Table des matières

Introduction

J'ai abordé ce stage en confiance, j'ai eu le temps de bien le préparer et j'ai eu la chance de pouvoir discuter de mon sujet et le définir en accord avec mon Maître de stage. Je suis arrivé en Allemagne en sachant que j'allais faire quelque chose qui me plairait, une sorte de mélange entre l'informatique, l'ergonomie IHM, les maths et un élément tout neuf pour moi : la Bio-informatique. A la frontière entre la biologie et l'informatique, cette discipline est en plein essor et Xapien en a fait son fond de commerce, comme beaucoup d'entreprise du secteur de Heidelberg. C'est avec un intérêt allant croissant que j'ai découvert une nouvelle saveur du monde informatique, et même si je n'ai pas fait de réelle bio-informatique, j'ai du apprendre les notions de bases de biochimie pour comprendre ce que l'on me demandait.

Maintenant c'est a mon tour de montrer que j'ai profité de cette expérience, mon stage aurait été beaucoup plus délicat sans ces préalables biologiques et probablement, je n'aurais pas réussi a proposer des solutions efficaces aux deux grands problèmes qui m'étaient posés ... On commence dans un petit instant, avec de la ..... biologie, bien sur !

Remerciements

A toute l'équipe de Xapien et en particulier a Alejandro, Venkata, Guillaume, Bernhart, Martin et David, mes collègues de bureau avec j'ai passé 5 mois avec le sourire,
A Guillaume, mon Maître de stage et ami de longue date,
Aux professeurs de l'université de Metz qui m'ont expliqué les problèmes de dessin de graphes,
A Martine Salm qui mérite bien ses vacances,
et aux futurs investisseurs de Xapien, pour que cette compagnie continue d'exister.

Sujet de stage tel que défini au départ

Représentation d'une protéine sous forme de domaine.

Il existe sur le marche de nombreux systèmes de représentation de protéine, sous forme de domaine ou autre mais les hypothèses graphiques choisies ne reflètent pas la fonctionnalité représentée. On me propose de définir une nouvelle méthode graphique pour représenter les domaines, en enrichissant les modèles existants de façon a intégrer les données supplémentaires venant de Xapien et en réfléchissant conjointement avec les biologistes à une signature graphique efficace et lisible.

Représentation automatique de pathway.

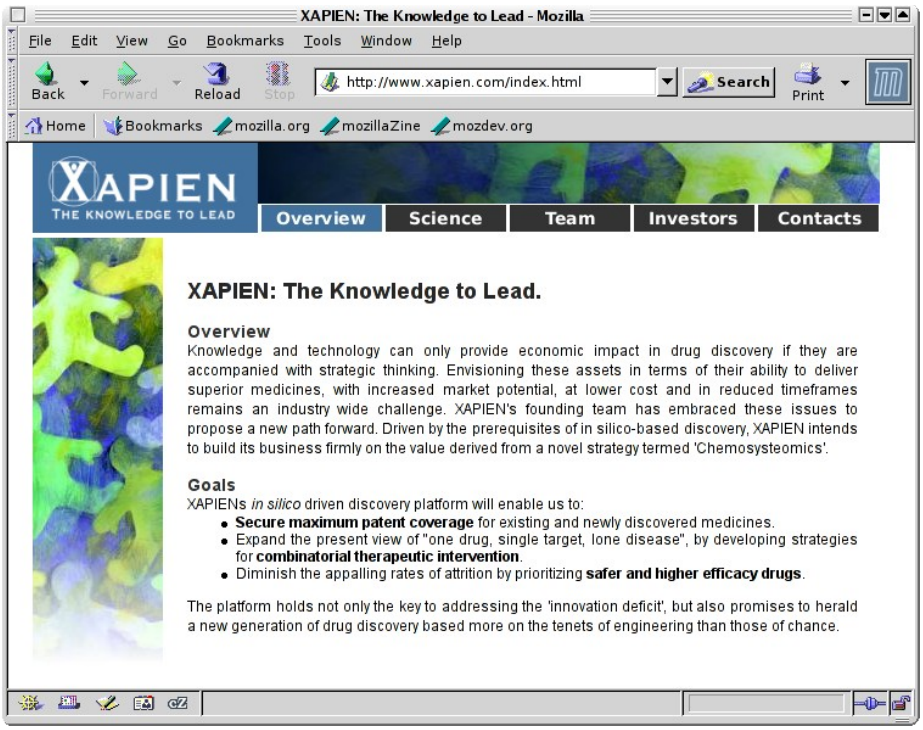
Les pathways biologiques sont représentés par des dessins de graphes pouvant être très complexes. Xapien voudrait mettre en place une méthode automatique ou au moins semi-automatique pour le dessin et l'annotation des graphes de pathway.

En production, la base de donnée devrait augmenter rapidement et il deviendrait vite impossible de dessiner à la main les pathways, comme c'est fait d'habitude. Il s'agit ici de comprendre ce qu'est un pathway, comment il fonctionne et trouver une méthode de dessin optimale en intégrant des contraintes de structure et de positionnement dans le dessin.

Site internet

En plus de ces deux grands projets, j'ai construit la première version du site web, dans un souci d'accessibilité et de respect des standards, bien sur. Comme je n'en parle pas ailleurs, je détaille un peu plus .. C'est un site statique de 5 pages en anglais dont le but est de présenter brièvement l'activité de Xapien et de permettre à des éventuels investisseurs de prendre contact avec la société.

J'ai voulu profiter de cette expérience pour construire un site réellement accessible, en suivant les recommandations du W3C. Le site est entièrement en XHTML strict valide, la mise en page passe par des calques CSS et non des tables, les images sont toutes commentées, le design est sobre et clair. Après des tests sue les navigateurs courants, le site fonctionne très bien, il passe aussi très bien en mode texte et je suppose qu'il pourrait être lu automatiquement sans mal.



chronologie du stage

23 - 27 février	Installation de mon bureau, préparation d'une machine de travail.
1 - 21 mars	Mise a niveau en Biologie, apprentissage des définitions de base, définition plus précise de mon sujet de stage.
22 mars - 4 avril	Représentations des protéines : étude des systèmes existants, leurs avantages et inconvénients. Propositions d'idées sous forme de dessins. début d'un prototype.
5 - 18 avril	développement du site Web de Xapien (www.xapien.com), graphisme et code pendant une semaine, contenu pendant l'autre semaine.
19 - 30 avril	Représentation de protéines : finalisation du premier prototype, correction et réécriture du code. Implémentation du mode "échelle variable".
3 - 8 mai	Représentation de protéines : version d'exploitation. Tests et corrections, choix du design final.
10 - 16 mai	Pathways : étude des pathways biologiques existants, test des outils automatiques de dessin.
17 - 23 mai	Pathways : étude des algorithmes pour le dessin de graphes, étude des caractéristiques communes des représentations des pathways.
24 - 28 mai	Pathways : écriture (papier) de mon algorithme de dessin, illustration de son fonctionnement par des dessins du résultat. Validation de la méthode par le reste de l'équipe.
31 mai - 12 juin	Pathways : premier prototype (pré simplification du graphe, détection et enregistrement des éléments, algorithme de pré-placement simple), test et corrections
14 - 19 juin	Pathways : réécriture du code, révision de l'algorithme de pré-placement (optimisation), écriture d'un jeu de test. Ajouts de la simplification de graphes par les noeuds virtuels.
21 - 27 juin	Pathways : second prototype, corrections de nombreux bogues, version finale de l'algorithme de pré-placement.
29 juin - 4 juillet	Pathways : étude des algorithmes dynamiques pour l'optimisation du placement des sous ensembles du graphes. écriture (papier) de mon algo "spring embed", validation par mon tuteur.
5 - 11 juillet	Pathways : ajout des méthodes pour le déplacement et la rotation des sous-arbres du graphe (beaucoup de géométrie).
12 - 17 juillet	Pathways : Détermination des points d'encrage des ressorts pour l'algo "spring embed". Nettoyage du code et commentaires. Je dois m'arrêter là.
19 - 31 juillet	Écriture du rapport de stage et préparation du diaporama pour la soutenance.

Notions biologiques de base

La biologie à été un domaine entièrement nouveau pour moi. Il a fallu que j'apprenne les bases du fonctionnement de la cellule, le rôle des différents éléments qui la composent pour bien pouvoir comprendre le sujet de mon stage et être sûr d'avancer dans la bonne direction. Je me propose dans cette première partie d'éclaircir les concepts essentiels en biologie cellulaire utile pour la compréhension du reste de mon stage. J'ai passe la majeure partie de mon premier mois de stage à explorer ce nouveau monde.

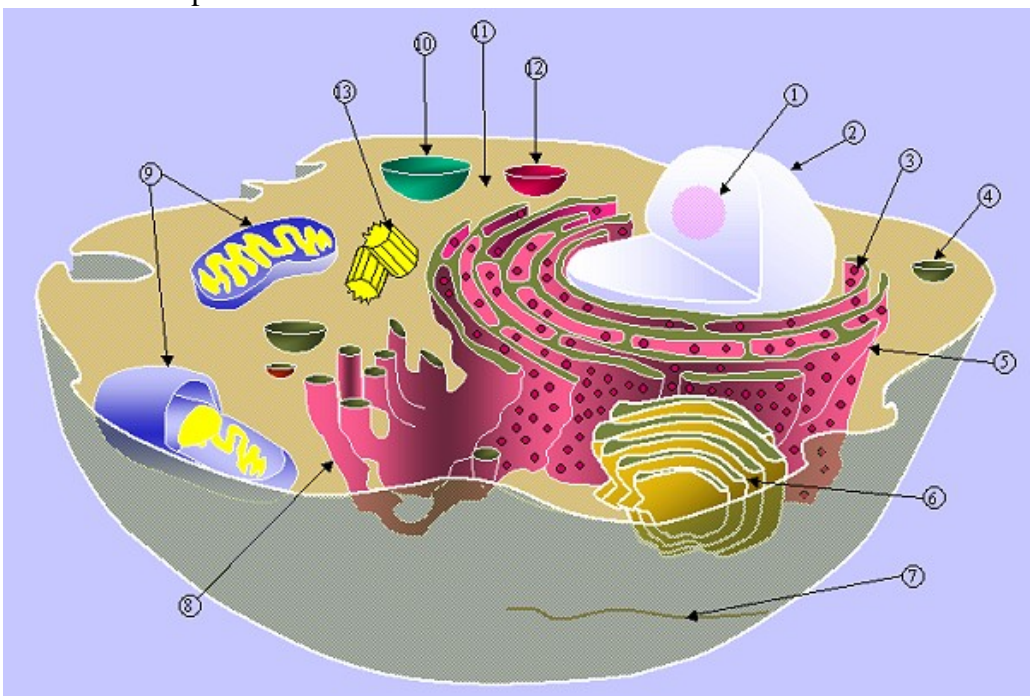
La cellule

La cellule est l'unité de base du vivant. Tout organisme, du plus simple au plus perfectionné est constitué de cellules, parfois une seule, parfois plusieurs dizaines, voire plusieurs milliards de types différents dans un ensemble très coordonné. Le mot cellule vient de la « cellule » monastique (car découverte par un moine). Il existe 3 lignées et 220 types de cellules qui les rendent différentes entre elles. Toutes celles que nous connaissons (mis à part quelques types spéciaux de cellules) contiennent cependant certains composants communs :

l'ADN, les protéines, les membranes cytoplasmiques, qui isolent la cellule de son environnement, agissent comme un filtre ou un système de communication avec l'extérieur, et compartimentent les cellules les plus complexes

Les cellules ont également en commun certaines capacités:

- la reproduction cellulaire, par division de la cellule
- le métabolisme cellulaire, utilisant de la matière brute et de l'énergie pour produire des composants de la cellule, et rejetant des produits dérivés
- la synthèse des protéines, par la transcription de l'ADN en ARN puis par la traduction par les ribosomes de l'ARN en protéine.



organisation d'une cellule typique (1. Nucléole, 8. Réticulum endoplasmique lisse, 2. Noyau, 9. Mitochondrie, 3. Ribosome, 10. Vacuole, 4. Vésicule, 11. Cytoplasme, 5. Réticulum endoplasmique rugueux (RER), 12. Lysosome, 6. Appareil de Golgi, 13. Centrioles, 7. Microtubule)

ADN, gènes et chromosomes

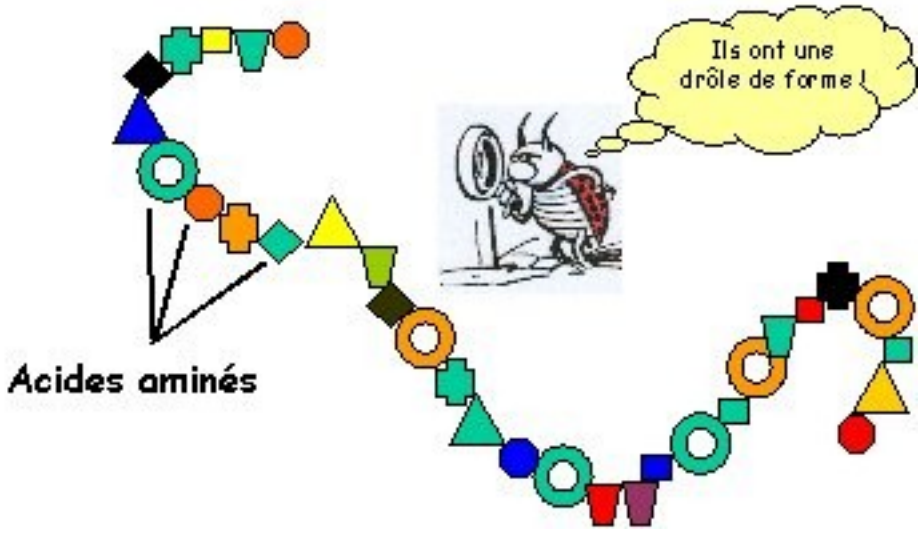
L'ADN, acronyme de acide désoxyribo-nucléique, est une longue molécule (groupement d'atomes) que l'on retrouve dans tous les organismes vivants. L'ADN est présent dans le noyau des cellules. On peut imaginer que la molécule d'ADN est le livre de recettes de toute cellule vivante. C'est là que se trouvent toutes les information nécessaires à la production des protéines dont les cellules vivantes ont besoin. Pour produire une protéine, des enzymes (protéines spécialises permettant une réaction chimique) vont lire l'ADN comme un livre et construire les protéines en suivant la recette décrite par l'ADN.

Les chaînes d'ADN. sont organisées en unités de fonction appelées "gènes", responsables de la synthèse d'une protéine ou d'une fraction de protéine. Les gènes ont une longueur moyenne de 1 000 à 2 000 paires de nucléotides et codent pour 300 à 600 acides aminés.

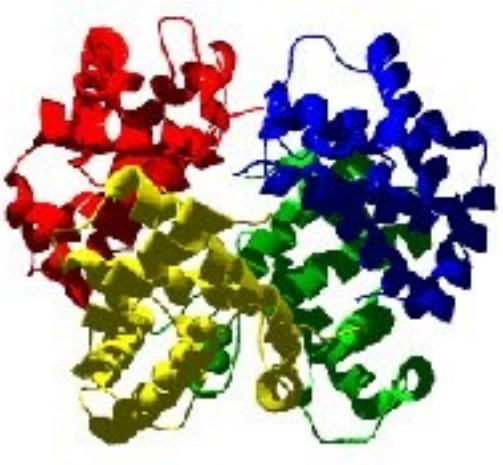
Les gènes sont portés par les chromosomes et occupent sur ceux-ci des emplacements fixes. Les chromosomes sont des filaments porteurs de l'information génétique. Ils sont constitués de protéines et d'ADN et logés dans le noyau de la cellule. Leur nombre est constant dans une espèce donnée (2 fois 23 chez l'être humain); un seul chromosome différenciant l'homme de la femme.
On peut dérouler environ deux mètres de ADN par cellule.

protéine

Les protéines sont des molécules essentielles dans la construction et le fonctionnement de tout les êtres vivants. On peut les comparer à des ouvriers spécialises, dans la mesure où elles assurent presque toutes les fonctions d'un organe. Parmi les plus connues on trouve les protéines de défenses, les fameux anticorps qui défendent l'organisme contre les virus et bactéries ; les protéines de transport comme l'hémoglobine qui assure le transport de l'oxygène ; les protéines structurelles comme le collagène ; les enzymes, qui ne sont pas utiles qu'à la lessive.



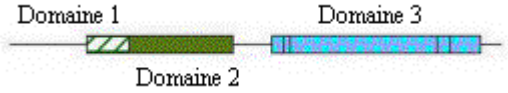
Une protéine est un long assemblage d'acides aminés (il en existe 20 différents), ils sont habituellement code par une lettre. L'ordre de succession des acides aminés dans la protéine s'appelle la séquence d'une protéine. Les protéines peuvent comporter entre 25 et 10000 acides aminés et être constitués de plusieurs chaînes qui s'assemblent et se replient sur elles mêmes en une structure tri-dimensionnelle particulière. Cette forme est essentielle pour le rôle de la protéine (c'est par exemple la forme de l'hémoglobine, sorte de velcrot à oxygène qui permet le transport par le sang).



Vue structurelle de la protéine

Caractérisation fonctionnelle des protéines : Les domaines

Une protéine effectue rarement une tache unique, elle peut par exemple se lier à la membrane et en même temps se lier à une autre protéine. Des différentes fonctions sont localisées dans des régions bien précises de la protéine appellées domaines. Si les acides aminés peuvent être considérés comme les lettres de l'alphabet, les domaines correspondent au mots d'une phrase.



ces domaines sont souvent conserves (même arrangement d'acides aminés) entre protéines dont la fonction est similaire.

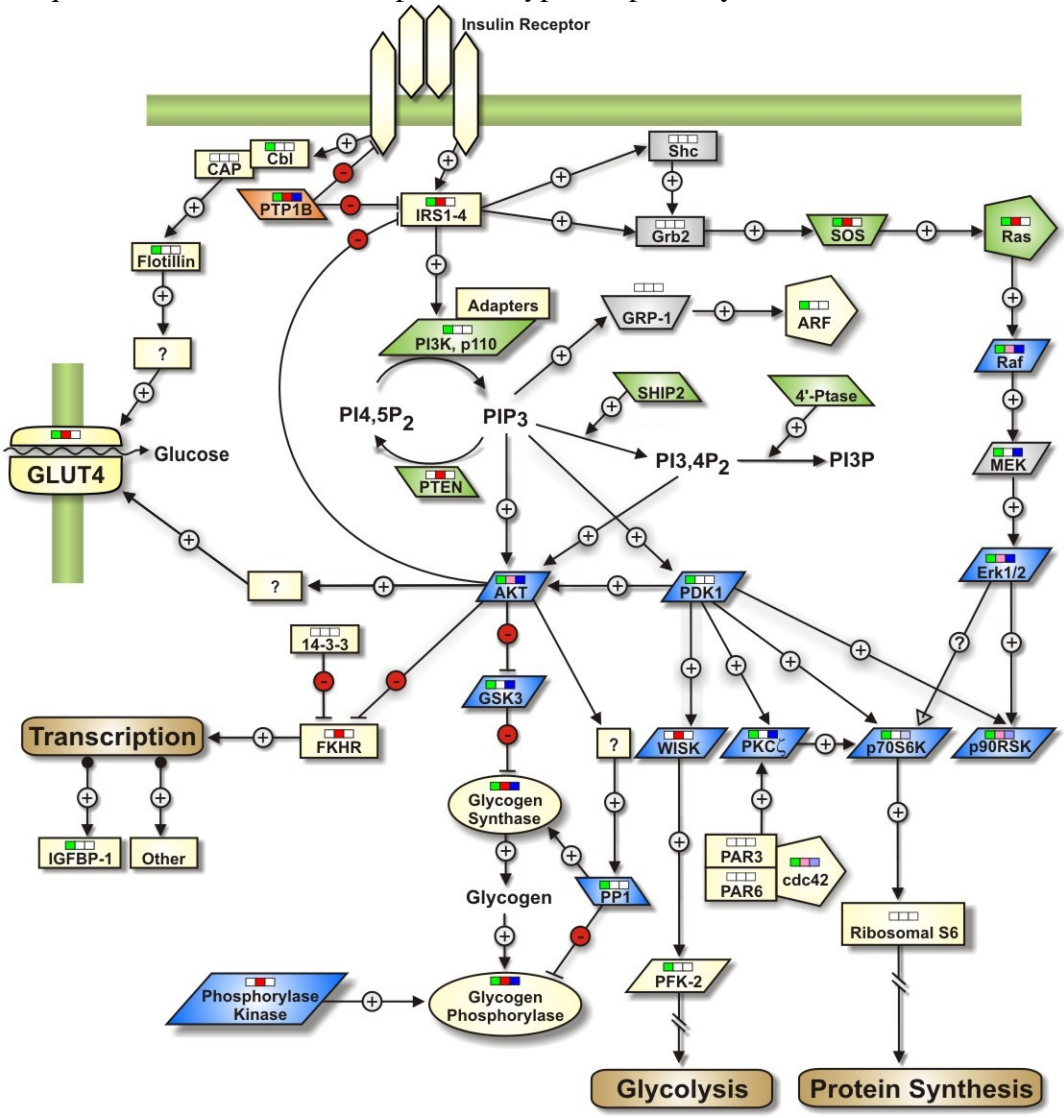
```
                          *         20          *         40
TMS3_HUMAN : DC--SGKYRCRSSFKCIELIARCDGVSDCKDGEDEYR--C :  36
TMS2_MOUSE : NC-STSEMECGSSGTCISSSLWCDGVAHCPNGEDENR--C :  37
TMS4_HUMAN : VILDKYYFLCGQPLHFIPRKQLCDGELDCPLGEDEEH--C :  38
TMS2_HUMAN : KC-SNSGIECDSSGTCINPSNWCDGVSHCPGGEDENR--C :  37
ENTK_BOVIN : PC-KEDNFQCKD-GECIPLVNLCDGFPHCKDGSDEAH--C :  36
ENTK_HUMAN : PC-KADHFQCKN-GECVPLVNLCDGHLHCEDGSDEAD--C :  36
ENTK_MOUSE : PC-QDDEFQCKD-GNCIPLGNLCDSYPHCRDGSDEAS--C :  36
ENTK_PIG-- : PC-KEDNFQCEN-GECVLLVNLCDGFSHCKDGSDEAH--C :  36
GRAAL_DRO- : KC-PNNYWLCHTSKECIPPAFVCDNTPDCADKSDECAAVC :  39
TEQUIL_DRO : KC-PNNYWLCHTSKECIPPAFVCDNTPDCADKSDECAAVC :  39
CORI_HUMAN : GCKERDLWECPSNKQCLKHTVICDGFPDCPDYMDEKN--C :  38
Q9Z319---- : GCKERALWECPFNKQCLKHTLICDGFPDCPDSMDEKN--C :  38
Q9NJS5---- : SC-PQDYWLCHASEECIPVQFLCDNVRDCADGSDESPDHC :  39
Q9NAT0---- : SC-PQDYWLCHASEECIPVQFLCDNVRDCADGSDESPDHC :  39
Q9U1I3---- : KC-PNNYWLCHTSKECIPPAFVCDNTPDCADKSDECAAVC :  39
Q9V4N6---- : KCD---GFQCDQ-NRCLPQEYVCDGHLDCMDQADEAK--C :  34
Q9Y1V3---- : DCAATNRYLCND-GSCIEHDQVCDFRDDCPNGEEETE--C :  37
Q9NRS4---- : VILDKYYFLCGQPLHFIPRKQLCDGELDCPLGEDEEH--C :  38
               c        C      c6       CD     C     dE     C
```

A partir d'une nouvelle protéine on peut identifier des domaines fonctionnels en recherchant des ressemblances avec des domaines dont la fonction biologique est déjà connue. C'est un travail informatique lourd, il s'agit de comparer entre eux des chaîne d'acides aminés pour trouver des similitudes.

Les protéines représentées sous forme de domaine sont une première simplification (une abstraction) par apport a la réalité chimique. On distingue deux types de représentation par domaine : la vue par activité où les domaines représentent des enzymes et la vue par liant ou le domaine représente l'élément avec lequel il interagit.

Pathway

En biologie cellulaire, un pathway est un ensemble de successif d'interaction entre protéines et éléments chimique à l'intérieur et à l'extérieur d'une cellule et qui correspond à un mécanisme du vivant. Un pathway renseigne sur les réactions chimiques qui régissent une fonctionnalité de la cellule. Il existe ici aussi plusieurs vues pour un même pathway. la vue métabolique explique les étapes de fabrication ou de modification d'une molécule, d'un compose chimique à l'intérieur de la cellule ; un pathway de signalisation montre la cascade d'évènements qui se produisent dans une cellule à la réception d'un signal extérieur à la cellule. Il existe encore d'autre types de pathways pour montrer par exemple le chemin parcouru par une protéine entre le moment de sa création et sa sortie de la cellule. Xapien s'intéresse uniquement à l'étude des deux premiers types de pathways.



prototype de représentation d'un pathway de signalisation Xapien

Xapien

A mi-chemin entre le monde de la recherche et celui de l'industrie, Xapien est une entreprise allemande initiée en début d'année par quatre membres fondateurs venant de la biologie et de l'informatique. Son domaine d'investigation est la recherche sur les molécules de médicaments
Pour y arriver, Xapien veut mettre en place des outils de tests et d'analyse, jusqu'ici réservé à la recherche, dans un système de automatisé.

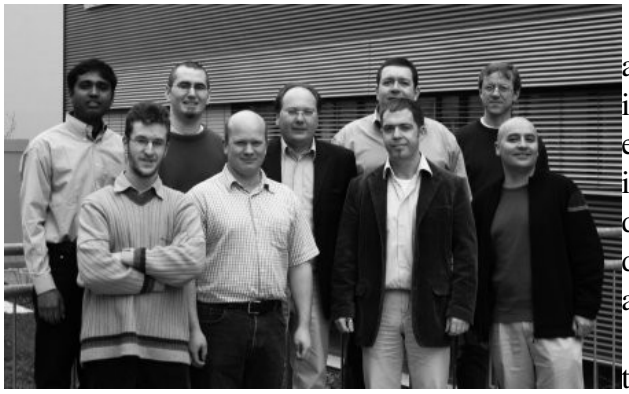
D'un point de vue bio-medical, Xapien cherche à comprendre les modes d'actions des médicaments sur l'ensemble du corps humain. La clef de cette approche est de comprendre les médicaments non pas en s'intéressant à leur structure ou leur propriétés physico-chimiques mais plutôt en considérant leurs effets au niveau biologique. Cette approche est novatrice n'a jamais été expérimentée à grande échelle. Elle doit permettre la prédictions des toxicité d'un médicament, la découverte de combinaisons de molécules existantes pour traiter plusieurs maladies en un même temps et enfin la découverte de nouveaux médicaments.

Fonctionnement

Le fonctionnement de Xapien en production se divise en trois grandes étapes :

- Design des expériences a partir de statistiques, de données existantes, de prédictions ...
- L'expérimentation biologique des nouvelles molécules sur des processeurs contenant des protéines humaines doit permettre de détecter quel médicament est assimilé par quelle protéine. Xapien travaille dans ce domaine avec une société spécialisée dans l'analyse de surface capable de détecter les très faibles variations de taille de la protéine générées par l'assimilation de la molécule testée.
- Une fois que les expérimentations biologiques ont été menées, on obtient une base de données de l'interaction des protéines avec des molécules de médicaments et à partir de cette base, les outils informatiques et bio-informatiques développés en interne doivent permettre de simuler le comportement général des cellules du corps humain. Une grande partie du travail bio-informatique concerne la restitution des informations en provenance des tests, recoupés avec les informations déjà en base de données. J'ai travaillé pendant mon stage sur la partie visualisation graphique de données du prototype, partie un peu moins biologique et plus orientée IHM, d'autres membres de l'équipe on fait du traitement comparaison de chaîne de protéine, d'autre encore du data-mining en provenance de bases de données du marché.

Équipe



l'équipe de Xapien est internationale : trois allemands, un irlandais, un écossais, un chilien, un indien et deux français et représente une longue expérience dans la recherche biologique et informatique. Leurs efforts réunis ont résulté dans plus de 25 brevets et plus de 100 publications de recherche dans des revues scientifiques. Nous travaillons tous en anglais.

Idéalement située a Heidelberg, ville d'histoire et de technologie, Xapien est au coeur de l'un des plus grands pôles pour les technologies biologiques en Europe. A Heidelberg se trouve aussi le laboratoire européen de biologie moléculaire (EMBL), le DKFZ : centre de recherche allemand sur le cancer et toute une foule d'entreprises tournant autour du domaine des Bio-technologies. La faculté de Heidelberg a aussi un important département de biologie. Toutes les ressources nécessaires (technologie, compétences) sont disponibles sur place. Tous les membres de l'équipe ont d'ailleurs eu une précédente expérience dans d'autres entreprises de Heidelberg.

Comme toute entreprise qui débute, le principal problème est le financement. Même si les

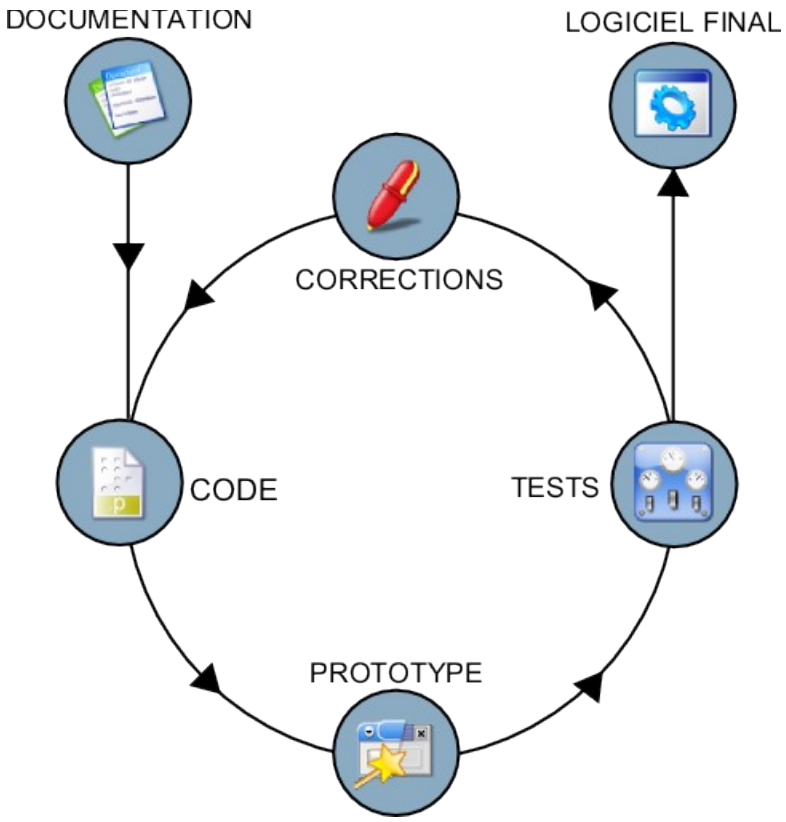
sources semblent nombreuses, c'est l'un des point les plus délicats à gérer, j'ai vu pendant mon stage passer beaucoup d'investisseurs potentiels, tous très intéressés mais très peu vont finalement investir. Même si Xapien a pu apporter les preuves de la méthode, aucun investisseur ne s'est encore décidé concrètement et l'équipe n'est toujours pas payée.

Méthodes et outils pour le développement.

Avant de plus développer les deux grandes parties de mon stage, je précise ici comment s'est effectué la conception et la mise en oeuvre de mes projets. Les deux projets, bien que différents reposent aussi sur les mêmes bases techniques. J'expliquerai dans cette partie aussi la pertinence de ces choix par rapport aux autres solutions qui auraient pu être adoptées.

Conception

Mon stage, même s'il reste très technique n'aboutit au final que sur de la génération d'image. Je n'ai pas eu de cahier des charges particulier et cela a été a moi de proposer des solutions. J'ai vite abandonné une démarche de conception classique type UML. L'absence de cahier des charges et ma méconnaissance du domaine biologique rendant quasi impossible une conception classique.
J'ai travaille selon le cycle suivant :



> Documentation : étude de l'existant, quelles sont les avantages, les inconvénients de ces solutions et dans quelle mesure celles-ci sont-elles exploitables dans ma solution.
> Code : développement graphique dans un premier temps puis codage dans les cycles suivants selon ce que j'ai pu apprendre de l'existant et l'idée que je me fais du programme.
> Prototype : Finalisation d'un prototype du programme dont les fonctionnalités vont être améliorées lors des prochains cycles.
> Tests : Avec les membres de l'équipe, nous testons et critiquons le prototype proposé. selon l'état de l'art, on décide de repartir sur un autre cycle ou alors le logiciel est près pour une version d'exploitation.
> Corrections : reprise du code du prototype et enrichissement, correction, débugage suivant le bilan de la phase de test.

A un moment ou l'autre dans chacun des deux projets, j'ai eu à restructurer complètement le code, cela intervient quand la complexité augmente de trop pour pouvoir continuer sur les bases du premier prototype. Une fois le code repris, on gagne en qualité et en lisibilité puisque nous avons déjà intègre beaucoup des contraintes du projet.

Les cycles durent Idéalement une quinzaine de jour. Cela a très bien fonctionné pour la représentation de protéines ou il n'y avait pas de problèmes de développement très important. Ces cycles

ont été beaucoup plus longs pour la représentation de pathways où il faut beaucoup plus de temps pour concevoir un prototype fonctionnel.

La phase de documentation a aussi été assez importante, j'ai trouve beaucoup d'information sur le Web, j'ai lu des papiers, je me suis documenté auprès des biologistes de l'équipe. Pour les pathways, c'est durant cette phase de documentation qu'il a fallu décider d'une méthode de dessin.

Globalement, j'ai codé beaucoup et très tôt, il n'y a pas eu beaucoup de concertation sur une démarche plus formelle. Xapien utilise la méthode Agile pour conception et l'implémentation des logiciels. Cette méthode est déjà très utilisée (par HP entre autres) ; l'idée est de donner la priorité aux versions, sans chercher a tout intégrer au départ et corriger au fur et à mesure que le projet avance.

Technologies retenues pour les deux parties de mon stage

Programmation en PERL

Perl (Practical Extraction and Report Langage ou langage pratique d'extraction et de génération de rapports ; ce nom est un rétro-acronyme) est un langage de programmation créé par Larry Wall en 1987 et reprenant des fonctionnalités du langage C et des langages de scripts sed, awk et shell (sh). Perl est considère comme le couteau suisse des langages de programmation. Sa souplesse autorise l'emploi de plusieurs modèles de programmation : programmation procédurale, programmation fonctionnelle et programmation orientée objet (bien que les puristes de ce dernier ne considèrent pas Perl comme en faisant partie). Son objectif est de s'adapter aux goûts du programmeur plutôt qu'une architecture stricte du langage. Perl est souvent considéré comme le langage de script par définition et a été qualifié de "ciment assurant la cohésion du web", étant un des langages CGI les plus populaires.
Une immense collection de modules Perl d'utilisation libre, allant des mathématiques avancés aux connexions aux bases de données, en passant par les réseaux et bien davantage encore, peuvent être téléchargés depuis un réseau de sites appelé CPAN.

Bien que Perl profite de la plupart des facilités d'un langage interprété, à proprement parler il n'interprète et n'exécute pas le code source une ligne à la fois. En fait, Perl compile d'abord le programme entier dans un bytecode intermédiaire (assez dans l'esprit du code objet Java), l'optimisant au passage, et exécute alors ce bytecode. Il est possible de compiler un programme Perl en bytecode pour s'épargner les phases de compilation lors d'exécutions futures, bien que l'"interpréteur" soit toujours requis pour exécuter ce code.

Perl est un langage toujours en cours de développement, nous en somme aux versions 5 et le langage continue doucement mais sûrement son évolution vers la version 6 qui constituera la version finale.

Perl se prête très bien aux problèmes de bio-informatiques, le langage permet un traitement très rapide des chaînes de caractères, il intègre les expressions rationnelles, des opérations de substitution de concaténation, et toutes sortes de fonctions de comparaison. De plus, CPAN propose un package nommé BioPerl qui est la librairie de référence en bio-informatique, elle contient tout ce qui est nécessaire aux biologistes.

J'ai choisi d'utiliser Perl car c'est le langage qui m'est le plus familier et que j'étais sûr de trouver dans CPAN touts les modules nécessaires à la visualisation, au rendu géométrique et à la recherche d'information dont j'avais besoin.

perl.com, ressource n°1 pour la programmation en Perl


Perl.com: The Source for Perl – perl development, perl conferences - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://www.perl.com/
O'REILLY perl.com THE SOURCE FOR PERL
Novell & O'REILLY present The Great Linux Desktop Migration Contest
LEARN MORE
SHARE YOUR OPEN SOURCE STORY AND ENTER TO WIN A TRIP TO NOVELL BRAINSHARE EUROPE '04
O'Reilly Network | oreilly.com | Safari Bookshelf | Conferences
Sign In/My Account | View Cart
Articles | Weblogs | Books | Learning Lab | News
O'Reilly Open Source Convention: July 26-30, Portland, OR.
Search
Go
Advanced Search »
Product Showcase
BEA Learning Channel
Perl.com
Downloads
Documentation
CPAN Mirror
FAQs
Training
Resources
Article Archive
O'Reilly Books
Search
Discover, Develop, Deliver
OSCON 2004
July 26 - 30
Portland, OR
Perl Columns
PERL pVoice
Writing accessible GUIs.
PERL Autopilots
Talk to X-Plane with Perl and Curses.
ONLamp
LINUX Sharp Zaurus SL-6000L Review
PDA or PC?
This Week on Perl 6, Week Ending 2004-07-04
Work begins on a Perl 6 regexp parser, and Unicode manipulation of strings prompts discussion on the language list. [Perl.com]
Building a Mailing List
There are dozens of free and proprietary systems for managing mailing lists, ranging from the trivial to bulletproof and exceedingly full-featured. Nearly all of them face the same challenges to provide the same behavior. Glenn Fleishman recently built his own LAMP-based mailing list in a few hours; here's how he did it. [ONLamp.com]
Application Design with POE
Matt Cashner provides a high-level introduction to POE, the Perl Object Environment, examining the concepts that POE brings to bear when designing long-running Perl applications. [Perl.com]
This Week on Perl 6, Week Ending 2004-06-27
Getting ready for the Piethon is a major concern, while the language list deals with various ways of modifying and annotating expressions. [Perl.com]
Profiling Perl
How do you know what your Perl programs are spending their time doing? How do you know where to start optimizing slow code? The answer to both these questions is "profiling," and Simon Cozens looks at how it's done. [Perl.com]
This Week on Perl 6, Fortnight Ending 2004-06-21
PERL RECIPE OF THE DAY
You have an aggregate data structure, such as an array or a hash. You want to know how often each element in the array (or value in the hash) occurs. For instance, if your array contains web server transactions, you might want to find the most commonly requested file. If your hash maps usernames to number of logins, you want to find the most common number of logins.
Do it now.
PERL VERSIONS
• Stable is 5.8.5.
• Latest is 5.8.5.
• Devel is 5.9.1.
WEBLOGS
WEBLOGS.OREILLY.COM
Sponsored By:
Novell O'REILLY
Share Your Open Source Success Story
The Great Linux Desktop Migration Contest
Enter to WIN A TRIP to Novell BrainShare Europe

SVG

SVG (Scalable Vector Graphics) est une norme du W3C basée sur le XML qui définit deux choses :

- un format de fichier permettant de créer des objets graphiques vectoriels en deux dimensions, des images ainsi que du texte. Il est possible de générer des objets à partir d'autres objets, d'appliquer des filtres, ...
- une API de programmation pour créer des applications. Il est possible d'agir sur ces objets pour en modifier les propriétés dynamiquement soit interactivement par des actions de l'utilisateur, soit par des scripts.

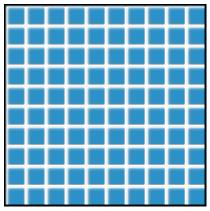
SVG permet la manipulation de trois types d'objets : des formes vectorielles, des images, du texte. Ces objets graphiques peuvent être groupés, transformés, composés dans d'autres objets et recevoir des attributs de style. tous les graphiques SVG sont zoomables ; ils reposent sur des standards du W3C
quelques exemples simples de SVG et leur résultat :

```
<svg width="100" height="100" xmlns="http://www.w3.org/2000/svg" >
    <rect x="10" y="10" width="80" height="40" fill="blue"/>
</svg>
```

```
<svg width="100" height="100" xmlns="http://www.w3.org/2000/svg" >
    <text x="50%" y="50%" font-family="sans-serif">Hello!</text>
</svg>
```

Hello!

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg" >
    <defs>
        <pattern id="pattern" x="0" y="0" width=".1" height=".1">
            <rect x="2" y="2" width="16" height="16" fill="#3399cc"/>
        </pattern>
    </defs>
    <rect x="0" y="0" width="100" height="100" fill="url(#pattern)" stroke="black"/>
</svg>
```



SVG apporte beaucoup d'avantages par rapport à des solutions comme flash :

- compatible xml : on peut l'intégrer dans une page html nativement
- supporté dans mozilla ou konqueror : le SVG est rendu directement par le navigateur
- scriptable : il est possible d'utiliser javascript ou ecmascript pour rendre les objets dynamiques
- supporté par Perl : la librairie SVG.pm permet d'écrire du SVG depuis un code Perl via une interface objet très bien documentée.
- Batik : c'est le renderer SVG du projet Apache. En plus du moteur de rendu, Batik propose un toolkit pour java permettant la transformation du SVG vers d'autres formats comme les images statiques .png ou .jpg.
- La communauté est de plus en plus active, le support peut être trouvé très rapidement

représentation de protéines

Ce qui a motivé l'équipe de Xapien a concevoir sa propre structure pour la représentation de protéines provient du fait que la plupart des systèmes existants n'apportent pas réellement d'informations et sont justes de belles images ; il n'existe d'ailleurs aucun standard de représentation même si certains scientifiques en ont propose une. Les dessins doivent aussi être les mêmes entre la vue d'une proteine et les domaines représentés sur un pathway.

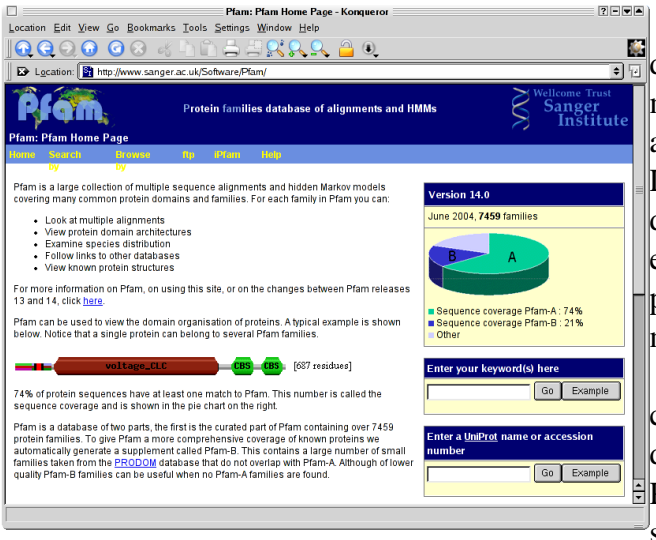
cahier des charges de départ :

- Les images doivent être simples à lire et contenir le maximum d'information utile sur le domaine possible. Il faut trouver une représentation graphique efficace.
- Les domaines peuvent et doivent se classer en famille. Il faut pouvoir distinguer chaque famille facilement.
- Un domaine est soit actif, c'est à dire qu'il est réactif à des molécules provenant par exemple de médicaments soit il est inactif. Xapien ne s'intéresse qu'à ce premier type.
- la représentation graphique choisie doit pouvoir être réutilisée sans changement dans la représentation des pathways.
- les informations caractérisant un domaine sont : famille, nom, est-il ou non actif, lui connaît-on des applications médicales.
- Deux types de représentations sont possibles : une vue par activité et une vue par liant. les notations graphiques doivent être propre à chaque vue.
- La lecture de la représentation doit être possible dans une majorité de cas, en donnant par exemple une importance plus grande aux domaines réactifs.
- Tenir compte des différences d'échelles très importantes et représenter aussi bien une petite protéines (25-100 amino-acides) qu'une très grande (+10000 amino acides).
- Optimiser le dessin pour la taille de l'écran. Faire tenir l'image dans la place disponible.

Étude des systèmes existants :

Les bibliothèques de domaines sont des systèmes contenant toutes les informations nécessaires à la compréhension des protéines. la représentation graphique des domaines n'est qu'une petite partie de ces bibliothèques qui contiennent en fait beaucoup plus d'informations.

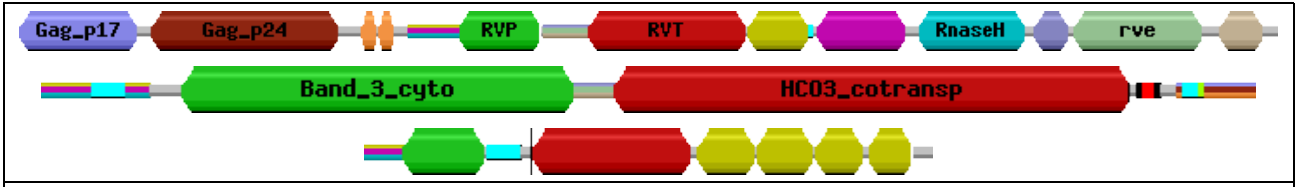
Pfam (http://www.sanger.ac.uk/Software/Pfam/) :



Pfam est une collection de protéines et de domaines qui permet de lancer des recherches par mots clef, par alignement de séquence d'amino-acides, par séquence d'ADN, par nom de domaine. Les protéines et domaines sont aussi répertoriés dans des catégories que l'ont peut accéder par génome, par espèces, par famille. Toutes les descriptions de protéines et de domaines sont actualisées au fur et a mesure des découvertes des équipes de recherche.

Pour le dessin des protéines, Pfam utilise un classement des domaines en plusieurs catégories, dont les deux plus importantes sont PfamA et PfamB. PfamA est l'ensemble des alignements qui sont répertoriés et validés par Pfam tandis que PfamB sont des éléments prédits mais non encore vérifiés. Dans le dessin, PfamA seul est documenté.

La représentation graphique Pfam autorise aussi le dessin d'un domaine à l'intérieur d'un autre. les domaines trans-membrane ou les peptides sont dessinés suivant une représentation qui leur est propre.

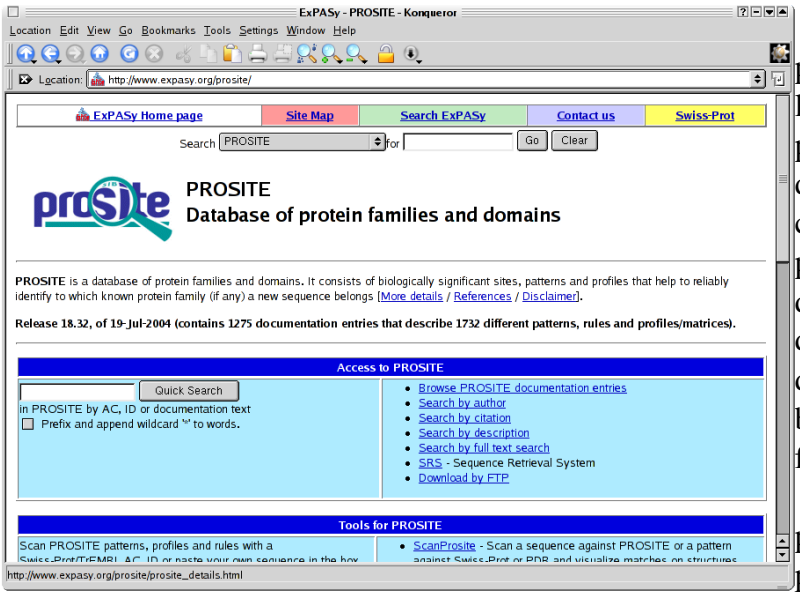


exemples de rendu de protéines selon Pfam. Les grands domaines sont des familles PfamA, en plus petit on trouve les domaines PfamB les domaines trans-membranes et le reste.

Le rendu Pfam n'est pas idéal :

- Les couleurs ne sont pas représentatives et n'ont pas de rôles dans le rendu.
- Le dessin des familles PfamB est dur à lire car trop petit
- le nom du domaine n'apparait que s'il y a suffisamment de place.
- par rapport à ce que veux Xapien, cette représentation n'est pas plus utile ou lisible que des informations textuelles.
- l'échelle du dessin n'est pas adaptable.
- la taille du dessin n'est pas adaptée à la résolution.
- Cette notation n'est pas utilisable dans les pathways.

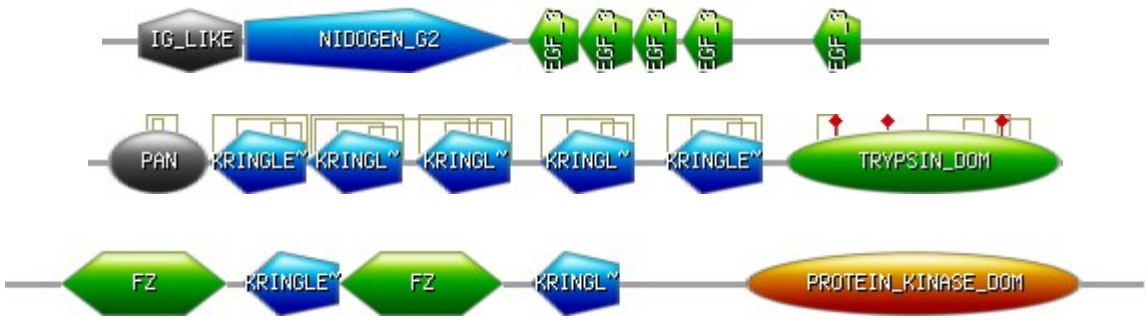
ProSite (http://www.expasy.org/prosite/)



Prosite est une base de donnée de protéines et de domaines qui se base sur les similitudes dans les séquences de protéines pour classer celles-ci. L'étude des similitudes permet de déterminer des comportements génériques entre des protéines appartenant à des espèces différentes. Prosite contient actuellement des informations pour environ un millier de protéines et familles de protéines. cette base est plus petite que Pfam (3500 familles actuellement).

Les recherches sont effectuées par protéines, par auteur, par description. On peut aussi chercher une entrée dans la liste complète des familles.

Comme pour Pfam, le système de représentation graphique de protéine ne sert qu'à illustrer une fiche plus complète.



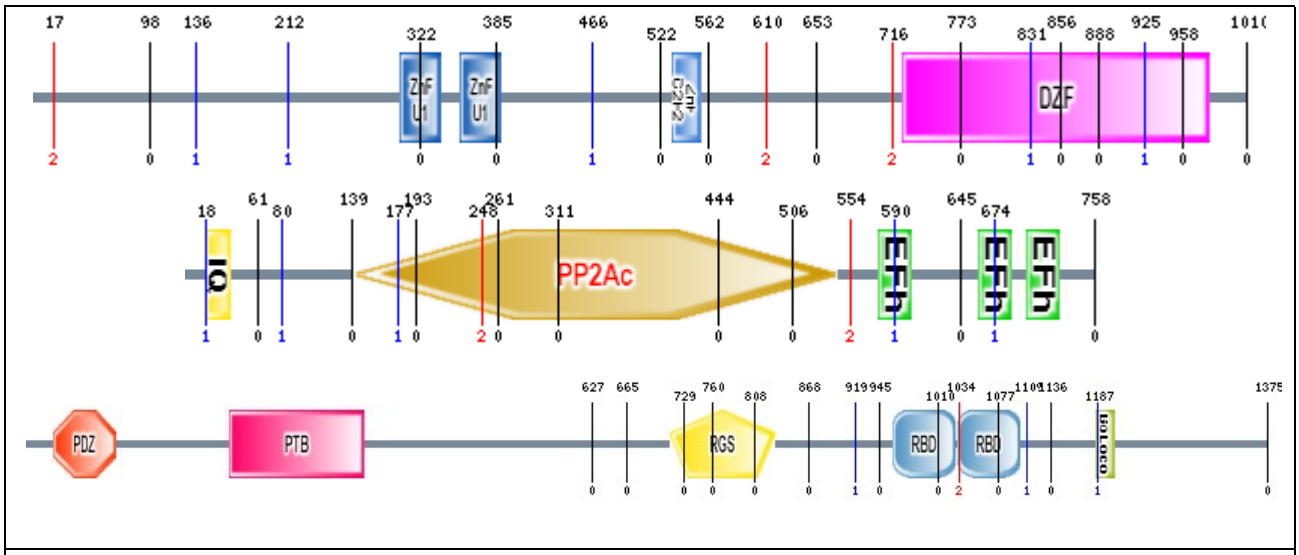
Le système Prosite définit un domaine par sa forme et sa couleur. Il y a des annotations supplémentaires concernant les sites actifs (losange rouge) et les lien intra-domaines (ponts gris).

Caractéristique du rendu Prosite :
- optimisation du placement des textes par des abréviations ou en verticale
- échelle du dessin réglable
- dessin de l'échelle et de la légende sur la page de résultat
- graphiques uniques pour chaque domaine.
- le nom est toujours présent

malgré ces améliorations par rapport a Pfam, il manque encore beaucoup de choses pour le modèle de Xapien :
- les couleurs et formes sont distribuées de façon arbitraire
- pas de distinction entre domaine actif et non actif
- les noms abrégés ne sont pas toujours clairs

Smart (http://smart.embl-heidelberg.de/)

Smart est la bibliothèque de l'institut européen de biologie moléculaire, basé a Heidelberg. les recherches sur les domaines et protéines peuvent être fait par séquence d'amino acides, par nom par mot clef. Smart propose aussi une liste de domaines classés dans 4 grands groupes (signalling, nuclear, extracellular et autres) ; ces groupes correspondent au même tri effectué par Pfam. En poursuivant les recherche dans cette liste, on peut connaître quelles sont les protéines contenant le domaine (toutes espèces confondues), on selectionne ensuite les protéines dont on veut comparer le rendu.



Smart incorpore en plus du dessin des informations sur la structure réelle de la protéine (barre verticales).

Même si le dessin est agréable à lire, ici aussi les informations de couleur et de forme sont distribuées au hasard. les noms abrégés sont clairs pour le concepteur de smart (j'ai dîné avec lui un soir ...) mais très obscurs pour la moyenne des biologistes à Xapien, ce qui est une tare quant on sait que smart est la base de données la plus utilisé. Les mêmes critiques peuvent être formulées pour Smart concernant l'utilité réelle des informations dessinées.

Design du système de représentation.

Le système que veut Xapien est beaucoup plus orienté production. Le rendu doit être non seulement une vue de la protéine mais aussi une information directe sur les propriétés des domaines ; en particulier sur les interactions connues avec des médicaments. La quantité d'information à rendre a travers le dessin impose d'exploiter toute les ressources graphiques disponibles , forme, couleurs, remplissage ...

Il faut aussi trouver une solution à la différence d'échelle existant entre les très grandes protéines et les très courtes.

Connaissant ces contraintes au niveau de la visualisation, j'ai commencé par écrire une bibliothèque de gestion de forme très simple sans me préoccuper pour le moment de donner une forme particulière aux domaines. J'ai juste cherché à résoudre les problèmes d'échelles et de taille de la protéine pour que tout puisse tenir dans l'écran. Le premier prototype est en fait l'écriture de la classe Protéine dont le rôle est de calculer le placement et la taille des domaines qui la composent.

Une fois le premier prototype achevé, c'est à dire le moteur de rendu à proprement parler, nous avons passé avec David (Bio-informaticien) une semaine pour déterminer quelle était la meilleure représentation, quelles couleurs adopter, quelle forme donner aux domaines, où et comment placer les informations textuelles. Je n'ai eu qu'a enrichir mon premier prototype avec la classe Domaine pour définir toutes les informations graphiques relatives aux domaines.

problèmes et solutions

relatif à l'utilisation et à l'optimisation de l'espace

Différences d'échelles entre petites et grandes protéines.	Observation : dans les protéines, beaucoup de domaines ne sont pas actifs et peuvent être négligés dans le rendu, de même, les zones vides entre deux domaines encombrent le dessin inutilement. solution : proposer un modèle de protéine à deux échelles, une grande pour les domaines actifs, une plus courte pour l'espace vide et les domaines non actifs**.**
Taille des grandes protéines	Donner une longueur maximale pour le dessin (1024 dans mon prototype) et adapter la protéine à cette longueur. La protéine sera réduite.
Taille des petites protéines	fixer une longueur maximale de dessin et agrandir (étirer) la protéine jusqu'à ce qu'elle remplisse l'espace de dessin.
Domaines les uns a l'intérieur des autres	Dessiner les domaines par ordre hiérarchique (le plus important au sens biologique) devant en réduisant légèrement la largeur du domaine à chaque couche.

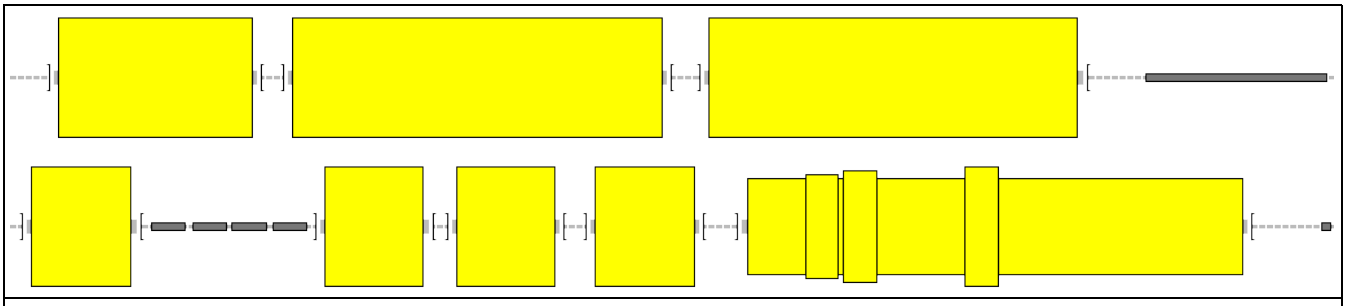
l'échelle variable permet de résoudre tous les problèmes de place du dessin mais on perd la possibilité de comparer la taille de deux protéines entre elle. Avec l'échelle variable, toutes les protéines occupent le même espace, peu importe leur taille.

Pour pouvoir comparer la taille des protéines, nous avons conservé un mode de fonctionnement "réel" dans le moteur de rendu pour pouvoir dessiner une protéine où un acide aminé vaut un pixel. et ainsi comparer les protéines sur la base d'une même échelle.

La taille disponible pour les domaines actifs est un ration de la taille totale, pour ce prototype,

nous avons décidé d'accorder 75% de la taille disponible, il reste donc 25% pour les domaines non actifs ou les parties de la protéine ne contenant pas de domaines.

A ce stade, le premier prototype est fonctionnel.



Les grands rectangles jaunes contiendront les domaines actifs sur 75% de la taille disponible.
Les rectangles plus petit en gris représentent les domaines non actifs
La barre de la protéine est en pointillé gris clair.
Pour bien indiquer que la barre est agrandie par endroit, on signale la rupture d'échelle par un jeu de crochets coupant la barre
Sur la seconde protéine, on peut voir que la largeur de l'espace attribué aux domaines diminue en fonction de l'importance des domaines. Le plus important reste toujours devant. Ce cas est toutefois assez rare, dans une protéine classique les domaines se suivent mais ne se chevauchent pas.

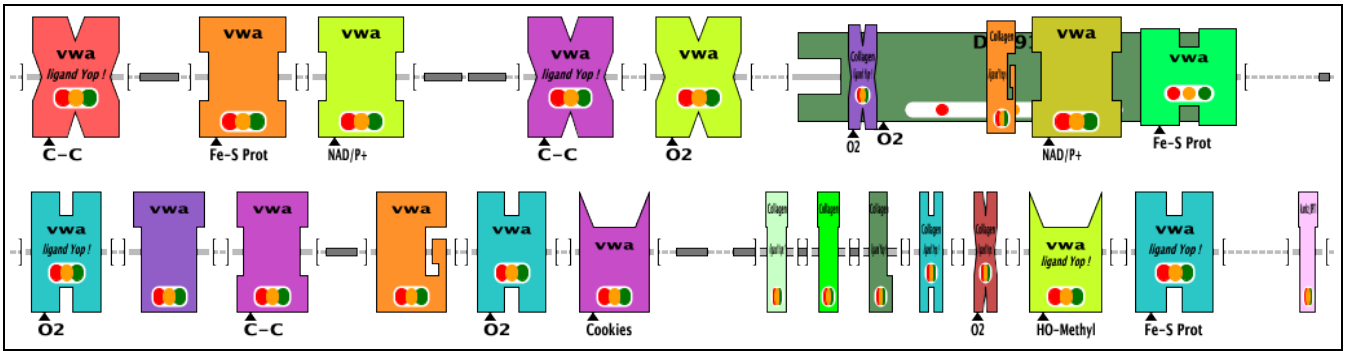
problèmes relatif au design propre des domaines

Rendre identifiable la famille du domaine	Utiliser un système de formes représentant le nom de la famille. Après plusieurs essais nous avons décidé d'utiliser la première lettre du nom de la famille. Il y a en tout 13 familles a distinguer, 5 dans la vue par liants et 8 dans la vue par activité, sachant aussi que ces deux vues sont distinctes et qu'il n'y a jamais de mélanges.
Différencier les deux types de vues (par activité et par liant)	Utiliser une distinction par la couleur. Nous donnerons des tons pastels pour les liant des tons plus colorés pour la vue d'activité.
Choisir les couleurs	Certains choix sont évidents : nous utiliserons par exemple du jaune pour représenter la famille des graisses (Aliphatic), orange rouille pour les Oxido-reducteurs ou encore rose bonbon pour les sucres. Les autres couleurs sont données en fonction de la fréquence d'apparition du domaine. Si le domaine appairait souvent, on lui donne une couleur pas trop agressive ou voyante, pour éviter de surcharger l'image. Les couleurs sont un autre moyen de différenciation des familles de domaines.
Informations concernant l'utilisation médicale du domaine	Dessin à l'intérieur du domaine d'un rectangle contenant trois cases de couleur, vert, orange ou rouge indiquant l'état d'avancement de la recherche médicale sur ce domaine (les couleurs représentant : impossible - à tester - possible). Ces informations doivent provenir des tests de Xapien et ne sont pas encore disponibles dans la base.

Rendre identifiable la famille du domaine	Utiliser un système de formes représentant le nom de la famille. Après plusieurs essais nous avons décidé d'utiliser la première lettre du nom de la famille. Il y a en tout 13 familles a distinguer, 5 dans la vue par liants et 8 dans la vue par activité, sachant aussi que ces deux vues sont distinctes et qu'il n'y a jamais de mélanges.
Textes	Le nom doit être ajouté à l'intérieur de la forme. Pour chaque forme on définit un point d'encrage du texte, là où il y a le plus de place. les textes sont raccourcis au cas où le domaine serait trop court. Pour la vue par liant, on souhaite aussi préciser la nom du liant a l'intérieur de la forme. Pour la vue par activité, on précise le nom de la molécule d'interaction en dessous du domaine, avec une petite flèche pointant sur le domaine.

Ces résultats sont le bilan de la semaine de tests ergonomiques passée avec David pour trouver les meilleures définitions possibles de formes et de couleurs et faire le tri dans les informations a afficher.

J'ai ensuite enrichi le premier prototype pour arriver aux résultats suivants :

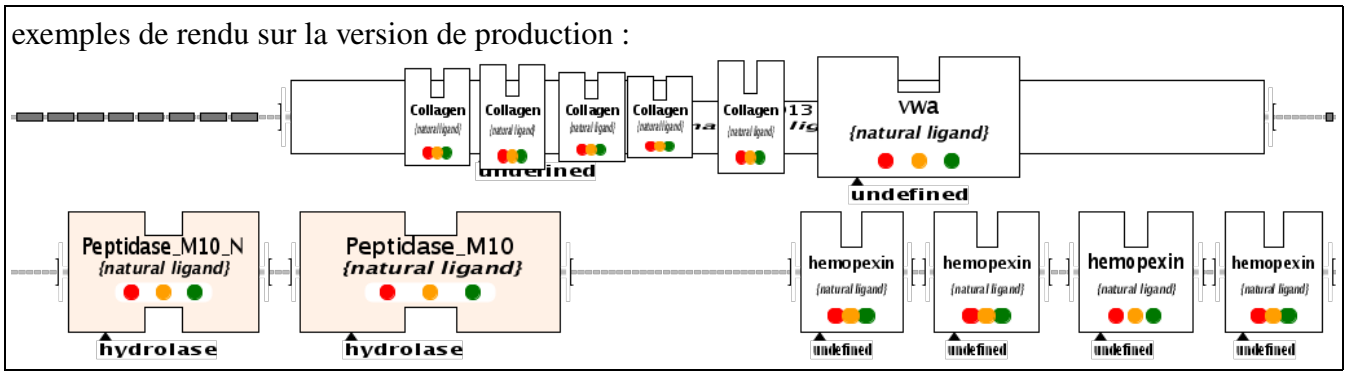


Ces deux protéines donnent un l'aperçu du rendu des formes de domaines, le positionnement des textes et le rectangle d'indication de l'utilisation médicale.

quelques significations de formes : X pour oXido-réducteur, I pour Isomer, G pour liGase, T pour Transferase, H pour Hydrolase ...

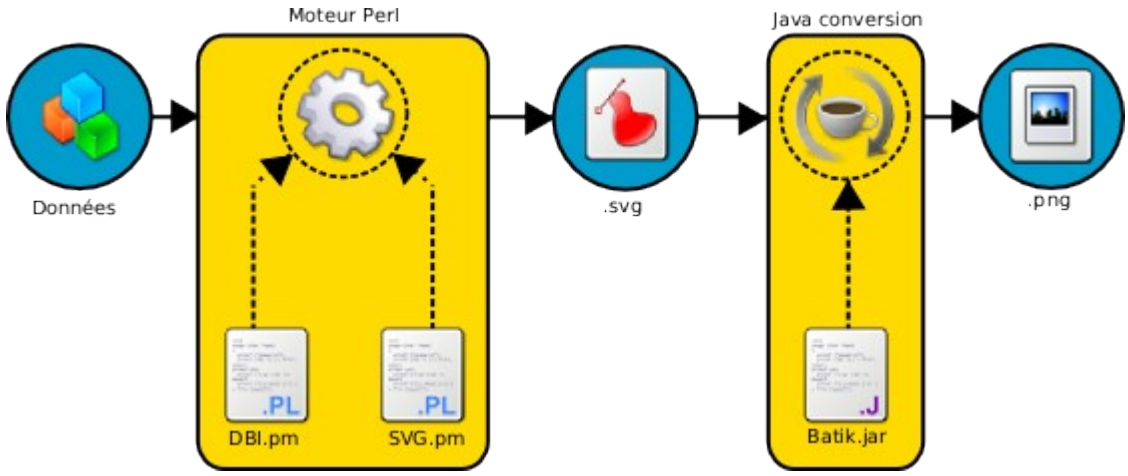
Les couleurs ne sont pas encore fixées dans ce prototype, elles le sont dans la version d'exploitation mais la base de donnée Xapien n'étant pas encore très riche, beaucoup de domaines ne sont pas définis (en particulier leur famille) et les premiers rendus réels sont assez pauvres.

exemples de rendu sur la version de production :



Conception du programme

Le projet est conçu comme une chaîne de traitement partant des données contenues dans la base de donnée Xapien pour arriver à l'image finale au format PNG.



Les données proviennent d'une base Postgresql qui est accédée depuis le moteur Perl à l'aide de la bibliothèque DBI.pm. Le moteur à proprement parler constitue le maillon principal de la chaîne. Il transforme les données en un fichier SVG en utilisant le module SVG.pm et les méthodes objets propres. Ce résultat en SVG n'est pas exploitable directement dans l'application finale, il faut encore transformer ce résultat en une image fixe non vectorielle. Cette étape ne peut être integrée au moteur Perl, il n'existe pas de bibliothèque de conversion pour SVG dans ce langage. C'est donc la tache dévolue a Java qui utilise pour cela les packages offerts par Batik pour convertir du SVG en PNG.

Le ciment entre les maillons de la chaîne est un script shell Bash (interpréteur de commande Unix) qui fait transiter les informations entres les maillons de la chaîne et va stocker les images finales aux bons endroits.

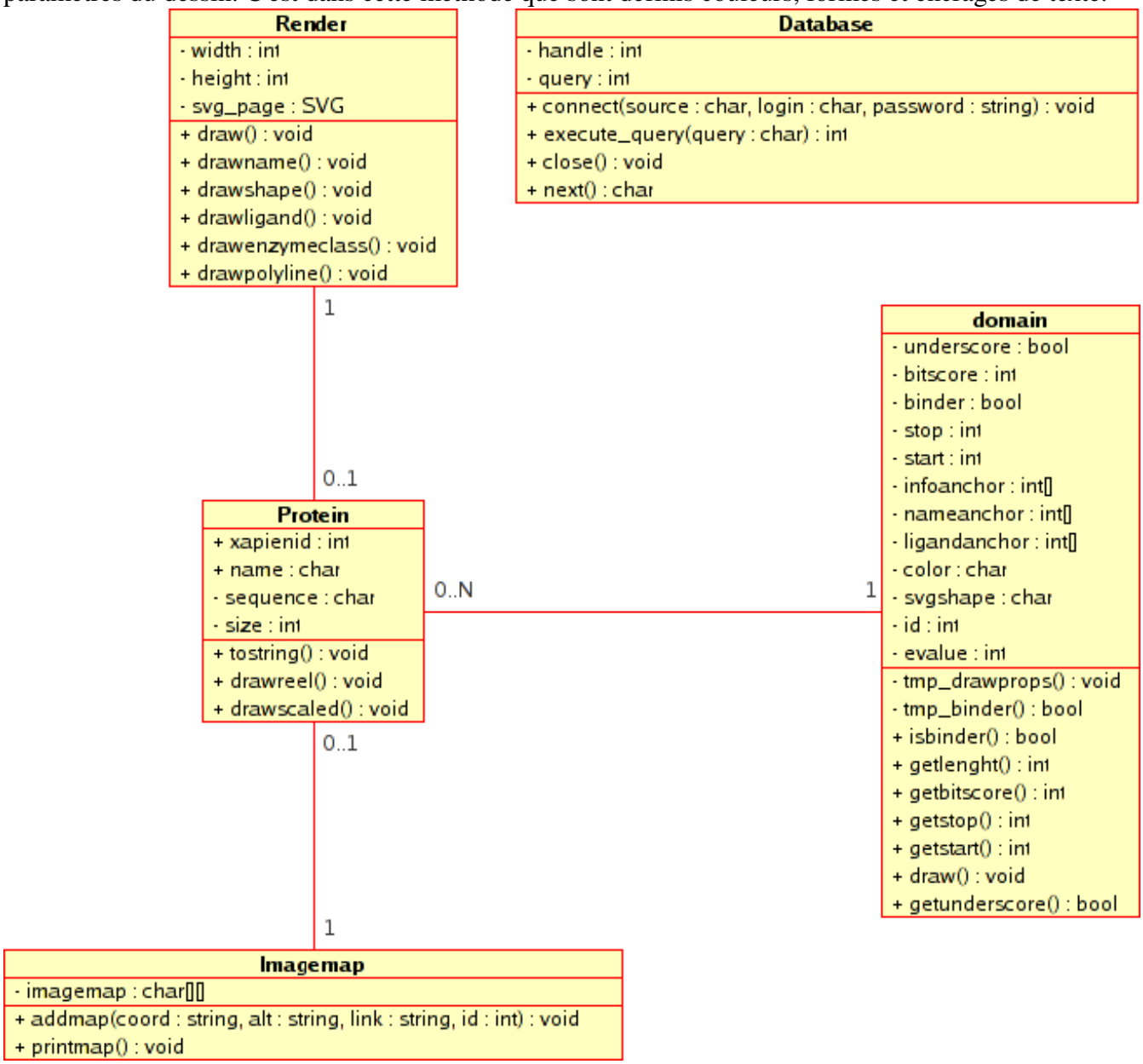
Organisation du Moteur Perl (Schéma de classes)

Le moteur Perl est constitué de deux classes principales, la classe Protein et la classe Domain. qui interagissent entres elles et avec les deux autres classes destinées au rendu soit du SVG soit du plan de l'image pour un l'utilisation combinée de HTML et javascript et créer des images de protéines dynamiques.

La classe Protein contient toutes les information relatives à la protéine, essentiellement sa taille, sa séquence (chaîne de caractère d'amino acides) et les identifiant de la base de données Xapien (xapienid) et son nom.
Une protéine peut contenir entre 0 et N domaines.
Une protéine contient un rendu SVG ou un rendu Imagemap.
les méthodes drawreel() et drawscaled() sont appelées depuis la boucle principale pour générer le rendu SVG

La classe Domain définit toutes les informations relatives aux domaines, particulièrement les caractéristiques de couleur, de forme, les points d'encrage des textes mais aussi la taille, la position dans la protéine, son bitscore (pour connaître son rang en cas de superposition), le booléen binder pour savoir s'il est actif ou non.
Parmi les méthodes importantes, draw permet de dessiner la forme SVG du domaine. Elle est appelée

par l'objet protein contenant le domaine lors du dessin de la protéine.
tmp_drawprops() est une méthode privée appelée par Domain.draw() pour déterminer tous les paramètres du dessin. C'est dans cette méthode que sont définis couleurs, formes et encrages de texte.



La classe Database est un accesseur générique à la base de données Xapien qui utilise Perl DBI et permet de construire une requête, de l'exécuter et de lister les résultats dans une boucle while (typiquement while($res->next()){}). Close ferme la connexion à la base.

Render est la classe contenant la page SVG, l'objet protein lui demande de dessiner des formes ou des textes a des endroits spécifiques.

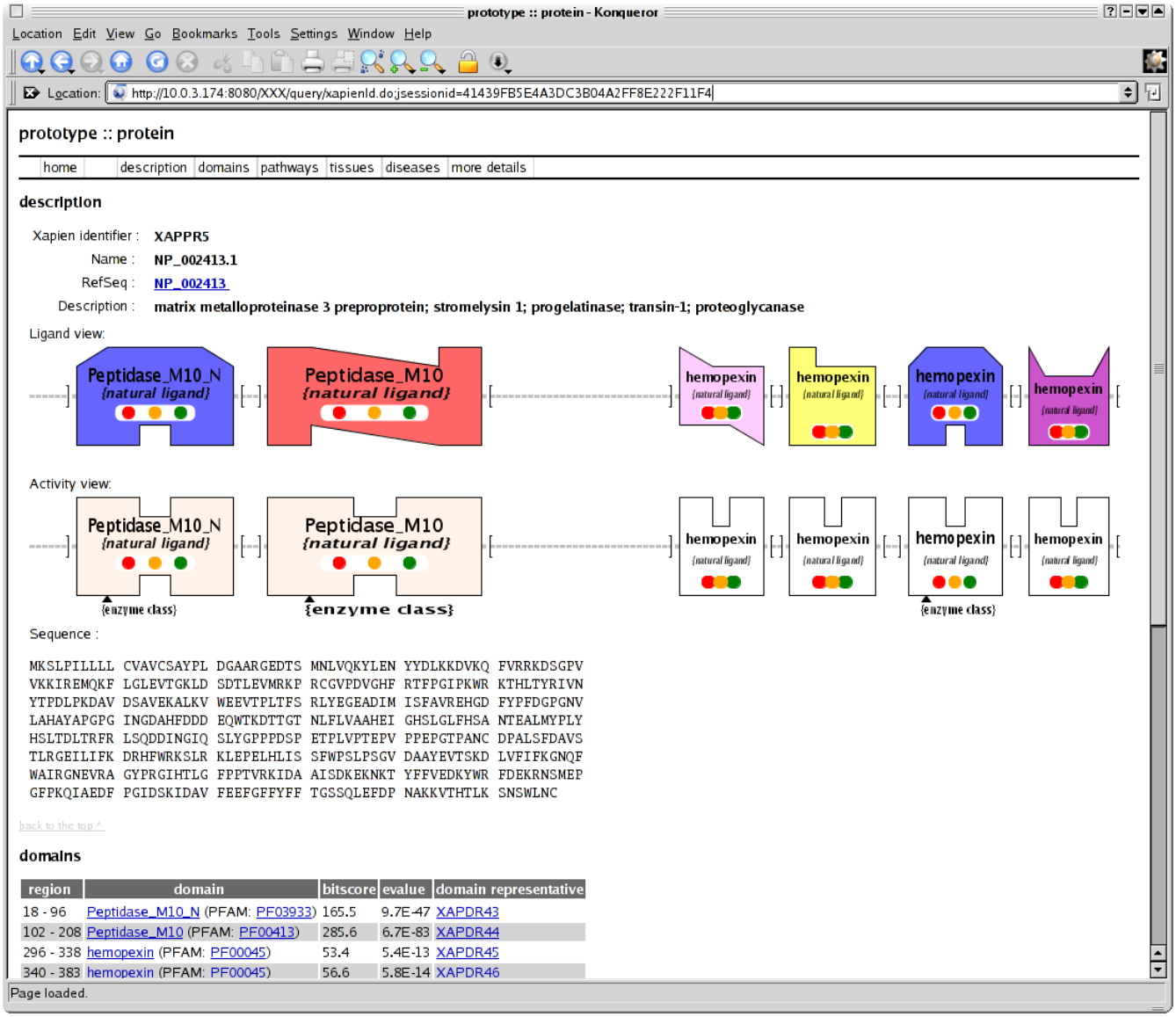
Résultats et améliorations

Même si elle est loin d'être complète pour le moment, la base de données Xapien contient environ 25000 entrées pour des protéines ce qui représente 48000 domaines soit un ensemble de plus de 300000 images à générer. Sans optimisations, le programme (la chaîne dans son ensemble) prend entre 5 jours et 6 jours pour produire l'ensemble.

La majorité du temps de traitement est utilisée pour la connexion à la base de donnée. Perl charge et décharge aussi les modules à chaque image. enfin le toolkit Batik est prévu pour un support complet de SVG et est peut être un peu sur-dimensionné pour l'utilisation que l'on en a.

Plusieurs solutions existent pour diminuer au moins le temps de traitement du moteur Perl :

- factoriser les traitements : ouvrir la connexion a la base une seule fois pour tout le traitement, ne charger les modules qu'une seule fois
- passer directement en perl pour faire le lien entre les maillons de la chaîne de traitement permettrait d'inclure dans le moteur les ouvertures de connections et de modules. Il ne resterait alors que la conversion vers le PNG à effectuer par java à l'extérieur du moteur.
- séparer entrée-sortie et rendu et distribuer ces taches sur des threads différents.
- la version de batik pour java est assez lente. Il existe une version compilée pour C++, on peut supposer que le code compilé générerait plus rapidement.



version du prototype Xapien intégrant mon système de représentation.

Représentation automatique de pathways

Vocabulaire de base sur les graphes

Graphe, sommet (ou noeud), arc (ou arête)

Intuitivement, un graphe est un schéma utilisant des points et des flèches ou des segments non orientés reliant certains de ces points. Les exemples de la vie courante abondent : réseaux routiers, de chemin de fer, d'ordinateurs, le Web, les relais de télévision mais aussi les circuits électroniques, la structuration d'un programme informatique.

L'étude des graphes est un domaine particulier des mathématiques que l'on appelle théorie des graphes. S'il fallait donner une définition mathématique à un graphe on pourrait donner :

Un graphe G est un couple (V,E) où
- V est un ensemble (fini) d'objets. Les éléments de V sont appelés les sommets du graphe.
- E est sous-ensemble de VxV. Les éléments de E sont appelés les arêtes du graphe.

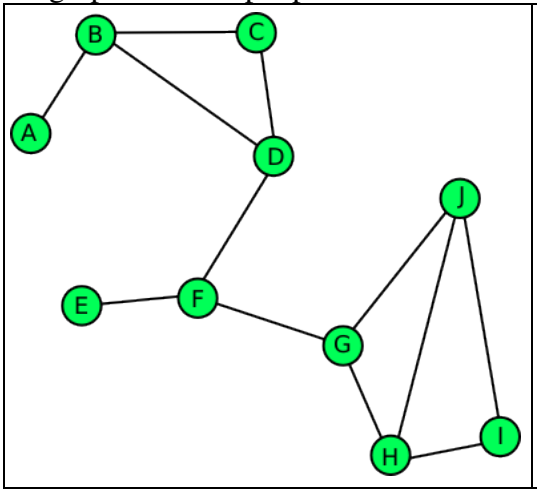
Une arête e du graphe est une paire e=(x,y) de sommets. Les sommets x et y sont les extrémités de l'arête.

C'est dit. Les graphes en mathématiques restent un domaine où l'on arrive très vite à des choses assez difficiles pour les non matheux, faisant appel à la théorie des ensembles qui est déjà bien complexe en elle même. Nous revenons donc à des définitions plus abordables. Si les solutions aux problèmes de graphes sont en général compliquées, les problèmes au moins restent simple à exprimer en langage clair.

Les graphes permettent de manipuler plus facilement des objets et leurs relations avec une représentation graphique naturelle. L'ensemble des techniques et outils mathématiques mis au point en Théorie des Graphes permet de démontrer facilement des propriétés, d'en déduire des méthodes de résolution, des algorithmes, ...

- Comment minimiser la longueur totale des connexions d'un circuit?
- Quel est le plus court chemin (en distance ou en temps) pour se rendre d'une ville à une autre?
- Peut-on mettre une rue en sens unique sans rendre impossible la circulation en ville?

Un graphe très simple pourrait ressembler a cela :



Un graphe est constitué de sommets, les points en vert et d'arcs, les segments qui relient ces sommets.
Ce graphe d'exemple comporte 11 sommets nommés de A a J et 12 arcs.
Si c'est un exemple simple, ce n'est pourtant pas le graphe le plus général possible. Ce graphe illustre déjà une catégorie particulière de graphe, les graphes simples dans lesquels il n'y a qu'un seul chemin direct, c'est à dire sans passer par des sommets intermédiaires pour aller d'un sommet X a un sommet Y. Pour faire l'analogie avec le réseau routier, la catégorie des graphes simples représenterait un réseau où il n'y aurait qu'une route entre deux villes.

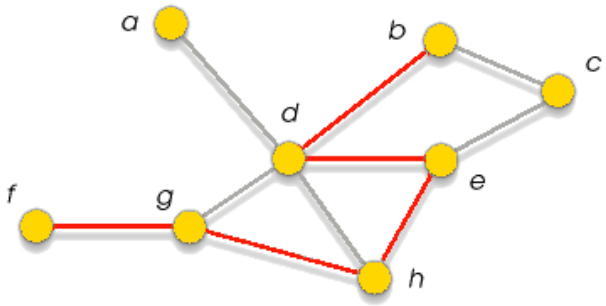
Chemin, cycle, longueur

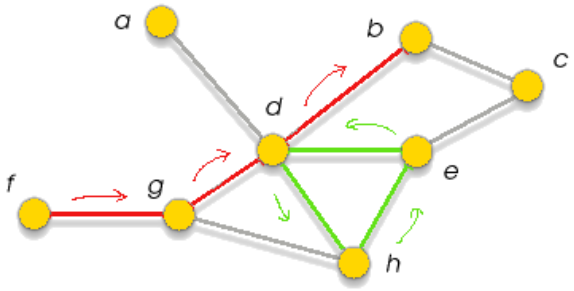
<u>chemin :</u>

Dans un graphe il est naturel de vouloir se déplacer de sommet en sommet en suivant les arêtes. Une telle marche est appelée une chaîne ou un chemin.

f-g-h-e-d-b est un chemin qui permet d'aller de f à b.

Il existe bien d'autres chemins pour aller de f à b : par exemple (f, g, d, b) et même un chemin infini {f,g,[h,e,d]*,b} puisque empruntant un cycle





<u>Cycle :</u>

un cycle est un chemin fermé sur lui même. Il permet de partir d'un sommet et d'arriver à ce même sommet.

{d,h,e} est un cycle.

Il y en a d'autres comme par exemple {h,e,c,b,d}

<u>longueur :</u>

La longueur d'un chemin ou d'un cycle est le nombre d'arc que l'on traverse pour compléter le chemin ou le cycle. la longueur d'un chemin ou d'un cycle vaut le nombre de sommets de contenus dans le parcours -1.

le chemin {f,g,h,e,d,b} a pour longueur 5.

<u>Chemin simple :</u>

Un chemin est simple si chaque arc du chemin est visité une seule fois.

Le chemin {f,g,d,h,e,d,h} n'est pas simple, on passe deux fois par l'arc d-h

Le chemin {f,g,d,h,e} est simple.

On peut ajouter la la définition du cycle que c'est un chemin simple qui finit à son point de départ

<u>Chemin élémentaire :</u>

Un chemin élémentaire est un chemin simple dans lequel on passe une seule fois par les sommets du chemin

Le chemin {f,g,d,h,e,d,b} est simple mais non élémentaire

Le chemin {f,g,d,b} est simple et élémentaire.

Les termes de chemin et de circuit s'emploient en propre pour les graphes orientés. Pour les graphes non orientés que nous manipulons ici, on parle de chaîne et de cycle. Cependant la définition formelle est exactement la même dans les 2 cas, seule change la structure (graphe orienté ou non) sur laquelle ils sont définis.

familles de graphes

Il est impossible de donner une définition unique d'un graphe pour le bonne raison qu'il existe

plusieurs sortes de graphes. Selon les relations, un graphe peut être orienté ou non, valué ou non, cyclique ou non, pondéré ou non ... nous allons détailler quelques familles de graphes pour illustrer cette diversité.

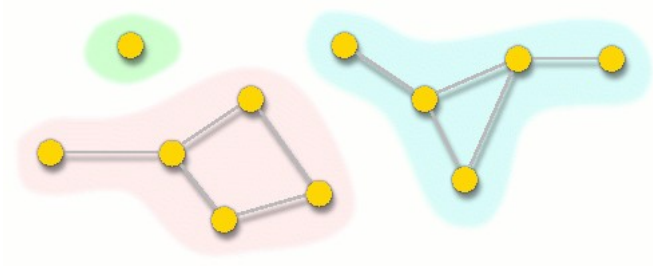
Graphe simple:
c'est un graphe caractérisé par
- pas d'arc dont les extrémités sont confondues
- Un seul arc joignant deux sommets

Graphe simple orienté

On conserve les mêmes règles que pour les graphes simples, on ajoute un sens aux arcs. Par exemple, on peut aller de A vers B mais pas de B vers A. Un graphe de rue en sens unique est un graphe orienté. dans la réalité, la majorité des graphes sont orientés et en informatique, on va représenter un graphe simple non orienté par un graphe orienté en doublant les arcs, comme les deux sens de parcours sont possibles.

On représente les graphes orientés en dessinant des flèches sur les arcs pour signifier le sens de la relation entre les deux sommets.

Graphe connexe



Un graphe est connexe s'il est en un seul tenant, s'il n'y tous les sommets peuvent être reliés a chacun des autres sommets en empruntant un ou plusieurs arcs. Autrement dit, il faut que le graphe soit en un seul bout. Les graphes donnés en exemple depuis le début sont tous connexes. Le graphe à gauche n'est pas connexe.
Il est en effet composé de trois sous graphes, on dit que son niveau de connexité vaut 3.

Arbre

C'est une catégorie de graphes particuliers. Ils n'ont pas de cycles et sont connexes. Un ensemble d'arbres est un graphe particulier appelé forêt (bravo!)

Sous graphe

Toute division d'un graphe en un ensemble plus petit est un sous graphe du graphe principal

Graphe valué

C'est un graphe pour lequel les arcs ont une valeur. Cette valeur peut exprimer une contrainte sur le graphe. C'est par exemple le nombre de kilomètres entre deux villes pour un réseau routier (contrainte de proximité). Les contraintes sont souvent très importantes quand il s'agit de représenter un graphe, comme nous le verrons plus loin.

Algorithmes sur les graphes

Des lors qu'il existe des arcs entre des sommets, l'esprit humain titillé se pose beaucoup de questions ... Quel peut être le chemin le plus court d'un sommet à un autre du graphe ? existe t'il un chemin élémentaire pour passer par tous les sommets ? Quel est le plus grand circuit possible dans le graphe ? On peut même imaginer des jeux passionnants sur les graphes comme par exemple "est-il

possible de parcourir une fois et une seule chacune des cases d'un échiquier avec un cavalier ?"

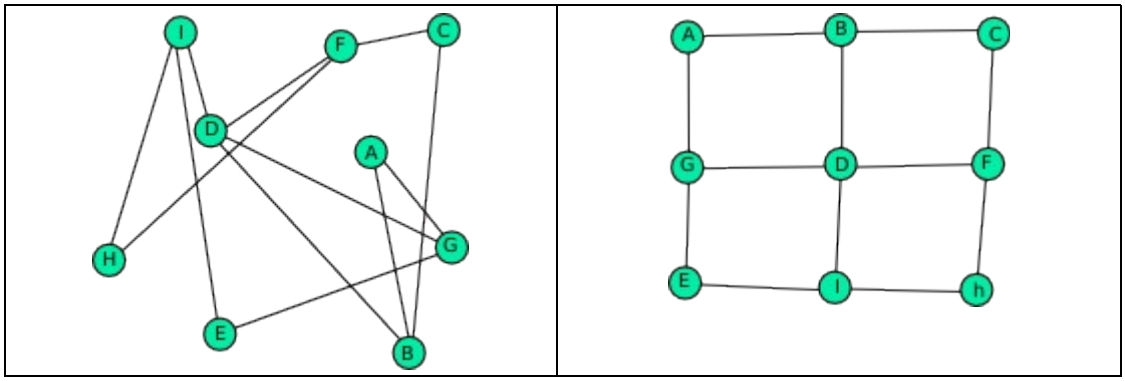
Tout ce qui vise à répondre à ces questions est du domaine de l'algorithmique des graphes. Si certains algorithmes sont déjà bien installés (le problème du plus court chemin à été très documenté) d'autres comme par exemple la détermination du plus long chemin élémentaire d'un graphe n'ont pas de solutions simples, on dit que ce sont des problèmes NP-complets auxquels il n'y a pas d'autres solutions que de parcourir l'ensemble des solutions possibles pour trouver celle qui répond au problème.

Rendu automatique de graphes

introduction à la complexité du rendu

Au moment où j'ai pris connaissance de mon sujet de stage, quant on m'a parlé de dessiner automatiquement des pathways et que j'ai vu des descriptions du résultat auquel s'attendait Xapien, j'ai pensé que c'était un bon mois de travail pour trouver et proposer une solution et que sans doute mon sujet méritait d'être étoffé un peu. Puis au fur et à mesure de mes recherches sur le domaine, sur internet, dans des bouquins, à l'université de Metz où j'ai discuté avec pas mal de professeurs, j'ai compris que la solution était loin d'être évidente et que la représentation graphique automatique de graphe est un problème encore très expérimental.

Pour donner une idée du défi que se propose de relever le rendu de graphe, voici deux graphes équivalents au sens mathématique. L'un a un rendu optimisé et l'autre, c'est un positionnement au hasard des noeuds.



Celui de droite est beaucoup plus simple à lire, on comprend tout de suite sa structure. L'organisation d'un graphe pour son rendu est une étape obligatoire des qu'il s'agit d'afficher le résultat d'un graphe pour qu'il puisse être rapidement interprété par le cerveaux humain. Pour cette partie de mon stage, je baigne dans les IHM .

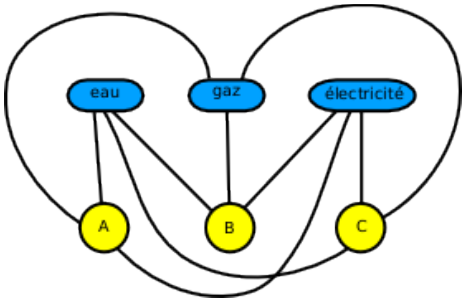
Le dessin de graphe est une tâche très complexe. Pour s'en rendre compte, il suffit d'essayer d' écrire une méthode utilisant des ordres simples pour organiser le graphe de gauche afin d'arriver à quelque chose d'approchant l'image de droite et espérer que cette méthode s'applique à tout types de graphes . Si nous arriverions assez facilement dans des graphes simples à dessiner à la main un rendu correct (le cerveau fait des regroupements instinctifs et arrive à organiser de façon géométrique très efficacement), la tâche est beaucoup plus dure pour un ordinateur qui n'a aucune idée de ce que signifie un rendu lisible.

En fait, il n'existe aucun algorithme suffisamment complet pour rendre tout type de graphes à coup sûr, il n'existe pas d'algorithme où l'on soit certain de la qualité du résultat et il n'existe pas de méthode pour être sr de l'efficacité d'un algorithme par rapport à un autre.

Pour concevoir un programme de rendu de graphe, il faut commencer par cerner au maximum le problème et bien comprendre quel type de graphe on a à représenter. Plus on arrive à caractériser clairement la catégorie des graphes à rendre plus on a de chance de trouver un algorithme efficace. Il est très rare d'avoir à concevoir un algorithme qui doive répondre à toute les situations de graphes et de

toutes façons un tel algorithme n'existe pas.

Voici, pour terminer cette introduction, un problème typique pour illustrer l'une des contrainte classiques du dessin de graphe : la minimisation des croisements d'arcs.



dans une ville, trois nouvelles maisons viennent d'être construites. On veux raccorder ces trois habitations aux services d'eau, de gaz et d'électricité. Y a t-il un moyen de dessiner le plan de raccordement sans qu'aucune des lignes de raccordement ne chevauche une autre ? Si la réponse est non, comment déterminer le dessin où il y a le moins de croisements possibles ?
Le dessin de gauche propose une solution, il existe en mathématiques un moyen de prouver qu'il n'y a pas de solutions sans croisement à ce problème. Pour répondre à la seconde question, celle de la minimisation des intersections, il faut étudier les algorithmes de rendu de graphes.. Les mathématiques peuvent encore prévoir quel est le nombre minimal de croisement, ensuite tout est une question d'efficacité de l'algorithme.
Certaines méthodes vont privilégier le temps au détriment de la qualité du rendu, d'autres algorithmes vont passer beaucoup plus de temps en calcul pour effectuer un rendu meilleur

méthodes pour le dessin de graphes

Il existe de nombreuses méthodes de dessin, privilégiant telles ou telles contraintes de rendu, certaines s'appliquent aux grands graphes, d'autres sont spécifiques aux arbres, d'autres encore ont des approches volumétriques.

Il est aussi évident qu'une méthode automatique générale pour le dessin de graphe n'a d'intérêt que si l'on a à représenter des graphes différents. C'est une perte de temps de concevoir un algorithme de rendu pour un graphe en particulier et qui ne s'adapte à aucun autre cas.
Avant de choisir une méthode de dessin, il faut étudier ce que l'on a à dessiner.

<u>déblayer le terrain</u>

Étape préalable à toute tentative, il faut caractériser au maximum les graphes que l'on a à rendre. On peut caractériser les graphes en se posant des questions simples sur leurs structures :
- Combien de sommets comporte en moyenne le graphe ?
- Quel est la proportion d'arcs par rapport aux sommets ?
- Y a t-il beaucoup de cycles ?
- En dehors des cycles, y a t-il des structures répétitives dans les graphes ?
- Quelle est le nombre moyen de noeuds voisins ?
- Structure proche d'un arbre ou plutôt graphe fortement cyclique ?
- Peut-on classer les structures des graphes en sous groupes généraux ?

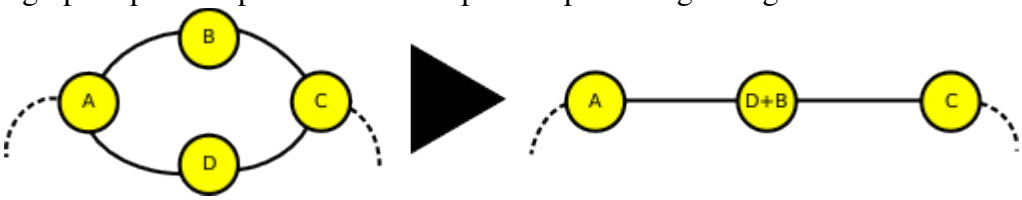
Avec des réponses à ces questions, on peut peut être déjà déterminer quel algorithme utiliser ; par exemple si l'on a que des arbres à dessiner, il y a beaucoup de chances qu'il existe des algorithmes adaptés spécifiquement au dessin d'arbre. En tout cas, on peut déjà savoir quels algorithmes sont inadaptés et ne fonctionneront pas.

Ensuite on peut s'intéresser à l'aspect que l'on veut donner au rendu en énonçant des contraintes. On cherche à répondre à des questions comme :
- La taille d'ensemble du graphe à t-elle une importance ?
- Y a t-il une orientation à donner (plutôt horizontale, plutôt verticale) ?

- Quelle importance accorder à la proximité du voisinage d'un noeud ?
- Si les arcs sont valués, cette valuation doit elle influer sur le rendu ?
- Comment représenter les arcs ? Segments, splines, lignes brisées ?
- Y a t-il des sommets fixes ?
- Faut-il à tout prix éviter les croisements ou cela n'a t-il pas trop d'importance ?
- Faut-il essayer de représenter des structures géométriques dans le graphe ?

Selon les priorités que l'on accorde aux contraintes, on va s'orienter vers un algorithme plutôt que vers un autre.

Enfin, il faut essayer de simplifier le graphe avant d'exécuter l'algorithme. Les structures répétitives du graphes peuvent peut-être être simplifiées pour alléger l'algorithme.



La simplification ci dessus élimine les petits cycles en les remplaçant par un chemin simple. La plupart des algorithmes seront plus efficaces après ces optimisations.

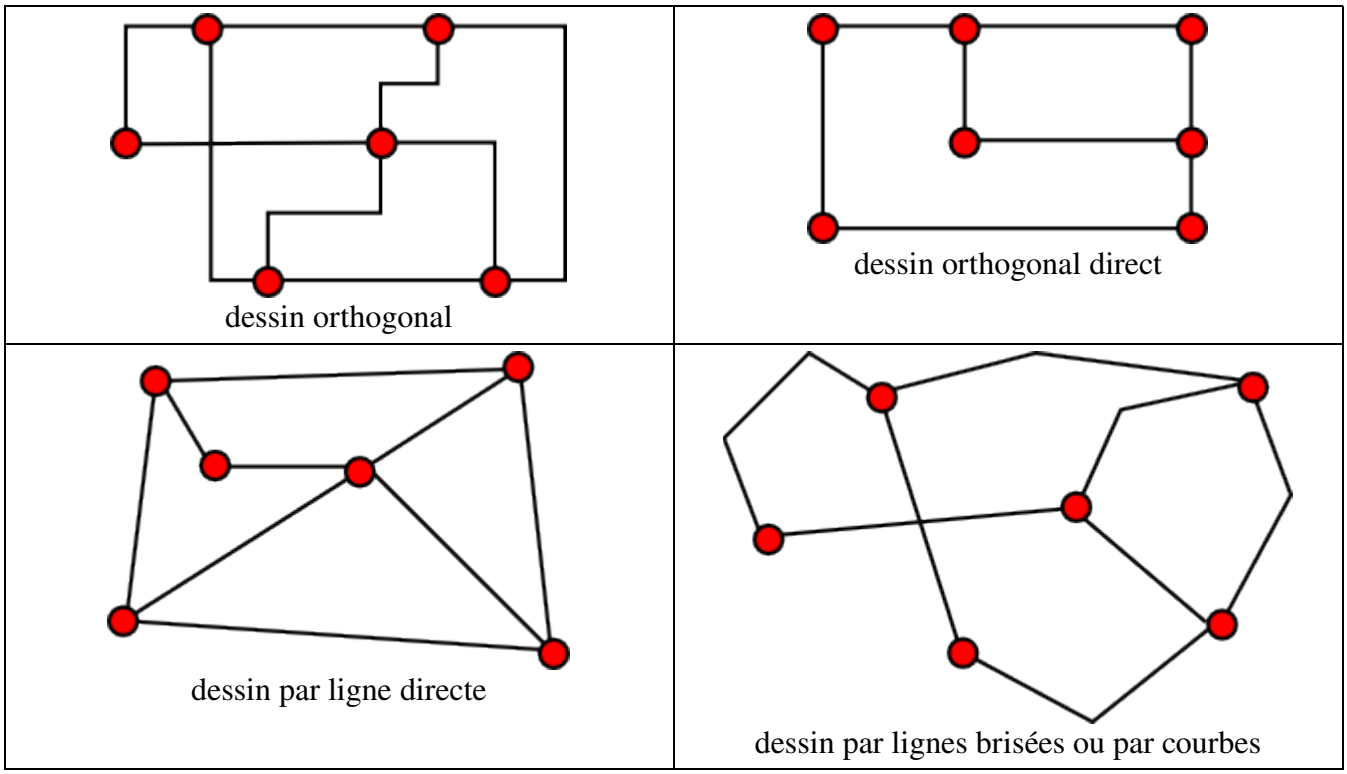
De même, il est souvent souhaitable de rendre un graphe connexe avant le rendu. On introduit alors des arcs pour lier les composantes connexes entres elles et ne plus avoir qu'un graphe d'un seul tenant.

Si l'on veux faire ressortir des figures géométriques dans un graphe, il arrive que l'on ait à compléter un graphe pour y inscrire une structure détectable par l'algorithme de rendu (s'il manque par exemple un sommet a un carré). On introduit alors de nouveaux sommets dont le but est d'équilibrer la figure. Ces sommets ne jouent pas de rôles dans le graphes, on les appelle les "dummy nodes".

Après le traitement par l'algorithme de rendu, il faut bien entendu inverser les simplifications et re-dessiner les noeuds effaces, effacer les liens rajoutés, effacer les dummy nodes et leurs arcs.

méthodes de dessin

On distingues quatre grandes familles de dessins de graphes, chacune d'elle résulte en des algorithmes spécifiques.



Les algorithmes les plus simples à programmer sont ceux qui représentent un graphe à l'aide de segments directs.

Algorithmes existants pour le rendu des graphes.

Je donne ici quelques classes d'algorithmes de rendu, sans entrer dans les détails de l'implémentation en précisant leurs domaines d'utilisations et leurs limites.

Algorithmes reprenant un modèle physique :

La plupart des algorithmes de dessin de graphes sont destinés à des classes de graphes possédant des propriétés spécifiques (graphes planaires, graphes orientés sans cycle, etc). Parmi les algorithmes qui sont en mesure de calculer un dessin d'un graphe sans propriété particulière, les algorithmes reprenant des modèles physiques présentent certaines qualités: les dessins sont "plaisants", la longueur des arêtes est quasiment uniforme dans tout le graphe, ils font ressortir les composantes (quasi)-isomorphes, etc. Le modèle correspond grossièrement à considérer les sommets du graphe comme des masses et les arcs comme des forces (ou des ressorts) attirant les sommets voisins.

Leur implémentation est à la fois facile et délicate. Le modèle relativement clair et explicite dépend de certains paramètres que seule une expérimentation soigneuse permet de régler. Cette situation est typique des problèmes d'optimisation, très sensible au choix des conditions initiales pour le calcul d'une solution.

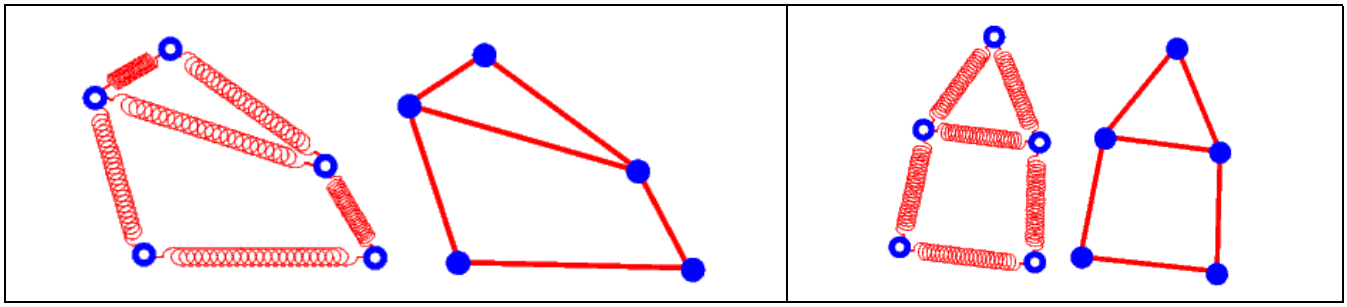


illustration du modèle physique par un système de ressorts

l'utilisation unique d'un algorithme de ce type apporte toutefois certains problèmes

- Incapacité à « sortir » d'une mauvaise configuration (problème typique des algorithmes d'optimisation).
- les forces peuvent maintenir une situation de blocages (croisement d'arêtes) esthétiquement insatisfaisantes.
- Difficulté du paramétrage (nombreuses constantes dont les effets sont corrélées).

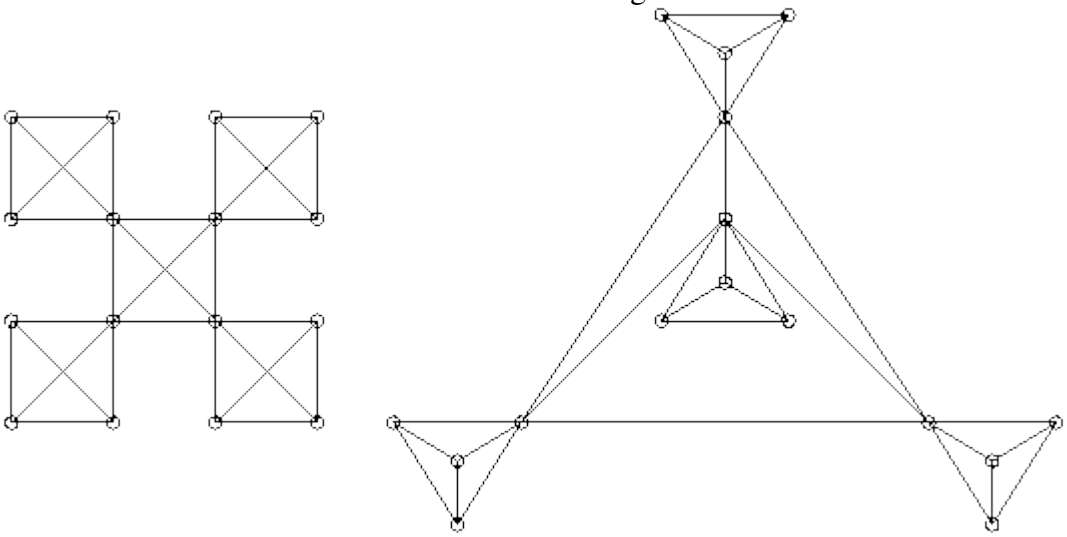
Il semble qu'en général, cet algorithme soit particulièrement efficace à la condition qu'un pré-placement est été effectué auparavant et qu'on se serve uniquement d'un algo à modèle physique pour fignoler le rendu. On évite ainsi beaucoup de problèmes de bloquages dûs aux croisements d'arcs.

Algorithme de rendu par communautés.

Le principe est de diviser la tache de rendu en des sous groupes que l'on va ensuite associer pour compléter le rendu final.

Il faut tout d'abord détecter les différentes communautés qui sont des sous-graphes, sachant qu'il peut y avoir plusieurs types de communautés. On calcule le rendu pour chaque type de communauté avec un algorithme propre a chacune d'elles. Enfin l'association des éléments peut se faire par un algorithme à modèle physique qui traite chacune des communautés comme un sommet du graphe d'ensemble.

Les communautés peuvent être des sous ensembles géométriques comme dans les exemples ci dessous. La caractéristique de ce rendu est qu'il a pour principal intérêt de faciliter la lecture, au détriment des contraintes comme la conservation du voisinage ou les intersections minimales.



Algorithme de rendu de graphe planaires

on appelle un graphe planaire un graphe qui peut être dessiné sans aucun croisement. Il existe des théorèmes de la théorie des graphes pour savoir si un graphe est planaire.

Si le graphe est planaire, il est sans doute possible de lui trouver une représentation sans croisement. L'algorithme considère qu'il existe dans le graphe un cycle pouvant contenir le graphe, au sens géométriques (sinon ce serait un arbre). A l'intérieur de ce cycle, les autres sommets et arcs décrivent des pièces, comme un puzzle. Certaines de ces pièces en chevauchent d'autres, il faut alors dessiner l'une des deux à l'extérieur du cycle. une fois que tout les chevauchement ont été résolus, on doit avoir rendu le graphe planaire sans intersection.

Cet algorithme est très systématique, il s'applique a coup sur et reste très efficace sur les petits et moyens graphes. Malheureusement, prouver la planarité d'un graphe est complexe, si le graphe n'est pas planaire, il existe des méthodes pour le rendre (en ajoutant des sommets sur chaque intersections d'arc par exemple) mais ici aussi c'est compliqué et coûteux en temps de calcul.

algorithmes pour le rendu d'arbres :

C'est l'un des problème de rendu de graphe qui est à priori plus simple à résoudre. les algorithmes sont assez intuitifs dans le cas d'arbres de petite et moyenne taille et il existe des algorithmes très bien documentés pour les arbres de grande taille.

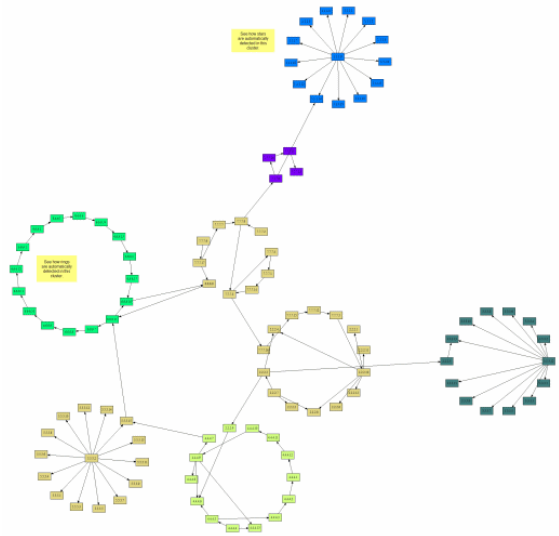
Habituellement, on représente un arbre sous la forme de plusieurs listes indentées, en empilant les générations les une en dessous des autres. Cela pose un problème dans le cas des grands arbres ou l'on arrive à un point ou il n'y a plus de place pour la génération suivante et l'arbre n'est jamais entièrement visible car le nombre de sommets affichés simultanément est forcement limité par l'espace disponible.

Des techniques spécifiques ont été développées pour la visualisation de grands graphes et font plus appel à des images dynamiques. On peut citer entre autre l'approche par diagramme, l'approche surfacique (cartes d'arbres circulaires), l'approche 3D ou encore l'utilisation des géométries non euclidiennes pour la représentation (voir TER de maîtrise).

outils existants

Il existe beaucoup de librairies généralistes pour le rendu de graphes, certaines sont très complète, d'autres proposent des solutions adaptés plus particulièrement à certains types de graphes. La plupart sont des outils propriétaires payant, les licences sont chères. Un produit gratuit néanmoins : graphviz, c'est aussi l'un des plus efficaces.
Petit tour d'horizon de ce que proposent les éditeurs.

Tom Sawyer Software :



Sans doute l'un des plus grand spécialiste du rendu de graphes. Il propose un toolkit pour java ou c++ permettant de gérer tout les aspects de la visualisation de graphes, depuis le stockage jusqu'à la modification du rendu par l'utilisateur. ils proposent plusieurs algorithmes pour le dessin, il est possible d'en mélanger plusieurs sur un même rendu.
Les sommets peuvent aussi être rassemblés en groupes qui pourront être maximisés ou minimisés. Un sommet peut être Dessiné comme un polygone de taille quelconque.
Le programme donne des résultats impressionnants sur le papier mais reste mal adapté aux cas des pathways biologiques ou les contraintes sont particulières.

Graphviz

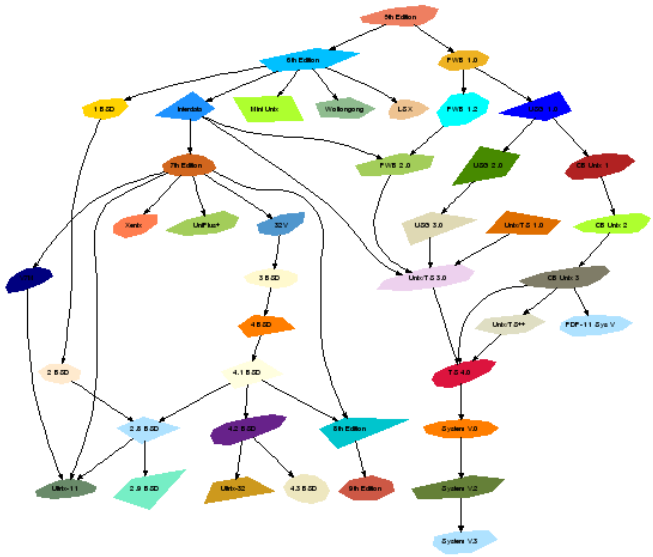
Graphviz est un projet mené par le laboratoire de recherche AT&T depuis plus de 15 ans. Ils ont été les précurseurs du rendu de graphe.

les deux principaux programmes pour le rendu sont dot (pour les graphes orientés) et neato (pour les graphes non orientés). Le but principal de ces deux algorithmes est de donner un résultat esthétique et compact et aussi de conserver au maximum ce comportement pour les grands graphes. Les dessins doivent ressembler à ce qui aurait pu être fait manuellement.

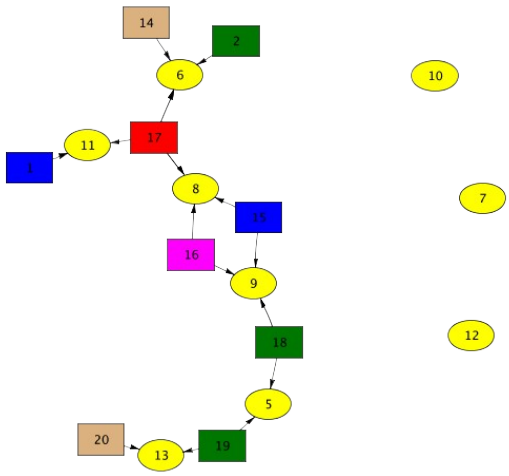
Dot dessine un rendu hiérarchique des graphes orientés. Il rends des résultats corrects et il est

possible de dessiner les sommets sous forme de polygones. Dot intègre aussi un algorithme de regroupement avec lequel il est possible de dessiner des clusters de sommets.

Si l'on doit dessiner un graphe non orienté, Netao est l'algorithme qui correspond le mieux. C'est un algorithme à modèle physique qui simule un système à ressorts et aimants. Neato est particulièrement efficace et rapide. Malheureusement, il n'est pas possible en Neato de dessiner des groupes de sommets et le dessin peut être différent entre deux rendus, ce qui limite grandement son utilisation pour les pathways.



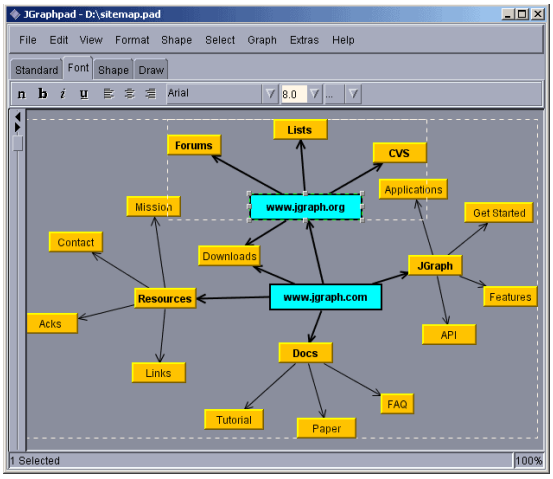
Rendu hiérarchique par Dot



Rendu d'un graphe non orienté par Neato (pathway inositol).

Le langage pour exprimer les graphes graphviz est le Dot. Il existe des modules en Perl pour convertir directement des structures de graphes en dot et pour convertir le rendu en image statique.

JGraph



JGraph est le composant de représentation de graphe le plus utilisé en java. Il consiste en une série de classes qui permettent de dessiner un graphe et d'interagir avec l'utilisateur. Il apporte des fonctionnalités de drag&drop, de déplacement, rotation l'homothétie des sommets par groupes ou à l'unité. En fait JGraph se charge de toutes les interactions utilisateur sur les graphes.

JGraph n'intègre pas d'algorithme de rendu (certains sont toutefois disponibles sous la forme de plugins). Il faut calculer le rendu du graphe par un autre programme ou en utilisant son propre algorithme et ensuite enregistrer ce graphe avec les positions des sommets dans le format de description de JGraph et le dessin peut être confié à ce dernier.

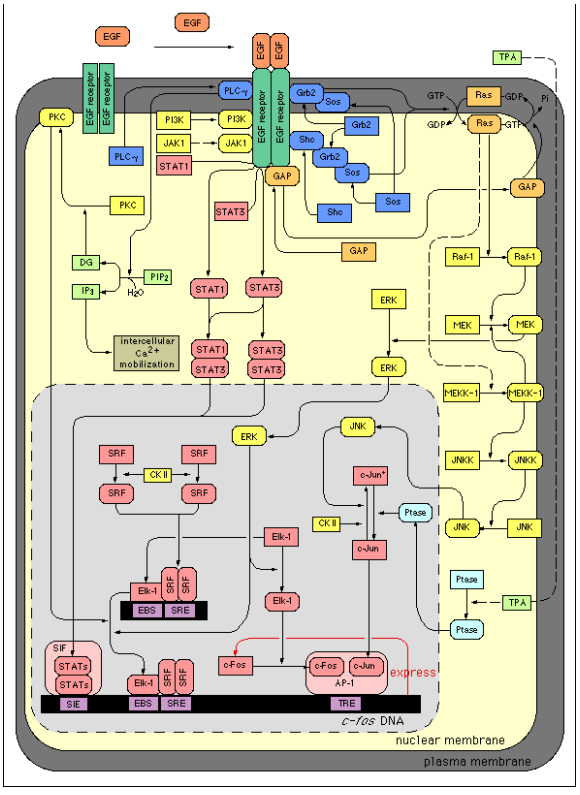
L'utilisation de JGraph est intéressante si l'on veut par exemple faire contrôler par un utilisateur le rendu automatique. Comme le tout-automatique n'est pas parfait, en passant par JGraph, un utilisateur va pouvoir replacer des sommets ou des groupes qui auraient été mal disposés par l'algorithme automatique.

Malgré toute la diversité des outils existants, il n'y a pas de librairie spéciale dédiée au rendu automatique de pathways biologiques.

Études des pathways Biologiques

Comme pour les protéines, il existe des bases de données pour les pathways. certaines sont publiques mais la plupart sont payantes et toutes ces bibliothèques proposent un nombre limité de pathways car ils sont dessinés a la main. Le choix des éléments biologiques représentes par un pathway dépend beaucoup de la base de données. certaines choisissent de représenter une vue très détaillée, d'autres sont beaucoup plus simples. L'étendue d'un pathway est aussi quelque chose de totalement arbitraire. Il est théoriquement possible de décrire le fonctionnement de l'être humain dans son ensemble dans un gigantesque pathway présentant les interactions moléculaires intra et extra cellulaires pour chacune des cellules du corps humain (cela reste de la science fiction). Choisir l'étendue d'un pathway n'est pas simple, il n'y a pas de règles pour prévoir à l'avance quel est le niveau de détail nécessaire à la compréhension d'un mécanisme dans son ensemble. Plus la vue est large et plus le système est difficile à appréhender et si le pathway n'est pas assez étendu, on risque de manquer des interactions importantes.
Seuls les pathways métaboliques bénéficient d'un consensus pour leur représentation. Comme ils illustrent des interactions chimiques, il n'y a pas d'ambiguïté sur le sens de l'une ou l'autre des réactions. Ces pathways ont été les premiers à êtres étudiés et sont maintenant bien connus. Malheureusement ce ne sont pas les plus utiles pour les biologistes.

quelques bases de données de pathways :



- Kegg : c'est la base la plus importante de pathways métaboliques avec plus de 100 à 120 entrées. Kegg propose aussi depuis sa version 2 des pathways de maladies (desease pathway). Les pathways kegg sont exprimés en un langage XML, le KGML où sont enregistrés les réactions, les relations, les composés chimiques. Avec ce langage, il est possible de récupérer la structure des pathways Kegg et de les représenter en enrichissant le modèle. C'est d'ailleurs ce que font beaucoup de base de données de pathways.
- AFCS : Alliance for cellular signalling ; représente des pathways de signalisation sur un modèle original. Leur rendu est agréable et simple à lire. ce sont ces pathways qui ont servi de base de réflexion à l'équipe de Xapien pour imaginer les premiers prototypes de leurs pathways.
- Gold.db : une base de données qui se concentre uniquement sur les interactions avec des lipides. Ce sont des transformations et enrichissements des pathways Kegg de base.
- SPAD : Pathways de signalisation ici aussi. Cette base n'est plus à jour mais disposait d'un rendu intéressant, en basant la représentation du graphe sur la position physique des composants dans la cellule (image ci-dessus). La résolution automatique du rendu peut être largement facilité si un certain nombre des noeuds ont déjà une position fixe.

Toutes les bibliothèques existantes proposent des pathways qui sont dessinés à la main ou à l'aide de logiciel de dessin de graphes mais aucune ne propose des pathways rendus automatiquement. Les recherches dans ce domaine ne sont pas très poussées, on trouve quelques tentatives d'automatisation de dessin de pathways métaboliques et l'un ou l'autre papiers qui décrivent théoriquement des méthodes

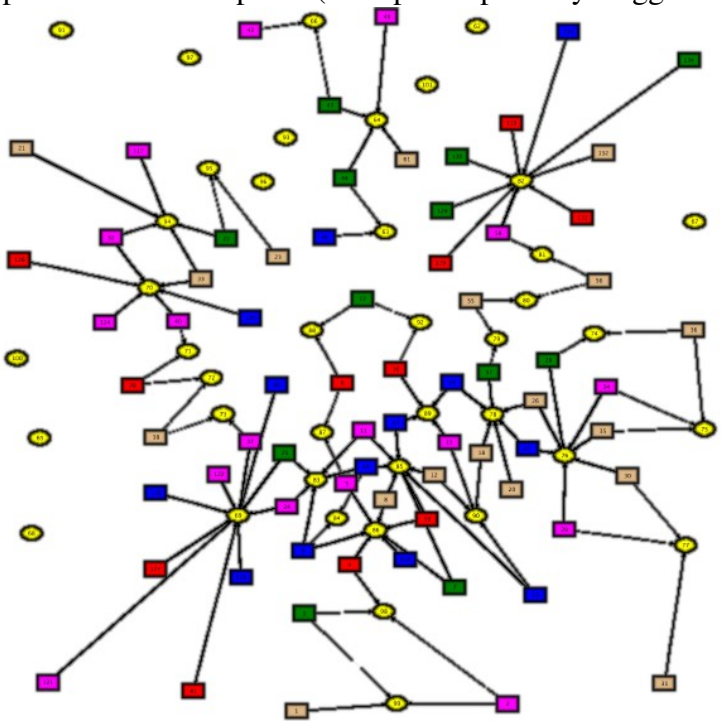
pour arriver à dessiner des pathways. Je me suis inspiré de l'un de ces papiers pour concevoir un algorithme de rendu propre à Xapien.

Caractéristisation des pathways.

Pour mieux comprendre l'organisation des pathways, j'ai étudié les exemples qui m'ont été fournis par Xapien mais mes principales recherches ont porté sur les pathways métaboliques kegg. et en particulier cinq d'entres eux que j'ai définis comme mon jeu de test pour tout le reste de ce projet.

test de rendu avec les outils automatiques

Avant de me lancer dans l'écriture d'un algorithme spécifique, j'ai d'abord testé ce qu'il était possible de réaliser avec les avec Dot et graphviz, le seul toolkit de rendu utilisable gratuitement :
Après l'écriture d'un convertisseur basique entre KGML et Dot, j'ai pu étudier le rendu automatique des pathway kegg par le logiciel qui est connu pour être l'un des meilleurs. Voici le résultat obtenu après optimisation de tous les paramètres de Graphviz (exemple du pathway Kegg Butanoate)



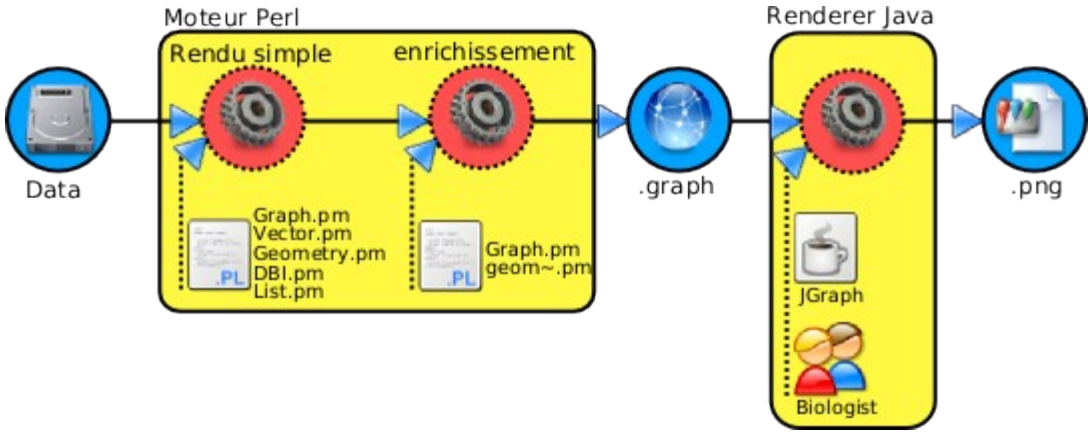
Bien que le graphe soit orienté, les résultats donnés par dot sont très mauvais et seul neato arrive a rendre quelque chose de lisible.

Le résultat n'est pas vraiment convainquant, bien qu'il n'y ait presque aucune intersection dans le graphe, il n'est pas facile à lire car il manque de structure.

Une solution pourrait être d'optimiser ce graphe en recherchant les structures remarquables et en optimisant la construction de celles-ci. Hélas, graphviz n'exporte pas les coordonnés calculées dans son modèle de données. Il n'y a donc pas moyen de récupérer le plan du graphe pour y faire des retouches (et passer par SVG est beaucoup trop fastidieux).

Principe de l'algorithme de rendu

Comme la tâche de rendu est assez complexe en soit, nous avons décidé de décomposer le programme en deux parties, le moteur de calcul de position en Perl s'occupe du calcul automatique du dessin et passe le résultat au renderer Java pour que l'utilisateur puisse effectuer des modifications si c'est nécessaire.



Les données proviennent de la base de donnée Xapien, il faut convertir dans un premier temps ces définitions biologiques de pathways en une structure plus informatique. Le moteur de rendu simple travaille sur une version du graphe où les sommets sont de simples points et où le graphe a été simplifié au maximum. Je détaille le fonctionnement du rendu simple dans la prochaine section.

L'enrichissement consiste à ajouter des informations au graphe simple, en inversant l'étape de simplification préalable au rendu simple, en ajoutant des éléments spécifiques aux pathways, en dessinant les sommets, en ajoutant toutes les informations textuelles et autres annotations ou les liens vers d'autres pathways.

Tout peut être réalisé en Perl en utilisant une série de modules spécialisés en géométrie, en manipulation de graphe et structures de listes (équivalents aux ensembles mathématiques). A la sortie, on doit obtenir un graphe avec les coordonnées calculées de tous ses éléments, prêt à être dessiné par le renderer java.

Celui-ci prend la forme d'une applet intégrée dans l'application finale de Xapien. Pour modifier le graphe, il suffit de passer dans un mode d'édition et de bouger les noeuds comme on le désire puis d'enregistrer les changements. la modification n'est donc pas une étape obligatoire, c'est simplement une possibilité offerte aux biologistes pour modifier le rendu dans l'application.

Il reste dans cette conception encore un point à étudier ... que ce passe-t-il pour les pathways qui ont été modifiés si l'on décide de les régénérer ? Il faut trouver un moyen d'enregistrer les modifications apportées par l'utilisateur entre deux générations.

La réalisation s'avère très ambitieuse pour une seule personne et je n'ai eu le temps que de construire le premier maillon de la chaîne, le moteur de rendu simple en Perl. pour achever toute la chaîne, j'estime qu'il faudrait au moins encore deux fois le temps que j'ai passe sur cette première partie, encore 4 à 5 mois de travail.

L'enrichissement du graphe en particulier est une tâche qui est au moins aussi complexe que le rendu simple. Cette partie fait d'ailleurs référence à un autre thème de recherche qui est l'annotation de graphe ou "Graph labelling". Le placement d'informations supplémentaire dans un graphe est typiquement un problème très simple à résoudre pour un humain mais compliqué à formaliser pour en faire un programme efficace.

La partie dynamique en Java et les modifications par l'utilisateur n'impose rien d'autre que de bien connaître JGraph et Java, c'est la partie la moins complexe de la chaîne.

Conception de l'algorithme

Après ces tests, il est rapidement apparu qu'il n'existait pas de solution toute faite efficace au problème d'un rendu de pathway. Il a fallu trouver une méthode spécifique en commençant par caractériser les pathways.

taille et ratios

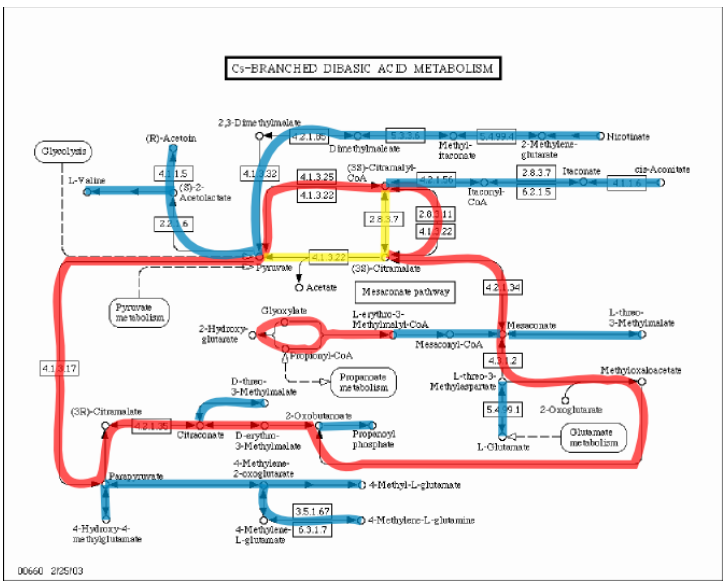
Les pathways sont en général des graphes de petite et moyenne taille, c'est à dire de moins de 500 noeuds, la moyenne se situant plutôt autour d'une centaine de noeuds par graphe. La proportion d'arc n'excède pas 2* le nombre de noeuds du graphe dans le cas général et ses arcs sont repartis équitablement entre les noeuds, globalement les pathways sont des graphes équilibrés. Quelques exceptions toutefois dans les pathways de signalisations où il arrive qu'un sommet soit une extrémité de beaucoup d'arcs.

Tous les pathways que j'ai analysé pouvaient être lisibles dans un seul écran.

Structures communes a beaucoup de graphes :

Après quelques temps d'étude des pathways kegg, on constate des structures qui reviennent souvent. ainsi, dans quasiment tous les pathways, on trouve des cycles. Après vérification auprès de David, il semble même que ce soit une propriété très intéressante dans le rendu des pathways et que généralement, on cherche à faire ressortir ce type de cycle lors du dessin. D'où l'idée de composer un algorithme de rendu construit autour de la notion de cycle de graphe.

Si l'on décide de se servir de la structure cyclique des domaines, il faut aussi chercher à caractériser tout ce qui n'est pas cyclique, les restes des noeuds du graphe. Ici encore, il suffit d'observer pour trouver la marche à suivre, une fois que l'on a tracé un cycle on constate que certains chemins du graphe vont d'un bout à l'autre de ce cycle, que d'autres font des arbres ou une extrémité est connectée à ce cycle, que d'autres enfin font le lien entre plusieurs cycles ...



L'exemple ci contre est un pathway Kegg sur lequel sur lequel les plus grands cycles ont été tracés en rouge, en jaune les liens internes, et en bleu les arbres externes ou liens externes.

En cherchant ces trois catégories dans quelques pathways kegg ou autres, j'ai constaté qu'elles étaient toujours présentes et que chacun des noeuds de chaque graphe peut être classé dans l'une de ces catégories.

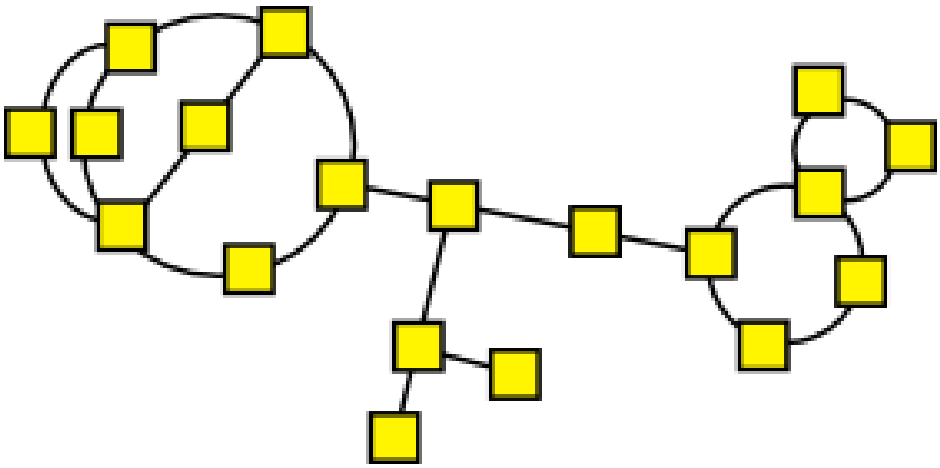
Avec un graphe caractérisé il est beaucoup plus simple d'imaginer à quoi pourrait ressembler un rendu de ce graphe et quelle méthode adopter pour y parvenir. J'ai dessiné sur papier des idées de rendu automatique de ces catégories et j'ai pu me mettre au travail après avoir discuté avec David et Guillaume pour savoir si le rendu que je proposais avait un sens au niveau biologique (bien sur).

algorithme de rendu de pathway

L'algorithme comporte trois grandes parties:

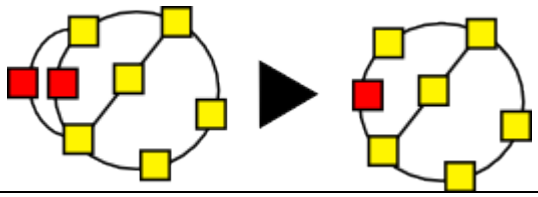
- Simplification : Dans cette première étape, on nettoie le graphe de tout ce qui est possible, en particulier, on simplifie des ensembles de noeuds en les groupant.
- Détection : toutes les structures qui composent les graphes doivent être classifiées et stockées. L'étape de détection est la plus longue de l'algorithme, certaines des opérations ne n'ont pas de solutions autre que le parcours en profondeur du graphe (ou en tous cas je n'en ai pas trouvé). C'est par exemple le cas de la recherche des cycles les plus long. Pour la plupart des problèmes de détection de chemins ou de cycles, il existe des méthodes heuristique de détermination mais ces solutions ne sont pas sûres a 100% et comme un bon rendu dépend de la sûreté de la détection, j'ai préféré implémenter une méthode certaine, quitte à perdre un peu de temps processeur.
- Rendu : malgré le nom, rien de très visuel à cette étape. On calcule simplement quelles doivent être les coordonnées des noeuds. Deux étapes dans le rendu, d'abord un calcul préalable des positions suivant un algorithme de rendu d'arbre sur des cercles concentriques. Puis on simplifie encore le graphe pour l'algorithme à modèle physique qui se charge d'optimiser la position des groupes d'éléments. Beaucoup de géométrie pour le rendu, surtout de la trigonométrie. J'ai résolu tous les déplacements du plan par des vecteurs .. tant qu'à refaire des maths, autant le faire proprement.

Comme l'algorithme est tout de même assez long, plutôt que de tenter une explication textuelle, j'ai préféré illustrer le déroulement de l'algorithme sur un exemple simple mais comportant toutefois tous les cas de figures que l'on peut trouver sur un pathway réel.



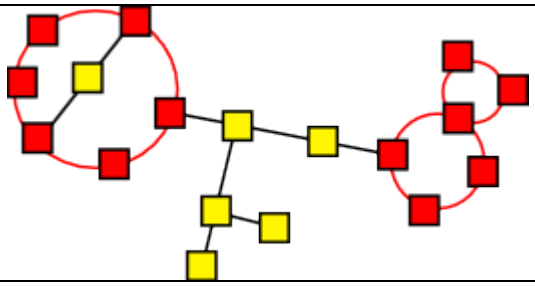
Plusieurs cycles, liens, chemins internes et un arbre externe sont présents; les noeuds sont représentés par les carrés en jaune.

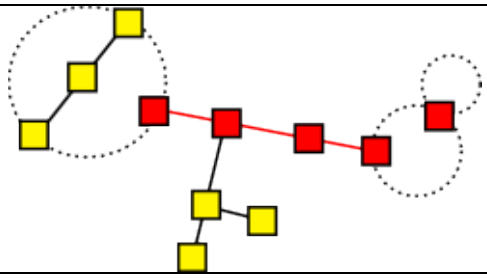
Dans les pages qui suivent, je déroule l'algorithme par étape avec une image illustrant les modifications dans le graphe à chacune des étapes.



1) On commence par simplifier le graphe en combinant les noeuds binaires qui ont mêmes extrémités.

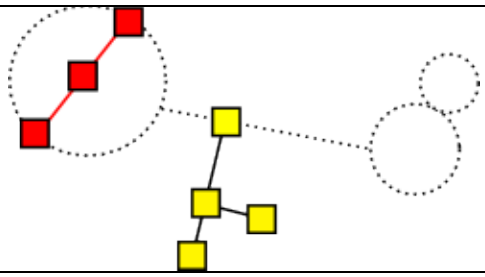
2) Ensuite, on recherche les plus longs cycles du graphe. dans l'exemple, il y en a trois dont un grand à gauche et deux petits qui sont liés par un noeud commun. La difficulté est de faire le tri entre les cycles pour prendre les plus grands. Deux cycles distincts sont liés au maximum par un noeud commun.

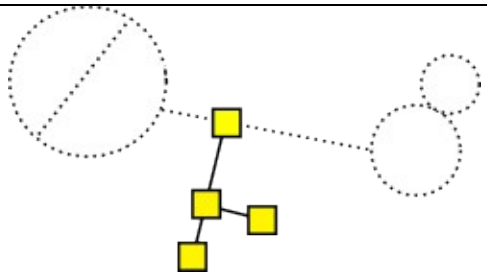




3) dans ce qui reste du graphe, on recherche maintenant les plus courts chemins entre deux cycles. On trouve deux chemins, dont un de longueur 1.

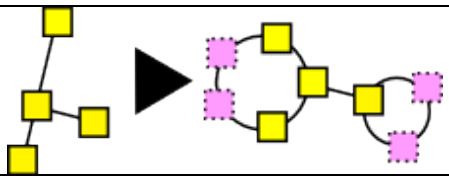
4) troisième type d'élément dans les pathways : les liens internes à un cycle. Ce sont des chemins d'éléments binaires qui commencent par un noeud appartenant à un cycle et terminant sur un autre noeud de ce même cycle.

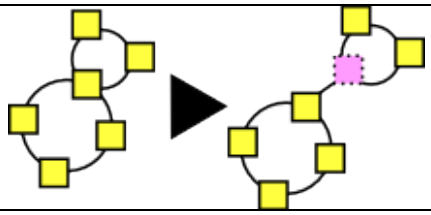




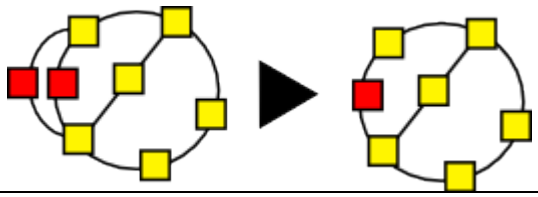
5) Il ne reste maintenant plus que des arbres ou des noeuds isolés. Avec ces derniers noeuds, on va opérer une transformation et en faire des cycles afin que la partie rendu qui suivra n'ait pas besoin de s'occuper de rendre autre chose que des cycles. La technique consiste à compléter le graphe en rajoutant des noeuds invisibles.

6) Transformation de l'arbre a 2 branches en un système de deux cycles en pointillé. On a ajouté des noeuds invisibles. On enregistre ces cycles et les nouveaux liens qui sont crés.





7) avec les liens de taille 1, il est aussi nécessaire d'effectuer une transformation pour le bon déroulement du rendu. On ajoute un noeud invisible pour qu'un lien soit toujours assimilable a une ligne que l'on peut étirer ou raccourcir

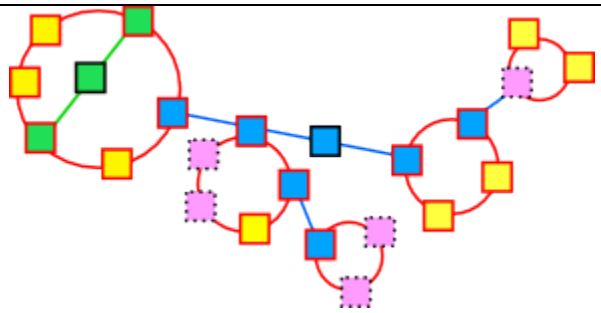


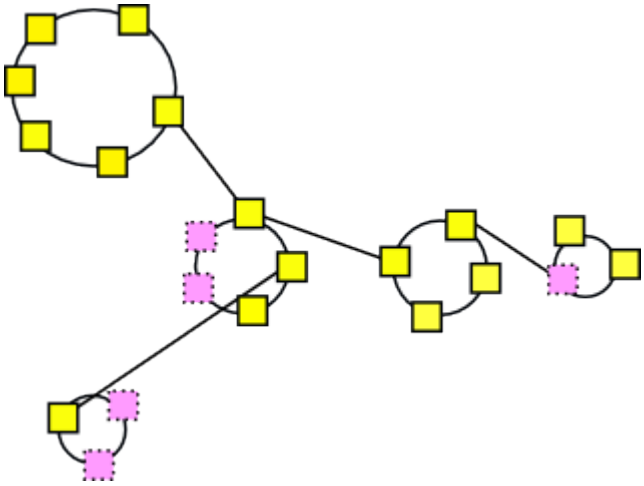
1) On commence par simplifier le graphe en combinant les noeuds binaires qui ont mêmes extrémités.

10) structure en mémoire du graphe après la partie de détection :

- cycles : avec des bordures rouges
- liens internes : contenu vert
- liens externes : contenu bleu
- noeuds invisibles : pointillé et rose

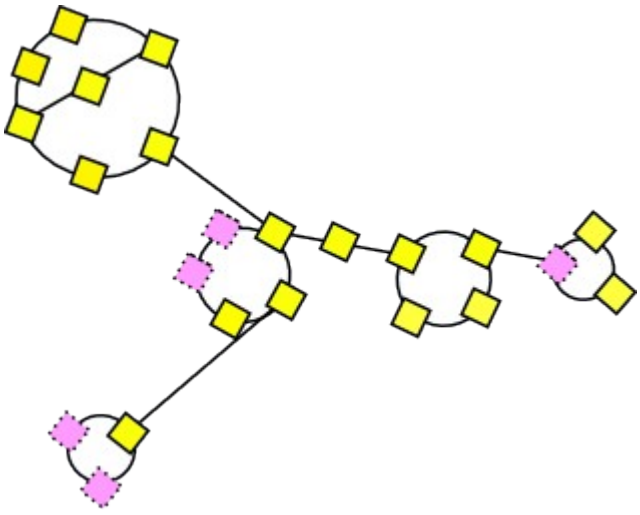
tous les noeuds appartiennent au moins à l'une de ces catégories.

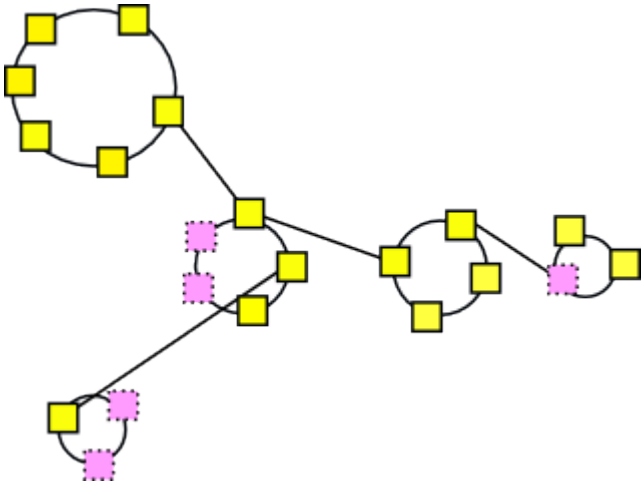




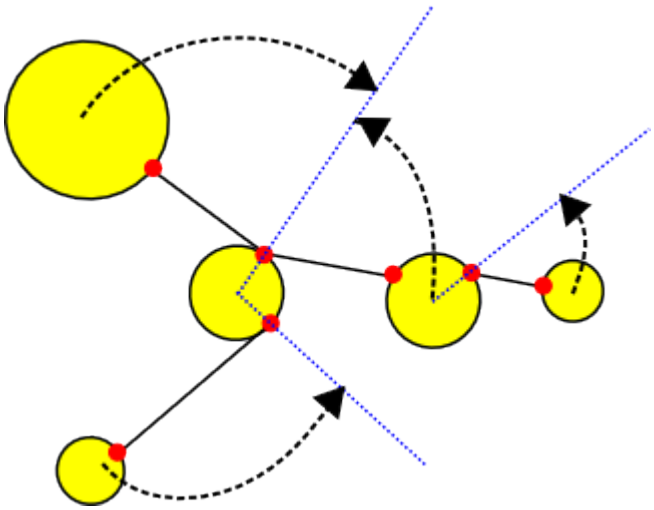
11) Ici commence le rendu à proprement parler. On commence par représenter les cycles en plaçant le cycle qui est source ou destination du plus de liens à l'origine du dessin. On répartit ses fils directs tout autour de lui (dans le bon ordre) et ensuite, récursivement on va placer tous les descendants. Le père repartit sur 360°, les fils repartissent leurs descendants directs sur 180°.
On obtient la répartition de gauche, qui n'est déjà pas si mauvaise.

12) Orientation des cycles : pour minimiser les croisements, on va orienter les cycles de façon à ce qu'ils "regardent" dans la bonne direction. Cela revient à simuler un système mécanique où mes cycles seraient des roues et où les liens seraient des élastiques. On cherche quelle est la position d'équilibre.
13) on calcule maintenant tous les autres noeuds, les liens internes et externes.



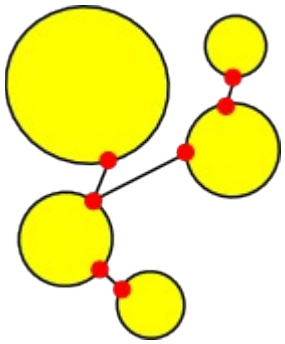


11) Ici commence le rendu à proprement parler. On commence par représenter les cycles en plaçant le cycle qui est source ou destination du plus de liens à l'origine du dessin. On répartit ses fils directs tout autour de lui (dans le bon ordre) et ensuite, récursivement on va placer tous les descendants. Le père repartit sur 360°, les fils repartissent leurs descendants directs sur 180°.
On obtient la répartition de gauche, qui n'est déjà pas si mauvaise.



14) Le graphe est simplifié encore une fois et est représenté par un système de cercles liés entre eux par des cordes attachées à leur bordure. Cette abstraction permet de vérifier très simplement les contraintes mécaniques du ...
15)... pseudo spring embed algorithme. On veut optimiser le placement des cercles en évitant toutefois qu'ils ne se rentrent les uns dans les autres.. on va donc utiliser un algo à modèle physique et effectuer des petits mouvements vers les lignes en bleu qui représentent les attaches des ressorts et s'arrêter quand on cogne contre un autre élément.
Les attaches sont sur la ligne (centre du père ->

extrémité) du lien vers le fils et ces positions sont recalculées à chaque tour de boucle (chaque petit mouvement).
16) Voila le résultat après un petit moment de dynamique, on a réduit l'aire occupée par les cercles.



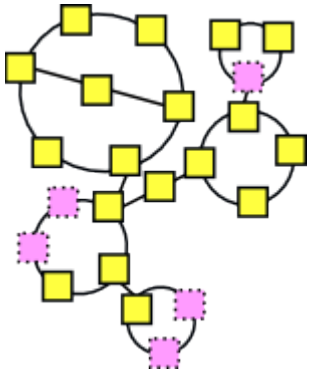
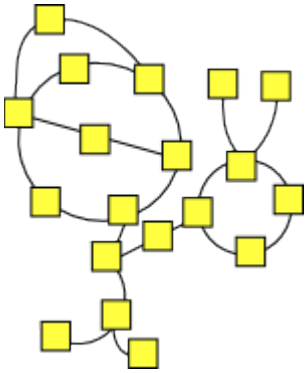
	17) On peut repasser en mode graphe complet ...
18) ... et la dernière étape consiste à re-complexifier le graphe, c'est à dire supprimer les dummy nodes et diviser à nouveau les noeuds qui en contiennent plusieurs. Le résultat final donne un graphe lisible, compact sans croisement et particulièrement adapté aux pathways puisqu'il fait ressortir les cycles chimiques.	

Diagramme de classe (page suivante)

Classe Pathway

Les classes sont organisées autour de Pathway qui fait le lien entre tous les objets. c'est Pathway qui commande l'ordre du programme : simplification, détection des cycles, des liens, re-simplification, rendu simple, spring algorithme, re-complexification. Cette ordre est demandé depuis le programme principal sur l'objet pathway. L'ordre doit absolument être respecté sous peine de non fonctionnement du programme.

Un pathway se compose de Cycles et de liens. Pathway stocke deux versions du graphe : graph est la version de travail, simplifiée et pathway est la version complète, telle qu'on la lit depuis la base de donnée.

Les Dummy sont des sommets comme les autres à l'exception que leur ID soit supérieur a 5000, ce qui signifie que l'on ne peut pas représenter un graphe de plus de 5000 sommets.

La méthode Pathway->coordinates() est l'une des plus importantes, c'est ici que l'on gère l'ordre de calcul des positions, si les cycles devaient être dessinés sur un mode différent que la hiérarchie ou pour toute autre modification influant sur le rendu final, tout se règle dans cette méthode.

Classe Cycle

Contient toutes les informations sur un cycle (ses nodes) ainsi que sur son père et propose des méthodes pour manipuler ses descendants.

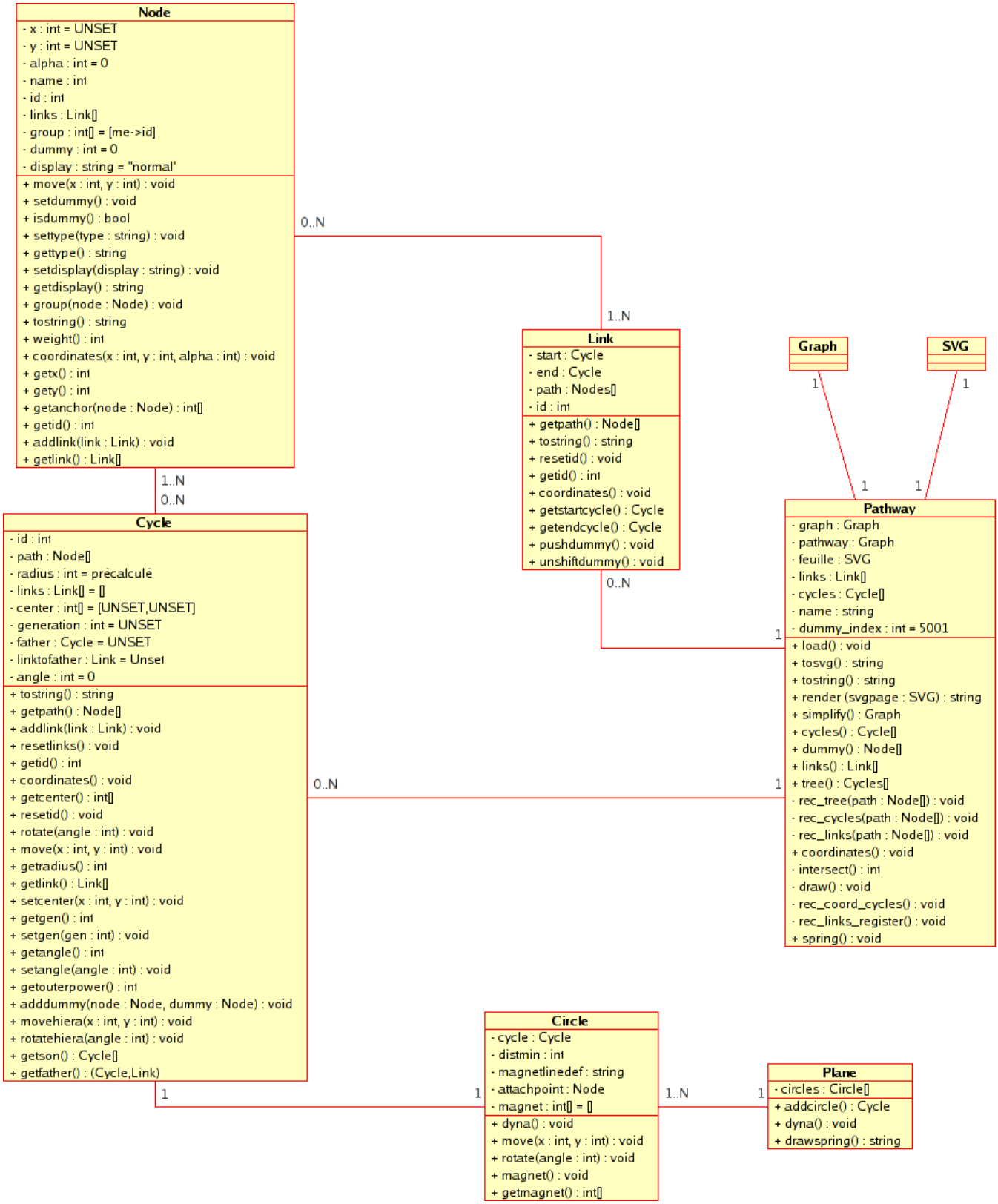
- Cycle->coordinates() : calcule les coordonnés des membres du cycles
- Cycle->rotate()/move() : permettent de déplacer ou de tourner sur lui même un cycle.
- Cycle->rotatehiera()/movehiera() : même chose sauf que le cycle emmène ses descendant dans le mouvement.
- Cycle->getouterpower : pour l'étape 12, orientation des cycles, cette méthode calcule le score pour une position donnée. plus ce score est bas, meilleure est la position.

Classe Node

Classe simple définissant un sommet (ou un groupe de sommet) du graphe, ses coordonnés et

tout ce qui peut caractériser son affichage.

- Node->group ajoute un noeud à l'intérieur d'un autre et définit un groupe de noeuds (étape 1)
- Node->move()/coordinates() : pour enregistrer les coordonnées et pour déplacer les noeuds.



Classe Link

Link représente indifféremment un lien interne où un lien externe entre deux cercles. La différence se situe juste au niveau des points de départ et d'arrivée ou le lien interne aura $end=$start. Les méthodes de cette classe sont très simples :

- push/unshiftdummy() permet d'ajouter un noeud invisible à un lien, soit au début soit à la fin du chemin qui compose le lien.
- coordinates() : calcule la position des noeuds dans le lien. on repartit simplement les noeuds équitablement entre le début et la fin.

Classes Circle et Plane

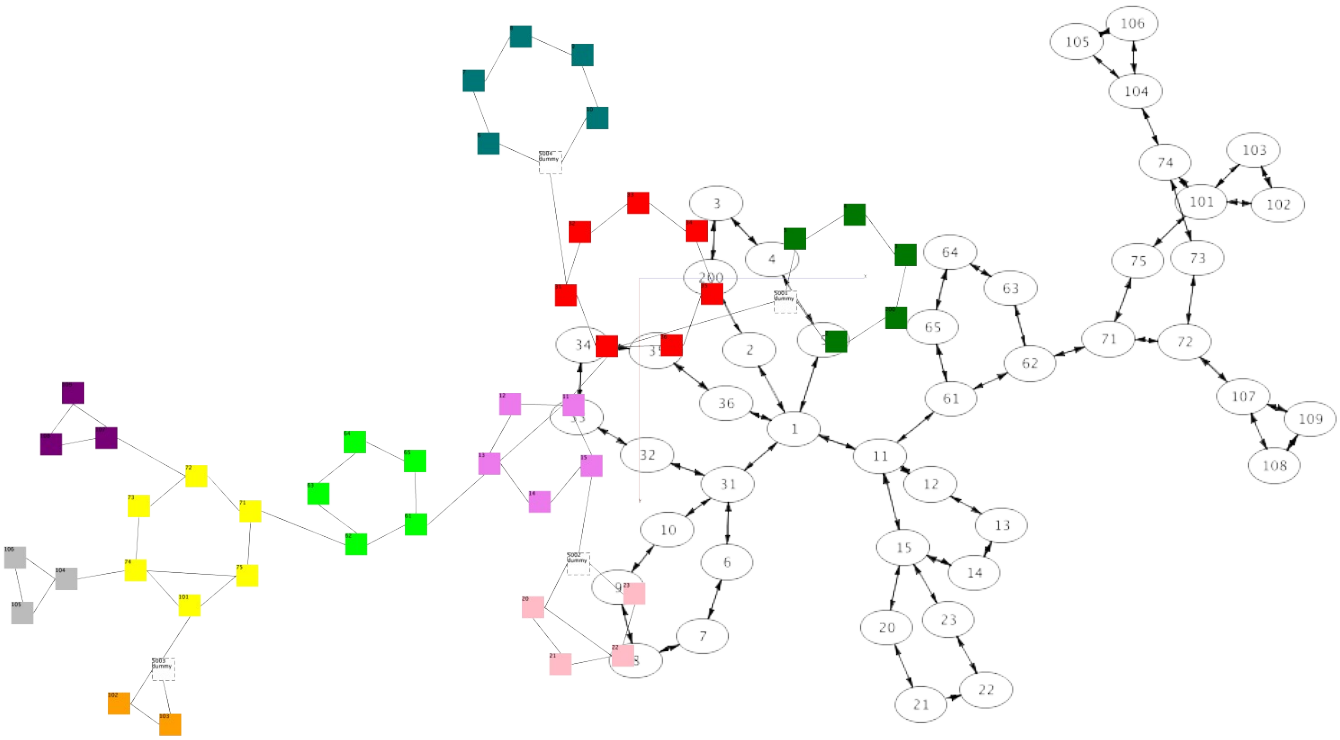
Elles servent à l'algorithme à modèle physique. Circle et Plane sont une vue du système pathway (interface ?) où le but est d'optimiser les temps de calcul des intersections.
C'est la méthode Plane->dyna() qui commande et contrôle l'exécution de cette partie de l'algorithme.

Tests et performance

Même si je n'ai pas eu le temps de finir le sujet dans son ensemble, j'ai quand même contrôlé a de nombreuses reprises le fonctionnement du programme. Le jeux de test fourni avec le code permet de tester le comportement du programme sur les cas extrêmes de pathways, pour tester la représentation de graphes particuliers et être sûr du comportement du programme. Jusqu'à la dernière version, le jeu de test a fonctionné correctement, je suis donc confiant pour la suite que Xapien pourra donner à ce projet.

Du point de vue de la performance, sur des graphes plus compliqués, le programme perd beaucoup de temps lors de la phase de détection. il doit exister des moyen pour optimiser l'algorithme et si le temps devient une question cruciale, il faudra songer à passer à un langage compilé de type C++.
Je n'ai pas eu le temps d'aller plus loin que l'algorithme à modèle physique, celui ci est déjà bien entamé, j'ai calculé les point magnétiques pour chacun des cercles, il n'y a plus qu'a compléter plane->dyna() et à créer les méthodes de détections d'intersections.

Pour conclure sur cette partie, voici le dernier rendu du premier exemple de mon jeu de tests (en haut) et son équivalent rendu par Graphviz (en bas), on peut estimer que le résultat est déjà très correct, sachant que ce graphe n'a pas encore été optimisé.



Conclusion

Cinq mois font-ils plus que des années d'études ? C'est l'impression en tout cas qu'il me restera de mon stage. J'ai le sentiment d'avoir appris énormément mais pas comme on pourrait le penser au niveau de mes connaissances informatiques mais plutôt sur tout le reste. J'ai vu à Xapien ce qu'est une équipe de travail, j'ai vu mes collègues se démener, passer des journées au téléphone pour faire avancer leur projet d'entreprise, croire en leurs capacités et arriver à communiquer cette confiance aux autres.

Xapien, nouvelle entreprise, Start-Up, m'a fait voir comment se construit une entreprise de l'intérieur, quelles sont les pièges a éviter, comment réussir à y apporter une valeur ajoutée, comment prévoir le long terme tout en gérant les galères au jour le jour ...

Je suis infiniment reconnaissant envers toute l'équipe de m'avoir si bien accueilli, de m'avoir complètement intégré à son fonctionnement, de m'avoir fait une place à toutes les réunions, les "monday-ten-o-clock meeting" ; j'ai le sentiment que cette expérience me sera hautement profitable et je vois déjà la création d'entreprise d'un oeil neuf.

Un seul regret toutefois : les financements n'arrivent pas assez vite pour pouvoir me garder une place au chaud en septembre et je remets ma sortie de l'école à un peu plus tard .. tant pis.

~ Littérature ~

Bibliographie

Pathways

- *H. Kitano* "The standard graphical notification for biological network"
- *R. Tasmassia* "The Graph Drawing tutorial"
- *R. Tasmassia* "Algorithms for drawing graphs : an annotated bibliography"
- *R. Lipton, S.north, J. Sandberg* "A method for drawind graphs"
- *P. Eades* "A Heuristic for graph Drawing"
- *T. Kamada, S. Kawai* "Automatic display of network structures for human understanding"
- *T. Kamada, S. Kawai* "An algorithm for drawing General Undirected Graphs"

représentation de protéines

- the xapien Bible
- papier "for Mathieu" d'explication
- "Lexique de biologie" Larousse

Webographie

Représentation de protéines

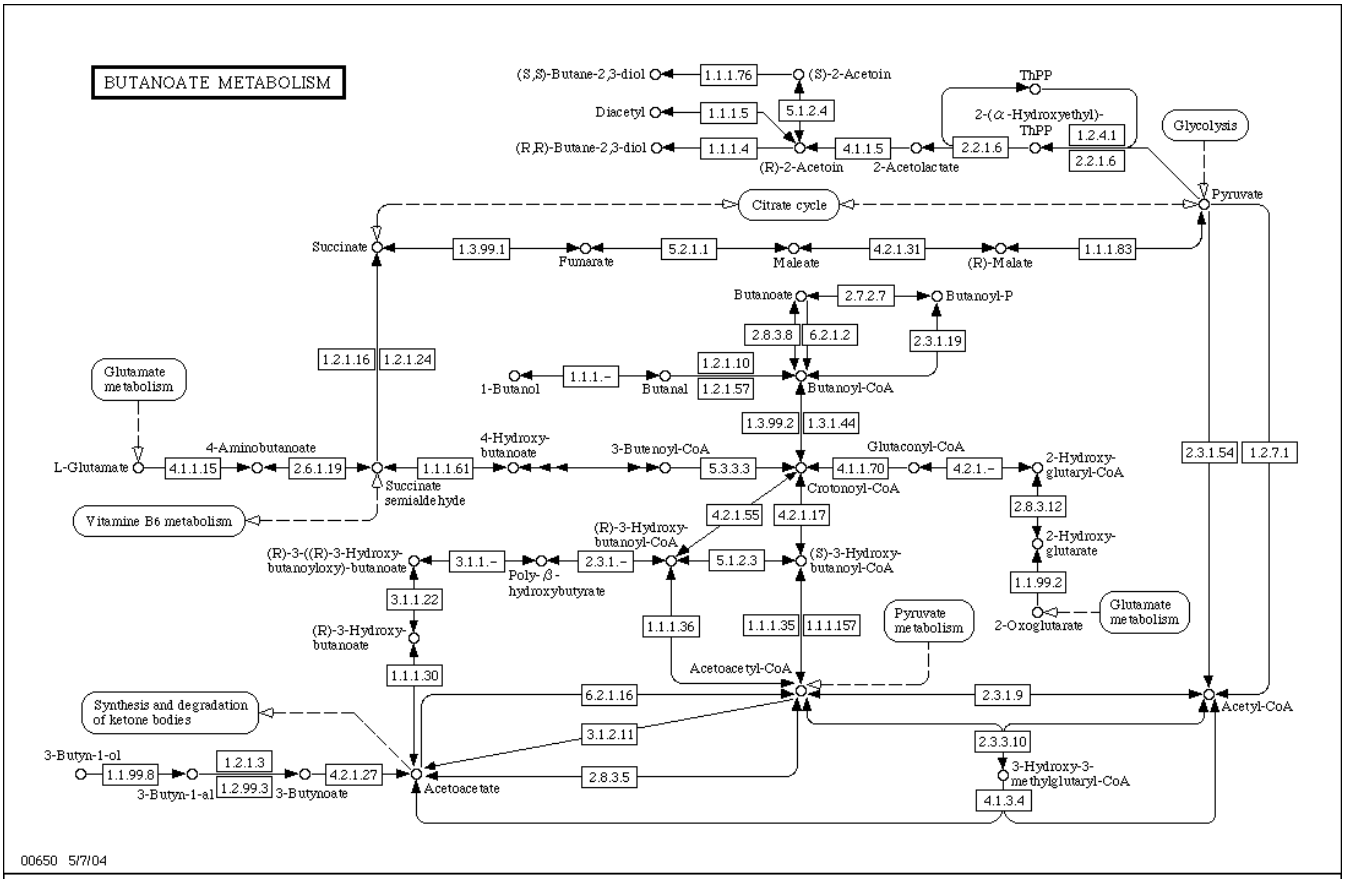
- Prosite : http://expasy.hcuge.ch/prosite/
- Smart : http://smart.embl-heidelberg.de/
- Pfam : http://pfam.wustl.edu/
- ProDom : http://protein.toulouse.inra.fr/prodom/current/html/home.php
- cours : http://www.people.virginia.edu/~wrp/cshl00/domain-lecture.html
- Blocks : http://www.blocks.fhcrc.org/

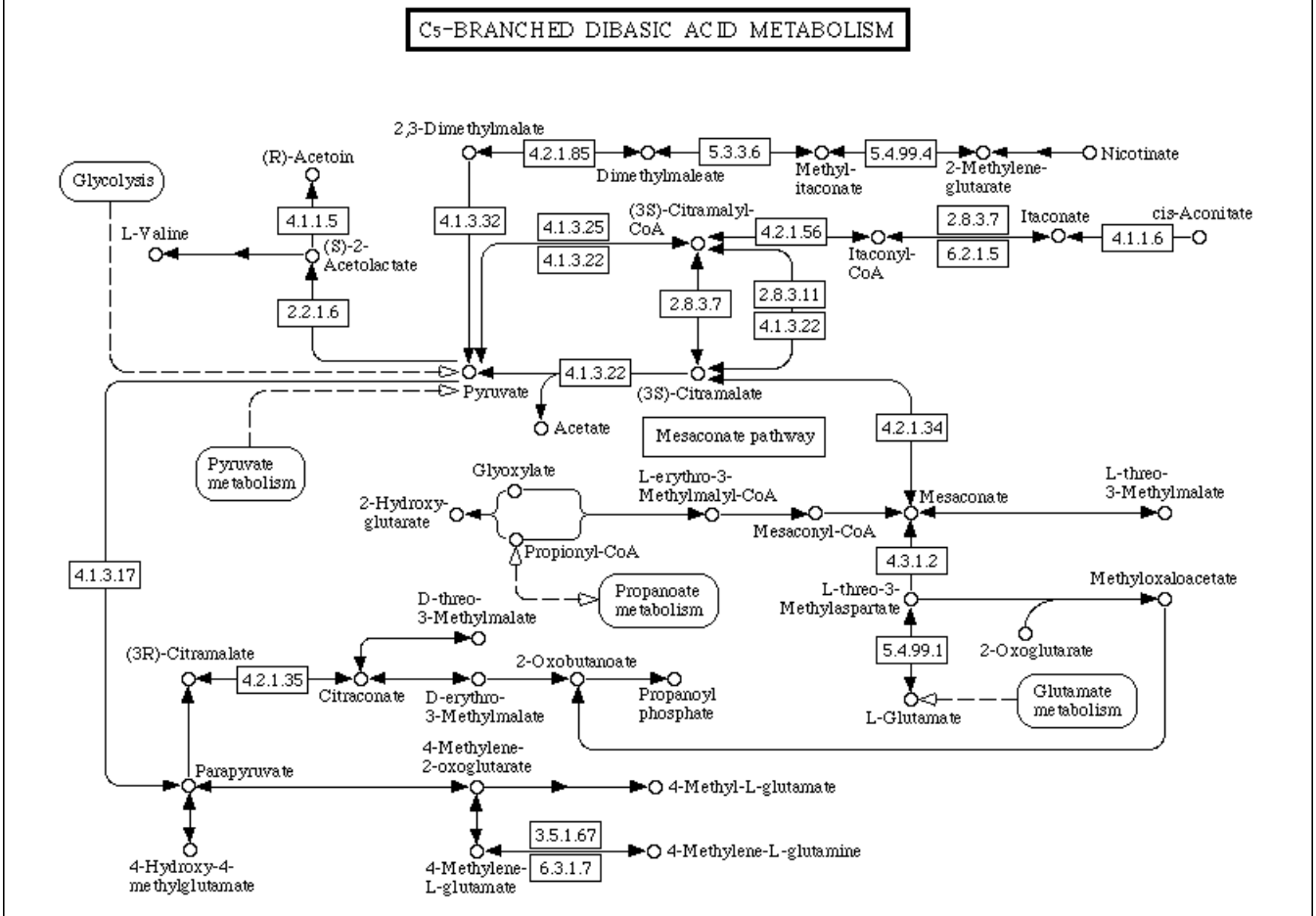
Pathways

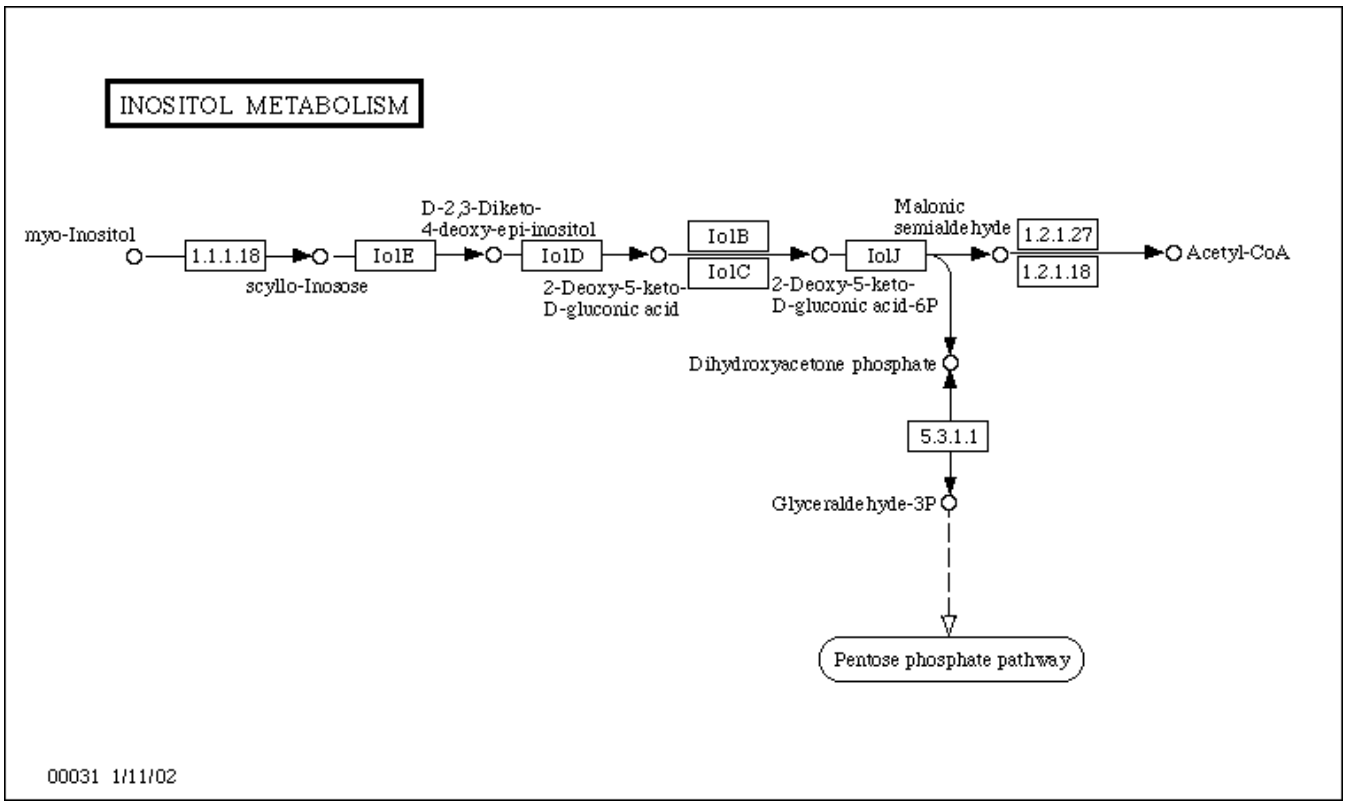
- R. Tasmassia : www.cs.brown.edu/people/rt/gd.html
- GraphViz : www.research.att.com/sw/tools/graphviz
- Govisual : www.oreas.com
- tomsawyer : www.tomsawyer.com
- links : http://rw4.cs.uni-sb.de/users/sander/html/gstools.html
- Wikipedia definition : http://en.wikipedia.org/wiki/Graph_drawing
- Spring layout definition : http://en.wikipedia.org/wiki/Spring_layout

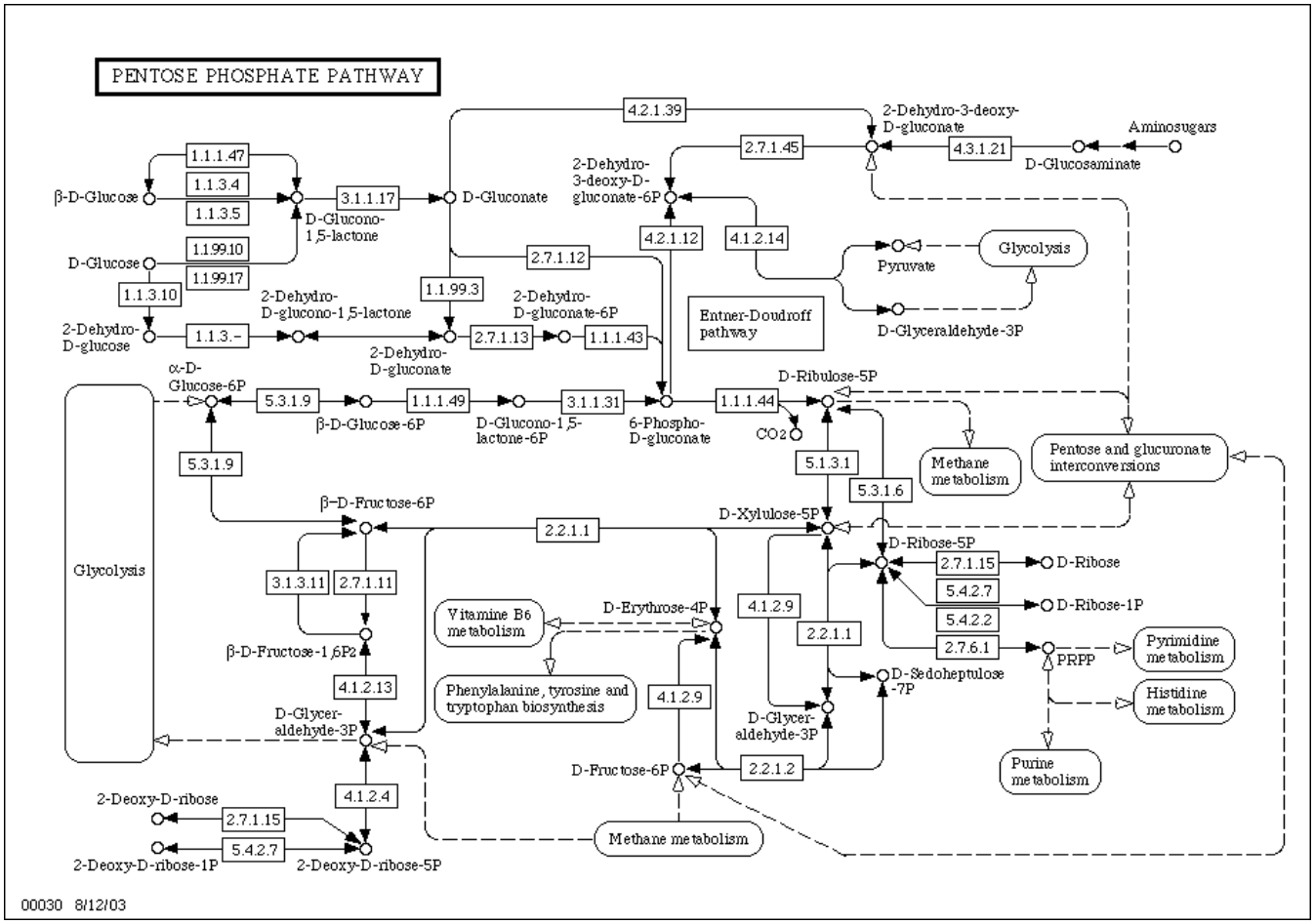
ANNEXE

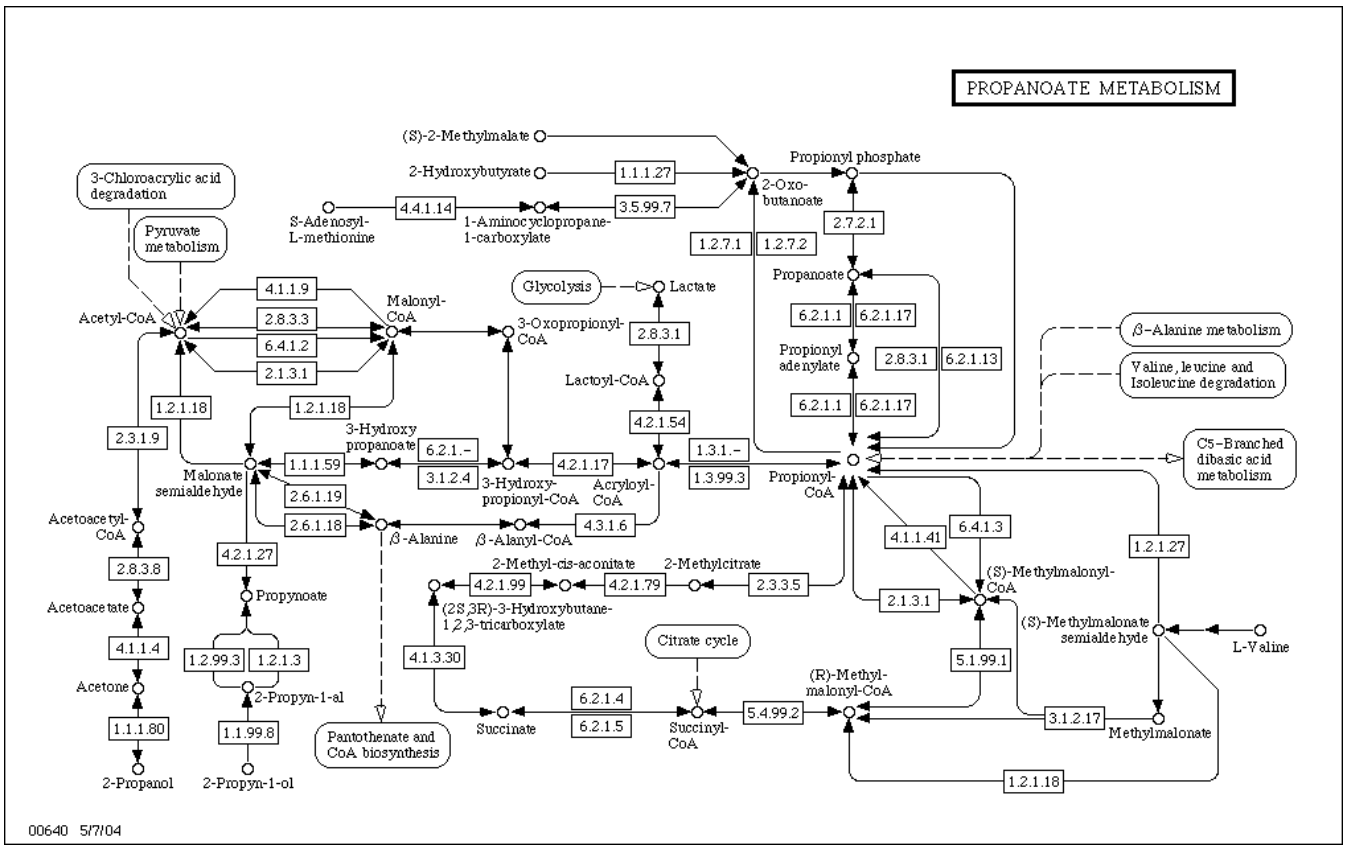
Pathways Kegg utilisés pour déterminer la méthode de dessin de graphes


BUTANOATE METABOLISM
(S,S)-Butane-2,3-diol
1.1.1.76
(S)-2-Acetoin
ThPP
Diacetyl
1.1.1.5
5.1.2.4
2-(α-Hydroxyethyl)-ThPP
1.2.4.1
Glycolysis
(R,R)-Butane-2,3-diol
1.1.1.4
4.1.1.5
2.2.1.6
(R)-2-Acetoin
2-Acetolactate
2.2.1.6
Pyruvate
Citrate cycle
Succinate
1.3.99.1
Fumarate
5.2.1.1
Maleate
4.2.1.31
(R)-Malate
1.1.1.83
Butanoate
2.7.2.7
Butanoyl-P
2.8.3.8
6.2.1.2
2.3.1.19
1.2.1.16
1.2.1.24
Glutamate metabolism
1-Butanol
1.1.1.-
Butanal
1.2.1.10
1.2.1.57
Butanoyl-CoA
1.3.99.2
1.3.1.44
L-Glutamate
4.1.1.15
4-Aminobutanoate
2.6.1.19
1.1.1.61
4-Hydroxybutanoate
3-Butenoyl-CoA
5.3.3.3
4.1.1.70
Glutaconyl-CoA
4.2.1.-
2-Hydroxyglutaryl-CoA
2.3.1.54
1.2.7.1
Succinate semialdehyde
Crotonoyl-CoA
Vitamine B6 metabolism
4.2.1.55
4.2.1.17
2.8.3.12
(R)-3-Hydroxybutanoyl-CoA
2-Hydroxyglutarate
(R)-3-((R)-3-Hydroxybutanoyloxy)-butanoate
3.1.1.-
2.3.1.-
Poly-β-hydroxybutyrate
5.1.2.3
(S)-3-Hydroxybutanoyl-CoA
1.1.99.2
3.1.1.22
Pyruvate metabolism
Glutamate metabolism
2-Oxoglutarate
(R)-3-Hydroxybutanoate
1.1.1.36
1.1.1.35
1.1.1.157
1.1.1.30
Acetoacetyl-CoA
Synthesis and degradation of ketone bodies
6.2.1.16
2.3.1.9
Acetyl-CoA
3.1.2.11
2.3.3.10
3-Butyn-1-ol
1.1.99.8
1.2.1.3
4.2.1.27
3-Hydroxy-3-methylglutaryl-CoA
1.2.99.3
3-Butyn-1-al
3-Butynoate
Acetoacetate
2.8.3.5
4.1.3.4
00650 5/7/04


C5-BRANCHED DIBASIC ACID METABOLISM
2,3-Dimethylmalate
4.2.1.85
5.3.3.6
5.4.99.4
Nicotinate
(R)-Acetoin
Dimethylmaleate
Methyl-itaconate
2-Methylene-glutarate
Glycolysis
4.1.3.32
4.1.3.25
(3S)-Citramalyl-CoA
4.2.1.56
2.8.3.7
Itaconate
cis-Aconitate
L-Valine
4.1.1.5
(S)-2-Acetolactate
4.1.3.22
Itaconyl-CoA
6.2.1.5
4.1.1.6
2.2.1.6
2.8.3.7
2.8.3.11
4.1.3.22
4.1.3.22
Pyruvate
(3S)-Citramalate
Acetate
Mesaconate pathway
4.2.1.34
Pyruvate metabolism
Glyoxylate
L-erythro-3-Methylmalyl-CoA
Mesaconate
L-threo-3-Methylmalate
2-Hydroxy-glutarate
Mesaconyl-CoA
Propionyl-CoA
4.3.1.2
4.1.3.17
Methyloxaloacetate
D-threo-3-Methylmalate
Propanoate metabolism
L-threo-3-Methylaspartate
(3R)-Citramalate
5.4.99.1
2-Oxoglutarate
4.2.1.35
2-Oxobutanoate
Citraconate
D-erythro-3-Methylmalate
Propanoyl phosphate
L-Glutamate
Glutamate metabolism
4-Methylene-2-oxoglutarate
Parapyruvate
4-Methyl-L-glutamate
3.5.1.67
4-Methylene-L-glutamine
4-Hydroxy-4-methylglutamate
4-Methylene-L-glutamate
6.3.1.7
00660 2/25/03


INOSITOL METABOLISM
myo-Inositol
1.1.1.18
scyllo-Inosose
IolE
D-2,3-Diketo-
4-deoxy-epi-inositol
IolD
2-Deoxy-5-keto-
D-gluconic acid
IolB
IolC
2-Deoxy-5-keto-
D-gluconic acid-6P
IolJ
Malonic
semialdehyde
1.2.1.27
1.2.1.18
Acetyl-CoA
Dihydroxyacetone phosphate
5.3.1.1
Glyceraldehyde-3P
Pentose phosphate pathway
00031 1/11/02


PENTOSE PHOSPHATE PATHWAY
β-D-Glucose
D-Glucose
2-Dehydro-
D-glucose
1.1.1.47
1.1.3.4
1.1.3.5
1.1.99.10
1.1.99.17
1.1.3.10
1.1.3.-
3.1.1.17
D-Glucono-
1,5-lactone
D-Gluconate
2-Dehydro-
D-glucono-1,5-lactone
2-Dehydro-
D-gluconate
1.1.99.3
2.7.1.13
2-Dehydro-
D-gluconate-6P
1.1.1.43
4.2.1.39
2-Dehydro-3-deoxy-
D-gluconate
2.7.1.45
4.3.1.21
Aminosugars
D-Glucosaminate
2-Dehydro-
3-deoxy-D-
gluconate-6P
4.2.1.12
4.1.2.14
2.7.1.12
Pyruvate
Glycolysis
D-Glyceraldehyde-3P
Entner-Doudroff
pathway
α-D-
Glucose-6P
5.3.1.9
β-D-Glucose-6P
1.1.1.49
D-Glucono-1,5-
lactone-6P
3.1.1.31
6-Phospho-
D-gluconate
1.1.1.44
CO2
D-Ribulose-5P
5.1.3.1
5.3.1.6
Methane
metabolism
Pentose and glucuronate
interconversions
β-D-Fructose-6P
2.2.1.1
D-Xylulose-5P
D-Ribose-5P
2.7.1.15
D-Ribose
5.4.2.7
D-Ribose-1P
5.4.2.2
2.7.6.1
PRPP
Pyrimidine
metabolism
Histidine
metabolism
Glycolysis
3.1.3.11
2.7.1.11
β-D-Fructose-1,6P2
Vitamine B6
metabolism
D-Erythrose-4P
4.1.2.9
2.2.1.1
Phenylalanine, tyrosine and
tryptophan biosynthesis
4.1.2.13
4.1.2.9
D-Sedoheptulose
-7P
D-Glycer-
aldehyde-3P
D-Glycer-
aldehyde-3P
D-Fructose-6P
2.2.1.2
Purine
metabolism
4.1.2.4
2-Deoxy-D-ribose
2.7.1.15
5.4.2.7
2-Deoxy-D-ribose-1P
2-Deoxy-D-ribose-5P
Methane metabolism
00030 8/12/03


PROPANOATE METABOLISM
(S)-2-Methylmalate
2-Hydroxybutyrate
1.1.1.27
Propionyl phosphate
3-Chloroacrylic acid degradation
2-Oxo-butanoate
4.4.1.14
3.5.99.7
S-Adenosyl-L-methionine
1-Aminocyclopropane-1-carboxylate
2.7.2.1
1.2.7.1
1.2.7.2
Pyruvate metabolism
Propanoate
4.1.1.9
Glycolysis
Lactate
Malonyl-CoA
Acetyl-CoA
2.8.3.3
6.4.1.2
2.1.3.1
3-Oxopropionyl-CoA
2.8.3.1
6.2.1.1
6.2.1.17
β-Alanine metabolism
Propionyl adenylate
2.8.3.1
6.2.1.13
Valine, leucine and Isoleucine degradation
Lactoyl-CoA
6.2.1.1
6.2.1.17
1.2.1.18
1.2.1.18
4.2.1.54
2.3.1.9
3-Hydroxy propanoate
6.2.1.-
1.3.1.-
C5-Branched dibasic acid metabolism
1.1.1.59
4.2.1.17
3.1.2.4
1.3.99.3
Malonate semialdehyde
3-Hydroxy-propionyl-CoA
Acryloyl-CoA
Propionyl-CoA
2.6.1.19
2.6.1.18
4.3.1.6
Acetoacetyl-CoA
β-Alanine
β-Alanyl-CoA
4.1.1.41
6.4.1.3
1.2.1.27
4.2.1.27
2-Methyl-cis-aconitate
2-Methylcitrate
2.8.3.8
4.2.1.99
4.2.1.79
2.3.3.5
(S)-Methylmalonyl-CoA
Propynoate
Acetoacetate
(2S,3R)-3-Hydroxybutane-1,2,3-tricarboxylate
2.1.3.1
(S)-Methylmalonate semialdehyde
L-Valine
4.1.1.4
1.2.99.3
1.2.1.3
Citrate cycle
4.1.3.30
5.1.99.1
Acetone
2-Propyn-1-al
(R)-Methyl-malonyl-CoA
6.2.1.4
6.2.1.5
5.4.99.2
3.1.2.17
1.1.1.80
1.1.99.8
Pantothenate and CoA biosynthesis
Succinate
Succinyl-CoA
Methylmalonate
2-Propanol
2-Propyn-1-ol
1.2.1.18
00640 5/7/04