Titre de l'ouvrage

Prénom NOM

January 24, 2024

Avant-propos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam maximus accumsan erat, sed ultricies augue ultricies et. Proin suscipit nunc ut urna venenatis, sit amet vehicula arcu suscipit. Phasellus id lorem quis magna tristique gravida elementum eu metus. Nam scelerisque justo quis elit ullamcorper, et dictum metus iaculis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque sed lacus nunc. Sed vel nulla mi. Nullam bibendum est cursus suscipit blandit. In eu ante id mi sagittis tristique eget vel ligula.

Nam lobortis tortor diam, vitae suscipit ligula placerat venenatis. Praesent eu auctor tellus, eget auctor velit. Nullam convallis placerat dui, sit amet porttitor est condimentum sed. Nunc dui lectus, eleifend efficitur ornare quis, consequat nec ligula. Sed vitae nibh iaculis augue lobortis dignissim ac ac ipsum. Praesent at vulputate tortor. Fusce et nibh ac velit porta tincidunt. Aliquam dignissim lorem at arcu varius semper. Integer et pharetra sem. Maecenas eget facilisis lectus. Suspendisse in ex augue. Cras non tortor in magna rhoncus dictum non in lectus. Interdum et malesuada fames ac ante ipsum primis in faucibus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed auctor, dui nec condimentum vulputate, nisl augue mollis diam, non pellentesque nulla turpis porttitor lorem. Praesent sapien magna, ullamcorper ac nisi quis, euismod tincidunt odio.

– Aliquam dignissim lorem at arcu varius semper. Integer et pharetra sem. Maecenas eget facilisis lectus. Suspendisse in ex augue. Cras non tortor in magna rhoncus dictum non in lectus.

– Quisque malesuada posuere luctus. Vivamus porttitor eget nibh ut eleifend. Donec sed faucibus lectus, elementum molestie eros. Etiam aliquet neque vitae porta malesuada. Aliquam at mauris vel arcu bibendum bibendum. Ut erat dolor, mattis id

ullamcorper nec, maximus at elit. Morbi tristique, erat nec condimentum auctor, sem dolor congue mi, vitae tincidunt nunc metus vel ipsum. Nullam vitae vulputate mauris, quis rutrum augue. Curabitur cursus interdum congue. Vestibulum consectetur velit at fermentum iaculis. Nullam ac facilisis nisl. Pellentesque posuere libero ut felis finibus, ut commodo augue porta. Fusce varius consectetur tellus. Cras ultricies tellus nec est pellentesque, nec fringilla turpis dictum. Fusce lacinia ex in euismod ultrices.

– Praesent malesuada iaculis velit a tincidunt. Nunc sed felis a risus semper dictum et tempus leo. Aenean id mauris ipsum. Nam bibendum non justo nec tincidunt. Aenean purus elit, tincidunt sed venenatis ut, mattis sit amet est. Phasellus ut magna dapibus felis vestibulum sodales. Aliquam rutrum dolor felis, euismod efficitur enim rutrum id. Nullam scelerisque hendrerit justo, vitae dignissim urna euismod non.

Nunc luctus augue lectus, et fringilla orci dignissim efficitur. Vestibulum faucibus, ligula et aliquet consectetur, tellus odio vestibulum magna, eu scelerisque elit elit ut elit. Nunc pretium auctor viverra. Suspendisse nec nisl a orci tempus cursus ultricies et orci. Vivamus viverra pharetra erat, pellentesque aliquet odio. Donec lacinia malesuada purus eu suscipit. Duis mattis mi justo, non interdum eros gravida accumsan. Donec ac placerat erat. Aenean fringilla in leo at hendrerit. Integer congue sem in nunc scelerisque, quis volutpat velit iaculis. Praesent eu purus tincidunt, gravida arcu a, porta quam. Pellentesque lobortis diam a interdum blandit.

Duis pellentesque nibh id ante imperdiet, non venenatis ante varius. Etiam in nisl in dolor mattis convallis. Maecenas nibh odio, ullamcorper a volutpat vel, hendrerit iaculis nisl. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin ante arcu, commodo eu orci eget, convallis pharetra purus. Mauris rhoncus nulla dolor. Quisque et laoreet eros.

Sed in nulla faucibus, dignissim odio quis, elementum nibh. Cras ultricies, augue feugiat tempor feugiat, arcu mi lacinia tellus, pharetra viverra ligula purus et odio. Suspendisse non ultrices odio. Donec rutrum nunc id mauris fermentum finibus. Etiam eget dignissim urna. Mauris dignissim ultricies tortor a vulputate. Aliquam erat volutpat. Sed at velit et libero commodo feugiat quis nec risus. Duis sit amet vestibulum lacus, id semper turpis. Etiam imperdiet molestie lorem, vel ultrices nulla. Phasellus efficitur nec elit eget varius. Curabitur tristique tortor in ultricies laoreet. Curabitur tristique mauris vitae lectus accumsan pulvinar. Sed tortor enim, commodo vel augue non, ornare tristique dolor. Donec eget augue dolor. Donec vitae lectus et nulla fermentum sodales vel ac ex.

Phasellus turpis tortor, aliquam ut feugiat a, vehicula rhoncus elit. Proin at nunc sed felis facilisis egestas. Sed facilisis feugiat auctor. Ut quis mattis mauris, sed efficitur justo. Fusce volutpat nulla et augue condimentum, faucibus laoreet arcu sollicitudin.

Sed eros arcu, fringilla sit amet ante ac, posuere laoreet ligula. Duis rhoncus orci sed suscipit feugiat. Etiam eleifend augue quis lacus scelerisque porta. Curabitur ultricies eget neque nec vestibulum. Vivamus pellentesque leo sed lacus rhoncus consectetur.

Quisque malesuada posuere luctus. Vivamus porttitor eget nibh ut eleifend. Donec sed faucibus lectus, elementum molestie eros. Etiam aliquet neque vitae porta malesuada. Aliquam at mauris vel arcu bibendum bibendum. Ut erat dolor, mattis id ullamcorper nec, maximus at elit. Morbi tristique, erat nec condimentum auctor, sem dolor congue mi, vitae tincidunt nunc metus vel ipsum. Nullam vitae vulputate mauris, quis rutrum augue. Curabitur cursus interdum congue. Vestibulum consectetur velit at fermentum iaculis. Nullam ac facilisis nisl. Pellentesque posuere libero ut felis finibus, ut commodo augue porta. Fusce varius consectetur tellus. Cras ultricies tellus nec est pellentesque, nec fringilla turpis dictum. Fusce lacinia ex in euismod ultrices.

Proin mollis tellus accumsan, laoreet purus id, congue nibh. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc rutrum urna accumsan massa dapibus, quis pellentesque elit ornare. Donec mollis nec leo a accumsan. Aliquam erat volutpat. Vivamus sodales sed leo ut congue. Nullam sed malesuada ex. Proin pellentesque ante elit, eget tincidunt ligula facilisis nec. Maecenas dapibus iaculis tortor id lobortis. Morbi a iaculis tortor. Nam lorem sem, ultricies eget justo eget, aliquet gravida nisi. Ut aliquam lacus ut purus pulvinar, ac tincidunt turpis congue. Phasellus non tristique sem. Praesent vitae aliquet purus, nec suscipit dolor.

Nunc rutrum efficitur dui, ut maximus leo dapibus eget. Morbi nisi velit, volutpat in erat non, viverra posuere mauris. Nullam mattis facilisis ligula, et iaculis arcu malesuada maximus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin malesuada, sem sed luctus aliquet, sem est feugiat ex, a dignissim mauris ante vitae magna. Donec viverra est magna, ac consectetur eros dapibus et. Vestibulum condimentum, felis ut scelerisque sodales, massa ligula placerat nunc, non volutpat nibh justo auctor nisi.

Proin tincidunt ex metus, in consequat tellus posuere vitae. Morbi suscipit sagittis nunc ut euismod. Fusce accumsan mattis diam sit amet suscipit. Donec id ex sed mi condimentum condimentum faucibus id est. Maecenas vitae pellentesque felis. Vestibulum ornare euismod ipsum, non tempor dolor tincidunt ut. Suspendisse at mi ornare, eleifend nulla sit amet, fringilla nulla. Curabitur fermentum ligula nisi, sit amet finibus dolor posuere ut. Etiam ullamcorper lectus ut iaculis vehicula.

1

Titre chapitre

Jean DUPONT et Sophie MARTIN
Université de Montpellier, Montpellier, France

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse luctus rutrum orci, non malesuada sapien sollicitudin vitae. Vivamus mauris nunc, bibendum sit amet dapibus eu, ultricies ut arcu. Vivamus fringilla fringilla ipsum auctor mollis. Vestibulum ut venenatis nisl, quis accumsan urna. Etiam in pharetra lacus. Suspendisse potenti. Sed sit amet tincidunt augue. Duis et risus nec dui semper auctor. Vivamus lacus dolor, commodo non viverra eu, volutpat in massa. Mauris iaculis lorem quis ligula molestie, in laoreet enim pretium. Curabitur varius ultrices purus, non consequat leo auctor ut. Mauris semper rhoncus ante, quis sollicitudin dolor varius eu.

1.1. Premier niveau d'intertitre

1.1.1. *Deuxième niveau d'intertitre*

1.1.1.1. *Troisième niveau d'intertitre*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse luctus rutrum orci, non malesuada sapien sollicitudin vitae. Vivamus mauris nunc, bibendum sit amet dapibus eu, ultricies ut arcu. Vivamus fringilla fringilla ipsum auctor mollis. Vestibulum ut venenatis nisl, quis accumsan urna. Etiam in pharetra lacus. Suspendisse potenti. Sed sit amet tincidunt augue. Duis et risus nec dui semper auctor. Vivamus lacus dolor, commodo non viverra eu, volutpat in massa. Mauris

iaculis lorem quis ligula molestie, in laoreet enim pretium. Curabitur varius ultrices purus, non consequat leo auctor ut. Mauris semper rhoncus ante, quis sollicitudin dolor varius eu.

Aenean commodo lectus nunc, facilisis luctus lectus pulvinar ac. Sed quis lectus tellus. Aenean vel sem sagittis, sollicitudin leo eget, molestie neque. Donec viverra at ipsum quis feugiat. Integer ac libero sed risus lacinia lobortis quis ac lacus. Proin felis urna, hendrerit id rutrum vitae, cursus id quam. Donec interdum facilisis nulla vitae fermentum. Sed dignissim magna tempus, vehicula ante a, iaculis lacus. Fusce condimentum, dolor id posuere cursus, erat odio laoreet mauris, sed mattis sem nulla in erat. Duis sagittis ligula odio, nec aliquam mauris lacinia fermentum. Praesent quis euismod velit. Nullam quis efficitur enim, a mattis sapien. Etiam nibh tortor, dapibus sit amet lacus at, pellentesque fringilla lacus. Morbi eget elementum enim.

Sed viverra, justo sed tempor congue, nibh nisi porta velit, quis tempus nunc diam quis ante. Curabitur maximus nisi quis quam viverra, at sodales odio interdum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vivamus aliquam elit ac dictum scelerisque. Donec imperdiet feugiat turpis ac gravida. Phasellus efficitur mollis dolor vitae tincidunt. Nullam facilisis, sapien sit amet vestibulum blandit, ex nibh gravida tortor, quis rhoncus lectus metus non sapien. Morbi bibendum ligula ut enim ultricies, quis semper nisl cursus.

Duis in ligula sit amet augue hendrerit volutpat. Aliquam viverra pellentesque consequat. Nulla vestibulum in ipsum vel dapibus. Aliquam interdum lacinia rutrum. Nullam sit amet sem in orci malesuada volutpat id vel tellus. Suspendisse potenti. Ut dapibus mauris non sapien gravida maximus. Cras molestie tellus orci, et mollis ipsum vulputate eget. Nunc iaculis luctus laoreet. Nulla eget nisi non tortor sodales aliquet.

1.1.1.2. *Troisième niveau d'intertitre*

Pellentesque semper est leo, vitae efficitur massa cursus eu. Ut odio nisi, imperdiet in erat ut, pulvinar malesuada lorem. Praesent nisi arcu, faucibus ac tellus nec, tincidunt aliquam nisi. Donec nec justo purus. Duis lacinia ante turpis, iaculis eleifend est eleifend iaculis. Fusce accumsan nunc sed felis convallis viverra. Pellentesque vestibulum massa nulla, tincidunt rutrum ex ultrices vitae. Fusce quis arcu ex. Vivamus sagittis faucibus velit, consectetur vulputate nisi tincidunt eget.

Donec dignissim mi nec dapibus dictum. Praesent quis diam fermentum, feugiat velit ac, dapibus magna. Nulla rhoncus lobortis massa ut interdum. Nulla quis ligula quis erat posuere tempus. Proin suscipit a leo ut suscipit. Nam vel lorem dictum, faucibus ipsum non, molestie felis. Fusce venenatis ipsum vitae turpis faucibus, sed feugiat felis lacinia. Fusce id ligula bibendum, semper felis at, accumsan nisl. Integer id massa vel ante sagittis facilisis.

Praesent sit amet nunc ornare, viverra odio id, interdum metus. Etiam justo nunc, pulvinar gravida sodales sit amet, consectetur eget nibh. Fusce sit amet vehicula felis. Sed in ipsum fermentum, vehicula velit et, gravida massa. Aenean commodo ullamcorper tortor, ac pellentesque turpis facilisis aliquam. Phasellus eget commodo eros. Sed ac sagittis felis, ultricies vulputate lectus. Nunc imperdiet diam id laoreet blandit. Sed laoreet massa nec ligula facilisis, id rhoncus sapien pellentesque. Integer iaculis mauris neque, ac feugiat eros laoreet eget. Nulla molestie, erat a convallis tristique, odio mi tempus purus, sit amet dapibus nisl nisi ut libero. Nam tincidunt eu orci et eleifend. Ut at nulla lacus.

1.1.2. *Deuxième niveau d'intertitre*

Morbi et neque quis ex pellentesque sagittis. Morbi molestie est at ex porta tempor. Nulla accumsan nibh nec nisi fringilla pretium. Mauris eget cursus massa. Cras blandit nec ligula et facilisis. Donec faucibus mauris convallis, tristique nulla ac, condimentum velit. Etiam a maximus diam, at lacinia arcu. Etiam sed est non metus sagittis sollicitudin. Morbi ut vestibulum enim. Suspendisse iaculis dui et sem bibendum viverra. Suspendisse sapien sapien, porta vitae bibendum et, ultrices id libero.

Interdum et malesuada fames ac ante ipsum primis in faucibus. Aliquam eget pretium lacus. Fusce turpis ipsum, sodales tempor nisi vitae, facilisis feugiat enim. Praesent commodo neque at neque aliquam sodales. Duis libero ante, sodales vel blandit quis, auctor non enim. Cras molestie elit vel arcu ultricies accumsan. Sed tristique justo et tincidunt aliquet. Nam fermentum luctus orci et volutpat.

Maecenas gravida, metus non tempus tempus, quam ante lobortis augue, sit amet accumsan neque elit feugiat urna. Praesent mauris massa, cursus vitae sagittis a, venenatis in orci. Nullam mollis eu diam in viverra. Maecenas odio augue, hendrerit auctor nulla lobortis, ultrices fermentum felis. Quisque at congue odio, eu ullamcorper ligula. Morbi a nisi cursus, laoreet sem et, molestie sem. Maecenas eu elit dolor. Ut vitae accumsan eros, sit amet sodales lectus. Phasellus a dui eget quam elementum iaculis sed nec sapien. Proin feugiat blandit suscipit. Curabitur ligula purus, viverra in congue iaculis, vulputate vitae mauris. Maecenas eget ex et lectus faucibus facilisis. Vestibulum et scelerisque ante. Vivamus laoreet velit nec laoreet pharetra. Suspendisse potenti.

1.1.3. *Deuxième niveau d'intertitre*

Morbi et neque quis ex pellentesque sagittis. Morbi molestie est at ex porta tempor. Nulla accumsan nibh nec nisi fringilla pretium. Mauris eget cursus massa. Cras blandit

nec ligula et facilisis. Donec faucibus mauris convallis, tristique nulla ac, condimentum velit. Etiam a maximus diam, at lacinia arcu. Etiam sed est non metus sagittis sollicitudin. Morbi ut vestibulum enim. Suspendisse iaculis dui et sem bibendum viverra. Suspendisse sapien sapien, porta vitae bibendum et, ultrices id libero.

Interdum et malesuada fames ac ante ipsum primis in faucibus. Aliquam eget pretium lacus. Fusce turpis ipsum, sodales tempor nisi vitae, facilisis feugiat enim. Praesent commodo neque at neque aliquam sodales. Duis libero ante, sodales vel blandit quis, auctor non enim. Cras molestie elit vel arcu ultricies accumsan. Sed tristique justo et tincidunt aliquet. Nam fermentum luctus orci et volutpat.

Maecenas gravida, metus non tempus tempus, quam ante lobortis augue, sit amet accumsan neque elit feugiat urna. Praesent mauris massa, cursus vitae sagittis a, venenatis in orci. Nullam mollis eu diam in viverra. Maecenas odio augue, hendrerit auctor nulla lobortis, ultrices fermentum felis. Quisque at congue odio, eu ullamcorper ligula. Morbi a nisi cursus, laoreet sem et, molestie sem. Maecenas eu elit dolor. Ut vitae accumsan eros, sit amet sodales lectus. Phasellus a dui eget quam elementum iaculis sed nec sapien. Proin feugiat blandit suscipit. Curabitur ligula purus, viverra in congue iaculis, vulputate vitae mauris. Maecenas eget ex et lectus faucibus facilisis. Vestibulum et scelerisque ante. Vivamus laoreet velit nec laoreet pharetra. Suspendisse potenti.

1.2. Premier niveau d'intertitre

Sed a lobortis massa. Pellentesque a neque rhoncus, malesuada velit vitae, semper ante. Quisque vehicula ultricies libero ac aliquam. Suspendisse potenti. Quisque ut tortor quis dolor semper vulputate. Nam non lobortis nulla. Suspendisse ornare cursus faucibus. Integer quis feugiat tellus. Nullam faucibus vehicula arcu quis laoreet. Nullam viverra, lacus id blandit blandit, urna elit mollis augue, vitae tempor neque sem quis lectus. Morbi sit amet tristique dolor. Fusce malesuada libero justo, ac interdum dui vestibulum ut. Aenean fringilla nulla et ante ultrices, in scelerisque ante pharetra. Donec luctus justo in orci dapibus, ut molestie nibh tempor.

Cras eu ex mattis metus finibus consectetur a in purus. Nunc vehicula imperdiet neque, in luctus ante. Donec velit dui, imperdiet sit amet vehicula at, tincidunt sodales quam. Nam eu placerat nunc, vel commodo metus. Suspendisse risus tortor, tincidunt ac fringilla sed, tincidunt non mi. Morbi condimentum massa nec velit pulvinar, eu sagittis urna venenatis. Aenean fermentum mattis erat nec tincidunt. Nunc in mi vitae leo elementum imperdiet vel ac erat. Sed placerat molestie elementum. Aliquam a massa eget ex facilisis posuere. Vivamus magna dui, lobortis sed nisl a, facilisis blandit leo. Sed pulvinar at nunc non sagittis. Nunc blandit volutpat nisl, eget efficitur felis bibendum id. Pellentesque et hendrerit ante. Pellentesque non aliquet est. Maecenas vehicula odio ac rhoncus congue.

Nunc quis orci dui. Duis eget lorem felis. Ut non tincidunt augue. Nulla sollicitudin laoreet euismod. Donec non imperdiet leo. Aliquam non quam et ipsum luctus vestibulum. Suspendisse sed orci non ipsum varius varius. Proin sed odio et diam sagittis bibendum. Nullam sollicitudin sem porttitor massa viverra, ut efficitur tellus mollis. Vestibulum ut felis orci. Quisque sapien libero, laoreet vel lobortis vel, mollis nec lectus. Mauris ullamcorper enim in fringilla commodo. Maecenas auctor eros vitae felis mollis pellentesque. Morbi hendrerit aliquam leo eu elementum.

Donec feugiat ligula vitae pharetra dignissim. Suspendisse potenti. Sed urna sapien, pellentesque at quam in, faucibus consectetur nibh. Aenean magna risus, semper at fermentum at, varius a libero. Morbi nec molestie dui. Suspendisse efficitur, est non bibendum tempus, augue diam dignissim ex, at pellentesque lectus dolor quis nulla. Vestibulum enim velit, porttitor non odio et, porta scelerisque ipsum.

1.2.1. *Deuxième niveau d'intertitre*

Donec malesuada ante dolor, sit amet molestie ante mollis sit amet. Sed pretium ultricies varius. Morbi lectus mauris, interdum at augue quis, semper hendrerit purus. Integer interdum, nunc ut vestibulum tincidunt, est ligula fermentum est, vel posuere leo lacus sit amet diam. Aliquam sollicitudin ante sit amet libero rhoncus convallis. Integer pretium commodo iaculis. Mauris aliquam, nibh vel tempus semper, orci mauris eleifend arcu, at facilisis neque tortor in urna. Duis vestibulum molestie enim, ultrices placerat odio posuere non.

Nulla imperdiet nulla at erat pellentesque faucibus. Sed vitae nisi dolor. Aenean aliquet molestie leo. Pellentesque scelerisque consectetur posuere. Integer eu sapien sit amet nunc imperdiet viverra vel quis mi. Morbi est arcu, venenatis mollis mollis id, euismod ac dui. Maecenas dapibus enim lacus, ac mattis purus venenatis at. Nullam a hendrerit neque, quis mattis augue. Vestibulum sed justo at metus suscipit laoreet eget vitae risus. Aliquam suscipit, ante a ullamcorper elementum, neque leo sagittis metus, sit amet mattis lectus est vitae diam. Praesent eu ligula a risus viverra rhoncus eu sed ante. Vivamus at nisi vitae odio aliquam mollis. Aenean porta ligula dolor, quis mattis velit fringilla at. Proin sollicitudin, est at volutpat dignissim, eros odio mollis lacus, nec hendrerit ipsum urna ac ipsum. Phasellus eros mi, maximus in nisi vel, sagittis viverra felis. Etiam vitae lacinia dui, ut fermentum tellus.

Morbi et neque quis ex pellentesque sagittis. Morbi molestie est at ex porta tempor. Nulla accumsan nibh nec nisi fringilla pretium. Mauris eget cursus massa. Cras blandit nec ligula et facilisis. Donec faucibus mauris convallis, tristique nulla ac, condimentum velit. Etiam a maximus diam, at lacinia arcu. Etiam sed est non metus sagittis sollicitudin. Morbi ut vestibulum enim. Suspendisse iaculis dui et sem bibendum viverra. Suspendisse sapien sapien, porta vitae bibendum et, ultrices id libero.

Interdum et malesuada fames ac ante ipsum primis in faucibus. Aliquam eget pretium lacus. Fusce turpis ipsum, sodales tempor nisi vitae, facilisis feugiat enim. Praesent commodo neque at neque aliquam sodales. Duis libero ante, sodales vel blandit quis, auctor non enim. Cras molestie elit vel arcu ultricies accumsan. Sed tristique justo et tincidunt aliquet. Nam fermentum luctus orci et volutpat.

1.2.2. *Deuxième niveau d'intertitre*

Sed a lobortis massa. Pellentesque a neque rhoncus, malesuada velit vitae, semper ante. Quisque vehicula ultricies libero ac aliquam. Suspendisse potenti. Quisque ut tortor quis dolor semper vulputate. Nam non lobortis nulla. Suspendisse ornare cursus faucibus. Integer quis feugiat tellus. Nullam faucibus vehicula arcu quis laoreet. Nullam viverra, lacus id blandit blandit, urna elit mollis augue, vitae tempor neque sem quis lectus. Morbi sit amet tristique dolor. Fusce malesuada libero justo, ac interdum dui vestibulum ut. Aenean fringilla nulla et ante ultrices, in scelerisque ante pharetra. Donec luctus justo in orci dapibus, ut molestie nibh tempor.

Cras eu ex mattis metus finibus consectetur a in purus. Nunc vehicula imperdiet neque, in luctus ante. Donec velit dui, imperdiet sit amet vehicula at, tincidunt sodales quam. Nam eu placerat nunc, vel commodo metus. Suspendisse risus tortor, tincidunt ac fringilla sed, tincidunt non mi. Morbi condimentum massa nec velit pulvinar, eu sagittis urna venenatis. Aenean fermentum mattis erat nec tincidunt. Nunc in mi vitae leo elementum imperdiet vel ac erat. Sed placerat molestie elementum. Aliquam a massa eget ex facilisis posuere. Vivamus magna dui, lobortis sed nisl a, facilisis blandit leo. Sed pulvinar at nunc non sagittis. Nunc blandit volutpat nisl, eget efficitur felis bibendum id. Pellentesque et hendrerit ante. Pellentesque non aliquet est. Maecenas vehicula odio ac rhoncus congue.

Nunc quis orci dui. Duis eget lorem felis. Ut non tincidunt augue. Nulla sollicitudin laoreet euismod. Donec non imperdiet leo. Aliquam non quam et ipsum luctus vestibulum. Suspendisse sed orci non ipsum varius varius. Proin sed odio et diam sagittis bibendum. Nullam sollicitudin sem porttitor massa viverra, ut efficitur tellus mollis. Vestibulum ut felis orci. Quisque sapien libero, laoreet vel lobortis vel, mollis nec lectus. Mauris ullamcorper enim in fringilla commodo. Maecenas auctor eros vitae felis mollis pellentesque. Morbi hendrerit aliquam leo eu elementum.

COMMENTAIRE – *Proin vestibulum eros in fringilla pretium. Curabitur sit amet orci elit. Praesent nec sollicitudin dolor. Phasellus nec venenatis massa, non facilisis ante. Donec rhoncus libero vitae malesuada interdum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas aliquam metus id lacus consequat, lacinia placerat tortor finibus. Aenean eleifend auctor pharetra. Donec elit justo, vulputate eget blandit in, aliquam in nunc. Aliquam at elit risus. In sed ligula*

non ex bibendum viverra. Cras tincidunt feugiat ipsum, a dictum leo consectetur ut. Phasellus mollis massa vitae maximus mollis. Quisque a laoreet nunc. Ut tellus nisi, pharetra non ipsum id, molestie tempus ex. Donec suscipit urna sed arcu accumsan fringilla.

Étant donné un graphe orienté, un des problèmes en théorie des graphes consiste à trouver un chemin ou un cycle passant par tous les sommets.

1.3. Conclusion

Proin vestibulum eros in fringilla pretium. Curabitur sit amet orci elit. Praesent nec sollicitudin dolor. Phasellus nec venenatis massa, non facilisis ante. Donec rhoncus libero vitae malesuada interdum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas aliquam metus id lacus consequat, lacinia placerat tortor finibus. Aenean eleifend auctor pharetra. Donec elit justo, vulputate eget blandit in, aliquam in nunc. Aliquam at elit risus. In sed ligula non ex bibendum viverra. Cras tincidunt feugiat ipsum, a dictum leo consectetur ut. Phasellus mollis massa vitae maximus mollis. Quisque a laoreet nunc. Ut tellus nisi, pharetra non ipsum id, molestie tempus ex. Donec suscipit urna sed arcu accumsan fringilla. ?

1.4. Bibliography

Alberts, B. (2004). *Biologie moléculaire de la cellule*. Flammarion.

Bondy, J.A. and Murty, U.S.R. (1985). *Graph Theory with Applications*. North Holland.

Cook, S.A. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, Shaker Heights, Ohio, May 3–5, 151–158.

Cori, R. and Lascar, D. (2003). *Logique mathématique 1 – Calcul propositionnel ; algèbre de Boole ; calcul des prédicats*. Dunod.

Crochemore, M., Hancart, C., Lecroq, T. (2001). *Algorithmique du texte*. Vuibert.

Diestel, R. (2017). *Graph Theory, Graduate Texts in Mathematics*, 5th edition. Springer.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8, 128–140.

Cette bibliographie est identique à celle de l'ouvrage correspondant en anglais publié par ISTE.

Garey, M.R. and Johnson, D.S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co.

Heather, J.M. and Chain, B. (2016). The sequence of sequencers: The history of sequencing DNA. *Genomics*, 107(1), 1–8.

Held, M. and Karp, R.M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1), 196–210.

Ladner, R.E. (1975). On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1), 155–171.

Letunic, I. and Bork, P. (2017). 20 years of the SMART protein domain annotation resource. *Nucleic Acids Research*, 46(D1), D493–D496.

Lewis, H.R. and Papadimitriou, C.H. (1981). *Elements of the Theory of Computation*. Prentice-Hall.

Petsko, G.A., Ringe, D., Sanlaville, C. (2009). *Structure et fonction des protéines*. De Boeck.

Planck Collaboration et al. (2018). Planck 2018 results VI. cosmological parameters. Cosmology and Nongalactic Astrophysics, arXiv:1807.06209.

Robertson, N. and Seymour, P.D. (1986). Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3), 309–322.

2

Titre Chapitre

Jean DUPONT[1] et Sophie MARTIN[2]
[1] *LITIS, UNIROUEN, Normandie Université, Rouen, France*
[2] *CRIStAL, CNRS, Université de Lille, Lille, France*

2.1. Premier niveau d'intertitre

Le développement de l'informatique et de ses capacités de calcul a provoqué depuis des décennies une croissance exponentielle de la quantité de données produite quotidiennement par des domaines allant de la climatologie à la géographie, en passant par la médecine et la sociologie (Mayer-Schönberger et Cukier 2013). On parle de déluge de données. C'est également le cas en biologie moléculaire où les séquenceurs de deuxième et maintenant de troisième générations permettent d'obtenir des séquences génomiques et transcriptomiques à haut débit et à des coûts relativement faibles (Schatz et Langmead 2013 ; Goodwin *et al*. 2016). Les séquences ainsi produites n'apportent en soi aucune information si elles ne sont pas remises en contexte et, en l'occurrence, comparées entre elles ou à des séquences déjà connues. Savoir, à partir des séquences obtenues, si un gène est surexprimé chez certains individus par rapport à d'autres nécessite de comparer des séquences pour identifier celles qui correspondent au gène d'intérêt. Plutôt que de se concentrer sur un gène particulier, les recherches se font plus souvent sans trop d'*a priori*, nécessitant de comparer chaque séquence avec la totalité d'un ou plusieurs génomes. De telles comparaisons entre séquences ne se restreignent pas à l'exemple donné mais sont nécessaires dans bien d'autres situations (étude de l'évolution phylogénétique, assemblage de génomes, comparaison de métagénomes, etc.).

Titre ouvrage,
coordonné par Jean DUPONT et Sophie MARTIN.

Si de telles comparaisons étaient réalisées de manière naïve, il faudrait comparer des milliards de séquences produites par le séquenceur à haut débit aux milliards de positions d'un seul génome. Pour éviter une telle explosion du nombre de comparaisons, il est indispensable d'effectuer un traitement permettant d'accélérer cette recherche. Cette optimisation s'appuie sur l'utilisation d'un index.

2.1.1. *L'indexation*

Dans notre vie quotidienne, nous pouvons utiliser un index pour localiser des mots dans un livre, comme dans celui-ci par exemple. Cet index ne contient que des termes déterminés à l'avance et ne réfère qu'une partie des occurrences de ces termes. Pour les séquences biologiques, on utilise également des index qui, contrairement aux index à la fin des livres, permettent en général d'accéder à la totalité des séquences et à toutes leurs occurrences. En raison de la taille des séquences biologiques à indexer (par exemple des génomes), ces index doivent être suffisamment économes pour pouvoir être utilisés sur des machines de calcul. Alors que dans les années 2010 les comparaisons se faisaient la grande majorité du temps contre une seule séquence de référence, avec la diversité des séquences dont nous disposons désormais, il devient nécessaire d'être en mesure d'indexer plusieurs séquences de référence (potentiellement des milliers, voire des millions).

De manière plus générale, l'indexation d'un ensemble de données consiste à construire une structure de données qui accélère la recherche d'informations particulières dans cet ensemble. La structure de données une fois construite va éviter un parcours séquentiel des données de l'ensemble lors de chaque recherche d'informations. Ainsi, le temps de la recherche pourra être, dans le meilleur des cas, proportionnel à la taille des informations recherchées et non pas à la taille de l'ensemble des données indexées. En raison du temps nécessaire à la construction d'une structure indexant les données, une telle approche n'est rentable que lorsque ce temps est largement compensé par le temps économisé lors des recherches. Autrement dit, l'indexation est envisageable, voire souhaitable, lorsque l'ensemble de données indexé varie rarement, voire pas du tout.

2.1.2. *Quand indexer ?*

Les séquences génomiques disponibles dans les banques de données publiques, comme la séquence du génome humain, sont des données variant assez rarement. À titre d'exemple, la version du génome humain de référence disponible dans les banques de données en 2019 était GRCh38 (NCBI 2019). La version précédente, GRCh37, datait de 2013 (NCBI 2013). Même en prenant en compte les modifications ponctuelles introduites sur GRCh38 (les *patches*), il n'y en a eu que 12 entre

2013 et 2019. Dans une telle situation, la création d'une structure d'indexation sur un génome humain sera largement rentabilisée par les milliards de recherches qui seront réalisées dans cette structure, avant que celle-ci ne doive être recréée sur une nouvelle version du génome.

Les structures d'indexation présentées dans ce chapitre permettent aisément de chercher des occurrences exactes de motifs. Elles constituent aussi des structures fondamentales pour la recherche de motifs approchés, autorisant des différences entre la séquence recherchée et les occurrences trouvées. Dans ce dernier cas, elles nécessitent des méthodes additionnelles dont certaines seront détaillées dans le chapitre suivant sur l'alignement de séquences.

2.1.3. *Qu'indexer ?*

L'indexation intéresse de nombreux domaines s'appuyant sur de grandes quantités de données. Les bases de données, les moteurs de recherche internet s'appuient sur des structures d'indexation afin de proposer des résultats rapides. Néanmoins, les données indexées en biologie ne sont pas forcément comparables à celles de ces domaines et donc les techniques d'indexation peuvent différer.

En bioinformatique, la spécificité est d'indexer de très grands textes peu ou pas structurés, à l'inverse des textes en langue naturelle dans lesquels on peut distinguer des mots, des paragraphes, des titres, etc.

Deux grands types de problèmes bioinformatiques ont historiquement eu largement recours à l'indexation : le positionnement des lectures issues du séquençage sur un génome de référence et l'assemblage des lectures pour reconstituer la séquence d'un génome.

Dans le problème du positionnement des lectures sur un génome de référence, on doit généralement positionner quelques millions de lectures sur un génome contenant quelques millions à quelques milliards de bases.

Pour permettre l'assemblage de lectures, de nombreuses méthodes reposent sur l'indexation des lectures afin de comparer les lectures entre elles et trouver efficacement les chevauchements pertinents entre ces lectures.

Bien entendu, l'utilisation de l'indexation ne se restreint pas à ces problèmes-là, ni sous cette forme-là. Le positionnement des lectures sur un génome peut aussi se faire en indexant les lectures plutôt que le génome, afin de tirer parti de la redondance des lectures. L'indexation du jeu de données de lectures facilite leur comparaison afin de déterminer les jeux de données les plus communs.

2.1.4. *Structures d'indexation et requêtes considérées*

Classiquement, une structure d'indexation permettra d'obtenir une réponse à certains types de requêtes. Les requêtes classiques, de la plus simple à la plus complexe, sont l'existence, le comptage et la localisation, qui peuvent être synthétisées dans les trois problèmes suivants.

Problème 2.1. Existence d'un élément dans des données indexées

Entrée : un élément et des données indexées.
Question : l'élément est-il présent dans les données indexées ?

Problème 2.2. Comptage d'un élément dans des données indexées

Entrée : un élément et des données indexées.
Sortie : nombre d'occurrences de l'élément dans les données indexées.

Problème 2.3. Localisation d'un élément dans des données indexées

Entrée : un élément et des données indexées.
Sortie : liste des emplacements de l'élément recherché dans les données indexées ?

L'objet de ce chapitre sera de présenter les méthodes couramment utilisées pour répondre à ces problèmes avec des séquences biologiques. Puisque les séquences biologiques à indexer peuvent être de différentes natures (un génome, un ensemble de génomes, un ou plusieurs jeux de données de séquençage, etc.), nous généraliserons le propos en considérant un ensemble de *séquences* à indexer sur un alphabet de quatre lettres. Dans un premier temps, nous allons présenter des structures permettant d'indexer les facteurs, d'une longueur fixe, des mots d'un ensemble : filtres de Bloom, listes inversées, tables de correspondance, tables de hachage et graphe de De Bruijn. Dans un deuxième temps, nous présenterons des structures d'indexation plein texte, qui ne sont pas restreintes à des mots de longueur fixe : arbres des suffixes, tables des suffixes et transformées de Burrows-Wheeler. Ensuite, nous énoncerons des critères pour choisir une solution d'indexation en fonction des données disponibles et des recherches à effectuer.

2.1.5. *Notions de base et vocabulaire*

Les structures d'indexation ont recours à certains concepts que nous commençons par définir. Nous introduisons également quelques notations (les trois premières sont

également détaillées dans le chapitre **??**) : Ces éléments seront utilisés dans le reste du chapitre.

- *séquence* : une *séquence* (aussi appelée texte) est une suite de symboles issus d'un alphabet, qui est un ensemble fini de symboles. Une séquence x de longueur n est notée $x = x[1\,..\,n]$. Les symboles sont indexés à partir de la position 1. La longueur de la séquence x est notée $|x|$. Un fragment d'une séquence est aussi appelé un *facteur*. Le facteur de la séquence x commençant à la position i et se terminant à la position j est noté $x[i\,..\,j]$ pour $1 \leq i \leq j \leq n$. Un facteur commençant au début de la séquence et donc de la forme $x[1\,..\,j]$ est appelé un *préfixe* et un facteur se terminant à la fin de la séquence et donc de la forme $x[i\,..\,n]$ est appelé un *suffixe* ;

- *graphe* : un *graphe* $G = (V, E)$ est composé d'un ensemble fini V de *sommets* et d'un d'ensemble d'*arêtes* E reliant ces sommets. Ces arêtes sont appelées des *arcs* dans le cas d'un graphe orienté. Ces arêtes peuvent être étiquetées ;

- *arbre* : un *arbre* (enraciné) est un graphe acyclique orienté. Les sommets sont appelés des *nœuds* et les arcs des *branches*. Un arbre non vide possède un unique nœud sans prédécesseur appelé *racine*. Les nœuds sans successeur sont appelés des *feuilles* ;

- $rank$ *et* $select$: soit T un tableau de n bits avec les deux opérations suivantes, où $b \in \{0, 1\}$:

  - $rank_b(T, i)$, qui retourne le nombre de bits b dans T jusqu'à la position i, soit $rank_b(T, i) = |\{j \mid T[j] = b \text{ et } j \leq i\}|$;
  - $select_b(T, i)$, qui retourne la position du i^{e} bit b dans T, soit $select_b(T, i) = \min\{j \mid rank_b(T, j) = i\}$.

  Par exemple, dans le tableau de bits $T = \overset{1\ 2\ 3\ 4\ 5\ 6\ 7}{0\ 1\ 1\ 0\ 1\ 1\ 0}$, il y a deux 0 jusqu'à la position 5, donc $rank_0(T, 5) = 2$, et trois 1 jusqu'à la position 5, donc $rank_1(T, 5) = 3$. D'autre part, le troisième 0 est en position 7 ($select_0(T, 3) = 7$) tandis que le troisième 1 est en position 5 ($select_1(T, 3) = 5$).

  Ces opérations peuvent être calculées en temps constant en utilisant $o(m)$ bits, en plus de T qui peut éventuellement être compressé (Navarro et Mäkinen 2007).

  Notons qu'une telle structure de données est une structure d'indexation qui, ici, indexe un tableau de bits. L'opération $rank$ est une opération de *comptage* alors que $select$ est une opération de *localisation*.

2.2. Indexation de mots

Nous commençons par étudier les structures indexant uniquement des mots d'une longueur fixe. L'origine de ces mots de longueur fixe peut être diverse : ils peuvent aussi bien provenir de génomes, de transcriptomes que de lectures de séquençage, etc. Afin de généraliser ces différents cas de figure, nous allons considérer l'indexation de tous les éléments d'un ensemble E de n mots de longueur fixe et possiblement le stockage des informations relatives à chaque mot. La longueur fixe de chacun de ces mots sera notée k et nous appelons ces mots des k-mers.

Tout au long de cette section, nous prendrons pour exemple l'indexation de quatre séquences de longueurs différentes, soit $S = \{\texttt{ACTCGA}, \texttt{TCGAT}, \texttt{CGATC}, \texttt{TCGC}\}$. Une fois décomposées sous forme de 3-mers, les séquences de S nous donnent un ensemble de 7 éléments :

$$E = \{\texttt{ACT}, \texttt{ATC}, \texttt{CGA}, \texttt{CGC}, \texttt{CTC}, \texttt{GAT}, \texttt{TCG}\}$$

2.2.1. *Filtre de Bloom*

Un filtre de Bloom (1970) est une structure de données probabiliste permettant de répondre à l'existence de k-mers dans l'ensemble E (problème **??**). La structure est dite *probabiliste* car il existe une certaine probabilité qu'une requête retourne une réponse fausse (un faux positif, voir section 1.9.1). Un filtre de Bloom n'est constitué que d'un vecteur de bits. Le stockage d'un élément ne se fait qu'en mettant des 1 aux positions indiquées par le résultat de plusieurs fonctions de hachage sur cet élément. Plus formellement, le filtre se compose d'un tableau T de p bits et d'un ensemble de h fonctions de hachage $f_i : E \mapsto [1, p]$ pour $1 \leq i \leq h$. Pour indiquer qu'un élément e appartient à E, on fixe à 1 les h bits aux positions $f_i(e)$ de T.

Pour tester si un élément e appartient à E, il suffit d'accéder aux h bits aux positions $f_i(e)$: si un d'entre eux est à 0, alors e n'est pas dans E, et s'ils sont tous à 1, alors il est *probable* que e soit dans E.

Prenons un exemple avec 10 bits et 2 fonctions de hachage (d'où $p = 10$ et $h = 2$) (voir figure 2.1). Définissons une fonction $rang$ qui associe un entier entre 0 et 3 à chaque nucléotide dans l'ordre alphabétique : $rang(\texttt{A}) = 0$, $rang(\texttt{C}) = 1$, $rang(\texttt{G}) = 2$ et $rang(\texttt{T}) = 3$. Soit f_1 et f_2 nos deux fonctions de hachage. La fonction de hachage f_1 consiste à sommer le $rang$ de chaque nucléotide d'un k-mer, modulo 10, et f_2 correspond à la valeur du k-mer en base 4, modulo 10. Plus formellement, $f_1 : A^3 \mapsto \{1, \ldots, 10\}$ définie comme suit $f_1(x) = (rang(x[1]) + rang(x[2]) + rang(x[3])) \bmod 10 + 1$ et $f_2 : A^3 \mapsto \{1, \ldots, 10\}$ définie comme suit $f_2(x) = (rang(x[1]) \times 16 + rang(x[2]) \times 4 + rang(x[3])) \bmod 10 + 1$.

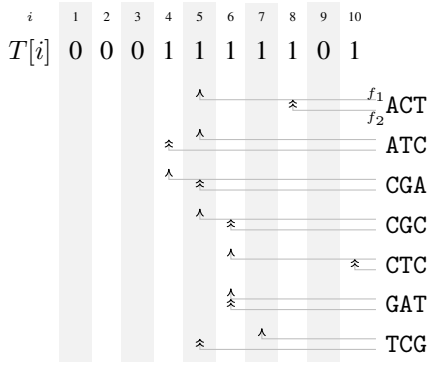


Figure 2.1. *Stockage de 3-mers avec un filtre de Bloom T de 10 bits et 2 fonctions de hachage (f_1 et f_2). L'application des fonctions f_1 (↟) et f_2 (↠) sur chacun des k-mers va indiquer les positions auxquelles stocker un 1 dans T*

Dans le filtre de Bloom défini à la figure 2.1, on sait que `CAC` n'appartient pas à E car $f_1(\texttt{CAC}) = 3$ et $T[3] = 0$. Par contre, pour `TCC`, $f_1(\texttt{TCC}) = 6$ et $f_2(\texttt{TCC}) = 4$ et $T[4] = T[6] = 1$. On en conclut que `TCC` appartient probablement à E. Or ce n'est pas le cas. Il s'agit d'un exemple de faux positifs dû à l'insertion de certains k-mers qui ont ajouté des 1 en positions 4 et 6.

Les fonctions de hachage utilisées dans cet exemple sont de mauvais choix dans le cas général. Il est préférable d'utiliser des fonctions distribuant uniformément les mots indexés dans les p positions de T, indépendamment de l'ordre lexicographique (Marçais *et al*. 2017).

Le taux de faux positifs d'un filtre de Bloom dépend des trois paramètres $|E|$, $n = |T|$ et h. Pour conclure, à tort, qu'un élément $e \notin E$ appartient à E, il faut que tous les bits aux positions $f_i(e)$ vaillent 1. La probabilité qu'un bit quelconque de T vaille 1 est $1 - (1 - \frac{1}{n})^{h|E|}$, car il y a une probabilité $(1 - \frac{1}{n})^{h|E|}$ qu'un bit de T vaille 0 après avoir inséré $|E|$ éléments avec h fonctions de hachage. Alors la probabilité d'avoir h bits à 1 aux positions $f_i(e)$, autrement dit le taux de faux positifs, est $\left(1 - \left(1 - \frac{1}{n}\right)^{h|E|}\right)^h$ (Broder et Mitzenmacher 2004).

Le taux de faux positifs est donc déterminé en fonction des choix du nombre de fonctions de hachage et de la taille du filtre. Cela rend ces filtres particulièrement souples et économes en espace. Bien entendu, la contrepartie est d'avoir un certain taux de faux positifs. D'autre part, les filtres de Bloom permettent de ne répondre (de manière probabiliste) qu'à la requête d'existence. Il est impossible avec un simple

filtre de Bloom de compter le nombre d'occurrences et, *a fortiori*, de localiser les occurrences même s'il existe des extensions des filtres de Bloom qui lèvent certaines de ces limites (voir par exemple (Luo *et al*. 2019)). D'autres méthodes existent pour stocker exactement l'ensemble E des k-mers d'intérêt, ce qui permet de les interroger avec ces différentes requêtes.

2.2.2. *Liste inversée*

Une liste inversée consiste à associer à un mot de E une liste des positions auxquelles ce mot apparaît dans S. L'indexation de mots par liste inversée est la méthode la plus commune, y compris pour des textes en langue naturelle. Par exemple, les index qui peuvent être présents à la fin des livres sont des listes inversées.

Les listes inversées peuvent être stockées de diverses manières. Deux sont principalement utilisées en bioinformatique : les tables de correspondance, utilisées pour de petits k-mers, et les tables de hachage.

2.2.2.1. *Table de correspondance*

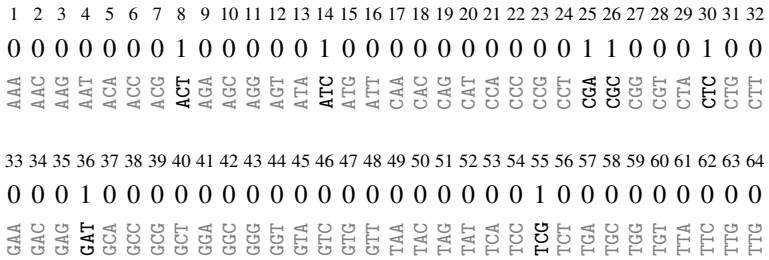
Une table de correspondance pour des k-mers d'un ensemble E est une table contenant σ^k cellules, où σ est la taille de l'alphabet (soit 4 classiquement pour l'ADN ou l'ARN). Les k-mers sont transformés par une fonction bijective en un entier compris entre 1 et σ^k. Cet entier est l'indice dans la table de correspondance où seront stockées les informations relatives au k-mer. La transformation des k-mers en entier se fait le plus souvent en considérant le k-mer comme étant l'écriture d'un nombre en base σ (où chaque lettre du k-mer est donc associée à un chiffre), comme pour la fonction $rang$ utilisée pour les filtres de Bloom (voir section **??**).

Les tables de correspondance sont notamment utilisées pour stocker la présence ou l'absence d'un k-mer dans E, ce qui permet de répondre au problème **??**. Il suffit donc d'un bit d'information par k-mer. Autrement, les tables de correspondance peuvent aussi stocker le nombre d'occurrences de chaque k-mer dans S, ce qui permet de répondre au problème **??**. Généralement, ces tables ne sont pas utilisées pour stocker toutes les positions auxquelles les k-mers apparaissent dans S. Dans ce cas, les tables de hachage sont privilégiées.

Par exemple, pour stocker les sept 3-mers de notre ensemble E, la table de correspondance serait composée de $4^3 = 64$ bits (voir figure 2.2).

2.2.2.2. *Table de hachage*

Une table de hachage permet de stocker les éléments de l'ensemble E dans un tableau T de N cases en utilisant une fonction de hachage $f : E \mapsto [1, N]$. Un



$$E = \{\texttt{ACT}, \texttt{ATC}, \texttt{CGA}, \texttt{CGC}, \texttt{CTC}, \texttt{GAT}, \texttt{TCG}\}.$$

Figure 2.2. *Stockage de 3-mers dans une table de correspondance. À titre d'information, nous affichons les k-mers correspondant à chacune des positions de la table de correspondance mais ils ne sont pas réellement stockés. Sont indiqués en noir les k-mers pour lesquels un 1 est stocké dans la table de correspondance.*

élément $e \in E$ est stocké dans $T[f(e)]$. La fonction f est en général non injective. Si deux éléments distincts e et e' de E sont tels que $f(e) = f(e')$ alors on dit qu'il y a *collision*. Dans une table de correspondance, il n'était pas possible d'avoir une collision puisque chaque élément potentiellement dans E possédait une case qui lui était propre.

Il existe plusieurs méthodes pour gérer les collisions dans les tables de hachage (Cormen *et al*. 2022). Le *hachage ouvert*, ou *adressage fermé*, consiste à utiliser des listes chaînées pour stocker les valeurs dans les cases du tableau T. Les listes chaînées sont des structures de données dans lesquelles chaque « case » stockant une valeur pointe également vers la case suivante (si elle existe). Cela permet d'avoir une liste qui peut grossir arbitrairement sans avoir de limite aussi fixe qu'avec un tableau. Ainsi, une table de hachage à adressage fermé n'a pas une taille limite fixée à la construction (d'où le nom de hachage ouvert), et l'adresse d'un élément est toujours le même (adressage fermé).

La figure 2.3 présente un exemple du stockage de E dans une table de hachage avec la fonction f_1 préalablement introduite (voir figure **??**). Pour savoir si `CGC` est présent dans la table de hachage, nous commençons par calculer $f_1(\texttt{CGC})$ qui vaut 5. Ensuite, nous parcourons les différents k-mers présents à cette position afin de vérifier que `CGC` s'y trouve, ce qui est le cas.

Si on recherche `TCC`, on a $f_1(\texttt{TCC}) = 5$. En parcourant la liste des k-mers stockés à cette position, on se rend compte que `TCC` n'y est pas. On en conclut que $\texttt{TCC} \notin E$.

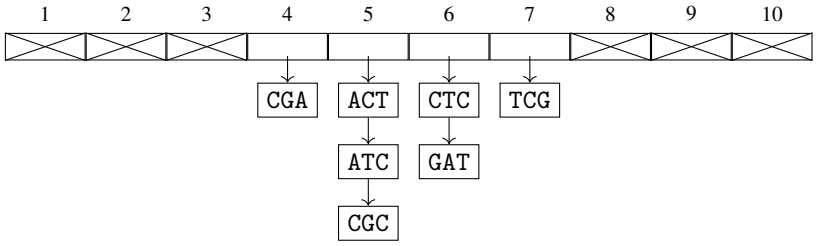


Figure 2.3. *Stockage de 3-mers dans une table de hachage avec adressage fermé. La fonction de hachage utilisée f est f_1, telle que définie dans la section* **??** *pour stocker E dans une table de hachage T de 10 cases*

Contrairement à la table de correspondance, l'espace requis pour une telle table de hachage n'est pas proportionnel au nombre de k-mers d'une taille donnée, mais principalement au nombre de k-mers distincts présents dans un jeu de données, ici dans E. En revanche, une telle table de hachage a deux inconvénients : d'une part, il est nécessaire de stocker le k-mer lui-même et, d'autre part, pour vérifier si un k-mer est présent à une position donnée, il est nécessaire de le rechercher dans la liste à cette position.

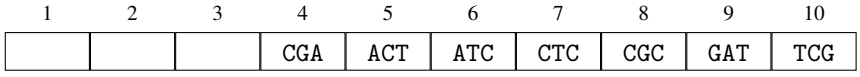


Figure 2.4. *Stockage des 3-mers [`ACT`, `ATC`, `CGA`, `CTC`, `CGC`, `GAT`, `TCG`], dans l'ordre, dans une table de hachage avec adressage ouvert*

COMMENTAIRE SUR LA FIGURE 2.4.– *La fonction de hachage utilisée f est f_1, telle que définie dans la section* **??** *pour stocker E dans une table de hachage de 10 cases. Par exemple, $f_1(\texttt{ATC}) = 5$, pourtant `ATC` est stocké en position 6 car `ACT` a été préalablement stocké en position 5. On cherche alors la position suivante qui est libre (il s'agit de la position 6).*

Une autre solution est le *hachage fermé*, ou *adressage ouvert*. Dans ce cas, la taille de la structure de données est fixée à la construction : un nombre fixe d'éléments pourra être stocké. Dans le cas d'une collision, il faudra trouver de manière déterministe un autre emplacement libre dans la structure de données. Cette recherche d'un emplacement libre peut se faire de manière linéaire (on essaie les emplacements qui suivent directement l'emplacement d'origine, voir figure 2.4) ou, plus généralement,

dépendant d'une fonction g qui donne la position à tenter. Avec une telle structure, un élément donné pourra être inséré à des positions très différentes selon les éléments qui auront été insérés préalablement. Avec une table de hachage à adressage ouvert, la table doit nécessairement être de taille strictement supérieure au nombre d'éléments à insérer, afin d'éviter trop de collisions et d'avoir à chercher l'emplacement où se trouve un élément. Une table à adressage fermé n'a pas cet inconvénient.

2.2.2.2.1. Fonction de hachage minimale parfaite

Une fonction de hachage minimale parfaite f permet de stocker les n éléments de l'ensemble E dans une table à n emplacements sans collision. La fonction f est donc telle que $1 \leq f(e) \leq n$ et $f(e) = f(e') \Leftrightarrow e = e'$.

L'utilisation d'une fonction de hachage minimale parfaite permet de ne pas avoir à gérer de collisions, comme dans une table de correspondance, et il suffit de réserver le nombre nécessaire d'emplacements en mémoire, à l'inverse d'une table de correspondance.

Le calcul et l'utilisation d'une fonction de hachage minimale parfaite nécessite de l'espace mémoire. En effet, une telle fonction repose sur un précalcul effectué sur l'ensemble des données qui doit donc être connu à l'avance. Une borne minimale d'environ 1,44 bits par élément est établie (Belazzougui *et al*. 2009).

Une fonction de hachage minimale parfaite permet donc, en quelque sorte, de bénéficier du meilleur des mondes avec un coût modique. En revanche, l'ensemble E doit être connu au départ. Il n'est pas possible, dans l'état actuel des connaissances, par exemple d'ajouter de nouveaux éléments à une fonction de hachage minimale parfaite.

2.2.3. *Graphes de De Bruijn*

Les graphes de De Bruijn ont été introduits dans un cadre théorique dans (Sainte-Marie 1894 ; Good 1946 ; de Bruijn 1950). Ils représentent des k-mers, ainsi que leurs chevauchements de longueur $k-1$. Dans un tel graphe, les sommets du graphe sont les k-mers.

Étant donnés un ensemble de n mots $S = \{w_1, \ldots, w_n\}$ et un entier k, le graphe de De Bruijn d'ordre k de S est un graphe orienté dont les sommets sont donc les k-mers des mots de S. Un arc existe entre un sommet u et un sommet v lorsque le suffixe de longueur $k-1$ de u est égal au préfixe de longueur $k-1$ de v, c'est-à-dire lorsqu'il y a un chevauchement de longueur $k-1$ entre la fin de u et le début de v.

Il est possible de restreindre cette définition pour que l'existence d'un arc soit conditionnée par le fait que le mot $u \cdot v[k]$, de longueur $k+1$, soit facteur d'un mot

de l'ensemble de départ dont sont issus les k-mers. Dans ce cas, il est également possible de pondérer les arcs par le nombre d'occurrences de $u \cdot v[k]$ dans les mots de l'ensemble de départ.

Un tel graphe donne accès à une information qu'il était difficile d'obtenir avec les structures précédentes. En suivant un chemin dans le graphe, il est possible d'énumérer les k-mers chevauchants. Cette structure est particulièrement utilisée pour l'assemblage de séquences (voir chapitre **??**) mais également pour détecter des événements biologiques, comme des mutations ponctuelles ou des événements d'épissage alternatif, directement à partir de la structure du graphe (Iqbal *et al*. 2012 ; Sacomoto *et al*. 2012).

Pour un exemple, le graphe de De Bruijn d'ordre 3 de S est donné en figure 2.5.

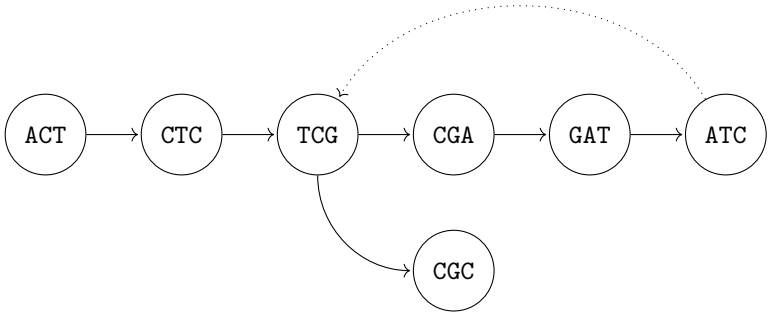


Figure 2.5. *Graphe de De Bruijn d'ordre 3 de* $S = \{\texttt{ACTCGA}, \texttt{TCGAT}, \texttt{CGATC}, \texttt{TCGC}\}$

COMMENTAIRE SUR LA FIGURE 2.5.– *Chaque sommet du graphe correspond à un élément de S. Chaque arête pleine correspond à un 4-mers qui existe dans S. Par exemple, il existe une arête de `TCG` vers `CGA` car le suffixe de `TCG` de longueur 2 est égal au préfixe de même longueur de `CGA`, c'est-à-dire à `CG`, et que `TCGA` apparaît dans le premier mot de S. L'arête en pointillé correspond à un 4-mers qui n'existe pas dans S (`ATCG`), mais pour lequel le suffixe de taille 2 du sommet d'origine est identique au préfixe de taille 2 du sommet d'arrivée (`TC`). Selon la définition du graphe de De Bruijn utilisée, cette arête pourra exister ou non.*

Les graphes de De Bruijn sont une manière de représenter des chevauchements entre k-mers et sont donc une manière de garder une information un peu plus complète sur les séquences indexées.

Cette représentation sous forme de graphe est une structure de données abstraite, qui peut être stockée avec les structures présentées dans ce chapitre. Une solution efficace est d'utiliser un filtre de Bloom. Le filtre stocke les k-mers, les chevauchements n'ont pas nécessairement besoin d'être stockés mais peuvent être déterminés en effectuant une requête pour chaque chevauchement possible (il n'y a que quatre possibilités avec l'ADN). Les filtres de Bloom ayant des faux positifs, on peut les éviter en stockant explicitement les k-mers faux positifs accessibles depuis un sommet du graphe (Chikhi et Rizk 2013). D'autres solutions sont détaillées dans une synthèse de (Chikhi *et al*. 2019).

2.2.4. *Des structures efficaces pour des requêtes ciblées*

Les structures précédentes sont efficaces en termes d'espace requis et permettent de requêter des mots dont la longueur est choisie à l'avance. Requêter un mot d'une autre longueur n'est pas possible à moins de reconstruire une structure dédiée à cette longueur-là. Ces structures sont donc adaptées à des requêtes ciblées, dont la longueur est fixée. Dans le cas où la longueur des requêtes ne peut être fixée à l'avance, il faut se reporter sur des structures indexant la totalité du texte.

2.3. Indexation plein texte

Contrairement à la section précédente, nous allons maintenant considérer le texte dans sa globalité et non pas découpé en k-mers. Cela ouvre de nouvelles possibilités en termes de requêtes : peuvent être recherchées des séquences de longueur arbitraire. Néanmoins, un tel bénéfice ne vient pas sans inconvénient et de telles structures sont généralement plus gourmandes en mémoire. Nous verrons dans cette section comment sont définies de telles structures et comment les utiliser.

L'indexation plein texte repose, en tout cas pour les structures présentées ici, sur l'indexation de tous les *suffixes* d'un texte. En effet, ces structures s'appuient sur ce principe : accéder aux préfixes de ces suffixes permet d'accéder à tous les facteurs du texte. Or rechercher une chaîne dans un texte revient bien à chercher si celle-ci est facteur du texte. Ces structures d'indexation plein texte permettent donc d'accéder facilement aux préfixes de tous les suffixes d'un texte.

2.3.1. *Arbre des suffixes*

Une solution intuitive pour accéder à tous les suffixes d'un texte est de stocker chacun d'entre eux dans un arbre, plus précisément un *trie*. Il s'agit d'un arbre dans lequel tout chemin de la racine à une feuille représente une séquence stockée, ici un

suffixe. Ainsi, avec un tel *trie*, tout chemin partant de la racine correspond à un préfixe d'un ou plusieurs suffixes, et donc à un facteur quelconque. À partir de la racine, il est donc possible d'accéder à tout facteur d'un texte.

L'arbre des suffixes $\mathcal{S}(y)$ d'une séquence y de longueur n est un *trie compact* représentant tous les suffixes de y. Il est défini par les propriétés suivantes :

- les branches de $\mathcal{S}(y)$ sont étiquetées par des séquences ;
- les branches sortantes d'un nœud interne sont étiquetées par des séquences commençant par des lettres différentes ;
- tout chemin de $\mathcal{S}(y)$ de la racine à une feuille est étiqueté par un suffixe de y ;
- les nœuds internes de $\mathcal{S}(y)$ ont au moins deux descendants, ce qui en fait un trie compact ;
- les séquences qui étiquettent les branches sont représentées par des couples (position de début dans y, longueur), ce qui permet de stocker chaque étiquette dans un espace constant.

De plus, si on ajoute un symbole terminateur \$ qui n'apparaît qu'à la fin de y, les suffixes de y correspondent aux feuilles de $\mathcal{S}(y)$. Cela implique que $\mathcal{S}(y)$ possède exactement $n+1$ feuilles, donc $O(n)$ nœuds et $O(n)$ branches. Puisque chaque étiquette de branche occupe un espace constant, l'arbre des suffixes de y occupe donc un espace linéaire. La figure 2.6 montre un exemple d'arbre des suffixes.

COMMENTAIRE SUR LA FIGURE 2.6.– *Les nœuds internes sont représentés par des cercles (les lettres grecques qu'ils contiennent servent uniquement à les identifier) alors que les feuilles sont représentées par des carrés et contiennent la position de début des suffixes auxquels elles sont associées. Tout chemin de la racine à une feuille correspond à un suffixe du texte. Par exemple, le chemin de la racine à la feuille 5 correspond au suffixe qui commence en position 5, soit* `ATTAT$` *qu'on peut lire sur l'arbre avec l'étiquette (3,2) =* `AT` *suivie de l'étiquette (7, 4) =* `TAT$`*.*

Il existe plusieurs algorithmes permettant de construire l'arbre des suffixes d'une séquence en temps et en espace linéaires avec un alphabet de taille constante (McCreight 1976 ; Ukkonen 1995 ; Farach 1997). Cette structure peut être étendue à un ensemble fini de séquences et s'appelle alors l'arbre des suffixes généralisé.

Savoir si une séquence x apparaît dans une séquence y indexée avec un arbre des suffixes $\mathcal{S}(y)$ revient à utiliser $\mathcal{S}(y)$ comme un automate fini déterministe et donc à accéder aux caractères de x un par un et à descendre dans $\mathcal{S}(y)$ en partant de la

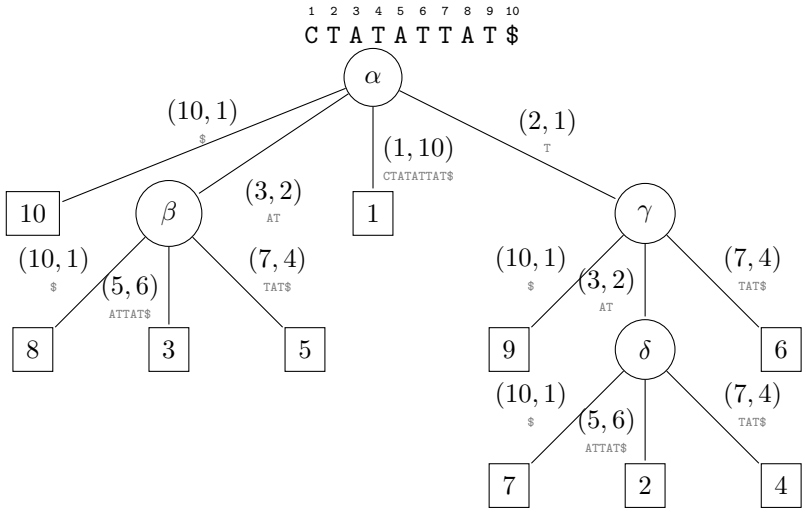


Figure 2.6. *L'arbre des suffixes de* `CTATATTAT$`

racine. Cela revient donc à accéder aux préfixes des suffixes du texte, qui sont stockés dans $\mathcal{S}(y)$. Si x peut être ainsi parcourue en entier alors elle apparaît dans y et les positions de toutes ses occurrences se trouvent dans les feuilles du sous-arbre enraciné au premier nœud (ou à la première feuille) au bout de la branche où s'est terminé le parcours de x.

Par exemple, recherchons `TA` dans l'arbre des suffixes de `CTATATTAT$` de la figure 2.6. La lecture de `T` en partant de la racine mène au nœud γ, puis la lecture de `A` mène sur la branche entre le nœud γ et le nœud δ. Donc `TA` apparaît dans `CTATATTAT$` aux positions 2, 4 et 7 contenues dans les feuilles du sous-arbre enraciné au nœud δ.

La recherche d'une séquence x de longueur m dans une séquence y peut donc se faire dans un temps $O(m)$ à partir de l'arbre des suffixes de y. Bien que linéaire en mémoire, une telle structure est néanmoins gourmande, la rendant finalement assez peu utilisée en pratique : les index présentés dans la suite, plus compacts, lui sont préférés.

2.3.2. *Table (étendue) des suffixes*

En lieu et place de stocker tout un arbre, il est possible de se contenter de l'information stockée dans les feuilles (les positions de début des suffixes) à condition qu'elles soient dans un ordre pertinent. La table des suffixes d'une séquence y correspond aux positions de début des suffixes de y triés dans l'ordre

lexicographique. Cette table des suffixes a été introduite indépendamment par deux groupes de chercheurs (Manber et Myers 1990 ; Baeza-Yates et Gonnet 1992).

Formellement, la table des suffixes de y, SA, est définie comme suit :

$$y[SA[1]\,..\,n] < y[SA[2]\,..\,n] < \cdots < y[SA[n]\,..\,n]$$

	1	2	3	4	5	6	7	8	9	10
	C	T	A	T	A	T	T	A	T	$

i	$SA[i]$	$LCP[i]$	$y[SA[i]\,..\,n]$
1	10		$
2	8	0	AT$
3	3	2	ATATTAT$
4	5	2	ATTAT$
5	1	0	CTATATTAT$
6	9	0	T$
7	7	1	TAT$
8	2	3	TATATTAT$
9	4	3	TATTAT$
10	6	1	TTAT$

Figure 2.7. *La table des suffixes de `CTATATTAT$` (colonne $SA[i]$)*

COMMENTAIRE SUR LA FIGURE 2.7.– *La colonne $LCP[i]$ donne la longueur des plus longs préfixes communs à deux suffixes consécutifs. Par exemple, $LCP[3] = 2$ car les suffixes en positions 3 et 5 ont un préfixe identifique de longueur 2. Les suffixes, en dernière colonne, ne sont pas réellement stockés. Les plus longs préfixes communs entre deux suffixes consécutifs de la table sont soulignés.*

L'information stockée dans la table des suffixes, bien que plus restreinte que dans l'arbre des suffixes, est malgré tout suffisante pour rechercher les occurrences d'une séquence quelconque.

La figure 2.7 montre un exemple de table des suffixes.

Il existe plusieurs algorithmes qui permettent de construire la table des suffixes d'une séquence en temps et en espace linéaires (Kärkkäinen and Sanders 2003 ; Kim *et al*. 2003 ; Ko et Aluru 2003). Néanmoins, en pratique, ces algorithmes linéaires et récursifs ne sont pas les plus efficaces. Des algorithmes ayant des complexités théoriques supralinéaires les dépassent, entre autres sur des séquences génomiques (Puglisi *et al*. 2007 ; Bahne *et al*. 2019). En particulier, l'algorithme

DivSufSort reste le meilleur algorithme de construction d'une table des suffixes en pratique depuis 2006, mais l'auteur a seulement publié le code source, il n'a pas publié d'article scientifique pour en décrire la méthode (Mori 2015).

La recherche des occurrences d'une séquence x de longueur m dans une séquence y de longueur n peut se faire à l'aide d'une recherche dichotomique dans la table des suffixes de y. La comparaison entre x et chacun des suffixes de la table pouvant nécessiter $O(m)$ comparaisons de symboles, cette recherche se fait en temps $O(m \times \log n)$.

Par exemple, effectuons la recherche de `TA` dans la table des suffixes de `CTATATTAT$` de la figure 2.7. La recherche, par dichotomie, commence en considérant l'ensemble de la table des suffixes. Afin de représenter cela, nous prendrons deux variables ℓ et r qui représentent respectivement la position de début et de fin de l'intervalle considéré dans la table des suffixes. Dans cet exemple, nous commençons avec les bornes $\ell = 1$ et $r = 10$. La position au milieu de l'intervalle étant égale à 5, on compare `TA` à $y[SA[5]\,..\,n] = y[1\,..\,10] =$ `CTATATTAT$`. `TA` est plus grand, les bornes deviennent donc $\ell = 5$ et $r = 11$. La position au milieu de l'intervalle est égale à 8 et `TA` est un préfixe de $y[SA[8]\,..\,n] = y[2\,..\,9] =$ `TATATTAT$`. Donc `TA` apparaît dans y à la position 2.

Il est possible de réduire la complexité de la recherche à $O(m + \log n)$ en utilisant la table des plus longs préfixes communs. Cette table est en deux parties. Les n premiers éléments correspondent aux longueurs des plus longs préfixes communs à deux suffixes consécutifs dans la table des suffixes. Les éléments suivants sont constitués par les longueurs des plus longs préfixes communs aux couples de suffixes qui peuvent être considérés lors de la recherche dichotomique dans la table des suffixes. Ces couples de positions ne dépendent pas du mot recherché. Formellement, $LCP[i] = lcp(y[SA[i-1]\,..\,n], y[SA[i]\,..\,n])$ pour $1 < i \leq n$, où $lcp(u, v)$ est la longueur du plus long préfixe commun aux séquences u et v. Et $LCP[n+(i+j)/2] = lcp(y[SA[i]\,..\,n], y[SA[j]\,..\,n])$ pour $1 \leq i, j \leq n$ et (i, j) est un couple qui apparaît dans la recherche dichotomique.

La première partie de la table LCP peut être calculée en temps et espace linéaires (Kasai *et al*. 2001) et la deuxième partie peut en être déduite par un simple parcours en profondeur de l'arbre (voir section **??**) de la recherche dichotomique à l'aide de l'observation suivante :

$$|lcp(y[SA[d]\,..\,n], y[SA[f]\,..\,n])|$$
$$=$$
$$\min\{|lcp(y[SA[d]\,..\,n], y[SA[i]\,..\,n])|, |lcp(y[SA[i]\,..\,n], y[SA[f]\,..\,n])|\}$$

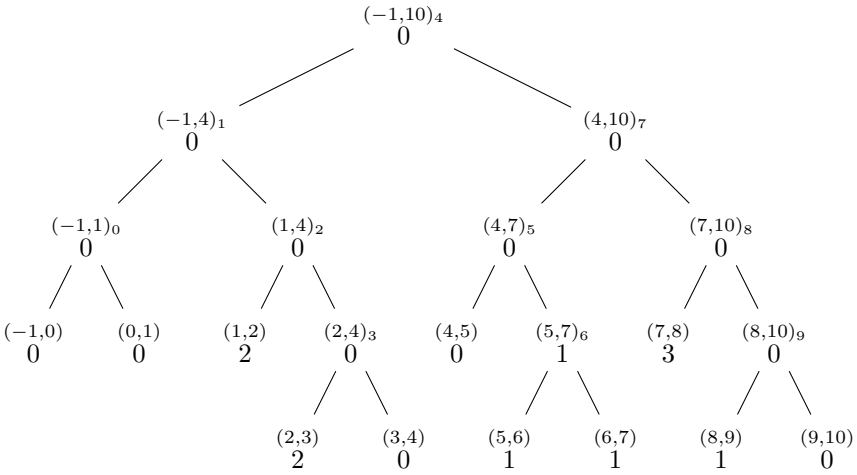


Figure 2.8. *L'arbre de la recherche dichotomique pour* `CTATATTAT$`. *Les feuilles* $(i-1,i)$ *correspondent aux premières valeurs de la table* $LCP[i]$ *pour* $1 \leq i \leq n-1$. *Les nœuds internes* $(i,j)_{\lfloor (i+j)/2 \rfloor}$ *correspondent à la deuxième partie et les longueurs des plus longs préfixes communs sont stockées dans* $LCP[n + \lfloor (i+j)/2 \rfloor]$.

pour $1 \leq d < i < f \leq n$.

La figure 2.8 montre un exemple d'arbre de recherche dichotomique. Celui-ci n'a pas besoin d'être stocké en tant que tel : la table LCP sert à le représenter. Les feuilles de l'arbre correspondent aux n premiers éléments de la table LCP. Quant aux nœuds, ils sont déduits des deux enfants : il s'agit du minimum des deux nœuds enfants.

La table LCP complète peut être utilisée lors de la recherche dichotomique pour conserver la longueur du plus long préfixe commun entre $y[\ell \,..\, n]$ et $y[r \,..\, n]$ (où ℓ et r sont les positions de début et fin de l'intervalle dans lequel s'opère la recherche dichotomique) de manière à obtenir une recherche avec une complexité en temps $O(m + \log n)$ (voir (Crochemore *et al*. 2007) pour les détails).

Adjoindre la table LCP à la table des suffixes permet de regagner des informations qui avaient été perdues en passant de l'arbre des suffixes à la table des suffixes. Néanmoins, cela rend la structure plus consommatrice en mémoire, le gain par rapport à un arbre des suffixes est donc plus modéré. La nécessité de construire ces structures d'indexation sur des séquences génomiques de plusieurs milliards de nucléotides rend nécessaire l'utilisation de structures plus compactes.

2.3.3. *Transformée de Burrows-Wheeler*

L'arbre et la table des suffixes ont l'avantage d'indexer l'intégralité du texte, permettant de rechercher des séquences de n'importe quelle longueur dans celui-ci. Néanmoins, un inconvénient majeur est la place qu'occupent ces structures. Elles ont besoin d'environ 10 fois plus de place que les structures de données n'indexant que des k-mers. Lorsqu'il s'agit d'indexer des séquences génomiques d'eucaryotes, cela peut devenir rédhibitoire.

Néanmoins, des chercheurs ont noté la grande proximité conceptuelle entre la table des suffixes et la transformée de Burrows-Wheeler. Cette transformée de Burrows-Wheeler est une permutation d'une séquence qui permet de favoriser le rapprochement de lettres identiques afin de mieux les compresser. Cette transformée a initialement été utilisée dans la compression de textes notamment dans l'utilitaire `bzip2`. Ferragina et Manzini ont montré comment on pouvait tirer parti de ces deux avantages de la transformée de Burrows-Wheeler en obtenant un index compressé (Ferragina et Manzini 2000).

Pour introduire la transformée de Burrows-Wheeler, nous allons d'abord présenter la notion de rotation cyclique d'un mot y de longueur n. Une *rotation cyclique* de y est un mot $v = y[i\,..\,n]y[1\,..\,i-1]$ avec $1 \leq i \leq n$.

Imaginons une matrice M_y formée par les n rotations cycliques de y triées dans l'ordre lexicographique. On note F_y et L_y les première et dernière colonnes de M_y. Alors, la transformée de Burrows-Wheeler (BWT) (Burrows et Wheeler 1994) de y correspond au couple (L_y, j) où $M_y[j] = y$. Autrement dit, la BWT de y est la dernière colonne de M_y et l'indice de la ligne dans M_y correspondant à y. En pratique, on ajoute à la fin du texte un terminateur \$, inférieur à toutes les lettres de l'alphabet. Cela permet de réduire la BWT à la dernière colonne de M_y : la première ligne de M_y correspond nécessairement à $\$y$. M_y n'est jamais réellement calculée ni stockée. Elle nous sert juste à introduire le concept de la BWT. Notons que M_y correspond au tri des rotations cycliques là où la table des suffixes est obtenue par un tri des suffixes, ce qui illustre la proximité entre les deux concepts.

La figure 2.9 montre un exemple de BWT.

La BWT de y est, en général, plus compressible que y car elle a tendance à regrouper les lettres identiques par paquets. De plus, elle est inversible, c'est-à-dire qu'il est possible de retrouver y à l'aide de sa BWT. Pour cela on utilise le fait que la BWT préserve la position respective d'une lettre dans les première et dernière colonnes.

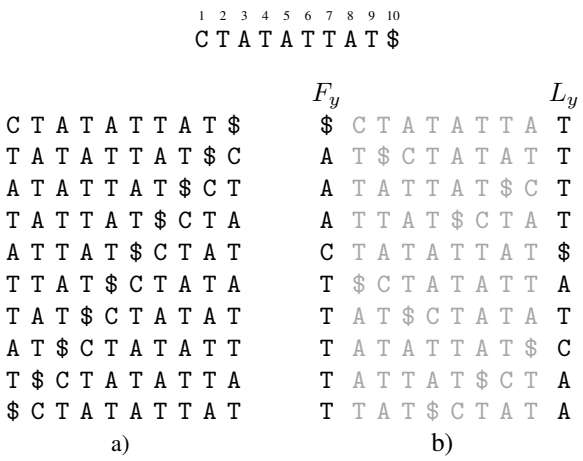


Figure 2.9. *BWT de $y =$ `CTATATTAT$` : a) rotations cycliques ; b) rotations cycliques triées. La BWT de y correspond à la dernière colonne L_y des rotations triées*

On peut tout d'abord remarquer que F_y peut être déduite de L_y puisque F_y est constituée par la suite triée des lettres de y et donc de L_y. Donc F_y peut être représentée par un tableau C défini par $C[c]$ égal au nombre de lettres strictement plus petites que c dans y pour $c \in A$.

On définit ensuite une fonction, appelée LF (pour *Last-First*), qui permet de passer d'une lettre dans la dernière colonne (L_y) à cette même lettre dans la première colonne (F_y). Par exemple, si $L_y[i] = c$ et qu'il s'agit du j^e c dans L_y jusqu'à la position i, alors $LF(i)$ donnera la position du j^e c dans F_y. La fonction est calculée comme suit : $LF(i) = C[c] + rank_c(L_y, i)$ où $rank_c(L_y, i)$ est le nombre total de c dans L_y jusqu'à la position i.

La figure 2.10 montre un exemple d'inversion de BWT utilisant la fonction LF.

Il est également possible de rechercher le nombre d'occurrences d'une séquence x de longueur m dans une séquence y, étant donnée la BWT L_y de y, grâce à une technique connue sous le nom de *backward search* (recherche en arrière). Le principe général de cette technique est d'identifier l'intervalle de M_y contenant toutes les rotations cycliques commençant par x. Cette recherche s'effectue en lisant les lettres de x de droite à gauche, d'où le nom de *backward search* (voir algorithme 2.1).

Le principe général de l'algorithme est, à partir d'un intervalle des positions ℓ à r de M_y, contenant toutes les occurrences de $x[i\,..\,|x|]$, de déduire l'intervalle qui contient toutes les occurrences de $x[i-1\,..\,|x|]$. Pour cela il faut considérer, parmi

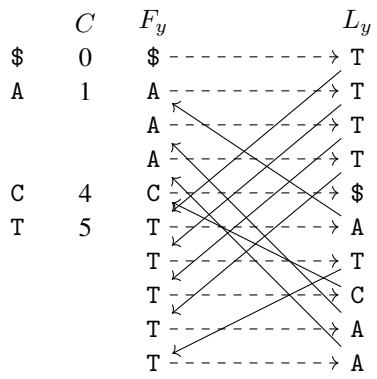


Figure 2.10. *Inversion de la BWT de $y =$ `CTATATTAT$`. La fonction LF est montrée par les flèches en trait plein. $F_y[1] =$ `$` puisque le terminateur est la plus petite lettre donc il est précédé par $L_y[1] =$ `T` qui est la dernière lettre de y. La fonction LF permet de repérer la position de ce `T` dans la première colonne. Il suffit d'itérer le processus pour reconstruire y de droite à gauche.*

Algorithme 2.1. Algorithme de recherche en arrière (*backward search*) dans une transformée de Burrows-Wheeler.

input : L – transformée de Burrows-Wheeler
C – tableau donnant pour chaque lettre le nombre de lettres inférieures à celle-ci.
x – séquence de longueur m à rechercher
output : ℓ, r – positions de début et fin correspondant aux occurrences de x dans M_y

Function `BackwardSearch`(L, C, x)
  `// Initialisation de la taille de l'intervalle à l'intégralité de` L;
  $\ell \leftarrow 1$;
  $r \leftarrow |L|$;
  pour $i \leftarrow |x|$ **à** 1 **faire**
    $c \leftarrow x[i]$;
    $\ell \leftarrow C[c] + rank_c(L, \ell - 1) + 1$;
    $r \leftarrow C[c] + rank_c(L, r)$;
  retourner ℓ, r

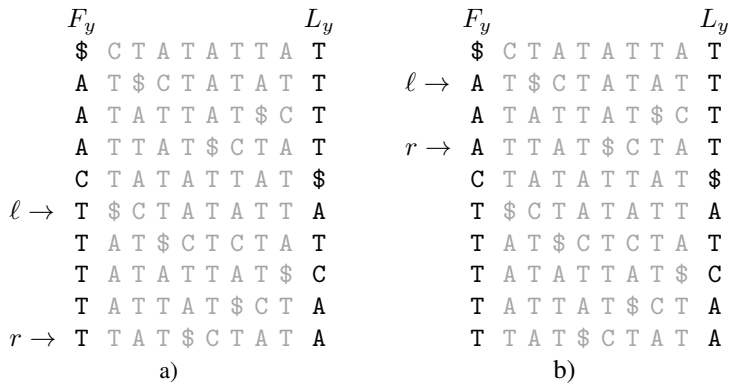


Figure 2.11. *Recherche de* `AT` *dans la BWT de* $y =$ `CTATATTAT$` *: a) les bornes de l'intervalle sont initialisées avec l'intervalle des* `T` *; b) parmi les cinq* `T` *(en première colonne), trois sont précédés par des* `A` *(en dernière colonne). Ces trois* `A` *sont les premier, deuxième et troisième de la dernière colonne et correspondent donc aux premier, deuxième et troisième* `A` *de la première colonne.* `AT` *apparaît donc trois fois dans* y*.*

les rotations du premier intervalle, celles qui sont précédées par $c = x[i-1]$. Il faut ensuite compter le nombre d'occurrences de c dans L_y entre les positions ℓ et r et mettre à jour les valeurs de ℓ et de r (voir lignes 6 et 7 de l'algorithme 2.1).

Il suffit de répéter le processus en considérant ainsi les lettres de x dans l'ordre décroissant des positions. Il est possible d'arrêter avant l'examen de toutes les lettres de y si l'intervalle se réduit au vide, auquel cas cela signifie que x n'apparaît pas dans y. Le nombre d'occurrences de x dans y correspond à la longueur de l'intervalle final. Pour obtenir les positions de ces occurrences, il faut disposer du tableau des suffixes de y. En pratique, ce tableau est trop volumineux. Il est possible de n'en conserver qu'un échantillon et de retrouver aisément les positions non échantillonnées, à l'aide de la fonction LF. C'est le principe du FM-index (Ferragina et Manzini 2000).

La figure 2.11 montre un exemple de recherche à l'aide de la BWT.

2.3.3.1. *Représentation d'un graphe de De Bruijn avec une transformée de Burrows-Wheeler*

Dans la section **??** sur les graphes de De Bruijn, nous avons indiqué qu'il s'agit d'une structure de données abstraite qui peut être représentée et stockée en mémoire de plusieurs manières : avec des filtres de Bloom, des tables de hachage, etc. Ces graphes peuvent également être représentés en utilisant une transformée de Burrows-Wheeler comme l'ont montré (Bowe *et al*. 2012).

Supposons que nous souhaitions construire un graphe de De Bruijn sur un ensemble S de m séquences. Commençons par transformer cet ensemble en une séquence. Toutes les séquences sont concaténées et séparées par un symbole différent de tous les autres [1]. Cette séquence est elle-même précédée par une suite de k symboles différents de tous les autres. Nous obtenons alors un mot $T = \$_1\$_1\$_1 S_1 \cdot \$_2 \cdot S_2 \cdot \$_3 \cdots S_m \cdot \$_{m+1}$. Le graphe de De Bruijn d'ordre k est construit sur T et possède un sommet par k-mer différent de T, soit E cet ensemble de k-mers, et n le nombre d'arcs dans le graphe.

La figure 2.12 montre un exemple de tel graphe.

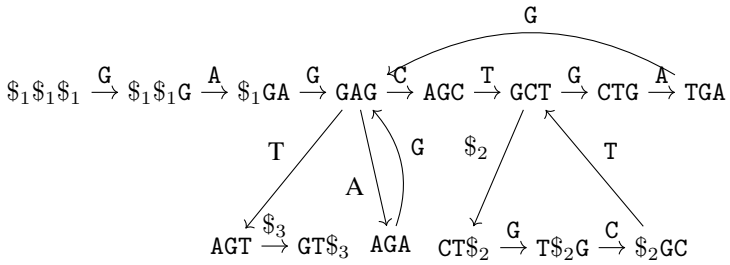


Figure 2.12. *Graphe de De Bruijn d'ordre 3 de E l'ensemble des 3-mers de $\$_1\$_1\$_1 GAGAGCT\$_2 GCTGAGT\$_3$. N'ont été dessinés que les arcs correspondant à des k-mers présents dans le jeu de données, et non les arcs théoriquement possibles*

Dans cette section, nous allons voir comment des concepts de la transformée de Burrows-Wheeler, notamment le principe de sa fonction LF, sont utilisés pour représenter un tel graphe. Les opérations indispensables sont de pouvoir identifier le sommet correspondant à un k-mer donné, s'il existe, et ensuite de naviguer dans le graphe en identifiant les prédécesseurs ou les successeurs d'un sommet.

Un graphe de De Bruijn n'est pas une structure linéaire, il ne suffit donc pas de construire une transformée de Burrows-Wheeler sur T. Néanmoins, comme pour la transformée de Burrows-Wheeler, imaginons une matrice. Celle-ci est composée de n lignes et $k+1$ colonnes. Chaque ligne consiste en un k-mer de T et de la lettre qui le suit dans T. Autrement dit, chaque ligne consiste en un sommet du graphe et d'un arc sortant. Nous appelons K les k premières colonnes de cette matrice ($K[1]$ correspond au premier k-mer dans la matrice) et W la dernière colonne. Les lignes sont ensuite triées en lisant les k premières colonnes de droite à gauche, suivies de la

Notons qu'en pratique, certaines astuces permettent de se dispenser d'employer un séparateur différent pour chaque séquence

dernière colonne W. Il s'agit de l'équivalent de la transformée de Burrows-Wheeler appliquée à un graphe de De Bruijn (voir les colonnes K et W dans la figure 2.13).

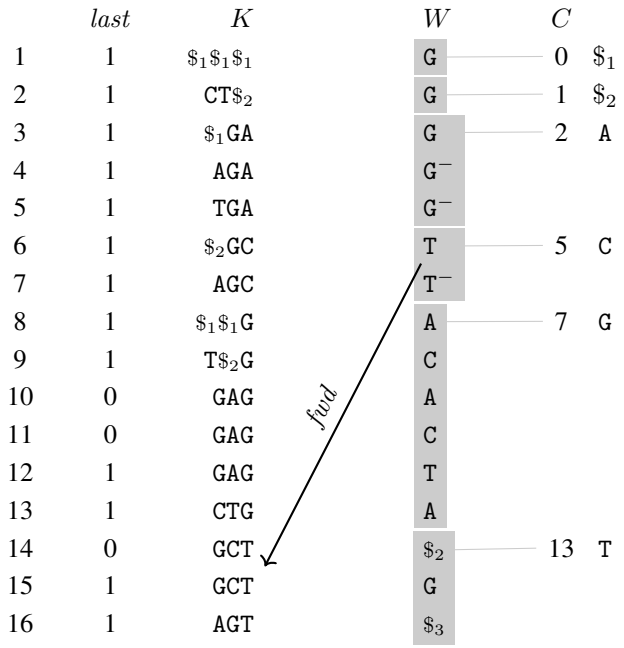


Figure 2.13. *Graphe de De Bruijn d'ordre 3 de E représenté à l'aide de W*

COMMENTAIRE SUR LA FIGURE 2.13.– *On note que le k-mer GAG est présent trois fois (positions 10 à 12), mais dans $last$ un 1 est stocké uniquement à la dernière de ces positions. D'autre part, à ces positions, chacune des lettres est différente dans W puisqu'elles correspondent toutes à un arc distinct sortant du sommet GAG, et elles sont triées dans l'ordre alphabétique. Trois k-mers se terminent par GA (positions 3 à 5). Les trois sommets correspondant ont un unique arc sortant par G. Néanmoins, un G n'est stocké dans W qu'à la première position, les autres stockent une version modifiée : G^-. Pour rappel, K n'a pas besoin d'être réellement stocké. La fonction fwd à la position 6 donne pour résultat la position 15.*

Néanmoins, avant d'utiliser cette transformée W, il nous faut encore procéder à quelques modifications pour parvenir à parcourir le graphe de De Bruijn.

Premièrement, dans K, plusieurs entrées (nécessairement consécutives) peuvent être identiques, c'est-à-dire qu'elles correspondent à un même k-mer. Cela arrive

lorsqu'un sommet a plusieurs arcs sortants. Nous voulons marquer chaque k-mer distinct une seule fois. Pour cela, nous utiliserons $last$, un vecteur de n bits. Il est défini tel que $last[i] = 0$ si et seulement si $K[i] = K[i+1]$, avec $1 \leq i < n$ (voir colonne $last$ dans la figure 2.13).

De manière similaire, nous allons marquer dans W les lettres qui représentent des arcs sortants, selon qu'il s'agit de la première à amener à un sommet donné ou non. S'il s'agit de la première, rien n'est modifié, dans le cas inverse nous modifions le caractère de W, en lui adjoignant par exemple un $^-$. Autrement dit, si $K[i] = s_1 s_2 \cdots s_k$ et $W[i] = c$ alors $W[i] \leftarrow c^-$ si et seulement si il existe une position $j < i$ telle que $K[j] = s'_1 s_2 \cdots s_k$ (les $k-1$ dernières lettres sont les mêmes que pour $K[i]$) et $W[j] = c$. Dans ce cas, les positions j et i correspondent à deux sommets amenant au même sommet représentant le k-mer $s_2 \cdots s_k c$. La taille de l'alphabet de W est donc potentiellement doublée par rapport à sa taille d'origine.

De manière similaire à la transformée de Burrows-Wheeler, nous aurons également un tableau C qui permettra de connaître l'intervalle des positions des k-mers finissant par une lettre donnée. Plus concrètement, $C[c]$ indique pour n'importe quelle lettre de l'alphabet la position de la première occurrence d'un k-mer finissant par c dans la matrice.

Un exemple est donné dans la figure 2.13.

Seuls C, W et $last$ sont stockés et suffisent à naviguer dans le graphe de De Bruijn. K n'est pas stocké.

2.3.3.1.1. Parcours du graphe

La fonction LF introduite pour la transformée de Burrows-Wheeler s'applique de manière similaire à partir du tableau C et de W mais en y ajoutant l'information du vecteur de bits $last$ afin de prendre en compte les lettres modifiées. Nous appellerons fwd cette fonction pour faire la différence avec LF. La fonction fwd s'applique à une position donnée, correspondant à un sommet s et à un arc sortant, et permet de trouver une des positions du sommet s' atteint à partir de ce sommet s en suivant cet arc. La fonction fwd ne peut s'appliquer qu'à des positions où W ne contient pas une lettre modifiée c^-. Dans le cas contraire, il faut se ramener à la position précédente la plus proche contenant la même lettre non modifiée c. Ensuite, l'opération $rank_c$ sur W nous donnera le nombre d'occurrences occ de c jusqu'à la position considérée. Il reste ensuite à utiliser le tableau C afin de savoir où se trouve l'intervalle des k-mers finissant par un c et de ne prendre en compte que les positions où $last$ vaut 1. Le occ-ème 1 de cet intervalle est à la position d'une des lignes correspondant au sommet s'. Plus formellement, $fwd(i) = C[c] + select_1(last[C[c]+1\ldots], rank_c(W,i))$, avec $c = W[i]$ et $fwd(i)$ est définie si et seulement si c n'est pas une lettre modifiée.

Par exemple, dans la figure 2.13, si nous sommes à la position 7, $W[7] = \texttt{T}^-$, il faut remonter à la précédente position qui contenait un `T`, il s'agit de la position 6. Ce `T` est le premier dans W, il faut maintenant aller à la position du premier k-mer qui se termine par un `T` et qui soit marqué par un 1 dans $last$. Pour cela, le tableau C nous permet de savoir que l'intervalle d'intérêt, correspondant aux k-mers finissant par un `T`, va de la position 14 à 16. Les opérations $rank$ et $select$ sur $last$ nous permettent d'identifier que la position correspondant à notre critère est la 15.

Cette fonction fwd peut notamment être utilisée pour identifier la position d'un sommet correspondant à un k-mer donné. Pour cela, il faut appliquer la fonction fwd à k reprises en démarrant de l'intervalle contenant toutes les lignes, de manière similaire à la recherche de motifs avec une transformée de Burrows-Wheeler. Par exemple, la recherche de `GCT` donnera respectivement les intervalles $[\ell, r] : [1, 16]$, puis $[8, 13]$, puis $[6, 7]$, puis $[15, 15]$, indiquant que le sommet correspondant au k-mer `GCT` se trouve en position 15.

2.3.3.1.2. Identifier le successeur d'un sommet par la lettre c

Supposons que nous voulions passer d'un sommet à un autre en suivant l'arc étiqueté par la lettre c. Prenons comme point de départ la ligne i de la matrice. Cette ligne ne représente pas un sommet mais un sommet ainsi qu'un de ses arcs sortants. Pour identifier toutes les lignes qui correspondent à ce même sommet, il faut utiliser le vecteur de bits $last$. La position qui suit le premier 1 avant $last[i]$ donnera la première position du sommet. De même, identifier la position du premier 1 à partir de $last[i]$ (inclus) donnera la dernière position de ce sommet. Nous obtenons alors un intervalle de toutes les positions correspondant au même sommet que celui à la position i. Ce calcul peut être fait en temps constant avec l'aide des opérations *rank* et *select*. Dans cet intervalle, supposons que W ne contienne pas c mais uniquement c^-. Il faut alors remonter à la position j de la précédente occurrence de c dans W. Par définition, cette position correspond à un k-mer finissant par les $k-1$ mêmes lettres que le sommet d'origine. Le sommet destination sera donc bien celui attendu. À partir de la position j (où $W[j] = c$), il suffit d'appliquer la fonction fwd afin d'obtenir une position du successeur du sommet courant par la lettre c.

Dans la suite, nous décrivons succinctement l'identification d'un k-mer associé à un sommet ainsi que l'identification d'un sommet en précédant un autre.

2.3.3.1.3. Identifier le k-mer associé à un sommet

À une position donnée, seule la dernière lettre du k-mer est accessible (grâce au tableau C). Pour reconstituer la séquence complète du k-mer, il est donc nécessaire de retourner aux $k-1$ sommets précédents afin d'extraire à chaque fois la dernière lettre de leurs k-mers associés. L'inverse de la fonction fwd est utilisée pour ce faire.

2.3.3.1.4. Identifier le prédécesseur d'un sommet commençant par la lettre c

Supposons que nous voulions passer d'un sommet à un sommet précédent dont le k-mer associé débute par la lettre c. Partons de la ligne i de la matrice et appelons c' la dernière lettre du k-mer correspondant à cette ligne. Nous pouvons remonter à un prédécesseur, pour lequel c' est stocké dans W et identifier tous les autres prédécesseurs possibles : ce sont toutes les occurrences de c'^{-} qui précèdent la prochaine occurrence de c'. Parmi ces occurrences potentielles de prédécesseurs, il suffit d'identifier par dichotomie le k-mer qui commence par la lettre c choisie.

2.3.3.1.5. Complexités en temps et en espace

Seuls W, $last$ et C sont réellement stockés. En supposant l'alphabet constant, la structure ayant la complexité en espace la plus importante est W. Le tableau W peut être représenté en $n \log \sigma + o(n \log \sigma)$ bits, de manière à disposer de l'opération $rank$ sur le tableau, avec n le nombre d'arcs dans le graphe. C peut être stocké avec $\sigma \log n$ bits et $last$ avec $n + o(n)$ bits. L'espace total utilisé est alors en $n \log(\sigma) + o(n \log \sigma)$ bits.

L'accès à un successeur se fait en temps constant, identifier le k-mer d'un sommet est effectué en $O(k)$ tout comme le fait d'accéder au prédécesseur.

2.4. Critères de choix d'indexation

Nous allons maintenant tenter de donner des indications sur le choix d'une structure d'indexation en fonction des besoins. Ou pourquoi choisir un filtre de Bloom, qui peut donner des résultats inexacts, quand on dispose de méthodes pour indexer, de manière exacte, tout le texte ?

2.4.1. *En fonction du type de requête nécessaire*

En fonction de l'application envisagée, il peut être suffisant d'effectuer des requêtes à partir de k-mers ou, à l'inverse, il est indispensable de pouvoir requêter des séquences de longueurs arbitraires. Dans le premier cas, les structures d'indexation présentées en section **??** seront suffisantes, d'autant qu'elles sont plus économes en mémoire, alors que dans le second cas il faudra avoir recours aux structures de la section **??**.

Au-delà de la simple requête sur une séquence de taille fixe ou non, le type de réponse que peut apporter la structure importe. Certaines structures ne permettent que de savoir si une séquence donnée existe (*existence*) alors que d'autres permettent de connaître le nombre d'occurrences (*comptage*) ainsi que la position de chacune d'entre

Structure	Existence	Comptage	Localisation	Exacte
Filtre de Bloom	O	N	N	N
Table de hachage	O	O	O	O
Graphes de De Bruijn	O	P	N	P
Arbre des suffixes	O	O	O	O
Table des suffixes	O	O	O	O
Transformée de Burrows-Wheeler	O	O	O	O

Table 2.1. *Fonctionnalités des structures d'indexation présentées. Les structures permettent (O) ou non (N) ou, parfois, peuvent permettre (P) certaines fonctionnalités*

elles (*localisation*). Connaître la localisation permet ensuite d'accéder à des informations sur le contexte de l'occurrence. Le tableau 2.1 récapitule les différentes fonctionnalités des structures présentées.

Diverses structures d'indexation fournissent les mêmes fonctionnalités : par exemple, les tables de hachage comme la transformée de Burrows-Wheeler permettent de localiser les occurrences d'une séquence recherchée. Au-delà des fonctionnalités se pose la question des ressources nécessaires pour construire et utiliser de telles structures de données.

2.4.2. *En fonction du compromis espace-temps et de la quantité de données*

Dans l'ordre d'occupation mémoire, la structure la plus économe sera évidemment le filtre de Bloom qui n'est composé que d'un vecteur de bits, dont la taille dépend notamment du taux de faux positifs choisi. Ensuite vient le graphe de De Bruijn, qui peut être construit avec des filtres de Bloom ou avec une transformée de Burrows-Wheeler. Le FM-index utilisant une transformée de Burrows-Wheeler est la structure d'indexation plein texte la plus économe parmi celles présentées : elle est compressée. Viennent ensuite la table des suffixes puis l'arbre des suffixes. Afin de donner une idée de la place requise par ces structures, nous pouvons considérer l'indexation d'un génome humain (ou de ses k-mers). Un génome humain est composé d'environ 3 milliards de nucléotides et, donc, environ autant de k-mers. Un ordre d'idée de l'espace nécessaire à chacune des structures d'indexation est indiqué dans le tableau 2.2.

D'un point de vue plus formel, on peut également exprimer l'espace occupé pour indexer un texte (ou ses k-mers) en fonction de sa taille n. Nous donnons ces complexités en espace dans le tableau 2.3.

Structure d'indexation	Espace occupé (Go)
Filtre de Bloom	1
Graphe de De Bruijn	1
Transformée de Burrows-Wheeler	2
Table de hachage	8
Table des suffixes	15
Arbre des suffixes	$\simeq 30$

a) Vecteur de 10^{10} bits, avec 2 fonctions de hachage, 20 % de faux positifs.
b) Pour le comptage uniquement.

Table 2.2. *Espace occupé par chaque structure d'indexation pour l'indexation d'un génome humain (environ 3 milliards de nucléotides), arrondi au giga-octet le plus proche. L'espace peut fluctuer en fonction des paramètres choisis (en particulier pour les structures les plus économes.)*

Structure d'indexation	Complexité en espace
Filtre de Bloom	$2n$ à $3n$ bits
Graphe de De Bruijn	$3,24n$ bits (Chikhi *et al.* 2015)
Transformée de Burrows-Wheeler	$nH_k(y) + o(n)$ bits
Table de hachage	$O(n)$
Table des suffixes	$n \log n$ bits
Table des suffixes	$O(n)$

Table 2.3. *Complexité en espace des structures d'indexation en fonction de n, la taille du texte indexé. $H_k(y)$ est l'entropie empirique d'ordre k du texte : elle mesure le nombre de bits moyen nécessaire pour représenter une lettre de la séquence y, en ayant connaissance des k lettres précédentes. Pour de l'ADN, cette valeur est nécessairement inférieure ou égale à 2.*

En réalité, certaines de ces structures offrent un compromis espace-temps. Il est possible de réduire l'espace utilisé par la structure à condition d'augmenter le temps des requêtes, ou inversement.

Le temps de requête, justement, est plus difficile à estimer. Les complexités théoriques en temps nous renseignent moins directement que celles en espace sur l'efficacité pratique. Par exemple, compter le nombre d'occurrences d'un mot de longueur m avec uniquement une table des suffixes se fait en temps $O(m \log n)$ alors que dans une transformée de Burrows-Wheeler elle est en temps optimal $O(m)$. Pourtant, en pratique, la recherche dans la table des suffixes est plus rapide de plusieurs ordres de grandeur. Cette différence est due aux nombreux accès mémoire requis pour interroger une transformée de Burrows-Wheeler ainsi que les structures qui l'accompagnent dans un FM-index et au fait que ces accès mémoire ne soient pas

contigus. De plus, l'utilisation d'une table étendue des suffixes améliore la complexité en temps de la recherche à $O(m + \log n)$.

2.4.3. *En fonction de la nécessité d'ajouter ou de modifier les données indexées*

Les données de séquençage utilisées en bioinformatique sont loin d'être immuables. De nouveaux jeux de données, de nouvelles espèces sont régulièrement séquencées. Il est alors nécessaire de mettre à jour l'index qui stocke les séquences d'intérêt afin de rester à jour.

Pour les structures indexant les k-mers, l'ajout de nouvelles données ne pose pas de problème particulier. En revanche, la suppression de données dans un filtre de Bloom n'est pas possible. On ne peut pas inverser un 1 en 0 car plusieurs k-mers ont pu mener au stockage d'un 1 à la même position. Pour permettre les suppressions, il faudrait donc un compteur, plutôt qu'un bit, pour savoir combien de fois une insertion a été faite à une position donnée (Fan *et al*. 2000). Une telle modification a évidemment des conséquences sur la consommation mémoire de la structure.

Les mises à jour des structures indexant tout le texte sont moins évidentes : ces structures indexent des suffixes du texte. Or une seule modification peut impacter, selon sa position dans le texte, une grande partie des suffixes. Néanmoins, en utilisant la fonction LF de la transformée de Burrows-Wheeler (voir section **??**), il est possible de mettre à jour une transformée de Burrows-Wheeler, et donc un FM-index, à condition d'utiliser des structures de données dynamiques qui sont plus lentes d'un facteur $\log n$ (Salson *et al*. 2010).

2.4.4. *Des choix d'indexation selon les applications*

2.4.4.1. *Indexation de collections de génomes*

Les index plein texte ne permettent pas de tirer parti d'une telle redondance, y compris le FM-index qui est pourtant une structure d'indexation compressée. Un FM-index peut compresser de courtes répétitions, mais n'est pas adapté à la compression de longues répétitions. Afin de gérer au mieux la redondance des données, il faut soit éliminer cette redondance en amont, soit en aval (Gagie et Navarro 2019). En amont, il est possible de compresser les séquences redondantes en utilisant des grammaires (comme la compression LZ-77) ou en indexant les séquences alignées, plutôt que les séquences brutes. En aval, après avoir indexé des séquences redondantes dans un FM-index, il faut supprimer une partie de la redondance restante en utilisant d'autres techniques de compression. Les solutions amonts semblent les plus efficaces à l'heure actuelle.

2.4.4.2. *Indexation de collection de jeux de séquençage*

L'indexation de collections de jeux de séquençage, comme l'indexation de collection de génomes, revient à indexer de grandes quantités de données génomiques. En revanche, la différence porte sur la nature des données indexées. Indexer des jeux de données de séquençage, c'est indexer des milliards de courtes séquences, éventuellement bruitées. En raison des profondeurs de séquençage communément rencontrées, dans un seul jeu de séquençage une même région peut être séquencée une centaine de fois. Nous avons donc des niveaux de redondance bien supérieurs à ceux rencontrés lorsqu'on indexe des génomes. Mais cette redondance est également moins parfaite, à cause des erreurs de séquençage. Bien qu'avec la deuxième génération de séquenceurs les erreurs de séquençage soient relativement rares, elles peuvent néanmoins constituer la majorité des k-mers distincts d'un jeu de séquençage. Dans ce cas, la redondance est plus difficile à exploiter, à cause des petites différences entre les différents k-mers.

En revanche, de nombreuses solutions permettent l'indexation de k-mers de milliers de jeux de données de séquençage, voir la synthèse de (Marchet *et al*. 2021). Soit chaque jeu de données est indexé indépendamment (par exemple avec un filtre de Bloom) et une structure additionnelle permet de requêter efficacement l'ensemble des jeux de données, soit tous les jeux de données sont indexés ensemble (par exemple avec un graphe de De Bruijn) et une *couleur* est associée à chaque k-mer, correspondant aux jeux de données où ce k-mer apparaît, on parle alors de graphes de De Bruijn colorés. Dans tous les cas, un filtrage est opéré sur les k-mers les moins abondants afin de retirer une partie des erreurs de séquençage, qui risqueraient de prendre une place démesurée.

2.5. Conclusions et perspectives

2.5.1. *Des méthodes efficaces pour indexer quelques génomes ou jeux de séquençage*

L'assemblage de génomes comme l'alignement de séquences sur un génome sont des tâches fondamentales en bioinformatique des séquences (voir les chapitres **??** et **??**). Or l'alignement comme l'assemblage ne pourraient être résolus efficacement sans le recours à des structures d'indexation. Les méthodes d'indexation employées s'appuient sur des structures déjà connues (graphe de De Bruijn, table des suffixes, FM-index, etc.) en les adaptant à la problématique et en les faisant passer à l'échelle. Désormais, ces structures d'indexation donnent satisfaction pour l'alignement et l'assemblage, lorsqu'ils sont restreints à l'étude d'un génome ou d'un jeu de données, en offrant de bons compromis entre l'espace consommé par la structure et les temps de requête (compromis espace-temps).

2.5.2. *Des méthodes tirant difficilement parti de la redondance des données*

Dans une certaine mesure, les graphes de De Bruijn permettent cela, mais en supprimant de l'information : l'indexation ne se fait que pour des k-mers, avec une valeur fixée à l'avance pour k.

En ce qui concerne l'indexation plein texte, à la fin des années 2010 diverses recherches tentent d'améliorer leur efficacité pour mieux leur faire tirer parti de la très forte redondance des données (Gagie *et al*. 2018 ; Na *et al*. 2018 ; Sirén *et al*. 2020). En particulier, certaines approches vont chercher à représenter, compresser et indexer un graphe plutôt que des milliers de séquences linéaires, en appliquant la transformée de Burrows-Wheeler à un tel graphe.

Notamment, il a été proposé de regrouper ces différentes approches sous la notion commune de graphes de Wheeler, qui correspondent à des automates non déterministes permettant de résoudre des problèmes de recherche de motifs. Ces graphes peuvent être indexés en utilisant peu d'espace tout en permettant un parcours rapide du graphe. Plus formellement, un graphe orienté $G = (V, E)$ doit respecter certaines propriétés pour être de Wheeler. Il doit exister un ordre sur les sommets tel que pour tout couple d'arcs (u, k, v) et (u', k', v'), avec $u, v, u', v' \in V$ et avec k et k' les étiquettes des arcs de u à v et de u' à v' respectivement, les deux propriétés suivantes sont vérifiées :

- $k < k' \rightarrow v < v'$;
- $k = k'$ et $u < u' \rightarrow v \leq v'$.

Au-delà de ces approches qui généralisent la notion de transformée de Burrows-Wheeler, d'autres recherches visent à développer des structures d'indexation au-dessus de méthodes de compression déjà connues pour leur grande efficacité à fortement réduire la grande redondance des données. En 2019, des chercheurs ont proposé un index théorique rassemblant toutes ces méthodes de compression sous une même notion, celle d'*attracteurs*, et proposant une méthode pour les indexer, quelle que soit la méthode de compression sous-jacente (Navarro et Prezza 2019). Les *attracteurs* d'une séquence constituent l'ensemble (le plus petit possible) des positions du texte tel que tout facteur du texte peut être retrouvé chevauchant une de ces positions.

Les différentes approches présentées dans cette section sont, en 2020, encore à l'état de prototypes de recherche et ne sont pas encore utilisées dans des outils bioinformatiques pour l'indexation de milliers de jeux de données. Néanmoins, ces approches illustrent la nécessité de structures d'indexation toujours plus frugales et préfigurent peut-être les méthodes qui seront prochainement utilisées pour de l'indexation à plus grande échelle.

2.6. Bibliography

Baeza-Yates, R. and Gonnet, G. (1992). A new approach to text searching. *Communications of the ACM*, 35(10), 74–82.

Bahne, J., Bertram, N., Böcker, M., Bode, J., Fischer, J., Foot, H., Grieskamp, F., Kurpicz, F., Löbel, M., Magiera, O. et al. (2019). SACABench: Benchmarking suffix array construction. *26th International Symposium on String Processing and Information Retrieval*, Segovia, October 7–9, 407–416.

Belazzougui, D., Botelho, F.C., Dietzfelbinger, M. (2009). Hash, displace, and compress. *European Symposium on Algorithms*, 682–693.

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

Bowe, A., Onodera, T., Sadakane, K., Shibuya, T. (2012). Succinct de Bruijn graphs. In *Algorithms in Bioinformatics – 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10–12, 2012 Proceedings*, Raphael, B.J. and Tang, J. (eds). Springer.

Broder, A. and Mitzenmacher, M. (2004). Network applications of Bloom filters: A survey, *Internet Mathematics*, 1(4), 485–509.

de Bruijn, N. (1950). On bases for the set of integers. *Publicationes Mathematicae Debrecen*, 1, 232–242.

Burrows, M. and Wheeler, D.J. (1994). A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation.

Chikhi, R. and Rizk, G. (2013). Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8(1), 22.

Chikhi, R., Limasset, A., Jackman, S., Simpson, J.T., Medvedev, P. (2015). On the representation of de Bruijn graphs. *Journal of Computational Biology*, 22(5), 336–352.

Cette bibliographie est identique à celle de l'ouvrage correspondant en anglais publié par ISTE.

Chikhi, R., Holub, J., Medvedev, P. (2019). Data structures to represent sets of k-long DNA sequences. arXiv:1903.12312.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2022). *Introduction to Algorithms*. MIT Press.

Crochemore, M., Hancart, C., Lecroq, T. (2007). *Algorithms on Strings*. Cambridge University Press.

Fan, L., Cao, P., Almeida, J., Broder, A.Z. (2000). Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3), 281–293.

Farach, M. (1997). Optimal suffix tree construction with large alphabets. *Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science*, Miami Beach, 137–143.

Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12–14 November 2000*, IEEE Computer Society, 390–398.

Gagie, T. and Navarro, G. (2019). Compressed indexes for repetitive textual datasets. In *Encyclopedia of Big Data Technologies*, Sakr, S. and Zomaya, A.Y. (eds). Springer.

Gagie, T., Navarro, G., Prezza, N. (2018). Optimal-time text indexing in BWT-runs bounded space. *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, SIAM, 1459–1477.

Good, I.J. (1946). Normal recurring decimals. *Journal of the London Mathematical Society*, 21(3), 167–169.

Goodwin, S., McPherson, J.D., McCombie, W.R. (2016). Coming of age: Ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6), 333–351.

Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G. (2012). De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2), 226.

Kärkkäinen, J. and Sanders, P. (2003). Simple linear work suffix array construction. In *Proceedings of the Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30–July 4 2003*, Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds). Springer.

Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K. (2001). Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proceedings of the Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1–4, 2001*, Amir, A. and Landau, G.M. (eds). Springer.

Kim, D.K., Sim, J.S., Park, H., Park, K. (2003). Linear-time construction of suffix arrays. In *Proceedings of the Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25–27, 2003*, Baeza-Yates, R.A., Chávez, E., Crochemore, M. (eds). Springer.

Ko, P. and Aluru, S. (2003). Space efficient linear time construction of suffix arrays. In *Proceedings of the Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25–27, 2003*, Baeza-Yates, R.A., Chávez, E., Crochemore, M. (eds). Springer.

Luo, L., Guo, D., Ma, R.T.B., Rottenstreich, O., Luo, X. (2019). Optimizing bloom filter: challenges, solutions, and comparisons. *IEEE Communications Surveys Tutorials*, 21(2), 1912–1949.

Manber, U. and Myers, G. (1990). Suffix arrays: A new method for on-line string searches. *SODA '90: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, Industrial and Applied Mathematics, Philadelphia, PA, 319–327.

Marçais, G., Pellow, D., Bork, D., Orenstein, Y., Shamir, R., Kingsford, C. (2017). Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14), i110–i117.

Marchet, C., Boucher, C., Puglisi, S.J., Medvedev, P., Salson, M., Chikhi, R. (2021). Data structures based on k-mers for querying large collections of sequencing datasets. *Genome Research*, 31, 1–12.

Mayer-Schönberger, V. and Cukier, K. (2013). *Big Data: A Revolution that Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt.

McCreight, E.M. (1976). A space-economical suffix tree construction algorithm. *Journal of Algorithms*, 23(2), 262–272.

Mori, Y. (2015). LibDivSufSort: A lightweight suffix-sorting library [Online]. Available at: https://github.com/y-256/libdivsufsort.

Na, J.C., Kim, H., Min, S., Park, H., Lecroq, T., Léonard, M., Mouchard, L., Park, K. (2018). FM-index of alignment with gaps. *Theoretical Computer Science*, 710, 148–157.

Navarro, G. and Mäkinen, V. (2007). Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2–es.

Navarro, G. and Prezza, N. (2019). Universal compressed text indexing. *Theoretical Computer Science*, 762, 41–50.

NCBI (2013). Grch37.p13-genome-assembly-ncbi [Online]. Available at: https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.25.

NCBI (2019). Grch38.p13-genome-assembly-ncbi [Online]. Available at: https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39.

Puglisi, S.J., Smyth, W.F., Turpin, A.H. (2007). A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2), 4–es.

Sacomoto, G.A., Kielbassa, J., Chikhi, R., Uricaru, R., Antoniou, P., Sagot, M.-F., Peterlongo, P., Lacroix, V. (2012). KIS SPLICE: De-novo calling alternative splicing events from RNA-seq data. *BMC Bioinformatics*, 13(S6).

Sainte-Marie, C.F. (1894). Question 48. *L'Intermédiaire des Mathématiciens*, 1, 107–110.

Salson, M., Lecroq, T., Léonard, M., Mouchard, L. (2010). Dynamic extended suffix arrays. *Journal of Discrete Algorithms*, 8, 241–257.

Schatz, M.C. and Langmead, B. (2013). The DNA data deluge. *IEEE Spectrum*, 50(7), 28–33.

Sirén, J., Garrison, E., Novak, A.M., Paten, B., Durbin, R. (2020). Haplotype-aware graph indexes. *Bioinformatics*, 36(2), 400–407.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3), 249–260.

Conclusion

Dans cet ouvrage, l'accent a été mis sur la question « comment faire ? ». L'objectif était de présenter une partie de la diversité des méthodes existant en bio-informatique, mais aussi d'illustrer à quel point le choix de la structure de représentation des données et des problèmes a son importance dans la facilité avec laquelle on peut résoudre ces problèmes. Nous avons vu différentes questions, et l'emphase a été mise sur leur signification, leur analyse, leur formalisation sous forme de problème mathématique, leur représentation sous forme de structure discrète, et sur l'aspect algorithmique de leur résolution. À travers les difficultés évoquées dans les différents chapitres, nous n'avons fait qu'effleurer un certain nombre d'autres aspects opérationnels de la bio-informatique. Nous leur consacrons ici quelques lignes, pour mieux vous donner l'envie de poursuivre l'exploration de cette discipline naissante.

Ainsi, pour mieux comprendre l'interaction entre la représentation et le choix des structures d'une part, et les données et problématiques d'autre part, nous pouvons évoquer l'aspect visualisation. Celle-ci consiste à proposer une représentation graphique des concepts et données non seulement en amont, mais également au niveau des solutions construites par les méthodes. Ainsi, pour les assemblages de génomes, la question de ce que nous obtenons en sortie, les séquences de contigs, est plus complexe qu'il ne semble à première vue, et il est très intéressant de pouvoir visualiser les graphes d'assemblages plutôt que simplement disposer du fichier des séquences linéaires correspondant à un organisme. Visualiser les parties de ce graphe ayant une complexité supérieure peut permettre de donner des outils pour mieux les comprendre et mieux les utiliser. Il en va de même pour chaque question présentée dans cet ouvrage, et chaque type de représentation mathématique s'accompagne d'un visuel « attendu » permettant aux experts et aux expertes d'appliquer leurs connaissances *a priori* ou de se laisser surprendre au contraire par une part exploratoire. Il appartient tout autant au métier de bio-informaticien ou bio-informaticienne de proposer

des modes de représentation visuelle des structures et des données. Cette part n'est pas forcément mise en avant dans cet ouvrage, mais nous soulignons son importance dans la pratique de tous les jours.

Un certain nombre de principes voient le jour dans ce sens, pour assurer cette reproductibilité aussi bien au niveau des données qu'au niveau des logiciels et scripts. Le nœud de l'affaire passe par la documentation, et l'articulation entre la compréhension humaine et l'automatisation des procédés. Ainsi, les cahiers de laboratoire électronique permettant d'intégrer code et explication, les outils de gestion des versions, les outils collaboratifs, les entrepôts de données publics, les outils d'encapsulation dans des systèmes virtuels minimaux autonomes pour éviter les conflits de version sans trop gonfler les ressources nécessaires, sont autant de concepts qui sont à disposition des bio-informaticiens et bio-informaticiennes pour proposer des solutions complètes et pérennes aux problèmes qui se posent.

Nous espérons, à travers cet ouvrage, vous avoir fait passer à la fois une ouverture vers des concepts fondamentaux, mais aussi une partie de l'enthousiasme que nous pouvons rencontrer dans notre pratique quotidienne de la recherche en bio-informatique, et cette étincelle qui nous anime lorsque l'on trouve « la bonne méthode », celle qui va donner « de beaux résultats », qu'il est satisfaisant de faire partager, évoluer et répondre à des problématiques très concrètes.

Index