Comparer deux Génomes contenant des Gènes Dupliqués est Algorithmiquement Difficile

Guillaume Fertin

LINA, Université de Nantes
Guillaume.Fertin@univ-nantes.fr

GTGC - Lyon, Janvier 2006

Plan de l’exposé

1. Parlons Algorithmique
2. Réarrangements Génomiques
3. Gènes Dupliqués
4. Quelques Résultats
5. Notre Solution (Travaux en Cours)

Plan de l'exposé

1 Parlons Algorithmique

2 Réarrangements Génomiques

3 Gènes Dupliqués

4 Quelques Résultats

5 Notre Solution (Travaux en Cours)

Algorithmique - Méthodologie Générale

Étude d'un problème du point de vue algorithmique

- Partir d'un problème déjà modélisé
- Généralement, *P* sera:
  - soit un problème de **décision** (Réponse: OUI ou NON)
  - soit un problème d'**optimisation** (Réponse: une valeur à MAXIMISER ou à MINIMISER)
- But: pouvoir répondre au mieux à ce problème

Algorithmique - Méthodologie Générale

Classification des problèmes

- Pour cela, on cherche à classifier le problème
- Ceci se fait généralement en fonction de sa **difficulté**
- Première classification (= premier réflexe d'un algorithmicien):
  - Le problème est-il **polynomial** ?
  - Ou bien le problème est-il **NP-dur** ?

Algorithmique - Méthodologie Générale

Cette classification n'est pas anodine...

- Cette notion est à la base de la théorie de l'informatique
- Diverses questions à ce sujet sont ouvertes depuis $\sim$ 50 ans
- Par exemple, le système de cryptage sur Internet (transactions bancaires) repose sur cela ! (et tout ce qui est crypté en général)

Polynomial vs NP-dur

Problème polynomial

- On peut y répondre **exactement** et **rapidement**
- Exactement: la valeur à calculer est obtenue
- Rapidement: même sur des données très grandes, un ordinateur répond en "temps raisonnable"

Polynomial vs NP-dur

Problème NP-dur

- Impossible de répondre exactement **et** en "temps raisonnable"
- Idée: pour de grandes données, l'ordinateur mettra plusieurs jours/semaines/mois/ans/siècles à répondre
- Même sur les ordinateurs les plus puissants, même si cette puissance se multiplie à l'avenir
- On pourra aussi parler de NP-difficile, NP-complet

Comment montre-t-on qu'un problème est NP-dur ?

Principe de réduction

- Repose sur l'idée de **réduction** à partir d'un problème P' connu
- P' étant lui-même NP-dur

Que faire face à un problème Polynomial ?

Si P est polynomial

- Situation la plus favorable
- Trouver le "meilleur" algorithme
- Meilleur = le plus rapide
- Meilleur = le moins gourmand en mémoire
- La priorité étant souvent mise sur le **temps** d'exécution

Que faire face à un problème NP-dur ?

Si P est NP-dur

- Premier constat: pas la peine de s'acharner à trouver un algorithme exact **et** rapide
- Il faut donc baisser ses exigences
  - soit sur la **rapidité** d'exécution
  - soit sur l'**exactitude** de la réponse
- Dans tous les cas, on cherche à *affiner* encore la classification.

Problème NP-dur: si on s'autorise une perte en exactitude

Les Heuristiques

- On "invente" un algorithme polynomial A
- On espère que le résultat qu'il donne reste proche du résultat optimal
- ...mais **rien** ne permet théoriquement de le dire !
- Souvent "validé" par quelques essais sur quelques jeux de données
- L'idéal: tester sur des benchmarks solides, reconnus et utilisés par tous

Problème NP-dur: si on s'autorise une perte en exactitude

Les Algorithmes d'approximation

Classification plus fine du problème P:

- Problèmes possédant un **schéma d'approximation** (ou PTAS)
  - on peut s'approcher aussi près du "vrai" résultat qu'on veut...
  - ... si on en paye le prix en temps
  - C'est le plus favorable des cas NP-durs

Problèmes PTAS

Exemple typique

- Problème de minimisation
- Il existe un algorithme polynomial A tel que
  1. SOLUTION(A) = (1+ϵ)· SOLUTION_OPTIMALE pour n'importe quel $\epsilon > 0$
  2. **Mais** A a un temps d'exécution proportionnel à $n^{c+\frac{1}{\epsilon}}$
  3. avec n=taille des données et $c = constante$
  4. $\Rightarrow$ On peut s'approcher aussi près du résultat qu'on veut... si on en paie le prix en temps

Problèmes PTAS

En résumé

- Problèmes PTAS: les plus favorables parmi les problèmes NP-durs
- Contrôle total temps d'exécution/qualité du résultat
- Permet d'obtenir des compromis évaluables

Problème NP-dur: si on s'autorise une perte en exactitude

Les Algorithmes d'approximation

Classification plus fine du problème P:

- Problèmes ne possédant pas de PTAS mais étant **approximables** (ou APX):
  - Impossible de s'approcher aussi près du "vrai" résultat qu'on veut...
  - $\Rightarrow$ Il y a une limite qu'on ne peut pas franchir
  - Mais il existe un algorithme dont on peut **montrer** théoriquement que son résultat reste **dans tous les cas** proportionnel au résultat optimal

Problème APX

Exemple typique

- Problème de minimisation
- Résultat négatif: le problème consistant à obtenir non plus la solution optimale mais une solution S telle que

$$S \leq 2 \cdot SOLUTION\_OPTIMALE$$

est NP-dur

- Résultat positif: il existe un algorithme polynomial A tel que

$$SOLUTION(A) \leq 3 \cdot SOLUTION\_OPTIMALE$$

Problème APX

En résumé

- Résultat négatif: pas la peine de chercher à s'approcher aussi près qu'on veut de l'optimal
- Indique un "obstacle" infranchissable
- Résultat positif: donne une **garantie** sur le résultat (exemple: facteur 3)
- Le facteur 3 est un résultat **au pire**...
- Il peut être meilleur !

Problème NP-dur - Si on s'autorise une perte en exactitude

En résumé

- **Heuristiques**: pas de contrôle sur l'écart à l'optimal
- **PTAS**: écart à l'optimal contrôlable, variable selon le temps d'exécution accordé
- **APX**: donne de bonnes bornes sur l'écart à l'optimal
  - borne inférieure fournie par la théorie (réduction similaire à celle utilisée pour les problèmes NP-durs, mais un peu plus contrainte)
  - borne supérieure fournie par un algorithme dédié

Problème NP-dur

Si on s'autorise une perte en temps

- On a alors un temps exponentiel pour conserver l'exactitude de la réponse. On peut alors:
- Réduire l'espace des solutions visitées
- Transformer le problème P en un autre problème informatique classique P'
  - P' sera bien sûr exponentiel
  - P' étant classique, de nombreuses recherches ont été effectuées pour essayer d'accélérer son temps d'exécution tut en maintenant une réponse exacte
  - On y reviendra plus tard

Problème NP-dur

Si on s'autorise une perte en temps

- On a donc un temps exponentiel pour conserver l'exactitude de la réponse.
- On peut alors déterminer un algorithme exponentiel:
  - uniquement en un paramètre annexe k
  - ...paramètre qu'on sait être petit dans la pratique
- Ces problèmes sont alors appelés **FPT**

Problème NP-dur

Si on s'autorise une perte en temps - Algorithmes FPT

Exemple:

- Problème P sur une séquence d'ADN de taille $n = 10kb$
- Soit A un algorithme permettant de répondre à P
- Si A est exponentiel en n: pas d'espoir
- Si A est exponentiel en k=taille de l'alphabet=4, OK
- Par exemple, temps d'exécution $T \sim n^2 \cdot 3^k$
- Remarque: on peut aussi démontrer de façon théorique que certains problèmes **ne sont pas** FPT...

Pour résumer

Ce que fait l'algorithmicien devant un problème

- Il le "décortique" en cherchant à le classifier finement
- Permet d'éviter divers écueils, en particulier:
  - ne pas inventer une heuristique si le problème est polynomial
  - éviter de perdre du temps si le problème est NP-dur, non PTAS, non APX, non FPT, etc.
- Demande de l'intuition et de "l'expérience"
- ...comme dans toute discipline scientifique

Pour résumer - Suite

Ce que fait l'algorithmicien devant un problème

- Les résultats négatifs:
  - s'obtiennent de façon théorique
  - souvent en utilisant le principe de *réduction* depuis un problème connu
- Les résultats positifs:
  - s'obtiennent en déterminant des algorithmes...
  - ...dont il faut démontrer leur écart à l'optimum de façon théorique.

Plan de l'exposé

1 Parlons Algorithmique

2 **Réarrangements Génomiques**

3 Gènes Dupliqués

4 Quelques Résultats

5 Notre Solution (Travaux en Cours)

Modélisation du problème

Génomes

- Génome : suite ordonnée de gènes
- Gènes représentés par des entiers
- Gènes pouvant avoir un signe

Hypothèses simplificatrices

- Pas de gène présent plusieurs fois dans un génome
- Tout gène apparaissant dans G_1 apparaît dans G_2

Exemple

$G_1 = +1 + 3 - 4 + 5 + 2 - 6$
$G_2 = +1 + 2 - 5 - 3 + 6 - 4$

Comment comparer deux Génomes ?

2 grandes familles:

1 - Distances de Réarrangement

- On se dote d'*opérations* permises (ex: inversion, transposition)
- On cherche à déterminer le nombre *minimum* d'opérations permettant de passer de G_1 à G_2 + scénario de réarrangement correspondant
- Suivant le principe de parcimonie
- *Remarque* : les opérations peuvent éventuellement être pondérées

Comment comparer deux Génomes ?

2 grandes familles:

2 - Mesures de similarité

- Se base sur la *structure* des génomes
- On calcule une *mesure* qui traduit la proximité (ou non) des deux génomes
- Ne permet pas de reconstruire un scénario de réarrangement
- Exemples: breakpoints, intervalles communs

Intervalles communs

Définitions

- Un *intervalle* d'un génome G_1 est une suite d'éléments consécutifs de G_1.
- Un intervalle I_1 de G_1 est dit *commun à* G_2 s'il existe dans G_2 un intervalle I_2, où I_1 et I_2 contiennent les mêmes éléments au signe près.

Exemple

$$G_1 = +1\ \underline{+2\ +3\ +4\ +5}\ +6\ +7$$

$$G_2 = -7\ +6\ +1\ \underline{+5\ -3\ +2\ +4}$$

L'intervalle $I_1 = +2\ +3\ +4\ +5$ de G_1 est commun à G_2

Sans Duplications vs Avec Duplications

En résumé

- Comparer des génomes est souvent "simple" (=polynomial) lorsque ceux-ci ne présentent pas de duplications
- Or, cette hypothèse est trop restrictive
  - $\sim$ 15% chez l'homme
  - $\sim$ 16% chez la levure
  - $\sim$ 25% chez Arabidopsis
- Comment en tenir compte ?

Plan de l'exposé

1 Parlons Algorithmique

2 Réarrangements Génomiques

3 **Gènes Dupliqués**

4 Quelques Résultats

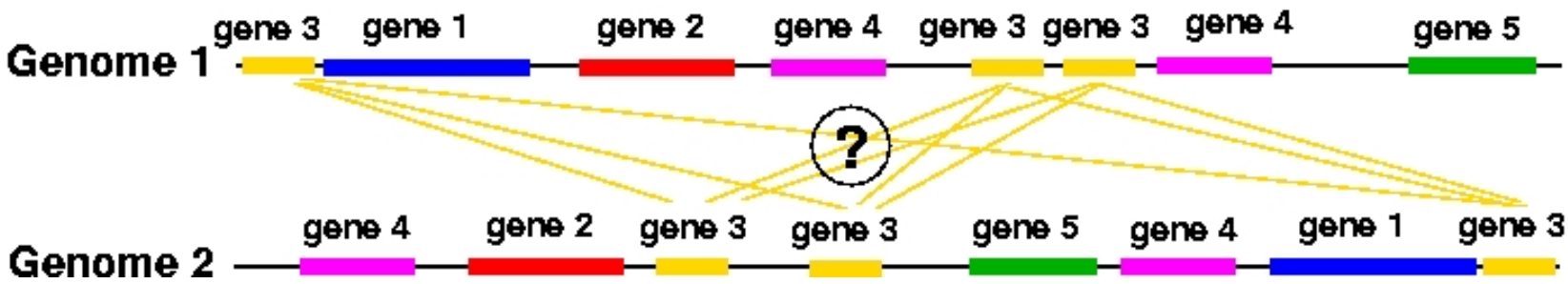
5 Notre Solution (Travaux en Cours)

Un premier problème

Sans duplication, pas d'ambiguïté

Un premier problème

Avec duplication, comment faire ?



Gestion des ambiguïtés

Pour chaque famille de gènes dupliqués

- faire correspondre les gènes 1 à 1
- éventuellement en supprimant des gènes

Deux façons de procéder

- **Exemplarisation** entre G_1 et G_2
- **Matching** de taille maximale entre G_1 et G_2

Exemplarisation vs Matching

Exemple

$$G_1 = +1\ +2\ +3\ -2\ +5\ +4\ +4$$

$$G_2 = -1\ +2\ +1\ +5\ -3\ +2\ +4$$

Version Exemplaire

Ne garder qu'**un seul** gène de chaque famille dans G_1 et G_2Il faut lever les ambiguïtés pour chaque famille de gènes dupliqués

Version Matching

Faire correspondre le nb **maximum** de gènes entre G_1 et G_2

Version Exemplaire

Exemple

$$G_1 = +1\ +2\ +3\ -2\ +5\ +4\ +4$$

$$G_2 = -1\ +2\ +1\ +5-3\ +2\ +4$$

Une exemplarisation:

$$G'_1 = +1\ +2\ +3\ +5\ +4$$

$$G'_2 = +1\ +5-3\ +2\ +4$$

La correspondance 1-1 est implicite

Version Matching

Exemple

$$G_1 = +1\ +2\ +3\ -2\ +5\ +4\ +4$$

$$G_2 = -1\ +2\ +1\ +5\ -3\ +2\ +4$$

Un matching maximal:

$$G'_1 = +1\ +2\ +3\ -2\ +5\ +4$$

$$G'_2 = +2\ +1\ +5\ -3\ +2\ +4$$

Version Matching

Exemple - Suite

Un matching maximal:

$$G'_1 = +1\ +2\ +3\ -2\ +5\ +4$$

$$G'_2 = +2\ +1\ +5\ -3\ +2\ +4$$

- Reste encore à lever l'ambiguïté pour le gène 2
- Demande en plus d'établir une *correspondance 1-1* entre gènes d'une même famille, pour toutes les familles

Exemplarisation ou Matching

Quel exemplarisation/matching choisir ?

- Pour toute instance (G_1, G_2), il y a un grand nombre de façons d'effectuer un matching ou une exemplarisation
- Il faut imposer un **critère de choix**
- **Critère**:
  - pour une distance ou une mesure de similarité D donnée
  - trouver le matching/l'exemplarisation qui **optimise** D
- Principe de parcimonie

Plan de l'exposé

1 Parlons Algorithmique

2 Réarrangements Génomiques

3 Gènes Dupliqués

4 **Quelques Résultats**

5 Notre Solution (Travaux en Cours)

Résultats

Quelques définitions

Soit G un génome.

- $occ(G, x)$: nombre d'occurrences du gènes x dans G
- $occ(G)$: maximum des $occ(G, x)$ pris sur toutes les familles x de gènes
- $f(G)$: nombre de famille de gènes différentes dans G

Résultats

Exemplarisation: résultats issus de:

- Bryant-2000
- Chen,Zheng,Fu,Nan,Zhang,Lonardi,Jiang-2004
- Chauve,Fertin,Rizzi,Vialette-2005

Matching: résultats issus de:

- Bryant-2000
- Chen,Zheng,Fu,Nan,Zhang,Lonardi,Jiang-2004
- Blin,Chauve,Fertin-2004
- Chauve,Fertin,Rizzi,Vialette-2005

Exemplarisation - Résultats

Exemplarisation	
Mesure	**Complexité**
Breakpoints	NP-dur même quand $occ(G_1) = 1$ et $occ(G_2) = 2$ même quand $f(G_1) = f(G_2) = 1$
Inversions	NP-dur même quand $occ(G_1) = occ(G_2) = 2$
Intervalles Communs	NP-dur même quand $occ(G_1) = 1$ et $occ(G_2) = 2$ même quand $f(G_1) = f(G_2) = 1$

Matching - Résultats

Matching	
Mesure	**Complexité**
Breakpoints	NP-dur même quand $occ(G_1) = 1$ et $occ(G_2) = 2$ même quand $f(G_1) = f(G_2) = 1$
Inversions	NP-dur même quand $occ(G_1) = occ(G_2) = 2$
Intervalles Communs	NP-dur même quand $occ(G_1) = 1$ et $occ(G_2) = 2$ même quand $f(G_1) = f(G_2) = 1$

Plan de l'exposé

1 Parlons Algorithmique

2 Réarrangements Génomiques

3 Gènes Dupliqués

4 Quelques Résultats

5 **Notre Solution (Travaux en Cours)**

Gènes Dupliqués et Mesures de Similarité

La solution que nous proposons

- Obtenir un résultat **exact**
- Donc un temps exponentiel
- Transformer le problème initial en un problème algorithmique classique
- Dans notre cas: le problème **SAT**
- Profiter de dizaines d'années de recherche concernant SAT
- Et notamment de logiciels **très puissants** (SAT solvers)
- SAT solvers capables de gérer de grandes données et de répondre rapidement

Gènes Dupliqués et Mesures de Similarité

Problème SAT

- Problème de logique booléenne (VRAI/FAUX)
- Variables booléennes x_1,$x_2 \ldots x_n$
- Clauses $C_1, C_2 \ldots C_q$: $C_4 = x_1$ ou $\overline{x_3}$ ou $x_1 1$ ou $\overline{x_1 3}$
- Expression booléenne: $E = C_1$ et C_2 et $C_3 \ldots$ et C_q
- But: satisfaire l'expression E
- C'est-à-dire déterminer une affectation de chaque variable x_i (à VRAI ou à FAUX)
- Telle que *chacune* des clauses soit vraie

Transformation en Problème SAT

Problème SAT

- Problème "bas niveau" (VRAI/FAUX)
- A la base de la théorie de la complexité algorithmique
- Très étudié depuis $\sim$ 35 ans
- Problème *NP*-dur
- ...mais -paradoxalement-, certaines stratégies permettent d'accélérer énormément les calculs

Transformation en Problème SAT

L'idée principale

- S'il est possible de transformer notre problème en problème SAT
- Utiliser un SAT solver qui saura le résoudre
- "Ramener" la réponse du SAT solver vers notre problème

Où en est-on ?

Les prémices...

- Plusieurs "traductions" vers SAT réalisées sur le papier:
  - Exemplarisation + breakpoints
  - Matching + breakpoints
  - Exemplarisation + intervalles communs
  - Matching + intervalles communs
- Pas toujours traduisible en SAT (ex: distance d'inversion)
- Restent la programmation et les tests (sujets de stage M2R)

Où en est-on ?

Que peut-on en attendre ?

- Écueils possibles:
  - Données trop grosses, même pour le SAT solveur
  - Trop de réponses optimales : laquelle/lesquelles choisir ?
- Avancées possibles:
  - Obtenir une réponse exacte et rapide (via "l'artifice" SAT)
  - Possibilité de moduler entre
    - Exemplarisation (**1** occurrence)
    - Matching (**Maximum** d'occurrences)

Remerciements

Travaux en collaboration avec

- Stéphane Vialette (LRI, Orsay)
- Cedric Chauve (LACIM, UQAM, Montréal)
- Romeo Rizzi (U. Udine, Italie)