folha existe na árvore
common_ancestor(leaf1, leaf2)	Retorna a árvore correspondente ao antecessor comum mais próximo entre duas folhas
distance_leaves(leafnum1, leafnum2)	Retorna a distancia entre duas folhas.

- **Construção de um grafo**

Esta passo consiste na construção de um grafo que traduz as relações entre as 11 sequências selecionadas pelo método Blast. Cada aresta entre dois nós do grafo deve representar a conexão entre nós adjacentes, isto é antecessores e predecessores. A tarefa foi completada pela implementação da classe MyGraph. O objeto da classe é definido pelo atributo grafo que consiste num dicionário onde a cada nó (chave) corresponde os respetivos nós adjacentes (valores). Novamente, a construção do grafo requer uma chamada à classe UPGMA descrita no passo anterior. Posteriormente, o grafo é construido com base na matriz de distâncias resultante e num valor de corte que define o critério de ligação entre sequências. Na Tabela 6 encontram-se descritas os métodos implementados na classe MyGraph.

Tabela 6. Descrição dos métodos implementados na classe MyGraph.

MyGraph	
Método	**Descrição**
print_graph()	Imprime o conteúdo do grafo na forma de uma lista de adjacência.
get_nodes()	Retorna a lista de nós (chaves) do grafo.
get_edges()	Retorna a lista das arestas do grafo na forma de tuplos (origem, destino).
size()	Retorna o numero nós e arestas no grafo na forma de um tuplo.
add_vertex(v)	Adiciona um nó ao dicionário.
add_edge(o,d)	Adiciona uma aresta direcionada ao grafo com base nos nós de origem e destino.
get_successors(v)	Retorna a lista dos sucessores de determinado nó.
get_predcessors(v)	Retorna a lista dos antecessores de determinado nó.
get_adjacents(v)	Retorna uma lista dos nós adjacentes (antecessores e predecessores) de um nó.
out_degree(v)	Retorna o numero de sucessores de um nó.
in_degrees(v)	Retorna o numero de antecessores de um nó.
degree(v)	Retorna o numero de nós adjacentes (antecessores e predecessores) de um nó.

MyGraph	
all_degrees(seg_type=”inout”all_ degrees(deg_type=”inout”)	Retorna um dicionário com o número total de nós adjacentes para cada nó do grafo, de acordo com o tipo definido (in: antecessores; out: sucessores; inout: ambos).
highest_degrees(all_deg=None, deg_type=”inout”, top=10)	Retorna a lista do top-n nós com nós adjacentes
mean_degree(deg_type=”inout”)	Retorna a média do número total de nós adjacentes (degree) de todos os nós do grafo.
prob_degree(deg_type=”inout”)	Retorna a um dicionário com a frequência de cada degree presente no grafo.
reachable_bfs(v)	Retorna a lista de nós que podem ser alcançados por uma pesquisa em largura a partir do nó v
reachable_dfs(v)	Retorna a lista de nós que podem ser alcançados por uma pesquisa em profundidade a partir do nó v
distance(s,d)	Retorna a distância entre dois nós (s e d) pela contagem de elementos entre os dois.
shortest_path(s,d)	Retorna o caminho mais curto entre dois nós (s e d) na forma de uma lista das ligações necessárias.
clustering_coef(v)	Retorna um coeficiente que traduz a tendência de os nós do grafo criarem grupos locais fortemente conectados
all_clustering_coefs()	Retorna um dicionário com os coeficientes de cluster de cada nó.
mean_clustering_coef()	Retorna a média dos coeficientes de cluster de todos os nós do grafo.

Resultados

A aplicação das classes descritas no tópico anterior nas tarefas que constituem a pipeline está implementada no script run_me.py. Para isso, foram desenvolvidos alguns métodos complementares que se encontram descritos na Tabela 7.

Tabela 7. Descrição dos métodos implementados no modulo run_me.py, correspondentes à construção da pipelina.

Método	Descrição
Blast	
top_n_blast(blast_object, ref_seq, sp, n)	Função que recebe como argumento um objeto da classe MyBlast (contém como atributo uma livraria de sequências), uma sequencia de referencia e a respetiva espécie e encontra as n sequencias mais semelhantes. Para isso, é encontrada removida a sequência que apresentam o melhor alinhamento à sequência de referencia. Este procedimento é repetido até se obter o numero total de sequências desejadas.
n_similar_seqs(ref_seq, ref_sp, lib_seq, lib_sp, n)	Realiza o chamamento da função descrita a cima e retorna as listas das n sequências mais semelhantes a sequencia de referencia e das respetivas espécies.
Alinhamento Múltiplo	
m_align(seqs, seqsID)	Função que realiza o alinhamento múltiplo das sequências selecionadas.
print_m_align(m_align,max_len, seqsID)	Função que imprime o alinhamento para a consola de forma compreensível, isto é, cada sequência é impressa em fragmentos de tamanho máximo definido pelo utilizador (max_len). Recebe também o argumento seqsID que corresponde a uma lista dos identificadores das sequências, neste caso, a respetiva espécie.

Método	Descrição
Árvore Filogenética	
phylo_tree(seqs,seqID)	Função de constrói, imprime para a consola e retorna a árvore filogenética como um objeto BinaryTree. Recebe como argumentos a lista das sequências a considerar e a lista dos identificadores de cada folha (opcional), neste caso as espécies correspondentes a cada sequência. Faz a chamada às classes PairwiseAlignmente e UPGMA necessárias à construção da árvore, conforme descrito anteriormente.
tree2Newick(tree, seqsID=None)	Função recursiva que realiza a conversão da árvore filogenética para o formato newick, o mais comum para a representação de árvores. Recebe como argumentos o objeto BinaryTree correspondente à árvore e a lista de identificadores das folhas (opcional), neste caso as espécies das sequências.
export_tree(tree)	Realiza a exportação da árvore em formato newick para o ficheiro t*ree_viz.txt.*
Grafo	
create_graph(seqs, dist_cut)	Cosntroi o grafo não direcionado e não pesado com base na lista de sequências a considerar (seqs) e numa distância de corte que define o critério de ligação entre as sequências. Retorna um objeto do tipo MyGraph.
export_graph(graph, seqsID)	Realiza a exportação do grafo para o ficheiro *graph_viz.txt* no formato dot, o mais comum para a representação de grafos.

Nas Figuras 1 e 2 estão representadas as visualizações gráficas da árvore filogenética e grafo construidos com recurso aos softwares *Trex-online* (http://www.trex.uqam.ca/index.php?action=newick) e *Erdos- Online Graphviz Viewer* (http://sandbox.kidstrythisathome.com/erdos/) , respetivamente.

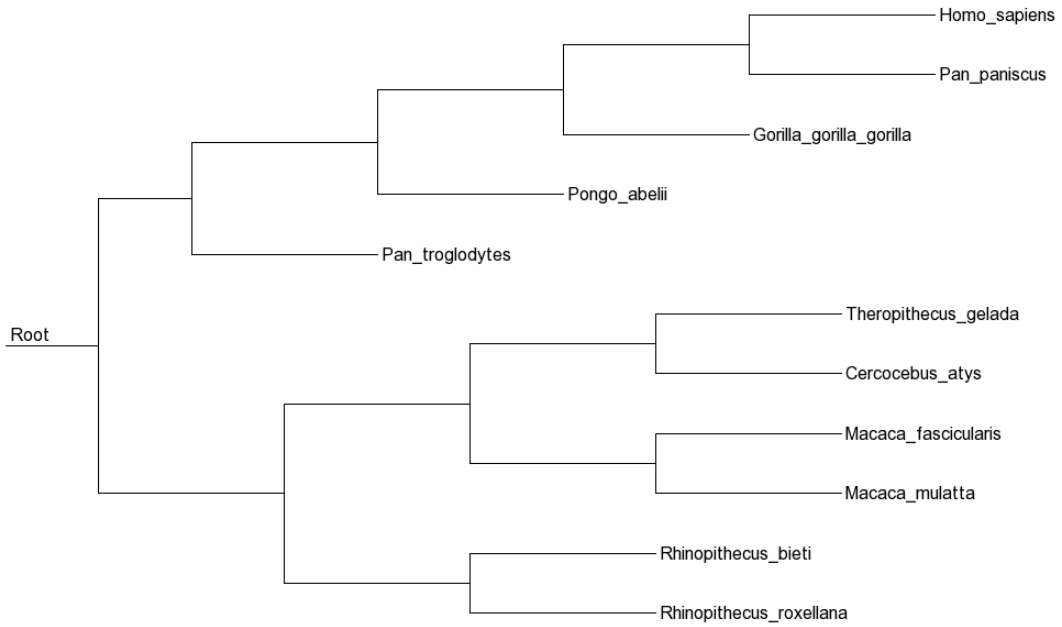

Figura 1. Representação gráfica da árvore filogenética

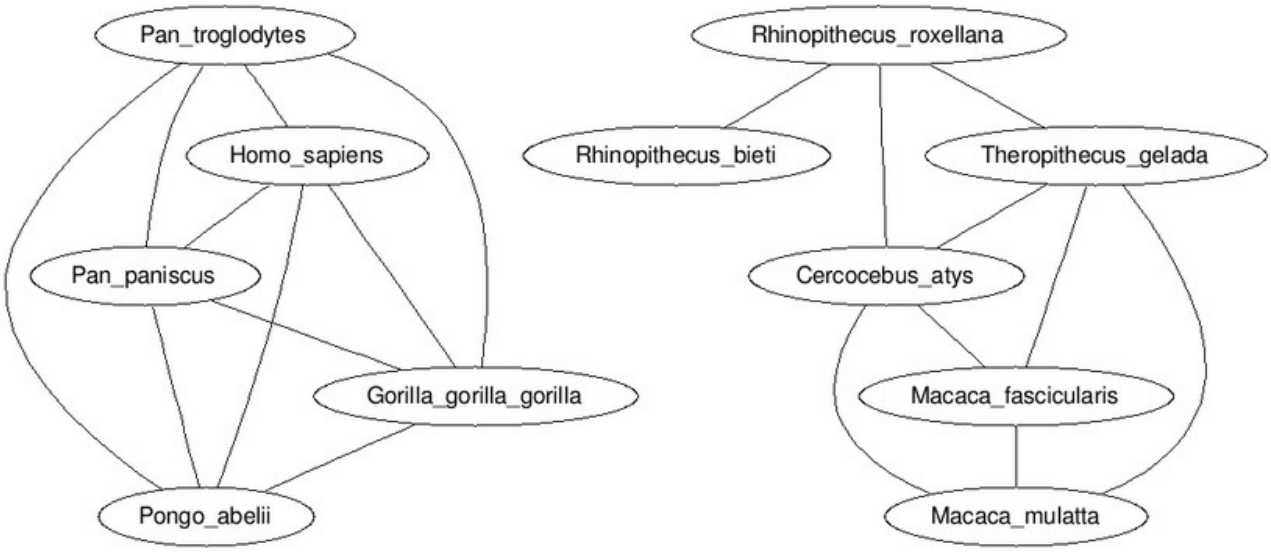

Figura 2. Representação gráfica do grafo.

Comentários e Conclusões

O trabalho solicitado foi executado na totalidade. De notar que, ao contrário de alguns trabalhos realizados anteriormente, não foi dado ao utilizador poder de decisão sobre as tarefas a realizar. Esta decisão prendeu-se ao facto de neste trabalho especifico ser solicitada a implementação de uma pipeline que, por definição, consiste na realização sequencial e autónoma de um conjunto de tarefas. De salientar ainda que nas tarefas de alinhamento múltiplo, construção da árvore filogenética e do grafo, os resultados foram exportados nos respetivos formatos mais adequados, fasta, newick e dot, respetivamente.