

IFT6521 - Projet Final

Mathieu Beaudoin

Session Hiver 2023

1. Description du problème

Deux fois par an, les étudiants à la maîtrise en finance mathématique et computationnelle de l'Université de Montréal doivent choisir quels cours entreprendre pour le trimestre à venir. Il s'agit d'une décision lourde de conséquences:

- d'un côté, on aimerait obtenir notre diplôme le plus vite possible, afin d'entrer rapidement sur le marché du travail (ou passer au doctorat, pour ceux dont c'est l'objectif), mais
- on ne veut pas se surmener au point d'avoir des mauvaises notes, voire d'échouer des cours (ce qui retarderait notre progrès). Il faut aussi dormir un peu à travers tout ça!

On a donc deux objectifs conflictuels. Il existe aussi d'autres considérations, comme le fait que tous les cours ne se donnent pas à chaque trimestre, et le fait que certains cours, une fois réussis, peuvent nous aider à mieux réussir d'autres cours qui partagent ou reprennent une partie de la matière ou bien des concepts similaires.

Muni de quelques suppositions et simplifications, j'ai choisi d'attaquer cet enjeu, qui revient essentiellement à un problème de décision séquentielle assorti de contraintes et de stochasticité.

1.1 Énoncé

1.1.1 Objectif

Minimiser le temps espéré (mesuré en nombre de mois) requis pour compléter la maîtrise. Pour simplifier, on ne considère **pas**:

- les *notes* espérées pour les cours (soit on réussit, soit on échoue)
- la dimension "santé mentale" (si on réussit tous nos cours rapidement mais qu'on tombe en *burnout*, c'est pas bien non plus... mais on va l'ignorer ici)
- les différences de difficulté entre les cours, au-delà du nombre de crédits associés

1.1.2 Décisions

Pour chaque trimestre, on doit choisir:

1. quels cours prendre;
2. quelle part de notre temps et nos énergies allouer à chacun de ceux qu'on a choisis.

1.1.3 Information disponible

Au moment de prendre notre décision, on sait:

- quels cours on a déjà réussis
- quels cours sont disponibles pendant la session à venir
- la charge que chaque cours représente (approximée par le nombre de crédits)

De façon implicite ou explicite (selon la formulation exacte du modèle), on sait aussi quels "boni" (qui augmentent la probabilité de réussir un cours pour un niveau donné de temps/effort) sont acquis pour un cours quelconque, ainsi que les boni supplémentaires pouvant être acquis en réussissant ledit cours.

1.1.4 Aléas

Le seul aspect aléatoire du problème est une variable succès/échec s'appliquant individuellement à chaque cours entrepris lors d'un trimestre quelconque. Cette probabilité est une fonction croissante de la proportion de notre temps alloué à un cours quelconque, divisé par la charge que représente ce cours.

Les paramètres fixes de cette distribution ont été choisis pour qu'ils reflètent ma propre situation: ayant un emploi à temps plein, deux cours par trimestre ça va, mais plus que ça, je ne penserais pas! On a donc une fonction qui donne une bonne chance de réussir si on distribue également notre temps sur 6-8 crédits, mais qui chute fortement au-delà de 10 crédits.

2. Modélisation

2.1 Notation

- N := nombre total de cours à réussir
- i := indice d'état, $i \in \{1, 2, \dots, 2^{N+1}\}$
- X_i := vecteur d'état $i \rightarrow \in \{0, 1\}^{N+1}$
 - $X_i = (1\{\text{session d'hiver}\}; I_i)$
 - $I_i := (1_{i,j}\{réussi\})_{j \in \{1, \dots, N\}} \rightarrow$ indicateurs de cours déjà réussis (on va souvent appeler cette quantité la "complétude" des cours)

Actions

- U_i := vecteur d'options disponibles à $i \rightarrow \in \{0, 1\}^N$
 - $U_{i,j} := 1\{\{\text{cours donné pendant le trimestre associé}\} \cap \{I_{i,j} = 0\}\} \in \{0, 1\}$
 - En français: l'intersection des cours qu'il est possible de prendre lors du trimestre associé à l'état i et des cours qu'on n'a pas déjà réussis
- u_i := vecteur de cours choisis à $i \rightarrow \in \{0, 1\}^N$
 - $u_{i,j} \in \{\{0\} \cup \{U_{i,j}\}\}$
- v_i := vecteur d'allocations $\rightarrow \in [0, 1]^N$
 - $v_{i,j} \in [0, u_{i,j}] \rightarrow$ prend des valeurs non-nulles seulement où $u_i = 1$
 - $|v_i| = \min\{1, |u_i|\} \rightarrow$ se somme à 1 (ou zéro s'il n'y a pas d'options disponibles)

Aléas

- $\omega_{i,j}$:= aléas, égal à 1 si le cours j est réussi à partir de l'état i , zéro sinon
 - b_j := charge (nombre de crédits) associée au cours j
 - γ_j := bonus de préalables pour la réussite du cours j
 - $p_{i,j} := P(\omega_{i,j} = 1 | v_{i,j}, b_j, \gamma_j) \rightarrow$ on simplifie la notation car on ne considérera jamais la probabilité inconditionnelle de réussite

Coûts

- $g(X_i)$:= coût immédiat de l'état i
- $J(X_i)$:= coût cumulé espéré de l'état i jusqu'à la fin

Approximation paramétrique

- $\phi(X_i)$:= transformation ("feature vector") de X_i
- \hat{r} := vecteur de coefficients d'approximation
- $\hat{J}(X_i)$:= approximation paramétrique de $J(X_i)$

2.2 Distribution de l'aléas

On modélise $\omega_{i,j}$ comme une Bernouilli, dont la densité est définie par la fonction suivante:

$$P(\omega_{i,j} = 1 | v_{i,j}, b_{i,j}, \gamma_{i,j}) = \frac{1}{1 + \exp\left(\alpha + \beta \frac{b_{i,j}}{\max\{\epsilon, v_{i,j}\}} + \gamma_{i,j}\right)}$$

où α et β sont des constantes arbitraires (j'ai pris $\alpha = -10$ et $\beta = 1$) et ϵ est un petit réel positif servant à garantir une certaine stabilité numérique.

2.3 Fonction de transition

Ici nous ferons une entorse à notre notation ci-haut, en remplaçant les indices i , qui réfèrent à des états définis de façon statique et qui ne sont pas ordonnés topologiquement, par des indices t , qui seront ordonnés dans le temps. Nous devons également séparer le traitement de la transition en deux: une première partie, déterministe, dénote simplement le passage d'un trimestre à un autre; la deuxième, stochastique, note l'évolution de la complétude des cours.

- $X_{t+1,0} = 1 - X_{t,0} \rightarrow$ on alterne simplement entre 0 et 1
- $X_{t+1,j} = X_{t,j} + \omega_{t,j-1} \cdot u_{t,j-1}$ pour $j \in \{2, \dots, N+1\} \rightarrow$ la complétude au temps $t+1$ est égale à celle du temps t , plus les cours réussis en t . Remarque: il peut sembler superflu de multiplier $\omega_{t,j-1}$ par $u_{t,j-1}$, car si $u_{t,j-1} = 0$, $\omega_{t,j-1}$ sera *presque* toujours zéro. Toutefois, la définition de l'aléas admet une probabilité minuscule (mais non-nulle) de réussir un cours sur lequel on aurait mis un temps strictement égal à zéro, une éventualité peu réaliste qu'on choisit d'ignorer.

2.4 Coûts

Le coût immédiat d'un état dépend de sa complétude et du trimestre concerné:

$$g(X_i) = \begin{cases} 0 & \text{si } |I_i| = N \\ 8 & \text{si } X_{i,1} = 0 \text{ et } |I_i| > 0 \\ 4 & \text{sinon} \end{cases}$$

On peut expliquer la logique comme suit:

- Si la maîtrise est terminée ($|I_i| = N$), on cesse d'accumuler des coûts;
- Chaque session coûte 4 mois, mais...
- Si on est rendus à l'automne et qu'on a déjà réussi des cours, ça sous-entend qu'on a commencé préalablement et qu'il s'est écoulé un été depuis le trimestre précédent, induisant une pénalité de 4 mois;
- Une exception à cette dernière règle est la cas où on vient tout juste de commencer, et donc $|I_i| = 0$. Notons qu'on va supposer, pour simplifier les choses, que $\{|I_i| = 0\}$ implique nécessairement qu'il s'agit d'un premier trimestre, alors que si on voulait être complètement rigoureux, on noterait qu'il est aussi possible qu'on ait commencé plus tôt *et réussi aucun cours*. Nous allons ignorer cette possibilité.

2.5 Équations de récurrence

$$v_i^* \leftarrow \arg \min_{v_i} \left\{ g(X_i) + E \left[J(f(X_i, v_i, \omega_i)) \right] \right\}$$

$$J^*(X_i) \leftarrow \min_{v_i} \left\{ g(X_i) + E \left[J(f(X_i, v_i, \omega_i)) \right] \right\}$$

où v_i^* et $J^*(X_i)$ représentent respectivement le vecteur de décisions optimales et le coût optimal pour l'état X_i , et $f(\cdot)$ est la fonction de transition définie à la section 2.3.

Les lecteurs attentifs auront peut-être noté que le vecteur u_i ne figure pas dans ces équations! Ceci découle du fait que l'information contenue dans u_i est entièrement contenue dans v_i ($u_i = 1 \Leftrightarrow v_i > 0$); aussi, on ne manipulera pas explicitement u_i sauf dans le cas de référence où nous utiliserons une politique aléatoire.

3. Algorithmes

3.1 Méthode exacte

Nous implémentons d'abord l'algorithme de chaînage-arrière. Notons ici que notre problème exigera une approximation pour un sous-problème précis que je décrirai sous peu, mais par ailleurs nous suivrons l'algorithme classique à la lettre.

3.1.1 Optimisation à une étape donnée

Avant d'expliquer l'algorithme de façon plus globale, définissons-en d'abord la partie qui servira à optimiser l'allocation v_i pour un état X_i quelconque, en supposant qu'on connaisse les $J(X)$ de chacun des états suivants possibles.

Cas où $|U_i| = 0$ (aucun choix possible)

Il n'y a aucune optimisation à faire dans ce cas: $v_i = 0$.

Cas où $|U_i| = 1$ (un seul choix possible)

Encore une fois, il n'y a pas d'optimisation à faire, on met simplement tout dans l'unique cours qui est disponible: $v_i = U_i$.

Cas où $|U_i| > 1$ (plusieurs choix possibles)

Ici c'est un peu plus compliqué, mais pourvu qu'on ait une fonction $h(v_i)$ qui puissent estimer (ou calculer exactement, ça ne change rien ici) $E[J(X'_i)|v_i]$ (où X'_i représente l'état suivant X_i), il est possible d'utiliser un solveur "off-the-shelf" (on en utilisera un provenant de la librairie *SciPy*) pour résoudre le problème suivant:

$$\begin{aligned} v_i^* &= \min_v h(v) \\ \text{tel que } & 0 \leq v \leq 1 \\ & 1^\top v = 1 \end{aligned}$$

On fournira $\frac{U_i}{|U_i|}$ comme valeur d'amorce (premier estimé de v); on supposera que la fonction h s'occupe de borner les v pour qu'ils soient conformes (le solveur de *SciPy* ne nous laisse pas borner directement si la borne inférieure est égale à la borne supérieure). Une fois que le solveur nous aura proposé un v_i^* , on s'assurera qu'il soit conforme en appliquant l'ajustement suivant (sinon on peut se retrouver avec de petites valeurs non-nulles aux endroits où $U_i = 0$):

$$v_i^* \leftarrow \frac{v_i^* \odot U_i}{|v_i^* \odot U_i|}$$

3.1.2 Algorithme - Chaînage arrière

1. $J \leftarrow 0$

2. Pour $k = \{N - 1, \dots, 0\}$:

- Pour $i \in \{i : |I_i| = k\} \cap \{i \leq 2^N\}$:
 - Prendre la paire d'indices $\mathcal{I} = (i, i + 2^N) = (i, j)$ (par construction, ils représentent les états qui ont exactement le même vecteur de complétude, mais à des trimestres différents)
 - $J^*(X_i), J^*(X_j), v_i^*, v_j^* \leftarrow$ optimisation par paires pour $X_{\mathcal{I}}$

3.1.3 Algorithme - Optimisation par paires

L'algorithme classique de chaînage arrière fonctionnerait pour notre application si on pouvait garantir qu'à chaque trimestre, on réussirait *au moins un cours*. Or, ce n'est pas le cas, ce qui nous met devant un problème d'identification: pour évaluer $J^*(X_i)$, on a besoin de $J^*(X_j)$, mais pour évaluer $J^*(X_j)$ on a besoin de $J^*(X_i)$!

On doit donc faire appel à une méthode approximative pour ce sous-aspect du problème. On parviendra à estimer les valeurs de chaque élément de la paire par un algorithme de balancement successif.

Soient $\tilde{J}_i^{(s)}$ et $\tilde{J}_j^{(s)}$ les valeurs estimées de J_i et J_j , respectivement, à l'itération s de l'algorithme. Notons M l'ensemble des indices tels que $|I_M| > k$ (puisque on ne pas "dé-réussir" un cours, tous les cas où $|I| < k$ peuvent être ignorés car leurs probabilités de transition à partir de $X_{\mathcal{I}}$ sont nulles). Puisqu'on a déjà estimé les J_M , leur espérance sachant v_i et v_j sera constante par rapport à nos estimations de $\tilde{J}_i^{(s)}$ et $\tilde{J}_j^{(s)}$.

Algorithme:

1. $\tilde{J}_i^{(s)} = \tilde{J}_j^{(s)} = 0$
2. Jusqu'à ce que la convergence soit atteinte, c'est-à-dire que $\{|\tilde{J}_i^{(s)} - \tilde{J}_i^{(s-1)}| < \epsilon\}$ et $\{|\tilde{J}_j^{(s)} - \tilde{J}_j^{(s-1)}| < \epsilon\}$, avec ϵ une petite constante positive:
 - $\tilde{J}_i^{(s)} \leftarrow \min_{v_i} \left\{ g(X_i) + E[J(X_M)|v_i] + P(\omega_i = 0|v_i)\tilde{J}_j^{(s-1)} \right\}$
 - $\tilde{J}_j^{(s)} \leftarrow \min_{v_j} \left\{ g(X_j) + E[J(X_M)|v_j] + P(\omega_j = 0|v_j)\tilde{J}_i^{(s)} \right\}$
3. On peut ensuite prendre les v_i^* et v_j^* comme les derniers vecteur v_i et v_j obtenus.

3.2 Méthode approximative

Nous allons comparer notre approche exacte à une approximation paramétrique itérative.

3.2.1 Représentation des états ("features")

Chaque vecteur d'état X_i est représenté par un vecteur de taille 7, contenant les données suivantes:

1. Le chiffre 1 ("*intercept*")
2. Le nombre de crédits restant à compléter: $(1 - I_i)^\top w$ (où w est le N -vecteur de nombres de crédits)
3. Ratio de complétude non-pondéré: $|I_i| \div N$
4. Indicateur de trimestre courant: 0 ou 1
5. Ratio de boni acquis: $I_i^\top B \div |B|$ (où B est le N -vecteur des sommes de rangées de la matrice de boni)
6. Nombre de cours atteignables lors du trimestre: $|U_i|$
7. Nombre de crédits atteignables lors du trimestre: $U_i^\top w$

3.2.2 Algorithme

Pour $k = N - 1, \dots, 0$:

1. $\mathcal{I}_k := \{i : |I_i| = k\} \rightarrow$ on travaille sur l'ensemble des i indiquant des états possédant le même niveau de complétude
2. $\hat{J}_{\mathcal{I}_k}^{(0)} \leftarrow g(X_{\mathcal{I}_k}^{(0)}) \rightarrow$ On initialise les \hat{J} du groupe par leur coûts immédiats d'état
3. Pour $s = 1, 2, \dots$ jusqu'à ce qu'on atteigne le critère de convergence $\|\hat{J}_{\mathcal{I}_k}^{(s)} - \hat{J}_{\mathcal{I}_k}^{(s-1)}\| < \epsilon$:
 - Pour chaque $i \in \mathcal{I}_k$:
 - $\mathcal{J}_i :=$ indices d'états atteignables en une étape à partir de X_i
 - $\Omega_{i \rightarrow \mathcal{J}_i} :=$ matrice $(|\mathcal{J}_i| \times N)$ de complétudes marginales associées à chaque état \mathcal{J}_i
 - * $\Omega_{i \rightarrow j} = I_j - I_i$
 - $\mathcal{V}_{i \rightarrow \mathcal{J}_i} :=$ matrice d'allocations, où $\mathcal{V}_{i \rightarrow j} = \frac{\Omega_{i \rightarrow j}}{\max\{1, |\Omega_{i \rightarrow j}|\}}$
 - * Heuristique: pour une sélection quelconque de cours, on accorde le même temps à chacun
 - $P_{i \rightarrow \mathcal{J}_i} :=$ matrice $(|\mathcal{J}_i| \times |\mathcal{J}_i|)$ de probabilités de transition (filtrée sur les transitions réalisables)
 - * chaque rangée correspond à un ensemble possible de choix de cours
 - * chaque colonne correspond aux probabilités conditionnelles de passer de X_i à un état $j \in \mathcal{J}_i$
 - Évaluer $\beta_i \leftarrow g(X_i) + \min(\langle P_{i \rightarrow \mathcal{J}_i}, \hat{J}_{\mathcal{J}_i} \rangle)$
 - Sauvegarder $(\phi(X_i), \beta_i)$
 - Entraîner un modèle linéaire (\hat{r}) sur les données $(\phi(X_{\mathcal{I}_k}), \beta_{\mathcal{I}_k})$
 - $\hat{J}_{\mathcal{I}_k}^{(s)} \leftarrow \langle \phi(X_{\mathcal{I}_k}), \hat{r} \rangle$

4. Résultats et analyse

Il n'a pas été très compliqué de produire différentes tailles de problèmes à partir seulement des quelques données originales dont je disposais:

- des instances "jouet" avec $N = 2$ ou 3 m'ont permis de développer la fonctionnalité en validant facilement que les opérations vectorielles/matricielles en particulier donnaient les résultats cherchés;
- une instance moyenne avec $N = 6$, qui a permis de valider rapidement que:
 - les deux méthodes étaient suffisamment bien implémentées pour ne pas planter ou donner de résultats absurdes sur des problèmes suffisamment complexes pour que cela soit loin d'être garanti
 - la méthode exacte performait mieux qu'une politique aléatoire (on l'espère, mais si on teste pas, on le sait pas..!)
 - la méthode approximative performait à un niveau similaire à celle de la méthode exacte
- une instance "grande" avec $N = 12$
 - ça a l'air de rien, mais avec $N = 12$, on a $2^{N+1} = 2^{13} = 8192$ états possibles, et 67 108 864 transitions potentielles à considérer!
 - il s'agit d'un problème suffisamment complexe pour que la méthode approximative ait un avantage majeur en termes de rapidité d'entraînement, comparativement à l'exacte

4.1 Structure des expériences

Au-delà de la pure validation de mes opérations vectorielles/matricielles avec les problèmes-jouets, le nerf de mon analyse repose sur la simulation. Le fonctionnement de celles-ci est simple. Quelques remarques avant de décrire l'algorithme de simulation:

- nous utiliserons ici des indices en "*superscript*" désignant des numéros d'étapes plutôt que des états précis
- nous allons noter par π la politique sujette à évaluation.
 - il s'agit essentiellement des v_i^* mentionnés dans la section (3);
 - nous ne nous étions pas attardés à parler explicitement de politique car notre emphase était sur l'estimation des coûts, les actions posées pour les évaluer n'étant que des moyens d'arriver à notre estimation
 - malgré ce silence, dans le code on enregistre quand même les décisions optimales lors de l'entraînement
 - ces politiques sont donc statiques, sachant qu'il existe un nombre fini d'états

4.1.1 Algorithme

1. Initialiser l'état de départ, avec $I^{(0)} = 0$
2. Initialiser le coût total, $J^{(0)} = 0$ (une entorse à la notation précédente: ici J désigne le coût cumulatif à partir du début de l'épisode)
3. Tant que $|I^{(t)}| < N$, pour $t = 1, 2, \dots$:
 - (a) On incrémente le coût: $J^{(t+1)} = J^{(t)} + g(X^{(t)})$ (voir la fonction à la section 2.4)
 - (b) $v^{(t)} \leftarrow \pi(X^{(t)}) \rightarrow$ (la politique renvoie l'action à prendre en fonction de l'état courant)
 - (c) On évalue $p_{i,j}$, les probabilités de réussir les cours entrepris
 - (d) Pour chaque j où $v_j^{(t)} > 0$, on pise $\omega_j^{(t)}$ avec la probabilité appropriée
 - (e) On met à jour l'état: $X^{(t+1)} \leftarrow (X_1^{(t)}; I^{(t)} + \omega^{(t)})$

Lorsqu'on a terminé, on enregistre la valeur terminale de $J^{(t)}$, puis on recommence l'expérience autant de fois qu'on le veut (j'ai fait 1000 simulations pour chaque test) pour pouvoir ensuite faire des statistiques sur les résultats.

4.2 Résultats

J'inclurai des graphiques à l'annexe A - j'ai de la difficulté à les placer comme je veux avec le logiciel d'édition de Markdown que j'utilise.

4.2.1 Petite instance ($N = 3$)

- Politique aléatoire contre chaînage-arrière:
 - Moyenne: 10.668 contre 4.26
 - Écart-type: 4.11 contre 1.53
- Politique chaînage-arrière contre approximation:
 - Moyenne: 4.344 contre 4.312
 - Écart-type: 1.737 contre 1.668

4.2.2 Moyenne instance ($N = 6$)

- Politique aléatoire contre chaînage-arrière:
 - Moyenne: 27.368 contre 16.856
 - Écart-type: 6.441 contre 2.568
- Politique chaînage-arrière contre approximation:
 - Moyenne: 16.94 contre 16.192
 - Écart-type: 2.695 contre 1.372
 - Durée de l'optimisation: 5.1 secondes contre 3.4

4.2.3 Grande instance ($N = 12$)

Il n'a pas encore été possible de compléter cette analyse: l'optimisation est trop longue! Toutefois, l'approximation a ici un avantage très clair en termes de rapidité:

- au bout d'environ 80 minutes de travail, l'approximation était rendue à $k = 6$
- après plus de 4 heures de travail, le chaînage-arrière était à $k = 7$ (les pannes d'électricités de la dernière semaine m'ont empêché de pouvoir laisser rouler cette optimisation plus longtemps)

À noter que les deux approches devraient bien se porter à la parallélisation, ce qui aurait le potentiel de les accélérer significativement. Toutefois, la parallélisation aurait ajouté une complexité considérable au développement de ce projet, et comme celui-ci ne servira jamais en "production", et que je ne m'attends pas à gagner de points supplémentaires pour un tel effort, j'ai choisi de ne pas m'aventurer par là!

4.3 Analyse

Ma plus grande surprise en effectuant ce travail a été que la performance de la méthode approximative ne fasse pas qu'approcher respectablement celle de la méthode exacte, mais qu'elle l'ait excédée (légèrement)! Deux hypothèses me viennent à l'esprit pour tenter d'expliquer ce résultat inattendu:

1. La partie approximative de l'algorithme autrement "exact" ne résoud peut-être pas adéquatement le problème d'identification abordé;
2. L'utilisation d'une fonction "boîte noire" pour trouver les v_i^* est également une source de risque. Lors du développement de mon code, je me suis parfois trouvé face à des résultats inusités pour lesquels j'ai du mettre en place des garde-fous explicites (ex: solutions non-admissibles). Je crois avoir somme toute bien géré cet aspect, notamment par l'utilisation de contraintes linéaires et la régularisation par la suite, mais il est entièrement possible qu'une subtilité-clé m'ait échappé.

J'ai aussi été agréablement surpris par la rapidité de la convergence de mon approximation: en général, 6-7 itérations semblaient suffire pour atteindre un critère de convergence somme toute assez serré!

Finalement, une de mes principales attentes s'est confirmée haut-la-main: que pour un problème non-linéaire complexe, une approximation sera beaucoup plus rapide à calculer qu'une optimisation exacte. Cet énoncé a un petit quelque chose de "thank you, Captain Obvious", mais cette analyse ne serait pas complète en passant ce fait sous silence!

5. Conclusion

5.1 Synthèse

Dans ce travail, j'ai exploré le problème du choix séquentiel de cours à la maîtrise. En posant quelques hypothèses sur la distribution probabiliste du succès ou de l'échec aux cours individuels, j'ai proposé deux manières distinctes d'optimiser les choix - l'une exacte, et l'autre approximative - si notre objectif est de terminer le programme le plus rapidement possible.

Les deux approches ont semblé donner de bien meilleurs résultats qu'une approche aléatoire. Toutefois, non seulement l'approche approximative a permis une optimisation bien plus rapide que l'approche exacte, elle semble avoir donné une politique plus optimale.

5.2 Choix d'algorithmes

5.2.1 Taxonomie du problème

La formulation que j'ai faite du problème possédait les caractéristiques suivantes:

- Horizon infini (en pratique il y a un temps limite mais j'ai ignoré cette contrainte)
- Espace d'états fini et discret
- Espace d'action mixte: discret et continu (mais plus continu que discret)
- Il s'agissait en quelque sorte d'un problème du plus court chemin, mais avec la particularité du parallélisme, c'est-à-dire la possibilité d'emprunter plus d'un chemin simultanément.

5.2.2 Justification des choix

La deuxième moitié du cours abordait notamment un enjeu que mon travail a mis en surbrillance: la malédiction de la dimensionalité. En effet, même avec un relativement petit nombre de cours, le nombre d'états explose, faisant croître de façon exponentielle le coût de la résolution exacte du problème avec chaque cours supplémentaire à considérer. Examinons tour à tour la pertinence des différents algorithmes qu'on aurait pu considérer:

- Estimation par simulation Monte Carlo: cette approche aurait souffert du fait que, pour le problème choisi, les résultats dépendent autant des choix effectués par l'agent. Avec autant d'influence sur le cours des événements, il aurait été absurde de simuler les réalisations comme des variables essentiellement exogènes.
- Simplification du problème ou approximation par heuristique: j'ai utilisé un peu ces idées en développant mon approximation paramétrique. Toutefois, je n'ai pas exploré davantage cette voie, puisque l'énoncé du projet mandatait expressément l'utilisation d'*au moins une* variante d'approximation paramétrique - et ajouter un algorithme d'approximation supplémentaire aurait considérablement augmenté la quantité de travail requis pour compléter ce projet.
- Modèle avec paramètres inconnus: ça aurait pu être un problème intéressant... si j'avais eu de vraies données de succès/échecs et d'efforts et charges simultanées! Cependant, puisque j'ai dû inventer la distribution de toutes pièces, il aurait été étrange de modéliser celle-ci avec un modèle bayésien (sachant que la simulation n'était pas une option viable - voir plus haut).
- Approximation par "rollout": c'est un peu ce que j'ai fait (ISL) pour obtenir des "données" d'entraînement pour mon approximation paramétrique. Intuitivement, une approche par "rollout", munie d'une bonne heuristique, devrait bien performer dans ce genre de problème où les décisions se ressemblent considérablement d'une étape à l'autre.
- Approximation séquentielle des Q-facteurs: il s'agit d'une approche relativement similaire à celle que j'ai adoptée. Toutefois, l'approche que j'ai utilisée a l'avantage d'être plus intuitive, puisqu'on demeure dans le monde des espérances absolues.

6. Instructions

6.1 Comment exécuter le code

Pour faire fonctionner le code, ouvrez-le dans un environnement avec support pour Jupyter Notebook en Python. La version de Python utilisée est la 3.9.13, mais le code devrait fonctionner avec n'importe quelle version 3.7+.

6.1.1 Dépendances non-incluses avec Python

- *Pandas*
- *NumPy*
- *matplotlib*
- *SciPy*
- *Scikit-Learn*
- *tqdm* (pas strictement nécessaire)

6.1.2 À savoir

- le fonctionnement du code dépend de deux fichiers de données en format .CSV, que j'inclus dans l'archive .ZIP sous le répertoire "data"
- le répertoire "data" contient aussi des fichiers .PKL permettant d'effectuer un "warm start" pour les algorithmes d'optimisation sur la grande instance, qui peut autrement prendre plusieurs heures à s'exécuter
- la 2e cellule du fichier de code permet de contrôler certaines constantes, dont les paramètres du modèle probabiliste et deux objets, nommés "TIMEOUT_AFTER_BC" et "TIMEOUT_AFTER_IA", qui imposent des limites de temps (en secondes) sur la durée de l'optimisation de la grande instance pour l'algorithme de chaînage-arrière et d'approximation paramétrique, respectivement. Si on leur assigne des valeurs de **None**, les optimisations ne seront pas limitées dans le temps.

6.2 Structure du code

1. Importation des librairies
2. Définition des constantes
3. Importation des données
4. Définition de la classe *DataContainer*, qui permet de manipuler plus facilement les données pour un problème d'une taille fixée
5. Définition de la distribution de probabilité de réussite de cours
6. Définition de la classe *ControlEnvironment*, qui facilite grandement l'interaction entre un modèle quelconque d'optimisation et les données d'un problème choisi
7. Définition d'une classe *Policy* générique et de la classe de politique aléatoire, *RandomPolicy*
8. Définition de la classe *ControlEpisode*, qui permet de simuler la performance d'une politique donnée
9. Optimisation exacte
 - (a) Définition d'une fonction qui donne un vecteur de probabilités de transitions, adapté aux dimensions du problème considéré
 - (b) Fonction d'optimisation à une étape donnée (voir 3.1.1)
 - (c) Définition de la classe *BackChainPolicy*, qui implémente l'algorithme de chaînage-arrière (3.1.2) et l'algorithme d'optimisation par paires (3.1.3).
 - (d) Comparaisons entre chaînage arrière et politique aléatoire (et définition d'une fonction qui permet de comparer facilement la performance de deux algorithmes d'optimisation)
10. Optimisation approximative - définition de la classe *IterativeApproximationPolicy*, qui implémente l'algorithme décrit en (3.2.2)
11. Comparaison entre les approches exactes et approximatives
 - (a) Petite instance
 - (b) Moyenne instances
 - (c) Optimisation des politiques pour la grande instance
 - (d) Comparaison des politiques pour la grande instance

7. Références

Je n'ai pas utilisé beaucoup de sources externes pour réaliser ce travail. En résumé, j'ai utilisé:

- Le site web de l'Université de Montréal pour constituer les données du problème;
- Les diapositives du cours, que j'ai consultées surtout pour m'aider avec les sections (3) et (5.2.2);
- De la documentation en ligne sur la librairie Python *itertools*
- ChatGPT pour chercher de l'inspiration initiale (voir l'échange en annexe). Finalement je ne me suis pas du tout servi de "ses" idées (mon "prompt" était un peu mal ciblé, et les réponses renvoyaient toutes à des algorithmes/problèmes déjà abordés dans le cours!)

7.1 Source des données

- Université de Montréal. (s.d.) *Structure du programme*. <https://admission.umontreal.ca/programmes/maitrise-en-finance-mathematique-et-computationnelle/structure-du-programme/>
 - **NB:** les données sur la disponibilité automne/hiver se trouvent dans les pages spécifiques à chaque cours, auxquelles on peut accéder directement à partir de cette page. Il me semble superflu de les énumérer une par une: voir simplement la référence que je viens de donner et c'est ultra-facile à trouver par la suite.

7.2 Diapositives du cours

- Frejinger, E. (23 janvier 2023). *Modèles déterministes et plus court chemin* [notes de cours]. Informatique et recherche opérationnelle, Université de Montréal. StudiUM. <https://studium.umontreal.ca/>
- Frejinger, E. (15 mars 2023). *Méthodes d'approximation dans l'espace des valeurs - Première et deuxième parties* [notes de cours]. Informatique et recherche opérationnelle, Université de Montréal. StudiUM. <https://studium.umontreal.ca/>
- Frejinger, E. (23 mars 2023). *Approximation paramétrique dans l'espace des valeurs - horizon fini* [notes de cours]. Informatique et recherche opérationnelle, Université de Montréal. StudiUM. <https://studium.umontreal.ca/>

7.3 Autres ressources

- Python Software Foundation. (12 avril 2023). *itertools — Functions creating iterators for efficient looping*. <https://docs.python.org/3/library/itertools.html>
- "Conversation" avec ChatGPT: <https://chat.openai.com/chat>

8. Auto-évaluation

1. Description du problème: A+ → le sujet est probablement assez original et je crois l'avoir énoncé très clairement
2. Modélisation: A+ → la notation, l'aléas, les transitions, les coûts et les équations de récurrence sont tous explicitement définis, et le raisonnement derrière est expliqué. La seule chose qui serait potentiellement à améliorer serait de trouver une formulation plus compacte de la fonction de transition, qui est en fait un agrégat de deux fonctions.
3. Algorithmes: A+ → corrects et décrits en détails (notation mathématique + explications des raisonnements), bien que l'approche exacte contienne une petite déviation (nécessaire, je crois, dans le contexte du problème) par rapport à l'algorithme vu en cours.
4. Résultats et analyse: A → Je crois avoir pris une approche très raisonnable (et équitable) pour comparer la performance des algorithmes. L'analyse aborde bien les principaux constats, y compris un résultat contre-intuitif pour lequel j'ai émis différentes explications plausibles. Cependant, la non-inclusion des résultats pour la grande instance représente une lacune.
5. Conclusion: B+ → cette partie manque un peu de viande, mais aborde quand même une variété de thèmes demandés, bien que de façon un peu sommaire.
6. Autre parties du rapport: A+ →
 - mes explications sur le code ne devraient pas laisser d'ambiguïté

- la bibliographie, bien que courte, reflète fidèlement mon utilisation de sources externes et respecte le style APA selon les explications fournies à l'adresse <https://bib.umontreal.ca/citer/styles-bibliographiques/apa?tab=5248896> (sauf peut-être l'entrée par rapport à ChatGPT).
7. Code: A+ → mon code manque un peu de commentaires (j'ai concentré mes explications dans le rapport écrit), mais autrement que ça, disons-le, il est de la bombe! Bien modularisé, extensible, raisonnablement efficient (il y aurait encore des améliorations à faire mais le temps me manque et ce n'est pas l'objet du cours), noms de variables/fonctions/classes intuitifs et/ou cohérents avec les explications du rapport (bien qu'en anglais, pour élargir l'audience qui pourra le comprendre lorsque je mettrai le tout sur mon GitHub), avec mêmes quelques "*bells and whistles*" comme la fonctionnalité de redémarrage "à chaud" des optimisations!
 8. Affiche: A- → elle est pas très belle, et je ne sais pas si on peut dire que j'ai bien résumé le projet. Disons qu'en étant limité à deux diapositives, on n'a pas de place pour inclure grand-chose d'autre qu'un survol hyper-sommaire..!

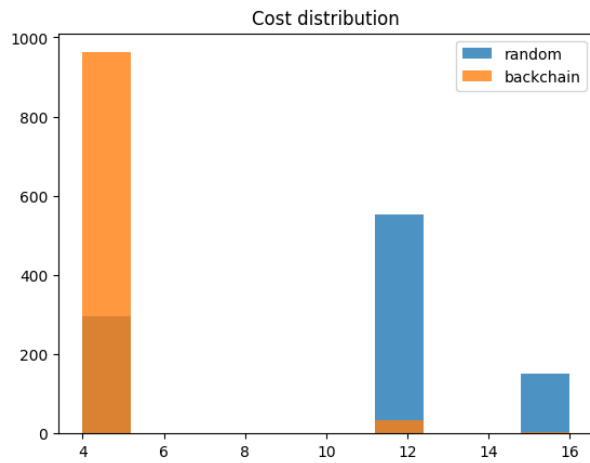
ANNEXE A

Performances simulées des différentes approches :

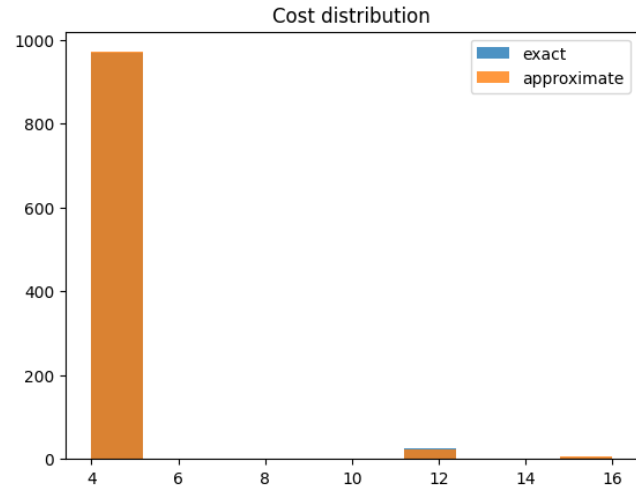
Histogrammes comparatifs

Petite instance

Politique aléatoire contre chaînage-arrière

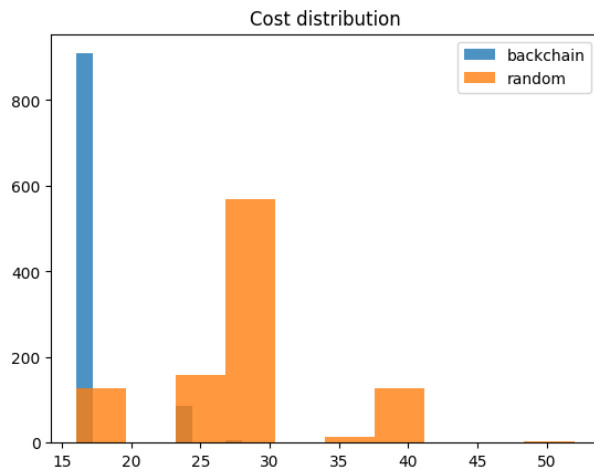


Politique exacte contre approximative

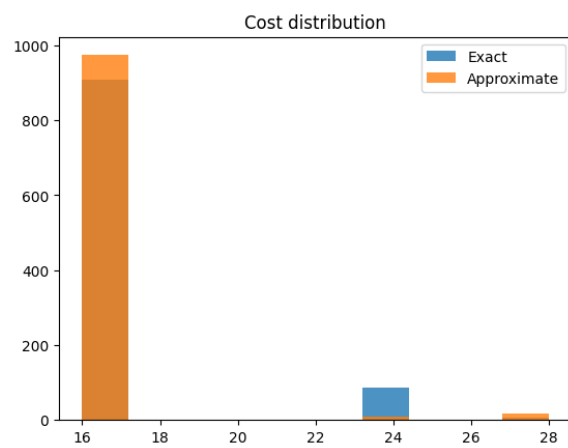


Moyenne instance

Politique aléatoire contre chaînage-arrière



Politique exacte contre approximative



ANNEXE B

Échange avec ChatGPT

(Phase de recherche de sujet pour le projet)

AR

Give me some examples of dynamic programming problems that can be approximated using the rollout algorithm.



Rollout algorithm is a simulation-based approach that can be used to approximate dynamic programming problems. Here are some examples of dynamic programming problems that can be approximated using the rollout algorithm:

1. Knapsack problem: In this problem, given a set of items with weights and values, the goal is to find the maximum value that can be obtained by selecting a subset of the items such that the total weight is less than or equal to a given capacity. Rollout algorithm can be used to simulate different item selections and approximate the optimal solution.
2. Shortest path problem: In this problem, given a weighted graph and a source and destination vertex, the goal is to find the shortest path from the source to the destination. Rollout algorithm can be used to simulate different paths and approximate the optimal solution.
3. Traveling salesman problem: In this problem, given a set of cities and their pairwise distances, the goal is to find the shortest possible route that visits each city exactly once and returns to the starting city. Rollout algorithm can be used to simulate different routes and approximate the optimal solution.
4. Stock market investment problem: In this problem, given a sequence of stock prices and a budget, the goal is to find the maximum profit that can be obtained by buying and selling stocks. Rollout algorithm can be used to simulate different buying and selling decisions and approximate the optimal solution.
5. Resource allocation problem: In this problem, given a set of tasks with resource requirements and a set of resources with capacities, the goal is to allocate resources to tasks in a way that maximizes some objective function. Rollout algorithm can be used to simulate different resource allocation strategies and approximate the optimal solution.