ALGORITHMIQUE AVANCÉE

Analyse *a posteriori* du projet en algorithmique pour le bloc 5 du DIU

Avant d'en donner une solution détaillée, il nous a semblé pertinent de proposer quelques reflexions sur la conception et le déroulement de ce projet.

Organisation de la formation

L'an dernier, en juin 2019, la centaine de stagiaires a été présente à l'Université de Bordeaux pour trois semaines à l'occasion des blocs 1,2 et 3. Cette année, les mêmes stagiaires ont suivi en distanciel pendant trois semaines de juin-juillet 2020 les blocs 4 et 5. Le bloc 4 occupait les deux premières semaines et le bloc 5 les deux dernières avec une semaine de chevauchement. Le projet d'algorithmique a occupé la dernière semaine pour éviter le chevauchement avec le bloc 4 et avoir suivi le cours du bloc 5 présentant le problème du voyageur de commerce. Les sources du projet autour de ce problème et les premiers énoncés ont été mis à disposition dans l'après-midi du vendredi précédent cette dernière semaine. Ce projet s'est alors déroulé jusqu'au vendredi suivant.

De la liberté puisqu'à l'impossible nul n'est tenu

Le programme officiel pour l'option NSI en première et terminale nous semble ambitieux puisqu'il semble couvrir le programme d'une L3 en informatique. C'est une forme de reconnaissance de la discipline informatique. Cependant il nous semble illusoire de pouvoir inculquer toutes ces notions dans le temps et les conditions de formation allouées. Cette impossibilité est finalement une source de liberté, nous laissant le choix sur ce que nous pouvons raisonnablement viser. Cela rentre en conflit avec les contraintes d'évaluation au BAC qui elles seront bien concrètes pour les professeurs et les lycéens.

Le distanciel nous a interdit une grande part de l'algorithmique qui se fait au tableau en TD, loin des machines. Une fois les principes algorithmiques acquis, on s'aperçoit fréquemment que l'écriture effective d'un programme nécessite de préciser les détails de programmation légitimement négligés dans un premier temps. De plus, cette incarnation d'algorithmes sous forme de programmes nécessite souvent lors de séance de TP sur machine des allers/retours précieux entre ces deux aspects algo/prog. Le distanciel interdit également ces séances de TP.

Dans ces conditions nous avons pris le parti de préparer un projet de programmation sur l'algorithmique autour du problème du voyageur de commerce. Ce fonctionnement par projet, suggéré par le programme de lycée, est fréquent dans le cursus d'études en informatique. Notre ambition était de montrer que sous cette forme, les stagiaires pouvaient être guidés, même en distanciel, dans une progression qui si possible devrait dépasser ce qu'ils auraient pu écrire intégralement par eux-même en partant de zéro.

Préparation du projet

Nous avons commencé par programmer intégralement une version du projet en notant au cours du développement les questions qui nous semblaient pertinentes. Celui-ci est la traduction un peu enrichie en python d'une version en C utilisée pour la moitié d'un enseignement sur un semestre de L3.

Lors du développement, nous avons notamment mis en place une batterie de tests. Des tests de correction pour vérifier que nos multiples algorithmes, traitant du même problème, donnaient bien toujours des solutions cohérentes sur un nombre significatif d'instances. Bien sûr, ces tests ne sont pas une preuve de correction mais permettent souvent de détecter les bugs.

Des tests de performance pour observer si des transformations d'algorithmes avaient une influence sur la vitesse de résolution ou l'utilisation de la mémoire. L'usage de ces tests de performance en python réserve quelques surprises, il est parfois difficile de comprendre ce qui se passe exactement sur la machine particulièrement avec ce langage. Ce n'est pas une critique du langage, simplement l'occasion pour nous de montrer notre incompréhension sur certains aspects entre la complexité théorique et la performance pratique.

Nous avons au maximum automatisé ces tests, les stagiaires n'ayant qu'à lancer une fonction déjà écrite pour effectuer le test. L'écriture des tests est souvent fastidieuse pour les débutants, il nous semble pertinent de les fournir clef en main dans un premier projet pour démontrer leur intérêt et motiver leur écriture dans les projets suivants. De plus, certains stagiaires nous ont confié que passer ces tests leur donnaient le sentiment d'être dans un escape game. Cette

« gamification » involontaire est peut-être un symptôme d'efficacité. Nous avons également utilisé les docstring et doctest pour certains tests, montrant l'intérêt de déclarer une fonction, puis d'écrire dans la docstring sa spécification ainsi que quelques cas d'usage. À cette occasion nous avons involontairement écrit un mauvais cas d'usage qui laissait passer une erreur classique de programmation, ce fut l'occasion de souligner l'importance d'un choix soigné des cas d'usage (et pas uniquement d'essayer de sauver la face par une pirouette).

Les batteries de tests étaient donc là pour compenser notre absence derrière le dos des stagiaires pendant les séances de TP. Leur écriture n'a été possible que parce que nous avions fait le choix d'imposer le squelette des fonctions à écrire. Le projet se présentait initialement aux stagiaires sous la forme d'un code à trou. Les trous apparaissaient sous la forme de assert(False) remplaçant des blocs de code *à l'intérieur* de fonctions. Même si le typage en python se fait entre adultes consentants, c'est à dire n'est vérifié qu'à l'exécution, nous avons utilisé le module typing permettant de déclarer le type de toutes les fonctions. Le respect de ce typage garantissait alors que les tests pourraient se dérouler correctement. C'est aussi une indication précieuse pour comprendre la spécification des fonctions. De plus nous avons choisi les « trous » les plus algorithmiques possibles, pour illustrer les notions vue dans le bloc 5. Ce format permet aussi de forcer les stagiaires à lire le code autour des trous et parfois encore plus loin.

Cette lecture du code nous semble un aspect très important de l'apprentissage. Une tendance des cours d'informatique est parfois de donner l'impression d'apprendre à lire un alphabet (syntaxe des boucles, affectations, ...) puis d'écrire un roman avant quelques années plus tard d'apprendre à lire de la poésie. Il est bien sûr prétentieux de comparer notre solution du projet à de la poésie, cependant sa lecture devrait être instructive : nous avons chercher à faire certains choix d'organisation qui s'incarnent concrètement, peuvent être discutables, mais sont visibles. Cela nous semble une bonne façon de s'interroger sur certains standards de codage (guidelines) plutôt que de les imposer d'office (surtout que le consensus n'existe pas, chacun pensant "sauf si on applique les miens"). Un risque de la lecture de code est de noyer le stagiaire lors de sa prise en main du projet. Les conditions de préparation compliquées de ce projet n'ont pas permis de baliser suffisamment ce projet pour éviter que certains stagiaires ne s'y perde. Une des ambitions de ce corrigé est d'ajouter ces balises pour traiter cette difficulté. De plus, ne demander que des modifications de fonctions permet d'éliminer toutes les questions d'écriture d'architecture logicielle qui sont peut-être un peu trop dures pour des lycéens (la lecture de ces architectures étant instructive en vue de leurs propres projets).

Déroulement du projet (point de vue des formateurs)

Le projet ne débutait officiellement que le lundi de la dernière semaine. Nous avons cependant mis à disposition les sources à trous et les deux premières feuilles (sur trois) dès le vendredi après-midi précédent. Nous avions ajouté une vidéo démontrant comment utiliser les tests et naviguer sommairement dans le code. Notre espoir était que les plus motivés essuient les plâtres de ce projet pour que nous soyons informés des éventuelles imperfections. Nous pouvions surveiller pendant le week-end leurs échanges sur la plateforme Slack (sans intervenir sauf cas de force majeure). Il y a environ cinq stagiaires qui ont traité la première feuille durant le week-end, relevant certaines incohérences mineures. Les plus avancés ont attaqué la deuxième feuille et certains l'avaient finie dès le lundi soir. Ce n'était pas le rythme attendu, le week-end pouvant aussi servir à se reposer, mais cela validait le fait que les feuilles pouvaient être faisables par certains.

Nous avions convenu de faire une séance BigBlueButton chaque jour de la semaine à 14h pour discuter du projet. Le lundi, à quatre formateurs en distanciel, pour tenir compte des différences, nous avons réparti 85 des 100 stagiaires en trois salles : ceux qui ne pouvaient pas installer et utiliser les sources initiales du projet, ceux qui étaient prêt à commencer le projet et ceux qui l'avaient déjà commencé. Cette dispersion n'a pas été rentable. Nous n'avons finalement pas eu à traiter de problème d'installation (ce qui ne signifie pas qu'il n'y en a pas eu). L'avalanche de questions dans le tchat pour ceux qui débutaient le projet était trop difficile à gérer pour un seul formateur. Les plus avancés, initialement laissé en autonomie, ont préféré repartir vers le Slack où ils avaient pris leurs habitudes et pouvaient plus facilement s'échanger du code et bénéficier des retours d'un des formateurs. Certains ont navigué entre les salles ne sachant pas où se passait ce qui pouvait les intéresser et ayant peur de le manquer.

Après discussion le lundi soir, nous avons trouvé notre rythme de croisière dès le mardi. Nous avons conservé en distanciel un formateur surveillant le Slack l'après-midi. Les trois autres formateurs se sont réunis dans une grande pièce aérée autour d'un micro commun, d'une tablette individuelle servant de tableau et d'ordinateurs. À 14h nous lançions l'enregistrement de l'unique salle BigBlueButton. La séance débutait par une présentation de ce qui était attendu sur l'après-midi. Cette présentation des algorithmes était censée compenser le manque de détails sur les feuilles de TP et correspondait à une présentation classique que l'on peut faire en début de séance en présentiel. Vue les difficultés rencontrées, nous avons ajouté la présentation de quelques solutions sous la forme d'un codage en direct. Cela permettait de discuter de certaines variantes et ceux qui étaient plus avancés pouvaient faire des commentaires en rapport avec leurs propres choix. Après une heure de présentation, nous laissions les stagiaires programmer, étant disponible pour des questions de debuggage jusqu'à 17h, heure à laquelle était prévu un debriefing de l'après-midi. Le partage d'écran et de micro nous a été précieux pour cette activité. Finalement, nous pensons que le partage de l'écran avec tous les stagiaires était un atout pour ces discussions de debuggage. Il y avait aussi quelques digressions par rapport au sujet mais les discussions nous semblaient toujours pertinentes. À 17h, jusqu'à 18h en général, une solution était proposée sous la forme d'un codage en direct de la solution.

L'organisation dans un bureau commun a permis de répartir plus efficacement les rôles. Une personne déroulait

le TP, les deux autres lisant les questions dans le tchat. Ils répondaient à certaines questions directement et parfois faisaient remonter la question pour que l'orateur principal s'en empare. Cela évite à l'orateur de se laisser distraire par les questions dont la lecture fragmenterait sa réflexion. De plus parfois, il a passé à la main à d'autres formateurs à la fois pour multiplier les avis et pouvoir se reposer. Avoir un retour critique sur la correction proposée par les autres formateurs est également très précieux. Les stagiaires se sont de plus en plus mêlés à ces discussions et c'est nous semble-t-il un intérêt de ce format. L'horaire tardif du debriefing ne permettait pas à tous les stagiaires de le suivre intégralement, nous espérons que les fréquentes demandes de mise à disposition rapide de ses débriefes sont le signe d'un intérêt des stagiaires. À la réécoute partielle, ces enregistrements sont un peu décousus, on a le sentiment qu'on pourrait aller plus vite à l'essentiel pour transmettre plus d'informations, mais prendre ce temps permet aussi de dédramatiser la difficulté.

Nous n'avons pas pu remplacer les conditions d'une formation en présentiel. Nous avons essayé de nous adapter. Ces après-midi sont épuisantes, gérer les outils de communication en distanciel, aussi bon soient ils, nécessite une couche supplémentaire de vigilance. Un autre énorme défaut est notre incapacité à suivre l'ensemble des stagiaires : seuls ceux qui se manifestent sont visibles. En particulier, nous perdons la possibilité comme dans un TP en présentiel d'observer ce que font les stagiaires et d'aller directement discuter avec une personne qui n'aurait pas initié de contact. Lorsque certains ont exprimé un « désespoir » nous nous sommes sentis démunis pour réagir de façon constructive. Les outils mettant au regard de tous les questions induisent une censure. Disperser trop les questions ne nous semble pas gérable d'un point de vue formateur car nous avons déjà dû nous organiser à plusieurs pour traiter les questions publiques. Le fait de mettre à disposition toutes les réponses nous semble être un maigre avantage de cette situation. Un autre avantage est de reconnaître qu'une solidarité entre stagiaires à pu apparaître dans les discussions sur le Slack certains y trouvant même l'occasion de faire de la pédagogie sur les autres stagiaires en observant leurs difficultés et les comparant aux leurs. Ces bénéfices pour les participants ne doivent pas masquer que certains lecteurs du Slack ont pu être intimidés ou démoralisés par ces discussions de gens qui "y arrivaient". Le distanciel est aussi la perte d'une garantie sur l'assiduité des stagiaires : compte-tenu des circonstances ils ont parfois été impliqués dans des réunions ou enseignements (sûrement légitimes) dans leurs lycées respectifs. Nous ne parlons pas non plus de l'intérêt des temps de pause et de leurs discussions pair à pair. Nous avons brièvement envisagé de les susciter en proposant des salles de discussions mais nous n'avons pas été très explicites et les stagiaires ne se sont pas emparés de cette possibilité.

Pour la centaine de stagiaires inscrits à cette semaine, les enregistrements sur BigBlueButton indiquent respectivement 85,85,56 et 53 participants. Nous ne savons pas si ce décompte, après exclusion des 4 formateurs est réaliste, ne sachant pas comment sont comptées les ré-entrées de participants. La participation à ces sessions était conseillée, mais pas obligatoire, certains pouvant travailler en autonomie. De plus, il était possible de visionner les enregistrements à des heures plus appropriées. Par les échanges explicites avec les stagiaires nous estimons qu'au moins 15 stagiaires ont fait la majeure partie des deux premières feuilles de ce TP ce qui était notre objectif et plusieurs ont traité les trois feuilles. Nous savons qu'il y a eu des abandons devant la difficulté durant la semaine avec quelques abandons explicites. Nous avons eu aussi connaissance de personnes ayant fait ces feuilles sans se manifester durant la formation. Ne pas savoir où en sont 80% des stagiaires est une difficulté de la formation en distanciel. Nous avons fait quelques sondages pendant les sessions, mais comme nous ne voulions pas ajouter de l'évaluation obligatoire à cette formation dans des conditions suffisamment difficiles, nous n'en savons pas plus. Nous lancerons probablement un sondage si possible anonyme pour essayer d'en savoir plus à la rentrée.

Ce texte est une introduction à une version plus détaillée des trois feuilles de TP, fort de notre expérience de la semaine et de temps de (post-)préparation, dans l'espoir que certains pourront les faire en autonomie à un rythme moins soutenu.

Contenu des feuilles

Pour un ensemble de points dans le plan, $(x_i)_i$ représente les positions de n villes, le problème du voyageur de commerce consiste à trouver une permutation σ des points $(x_\sigma(i))_i$ minimisant la longueur du trajet total pour visiter toutes les villes en partant et arrivant de la même ville.

Feuille 1

L'espace des solutions est donc l'ensemble des $n!$ permutations. La taille de cet espace est donc source d'une explosion combinatoire indiquant une difficulté pour l'approche naïve consistant à énumérer toutes les solutions possibles pour n'en retenir qu'une des plus courtes. Le thème de la feuille 1 est d'expérimenter cette explosion combinatoire puis de montrer que des raisonnements permettent de restreindre l'exploration à un sous-ensemble de l'espace des solutions. La conclusion espérée est que l'explosion combinatoire de l'espace des solutions n'implique pas la difficulté algorithmique parce que justement les algorithmes exploitent des arguments permettant de limiter l'exploration de cet espace.

Nous partons de l'exploration naïve correspondant à une énumération explicite (fournie) de l'ensemble des permutations avec un espace mémoire de l'ordre de $\mathcal{O}(n \times n!)$ qui s'observe expérimentalement. Elle permet de mettre au point la recherche d'un minimum sur cet espace.

Nous fournissons alors un autre mécanisme de l'exploration de l'espace des solutions basée sur deux primitives : l'une fournissant la première solution selon un ordre donné (lexicographique) et l'autre fournissant l'élément suivant

s'il existe en travaillant dans un espace mémoire de taille $\mathcal{O}(n)$. Ces deux primitives, algorithmiquement subtiles, sont fournies. La réécriture de la boucle for en une boucle while et la sauvegarde de la solution optimale sont déjà un exercice.

Ensuite, nous proposons deux stratégies pour limiter l'exploration de l'espace des solutions.

La première consiste à détecter de mauvais préfixes dans les solutions partielles : si après avoir visité 5 des 30 villes, mon trajet est déjà plus long que ma meilleure solution courante sur les 30, je peux me dispenser de visiter toutes les permutations débutant par ces 5 villes dans cet ordre. Il faut donc à la fois détecter ces mauvais préfixes au plus tôt puis savoir court-circuiter un préfixe donné dans le parcours des permutations. Le court-circuit dans le parcours des permutations est l'occasion d'observer que pour des propriétés spécifiques à l'algorithme de parcours, ce qui aurait dû être un tri en $\mathcal{O}(n \log n)$ d'un sous-tableau devient un simple renversement de sous-tableau en $\mathcal{O}(n)$. Nous améliorons dans deux variantes la détection des mauvais préfixes en prenant en compte grâce à l'inégalité triangulaire d'abord le trajet de retour entre la 5ème ville et la première, puis entre la 5ème, une des 25 restantes et la première.

La seconde stratégie consiste à prendre en compte les symétries dans l'espace des solutions. Pour une solution donnée sur n villes, il y a $2n$ solutions équivalentes modulo le choix de la ville de départ (n choix) et le sens du parcours (en général 2 choix). Il faut prendre soin de remarquer que toutes les solutions optimales ne sont en général pas équivalentes par ces symétries, même si dans le modèle aléatoire tirant des points au hasard c'est presque toujours le cas. La solution retenue consiste à supposer que la permutation σ sur $\{0, 1, \dots n-1\}$ satisfait les contraintes $\sigma(1) = 1$ et $\sigma(0) < \sigma(2)$. Ce choix permet d'observer que cette contrainte sur les permutations peut s'exprimer aussi comme une contrainte sur les préfixes et donc revisite les raccourcis sur les préfixes utilisés juste avant. Souvent, ce type d'optimisation n'est pas la dernière, cependant il nous semble plus sage de commencer à écrire les versions les plus simples sans ajouter des conditions de bords particulières qui peuvent poser des problèmes initialement inutiles.

À la fin de cette feuille, nous espérons avoir convaincu les stagiaires que l'exploration de tout l'espace des solutions n'est pas nécessaire et que cela permet des gains significatifs en performance. Nous espérons aussi fournir une première illustration de la progression partant d'un algorithme simple, correct mais peu efficace vers des algorithmes plus sophistiqués obtenu par des améliorations successives. Cela aurait dû être l'occasion de signaler qu'écrire directement la dernière solution est en général quasi-impossible pour les programmeurs de la rue (que nous sommes).

Digression Au cours des discussions sur cette feuille est apparue cette question que nous souhaitons mettre en exergue : Combien de boucle for sont cachées dans cette instruction

```
imax = p[j:].index(max(p[j:]))
```

qui est une étape d'un tri par sélection repérant l'indice imax d'un élément maximal du sous-tableau p[j :]. L'occasion de discuter du confort d'expressivité du langage python au prix d'une complexité cachée.

Feuille 2

Pour faire simple, l'objectif de cette feuille est de présenter la programmation dynamique comme une transformation à partir d'une résolution récursive en utilisant la mémoïsation.

Nous débutons par un exemple simple de mémoïsation partant de l'observation que la solution exhaustive sur 8 villes implique de l'ordre de 300 000 calculs de distances parmi les $\frac{8\times7}{2} = 28$ distances possibles. Les stagiaires utilisent explicitement un dictionnaire global pour remplacer les répétitions de ces calculs de distances par une mémorisation et un accès mémoire. Nous leur montrons également que cette méthode est automatisé par le décorateur `@lru_cache` du module functools.

La difficile question suivante consiste à implémenter une variante de la récurrence mathématique vue en cours caractérisant une solution optimale

$$D(S,t) = \min_{tt \in S-\{t\}} (D(S-\{t\}, tt) + \mathsf{distance}(tt, t))$$

où $D(S,t)$ est la longueur minimale d'une tournée débutant en la dernière ville $n-1$ puis visitant exactement une fois toutes les villes du sous-ensemble de villes S en terminant par la ville $t \in S$. La discussion porte donc sur le choix de l'avant-dernière ville tt pour identifier récursivement un plus petit sous-problème. Nous remarquons notamment que $D(\{0, \dots, n-1\}, n-1)$ donne une solution optimale au problème du voyageur de commerce. Sous cette forme, nous observons que la description est récursive et nous incitons les stagiaires à coller au plus prêt à cette récurrence dans la première solution en leur fournissant la sous-fonction récursive `rec_solve(S,t)` collant au plus prêt à $D(t,s)$ (avec la solution en plus) et en allant jusqu'à utiliser les compréhensions de listes pour calculer $S-\{t\}$ et le min sur l'ensemble des possibilités. Le but est d'utiliser le confort maximal de la syntaxe python pour traduire la description mathématique. Une fois cette solution correcte, nous n'allons plus qu'effectuer des transformations de ce programme pour lequel il est plus facile de préserver la correction plutôt que de s'en assurer en plus de tous les autres détails. Nous constatons que la mémoïsation automatique sur la fonction `rec_solve` ne fonctionne pas car l'argument S étant une liste, il n'est pas hashable. C'est une raison mineure car nous pourrions utiliser des tuples, mais c'est l'occasion de présenter ce prérequis à l'utilisation de `@lru_cache`.

La première transformation consiste à transformer l'ensemble S :List[int] décrit par une liste d'entiers par un unique entier s_int :int dont le codage en binaire code les éléments de l'ensemble. Par exemple, $42 = 2^5 + 2^3 + 2^1$ code pour l'ensemble $[1, 3, 5]$. Après la programmation des deux conversions entre ces deux représentations, nous les utilisons pour changer le type de l'argument S :List[int] de `rec_solve` en `s_int:int` en observant qu'il peut suffire pour travailler au moment des passages d'arguments afin de modifier le moins possible le programme précédent de résolution. Nous observons alors que le décorateur `@lru_cache` fonctionne au prix d'une augmentation importante de la mémoire utilisée. Cela annonce d'ailleurs ce que nous allons explicitement implémenter dans la suite de l'énoncé.

La seconde transformation est optionnelle et l'usage a montré qu'elle aurait plus sagement dû se placer à la toute fin de la feuille. Elle consiste à faire des manipulations de bits pour mettre à jour l'entier `s_int` représentant les ensembles de villes en évitant totalement de passer par les conversions en liste d'entiers. Il est intéressant de montrer l'existence des opérateurs <<,&, `int.bit_length`...

La troisième transformation est le début de la mémoïsation. Elle consiste à utiliser un dictionnaire $D = \{\}$ initialement vide, dont les clefs `(s_int,t)` indiquerons la mémorisation du résultat de `rec_solve(s_int,t)` lors de son premier appel. Elle permet d'ajouter directement en début de fonction le test de l'appartenance du résultat à D ce qui évite de répéter le calcul.

La quatrième transformation consiste à éliminer les appels récursifs en utilisant un ordre bien choisi sur les valeurs de `s_int` bien qu'il soit simplement

```
for s_int in range(1,2**n):
```

et conduise à s'interroger sur le codage astucieux des sous-ensembles. Il faut en particulier bien réaliser que si u est un sous ensemble de s alors u_int $\leqslant$ s_int donc que les appels récursifs de la récurrence sont compatibles avec cet ordre.

La cinquième transformation remplace le dictionnaire D par une paire de tableaux à double entrée `numpy.array` de réels Dlength et d'entiers Dtt alloué dès le début de l'algorithme.

La dernière transformation consiste à remarquer que pour qu'un ensemble S puisse être utilisé dans la récurrence, la ville $n-1$ doit nécessairement être la dernière visitée. Ainsi tous les ensembles S contenant $n-1$, à l'exception de l'ensemble entier, peuvent être oubliés divisant par deux le nombre de cas à traiter. Cette remarque induit une programmation légèrement différente du cas de cet unique ensemble contenant $n-1$ que nous n'abordons qu'à cette dernière question plutôt que d'avoir été perturbé lors de toutes les transformations précédentes.

Notre impression est que cette présentation très graduelle des difficultés sous forme de transformation de programme est très pertinente. Elle permet d'une part de lisser la difficulté, d'autre part, d'être plus réaliste sur la pratique de l'écriture d'un programme. Un regret est que les performances progressives ne sont pas au rendez-vous puisqu'il est finalement difficile d'améliorer la memoïsation automatique en la remplaçant par une mémoïsation explicite.

Feuille 3

Cette feuille présentait la notion d'heuristique et d'algorithme d'approximation. Comme elle a juste été évoquée lors de la dernière session nous la détaillons moins.

La notion d'heuristique est présentée sous la forme suivante : je tire 10 000 permutations au hasard et je garde la meilleure rencontrée comme résultat. Elle permet de donner finalement à peu de frais une méthode pouvant donner la solution optimale bien qu'elle ne donne aucune garantie. Par exemple, on peut imaginer avoir tirer 10 000 fois la pire tournée pour le voyageur. Les tests fournissent une étude statistique de la qualité des solutions fournie par l'heuristique. Comprendre qu'il est dur de transformer ces observations statistiques en preuve probabiliste fait partie de l'expérience.

Nous proposons, un peu artificiellement ici, d'utiliser le résultat générique d'une heuristique pour trouver une solution quasi-optimale permettant de prendre dès le départ plus de raccourcis dans l'exploration avec raccourcis de la feuille 1. Cela permet de souligner que l'heuristique peut se combiner avec une exploration exhaustive tronquée donnant elle des garanties sur le résultat. Techniquement, c'est l'occasion de passer une fonction comme argument et de discuter de l'évaluation partielle pour cacher par exemple certains paramètre des heuristiques comme le nombre de tirage.

Nous proposons ensuite une heuristique par flips améliorant une solution courante. Ces flips "locaux" peuvent aboutir à des minimums locaux qui ne sont pas globalement optimaux. En option, nous présentons l'algorithme de Metropolis-Hasting qui garantit de naviguer dans l'espace de solutions selon des probabilités en lien avec la fonction de partition utilisée par les chimistes et les physiciens.

Enfin, nous présentions une probable 2-approximation basée sur la construction d'une tournée débutant par les deux points les plus proches et augmenté gloutonnement par la ville qui rallonge le moins la tournée partielle courante en s'insérant entre deux villes successives dans la tournée. La notion d'algorithme d'approximation n'est pas au programme de NSI, mais savoir que l'on peut avoir des garanties sur l'approximation nous semblait compléter le tableau de manière appropriée.