Intelligence Artificielle : Logique et Contraintes - Devoir n° 1

Jeux Olympiques 2016 à Rio

I Le contexte du problème

La ville de Rio de Janeiro, au Brésil, accueille du 3 au 21 aout 2016 les prochains jeux olympiques. L'opportunité de pouvoir admirer certains des plus grands athlètes de la planète ainsi que caractère mythique de cette ville, avec son cadre remarquable, ses longues plages magnifiques, son ambiance festive et son patrimoine culturel très riche, suscite un attrait sans égal. De nombreux particuliers et groupes de supporters ont déjà bloqué leurs dates de congés afin de pouvoir prendre part à cet événement exceptionnel. Certains ont déjà acheté les billets pour pouvoir assister à leurs épreuves favorites. D'autres, à l'approche de l'ouverture des jeux, se laissent peu à peu gagner par cette atmosphère fébrile et se demandent s'il est encore temps d'organiser quelque chose à cette occasion. Naturellement pour nombre d'entre eux, ce sera aussi (re-)découvrir cette ville fantastique et quelques uns de ses haut-lieux touristique.

L'agence *Sport et Voyages* se propose d'aider ses clients à organiser leur voyage au Brésil et notamment de les informer sur la fiabilité de leurs projets et s'occuper de l'achat de billets, que ce soit pour les manifestations sportives organisées dans le cadre des JO ou encore pour certains points d'intérêt au niveau touristique.

Une des difficultés auxquelles l'agence est confrontée vient du fait que les épreuves des JO se déroulent sur différents lieux, répartis en gros dans quatre zones de l'agglomération de Rio (cf figure 1) :

Barra à l'ouest, où se situe la majorité des installations et notamment le village olympique.

Copacabana près d'une des plus célèbres plages du monde, quasiment au centre de Rio.

Maracanã sur les lieux de l'un des plus fameux stades de football, où s'est notamment déroulée la finale de la dernière coupe du monde en 2014, mais également le très célèbre Sambodrome, où défilent chaque année les meilleurs écoles de samba à l'occasion du légendaire carnaval.

Deodoro au nord de Barra, où se sont déjà déroulés les jeux Panaméricains en 2007.

FIGURE 1 – Les sites des épreuves

Mais, coincée entre la mer et les montagnes, Rio est aussi célèbre pour ses très nombreux embouteillages. Les visiteurs sont naturellement invités à utiliser au maximum les transports en communs. Cependant, bien qu'en constante amélioration, ces derniers ne sont pas aussi efficaces qu'on pourrait le souhaiter. Les temps de déplacements sont souvent loin d'être négligeables et doivent être pris en compte dans le planning du voyage.

Afin de pouvoir répondre au mieux aux attentes de ses clients, l'agence dispose d'un certain nombre d'informations factuelles, tant sur les *points d'intérêt touristiques* (pit) de la ville (I.e. sites remarquables, musées, événements, etc) que sur les différentes sessions organisées dans le cadre des jeux olympiques. Elles dispose également d'informations sur les temps de transport permettant de relier les différents lieux concernés (que, pour simplifier, nous supposerons indépendants de l'heure de la journée), cruciales pour pouvoir organiser un séjour dans les meilleurs conditions. Toutes ces informations sont représentées dans un ou plusieurs fichiers texte, répondant à une syntaxe précise. Nous supposerons que chaque information pertinente figure sur une ligne constituée d'une suite de chaines de caractères (séparées par un nombre quelconque d'espaces et/ou de tabulations) et débute par un mot clé particulier. Toute ligne ne commençant pas par un des mots clés mentionnés ci-dessous sera considérée comme un commentaire et devra donc être ignorée. Dans la suite nous retiendrons notamment lignes répondant à la syntaxe suivante (les éléments figurant entre crochets étant considérés comme facultatifs) :


Rio : Pão de acucar (pit)

- `pit <idP> <hv:mv> <hd:md> <hf:mf> [<joursDescr>]`,
  représente un pit dont l'identifiant est `<idP>` nécessitant d'y consacrer une durée minimale de `<hv:mv>` et qui est accessible au public à partir de l'heure `<hd:md>`, jusqu'à l'heure `<hf:mf>`. La partie `<joursDescr>`, facultative, permet de restreindre la liste des dates auxquelles ce point d'intérêt est effectivement accessible (en cas d'absence, le pit est supposé accessible tous les jours). Il peut s'agir :
  — soit d'un simple entier `j` désignant un jour précis dans le mois (d'août 2016),
  — soit d'une paire d'entiers `j1-j2` (t.q. $j1 \leq j2$), représentant l'ensemble des jours de `j1` à `j2`
  — soit d'une suite d'entiers et/ou de paires, séparées par des virgules (mais sans espace ni tabulation), devant être interprétée comme une union de cas précédents
- `session <idS> <sport> <j> <hd:md> <hf:mf> <installation>` décrit une session des jeux olympiques (i.e. une épreuve ou un groupe d'épreuve d'une même discipline sportive pour laquelle il est possible d'acheter un billet) où `<sId>`, `<sport>` et `<installation>` sont des chaines de caractères caractérisant respectivement l'identifiant (unique) de la session, la classe des sports de cette session et le l'installation sportive où se déroule la session, `<j>` est un entier représentant la date (i.e. ici le jour du mois d'août) et où `<hd:md>` (resp `<hf:mf>`) représente l'heure de début (resp. de fin) de la session.
- `zone <idZ> <installation> [<avant> <apres>]` qui précise l'identifiant `<idZ>` de la zone olympique où se trouve une `<installation>` utilisé par au moins une session. Lorsqu'ils sont mentionnés, `<avant>` (resp. `<apres>`) représente le temps minimal (en mn) nécessaire pour pouvoir accéder à l'installation une fois arrivé sur la zone, et passer les contrôles de sécurité (resp. quitter l'installation pour rejoindre les réseaux de transport). Lorsqu'elles ne sont pas mentionnées on prendra respectivement comme valeurs 3 pour avant, et 20 pour après.
- `transfert <id1> <id2>` représente le temps (en mn) de transfert nécessaire pour passer d'un lieu `<id1>` à un lieu `<id1>` à l'aide des transports en commun.

Par exemple, le texte ci-dessous, extrait d'un tel fichier d'informations, indique notamment que la session `HB022` est une session de Handball, qui a lieu le 13/08 à la Future Arena, de 8h30 à 13h. Lle Musée d'Art Moderne de Rio est ouvert de 10h à 17h tous les jours sauf les lundis et le temps consacré à sa visite doit être au minimum de 4h. Le Corcovado est lui ouvert tous les jours sans exception de 8h à 19h et il faut compter 90mn de transport pour s'y rendre depuis la zone de Maracanã.

```
// --- Sessions ---
session  AT002  Athletics   12 20:20 23:25 OlympicStadium
session  HB022 Handball 13 09:30 13:00 FutureArena
..
// --- Zones ---
zone Maracanã OlympicStadium  60  30
zone Barra FutureArena
...
// --- Points d'intérêt touristique ---
pit Corcovado 3:00     08:00 19:00
pit MuseuDeArte 4:00 010:00 17:00  2-7,9-14,16-21,30,31
...
// --- Temps de transfert entre points ---
transfert Barra Maracanã  90
transfert Maracanã  Corcovado 60
...
```

II Problèmes à résoudre

II.1 Aide à la personnalisation d'un séjour

Si certains clients ont déjà bien anticipé le programme de leur séjour à Rio, d'autres n'ont pas pris le temps d'y réfléchir ou encore peuvent s'interroger sur ce qu'ils pourraient faire en plus de ce qu'ils ont déjà prévu. Afin de pouvoir répondre à leurs attentes, l'agence souhaite mettre au point un modèle Opl permettant de leur décrire tout ce qu'il est possible de faire à une date donnée, tout en prenant compte de leurs souhaits éventuels et respectants toutes les contraintes d'accès/évacuation aux installations sportives, ainsi que les temps de transfert entre différents sites.

Nous supposerons qu'une requête formulée par un client sera exprimée par une ou plusieurs lignes de la forme :

L'interprétation

- `demande <idD> <jour> <max> [(+<A> | -<A>)*]`
  où `<idD>` est une chaine (unique) identifiant la demande, `<jour>` et `<max>` sont des entiers, et où la partie facultative correspond à une suite de longueur finie de chaines de la forme `+<A>` où `-<A>` chaque `<A>` correspondent à un identifiant d'activité (i.e. session ou pit).

L'interprétation d'une telle ligne est que le client souhaite connaître toutes les séquences valides d'au plus `max` d'activités, représentées par la suite de leurs identifiants, réalisables le jour `<j>`. Chacune de ces séquences devra contenir tous les codes préfixés par un `+` sur la ligne et aucun des éléments préfixés par un `-`. Une séquence sera considérée comme valide s'il est possible d'enchainer deux activités consécutives de la séquence en respectant les temps de transfert entre les différents sites et les temps d'accès/évacuation des installations sportives pour les sessions. Par exemple :

```
demande  d1 12   3  +AT002  -Corcovado
```

indique qu'une demande `d1` a été formulée par un client, pour connaitre toutes les séquences d'au plus 3 activités réalisables le 12/08, sachant qu'il a déjà prévu d'assister à la session `AT002` et ne souhaite pas aller au `Corcovado` ce jour là.

Notez que si certaines demandes peuvent ne pas avoir de réponse, (par exemple, si l'on inclue et exclue une même activité, où encore si le nombre d'activités requises excède la valeur de `max`, dans la plupart des cas il y a au moins une réponse : la séquence vide. Et pour les requêtes dont la partie facultative est vide, le nombre de séquences valides peut être important.

Vous réaliser un premier modèle `choix.mod` qui, étant donné une instance d'un tel problème renverra, pour chaque demande formulée, l'ensemble séquences valides correspondantes. Chaque séquence `<A1> ... <Ak>` valide pour une demande identifiée par `<idD>` sera décrite par une ligne de la forme :

```
programmePossible  <idD>  <j>  <A1> ... <Ak>
```

Pour ce problème, comme pour ceux de la partie suivante, une instance sera représentée par un fichier `.dat` contenant deux informations : une chaine `nom` précisant le nom de l'instance et uen

ensemble de chaines `fichiersDonnees` qui mentionne l'ensemble des chemins d'accès vers les fichiers décrivant les données à prendre en compte pour le problème. Le fait de pouvoir répartir les données dans différents fichiers texte facilite le partage de données communes entre plusieurs instances.

Votre modèle devra permettre de sauvegarder la réponse (conformément à la syntaxe précédente) dans un fichier. Si au moins une des demandes formulées n'a pas de solution, le fichier devra contenir au moins une ligne réduite au mot clé `ECHEC` (selon la façon dont vous vous y prenez pour coder votre modèle vous pourrez ou non indiquer les programmes possibles pour les demandes ayant une réponse, mais ce n'est pas indispensable).

II.2 Réservation d'hébergement pour un séjour

Un autre service assuré par l'agence, est de permettre la réservation d'hébergement, tâche toujours délicate lorsque l'on ne connait pas une ville et qui risque de s'avérer compliquée compte tenu de la forte demande au moment des jeux. Prévoyante, l'agence a déjà négocié avec un certain nombre d'hôtels de l'agglomération, des quotas de chambres. Il ne s'agit pas vraiment de réservations définitives mais les hôteliers se sont engagés à ne pas les louer à d'autres clients, au moins jusqu'à une certaine date, sous réserve que l'agence leur précise chaque jour, le nombre de réservations confirmées. Nous supposerons que les informations relatives à ces hotels sont représentées dans les fichiers de données par des lignes de la forme :

- `hotel <idH> <categorie> <quota>` où `<idH>` est l'identifiant de l'hôtel, `<categorie>` est un entier (de 1 à 5) représentant le niveau de prestation de l'hôtel `<quota>` est un entier représentant le quota de chambres qui ont été bloquées pour l'agence (que pour simplifier nous supposerons toutes identiques.

On suppose à présent qu'un client veut faire une demande de réservation d'hôtel. Chaque client souhaite naturellement réduire autant que possible le temp qu'il va passer dans les transports et indique donc à l'agence son projet de planning, de façon à ce que l'agence puisse tenir compte autant que possible de ses projets. On suppose que chaque demande de reservation est décrite sous la forme :

- `reservation <idR> <d> <nj> <nc> <cat> programme <P1> | <P2> | ... | <Pnj>` où `IdR` identifie (de façon unique) décrivant demande une réservation de `<nc>` chambres, à compter du jour `<d>`, pour `<nj>` nuits, dans un hotel de catégorie compatible avec `<cat>`, où `<cat>` est soit un entier (entre 1 et 5), soit une paire `cinf-csup` d'entiers. Le premier cas indique que le client ne n'accepte qu'un hotel dont la catégorie est exactement `<cat>` alors que le second accepte n'importe quel hôtel de catégorie comprise entre `cinf` et `csup`.
  La fin de la ligne décrit le programme choisi pour chaque journée du séjour. Chaque `<Pi>` peut valoir soit le symbole `jourLibre`, soit un identifiant d'activité unique (si une seule activité est prévue le i-ème jour), soit les codes de la première et de la dernière activité de la journée, séparés par un espace.

II.2.1 Traitement d'une demande unique de réservation

Ecrire un modèle `meilleurHotel.mod` qui étant donné une instance contenant une unique demande de réservation sélectionne un hôtel de catégorie compatible avec la demande, dont le quota est suffisant pour couvrir la demande, et permettant de minimiser les temps de transport. Pour évaluer le temps de transport on ne comptabilise que le temps passé pour rejoindre depuis l'hôtel le lieu de la première activité et celui pour rentrer depuis le lieu de la dernière activité. Les temps intermédiaires pour passer d'un lieu à l'autre dans la journée sont de toutes façons incompressibles. Pour les journées déclarées libres le temps comptabilisé est de 0.
Remarque : Le calcul des temps de transfert, nécessite de connaître les temps de transfert entre les hotels et les lieux d'activité, ce qui suppose qu'ils aient été déclarés dans l'instance. Si l'information n'est pas disponible, nous supposerons arbitrairement un temps par défaut de 150mn.

Vous enregistrerez la solution obtenue pour une instance dans le fichier résultat sous la forme d'une ligne :

```
affectation  <idRes>  <idHotel>
```

II.2.2 Traitement de demandes multiples de réservation

On suppose a présent qu'on doit traiter simultanément un ensemble de demandes de réservations reçues par l'agence durant une même journée. Implicitement, nous supposons que toutes les personnes concernées par une même réservation suivent le même programme. Il peut cependant peut arriver que certains clients aient des programment différents mais souhaitent malgré tout être logé dans le même hôtel. On autorise donc en plus l'ajout, dans un fichier d'instance, de lignes de la forme :

`memeHotel  <idR1> ... <idRk>`

qui impose que l'hôtel réservé pour satisfaire les demandes de réservation `<idR1>` ... `<idRk>` soit identique.

Ecrire un modèle `repartitionHotels.mod` qui étant donné une instance contenant au moins une demande de réservation réalise une affectation optimale pour ces demandes, respectant les contraintes précédentes, sans excéder les quotas disponibles. On retiendra comme critère d'optimisation, la somme des temps de transferts correspondant à chaque réservation, pondérés par le nombre de chambres correspondant à ces réservations.

Dans le fichier résultat, vous décrirez la solution obtenue en précisant l'hotel affecté à chaque réservation par une ligne indiquer une ligne `affectation <idRes> <idHotel>` .

III Consignes à respecter

Ce devoir doit être réalisé **en binôme**. Vous rendrez ce devoir à la fois sous la forme d'un compte rendu papier, à déposer au secrétariat, et d'une archive au format `.tgz` dont le nom sera **impérativement** de la forme `Devoir_M1-App_IALC_Nom1_Nom2.tgz` (où Nom1 et Nom2 sont les noms respectifs des deux membres du binôme, dans l'ordre alphabétique).
Le sujet du mail devra être : `[M1-App] : Devoir IALC Nom1 Nom2`
L'archive devra contenir :

- un répertoire `doc` contenant une version **en pdf**, du compte rendu papier, nommée `cr_devoir_ialc_Nom1_Nom2.pdf`, et qui devra détailler votre modélisation, vos choix de mise en oeuvre et votre analyse des résultats obtenus sur les différentes instances (maximum 10 pages).
- un répertoire `src` contenant le code source de votre devoir, i.e. les fichiers de modèles OPL et les fichiers `.dat` décrivant les instances, regroupés dans un sous dossier `dat` que vous pourrez organiser librement. Vous penserez à mettre les noms de votre binôme en commentaire en tête de chaque fichier source.
- un répertoire `donneesInstances` qui devra contenir les fichiers `.txt` décrivant les différentes instances que vous aurez traitées. Vous pourrez structurer ce répertoire librement, du moment que les fichiers .dat référencent ces fichiers correctement.
- un répertoire `resultats` qui devra contenir les fichiers `.txt` décrivant les résultats obtenus pour les instances traitées avec vos différents modèles. Pour chaque instance de nom `<nomInstance>`, le résultat obtenu via un modèle `Mod` sera écrit dans un fichier `<nomInstance>_<Mod>.txt` dans le répertoire `resultats`.

Votre archive est à rendre par email **au plus tard le vendredi 22/04/16 à 23h59**. La version papier doit être déposée au secrétariat ce même jour. Prenez donc vos dispositions pour être certains d'avoir rendu au moins une version de votre devoir avant cette limite et d'anticiper sur tout problème de transport, panne de réseau, machine, etc... Vous pouvez éventuellement envoyer une version modifiée de votre devoir jusqu'à la limite fixée. Vous recevrez un accusé de réception dès que possible (durant la semaine suivante). Si ce n'est pas le cas. Afin limiter les risques de mails égarés, merci d'utiliser votre adresse `prenom.nom@u-psud.fr` pour l'envoi et pensez toujours à bien mettre votre binôme en copie du mail.

Enfin, il vous est demandé de rendre **un travail personnel**. Sachez que les codes seront tous vérifiés attentivement et **toute copie de code entre binômes sera sévèrement sanctionnée.**

IV Quelques conseils pour une bonne réalisation du devoir

Pour réaliser ce devoir vous aurez besoin d'effectuer des opérations d'entrées sorties sur des fichiers (pour lire les données des instances et écrire les résultats). Vous trouverez, les informations utiles sur la façon de procéder, dans la partie de la documentation en ligne portant sur *IBM Ilog Script Reference Manual* et plus particulièrement dans la rubrique *OPL Classes* et notamment au sujet des classes `IloOplInputFile` et `IloOplOutputFile`. De façon générale, il faudra utiliser des primitives du langage *IBM Ilog Script* pour ouvrir et lire le fichier ligne a ligne, et en extraire les informations utiles pour alimenter des structures de données *OPL* modélisant les informations brutes de l'instances. Ensuite, vous pourrez faire les prétraitement adéquats sur ces données, pour alimenter les données dont vous avez besoin pour vos modèles. Il est également possible de concevoir des fonctions permettant de charger dynamiquement des modèles et donc d'automatiser la résolution d'une (ou plusieurs instances) avec différents modèles.

Rappelez vous notamment que par rapport aux possibilités offertes par le lange `OPL il existe des limitations` sur ce que l'on peut faire dans *IBM Ilog Script for Opl*. Notamment, il n'est possible de construire que des tuples dont les attributs correspondent à des valeurs **atomiques** (donc, ni des ensembles, ni des tableaux). Il peut donc être nécessaire de passer par certaines structures de données auxiliaires, utiles juste pour enregistrer de façon "plate" les informations lues dans les fichiers de l'instance, puis de faire ensuite quelques pré-traitements, pour reconstruire d'autres structures plus élaborées, mieux adaptées à résolution de votre problème.

Autres remarques :

- Vous trouverez sur la page du cours : `http://www.lri.fr/~chatalic/Enseignement/ia-lc-m1` une archive correspondant à la structure demandée et contenant une pseudo-instances avec au moins une exemple de ligne pour chaque type d'entrée pouvant être rencontrée dans les données dérivant une instance. Des données plus conséquentes seront ajoutées sur la page ultérieurement
- Si certains points s'avèrent ambigus et nécessitent des précisions particulières, dans un soucis d'équité, les réponses à vos éventuelles questions seront faites sur la page du cours. Attention toutefois, je serai en déplacement à compter du 15/05 et très difficilement joignable. Donc n'attendez pas pour vous y mettre dès à présent.
- Par rapport à ce que ce que vous avez déjà pu réaliser en td, la principale différence est qu'il faut ici commencer lire les données du problème, en extraire les informations pertinentes et choisir comment les représenter de façon efficace et pratique pour vos modèles. Mais il s'agit là essentiellement de programmation *classique*, indépendant de l'aspect *contraintes*.
- Les modèles eux même ne posent pas de grande difficulté par rapport à ce que vous avez déjà rencontré en cours et en td. Notez que la partie concernant la lecture des données peut-être mutualisée entre les différents modèles.
- Il faudra vous assurer pour chaque modèle proposé, les propriété à satisfaire sont modélisée de façon adéquate. Afin de s'en convaincre, il est fortement conseillé de se créer différentes instances très simples (i.e. avec très peu de données) pour tester séparément les différents aspects du problème.
- Lorsque votre code vous semblera aboutit, vous pourrez concevoir des instances plus complexes, qui mélangent les différents aspects du problème et tester le passage à l'échelle.
- Les bons principes de génie logiciel s'appliquent quels que soient les langages utilisés. En particulier, la modularité est toujours un élément essentiel de la qualité du code produit. Cela facilite non seulement la lisibilité mais cela permet également d'avoir les idées plus claires lors de la conception, et facilite le déboggage. Donc n'hésitez pas ici à décomposer votre code OPL Script en un ensemble de petites fonctions indépendantes, qui font des choses précises et limités.
- Pour ceux qui n'ont pas installé la suite d'IBM sur leur propre machine (ce qui n'est pas vraiment indispensable) il vous est rappelé que vous pouvez vous connecter à distance via ssh sur les machines `tp-ssh1.dep-informatique.u-psud.fr` ou `tp-ssh2.dep-informatique.u-psud.fr` et lancer `oplrun` en mode console. Il est donc tout à fait possible de travailler à distance.
- Ceux qui ont l'habitude d'utiliser des outils de gestion de version (git, svn,...) savent naturel-

lement que cela leur facilitera la vie pour synchroniser facilement du code entre membres d'un même binôme, ou encore entre machine vos machine locales et celles du département.

- Enfin, **n'attendez pas le dernier moment pour vous y mettre !** Votre objectif doit être d'arriver à lire les données des fichiers d'instance dans les meilleurs délais. Cela peut se faire progressivement. Commencez par voir comment récupérer certaines lignes seulement, en ignorant les autres. Puis enrichissez votre code jusqu'à arriver à lire correctement les informations de toutes les lignes du format décrit.
- Soignez la rédaction de votre petit rapport de synthèse, en décrivant bien vos modèles et la façon de traduire les contraintes, et commentez vos résultats.

Après l'effort... le réconfort !

Rio : La plage d'Ipanema