ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Hoàng Thị Điệp

PHƯƠNG PHÁP TÍNH TOÁN THÔNG MINH GIẢI MỘT SỐ BÀI TOÁN TIN SINH

Chuyên ngành: Khoa học Máy tính
Mã số: 62 48 01 01

TÓM TẮT LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2017

Công trình được hoàn thành tại: Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội

Người hướng dẫn khoa học:

1. PGS.TS. Lê Sỹ Vinh
2. PGS.TS. Hoàng Xuân Huấn

Phản biện:.....

.....

Phản biện:.....

.....

Phản biện:.....

.....

Luận án sẽ được bảo vệ trước Hội đồng cấp Đại học Quốc gia chấm luận án tiến sĩ họp tại .....

vào hồi giờ ngày tháng năm

Có thể tìm hiểu luận án tại:

- Thư viện Quốc gia Việt Nam
- Trung tâm Thông tin - Thư viện, Đại học Quốc gia Hà Nội

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

1. **Hoang, D. T.**, Le, S. V., Flouri, T., Stamatakis, A., von Haeseler, A. and Minh, B. Q. (2018) 'MPBoot: fast phylogenetic maximum parsimony tree inference and bootstrap approximation', *BMC Evolutionary Biology*, 18(1), p. 11. doi: 10.1186/s12862-018-1131-3. ***Tạp chí thuộc danh mục cơ sở dữ liệu của ISI.***
   (Tiêu đề của bản nộp đầu tiên: 'MPBoot: Novel and fast approximation for maximum parsimony bootstrap')
2. **Hoang, D. T.**, Chernomor, O., von Haeseler, A., Minh, B. Q. and Le, S. V. (2017) 'UFBoot2: Improving the ultrafast bootstrap approximation', *Molecular Biology and Evolution*. doi: 10.1093/molbev/msx281. ***Tạp chí thuộc danh mục cơ sở dữ liệu của ISI.***
3. **Hoang, D. T.**, Le, S. V., Flouri, T., Stamatakis, A., von Haeseler, A. and Minh, B. Q. (2016) 'A new phylogenetic tree sampling method for maximum parsimony bootstrapping and proof-of-concept implementation', in 2016 Eighth International Conference on Knowledge and Systems Engineering (KSE), pp. 1–6. doi: 10.1109/KSE.2016.7758020. ***Giải nhì bài báo xuất sắc cho học viên (Runner up Best Student Paper).***

MỞ ĐẦU

1. Tính cấp thiết của luận án

Tin sinh liên quan tới những công nghệ sử dụng máy tính để lưu trữ, truy xuất và phân tích các thông tin liên quan tới các đại phân tử sinh học: DNA, RNA và protein. Tin sinh học đang thu hút được nhiều quan tâm của cộng đồng khoa học và các nhà đầu tư do khả năng mang lại sự tiến bộ về khoa học và hiệu quả kinh tế thông qua việc thúc đẩy sự phát triển công nghệ sinh học và ứng dụng trong y tế, nông nghiệp và các lĩnh vực khác.

Nhiều khái niệm cơ bản trong tin sinh như sắp hàng chuỗi, tìm kiếm chuỗi tương đồng và xây dựng cây tiến hóa đều có liên hệ tới tiến hóa. Hơn nữa, những bước tiến công nghệ sinh học cho phép thu được trình tự của cả bộ gen hoàn chỉnh đã và đang cung cấp nguồn dữ liệu mới, vô cùng phong phú cho các nghiên cứu về tiến hóa ở mức độ toàn bộ gen. Điều này đặt ra đòi hỏi phát triển các phương pháp mới để có thể phân tích quan hệ tiến hóa trên các bộ dữ liệu nhiều chuỗi và dữ liệu ở mức bộ gen này.

Dưới góc nhìn thống kê, xây dựng cây tiến hóa là bài toán ước lượng tham số thống kê. Ta ước lượng cây tiến hóa từ mẫu gồm các phần tử là các vị trí trên sắp hàng. Do đó, để kết quả có ý nghĩa thì phải có độ tin cậy đi kèm. Làm bootstrap cây tiến hóa là kĩ thuật phổ biến để xác định độ tin cậy cây tiến hóa. Kĩ thuật này được phát triển từ kĩ thuật bootstrap của thống kê phục vụ việc xác định biến thiên của một ước lượng bằng thực nghiệm. Làm bootstrap cây tiến hóa thường được tiến hành ngay sau xây dựng cây tiến hóa với các bước sau: Với dữ liệu vào là sắp hàng gốc m vị trí, trước tiên, ta lấy mẫu có hoàn lại các vị trí trên sắp hàng gốc đúng m lần nhằm tạo một sắp hàng bootstrap có kích thước giống hệt sắp hàng gốc. Thông thường, 100 hoặc 1000 sắp hàng bootstrap được tạo ra theo cách này. Ta xây dựng cây tiến hóa cho mỗi sắp hàng bootstrap, gọi là cây bootstrap, theo cùng cách đã xây dựng cây tiến hóa cho sắp hàng gốc. Với mỗi cạnh của cây xây dựng trên sắp hàng gốc, giá trị hỗ trợ bootstrap của nó được tính bằng tỷ lệ cây bootstrap có chứa cạnh đó.

Làm bootstrap cây tiến hóa theo lược đồ chuẩn nói trên có một số hạn chế lớn là:

- Bootstrap chuẩn tiêu tốn rất nhiều thời gian. Vấn đề trở nên nghiêm trọng đặc biệt với sự ra đời của các công nghệ giải trình tự thế hệ tiếp theo cho phép tạo ra những bộ dữ liệu khổng lồ.

- Một vấn đề khác là độ chính xác bootstrap: giá trị hỗ trợ bootstrap mà tiếp cận bootstrap chuẩn gán cho từng cạnh của cây cho ước lượng thấp hơn xác suất đúng (tức xác suất thuộc về cây đúng) của cạnh.
- Riêng với các phương pháp xây dựng cây theo tiêu chuẩn hợp lý nhất (maximum likelihood - ML), còn có các vấn đề về ảnh hưởng của vi phạm giả thiết mô hình (vi phạm mô hình) và hiện tượng đa phân tới độ chính xác bootstrap.

2. Mục tiêu của luận án

1) Nghiên cứu phương pháp chuẩn và các phương pháp xấp xỉ đã có cho bootstrap cây tiến hóa ML, từ đó đề xuất phương pháp giải quyết tốt hơn từng thách thức của bài toán: thời gian chạy, độ chính xác, ảnh hưởng của vi phạm mô hình và hiện tượng đa phân, mở rộng cho dữ liệu bộ gen.
2) Nghiên cứu phương pháp chuẩn cho bootstrap cây tiến hóa theo tiêu chuẩn tiết kiệm nhất (maximum parsimony - MP), từ đó đề xuất phương pháp xấp xỉ mới hiệu quả cho bài toán.
3) Cài đặt các phương pháp đề xuất cho bootstrap cây tiến hóa ML và bootstrap cây tiến hóa MP để phục vụ nhu cầu của các nhà khoa học phân tích cây tiến hóa.

3. Các đóng góp của luận án

Trong luận án này, với bài toán bootstrap cây tiến hóa ML, chúng tôi đề xuất phương pháp UFBoot2 dựa trên phương pháp UFBoot (Minh et al. 2013) với 4 cải tiến quan trọng. UFBoot2 cải thiện đáng kể tốc độ và độ chính xác của giá trị bootstrap so với UFBoot. Hơn nữa, UFBoot2 có các cải tiến để xử lý đỉnh đa phân tốt hơn, giảm ảnh hưởng của vi phạm mô hình và mở rộng để phân tích sắp hàng các bộ gen. Với bài toán bootstrap cây tiến hóa MP, luận án đề xuất phương pháp mới MPBoot để xấp xỉ nhanh bootstrap chuẩn. MPBoot được phát triển từ ý tưởng của UFBoot với các điều chỉnh quan trọng để phù hợp với tiêu chuẩn tiết kiệm nhất: tính toán hiệu quả điểm MP cho cây trên sắp hàng bootstrap, các kỹ thuật phức tạp hơn cho tìm kiếm cây như cắt và ghép cây con (SPR), ratchet và bước tinh chỉnh các cây bootstrap ứng viên. Chúng tôi đã kết hợp với Trung tâm Tin sinh Tích hợp Vienna, Cộng hòa Áo phát triển và tích hợp UFBoot2 vào hệ thống mã nguồn mở tốt nhất hiện nay cho phân tích cây tiến hóa theo tiêu chuẩn hợp lý nhất IQ-TREE; và phát triển phần mềm mã nguồn mở MPBoot cài đặt phương pháp MPBoot.

Các kết quả của luận án đã được công bố trong 2 bài báo ở tạp chí SCI quốc tế và 1 báo cáo ở hội nghị quốc tế.

4. Bố cục của luận án

Ngoài phần kết luận, luận án được tổ chức như sau.

Chương 1 giới thiệu các khái niệm cơ bản trong phân tích cây tiến hóa dựa trên dữ liệu sinh học phân tử, phương pháp bootstrap trong thống kê và phát biểu bài toán làm bootstrap cây tiến hóa.

Chương 2 đề xuất thuật toán pruning nhanh trong trường hợp mô hình tiến hóa có tính thuận nghịch thời gian; sau đó đề xuất phương pháp UFBoot2 để xấp xỉ nhanh bootstrap cây tiến hóa ML. Cuối chương trình bày thiết kế thực nghiệm và kết quả.

Chương 3 đề cập tới các vấn đề về dữ liệu và mô hình mà UFBoot không hỗ trợ. Từ đó, đề xuất ba cải tiến quan trọng: (i) cải tiến để xử lý các đỉnh đa phân (ii) cải tiến để giảm ảnh hưởng của vi phạm mô hình và (iii) cải tiến mở rộng để phân tích sắp hàng nhiều gen. Cuối chương trình bày thiết kế thực nghiệm và kết quả.

Chương 4 đề xuất một phương pháp mới (MPBoot) để tìm kiếm hiệu quả cây MP, đồng thời xấp xỉ hiệu quả bootstrap cây tiến hóa MP. Cuối chương trình bày thiết kế thực nghiệm và kết quả.

Chương 1 BÀI TOÁN BOOTSTRAP CÂY TIẾN HÓA

1.1 Một số khái niệm cơ bản trong tiến hóa phân tử

1.1.1 Thông tin di truyền

Kiểu hình của sinh vật sống luôn là kết quả của thông tin di truyền (mà sinh vật mang và truyền cho thế hệ kế tiếp) và tương tác với môi trường. Vì vậy, để nghiên cứu về tiến hóa, ta cần tìm hiểu về biến đổi trong thông tin di truyền của sinh vật. Phần này trình bày một số khái niệm cơ bản trong sinh học phân tử (bộ gen, DNA, RNA, protein) và quan hệ giữa chúng.

1.1.2 Sắp hàng đa chuỗi

Sắp hàng đa chuỗi có thể được hiểu như một ma trận các phân tử sinh học. Trong đó mỗi hàng chính là một chuỗi phân tử sinh học còn mỗi cột chứa các phân tử sinh học tương đồng của các chuỗi. Mỗi cột được gọi là một vị trí. Từ sắp hàng đa chuỗi, các chuỗi tương đồng có thể sử dụng để phân tích cây tiến hóa giúp đánh giá nguồn gốc tiến hóa của các chuỗi.

1.1.3 Cây tiến hóa

Cây tiến hóa là một dạng sơ đồ phân nhánh thể hiện mối quan hệ tiến hóa giữa các loài (cũng có thể mở rộng cho các cá thể hay các gen) dựa trên sự tương đồng và khác biệt về đặc điểm di truyền. Các loài đưa vào phân tích quan hệ tiến hóa được cho là có một tổ tiên chung.

Mối quan hệ tiến hóa giữa các loài thường được biểu diễn bởi một cây nhị phân với cấu trúc như sau: mỗi đỉnh lá của cây biểu diễn một loài hiện tại (cho bởi dữ liệu vào); mỗi đỉnh trong của cây biểu diễn một loài tổ tiên

(ta không có thông tin về các loài tổ tiên); một cạnh của cây nối hai đỉnh của cây và biểu diễn mối quan hệ trực tiếp giữa hai loài ở hai đỉnh của cây; độ dài của cạnh nối hai loài trên cây cho biết khoảng cách tiến hóa giữa chúng. Khoảng cách tiến hóa này có thể được biểu diễn bằng thời gian, hay số lượng các biến đổi nucleotide/axít amin giữa hai chuỗi DNA/axít amin được sử dụng để so sánh hai loài.

Ta thường không có thông tin về các loài tổ tiên, cho nên không xác định được chính xác gốc của cây tiến hóa. Chính vì vậy, cây tiến hóa thường được biểu diễn bằng một cây nhị phân không gốc. Sau đó, khi có thêm thông tin tiến hóa, ta sẽ dùng một kĩ thuật định gốc, ví dụ kĩ thuật định gốc bằng một loài khác xa với nhóm loài đang xét (outgroup rooting).

1.1.4 Xây dựng cây tiến hóa

Cây tiến hóa không quan sát trực tiếp được mà phải suy luận từ chuỗi hoặc các dữ liệu khác. Các phương pháp xây dựng cây tiến hóa được xếp vào hai hướng tiếp cận: dựa trên khoảng cách (các phương pháp khoảng cách) hoặc dựa trên ký tự (các phương pháp tiệm nhất –MP , các phương pháp hợp lý nhất - ML, các phương pháp suy luận Bayesian). Luận án sẽ tập trung vào xây dựng cây tiến hóa theo MP và ML do độ phổ dụng cao của chúng.

1.1.4.1 Tiêu chuẩn tiết kiệm nhất (maximum parsimony – MP)

Phương pháp MP tính số lượng biến đổi cực tiểu trên một cây tiến hóa bằng cách gán các trạng thái ký tự cho các đỉnh trong của cây. Chiều dài ký tự (hoặc vị trí) là số lượng biến đổi tối thiểu cần thiết cho vị trí đó. Điểm MP của cây là tổng các chiều dài ký tự tính trên tất cả các vị trí. Cây MP là cây có điểm MP nhỏ nhất.

1.1.4.2 Tiêu chuẩn hợp lý nhất (maximum likelihood – ML)

Để ước lượng cây tiến hóa theo ML cần hai bước tối ưu: tối ưu độ dài các cạnh để tính điểm cây cho mỗi cây ứng viên và duyệt tìm kiếm trong không gian cây để thu được cây làm cực đại hàm likelihood. Suy luận cây theo ML tương đương với việc so sánh nhiều giả thuyết thống kê có cùng số lượng tham số.

1.1.4.3 Một số kĩ thuật biến đổi cục bộ trên cây dùng trong xây dựng cây tiến hóa

Hoán-đổi-cạnh (branch-swapping) là một lớp các kĩ thuật dùng để cải thiện lời giải trong các thuật toán xây dựng cây tiến hóa có liên quan tới việc xáo trộn cấu trúc cây. Ba kĩ thuật hoán-đổi-cạnh từ đơn giản nhất đến phức tạp nhất là: (i) hoán đổi hàng xóm gần nhất (nearest-neighbor interchange - NNI), (ii) cắt và ghép cây con (subtree pruning and regrafting - SPR) và (iii) chặt đôi và nối lại (tree bisection and reconnection - TBR).

1.1.5 Mô hình tiến hóa

Tất cả các phương pháp phân tích quan hệ tiến hóa dựa trên mô hình (trong đó có các phương pháp ML) đều giả thiết rằng biến đổi tiến hóa của một kí tự (nucleotide hay axit amin) tuân theo xích Markov thời gian liên tục với tập trạng thái $\mathcal{E}$ chính là tập các trạng thái kí tự. Phần này sẽ trình bày các công thức với mô hình biến đổi nucleotide ($\mathcal{E} = \{A, C, G, T\}$); công thức cho mô hình biến đổi axit amin ($\mathcal{E}$ là tập các axit amin) hoạt động tương tự.

1.1.5.1 Ma trận tốc độ biến đổi tức thì

Tâm điểm của xích Markov thời gian liên tục là một ma trận $\mathbf{Q}$ (trừ những ô nằm trên đường chéo) của tốc độ biến đổi tức thì từ một trạng thái kí tự sang một trạng thái kí tự khác. Các ô đường chéo được tính theo các ô cùng hàng để đảm bảo tổng hàng bằng 0. Trong Hình 1.4, các hàng sắp theo thứ tự A,C,G,T. $\pi_A, \pi_C, \pi_G, \pi_T$ là tần suất của các nucleotide. Đại lượng μ là trung bình tốc độ biến đổi tức thì. Trong tính toán, $\mathbf{Q}$ được chuẩn hóa để $\mu = 1$. Khi ấy, độ dài cạnh của cây tiến hóa là trung bình số lượng biến đổi nucleotide trên vị trí sắp hàng.

$$\mathbf{Q} = (q_{ij}) = \begin{array}{c} \begin{matrix} \mathbf{A} & \mathbf{C} & \mathbf{G} & \mathbf{T} \end{matrix} \\ \begin{pmatrix} -(a\pi_C + b\pi_G + c\pi_T) & a\pi_C & b\pi_G & c\pi_T \\ g\pi_A & -(g\pi_A + d\pi_G + e\pi_T) & d\pi_G & e\pi_T \\ h\pi_A & i\pi_C & -(h\pi_A + i\pi_C + f\pi_T) & f\pi_T \\ j\pi_A & k\pi_C & l\pi_G & -(j\pi_A + k\pi_C + l\pi_G) \end{pmatrix} \end{array} \mu$$

Hình 1.1. Ma trận tốc độ biến đổi tức thì Q cho mô hình biến đổi nucleotide.

Trong Chương 2, luận án sẽ khảo sát một lớp cụ thể hơn của mô hình biến đổi nucleotide gọi là các mô hình có tính thuận nghịch thời gian (time-reversible), tức $a = g, b = h, c = j, d = i, e = k, f = l$.

Khi biết $\mathbf{Q}$ ta có thể tính xác suất chuyển từ trạng thái kí tự này sang trạng thái kí tự khác trong thời gian tiến hóa t bằng cách tính hàm mũ ma trận:

$$\mathbf{P}(t) = e^{\mathbf{Q}t} \qquad \textbf{(1.1)}$$

Ma trận xác suất chuyển trạng thái $\mathbf{P}(t)$ chính là chìa khóa để tính likelihood trong các phương pháp xây dựng cây tiến hóa theo ML.

1.1.5.2 Một số mô hình biến đổi nucleotide

Bảng 1.3 liệt kê các mô hình biến đổi nucleotide điển hình thuộc lớp thuận nghịch thời gian. Ta nhận thấy GTR là mô hình tổng quát nhất với 8 tham số tự do. JC69 là mô hình đơn giản nhất, nó không chứa một tham số tự do nào.

1.1.5.3 Tính không đồng nhất của tốc độ biến đổi giữa các vị trí trên trình tự

Thực tế, tốc độ biến đổi nucleotide có thể khác biệt đáng kể ở các vị trí khác nhau trên trình tự. Để tính tới tình huống này, người ta đưa vào một

mô hình hợp lý cho phân bố của tốc độ theo vị trí. Tiếp cận phổ biến nhất là dùng phân bố gamma (Γ) với kì vọng bằng 1.0 và phương sai bằng $1/\alpha$, từ đó có các mô hình $JC69 + \Gamma$, $HKY85 + \Gamma$ hay $GTR + \Gamma$ có nhiều hơn 1 tham số tự do so với mô hình gốc (tham số α).

1.1.6 Giới thiệu phương pháp bootstrap trong thống kê

Bootstrap là kĩ thuật trong thống kê để ước lượng theo cách thực nghiệm mức độ biến thiên của một ước lượng. Nó lấy mẫu có hoàn lại từ mẫu ban đầu để tạo ra một mẫu hư cấu có cùng kích thước.

Giả sử mẫu gốc có m điểm dữ liệu $(x_1, x_2, \dots, x_m)$ được sinh độc lập từ phân bố $F(\theta)$ phụ thuộc vào tham số θ. Từ mẫu gốc ta tính được ước lượng $\hat{\theta} = t(x_1, x_2, \dots, x_m)$ cho tham số θ. Ta muốn biết mức độ biến thiên của phân bố của ước lượng này. Việc này trước khi phương pháp bootstrap ra đời là không thể thực hiện được nếu phân bố F chưa biết hoặc hàm ước lượng $t(\mathbf{x})$ phức tạp về mặt toán học. Bootstrap suy luận ra biến thiên này bằng cách sử dụng mẫu gốc, thông qua việc sinh các mẫu mới không phải từ F mà từ phân bố thực nghiệm $\hat{F}$ cho các điểm dữ liệu trên mẫu gốc. Sinh một mẫu cùng cỡ m từ phân bố thực nghiệm cũng giống như lấy một mẫu các điểm $(x_1^*, x_2^*, \dots, x_m^*)$ từ chính mẫu gốc. Mẫu này được gọi là bản sao bootstrap $\mathbf{x}^*$. Từ bản sao ta cũng tính được ước lượng cho tham số $\widehat{\boldsymbol{\theta}^*} = t(\mathbf{x}^*)$. Để thấy mức độ biến thiên của các ước lượng cho θ, ta chỉ cần sinh thật nhiều bản sao bootstrap và làm ước lượng trên đó. Các nghiên cứu đã chỉ ra rằng, m lớn và làm bootstrap với số lượng lớn bản sao sẽ cho biến thiên chính xác của $\hat{\theta}$.

1.2 Bài toán bootstrap cây tiến hóa

1.2.1 Phát biểu bài toán

Dữ liệu vào: Dữ liệu đầu vào là một sắp hàng của n chuỗi của các phân tử sinh học (chuỗi nucleotide/axít amin/codon), mỗi chuỗi có m vị trí (ký tự), sau đây gọi là sắp hàng gốc. Với phân tích ML, dữ liệu vào có thêm mô hình tiến hóa.

Bài toán: Lặp lại B lần việc sinh sắp hàng bootstrap có kích thước giống hệt sắp hàng mẫu bằng cách lấy mẫu có hoàn lại các vị trí của sắp hàng gốc ta thu được B sắp hàng bootstrap. Xây dựng cây tiến hóa cho sắp hàng gốc và cho từng sắp hàng bootstrap. Do có nhiều phương pháp khác nhau cho kết quả với độ chính xác cũng như thời gian thực hiện khác nhau, chúng ta cần đề xuất các phương pháp cho kết quả chính xác với thời gian thực hiện thấp nhất có thể.

Dữ liệu ra: Cây tiến hóa cho sắp hàng gốc có gắn giá trị hỗ trợ bootstrap cho mỗi cạnh.

Minh họa lược đồ của tiếp cận bootstrap chuẩn cây tiến hóa (standard bootstrap hay SBS) được minh họa trong Hình 1.6.

1.2.2 Các tiêu chí đánh giá một phương pháp bootstrap cây tiến hóa

1.2.2.1 Đánh giá thời gian chạy

1.2.2.2 Đánh giá độ chính xác

Độ chính xác của một phương pháp, Z, được định nghĩa bởi $f_Z(x)$, là tỷ lệ của số cạnh có mặt trong cây đúng trong số tất cả các cạnh có giá trị hỗ trợ bootstrap $x\%$ (đếm trên tất cả các cây xây dựng được). $f_Z(x)$ phản ánh xác suất một cạnh với giá trị hỗ trợ bootstrap $x\%$ là một cạnh đúng. Phương pháp Z được gọi là không chệch nếu $f_Z(x) = x\%$ cho tất cả các giá trị của x (trên đồ thị đây chính là đường chéo). Nếu đường cong của $f_Z(x)$ nằm phía trên đường chéo, thì phương pháp bootstrap cho ước lượng thấp hơn xác suất đúng của cạnh (nghĩa là phương pháp này bảo thủ). Ngược lại (đường cong cho $f_Z(x)$ nằm bên dưới đường chéo), phương pháp cho ước lượng cao hơn xác suất đúng của cạnh (nghĩa là phương pháp này lạc quan).

Việc khảo sát độ chính xác của phương pháp bootstrap Z có thể được thực hiện thông qua thực nghiệm mô phỏng.

1.2.2.3 Các tiêu chí khác

a) Đánh giá ảnh hưởng của vi phạm giả thiết mô hình tiến hóa

Ta giả sử dữ liệu sắp hàng đã được sinh từ một cây tiến hóa đúng $T^{đúng}$ theo một mô hình tiến hóa đúng $M^{đúng}$. Khi xây dựng cây tiến hóa theo tiêu chuẩn hợp lý nhất, một trong những bước đầu tiên là xác định mô hình tiến hóa M. Khi mô hình M được chọn là khác $M^{đúng}$, ta gọi nó là mô hình vi phạm giả thiết. Việc khảo sát ảnh hưởng của mô hình tiến hóa sai có thể được thực hiện thông qua thực nghiệm mô phỏng.

b) Đánh giá ảnh hưởng của hiện tượng đa phân (polytomy)

Các phương pháp xây dựng cây tiến hóa chỉ duyệt các cây có cấu trúc nhị phân (nghĩa là mỗi đỉnh trong luôn kết nối với ba cạnh khác). Đôi khi, dữ liệu đầu vào không cho ta đủ thông tin tiến hóa, dẫn đến việc biểu diễn tốt nhất lại là một cây có chứa đỉnh đa phân. Những đỉnh này gọi là polytomy. Các phương pháp khác nhau cho kết quả khác nhau khi tìm dạng nhị phân cho đỉnh đa phân này. Việc khảo sát ảnh hưởng của hiện tượng đa phân có thể được thực hiện thông qua thực nghiệm mô phỏng.

1.3 Các nghiên cứu bootstrap cây tiến hóa

Tốn nhiều thời gian chạy là nút thắt lớn nhất của phân tích bootstrap cây tiến hóa. Vấn đề trở nên nghiêm trọng hơn với phân tích dựa trên ML bởi tính toán likehood tốn kém và bởi xây dựng cây tiến hóa theo ML thuộc lớp bài toán NP-khó. Luận án không tìm hiểu được công trình nào nghiên cứu xấp xỉ bootstrap cây tiến hóa theo MP nhưng lại có khá nhiều công

trình nghiên cứu xấp xỉ bootstrap cây tiến hóa theo ML). Trong đó, UFBoot (Minh et al. 2013) là phương pháp mới và vượt trội về tốc độ so với các phương pháp khác. Ngoài ưu điểm về thời gian chạy, UFBoot cho ước lượng sát với xác suất đúng của cạnh khi mô hình đúng hoặc vi phạm ít. Hơn nữa, UFBoot cài đặt trong hệ thống IQ-TREE có mã nguồn mở miễn phí để các nhà nghiên cứu quan tâm có thể tham gia cùng phát triển.

Chương 2 PHƯƠNG PHÁP UFBOOT2 CHO BÀI TOÁN BOOTSTRAP CÂY TIẾN HÓA THEO TIÊU CHUẨN HỢP LÝ NHẤT

2.1 Bootstrap cây tiến hóa theo tiêu chuẩn hợp lý nhất

Trong bootstrap cây tiến hóa theo tiêu chuẩn hợp lý nhất, việc duyệt tìm cây tốt nhất cho sắp hàng gốc và việc duyệt tìm tập cây bootstrap (hay tập hợp các cây tốt nhất cho từng sắp hàng bootstrap) đều sử dụng tiêu chuẩn hợp lý nhất.

Xây dựng cây tiến hóa theo tiêu chuẩn hợp lý nhất

Gọi $A = (X_1, \dots, X_n)$ là một dóng hàng gồm n chuỗi với độ dài m; M là mô hình tiến hóa. Chúng ta cần xây dựng một cây tiến hóa T sao cho T có thể giải thích một cách hợp lý nhất quá trình biến đổi thành các chuỗi trong A theo cây T và mô hình biến đổi M. Giá trị hợp lý $L(T|A, M)$ của cây T để giải thích sắp hàng A với mô hình tiến hóa M được tính theo công thức sau:

$$L(T|A, M) = P(A|T, M)$$

trong đó $P(A|T, M)$ là xác suất để quan sát được dữ liệu A với điều kiện cây T và mô hình biến đổi M là đúng.

Để không bị mất mát thông tin khi tính toán trên các số bé, thay vì tính likelihood, người ta thường tính log-likelihood của cây. Kí hiệu log-likelihood của cây tính trên cả sắp hàng là $\ell(T|A)$, tính trên vị trí j của sắp hàng là $\ell(T|A_j)$. Ta có:

$$\ell(T|A) = \sum_{j=1}^{m} \ell(T|A_j) \qquad \textbf{(2.1)}$$

Xây dựng cây tiến hóa theo tiêu chuẩn hợp lý cực đại (ML) là tìm cây T^* để $\ell(T^*|A)$ đạt giá trị lớn nhất. Duyệt tất cả các cây tiến hóa (cấu trúc phân nhánh + độ dài các cạnh) để tìm ra cây tốt nhất theo tiêu chuẩn cực đại hợp lý một bài toán khó và phức tạp, nó thuộc lớp các bài toán NP-khó.

2.2 Xấp xỉ nhanh bootstrap cây tiến hóa theo tiêu chuẩn ML: Thuật toán UFBoot

2.2.1 Tóm tắt ý tưởng

Ý tưởng chính của UFBoot (Minh et al. 2013) (**Thuật toán 2.1**) là giữ lại các cây duyệt khi thực hiện tìm kiếm trên sắp hàng gốc và dùng chúng để tính likelihood cho các sắp hàng bootstrap. Để tăng tốc độ tính toán likelihood hơn nữa đối với các sắp hàng bootstrap, UFBoot sử dụng chiến lược resampling estimated log-likelihood (RELL). Đối với mỗi sắp hàng bootstrap cây có điểm RELL cao nhất (RELL-tree) biểu thị cây ML-bootstrap. Trái ngược với SBS, UFBoot không tối ưu cây này. Sự khác biệt giữa UFBoot và SBS trong giá trị hỗ trợ bootstrap gán cho các cạnh là do UFBoot và SBS chọn cây bootstrap theo cách khác nhau.

2.2.2 Công thức RELL

Kí hiệu A^{data} là sắp hàng gốc của n chuỗi và m vị trí; các vị trí này được nhóm thành các mẫu-vị trí $D_1, D_2, \dots, D_k$ với tần suất tương ứng là $d_1, d_2, \dots, d_k$. Điểm log-likelihood cho cấu trúc cây T khi biết A^{data} được tính bởi công thức:

$$\ell(T|A^{data}) = \sum_{i=1}^{k} \ell(T|D_i) \times d_i \qquad \textbf{(2.2)}$$

trong đó $\ell(T|D_i)$ là điểm log-likelihood cho cây T tại mẫu-vị trí D_i, được tính toán hiệu quả với thuật toán pruning (Felsenstein 1973; Felsenstein 1981). Thuật toán này sẽ được trình bày kĩ ở phần 2.3.

Với một cây T đã được tính điểm log-likelihood tại các mẫu-vị trí của sắp hàng gốc, chiến lược RELL cho phép ta tính xấp xỉ điểm log-likelihood của T trên sắp hàng bootstrap $A^{bootstrap}$ cực nhanh chỉ với phép tính tổng:

$$\ell(T|A^{bootstrap}) \approx \hat{\ell}(T|A^{bootstrap}) = \sum_{i=1}^{k} \ell(T|D_i) \times d_i^{bootstrap} \qquad \textbf{(2.3)}$$

Trong đó $d_i^{bootstrap}$ là tần suất của D_i trong $A^{bootstrap}$. Nhờ vậy, ta tránh được việc gọi tới thuật toán pruning khi tính log-likelihood cho T tại mỗi mẫu-vị trí của $A^{bootstrap}$.

2.2.3 Giả mã của thuật toán UFBoot

Thuật toán 2.1. Thuật toán UFBoot

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa). Mô hình tiến hóa M cho phép tính các xác suất biến đổi trạng thái trên cây.

Dữ liệu ra: Cây ML xây dựng cho A^{data} được gắn các giá trị hỗ trợ bootstrap cho mỗi cạnh.

Bắt đầu

1) Bước khởi đầu: Tạo B sắp hàng bootstrap, $A_1, A_2, \dots, A_B$. Với mỗi sắp hàng bootstrap A_b khởi tạo cây bootstrap $T_b := null$ và điểm RELL bootstrap $\hat{\ell}(T_b|A_b) := -\infty$. Khởi tạo một tập cây $S := \{\}$ và ngưỡng log-likelihood $\ell_{min} := -\infty$.
2) Bước khám phá: Tiến hành tìm kiếm cây sử dụng thuật toán IQPNNI (Vinh and von Haeseler 2004) cho sắp hàng gốc A^{data}. Mỗi khi duyệt một cây mới $T \notin S$ thỏa mãn $\ell(T|A^{data}) \geq \ell_{min}$, thêm T vào S và tính $\hat{\ell}(T|A_b)$, cho các $b = 1, \dots, B$ dựa trên **Công thức 2.4**. Nếu $\hat{\ell}(T|A_b) > \hat{\ell}(T_b|A_b)$, cập nhật $T_b := T$.

   Khi hết mỗi lượt lặp tìm kiếm của thuật toán IQPNNI, cập nhật ℓ_{min}.
3) Bước tóm tắt: Xây dựng một cây đồng thuận từ các cây bootstrap $\{T_1, T_2, \dots, T_B\}$, hoặc ánh xạ giá trị hỗ trợ bootstrap lên cây ML mà IQPNNI tìm được cho A^{data}.

Kết thúc

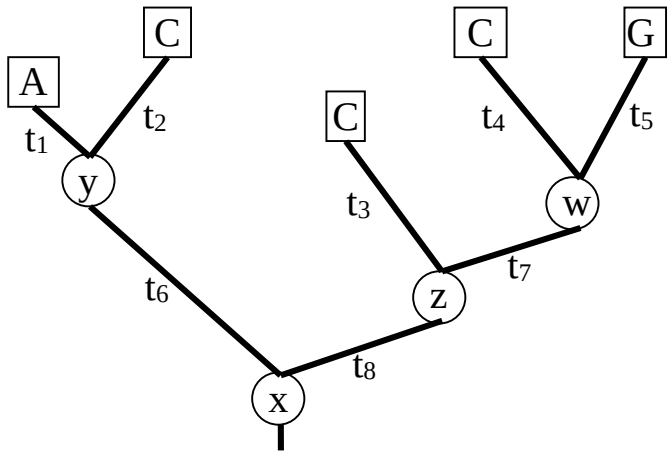
2.3 Tính độ hợp lý (likelihood) cho một cây: Thuật toán pruning

2.3.1 Tính likelihood cho một cây theo định nghĩa

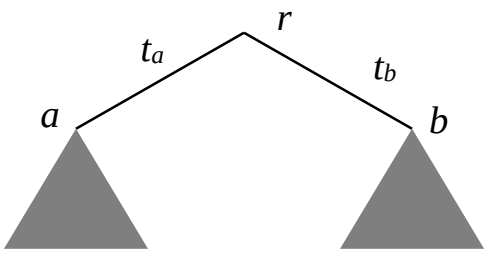
Ta sẽ khảo sát việc tính likelihood cho một cây đã biết độ dài cạnh. Ví dụ minh họa việc tính toán cho các chuỗi DNA nhưng có thể tổng quát hóa cho tất cả các mô hình kí tự rời rạc. Ta được cho một cây biết độ dài cạnh (Hình 2.1) và một mô hình tiến hóa cho phép tính các xác suất biến đổi trạng thái trên cây này, tức tính $P_{ij}(t)$ (xác suất trạng thái j sẽ tồn tại ở điểm kết thúc của một cạnh độ dài t, nếu trạng thái ở điểm bắt đầu của cạnh là i). Tính toán sử dụng 2 giả thiết độc lập. Theo đó, việc tính likelihood trên một sắp hàng quy về việc tính likelihood cho cây tại một vị trí đơn lẻ.

$$P(A_i|T) = \sum_x \sum_y \sum_z \sum_w P(A, C, C, C, G, x, y, z, t, w|T) \qquad \textbf{(2.4)}$$

mỗi tổng chạy trên tất cả bốn nucleotide.



Hình 2.1. Cây minh họa tính likelihood bằng định nghĩa chuẩn và bằng thuật toán pruning. Đỉnh gốc là x.



Hình 2.2. Một cây tiến hóa T để minh họa thuật toán pruning và pruning nhanh. Nó được định gốc ngẫu nhiên tại một đỉnh trong r với hai đỉnh con trực tiếp là a và b với các độ dài cạnh tương ứng là t_a và t_b.

Xác suất ở vế phải của **Công thức 2.5** có thể phân rã thành một tích của các nhân tử:

$$P(A,C,C,C,G,x,y,z,t,w|T) = P(x)P(y|x,t_6)P(A|y,t_1)P(C|y,t_2) \quad \textbf{(2.5)}$$
$$P(z|x,t_8)P(C|z,t_3)P(w|z,t_7)\,P(C|w,t_4)P(G|w,t_5)$$

$P(x)$ có thể đặt bằng xác suất cân bằng của bazơ x theo mô hình đó. Những xác suất khác được tính từ mô hình biến đổi nucleotide. Biến đổi ở mỗi cạnh độc lập với tất cả các cạnh khác nếu bazơ ở điểm bắt đầu cạnh ấy đã xác định.

Công thức 2.6 có rất nhiều hạng tử. Với mỗi vị trí sắp hàng, ta tính tổng của $4^4 = 256$ hạng tử. Số hạng tử tăng theo hàm mũ so với n (số loài). Trên một cây (có gốc) có n loài, ta có $n-1$ đỉnh trong, mỗi đỉnh trong có thể nhận một trong 4 trạng thái. Do đó ta cần 4^{n-1} hạng tử.

2.3.2 Tính likelihood cho một cây theo thuật toán pruning

Thuật toán pruning tính toán hiệu quả likelihood của cây nhờ sử dụng tiếp cận quy hoạch động. Ý tưởng là đẩy các kí hiệu tổng trong **Công thức 2.5** đi càng xa càng tốt và nhóm nó trong cặp ngoặc khi có thể.

$$P(A_i|T) = \sum_x P(x)\left(\sum_y P(y|x,t_6)P(A|y,t_1)P(C|y,t_2)\right) \times\left(\sum_z P(z|x,t_8)P(C|z,t_3)\right) \times\left(\sum_w P(w|z,t_7)P(C|w,t_4)P(G|w,t_5)\right)\right) \quad \textbf{(2.6)}$$

Việc tính toán theo **Công thức 2.8** diễn ra từ cặp ngoặc sâu nhất ra ngoài. Điều này gợi ý luồng tính toán trong cây là hướng xuống gốc.

Thuật toán pruning dựa trên hàm likelihood riêng phần của một cây con, kí hiệu là $L_i^r(s)$. Đây là xác suất của mọi thứ quan sát được từ nút r trở đến các lá, tại vị trí sắp hàng thứ i, với điều kiện là nút r có trạng thái s. Trong **Công thức 2.8**, hạng tử

$$P(C|w,t_4)P(G|w,t_5)$$

là một trong những đại lượng này. Có 4 đại lượng như vậy ứng với các giá trị khác nhau tại đỉnh w. Điểm mấu chốt của thuật toán pruning là, một khi 4 con số này đã được tính thì ta không cần liên tục tính lại chúng.

Thuật toán pruning tiến hành nhờ tính toán đệ quy hàm $L_i^r(s)$ tại mỗi đỉnh trên cây theo chính hàm này này tại các đỉnh hậu duệ gần nhất. Giả sử đỉnh r có 2 hậu duệ gần nhất là a và b, là các đỉnh kết thúc của các cạnh có độ dài t_a và t_b. Ta có

$$L_i^r(s) = \left(\sum_x P(x|s,t_a)L_i^a(x)\right)\left(\sum_y P(y|s,t_b)L_i^b(y)\right) \quad \textbf{(2.7)}$$

Trước khi bắt đầu tính toán, ta khởi tạo likelihood riêng phần ở các đỉnh lá như sau: nếu đỉnh lá có trạng thái A thì các giá trị L_i của đỉnh lá đó sẽ là

$$\left(L_i\ (A), L_i\ (C), L_i\ (G), L_i\ (T)\right) = (1,0,0,0) \quad \textbf{(2.8)}$$

Thuật toán bắt đầu từ một đỉnh có tất cả đỉnh hậu duệ gần nhất đều là lá (trong cây luôn có ít nhất một đỉnh trong như vậy). Sau đó nó lần lượt tính cho các đỉnh gần gốc hơn, chỉ áp dụng với các đỉnh có tất cả hậu duệ gần nhất đã được tính likelihood. Kết qủa là L_i^O cho đỉnh gốc. Ta hoàn thiện việc tính likelihood cho vị trí sắp hàng này bằng cách tính trung bình có trọng số trên tất cả 4 bazơ, sử dụng trọng số là xác suất tiền nghiệm theo mô hình xác suất:

$$L_i = \sum_x \pi_x L_i^O(x) \quad \textbf{(2.9)}$$

Công thức 2.11 cho kết quả giống **Công thức 2.8** nhưng không tốn nhiều chi phí tính toán. Với mỗi vị trí sắp hàng thì công thức đệ quy được áp dụng $n-1$ lần, mỗi lần đệ quy gồm 4 lần tính, mỗi lần tính là tích của 2 hạng tử, mỗi hạng tử là tổng của 4 tích. Như vậy, với 1 cây có n lá, sắp hàng có m vị trí và mỗi kí tự có thể nhận c trạng thái, thì chi phí tính toán tỉ lệ với $m(n-1)c^2$.

Ý nghĩa của thuật toán pruning

Thuật toán pruning cho phép tính nhanh likelihood của cây, giúp nhanh chóng tìm được độ dài tối ưu của các cạnh ứng với một cấu trúc phân nhánh cho trước. Với việc duyệt tìm cấu trúc cây tối ưu, thuật toán pruning cho phép đánh giá một cách hiệu quả các biến đổi cấu trúc cục bộ.

2.4 Đề xuất thuật toán pruning nhanh

Giả sử mô hình tiến hóa của chuỗi là mô hình Markov thuận nghịch với ma trận tốc độ Q. Không mất tính tổng quát, dưới đây chúng tôi sẽ trình bày phương pháp đề xuất cho các chuỗi DNA với các ký tự trạng thái là {A,C,G,T} và các tốc độ là độc lập và cùng phân bố.

2.4.1 Phân tích thuật toán pruning của Felsenstein

Cho trước một sắp sắp hàng sắp hàng $\boldsymbol{A}$ với các vị trí $A_1, \dots, A_m$, log-likelihood của cây T (có độ dài cạnh) và ma trận tốc độ Q được tính bởi công thức:

$$\ell(\boldsymbol{A}\,|T,Q) = \sum_{i=1}^{m} \log\big(L(A_i|T,Q)\big) \quad \textbf{(2.10)}$$

trong đó $L(A_i|T,Q)$ là *likelihood cho vị trí i* của sắp sắp hàng.

Để thu nhỏ chi phí tính toán $L(A_i|T,Q)$, Felsenstein (1981) đã đề xuất thuật toán *pruning*. Theo đó, cây T được định gốc ngẫu nhiên tại một đỉnh r với hai đỉnh con là a và b. Thuật toán sẽ duyệt đệ quy cây này để tính *vector likelihood riêng phần* $L_i^a(x)$, với $x \in \{A,C,G,T\}$ cho cây con có gốc tại a (Hình 2.2). Tương tự, ta tính vector likelihood riêng phần tại đỉnh b: $L_i^b(x)$. Các đại lượng này sẽ được kết hợp lại để tính vector likelihood riêng phần tại gốc:

$$L_i^r(x) = \left(\sum_y p_{xy}(t_a) L_i^a(y)\right)\left(\sum_y p_{xy}(t_b) L_i^b(y)\right) \quad \textbf{(2.11)}$$

trong đó

$$\left(p_{xy}(t)\right)_{x,y\in\{A,C,G,T\}} = P(t) = e^{Qt}$$

là xác suất biến đổi từ trạng thái *x* sang trạng thái *y* trong khoảng thời gian tiến hóa *t*. Lưu ý là $L_i^a(x)$ và $L_i^b(x)$ được tính theo cách thức tương tự như trong Công thức 2.13 tương ứng từ các hậu duệ của *a* và *b*.

Likelihood cho vị trí sắp hàng *i* sẽ được tính bằng:

$$L(A_i|T,Q) = \sum_x \pi_x L_i^r(x) \quad \textbf{(2.12)}$$

với π_x là tần suất cân bằng của trạng thái x.

Theo nguyên lý Pulley, khi Q thuận nghịch, likelihood của một vị trí sắp hàng *i* nào đó không thay đổi miễn là $t_a + t_b = t$ là không đổi. Nói cách khác, ta có thể đặt $t_a = 0$ (di chuyển *r* tới *a*). Khi đó, $P(t_a) = P(0)$ trở thành ma trận đơn vị và kết hợp Công thức 2.13 và Công thức 2.14 cho ta:

$$L(A_i|T,Q) = \sum_x \pi_x \, L_i^a(x) \left(\sum_y p_{xy}(t) L_i^b(y) \right) \quad \textbf{(2.13)}$$

Về cơ bản, thuật toán của Felsenstein là thuật toán quy hoạch động. Nó có độ phức tạp thời gian là $O(nmc^2)$, trong đó *n*, *m* và *c* lần lượt là số chuỗi, số vị trí sắp hàng và số trạng thái ký tự (dữ liệu DNA thì *c*=4). Độ phức tạp không gian là $O(nmc)$ nhằm lưu các vector likelihood riêng phần cho tất cả các đỉnh trong của cây.

2.4.2 Phân tích giá trị đặc trưng

Trước khi giải thích về phiên bản đề xuất cho việc cải thiện tốc độ tính toán chiều dài cạnh, luận án sẽ bắt đầu với một đặc điểm cụ thể của ma trận tốc độ thuận nghịch và dạng biến đổi tương tự của nó.

Gọi Q là ma trận tốc độ của mô hình thuận nghịch thời gian tổng quát và $\Pi = \mathrm{diag}(\pi_A, \pi_C, \pi_G, \pi_T)$ là ma trận chéo của tần suất trạng thái cân bằng. Ta có, ma trận

$$Q_1 = \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2}$$

là đối xứng với các giá trị đặc trưng thực và các vector đặc trưng thực. Ngoài ra, ta có thể tính một ma trận trực giao W từ các vector đặc trưng của Q_1 sao cho

$\Lambda = W^T \cdot Q_1 \cdot W$, với $W^T \cdot W = I = \mathrm{diag}(1,1,1,1)$.

Λ là ma trận chéo với các giá trị đặc trưng của Q_1 (và cũng là của Q).

Ta thu được

$$\Lambda = W^T \cdot \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2} \cdot W$$

Do tính kết hợp của phép nhân ma trận

$$W^T \cdot \Pi^{1/2} \cdot \Pi^{-1/2} \cdot W = I.$$

Do đó, $U^{-1} = W^T \cdot \Pi^{1/2}$ và $U = \Pi^{-1/2} \cdot W$; với U là ma trận của các vector đặc trưng của Q.

$u_{xy}^{-1} = w_{xy}^T \sqrt{\pi_y}$ and $u_{yx} = w_{yx}/\sqrt{\pi_y}$. Vì $w_{xy}^T = w_{yx}$ ta thu được

$$u_{xy}^{-1} = \pi_y u_{yx} \quad \textbf{(2.14)}$$

Công thức 2.16 sẽ được dùng về sau.

2.4.3 Tăng tốc ước lượng độ dài cạnh

Để tính $\ell(\boldsymbol{A}|T, Q)$, ta cần ước lượng độ dài tất cả các cạnh của T. Việc này chiếm hầu hết thời gian chạy. Ở đây, ta duyệt cây để tối ưu mỗi lần một cạnh, chẳng hạn bằng phương pháp Newton-Raphson (Olsen et al. 1994). Duyệt cây được lặp lại tới khi log-likelihood hội tụ. Do đó, một thao tác phổ biến là tính $\ell(\boldsymbol{A}|T, Q)$ khi biết cạnh (a, b) nối đỉnh a và b có độ dài t. Bởi Công thức 2.15 được áp dụng lặp lại khi tối ưu t, ta cần tính sẵn các vector likelihood riêng phần $L_i^a(.)$ và $L_i^b(.)$ để tiết kiệm tính toán. Chi phí tính toán cho Công thức 2.15 là mc^2 với một độ dài t cho trước. Ở phần sau đây, luận án trình bày kĩ thuật để giảm chi phí này thành mc, tức là nhanh hơn c lần phiên bản thuần túy của Công thức 2.15.

Như đã biến đổi ở phần trước:

$$Q = U \cdot \Lambda \cdot U^{-1}.$$

Do đó,

$$P(t) = e^{Qt} = U \cdot e^{\Lambda t} \cdot U^{-1}.$$

$e^{\Lambda t}$ là ma trận chéo của các hàm mũ của giá trị đặc trưng. Nói cách khác, ta có:

$$p_{xy}(t) = \sum_z u_{xz} e^{\lambda_z t} u_{zy}^{-1} \quad \textbf{(2.15)}$$

với mọi trạng thái x và y, trong đó u_{xz} và u_{zy}^{-1} là các phần tử của ma trận các vector đặc trưng U và U^{-1}. Thay vế phải của Công thức 2.17 vào Công thức 2.15 ta được

$$L(A_i|T, Q) = \sum_x \pi_x L_i^a(x) \left(\sum_y \sum_z u_{xz} e^{\lambda_z t} u_{zy}^{-1} L_i^b(y) \right).$$

Sắp xếp lại các thành phần trong công thức để có thể rút gọn bằng $\pi_x u_{xz} = u_{zx}^{-1}$ (Công thức 2.16), ta được:

$$L(A_i|T, Q) = \sum_z e^{\lambda_z t} \left(\sum_x u_{zx}^{-1} L_i^a(x) \right) \left(\sum_y u_{zy}^{-1} L_i^b(y) \right) \quad \textbf{(2.16)}$$

Kí hiệu 2 tổng trong ngoặc tròn của Công thức 2.18 là $V_i^a(z)$ và $V_i^b(z)$, ta có:

$$L(A_i|T,Q) = \sum_z e^{\lambda_z t} V_i^a(z) V_i^b(z) \quad \textbf{(2.17)}$$

So sánh Công thức 2.15 và Công thức 2.19, ta thấy đã rút gọn từ 2 phép tổng lồng nhau thành chỉ 1 phép tổng, nhờ việc lưu trữ 2 vector $V_i^a(z)$ và $V_i^b(z)$ thay cho các vector likelihood riêng phần $L_i^a(x)$ và $L_i^b(x)$. Chi phí tính toán các vector V tốn gấp đôi chi phí cho các vector likelihood riêng phần, nhưng bù lại, việc ước lượng độ dài cạnh sử dụng Công thức 2.19 nhanh hơn *c* lần sử dụng Công thức 2.15.

2.4.4 Thuật toán pruning nhanh đề xuất

Thuật toán 2.2. Thuật toán pruning nhanh

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa); mô hình tiến hóa; cây T có độ dài cạnh; cạnh (a,b) cần tối ưu độ dài cạnh

Dữ liệu ra: Cây T có cạnh (a,b) đã được tối ưu và log-likelihood tương ứng cho cây

Bắt đầu

1) Thực hiện duyệt cây theo thứ thự sau để tính các vector V cho tất cả các đỉnh dựa trên các vector V của hậu duệ sử dụng Công thức 2.20 và Công thức 2.21. Chi phí tính toán của phần này gấp đôi chi phí cho việc tính các vector likelihood riêng phần bằng Công thức 2.13.
2) Vận dụng Công thức 2.19 để ước lượng độ dài cho cạnh *(a,b)* bất kì biết rằng V_i^a và V_i^b đã được tính trước đó. Chi phí tính toán của phần này nhanh hơn 4, 20 và 61 lần so với việc sử dụng Công thức 2.15 tương ứng trên các mô hình DNA, protein và codon.

Kết thúc

Thuật toán pruning mới sẽ tính và lưu *V* thay cho việc lưu vector likelihood riêng phần cho từng đỉnh trong của cây. Để làm việc này, thay vế phải của Công thức 2.17 vào Công thức 2.13 cho ta:

$$L_i^r(x) = \left(\sum_y \sum_z u_{xz} e^{\lambda_z t_a} u_{zy}^{-1} L_i^a(y)\right)\left(\sum_y \sum_z u_{xz} e^{\lambda_z t_b} u_{zy}^{-1} L_i^b(y)\right).$$

Sắp xếp lại các thành phần trong công thức và thay *L* bằng *V*:

$$L_i^r(x) = \left(\sum_z u_{xz} e^{\lambda_z t_a} V_i^a(z)\right)\left(\sum_z u_{xz} e^{\lambda_z t_b} V_i^b(z)\right) \quad \textbf{(2.18)}$$

Vector *V* của gốc được tính bởi công thức:

$$V_i^r(z) = \sum_x u_{zx}^{-1} L_i^r(x) \quad \textbf{(2.19)}$$

Kết hợp các thành phần nói trên với nhau ta có thuật toán pruning nhanh (**Thuật toán 2.2**).

2.5 Đề xuất thuật toán UFBoot2

Chúng tôi đề xuất phương pháp UFBoot2 để xấp xỉ bootstrap cây tiến hóa ML bằng việc thay thế toàn bộ thuật toán pruning trong UFBoot (**Thuật toán 2.1**) bằng thuật toán pruning nhanh (**Thuật toán 2.2**). UFBoot2 đã được cài đặt thành công trong hệ thống IQ-TREE (mã nguồn mở cung cấp tại http://www.iqtree.org).

2.5.1 Cải tiến tốc độ bằng kĩ thuật tối ưu code

Ngoài cải tiến về thuật toán, luận án đề xuất tận dụng khả năng tính toán đơn lệnh đa dữ liệu (SIMD - single instruction, multiple data) của kiến trúc máy tính hiện đại để tính toán đồng thời likelihood cho 2 vị trí sắp hàng khi sử dụng kiến trúc SSE (streaming SIMD extensions) và cho 4 vị trí sắp hàng khi sử dụng kiến trúc AVX (advanced vector extensions), dẫn đến tăng tốc lý thuyết hai lần hoặc 4 lần so với cài đặt không sử dụng SIMD.

2.6 Kết quả thực nghiệm

Luận án thực hiện đánh giá thời gian chạy của 70 sắp hàng DNA and 45 sắp hàng protein từ TreeBASE, trước đó đã được phân tích trong Nguyen et al. (2015). UFBoot2 nhanh hơn trung bình 2,4 lần (tối đa: 77,3) so với UFBoot 0.9.6.

Chương 3 CẢI TIẾN UFBOOT2 XỬ LÝ CÁC VẤN ĐỀ DỮ LIỆU VÀ MÔ HÌNH

Chương này tập trung vào bài toán bootstrap cây tiến hóa ML và tổng hợp các kết quả nghiên cứu của luận án về cải tiến khả năng xử lý các vấn đề dữ liệu và mô hình của phương pháp UFBoot.

3.1 Cải tiến để xử lý polytomy tốt hơn

Polytomy là các đỉnh đa phân trong cây mà không thể phân giải về dạng nhị phân do không có đủ thông tin tiến hóa trong dữ liệu. Tuy nhiên, việc xây dựng cây tiến hóa lại luôn giả thiết cây có dạng nhị phân. Khi phân giải đỉnh đa phân có thể có nhiều cấu trúc nhị phân tối ưu ngang nhau. Vì UFBoot (và các cách tiếp cận bootstrap khác) chỉ lưu một cây duy nhất tối ưu cho mỗi sắp hàng bootstrap, nó có thể gán giá trị hỗ trợ bootstrap quá cao cho các cạnh ngắn.

Để khắc phục thiếu sót này UFBoot2 đã thực hiện kỹ thuật sau đây. Thay vì chọn cây bootstrap với điểm số RELL cao nhất cho mỗi sắp hàng bootstrap, UFBoot2 sẽ chọn ngẫu nhiên một trong số các cây có điểm số

RELL khác biệt không quá ε_{boot} (mặc định là 0.5) so với RELL cao nhất. Thực nghiệm trên dữ liệu mô phỏng tiến hóa từ cây hình sao (với cùng thiết kế như đề xuất trong (Simmons and Norton 2014)) đã chỉ ra rằng: UFBoot2 giống với bản UFBoot gốc, nó không bao giờ hỗ trợ các cạnh không tồn tại (các giá trị hỗ trợ bootstrap ≤88%).

3.2 Cải tiến để giảm ảnh hưởng của vi phạm mô hình

Minh và cộng sự (2013) đã chỉ ra rằng vi phạm mô hình nhiều sẽ làm tăng giá trị hỗ trợ bootstrap của UFBoot. Để giải quyết vấn đề này UFBoot2 cung cấp một tùy chọn để tiến hành một bước bổ sung khi tìm kiếm cây trên sắp hàng gốc hoàn tất. Theo đó, các cây RELL tốt nhất tiếp tục được tối ưu nhờ tìm kiếm leo đồi dùng NNI dựa trực tiếp trên sắp hàng bootstrap tương ứng. Do đó, bước bổ sung này hoạt động như SBS, nhưng là dạng tìm kiếm cây nhanh nhằm tiết kiệm thời gian. Các giá trị hỗ trợ bootstrap sau đó được tóm tắt từ các cây bootstrap đã tinh chỉnh tối ưu. Sau đây, chúng tôi gọi tùy chọn này là UFBoot2+NNI, có thể được bật lên khi dùng IQ-TREE thông qua tùy chọn "-bnni".

Luận án thực hiện lại thực nghiệm với dữ liệu mô phỏng PANDIT (Minh et al. 2013) để so sánh *độ chính xác* bootstrap của UFBoot2 và UFBoot2+NNI với SBS (bootstrap chuẩn 1000 bản sao bootstrap bằng IQ-TREE) và RBS (sử dụng tiêu chuẩn bootstopping của RAxML).

Luận án kết luận rằng UFBoot2 và UFBoot2+NNI là những phương pháp nhanh thay thế cho các tiếp cận bootstrap khác. Khi mô hình không sai hoặc sai ít, ta có thể dùng các giá trị hỗ trợ bootstrap UFBoot2 với ý nghĩa không chệch đã đề xuất cho UFBoot (Minh et al. 2013). Nghĩa là, người dùng có thể tin tưởng các cạnh có giá trị hỗ trợ bootstrap từ UFBoot2 $\geq$ 95%. Người dùng nên sử dụng các phương pháp phát hiện vi phạm mô hình (Goldman 1993; Weiss and von Haeseler 2003; Nguyen et al. 2011) trước khi làm phân tích bootstrap. Khi có khả năng cao xảy ra vi phạm mô hình so với dữ liệu quan sát thì người dùng cần sử dụng UFBoot2+NNI.

3.3 Cải tiến mở rộng để phân tích sắp hàng các bộ gen

Để hỗ trợ phân tích sắp hàng đa chuỗi đa gen, UFBoot2 đề xuất và thực hiện một số chiến lược sinh sắp hàng bootstrap như sau: (i) lấy mẫu có hoàn lại các vị trí sắp hàng trong từng gen của sắp hàng gốc (gọi là lấy mẫu theo vị trí sắp hàng; được đặt làm tùy chọn mặc định trong UFBoot2), (ii) lấy mẫu có hoàn lại các gen thay vì các vị trí trên sắp hàng (gọi là lấy mẫu theo gen; được gọi thông qua tham số dòng lệnh "-bsam GENE") và (iii) lấy mẫu có hoàn lại các gen và sau đó tiếp tục lấy mẫu có hoàn lại các vị trí sắp hàng trong mỗi gen (gọi là lấy mẫu theo gen-vị trí; được gọi thông qua tham số dòng lệnh "-bsam GENESITE") (Gadagkar et al. 2005).

Để khảo sát ảnh hưởng của ba chiến lược lấy mẫu, luận án phân tích lại dữ liệu metazoan trong đó có 21 loài, 225 gen và tổng số 171077 vị trí amino-acid và mở rộng khảo sát với 14 bộ dữ liệu sinh học khác. Chúng tôi quan sát được những kết quả nhất quán và cả những kết quả khác biệt giữa các chiến lược lấy mẫu. Trên một số bộ dữ liệu, có những cạnh gần như không nhận được hỗ trợ nào (≤10%) từ một chiến lược lấy mẫu nhưng lại có hỗ trợ cao (≥95%) từ hai chiến lược lấy mẫu kia. Tuy nhiên, không có dấu hiệu nào về việc một chiến lược luôn luôn cho giá trị hỗ trợ bootstrap thấp. Từ những phát hiện nói trên, luận án khuyến cáo người dùng áp dụng tất cả các chiến lược lấy mẫu. Nếu giá trị hỗ trợ bootstrap thu được là tương tự nhau thì kết quả có độ tin cậy cao hơn.

Chương 4 ĐỀ XUẤT PHƯƠNG PHÁP MỚI MPBOOT XẤP XỈ NHANH BOOTSTRAP CÂY TIẾN HÓA THEO TIÊU CHUẨN TIẾT KIỆM NHẤT

4.1 Giới thiệu

Bootstrap cây tiến hóa MP theo lược đồ chuẩn có 2 vấn đề: (i) nó tiêu tốn nhiều thời gian khi số lượng loài lớn; (ii) nó cho ước lượng chệch về xác suất đúng của cạnh.

4.2 Xây dựng cây tiến hóa theo tiêu chuẩn MP

Kí hiệu A^{data} là sắp hàng của n chuỗi và m vị trí có thông tin parsimony. Vị trí có thông tin parsimony là vị trí có chứa ít nhất hai ký tự trạng thái. m vị trí này được nhóm thành các mẫu-vị trí $D_1, D_2, \dots, D_k$ với tần suất tương ứng là $d_1, d_2, \dots, d_k$. Điểm MP cho cấu trúc cây T khi biết A^{data} được tính bởi công thức:

$$MP(T|A^{data}) = \sum_{i=1}^{k} MP(T|D_i) \times d_i \qquad \textbf{(3.1)}$$

trong đó $MP(T|D_i)$ là điểm MP cho cây T tại mẫu-vị trí D_i.

Với một cây T cho trước, $MP(T|D_i)$ có thể tính được một cách hiệu quả nhờ thuật toán Fitch (Fitch 1971) trong trường hợp chi phí biến đổi giữa các trạng thái là bằng nhau, tức *ma trận chi phí đều*. Trường hợp *ma trận chí phí không đều*, ta dùng thuật toán Sankoff (Sankoff 1975) để tính $MP(T|D_i)$. Một phương pháp tìm kiếm cây theo tiêu chuẩn MP có mục tiêu tìm một cây có điểm MP nhỏ nhất. Bài toán tìm cây MP tốt nhất thuộc lớp NP-đầy đủ, do đó, cần sử dụng các heuristic cho tìm kiếm cây.

4.3 Phương pháp đề xuất để xấp xỉ bootstrap theo tiêu chuẩn MP (MPBoot)

4.3.1 Lấy mẫu cây trên sắp hàng gốc

Trước tiên, MPBoot tạo ra một tập các sắp hàng bootstrap và sau đó tìm kiếm trên không gian cây dựa trên sắp hàng gốc. Các cây được duyệt (còn gọi là lấy mẫu) trong quá trình tìm kiếm cây này được coi là cây MP tiềm năng cho mỗi sắp hàng bootstrap.

Thuật toán MPBoot tìm kiếm cây MP cho sắp hàng gốc hoạt động nhờ chỉnh sửa dần dần một tập ứng viên *C* chứa các cây tối ưu cục bộ khác nhau. Để khởi tạo tập ứng viên *C*, MPBoot tạo ra 100 cây tối ưu cục bộ theo tiêu chuẩn MP nhờ chiến lược xây dựng cây từng bước ngẫu nhiên (randomized stepwise addition) (Wagner 1961) theo sau là tìm kiếm leo đồi với kĩ thuật SPR (Stamatakis et al. 2008). Thuật toán sắp xếp các cây này tăng dần theo điểm MP của chúng, rồi chọn 5 cây phân biệt đầu tiên để tạo tập ứng viên ban đầu. Trong quá trình tìm kiếm cây, *C* sẽ được cập nhật liên tục với các cây tốt hơn. Điều này hoàn tất *bước khởi đầu (initial step)* của MPBoot.

Trong *bước khám phá (exploration step)* tiếp sau đó, MPBoot sẽ luân phiên giữa phép xáo trộn cấu trúc cây (perturbation) và phép leo đồi (hill-climbing). Việc này sẽ được lặp lại nhiều lần nhằm thoát khỏi cực trị địa phương trong không gian tìm kiếm cây.

Phép xáo trộn cấu trúc cây: MPBoot trước tiên chọn ngẫu nhiên một cây T_C từ tập ứng viên *C*. Sau đó, T_C bị xáo trộn theo một trong 2 cách sau: (i) thực hiện trao đổi ngẫu nhiên hàng xóm gần nhất (nearest neighbor interchange, NNI) trên 50% các cạnh trong chọn ngẫu nhiên để tạo ra cây T^* hoặc (ii) áp dụng parsimony ratchet (Nixon 1999). Parsimony ratchet sẽ nhân đôi 50% các vị trí có thông tin parsimony trên sắp hàng gốc để sinh ra một sắp hàng xáo trộn. Sau đó, ratchet thực hiện một tìm kiếm leo đồi với kĩ thuật SPR trên sắp hàng xáo trộn này, bắt đầu từ cây T_C để tìm ra một cây tối ưu cục bộ T^*. Nói tóm lại, T^* được tạo ra bằng chiến lược xáo trộn cây (các NNI ngẫu nhiên) hoặc bằng chiến lược xáo trộn sắp hàng.

Phép leo đồi: Cây T^* sau đó đóng vai trò cây ban đầu cho tìm kiếm leo đồi SPR để tìm ra một cây tối ưu cục bộ T^{**} cho A^{data}. Nếu $MP(T^{**}|A^{data})$ nhỏ hơn hoặc bằng điểm MP lớn nhất trên A^{data} của các cây trong tập ứng viên thì T^{**} sẽ thế chỗ cây tương ứng trong tập ứng viên. Luận án gọi phép leo đồi là thành công nếu $MP(T^{**}|A^{data})$ nhỏ hơn chặt điểm MP nhỏ nhất trên A^{data} của các cây trong tập ứng viên. Ngược lại, luận án gọi phép leo đồi là không thành công, nghĩa là thuật toán đã không tìm thấy một cây tốt hơn.

Bước khám phá áp dụng các bước xáo trộn theo sau là bước leo đồi cho đến khi n' (làm tròn số lượng chuỗi n lên tới hàng trăm) bước tìm kiếm leo đồi cuối cùng đều không thành công. Điều này cho phép tìm kiếm kỹ hơn với các sắp hàng có rất nhiều chuỗi. Do vậy, MPBoot dừng lại bởi vì nó không thể tìm được cây tốt hơn. Bước khám phá hoàn tất.

4.3.2 Lấy mẫu điểm MP (Resampling parsimony score - REPS)

Với cây T và các điểm số mẫu-vị trí $MP(T|D_i)$ tính trên A^{data}, điểm MP cho $A^{bootstrap}$ được tính nhanh bằng tổng có trọng số của các điểm số mẫu-vị trí:

$$MP(T\big|A^{bootstrap}) = \sum_{i=1}^{k} MP(T|D_i) \times d_i^{bootstrap} \quad \textbf{(3.2)}$$

trong đó $d_i^{bootstrap}$ là số lần được lấy mẫu của mẫu-vị trí D_i khi tạo $A^{bootstrap}$. Vì vậy, không cần phải lặp lại việc tính điểm MP cho mỗi mẫu-vị trí, mỗi bản sao bootstrap và mỗi cây.

4.3.3 Tăng tốc tính toán REPS

Để tăng tốc việc tính điểm MP, MPBoot sử dụng thêm hai kĩ thuật tối ưu hóa thuật toán.

Thứ nhất, luận án đề xuất sử dụng một ngưỡng MP_{max}, sao cho điểm MP tính trên các sắp hàng bootstrap sử dụng Công thức 3.2 chỉ áp dụng cho những cây T, được tìm thấy trong bước leo đồi, thỏa mãn $MP(T\big|A^{data}) < MP_{max}$. Ban đầu, $MP_{max} = \infty$. Sau bước leo đồi đầu tiên, thuật toán thiết lập MP_{max} là bách phân vị thứ 90 của phân bố điểm MP trên sắp hàng gốc của tất cả các cây duyệt trong bước leo đồi này. Trong các bước leo núi tiếp theo, thuật toán chỉ xem xét các cây có điểm MP trên A^{data} nhỏ hơn MP_{max}. Điểm MP của những cây này tạo thành một phân bố sau đó sẽ được sử dụng để cập nhật MP_{max} sau mỗi bước leo đồi như trên.

Thứ hai, thuật toán dừng việc tính toán REPS cho một $A^{bootstrap}$ nếu ta không thể mong đợi rằng $MP(T\big|A^{bootstrap})$ sẽ nhỏ hơn điểm MP tốt nhất tính đến hiện tại $MP_{best}(A^{bootstrap})$. Để làm điều này, thuật toán sắp xếp các mẫu-vị trí D_i theo điểm MP giảm dần dựa trên cây đầu tiên xây dựng được ở bước bắt đầu. Điểm MP nhỏ nhất về mặt lý thuyết, $MP_{min}(D_i)$, bằng số các trạng thái ký tự riêng biệt có mặt trong D_i trừ đi 1 trong trường hợp ma trận chi phí đều. Trường hợp ma trận chi phí không đều, $MP_{min}(D_i)$ bằng độ dài của cây khung nhỏ nhất trên đồ thị chi phí. Đồ thị này có các đỉnh tương ứng các trạng thái ký tự có mặt trong D_i và trọng số

cạnh bằng chi phí biến đổi giữa các trạng thái. Thuật toán dừng việc tính toán REPS nếu

$$\sum_{i=1}^{j} MP(T|D_i) \times d_i^{bootstrap} + \sum_{i=j+1}^{k} MP_{min}(D_i) \times d_i^{bootstrap} > MP_{best}(A^{bootstrap}) \quad \textbf{(3.3)}$$

với một số giá trị j $(1 \leq j \leq k)$, bởi T không thể là cây MP cho $A^{bootstrap}$.

4.3.4 Thuật toán MPBoot

Ta có thể tóm tắt toàn bộ hoạt động của MPBoot như sau:

Thuật toán 3.1. Thuật toán MPBoot

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa).

Dữ liệu ra: Cây MP xây dựng cho sắp hàng gốc được gắn các giá trị bootstrap cho mỗi cạnh

Bắt đầu

4) Bước khởi đầu: Tạo B sắp hàng bootstrap, $A_1, A_2, \dots, A_B$. Với mỗi A_b khởi tạo cây bootstrap $T_b := null$ và $MP(T_b|A_b) := +\infty$. Khởi tạo một tập các cây $S := \{\}$ và ngưỡng $MP_{max} := +\infty$. Khởi tạo tập ứng viên C trên A^{data} như giải thích trong Phần 3.2.1.
5) Bước khám phá: Từ một cây chọn ngẫu nhiên trong tập ứng viên C, thực hiện bước xáo trộn theo sau là bước leo đồi. Mỗi khi duyệt một cây T mới thỏa mãn $MP(T|A^{data}) < MP_{max}$, thêm T vào S và tính $MP(T|A_b)$, cho các $b = 1, \dots, B$ dựa trên Công thức 3.2. Nếu $MP(T|A_b) < MP(T_b|A_b)$, cập nhật $T_b := T$. Khi bước leo đồi thực hiện xong, cập nhật MP_{max} bằng bách vị phân thứ 90 của phân bố các điểm MP của các cây trong *S*.
6) Điều kiện dừng: Nếu n' bước leo đồi liên tiếp đều không thành công, thực hiện bước 4. Ngược lại, quay về bước 2.
7) Bước tinh chỉnh: Với mỗi cây MP T_b $(b = 1, \dots, B)$ và sắp hàng tương ứng A_b , tiến hành một lượt tìm kiếm leo đồi dùng SPR và thay thế T_b bằng cây MP mới nếu cây mới có điểm MP tốt hơn.
8) Bước tóm tắt: Xây dựng một cây đồng thuận từ các cây bootstrap $\{T_1, T_2, \dots, T_B\}$, hoặc ánh xạ giá trị hỗ trợ bootstrap lên cây MP tốt nhất trên A^{data}.

Kết thúc

MPBoot sử dụng thư viện tính toán likelihood cho phân tích tiến hóa PLL để tính toán parsimony hiệu quả. Ngoài ra, tất cả các tính toán cốt lõi

parsimony và tính toán REPS được vector hoá bằng các khả năng tính toán đơn lệnh đa dữ liệu (SIMD) của kiến trúc máy tính x86 hiện đại.

4.4 Thiết kế thực nghiệm

Luận án so sánh hiệu năng của MPBoot SPR3 và SPR6 (được biên dịch với SSE4) với bootstrap chuẩn (1000 bản sao bootstrap), cài đặt trong TNT phiên bản 1 .1 (tháng 10 năm 2014) và PAUP * phiên bản 4.0a152 (tháng 1 năm 2017). Tất cả các phương pháp lưu một và chỉ một cây tốt nhất cho mỗi sắp hàng bootstrap. Luận án so sánh các sắp hàng DNA và protein sử dụng ma trận chi phí đều và không đều.

Chúng tôi sử dụng hai phân tích bootstrap trong TNT: fast-TNT tìm kiếm nhanh trên sắp hàng gốc và các sắp hàng bootstrap. intensive-TNT tìm kiếm kĩ trên sắp hàng gốc và tìm kiếm nhanh các sắp hàng bootstrap. Chúng tôi cũng khảo sát bootstrap chuẩn cài đặt trong PAUP* với chiến lược giống fast-TNT.

4.5 Kết quả

MPBoot cho điểm MP tốt hơn fast-TNT và PAUP* và tương đương với intensive-TNT. MPBoot SPR6 thu được các giá trị bootstrap gần như không chệch bất kể loại dữ liệu sử dụng và ma trận chi phí cụ thể. MPBoot có thời gian chạy ít hơn đáng kể so với PAUP*.

KẾT LUẬN

Cũng giống như ý nghĩa quan trọng của thông số độ tin cậy trong các bài toán ước lượng tham số thống kê, việc làm bootstrap cây tiến hóa để xác định độ tin cậy cho từng cạnh của cây tiến hóa xây dựng được là bước không thể thiếu trong phân tích cây tiến hóa. Luận án tập trung vào bài toán này cho tiêu chuẩn hợp lý nhất và tiêu chuẩn tiết kiệm nhất vì đây là hai hướng tiêu biểu trong phân tích tiến hóa phân tử. Phương pháp bootstrap chuẩn có hai hạn chế chính là (i) vô cùng tốn kém về thời gian thực hiện và (ii) cho ước lượng thấp hơn xác suất đúng của cạnh. Sự gia tăng không ngừng về quy mô dữ liệu sinh học phân tử phục vụ phân tích tiến hóa đặt ra đòi hỏi phải khắc phục các hạn chế này.

Dựa trên phân tích điểm mạnh, điểm yếu của phương pháp UFBoot xấp xỉ nhanh bootstrap chuẩn theo tiêu chuẩn hợp lý nhất, luận án đã đề xuất phương pháp UFBoot2 với bốn cải tiến quan trọng: (i) Cải tiến tốc độ với đề xuất thuật toán pruning nhanh và các kĩ thuật tối ưu code, nhờ đó nhanh hơn phiên bản cũ trung bình 2,4 lần; (ii) Cải tiến để xử lý các đỉnh đa phân tốt hơn, nhờ đó thể hiện thành công trên thực nghiệm mô phỏng kiểm tra cây hình sao (iii) Cải tiến để giảm ảnh hưởng của vi phạm mô hình , nhờ

đó cho ước lượng sát với xác suất đúng của cạnh nếu vi phạm mô hình nhiều và (iv) Cải tiến mở rộng để phân tích sắp hàng nhiều gen.

Phỏng theo ý tưởng của UFBoot cho ML, luận án đã đề xuất một phương pháp MPBoot mới để tìm kiếm hiệu quả cây MP, đồng thời xấp xỉ hiệu quả bootstrap chuẩn theo tiêu chuẩn MP. MPBoot có thể sử dụng ma trận chi phí đều hoặc không đều. Luận án đã thực hiện các thực nghiệm trên cả dữ liệu mô phỏng và các bộ dữ liệu sinh học lớn để so sánh MPBoot và các phương pháp cài đặt trong hai công cụ phân tích parsimony tốt nhất hiện nay là TNT và PAUP* trên các mặt: thời gian tính toán, khả năng tìm ra cây MP và độ chính xác của ước lượng bootstrap. Xấp xỉ bootstrap của MPBoot thực hiện nhanh hơn bootstrap chuẩn cài đặt trong PAUP* khi sử dụng ma trận đều; nhanh hơn fast-TNT khi sử dụng ma trận chi phí không đều. MPBoot xây dựng được cây MP có điểm MP tương đương với intensive-TNT. Trong khi các bản cài đặt bootstrap chuẩn đều cho ước lượng thấp về xác suất đúng của cạnh, MPBoot SPR6 thu được các giá trị bootstrap gần như không chệch bất kể loại dữ liệu sử dụng và ma trận chi phí cụ thể. MPBoot, do đó, có thể thay thế hiệu quả các tiếp cận bootstrap chuẩn theo tiêu chuẩn MP.

Cuối cùng, phương pháp UFBoot2 đề xuất trong luận án đã được tích hợp vào hệ thống IQ-TREE (địa chỉ website: http://www.iqtree.org) là phần mềm mã nguồn mở tốt nhất hiện nay cho phân tích cây tiến hóa theo tiêu chuẩn hợp lý nhất và có nhiều người sử dụng. Các đề xuất về phương pháp mới MPBoot đã được cài đặt trong phần mềm mã nguồn mở MPBoot (địa chỉ website: http://www.cibiv.at/software/mpboot).