Nouvelles heuristiques de voisinage et mémétiques pour le problème Maximum de Parcimonie

Adrien Goëffon

UNIVERSITÉ
D'ANGERS

Année : 2006
N° d'ordre : 775

Nouvelles heuristiques de voisinage et mémétiques pour le problème Maximum de Parcimonie

Thèse de doctorat

Spécialité : Informatique

École Doctorale d'Angers

Présentée et soutenue publiquement

Le 21 novembre 2006

À Angers

Par **Adrien GOËFFON**

Devant le jury ci-dessous :

Président :	Jean-Jacques Chabrier,	Professeur à l'Université de Bourgogne
Rapporteurs :	Philippe Galinier, El-Ghazali Talbi,	Professeur adjoint à l'École Polytechnique de Montréal Professeur à l'Université de Lille
Examinateur :	Rumen Andonov,	Professeur à l'Université de Rennes
Directeur de thèse :	Jin-Kao Hao,	Professeur à l'Université d'Angers
Co-encadrant :	Jean-Michel Richer,	Maitre de conférences à l'Université d'Angers

Avertissement

Ce manuscrit se veut être au maximum en adéquation avec les rectifications orthographiques du 6 décembre 1990, parues dans le *Journal officiel de la République française*. Ainsi, parmi les modifications les plus notables dans ces pages, certains mots perdent leur accent circonflexe (**apparaitre**, **boite**, **chaine**, **connaitre**, **cout**, **(dé)croitre** sauf *croît*, **disparaitre**, **entrainer**, **maitriser**, **naitre**, **traiter**), et d'autres voient leur accent différer (**évènement**, **règnant**) ou leur tréma être déplacé (**ambigüité**). De plus, les pluriels de mots empruntés, d'origine latine ou étrangère, sont francisés (**minimums**, **optimums**, **scénarios**).

Remerciements

Mes premiers remerciements s'adressent à Jin-Kao Hao, mon directeur de thèse, pour le soutien constant et les nombreux conseils qu'il m'a prodigués tout au long de ces trois années de thèse. Je lui suis particulièrement reconnaissant de m'avoir laissé une grande liberté scientifique, ce qui m'a permis de recevoir un apprentissage idéal et privilégié de la recherche. La confiance que M. Hao m'a accordée, depuis qu'il m'a offert l'opportunité d'effectuer cette thèse de doctorat sous sa responsabilité, m'a toujours beaucoup honoré.

Je remercie très chaleureusement Jean-Michel Richer, qui a toujours été à mes côtés dans la réalisation de cette thèse, n'hésitant pas à prendre sur son temps pour m'aider ou parfaire nos travaux. Sa gentillesse inaltérable, ses remarques constructives et ses fréquents encouragements ont été des atouts majeurs dans l'efficacité et l'entrain de notre travail.

Je souhaite exprimer ma profonde gratitude envers Philippe Galinier et El-Ghazali Talbi pour avoir accepté d'être rapporteurs de ma thèse. Je sais également gré à Rumen Andonov et Jean-Jacques Chabrier, qui ont accepté de participer au jury en tant qu'examinateurs.

Les membres permanents du LERIA et du département informatique ont une place prépondérante dans la réalisation de cette thèse, pour plusieurs raisons. Tout d'abord, la plupart d'entre eux ont été mes enseignants lorsque j'étais étudiant en informatique, certains m'ayant transmis la passion pour la recherche. Ensuite, j'ai beaucoup appris au contact de l'ensemble des enseignants-chercheurs et techniciens, et apprécié l'ambiance conviviale règnant au laboratoire. Je remercie tout particulièrement Vincent Barichard, David Genest, Frédéric Lardeux et Frédéric Saubion pour leur disponibilité, leur aide et leurs nombreux conseils, ainsi que Gilles Hunault et Igor Stéphan pour — entre autres — leurs remarques pertinentes et leur participation à la tâche laborieuse que constitue la relecture du présent manuscrit.

Je ne peux qu'adresser un remerciement particulier et amical à Tony Lambert, qui a supporté mes humeurs presque quotidiennement depuis plus de trois ans, notre bureau commun étant devenu à chacun notre second chez-soi. Je remercie par la même occasion mes amis doctorants — Olivier Cantin, Vincent Derrien, Antoine Robin, Eduardo Rodriguez-Tello, Thomas Raimbault, Sylvain Lamprier, Marc-Olivier Buob, sans oublier Hervé Deleau —, pour les bons moments de détente passés en leur compagnie.

Le début de la thèse a coïncidé pour ma part à un nouveau départ, tant du point de vue des aspirations professionnelles que sur le plan personnel. J'ai eu la chance d'y rencontrer des amis remarquables, doctorants pour la plupart, qui à n'en pas douter ont joué un rôle important dans l'efficacité de mon travail car j'ai pu retrouver sérénité et équilibre. J'adresse donc un grand merci à Sébastien, Clothilde, Laurent, Jean-Fred, Hélène, Sophie, Magali, Xavier, Clémentine, Gyasi, Ludovic, Marie-Aude, Daphné et Vincent. Je remercie

également Audrey, Thomas, Didier, Charline, Gaëtan et Jean-Philippe pour leur amitié sincère.

L'équilibre et le bonheur au quotidien, il est évident que je le dois principalement à Émilie, que je remercie du fond du cœur pour toutes les attentions qu'elle me porte. Un grand mérite lui revient dans la réalisation de cette thèse, tant elle a partagé les doutes et les satisfactions que ce doctorat a pu me procurer. Je remercie également ses parents pour la gentillesse et la générosité dont ils font preuve à mon égard.

Enfin, je remercie très affectueusement mes parents pour le soutien considérable qu'ils m'ont apporté et la confiance dont ils m'ont toujours témoigné. Pour tous les efforts qu'ils ont consentis durant de nombreuses années afin que je mène à bien mes études, le moins que je puisse faire est de leur dédier cette thèse.

Table des matières

Introduction

Contexte de travail

Le terme **bio-informatique** regroupe un ensemble de méthodes, techniques et outils liés aux mathématiques et à l'informatique dont le but est l'interprétation des données biologiques. Celles-ci concernent essentiellement des macromolécules (acides nucléiques et protéines). La bio-informatique [Mount, 2001] est devenue une discipline à part entière qui ne se cantonne pas aux deux disciplines qui la définissent terminologiquement mais englobe l'ensemble des sciences exactes et naturelles. Si nous voulons progresser dans notre connaissance des phénomènes du vivant, il nous faut allier les compétences des chercheurs en biologie, chimie, physique, mathématiques et informatique. Si la bio-informatique s'attache davantage à l'analyse, l'organisation et la visualisation de données biologiques via l'utilisation d'outils informatiques, la **biologie computationnelle**, discipline sœur, comprend le développement de méthodes basées sur la modélisation mathématique et la simulation computationnelle pour l'étude de systèmes biologiques. Par abus de langage, l'appellation bio-informatique est souvent employée pour regrouper ces deux domaines proches.

Les récents progrès des programmes de séquençage des génomes (comme le projet Génome Humain achevé en 2003 [Collins *et al.*, 2003]) ont ouvert de nouvelles perspectives dans les domaines de la biologie, de la santé et de l'agronomie. Parmi les principaux domaines d'application de la bio-informatique, on trouve :

- l'analyse de séquences (alignement de séquences, recherche de séquences et de motifs dans des bases de données, *etc.*),
- la reconstruction d'arbres phylogénétiques,
- la prédiction de structures protéiques,
- l'annotation de génomes.

La **reconstruction phylogénétique** [Nei and Kumar, 2000 ; Felsenstein, 2003] a pour but de reconstruire l'histoire évolutive d'organismes. Si l'on écarte le créationnisme, il est admis en biologie que les espèces vivantes actuelles sont issues d'un même ancêtre commun, qui durant des millions d'années n'a cessé de muter, de se transformer pour donner naissance à différentes espèces ayant chacunes leurs caractéristiques propres. Le résultat d'une reconstruction phylogénétique est un arbre, appelé *arbre phylogénétique* ou *phylogénie*, dont les feuilles représentent les espèces contemporaines et les nœuds internes des ancêtres hypothétiques.

Cette branche de la bio-informatique trouve aujourd'hui, grâce à l'utilisation des séquences d'ADN ou de protéines, de nombreuses applications en biologie et en médecine,

notamment pour ce qui concerne la détermination de l'origine d'agents pathogènes [Carrington and Hoelzel, 2001 ; Greenblatt and Spigelman, 2003] ou l'évolution de souches virales (grippe [Bush *et al.*, 1999], sida [Rambaut *et al.*, 2001], anthrax [Keim *et al.*, 2004], *etc.*).

Il existe trois types d'analyses phylogénétiques. L'**analyse phénétique** se fonde sur l'utilisation de mesures de distances entre espèces. L'**analyse probabiliste**, quant à elle, nécessite un modèle de l'évolution qui établit des probabilités de mutation entre nucléotides. Enfin, une **analyse cladistique** sur des séquences d'ADN préalablement alignées part du postulat que l'arborescence la plus probable est censée correspondre à celle qui possède le nombre minimal de mutations. La recherche d'un ou des arbres minimaux correspond au problème de la recherche du Maximum de Parcimonie — généralement abrégé en problème du **Maximum de Parcimonie** (MP) — qui fait l'objet de cette thèse.

Étant donné un ensemble de séquences composées d'un même nombre de caractères (généralement des séquences ADN), le problème MP consiste à retrouver un arbre binaire dont les feuilles sont associées aux séquences données, les nœuds internes à des séquences hypothétiques correspondant à d'éventuels ancêtres communs, et qui minimise le score de parcimonie, *i.e.* la somme des différences entre les séquences associées à des nœuds adjacents. Le critère de parcimonie, élément permettant l'évaluation des phylogénies, est très simple à appréhender et constitue en conséquence une approche très populaire pour la reconstruction phylogénétique. En revanche, résoudre le problème MP, *i.e.* déterminer les arbres les plus parcimonieux, est une tâche difficile en raison de sa NP-complétude [Foulds and Graham, 1982]. Traiter ce problème de minimisation consiste alors à concevoir une méthode permettant de trouver des solutions approchées en des temps de calcul raisonnables.

Problématique et contribution

Le problème MP peut être considéré comme un problème de minimisation sur un espace d'arbres. À partir d'une quinzaine de séquences (plus de 10^{14} configurations pour 15 séquences, et une croissance factorielle de l'espace de recherche en fonction du nombre de séquences), sa résolution nécessite le recours à des heuristiques. Les algorithmes les plus connus utilisent la recherche locale (descente pure, itérée, à voisinage variable, *etc.*), généralement couplée à d'autres techniques comme les algorithmes gloutons, génétiques ou de décomposition. Cependant, parmi les nombreux logiciels de parcimonie existants, peu sont efficaces lorsqu'il s'agit de résoudre des instances de grande taille. L'instance réelle *zilla* [Chase *et al.*, 1993] par exemple, très populaire dans la littérature pour tester les méthodes de résolution du problème MP, n'a été résolue efficacement que par des logiciels très pointus et qui utilisent de nombreuses méthodes interconnectées (logiciel `TNT` [Goloboff *et al.*, 2000]), ou bien par des algorithmes parallélisés [Du *et al.*, 2005]. Dans les deux cas, ces logiciels ont été conçus de manière à accélérer les temps de calcul afin de passer en revue des milliards de solutions éventuelles, mais ne proposent pas d'heu-

ristiques innovantes permettant une approche moins brutale du problème. Il est vrai que concernant l'instance *zilla*, comme sur d'autres instances structurées de grande taille, la motivation concerne davantage les temps de calcul que la recherche de nouvelles solutions ; en effet, l'expérience montre qu'un algorithme de descente utilisant des voisinages d'arbres larges (SPR, TBR [Swofford and Olsen, 1990]) et exécuté durant des millions voire milliards d'itérations suffit à trouver de bonnes solutions. Cependant, sur des instances non structurées, même de petite taille, ou sur des instances réelles de très grande taille, ces algorithmes montrent leurs limites.

Dans cette thèse, nous appliquons des techniques d'optimisation combinatoire éprouvées (recherche locale, algorithmes mémétiques) et y introduisons de nouvelles stratégies de voisinages et de croisements d'arbres, afin d'améliorer les résultats obtenus par les algorithmes existants pour constituer un algorithme de résolution du problème MP plus performant. La principale difficulté de ce problème concerne l'espace de recherche constitué d'arbres binaires de taille fixe et dont les feuilles représentent les données initiales du problème, nous conduisant ainsi à utiliser des voisinages et des croisements spécifiques adaptatés au problème.

Au vu de l'importance des phases de recherche locale dans les algorithmes existants, un premier axe de recherche consiste à étudier empiriquement les voisinages traditionnels (NNI, SPR et TBR, qui consistent à déplacer des sous-arbres). Cette étude nous a permis d'isoler les transformations les plus pertinentes en fonction de la qualité des arbres, et de définir en conséquence des stratégies visant à rendre les phases de recherche locale plus efficaces. C'est ainsi que nous avons introduit le concept de voisinage progressif [Goëffon *et al.*, 2005b], qui se restreint au fur et à mesure de la recherche de manière à ne considérer que les solutions potentiellement les plus pertinentes.

Pour résoudre les instances très difficiles, nous avons conçu un algorithme mémétique, qui hybride cette descente à voisinage progressif à un croisement d'arbres novateur [Goëffon *et al.*, 2006]. Les croisements d'arbres existants [Moilanen, 1999 ; Congdon and Septor, 2003 ; Ribeiro and Vianna, 2003] génèrent une descendance à partir de morceaux de parents, et perdrent en conséquence la moitié de leur information topologique. Au contraire, le croisement que nous introduisons résume l'ensemble des informations sémantiques de chaque parent dans une matrice de distances, puis combine ces deux matrices en une seule pour reconstruire un arbre fils. Extraire les informations sémantiquement fortes de chaque parent et les considérer comme uniques données pour la reconstruction d'une nouvelle configuration est une nouvelle stratégie de croisement pour les algorithmes génétiques et mémétiques.

L'évaluation de nos méthodes a été réalisée sur un large panel d'instances, comprenant notamment des *benchmarks* issus de la littérature, traduisant des problèmes réels ou bien générés alétoirement.

Les principaux résultats obtenus indiquent que la descente à voisinage progressif permet d'atteindre de bien meilleures solutions que les autres heuristiques de voisinage, en un

nombre d'itération plus faible. Enfin, l'algorithme mémétique, que nous avons baptisé `HYDRA` pour *HYbrid Distance Recombination Algorithm*, obtient de meilleurs résultats — en terme de qualité et de faible dispersion des scores de parcimonie atteints — que les logiciels existants, en particulier sur les instances les plus difficiles.

Organisation de la thèse

Cette thèse s'articule en deux parties. La première est un état de l'art du problème MP et des méthodes de résolution associées.

- Dans le premier chapitre, après un préambule historique sur les théories de l'évolution et la naissance du concept de phylogénie, nous introduisons les arbres phylogénétiques en tant qu'objets mathématiques, puis décrivons les différentes analyses phylogénétiques existantes ;
- Le chapitre 2 constitue un exposé précis du critère de parcimonie, de la signification du score de parcimonie d'un arbre et des termes en lesquels se pose le problème MP ;
- Enfin, le chapitre 3 expose les différentes techniques de résolution appliquées au problème MP dans les logiciels couramment utilisés par les biologistes.

La seconde partie décrit les contributions de cette thèse, c'est-à-dire la conception de nouvelles heuristiques pour le problème MP, et est décomposée en trois chapitres :

- L'étude empirique d'heuristiques de voisinage traditionnelles appliquées à ce problème, décrite au chapitre 4, permet de dégager des enseignements importants et d'envisager des pistes de travail ;
- Le fonctionnement du voisinage progressif et les justifications de son utilisation sont décrits dans le chapitre 5 ;
- Quant au dernier chapitre, il présente le croisement d'arbres que nous avons conçu, et son intégration au sein d'un algorithme mémétique performant.

Nous espérons que le lecteur trouvera autant de satisfaction à lire ce manuscrit que nous en avons éprouvé à le rédiger.

Première partie

État de l'Art

Chapitre 1

La reconstruction phylogénétique : histoire et méthodes

La phylogénétique, du grec *phylon* (tribu, race), et *genetikos* (relatif aux gènes, de *genesis* : naissance) est l'étude des relations évolutionnaires entre les organismes. Depuis toujours l'Homme connait le besoin de classer les objets, de construire des ontologies. Toute connaissance se doit d'être nommée au sein d'une classification pour être identifiée, pour exister. Dans tous les domaines, on peut remarquer cette volonté de décrire l'existant tel un arbre composé de groupes, où peuvent s'exprimer par hiérarchies des relations horizontales ou verticales. En particulier, la connaissance du vivant fut l'une des premières motivations de l'Homme. Les premières classifications tentaient de répondre aux interrogations du type « *qui est proche de qui ?* » , puis au fil du temps à « *qui descend de qui ?* » . De ce processus est née la notion d'arbre phylogénétique, avant que ne se pose la question des méthodes de reconstruction.

Ce chapitre débute par une évocation rapide de cette évolution des idées, conforme à celle de nos connaissances universelles. Elles aboutirent il y a deux siècles au Darwinisme, époque où naquirent les arbres d'évolution ; nous passerons outre les débats contemporains sur les théories de l'évolution et nous concentrerons sur les définitions et représentations de ces arbres dits phylogénétiques. Par la suite, nous verrons qu'il existe trois critères de reconstruction phylogénétique. Comme l'ensemble des chapitres suivants est consacré à l'analyse cladistique, nous évoquerons surtout ici les deux autres concepts de reconstruction : l'analyse phénétique et l'analyse probabiliste.

Sommaire

1.1 Les théories de l'évolution

Aborder un thème de recherche situé aux confins d'une science ancienne telle la biologie et d'une autre plus moderne comme la science de l'information nécessite en premier lieu de posséder les moyens de les réunir, en particulier en plaçant les connaissances actuelles dans leur contexte philosophique et scientifique. Nous dressons ici un aperçu des différentes théories avancées dans le domaine de la classification et de l'évolution présumée des espèces, et montrons par quel cheminement d'idées le concept d'arbre phylogénétique est né.

1.1.1 La génération spontanée

Le philosophe grec Démocrite d'Abdère et son mentor Leucippe préfigurent déjà l'évolution des espèces quatre siècles avant JC. Leur théorie atomiste selon laquelle l'Univers serait composé de vide et d'un nombre infini d'atomes, exclut pour la première fois l'intervention des Dieux dans l'explication de l'Univers et des êtres vivants. Dans la foulée, Aristote[1] pose la théorie de la *génération spontanée* (aujourd'hui existe le terme *abiogénèse*) qui voit se former spontanément de nouvelles espèces à partir de matières minérales en décomposition. Soutenue jusqu'au XVIIe siècle par de grands scientifiques comme Newton ou Descartes, ainsi que par le médecin-alchimiste Jan Baptist Van Helmont qui a prétendu obtenir des souris avec des grains de blé et de la sueur humaine, cette théorie n'a réellement pu être infirmée qu'avec Pasteur vers 1860. Le microbiologiste français a en effet démontré expérimentalement que même les micro-organismes ne pouvaient provenir d'une génération spontanée.

1.1.2 Le créationnisme

Durant les deux mille ans qui suivirent les premiers écrits d'Aristote, le créationnisme et le fixisme sont les théories qui prévalent et unissent alors la plupart des savants. Cette doctrine est basée sur la croyance selon laquelle l'univers a été créé par Dieu. Les espèces vivantes aussi, lors de la Génèse, de manière indépendante et permanente. Ainsi, les espèces sont immuables, ce qui rejette toute possibilité d'évolution (fixisme). Carl von Linné, naturaliste suédois très influent du XVIIIe siècle, fils de pasteur et fixiste convaincu, publie treize éditions de son *Systema Naturae* [von Linné, 1735 1770]. L'ouvrage est ambitieux ; bien que ses 3 000 pages de classifications des espèces recèlent quelques libertés, il constitue la base de la classification scientifique contemporaine des espèces[2]. Cette classification systématique des êtres vivants est devenue une branche bien à part de la biologie, appelée taxonomie par le botaniste Augustin Pyrame de Candolle [de Candolle, 1813], puis taxinomie (du grec *taxis* : rangement et *nomos* : loi) par Emile Littré (en 1842), médecin reconverti philologue. Ce classement, arborescent mais pas généalogique, a survécu à l'avènement des théories évolutionnistes.

[1] Un tiers des écrits du philosophe concerne la biologie. Dans l'ouvrage *Des Parties des Animaux* [Aristote, IVe s av J C] il établit une classification par genres et espèces qui perdurera jusqu'à Buffon.

[2] Dès la première édition du *Systema Naturae*, Linné désigne les espèces par un nom de genre suivi d'un nom d'espèce. Cette nomenclature binomiale est toujours utilisée.

1.1.3 Le transformisme

Au milieu du XVIIIe siècle, le naturaliste français Buffon, scientifique et écrivain de grand talent, envisage dans son *Histoire naturelle, générale et particulière* (36 volumes) la possibilité d'une généalogie commune entre l'Homme et le singe. Pourtant partisan de la génération spontanée, il envisage la transformation des espèces existantes selon trois facteurs : le climat, la nourriture et la domesticité. Le mathématicien Maupertuis, en revanche, introduit les hypothèses de variations héréditaires et de sélection [de Maupertuis, 1745].

Plus aboutie, la thèse de Lamarck soutient que les espèces se transforment au cours du temps en s'adaptant aux variations de leur milieu. Il explique ainsi son principe d'adaptation dans [de Lamarck, 1809] :

> « *Ce n'est point la forme du corps, soit de ses parties, qui donne lieu aux habitudes, à la manière de vivre des animaux ; mais que ce sont au contraire les habitudes, la manière de vivre et toutes les circonstances influentes qui ont avec le temps constitué la forme des animaux.* »

Pionnier de l'évolutionnisme, Lamarck est surtout célèbre pour son principe des caractères acquis, hypothèse — avérée fausse — qui explique son idée d'évolution ; selon lui, les modifications (imperceptibles à l'échelle de l'individu) des organismes en fonction de leur environnement, sont transmissibles.

Si les théories de Lamarck n'eurent pas grande crédibilité au début du XIXème siècle, c'est que le très respecté scientifique Cuvier s'y est violemment opposé. Malgré ses découvertes de fossiles d'animaux disparus [Cuvier, 1812], Cuvier restait convaincu par le créationnisme et formule le catastrophisme : une création divine remplace les espèces disparues après chaque catastrophe anéantissant toute vie sur Terre.

1.1.4 Le Darwinisme

Né en 1809, Darwin commence à l'âge de seize ans des études de Médecine à Edimbourg. Son manque de motivation le pousse à migrer à Cambridge afin de devenir pasteur. Au fil de rencontres, Darwin va finalement se retrouver à bord du *Beagle*, un navire de recherche destiné à améliorer les relevés des côtes de Patagonie. On dit traditionnellement que Darwin était à bord de ce navire en temps que naturaliste, mais le *Beagle* en possédait déjà un. En fait, le capitaine recherchait plus une compagnie intéressante pour passer de nombreuses années en mer. Darwin pouvait ainsi se laisser aller à ses recherches.

De 1831 à 1836, Charles Darwin va ainsi recueillir une somme considérable d'informations géologiques et biologiques. Durant ces cinq années d'observation, notamment au Cap-Vert, en Amérique du Sud, à Tahiti, en Australie, aux Açores, Darwin recense et classe de nombreuses variétés de fossiles et d'espèces vivantes.

Dès son retour, Darwin va soumettre les théories de son époque à ses observations. Dans un premier temps, ses observations géologiques vont confirmer celles de certains de ses contemporains qui contestent le catastrophisme, comme Sir Charles Lyell [Lyell, 1830 1833] ou surtout Alfred Russel Wallace qui a développé indépendamment la même théorie que Darwin sur l'évolution des espèces. Leur rencontre en 1858 a incité Darwin à publier

rapidement son ouvrage *Sur l'origine des espèces* [Darwin, 1859], qui fut épuisé dès le premier jour.

Dans son ouvrage, Darwin remet intégralement en cause la fixité des espèces, en notant que les fossiles d'espèces éteintes ressemblent en grande partie à certaines espèces vivantes. De plus, il remarque que les variations entre espèces provenant d'iles différentes sont minimes. De ces observations, Darwin en conclut que les espèces ne sont pas fixes mais se modifient. Pour lui, la compétition entre les jeunes de chaque espèce pour survivre entraine une sélection naturelle : les survivants engendreront une génération qui possède les caractéristiques naturelles ayant permis à leurs ancêtres de survivre. D'après Darwin, la sélection naturelle est si rigoureuse que la moindre variation utile fait triompher la lignée qui la possède.

C'est véritablement depuis cet ouvrage que nait la notion d'arbre phylogénétique — même si le terme *phylogénie* n'apparait pour la première fois que sept ans plus tard [Haeckel, 1866]). En effet, d'après [Darwin, 1859], toutes les espèces vivantes sont originaires d'une seule forme de vie à travers un processus de branchement :

> « *Les espèces qui appartiennent à ce que nous appelons le même genre descendent directement de quelque autre espèce ordinairement éteinte.* »

La plupart des scientifiques de l'époque vont dénigrer Darwin en remarquant qu'il est incapable de prouver sa théorie. Cependant, les lois de Mendel (1822-1884), le fondateur de la génétique moderne, iront dans le sens de la théorie de la sélection naturelle. Bien sûr, les hommes d'Église — entre autres — continueront de s'insurger contre ces théories qui vont à l'encontre des Écritures sur la création de l'Homme. L'évolution place en effet l'Homme au niveau de l'animal et, surtout, le fait descendre du singe. Darwin avait d'ailleurs anticipé le tracas idéologique que cette théorie engendrerait [Darwin, 1871] :

> « *La conclusion fondamentale à laquelle nous sommes arrivés dans cet ouvrage, à savoir que l'homme descend de quelque forme d'organisation inférieure, sera, je le regrette de le penser, fort désagréable à beaucoup de personnes.* »

1.2 Les arbres phylogénétiques

Après cette brève introduction historique, nous décrivons dans cette section les arbres phylogénétiques et les concepts de bases qui leur sont associés.

1.2.1 Théorie des graphes : définitions préliminaires

Il s'agit ici de rappeler au fil de définitions simples ce qu'est un arbre au sens mathématique du terme. Nous incluons d'autres définitions et propriétés de base qui seront abordées par la suite [Berge, 1958 ; Essam and Fisher, 1970].

Définition 1 Graphe : *Objet mathématique composé de points appelés* sommets *ou* nœuds*, et de lignes appelées* arêtes *connectant un ensemble de ces sommets.*
Un graphe (non orienté) se définit alors par un couple $(\mathcal{V}, \mathcal{E})$, où $\mathcal{V}$ est un ensemble de sommets et $\mathcal{E} \subseteq \mathcal{V}^2$ est un ensemble de paires de sommets (arêtes).

Graphes : définitions annexes

Etiquette : nom, valeur que l'on attribue à un sommet ou une arête.
Degré d'un sommet : nombre d'arêtes connectées à ce sommet.
Chaine : suite de sommets reliés par des arêtes.
Cycle : chaine dont les extrémités coïncident.
Connexité : un graphe est *connexe* si chaque paire de sommets est reliée par une chaine.

Définition 2 Arbre : *Graphe connexe sans cycle.*

Lorsque l'on travaille sur des arbres, on emploie plus usuellement le terme *nœud* que *sommet*. De même, on peut également désigner ses arêtes par des *branches*.

Arbres : définitions annexes

Feuille : dans un arbre, nœud de degré 1.
Arbre binaire : arbre dont chaque nœud est de degré 3 au maximum.
Arbre enraciné : arbre possédant un nœud distingué appelé *racine*.
Arbre binaire entier : arbre binaire dont au maximum un nœud (la racine, suivant que l'arbre soit enraciné ou non) est de degré 2.

L'enracinement d'un arbre entraine une hiérarchie entre les sommets séparés par des arêtes. Cela revient à orienter l'arbre ; dans ce cas on parle de *chemins* plutôt que de *chaines*.

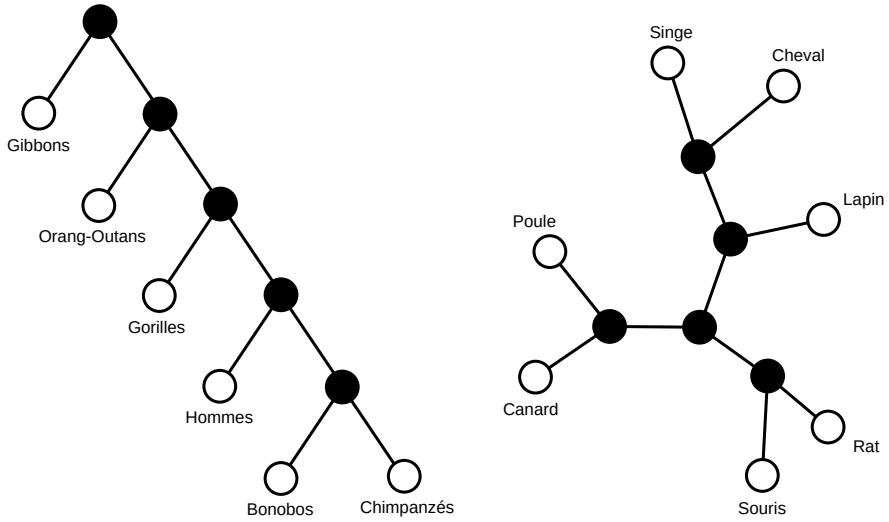
Définition 3 Ascendant et descendants d'un nœud : *Dans un arbre enraciné, l'ascendant d'un nœud E est le nœud relié à E par une arête et appartenant au chemin menant de E vers la racine. Notons que la racine n'a aucun ascendant. L'ascendant est également appelé* parent *de E.*
Les descendants de E sont les nœuds dont E est l'ascendant. Là encore, on parle plus communément de fils *plutôt que de descendants.*

Par la suite et sauf mention contraire, tous les arbres considérés seront des arbres binaires entiers. En informatique, les arbres binaires sont une structure de données à part entière, et ils possèdent toujours une racine. En reconstruction phylogénétique, on pourra au contraire aussi bien utiliser des arbres enracinés ou non, en fonction des méthodes de reconstruction.

1.2.2 Types d'arbres phylogénétiques et taxons

Un arbre phylogénétique est censé montrer les relations évolutionnaires d'un ensemble d'espèces. À quelques détails près, un arbre phylogénétique est un arbre généalogique. Les différences notables sont que :

1. il est construit à partir de groupes, d'espèces, et non d'individus,
2. il exprime les relations de parenté et non de descendance ; c'est-à-dire que l'identification des ascendants importe moins que la parenté relative de deux espèces contemporaines, et



Arbre enraciné Arbre non enraciné

FIG. 1.1 – Exemples d'arbres phylogénétiques enracinés ou non

3. il infère le passé et en ce sens se pose comme hypothèse.

Les données d'un arbre phylogénétique se trouvent aux feuilles, tandis que les nœuds internes et les arêtes de l'arbre sont en quelque sorte les inconnues. En phylogénie, les données, qui donnent un sens à chaque feuille des arbres, sont des *taxons*.

Définition 4 Taxon [Simpson, 1961] : *Groupe d'organismes reconnu en tant qu'unité formelle à chacun des niveaux de la classification.*

Dans la littérature, nous pouvons parfois trouver le concept de OTU (*Operational Taxonomic Unit* [Sokal and Sneath, 1963]), qui exprime une unité pragmatique soumise à l'investigation. Le terme associé HTU (*Hypothetical Taxonomic Unit* [Farris, 1970]) désigne lui un ancêtre hypothétique de plusieurs OTU, inféré par un arbre phylogénétique. D'autres termes ont depuis été définis (unités évolutives [Meacham, 1984], taxons liminaux [Dupuis, 1988], *etc.*), n'aidant pas le lecteur informaticien à une compréhension rapide de la spécialité. Ici, nous parlerons simplement de *feuilles* lorsque le problème sera abstrait de ces considérations biologiques, voire de *taxons* ou d'*espèces* si nous tenons à donner un sens ou une identité aux données.

Il existe ensuite plusieurs variétés d'arbres. Tout d'abord, les *dendrogrammes* expriment uniquement des liens entre les taxons. Les branches ne sont pas étiquetées, donc leur longueur n'a pas de signification biologique ; c'est le cas des arbres de la figure 1.1. Les *cladogrammes*, *phénogrammes* et *phylogrammes* sont des dendrogrammes particuliers, où les nœuds et les arêtes expriment des données propres. Ces trois types d'arbres sont générés par les trois différents critères de reconstruction phylogénétique, que nous détaillerons dans la prochaine section.

En fonction du type d'arbre utilisé, la racine peut avoir une signification biologique ou non. Intuitivement, elle représente l'ancêtre commun hypothétique à toutes les espèces considérées, mais en fonction de la méthode employée, elle peut n'avoir qu'un but pratique, par exemple pour une meilleure lecture de l'arbre.

1.2.3 Les caractères

L'analyse phylogénétique est basée sur les différences entre organismes. Pour cela, le biologiste doit préalablement identifier des caractères sur lesquels fonder l'analyse. Ces caractères (attributs observables) peuvent ensuite être instanciés par différents états.

On distingue plusieurs types de caractères qui permettent de différencier les espèces [Huxley, 1940]. Parmi eux, la différence morphologique et la distinction génétique sont les plus utilisés. Les données sur lesquelles s'appuyer sont alors une liste d'états de caractères, discrets et/ou continus. Les *séquences* de nucléotides ou d'acides aminés en sont un exemple, tout comme les séquences binaires, composées de caractères pouvant prendre comme valeur deux états seulement (présence / absence).

Exemple : caractères binaires

Alismatanae	10111110
Aranae	11110000
Liliales	11100011
Asparagales	10111001
Pandanales	01001010
Dioscoreales	10001011

Ces ensembles de séquences peuvent être représentés par une matrice où chaque élément est un caractère prenant sa valeur dans un ensemble d'états Σ appelé alphabet. Dans le cas de séquences binaires, $\Sigma = \{0, 1\}$; pour les séquences ADN (de nucléotides), $\Sigma = \{A, C, G, T\}$ où A représente l'adénine, C la cytosine, G la guanine et T la thymine. En utilisant cette représentation, nous définissons un *site* comme un vecteur colonne ; le k-ième site est la réunion des k-ièmes caractères des séquences. En outre, ces *séquences* peuvent se définir simplement comme des mots de même longueur m, soit des éléments de Σ^m.

Nous verrons dans la section suivante que toutes les méthodes de reconstruction phylogénétique ne sont pas fondées sur des ensembles de séquences de caractères. Certaines méthodes se basent en effet sur des matrices de distances exprimant la similarité ou disimilarité entre chaque couple d'espèces considérées.

1.2.4 Nombre d'arbres

À partir d'un ensemble d'espèces actuelles (taxons), et si nous acceptons le postulat selon lequel elles dérivent toutes d'un ancêtre commun, il n'existe qu'un arbre phylogénétique *vrai*, qui retrace avec exactitude l'histoire évolutive de ces espèces. Bien sûr, nous ne connaissons pas cet arbre, et c'est par hypothèse que nous allons tenter de le reconstruire.

Nous proposons en premier lieu de regarder combien d'arbres phylogénétiques distincts sont envisageables, en considérant un ensemble de n taxons. Ce problème de dénombrement d'arbres a 150 ans ; on le trouve dans [Cayley, 1857].

Par commodité, commençons par dénombrer les arbres non enracinés. Si $n = 2$, il n'existe qu'un arbre non enraciné possédant 2 feuilles (Figure 1.2). À partir de cet arbre, il n'y a qu'un endroit où insérer une troisième espèce, donc il n'existe également qu'un seul arbre non enraciné pour $n = 3$. Chaque arête de l'arbre est triviale au sens où elle ne fait que séparer chaque espèce des deux autres.

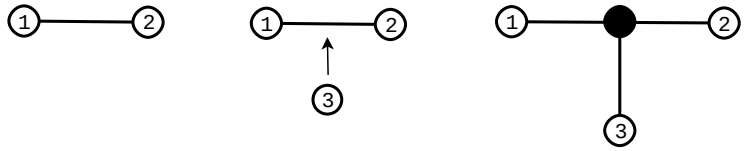


Fig. 1.2 – Arbres triviaux à deux ou trois taxons

Il existe trois arbres non enracinés envisageables lorsque $n = 4$. Considérant quatre taxons numérotés de 1 à 4, un arbre non enraciné ne fera que les partitionner en deux groupes de deux. Les trois partitionnements possibles sont $\{\{1,2\},\{3,4\}\}$, $\{\{1,3\},\{2,4\}\}$ et $\{\{1,4\},\{2,3\}\}$. On le remarque sur la figure 1.3 lorsque l'on insère un quatrième taxon à l'unique arbre composé de trois feuilles. Dans ce cas, l'arête interne (non triviale) de l'arbre symbolise la bipartition. Précisons que l'ordre dans lequel les taxons sont considérés n'a pas d'incidence ; le résultat aurait été identique si, par exemple, les taxons numérotés de 2 à 4 avaient été insérés avant le premier.

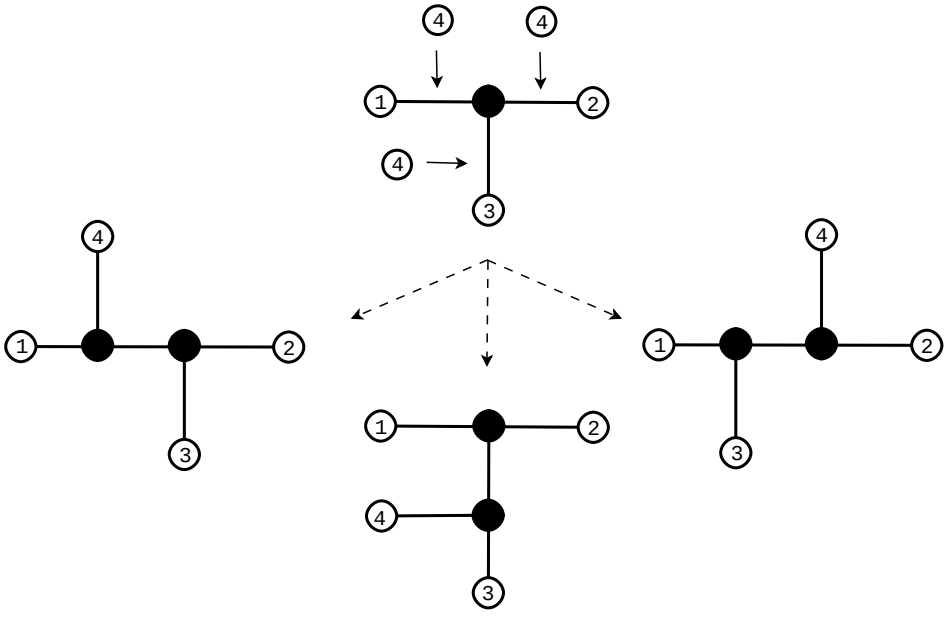


Fig. 1.3 – Différents placements possibles pour un quatrième taxon

À partir d'un arbre de taille 4 (*i.e.* comportant quatre feuilles), un cinquième taxon

peut être inséré à cinq endroits différents, car l'arbre de taille 4 comporte cinq arêtes. Il existe donc cinq arbres distincts générés à partir de chaque arbre de taille 4. Pour affirmer que le nombre d'arbres de taille 5 est égal à 15 (cinq arbres générés à partir de chacun des trois arbres de taille 4), il suffit de montrer qu'un même arbre ne puisse pas être généré à partir de deux arbres distincts de taille inférieure. Or si l'on ôte un taxon (ou un sous-ensemble de taxons) à un arbre, l'arbre obtenu sera unique. Par conséquent, il existe bien quinze arbres non enracinés de taille 5, et cette méthode de calcul peut être généralisée de manière récurrente [Cavalli-Sforza and Edwards, 1967] :

Lemme 1 *Un arbre binaire entier non enraciné de taille $n \geqslant 2$ possède $2n-3$ arêtes.*

Propriété 1 *Si τ_n est le nombre d'arbres non enracinés de taille n, alors $\tau_{n+1} = \tau_n(2n-3)$. Il en découle [Schröder, 1870 ; Stanley, 1997] :*

$$\tau_n = \prod_{i=3}^{n}(2i-5) = \frac{(2(n-1))!}{2^{n-1}(n-1)!}$$

La racine est un nœud additionnel qui sépare une arête en deux. Ainsi, il existe autant de versions enracinées d'un arbre non enraciné que d'arêtes dans cet arbre. Il vient :

Propriété 2 *Si $\dot{\tau}_n$ est le nombre d'arbres enracinés de taille n, alors $\dot{\tau}_n = \tau_n(2n-3)$, soit :*

$$\dot{\tau}_n = \prod_{i=2}^{n}(2i-3)$$

En fait, que ce soit pour y placer une racine ou bien pour y insérer un nouveau taxon, le nombre de choix d'arêtes possibles reste le même : $\dot{\tau}_n = \tau_{n+1}$ (ce qui apparait immédiatement en corollaire des propriétés 1 et 2).

Il est possible d'obtenir une approximation asymptotique de ce nombre en utilisant la formule de Stirling :

$$\dot{\tau}_n \sim \sqrt{2}\,.\,[2e^{-1}(n-1)]^{n-1}$$

Cela nous permet de calculer le nombre d'arbres phylogénétiques envisageables en fonction du nombre de taxons (tableau 1.1).

1.3 Les méthodes de reconstruction phylogénétique

Une analyse phylogénétique vise à reconstruire l'histoire évolutive d'un ensemble d'espèces. Savoir comment des espèces ont dérivé durant l'évolution, en considérant uniquement leurs caractères respectifs actuels, est un problème complexe. Sur quels critères se baser ? Quelle connaissance avons-nous des processus d'évolution ? Est-il possible de formaliser de manière exacte l'ensemble des mécanismes évolutifs ? Comme ces questions amènent souvent plusieurs réponses, il n'est pas étonnant de constater qu'il existe plusieurs critères de reconstruction phylogénétique, qui se basent parfois sur des types de

Taxons	Arbres non enracinés	Arbres enracinés	Taxons	Arbres non enracinés	Arbres enracinés
1	1	1	40	$1,31.10^{55}$	$1,01.10^{57}$
2	1	1	50	$2,84.10^{74}$	$2,75.10^{76}$
3	1	3	60	$5,01.10^{94}$	$5,86.10^{96}$
4	3	15	80	$2,18.10^{137}$	$3,43.10^{139}$
5	15	105	100	$1,70.10^{182}$	$3,35.10^{184}$
6	105	945	200	$3,19.10^{428}$	$1,27.10^{431}$
7	945	10 395	500	$1,01.10^{1277}$	$1,01.10^{1280}$
8	10 395	135 135	1 000	$1,93.10^{2860}$	$3,85.10^{2863}$
9	135 135	2 027 025	1 500	$2,74.10^{4557}$	$8,20.10^{4560}$
10	2 027 025	34 459 425	2 000	$3,00.10^{6328}$	$1,20.10^{6332}$
15	$7,91.10^{12}$	$2,13.10^{14}$	3 000	$1,46.10^{10024}$	$8,76.10^{10027}$
20	$2,20.10^{20}$	$8,20.10^{21}$	4 000	$3,36.10^{13867}$	$2,69.10^{13871}$
25	$2,54.10^{28}$	$1,19.10^{30}$	5 000	$4,77.10^{17820}$	$4,77.10^{17824}$
30	$8,69.10^{36}$	$4,95.10^{38}$	10 000	$8,01.10^{38658}$	$1,60.10^{38663}$

Tab. 1.1 – Nombre d'arbres en fonctions du nombre de taxons

données différents. Et même si les processus évolutifs obéissent à certaines règles que les spécialistes tentent de modéliser au mieux, le hasard de la nature fait que dans la plupart des cas, plusieurs explications d'une même observation peuvent être envisagées.

N'étant ni biologiste ni philosophe, nous ne prendrons aucunement parti sur les différentes méthodes de reconstruction. Nous nous efforcerons ici de les présenter de manière simple et concise, et d'en dégager les aspects combinatoires.

1.3.1 L'analyse phénétique

Le concept de phénétique, ou *taxinomie numérique* est apparu dans [Sokal and Sneath, 1963]. Pour classer les espèces, Sokal et Sneath privilégient les notions de ressemblance et de similitude. Arguant qu'à cette époque, les connaissances évolutives étaient trop superficielles, ils ont basé leurs classifications sur les caractères observables des espèces. Ainsi, les espèces ayant le plus de similitudes, c'est-à-dire le plus de caractères en commun, seront regroupées.

Ce principe statistique induit que le plus grand nombre possible de caractères doit être pris en compte pour établir des indices de similarité entre espèces. Ces caractères (voir section 1.2.3), qu'ils soient continus (dimensions, fréquence allélique) ou discrets (nombre de pattes, présence ou absence d'un organe, *etc.*), sont utilisés pour le calcul d'indices de similitude globale. À partir d'une matrice de similarités (distances) composée des indices de similitude entre chaque paire d'espèces, on applique un algorithme de classification (*clustering*) qui regroupe itérativement les espèces les plus proches, jusqu'à obtenir une classification hiérarchique. La méthode de *clustering* (ou méthode de distances) la plus célèbre, qui dépasse d'ailleurs le simple cadre de la reconstruction phylogénétique,

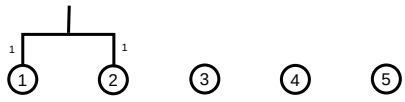
est l'algorithme UPGMA (*Unweighted Pair Group Method using Arithmetic Mean*) ; ses fondements remontent à [Sokal and Michener, 1957]. Un dendrogramme produit par la taxinomie numérique est un *phénogramme*.

Nous proposons d'illustrer l'algorithme UPGMA par un exemple simple. Pour cela, considérons un ensemble de cinq taxons $\mathfrak{T} = \{T_1, T_2, T_3, T_4, T_5\}$ et la matrice de distances associée suivante :

	T_1	T_2	T_3	T_4	T_5
T_1	0				
T_2	2	0			
T_3	6	5	0		
T_4	10	9	4	0	
T_5	9	8	5	3	0

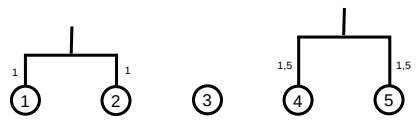
D'après la matrice, les deux espèces les plus proches sont T_1 et T_2, distants de 2 unités. Nous les regroupons pour former la classe (*cluster*) (T_1, T_2). T_1 et T_2 sont alors supprimés de $\mathfrak{T}$ et donc de la matrice, remplacés par le cluster (T_1, T_2), noté$T_{1,2}$. Nous obtenons la distance entre $T_{1,2}$ et chacun des autres taxons en calculant la moyenne arithmétique des deux distances entre ces taxons et T_1 et T_2. La distance entre $T_{1,2}$ et T_3, $d(T_{1,2}, T_3)$, est ainsi égale à $(d(T_1, T_3) \times 1 + d(T_2, T_3) \times 1)/(1 + 1) = (6 + 5)/2 = 5,5$. Chaque cluster ajouté dans $\mathfrak{T}$ est à ce stade considéré de la même manière qu'un taxon.

	T_{12}	T_3	T_4	T_5
$T_{1,2}$	0			
T_3	5,5	0		
T_4	9,5	4	0	
T_5	8,5	5	3	0



Les deux taxons / clusters les plus proches, $\arg\min_{(X,Y)\in\mathfrak{T}^2} d(X,Y)$ $(X \neq Y)$[3], sont à ce stade T_4 et T_5, avec $d(T_4, T_5) = 3$. Nous les agglomérons en un cluster (T_4, T_5), soit $T_{4,5}$, puis on calcule la matrice de distances des taxons / clusters restants : $\mathcal{T} = \{T_{1,2}, T_3, T_{4,5}\}$.

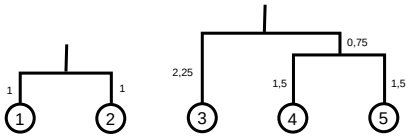
	T_{12}	T_3	T_{45}
$T_{1,2}$	0		
T_3	5,5	0	
$T_{4,5}$	9	4,5	0



On regroupe ici T_3 avec T_{45}, le nouveau cluster est $T_{3,4,5}$. D'après l'algorithme UPGMA, $d(T_{1,2}, T_{3,4,5}) = (d(T_3, T_{1,2}) \times 1 + d(T_{4,5}, T_{1,2}) \times 2)/(1 + 2) = 7,833$ car la moyenne est pondérée par le nombre de taxons contenus dans chacun des deux clusters (1 s'il s'agit d'un simple taxon) nouvellement regroupés.

[3] $\arg\min_{(X,Y)\in\mathfrak{T}^2} d(X,Y) = \{(\bar{X}, \bar{Y}), d(\bar{X}, \bar{Y}) = \min_{(X,Y)\in\mathfrak{T}^2} d(X,Y)\}$

	$T_{1,2}$	T_{345}
T_{12}	0	
$T_{3,4,5}$	7,833	0



Il ne reste plus que deux clusters à agglomérer, ce qui nous amène à l'arbre final obtenu avec l'algorithme UPGMA (Fig. 1.4). Dans ce phénogramme, les longueurs des branches sont proportionnelles aux distances calculées, et il se vérifie aisément que dans ce cas, tous les taxons sont à égale distance de la racine.

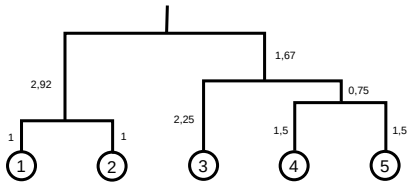


Fig. 1.4 – Exemple de dendrogramme obtenu par l'algorithme UPGMA

L'algorithme UPGMA, le U signifiant *non pondéré* (*Unweighted*), utilise une moyenne pondérée afin que les taxons ou clusters nouvellement ajoutés ne pèsent pas plus que leur poids réel dans le calcul de la distance. Par exemple, si le simple taxon X est distant de 10 unités d'un taxon Y, et de 20 unités *en moyenne* d'un agglomérat $\mathfrak{A}$ de 100 taxons, alors sa distance moyenne avec le cluster formé par $\mathfrak{A}$ et Y devra être très proche de 20. Or si nous ne pondérons pas les distances lors du calcul de la moyenne, la distance entre X et le reste des taxons chutera à 15. Dans ce cas, le calcul n'utilise aucune pondération, mais la moyenne est bien pondérée, puisque le taxon Y pèse alors beaucoup plus que chacun des 100 taxons de $\mathfrak{A}$ dans la moyenne résultante. L'algorithme utilisant ce calcul est appelé WPGMA (*Weighted Pair Group Method using Arithmetic Mean*), le W signifiant *pondéré*. Cette petite subtilité échappe à certains auteurs, et l'on trouve fréquemment des inversions de dénomination entre UPGMA et WPGMA. Il existe également deux variantes, UPGMC et WPGMC (*(Un)Weighted Pair Group Method using the Centroid average*), qui calculent les moyennes géométriques (barycentres) plutôt qu'arithmétiques.

Nous pouvons trouver une présentation de ces quatre algorithmes de regroupement par moyenne dans [Sneath and Sokal, 1973], bien qu'une première méthode de ce type ait été utilisée 15 ans auparavant [Michener and Sokal, 1958]. Cependant, ces méthodes de clustering ne sont pas propres à l'analyse phylogénétique et sont très employées en statistique. D'autres algorithmes utilisent pour le recalcul de la matrice le minimum ou le maximum (*single-link*, *complete-link*) [Johnson, 1967], ou encore la variance minimale [Ward, 1963]. Et d'un point de vue historique, le premier algorithme de clustering est à mettre au crédit du psychologue Robert Choate Tryon [Tryon, 1939].

Les méthodes phénétiques ne sont pas incompatibles avec les données génétiques, bien au contraire. Devant l'essor commun de la génomique et de l'informatique, de grandes

classifications peuvent être établies à partir de séquences ADN des espèces. Par exemple, la *distance* entre deux séquences ADN de même longueur peut être définie comme étant leur nombre de caractères différents (*distance de Hamming*).

Exemple : Transformation d'une matrice de caractères en matrice de distances

T_1 - Circus	CACACTATTCCGCAGACACTACCCTGGCTTTCTCATCCGT
T_2 - Aquila	TACACTACACGGCAGACACCACCCTAGCCTTCTCATCCGT
T_3 - Butastus	TACACTACACCGCAGACACCACCCTAGCCTTTTCATCAGT
T_4 - Accipiter	TACACTACACCGAAGACACTACCCTAGCCTTTTCATCAGT

En comptabilisant les différences entre les séquences deux à deux, la matrice de distances issue de ces données est la suivante :

	T_1	T_2	T_3	T_4
T_1	0			
T_2	7	0		
T_3	8	3	0	
T_4	8	5	2	0

Il existe bien sûr des méthodes plus fines et fiables pour transformer un ensemble de séquences en matrice de distances, prenant en compte des modèles évolutifs (voir section 1.3.3).

Bien qu'elles soient simples à implémenter et très rapides, les méthodes de distances souffrent généralement de la plus faible information contenue dans les matrices de distances. Les algorithmes les plus communément utilisés en analyse phénétique sont ceux des moindres carrés [Cavalli-Sforza and Edwards, 1967 ; Rzhetsky and Nei, 1992], du *Minimum-Evolution* [Rzhetsky and Nei, 1993] et du *Neighbor-Joining* (`NJ` [Saitou and Nei, 1987] puis `BIONJ` [Gascuel, 1997]).

1.3.2 L'analyse cladistique

Depuis une cinquantaine d'années, un nouveau mode de classification divise les spécialistes. *Phylogenetic systematic* [Hennig, 1966] est un ouvrage clé dans l'histoire de la systématique. Hennig propose de reconstituer le scénario évolutif des espèces afin d'élucider leur relation de parenté. C'est alors l'analyse qualitative des caractères qui doit permettre d'identifier les états primitifs des états dérivés (on trouve aussi états *plésiomorphes* et *apomorphes* [Tassy, 1986]). Lorsqu'une apomorphie est partagée par plusieurs taxons (on parle alors de *synapomorphie*), ces derniers sont regroupés sous la forme d'un *groupe monophylétique* (figure 1.5). En revanche, la présence d'*homoplasies* (substitutions parallèles, apparaissant de manière indépendante) peut conduire à de fausses pistes.

Outre ces quelques considérations introductives, nous allons faire l'impasse sur le vocabulaire complexe et étendu de la cladistique, qui trouve un intérêt plus large d'un point de vue biologique que combinatoire. Cependant, le lecteur intéressé peut trouver ces informations dans l'ouvrage [Darlu and Tassy, 1993].

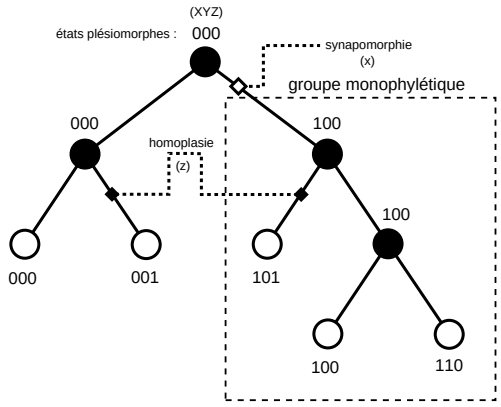
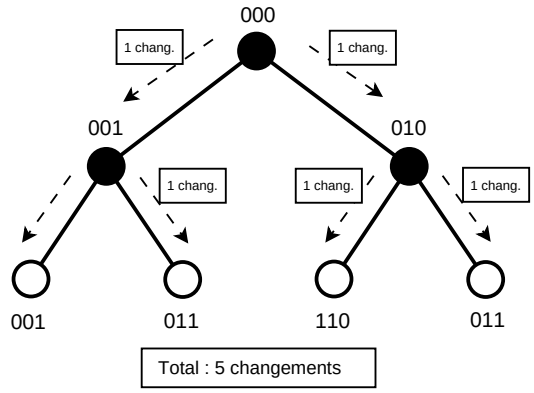


FIG. 1.5 – Constitution d'un groupe monophylétique

Dans l'esprit de la cladistique, un arbre est plus valable qu'un autre si le nombre de changements d'état observés est inférieur. La méthode du **maximum de parcimonie**, qui est la méthode cladistique la plus utilisée, consiste à rechercher, à partir de séquences de caractères d'un ensemble d'espèces qui seront placées aux feuilles, l'arbre qui minimise ce nombre de changements d'état. Un avantage à cette méthode est de permettre la reconstitution des états pris par les espèces ancestrales, pour chaque caractère. Un *cladogramme* est un arbre résultant d'une analyse cladistique.

Exemple : Comptabilisation des changements d'état
Soient quatre taxons dont les chaines de caractères binaires sont respectivement 001, 011, 110 et 011, ainsi qu'un cladogramme regroupant ces taxons. Nous reconstituons de manière assez arbitraire les séquences associées aux nœuds internes de l'arbre, puis comptabilisons le nombre de changements d'état total.



Le problème Maximum de Parcimonie fait l'objet de cette thèse et sera détaillé plus précisément dans le chapitre suivant.

1.3.3 L'analyse probabiliste

Les phylogénies proposées par une analyse cladistique des caractères ne prennent en compte aucune information biologique, aucune connaissance *a priori* sur l'évolution de ces caractères. L'arbre phylogénétique le plus parcimonieux est celui qui minimise le nombre de changements évolutifs survenus au cours de l'évolution. Cela implique que l'on pose comme hypothèse que la probabilité de mutation est constante et à tout instant très faible. Si nous considérons des espèces relativement éloignées les unes des autres, une analyse cladistique ne prendra pas en compte la possibilité que des lignées évoluent plus rapidement que d'autres alors que c'est probablement le cas. Un autre paradigme est de reconstruire l'arbre phylogénétique le plus probable en fonction des données. Dans ce cas, les données regroupent non seulement les séquences de caractères associées aux espèces considérées, mais aussi un modèle mathématique traduisant plus ou moins fidèlement les connaissances évolutives que l'on souhaite considérer.

En statistique, estimer la probabilité qu'une hypothèse soit fidèle à l'observation revient à calculer sa *vraisemblance*. Le problème *Maximum de Vraisemblance*, qui consiste à trouver l'hypothèse la plus probable en fonction des données observées, a été énoncé au début du siècle dernier au fil de trois publications [Fisher, 1912 ; Fisher, 1921 ; Fisher, 1922]. D'après [Aldrich, 1997], il s'agit d'une des plus importantes contributions du XXème siècle dans le domaine des statistiques.

La première application de cette méthode (ML pour *Maximum Likelihood*) à la reconstruction phylogénétique est à mettre au crédit de [Edwards and Cavalli-Sforza, 1964], malgré de nombreux problèmes d'implémentation. C'est finalement Neyman, statisticien pourtant peu convaincu par les méthodes de vraisemblance, qui appliqua le premier cette technique à des données moléculaires [Neyman, 1971]. Enfin, le premier algorithme de reconstruction d'arbres phylogénétiques par maximum de vraisemblance à partir de séquences de nucléotides a été réalisé par [Felsenstein, 1981].

Pour comprendre le calcul de la vraisemblance d'un arbre phylogénétique T en fonction des données (un alignement de séquences $\mathcal{A}$ et un modèle de l'évolution $\mathfrak{M}$), nous procéderons par étapes.

Modèle de l'évolution

L'estimation de la vraisemblance d'une phylogénie est dépendante d'un paramètre $\mathfrak{M}$, qui décrit de quelle manière les caractères évoluent. Nous considérons ici que les données sont des séquences ADN, *i.e.* $\Sigma = \{A, C, G, T\}$. Plusieurs modèles de l'évolution ont été définis affectant à chaque changement d'état éventuel $i \to j, (i, j) \in \Sigma^2$, un taux d'apparition. Le modèle le plus simple, que nous allons utiliser pour l'exemple, est celui de Jukes et Cantor [Jukes and Cantor, 1969], où tous les changements d'état sont équiprobables. La matrice de substitution $\mathcal{R}$ de ce modèle est représentée de cette manière :

$$\mathcal{R} = \begin{pmatrix} \bullet & \alpha & \alpha & \alpha \\ \alpha & \bullet & \alpha & \alpha \\ \alpha & \alpha & \bullet & \alpha \\ \alpha & \alpha & \alpha & \bullet \end{pmatrix}$$

Le paramètre α signifie que tous les changements d'état ont le même taux d'apparition, à savoir la probabilité $\alpha.\varepsilon$ sur une durée infinitésimale ε. Définissons alors la matrice Markovienne $P(t)$ associée, telle que $P_{ij}(t)$ représente la probabilité que l'état i soit transformé en l'état j après t unités de temps : $P_{ij} = p(j|i,t)$.

Nous avons en premier lieu la relation suivante :

$$\exists \varepsilon_0 \text{ tel que } \forall \varepsilon < \varepsilon_0, P_{ij}(\varepsilon) = \begin{cases} \alpha.\varepsilon, \text{ si } i \neq j \\ 1 - 3\alpha.\varepsilon, \text{ sinon} \end{cases} \tag{1.1}$$

Comme tous les évènements ont une probabilité d'apparition équivalente, et considérant l'hypothèse selon laquelle tout processus évolutif est réversible, nous avons :

$$\lim_{t \to \infty} P_{ij}(t) = \frac{1}{4}, \forall (i,j) \in \Sigma^2$$

D'après (1.1), il vient $P'_{ij}(0) = \alpha$, et après quelques calculs différentiels, nous pouvons exprimer $P_{ij}(t)$ en fonction de t de manière analytique :

$$P(t) = \begin{pmatrix} p(A|A,t) & p(A|C,t) & p(A|G,t) & p(A|T,t) \\ p(C|A,t) & p(C|C,t) & p(C|G,t) & p(C|T,t) \\ p(G|A,t) & p(G|C,t) & p(G|G,t) & p(G|T,t) \\ p(T|A,t) & p(T|C,t) & p(T|G,t) & p(T|T,t) \end{pmatrix} = \begin{pmatrix} p_0 & p_1 & p_1 & p_1 \\ p_1 & p_0 & p_1 & p_1 \\ p_1 & p_1 & p_0 & p_1 \\ p_1 & p_1 & p_1 & p_0 \end{pmatrix} \tag{1.2}$$

où

$$p_0 = \frac{1}{4}(1 + e^{-4\alpha t}) \text{ et } p_1 = \frac{1}{4}(1 - e^{-4\alpha t})$$

Les $P_{ij}(t)$ expriment des probabilités et vérifient les contraintes suivantes :

$$\sum_{i \in \Sigma} P_{ij}(t) = 1, \forall j \in \Sigma$$

$$\sum_{j \in \Sigma} P_{ij}(t) = 1, \forall i \in \Sigma$$

$$P_{ij}(t) \geqslant 0, \forall i,j \in \Sigma$$

Le modèle de Jukes et Cantor, que nous venons d'introduire, est le plus simple du fait qu'il ne comporte qu'un paramètre α. D'autres modèles plus évolués ont été définis, comme celui de Kimura à deux paramètres [Kimura, 1980] qui définit un taux différent entre les transitions et les tranversions[4], ou plus généralement le modèle GTR à dix paramètres [Lanave *et al.*, 1984 ; Tavaré, 1986]. En fait, une quantité impressionnante de modèles ont été définis, comme ceux de [Felsenstein, 1981 ; Hasegawa *et al.*, 1985 ; Jin and Nei, 1990 ; Tamura, 1992 ; Tamura and Nei, 1993] et leurs variantes. Les défenseurs de l'analyse probabiliste dénoncent souvent le manque de fiabilité des analyses cladistiques, eu égard à l'absence de considération biologique soujacente, alors que les modèles de l'évolution utilisés sont nombreux, multiparamétriques et parfois contradictoires. De plus, un changement d'alphabet entraine inévitablement une reconsidération du modèle [Neyman, 1971 ; Farris, 1973 ; Cavender, 1978 ; Dayhoff *et al.*, 1978 ; Jones *et al.*, 1992].

[4]Une *transition* transforme une purine (adénine, guanine) en une autre purine ou une pyrimidine (cytosine, thymine) en une autre pyrimidine. Une *transversion* modifie une purine en pyrimidine, ou inversement.

Calcul de la vraisemblance

Soient quatre séquences de nucléotides S_1, S_2, S_3 et S_4 de longueur m décrivant respectivement quatre taxons numérotés de 1 à 4, et un arbre phylogénétique associé (figure 1.6).

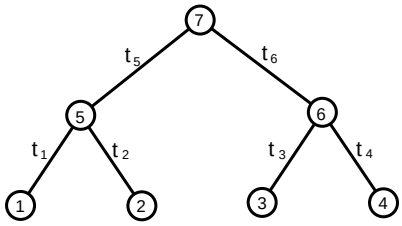


FIG. 1.6 – Pour le calcul de la vraisemblance d'un arbre, on doit inférer la longueur des branches

Comme les sites sont indépendants, considérons pour l'instant uniquement le calcul de la vraisemblance sur le k-ième caractère de chaque séquence. Notons x_i^k, l'état du k-ième caractère de la séquence S_i. Les caractères inconnus des trois ascendants de l'arbre sont notés x_5^k à x_7^k, suivant la numérotation des nœuds internes sur la figure. $p(X|Y,t)$ représente la probabilité qu'un caractère prenne l'état X sachant qu'il prenait l'état Y, t unités de temps auparavant. Sur la figure, les délais sont notés t_1 à t_6 ; par exemple, t_1 représente le temps d'évolution écoulé entre l'ancêtre hypothétique 5 et le taxon 1. Enfin, par souci de lisibilité, notons $\mathfrak{t} = (t_1, t_2, t_3, t_4, t_5, t_6)$ le vecteur d'inconnues à optimiser, et $\mathfrak{p}_i^k$ la probabilité $p(x_i^k|x_{\text{asc}(i)}^k, t_i)$, asc$(i)$ signifiant ascendant de i. Par exemple, $\mathfrak{p}_1^k = p(x_1^k|x_5^k, t_1)$.

La fonction de vraisemblance associée à cet arbre, et pour le k-ième site est donnée par :

$$L^k(\mathfrak{t}) = \sum_{x_5^k \in \Sigma} \sum_{x_6^k \in \Sigma} \left[p(x_5^k|x_6^k, t_5 + t_6). \prod_{i=1}^{4} \mathfrak{p}_i^k \right]$$

Le $p(x_5^k|x_6^k, t_5 + t_6)$ permet de simplifier le problème en utilisant la réversibilité du modèle. Il n'est ainsi pas nécessaire d'inférer la racine de l'arbre, ce qui permet notamment d'économiser une somme et un produit. On en déduit que la position de la racine n'a aucune influence sur la vraisemblance d'un arbre.

Les deux sommes proviennent du fait que x_5 et x_6 sont des inconnues, et qu'il faut prendre en considération tous les cas possibles. Enfin, la vraisemblance de cet arbre étant donné le vecteur de distances $\mathfrak{t}$ est égale au produit des vraisemblances pour chaque site :

$$L(\mathfrak{t}) = \prod_{k=1}^{m} L^k(\mathfrak{t})$$

Pour connaitre la vraisemblance L de l'arbre, il faut déterminer les valeurs t_i telles que $L(\mathfrak{t})$ soit maximale. Comme il s'agit de très petits nombres (produits de probabilités), les

vraisemblances s'expriment sous forme logarithmique. On a finalement :

$$\ln L = \max_{\mathfrak{t}} \ln L(\mathfrak{t}) = \max_{\mathfrak{t}} \sum_{k=1}^{m} \ln L^k(\mathfrak{t})$$

Comme nous pouvons le constater, le calcul de la vraisemblance d'un arbre requiert un effort calculatoire important. Le problème Maximum de Vraisemblance, qui consiste à déterminer l'arbre dont la vraisemblance est la plus élevée, est une méthode de reconstruction phylogénétique très populaire car elle est considérée comme consistante. En revanche, sa complexité est telle que pendant longtemps, les jeux de données de grande taille furent très difficiles à résoudre, même de manière approchée. Récemment, [Guindon and Gascuel, 2003] ont proposé à travers le logiciel `PHYML` une méthode de résolution efficace et dont les temps de calcul sont du même ordre de grandeur que ceux des logiciels d'analyse phénétique ou cladistique.

1.3.4 La fiabilité des arbres

Quelle que soit la méthode basée sur les caractères employée pour reconstituer une phylogénie, les biologistes la complètent généralement par un traitement, le *bootstrap* [Efron, 1979], qui évalue la fiabilité du résultat obtenu. Il s'agit d'une procédure d'inférence statistique appliquée notamment en reconstruction phylogénétique [Felsenstein, 1985] permettant de tester à quel point la phylogénie est sensible au bruitage des données.

Soit un ensemble X comportant n séquences de taille m, et T un arbre reconstruit avec un algorithme Λ à partir de X. Générons maintenant un ensemble conséquent (par exemple un millier) de jeux de séquences bruités (*bootstrappés*) $\{X_1^*, \ldots, X_{1000}^*\}$. Pour générer chaque X_i^*, nous tirons au hasard et avec relance m sites issus de X. La conséquence est que certains sites de X seront présents plusieurs fois dans un X_i^*, et d'autres seront absents. Ensuite, nous reconstruisons 1000 arbres $T_1^*, \ldots, T_{1000}^*$, suivant respectivement $X_1^*, \ldots, X_{1000}^*$ et avec la même méthode de reconstruction Λ. Nous comptabilisons ensuite, pour chacune des $n-3$ arêtes internes (non triviales) de la représentation non enracinée de T, leur taux d'apparition dans $\{T_1^*, \ldots, T_{1000}^*\}$. Ce taux, ramené à un entier de 0 à 100, permet d'identifier les informations (véhiculées par les arêtes) quasi insensibles au bruit de fond des séquences. Une représentation finale de T peut être un arbre non binaire, dont les arêtes internes sont celles dont le score de bootstrap dépasse un certain taux, par exemple 90. Ces arêtes sont considérées comme des branches robustes.

1.4 Conclusion

La reconstruction phylogénétique consiste à inférer un arbre d'évolution en fonction de données relatives à un ensemble d'espèces. Il s'agit d'un axe de recherche majeur de la bio-informatique, tant les nombreuses applications motivent les biologistes. [Hillis *et al.*, 1996] en répertorient un grand nombre : évolution génétique, classification et taxonomie, subdivisions de populations, variations géographiques, tests de paternité, mise en évidence

de nouvelles espèces, analyse des comportements reproducteurs, recherche virale, *etc.* [Harvey *et al.*, 1996] rappellent également l'indispensabilité de la reconstruction phylogénétique pour beaucoup d'études biologiques.

Il existe plusieurs manières d'aborder la question, selon le type de données mises à disposition et l'utilisation éventuelle de tel ou tel modèle de l'évolution. Parmi elles, le problème Maximum de Parcimonie, qui fait l'objet de cette thèse, consiste à trouver un arbre qui minimise le nombre de changements d'état survenus au cours de l'évolution. C'est ce que nous rappelons et détaillons dans le chapitre suivant.

Chapitre 2

Le problème Maximum de Parcimonie

L'analyse cladistique est une méthode de reconstruction qui ne recourt pas à un modèle de l'évolution. Dans ce chapitre, nous présentons en détail le problème Maximum de Parcimonie afin de l'envisager comme un problème d'optimisation. La présentation se veut précise et exhaustive et s'appuiera sur des exemples simples. Les nombreux ouvrages évoquant ce problème, généralement rédigés pour des biologistes, l'abordent de diverses manières et la plupart du temps de manière intuitive. Au contraire, nous proposons dans ce chapitre une définition plus formelle du problème.

Sommaire

2.1 Problématique

Le critère de parcimonie constitue une méthode de reconstruction phylogénétique très simple à appréhender. À contre courant des méthodes utilisant une modélisation des processus évolutifs, le critère de parcimonie part d'un postulat métascientifique bien connu des philosophes, le rasoir d'Ockham, qui cherche à expliquer la plus grande diversité possible d'objets et de processus par le plus petit nombre possible d'idées.

Si l'on attribue à chaque espèce considérée une séquence de caractères — chaque caractère pouvant prendre plusieurs états possibles —, alors l'arbre le plus parcimonieux sera celui qui minimise le nombre de changements d'état survenus au cours de l'évolution, les espèces étant placées aux feuilles de l'arbre.

Au départ, cette méthode fut développée pour des caractères morphologiques [Hennig, 1966]. Elle est majoritairement utilisée dans le cas de données moléculaires, c'est-à-dire de séquences de nucléotides ou d'acides aminés.

Prenons l'exemple où nous voulons reconstituer l'arbre le plus parcimonieux de quatre espèces représentées par les séquences suivantes :

> Esp 1 AAA
> Esp 2 AGA
> Esp 3 TAC
> Esp 4 TGC

Bien sûr, un tel exemple n'est pas valable en pratique puisque le nombre de caractères sur lequel porte l'analyse est bien trop faible pour présenter un intérêt biologique. Il existe néanmoins un ou plusieurs arbres qui minimise le *score de parcimonie*, c'est à dire le nombre de changements d'état. Nous voulons ici construire un de ces arbres optimaux ; les données sont les feuilles de l'arbre, il nous faut reconstruire sa topologie (Figure 2.1.a).

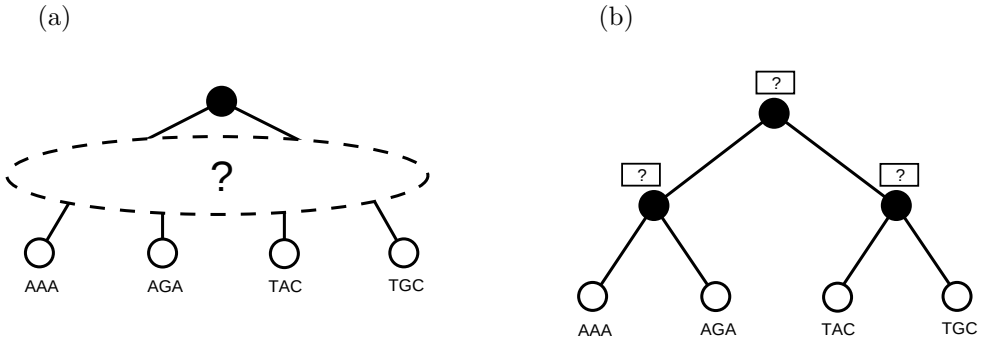


Fig. 2.1 – Deux niveaux d'inférence pour minimiser le score de parcimonie

D'après le tableau 1.1 (chapitre précédent), il existe 15 arbres enracinés de taille 4. Une possibilité pour trouver l'arbre le plus parcimonieux est de calculer le score de parcimonie de chacun de ces 15 arbres. Prenons par exemple l'arbre de la figure 2.1.b. Pour calculer

le nombre de changements d'état total de l'arbre, nous avons besoin de connaitre les séquences hypothétiques de tous les noeuds internes de l'arbre. Or, puisque ces séquences ne sont pas données, nous devons les inférer. Si l'on considère l'alphabet des nucléotides ($\Sigma = \{A, C, G, T\}$), il existe quatre états possibles pour chaque caractère et pour chaque séquence hypothétique, soit $4^9 = 262\ 144$ configurations pour l'ensemble des 9 caractères (3 séquences $\times$ 3 caractères par séquence).

Nous pouvons remarquer sur la figure 2.2 que le nombre de changements d'état dépend des affectations des séquences hypothétiques. Le score de parcimonie d'un arbre désigne en fait le nombre minimal de changements en envisageant toutes les affectations possibles. Calculer le score de parcimonie d'un arbre donné revient à résoudre le *problème de parcimonie restreint*, pour lequel il existe un algorithme de résolution simple qui construit une affectation valide de séquences hypothétiques. Il s'agit de l'algorithme de Fitch présenté dans la section suivante.

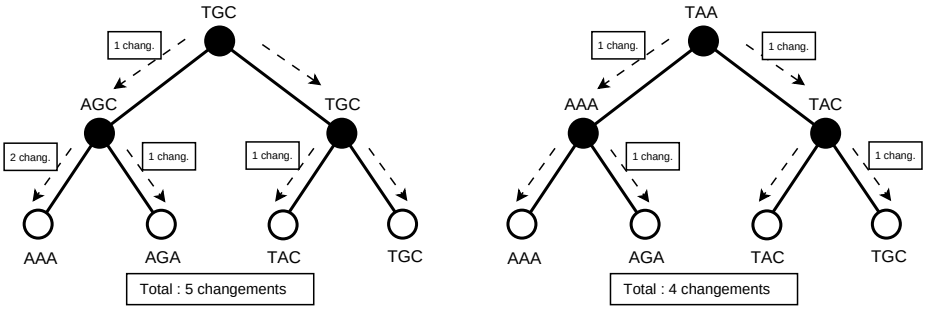


FIG. 2.2 – Différentes séquences hypothétiques pour un même arbre

2.2 Problème de parcimonie restreint et algorithme de Fitch

Le problème de parcimonie restreint (*small parsimony problem*) a trait au calcul du score de parcimonie d'un arbre phylogénétique dont la topologie est donnée.

Problème de parcimonie restreint

> **Donnée** : la topologie d'un arbre phylogénétique, dont les feuilles sont étiquetées par des séquences de même longueur.
> **Questions**
> 1. Quel est le nombre minimum de changements pour cette topologie ?
> 2. Quel est l'étiquetage optimal des nœuds internes de l'arbre ?

L'algorithme de résolution (**algorithme de Fitch**) est très intuitif. Il constitue l'étiquetage optimal des nœuds internes, duquel découle le score de parcimonie. Les séquences des nœuds internes ainsi reconstruites sont appelées *séquences de parcimonie*. Nous proposons d'inclure l'algorithme de Fitch à la définition d'une séquence de parcimonie, puisque l'un dépend de l'autre.

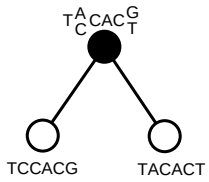


FIG. 2.3 – Séquence de parcimonie

Définition 5 (Séquence de parcimonie et Algorithme de Fitch) *Soient deux séquences S_1 et S_2 de même longueur m telles que $S_1 = (s_1^1, \ldots, s_1^m)$, $S_2 = (s_2^1, \ldots, s_2^m)$ avec $\forall i \in \{1,2\}$ et $k \in \{1..m\}$, $s_i^k \in \mathcal{P}(\Sigma)$, ensemble des parties de l'alphabet Σ. La séquence de parcimonie de S_1 et S_2, notée $F(S_1, S_2) = (f(s_1^1, s_2^1), \ldots, f(s_1^m, s_2^m))$ est obtenue par l'algorithme suivant ([Fitch, 1971]) :*

$$\forall k \in \{1..m\}, f(s_1^k, s_2^k) = \begin{cases} s_1^k \cup s_2^k, \text{si } s_1^k \cap s_2^k = \emptyset \\ s_1^k \cap s_2^k, \text{sinon} \end{cases}$$

Le cout de la séquence de parcimonie $F(S_1, S_2)$ dépend de S_1 et S_2 et non de $F(S_1, S_2)$. Il est défini par :

$$\phi(S_1, S_2) = \sum_{k=1}^{m} \phi(s_1^k, s_2^k), \quad \text{avec} \quad \phi(s_1^k, s_2^k) = \begin{cases} 1, \text{si } s_1^k \cap s_2^k = \emptyset \\ 0, \text{sinon} \end{cases}$$

À titre d'exemple, considérons deux séquences nucléotidiques $S_1 =$ `TCCACG` et $S_2 =$ `TACACT`, s'exprimant sur l'alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. s_i^k représente l'ensemble des états que le k-ième caractère de la séquence S_i peut prendre pour que le score de parcimonie soit minimum.

En utilisant l'algorithme de Fitch, nous pouvons calculer la séquence de parcimonie associée à S_1 et S_2 de cout minimal.

k	1	2	3	4	5	6	
s_1^k	{T}	{C}	{C}	{A}	{C}	{G}	
s_2^k	{T}	{A}	{C}	{A}	{C}	{T}	
$s_1^k \cap s_2^k$	**{T}**	{}	**{C}**	**{A}**	**{C}**	{}	
$s_1^k \cup s_2^k$	{T}	**{A,C}**	{C}	{A}	{C}	**{G,T}**	
$f(s_1^k, s_2^k)$	{T}	{A,C}	{C}	{A}	{C}	{G,T}	
$\phi(s_1^k, s_2^k)$	0	1	0	0	0	1	Total : 2

Le cout de de la séquence de parcimonie de S_1 et S_2, noté $\phi(S_1, S_2)$, est égal à 2. Ainsi, comme nous le voyons sur la figure 2.3, deux changements d'état au minimum suffisent à expliquer l'évolution de S_1 et S_2 depuis un ancêtre commun hypothétique. Sur les figures, plusieurs états possibles pour un caractère sont représentés verticalement.

De la même manière, pour les deux séquences $S_3 =$ `AAATCC` et $S_4 =$ `TAACCT`, nous obtenons un score de 3 (voir tableau suivant). Nous remarquons que lorsque les deux séquences

ne sont composées que de singletons (un seul état possible pour chaque caractère), ce qui est généralement le cas des séquences initiales, alors le cout de leur séquence de parcimonie est égal à leur distance de Hamming[1].

k	1	2	3	4	5	6	
s_3^k	{A}	{A}	{A}	{T}	{C}	{C}	
s_4^k	{T}	{A}	{A}	{C}	{C}	{T}	
$s_3^k \cap s_4^k$	{}	**{A}**	**{A}**	{}	**{C}**	{}	
$s_3^k \cup s_4^k$	**{A,T}**	{A}	{A}	**{C,T}**	{C}	**{C,T}**	
$f(s_3^k, s_4^k)$	{A,T}	{A}	{A}	{C,T}	{C}	{C,T}	
$\phi(s_3^k, s_4^k)$	1	0	0	1	0	1	Total : 3

Enfin, calculons la séquence de parcimonie de ces deux séquences résultantes, $F(S_1, S_2)$ et $F(S_3, S_4)$. Par commodité, nommons $H_1 = F(S_1, S_2) = (h_1^1, \ldots, h_1^m)$ et $H_2 = F(S_3, S_4) = (h_2^1, \ldots, h_2^m)$.

k	1	2	3	4	5	6	
h_1^k	{T}	{A,C}	{C}	{A}	{C}	{G,T}	
h_2^k	{A,T}	{A}	{A}	{C,T}	{C}	{C,T}	
$h_1^k \cap h_2^k$	**{T}**	**{A}**	{}	{}	**{C}**	**{T}**	
$h_1^k \cup h_2^k$	{A,T}	{A,C}	**{A,C}**	**{A,C,T}**	{C}	{C,G,T}	
$f(h_1^k, h_2^k)$	{T}	{A}	{A,C}	{A,C,T}	{C}	{T}	
$\phi(h_1^k, h_2^k)$	0	0	1	1	0	0	Total : 2

À ce stade, nous avons calculé les trois séquences de parcimonie de l'arbre $((\mathtt{S}_1, \mathtt{S}_2), (\mathtt{S}_3, \mathtt{S}_4))$[2], ainsi que leurs couts respectifs.

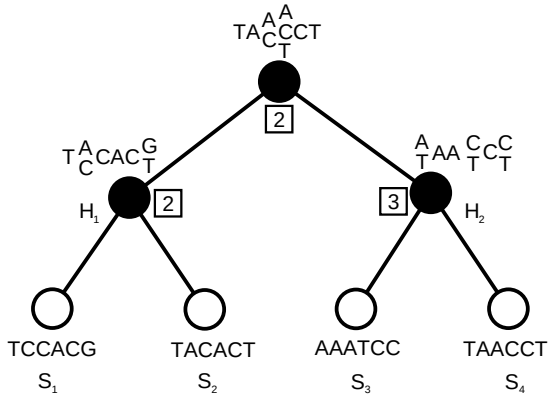


Fig. 2.4 – Exemple d'application de l'algorithme de Fitch

[1] La *distance de Hamming* entre deux séquences $x = (x^1, \ldots, x^m)$ et $y = (y^1, \ldots, y^m)$ est égale à $|\{k : x^k \neq y^k\}|$.

[2] Cette écriture parenthésée d'un arbre est appelée *format Newick* [Cayley, 1857].

Cela nous amène à définir plus précisément la notion d'arbre de parcimonie. Dans la littérature, on trouve très peu de définitions formelles du problème de parcimonie en général, et d'un arbre de parcimonie en particulier. Le manque de consensus à ce sujet vient probablement du fait que ces applications sont à la frontière entre l'informatique, la biologie et les mathématiques. Il faut néanmoins admettre que ce problème, relativement simple à appréhender, est plus délicat à définir formellement.

Tout d'abord, la donnée initiale du problème est un ensemble de n séquences, comportant toutes un même nombre de m caractères s'exprimant sur un alphabet Σ (ensemble des états possibles pris par chaque caractère). Parfois, à un caractère particulier peut correspondre plusieurs états possibles. Nous considérons alors d'une manière générale une séquence par un vecteur d'ensembles d'états possibles. Cette définition d'une séquence est en adéquation avec celles de la définition 5.

Un ensemble de séquences (généralement un *alignement* de séquences dans le cas d'applications réelles) se définit alors par une matrice $\mathcal{A}$ de taille $n \times m$ à coefficients dans $\mathcal{P}(\Sigma)$. Nous notons les lignes de la matrice $\mathcal{A}_i \in \mathcal{P}^m(\Sigma)$ ($i \in \{1..n\}$) et les colonnes $\mathcal{A}^k \in \mathcal{P}^n(\Sigma)$ ($k \in \{1..m\}$). Les $\mathcal{A}_i$ et les $\mathcal{A}^k$ sont des vecteurs qui représentent respectivement les séquences et les sites de l'alignement. Enfin, chaque $\mathcal{A}_i^k \in \mathcal{P}(\Sigma)$ est l'ensemble des états possibles pris par le k-ième caractère de la séquence i.

Un arbre de parcimonie est un arbre dont les feuilles sont étiquetées par chacune des séquences $\mathcal{A}_i$ et dont les autres nœuds sont étiquetés par des séquences de parcimonie. Nous choisissons de définir un arbre de parcimonie comme arbre enraciné, mais il est possible d'adapter cette définition pour le cas d'arbres non enracinés.

Définition 6 (Arbre de parcimonie) *Soit $\mathcal{A}$ une matrice $n \times m$ à coefficients dans $\mathcal{P}(\Sigma)$. Soit $T = (\mathcal{V}, \mathcal{E}, \zeta)$ un arbre binaire entier enraciné, où $\mathcal{V} = \{v_1, \ldots, v_r\}$ est l'ensemble des nœuds, $\mathcal{E} \subseteq \mathcal{V}^2$ est l'ensemble des arêtes, et $\zeta : \mathcal{V} \to \mathcal{P}^m(\Sigma)$ est une application qui attribue une séquence (ligne de la matrice) à chaque nœud. Notons $\mathcal{L} \subseteq \mathcal{V}$ l'ensemble des n nœuds n'ayant aucun descendant (feuilles), et $\mathcal{S}$ l'ensemble des n séquences $\mathcal{A}_i$ ($i \in \{1..n\}$). T est un arbre de parcimonie de $\mathcal{A}$ si :*

i. $\zeta_{|\mathcal{L}}$ est une bijection de $\mathcal{L}$ dans $\mathcal{S}$, et

ii. $\forall v \in \mathcal{V} \setminus \mathcal{L}$, $\zeta(v)$ est la séquence de parcimonie des deux séquences $\zeta(w_1)$ et $\zeta(w_2)$, (w_1, w_2) distincts et tels que $\{(v, w_1), (v, w_2)\} \subseteq \mathcal{E}$.

Le cout *(ou* score*) de T, noté $\phi(T)$ est défini ainsi :*

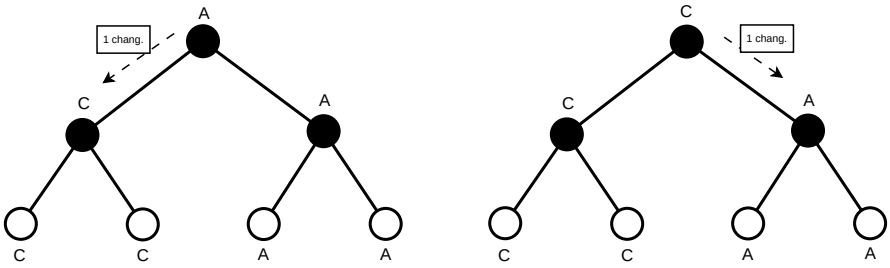
$$\phi(T) = \sum_{\substack{v \in \mathcal{V} \setminus \mathcal{L} \\ (v,w_1),(v,w_2) \in \mathcal{E} \\ w_1 \neq w_2}} \phi(\zeta(w_1), \zeta(w_2))$$

Propriété 3 *Soit $T = (\mathcal{V}, \mathcal{E}, \zeta)$ un arbre de parcimonie. Soit T^k l'arbre T dont les feuilles sont étiquetées uniquement suivant $\mathcal{A}^k$. L'étiquetage de l'arbre est donc celui du k-ième site des séquences. Nous avons alors l'égalité suivante :*

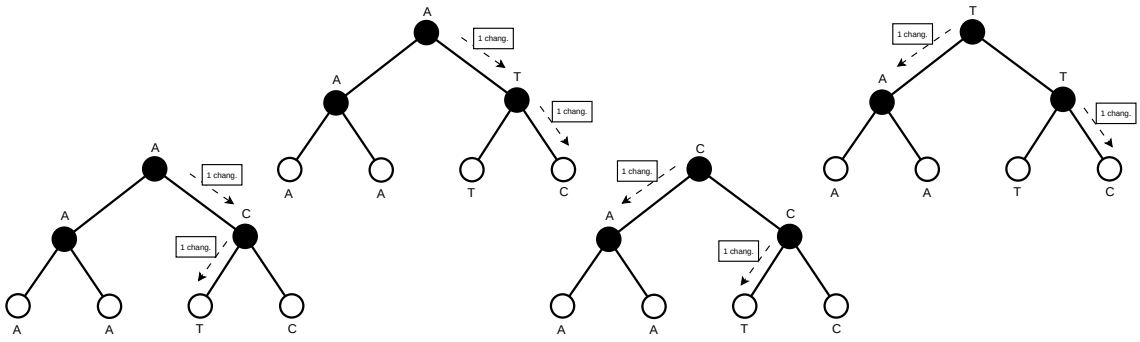
$$\phi(T) = \sum_{k=1}^{m} \phi(T^k)$$

Le cout d'un arbre est égal à la somme des couts de toutes les séquences de parcimonie associées à ses nœuds internes. Par conséquent, dans notre exemple, le cout de l'arbre est égal à 7 $(3 + 2 + 2)$. Ce cout peut également être présenté comme la somme des couts des m arbres T^k dont les feuilles sont étiquetées par le k-ième caractère des séquences correspondantes données (propriété 3). Ces deux manières de calculer sont équivalentes, car l'ordre des sommes ne modifie pas le nombre d'opérations à effectuer ; ce constat illustre l'indépendance des sites. Précisons que cette définition d'un arbre de parcimonie exclut la possibilité que plusieurs séquences données soient identiques, à moins de considérer des multiensembles.

L'algorithme de Fitch retourne des séquences de parcimonie dans lesquelles certains caractères peuvent être indéterminés : dans l'exemple précédent (figure 2.4), le troisième caractère de la séquence racine peut être `A` ou `C`. Quelle que soit la valeur choisie, le cout de l'arbre sera inchangé. On peut distinguer ces deux étiquetages pour le troisième site :



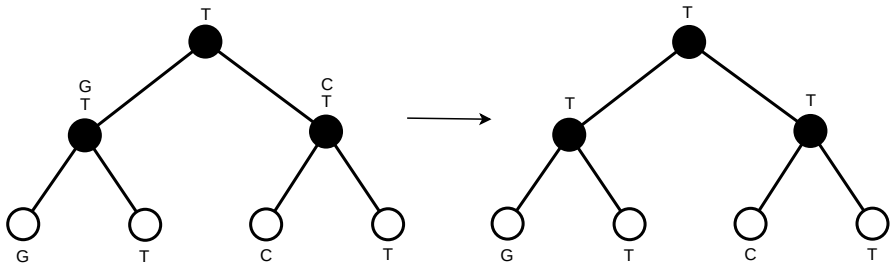
Ces deux étiquetages sont équivalents, et permettent de distinguer les deux scénarios envisageables en fixant chaque caractère à un unique état. De la même manière, pour le quatrième site, il existe quatre explications possibles :



Pour l'ensemble des six caractères, le tableau suivant comptabilise le nombre d'étiquetages possibles :

Caractère	1	2	3	4	5	6
Etiquetages	2	2	2	4	1	1

Les caractères 5 et 6 ne permettent qu'un unique étiquetage, mais pour des raisons différentes. Le caractère 5 est ce que l'on appelle un *site invariant*, c'est-à-dire que chaque taxon a le même état pour ce caractère (voir section 2.4.1). Pour le caractère 6 en revanche, on ne peut déduire l'étiquetage unique pour cet arbre (en l'occurence `T` pour chaque nœud interne) qu'une fois calculées toutes les séquences de parcimonie. Ici, le `T` de la racine se propage sur ses decendants, car une autre affectation entrainerait une augmentation du score de parcimonie.



Lorsque les caractères sont concaténés, *i.e.* lorsque l'on considère les séquences, il existe alors pour l'exemple considéré 32 étiquetages possibles (le cardinal du produit cartésien des différents étiquetages), pour un même score de parcimonie de 7.

Complexité de l'algorithme de Fitch

Le cout (comme la construction) d'une séquence de parcimonie se calcule en $O(m)$, m étant la taille des séquences. Le cout de l'arbre s'obtient en calculant toutes les séquences de parcimonie. Le nombre de séquences à calculer est égal au nombre de nœuds internes de l'arbre, soit $n-1$ (n est le nombre de séquences, donc le nombre de feuilles). Au final, l'algorithme de Fitch a une complexité de $O(m.n)$.

2.3 Problème Maximum de Parcimonie et arbre de Steiner

Le problème Maximum de Parcimonie (MP), encore appelé problème de parcimonie étendu (*large parsimony*), consiste à retrouver un arbre phylogénétique cohérent avec les données et qui minimise le score de parcimonie.

Problème Maximum de Parcimonie

> **Donnée** : une matrice de $n \times m$ caractères décrivant un ensemble de n taxons.
> **Question** : quel est l'arbre phylogénétique optimal, *i.e.* celui qui minimise le score de parcimonie ?

Comme le rappelle [Gusfield, 1997], le problème MP est équivalent à un autre problème combinatoire, à savoir le problème de l'arbre de Steiner dans un hypercube.

Définition 7 *Un* hypercube *de dimension d est un graphe non orienté contenant* 2^d *sommets bijectivement étiquetés par les entiers compris entre 0 et* $2^d - 1$. *Deux sommets d'un hypercube sont adjacents si et seulement si les représentations binaires de leurs étiquettes diffèrent par un bit uniquement.*

Un arbre phylogénétique peut être vu comme un arbre connectant dans l'espace des séquences, des points particuliers. Pour pouvoir le représenter aisément ici, prenons deux séquences binaires S_1 et S_2, avec $\Sigma = \{0, 1\}$. Nous considérons simultanément trois exemples triviaux, chaque séquence comportant un ou deux caractères.

Exemple 1		Exemple 2		Exemple 3	
S_1	0	S_1	00	S_1	00
S_2	1	S_2	01	S_2	11

Maintenant, imaginons un hypercube (définition 7) de dimension m (la taille des séquences), dont les sommets représentent chacune des 2^m séquences binaires distinctes. Par définition, deux sommets de cet hypercube sont adjacents si et seulement si la distance de Hamming des séquences qu'ils étiquettent est égale à 1.

Comme nous l'avons vu précédemment, il n'existe qu'un seul arbre phylogénétique comportant deux taxons. Nous voyons sur la figure 2.5 comment représenter cet arbre dans un hypercube. Les sommets symbolisés par les points en gras sont les séquences du problème (données). Les autres sommets représentent chacune des $2^m - n$ séquences possibles. Le problème MP consiste dans ce cadre à chercher l'arbre le plus court, sous-graphe de l'hypercube et contenant tous les sommets particuliers qui correspondent aux données. Chaque arête de l'arbre symbolise un changement d'état, et le score de parcimonie de l'arbre est égal à son nombre d'arêtes (*longueur*). L'arbre ainsi obtenu n'est pas nécessairement binaire, mais il est aisé de le convertir en arbre binaire. Premièrement en insérant des nœuds de degré 3 lorsque certains nœuds ont un degré supérieur, puis en sortant les sommets particuliers (les transformer en feuilles) si leur degré est égal à 2. Enfin, pour rendre l'arbre binaire ainsi obtenu entier, il suffit de remplacer tous les chemins $(x_i, \ldots, x_{i+l})$ où seuls x_i et x_{i+l} ont un degré différent de 2, par une arête simple (x_i, x_{i+l}) étiquetée par la longueur l du chemin. Nous obtenons alors une solution non enracinée au problème, mais il est possible d'insérer à tout endroit une racine.

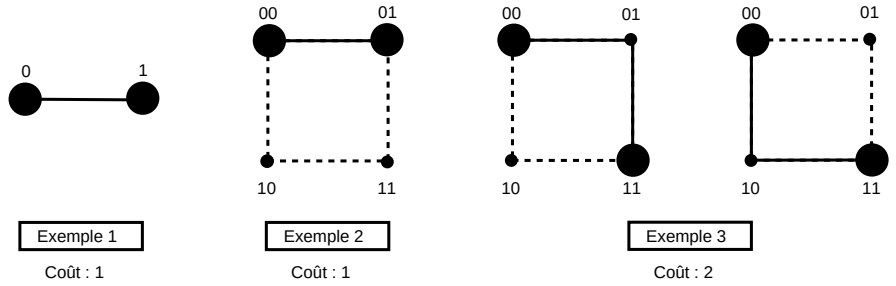


FIG. 2.5 – Une manière différente d'appréhender le problème MP

Sur la figure 2.5, l'exemple 1 représente un hypercube de dimension 1, unique solution concevable au problème. Dans l'exemple 2, en plus de la topologie triviale d'un arbre comportant deux feuilles, nous remarquons que la recherche de l'arbre couvrant (arbre partiel de l'hypercube) le plus court permet d'optimiser le score de parcimonie. De la

même manière dans l'exemple 3, deux explications équivalentes sont imaginables : l'une passant par ancêtre hypothétique 01 et l'autre par 10.

Étant donné un graphe et des sommets particuliers, rechercher l'arbre le plus court qui contient tous les sommets particuliers revient à résoudre le problème de Steiner dans un graphe.

Définition 8 (Arbre de Steiner) *Soit $G = (\mathcal{V}, \mathcal{E})$ un graphe non orienté. $\mathcal{V}$ est l'ensemble de ses sommets et $\mathcal{E} \subseteq \mathcal{V}^2$ l'ensemble de ses arêtes. À chaque arête $(v_i, v_j) \in \mathcal{E}$ est associé un poids non négatif $\omega_{ij} \in \mathbb{N}^*$. Soit $\mathcal{X} \subseteq \mathcal{V}$ un sous-ensemble de sommets. Un* arbre de Steiner *$ST_G(\mathcal{X}) = (\mathcal{V}', \mathcal{E}')$ est un arbre tel que $\mathcal{X} \subseteq \mathcal{V}' \subseteq \mathcal{V}$, et $\mathcal{E}' \subseteq \mathcal{E}$. Les sommets de $\mathcal{V}' \setminus \mathcal{X}$ sont appelés* points de Steiner*. La longueur (ou poids) de $ST_G(\mathcal{X})$, notée $\Omega(ST_G(\mathcal{X}))$, est définie par :*

$$\Omega(ST_G(\mathcal{X})) = \sum_{\substack{i,j \\ (v_i,v_j)\in\mathcal{E}'}} \omega_{ij}$$

Étant donné G et $\mathcal{X}$, le problème de Steiner, du nom du mathématicien Jakob Steiner (1796-1863), consiste à trouver l'arbre de Steiner de longueur minimum. Le problème de Steiner non pondéré est défini lorsque les arêtes de G ne sont pas étiquetées. Dans ce cas, $\omega_{ij} = 1$, pour tous i, j tels que $(v_i, v_j) \in \mathcal{E}'$.

Le problème MP sur des séquences binaires est en réalité équivalent au problème de Steiner non pondéré dans un hypercube. Un hypercube de dimension m possède un sommet pour chacune des 2^m chaines de m bits, donc des 2^m séquences possibles de m caractères. Deux séquences séparées par un changement d'état sont représentées par deux sommets adjacents dans l'hypercube. Il apparait que l'arbre de Steiner le plus court est équivalent à l'arbre le plus parcimonieux — en tolérant un possible arbre non binaire, mais comme nous l'avons vu la transformation est immédiate. Le nombre d'arêtes de l'arbre de Steiner est alors égal au score de parcimonie minimal. Le premier lien entre le problème MP et le problème de Steiner est à mettre au crédit de [Sankoff and Rousseau, 1975].

Nous proposons d'illustrer l'équivalence entre les deux problèmes à l'aide d'un exemple simple mais cette fois-ci non trivial. Soient cinq séquences de longueur 4 définies comme suit :

> Esp 1 0000
> Esp 2 0011
> Esp 3 0101
> Esp 4 0110
> Esp 4 1001

Le problème est de trouver un des arbres les plus parcimonieux. Pour résoudre le problème de Steiner associé, considérons un hypercube $H = (\mathcal{V}, \mathcal{E})$ de dimension 4 et un ensemble de cinq sommets $\mathcal{X} = \{0000, 0011, 0101, 0110, 1001\}$. Sur la figure 2.6.a, nous représentons dans H ces cinq sommets, ainsi qu'un arbre de Steiner de longueur 6. Cet arbre est représenté plus clairement sur la figure 2.6.b, où l'on rappelle que chaque arête

symbolise un changement d'état. Les sommets initiaux sont indiqués en gras, tandis que les deux autres sommets sont les points de Steiner.

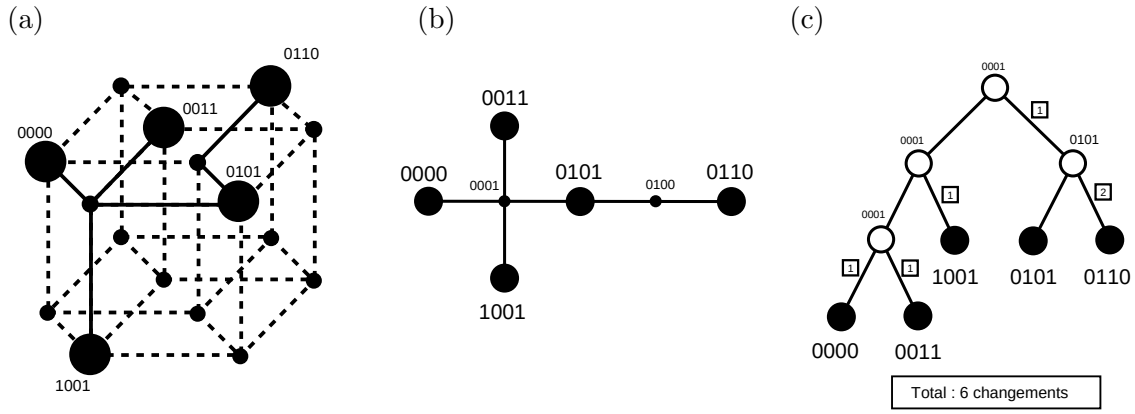


FIG. 2.6 – Problème de Steiner dans un hypercube de dimension 4

En pratique, il n'est pas nécessaire d'aller plus loin, car l'arbre de la figure 2.6.b, en ôtant éventuellement le point de Steiner superflu (de degré 2), contient toutes les informations pertinentes. Cependant, pour apporter plus de rigueur et se conformer aux définitions énoncées dans la section précédente, nous représentons cet arbre plus lisiblement (figure 2.6.c). Celui-ci, dont le score est 6, semble être optimal ; c'est-à-dire qu'il n'existerait pas d'arbre plus parcimonieux.

Le problème de Steiner dans un hypercube est NP-complet, et par équivalence le problème MP également [Foulds and Graham, 1982] ; en conséquence, il est impossible dans l'état actuel des connaissances de produire une méthode systématique permettant de générer rapidement une solution optimale. Dans cet exemple, la construction de l'arbre a été réalisée de manière arbitraire, dans le but de distinguer les analogies entre le problème de Steiner dans un hypercube et le problème MP. Les approches de résolution feront quant à elles l'objet du chapitre 3.

Nous venons de constater l'équivalence entre le problème de Steiner dans un hypercube et le problème MP dans le cas d'un alphabet binaire. Il ne faut que quelques transformations pour généraliser cette équivalence au problème MP dans le cas général [Day *et al.*, 1986].

2.4 Calcul du score : propriétés de base

Une partie des données du problème peuvent n'avoir aucune incidence sur sa solution. C'est le cas des sites non informatifs, fréquemment mentionnés dans la littérature, mais rarement décrits de manière précise. Nous proposons dans cette section une formalisation de toutes les simplifications possibles des séquences initiales.

2.4.1 Sites non informatifs

Pour calculer les séquences hypothétiques et le score de parcimonie d'un arbre à partir d'un alignement de séquences, il suffit d'isoler chaque site k et de calculer les m scores $\phi(T^k)$ (voir propriété 3). Or sur certains sites, les x_i^k sont tels que quelle que soit la topologie de l'arbre T, $\phi(T^k)$ aura toujours la même valeur. Ces sites sont aisément identifiables, et ne jouent aucun rôle dans l'établissement de la solution optimale.

Motivation. Soit $\mathcal{A}$ une matrice $n \times m$ décrivant un alignement de séquences, $I \subseteq \{1..m\}$ un ensemble d'indices correspondants à des sites non informatifs dans $\mathcal{A}$, et $\overline{I} = \{1..m\} \setminus I$ son complément. Notons $\mathcal{A}^{\overline{I}}$ la sous-matrice de taille $n \times |\overline{I}|$, obtenue en supprimant les colonnes indicées par les éléments de I, et $\hat{T}$ un arbre parcimonieux optimal décrivant $\mathcal{A}$. Alors $\hat{T}$ est également optimal au sens de $\mathcal{A}^{\overline{I}}$.

Établir l'alignement $\mathcal{A}^{\overline{I}}$ le plus court possible permet de réduire le traitement à un sous-ensemble de sites. Il existe trois types de sites qui n'ont aucun effet sur la topologie des solutions optimales :

Les sites invariants. Un site $\mathcal{A}^k = (x_1^k, \ldots, x_n^k) \in \Sigma^n$ est dit *invariant* si tous les x_i^k ($i \in \{1..n\}$) sont égaux[3]. Quelle que soit la position des taxons dans l'arbre, il n'y aura aucun changement évolutif sur ce site (Figure 2.7.a). Dans ce cas, les arbres les plus parcimonieux d'après $\mathcal{A}$ le seront également d'après $\mathcal{A}^{\overline{\{k\}}}$.

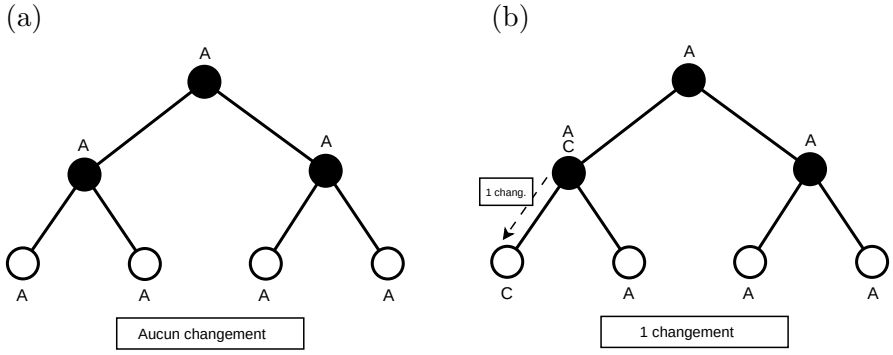


FIG. 2.7 – Scores de parcimonie constants sur les sites informatifs

Les sites quasi-invariants. Un site $\mathcal{A}^k$ est *quasi-invariant* (*singleton* dans la littérature) si tous les x_i^k sont égaux, sauf un (Figure 2.7.b). Ici, bien que les arbres optimaux soient les mêmes avec $\mathcal{A}$ qu'avec $\mathcal{A}^{\overline{\{k\}}}$, leurs scores de parcimonie diffèrent d'une unité.

Les autres sites non informatifs. En généralisant la propriété des sites quasi-invariants, on constate que pour qu'un site soit informatif (*i.e.* nous ne pouvons pas déterminer *a priori*

[3]Il est possible d'étendre la définition aux sites $\mathcal{A}^k \in \mathcal{P}^n(\Sigma)$. Dans ce cas, $\mathcal{A}^k$ est invariant s'il existe un $\mathcal{A}^* = (x_1^*, \ldots, x_n^*) \in \Sigma^n$ tel que $\forall i \in \{1..n\}, x_i^* \in x_i^k$.

que l'information qu'il contient soit indépendante des facteurs d'optimalité des arbres les plus parcimonieux), il faut et il suffit qu'au moins deux états soient représentés au moins deux fois.

Exemple : Soit $\mathcal{A}$ l'alignement de séquences suivant :

$$\mathcal{A} = \begin{array}{|cccccc|} \hline 1 & 2 & 3 & 4 & 5 & 6 \\ \hline C & A & A & C & A & T \\ C & T & A & G & T & T \\ G & C & A & G & C & A \\ C & G & A & C & C & A \\ \hline \end{array}$$

Le site 3 est invariant, et le site 1 quasi-invariant. De plus, les sites 2 et 5 ne sont pas informatifs : pour le 2, aucun état n'est représenté plus d'une fois, et pour le 5, un seul état est représenté au moins deux fois. Pour tous ces sites, cela signifie que, puisque les taxons sont situés aux feuilles, le changement d'état éventuel qui minimise le score de parcimonie peut survenir entre ce taxon et son ancêtre. Et quelle que soit leur position dans l'arbre, ce changement éventuel sera nécessaire.

L'alignement simplifié s'obtient en retirant les sites non informatifs de l'alignement initial. Le problème sera donc équivalent en utilisant l'alignement suivant :

$$\mathcal{A}^{\{4,6\}} = \begin{array}{|cc|} \hline 4 & 6 \\ \hline C & T \\ G & T \\ G & A \\ C & A \\ \hline \end{array}$$

En revanche, le score de parcimonie du ou des arbres optimaux ne sera pas le même avec l'alignement simplifié qu'avec l'alignement initial. Pour connaitre le score de parcimonie optimal en fonction des données initiales, il faut ajouter au premier score autant d'unités que de caractères isolés dans les sites supprimés, soit $1 + 4 + 0 + 2 = 7$.

2.4.2 Sites équivalents

Parmi les sites informatifs, certains peuvent véhiculer la même information, au regard du critère de parcimonie. Par exemple, si plusieurs sites sont parfaitement identiques (vecteurs colonnes identiques dans la matrice $\mathcal{A}$), il est possible de n'en garder qu'un en ajoutant une pondération au site. De même, s'il existe une permutation $\xi : \Sigma \to \Sigma$ telle que $\forall i \in \{1..n\}$, $\xi(x_i^k) = x_i^{k'}$, alors nous considérons que les sites k et k' sont équivalents et peuvent être traités comme des sites identiques.

Dans l'exemple ci-après, les sites 1, 3 et 4 sont équivalents, de même que les sites 2 et 6. Cela nous permet de simplifier l'alignement initial.

$$\mathcal{A} = \begin{array}{|cccccc|}\hline 1 & 2 & 3 & 4 & 5 & 6 \\ \hline A & T & A & C & A & G \\ C & T & C & G & T & G \\ C & C & C & G & A & C \\ A & C & A & C & T & C \\ \hline \end{array} \qquad \mathcal{A}_*^{\{1,2,5\}} = \begin{array}{|ccc|}\hline 1 & 2 & 5 \\ \hline \times 3 & \times\ 2 & \times 1 \\ \hline A & T & A \\ C & T & T \\ C & C & A \\ A & C & T \\ \hline \end{array}$$

Bien qu'à première vue, cette nouvelle simplification permette d'économiser quelques calculs, elle n'est pas utilisée en pratique car elle nécessite une implémentation différente de la fonction de score (due à la pondération des sites), ainsi que des calculs supplémentaires afin de détecter les permutations. De plus, lorsque le nombre de taxons croît, les sites équivalents deviennent de plus en plus rares.

2.4.3 Position de la racine

Pour calculer le score de parcimonie d'un arbre, il faut préalablement calculer toutes les séquences de parcimonie associées aux nœuds internes. Une séquence de parcimonie s'obtient uniquement à partir des séquences associées aux deux descendants du nœud. Cela induit que l'arbre est enraciné, ce qui permet d'associer un sens aux arêtes. La figure 2.8 montre qu'un même arbre enraciné de différentes manières peut conduire à un étiquetage différents de ses nœuds internes ((a) et (b)). La présence d'une racine est cependant nécessaire afin d'orienter l'arbre et de pouvoir distinguer deux descendants et un ascendant pour chaque nœud interne. Lorsque l'arbre est non enraciné (c), l'algorithme de Fitch ne peut être appliqué car un parcours postfixe de l'arbre est impossible.

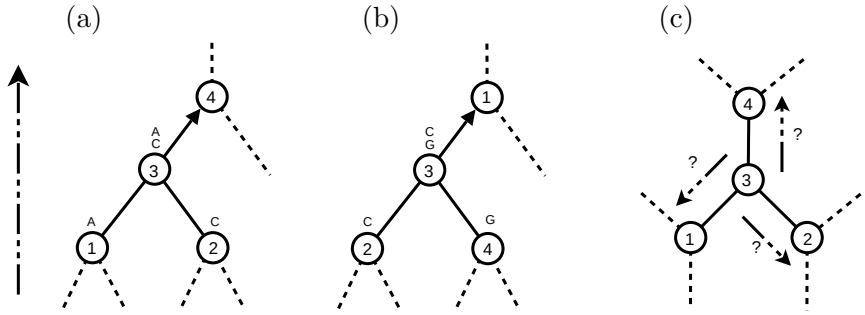


Fig. 2.8 – La présence d'une racine entraine une hiérarchisation des nœuds

Une propriété importante est que la position de la racine est indépendante du score de parcimonie de l'arbre. Bien que l'on place arbitrairement une racine afin de calculer le score d'un arbre — identifier une arête particulière pour hiérarchiser les nœuds —, celle-ci ne constitue pas un élément de la solution. Lorsque l'on considère un arbre enraciné comme solution du problème MP, cela signifie que tous les arbres enracinés construits en déplaçant la racine sont également solutions.

L'espace de recherche associé au problème MP se restreint donc à l'ensemble des arbres non enracinés, bien que l'on puisse travailler sur des arbres enracinés pour sa résolution (la proportion de solutions optimales reste la même). Certains auteurs, qui modélisent le pro-

blème directement à partir d'arbres non enracinés, parlent de racines de calcul (*calculation root*) pour orienter temporairement l'arbre afin d'en calculer le cout.

2.5 Parcimonie pondérée

Dans ce qui précède, le critère à optimiser afin de déterminer le *meilleur* arbre est le nombre de changements évolutifs. L'algorithme de Fitch comptabilise les changements sans pondération : quelle qu'elle soit, toute modification coute une unité. Dans ce cas, les matrices de substitutions qui correspondent au cout de chaque changement d'état éventuel sont de cette forme :

$\mathcal{C}$ =

	0	1
0	0	1
1	1	0

$\mathcal{C}$ =

	A	C	G	T
A	0	1	1	1
C	1	0	1	1
G	1	1	0	1
T	1	1	1	0

Ce modèle d'évolution des caractères est appelé *parcimonie de Fitch* [Fitch, 1971], et c'est celle que l'on traitera. Cependant, si l'on souhaite assigner des couts différents à certains changements d'état, alors l'algorithme de Fitch ne permet plus de reconstruire de manière optimale les séquences de parcimonie et de calculer le cout de l'arbre. Par extension, on peut imaginer que certains changements soient impossibles.

Un problème proposé par [Camin and Sokal, 1965] est de considérer un modèle où les caractères prennent leur valeur parmi plusieurs états arrangés linéairement, et où chaque substitution est irréversible. Soit $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, et le graphe des substitutions possibles suivant : $\sigma_1 \leftarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_4$. On remarque, par exemple, que le changement $\sigma_1 \rightarrow \sigma_3$ est impossible. Une alternative est que, quelle que soit la substitution, elle est irréversible et unique. Dans ce cas, le graphe des substitutions est un chemin : $\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_4$.

Un autre modèle [Farris, 1970] utilise cet arrangement linéaire des caractères, mais conserve la réversibilité des substitutions. Sur un alphabet à quatre caractères ordonnés ($\sigma_1 \leftrightarrow \sigma_2 \leftrightarrow \sigma_3 \leftrightarrow \sigma_4$), le poids d'une substitution est égal à la distance entre les deux caractères dans l'arrangement linéaire. En particulier la substitution $\sigma_2 \leftrightarrow \sigma_4$ doit être décomposée en deux substitutions, $\sigma_2 \leftrightarrow \sigma_3$ et $\sigma_3 \leftrightarrow \sigma_4$. Ce modèle est traditionnellement appelé *parcimonie de Wagner*.

Les matrices de substitutions associées à ces deux modèles, respectivement irréversibles (figure 2.9.a et 2.9.b) et ordonnés (figure 2.9.c), permettent d'associer au critère de parcimonie un modèle d'évolution des caractères. Enfin, évoquons un modèle alternatif basé sur l'alphabet des nucléotides qui alloue un poids plus fort aux transversions qu'aux transitions (figure 2.9.d).

Nous pouvons remarquer que les matrices d'évolution des caractères sont symétriques si et seulement si le processus est réversible.

Calculer le score de parcimonie pondérée d'un arbre est également nécessaire lorsque l'alphabet est celui des acides aminés. Il existe plusieurs matrices de substitutions d'acides aminés (PAM [Dayhoff *et al.*, 1978], BLOSUM [Henikoff and Henikoff, 1992], Gonnet

(a)

$\sigma_1 \leftarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_4$

	σ_1	σ_2	σ_3	σ_4
σ_1	0	∞	∞	∞
σ_2	1	0	1	2
σ_3	∞	∞	0	1
σ_4	∞	∞	∞	0

(b)

$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_4$

	σ_1	σ_2	σ_3	σ_4
σ_1	0	1	2	3
σ_2	∞	0	1	2
σ_3	∞	∞	0	1
σ_4	∞	∞	∞	0

(c)

	σ_1	σ_2	σ_3	σ_4
σ_1	0	1	2	3
σ_2	1	0	1	2
σ_3	2	1	0	1
σ_4	3	2	1	0

(d)

	A	C	G	T
A	0	2	1	2
C	2	0	2	1
G	1	2	0	2
T	2	1	2	0

FIG. 2.9 – Quatre modèles courants de parcimonie pondérée

[Gonnet *et al.*, 1992]), et dans aucun cas la parcimonie de Fitch ne peut être appliquée. Pour calculer le score d'un arbre en utilisant un tel modèle, on utilise l'algorithme de Sankoff [Sankoff, 1975], qui est une généralisation de l'algorithme de Fitch. L'algorithme de Sankoff, que nous ne détaillerons pas ici, est plus gourmand que l'algorithme de Fitch puisque sa complexité est en $O(m.n.|\Sigma|^2)$. Cette complexité peut parfois constituer un frein à la reconstitution d'arbres parcimonieux à partir de séquences d'acides aminés (où $|\Sigma| = 20$).

2.6 Discussion

Le problème Maximum de Parcimonie (non pondéré) est un pur problème d'optimisation qui ne prend en compte aucune hypothèse sur l'évolution des espèces ; en ce sens il est libre de toute interprétation erronée ou incomplète des mécanismes évolutifs. Toutes les méthodes de résolution du problème MP, que nous détaillerons au fil de ce manuscrit, pourront être exemptes de toute discussion biologique.

Un débat récurrent au sein de la communauté des phylogénéticiens concerne le type d'analyse auquel soumettre les données. Il s'avère qu'une analyse cladistique est un moyen efficace de produire des arbres fiables lorsque les séquences sont relativement proches les unes des autres, ou que le nombre de sites examinés est important [Sourdis and Nei, 1988]. En revanche, le critère de parcimonie qui part du postulat que les substitutions sont peu probables, néglige entre autres la possibilité que des substitutions successives apparaissent sur une même arête de l'arbre. De même, les homoplasies (chapitre 1, figure 1.5) et réversions sont trompeuses au regard du critère de parcimonie. Un *index de consistance* a

d'ailleurs été introduit par [Kluge and Farris, 1969] pour tenir compte de ces phénomènes. De nombreuses études portent sur la comparaison des performances entre la vraisemblance et la parcimonie, et un certain nombre montrent que la parcimonie est dans la plupart des cas le critère le plus précis pour reconstruire les arbres d'évolution [Rice and Warnow, 1997 ; Kolaczkowski and Thornton, 2004].

Insistons malgré cela sur le fait que l'évolution des caractères suit un processus stochastique, et qu'aucune modélisation hypothétique ne peut vérifier toute observation du monde du vivant. Avec le critère de parcimonie, nous partons de l'hypothèse que le meilleur arbre est celui qui minimise les mutations, alors qu'il est fort possible que ce ne soit pas exactement le reflet de la réalité. Il nous parait néanmoins intéressant d'obtenir une vision formalisée d'un processus naturel, ce qui permet de fournir des informations que l'on n'avait pu inférer au préalable, et que l'on peut ensuite confronter à l'observation. Dans tous les cas, les méthodes utilisées ont pour but d'orienter la recherche, mais au final seul le spécialiste sera susceptible de confirmer, si ce n'est l'exactitude de la phylogénie, au moins sa concordance avec des connaissances établies.

Chapitre 3

Méthodes de Résolution pour le problème MP

PROPOSER UNE MÉTHODE DE RÉSOLUTION pour le problème MP qui soit efficace et relativement rapide quelle que soit l'instance à traiter est une tâche difficile. Ce chapitre dresse un panorama des principaux travaux réalisés dans le domaine.

Sommaire

3.1 Approches de résolution

Le chapitre précédent présentait le problème Maximum de Parcimonie, ou problème de parcimonie au sens large. La donnée du problème est un ensemble de séquences de même longueur, exprimé sous forme d'une matrice de caractères. Il s'agit de trouver un arbre binaire dont chaque feuille est associée à un taxon, et qui minimise le score de parcimonie. L'algorithme de Fitch calcule en temps polynomial le score d'un arbre, et le problème revient à trouver un arbre (*tous* les arbres, selon certaines versions) dont le score soit le plus petit possible. Il s'agit d'un problème hautement combinatoire, puisque le nombre d'arbres envisageables croît factoriellement en fonction du nombre de séquences. De plus, nous avons mentionné que ce problème est NP-complet[1]. Par conséquent, en conjecturant la vraisemblable inégalité $\mathcal{P} \neq \mathcal{NP}$, il n'y a aucun espoir de trouver un algorithme qui résolve le problème MP en temps polynomial par rapport à la taille des données.

3.1.1 Métaheuristiques

Les approches exactes pour résoudre le problème MP sont non-polynomiales, et en conséquence rapidement inexploitables (section 3.2). Ce constat établi, l'objectif est de concevoir un algorithme approché pour sa résolution. Pour cela, différentes classes de méthodes approchées fondamentales, appelées *métaheuristiques* [Reeves, 1993], sont appliquées, comme les algorithmes de recherche locale stochastique ou les algorithmes mémétiques. On y distingue notamment les *heuristiques de construction*, qui génèrent au mieux une solution au problème et les *heuristiques de voisinage*, procédures « intelligentes » qui n'envisagent qu'une infime partie des solutions possibles tout en maximisant les chances de rencontrer une solution au problème. Sans connaissance annexe, il est impossible de s'assurer que la solution retournée par une heuristique soit une solution optimale au problème ; il ne s'agit que d'une solution approchée — ce qui ne signifie pas systématiquement non-optimale.

La plupart des algorithmes heuristiques ne sont pas déterministes[2], c'est-à-dire que deux exécutions successives utilisant les mêmes données peuvent retourner deux résultats différents. Le (pseudo-)hasard, lorsqu'il est avantageusement utilisé, est une composante importante dans le succès d'une heuristique (calcul *stochastique*). Des algorithmes dits robustes retourneront toujours des résultats similaires ou de pertinence proche. Les algorithmes heuristiques sont évalués suivant leur efficacité (capacité à retourner des solutions approchées de bonne qualité), leur robustesse et leur rapidité. Bien qu'il soit préférable

[1] La classe de complexité $\mathcal{P}$ (pour *Polynomial*) regroupe les problèmes qui peuvent être résolus par un algorithme déterministe en un temps polynomial par rapport à la taille des données. Par définition, la classe de complexité $\mathcal{NP}$ (*Non déterministe Polynomial*) regroupe les problèmes de décision dont il existe un algorithme permettant de vérifier en temps polynomial qu'une solution convienne ou non. Par extension, pour les problèmes d'optimisation, il suffit de pouvoir vérifier (toujours en un temps polynomial) qu'une solution soit meilleure qu'une borne donnée. Enfin, parmi eux, les problèmes NP-complets sont les plus difficiles, ceux auxquels chaque problème NP peut leur être réduit polynomialement [Garey and Johnson, 1979 ; Papadimitriou, 1994].

[2] En pratique, le non-déterminisme de ces algorithmes est simulé par un paramètre annexe, car les connaissances actuelles ne permettent pas de concevoir un automate non déterministe.

de sacrifier en rapidité pour gagner en efficacité, les temps de calculs de ces méthodes approchées doivent rester raisonnables.

3.1.2 Espace de recherche

Les heuristiques de voisinage, qui sont au centre de nos motivations, utilisent comme référence un *espace de recherche*. Tout d'abord, les arbres phylogénétiques peuvent se définir comme des individus d'une population comportant toutes les solutions potentielles au problème (*configurations*). Nous avons vu précédemment (chapitre 1, propriété 2 et tableau 1.1) le nombre d'arbres distincts en fonction du nombre de feuilles identifiables. Nous pouvons représenter l'espace de recherche comme un ensemble de points identifiant les arbres, étiquetés par une valeur numérique entière qui désigne leur score de parcimonie respectif (figure 3.1). Résoudre le problème MP revient à identifier un des points associé à la plus faible valeur, voire à tous ces points si l'on souhaite répondre de manière exhaustive à la question posée.

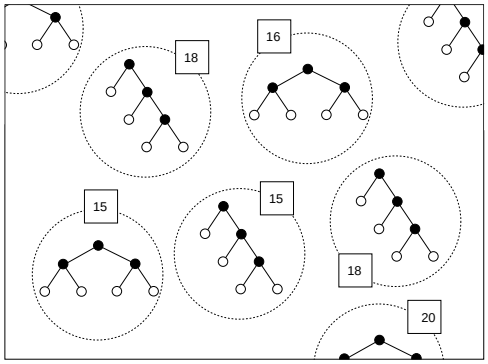


FIG. 3.1 – Espace de recherche d'arbres parcimonieux

Nous formulons alors le problème MP comme un problème combinatoire de minimisation $(\mathcal{T}, \phi)$, tel que :

1. l'espace de recherche $\mathcal{T}$ est défini par l'ensemble des configurations possibles, et
2. la fonction de cout, ou fonction objectif $\phi : \mathcal{T} \rightarrow \mathbb{N}$ est telle que $\forall t \in \mathcal{T}$, $\phi(t)$ représente le score de parcimonie de t.

La taille de l'espace de recherche associé à une instanciation de taille moyenne du problème MP est déjà astronomique — il existe autant d'arbres différents que d'atomes dans l'univers pour 50 taxons environ —, et les scores de parcimonie des individus sont inconnus au départ. Le rôle de l'heuristique est de trouver de *bons* individus en passant en revue quelques milliers ou millions d'entre eux seulement, sélectionnés par une heuristique. Pour cela on utilise une relation de *voisinage*, qui définit des correspondances entre individus. Ainsi l'heuristique de recherche, ou *mécanisme de voisinage*, naviguera dans l'espace de recherche selon des règles définies et suivant l'éventail des chemins tracés par le voisinage (définition 9 et figure 3.2).

Définition 9 (Voisinage [Hao *et al.*, 1999]) *Soit X l'ensemble des configurations admissibles d'un problème. On appelle* voisinage *toute application $\mathcal{N} : X \rightarrow 2^X$, et* mécanisme d'exploration du voisinage *toute procédure qui précise comment la recherche passe d'une configuration $s \in X$ à une configuration $s' \in \mathcal{N}(s)$. Une configuration s est un* optimum local *par rapport au voisinage $\mathcal{N}$ si $\forall s' \in \mathcal{N}(s), \phi(s) \leqslant \phi(s')$.*

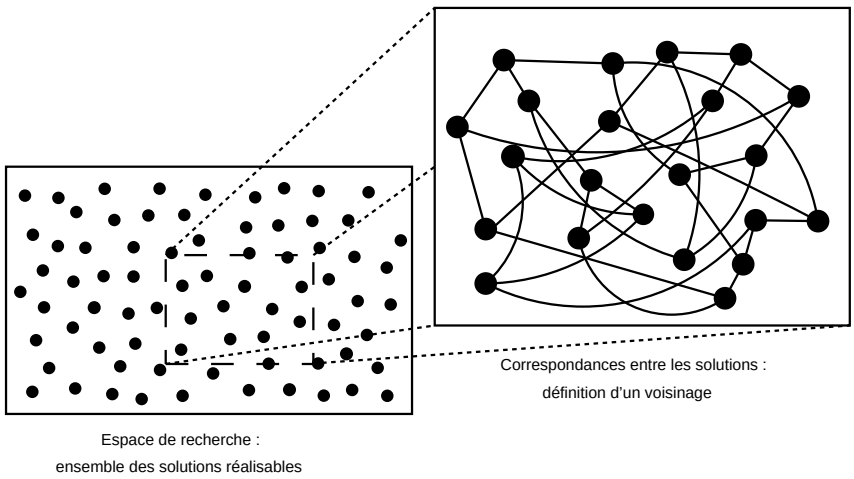


FIG. 3.2 – Espace de recherche et voisinage

Passé ce rappel des notions d'*heuristique*, d'*espace de recherche* et de *voisinage*, nous synthétisons dans ce chapitre l'ensemble des méthodes de résolution ayant été appliquées au problème MP, éventuellement combinées entre elles.

3.2 Méthodes exactes

3.2.1 Recherche exhaustive

Cette méthode complète consiste simplement à énumérer tous les arbres possibles, puis à calculer leur score de parcimonie. La recherche exhaustive permet de renvoyer tous les arbres dont le score de parcimonie est optimal. Cependant, le nombre d'arbres augmentant factoriellement en fonction du nombre d'espèces, cette méthode devient très vite inutilisable. En pratique, personne n'utilise la recherche exhaustive au-delà de dix taxons.

3.2.2 Branch and bound

L'algorithme du *branch and bound*, proposé initialement en programmation linéaire en nombres entiers [Land and Doig, 1960], est un grand classique pour résoudre de manière exacte les problèmes d'optimisation. Il s'agit d'une méthode générale, adaptable à de nombreux problèmes combinatoires. On le trouve pour la première fois appliqué au problème

MP dans [Hendy and Penny, 1982]. Il consiste à construire progressivement tous les arbres possibles par insertion de taxons, et abandonner la construction des arbres partiellement construits dès que le score de parcimonie dépasse une borne c, que l'on sait être supérieure au score de parcimonie optimal. La borne c est préalablement choisie en calculant le score d'un arbre de parcimonie aléatoire ou généré avec une heuristique, puis ajusté au cours du traitement.

Bien que les temps de calcul de l'algorithme du *branch and bound* dépendent largement du choix de cette borne [Swofford *et al.*, 1996], la NP-complétude du problème MP fait qu'il n'est pas concevable d'utiliser une méthode exacte pour sa résolution dès lors que le nombre de taxons à considérer dépasse la quinzaine. C'est pourquoi peu de logiciels communément utilisés pour la résolution du problème MP n'utilise de méthode exacte.

3.3 Méthodes de construction

Les heuristiques de construction, comme l'indique cette dénomination, construisent itérativement une solution approchée au problème. Dans le cas du problème MP, lorsque des heuristiques de construction sont utilisées, c'est pour servir de base à un algorithme de recherche locale (section 3.4). Des algorithmes de construction simples et rapides peuvent alors être préférés à d'autres plus efficaces mais plus couteux.

3.3.1 Méthodes de distances

Nous avons vu dans la section 1.3.1 que les méthodes de distances peuvent être utilisées à partir d'un ensemble de séquences de même longueur. Un algorithme de classification comme UPGMA peut donc très bien être utilisé pour construire une solution approchée au problème MP. Quelques essais suffisent à montrer que les scores de parcimonie des arbres construits de cette manière sont généralement plus élevés que ceux qui peuvent être atteints avec les autres méthodes. Seulement, cela permet d'obtenir rapidement une première solution acceptable dans le cadre d'une amélioration par recherche locale.

3.3.2 Algorithmes gloutons

En reconstruction phylogénétique, les algorithmes gloutons sont très utilisés, la plupart du temps sous l'appellation *stepwise addition*. Les taxons sont insérés un à un sur une branche d'un arbre partiel. Communément, avec les algorithmes gloutons purs (non couplés avec une heuristique de recherche locale), les insertions ont lieu aux endroits qui minimisent l'augmentation du score de parcimonie. Les algorithmes existants diffèrent alors essentiellement sur l'ordre dans lequel les taxons sont ajoutés. Ces derniers peuvent en effet être insérés dans l'ordre de la matrice de caractères donnée, dans un ordre aléatoire, ou bien de telle manière à ce que chaque insertion minimise ou maximise le score de parcimonie (dans ce dernier cas c'est la variation minimale qui est maximisée). Comme on l'imagine, seules certaines de ces variantes rendent l'algorithme non déterministe.

Peu d'articles traitent des différents algorithmes gloutons qui sont pourtant simples à implémenter et offrent un bon rapport temps de calcul / efficacité sur les instances

comportant quelques dizaines de taxons. [Andreatta and Ribeiro, 2002] proposent une comparaison de trois algorithmes gloutons de complexité et d'efficacité différentes :

1. `1stRotuGbr` sélectionne à chaque itération un taxon aléatoirement et l'insère à l'endroit qui minimise le score de parcimonie ;
2. `Gstep_wR` sélectionne un taxon et une branche de telle manière à ce que l'augmentation du score engendré par l'ajout de ce taxon à cette position soit minimale ;
3. `GRstep` est une variante de `Gstep_wR` où le couple taxon-branche sélectionné n'augmente pas le score de parcimonie de plus de 10% de celui engendré par le meilleur couple.

D'après leurs expérimentations, `Gstep_wR` est légèrement plus efficace que `1stRotuGbr` mais sa complexité accrue ($O(n^3)$ contre $O(n^2)$) augmente radicalement les temps de calculs. `GRstep` n'a d'intérêt que lorsqu'il est couplé avec une méthode de recherche locale (voir section suivante).

Lorsque les instances à traiter sont plus grandes (une ou plusieurs centaines de taxons), les algorithmes gloutons sont trop gourmands en temps de calcul en regard de leur efficacité. Utilisés seuls, il est en effet peu probable qu'ils approchent les résultats obtenus par d'autres algorithmes heuristiques.

3.4 Recherche locale stochastique

Considérant les très larges espaces de recherche, il se vérifie empiriquement que les heuristiques de recherche locale stochastique (ou *méthodes de voisinages*) sont particulièrement adaptées au problème MP, à la condition d'utiliser un voisinage approprié [Ganapathy *et al.*, 2004].

3.4.1 Descente pure et voisinages : *branch-swapping*

Un algorithme de descente pure consiste à générer un premier arbre, puis à rechercher un arbre voisin (au sens d'une relation de voisinage) dont le score est inférieur, et ainsi de suite jusqu'à ce que l'arbre courant n'ait aucun voisin dont le score soit strictement inférieur (algorithme 3.1). La solution finale est alors optimale par rapport à ses voisins (optimum local), mais pas nécessairement optimale au problème. Il existe plusieurs manières de sélectionner un voisin : soit en sélectionnant le premier voisin trouvé de score inférieur (méthode *first improve*, illustrée par l'algorithme 3.1), soit en calculant tous les voisins avant de sélectionner le meilleur, s'il améliore l'arbre courant (méthode *best improve*, bien plus couteuse lorsque le voisinage est large). Suivant le voisinage utilisé, certains algorithmes sont capables de calculer efficacement un sous-ensemble de voisins, et sélectionnent alors le meilleur d'entre eux ; il s'agit alors d'une alternative intermédiaire.

L'approche de descente, qui est la méthode de recherche locale la plus simple, dépend essentiellement de la relation de voisinage à laquelle elle est associée. Même s'il existe de nombreuses techniques pour tenter d'améliorer la qualité des solutions fournies par les algorithmes de descente, ces derniers sont à la base de toutes les méthodes efficaces de résolution du problème MP.

Algorithme 3.1 : Algorithme de descente

Données : une matrice de caractères $\mathcal{A}$, un voisinage $\mathcal{N}$
Résultat : un arbre t correspondant à un optimum local
début
  construire un arbre initial t ;
  tant que t *n'est pas un optimum local* **faire**
    sélectionner $t' \in \mathcal{N}(t)$;
    si $\phi(t') < \phi(t)$ **alors**
      $t \leftarrow t'$;
    fin
  fin
  retourner t ;
fin

Dans la littérature, on retrouve majoritairement trois voisinages d'arbres : NNI, SPR et TBR. Lorsque l'on évoque l'un de ces voisinages en reconstruction phylogénétique, cela sous-entend la plupart du temps que l'on parle de descente utilisant ce voisinage. Les termes descente, recherche locale et voisinage ne sont d'ailleurs pas utilisés dans tous les articles et ouvrages traitant du sujet, car certains des auteurs sont avant tout biologistes. La plupart du temps, on parle alors de *branch-swapping*.

Dans la suite de cette section, nous passons en revue les voisinages d'arbres existants proposés pour la résolution du problème MP.

Nearest Neighbor Interchange

NNI (*Nearest Neighbor Interchange*) [Waterman and Smith, 1978] consiste à échanger deux sous-arbres séparés par une arête. C'est un voisinage restreint de taille linéaire par rapport à la taille de l'arbre ($O(n)$), car un arbre à n feuilles compte $2n - 6$ voisins NNI [Robinson, 1971] ($n - 3$ arêtes internes, et deux mouvements possibles par arête).

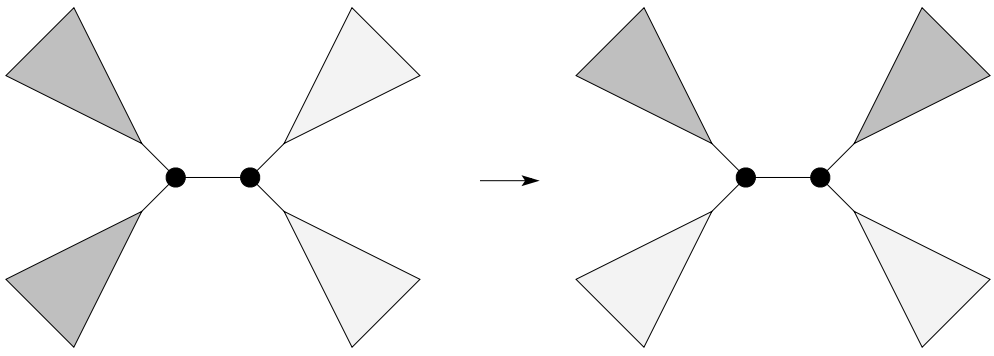


Fig. 3.3 – Transformation NNI

Subtree Pruning Regrafting

Un mouvement SPR (*Subtree Pruning Regrafting*, ou *Subtree-Prune-and-Regraft*) [Swofford and Olsen, 1990] détache un sous-arbre (supprime une arête existante) et le réinsère ailleurs (crée une nouvelle arête). À partir de chaque arbre, il existe $2(n-3)(2n-7)$ réarrangements SPR possibles [Allen and Steel, 2001] ; la taille du voisinage est en $O(n^2)$. Connaissant le score et les séquences de parcimonie d'un arbre, il n'est pas nécessaire d'appliquer globalement l'algorithme de Fitch pour calculer le score d'un voisin SPR. Il est d'usage de ne recalculer uniquement que les séquences de parcimonie des nœuds situés sur les chemins allant de la racine vers : (1) l'ascendant du nœud supprimé, et (2) le nœud créé lors de l'insertion. [Goloboff, 1993] propose une technique qui optimise le recalcul du score de plusieurs voisins. Lorsqu'un sous-arbre est détaché, l'algorithme effectue un pré-traitement sur les deux sous-arbres résultants tel que l'évaluation de chaque insertion devienne pratiquement immédiate ($O(m)$ au lieu de $O(n.m)$). D'autres solutions similaires on été proposées par [Gladstein, 1997 ; Yan and Bader, 2003].

Tree-Bisection-Reconnection

TBR (*Tree-Bisection-Reconnection*) [Swofford and Olsen, 1990] est un voisinage plus large qui casse l'arbre en deux sous-arbres, ensuite reconnectés à partir d'une de leurs arêtes. Ici, le nombre de voisins dépend de la topologie de l'arbre, mais il est d'au moins $(2n-3)(n-3)^2$, soit $O(n^3)$ [Allen and Steel, 2001]. Notons que les optimisations du recalcul du score utilisées pour SPR s'adaptent également à ce voisinage. Une représentation possible des transformations SPR et TBR est indiquée figure 3.4.

Nous pouvons remarquer que $NNI \subseteq SPR \subseteq TBR$[3] [Maddison, 1991]. Dans chaque logiciel de reconstruction phylogénétique basé sur le *branch-swapping*, au moins un de ces trois voisinages imbriqués est utilisé. Hors de ce cadre spécifique, ces voisinages sont très étudiés en théorie des graphes, et de nombreux travaux traitent de leurs espaces de recherche associés [Allen and Steel, 2001 ; Bastert *et al.*, 2002 ; Bryant, 2004 ; Bordewich and Semple, 2004]. Pouvoir prédire et localiser les optimums locaux constituerait un grand pas en avant dans la conception des algorithmes de reconstruction.

Ces dernières années, une volonté particulière d'améliorer l'efficacité de la recherche locale a amené certains auteurs à proposer d'autres voisinages :

Single Step

STEP [Swofford and Olsen, 1990 ; Andreatta and Ribeiro, 2002] est un sous-voisinage de SPR. La condition supplémentaire est que le sous-arbre détaché doit être un taxon. En d'autres termes, une transformation STEP consiste à déplacer une feuille dans l'arbre. Chaque configuration possède $2n(n-3)$ voisins STEP.

[3] Étant donnés deux voisinages $\mathcal{N}^1$ et $\mathcal{N}^2 : X \to 2^X$, nous notons $\mathcal{N}^1 \subseteq \mathcal{N}^2$ si $\forall x \in X, \mathcal{N}^1(x) \subseteq \mathcal{N}^2(x)$.

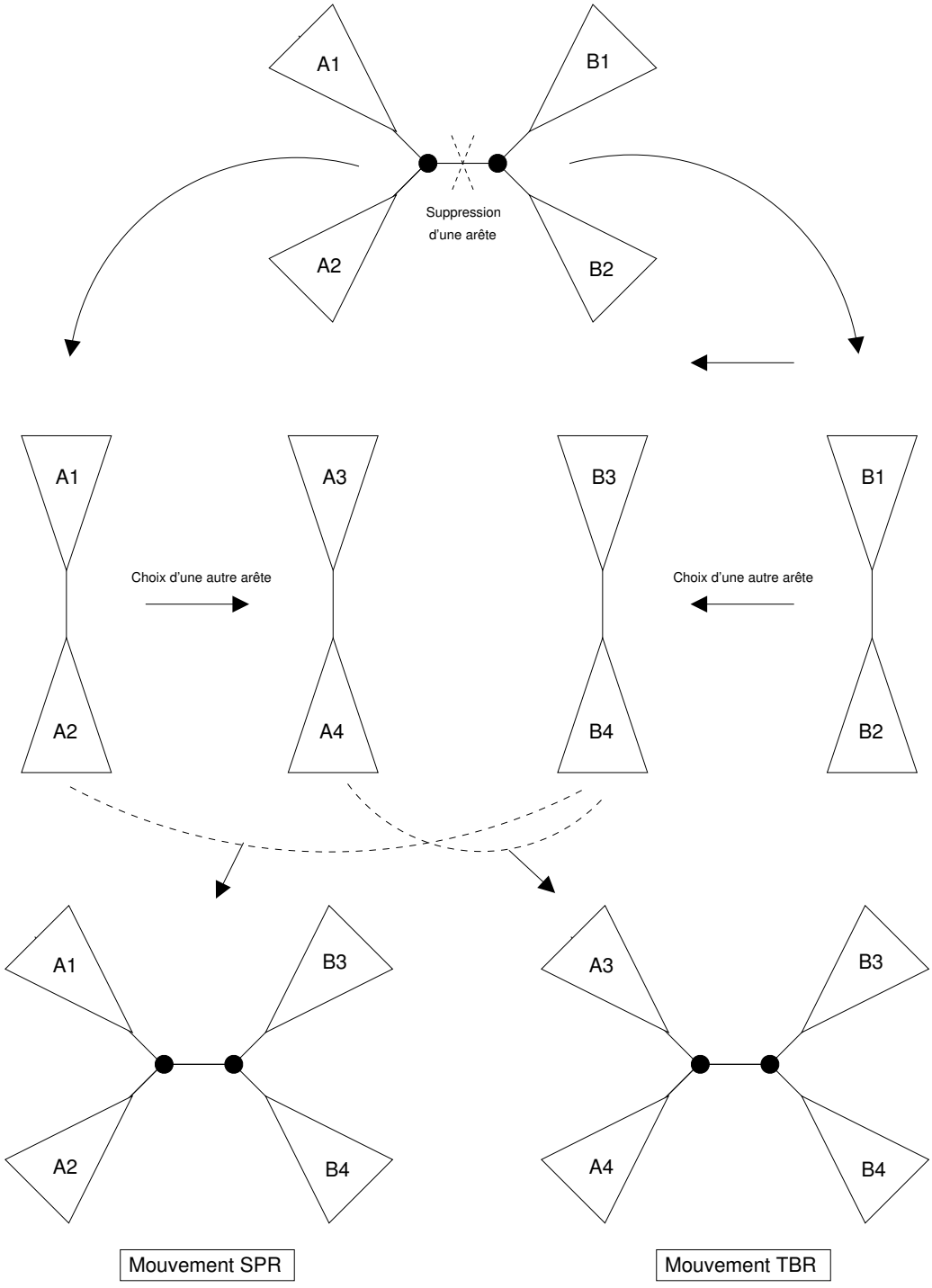


FIG. 3.4 – Transformations SPR et TBR

p-ECR

p-ECR [Ganapathy *et al.*, 2003] contracte p arêtes adjacentes, avant que le nœud de degré $p + 3$ né de cette contraction ne soit décomposé pour générer p nouvelles arêtes (*raffinement*). p-ECR est une généralisation de NNI dont il lui est fortement inspiré ; 1-ECR équivaut d'ailleurs au voisinage NNI. Les auteurs démontrent par un théorème que 2-ECR, de taille $O(n^2)$, possède un nombre de voisins en commun avec TBR de l'ordre de $O(n)$. Ce résultat n'a rien de surprenant, car ces deux voisinages généralisent NNI. En revanche, 2-ECR et TBR privés de NNI sont disjoints. Dans [Ganapathy *et al.*, 2003], l'étude s'est restreinte au cas où $p = 2$. Des propriétés plus générales ont été présentées dans [Ganapathy *et al.*, 2004], mais aucune application n'a été effectuée par les auteurs.

l-SPR

l-SPR [Ribeiro and Vianna, 2005] est un voisinage dont chaque transformation est une succession de l mouvements SPR. En pratique, l ne dépasse pas 2 car le voisinage, de taille $O(n^{2^l})$, devient vite trop large. l-SPR est utilisé dans le cadre d'une descente à voisinage variable (voir section 3.4.2), méthode généralisée par [Mladenović and Hansen, 1997].

En optimisation, les algorithmes de descente pure sont rarement efficaces pour résoudre les problèmes difficiles, ou pour trouver une solution approchée de bonne qualité. Les algorithmes de descente dépendent essentiellement du ou des voisinages utilisés ; mais même en utilisant un voisinage approprié, ils peuvent retourner des optimums locaux (solutions dont tous les voisins sont de cout égal ou supérieur) de mauvaise qualité. De nombreux algorithmes sont utilisés en optimisation pour sortir des optimums locaux : descente à voisinage variable, recuit simulé, recherche tabou, *etc.* [Hoos and Stützle, 2005]. Dans les sections suivantes nous présentons les algorithmes de recherche locale qui ont été appliqués avec succès au problème MP.

3.4.2 Descente à Voisinage Variable

Bien qu'il ne s'agisse pas de l'unique facteur, il apparait aisément que la probabilité de rencontrer tôt un optimum local est fortement corrélée à la taille du voisinage. Un algorithme de descente utilisant un voisinage trop étroit risque de retourner rapidement un optimum local de piètre qualité, tandis qu'un voisinage trop large apparenterait la recherche à un parcours aléatoire de l'espace.

Lorsque la descente retourne un optimum local, il existe plusieurs manières de s'en extraire. La plupart des techniques perturbent la configuration courante, ce qui a pour effet de détériorer son cout (voir section 3.4.3). L'unique moyen d'améliorer la configuration courante est de changer de voisinage. Ce concept fut formalisé en un cadre assez large nommé VNS (*Variable Neighborhood Search*) [Mladenović and Hansen, 1997 ; Hansen and Mladenović, 1999]. La descente à voisinage variable (*Variable Neighborhood Descent*) s'inscrit dans ce schéma. [Ribeiro and Vianna, 2005] en proposent une application au problème MP (algorithme 3.2). Leur algorithme obtient de bons résultats, mais dans des temps de calcul relativement élevés.

Algorithme 3.2 : Exemple de descente à voisinage variable

Données : une matrice de caractères $\mathcal{A}$,
un ensemble de voisinages $\mathcal{N}^1, \mathcal{N}^2, \ldots, \mathcal{N}^{l_{\max}}$
Résultat : un arbre t
début
  construire un arbre initial t ;
  $l \leftarrow 1$;
  tant que $l \leqslant l_{\max}$ **faire**
    choisir $t' \in \arg\min_\phi \mathcal{N}^l(t)$;
    si $\phi(t') < \phi(t)$ **alors**
      $t \leftarrow t'$;
      $l \leftarrow 1$;
    sinon
      $l \leftarrow l + 1$;
    fin
  fin
  retourner t;
fin

Plutôt que d'élargir le voisinage, nous proposons un mécanisme qui permet de le réduire au fur et à mesure de la recherche [Goëffon *et al.*, 2005b]. Il s'agit du concept de descente à voisinage progressif, qui fait l'objet du chapitre 5.

3.4.3 Recherche locale itérée et *Parsimony Ratchet*

Lorsque l'algorithme de descente reste bloqué dans un optimum local, il existe plusieurs moyens de s'en extraire afin de relancer la recherche, sans nécessairement changer de voisinage. Si certains algorithmes sont relancés avec une nouvelle solution initiale pour explorer une autre zone de l'espace de recherche, d'autres perturbent l'optimum avant de relancer la descente. C'est le cas de l'algorithme de *recherche locale itérée* (ILS pour *Iterated Local Search*) [Lourenço *et al.*, 2002].

Une technique souvent utilisée en optimisation consiste à naviguer de manière plus ou moins aléatoire dans l'espace de recherche depuis l'optimum local, puis de s'arrêter sur une configuration qui sera le point de départ d'une nouvelle descente. Les perturbations doivent être effectuées de manière judicieuse, afin de minimiser les chances de retrouver l'optimum obtenu précédemment lors de la prochaine phase de descente. Certains espaces de recherche sont structurés de telle manière qu'un nombre insuffisant de perturbations ou l'acceptation de voisins trop dégradés peut ne pas diversifier suffisamment la recherche pour rencontrer un optimum différent. En optimisation, il est souvent préférable d'utiliser un voisinage distinct (plus large ou dont l'intersection avec le voisinage courant est restreinte) pour la phase de perturbation. Ainsi, quelques pas aléatoires peuvent suffire à explorer une nouvelle zone de l'espace de recherche original tout en assurant à la nouvelle configuration initiale la conservation de certaines propriétés antérieures.

Bien que le terme ILS n'apparaisse pas dans les publications concernées, c'est un algorithme de descente avec perturbations qui figure en tête de liste des techniques les plus utilisées pour résoudre le problème MP. Présentée comme une nouvelle méthode par [Nixon, 1999 ; Horovitz, 1999] et améliorant sensiblement l'efficacité des algorithmes de *branch-swapping*, le *Parsimony Ratchet* est en fait une application directe de la méthode de bruitage proposée par [Charon and Hudry, 1993 ; Charon and Hudry, 2002].

Après avoir calculé un premier arbre de parcimonie t avec un algorithme de descente pure utilisant un voisinage $\mathcal{N}$, l'algorithme du *Parsimony Ratchet* sélectionne aléatoirement un certain nombre de sites de l'alignement initial (usuellement dans une proportion de 5 à 25%). Ces sites sont soit supprimés (pondération nulle, méthode statistique du *jackknife* [Tukey, 1958 ; Gray and Schucany, 1972]), soit dupliqués (poids double), et l'alignement bruité obtenu sert de nouvelle référence, comme pour les techniques de bootstrap (section 1.3.4). Le score de parcimonie de l'arbre t sera *a priori* différent de celui calculé à partir des séquences initiales. La fonction d'évaluation initiale calculant le score de parcimonie, que l'on note ϕ, est modifiée en ϕ^* par les données bruitées. Cette nouvelle fonction d'évaluation est utilisée pour une seconde phase de recherche locale, où l'arbre t est amélioré en un arbre t^* au sens de ϕ^*, mais normalement dégradé selon ϕ. Pour perturber t en t^*, l'algorithme a utilisé le même voisinage $\mathcal{N}$ que pour la phase d'amélioration, seule la fonction d'évaluation a été modifiée par bruitage de la matrice de caractères. L'algorithme utilise ici un espace de recherche possédant toujours la même structure : en considérant cet espace de recherche comme un graphe étiqueté par les variations de cout entre les arbres, seul le poids des arêtes change pendant les phases de perturbations. Un certain nombre d'itérations descente-perturbation est effectué, et chaque phase de perturbation s'effectue avec une nouvelle fonction ϕ^*. L'arbre retourné au final est le meilleur arbre rencontré durant la recherche, au sens de ϕ.

[Quicke *et al.*, 2001] ont développé indépendamment un algorithme très proche. L'unique différence provient de la manière de pondérer les caractères. Cette pondération n'est pas effectuée de manière aléatoire, mais en fonction de l'importance de chaque caractère dans le score de parcimonie des arbres préalablement calculés.

3.4.4 Recuit simulé

Introduit par [Kirkpatrick *et al.*, 1983] et de manière indépendante par [Černý, 1985], le recuit simulé est une des méthodes de recherche locale les plus connues et utilisées. Son fonctionnement, inspiré du recuit physique en métallurgie [Metropolis *et al.*, 1953], est relativement simple. Il s'agit d'une méthode de voisinage dont le principe est basé sur celui de la descente. À chaque itération, une configuration voisine (sélectionnée aléatoirement) $t' \in \mathcal{N}(t)$ est estimée à l'aide d'une fonction d'évaluation ϕ, puis acceptée sous certaines conditions. Alors que pour la descente, seules les modifications qui améliorent la configuration courante sont acceptées, ici certaines détériorations sont admises suivant une certaine probabilité, fonction de l'importance de la dégradation et de l'avancée de la recherche. Si $\phi(t') < \phi(t)$, alors t' remplace t et devient la configuration courante. Dans le cas contraire, t' est accepté selon une probabilité $\exp(-\frac{|\phi(t')-\phi(t)|}{\tau})$. Le paramètre τ est appelé température, et communément décroit au fur et à mesure de la recherche. En fin

de recherche, lorsque τ est très proche de 0, plus aucune dégradation n'est autorisée et l'algorithme se comporte comme une descente pure. C'est généralement à ce moment que l'algorithme se termine.

Le recuit simulé évite de stagner dans un optimum local de mauvaise qualité, ce qui entraine classiquement une robustesse et un net gain d'efficacité par rapport à la descente classique. Il a été utilisé avec succès pour bon nombre de problèmes combinatoires [Bonomi and Lutton, 1984 ; Vidal, 1993]. En revanche, il est bien moins rapide qu'une descente pure et, surtout, son paramétrage est très délicat. Il faut définir une fonction de réduction de la température, une valeur initiale de la température τ_0 et une condition d'arrêt qui permette de maximiser l'efficacité du recuit. De nombreuses expérimentations sont nécessaires à l'affinage de ces paramètres, et leur adéquation est souvent dépendante de l'instance considérée. Le fonctionnement du recuit simulé est relativement proche de l'algorithme de recherche locale appelé *marche aléatoire* (*random walk*), qui se comporte comme une descente tout en générant des mouvements aléatoires avec une probabilité de bruit constante.

Il existe peu d'applications du recuit simulé à la reconstruction phylogénétique : `LVB` [Barker, 2004] pour le problème MP, et `RAxMP-SA` [Stamatakis, 2005] pour le problème ML. Nous avons également réalisé un algorithme de recuit simulé pour le problème MP, et comparé son efficacité avec d'autres méthodes de recherche locale, comme la recherche locale itérative ou la marche aléatoire [Goëffon *et al.*, 2005a]. Ces travaux sont évoqués dans le chapitre 4.

3.4.5 Hybridation *stepwise addition* et *branch-swapping*

Le problème initial des algorithmes gloutons de type *stepwise addition* est de ne pas remettre en cause la position des taxons nouvellement insérés. La plupart du temps, lorsqu'un taxon est inséré, des positions alternatives de taxons précédemment ajoutés se seraient révélées meilleures ; et ce même si chaque nouvelle insertion minimise l'augmentation du score de parcimonie de l'arbre partiel.

C'est pourquoi, à plusieurs reprises durant la formation de l'arbre (éventuellement après chaque taxon inséré), une phase de *branch-swapping* peut être réalisée afin de rectifier certains choix et ainsi tenter d'améliorer le cout de l'arbre.

Nous retrouvons cette procédure lors de la première étape des algorithmes de type `GRASP` (*Greedy Randomised Adaptive Search Procedure*) [Feo and Resende, 1995]. De plus, dès que la solution est entièrement construite, une phase plus intensive de recherche locale est appliquée. Plusieurs solutions sont obtenues de cette manière (construction avec un algorithme glouton + recherche locale), et la meilleure est retournée. L'algorithme `GRASP` a été appliqué au problème MP dans [Ribeiro and Vianna, 2005].

3.5 Algorithmes mémétiques

En optimisation, certains algorithmes sont inspirés de phénomènes naturels. C'est le cas de l'*optimisation par colonies de fourmis* [Dorigo and Caro, 1999], métaphore où le comportement et la communication des fourmis par le biais de phéromones est reproduit

pour la résolution de problèmes combinatoires. Quant aux *algorithmes génétiques*, introduits par [Holland, 1975] puis vulgarisés par [Goldberg, 1989], ils tirent leur inspiration du processus de sélection naturelle.

Le schéma général des algorithmes génétiques est simple. On dispose d'une population de configurations appelées individus. Ceux-ci peuvent se croiser (se reproduire), muter, et disparaitre au fil des générations. Les descendants (fils) héritent de spécificités propres à leurs parents, et les individus les plus adaptés ont de meilleures chances de survie. L'efficacité d'un algorithme génétique est régie par un grand nombre de paramètres : la taille de la population et la méthode de génération des individus initiaux, la sélection des individus à recombiner, le mécanisme de croisement, l'opérateur de mutation (généralement appliqué aux fils), ainsi que la condition d'insertion des fils et *a fortiori* la mise à jour de la population. Le codage des individus, dont dépend les possibilités et la complexité des croisements, est également un choix déterminant [Michalewicz, 1996].

Dans la plupart des cas, comme le rappellent [Hoos and Stützle, 2005], un algorithme génétique défini comme tel n'est pas suffisamment efficace, car les opérateurs de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche. L'opérateur de mutation, typiquement, n'est censé apporter qu'une légère modification à l'individu. Son rôle est de favoriser la diversification des individus alors que la sélection se charge de conserver les meilleurs. C'est pourquoi les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale. L'opérateur de recherche locale remplace ou succède à la mutation, et permet d'intensifier la recherche dans diverses zones pointées par les mécanismes génétiques (sélections, croisement) ; on parle alors d'*algorithme mémétique* [Moscato, 1999] ou plus marginalement de *recherche locale génétique* [Ulder *et al.*, 1991 ; Kolen and Pesch, 1994]. Les possibilités d'hybridation entre les algorithmes génétiques et de recherche locale sont multiples, comme le montrent [Nagata and Kobayashi, 1997] en incorporant la recherche locale dans l'opérateur de croisement.

De nombreuses applications ont prouvé à quel point les algorithmes mémétiques pouvaient être performants, sur des problèmes bien connus comme celui du voyageur de commerce [Freisleben and Merz, 1996], du sac à dos [Falkenauer, 1996], de l'affectation quadratique [Merz and Freisleben, 1997], de coloration [Dorne and Hao, 1998 ; Galinier and Hao, 1999] ou de satisfiabilité [Lardeux *et al.*, 2006]. Quelques uns ont été appliqués à la reconstruction phylogénétique : [Matsuda, 1996 ; Lewis, 1998] pour le problème ML, [Moilanen, 1999 ; Congdon, 2002 ; Congdon and Septor, 2003 ; Ribeiro and Vianna, 2003] pour le problème MP, et [Cotta and Moscato, 2002] pour l'analyse phénétique (méthodes de distances).

Les opérateurs de croisement définis dans ce cadre recourent souvent à la même stratégie, sans doute inspirée des croisements monopoints de chaines de bits, et étendue à un mouvement SPR généralisé sur deux arbres. Étant donné deux arbres parents, l'opération dans sa forme générale consiste en premier lieu à sélectionner un sous-arbre depuis un des parents (*donneur*). Les feuilles correspondant aux taxons de ce sous-arbres sont supprimées dans le second parent (*receveur*) pour retourner un fils intermédiaire, auquel on identifie une arête que nous appellerons *point de fusion*. L'arbre fils est obtenu en connectant le sous-arbre du donneur au point de fusion du fils intermédiaire. Le fils dépend du point de coupe du donneur et du point de fusion du receveur ; il est également possible de générer

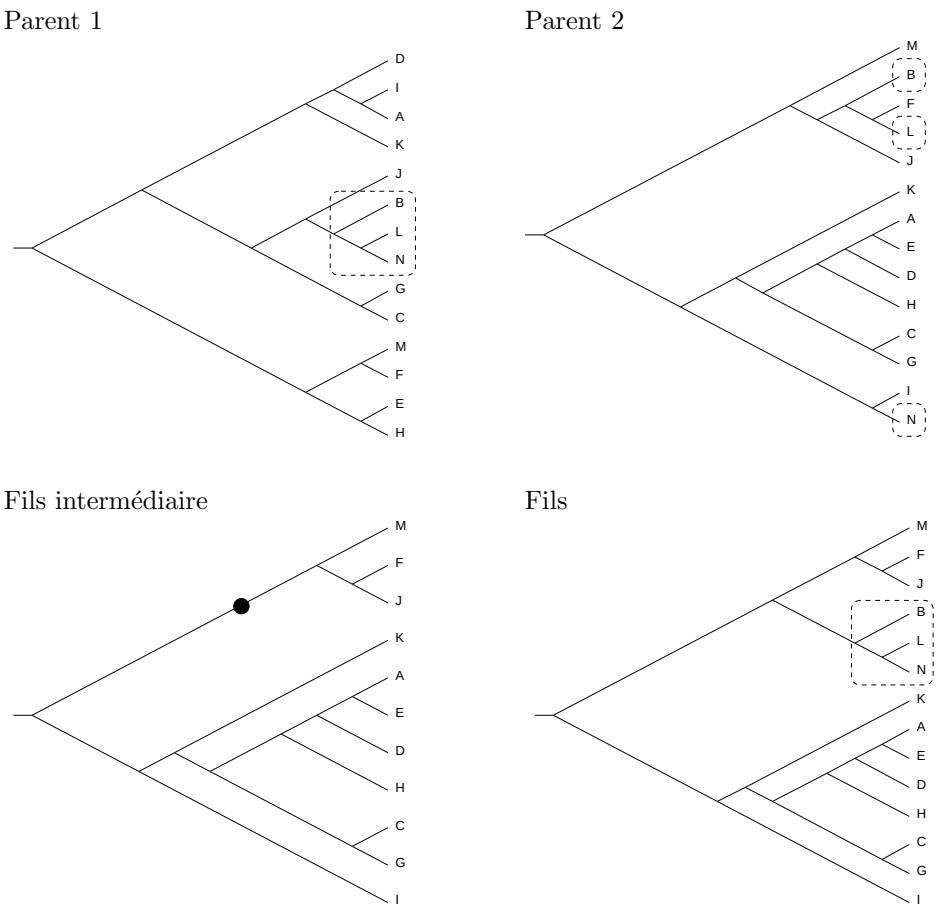


FIG. 3.5 – Exemple de croisement d'arbres couramment utilisé

des fils différents en inversant les rôles (donneur / receveur) des deux parents.

La figure 3.5 montre un exemple avec 14 taxons, où le sous-arbre $(B,(L,N))$ est coupé du premier parent et réinséré dans le second parent entre la racine et le sous-arbre $((F,J),M)$ après suppression des trois espèces B, L et N.

Nous montrons dans [Goëffon *et al.*, 2006] qu'avec ce croisement et ses variantes, trop peu d'information est transmise des parents vers le fils. Dans le chapitre 6, nous présentons un croisement original qui permet d'obtenir de meilleurs résultats que les croisements d'arbres classiques.

3.6 Algorithmes de décomposition

Les algorithmes de recherche locale et mémétiques constituent un cadre adapté à la résolution d'instances de taille moyenne à grande (moins de 500 taxons). Mais en traitant

des instances comportant des milliers de taxons, la résolution devient fortement problématique. Les données en provenance des biologistes sont de plus en plus grandes : 500 taxons [Chase *et al.*, 1993], puis 2 538 [Källersjö *et al.*, 1998] ou plus récemment 13 921 [Maidak *et al.*, 2000]. Il est évident qu'aucune heuristique de recherche ne peut traiter cette dernière instance dans sa globalité, la taille de l'espace de recherche dépassant $10^{55\ 000}$. Pour contrer cet obstacle, il est nécessaire de décomposer les instances en sous-instances (sous-ensembles de taxons qui se chevauchent), puis de recombiner les informations pour obtenir une solution unique au problème initial. Il s'agit du concept des *superarbres* [Gordon, 1986 ; Bininda-Emonds, 2002 ; Bininda-Emonds, 2004], déclinaison des algorithmes de type *diviser pour régner* et qui se pose comme un problème propre. Plusieurs techniques sont utilisées pour combiner les arbres issus de la décomposition de l'instance ; la plus utilisée est la représentation matricielle par parcimonie (MRP pour *Matrix Representation using Parsimony* [Bininda-Emonds, 1998]) basée sur les travaux indépendants de [Baum, 1992] et de [Ragan, 1992].

Les travaux sur les superarbres, nécessaires pour appréhender les très larges instances, sont complémentaires à ceux portant sur les méthodes globales, comme les heuristiques de recherche locale. Les algorithmes de décomposition divisent le problème en sous-problèmes et utilisent des algorithmes de reconstruction classiques pour les résoudre. Des deux côtés, les recherches connaissent un fort engouement depuis quelques années, à l'heure où de nombreux efforts sont concentrés dans la réalisation de l'*Arbre de la Vie* (*Tree of Life*) [Pennisi, 2003].

Pour des instances larges, [Snell *et al.*, 2000] et [Du *et al.*, 2005] proposent quant à eux une parallélisation d'algorithmes existants.

3.7 Principaux logiciels

La plupart des logiciels de résolution du problème MP sont conçus pour et par des biologistes, et les publications qui leur sont associés paraissent généralement dans des revues biologiques (*Cladistics*, *Molecular Biology and Evolution*, *Journal of Molecular Evolution*, *Systematic Biology*, *etc*). C'est pourquoi les algorithmes des logiciels couramment utilisés sont rarement décrits dans la littérature, et il est souvent difficile de connaitre les méthodes employées avec exactitude. C'est le cas du logiciel `PAUP*` [Swofford, 2002], très utilisé par les biologistes. `TNT` [Goloboff *et al.*, 2000] (dérivé de `NONA` [Goloboff, 1997]), réputé comme étant le logiciel de MP disponible le plus efficace avec `PAUP*`, ne comporte pas moins de seize nouvelles techniques originales ou combinées, regroupées sous l'appellation *New Technology* et évoquées uniquement dans [Goloboff, 1999] sans que le bien-fondé de telle ou telle méthode ne soit justifié. Parmi elles, les *recherches sectorielles* sont des recherches locales appliquées à des sous-arbres, la *fusion d'arbre* (*tree-fusing*) teste des échanges valides entre des sous-arbres de différents arbres et le *tree-drifting* est une opération de perturbation similaire à celles utilisées en recherche locale itérée. Au final, `TNT` est une machinerie qui permet de passer en revue des millions d'arbres par seconde, dont [Hovenkamp, 2004] recense 117 commandes, la plupart comportant entre 5 et 20 paramètres.

Cependant, il est connu que ces programmes utilisent en grande partie le *branch-*

swapping (sur les trois principaux voisinages NNI, SPR et TBR). Précisons que `PAUP*` et `TNT` ne sont pas des logiciels libres, contrairement au vénérable `DNAPARS` du *package* `PHYLIP` [Felsenstein, 1989].

3.8 Conclusion

Tous les algorithmes de résolution du problème MP actuels utilisent des méthodes de voisinage simples, typiquement la descente pure ou bruitée. Celles-ci sont incluses dans des algorithmes plus élaborés : algorithmes mémétiques, GRASP, de décomposition ou parallélisés. En dressant cet état de l'art, il nous est clairement apparu que différents axes de recherche, complémentaires, pouvaient être envisagés. Dans la seconde partie de cette thèse, qui traite des méthodes globales de résolution du problème MP, nous proposons des solutions alternatives qui visent en un premier temps à rendre les algorithmes de descente plus efficaces, puis à utiliser ces derniers au mieux par le biais d'un algorithme mémétique.

Deuxième partie

Nouvelles heuristiques pour le problème MP

Chapitre 4

Étude empirique de métaheuristiques de voisinage

Beaucoup de logiciels résolvent le problème MP en utilisant des techniques de recherche locale. Cependant, il s'agit la plupart du temps, et ce pour les logiciels les plus efficaces, de combinaisons d'heuristiques simples, appliquées sans pertinence particulière mais fortement optimisées afin d'explorer intensivement l'espace de recherche. Il n'existe en revanche aucune étude empirique des métaheuristiques de voisinage appliquées au problème MP. Dans ce chapitre, nous nous sommes attachés à implémenter des métaheuristiques traditionnelles de recherche locale, et à étudier expérimentalement les choix efficaces en fonction des méthodes ou des voisinages employés. Cette étude préliminaire nous a permis d'orienter notre recherche afin de concevoir, dans un second temps, des algorithmes plus compétitifs.

Ce chapitre assure la continuité du précédent, et notamment de la section 3.4, en montrant expérimentalement l'efficacité de certaines métaheuristiques appliquées au problème MP. Une partie de cette étude a été publiée dans [Goëffon *et al.*, 2005a].

Sommaire

4.1 Fonctionnement général des algorithmes de recherche locale

Un algorithme de recherche locale est défini par trois éléments :

1. Une **fonction d'évaluation** f qui estime la qualité de chaque configuration ;
2. Une **relation de voisinage** $\mathcal{N}$ qui génère des configurations voisines ;
3. Une **stratégie de mouvement** qui décide de l'acceptation ou du rejet d'une configuration voisine.

Typiquement, un tel algorithme débute avec une configuration initiale issue de l'espace de recherche et procède à une série d'itérations. À chaque itération, un voisin de la configuration courante est généré selon la relation de voisinage $\mathcal{N}$, et évalué par f avant d'être accepté pour remplacer la configuration courante, ou rejeté par la stratégie de mouvement. Plus précisément, un algorithme de recherche locale visite une série de configurations $(c^0, c^1, \ldots, c^i, c^{i+1}, \ldots)$, tel que c^0 est la configuration initiale et $\forall i, c^{i+1} \in \mathcal{N}(c^i)$.

L'algorithme s'arrête lorsqu'une condition préfixée est vérifiée ; par exemple quand un nombre maximum d'itérations ou une solution de qualité jugée suffisamment bonne est atteinte. Il retourne alors la *meilleure* solution trouvée durant la recherche. Rappelons que cette *meilleure* solution n'est pas nécessairement une solution optimale au problème.

La recherche locale vise à trouver rapidement une solution de bonne qualité pour un problème donné. Ce type d'algorithme est très utilisé lorsque l'on s'attaque à des problèmes NP-complets pour lesquels des solutions optimales peuvent être trouvées avec certitude uniquement pour des instances de petite taille.

4.2 Application de la recherche locale au problème MP

4.2.1 Schéma général

Le problème MP étant NP-complet [Foulds and Graham, 1982], les approches de résolution par recherche locale semblent bien appropriées pour trouver des solutions approchées quelle que soit l'instance à traiter. Nous avons transcrit le problème MP comme problème combinatoire de minimisation $(\mathcal{T}, \phi)$ dans la section 3.1.2 du chapitre 3. Dans ce qui suit, la fonction d'évaluation utilisée dans tous les algorithmes sera la fonction objectif ϕ du problème, qui s'attache à calculer le score de parcimonie des arbres ; elle sera notée en conséquence ϕ. En effet, aucune définition alternative d'une fonction d'évaluation ne nous a semblée judicieuse dans le cadre de ce problème — et il n'en existe aucune dans la littérature. L'algorithme 4.1 montre notre approche générale de recherche locale appliquée au problème MP.

Dans les sections suivantes, nous présentons les différents éléments qui constituent les algorithmes de recherche locale.

Algorithme 4.1 : Recherche locale pour le problème MP

Données : une matrice de caractères $\mathcal{A}$ induisant un problème de minimisation $(\mathcal{T}, \phi)$.
Résultat : le meilleur arbre trouvé.
début
  générer un arbre initial $t \in \mathcal{T}$ (aléatoirement ou par une heuristique de construction rapide) ;
  $b \leftarrow t$ # b stocke le meilleur arbre trouvé jusqu'ici ;
  tant que *la condition d'arrêt n'est pas satisfaite* **faire**
    générer un voisin t' de l'arbre courant t selon une relation de voisinage $\mathcal{N}$ ($t' \in \mathcal{N}(t)$) ;
    calculer $\phi(t')$ # évaluer la qualité de t' par la fonction d'évaluation ϕ;
    si *la stratégie de mouvement accepte* t' **alors**
      $t \leftarrow t'$;
    fin
    si $\phi(t) < \phi(b)$ **alors**
      $b \leftarrow t$ # mettre à jour la meilleure solution trouvée;
    fin
  fin
  retourner b ;
fin

4.2.2 Construction de l'arbre initial

Une première possibilité consiste à débuter la recherche locale avec un arbre construit par un processus aléatoire. Nous utilisons une méthode simple qui construit progressivement l'arbre, selon un cheminement identique à celui de l'algorithme UPGMA (chapitre 2, section 1.3.1), mais sans se référer à une matrice de distances. L'algorithme suit un processus itératif. Tout d'abord, deux taxons sont sélectionnés aléatoirement parmi les données selon une distribution uniforme, puis regroupés en créant un nouveau nœud parent qui représente l'ancêtre commun hypothétique des deux taxons. Ce nœud remplace ses deux descendants dans l'ensemble contenant initialement tous les taxons (futures feuilles de l'arbre), afin de continuer la construction progressive de l'arbre. Celle-ci s'arrête lorsque les deux derniers nœuds (dans ce cas les dernières classes) sont regroupés par un nœud qui devient la racine de l'arbre. Cet algorithme a une complexité de $O(n)$, où n est le nombre de taxons.

Les heuristiques qui construisent des arbres initiaux de meilleur cout (méthodes de distances, algorithmes gloutons) sont détaillées dans la section 3.3 du chapitre 3.

4.2.3 Voisinages

En plus des trois voisinages d'arbres NNI, SPR et TBR, nous avions préalablement défini dans le cadre de cette étude un voisinage de taille quadratique comme l'est SPR. Ce voisinage, appelé SSN (*Subtree Swapping Neighborhood)* [Goëffon *et al.*, 2005a] consiste

simplement à échanger deux sous-arbres de l'arbre courant. Il possède quelques propriétés immédiates qui justifiaient de prime abord son utilisation, comme la propriété $SSN = SSN^{-1}$ qui permet de rejeter un voisin en répétant la dernière transformation effectuée. Les performances du voisinage SSN, bien que proches de celles du voisinage SPR sur les instances les plus petites, se détériorent en fonction de la taille des instances. Il s'avère en fait que dans le cas du problème MP, le voisinage SPR — qui consiste à déplacer un sous-arbre — a une sémantique beaucoup plus forte que SSN — qui déplace les sous-arbres par deux et aux mêmes positions. Cette transformation perturbe beaucoup plus la solution courante, ce qui réduit les possibilités d'améliorations en fin de recherche. Cette piste sur l'importance de la sémantique des voisinages utilisés et leurs effets en fonction de la qualité de l'arbre courant fait l'objet des deux chapitres suivants.

4.2.4 Représentation interne des arbres

Le codage des séquences de parcimonie et des topologies d'arbres est un point important dans l'implémentation et la rapidité des algorithmes de reconstruction. Cette courte section ne s'attache pas à détailler les implémentations des arbres et des algorithmes de recherche locale, mais à mentionner quelques enseignements que nous avons pu tirer de nos expérimentations.

Caractères

Nous représentons les caractères des séquences de parcimonie par des entiers. Si Σ est l'alphabet des états possibles pris pour chaque caractère, et puisqu'un caractère peut prendre plusieurs états dans le cas d'indéterminations, il doit exister une bijection entre l'ensemble des entiers utilisés pour coder les caractères, et l'ensemble des parties de Σ, noté $\mathcal{P}(\Sigma)$.

Soit ψ une bijection de Σ vers $\{1, \ldots, |\Sigma|\}$. Nous pouvons alors définir une bijection $\Psi : \mathcal{P}(\Sigma) \rightarrow \{0, \ldots, |\mathcal{P}(\Sigma)| - 1\}$ telle que :

$$\Psi(c) = \sum_{i=1}^{|\Sigma|} a_i(c).2^{\psi^{-1}(i)}, \text{ avec } a_i(c) = \begin{cases} 1, \text{ si } \psi^{-1}(i) \in c \\ 0, \text{ sinon} \end{cases}$$

De cette manière, chaque ensemble d'états c peut être codé par un entier naturel, *i.e.* par un vecteur de bits, et les millions voire milliards d'opérations ensemblistes que nécessitent les recours à l'algorithme de Fitch peuvent être traitées rapidement par des opérations binaires et des décalages de bits.

Topologies

Il existe traditionnellement deux manières de représenter les arbres binaires : soit par représentation matricielle, soit par pointeurs. Nous avons réalisé deux implémentations dans le cadre de cette étude, en utilisant pour chacune une représentation distincte. Celle par pointeurs s'avère être plus efficace au niveau des temps d'exécution des algorithmes, mais souvent plus complexe à traiter algorithmiquement. Certaines opérations s'adaptent

mieux à l'une ou l'autre représentation ; par exemple le voisinage TBR nécessite une représentation par pointeurs des arbres, tandis que la prise en compte d'informations topologiques — telles que la position relative des nœuds deux à deux — est moins couteuse en utilisant une représentation matricielle.

4.3 Instances de tests

Toutes les expérimentations présentées dans ce manuscrit ont été réalisées sur des instances aléatoires et réelles, de taille et de constitution diverses. Une grande difficulté est de trouver de telles instances, car celles utilisées dans diverses publications ne sont généralement pas disponibles sur le web. En outre, il n'existe aucun site qui répertorie ou propose des instances pour ce problème.

Instances aléatoires simulées. Nous avons composé un échantillon de 180 instances aléatoires afin de réaliser des tests nécessitant un large panel d'instances [Goëffon *et al.*, 2005c]. Elles ont été générées avec le logiciel *dnatree* [Kuhner and Felsenstein, 1994] utilisant le modèle de Kimura à 2 paramètres [Kimura, 1980] avec un taux transition / transversion fixé à 2. Cette méthode de génération simule un processus d'évolution à partir d'un taux de substitution donné, ce qui permet d'obtenir des instances structurées[1]. Toutes ces instances ont été générées suivant des critères différents : de 20 à 300 séquences ADN de longueur 100 à 2 000, et utilisant un taux de substitution allant de 2% à 50%. Leur dénomination est composée du nombre de séquences, de leur longueur et du taux de substitution. Pour les tests plus intensifs présentés dans cette thèse, nous avons fixé le taux de substitution à 5, afin de simuler au mieux des instances réelles, puis sélectionné arbitrairement quelques unes de ces instances : `100-100-5`, `100-1000-5`, `300-100-5` ou `300-1000-5`. Nous avons également généré deux instances aléatoires plus larges : `500-100-5` et `500-1000-5`, utilisées dans [Goëffon *et al.*, 2005b].

Instances réelles. L'instance `zilla`, réputée très difficile, constitue le principal challenge du problème MP dans la littérature. Elle est composée de 500 séquences ADN (de 759 nucléotides, dont certains peuvent être indéterminés) d'angiospermes (plantes à fleurs). Publiée par [Chase *et al.*, 1993] avec 16 305 comme meilleur score de parcimonie trouvé en quatre semaines de calcul, l'instance `zilla` sert depuis de référence pour attester de la qualité des programmes de parcimonie. Ainsi, [Rice and Warnow, 1997], après une exécution de près d'un an (11,6 mois) sur trois stations Sun du logiciel `PAUP 3.0` (*stepwise addition* + descente TBR) et 28 milliards d'arbres passés en revue, ont pu atteindre le score minimal de 16 220. Deux ans plus tard, ce score parvient à être amélioré de deux unités — 16 218 — avec la méthode du *Ratchet* [Nixon, 1999], en 22 heures de calcul. Il s'agit à ce jour du plus petit score connu, même si les temps de calcul parviennent à être améliorés, notamment par [Goloboff, 1999] dont le logiciel `TNT` parvient à trouver un

[1] Les instances aléatoires dites *structurées* ont une constitution proche des instances réelles, au contraire d'instances purement aléatoires. Les données sont alors plus robustes et moins sensibles au bruit. Ici, plus le taux de substitution est faible, plus les instances sont structurées.

arbre de ce score en moins de 10 minutes. Néanmoins, le paramétrage y est très délicat et les temps de calcul réduits s'obtiennent au prix d'une faible robustesse : sur les cinq méthodes d'analyse proposées par Goloboff et selon ses propres expérimentations, seules deux parviennent à trouver le score minimal, avec des fréquences très faibles de $0,2\%$ et $1,3\%$. Depuis, seuls les algorithmes parallélisant le *Ratchet* parviennent à être plus rapides et fiables sur cette instance. Pour nos expérimentations, Dr Olaf R.P. Bininda-Emonds nous a généreusement mis à disposition l'instance `zilla`.

Nous disposons en outre d'une petite instance `droso` composée de 17 séquences ADN (de 222 nucléotides) de drosophiles (mouches des fruits), idéale pour des tests préliminaires.

Instances réelles binaires. [Ribeiro and Vianna, 2003 ; Ribeiro and Vianna, 2005] emploient la descente à voisinage variable ou les algorithmes génétiques pour résoudre le problème MP. Nous incluons dans nos jeux de test les huit instances réelles de taille moyenne et de difficultés très variables utilisées dans leurs expérimentations. Il s'agit d'instances binaires, sept tirées de [Luckow and Pimentel, 1985 ; Platnick, 1987 ; Platnick, 1989], et une fournie par Dr Pablo Goloboff (instance `bin-GOLO`). Nous identifions ces instances par le préfixe `bin-`.

Instances binaires aléatoires avec données manquantes. En plus des instances réelles, [Ribeiro and Vianna, 2003 ; Ribeiro and Vianna, 2005] utilisent vingt instances binaires aléatoires, qui possèdent un certain pourcentage de caractères indéfinis. Contrairement à tous les autres jeux de tests utilisés, ceux-ci ne sont pas structurés. En pratique, cela confère à ces instances une très grande difficulté. Ces vingt instances aléatoires (`tst01` à `tst20`), ainsi que les huit instances réelles précédentes, nous ont été gracieusement fournies par Pr Celso C. Ribeiro.

4.4 Comparaisons d'algorithmes de recherche locale

Nous présentons dans cette section quatre algorithmes de recherche locale, et comparons leurs efficacités respectives en les soumettant à différents paramétrages. Toutes les implémentations ont été réalisées de manière rigoureuse et homogène, afin de biaiser le moins possible les comparaisons.

4.4.1 Descente

Traditionnellement, l'algorithme de *descente pure* accepte uniquement des voisins de meilleur cout, et termine lorsqu'un optimum local est atteint, c'est-à-dire qu'il n'existe plus aucun voisin de la solution courante dont le cout lui soit strictement inférieur. Dans le cadre du problème MP, cela signifie qu'un arbre voisin t' remplace l'arbre courant t si et seulement si $\phi(t') < \phi(t)$ (t' est plus parcimonieux que t). Quatre algorithmes de descente ont été implémentés, le dernier comportant une légère variante par rapport aux définitions standard.

- $D^p_<$: À partir de chaque nouvel arbre, l'algorithme parcourt l'ensemble des voisins dans un ordre aléatoire jusqu'à ce qu'un voisin de cout strictement inférieur soit sélectionné (*premier améliorant*) ;
- $D^m_<$: Tous les voisins sont évalués, et si l'arbre courant n'est pas un optimum local, alors l'algorithme sélectionne aléatoirement un des voisins dont le cout est le plus faible (*meilleur améliorant*) ;
- $D^a_<$: À chaque itération, un voisin est généré aléatoirement, et sélectionné si son évaluation est strictement meilleure que celle de l'arbre courant. Cette technique est plus simple mais ne retourne pas systématiquement un optimum local. L'algorithme s'arrête alors lorsqu'un nombre prédéterminé d'itérations à effectuer est atteint ;
- $D^a_\leqslant$ fonctionne suivant le même procédé que $D^a_<$, mais accepte également les voisins dont le score est égal à celui de l'arbre courant.

Ces quatre algorithmes de descente sont combinés aux voisinages NNI, SPR et TBR, et exécutés 100 fois chacun sur des instances de tailles diverses. Les nombres d'itérations de recherche locale sont généralement choisis afin de maximiser l'efficacité des différents algorithmes. Nous rappelons qu'il s'agit ici d'une étude préliminaire permettant de mieux comprendre le comportement des différentes méthodes de recherche locale et des différents voisinages. Par la même occasion, des informations sur la difficulté des instances utilisées seront collectées.

Les tableaux de résultats comportent le nom de l'instance, la méthode de construction $\mathfrak{C}$ des arbres initiaux ($\mathfrak{C}_\mathfrak{a}$ pour un algorithme de construction aléatoire et $\mathfrak{C}_\mathfrak{g}$ pour un algorithme glouton), l'algorithme de recherche locale utilisé, son nombre d'itérations si celui-ci doit être fixé, et le voisinage employé. Puis, parmi les solutions retournées par l'ensemble des exécutions, nous indiquons le meilleur score ϕ^*, sa fréquence $\mathfrak{f}$, le score moyen $\phi^\sim$ et son écart-type σ, ainsi que le temps de calcul moyen d'une exécution en secondes (sauf mention contraire). Certains tableaux de résultats comportent en sus les paramètres des algorithmes utilisés. Toutes ces expérimentations ont été réalisées sur un serveur Dataswift Poseïdon disposant d'un processeur AMD Opteron 850 cadencé à 2,4 GHz.

Instance réelle à 17 séquences. L'instance réelle `droso`, bien que comportant uniquement 17 séquences, nous permet de dégager des premières tendances et éventuellement de pointer des combinaisons de méthodes inefficaces.

Dans un premier temps, nous utilisons le voisinage SPR. Avec chacune des quatre méthodes de descente, 100 descentes SPR suffisent amplement à trouver une solution qui semble être optimale, tant un score minimal se dégage des tests. Cependant, nous pouvons remarquer, et cela sera largement vérifié par la suite, que $D^a_<$ est la variante la moins efficace, et que $D^a_\leqslant$ est plus compétitive que $D^p_<$ et $D^m_<$. Nous indiquons également dans le tableau suivant la distribution d'un millier d'arbres générés suivant un algorithme de construction aléatoire ($\mathfrak{C}_\mathfrak{a}$).

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	ƒ	$\phi^\sim$	σ	tps (s)
droso	$\mathfrak{C}_\mathfrak{a}$	-	-	-	243	1/100	272,0	8,0	$\ll 0,01$
	$\mathfrak{C}_\mathfrak{a}$	$D^p_<$	-	SPR	203	60/100	203,4	0,5	0,01
	$\mathfrak{C}_\mathfrak{a}$	$D^m_<$	-	SPR	203	57/100	203,5	0,6	0,05
	$\mathfrak{C}_\mathfrak{a}$	$D^a_<$	2 000	SPR	203	36/100	204,3	2,2	0,01
	$\mathfrak{C}_\mathfrak{a}$	$D^a_<$	5 000	SPR	203	45/100	204,3	2,5	0,03
	$\mathfrak{C}_\mathfrak{a}$	$D^a_\leqslant$	2 000	SPR	203	80/100	203,3	1,2	0,01
	$\mathfrak{C}_\mathfrak{a}$	$D^a_\leqslant$	5 000	SPR	203	97/100	203,0	0,2	0,03

L'algorithme $D^a_\leqslant$, en autorisant le déplacement vers des voisins de même cout, est capable de s'extraire des optimums locaux, tandis que $D^p_<$ et $D^m_<$, qui retournent systématiquement des optimums locaux, échouent dans la recherche de la meilleure solution dans 40% des cas. De plus, cette stratégie laisse la possibilité de rallonger la recherche de $D^a_\leqslant$ en espérant trouver de meilleurs résultats, comme nous le remarquons en passant de 2 000 à 5 000 itérations. Une extension similaire de la recherche n'a logiquement généré aucune amélioration pour $D^a_<$, les résultats étant globalement les mêmes avec 2 000 et 5 000 itérations.

La deuxième étape consiste à comparer les résultats obtenus avec le voisinage SPR à ceux des descentes NNI et TBR. Nous rappelons que les voisinages NNI, SPR et TBR suivent le même principe du déplacement d'arêtes, mais sont très différents par la taille. NNI est un voisinage de taille linéaire par rapport à la taille de l'arbre, SPR est quadratique et TBR est cubique. Considérant leurs tailles respectives et le fait qu'ils soient imbriqués entre eux, il n'est pas surprenant que sur une petite instance comme celle-ci, leur efficacité est fonction de leur taille.

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	ƒ	$\phi^\sim$	σ	tps (s)
droso	$\mathfrak{C}_\mathfrak{a}$	$D^a_<$	2000	NNI	209	1/100	242,8	16,2	0,01
	$\mathfrak{C}_\mathfrak{a}$	$D^a_\leqslant$	2000	NNI	203	17/100	209,6	6,9	0,01
	$\mathfrak{C}_\mathfrak{a}$	$D^a_\leqslant$	2000	TBR	203	72/100	203,3	0,5	0,04
	$\mathfrak{C}_\mathfrak{a}$	$D^a_<$	5000	TBR	203	59/100	203,5	0,6	0,11
	$\mathfrak{C}_\mathfrak{a}$	$D^a_\leqslant$	5000	TBR	203	100/100	203,0	0	0,12

Combiné à une descente pure qui n'accepte que les voisins strictement meilleurs, NNI rencontre très rapidement des optimums locaux de très mauvaise qualité ($\phi^\sim = 242,8$). Ce problème s'estompe avec $D^a_\leqslant$, mais NNI reste peu efficace bien que l'instance soit de petite taille. TBR est le voisinage le plus efficace, et d'une robustesse exemplaire combiné à la méthode $D^a_\leqslant$.

Instance aléatoire à 100 séquences. Comparons à présent ces descentes sur une instance de taille plus importante, comportant 100 séquences de 100 caractères.

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	∫	$\phi^\sim$	σ	tps (s)
100–100–5	$\mathfrak{C}_a$	-	-	-	1 477	1/100	1 567,0	25,9	$\ll 0,01$
	$\mathfrak{C}_a$	$D^p_<$	-	SPR	534	20/100	535,1	0,9	1,3
	$\mathfrak{C}_a$	$D^m_<$	-	SPR	534	17/100	535,1	0,7	1,2
	$\mathfrak{C}_a$	$D^a_<$	50 000	SPR	534	1/100	539,7	4,2	0,4
	$\mathfrak{C}_a$	$D^a_<$	150 000	SPR	534	10/100	537,6	6,1	1,2
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	SPR	534	4/100	536,2	1,5	0,4
	$\mathfrak{C}_a$	$D^a_\leqslant$	150 000	SPR	534	57/100	534,4	0,5	1,2
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	NNI	534	6/100	566,7	32,4	0,4
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	TBR	534	1/100	537,8	1,8	3,5
	$\mathfrak{C}_a$	$D^a_\leqslant$	150 000	TBR	534	28/100	534,8	0,5	9,6
	$\mathfrak{C}_a$	$D^a_\leqslant$	500 000	TBR	534	69/100	534,3	0,5	33,4

Sur l'ensemble des 100 exécutions, $D^a_\leqslant$ est parvenu à atteindre les mêmes résultats que $D^p_<$ et $D^m_<$ en 50 000 itérations de recherche locale, mais en beaucoup moins d'occasions. Nous avons alors triplé l'effort de recherche, afin de comparer leurs résultats en des temps de calcul équivalents, et $D^a_\leqslant$ domine alors sensiblement les deux descentes pures.

Dans les mêmes conditions, NNI parvient une fois sur vingt à trouver les meileurs résultats, et reste très éloigné le reste du temps. Cela montre que ses résultats sont fortement conditionnés par les solutions initiales et autres aléas stochastiques. Enfin, TBR n'améliore les résultats de SPR qu'au prix d'un temps de calcul largement supérieur (près de 30 fois).

Instance aléatoire à 300 séquences. Cette instance montre les limites des descentes et des voisinages classiques. Combiné à descente $D^a_\leqslant$, NNI est incapable de trouver une solution acceptable, tandis que TBR se perd dans son voisinage. Enfin, nous avons dû étendre le nombre d'itérations à un million pour que SPR retourne des résultats homogènes. Sans surprise, nous constatons que la méthode $D^m_<$, combinée à un voisinage suffisamment large pour traiter ce type d'instance, est inadaptée car trop gourmande en temps de calcul (plus d'une journée de traitement pour l'ensemble des 100 exécutions).

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	∫	$\phi^\sim$	σ	tps (s)
300–100–5	$\mathfrak{C}_a$	-	-	-	5 666	1/100	5 857,0	57,4	$\ll 0,01$
	$\mathfrak{C}_a$	$D^p_<$	-	SPR	1 301	2/100	1 305,4	2,5	26,5
	$\mathfrak{C}_a$	$D^m_<$	-	SPR	1 301	2/100	1 305,5	2,1	980,1
	$\mathfrak{C}_a$	$D^a_<$	50 000	SPR	1 606	1/100	1 674,4	29,0	0,8
	$\mathfrak{C}_a$	$D^a_<$	500 000	SPR	1 305	1/100	1 318,5	10,4	6,2
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	SPR	1 558	1/100	1 649,0	30,0	0,8
	$\mathfrak{C}_a$	$D^a_\leqslant$	500 000	SPR	1 302	1/100	1 307,5	4,5	6,5
	$\mathfrak{C}_a$	$D^a_\leqslant$	1 000 000	SPR	1 300	4/100	1 302,4	1,4	12,5
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	NNI	1 582	1/100	1 924,2	146,3	0,6
	$\mathfrak{C}_a$	$D^a_\leqslant$	50 000	TBR	1 601	1/100	1 671,3	36,2	10,5
	$\mathfrak{C}_a$	$D^a_\leqslant$	500 000	TBR	1 307	1/100	1 314,0	3,6	105,1

Influence de l'arbre initial. Les descentes NNI observées jusqu'à présent semblent fortement dépendantes des arbres initiaux. Jusqu'à présent, ces derniers ont été générés de manière aléatoire ; nous expérimentons ici les mêmes algorithmes, mais en construisant l'arbre initial avec un algorithme glouton $\mathfrak{C}_{\mathfrak{g}}$ dont les résultats indépendants sont indiqués. Pour les descentes, les temps de calculs reportés tiennent compte de la construction de l'arbre initial (0,4 seconde).

Une information majeure est l'efficacité de NNI — très influencé par les solutions initiales — par rapport aux autres voisinages, qui parvient à trouver des arbres de plus faible cout en un temps de calcul réduit.

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	ƒ	$\phi^\sim$	σ	tps (s)
300-100-5	$\mathfrak{C}_{\mathfrak{g}}$	-	-	-	1 313	3/100	1 323,9	5,9	0,4
	$\mathfrak{C}_{\mathfrak{g}}$	$D^p_<$	-	SPR	1 301	1/100	1 305,1	2,0	10,6
	$\mathfrak{C}_{\mathfrak{g}}$	$D^m_<$	-	SPR	1 301	2/100	1 305,5	1,8	46,0
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_<$	50 000	SPR	1 307	3/100	1 315,5	3,5	1,0
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_<$	500 000	SPR	1 302	3/100	1 306,5	2,3	6,9
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	50 000	SPR	1 305	1/100	1 315,3	4,0	1,0
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	500 000	SPR	1 301	14/100	1 303,1	1,5	6,8
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	50 000	NNI	1 300	2/100	1 304,9	3,0	0,8
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	50 000	TBR	1 306	1/100	1 320,1	4,5	11,6
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	500 000	TBR	1 303	3/100	1 307,4	2,1	121,8

Comportement des descentes sur une instance difficile. Les instances aléatoires structurées, de tailles diverses, permettent de comprendre et d'anticiper les bénéfices des certains voisinages et critères de sélection, en fonction de la qualité des solutions initiales. Cependant, ce sont sur des benchmarks difficiles, comme l'intance `zilla`, que le choix de telle ou telle méthode devient primordial. Nous constatons que seule une descente SPR suffisamment longue depuis un arbre de bonne qualité donne de bons résultats. Rappelons que le meilleur score jamais obtenu sur cette instance est 16 218.

Inst.	$\mathfrak{C}$	Algo.	itérations	Voisinage	ϕ^*	ƒ	$\phi^\sim$	σ	tps (s)
zilla	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	100 000	SPR	18 013	1/100	18 470,2	126,9	18,5
	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	1 000 000	SPR	16 272	1/100	16 312,8	20,4	135,3
	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	10 000 000	SPR	16 220	3/100	16 228,9	8,3	1 268
	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	100 000	NNI	19 875	1/100	21 863,2	894,9	10,6
	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	1 000 000	NNI	19 142	1/100	21 342,4	1 102,9	118,4
	$\mathfrak{C}_{\mathfrak{a}}$	$D^a_\leqslant$	10 000 000	NNI	19 133	1/100	21 318,7	996,9	1 224,4
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	100 000	SPR	16 335	1/100	16 384,1	21,9	69,1
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	1 000 000	SPR	16 237	1/100	16 256,4	8,3	203,8
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	10 000 000	SPR	16 221	1/20	16 225,2	5,4	1 339
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	100 000	NNI	16 255	1/100	16 313,0	22,4	39,6
	$\mathfrak{C}_{\mathfrak{g}}$	$D^a_\leqslant$	1 000 000	NNI	16 257	1/100	16 311,3	28,1	141,3

4.4.2 Descente randomisée

La *descente randomisée*, ou *Random Walk* (littéralement traduit par « marche aléatoire »), est basé sur l'algorithme D^a. La différence est qu'ici, tout voisin estimé mais rejeté par D^a peut être accepté selon une probabilité p_{rw} donnée en paramètre. Nous comparons cet algorithme (noté RW) à la descente $D^a_{\leqslant}$, qui fournit de meilleurs résultats que $D^a_{<}$. Tout comme $D^a_{\leqslant}$, RW accepte systématiquement au cours de la recherche les voisins de cout équivalent. Les expérimentations menées n'ont dégagé aucune amélioration particulière par rapport à l'algorithme $D^a_{\leqslant}$.

Inst.	$\mathfrak{C}$	Algo.	itérations	p_{rw}	Voisinage	ϕ^*	∮	$\phi^\sim$	σ	tps (s)
300-100	$\mathfrak{C}_\mathfrak{a}$	$D^a_{\leqslant}$	1 000 000	-	SPR	1 300	4/100	1 302,4	1,4	12,5
	$\mathfrak{C}_\mathfrak{a}$	RW	1 000 000	0,1	SPR	1 300	7/100	1 302,4	2,9	12,6
	$\mathfrak{C}_\mathfrak{a}$	RW	1 000 000	0,01	SPR	1 300	8/100	1 302,4	2,5	12,3
	$\mathfrak{C}_\mathfrak{a}$	$D^r_{\leqslant}$	50 000	-	NNI	1 582	1/100	1 924,2	146,3	0,6
	$\mathfrak{C}_\mathfrak{a}$	RW	50 000	0,1	NNI	1 532	1/100	1 941,1	170,2	0,6
	$\mathfrak{C}_\mathfrak{a}$	RW	50 000	0,01	NNI	1 620	1/100	1 934,7	154,2	0,6

Inst.	Constr.	Algo.	it.	p_{rw}	Voisinage	ϕ^*	∮	$\phi^\sim$	σ	tps (s)
	$\mathfrak{C}_\mathfrak{g}$	-	-	-	-	16 434	1/20	16 485,4		
zilla	$\mathfrak{C}_\mathfrak{g}$	$D^a_{\leqslant}$	10^7	-	SPR	16 221	1/20	16 225,2	5,4	1 339
	$\mathfrak{C}_\mathfrak{g}$	RW	10^7	0,01	SPR	16 220	1/20	16 225,9	4,6	1 285
	$\mathfrak{C}_\mathfrak{g}$	RW	10^7	0,1	NNI	16 266	1/20	16 311,2	19,4	1 176
	$\mathfrak{C}_\mathfrak{g}$	RW	10^7	0,01	NNI	16 270	1/20	16 314,3	20,7	1 161

4.4.3 Recherche locale itérée

Dans la littérature, les recherches locales itérées sont considérées comme les méthodes les plus efficaces pour ce problème. Leur fonctionnement est décrit au chapitre précédent (section 3.4.3). La recherche locale itérée que nous avons implémentée est simple. Tout d'abord, une longue descente de type $D^a_{\leqslant}$ est effectuée ; ensuite, la solution retournée subit une série de perturbations, *i.e.* un certain nombre de voisins sont successivement sélectionnés, sans critère. La nouvelle solution obtenue est alors le point de départ d'une nouvelle descente. Ce processus est répété vingt fois (il s'agit du nombre de relances).

Nous constatons que cet algorithme est très efficace et n'a rien à envier à la méthode du *Ratchet*. En revanche, les temps de calcul sont relativement importants (ici sept heures pour vingt relances).

Inst.	Constr.	Algo.	itérations	perturb	relances	Voisinage	ϕ^*	tps
zilla	$\mathfrak{C}_\mathfrak{g}$	$D^a_{\leqslant}$	10 000 000	0	20	SPR	16 221	7h
	$\mathfrak{C}_\mathfrak{g}$	ILS	10 000 000	15	20	SPR	16 219	7h
	$\mathfrak{C}_\mathfrak{g}$	ILS	10 000 000	10	20	SPR	16 219	7h
	$\mathfrak{C}_\mathfrak{g}$	ILS	10 000 000	5	20	SPR	16 218	7h

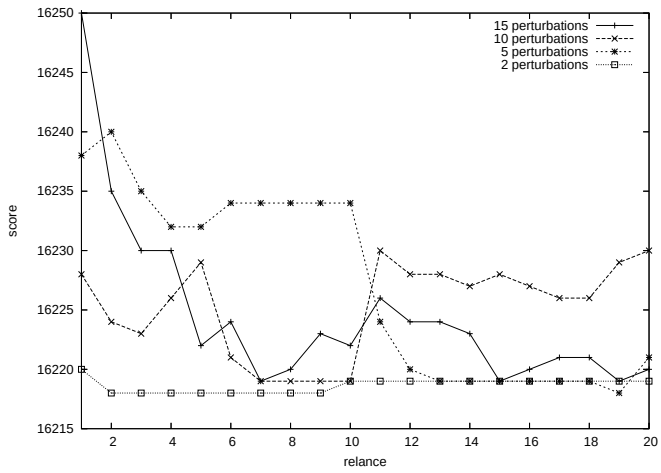


FIG. 4.1 – Évolution des recherches locales itérées

La figure 4.1 indique les couts des vingt solutions successivement retournées par les descentes. Nous pouvons dresser deux constats sur cette instance `zilla` : tout d'abord que deux mouvements SPR seulement peuvent perturber efficacement la solution ; ensuite que quelle que soit la solution courante, il suffit de quelques relances et de longues recherches pour trouver de bonnes solutions.

4.4.4 Recuit simulé

Le recuit simulé est la méthode de voisinage dont la stratégie de mouvement est la plus fine. Dans l'algorithme RW, les mouvements qui détériorent la solution courante sont acceptés suivant une probabilité constante. Ici, cette probabilité est fonction de l'importance de la dégradation $\Delta\phi$ et du nombre d'itérations de recherche locale déjà accomplis (i). Depuis une solution courante t, la probabilité d'acceptation $p_i(t'|t)$ d'un voisin t' tel que $\phi(t') > \phi(t)$ lors de la i-ème itération de recherche locale, est égale à $e^{\frac{\phi(t)-\phi(t')}{\tau(i)}}$. $\tau : \mathbb{N} \rightarrow \mathbb{R}$ est une fonction de réduction appelée température qui règle le comportement du recuit simulé en faisant décroitre progressivement la probabilité d'accepter des voisins détériorants. Nous avons expérimenté trois types de schémas de réduction : une réduction géométrique $\tau^{\mathfrak{g}}$, une réduction par paliers $\tau^{\mathfrak{p}}$ et une réduction non-monotone $\tau^{\mathfrak{s}}$.

Réduction géométrique de la température

La suite à valeur réelles $\tau^{\mathfrak{g}}$ n'utilise qu'un paramètre α (le taux de réduction), qui doit être très légèrement inférieur à 1 afin que $\tau^{\mathfrak{g}}$ ne converge pas trop rapidement. Elle est définie récursivement par $\tau^{\mathfrak{g}}_{i+1} = \alpha.\tau^{\mathfrak{g}}_i$, avec $\tau^{\mathfrak{g}}_0$ fixé. On en déduit son terme général : $\tau^{\mathfrak{g}}_i = \tau^{\mathfrak{g}}_0.\alpha^i$.

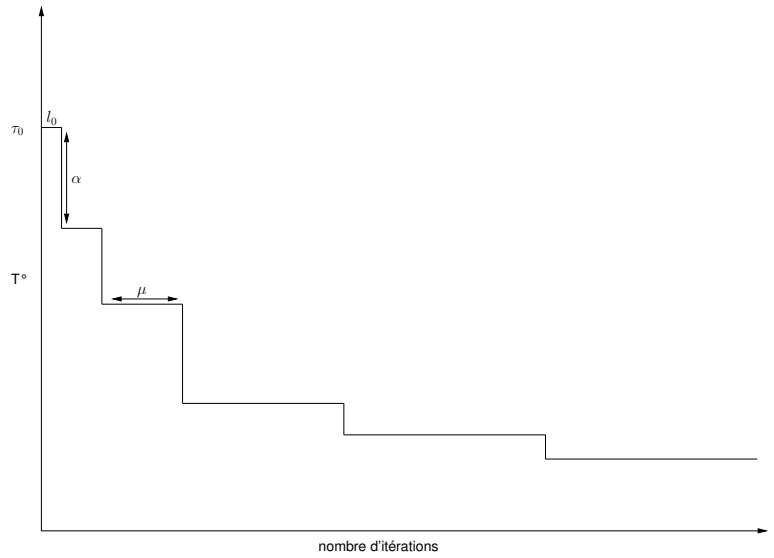


FIG. 4.2 – Schéma de réduction de la température par paliers

Réduction par paliers

La réduction de la température par paliers permet un paramétrage plus fin que la réduction géométrique, dont elle est une extension. Ici, la température n'est pas mise à jour systématiquement mais reste inchangée durant chaque palier, *i.e.* un certain nombre d'itérations. Les paliers sont plus longs en fin de recherche, lorsque la température est plus faible.

Nous avons défini un schéma de réduction par paliers comportant, outre la valeur initiale de la température τ_0, trois paramètres : le taux de réduction α, la longueur du palier initial l_0, et le coefficient $\mu > 1$ qui allonge les paliers au fil de la recherche.

Une telle évolution de la température est représentée graphiquement figure 4.2.

Plus formellement, $\tau^{\mathfrak{p}} : \mathbb{N} \to \mathbb{R}$ est définie par :

$$\tau_n^{\mathfrak{p}} = \tau_0^{\mathfrak{p}}.\alpha^{[\frac{\ln(\frac{(\mu-1)n}{l_0}+1)}{\ln \mu}]}$$

Justification. Soit $\mathcal{L} : \mathbb{N} \to \mathbb{N}$ la suite dont les valeurs successives sont égales aux n tels que $\tau_n^{\mathfrak{p}} \neq \tau_{n-1}^{\mathfrak{p}}$, c'est-à-dire aux indices des itérations qui marquent le début d'un palier. Nous avons $\mathcal{L}_0 = l_0, \mathcal{L}_1 = l_0 + \mu.l_0, \ldots, \mathcal{L}_N = \sum_{i=0}^{N} \mu^i.l_0$.
En premier lieu, déterminons N en fonction de n :

$$\sum_{i=0}^{N} \mu^i.l_0 \leqslant n < \sum_{i=0}^{N+1} \mu^i.l_0$$

$$\frac{\mu^{N+1}-1}{\mu-1} l_0 \leqslant n < \frac{\mu^{N+2}-1}{\mu-1} l_0 \quad (\mu \neq 1)$$

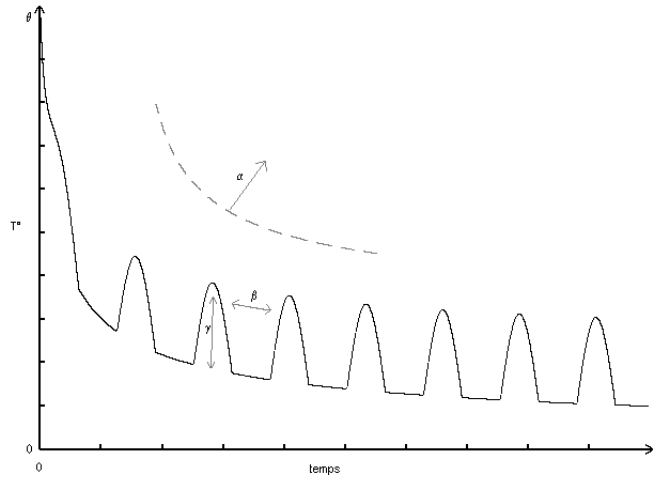


FIG. 4.3 – Schéma de réduction non-monotone de la température

$$N \leqslant \log_\mu \left(\frac{n.(\mu-1)}{l_0} + 1 \right) - 1 < N + 1$$

$$N = \left[\log_\mu \left(\frac{n.(\mu-1)}{l_0} + 1 \right) - 1 \right]$$

Nous pouvons alors calculer $\tau^{\mathfrak{p}}_{\mathcal{L}_N}$:

$$\tau^{\mathfrak{p}}_{\mathcal{L}_N} \stackrel{\text{def}}{=} \alpha.\tau^{\mathfrak{p}}_{\mathcal{L}_N - 1} = \alpha^{N+1}.\tau_0$$

$\forall n, N$ tels que $\mathcal{L}_N < n < \mathcal{L}_{N+1}, \tau^{\mathfrak{p}}_n \stackrel{\text{def}}{=} \tau^{\mathfrak{p}}_{n+1}$. D'où $\tau^{\mathfrak{p}}_n = \tau^{\mathfrak{p}}_{\mathcal{L}_N} = \alpha^{N+1}.\tau^{\mathfrak{p}}_0$. En conséquence :

$$\tau^{\mathfrak{p}}_n = \tau^{\mathfrak{p}}_0.\alpha^{[\log_\mu (\frac{n.(\mu-1)}{l_0}+1)]} = \tau^{\mathfrak{p}}_0.\alpha^{[\frac{\ln (\frac{n.(\mu-1)}{l_0}+1)}{\ln \mu}]}$$

Réduction non-monotone

Enfin, nous avons expérimenté un schéma inédit de refroidissement non-monotone de la température. Il s'agit de concevoir un modèle de régulation de la température tel qu'un paramétrage approprié combine les atouts d'un recuit simulé classique et d'une recherche locale itérée. Ce schéma de réduction comporte quatre paramètres (α, β, γ et θ) dont les effets sont indiqués figure 4.3.

La définition — récursive — de $\tau^{\mathfrak{s}}$ ($\mathfrak{s}$ pour sinusoïde) est donnée par :

$$\tau^{\mathfrak{s}}_n = \theta.\tau^{\mathfrak{s}}_{n-1}.\left(1 - \frac{(\tau^{\mathfrak{s}}_{n-1})^2}{\alpha}\right) + \max\left(0\ ,\ \frac{\sin\frac{n}{\beta}}{\gamma}\right)$$

Performances des recuits simulés sur l'instance `zilla`

Lorsque le temps de calcul alloué est suffisant, les algorithmes de descente parviennent à trouver des solutions acceptables, souvent proches des meilleurs scores connus. Pour améliorer les solutions trouvées par les descentes, les recuits simulés ont généralement besoin d'un effort calculatoire plus important. Il s'avère que pour le problème MP et pour les instances traditionnelles, le recuit simulé exhibe des résultats décevants, pas meilleurs que ceux de la descente classique. Le schéma de réduction non-monotone offre cependant un gain en terme de robustesse.

Inst.	Algo.	itérations	réduction	Voisinage	ϕ^*	ƒ	$\phi^\sim$	σ	tps (s)
	$\mathfrak{C}_{\mathfrak{a}}$	-	-	-	37 005	1/20	16 422,9		
`zilla`	$D^a_{\leqslant}$	2.10^7	-	SPR	16 219	1/20	16 225,8	4,8	2 325
	RS	2.10^7	$\tau^{\mathfrak{g}}$	SPR	16 220	2/20	16 227,2	6,5	2 362
	RS	2.10^7	$\tau^{\mathfrak{p}}$	SPR	16 219	1/20	16 226,1	6,5	2 336
	RS	2.10^7	$\tau^{\mathfrak{s}}$	SPR	16 219	1/20	16 224,9	3,8	2 372

En revanche, notre algorithme du recuit simulé obtient de bien meilleurs résultats que `LVB` [Barker, 2004], unique logiciel de reconstruction phylogénétique par parcimonie utilisant un recuit simulé. Des comparaisons expérimentales sur des instances aléatoires structurées sont disponibles dans [Goëffon *et al.*, 2005a], et voient notre algorithme obtenir de meilleurs résultats en terme d'efficacité et de temps de calcul que `LVB` et `DNAPARS`, le logiciel libre de résolution du problème MP le plus utilisé. De plus, ni l'un ni l'autre de ces deux logiciels ne parvient à traiter l'instance `zilla`.

4.5 Conclusion

L'étude de nombreuses heuristiques de voisinage au problème MP a permis d'identifier les facteurs régissant l'efficacité et la rapidité des algorithmes. Parmi les enseignements principaux, nous retenons :

- la puissance de recherche du voisinage SPR,
- la nécessité de la stratégie de mouvement à accepter des voisins d'évaluations identiques,
- la nécessité de partir d'un arbre aléatoire ou bien construit en fonction du voisinage et de la méthode utilisés,
- le peu d'intérêt d'utiliser des stratégies plus élaborées propres à la descente randomisée ou au recuit simulé.

Cette étude empirique des métaheuristiques de voisinage nous permet de définir des pistes de recherches plus précises afin d'améliorer leurs performances. De nouvelles problématiques se dégagent de ces résultats, comme les possibilités de définir un voisinage alternatif plus puissant que SPR, traité dans le chapitre suivant.

Chapitre 5

Un voisinage progressif

Les méthodes de recherche locale sont capables de traiter de manière satisfaisante des instances du problème MP comportant plusieurs centaines d'espèces, à condition d'utiliser un voisinage approprié et de tolérer des temps de calcul importants. Nous avons observé dans le chapitre 4 que le voisinage SPR était suffisamment large pour permettre aux algorithmes de descente de ne pas converger rapidement vers un optimum local de mauvaise qualité. En revanche, la taille du voisinage est un handicap au niveau des temps de calcul car chaque solution possède de nombreux voisins, dont la plupart ne sont pas pertinents.

Nous présentons dans ce chapitre un nouveau mécanisme qui permet de cibler les voisins les plus pertinents d'un voisinage large comme SPR, afin de réduire les temps de calcul tout en conservant une efficacité similaire. Une grande partie de ce chapitre a été publiée dans [Goëffon *et al.*, 2005b].

Sommaire

5.1 Problématique et principe général

Dans le chapitre précédent, nous avons observé que la plupart des algorithmes de recherche locale utilisant le voisinage SPR convergent fréquemment vers des solutions de bonne qualité. La taille importante du voisinage réduit les possibilités de bloquer sur une solution dont aucun voisin ne satisfasse la condition d'acceptation, surtout lorsque cette dernière autorise les mouvements vers des voisins de scores identiques. Cependant, et en toute logique, ces algorithmes ont une très faible capacité d'amélioration en fin de recherche et évaluent de nombreux voisins non pertinents, dont les scores sont largement supérieurs à celui de la solution courante. Les mouvements sont alors peu fréquents et le nombre d'itérations de recherche locale sans amélioration très important.

En utilisant un voisinage restreint comme NNI, les descentes sont très rapides, car elles convergent rapidement vers des optimums locaux, où il n'existe plus de possibilité d'amélioration. Lorsque les instances sont difficiles, les solutions retournées par des recherches locales utilisant ce voisinage sont très mauvaises, à moins que la solution initiale soit déjà de bonne qualité ; dans ce cas, nous observons qu'il est possible d'améliorer cette solution en quelques pas de recherche NNI.

Afin de combiner les propriétés intéressantes des voisinages larges et faibles, nous proposons d'effectuer une recherche locale utilisant un voisinage large ou restreint selon l'avancée de la recherche, et la fréquence d'apparition de voisins pertinents. Cette première idée est typiquement une variante de la recherche à voisinage variable (chapitre 3, section 3.4.2), mais où la succession de voisinages ne s'appliquerait pas dans le même ordre et aux mêmes conditions. En effet, contrairement à la méthode VNS, partir du voisinage le plus large peut s'avérer ici pertinent, en construisant les bases de la topologie de la future solution. Évaluer plus de voisins — avec des modifications plus sensibles — en début de recherche va permettre d'améliorer sensiblement le cout des solutions dès les premiers pas de la recherche locale, grâce à une recherche plus intensive. En fin de recherche, nous imaginons n'intervenir que très localement sur la topologie de l'arbre. Nous pouvons obtenir ce schéma en réduisant petit à petit l'étendue du voisinage exploré au fil de la recherche tout en conservant le plus possible de voisins pertinents.

L'idée est donc de définir une relation de voisinage paramétrique qui évolue dans le temps, soit de manière prédéfinie, soit de manière réactive en prenant en compte les informations sur la qualité des voisins visités précédemment. Nous parlerons de voisinage *progressif* s'il suit un schéma d'évolution préfixé, ou *réactif* dans le cas contraire.

Dans la suite de ce chapitre, nous définissons un voisinage paramétrique basé sur le voisinage SPR, tout en généralisant le concept qui peut s'appliquer à tout voisinage. Nous montrons ensuite de quelle manière l'évolution du paramètre a une influence sur la qualité du voisinage, puis mesurerons le gain apporté par cette technique.

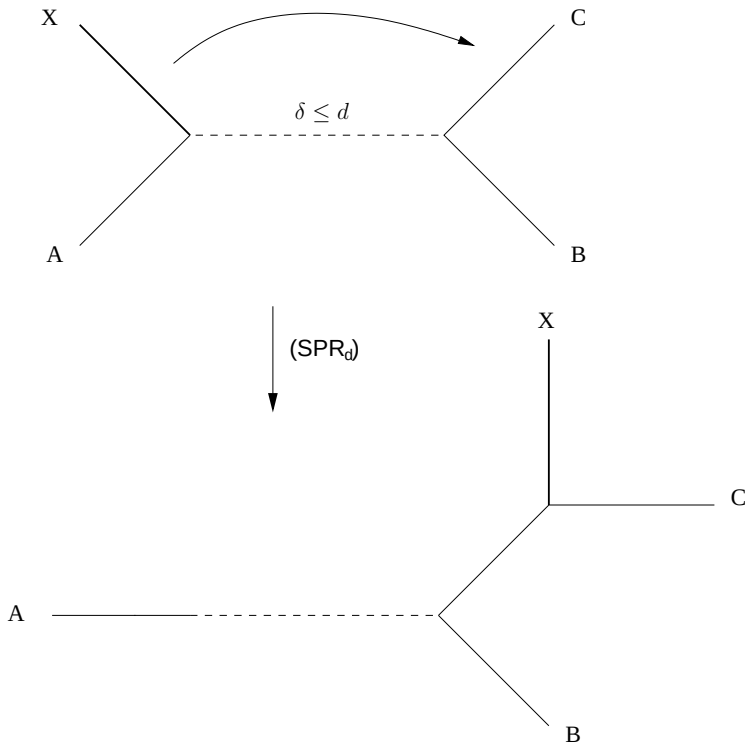


FIG. 5.1 – SPR et le voisinage paramétrique : SPR correspond au cas où $d = \infty$

5.2 Un exemple de voisinage paramétrique

5.2.1 Schéma de construction

À titre d'exemple, nous prenons deux voisinages de tailles différentes $\mathcal{N}^1$ et $\mathcal{N}^2$, de sorte que $\mathcal{N}^2 \subseteq \mathcal{N}^1$. L'idée est de définir un voisinage paramétrique $\mathcal{N}_d$ tel que $\mathcal{N}_{dmax} = \mathcal{N}^1$, $\mathcal{N}_{dmin} = \mathcal{N}^2$, et $\forall d \in \{dmin, \ldots, dmax\}, \mathcal{N}^2 \subseteq \mathcal{N}_d \subseteq \mathcal{N}^1$.

Prenons $\mathcal{N}^1 = \mathcal{N}^{SPR}$ et $\mathcal{N}^2 = \mathcal{N}^{NNI}$. Avec SPR, un sous-arbre est détaché et reconnecté ailleurs, sans contrainte particulière si ce n'est d'obtenir un arbre valide et distinct. On peut voir NNI comme un SPR particulier, où une branche doit être insérée sur une arête voisine d'où elle provient dans l'arbre courant.

Par extension, nous imaginons alors un voisinage de type SPR où la distance entre l'arête supprimée et l'arête insérée soit bornée. Si cette borne est maximale (Figure 5.1), alors il s'agit du voisinage SPR, sans contrainte. Si celle-ci elle minimale, alors nous nous retrouvons dans le cas du voisinage NNI (Figure 5.2). Une telle définition de voisinage paramétrique peut être vue comme une simple généralisation de voisinages inclus.

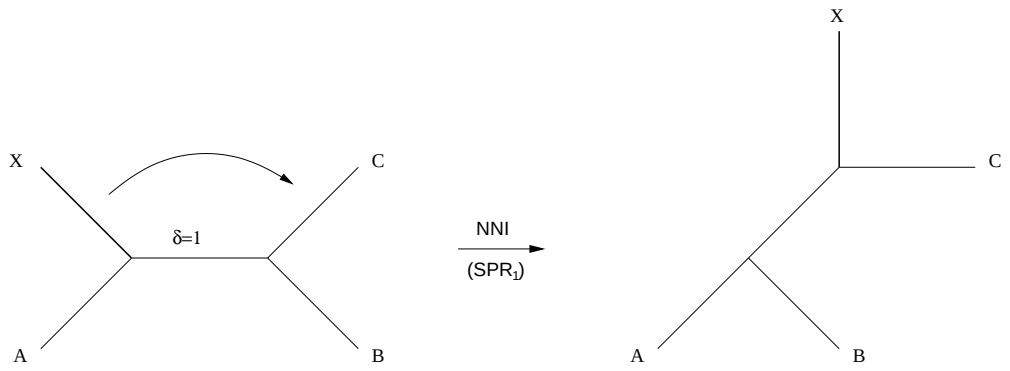


FIG. 5.2 – NNI et le voisinage paramétrique : NNI correspond au cas où $d = 1$

5.2.2 Distance entre nœuds

Soit $f^{SPR} : (\mathcal{T}, \mathcal{V}, \mathcal{V}) \rightarrow \mathcal{T}$ la transformation telle que $f^{SPR}(t, v_i, v_j)$ soit l'arbre obtenu en détachant dans $t = (\mathcal{V}, \mathcal{E})$ le sous-arbre de racine v_i et en l'insérant entre v_j et son ascendant direct. Alors $\mathcal{N}^{SPR}(t) = \{t' \in \mathcal{T}, \exists (v_i, v_j) \in \mathcal{V}^2, f^{SPR}(t, v_i, v_j) = t'\}$.

Pour contraindre SPR, nous introduisons un paramètre d, tel que $\mathcal{N}_d^{SPR}(T)$ représente l'ensemble des arbres obtenus par transformation $f^{SPR}(t, v_i, v_j)$ et dont v_i et v_j sont *distants* de d au maximum.

Notons $\delta(v_i, v_j)$ la *distance* entre v_i et v_j, comme étant égale à la longueur du chemin élémentaire séparant les ascendants respectifs de v_i et v_j, à laquelle nous retranchons 1 si le chemin contient la racine (si l'on travaille sur des arbres enracinés). Ainsi, deux nœuds frères sont distants de 0, et la distance reste la même dans le cas d'arbres non enracinés.

Puisque nous souhaitons maitriser la taille du voisinage durant la recherche, nous définissons le voisinage $\mathcal{N}_d^{SPR}(t)$ comme étant l'ensemble des voisins $\mathcal{N}^{SPR}(t)$ tels que la distance entre l'arête supprimée et l'arête insérée (qui est égale à la distance δ entre leurs deux nœuds fils) n'excède pas le paramètre d. Plus formellement, $\mathcal{N}_d^{SPR}(t) = \{t' \in \mathcal{T}, \exists (v_i, v_j) \in V^2, f^{SPR}(t, v_i, v_j) = t' \wedge \delta(v_i, v_j) \leqslant d\}$.

5.3 Schéma de voisinage progressif

5.3.1 Étude des espaces de recherche

Afin de justifier l'utilisation et la pertinence du voisinage progressif, nous avons étudié le comportement des algorithmes de descente utilisant le voisinage SPR. Dans cette section, nous utilisons la distance $\delta(v_i, v_j)$ pour définir une *distance d'arbre* au sens de δ, entre un arbre t et l'arbre voisin évoqué $f^{SPR}(t, v_i, v_j)$.

Localisation des meilleurs voisins

La première étude que nous avons réalisée consiste à observer l'évolution de la distance δ entre un arbre donné t et ses *meilleurs* voisins t'. Pour cela, nous avons adapté un

algorithme de descente de type *meilleur améliorant* (Algorithme 5.1). À chaque itération de l'algorithme, un voisin de score minimal est choisi et remplace l'arbre courant, jusqu'à ce que celui-ci soit un optimum local, c'est-à-dire qu'il ne possède aucun voisin améliorant.

Algorithme 5.1 : Algorithme de type D^m mémorisant les distances entre les solutions sélectionnées

début
  $r \leftarrow 1$;
  tant que $r \leqslant R$ **faire**
    construire un arbre initial t_0 ;
    $i \leftarrow 0$;
    tant que t_i *n'est pas un optimum local* **faire**
      $i \leftarrow i + 1$;
      sélectionner $t_i \in \mathcal{N}^{SPR}(t_{i-1})$ tel que $\forall t' \in \mathcal{N}^{SPR}(t_{i-1}), \phi(t_i) \leqslant \phi(t')$;
    fin
    $p \leftarrow i$ (le nombre final d'itérations);
    pour tous les $i \in \{1..p\}$ **faire**
      marquer les coordonnées $(\frac{i}{p}, \delta(t_{i-1}, t_i))$;
    fin
    $r \leftarrow r + 1$;
  fin
fin

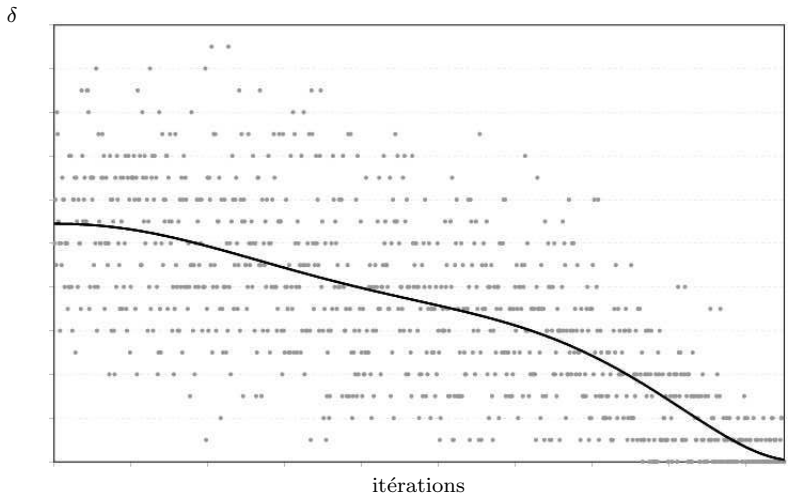


FIG. 5.3 – Évolution de la distance δ entre des arbres et leurs meilleurs voisins SPR, en fonction du nombre d'itérations.

Nous avons exécuté cet algorithme plusieurs fois ($R = 10$) sur plusieurs instances, avec des résultats similaires. La figure 5.3 montre les résultats obtenus sur l'instance aléatoire

100-1000-5. La ligne noire est une approximation polynomiale de degré 6 de tous les points. Sur cette figure, nous observons qu'en début de recherche, les meilleurs voisins sont à des distances δ variables, et généralement autour de la moyenne des valeurs possibles de δ. Cette distance décroit lorsque la recherche progresse. En fin de recherche, les meilleurs voisins des arbres courants sont toujours très proches, le plus souvent séparés par une faible distance δ, proche de 1 (le minimum).

Cette observation est en accord avec les résultats des algorithmes du chapitre précédent, et montre la nécessité d'utiliser un voisinage progressif pour réduire les calculs *a priori* dispensables. Nous remarquons sur la figure 5.3 que pour améliorer sensiblement la solution courante en début de recherche, il est nécessaire de tester d'importantes transformations, en utilisant un voisinage suffisamment large comme SPR. Puisque la distance entre un arbre et son meilleur voisin décroit tout au long de la recherche, des transformations mineures sont plus appropriées en fin de recherche ; cela correspond à l'emploi d'un voisinage restreint comme NNI.

Sous-voisinage améliorant

La seconde étude évalue la taille des sous-voisinages améliorants et détériorants d'une solution quelconque en dénombrant le nombre de voisins de meilleur cout. Dans l'optique de lier cette information avec la valeur du paramètre d, nous considérons, pour une distance δ donnée, l'évolution du ratio $\frac{|\{\text{voisins\_ameliorants}\}|}{|\{\text{voisins}\}|}$ durant la recherche. Pour réaliser cette mesure, nous sélectionnons un grand nombre de configurations de scores divers — pour la plupart proches du score optimal —, et dénombrons pour chaque configuration t les voisins de meilleur score, en fonction de la distance $\delta(v_i, v_j)$ d'une transformation $f^{SPR}(t, v_i, v_j)$.

À partir d'un problème (τ, ϕ), nous définissons deux fonctions $\chi_+ : \mathbb{N} \times \mathbb{N} \to \mathbb{Q} \cap [0,1]$ et $\chi_- : \mathbb{N} \times \mathbb{N} \to \mathbb{Q} \cap [0,1]$ telles que $\chi_+(\delta, n)$ représente le pourcentage de voisins $\mathcal{N}_\delta^{SPR}(t) \setminus \mathcal{N}_{\delta-1}^{SPR}(t)$ améliorants pour l'ensemble des configurations $\{t \in \tau, \phi(t) = n\}$, et $\chi_-(\delta, n)$ représente le complément $1 - \chi_+(\delta, n)$.

Comme il n'est pas envisageable d'évaluer les voisins de l'ensemble des configurations de τ, nous estimons les valeurs de $\chi_+(\delta, n)$ et $\chi_-(\delta, n)$ à partir d'un échantillon $\tau' \subseteq \tau$ de configurations. Considérant la distribution en cloche des configurations de τ, nous générons l'échantillon τ' non pas de manière purement aléatoire, mais selon une distribution similaire à l'ensemble des configurations visitées pendant une recherche locale de type SPR. Ainsi, les configurations intéressantes seront mieux représentées et la variété de leurs scores préservée.

Les approximations des χ_+ sont alors données par :

$$\chi_+^*(\delta, n) = \frac{|\{t' \in \mathcal{N}_\delta^{SPR}(t) \setminus \mathcal{N}_{\delta-1}^{SPR}(t), (\phi(t') < n) \wedge (\forall t \in \tau', \phi(t) = n)\}|}{|\{\mathcal{N}_\delta^{SPR}(t) \setminus \mathcal{N}_{\delta-1}^{SPR}(T), \forall t \in \tau', \phi(t) = n\}|}$$

Par exemple, $\chi_+^*(10, 10\,000) \simeq 0,45$ signifie que pour une configuration dont le score est égal à 10 000, une transformation SPR dont la longueur du chemin entre l'arête supprimée et l'arête créée est de 10 améliore le score de parcimonie avec une probabilité estimée de 45% (sur une instance donnée).

Le but de cette étude est d'anticiper le schéma d'évolution optimal du paramètre d de notre voisinage $\mathcal{N}_d^{SPR}$. Une nouvelle fois, les tests effectués sur plusieurs types d'instances et plusieurs échantillons τ' ont mis en évidence les mêmes informations. La figure 5.4 montre les valeurs de χ_+^* (exprimées en poucentage), en fonction du score de la configuration courante et du paramètre δ (distance). La taille des points est, pour chaque score, fonction du rang de la probabilité de trouver un voisin qui améliore en fonction de δ.

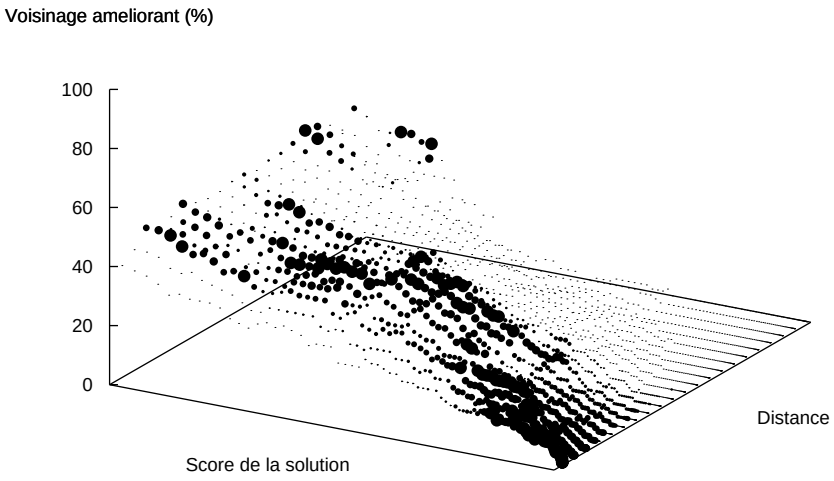


FIG. 5.4 – Proportions de voisins améliorants en fonction du score de la solution courante et de la distance δ

Nous remarquons que lors du début de la recherche locale, il est plus probable de trouver des voisins améliorants qu'en fin de recherche, mais qu'il n'est pas nécessaire de restreindre le voisinage pour augmenter la probabilité d'en trouver un. En revanche, lorsque la capacité d'améliorer la solution initiale est réduite de moitié — nous nous plaçons dans le cas où celle-ci est générée aléatoirement —, la valeur optimale de δ, *i.e.* $\arg\max_\delta(\chi_+^*(\delta, n))$, décroit linéairement. En fin de recherche, utiliser une faible valeur de d pour contraindre notre voisinage $\mathcal{N}_d^{SPR}$ nous semble alors approprié.

Pour une meilleure lisibilité des informations de la figure 5.4, nous proposons d'isoler l'évolution du rapport $\frac{|\{\text{voisins\_ameliorants}\}|}{|\{\text{voisins}\}|}$ pour quelques valeurs de δ (figures 5.5 et 5.6).

Il apparait clairement que plus la recherche progresse, plus la part de voisins améliorants décroit, mais de manière plus radicale avec des valeurs de δ plus importantes. En fin de recherche, il n'est plus possible d'améliorer les solutions avec des valeurs de δ élevées, tandis qu'une réduction de δ entraine une décroissance moins forte de la probabilité de rencontrer des voisins améliorants. Du point de vue des arbres, cela signifie qu'il est inutile d'effectuer des modifications topologiques importantes en fin de recherche. Cette

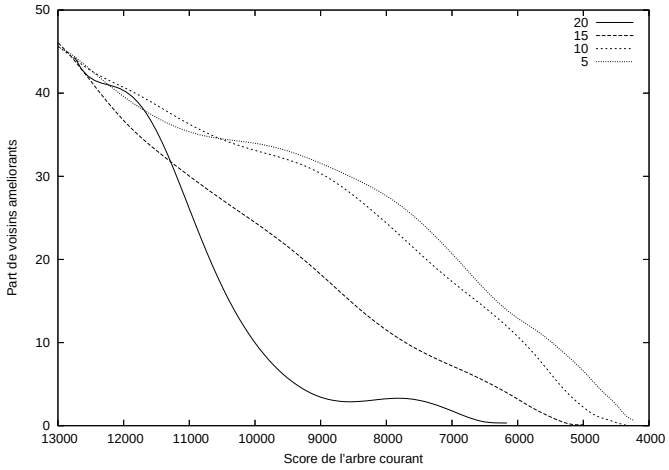


FIG. 5.5 – Évolution de la proportion de voisins améliorants en fonction du score des solutions, pour une distance δ donnée (20, 15, 10, 5)

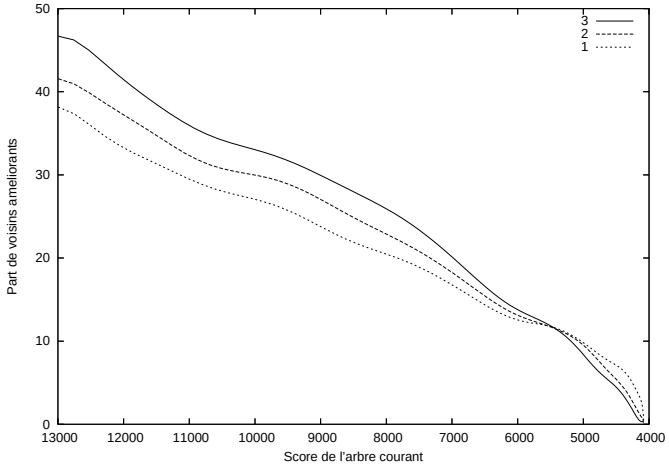


FIG. 5.6 – Évolution de la proportion de voisins améliorants en fonction du score des solutions, pour une distance δ donnée (3, 2, 1)

observation constitue une première justification empirique de l'utilisation d'un voisinage progressif.

5.3.2 Schéma de réduction du paramètre d

La réduction du voisinage $\mathcal{N}_d^{SPR}$ permet de débuter avec un voisinage quadratique (SPR) et de terminer avec le voisinage NNI, linéaire par rapport au nombre de nœuds des

arbres. Les valeurs initiales et finales de d se calculent alors de cette manière :

$$\begin{cases} \mathcal{N}^{SPR}_{d_{init}} \equiv \mathcal{N}^{SPR} \\ \\ \mathcal{N}^{SPR}_{d_{final}} \equiv \mathcal{N}^{NNI} \end{cases} \Rightarrow \begin{pmatrix} d_{init} \\ d_{final} \end{pmatrix} = \begin{pmatrix} \max_{V^2} \delta(v_i, v_j) \\ 1 \end{pmatrix}$$

d_{init} correspond au plus petit majorant des distances entre nœuds, qui est égale à la plus grande distance entre deux feuilles de l'arbre.

Dans un premier temps, nous réduisons d de manière linéaire. Si M est le nombre d'itérations de recherche locale prévus, alors le paramètre d du voisinage $\mathcal{N}^{SPR}_d$ à la i-ème itération de recherche locale sera égal à $\lceil d_{init} \left(1 - \frac{i-1}{M}\right) \rceil$.

Cependant, si les figures 5.3 à 5.6 suggèrent une réduction linéaire en fonction du score de l'arbre, il n'en va pas de même de la réduction de la distance en fonction du nombre d'itérations. En effet, plus le rapport de voisins améliorants baisse, plus le nombre d'itérations nécessaires pour améliorer les solutions courantes augmente. Pour tenir compte de cela, et plutôt que de définir un schéma de réduction non linéaire, nous avons modifié la méthode de génération de voisins. À chaque itération de recherche locale, le choix d'un voisin $f^{SPR}(t, v_i, v_j)$ (avec $\delta(v_i, v_j) \leqslant d$) se fait aléatoirement mais selon une distribution non uniforme. Une distance d' (telle que $1 \leqslant d' \leqslant d$) et un nœud v_i sont tout d'abord sélectionnés aléatoirement. Ensuite, l'algorithme recherche un nœud v_j en parcourant un chemin élémentaire aléatoire de longueur $d' + 2$. Si durant le parcours il rencontre une feuille de l'arbre, v_j prend la valeur de cette feuille même si $\delta(v_i, v_j) \leqslant d'$. Ainsi les feuilles de l'arbre ont une probabilité plus importante d'être sélectionnées, qui est fonction de leur distance avec la racine du sous-arbre à détacher. Cette technique possède deux avantages. Tout d'abord, elle permet de s'approcher d'une réduction quadratique du voisinage, mais sans le restreindre trop rapidement. Nous rappellons que l'intérêt du voisinage progressif est d'augmenter à chaque itération la probabilité de rencontrer des voisins améliorants, tout en réduisant le moins possible la puissance de recherche d'un voisinage large. Le second avantage constaté de cette méthode de génération de voisins est une réduction des temps de calcul.

Dans la section suivante, nous évaluons l'influence du voisinage progressif $\mathcal{N}^{SPR}_d$ par rapport à leurs voisinages fixes associés $\mathcal{N}^{SPR}$ et $\mathcal{N}^{NNI}$.

5.4 Expérimentations

5.4.1 Conditions d'expérimentation

Dans cette section, nous testons un algorithme de descente utilisant les trois voisinages suivants : $\mathcal{N}^{SPR}$ (descente SPR), $\mathcal{N}^{NNI}$ (descente NNI) et $\mathcal{N}^{SPR}_d$ (descente avec le voisinage progressif décrit dans la section 5.3.2, que nous appelons DPN pour *Descent with Progressive Neighborhood*). L'idée intuitive du voisinage progressif $\mathcal{N}^{SPR}_d$ est de combiner l'efficacité de $\mathcal{N}^{SPR}$ et la rapidité de $\mathcal{N}^{NNI}$. Nous vérifions ici ces points sur une sélection d'instances de tests. Puisque $\mathcal{N}^{NNI} \subseteq \mathcal{N}^{SPR}_d \subseteq \mathcal{N}^{SPR}$, une descente classique de type $D^p_<$ ou $D^m_<$ (qui retourne un optimum local) utilisant $\mathcal{N}^{SPR}$ retournera majoritairement des

solutions de cout meilleur ou égal, car un optimum local au sens d'un voisinage l'est également pour tous ses voisinages inclus, alors que la réciproque est fausse. Mais déterminer un optimum local nécessite en particulier d'avoir calculé tous ses voisins.

Le tableau 5.1 indique le nombre de voisins $|\mathcal{N}(t)|$ d'un arbre t en fonction du nombre de séquences N, pour les deux voisinages à taille fixe utilisés (les formules sont évoquées dans la section 3.4.1 du chapitre 3).

N	$\lvert\mathcal{N}^{NNI}(T)\rvert$	$\lvert\mathcal{N}^{SPR}(T)\rvert$	$\lvert\mathcal{T}\rvert$
50	94	8 742	$2,8.10^{76}$
100	194	37 442	$3,3.10^{184}$
300	594	352 242	$3,4.10^{700}$
500	994	987 042	$1,0.10^{1280}$

Tab. 5.1 – Taille des voisinages et de l'espace de recherche en fonction du nombre de séquences

Nous pouvons clairement remarquer que l'effort calculatoire à fournir pour trouver un optimum local peut devenir fortement problématique sur des voisinages larges.

Pour mesurer efficacement l'influence du voisinage sur la qualité de la solution retournée pour un effort calculatoire équivalent (et donc des temps de calcul plus raisonnables), nous fixons un nombre maximal M d'itérations de recherche locale. Dans nos expérimentations et pour les instances aléatoires structurées composées de 100 séquences, nous fixons M à 50 000, c'est-à-dire que 50 000 arbres au maximum seront évalués. Pour les instances plus larges (300 ou 500 séquences), nous avons fait varier le paramètre M.

L'algorithme 5.2 montre la procédure utilisée pour nos tests. Il s'agit de l'algorithme $D^a_{\leqslant}$, évoqué dans la section 4.4.1 du chapitre précédent. Nous avons vu que cet algorithme, qui accepte sytématiquement les voisins de scores identiques, est plus performant qu'une descente classique, qui s'arrête dès qu'un optimum local est atteint. En particulier, cela permet au voisinage restreint NNI de bloquer moins rapidement sur un optimum local.

Pour cette étude, nous avons sélectionné six instances aléatoires structurées : `100-100-5`, `100-1000-5`, `300-100-5`, `300-1000-5`, `500-100-5`, `300-1000-5`, ainsi que l'instance `zilla`. Pour chacune d'entre elles, nous avons réalisé 100 exécutions pour chaque combinaison $(\mathfrak{C}, \mathcal{N})$ (construction et voisinage). Rappelons que la méthode de construction $\mathfrak{C}_\mathfrak{a}$ génère aléatoirement un arbre initial, tandis que $\mathfrak{C}_\mathfrak{g}$ est un algorithme glouton (*stepwise addition*). Nous pouvons ainsi observer le comportement des descentes SPR, NNI et DPN en fonction de la qualité de la solution initiale.

5.4.2 Comparaison entre les recherches locales SPR, NNI et DPN

Les tableaux de résultats contiennent, pour chaque instance et chaque méthode, le score minimum ϕ^*, sa fréquence $\mathfrak{f}$, le score moyen $\phi^\sim$ et son écart-type σ, ainsi que le temps d'exécution moyen d'une exécution en secondes. Nous précisons également le score moyen des arbres initiaux $\phi_0^\sim$ suivant la méthode de construction $\mathfrak{C}_\mathfrak{a}$ ou $\mathfrak{C}_\mathfrak{g}$.

Algorithme 5.2 : Descente $D^a_{\leqslant}$

Données : une matrice de caractères $\mathcal{A}$ induisant un problème de minimisation $(\mathcal{T}, \phi)$, une méthode de construction $\mathfrak{C}$, un voisinage $\mathcal{N}$.
Résultat : le meilleur arbre trouvé.
début
  construire un arbre initial t selon $\mathfrak{C}$;
  $i \leftarrow 1$;
  tant que $i \leqslant M$ **faire**
    sélectionner $t' \in \mathcal{N}(t)$;
    si $\phi(t') \leqslant \phi(t)$ **alors**
      $t \leftarrow t'$;
    fin
    $i \leftarrow i + 1$;
  fin
fin

Sur les deux instances composées de 100 séquences (tableau 5.2), nous constatons que DPN retourne majoritairement ou systématiquement des solutions minimales (par rapport à l'ensemble des méthodes), quelle que soit la qualité de la solution initiale. NNI est très peu fiable lorsque la solution initiale est générée aléatoirement. Dans le cas contraire, SPR et NNI obtiennent des résultats satisfaisants, mais ceci est dû à la relative facilité de ces instances et par conséquent les bonnes qualités des solutions initiales (durant les tests, l'algorithme $\mathfrak{C}_{\mathfrak{g}}$ est parvenu deux fois sur cent à construire une solution de score minimal sur l'instance `100-1000-5`).

Lorsque le nombre de séquences est plus important, comme pour l'instance reportée dans le tableau 5.3, SPR n'est pas du tout approprié pour une recherche locale courte ($M = 50\ 000$) ; ses performances sont très éloignées de celles de la descente à voisinage progressif DPN. La qualité des solutions retournées par NNI est fortement dépendante des arbres initiaux ; l'algorithme peut converger dans ce cas très rapidement vers un optimum local, ce qui montre l'instabilité de la méthode en fonction des facteurs stochastiques. Ces résultats confirment que NNI n'est efficace (en terme de performance, robustesse et temps de calcul) que lorsque les séquences sont longues et à condition de débuter la recherche avec une solution de bonne qualité. En revanche, DPN obtient de bons résultats depuis toute solution initiale, malgré le nombre réduit d'itérations comparé à la taille du problème.

La figure 5.7 montre le score de l'arbre courant en fonction de l'avancée de la recherche, pour l'instance `300-100-5` et en partant d'un arbre aléatoire (la recherche reportée est celle retournant l'arbre de score médian). La recherche locale à voisinage progressif (DPN) y domine clairement SPR et NNI.

Pour des instances très larges (tableau 5.4), il se confirme que SPR donne dans tous les cas de mauvais résultats. Les scores retournés par NNI sont très inconstants, avec des écarts-types importants. La puissance du voisinage progressif par rapport aux voisinages standard est illustré sur la figure 5.8, qui représente la répartition de l'ensemble des

		$\phi_0^{\sim}$	ϕ^*	∱	$\phi^{\sim}$	σ	tps (s)
100–100–5	$\mathfrak{C}_a$+SPR	*1 570*	534	6/100	536,3	1,5	0,4
	$\mathfrak{C}_a$+NNI		534	5/100	566,7	32,4	0,3
	$\mathfrak{C}_a$+DPN		**534**	**87/100**	**534,4**	**2,3**	**0,3**
	$\mathfrak{C}_g$+SPR	*544*	534	26/100	534,9	0,6	0,5
	$\mathfrak{C}_g$+NNI		534	42/100	536,2	2,7	0,4
	$\mathfrak{C}_g$+DPN		**534**	**82/100**	**534,2**	**0,4**	0,4
100–1000–5	$\mathfrak{C}_a$+SPR	*12 914*	4 080	10/100	4 093,7	14,3	3,7
	$\mathfrak{C}_a$+NNI		4 080	46/100	4 121,9	91,4	3,2
	$\mathfrak{C}_a$+DPN		**4 080**	**100/100**	**4 080.0**	**0**	**2,2**
	$\mathfrak{C}_g$+SPR	*4 085*	4 080	85/100	4 080,3	0,5	4,0
	$\mathfrak{C}_g$+NNI		4 080	75/100	4 080,3	0,6	3,4
	$\mathfrak{C}_g$+DPN		**4 080**	**100/100**	**4 080.0**	**0**	**2,8**

Tab. 5.2 – Comparaisons des descentes SPR, NNI et DPN sur des instances aléatoires comportant 100 séquences

solutions retournées par les descentes NNI, SPR et DPN, après 50, 100 et 500 milliers d'itérations de recherche locale depuis un arbre aléatoire. Chaque boite est définie par cinq valeurs : minimum, quartile inférieur, médiane, quartile supérieur et maximum. Cette figure illustre parfaitement la rapidité de convergence de l'algorithme DPN.

Avec l'instance `zilla`, connue pour sa difficulté, trouver le score minimum calculé à ce jour (16 218) par un algorithme de descente en un faible nombre d'itérations reste peu probable. Les résultats de notre descente à voisinage progressif sur cette instance nous permettront de situer son efficacité globale et de jauger l'écart en terme de performances qu'il reste à combler.

Nous remarquons particulièrement la puissance du voisinage progressif sur cette instance (tableau 5.5) ; lorsque la recherche est courte, *i.e.* entre 100 000 et 1 000 000 itérations, la recherche DPN pointe des solutions de bien meilleure qualité que les recherches traditionnelles. De plus, avec un effort de recherche plus important (20 000 000 itérations), l'algorithme DPN trouve des solutions d'excellente qualité. Sur les vingt exécutions, le meilleur score connu à ce jour a été trouvé une fois par cet algorithme. Le point intéressant est que les recherches DPN parviennent à retourner en moyenne de meilleures solutions que des recherches SPR prolongées jusqu'à la convergence.

Sur cette instance très large, nous avons également expérimenté des descentes depuis des arbres intermédiaires, générés avec une troisième méthode de construction : l'algorithme UPGMA. C'est l'objet de la figure 5.9, qui montre le comportement des trois recherches sur l'instance `zilla`. Une fois encore, nous remarquons la supériorité de la descente à voisinage progressif qui converge rapidement vers des solutions de bonne qualité.

		$\phi_0^{\sim}$	M	ϕ^*	∫	$\phi^{\sim}$	σ	tps (s)
300−100−5	$\mathfrak{C}_\mathfrak{a}$+SPR			1 583	1/100	1 649,5	30,3	0,8
	$\mathfrak{C}_\mathfrak{a}$+NNI		50 000	1 569	1/100	1 954,3	151,6	0,4
	$\mathfrak{C}_\mathfrak{a}$+DPN			**1 300**	**1/100**	**1 329,1**	**38,6**	**0,6**
	$\mathfrak{C}_\mathfrak{a}$+SPR			1 413	1/100	1 456,0	21,2	1,6
	$\mathfrak{C}_\mathfrak{a}$+NNI	*5 864*	100 000	1 307	1/100	1 667,1	148,2	1,2
	$\mathfrak{C}_\mathfrak{a}$+DPN			**1 300**	**7/100**	**1 306,5**	**18,7**	**1,4**
	$\mathfrak{C}_\mathfrak{a}$+SPR			1 301	1/100	1 307,3	3,2	6,5
	$\mathfrak{C}_\mathfrak{a}$+NNI		500 000	1 329	1/100	1 632,4	143,2	5,7
	$\mathfrak{C}_\mathfrak{a}$+DPN			**1 300**	**30/100**	**1 301,9**	**5,5**	**6,6**
	$\mathfrak{C}_\mathfrak{g}$+SPR			1 307	2/100	1 315,4	4,3	1,1
	$\mathfrak{C}_\mathfrak{g}$+NNI		50 000	1 300	1/100	1 304,7	2,7	0,8
	$\mathfrak{C}_\mathfrak{g}$+DPN			**1 300**	**4/100**	**1 302,6**	**1,4**	**1,2**
	$\mathfrak{C}_\mathfrak{g}$+SPR			1 304	4/100	1 311,0	3,3	1,9
	$\mathfrak{C}_\mathfrak{g}$+NNI	*1 325*	100 000	1 300	2/100	1 304,2	2,6	1,4
	$\mathfrak{C}_\mathfrak{g}$+DPN			**1 300**	**3/100**	**1 301,9**	**1,0**	**2,6**
	$\mathfrak{C}_\mathfrak{g}$+SPR			1 300	1/100	1 302,8	1,4	7,0
	$\mathfrak{C}_\mathfrak{g}$+NNI		500 000	1 301	10/100	1 304,2	2,7	5,8
	$\mathfrak{C}_\mathfrak{g}$+DPN			**1 300**	**19/100**	**1 301,0**	**0,7**	**8,0**
300−1000−5	$\mathfrak{C}_\mathfrak{a}$+SPR			16 789	1/100	17 422,8	253,1	9,2
	$\mathfrak{C}_\mathfrak{a}$+NNI		50 000	14 209	15/100	14 381,8	213,8	5,8
	$\mathfrak{C}_\mathfrak{a}$+DPN			**14 209**	**94/100**	**14 209,5**	**1,8**	**4,2**
	$\mathfrak{C}_\mathfrak{a}$+SPR			15 175	1/100	15 613,2	171,6	18,3
	$\mathfrak{C}_\mathfrak{a}$+NNI	*51 387*	100 000	14 209	14/100	14 412,2	309,1	11,1
	$\mathfrak{C}_\mathfrak{a}$+DPN			**14 209**	**100/100**	**14 209,0**	**0**	**40,9**
	$\mathfrak{C}_\mathfrak{a}$+SPR			14 209	1/100	14 234,7	16,4	64,2
	$\mathfrak{C}_\mathfrak{a}$+NNI		500 000	14 209	33/100	14 352,6	202,2	51,3
	$\mathfrak{C}_\mathfrak{a}$+DPN			**14 209**	**100/100**	**14 209,0**	**0**	**40,9**
	$\mathfrak{C}_\mathfrak{g}$+SPR			14 209	1/100	14 221,4	8,0	22,0
	$\mathfrak{C}_\mathfrak{g}$+NNI		50 000	14 209	76/100	14 209,7	1,9	15,1
	$\mathfrak{C}_\mathfrak{g}$+DPN			**14 209**	**100/100**	**14 209,0**	**0**	**20,3**
	$\mathfrak{C}_\mathfrak{g}$+SPR			14 209	3/100	14 217,1	7,2	32,6
	$\mathfrak{C}_\mathfrak{g}$+NNI	*14 230*	100 000	14 209	73/100	14 209,9	2,0	19,5
	$\mathfrak{C}_\mathfrak{g}$+DPN			**14 209**	**100/100**	**14 209,0**	**0**	**26,7**
	$\mathfrak{C}_\mathfrak{g}$+SPR			14 209	87/100	14 209,3	0,8	76,2
	$\mathfrak{C}_\mathfrak{g}$+NNI		500 000	14 209	78/100	14 209,5	1,8	59,7
	$\mathfrak{C}_\mathfrak{g}$+DPN			**14 209**	**100/100**	**14 209,0**	**0**	**60,0**

Tab. 5.3 – Comparaisons des descentes SPR, NNI et DPN sur des instances aléatoires comportant 300 séquences

		$\phi_0^{\sim}$	M	ϕ^*	∱	$\phi^{\sim}$	σ	tps (s)
500−100−5	$\mathfrak{C}_\alpha$+SPR	*9 164*	50 000	3 493	1/100	3 671,6	68,6	1,1
	$\mathfrak{C}_\alpha$+NNI			4 039	1/100	4 410,9	174,7	0,9
	$\mathfrak{C}_\alpha$+DPN			**2 279**	**1/100**	**2 367,4**	**53,2**	**1,1**
	$\mathfrak{C}_\alpha$+SPR		100 000	2 907	1/100	3 037,4	46,5	2,5
	$\mathfrak{C}_\alpha$+NNI			2 517	1/100	3 005,8	228,2	1,6
	$\mathfrak{C}_\alpha$+DPN			**2 243**	**5/100**	**2 248,0**	**6,5**	**2,2**
	$\mathfrak{C}_\alpha$+SPR		500 000	2 312	1/100	2 344,7	15,7	8,5
	$\mathfrak{C}_\alpha$+NNI			2 263	1/100	2 428,2	108,2	6,5
	$\mathfrak{C}_\alpha$+DPN			**2 243**	**48/100**	**2 244,0**	**3,0**	**9,8**
	$\mathfrak{C}_\beta$+SPR	*2 265*	50 000	2 259	1/100	2 273,1	6,7	2,6
	$\mathfrak{C}_\beta$+NNI			2 243	7/100	2 247,2	3,6	1,6
	$\mathfrak{C}_\beta$+DPN			**2 243**	**4/100**	**2 245,3**	**1,5**	**3,4**
	$\mathfrak{C}_\beta$+SPR		100 000	2 256	2/100	2 268,0	5,9	4,0
	$\mathfrak{C}_\beta$+NNI			2 243	10/100	2 246,6	3,7	2,6
	$\mathfrak{C}_\beta$+DPN			**2 243**	**21/100**	**2 244,2**	**1,1**	**4,2**
	$\mathfrak{C}_\beta$+SPR		500 000	2 246	2/100	2 251,3	2,8	9,1
	$\mathfrak{C}_\beta$+NNI			2 243	28/100	2 245,6	3,0	7,2
	$\mathfrak{C}_\beta$+DPN			**2 243**	**61/100**	**2 243,5**	**0,7**	**13,1**
500−1000−5	$\mathfrak{C}_\alpha$+SPR	*100 388*	50 000	35 980	1/100	37 690,7	580,4	17,6
	$\mathfrak{C}_\alpha$+NNI			25 821	1/100	29 080,7	1 767,8	18,3
	$\mathfrak{C}_\alpha$+DPN			**24 319**	**5/100**	**24 391,9**	**141,6**	**5,2**
	$\mathfrak{C}_\alpha$+SPR		100 000	30 837	1/100	31 917,0	395,2	30,9
	$\mathfrak{C}_\alpha$+NNI			24 319	3/100	24 693,6	754,5	33,0
	$\mathfrak{C}_\alpha$+DPN			**24 319**	**91/100**	**24 319,6**	**1,9**	**10,2**
	$\mathfrak{C}_\alpha$+SPR		500 000	24 938	1/100	25 205,6	116,0	100,4
	$\mathfrak{C}_\alpha$+NNI			24 319	3/100	24 618,5	593,2	66,3
	$\mathfrak{C}_\alpha$+DPN			**24 319**	**100/100**	**24 319,0**	**0**	**54,2**
	$\mathfrak{C}_\beta$+SPR	*24 359*	50 000	24 326 (-3)	1/100	24 351,4	13,6	85,2
	$\mathfrak{C}_\beta$+NNI			24 319	70/100	24 321,0	3,4	49,4
	$\mathfrak{C}_\beta$+DPN			**24 319**	**98/100**	**24 219,1**	**0,7**	**81,1**
	$\mathfrak{C}_\beta$+SPR		100 000	24 326	1/100	24 346,8	12,2	102,1
	$\mathfrak{C}_\beta$+NNI			24 319	76/100	24 320,5	2,9	53,8
	$\mathfrak{C}_\beta$+DPN			**24 319**	**100/100**	**24 219,0**	**0**	**86,4**
	$\mathfrak{C}_\beta$+SPR		500 000	24 319	6/100	24 325,8	5,3	160,1
	$\mathfrak{C}_\beta$+NNI			24 319	76/100	24 320,6	3,1	104,0
	$\mathfrak{C}_\beta$+DPN			**24 319**	**100/100**	**24 219,0**	**0**	**143,8**

Tab. 5.4 – Comparaisons des descentes SPR, NNI et DPN sur des instances aléatoires comportant 500 séquences

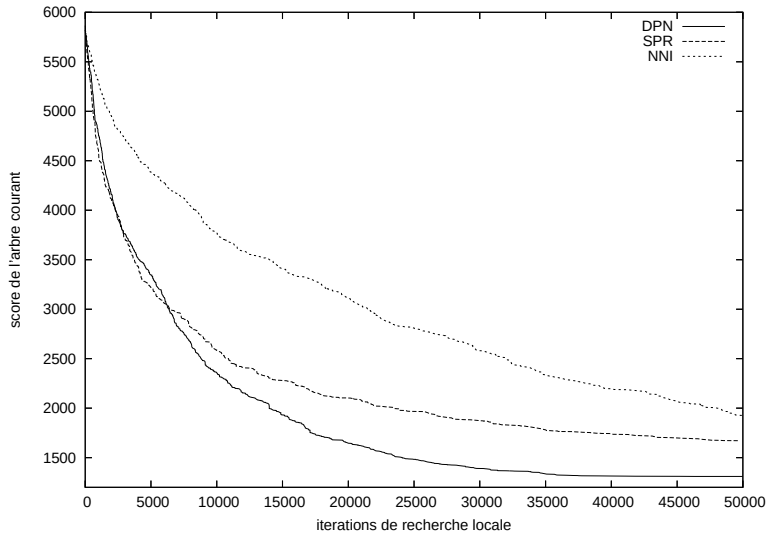


FIG. 5.7 – Évolution des scores des arbres courants pour les descentes SPR, NNI et DPN sur l'instance 300-100-5 depuis un arbre aléatoire

5.5 Performances comparatives par rapport à un logiciel GRASP+VND

Dans cette section, nous estimons la performance de la descente à voisinage progressif DPN par rapport à un algorithme GRASP+VND [Ribeiro and Vianna, 2005], très efficace sur ce problème. Puisque le logiciel n'est pas disponible, nous utilisons pour les comparaisons l'ensemble de huit instances réelles (tableau 5.6) utilisées dans l'article cité.

GRASP+VND (voir chapitre 3, sections 3.4.2 et 3.4.5) est une technique récente qui applique deux métaheuristiques éprouvées, GRASP [Feo and Resende, 1995] et VND [Hansen and Mladenović, 1999], au problème MP. Une solution initiale est construite avec un algorithme glouton randomisé, puis est amélioré avec une descente à voisinage variable utilisant le voisinage l-SPR (chapitre 3, section 3.4.1). Cet algorithme détient, sur les huit instances utilisées ici, les meilleurs résultats connus.

Dans un premier temps, nous limitons le nombre d'itérations de l'algorithme DPN, en fixant $M = 50\ 000$ et en débutant la recherche avec un arbre construit par un algorithme glouton. Nous avons procédé à 100 exécutions de l'algorithme DPN sur chaque instance. Les auteurs n'indiquent que le score retourné par la meilleure exécution, et le temps de calcul moyen.

Sur la première partie du tableau 5.6, nous observons qu'en moins de 50 000 iterations, DPN est capable d'égaler les résultats de GRASP+VND pour cinq des huit instances en un temps de calcul réduit (quelques minutes pour l'ensemble des 100 exécutions). Si nous étendons la recherche à 200 000 itérations (seconde partie du tableau), DPN est capable de trouver les meilleurs résultats pour les autres instances. Pour des résultats équivalents, GRASP+VND requiert plusieurs heures voire plusieurs jours de calcul. Étant donnée la sim-

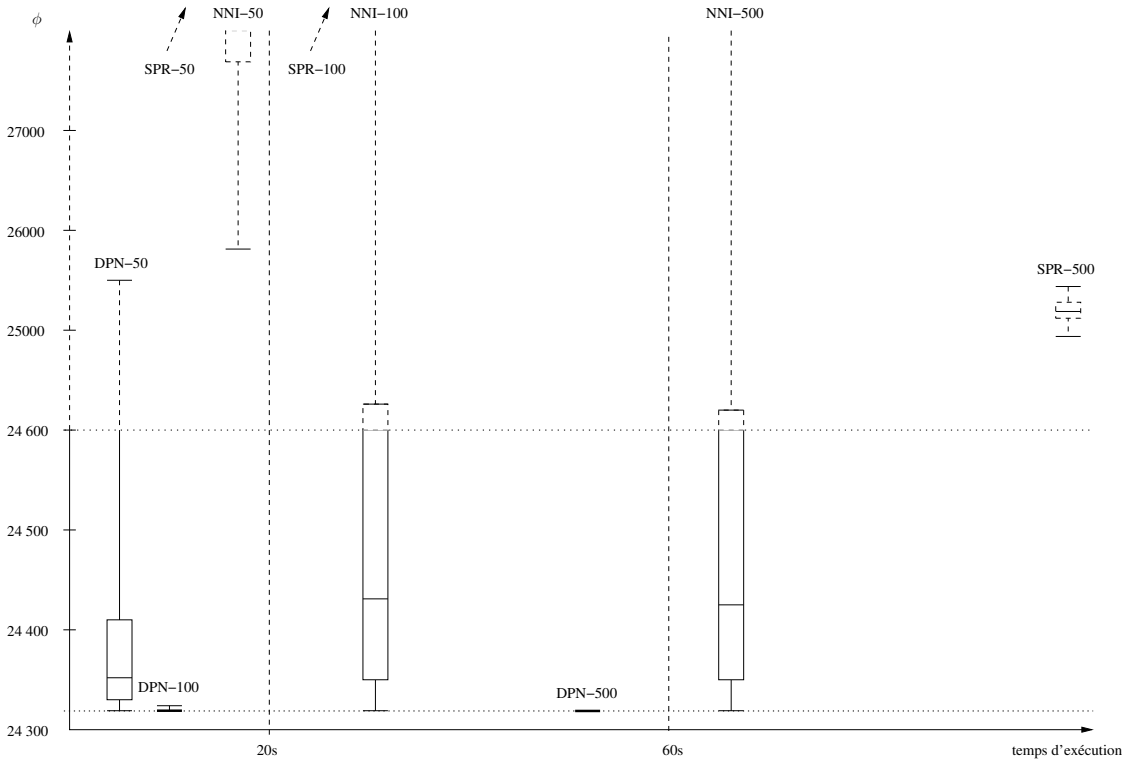


FIG. 5.8 – Performances comparées sur l'ensemble des exécutions (instance `500-1000-5`)

plicité de DPN et le nombre limité d'arbres visités durant la recherche, nous pouvons considérer cette recherche locale à voisinage progressif comme un procédé très intéressant, et une excellente base pour la conception d'un algorithme efficace, fiable et rapide.

5.6 Vers un voisinage *réactif*

La descente à voisinage progressif permet de réduire considérablement les temps d'excution des descentes classiques en ciblant les voisins intéressants avant même leur évaluation. Cela passe par la définition d'un schéma d'évolution du voisinage paramétrique. Le schéma de réduction est censé reproduire la distribution optimale des meilleurs voisins, que nous avons observée à partir d'un arbre aléatoire et sur quelques instances aléatoires ou réelles.

Néanmoins, ce schéma ne s'adapte pas idéalement si l'arbre initial est de bonne qualité — ce qui est le cas lorsqu'il est construit avec un algorithme glouton —, ou sur une instance non structurée dont la répartition des meilleurs voisins pourrait suivre une évolution différente de celle observée dans la section 5.3.1. Nous remarquons ceci sur la figure 5.9 où, partant d'une solution ayant subi un pré-traitement, NNI améliore la solution courante plus rapidement que DPN durant les premières itérations. Enfin, le schéma de voisinage progressif que nous avons défini nécessite un paramètre — le nombre d'itérations de re-

		$\phi_0^{\sim}$	M	ϕ^*	ƒ	$\phi^{\sim}$	σ	tps (s)
zilla	$\mathfrak{C}_\alpha$+SPR	*37 427*	10^5	18 013	1/100	18 470,2	126,9	18,5
	$\mathfrak{C}_\alpha$+NNI			19 875	1/100	21 863,2	894,9	10,6
	$\mathfrak{C}_\alpha$+**DPN**			**16 263**	**1/100**	**16 553,0**	**204,3**	**7,7**
	$\mathfrak{C}_\alpha$+SPR		10^6	16 272	1/100	16 312,8	20,4	135,3
	$\mathfrak{C}_\alpha$+NNI			19 142	1/100	21 342,4	1 102,9	118,4
	$\mathfrak{C}_\alpha$+**DPN**			**16 222**	**3/100**	**16 257,0**	**45,3**	**78,5**
	$\mathfrak{C}_\alpha$+SPR		2.10^7	16 219	1/20	16 225,8	4,8	2 325
	$\mathfrak{C}_\alpha$+**DPN**			**16 218**	**1/20**	**16 223,3**	**3,9**	**1 609**
	$\mathfrak{C}_\mathfrak{g}$+SPR	*16 475*	10^5	16 335	1/100	16 384,1	21,9	69,1
	$\mathfrak{C}_\mathfrak{g}$+NNI			16 255	1/100	16 313,0	22,4	39,6
	$\mathfrak{C}_\mathfrak{g}$+**DPN**			**16 226**	**1/100**	**16 245,9**	**11,8**	**66,8**
	$\mathfrak{C}_\mathfrak{g}$+SPR		10^6	16 237	1/100	16 256,4	8,3	203,8
	$\mathfrak{C}_\mathfrak{g}$+NNI			16 257	1/100	16 311,3	28,1	141,3
	$\mathfrak{C}_\mathfrak{g}$+**DPN**			**16 219**	1/100	**16 228,5**	**6,7**	**140,9**

TAB. 5.5 – Comparaisons des descentes SPR, NNI et DPN sur l'instance `zilla`

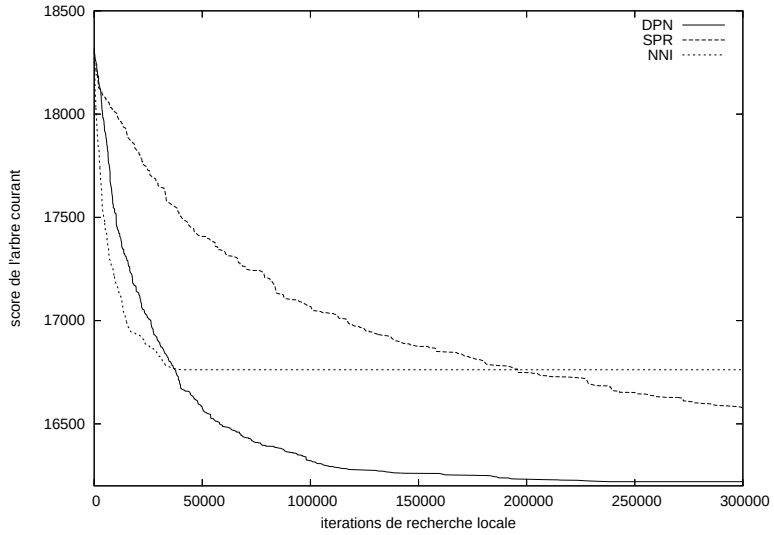


FIG. 5.9 – Évolution du score de la solution courante depuis une solution intermédiaire construite par UPGMA (instance `zilla`)

cherche locale désiré — dont la valeur adéquate n'est pas toujours évidente à déterminer.

De cette observation, nous avons imaginé utiliser ce même voisinage paramétrique, mais sans schéma prédéterminé et sans paramètre spécifique à chaque utilisation. Il s'agit de concevoir un voisinage réactif, où le paramètre d se régule automatiquement à partir des propriétés des solutions précédemment explorées. L'algorithme 5.3 décrit un tel mécanisme.

La mise à jour du paramètre d se fait périodiquement, après une série d'itérations

			DPN						GRASP+VND	
Instance	n	m	ϕ^*	tps (s)	∮	$\phi^\sim$	σ	M	ϕ^*	tps (s)
bin-GRIS	47	93	**172**	1,7	100/100	172.0	0	5.10^4	**172**	3 505
bin-ANGI	49	58	**216**	1,3	8/100	218.0	1.4		**216**	5 099
bin-TENU	56	179	**682**	2,6	9/100	686.6	3.7		**682**	7 497
bin-ETHE	58	86	**372**	1,5	2/100	374.3	1.0		**372**	10 042
bin-ROPA	75	82	326	1,6	5/100	328.7	1.6		**325**	15 764
bin-GOLO	77	97	497	2,1	1/100	505.8	4.2		**496**	32 836
bin-SCHU	113	146	**759**	3,5	1/100	768.6	5.1		**759**	113 391
bin-CARP	117	110	549	3,0	1/100	554.8	3.1		**548**	82 176
bin-ROPA	75	82	**325**	7,3	3/100	328.1	1.7	2.10^5	**325**	15 764
bin-GOLO	77	97	**496**	8,0	2/100	502.7	2.9		**496**	32 836
bin-SCHU	113	146	**759**	12,4	10/100	766.4	4.5		**759**	113 391
bin-CARP	117	110	**548**	10,9	4/100	553.4	3.5		**548**	82 176

Tab. 5.6 – Comparaisons entre DPN et l'algorithme `GRASP+VND`

correspondant à la longueur d'un intervalle. Si, durant cette période, aucune amélioration de la solution n'est survenue, alors le voisinage est étendu pour augmenter les possibilités d'améliorer la solution courante. Si l'échec est répété lors de plusieurs intervalles consécutifs — en l'occurence trois, tel que nous l'avons défini —, alors le voisinage se rétracte. Il s'agit du cas où le nombre de voisins améliorants semble réduit, ce qui requiert une évaluation de voisins plus proches (voir section 5.3).

Ce nouveau schéma d'évolution du paramètre d permet de s'adapter à toute instance, et surtout à tout arbre de départ. Par rapport au schéma linéaire vu précédemment, l'intérêt est double : tout d'abord la suppression du paramètre M (l'effort fixé de recherche), ensuite un gain de temps supplémentaire avec une définition adaptative de la valeur initiale du paramètre d et un schéma adapté à la recherche courante.

D'après nos premiers tests, l'utilisation de ce voisinage permet généralement d'aboutir à des résultats similaires à ceux de DPN, en des temps de calculs moindres et sans paramétrage initial de la durée allouée de la recherche. Il s'agit de travaux très récents dont les premiers résultats sont encourageants bien qu'ils soient encore à affiner ; nous pensons que quelques ajustements de l'algorithme permettront une amélioration complémentaire en terme de robustesse et de temps de calcul par rapport au voisinage progressif de base.

5.7 Conclusion

L'objectif de ce chapitre était de comprendre l'influence du voisinage utilisé en fonction des instances et de l'avancée de la recherche locale, puis de proposer une alternative qui combine les propriétés intéressantes des voisinages SPR et NNI. Nous avons alors introduit la notion de voisinage progressif, et la série d'expérimentations effectuée montre un gain d'efficacité sensible par rapport à SPR et NNI, notamment sur les instances plus difficiles.

Algorithme 5.3 : Descente à voisinage réactif

Données : une matrice de caractères $n \times m$ induisant un problème de minimisation $(\mathcal{T}, \phi)$, une méthode de construction $\mathfrak{C}$, un voisinage paramétrique $\mathcal{N}_d$.
Résultat : le meilleur arbre trouvé.
début
  $longueurIntervalle \leftarrow \lfloor 500 + \frac{n^2}{10} \rfloor$ (constante);
  $maxIntervalle \leftarrow 3$ (constante);
  construire un arbre initial t selon $\mathfrak{C}$;
  choisir arbitrairement une valeur initiale strictement positive de d ;
  $numIteration \leftarrow 1$;
  $numIntervalle \leftarrow 0$;
  $maxd \leftarrow 0$;
  tant que $d > 0$ **faire**
    si $numIteration \% longueurIntervalle = 0$ **alors**
      si $\neg amelioration$ **alors**
        si $numIntervalle < maxIntervalles$ **alors**
          $d \leftarrow d + 1$;
        sinon
          $d \leftarrow d - 1$;
        fin
        $numintervalle \leftarrow numintervalle + 1$;
      sinon
        $d \leftarrow maxd + 1$;
        $numintervalle \leftarrow 0$;
      fin
      $amelioration \leftarrow Faux$;
      $maxd \leftarrow 0$;
    fin
    sélectionner $t' \in \mathcal{N}_d(t)$;
    $\delta \leftarrow distance(t', t)$;
    si $\phi(t') \leqslant \phi(t)$ **alors**
      $t \leftarrow t'$;
      si $\phi(t') < \phi(t)$ **alors**
        $amelioration \leftarrow Vrai$;
        si $\delta > maxd$ **alors**
          $maxd \leftarrow \delta$;
        fin
      fin
    fin
    $numiteration \leftarrow numiteration + 1$;
  fin
fin

De plus, sa robustesse entraine un gain de temps considérable, car elle permet de minimiser le nombre de relances. En effet, l'arbre initial influe très peu sur la qualité des solutions retournées par la descente à voisinage progressif.

Chapitre 6

Croisement d'arbres spécifique et algorithme mémétique

La conception d'un algorithme performant pour la résolution du problème MP nécessite deux mécanismes qui régissent le parcours de l'espace de recherche : une procédure d'intensification efficace et rapide, ce qui est le cas de notre recherche locale à voisinage évolutif décrit au chapitre précédent, et une procédure de diversification capable de cibler des zones pertinentes de l'espace de recherche. Ce second point est l'objet de ce chapitre. Nous définissons ici un croisement d'arbres permettant une diversification plus fine des recherches locales, en vue de résoudre des instances difficiles. Cela aboutit à la conception d'un algorithme mémétique qui hybride la recherche locale à voisinage progressif avec un algorithme génétique basé sur ce nouveau croisement d'arbres.

Une partie de ce travail a été publiée dans [Goëffon *et al.*, 2006].

Sommaire

6.1 Problématique

Un algorithme de descente tel que nous l'avons présenté au chapitre précédent permet de trouver rapidement une solution approchée au problème MP. Pour certaines instances, notamment lorsque le nombre de séquences considérées est restreint, l'expérience montre que ces méthodes sont efficaces — dans le sens où les meilleurs scores de parcimonie calculés sont souvent équivalents ou très proches des minimums connus. Pour maximiser l'efficacité et la fiabilité du résultat, on effectue traditionnellement plusieurs descentes successives depuis plusieurs arbres de départ. Cependant, lorsque les espaces de recherche sont très grands comme c'est le cas pour des instances comportant plusieurs centaines de séquences, les minimums locaux deviennent trop nombreux pour s'assurer que les solutions trouvées par les descentes soient suffisamment proches des solutions optimales.

Certaines instances, de par leur taille ou leur structure, sont plus difficiles à résoudre par des algorithmes de descente, où l'on remarque davantage une corrélation entre les points de départ et les résultats. Typiquement, ces solutions de départ, sur lesquelles sont appliquées les descentes, sont construites aléatoirement ou bien suivant une heuristique de construction. Générer plusieurs solutions de départ de manière aléatoire assure une certaine diversité mais reste pauvre en information ; lorsque les instances sont difficiles, les descentes associées donnent généralement de mauvais résultats. En revanche, générer plusieurs arbres de départ avec la même heuristique de construction, même si celle-ci est non-déterministe, fait perdre de la diversité. Il sera en effet probable que dans ce cas, les solutions de départ soient proches les unes des autres, et les descentes associées risquent d'explorer les mêmes zones de l'espace de recherche. Enfin, les recherches locales itérées comme la méthode du *Ratchet* (voir chapitre 3, section 3.4.3) qui utilisent comme nouveau point de départ un précédent optimum bruité, gagnent en diversité mais seulement au prix d'une plus grande perte d'information : plus le précédent optimum sera perturbé, plus la diversification de la recherche sera importante, mais plus la génération de ce nouveau point de départ s'apparentera à une construction aléatoire.

Les algorithmes mémétiques permettent de diversifier la recherche sans trop détériorer les solutions, et en évitant d'incorporer des mécanismes stochastiques dispensables. Ceci dans le but de traiter des instances plus importantes, de renforcer la robustesse tout en conservant des temps de calcul raisonnables. La figure 6.1 résume les différentes stratégies de recherche.

Un algorithme mémétique est un algorithme génétique hybridé avec une méthode de recherche locale. La partie génétique s'attache à générer une population d'individus solutions, à en sélectionner des paires (voire des sous-ensembles plus importants), puis à en croiser les éléments pour générer de nouveaux individus. Ceux-ci prendront alors la place d'autres individus, et ainsi de suite sur plusieurs générations. La spécificité de chaque algorithme génétique réside essentiellement dans le croisement utilisé. Quelques algorithmes mémétiques ont déjà été appliqués au problème MP (chapitre 3, section 3.5).

Comme nous l'avons vu, les croisements d'arbres communément utilisés dans ces algorithmes combinent des sous-arbres de chacun des parents. La démarche classique (figure 6.2) consiste à prendre un sous-arbre T_1^* d'un parent T_1, puis à l'insérer dans le second parent T_2 — après suppression dans T_2 des taxons présents dans T_1^*.

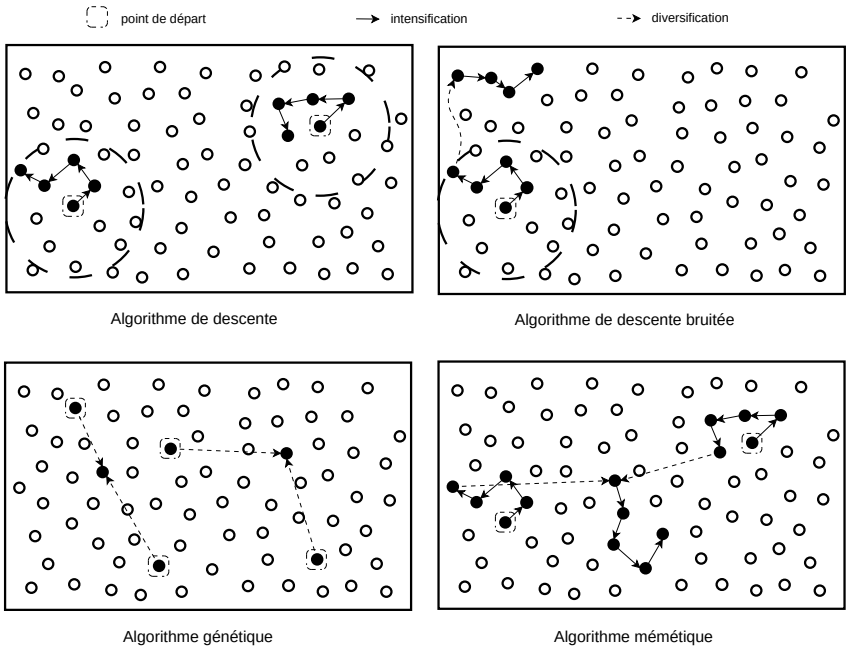


FIG. 6.1 – Fonctionnement d'un algorithme mémétique

La topologie d'un arbre phylogénétique — et par conséquent son score de parcimonie — est déterminée par la position de ses taxons. Or, dans cet exemple, seulement la moitié des informations sont prises en compte dans l'élaboration du croisement. En effet, les positions des taxons de T_1^* dans T_2, et celles des autres taxons dans T_1, n'a aucune incidence sur le résultat. Dans un algorithme mémétique où les parents ont pu subir une phase de recherche locale, se priver de la moitié de leur information topologique semble pénalisant. De plus, et c'est à notre avis le point le plus important, aucune concordance entre les informations des deux parents n'est vérifiée lors de cette procédure. Ainsi, les informations fortes, partagées par les deux parents, ne sont pas conservées en priorité. Par exemple sur la figure 6.2, nous constatons que chez l'enfant, les espèces F et N sont assez proches alors qu'elles ne le sont chez aucun des deux parents.

Nous remarquons en pratique que le fait de conserver des sous-topologies[1] de grande taille — c'est le cas dans notre exemple précédent où une bipartition des taxons suffit à générer une sous-topologie pour chacun des parents —, entraine fréquemment une grande perte d'informations tout en n'assurant pas une grande diversité. Plus précisément, en répliquant des morceaux topologiques des parents, on risque de faire face à de nombreuses ambigüités[2]. Pour les lever, les méthodes actuelles font des choix, généralement arbitraires ;

[1]Nous définissons une *sous-topologie* $T^* = (V^*, E^*)$ d'un arbre $T = (V, E)$ comme un arbre tel que $V^* \subseteq V$ et $\forall (v_1, v_2) \in E^*$, il existe un chemin allant de v_1 à v_2 dans T. Un *sous-arbre* T^* de T, *i.e.* tel que $V^* \subseteq V$ et $E^* \subseteq E$, en est un cas particulier.

[2]Deux arbres distincts possédant les même taxons véhiculent nécessairement des informations contradictoires.

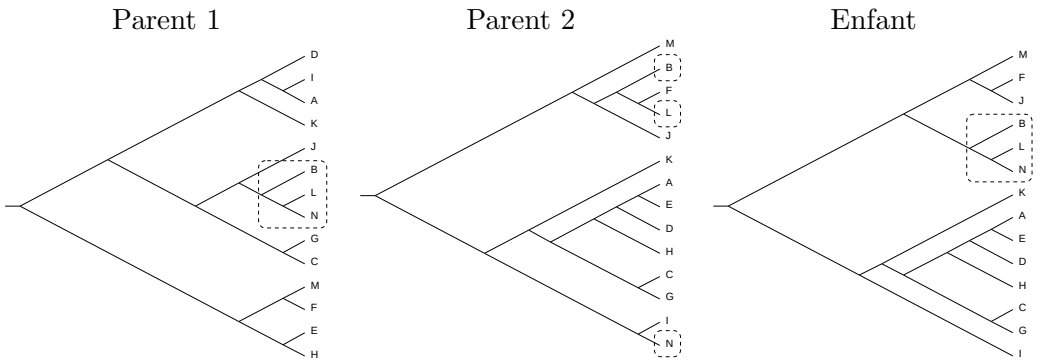


FIG. 6.2 – Résultat d'un croisement d'arbres classique

de plus, les fils peuvent être très proches topologiquement de l'un des deux parents. Nous proposons dans ce qui suit un moyen d'utiliser l'ensemble des informations véhiculées par chacun des parents à l'aide d'un croisement qui construit un fils en fonction des informations topologiques de ses deux parents, mais sans conserver nécessairement les topologies elles-mêmes. Les ambigüités seront levées par des décisions consensuelles et non arbitraires.

Dans ce chapitre, nous proposons en premier lieu un cadre général de croisement d'arbres qui préserve au maximum les informations pertinentes contenues dans chacun des parents. Nous laissons la possibilité de modifier chaque étape du croisement, et éventuellement de l'adapter à d'autres problèmes. Il s'agit d'un croisement préservant les propriétés représentatives des parents et basé sur des distances entre objets (ici des arbres), nommé *`DiBIP` crossover* (*Distance-Based Information Preservation Tree Crossover*). Quelques exemples de croisements pour le problème MP et des résultats expérimentaux confirmeront le bien-fondé de cette approche.

6.2 Croisement `DiBIP` : cadre général

L'idée du croisement `DiBIP` est de synthétiser les propriétés topologiques des deux parents, et de construire un fils qui s'éloigne le moins possible de ces propriétés. L'approche générale peut être résumée en trois étapes :

1. Représenter chaque parent par une matrice de distances ;
2. Combiner les matrices des deux parents pour obtenir une troisième matrice ;
3. Construire un fils à partir de cette troisième matrice.

Afin de décrire plus formellement le schéma général du croisement `DiBIP`, nous proposons d'introduire quelques notations :

- T_1 et T_2 représentent les deux arbres parents utilisés pour le croisement ;
- $\delta_T : L \times L \rightarrow \mathbb{Q}$ est une distance topologique entre chaque paire de taxons $(i, j) \in L^2$ d'un arbre T ;

- $\Delta : \mathcal{T} \to \mathcal{D}$ est une application qui transforme un arbre en matrice de distances ;
- $\oplus : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ est un opérateur matriciel qui combine deux matrices de distances en une nouvelle matrice ;
- $\Lambda : \mathcal{D} \to \mathcal{T}$ est une application qui construit un arbre à partir d'une matrice de distances entre taxons.

À partir de ces notations, le schéma général du croisement DiBIP peut être décrit par l'algorithme 6.1.

Algorithme 6.1 : Schéma général du croisement DiBIP

Données : T_1, T_2, δ_T, Δ, $\oplus$, Λ
Résultat : Un arbre fils T^*
début
  $D_1 \leftarrow \Delta(T_1)$;
  $D_2 \leftarrow \Delta(T_2)$;
  $D^* \leftarrow D_1 \oplus D_2$;
  $T^* \leftarrow \Lambda(D^*)$;
  Retourner T^* ;
fin

Nous allons à présent proposer un cadre de définition de chacun des opérateurs Δ, $\oplus$ et Λ.

6.2.1 Conversion Arbre → Matrice

La mesure de distances δ, unique paramètre de l'opérateur Δ, doit idéalement être liée à la position relative des taxons dans un arbre ; ce qui en particulier nécessite que δ soit fonction de T. La conventionnelle distance de Hamming entre les séquences associées à deux taxons i et j, par exemple, n'est pas applicable ici car cette métrique est totalement indépendante de la topologie de l'arbre concerné. En revanche, la distance topologique entre deux taxons i et j dans un arbre T, *i.e.* la longueur du chemin les séparant, est une donnée simple à calculer et bien adaptée à la transcription Arbre → Matrice. Il est également possible d'étendre cette distance au nombre de changements d'états présents sur le chemin menant de i à j.

6.2.2 Croisement de matrices

Propriétés de l'opérateur

Pour transmettre durant le croisement les propriétés représentatives partagées par les parents, l'opérateur matriciel $\oplus$ devra vérifier quelques conditions spécifiques. Par exemple, si une paire de taxons (a, b) est plus proche d'une autre paire (c, d) dans chacun des deux parents, alors cette information sémantique devra dans la mesure du possible être conservée par le processus de croisement, et transmise au fils en résultant. Nous appelons cette condition la *propriété de préservation des relations* (propriété 4).

Propriété 4 (Préservation des relations) *Soient D_1 et D_2 les matrices de distances de deux arbres T_1 et T_2 telles que $D_i(x,y) = \delta_{T_i}(x,y)$ ($i \in \{1,2\}$), et $D^* = D_1 \oplus D_2$ la matrice issue de D_1 et D_2 par l'opérateur $\oplus$. L'opérateur matriciel $\oplus$ vérifie la propriété de préservation des relations si et seulement si, étant donnés quatre taxons a, b, c, d et une relation $\lhd \in \{<, =, >\}$, l'implication suivante est toujours vérifiée :*

$$(D_1(a,b) \lhd D_1(c,d)) \wedge (D_2(a,b) \lhd D_2(c,d)) \Rightarrow (D^*(a,b) \lhd D^*(c,d)) \tag{6.1}$$

Pour des raisons pratiques et sémantiques, $\oplus$ doit également vérifier certaines propriétés algébriques (propriété 5).

Propriété 5 *1. $\oplus$ doit être commutatif ;*

2. La multiplication par un scalaire $\lambda \in \mathbb{Q}_+^*$ doit être distributive par rapport à $\oplus$: $\lambda.(D_1 \oplus D_2) = \lambda.D_1 \oplus \lambda.D_2$.

La commutativité de $\oplus$ impose que les deux parents, symbolisés par les deux matrices de distances, ne soient pas ordonnés. Il n'y a en particulier aucune notion de *donneur* ou de *receveur*. La seconde propriété est liée à la définition de Λ et nous verrons qu'elle permet de procéder à quelques simplifications.

Exemples d'opérateurs valides

Les deux matrices D_1 et D_2 de deux parents T_1 et T_2 donnent, pour chaque couple de taxons (i,j), deux distances : $\delta_{T_1}(i,j)$ et $\delta_{T_2}(i,j)$. Définir $\oplus$ tel que chaque élément de la matrice fille soit la moyenne arithmétique des deux matrices parentes, *i.e.* $(D_1 \oplus D_2)(i,j) = 0,5.D_1(i,j) + 0,5.D_2(i,j)$, vérifie les contraintes de l'opérateur et tient compte uniformément des topologies de T_1 et de T_2 (cas (a) dans le tableau 6.1). Il est également possible de privilégier les taxons très proches dans l'un des deux parents, ou plus éloignés, (cas (b) à (e)), suivant l'information que l'on souhaite retrouver en priorité dans l'arbre fils, qui sera construit à partir de cette matrice fille. Enfin, les cas (f) et (g), avec $\epsilon < [\max_{i,j}(\delta_1, \delta_2)]^{-1}$, constituent une alternative en ordonnant les valeurs égales suivant un second critère, comme l'écart entre les deux valeurs.

Un opérateur matriciel $\oplus$ défini par $(D_1 \oplus D_2)(i,j) = \alpha.\min\{D_1(i,j), D_2(i,j)\} + (1-\alpha).\max\{D_1(i,j), D_2(i,j)\}$, avec $\alpha \in [0,1]$, semble particulièrement adapté au principe du croisement `DiBIP`. La moyenne arithmétique ($\alpha = 0,5$) et l'opérateur max ($\alpha = 0$) en sont deux cas particuliers. Le paramètre α permet d'ajuster à quel point l'on privilège, lors de contradictions entre les deux parents, les paires de taxons plus proches ou bien plus éloignées. Nous pouvons démontrer que cet opérateur vérifie les propriétés 4 et 5 :

Propriété 6 *Soit $\oplus : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ défini tel que $(D_1 \oplus D_2)(i,j) = \alpha.\min\{D_1(i,j), D_2(i,j)\} + (1-\alpha).\max\{D_1(i,j), D_2(i,j)\}$, avec $\alpha \in [0,1]$. Alors $\oplus$ vérifie la propriété de préservation des relations (1), est commutatif (2) et $\forall \lambda \in \mathbb{Q}_+^*, \lambda.(D_1 \oplus D_2) = \lambda.D_1 \oplus \lambda.D_2$ (3).*

		(1)	(2)	(3)	(4)
	$\delta_1 = \delta_{T_1}(i,j)$	3	1	5	1
	$\delta_2 = \delta_{T_2}(i,j)$	3	5	1	9
(a)	$0,5.\delta_1 + 0,5.\delta_2$	3	3	3	5
(b)	$\min(\delta_1, \delta_2)$	3	1	1	1
(c)	$\max(\delta_1, \delta_2)$	3	5	5	9
(b)	$0,75.\min(\delta_1, \delta_2) + 0,25.\max(\delta_1, \delta_2)$	3	2	2	3
(e)	$0,25.\min(\delta_1, \delta_2) + 0,75.\max(\delta_1, \delta_2)$	3	4	4	7
(f)	$\delta_1 + \delta_2 + \epsilon.\lvert\delta_1 - \delta_2\rvert$	6	$10 + 3\epsilon$	$6 + 3\epsilon$	$10 + 5\epsilon$
(g)	$\delta_1 + \delta_2 - \epsilon.\lvert\delta_1 - \delta_2\rvert$	6	$10 - 3\epsilon$	$6 - 3\epsilon$	$10 - 5\epsilon$

TAB. 6.1 – Exemples de différentes définitions de $\oplus$

Preuve :

(1) Soit a, b, c et d quatre taxons tels que $0 \leqslant D_1(a,b) < D_1(c,d)$ et $0 \leqslant D_2(a,b) < D_2(c,d)$. Alors $\min\{D_1(a,b), D_2(a,b)\} < \min\{D_1(c,d), D_2(c,d)\}$.
De même, $\max\{D_1(a,b), D_2(a,b)\} < \max\{D_1(c,d), D_2(c,d)\}$.
Soit $\alpha \in [0,1]$, alors $\alpha - 1 \in [0,1] \geqslant 0$. Par conséquent, $\alpha.\min\{D_1(a,b), D_2(a,b)\}$
$< \alpha.\min\{D_1(c,d), D_2(c,d)\}$ et $(\alpha-1).\max\{D_1(a,b), D_2(a,b)\}$
$< (\alpha-1).\max\{D_1(c,d), D_2(c,d)\}$. Tous les facteurs sont positifs, on en déduit donc :
$\alpha.\min\{D_1(a,b), D_2(a,b)\} + (\alpha-1).\max\{D_1(a,b), D_2(a,b)\} < \alpha.\min\{D_1(c,d), D_2(c,d)\} + (\alpha-1).\max\{D_1(c,d), D_2(c,d)\}$.
On montre de la même manière la préservation des relations $=$ et $>$.
(2) $\forall (i,j) \in L^2, (D_1 \oplus D_2)(i,j) = \alpha.\min\{D_1(i,j), D_2(i,j)\} + (1-\alpha).\max\{D_1(i,j), D_2(i,j)\}$
$= \alpha.\min\{D_2(i,j), D_1(i,j)\} + (1-\alpha).\max\{D_2(i,j), D_1(i,j)\} = (D_2 \oplus D_1)(i,j)$.
(3) Soit $\lambda \in \mathbb{Q}_+^*$. $\forall (i,j) \in L^2, \lambda.(D_1(i,j) \oplus D_2(i,j))$
$= \lambda.(\alpha.\min\{D_1(i,j), D_2(i,j)\} + (1-\alpha).\max\{D_1(i,j), D_2(i,j)\})$
$= \lambda.\alpha.\min\{D_1(i,j), D_2(i,j)\} + \lambda.(1-\alpha).\max\{D_1(i,j), D_2(i,j)\}$
$= \alpha.\min\{\lambda.D_1(i,j), \lambda.D_2(i,j)\} + \lambda.(1-\alpha).\max\{\lambda.D_1(i,j), \lambda.D_2(i,j)\}$
$= \lambda.D_1(i,j) \oplus \lambda.D_2(i,j)$.

□

6.2.3 Construction de l'arbre fils

La troisième et dernière étape du croisement est la construction de l'arbre T^* à partir de la matrice $D^* = D_1 \oplus D_2$. La méthode utilisée, notée Λ, doit être rapide tout en préservant au maximum les informations présentes dans D^*. De plus, Λ doit vérifier la propriété 7.

Propriété 7 $\forall \lambda \in \mathbb{Q}_+^*$, $\Lambda(\lambda.D^*) = \Lambda(D^*)$.

Λ peut être non-déterministe ; si tel est le cas, alors elle prend un facteur stochastique ζ (*graine*) en paramètre supplémentaire. Dans ce cas la propriété 7 devient $\Lambda(\lambda.D^*, \zeta) = \Lambda(D^*, \zeta)$.

Les propriétés 5.2 et 7 permettent d'assurer l'égalité suivante :

Propriété 8 $\forall \lambda \in \mathbb{Q}_+^*, \Lambda(\lambda.D_1 \oplus \lambda.D_2) = \Lambda(D_1 \oplus D_2)$

Notons que $\Lambda \neq \Delta^{-1}$, et que la matrice de distance de l'arbre obtenu $\Delta(T^*)$ est en général différente de D^*. De plus, quel que soit l'opérateur $\oplus$ utilisé, il n'existe en général aucun arbre T tel que $\Delta(T) = \Delta(T_1) \oplus \Delta(T_2)$.

Des définitions appropriées de $\oplus$ et Λ, *i.e.* qui vérifient toutes les propriétés énoncées, assurent au maximum la préservation des informations topologiques communes aux deux parents.

6.2.4 Applications

Nous venons de définir un cadre général pour un croisement d'arbre original, fondamentalement différent des croisements communément utilisés. Le croisement `DiBIP` dispose en outre de trois paramètres qui pourront être redéfinis afin de l'employer dans d'autres problèmes combinatoires. Seules les relations entre les feuilles de l'arbre ont un sens pour la résolution du problème MP, mais `DiBIP` prévoit la possibilité de considérer tous les nœuds de l'arbre si l'on traite un autre problème, voire de l'adapter à un croisement de graphes.

Le fait d'abstraire des informations sémantiques des deux parents puis de construire un individu fils uniquement à partir de ces informations, permet à la fois de diversifier la recherche puisqu'il n'y a pas de recopie de morceaux de parents, et d'éviter d'incorporer du bruit dans le croisement en reprenant des propriétés non représentatives des parents. De plus, cela permet également de préserver les parties communes des deux parents.

Ce schéma général de croisement est rappelé ci-après :

$$\begin{array}{lll} T_i & \rightarrow_\Delta & D_i \\ & & \oplus \quad \rightarrow \quad D^* \quad \rightarrow_\Lambda \quad T^* \\ T_j & \rightarrow_\Delta & D_j \end{array}$$

6.3 Exemple de croisement pour le problème MP

Nous montrons dans cette section un exemple simple mais concret de croisement `DiBIP` appliqué au problème MP, en procédant aux choix suivants :

Distance δ et opérateur Δ. La mesure de distance δ_T entre deux taxons i et j est définie par la longueur du chemin élémentaire entre les ascendants respectifs de i et de j, moins une unité si le chemin contient la racine de l'arbre T — dans cet exemple nous utilisons des arbres enracinés. Puisque la position de la racine n'a aucun effet sur le score de parcimonie de l'arbre, cet élément ne doit pas affecter la matrice de distances Δ composée de tous les $\delta(i, j)$. Cette distance topologique est la même que celle utilisée dans le cadre du voisinage progressif (chapitre 5), et néglige les arêtes élémentaires de l'arbre (reliées aux taxons). Néanmoins, dans ce cadre-ci et compte tenu des opérateurs $\oplus$ et Λ qui seront

utilisés dans cet exemple, l'utilisation d'une distance topologique simple qui tient compte de ces arêtes élémentaires n'aurait pas d'incidence sur le résultat du croisement.

La figure 6.3 montre la matrice de distances $D = \Delta(T)$ obtenue à partir de l'arbre T donné en exemple. Pour une meilleure compréhension, nous construisons un arbre intermédiaire qui est égal à T privé de ses feuilles et de sa racine, où les nœuds sont étiquetés par la référence aux feuilles supprimées (chaque nœud possède 0, 1 ou 2 références). $\delta_T(i, j)$ représente alors la longueur du chemin entre les nœuds étiquetés par les références i et j.

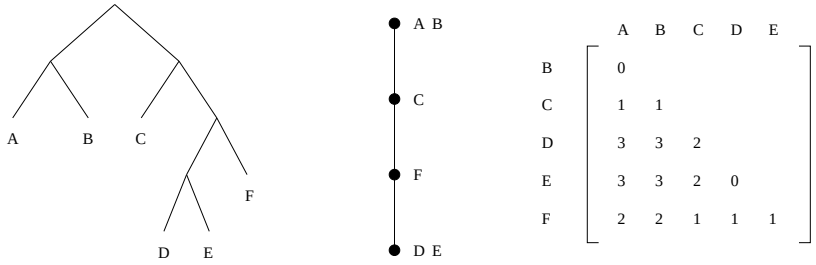


FIG. 6.3 – Distance utilisée

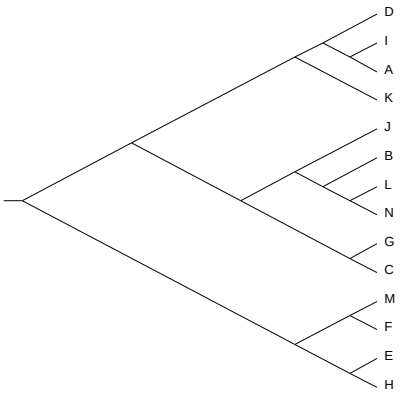
Opérateur $\oplus$ et algorithme de reconstruction Λ. En guise d'opérateur de combinaison de matrices $\oplus$, nous utilisons dans cet exemple l'addition arithmétique, qui est équivalente à la moyenne arithmétique (propriété 5.2). Enfin nous définissons comme méthode de reconstruction Υ une variante non déterministe de `UPGMA` (Algorithme 6.2). Ce choix se justifie par l'objectif du croisement `DiBIP` et la différence fondamentale qui existe entre les algorithmes de classification comme `UPGMA` et les algorithmes de reconstruction phylogénétique plus consistants basés sur les distances. En effet, D^* comporte des valeurs calculées à partir de distances topologiques et non de distances évolutives. De plus, afin de préserver les relations topologiques, l'algorithme utilisé devra en priorité regrouper les taxons les plus proches d'après D^*, et non pas optimiser un critère de consistance basé sur un modèle de l'évolution.

Exemple détaillé de croisement et dénombrement des relations préservées. La figure 6.4 montre par l'exemple cette application du croisement `DiBIP`. Nous remarquons que les proximités éventuelles des couples de taxons chez les deux parents sont conservées chez le fils, bien que celui-ci soit suffisamment éloigné de chacun des deux parents pour assurer la diversification du croisement.

Bien que $D_1 \oplus D_2$ préserve les relations partagées par D_1 et D_2, il n'en va pas de même pour $\Delta(\Lambda(D_1 \oplus D_2))$. Définir un couple d'opérateurs $(\oplus, \Lambda)$ tel que $\Delta(\Lambda(D_1 \oplus D_2))$ vérifie la propriété de préservation des relations semble être un problème difficile, voire impossible dans le cas général. Le but dans le cadre du croisement `DiBIP` est de minimiser le nombre de relations violées tout en utilisant des opérateurs $\oplus$ et Δ polynomiaux.

Afin d'évaluer la qualité du fils par rapport aux deux parents en terme de préservation des relations entre couples de taxons, nous proposons de dénombrer les contraintes violées par la transformation $D^* \rightarrow T^*$ dans l'exemple proposé.

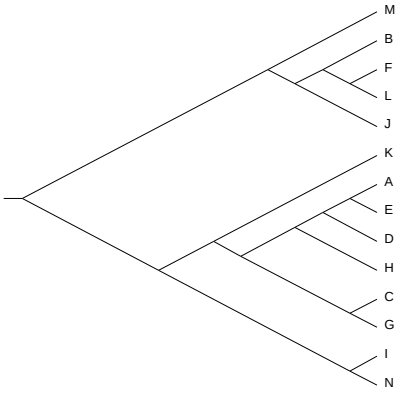
Parent 1 : T_1



$D_1 = \Delta(T_1)$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	6	-	**C**											
C	5	3	-	**D**										
D	1	5	4	-	**E**									
E	5	5	4	4	-	**F**								
F	5	5	4	4	2	-	**G**							
G	5	3	0	4	4	4	-	**H**						
H	5	5	4	4	0	2	4	-	**I**					
I	0	6	5	1	5	5	5	5	-	**J**				
J	5	1	2	4	4	4	2	4	5	-	**K**			
K	2	4	3	1	3	3	3	3	2	3	-	**L**		
L	7	1	4	6	6	6	4	6	7	2	5	-	**M**	
M	5	5	4	4	2	0	4	2	5	4	3	6	-	**N**
N	7	1	4	6	6	6	4	6	7	2	5	0	6	-

Parent 2 : T_2



$D_2 = \Delta(T_2)$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	8	-	**C**											
C	4	6	-	**D**										
D	1	7	3	-	**E**									
E	0	8	4	1	-	**F**								
F	9	1	7	8	9	-	**G**							
G	4	6	0	3	4	7	-	**H**						
H	2	6	2	1	2	7	2	-	**I**					
I	6	4	4	5	6	5	4	4	-	**J**				
J	7	1	5	6	7	2	5	5	3	-	**K**			
K	4	4	2	3	4	5	2	2	2	3	-	**L**		
L	9	1	7	8	9	0	7	7	5	2	5	-	**M**	
M	6	2	4	5	6	3	4	4	2	1	2	3	-	**N**
N	6	4	4	5	6	5	4	4	0	3	2	5	2	-

$D^* = D_1 \oplus D_2$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	14	-	**C**											
C	9	9	-	**D**										
D	2	12	7	-	**E**									
E	5	13	8	5	-	**F**								
F	14	6	11	12	11	-	**G**							
G	9	9	0	7	8	11	-	**H**						
H	7	11	6	5	2	9	6	-	**I**					
I	6	10	9	6	11	10	9	9	-	**J**				
J	12	2	7	10	11	6	7	9	8	-	**K**			
K	6	8	5	4	7	8	5	5	4	6	-	**L**		
L	16	2	11	14	15	6	11	13	12	4	10	-	**M**	
M	11	7	8	9	8	3	8	6	7	5	5	9	-	**N**
N	13	5	8	11	12	11	8	10	7	5	7	5	8	-

Fils : $T^* = \Lambda(D^*)$

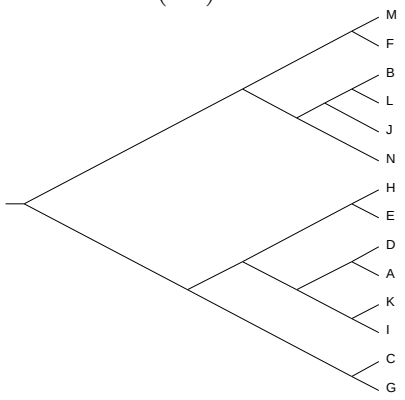


FIG. 6.4 – Exemple d'application du croisement `DiBIP`

Algorithme 6.2 : `UPGMA` non-déterministe

Données : un ensemble de taxons L, une matrice D^* de taille $|L| \times |L|$
Résultat : un arbre phylogénétique $T^* = (V, E)$

début
  $V \leftarrow L \cup \{n_1, \ldots, n_{|L|-1}\}$;
  $E \leftarrow \{\}$;
  $k \leftarrow 0$;
  $D_0^* \leftarrow D^*$;
  tant que $|L| \geqslant 1$ **faire**
    $k \leftarrow k + 1$;
    Choisir aléatoirement a et b distincts tels que $D^*(a, b) = \min_{i,j} D^*(i, j)$;
    $E \leftarrow E \cup \{(a, n_k), (b, n_k)\}$;
    $L \leftarrow (L \setminus \{a, b\}) \cup \{n_k\}$;
    pour tous les $i \in L \setminus \{n_k\}$ **faire**
      pour tous les $j \in L \setminus \{i, n_k\}$ **faire**
        $D_k^*(i, j) = D_{k-1}^*(i, j)$;
      fin
      $D_k^*(i, n_k) = (D_{k-1}^*(a, i) + D_{k-1}^*(b, i))/2$;
    fin
  fin
  Retourner T^* ;
fin

Si T_1 et T_2 avaient été égaux, $\Delta(T_1)$ et $\Delta(T_2)$ auraient partagé 4 095 relations (non triviales) en commun ; dans notre exemple, elles en partagent 1 878. C'est-à-dire qu'il existe 1 878 paires de deux taxons $((a, b), (c, d))$ telles que $a \neq b$, $c \neq d$, $\{a, b\} \neq \{c, d\}$, et symétries mises à part, $\exists \lhd \in \{<, =, >\}, (D_1(a, b) \lhd D_1(c, d)) \wedge (D_2(a, b) \lhd D_2(c, d))$. L'intégralité de ces relations est préservée dans $\Delta(T_1) \oplus \Delta(T_2)$ — par définition de l'opérateur $\oplus$ — et 1 515 paires, soit plus de 80%, sont conservées dans la matrice $\Delta(\Lambda(D_1 \oplus D_2))$ calculée à partir de l'arbre fils de la figure 6.4. Précisons que Λ est ici non-déterministe et qu'une nouvelle construction pourrait générer un résultat légèrement différent.

En approfondissant l'étude, il apparait que les relations d'égalité sont les plus difficiles à conserver, mais sont également les moins fortes sémantiquement, par rapport aux relations d'inégalités fortes. Par exemple, G et H sont à la même distance topologique de K dans chacun des deux parents (3 chez le parent 1, et 2 chez le parent 2). Cette relation d'égalité n'a pu être conservée chez l'enfant, où $\delta_{T^*}(G, K) = 4$ et $\delta_{T^*}(H, K) = 3$, mais seulement approchée. Néanmoins, cette information est moins forte que le grand écart en terme de distance topologique qui existe, chez les deux parents, entre les couples de taxons (E, H) et (A, L) : E et H sont beaucoup plus proches que A et L chez les deux parents (de 7 unités dans les deux cas), et cette relation est préservée chez l'arbre fils. De manière générale, plus la différence entre deux distances topologiques est marquée chez les deux parents, plus la relation est forte, et donc doit se retrouver chez l'enfant.

Nous mesurons le poids d'une relation de la manière suivante :

$$\Xi((a,b),(c,d)) = \min\{|D_1(a,b) - D_1(c,d)|, |D_2(a,b) - D_2(c,d)|\}$$

Cette mesure ne s'applique que si $\exists \lhd \in \{<,=,>\}, (D_1(a,b) \lhd D_1(c,d)) \wedge (D_2(a,b) \lhd D_2(c,d))$. Le tableau 6.2 montre le taux de relations préservées dans l'exemple de la figure 6.4 en fonction de leur poids. Nous remarquons que les relations exprimant des informations très fortes sont systématiquement conservées chez le fils.

Ξ	relations préservées	relations violées	%
0	37	39	49%
1	596	239	71%
2	435	71	86%
3	250	13	95%
4	131	1	99%
5	49	0	100%
6	15	0	100%
7	2	0	100%
Total	1 515	363	81%
Relations <,>	1 478	324	82%

Tab. 6.2 – Relations préservées dans l'exemple

Rappelons que ces relations sont préservées chez le fils tout en assurant une réelle diversité par rapport aux deux parents.

Complexité du croisement. À partir d'un arbre T, $\Delta(T)$ se calcule en $O(n^2 \log_2(n))$, où n est le nombre de feuilles de l'arbre. Ce calcul n'est effectué qu'une seule fois pour chaque arbre ; les matrices sont stockées car les arbres peuvent servir pour plusieurs croisements. L'addition de matrices ainsi que l'algorithme `UPGMA` ont une complexité de $O(n^2)$. Par conséquent, cette instanciation de l'opérateur de croisement `DiBIP` a une complexité temporelle totale de $O(n^2 \log_2(n))$.

6.4 Un algorithme mémétique pour le problème MP

Nous avons utilisé le croisement d'arbres `DiBIP` au sein d'un algorithme mémétique. Celui-ci, nommé `HYDRA` pour *HYbrid Distance Recombination Algorithm* est un algorithme de recherche locale génétique hybride utilisant le croisement `DiBIP` et l'opérateur de recherche locale à voisinage progressif `DPN` présenté au chapitre 5.

L'algorithme `HYDRA` (Algorithme 6.3) débute par la génération aléatoire d'une population initiale où chaque individu est un arbre phylogénétique (*GenererPopulation*). L'algorithme entre alors dans un processus itératif. À chaque pas, deux individus (parents) de la population sont sélectionnés (*ChoisirParents*) et recombinés (croisement `DiBIP`) afin d'obtenir un nouvel individu (fils). La recherche locale `DPN` est appliquée pour améliorer

le fils durant l itérations. Sous certaines conditions d'insertion, l'arbre résultant est ajouté à la population où il remplace alors un individu existant (*Remplacer*). Ce processus est répété jusqu'à ce que la condition d'arrêt soit vérifiée, usuellement lorsqu'un nombre maximum d'itérations max_{it} est atteint ou bien lorsque le temps d'exécution excède une durée maximale max_{tps}.

Algorithme 6.3 : Algorithme mémétique `HYDRA`

Données : une matrice de caractères $\mathcal{A}$, la taille de la population N, le nombre d'itérations de recherche locale l
Résultat : l'arbre le plus parcimonieux trouvé

```
1  début
2      P ← GenererPopulation(N) ;
3      tant que la condition d'arrêt n'est pas satisfaite faire
4          (T1, T2) ← ChoisirParents(P) ;
5          T ← DiBIP(T1, T2) ;
6          T ← DPN(T, l) ;
7          P ← Remplacer(P, T) ;
8      fin
9      Retourner le meilleur arbre trouvé ;
10 fin
```

La fonction *ChoisirParents* pratique une stratégie de sélection par tournoi [Miller and Goldberg, 1995]. Deux groupes de 20% des individus sont constitués aléatoirement. Les deux solutions représentant les meilleurs individus de chaque groupe sont sélectionnés pour le croisement. La fonction *Remplacer* remplace un des individus de la solution par l'arbre fils T obtenu après le processus de croisement et de recherche locale si et seulement si T n'est pas déjà présent dans la population. Dans ce cas, l'individu supprimé de P est le plus âgé de la population (celui qui est présent depuis le plus longtemps) si son âge en terme d'itérations dépasse une limite préfixée, ou le plus proche dans le cas contraire. Nous mesurons la proximité de deux solutions par la variance des valeurs de leurs matrices de distances associées. Cet algorithme mémétique n'est pas élitiste ; c'est-à-dire que l'individu remplacé peut être le meilleur de la population. Dans ce cas, il est sauvegardé jusqu'à ce qu'un meilleur individu soit trouvé, afin de pouvoir retourner la meilleure solution rencontrée durant le processus.

6.5 Résultats expérimentaux

Les instances aléatoires avec données manquantes sont les plus difficiles à résoudre par les algorithmes de recherche locale. Par conséquent, cela laisse entrevoir une possibilité d'améliorer pour ces instances les meilleurs résultats reportés dans la littérature.

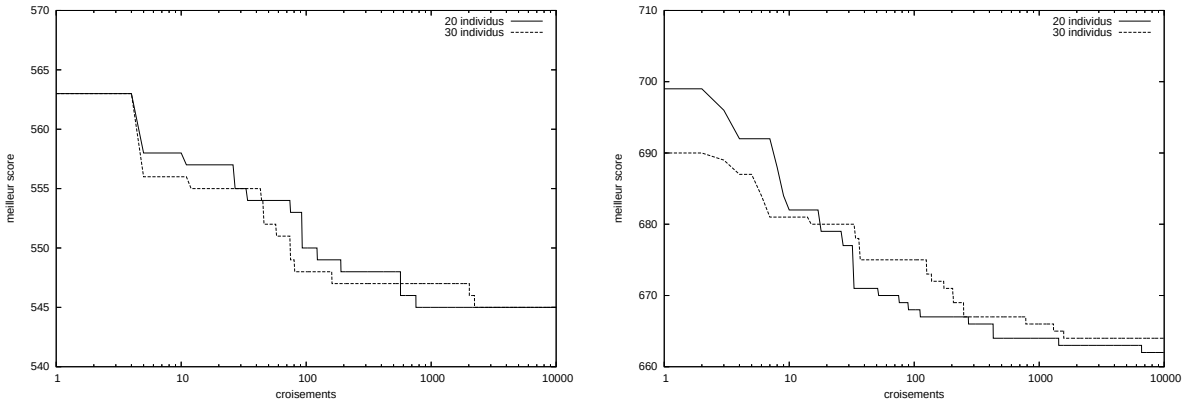


FIG. 6.5 – Influence de la taille de la population

6.5.1 Paramétrage

La première partie des expérimentations de l'algorithme `HYDRA` s'attache à la définition de paramètres adéquats. Afin d'optimiser l'algorithme, nous avons envisagé différents paramétrages :

- une taille de population de 20 ou 30 individus ;
- l'effort de recherche `DPN` (10 000, 50 000 ou 100 000 itérations) ;
- deux métriques utilisées pour la conversion `Arbre` $\rightarrow$ `Matrice` (δ^1 est la distance topologique entre deux nœuds définie dans la section 5.2.2 du chapitre 5, et δ^2 est la longueur du chemin entre deux nœuds, en prenant en compte les distances de parcimonie entre chaque couple de nœuds adjacents) ;
- un opérateur de croisement matriciel $\oplus$ de type $\alpha.\min + (1-\alpha).\max$ avec différentes valeurs possibles de α.

Nous avons fixé empiriquement les valeurs de ces paramètres de telle manière à ce que l'algorithme soit performant et robuste en des temps de calculs raisonnables (de l'ordre d'une dizaine de minutes) : $|P| = 20$, $l = 50\ 000$, δ^2 et $\alpha = 0,5$.

Les figures 6.5 à 6.8 illustrent les effets de chaque modification paramétrique, par rapport à notre paramétrage par défaut. Chaque courbe reportée symbolise l'évolution de la recherche ayant retourné, parmi cinq exécutions effectuées selon un paramétrage identique, la solution de cout médian[3]. L'axe temporel, échelonné logarithmiquement, est placé en abscisse ; l'unité choisie est le nombre d'itérations de recherche locale (ou bien le nombre de croisements si chaque croisement est suivi d'une recherche locale de même intensité). Ce traitement a été effectué sur les instances `tst01` (figures de gauche) et `tst20` (figures de droite). Il s'agit respectivement de la plus petite et de la plus grande des instances aléatoires avec données manquantes.

[3]Le score de la meilleure solution constitue le premier critère à optimiser, le second étant le temps de calcul employé pour trouver cette solution.

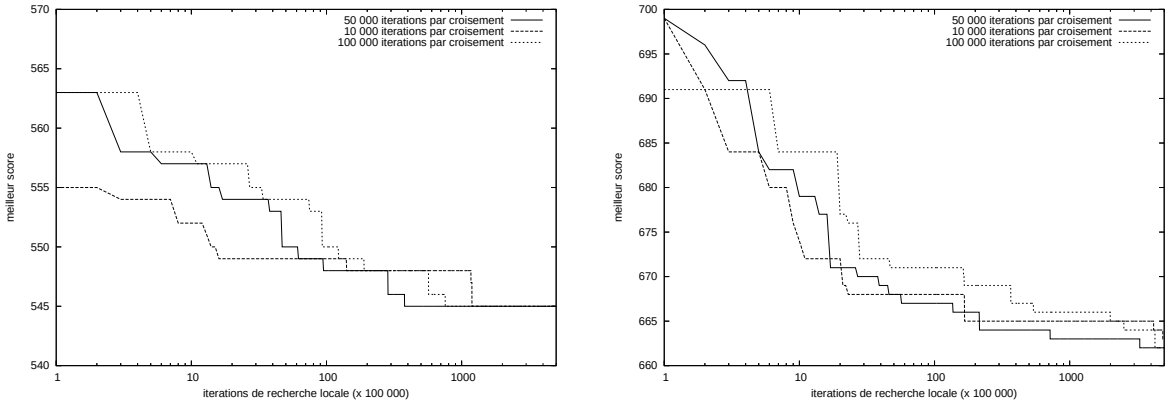


FIG. 6.6 – Influence de la longueur de la recherche locale

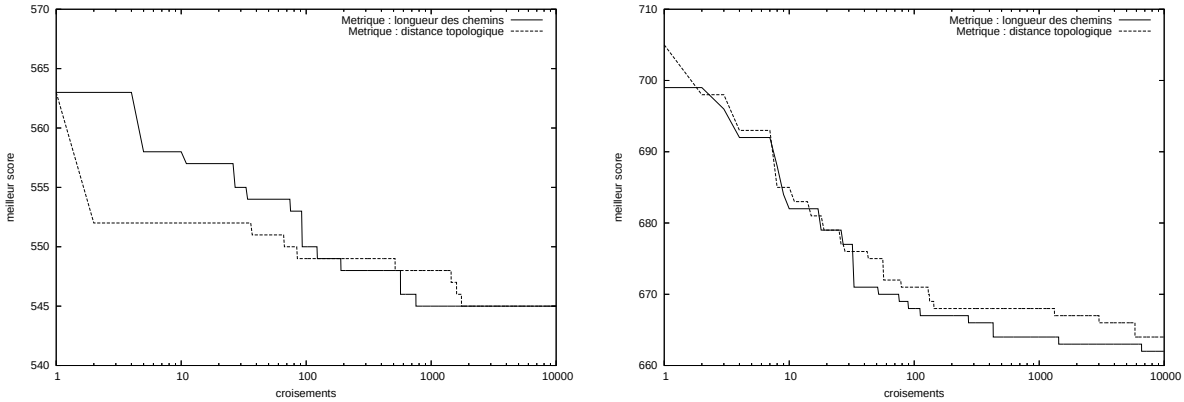


FIG. 6.7 – Influence de la définition de la métrique δ dans le croisement

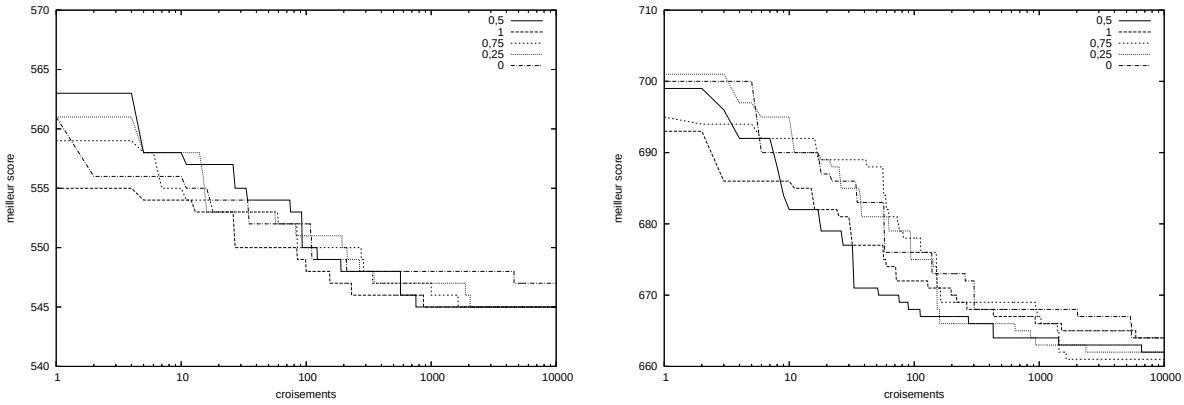


FIG. 6.8 – Influence du paramètre α de l'opérateur $\oplus$ dans le croisement

Dans tous les cas et avec chaque combinaison de paramètres, HYDRA retourne les meilleurs scores connus sur ces deux instances.

6.5.2 Résultats comparatifs par rapport aux logiciels existants

Dans cette section, nous comparons l'algorithme HYDRA utilisant les paramètres par défaut avec deux algorithmes de référence :

– l'algorithme génétique AG+PR [Ribeiro and Vianna, 2003], et
– le logiciel TNT[4] [Goloboff *et al.*, 2000].

Les comparaisons ont été effectuées sur l'ensemble des instances binaires, puisque l'algorithme AG+PR n'est pas disponible et que ses résultats sur ces instances sont communiqués. Dans les tableaux 6.3 à 6.5, nous reportons les résultats mentionnés par les auteurs, à savoir pour chaque instance le meilleur et la moyenne des scores retournés par dix exécutions d'une durée de 1 000 secondes de cet algorithme (les écarts-types ne sont pas communiqués). Nous avons réalisé autant d'exécutions avec l'algorithme HYDRA et le logiciel TNT (en utilisant l'ensemble des techniques) sur les mêmes instances. Le temps alloué par HYDRA est de 300 secondes (cinq minutes) sur les instances réelles, et 1 000 secondes sur les instances aléatoires, plus difficiles. TNT ne requiert pas de spécification de durée et termine de lui-même ; exécuté sur un AMD Athlon 64 X2 3800, ses temps d'exécution ne dépassent pas 5 secondes.

L'algorithme HYDRA, implémenté en C++ puis compilé avec gcc utilisant l'option d'optimisation -O3, est exécuté séquentiellement sur un cluster de dix nœuds, chacun étant composé d'un processeur Xeon cadencé à 2 GHz et disposant de 1 Go de mémoire vive. Le processeur utilisé dans [Ribeiro and Vianna, 2003] est un Pentium IV cadencé à 2 GHz. Les machines étant différentes, la comparaison des temps de calcul doit être effectuée avec précaution.

Sur les instances réelles (tableau 6.3), les trois logiciels obtiennent des résultats semblables, même si seul HYDRA parvient à retourner le meilleur résultat pour chaque exécution, ce qui démontre son excellente robustesse. En revanche, il existe une grande disparité au niveau des temps de calculs. TNT est le plus rapide, chaque exécution étant réalisée en quelques secondes. Quelques secondes suffisent également à la recherche locale DPN pour certaines de ces instances (voir chapitre 5, section 5.5), il en va donc de même pour l'algorithme HYDRA. Puisque le temps de calcul alloué est un paramètre de notre algorithme, nous avons spécifié une durée de cinq minutes afin de maximiser l'effort de recherche.

Nous constatons que les instances réelles de taille raisonnable disponibles dans la littérature (y compris l'instance zilla) n'offrent pas de réelle challenge, tant les meilleurs résultats connus peuvent être égalés, au prix d'un effort de recherche plus ou moins important — les motivations actuelles concernent alors davantage les temps de calcul que les solutions elles-mêmes —, mais peut-être pas améliorés. Ces instances sont structurées et

[4] TNT (*Tree analysis using New Technology*, présenté brièvement au chapitre 3) est connu pour trouver des arbres parcimonieux plusieurs milliers de fois plus rapidement que les autres logiciels. TNT utilise plusieurs stratégies, également basées sur la recherche locale [Goloboff, 1999], et parvient à évaluer plusieurs millions de solutions par seconde grâce à des optimisations dans le calcul des scores des voisins SPR et TBR.

Instance	n	m	Logiciel	ϕ^*	$\phi^\sim$	σ
GRIS	47	93	HYDRA	172	172,0	0
			AG+PR	172	172,0	0
			TNT	172	172,0	0
ANGI	49	59	HYDRA	216	216,0	0
			AG+PR	216	216,0	0
			TNT	216	216,0	0
TENU	56	179	HYDRA	682	682,0	0
			AG+PR	682	682,0	0
			TNT	682	682,0	0
ETHE	58	86	HYDRA	372	372,0	0
			AG+PR	372	372,4	nc
			TNT	372	372,2	0,4
ROPA	75	82	HYDRA	325	325,0	0
			AG+PR	325	325,8	nc
			TNT	325	325,0	0
GOLO	77	97	HYDRA	496	496,0	0
			AG+PR	496	496,2	nc
			TNT	496	496,3	0,7
SCHU	113	146	HYDRA	759	759,0	0
			AG+PR	759	759,0	0
			TNT	759	759,0	0
CARP	117	110	HYDRA	548	548,0	0
			AG+PR	548	548,6	nc
			TNT	548	548,0	0

Tab. 6.3 – Performances de l'algorithme HYDRA sur des instances réelles

leur difficulté est vraisemblablement moins importante que ne le laisse supposer la taille de leurs espaces de recherche associés.

En revanche, les instances aléatoires avec données manquantes sont difficiles : avec elles, la robustesse des meilleurs logiciels est mise à mal. Les tableaux 6.4 et 6.5 indiquent pour chaque instance, outre le nombre n de séquences (de longueur m), le pourcentage de caractères indéfinis (ind.). Nous avons effectué, pour chacune de ces vingt instances, dix exécutions de l'algorithme `HYDRA`, chacune d'une durée de vingt minutes. Sur ces instances difficiles, `HYDRA` est réellement efficace, puisqu'il permet d'améliorer significativement les meilleurs résultats connus (de près de 10 unités en moyenne). De plus, il est très robuste, y compris sur les instances possédant un fort pourcentage de caractères indéfinis.

Le chapitre précédent montrait à quel point l'algorithme `DPN` peut atteindre de bons résultats sur des instances traditionnelles. Hybridé à un algorithme génétique utilisant le croisement d'arbres `DiBIP`, nous pouvons constater que les résultats sont extrêmement compétitifs, même sur des instances purement aléatoires sortant du cadre d'application habituel du problème MP et difficiles à résoudre par les algorithmes existants. Ce constat nous laisse penser que ce mécanisme, adapté à d'autres problèmes combinatoires, pourrait obtenir des résultats prometteurs.

6.6 Conclusion

Nous avons introduit dans ce chapitre un nouveau mécanisme de croisement d'arbres, qui consiste à considérer des informations sémantiques extraites de chacun des parents pour reconstruire indépendamment un enfant. L'idée est alors d'utiliser une matrice de distances pour caractériser chaque arbre. En conséquence, deux arbres peuvent être aisément combinés par une opération sur les deux matrices de distances. Contrairement aux croisements d'arbres existants, ce schéma de croisement offre une manière simple et naturelle pour assurer une combinaison et une transmission d'information globale durant le processus de croisement.

Son application au problème MP par le biais d'un algorithme mémétique utilisant la descente à voisinage progressif permet de résoudre des instances structurées ou non de manière efficace et fiable. Toutes les expérimentations ont montré une puissance de recherche nettement supérieure aux algorithmes et programmes existants.

Instance	n	m	ind.	Logiciel	ϕ^*	$\phi^\sim$	σ
tst01	45	61	20%	HYDRA	**545**	545,4	0,7
				AG+PR	549	549,6	nc
				TNT	547	549,2	1,3
tst02	47	151	30%	HYDRA	**1 354**	1 356,1	1,6
				AG+PR	1 358	1 363,6	nc
				TNT	1 361	1 365,5	2,1
tst03	49	111	40%	HYDRA	**833**	833,9	0,6
				AG+PR	838	840,6	nc
				TNT	840	842,6	1,9
tst04	50	97	50%	HYDRA	**588**	589,4	0,7
				AG+PR	592	595,0	nc
				TNT	595	598,0	2,4
tst05	52	75	20%	HYDRA	**789**	789,0	0
				AG+PR	790	794,0	nc
				TNT	**789**	794,1	2,9
tst06	54	65	30%	HYDRA	**596**	597,3	0,7
				AG+PR	603	605,4	nc
				TNT	601	602,6	1,3
tst07	56	143	40%	HYDRA	**1 269**	1 270,7	0,7
				AG+PR	1 276	1 280,6	nc
				TNT	1 272	1 283,3	7,5
tst08	57	119	50%	HYDRA	**852**	854,1	2,2
				AG+PR	863	867,4	nc
				TNT	866	868,8	2,7
tst09	59	93	20%	HYDRA	**1 144**	1 145,2	1,2
				AG+PR	1 150	1 154,2	nc
				TNT	1 146	1 151,6	2,9
tst10	60	71	30%	HYDRA	**721**	721,3	0,5
				AG+PR	725	728,6	nc
				TNT	721	726,7	3,1

Tab. 6.4 – Performances de l'algorithme HYDRA sur des instances aléatoires (première partie)

Instance	n	m	ind.	Logiciel	ϕ^*	$\phi^\sim$	σ
tst11	62	63	40%	HYDRA	**542**	542,6	0,5
				AG+PR	544	546,8	nc
				TNT	547	550,2	2,4
tst12	64	147	50%	HYDRA	**1 211**	1 213,6	2,3
				AG+PR	1 229	1 233,0	nc
				TNT	1 228	1 233,3	3,3
tst13	65	113	20%	HYDRA	**1 515**	1 518,5	2,6
				AG+PR	1 526	1 530,6	nc
				TNT	1 522	1 528	3,2
tst14	67	99	30%	HYDRA	**1 160**	1 161,9	1,2
				AG+PR	1 174	1 177,4	nc
				TNT	1 163	1 173,9	4,7
tst15	69	77	40%	HYDRA	**752**	754,5	2,2
				AG+PR	765	766,4	nc
				TNT	760	765,2	2,6
tst16	70	69	50%	HYDRA	**529**	530,8	1,5
				AG+PR	545	547,6	nc
				TNT	540	544,4	3,0
tst17	71	159	20%	HYDRA	**2 453**	2 455,2	2,5
				AG+PR	2 468	2 470,8	nc
				TNT	2 468	2 471,5	2,0
tst18	73	117	30%	HYDRA	**1 522**	1 523,7	3,1
				AG+PR	1 542	1 548,2	nc
				TNT	1 539	1 544,4	4,6
tst19	74	95	40%	HYDRA	**1 013**	1 016,9	2,8
				AG+PR	1 028	1 033,0	nc
				TNT	1 025	1 032,7	5,5
tst20	75	79	50%	HYDRA	**661**	663,9	1,4
				AG+PR	676	678,8	nc
				TNT	675	678,9	2,9

Tab. 6.5 – Performances de l'algorithme HYDRA sur des instances aléatoires (seconde partie)

Conclusion générale

Principales contributions

Les travaux réalisés durant cette thèse s'articulent autour de la résolution du problème Maximum de Parcimonie. Dans la première partie, nous avons présenté le problème MP et recensé les différentes approches utilisées pour sa résolution. Il s'agit à notre connaissance de la première revue détaillée des différentes techniques de résolution de ce problème. Par la suite, nous avons mené une étude empirique concernant les voisinages d'arbres et leur influence sur la capacité des algorithmes de recherche locale à trouver rapidement une *bonne* solution. Ces expérimentations préliminaires ont indiqué qu'une descente utilisant le voisinage SPR — qui consiste à détacher un sous-arbre pour le reconnecter sur une branche de l'arbre amputé — est efficace dans le sens où elle n'échoue pas sur des optimums locaux de mauvaise qualité, mais demeure lente en fin de parcours étant donnés la taille du voisinage et la faible proportion de voisins pertinents.

De ce constat, nous avons défini un **voisinage progressif** qui se restreint en fonction de la distribution de plus en plus faible des voisins pertinents. Une étude sur la distribution des voisins améliorants, montrant notamment que seuls les mouvements SPR qui ne modifient que localement la topologie des arbres de faible cout permet de les améliorer, a justifié l'utilisation d'un tel voisinage.

Afin de minimiser le nombre d'évaluations de voisins non pertinents durant la recherche, nous avons défini un voisinage SPR paramétrique, où la distance topologique δ entre le point de coupe et le point d'insertion du sous-arbre détaché est contrainte. Ce paramètre, qui est la distance δ maximale autorisée, se réduit au fur et à mesure de la recherche, établissant un schéma de voisinage progressif. D'après nos expérimentations, la descente à voisinage progressif est robuste et permet de trouver sur des benchmarks structurés de bonnes solutions approchées en des temps de calcul beaucoup plus faibles qu'en utilisant les voisinages traditionnels.

Dans un second temps, afin d'améliorer l'efficacité de la recherche, nous avons conçu un algorithme mémétique qui utilise un **croisement d'arbres spécifique**. Les croisements d'arbres présentés dans la littérature utilisent tous un principe similaire, qui consiste à greffer à l'un des parents un sous-arbre de l'autre parent, puis à réparer l'arbre résultant afin qu'il constitue une configuration valide du problème. La conséquence directe est une perte de la moitié des informations topologiques contenues dans les deux parents. Au contraire, le croisement `DiBIP`, présenté dans cette thèse, extrait les informations topologiques de chacun des parents dans deux matrices de distances, ensuite combinées en une matrice fille via un opérateur préservant les informations sémantiques partagées par les deux parents. Celles-ci sont véhiculées par les positions relatives dans l'arbre des données du problème (taxons), placées aux feuilles et par définition directement liées au cout de

l'arbre. L'arbre fils, préservant l'essentiel des informations sémantiques partagées par les deux parents, est entièrement reconstruit à partir de la matrice fille par un algorithme de classification (UPGMA). L'algorithme mémétique `HYDRA`, conçu autour de ce croisement d'arbres et de la descente à voisinage progressif, obtient sur des benchmarks difficiles issus de la littérature de meilleurs résultats que les logiciels existants, aussi bien en terme de qualité des solutions que de robustesse.

Que ce soit dans les phases d'intensification (recherche locale) que de diversification (croisement), parcourir l'espace de recherche en prenant en compte les informations sémantiques des configurations courantes permet une nette économie de mouvements et un parcours maitrisé de l'espace de recherche. En appréhendant le problème MP de cette manière, nous avons pu diminuer le nombre d'itérations de recherche locale, et atteindre pour les instances les plus difficiles de nouvelles solutions minimales.

Bien sûr, le travail réalisé comporte quelques limites. La principale est que le logiciel `TNT` [Goloboff *et al.*, 2000] reste la référence en terme de rapidité. En conséquence, il parvient à faire la différence sur des instances structurées, où notre algorithme trouve généralement des solutions équivalentes mais en des temps de calcul plus importants. Même si le voisinage progressif que nous avons conçu permet de passer en revue beaucoup moins de solutions pour un même résultat, cela n'est pas suffisant pour rivaliser avec les optimisations d'implémentation de `TNT`.

Perspectives de recherche

L'état actuel de nos travaux laisse entrevoir de nombreuses perspectives de recherche.

La première, qui est de définir un voisinage réactif, ou *intelligent*, est entrevue à la fin du chapitre 5. Il s'agit d'une descente qui collecte des informations sur la qualité des voisins évalués durant la recherche, afin d'étendre ou de restreindre son voisinage. Parallèlement, considérant l'extrême rapidité du logiciel `TNT` malgré la grande quantité de voisins évalués et la taille des voisinages utilisés, il serait très intéressant de combiner certaines techniques de ce logiciel à ceux de la descente à voisinage progressif. Cependant, introduire l'optimisation du calcul des couts des arbres [Goloboff, 1993], qui apporte un bénéfice proportionnel au nombre de voisins évalués, ne semble pas compatible avec l'idée du voisinage progressif — qui est de réduire le nombre d'arbres à évaluer.

Nous pensons également que l'algorithme mémétique `HYDRA` peut être plus performant que lors de nos expérimentations, en réalisant un paramétrage plus fin. De la même manière, d'autres opérateurs peuvent être envisagés dans le cadre de définition du croisement `DiBIP`, et ce pour chacune des trois étapes du croisement. Un travail plus en profondeur pourrait également être réalisé afin de mesurer les distances (par exemple distances de Robinson-Foulds [Robinson and Foulds, 1981]) entre les arbres issus de différents croisements et leurs parents ; ceci dans le but de justifier davantage de l'utilisation de tel ou tel croisement. Une formalisation du concept de croisement basé sur la sémantique des parents est également à l'étude.

Enfin, le voisinage progressif et le croisement décrits dans cette thèse ont été définis de manière générale et peuvent alors constituer un cadre de travail pour d'autres problèmes combinatoires. Nous prévoyons ainsi d'appliquer une telle approche à des problèmes académiques comme le problème de l'arbre de Steiner dans un graphe, du voyageur de commerce ou du coloriage de graphes. Nous pensons en effet que ces problèmes ont de nombreuses analogies avec le problème MP.

Références bibliographiques

[Aldrich, 1997] cité p. 22
John Aldrich. R. A. Fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12:162–176, 1997.

[Allen and Steel, 2001] cité p. 52, 52, 52
Benjamin L. Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.

[Andreatta and Ribeiro, 2002] cité p. 49, 52
Alexandre Andreatta and Celso C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002.

[Aristote, IVe s av J C] cité p. 9
Aristote. *Des Parties des Animaux*. IVe s. av. J.-C.

[Barker, 2004] cité p. 57, 79
Daniel Barker. LVB: parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20(2):274–275, 2004.

[Bastert *et al.*, 2002] cité p. 52
Oliver Bastert, Dan Rockmore, Peter F. Stadler, and Gottfried Tinhofer. Landscapes on spaces of trees. *Applied Mathematics and Computation*, 131:439–459, 2002.

[Baum, 1992] cité p. 59
Bernard R. Baum. Combining trees as a way of combining data for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992.

[Berge, 1958] cité p. 11
Claude Berge. *Theorie des Graphes et ses Applications*. Dunod (Paris), 1958.

[Bininda-Emonds, 1998] cité p. 59
Olaf R. P. Bininda-Emonds. Properties of matrix representation with parsimony analyses. *Systematic Biology*, 47(3):497–508, 1998.

[Bininda-Emonds, 2002] cité p. 59
Olaf R. P. Bininda-Emonds. The (super)tree of life: procedures, problems, and prospects. *Annual Review of Ecology and Systematics*, 33:265–289, 2002.

[Bininda-Emonds, 2004] cité p. 59
Olaf R. P. Bininda-Emonds. The evolution of supertrees. *Trends in Ecology and Evolution*, 19(6):315–322, 2004.

[Bonomi and Lutton, 1984] cité p. 57
Ernesto Bonomi and Jean-Luc Lutton. The N-city travelling salesman problem: statistical mechanics and the metropolis algorithm. *Society for Industrial and Applied Mathematics Review*, 26(4):551–568, 1984.

[Bordewich and Semple, 2004] cité p. 52
Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8:409–423, 2004.

[Bryant, 2004] cité p. 52
David Bryant. The splits in the neighborhood of a tree. *Annals of Combinatorics*, 8:1–11, 2004.

[Bush *et al.*, 1999] cité p. 1
Robin M. Bush, Catherine A. Bender, Kanta Subbarao, Nancy J. Cox, and Walter M. Fitch. Predicting the evolution of human influenza a. *Science*, 286:1921–1925, 1999.

[Camin and Sokal, 1965] cité p. 41
Joseph H. Camin and Robert R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1965.

[Carrington and Hoelzel, 2001] cité p. 1
Mary Carrington and A. Rus Hoelzel. *Molecular Epidemiology*. Oxford University Press, 2001.

[Cavalli-Sforza and Edwards, 1967] cité p. 15, 20
Luca L. Cavalli-Sforza and Anthony W. F. Edwards. Phylogenetic analysis models and estimation procedures. *American Journal of Human Genetics*, 19:233–257, 1967.

[Cavender, 1978] cité p. 23
James A. Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40:271–280, 1978.

[Cayley, 1857] cité p. 14, 31
Arthur Cayley. On the theory of the analytical forms called trees. *Philosophy Magazine*, 13:172–176, 1857.

[Černý, 1985] cité p. 56
Vladimír Černý. A thermodynamic approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[Charon and Hudry, 1993] cité p. 55
Irène Charon and Olivier Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14:133–137, 1993.

[Charon and Hudry, 2002] cité p. 55
Irène Charon and Olivier Hudry. The noising methods: a survey. In P. Hansen and C. C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*, pages 245–261. Kluwer, 2002.

[Chase *et al.*, 1993] cité p. 2, 59, 69
M. W. Chase, D. E. Soltis, R. G. Olmstead, D. Morgan, R. H. Les, B. D. Mishler, M. R. Duvall, R. A. Price, H. G. Hills, Y.-L. Qui, K. A. Kron, J. H. Rettig, E. Conti, J. D. Palmer, J. R. Manhart, K. J. Sytsma, H. J. Michaels, W. J. Kress, K. G. Karol, W. D. Clark, M. Hedrén, B. S. Gaut, R. K. Jansen, K.-J. Kim, C. F. Wimpee, J. F. Smith, G. R. Furnier, S. H. Strauss, Q.-Y. Xiang, G. M. Plunkett, P. S. Soltis, S. M. Swensen, S. E. Williams, P. A. Gadek, C. J. Quinn, L. E. Eguiarte, E. Golenberg, G. H. Learn, S. W. Graham, S. C. H. Barrett, S. Dayanandan, and V. A. Albert. Phylogenetics of seed plants: an analysis of nucleotide sequences from the plastid gene *rbc*L. *Annals of the Missouri Botanical Gardens*, 80:528–580, 1993.

[Collins *et al.*, 2003] cité p. 1
Francis S. Collins, Michael Morgan, and Aristides Patrinos. The human genome project: Lessons from large-scale biology. *Science*, 300:286–290, 2003.

[Congdon and Septor, 2003] cité p. 3, 58
Clare B. Congdon and Kevin J. Septor. Phylogenetic trees using evolutionary search: Initial progress in extending gaphyl to work with genetic data. In *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 320–326. IEEE press, 2003.

[Congdon, 2002] cité p. 58
Clare B. Congdon. Gaphyl: An evolutionary algorithms approach for the study of natural evolution. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1057–1064, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Cotta and Moscato, 2002] cité p. 58
Carlos Cotta and Pablo Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villaca nas, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 720–729. Springer-Verlag, Berlin, 2002.

[Cuvier, 1812] cité p. 10
Georges Cuvier. *Recherches sur les ossements fossiles de quadrupèdes*. Déterville Paris, 1812.

[Darlu and Tassy, 1993] cité p. 20
Pierre Darlu and Pascal Tassy. *Reconstruction phylogénétique : concepts et méthodes*. Masson (Paris), 1993.

[Darwin, 1859] cité p. 10, 11
Charles Darwin. *On the Origin of Species*. Harvard University Press (Cambridge), 1859.

[Darwin, 1871] cité p. 11
Charles Darwin. *The Descent of Man, and Selection in Relation to Sex*. John Murray (London), 1871.

[Day *et al.*, 1986] cité p. 37
William H. E. Day, David S. Johnson, and David Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81:33–42, 1986.

[Dayhoff *et al.*, 1978] cité p. 23, 41
Margaret O. Dayhoff, Robert M. Schwartz, and Bruce C. Orcutt. A model of evolutionary change in proteins. In M. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. National Biomedical Research Foundation, Silver Spring, MD, 1978. Supplement 3.

[de Candolle, 1813] cité p. 9
Augustin Pyrame de Candolle. *Théorie élémentaire de la botanique*. 1813.

[de Lamarck, 1809] cité p. 10
Jean-Baptiste de Lamarck. *Philosophie zoologique*. Sady (ed. 1873), 1809.

[de Maupertuis, 1745] cité p. 9
Pierre-Louis Moreau de Maupertuis. *Vénus Physique*. 1745.

[Dorigo and Caro, 1999] cité p. 57
Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[Dorne and Hao, 1998] cité p. 58
R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 745–754, 1998.

[Du *et al.*, 2005] cité p. 2, 60
Zhihua Du, Feng Lin, and Usman W. Roshan. Reconstruction of large phylogenetic trees: a parallel approach. *Computational biology and chemistry*, 29(4):273–280, 2005.

[Dupuis, 1988] cité p. 13
Claude Dupuis. Le taxinomiste face aux catégories. *Cahiers de Naturalistes*, 44:49–109, 1988.

[Edwards and Cavalli-Sforza, 1964] cité p. 22
Anthony W. F. Edwards and Luca L. Cavalli-Sforza. Reconstruction of evolutionary trees. *Phenetic and Phylogenetic Classification*, pages 67–76, 1964.

[Efron, 1979] cité p. 25
Bradley Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7:1–26, 1979.

[Essam and Fisher, 1970] cité p. 11
John W. Essam and Michael E. Fisher. Some basic definitions in graph theory. *Reviews of Modern Physics*, 42(2):271–288, 1970.

[Falkenauer, 1996] cité p. 58
Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[Farris, 1970] cité p. 13, 41
James S. Farris. Methods for computing Wagner trees. *Systematic Zoology*, 19:83–92, 1970.

[Farris, 1973] cité p. 23
James S. Farris. A probability model for inferring evolutionary trees. *Systematic Zoology*, 22:250–256, 1973.

[Felsenstein, 1981] cité p. 22, 23
Joseph Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[Felsenstein, 1985] cité p. 25
Joseph Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[Felsenstein, 1989] cité p. 60
Joseph Felsenstein. PHYLIP – Phylogeny Inference Package. *Cladistics*, 5:164–166, 1989.

[Felsenstein, 2003] cité p. 1
Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2003.

[Feo and Resende, 1995] cité p. 57, 95
Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[Fisher, 1912] cité p. 22
Ronald Aylmer Fisher. On an absolute criterion for fitting frequency curves. *Messenger of Mathematics*, 41:155–160, 1912.

[Fisher, 1921] cité p. 22
Ronald Aylmer Fisher. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, 1:3–32, 1921.

[Fisher, 1922] cité p. 22
Ronald Aylmer Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Series A*, 222:309–368, 1922.

[Fitch, 1971] cité p. 29, 41
Walter M. Fitch. Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971.

[Foulds and Graham, 1982] cité p. 2, 37, 66
Leslie R. Foulds and Ron L. Graham. The Steiner tree problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[Freisleben and Merz, 1996] cité p. 58
Bernd Freisleben and Peter Merz. New genetic local search operators for the traveling salesman problem. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, volume 1141 of *Lecture Notes in Computer Science*, pages 890–900, Berlin, 1996. Springer.

[Galinier and Hao, 1999] cité p. 58
Phillippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Ganapathy *et al.*, 2003] cité p. 52, 52
Ganeshkumar Ganapathy, Vijaya Ramachandran, and Tandy Warnow. Better hill-climbing searches for parsimony. In *Proceedings of the Third International Workshop on Algorithms in Bioinformatics (WABI)*, pages 245–258, 2003.

[Ganapathy *et al.*, 2004] cité p. 50, 52
Ganeshkumar Ganapathy, Vijaya Ramachandran, and Tandy Warnow. On contract-and-refine transformations between phylogenetic trees. In *Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 900–909, 2004.

[Garey and Johnson, 1979] cité p. 46
Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[Gascuel, 1997] cité p. 20
Olivier Gascuel. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14:685–695, 1997.

[Gladstein, 1997] cité p. 52
David S. Gladstein. Efficient incremental character optimization. *Cladistics*, 13:21–26, 1997.

[Goëffon *et al.*, 2005a] cité p. 57, 65, 67, 79
Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. Local search for the maximum parsimony problem. In *Advances in Natural Computation, First International Conference*, volume 3612 of *Lecture Notes in Computer Science*, pages 678–683. Springer, 2005.

[Goëffon *et al.*, 2005b] cité p. 3, 54, 69, 81
Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. Progressive tree neighborhood applied to the maximum parsimony problem. In *Proceedings of the Mini EURO Conference on Variable Neighborhood Search*, 2005.

[Goëffon *et al.*, 2005c] cité p. 69
Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. Recherche locale à voisinage evolutif pour la reconstruction de phylogénies. In *Actes du Sixième Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, pages 187–188, 2005.

[Goëffon *et al.*, 2006] cité p. 3, 59, 101
Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao. A distance-based information preservation tree crossover for the maximum parsimony problem. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*, volume 4193 of *Lecture Notes in Computer Science*, pages 761–770. Springer, 2006.

[Goldberg, 1989] cité p. 57
David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[Goloboff *et al.*, 2000] cité p. 2, 60, 116, 122
Pablo A. Goloboff, Steve Farris, and Kevin C. Nixon. TNT (Tree analysis using New Technology) (BETA), 2000.

[Goloboff, 1993] cité p. 52, 122
Pablo A. Goloboff. Character optimization and calculation of tree lengths. *Cladistics*, 9:433–436, 1993.

[Goloboff, 1997] cité p. 60
Pablo A. Goloboff. NONA, version 1.5 (32 bit), 1997.

[Goloboff, 1999] cité p. 60, 69, 116
Pablo A. Goloboff. Analyzing large data sets in reasonable times: Solutions for composite optima. *Cladistics*, 15:415–428, 1999.

[Gonnet *et al.*, 1992] cité p. 41
Gaston H. Gonnet, Mark A. Cohen, and Steven A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[Gordon, 1986] cité p. 59
Allan D. Gordon. Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification*, 3:335–348, 1986.

[Gray and Schucany, 1972] cité p. 56
Henry L. Gray and William R. Schucany. *The Generalized Jackknife Statistic*. Dekker, New York, 1972.

[Greenblatt and Spigelman, 2003] cité p. 1
Charles Greenblatt and Mark Spigelman. *Emerging Pathogens: Archaeology, Ecology and Evolution of Infectious Disease*. Oxford University Press, 2003.

[Guindon and Gascuel, 2003] cité p. 25
Stéphane Guindon and Olivier Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003.

[Gusfield, 1997] cité p. 34
Daniel M. Gusfield. *Algorithms on strings, trees, and sequences: Computer science and computational biology*. University of Cambridge, 1997.

[Haeckel, 1866] cité p. 11
Ernst Haeckel. *Generelle Morphologie der Organismen*. Georg Riemer (Berlin), 1866.

[Hansen and Mladenović, 1999] cité p. 54, 95
Pierre Hansen and Nenad Mladenović. An introduction to variable neighborhood search. In *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1999.

[Hao *et al.*, 1999] cité p. 47
Jin-Kao Hao, Philippe Galinier, and Michel Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contrainte. *Revue d'Intelligence Artificielle*, 13(2):283–324, 1999.

[Harvey *et al.*, 1996] cité p. 25
Paul H. Harvey, John Maynard Smith, Sean Nee, and Andrew J. Leigh Brown. *New uses for new phylogenies*. Oxford University Press, 1996.

[Hasegawa *et al.*, 1985] cité p. 23
Masami Hasegawa, Hirohisa Kishino, and Taka aki Yano. Dating the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22:160–174, 1985.

[Hendy and Penny, 1982] cité p. 48
Michael D. Hendy and David Penny. *Mathematical Biosciences*, 60:133–142, 1982.

[Henikoff and Henikoff, 1992] cité p. 41
Steven Henikoff and Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science, USA*, 89:10915–10919, 1992.

[Hennig, 1966] cité p. 20, 28
Willi Hennig. *Phylogenetic systematic*. University of Illinois Press (Urbana), 1966.

[Hillis *et al.*, 1996] cité p. 25
David M. Hillis, Barbara K. Mable, and Craig Moritz. Applications of molecular systematics and the future of the field. In D.M. Hillis, D.M. Moritz, and B.K. Mable, editors, *Molecular systematics (2nd edition)*, pages 515–543. Sinauer (Sunderland), 1996.

[Holland, 1975] cité p. 57
John H. Holland. *Adaption in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[Hoos and Stützle, 2005] cité p. 54, 58
Holger H. Hoos and Thomas Stützle. *Stochastic Local Search, Foundations and Applications*. Morgan Kaufmann Publishers, 2005.

[Horovitz, 1999] cité p. 55
Inés Horovitz. A report on one day symposium on numerical cladistics. *Cladistics*, 15:177–182, 1999.

[Hovenkamp, 2004] cité p. 60
Peter Hovenkamp. Review of: T.N.T. – Tree analysis using New Technology. Version 1.0, by P. Goloboff, J. S. Farris and K. Nixon. *Cladistics*, 20:378–383, 2004.

[Huxley, 1940] cité p. 14
Julian Huxley. *The new systematics*. Oxford University Press (London), 1940.

[Jin and Nei, 1990] cité p. 23
Li Jin and Masatoshi Nei. Limitations of the evolutionary parsimony method of phylogenetic analysis. *Molecular Biology and Evolution*, 7:82–102, 1990.

[Johnson, 1967] cité p. 19
Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–245, 1967.

[Jones *et al.*, 1992] cité p. 23
David T. Jones, William R. Taylor, and Janet M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences*, 8:275–282, 1992.

[Jukes and Cantor, 1969] cité p. 22
Thomas H. Jukes and Charles R. Cantor. Evolution of protein molecules. *Mammalian protein metabolism*, III:21–123, 1969.

[Keim *et al.*, 2004] cité p. 1
Paul Keim, Matthew N. Van Ert, Talima Pearson, Amy J. Vogler, Lynn Y. Huynh, and David M. Wagner. Anthrax molecular epidemiology and forensics: using the appropriate marker for different evolutionary scales. *Infection, Genetics and Evolution*, 4:205–213, 2004.

[Kimura, 1980] cité p. 23, 69
Motoo Kimura. A simple method for estimating evolutionary rate of base substitution through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111–120, 1980.

[Kirkpatrick *et al.*, 1983] cité p. 56
Scott Kirkpatrick, Charles D. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(13 May 1983):671–680, 1983.

[Källersjö *et al.*, 1998] cité p. 59
M. Källersjö, J. S. Farris, M. W. Chase, B. Bremer, M. F. Fay, C. J. Humphries, G. Peterson, O. Seberg, and K. Bremer. Simultaneous parsimony jackknife analysis of 2538 *rbc*L DNA sequences reveals support for major glades of green plants, land plants, seed plants and flowering plants. *Plant Systematics and Evolution*, 213:259–287, 1998.

[Kluge and Farris, 1969] cité p. 42
Arnold G. Kluge and James S. Farris. Quantitative phyletics and the evolution of anurans. *Systematic Zoology*, 18(1):1–32, 1969.

[Kolaczkowski and Thornton, 2004] cité p. 42
Bryan Kolaczkowski and Joe W. Thornton. Performance of maximum parsimony and likelihood phylogenetics when evolution is heterogeneous. *Nature*, 431:980–984, 2004.

[Kolen and Pesch, 1994] cité p. 58
Antoon W. J. Kolen and Erwin Pesch. Genetic local search in combinatorial optimization. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 48, 1994.

[Kuhner and Felsenstein, 1994] cité p. 69
Mary K. Kuhner and Joseph Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–468, 1994.

[Lanave *et al.*, 1984] cité p. 23
Cecilia Lanave, Giuliano Preparata, Cecilia Saccone, and Giovanni Serio. A new method for calculating evolutionary substitution rates. *Journal of Molecular Evolution*, 20(1):86–93, 1984.

[Land and Doig, 1960] cité p. 48
Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[Lardeux *et al.*, 2006] cité p. 58
Frédéric Lardeux, Frédéric Saubion, and Jin-Kao Hao. GASAT: a genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253, 2006.

[Lewis, 1998] cité p. 58
Paul O. Lewis. A genetic algorithm for maximum likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15:277–283, 1998.

[Lourenço *et al.*, 2002] cité p. 55
Helena R. Lourenço, Olivier Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 2002.

[Luckow and Pimentel, 1985] cité p. 70
Melissa Luckow and R. A. Pimentel. An empirical comparison of numerical wagner computer programs. *Cladistics*, 1:47–66, 1985.

[Lyell, 1830 1833] cité p. 10
Charles Lyell. *Principles of Geology*. 1830–1833.

[Maddison, 1991] cité p. 52
David R. Maddison. The discovery and importance of multiple islands of most parsimonious trees. *Systematic Zoology*, 43(3):315–328, 1991.

[Maidak *et al.*, 2000] cité p. 59
Bonnie L. Maidak, James R. Cole, Timothy G. Lilburn, Charles T. Parker Jr., Paul Saxman, Jason M. Stredwick, George M. Garrity, Bing Li, Gary J. Olsen, Sakti Pramanik, Thomas M. Schmidt, and James M. Tiedje. The RDP-II (Ribosomal Database Project. *Nucleic Acids Research*, 28:173–174, 2000.

[Matsuda, 1996] cité p. 58
Hideo Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. *Proceedings of 1st Pacific Symposium on Biocomputing*, pages 512–523, 1996.

[Meacham, 1984] cité p. 13
Christopher A. Meacham. The role of hypothesized direction of characters in the estimation of evolutionary history. *Taxon*, 33(1):26–38, 1984.

[Merz and Freisleben, 1997] cité p. 58
Peter Merz and Bernd Freisleben. A genetic local search approach to the quadratic assignment problem. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 465–472. Morgan Kaufmann Publishers Inc., 1997.

[Metropolis *et al.*, 1953] cité p. 56
Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, and Augusta H. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[Michalewicz, 1996] cité p. 58
Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[Michener and Sokal, 1958] cité p. 19
Charles D. Michener and Robert R. Sokal. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[Miller and Goldberg, 1995] cité p. 113
Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[Mladenović and Hansen, 1997] cité p. 54, 54
Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[Moilanen, 1999] cité p. 3, 58
Atte Moilanen. Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics*, 15(1):39–50, 1999.

[Moscato, 1999] cité p. 58
Pablo Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, London, 1999.

[Mount, 2001] cité p. 1
David W. Mount. *Bioinformatics: sequence and genome analysis*. CSHL Press, 2001.

[Nagata and Kobayashi, 1997] cité p. 58
Yuichi Nagata and Shigenobu Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 450–457. Morgan Kaufmann, 1997.

[Nei and Kumar, 2000] cité p. 1
Masatoshi Nei and Sudhir Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, 2000.

[Neyman, 1971] cité p. 22, 23
J. Neyman. Molecular studies of evolution: A source of novel statistical problems. *Statistical Decision Theory and Related Topics*, pages 1–27, 1971.

[Nixon, 1999] cité p. 55, 69
Kevin C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15(4):407–414, 1999.

[Papadimitriou, 1994] cité p. 46
Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pennisi, 2003] cité p. 60
Elizabeth Pennisi. Modernizing the tree of life. *Science*, 300:1692–1697, 2003.

[Platnick, 1987] cité p. 70
Norman I. Platnick. An empirical comparison of microcomputer parsimony programs. *Cladistics*, 3:121–144, 1987.

[Platnick, 1989] cité p. 70
Norman I. Platnick. An empirical comparison of microcomputer parsimony programs II. *Cladistics*, 5:145–161, 1989.

[Quicke *et al.*, 2001] cité p. 56
Donald L. J. Quicke, Jason Taylor, and Andy Purvis. Changing the landscape: A new strategy for estimating large phylogenies. *Systematic Biology*, 50(1):60–66, 2001.

[Ragan, 1992] cité p. 59
M. A. Ragan. Phylogenetic inference based on matrix representation of trees. *Molecular phylogenetics and evolution*, 1:53–58, 1992.

[Rambaut *et al.*, 2001] cité p. 1
Andrew Rambaut, David L. Robertson, Oliver G. Pybus, Martine Peeters, and Edward C. Holmes. Phylogeny and the origin of hiv-1. *Nature*, 410:1047–1048, 2001.

[Reeves, 1993] cité p. 46
Colin C. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Press, Oxford, 1993.

[Ribeiro and Vianna, 2003] cité p. 3, 58, 70, 70, 116, 116
Celso C. Ribeiro and Dalessandro S. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In *II Workshop Brasileiro de Bioinformática*, pages 97–102, 2003.

[Ribeiro and Vianna, 2005] cité p. 54, 54, 57, 70, 70, 92
Celso C. Ribeiro and Dalessandro S. Vianna. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research*, 12:325–338, 2005.

[Rice and Warnow, 1997] cité p. 42, 69
Kenneth Rice and Tandy Warnow. Parsimony is hard to beat. In *Computing and Combinatorics*, volume 1276 of *Lecture Notes in Computer Science*, pages 124–133. Springer-Verlag, 1997.

[Robinson and Foulds, 1981] cité p. 122
David F. Robinson and Leslie R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[Robinson, 1971] cité p. 51
David F. Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory, Series B*, 11:105–119, 1971.

[Rzhetsky and Nei, 1992] cité p. 20
Andrey Rzhetsky and Masatoshi Nei. A simple method for estimating and testing minimum-evolution trees. *Molecular Biology and Evolution*, 9:945–967, 1992.

[Rzhetsky and Nei, 1993] cité p. 20
Andrey Rzhetsky and Masatoshi Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10:1073–1095, 1993.

[Saitou and Nei, 1987] cité p. 20
Naruya Saitou and Masatoshi Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[Sankoff and Rousseau, 1975] cité p. 36
David Sankoff and Pascale Rousseau. Locating the vertices of a Steiner tree in an arbitrary metric space. *Mathematical Programming*, 9(2):240–246, 1975.

[Sankoff, 1975] cité p. 41
David Sankoff. Minimal mutation trees of sequences. *Journal of Applied Mathematics*, 28(1):35–42, 1975.

[Schröder, 1870] cité p. 16
Ernst Schröder. Vier kombinatorische probleme. *Zeitschrift für Mathematik und Physik*, 15:361–376, 1870.

[Simpson, 1961] cité p. 13
George Gaylord Simpson. *Principles of animal taxonomy*. Columbia University Press (New York), 1961.

[Sneath and Sokal, 1973] cité p. 19
Peter H. A. Sneath and Robert R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. W. H. Freeman and Company (San Francisco), 1973.

[Snell *et al.*, 2000] cité p. 60
Quinn Snell, Michael Whiting, Mark Clement, and David McLaughlin. Parallel phylogenetic inference. In *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA*, pages 62–62. IEEE ACM, 2000.

[Sokal and Michener, 1957] cité p. 17
Robert R. Sokal and Charles D. Michener. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[Sokal and Sneath, 1963] cité p. 13, 17
Robert R. Sokal and Peter H. A. Sneath. *Principles of numerical taxonomy*. W. H. Freeman and Company, 1963.

[Sourdis and Nei, 1988] cité p. 42
John Sourdis and Masatoshi Nei. Relative efficiencies of the maximum parsimony and distance-matrix methods in obtaining the correct phylogenetic tree. *Molecular Biology and Evolution*, 5:298–311, 1988.

[Stamatakis, 2005] cité p. 57
Alexandros Stamatakis. An efficient program for phylogenetic inference using simulated annealing. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 7*, page 198, Washington, DC, USA, 2005. IEEE Computer Society.

[Stanley, 1997] cité p. 16
Richard P. Stanley. *Enumerative combinatorics. Vol. 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1997.

[Swofford and Olsen, 1990] cité p. 2, 52, 52, 52
David L. Swofford and Gary J. Olsen. Phylogeny reconstruction. In D. M. Hillis, D. M. Moritz, and B. K. Mable, editors, *Molecular systematics*, pages 411–501. Sinauer Associates, Inc., Sunderland, USA, 1990.

[Swofford *et al.*, 1996] cité p. 49
David L. Swofford, Gary J. Olsen, Peter J. Waddell, and David M. Hillis. Phylogenetic inference. In D. M. Hillis, D. M. Moritz, and B. K. Mable, editors, *Molecular systematics (2nd edition)*, pages 407–514. Sunderland, USA, 1996.

[Swofford, 2002] cité p. 60
David L. Swofford. PAUP*. Phylogenetic Analysis Using Parsimony (*and other methods). Version 4. Sinauer Associates, Sunderland, Massachusetts, 2002.

[Tamura and Nei, 1993] cité p. 23
Koichiro Tamura and Masatoshi Nei. Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution*, 10:512–526, 1993.

[Tamura, 1992] cité p. 23
Koichiro Tamura. Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+C content biases. *Molecular Biology and Evolution*, 9:678–687, 1992.

[Tassy, 1986] cité p. 20
Pascal Tassy. *L'ordre et la diversité du vivant. Quel statut scientifique pour les classifications biologiques ?* Fondation Diderot (Paris), 1986.

[Tavaré, 1986] cité p. 23
Simon Tavaré. Some probabilistic and statistical problems on the analysis of DNA sequences. *Lectures on Mathematics in the Life Sciences*, 17:57–86, 1986.

[Tryon, 1939] cité p. 19
Robert C. Tryon. *Cluster Analysis*. Edwards Brothers (Ann Arbor), 1939.

[Tukey, 1958] cité p. 56
John W. Tukey. Bias and confidence in not-quite large samples. *Annals of Mathematical Statistics*, 614, 1958.

[Ulder *et al.*, 1991] cité p. 58
Nico L. J. Ulder, Emile H. L. Aarts, Hans-Jürgen Bandelt, Peter J. M. van Laarhoven, and Erwin Pesch. Genetic local search algorithms for the traveling salesman problem. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN*

I, volume 496 of *Lecture Notes in Computer Science*, pages 109–116, Berlin, Germany, 1991. Springer.

[Vidal, 1993] cité p. 57
Rene V. V. Vidal. *Applied Simulated Annealing*. Springer, 1993.

[von Linné, 1735 1770] cité p. 9
Carl von Linné. *Systema Naturae per regna tria naturae, secundum classes, ordines, genera, species, cum characteribus differentiis, synonymis, locis*. 1735-1770.

[Ward, 1963] cité p. 19
Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[Waterman and Smith, 1978] cité p. 51
Michael S. Waterman and Temple F. Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73:784–900, 1978.

[Yan and Bader, 2003] cité p. 52
Mi Yan and David A. Bader. Fast character optimization in parsimony phylogeny reconstruction. Technical report, University of New Mexico, Albuquerque, 2003.

Liste des publications personnelles

Revues internationales avec comité de lecture

1. Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao
   Progressive Tree Neighborhood Applied to the Maximum Parsimony Problem
   IEEE/ACM Transactions on Computational Biology and Bioinformatics
   En révision

Conférences internationales avec comité de selection

1. Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao
   A Distance-based Information Preservation Tree Crossover for the Maximum Parsimony Problem
   Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (Reykjavik, Islande, Septembre 2006)
   Lecture Notes in Computer Science 4193 : 761-770, Springer-Verlag, 2006.
2. Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao
   Progressive Tree Neighborhood Applied to the Maximum Parsimony Problem
   Proceedings of the Mini EURO Conference on Variable Neighborhood Search (Tenerife, Espagne, Novembre 2005)
3. Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao
   Local Search for the Maximum Parsimony Problem
   Proceedings of the First International Conference on Natural Computation (Changsha, Chine, Aout 2005)
   Lecture Notes in Computer Science 3612 : 678-683, Springer-Verlag, 2005.

Conférences francophones avec comité de selection

1. Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao
   Hybrid Evolutionary Algorithm for reconstructing Phylogenetic Trees
   Actes des Septièmes Journées Ouvertes Biologie Informatique Mathématiques, p.241-249 (Bordeaux, France, Juillet 2006)

2. Adrien Goëffon, Jean-Michel Richer et Jin-Kao Hao
   Voisinage d'arbre évolutif appliqué au problème Maximum Parcimonie
   Actes des Sixièmes Journées Ouvertes Biologie Informatique Mathématiques, p.245-255 (Lyon, France, Juillet 2005)
3. Adrien Goëffon, Jean-Michel Richer et Jin-Kao Hao
   Voisinage d'arbre évolutif appliqué au problème Maximum Parcimonie
   Actes des Premières Journées Francophones de Programmation par Contraintes, p.443-446 (Lens, France, Juin 2005)

Congrès nationaux avec actes de résumés étendus

1. Adrien Goëffon, Jean-Michel Richer et Jin-Kao Hao
   Recherche locale à voisinage évolutif pour la reconstruction de phylogénies
   Actes du Sixième Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, p.187-188 (Tours, France, Mars 2005)

Nouvelles heuristiques de voisinage et mémétiques pour le problème Maximum de Parcimonie

Résumé

La reconstruction phylogénétique vise à reconstituer l'histoire évolutive d'un ensemble d'espèces sous forme d'un arbre. Parmi les méthodes de reconstruction, le problème Maximum de Parcimonie (MP) consiste à trouver un arbre binaire dont les feuilles sont associées à des séquences de caractères données, et qui minimise le *score de parcimonie*. Les méthodes de résolution existantes de ce problème NP-complet s'attachent généralement à appliquer des méthodes heuristiques traditionnelles, comme des algorithmes gloutons et de recherche locale. L'une des difficultés du problème repose sur la manipulation d'arbres et la définition de voisinages d'arbres.

Dans cette thèse, nous nous intéressons en premier lieu à l'amélioration des techniques de résolution du problème MP basées sur un algorithme de descente. Après avoir montré de manière empirique les limites des voisinages existants, nous introduisons un voisinage progressif qui évolue au cours de la recherche afin de limiter l'évaluation de voisins infructueux lors d'une descente. L'algorithme obtenu est ensuite hybridé à un algorithme génétique utilisant un croisement d'arbres spécifique fondé sur les mesures de distance entre chaque couple d'espèces dans l'arbre. Cet algorithme mémétique exhibe des résultats très compétitifs, tant sur des jeux de test tirés de la littérature que sur des jeux générés aléatoirement.

Mots-clés : reconstruction phylogénétique, problème Maximum de Parcimonie, optimisation, recherche locale, algorithmes mémétiques, voisinages d'arbres, croisements d'arbres

New neighborhood and memetic heuristics for the Maximum Parsimony problem

Abstract

Phylogenetic reconstruction aims at reconstructing the evolutionary history of a set of species, represented by a tree. Among the reconstruction methods, the Maximum Parsimony (MP) problem consists, given a set of aligned sequences to find a binary tree, whose leaves are associated to the sequences and which minimizes the parsimony score. Traditionally, existing resolution approaches of this NP-complete problem apply basic heuristic methods, like greedy algorithms and local search. One of the difficulties concerns the handling of binary trees and the definition of tree neighborhoods. In this thesis, we first focus on an improvement of descent algorithms. We empirically show the limits of the existing tree neighborhoods, and introduce a progressive neighborhood which evolves during the search to limit the evaluation of inappropriate neighbors. This algorithm is combined with a genetic algorithm which uses a specific tree crossover based on topological distances between each pair of leaves. This memetic algorithm shows very competitive results, both on real benchmarks taken from the literature as well as with randomly generated instances.

Keywords : phylogenetic reconstruction, Maximum Parsimony problem, optimization, local search, memetic algorithms, tree neighborhoods, tree crossovers