Projet Algorithmie

Problème d'emploi du temps

Paul Delarue DELP02049907
Louis Ledru LEDL13040001
Julia Mourier MOUJ21579909

Hiver 2022

Abstract

Les problèmes d'emplois du temps sont des problèmes complexes et extrêmement intéressants. C'est un problème auquel n'importe qui peut se trouver confronter, mais chacun aura sa propre version du problème, avec ses propres contraintes et objectifs.

Il s'agit dans cet article d'expliquer notre travail sur la résolution d'un problème d'emplois du temps.

Après quelques recherches sur le sujet, nous avons décidé d'écrire nos propres implémentations des algorithmes. A savoir, un algorithme exact et un autre approché. L'algorithme exact reprend les fondements de la méthode gloutonne et l'algorithme approché utilise les nouvelles avancées des ordinateurs quantiques.

Notre problème d'emplois du temps consiste en un groupe d'élèves devant se voir attribuer des cours qu'ils auront choisis et triés par ordre de préférence. Le but de nos algorithmes est de retourner les cours assignés aux élèves pour que ceux-ci soient le plus satisfaits possible. L'algorithme exact est capable de résoudre une instance de 200 élèves en un temps raisonnable tandis que celui approché peut gérer plus de 700 élèves sans soucis.

La résolution de ce problème nous a permis d'appliquer et d'approfondir nos connaissances, mais nous a aussi permis de s'intéresser à des problématiques d'optimisation ainsi qu'aux ordinateurs quantiques.

Contents

1 Introduction

1.1 Description et formulation du problème

Le problème d'emploi du temps porte bien son nom, c'est l'utilisation du temps que l'on cherche à optimiser. Il peut être utilisé pour tout un tas d'applications que l'on exprimera plus tard avec quelques exemples. Néanmoins, chaque application est unique et possède ses normes et critères qui lui sont propres et le caractérisent, ces contraintes seront explicitées dans le prochain point.

Le but de ce problème est d'affecter à des ressources très variées, avec des contraintes toutes aussi variées, un temps d'utilisation de façon à ce que l'utilisation de ces ressources soit optimale. Le tout étant fortement combinatoire, cela fait que ce problème est NP-Difficile[6].

Pendant très longtemps il a fallu effectuer ce travail fastidieux à la main, demandant parfois à plusieurs personnes de travailler dessus pendant plusieurs jours, espérant qu'une contrainte impromptue ne se montre pas, cela pouvant remettre en cause tout le travail effectué auparavant. C'est pour cela que dès la fin des années 50 des recherches ont été effectuées pour essayer d'automatiser le problème. C'est aujourd'hui chose faite et nous allons vous expliquer comment fonctionne le problème d'emploi du temps, et comment il peut être résolu.

1.2 Contraintes et entrées

Comme dit dans le point précédent, les contraintes et entrées du problème dépendent de l'application même du problème. C'est un problème de répartition donc il faut tout d'abord déterminer la nature de ce qui doit être réparti et aussi l'espace de répartition. Par exemple, dans le cas d'un projet en entreprise, il peut être nécessaire de répartir des tâches sur un temps donné. Les tâches sont les données à répartir et le temps est l'espace de répartition.

Il se peut que les données aient des contraintes entre elles. Deux données peuvent par exemple être liées et ce lien constitue une contrainte. Si on prend le même exemple, si une tâche nécessite la réalisation d'une autre pour pouvoir être réalisée, alors elles ne pourront pas s'effectuer en même temps et devront être agencées d'une certaine manière (l'une avant l'autre). Cet agencement constitue alors la contrainte.

Il est dit dans l'article[2] qu'il existe deux types de contraintes. Les contraintes dures qui nécessitent d'être respectées et les contraintes souples qui ne sont pas forcément nécessaires, mais qui devraient être respectées le plus possible.

En d'autres termes, les entrées constituent les données à répartir et peuvent être de natures multiples et les contraintes constituent les exigences appliquées à ces données. C'est-à-dire les façons dont elles peuvent être liées ou non.

1.3 Exemples d'utilisation

Il existe de nombreuses applications à ce problème avec de nombreuses méthodes de résolution. On peut citer 3 domaines dans lesquels ces problèmes sont récurrents : la pédagogie, la santé et le transport[3].

Dans le domaine de la pédagogie, il nous est facile d'imaginer les problèmes possibles. Ils sont souvent appelés les problèmes d'emploi du temps . Comment répartir les étudiants dans les horaires de cours sans créer de conflit et tout en s'assurant que les étudiants sont satisfaits ? Dans ce cas, les données d'entrée sont les étudiants et les horaires de cours, les contraintes sont les demandes de cours données par les étudiants ainsi que le nombre d'étudiants disponibles pour un même cours. Comment répartir les examens pour que les étudiants n'aient pas plusieurs examens sur le même horaire ? Qu'ils n'aient pas 3 examens le même jour[2] ? Dans ce cas, les données d'entrée sont les étudiants et la liste des examens disponible, les contraintes sont les listes d'examens que chaque étudiant doit passer ainsi que le nombre maximum d'examens qu'un étudiant peut passer le même jour.

Pour le transport, il s'agit en général de problèmes liés à la fréquence de passage de bus, voitures avec l'attribution du personnel qualifié, ces données constituent les entrées du problème. Il faut prendre en compte les compétences des travailleurs ainsi que leurs horaires de repos, ces données constituent les contraintes du problème. L'article[5] propose une résolution par intégration de l'entièreté du problème et non un découpage de celui-ci.

Dans la santé, le problème est sensiblement

identique, il faut attribuer du personnel, du matériel médical aux patients qui en ont besoin à des horaires précis. Dans l'article[4], l'auteur présente le problème de répartition des infirmières résolu grâce à un programme utilisant des ordinateurs à recuit quantique.

1.4 Possibilités d'implémentation

Il existe beaucoup de méthodes utilisées pour la résolution du problème d'emploi du temps ; voici une liste non-exhaustive des méthodes que nous avons trouvées :

Les algorithmes donnant la solution exacte :

- algorithme glouton
- méthode de retour en arrière
- programmation dynamique
- diviser et régner...

Le principal problème des méthodes exactes est la difficulté de résolution face aux instances de grande importance menant souvent à une utilisation des ressources trop importantes.

Les algorithmes aux solutions approchées :

- à base de voisinage
- de descente
- recuit simulé
- méthode d'acceptation à seuil
- bruitage
- méthode tabou
- évolutionnistes (dont génétique, colonies de fourmis...)
- d'autres méthodes hybrides...

Comme son nom l'indique, le principal problème des algorithmes approchés est que l'on n'a pas de certitude d'avoir la solution optimale. Cependant ils sont aujourd'hui très utilisés par leur rapidité et leur moindre coût.

1.5 Formulation de notre problème

Dans notre cas, notre projet porte sur la distribution d'étudiants dans des cours. Les étudiants devront être répartis dans un nombre de cours établis à l'avance. Ils devront faire des voeux sur le choix des cours dans une base de données. Le nombre de voeux est supérieur au nombre de cours qui sera affecté à l'étudiant. Les voeux sont classés par ordre de préférence.

Par exemple, les étudiants devront choisir 7 voeux de cours dans tous les cours disponibles. 5 leur seront attribués.

Les cours à choisir auront une capacité d'étudiant maximale.

Le but est de satisfaire le plus possible les étudiants.

1.6 Choix d'implémentation

Parmi les possibilités énoncées plus haut, nous avons choisi d'implémenter un algorithme de backtracking (retour en arrière en français) pour la méthode exacte. La méthode du retour en arrière est proposée dans l'article [2] et nous l'avons choisi car nous étions familiers avec elle. En effet, nous avons déjà implémenté cette méthode dans des projets passés.

Intéréssés par les technologies naissantes et l'informatique quantique nous avons choisi d'implémenter une méthode approchée grâce à des ordinateurs à recuit simulé quantique mis en libre accès par la société D-Wave.

2 Algorithme exact

Pour la résolution exacte de notre problème, notre but est d'attribuer le bon nombre de cours à tous les élèves en maximisant la satisfaction globale de la solution. Pour cela, il faut donc avoir une satisfaction maximale pour chacun des élèves, cette satisfaction est calculée selon les cours qui lui sont attribués. Si un élève doit se faire attribuer 4 cours alors les 4 premiers cours de sa liste de vœux ont une satisfaction de 100, le prochain aura une satisfaction de 80, le suivant de 60 et ainsi de suite jusqu'au dernier vœu. Nous n'avons aucunement le droit de faire autre chose que d'assigner des cours aux élèves, comme changer le nombre des places dans les cours ou changer la liste de vœux des élèves par exemple.

2.1 Création de l'algorithme

2.1.1 Le CSP

Notre première idée, en utilisant le backtracking a donc été d'utiliser un CSP ou problème à satisfaction de contrainte. En effet c'est une méthode extrêmement pratique pour une solution à un problème donné en respectant certaines contraintes, cet algorithme porte très bien son nom. Il peut notamment être utilisé pour trouver la solution d'un sudoku, il est très rapide et très efficace, ce qui en fait un algorithme de choix. Il a d'ailleurs été assez simple de l'implémenter et de trouver une solution pour chaque instance que nous lui donnions. Le problème est que pour qu'une méthode soit exacte il faut avoir en sortie de celle-ci la meilleure solution possible, ce qui, et vous le comprendrez, n'était pas du tout le cas. En effet, elle se contentait de renvoyer la première solution qu'elle trouvait.

Après amélioration du CSP pour énumérer toutes les solutions possibles, avec tout de même une notion de borne empruntée au branch and bound, celui-ci ressemblait plus à un algorithme vorace qu'autre chose, et l'efficacité propre au CSP n'était plus qu'un vague souvenir, peinant à trouver une solution pour une instance contenant 10 élèves. Cet algorithme a donc été abandonné au profit d'une méthode un peu plus réfléchie.

2.1.2 Course Trading

La méthode que nous avons développée ici n'est à notre connaissance utilisé nulle part ailleurs, nous l'avons nommée Course Trading, par rapport à son fonctionnement que nous allons définir maintenant.

L'idée nous est venue en partant d'un postulat simple, il est impossible de tester toutes les solutions d'une instance d'un problème, celles-ci étant beaucoup trop nombreuses. Prenons un exemple, pour que vous compreniez bien qu'imaginer lister toutes les solutions d'une instance est un rêve beaucoup trop fou.

Prenons pour commencer une instance simple : 1 élève, 5 choix de cours et 3 cours attribués. Ici disons que le nombre de cours total est de 7 et que le nombre de places par cours est de 3, ces dernières valeurs ne seront toutefois pas utilisées dans le calcul du nombre de possibilités.

Dans la liste de cours ci-dessous, les cours en bleu sont ceux qui ont été choisis par un étudiant :

1	2	3	4	5	6	7

L'étudiant a donc choisi les cours 1, 3, 4, 5 et 7. Dans ces 5 cours choisis, 3 lui seront attribués. Les solutions possibles sont les suivantes (en rouges les cours attribués) :

1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7

Le nombre de solution possible pour un étudiant est $\binom{3}{5} = 10$.

Imaginons que nous ayons maintenant cinq élèves au lieu d'un. Nous avons 10 possibilités de solution par élève, ce qui nous donne le nombre de possibilités de solution suivant :

$$10 * 10 * 10 * 10 * 10 = 10^5$$

Ce nombre est extrêmement grand pour un si petit nombre d'élèves. Même si c'est un nombre à relativiser, dû notamment aux contraintes de nombre d'élèves par cours, nous n'avons tout de même que très peu de chance d'énumérer toutes les solutions dans notre problème.

Néanmoins, nous n'avons pas besoin de toutes les solutions mais seulement de a meilleure. Ce qui veut dire que le plus grand nombre d'élèves doit recevoir ses premiers choix.

Ainsi, après l'attribution systématique des premiers choix des élèves, dans la limite du nombre de places disponibles dans un cours, le nombre de cours restant se trouve réduit. Il varie même significativement selon le type d'instance.

Nous sommes donc passés d'un problème d'attribution de cours à un problème d'échange de cours afin de satisfaire le plus possible les élèves.

Pour résoudre ce problème nous avons utilisé un système de profondeur, plus la profondeur est grande lors de l'échange d'un cours, plus la satisfaction globale sera réduite. Par exemple pour une profondeur de 1, il n'y a qu'un seul élève qui se sera vu attribuer son premier cours de remplacement. Pour une profondeur de 2, soit un élève se voit attribuer son second cours de remplacement, soit 2 élèves ont leur premier cours de remplacement.

Pour expliquer grossièrement l'algorithme : pour chaque cours à attribuer, on vérifie si on peut le faire avec une profondeur de 1, en regardant son propre cours de remplacement ou en regardant si un élève du cours qui n'a pas été attribué peut lui-même prendre en cours son cours de remplacement à la place. Cela libère donc une place dans le pour que l'élève n'ayant pas pu entrer dans ce cours puisse finalement y accéder.

Si cela ne suffit pas à attribuer tous les cours, on continue avec une profondeur de 2, puis 3 et 4. Une profondeur de 4 est une limite que nous avons jugé pertinente, englobant un maximum de solutions mais ne cherchant pas indéfiniment si la résolution de l'instance est impossible.

Cette solution est assez stable, elle permet de trouver une solution assez rapidement, même dans des instances compliquées ou assez grandes.Malheureusement, même si la plupart du temps la solution proposée est la meilleure, il peut arriver, dans de rares cas (environ 1/100 selon nos tests) que l'on ne trouve pas la meilleure solution. Cela est dû à un blocage dans l'attribution des cours qui amène à l'impossibilité de retrouver la meilleure des solutions. Nous avons notamment observé ce blocage lorsque deux élèves se sont vu refuser l'accès au même cours : le premier se voit attribuer son cours avec une profondeur de 1 mais cela bloque l'autre, qui ne peut avoir son cours seulement avec une profondeur de 3, alors que dans la solution optimale le premier élève obtient son cours avec une profondeur de 2, ce qui permet au second d'obtenir le sien avec une profondeur de 1.

Notre solution n'étant pas exacte, nous avons dû quelque peu l'améliorer.

2.1.3 Améliorations

La première amélioration apportée, dans le but d'avoir toujours la solution optimale est de tester toutes les solutions différentes permettant d'accéder à l'attribution de tous les cours. Ainsi, avec cette amélioration, les problèmes d'exactitudes n'en sont plus, la solution retournée par l'algorithme est dans tous les cas la meilleure. Cela entraîne par contre un contre coup assez énorme au niveau du temps de calcul, très bien connu des algorithmes gloutons. Pour exemple, l'attribution de 5 cours dans une instance composée de 25 élèves, 7 cours, 5 choix de cours, 4 cours à attribuer et 17 places par cours nous a pris 156 227s, ce qui donne 43h20. Ce qui est complètement inacceptable, surtout pour une instance aussi petite.

La seconde amélioration a été implémentée spécialement pour les instances très compliquées, et par compliquées, nous entendons une instance avec très peu de places de cours inutilisées, ce qui veut dire que le nombre d'élèves(1) multiplié par le nombre de cours à attribuer(2) se rapproche fortement du nombre de cours(3) multiplié le nombre de places par cours(4). L'amélioration consiste à éliminer les instances mathématiquement impossibles à résoudre, c'est-à-dire s'il n'y a pas assez de places dans les cours pour le nombre d'élèves, ce qui comprends donc le fait que $(1) * (2) > (3) * (4)$ ou alors, cela peut arriver notamment si un ou plusieurs cours sont délaissés au profit d'autres, qu'il y ait plus de places libres dans les cours(5) que le surplus de places. Ainsi si $(5) > (3)*(4)-(1)*(2)$, l'instance est impossible à résoudre.

Le cas inverse, beaucoup plus courant en pratique, est le fait qu'il ne faille qu'une profondeur de 1 pour attribuer tous les cours. Dans ce cas-là, il suffit de calculer le pourcentage de satisfaction maximale que peut atteindre une solution d'instance, et de renvoyer la première solution ayant exactement ce pourcentage de satisfaction. Le calcul de la satisfaction maximale n'est pas très com-

pliqué, il suffit de calculer le pourcentage d'un élève s'étant vu attribuer son premier cours de secours, de le multiplier par le nombre de cours à attribuer, d'y ajouter 100 * le nombre d'élèves ayant tous leurs premiers cours et de diviser par le nombre d'élèves. Cela permet de s'arrêter à la première solution optimale, sans avoir à traverser toutes les solutions. C'est cette amélioration qui nous a par exemple permis de résoudre 100 instances de 200 élèves en 43 767s, ce qui équivaut à seulement un peu plus de 12h.

2.2 Limites de la méthode

Bien sûr, cette méthode a des limites, notamment en temps de calcul.

Avec une instance pouvant se résoudre exclusivement avec une profondeur de 1, le temps moyen de résolution d'une instance est de 20 min, mais il augmente significativement avec le nombre de cours à assigner. Il peut même être assez changeant avec entre deux instances ayant le même nombre de cours à assigner.

De plus, si l'instance devient très compliquée, c'est-à-dire qu'il faut accéder à la profondeur 2, s'il y a plus de 15 élèves, il ne vaut mieux pas utiliser cette méthode. La méthode est assez efficace pour les instances n'étant pas trop complexes, mais est difficilement utilisable autrement.

D'autres fonctionnalités pourraient aussi améliorer la méthode comme ajouter une place dans un cours ou même ouvrir un doublon d'un même cours. Mais cela nous sortirait alors des limites que nous nous sommes fixées pour la résolution exacte. Nous en reparlerons un peu plus en détail dans la partie résolution approchée.

3 Algorithme approché

L'un d'entre nous avait un peu d'expérience sur les technologies de recuit quantique. C'est pour cela que nous avons choisi de découvrir et de pousser plus loin nos connaissances sur ces technologies naissantes. L'ordinateur utilisé ici est un ordinateur hybride de la société D-Wave.

3.1 Ordinateur quantique à recuit simulé

3.1.1 D-wave

D-wave est une société Canadienne, fondée en 1999. Elle est la première entreprise à avoir commercialisé un calculateur quantique basé sur le recuit simulé.

Les ordinateurs utilisés pour l'algorithme qui va suivre sont les ordinateurs de la société D-Wave mis à disposition gratuitement. La société garantie 20 min d'utilisation de leurs ordinateurs hybrides et 1 min pour leurs ordinateurs quantiques. Vous trouverez en annexe comment utiliser leurs ordinateurs.

Pour donner un ordre de grandeur, l'ordinateur le plus récent mis à disposition possède 5619 qubits. Cela peut sembler beaucoup lorsque l'on compare aux ordinateur quantique tel que l'on entend parler par les grande sociétés tels que Google ou Amazon. Mais la technologie de ceux-ci est nettement différente et ne vise pas du tout le même genre de problèmes.

Nous avons principalement utilisé les ordinateurs hybrides qui allient les technologies classiques et quantique pour la résolution du problème.

3.1.2 Recuit simulé quantique

Le recuit simulé quantique, tout comme le recuit simulé classique tient son nom du processus de métallurgie qui permet de minimiser l'énergie du matériau. Son utilisation principale est pour les problèmes d'optimisation.

Ce qui le différencie du recuit simulé classique est l'utilisation de qubit (ou bits quantique) [1]. Comme son équivalent classique, le qubit peut être dans un état 0 ou 1, il se différencie par sa capacité à être dans un état de superposition de 0 et 1 en même temps. Lors du processus de recuit quantique, chaque qubit est forcé dans un état 0 ou 1.

De plus, il est possible de lier deux qubits entre eux afin qu'ils soient capables de s'influencer. L'utilisation d'un coupleur devient nécessaire pour coupler les photons. Les coupleurs ont en général deux champs d'actions : ils peuvent forcer deux qubits à être dans un même état ou bien les forcer à être dans des états opposés. En plus de ce phénomène, les coupleurs permettent de mettre d'intriquer les photons. Les photons intriqués peuvent être perçu comme un unique objet pouvant couvrir 4 états (combinaisons linéaires des photons seuls : 00, 01, 10, 11)

Le résultat du recuit est donc influencé par les différents biais et couplages appliqués sur l'ensemble des photons. C'est lors de la formulation du problème que l'ordinateur est agencé de telle sorte que le résultat du recuit soit un optimum local au problème. Il est donc nécessaire de modéliser mathématiquement le problème avant qu'il soit compréhensible par la machine quantique.

3.2 Modèle mathématique

Dans un premier temps, afin de bien appliquer le problème afin de créer un algorithme quantique, il est nécessaire de mettre à jour un modèle mathématique cohérent.

La fonction objectif se traduit de la façon suivante :

$$obj = min \sum_{i=0}^{ne} \sum_{j=0}^{nc} x_{ij} w_{ij}$$

où ne est le nombre d'étudiants et nc est le nombre de cours

où x_{ij} est une booléen indiquant si l'on a attribué à l'étudiant i le cours j

et w_{ij} représente l'importance du poids du choix de l'étudiant i pour le cours j.

Ainsi, un poids plus faible sera attribué pour un cours qu'un étudiant à sélectionné comme premier choix et un poids un peu plus élevé sera attribué à un cours choisi en 2e choix, ect... Cela permet de prioriser les premiers choix de l'étudiant puisque le solveur cherchera à minimiser la somme.

Si l'on s'arrêtait là, le solveur attribuerait tous les cours à tous les étudiants. Il est donc nécessaire d'intégrer les contraintes.

3.3 Contraintes

Dans ce paragraphe nous allons écrire mathématiquement les deux principales contraintes de notre problème.

3.3.1 Nombre de cours par étudiant

Chaque étudiant doit recevoir un nombre précis de cours. Cela se traduit de la façon suivante : Pour chaque étudiant; il est nécessaire de vérifier ceci :

$$\sum_{j=0}^{nbcours} x_{ij} = A$$

avec A = nombre de cours à attribuer.

3.3.2 Nombre de places par cours

Chaque cours ne doit pas avoir plus d'un certain nombre d'étudiants : Pour chaque cours il faut que le problème vérifie :

$$\sum_{i=0}^{nbetudiants} x_{ij} <= P$$

avec P = nombre de place maximum.

3.4 Relaxation des contraintes

Pour que les contraintes soient intégrables à l'ordinateur quantique, il est nécessaire de les relaxer dans la fonction objectif. Mathématiquement la relaxation se traduit par :

$$objectif\_fonction + \gamma(contrainte)$$

Ici, γ, est ce qu'on appelle un multiplicateur de Lagrange. C'est un scalaire qui permet de donner plus ou moins d'importance à la contrainte. La meilleure façon de déterminer sa valeur est de faire des tests empiriques et d'observer le résultat.

Afin de les intégrer à la fonction objectif, il est nécessaire de retravailler légèrement les contraintes.

3.4.1 Nombre de cours par étudiant

Pour obtenir une égalité stricte, il suffit de mettre le résultat au carré après avoir regroupé le tout du même côté :

$$\gamma(\sum_{j=0}^{nbcours} x_{ij} - A)^2$$

Cette équation se traduit par une parabole centrée en la valeur du nombre de cours à attribuer par étudiant.

3.4.2 Nombre de places par cours

L'équation de la section précédente concernant le nombre de place par cours peut s'écrire ainsi : Pour tous les cours :

$$\gamma(\sum_{i=0}^{nbetudiants} x_{ij} - P_i)$$

La fonction $y = x_{ij} - A$ correspond géométriquement à une droite de coefficient directeur 1 et d'ordonnée à l'origine la valeur de nombre de place maximum du cours. Néanmoins, après quelques tests nous avons décidé de passer également cette contrainte en égalité stricte afin d'obtenir de meilleurs résultats :

$$\gamma(\sum_{i=0}^{nbetudiants} x_{ij} - P_j)^2$$

3.4.3 Coefficient de relaxation

Puisqu'il y a deux types de contraintes, nous avons décidé que chaque gamma d'une même contrainte seraient équivalents. En effet, chacune des parties des contraintes ont la même importance sur le problème global. Ainsi, posons par la suite γ_0 le paramètre lagrangien de la contrainte du nombre de cours à attribuer par étudiant et γ_1 celui de la contrainte sur les places dans chaque cours.

3.5 Formule finale

Avec la fonction objectif et la relaxation des contraintes des sections précédentes :

$$f = min(\sum_{i=0}^{ne}\sum_{j=0}^{nc} x_{ij}w_{ij} + \gamma_0(\sum_{j=0}^{nc}\sum_{i=0}^{ne} x_{ij} - P_j)^2 + \gamma_1(\sum_{i=0}^{ne}\sum_{j=0}^{nc} x_{ij} - A_i)^2)$$

On peut utiliser cette formule pour développer les sommes :

$$((\sum_{i=0}^{n} x_i) - C)^2 = \sum_{i=0}^{n} {x_i}^2 + 2\sum_{i=0}^{n}\sum_{j>i}^{n} x_i x_j - 2C\sum_{i=0}^{n} x_i + C^2$$

Dans notre cas, on peut souligner que $x_{ij}^2 = x_{ij}$, puisque chaque x_{ij} est un booléen. On peut simplifier la formule précédente avec ce résultat (spécifique à notre cas) :

$$((\sum_{i=0}^{n} x_i) - C)^2 = 2\sum_{i=0}^{n}\sum_{j>i}^{n} x_i x_j - (2C-1)\sum_{i=0}^{n} x_i + C^2$$

Avec ceci on a :

$$f = min(\sum_{i}^{ne}\sum_{j}^{nc} x_{ij}w_{ij} + \gamma_0(2\sum_{j}^{nc}(\sum_{i=0}^{ne}\sum_{m>i}^{ne} x_{ij}x_{mj} - (2P_j - 1)\sum_{i=0}^{ne} x_{ij} + P_j^2) + \gamma_1(2\sum_{i}^{ne}(\sum_{j=0}^{ne}\sum_{m>j}^{ne} x_{ij}x_{im} - (2A_i - 1)\sum_{j=0}^{ne} x_{ij} + A_i^2)$$

3.6 Implémentation

Il ne reste plus qu'à entrer cette dernière formule dans notre calculateur par le biais d'une matrice de coefficients qui est diagonale supérieure.

3.6.1 Fonction objectif

$$\sum_{i}^{ne}\sum_{j}^{nc} x_{ij}w_{ij}$$

Dans un premier temps on entre les coefficients propres à la fonction objectif de départ :

```
for student_index in students:
    for course_index in courses:
        ind1 = getIndex(student_index, course_index, len(courses))
        Q[(ind1, ind1)] += w[(student_index, course_index)]
```

3.6.2 Nombre de cours par étudiant

$$\gamma_0(2\sum_{j}^{nc}(\sum_{i=0}^{ne}\sum_{m>i}^{ne} x_{ij}x_{mj} - (2P_j - 1)\sum_{i=0}^{ne} x_{ij} + P_j^2)$$

On peut faire de même pour la première contrainte :

```
for course_index in courses:
    for student_index in students:
        ind1 = getIndex(student_index, course_index, len(courses))
        Q[(ind1, ind1)] -= (2*(room[course_index]-1))*lagrange_nb_course


for course_index in courses:
    for student_index in range(len(students)):
        for student_index_2 in range(student_index, len(students)):
            ind1 = getIndex(student_index, course_index, len(courses))
            ind2 = getIndex(student_index_2, course_index, len(courses))
            Q[(ind1, ind2)] += 2*lagrange_nb_course
```

La fonction getIndex() permet de remettre à plat la 3D du problème dans une matrice 2D.

3.6.3 Nombre de places par cours

$$\gamma_1(2\sum_{i}^{ne}(\sum_{j=0}^{nc}\sum_{m>j}^{nc} x_{ij}x_{im} - (2A_i - 1)\sum_{j=0}^{ne} x_{ij} + A_i^2)$$

Enfin, pour la deuxième contrainte :

```
for student_index in students:
    for course_index in courses:
        ind1 = getIndex(student_index, course_index, len(courses))
        Q[(ind1, ind1)] -= (2*instance.students[0].nbr_courses -1)*lagrange_parameter_only_one

for student_index in students:
    for j in range(len(courses)):
        for m in range(j, len(courses)):
            ind1 = getIndex(student_index, j, len(courses))
            ind2 = getIndex(student_index, m, len(courses))
            Q[(ind1, ind2)] += 2*lagrange_parameter_only_one
```

3.6.4 Envoi à l'ordinateur quantique

L'implémentation liée à l'ordinateur quantique est maintenant très simple :

```
bqm = BinaryQuadraticModel.from_qubo(Q, offset=0)
sampler = LeapHybridSampler()
results = sampler.sample(bqm, label='TimeStabling')
smpl = results.first.sample
energy = results.first.energy
```

Ici, l'ordinateur quantique que nous utilisons est un ordinateur quantique hybride de la société D-wave : hybrid_binary_quadratic_model_version2.

La valeur d'energie récupérée correspond au résultat de notre fonction objectif de la section Formule finale.

L'un des avantages de cet algorithme est qu'il fournit une nouvelle solution souvent très bonne pour une même instance. Cela pourrait permettre à l'administration de sélectionner une solution préférable ou bien de mixer plusieurs solutions.

3.7 Améliorations

Afin de rendre l'algorithme encore meilleur, il aurait fallu améliorer le choix des paramètres Lagrangiens pour que les solutions obtenues dépassent moins les contraintes. Il a donc fallu ajouter un algorithme corrigeant les solutions pour qu'elles soient comparables aux résultats de la méthode exacte.

4 Résultats et Comparaisons

4.1 Solutions

Prenons un exemple de solution. Nous avons choisi un problème de dimension 54 (9 étudiants*6cours) particulièrement difficile car le nombre de places totales disponibles est également le nombre d'étudiants. Chaque étudiant choisit 3 parmi 6 et s'en voit attribuer 2.
Dans le cas de l'algorithme exact, la solution trouvée permet un taux de satisfaction de 95,5%. Bien évidemment, toutes les contraintes sont respectées.

```
Satisfaction Totale : 95.55555555555556%

Course 0:           Course 1:           Course 2:
[5,6,7]             [0,3,4]             [2,7,8]

Course 3:           Course 4:           Course 5:
[0,2,6]             [1,4,5]             [1,3,8]

Student 0:          Student 1:          Student 2:          Student 3:          Student 4:
[0,1,3,5]           [0,4,5,3]           [3,4,2,0]           [5,2,1,0]           [1,4,2,5]
[1,3]               [5,4]               [3,2]               [5,1]               [1,4]
90.0%               90.0%               90.0%               90.0%               100.0%

Student 5:          Student 6:          Student 7:          Student 8:
[0,4,3,1]           [0,3,5,1]           [0,2,5,4]           [5,2,0,3]
[0,4]               [3,0]               [2,0]               [5,2]
100.0%              100.0%              100.0%              100.0%
```

Mais dans le cas de l'algorithme approché, on a un exemple de mauvais respects des contraintes (un dépassement de 2 étudiants pour le projet 0). Dans certains cas, surtout pour des instances plus grandes, il peut être intéressant d'avoir des dépassements, surtout pour un taux de satisfaction plus élevé. En effet, notre problème s'y prête particulièrement bien puisque les cours sont souvent flexibles sur leur nombre de place. Cependant pour une instance aussi petite ce résultat est difficilement acceptable. Ainsi pour cette instance en particulier et en général pour des instances de faibles dimensions il est bien plus intéressant d'utiliser une méthode exacte.

```
Leap IDE /workspace/Projet_algo $ /usr/local/bin/python /workspace/Projet_algo/Approached/main.py
[-255000, -10000, 25050, 50000, 100000000]
Number of students : 9
Number of courses : 6

Choose Hybrid Computer (h) or Quantum Computer (q) :
h
You chose the hybrid D-wave computer
Sending to the d-wave computer...

Execution Time (s) : 1.0808300971984863
Embedding Time (s) : 1.0592327117919922
Recovery of the results...
Recovery time (s) : 5.187521696090698
Total time : 7.32806396484375
Size  54
Energy  -4269950.0

Le taux de satisfaction globale est de : 98.88888888888889
```

```
Pour le projet 0 (5/3)
   étudiant : 0 => Student 0
   étudiant : 1 => Student 1
   étudiant : 5 => Student 5
   étudiant : 6 => Student 6
   étudiant : 7 => Student 7
Pour le projet 1 (2/3)
   étudiant : 0 => Student 0
   étudiant : 4 => Student 4
Pour le projet 2 (3/3)
   étudiant : 3 => Student 3
   étudiant : 7 => Student 7
   étudiant : 8 => Student 8
Pour le projet 3 (3/3)
   étudiant : 2 => Student 2
   étudiant : 5 => Student 5
   étudiant : 6 => Student 6
Pour le projet 4 (3/3)
   étudiant : 1 => Student 1
   étudiant : 2 => Student 2
   étudiant : 4 => Student 4
Pour le projet 5 (2/3)
   étudiant : 3 => Student 3
   étudiant : 8 => Student 8
Le fichier results1648854796.5670533all.txt a été créé
au total :18

Si vous voulez voir le nombre d'étudiants pour chaque projet, appuyez sur A
Si vous voulez voir le nom des étudiants pour tous les projets, appuyez sur Z
Si vous voulez voir le nombre d'étudiants pour un projet particulier, appuyez sur E
Si vous voulez voir le taux de satisfaction, appuyez sur R
Si vous voulez quitter, appuyer sur Q
R
Pour l'option 1 : 100.0%
Pour l'option 2 : 88.88888888888889%
Pour l'option 3 : 11.111111111111111%
Pour l'option 4 : 0.0%
Pour l'option 5 : 0.0%
```

On peut également observer un résultat pour une plus grande instance (voir Annexe C) :

```
[-45000, -40000, -30000, -10000, 1000, 10000000]
Number of students : 50
Number of courses : 10

Choose Hybrid Computer (h) or Quantum Computer (q) :
h
You chose the hybrid D-wave computer
Sending to the d-wave computer...

Execution Time (s) : 0.9201710224151611
Embedding Time (s) : 1.0862214565277
Recovery of the results...
Recovery time (s) : 5.348242521286011
Total time : 7.354772090911865
Size  500
Energy  -185912000.0

Le taux de satisfaction globale est de : 98.2


Si vous voulez voir le nombre d'étudiants pour chaque cours, appuyez sur A
Si vous voulez voir le nom des étudiants pour tous les courss, appuyez sur Z
Si vous voulez voir le nombre d'étudiants pour un cours particulier, appuyez sur E
Si vous voulez voir le taux de satisfaction, appuyez sur R
Si vous voulez quitter, appuyer sur Q
R
Pour l'option 1 : 100.0%
Pour l'option 2 : 100.0%
Pour l'option 3 : 96.0%
Pour l'option 4 : 68.0%
Pour l'option 5 : 36.0%
Pour l'option 6 : 0.0%
```

Ce résultat respecte parfaitement les contraintes et garantit un taux de satisfaction de 98.4%.

4.2 Temps d'exécution

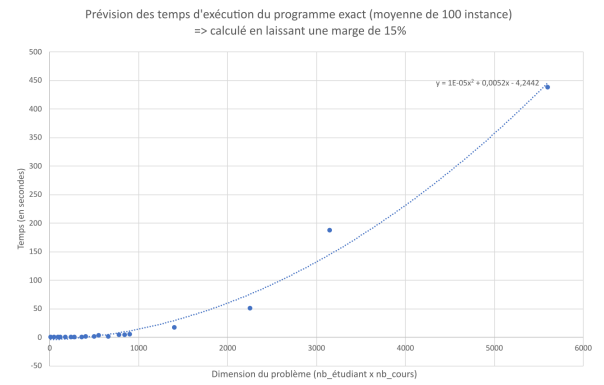
Dans les figures présentées ci-dessous, chaque point représente une moyenne du résultat de 100 fois la même instance pour l'algorithme exact et le résultat d'une fois la même instance pour la méthode approchée (malheureusement nous étions limités dans le temps d'utilisation des machines D-Wave). Ainsi, bien que ces résultats soient un bon indicateur du temps d'exécution, l'incertitude sur les données reste assez élevée. (principalement pour la méthode approchée)
Nous avons fait le choix de prendre des instances avec une marge de 15%, c'est à dire telles que :

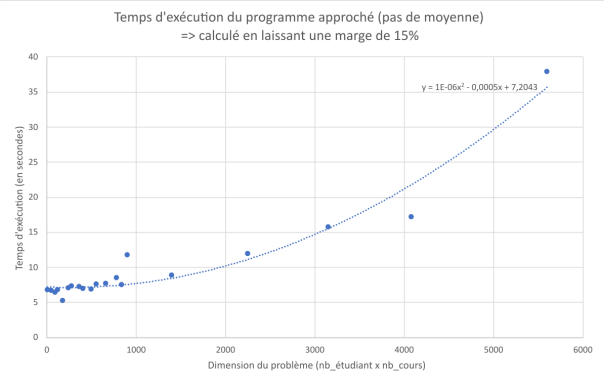
$$ne * nca \approx 1.15 * nc * npc$$

où ne est le nombre d'étudiant,
nca est le nombre de cours à attribuer,
nc est le nombre de cours total,
et npc est le nombre de place par cours

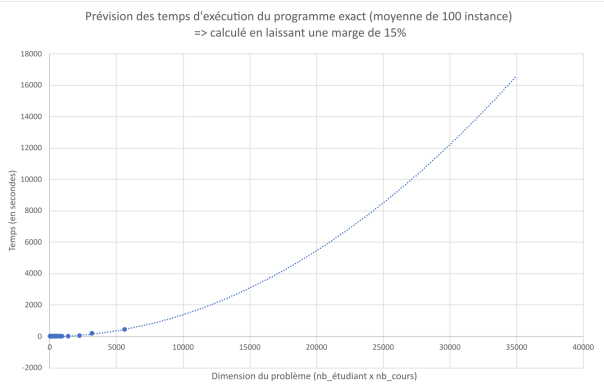
Les courbes suivantes représentent une approximation du temps de calcul en fonction de la dimension

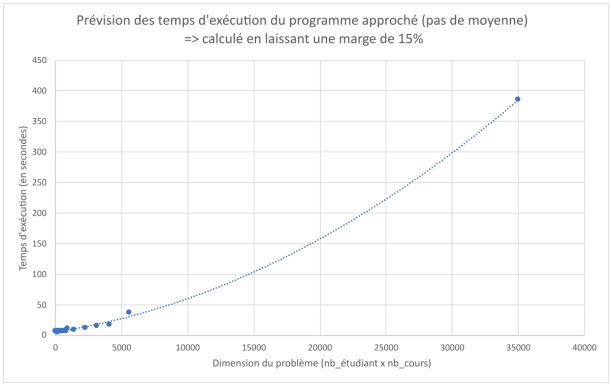
de l'instance pour les deux méthodes (la première est la courbe de la méthode exacte, la seconde est celle de la méthode approchée).



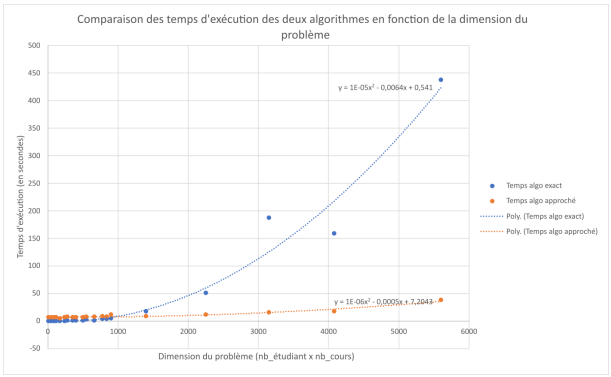


Les deux courbes progressent de manière polynomiales. Cependant, on peut observer que la méthode exacte croît beaucoup plus fort que la méthode approchée, on multiplie la croissance par 10 environ, en terme de temps de calcul.
Les deux courbes suivantes sont des extensions des courbes précédentes pour des dimensions beaucoup plus grandes :





En superposant les courbes, l'écart entre les deux résultats est bien plus visible. On voit notamment le point où le temps de la solution approchée devient définitivement plus faible que celui de la solution exacte. Ainsi, si l'on se base uniquement sur cette valeur, la solution approchée devient plus intéressante dès que les dimensions dépassent 1000 (autour de 100 étudiants pour 10 cours disponibles par exemple).



Si on fixe à une heure le maximum de l'acceptabilité du temps d'exécution du programme, on devrait s'arrêter à des instances de dimension légèrement supérieures à 2000 pour la méthode exacte, alors que pour la méthode approchée, on peut monter sur des instances beaucoup plus grandes (95000). Après cette valeur, l'IDE "kill" malheureusement tout seul le programme, l'algorithme pourrait continuer bien plus haut sans les contraintes forcées par l'IDE D-Wave.

Il est important de noter que même avec une instance ayant les mêmes paramètres, le temps d'exécution de la méthode exacte peut extrêmement varier en fonction du nombre de cours à attribuer grâce aux échanges. De plus, même avec ce même nombre d'attributions de cours, nous pouvons observer une grande disparité des résultats. L'allongement du temps d'exécution du programme

peut être impacté par l'ordre de recherche des solutions, si l'on décide de chercher toutes les solutions d'une branche alors qu'une attribution précédente est bloquante, le temps de calcul en sera forcément impacté. Tous ces résultats peuvent être observés en profondeur dans notre rendu de code (Projet_Algo).

Sur ces résultats il faut tout de même souligner qu'une parties des solutions dépassent légèrement les contraintes, d'autres trouvant l'optimalité. On compte au maximum une différence de 2 étudiants, ce qui est sur de grandes instances, est négligeable, comme expliqué plus tôt. Nous avons tout de même essayer de construire un algorithme de reconstruction de solution qui aurait permis de supprimer les dépassements de contraintes, mais par manque de temps il n'a pas été abouti.

5 Conclusion

Chacune des deux méthodes a des avantages et des inconvénients. Pour des instances de petite taille, nous conseillons l'algorithme exact qui permet une solution optimale en un temps acceptable. Mais pour de plus grandes instances, la méthode approchée convient mieux à notre problème, d'une part grâce à son temps d'exécution pour des instances moyennes et aussi par la possibilité d'obtention de plusieurs très bonnes solutions pour une même instance.

Bien que faible le non-respect des contraintes a été corrigé par un algorithme supplémentaire, celui-ci n'est pas forcément nécessaire au vu de la flexibilité des places dans les cours pour ce genre de problème. En effet, si l'on prend l'exemple de l'UQAC, il est en général acceptable d'ajouter un étudiant à un cours. Le principal inconvénient de cette méthode, comme énoncé plus haut est sa dépendance de la société et des ordinateurs D-Wave qui sont aujourd'hui en libre service mais rien ne garantis la pérennité de cette mise à disposition. Rien n'empêche non plus un développement positif de cette technologie qui améliorerait les performances de l'algorithme.

Enfin, quelques améliorations auraient été possibles si nous avions eu plus de temps. En effet, dans un premier temps une meilleure estimation des paramètres Lagrangiens auraient permis des résultats qui ne dépassent pas ou très peu les contraintes. On aurait également pu complexifier le problème en autorisant le doublement de cours dans le cas ou trop d'élèves ont choisi un cours comme premiers choix. Il aurait été également envisageable d'ajouter la répartition des professeurs à chaque cours.

Pour aller plus loin, une fois les algorithmes créés, il est possible d'ajouter de plus en plus de contraintes. Nous avions pensé à ajouter celles énoncées dans le paragraphe précédent. Comme le doublement des cours ou la répartition des professeurs dans les cours. Il aurait aussi été possible de créer la répartition hebdomadaire des cours dans un emploi du temps. Ce faisant, nous aurions dû ajouter d'autres contraintes comme la prise en compte qu'un élève ne puisse pas suivre deux cours à la même heure par exemple. Une autre contrainte aurait été de faire en sorte qu'un élève n'ait pas tous ses cours le même jour afin de répartir son temps de travail dans la semaine. Une dernière contrainte aurait été de prendre en compte certains aspects de la vie personnelle d'élèves, par exemple, pour les étudiants parents, il aurait été bien d'arranger leurs horaires de cours selon les horaires de garderie ou alors pour les étudiants sportifs ou musiciens de haut niveau de pouvoir éviter certains créneaux horaires. Ces contraintes s'éloignant un peu du problème initial, nous avons décidé de les garder pour un autre projet. Mais elles constituent une piste d'ouverture.

References

[1] What is quantum annealing. https://docs.dwavesys.com/docs/latest/c$_g s_2$.*html*.

[2] Masri Ayob, Ariff Malik, Salwani Abdullah, Abdul Hamdan, Graham Kendall, and Rong Qu. Solving a practical examination timetabling problem: A case study. pages 611–624, 08 2007.

[3] Troudi FATIHA. Résolution du problème de l'emploi du temps: Proposition d'un algorithme évolutionnaire multi objectif. Université Mentouri-Constantine, 2006.

[4] Travis S. Humble Kazuki Ikeda, Yuma Nakamura. Application of quantum annealing to nurse scheduling problem. *Scientific Reports*, 9:1–10, 2019.

[5] Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

[6] H. M. M. ten Eikelder and R. J. Willemen. Some complexity aspects of secondary school timetabling problems. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, pages 18–27, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Annexes

A Lancer l'algorithme exact

A.1 Créer une instance

Pour résoudre une instance, il suffit de la créer dans le fichier "Exact/main.py" disponible dans le code source du projet. Le code par défaut est celui-ci :

```
instance = Instance.Instance(9,6,4,2,3)
```

Dans cette instance, on réparti 9 étudiants dans 6 cours. Chaque étudiant choisi 4 des 6 cours dans l'ordre souhaité. 2 cours leurs seront affectés. Chaque cours ne peut contenir que 3 étudiants
Puis de créer un solver, par défaut, le code présent dans le fichier "main.py" est :

```
solver = Solver\_exact.Solver\_exact(copy.deepcopy(instance))
```

Et enfin de lancer la résolution :

```
solver.attribute\_courses(2)
```

Le programme a ainsi créé l'instance et l'a résoud en attribuant 2 cours pour chaque étudiant.

A.2 Lecture des résultats

Lors de la résolution de l'instance le fichier instance.txt est modifié, c'est l'instance précomplétée avec l'attribution des cours principaux pour tous les élèves.
Lecture d'un cours :

```
Course 6:
--> Le num ro du cours
None
--> Liste des   lves   ayant ce cours dans leurs voeux (inutilis  dans la r solution 2)
[1,3,5,8,10,11,16,21,27,30,37,41,43,44,50,51,52,54,55,56,57,67,68,72,73,75,76,83,86,87,89,90]

--> Les   lves   inscrits dans ce cours
```

Lecture d'un élève :

```
Student 124:            --> Le num ro de l'  lve
[3,9,4,6,7]             --> Les voeux de l'  lve
[3,6,7]                 --> La liste des cours non attribu s
[9,4]                   --> La liste des cours attribu s
100.0%                  --> La satisfaction
```

Si un cours de remplacement est choisi la satisfaction diminue, si le bon nombre de cours n'est pas encore attribué la satisfaction n'est pas impactée.
Il est possible d'enregistrer la solution retourner dans par la résolution d'instance :

```
solver._best_result[2].save_as_txt('exact\solution2.txt')
```

La lecture de l'instance se fait alors exactement comme décrit précedemment.
Les fichiers présents dans resultat sont les résultats des temps des tests effectués Le nom du fichier correspond à l'instance créée si le suffixe __2 apparait c'est que l'amélioration en fonction de la meilleure solution calculée mathématiquement a été ajoutée

```
nbr total : 100                                 --> nombre total d'instances r solues
temps total : 0.3081636428833008                --> temps total pour la r solution de ces
    instances
temps moyen total : 0.0030816364288330077       --> temps moyen pour la r solution de ces
    instances
-------------0-------------                      --> nombre de cours devant  tre  attribu s
    apr s la premi re attribution syst matique des meilleurs cours aux   lves   (visible
    dans instance.txt)
nbr : 56                                        --> nombre d'instance avec (ici) 0 cours
    devant  tre  attribu s
```

```
temps min : 0.0009965896606445312              --> temps minimum pour la r solution d'une
    des (ici) 56 instances
temps moyen : 0.0017386980272353183            --> temps moyen pour la r solution des
    instances
temps max : 0.005981922149658203               --> temps maximum pour la r solution d'une
    des (ici) 56 instances
```

B Lancer l'algorithme approché

Note : A chaque compilation, vous obtiendrez des répartitions différentes, c'est l'intérêt des ordinateurs quantique à annealing !

B.1 Etape 1

Créez un compte Leap sur le site : https://cloud.dwavesys.com/leap/signup/.
Celui-ci sera valide pendant 1 mois. Vous avez le droit à 1 minute sur les ordinateurs quantiques et 20 minutes sur les ordinateurs hybrides.

Après avoir vérifié votre adresse mail et vous être connecté, dirigez vous vers l'IDE Workspaces. Chargez un des exemples puis extrayez l'ensemble des fichiers source de l'archive de notre projet en les glissant dans l'IDE (Integrated Development Environment).

B.2 Etape 2

Vous pouvez maintenant exécuter le programme en vous plaçant dans le fichier /Aproached/main.py et en cliquant sur le bouton run situé en haut à droite de l'IDE

B.3 Etape 3

Choisissez en console l'ordinateur sur lequel s'effectue les calculs. Si vous choisissez l'ordinateur quantique,appuyez sur Q, si vous utilisez l'ordinateur hybride, entrez H.

Note : Nous vous conseillons d'utiliser l'ordinateur hybride. En effet, l'ordinateur quantique peut avoir des difficultés au moment de l'intégration.

B.4 Etape 4

Par la suite, plusieurs choix s'offrent à vous, comme pour le programme classique :

- En appuyant sur A, le nombre d'étudiants pour chaque cours s'affiche en console et un fichier commançant par "results" et terminant par "nbStudent.txt" est exporté.
- En appuyant sur Z, la répartition des étudiants pour chaque cours s'affiche en console et un fichier commançant par "results" et terminant par "all.txt" est exporté.
- En appuyant sur E, vous pouvez rechercher la répartition des étudiants pour un cours en particulier. Celui-ci doit être indiqué par son numéro.
- En appuyant sur Q, vous quitterez le programme.

Note : Les fichiers sont là aussi créé avec un identifiant unique (lié à l'heure) pour ne pas écraser les données.
Mot des développeurs : Pour toute question sur l'utilisation de ce programme, vous pouvez joindre notre groupe.

C Instance

```
Instance :
[
Course 0 :
    Students : []
    Students choice : [6, 8, 10, 13, 16, 20, 21, 27, 31, 32, 33, 36, 42, 43, 46, 47, 52, 53,
     54, 55, 56, 58, 61, 64, 65, 70, 77, 78, 82, 83, 84, 85, 86, 87, 91, 92, 103, 105, 110],
Course 1 :
    Students : []
    Students choice : [0, 11, 16, 24, 26, 29, 30, 31, 34, 37, 38, 40, 41, 45, 51, 55, 61,
    62, 64, 68, 72, 75, 76, 79, 92, 93, 95, 98, 99, 100, 102, 104, 105, 112, 117, 120, 121,
    124],
Course 2 :
    Students : []
    Students choice : [1, 3, 9, 10, 16, 22, 23, 24, 28, 39, 41, 43, 44, 46, 49, 53, 54, 56,
    58, 60, 61, 63, 64, 66, 67, 69, 77, 80, 97, 98, 100, 110, 111],
Course 3 :
    Students : []
    Students choice : [0, 7, 9, 12, 14, 15, 23, 26, 33, 34, 35, 44, 45, 48, 50, 52, 57, 59,
    60, 62, 77, 80, 83, 90, 91, 95, 99, 104, 106, 107, 108, 114, 115, 118, 121],
Course 4 :
    Students : []
    Students choice : [2, 4, 5, 8, 17, 18, 28, 32, 39, 42, 49, 50, 52, 59, 62, 66, 67, 69,
    72, 73, 98, 106, 110, 111, 116, 118, 122],
Course 5 :
    Students : []
    Students choice : [1, 6, 7, 14, 15, 17, 20, 24, 27, 29, 31, 34, 35, 38, 45, 47, 73, 75,
    76, 83, 89, 92, 93, 94, 96, 101, 102, 107, 112, 116, 119, 120, 123],
Course 6 :
    Students : []
    Students choice : [5, 9, 11, 18, 19, 22, 24, 25, 33, 39, 40, 46, 49, 57, 60, 62, 64, 65,
     66, 68, 69, 70, 76, 78, 85, 86, 87, 88, 93, 96, 97, 100, 102, 104, 108, 113, 115, 122,
    123],
Course 7 :
    Students : []
    Students choice : [0, 2, 4, 5, 13, 22, 28, 29, 32, 35, 36, 47, 56, 59, 61, 63, 73, 76,
    77, 82, 91, 93, 95, 98, 103, 111, 115, 117, 119, 124],
Course 8 :
    Students : []
    Students choice : [0, 1, 2, 3, 4, 8, 11, 13, 18, 25, 27, 29, 34, 39, 43, 47, 48, 58, 60,
     62, 65, 71, 74, 75, 79, 81, 82, 85, 86, 95, 96, 98, 101, 112, 114, 115, 119],
Course 9 :
    Students : []
    Students choice : [7, 12, 15, 16, 23, 27, 36, 37, 38, 39, 40, 43, 49, 56, 57, 72, 74,
    76, 79, 84, 86, 87, 90, 91, 99, 100, 103, 106, 109, 110, 112, 113, 118],
Course 10 :
    Students : []
    Students choice : [2, 3, 12, 19, 21, 25, 26, 30, 47, 53, 56, 57, 65, 67, 71, 73, 74, 75,
     78, 80, 81, 83, 84, 87, 94, 95, 97, 101, 102, 104, 107, 108, 109, 110, 114, 115, 119,
    122, 124],
Course 11 :
    Students : []
    Students choice : [1, 3, 4, 5, 6, 9, 10, 13, 14, 15, 17, 22, 23, 24, 29, 30, 36, 40, 45,
     46, 50, 54, 63, 65, 70, 73, 78, 79, 80, 84, 88, 89, 93, 96, 105, 106, 107, 108, 116,
    119, 121, 123],
Course 12 :
    Students : []
    Students choice : [6, 12, 14, 16, 18, 19, 21, 38, 42, 48, 51, 59, 63, 67, 68, 69, 70,
    71, 72, 74, 77, 82, 85, 86, 87, 88, 90, 94, 97, 99, 105, 109, 112, 113, 116, 120, 121,
    123, 124],
Course 13 :
    Students : []
    Students choice : [1, 8, 11, 17, 20, 25, 28, 34, 36, 37, 38, 41, 44, 51, 55, 58, 59, 64,
     67, 71, 72, 80, 81, 90, 92, 108, 111, 113, 114, 117, 120],
Course 14 :
    Students : []
    Students choice : [3, 11, 13, 15, 19, 20, 21, 23, 30, 31, 35, 37, 42, 43, 48, 51, 53,
```

```
    54, 68, 71, 84, 88, 101, 104, 106, 109, 111, 117, 118, 122],
Course 15 :
    Students : []
    Students choice : [2, 4, 6, 7, 8, 10, 12, 14, 20, 26, 30, 32, 33, 40, 45, 46, 50, 51,
    55, 61, 63, 66, 70, 81, 89, 90, 96, 97, 99, 102, 103, 107, 109, 113, 114, 117, 122,
    124],
Course 16 :
    Students : []
    Students choice : [0, 5, 7, 10, 19, 21, 25, 26, 27, 32, 33, 37, 41, 44, 49, 50, 52, 54,
    60, 69, 74, 79, 81, 82, 85, 88, 89, 91, 94, 103, 123],
Course 17 :
    Students : []
    Students choice : [9, 17, 18, 22, 28, 31, 35, 41, 42, 44, 48, 52, 53, 55, 57, 58, 66,
    68, 75, 78, 83, 89, 92, 94, 100, 101, 105, 116, 118, 120, 121]]
```