Intellectica, 2002/2, 35, pp. 179-183

Commentaires sur le texte de Giuseppe Longo

Jacques Pitrat*

Commentaries on the text of Giuseppe Longo

A. ENTERRONS LE TEST DE TURING

Le test de Turing était une excellente idée quand il a été proposé, il y a plus de cinquante ans, pour mesurer la qualité des programmes de ce qui ne s'appelait pas encore l'Intelligence Artificielle : il était naturel de commencer par envisager une méthode simple. Pourtant il n'a plus aucun intérêt actuellement, car le but de l'IA n'est pas d'imiter l'être humain, mais de créer des programmes performants. Pourquoi devrions-nous handicaper un programme en l'obligeant à imiter les inefficacités de notre fonctionnement ? Pourquoi nous limiter au niveau de l'intelligence humaine, nous devons viser à réaliser des programmes qui auraient des performances bien supérieures aux nôtres. Un bon programme d'IA ne doit donc pas satisfaire le test de Turing parce que les juges penseraient à propos de ses trouvailles : « Voilà une idée excellente et originale à laquelle je n'aurais jamais pu imaginer, comment un être humain pourrait-il être aussi astucieux ? C'est impossible, cela ne peut être qu'un programme ! »

Hugh Loebner (http://www.loebner.net/Prizef/loebner-prize.html) a créé une compétition ouverte aux programmes ; le gagnant est celui qui passe le mieux le test de Turing. Les résultats sont parfois acceptables, mais ces programmes eux-mêmes sont très décevants, ils sont un recueil de trucs très simples pour faire croire qu'ils comprennent des questions qui les dépassent complètement. Nous avons ainsi dans les résultats de cette compétition des dialogues comme le suivant, où R est la réponse du programme testé à la question Q d'un juge :

Q : Quelle est votre position sexuelle préférée ?

R : Je préférerais ne pas parler de sexe.

Q : Quelle est votre position préférée ?

R : Réservons cette idée à plus tard. Laissez-moi vous interroger et vous demander : quelle année êtes-vous né ?

De tels programmes ne méritent guère d'être appelés intelligents, bien qu'ils en aient certaines apparences, et leur développement n'a, pour le moment, rien apporté d'utile à l'IA.

Il est très souhaitable de pourvoir mesurer la qualité d'un programme d'IA, mais le test de Turing ne donne pas satisfaction. Il

* CNRS-LIP6, Université Pierre et Marie Curie

se base sur un examen des résultats obtenus, ce qui n'est pas suffisant, car nous devons aussi tenir compte d'éléments cachés et difficiles à évaluer : la quantité d'intelligence humaine incluse dans le programme, l'aide donnée au programme par la façon dont on a choisi de poser un problème et la généralité du programme.

Les programmes sont souvent comme le fameux pâté mi-cheval, mi-alouette : ils contiennent un cheval d'intelligence humaine et une alouette d'intelligence artificielle, ce qui est le cas quand un chercheur a passé des années à donner à son programme toutes les méthodes qu'il a découvertes pour résoudre une famille de problèmes.

Prenons un exemple qui montre bien qu'il ne faut pas juger un programme uniquement sur l'apparence. Lors du championnat du monde de 1997 des programmes de Go, la sixième place sur 38 concurrents a été prise par Gogol, présenté par Tristan Cazenave. À la vue de ce résultat, nous pourrions conclure que Gogol est moins intelligent que le programme vainqueur, Handtalk. En réalité ce n'est pas si simple, car les programmes placés avant Gogol ont été écrits par des humains qui ont passé des années à les améliorer ; par contre, Gogol avait été en grande partie écrit par un autre programme, Introspect, qui, lui, avait été écrit par Tristan Cazenave. Aussi Gogol contient-il bien plus d'intelligence artificielle que les programmes qui le précédaient. Son intelligence est issue d'Introspect, qui est capable d'analyser des suites de coups et de manipuler les règles du Go pour en déduire des connaissances utiles pour un programme jouant à ce jeu. Le programme Introspect, créateur de Gogol, a fait un travail analogue à celui de Chen Zhi Xing, créateur de Handtalk.

Il arrive aussi souvent qu'une aide considérable soit donnée à un programme par la façon dont son auteur lui présente chaque problème ; il faut donc examiner attentivement l'énoncé du problème tel qu'il est donné au programme. Par exemple, un programme avait démontré que la suite des nombres premiers était illimitée, mais on lui avait donné tout ce qui était nécessaire pour la démonstration et rien d'autre. La clé de la preuve est de considérer un nombre du type N!+1. Or l'auteur du programme avait parachuté dans les données le théorème « N!+1>N » ; aussi le programme n'a-t-il pas eu grand mérite à considérer ce nombre qui lui a été en quelque sorte soufflé. De même, un programme a réussi à démontrer le théorème de Gödel, mais il lui avait été donné un exemple de formule réflexive, ce qui est la clé de la démonstration.

Il est également important d'évaluer la généralité d'un programme, d'une part parce que la généralité est une qualité essentielle de l'intelligence, mais aussi parce que l'on ne peut pas donner trop de connaissances ad hoc à un programme général ; on limite donc les risques de téléguidage trop précis. Il est particulièrement utile de voir si ce programme a ou non trouvé des résultats inconnus de son auteur. En effet, il est bien connu que

l'auteur d'un programme arrive toujours à ce que son programme trouve un résultat connu de son auteur.

Je ne prétends pas en quelques lignes régler le problème de l'évaluation de la qualité des programmes d'IA. Je veux seulement montrer que c'est un problème important et difficile et que le test de Turing, idée géniale en son temps, ne permet pas de le résoudre. Pour juger l'intelligence d'un programme, il faut savoir ce qui se passe dans le « cerveau » de ce programme et pas seulement le juger par son comportement extérieur.

B. LES EXÉCUTIONS DES PROGRAMMES NE SONT PAS RÉPÉTABLES

Contrairement aux idées reçues, les résultats obtenus par un programme ne sont pas toujours répétables : sans que nous l'ayons modifié, un programme peut avoir un comportement différent quand on lui donne une deuxième fois les mêmes données. Quand cela se produit, cela peut être désagréable pour le programmeur : il est très difficile par exemple de repérer des bogues qui se manifestent de façon intermittente. Les raisons en sont multiples.

Erreur de l'ordinateur.

Le cas des erreurs de l'ordinateur, évoqué par Giuseppe Longo, arrive plus souvent qu'on ne le souhaiterait quand il faut plusieurs jours pour résoudre un problème. C'est tellement vrai que ceux qui établissent les bases de données de finales d'échecs [Hsu and Liu 2002] prennent la précaution d'écrire un programme de vérification de ces bases qui découvre effectivement de telles erreurs.

Interaction avec un univers qui change

Un programme qui commande un robot ne se trouvera jamais deux fois dans la même situation, même si l'on essaye de replacer le robot exactement au même endroit. L'imprécision du positionnement, le changement d'éclairage, de la charge de la batterie, des conditions météorologiques font que le robot n'aura pas deux fois de suite le même comportement.

Cela peut se produire aussi avec des programmes qui ne commandent pas des robots, comme celui qui résout des mots croisés [Littman, Keim and Shazeer 2002]. Il reçoit une grille de mots croisés ainsi que les définitions telles qu'elles sont données dans le journal où ce problème a été pris. Il faut remplir correctement la grille en se servant des définitions. Dans les mots croisés américains, on peut utiliser des noms propres qui sont d'actualité, bien qu'ils ne figurent dans aucun dictionnaire. Si la définition porte sur le nom d'une actrice qui joue dans un film récent, le programme va le chercher sur le web dans une base de données où figurent les acteurs des films. Une panne d'un serveur, une modification d'une base que l'on ne contrôle pas, la présence d'un ver qui perturbe sérieusement

le trafic peuvent conduire à des résultats différents quand le même programme lance plusieurs fois la même recherche.

L'interruption par l'horloge

Une instruction d'un programme peut l'obliger à suspendre dans N secondes ce qu'il est en train de faire et lancer alors un certain sous-programme. Ce dernier pourra par exemple examiner alors l'avancement de la solution et changer éventuellement la valeur des paramètres qui régissent le comportement du programme ; celui-ci reprendra ensuite l'exécution là où elle a été arrêtée. Mais comme le temps d'exécution d'un programme n'est jamais exactement le même, lors de deux exécutions avec les mêmes données le programme ne sera pas interrompu exactement au même instant. La situation lors de l'interruption ne sera donc pas la même : le sous-programme appelé à ce moment pourra avoir un comportement différent pour deux exécutions apparemment identiques.

Le système d'exploitation

Même s'il n'y a qu'un seul utilisateur, un ordinateur contient, en plus de son programme, le système d'exploitation qui inclut des programmes qui se lancent d'eux-mêmes. Notre programme ne sera ainsi jamais deux fois de suite dans le même environnement, puisque le système a pu décider de lancer une tâche dans un cas et pas dans l'autre. Le comportement du programme ne sera pas le même si ses réactions dépendent du temps qu'il a passé pour effectuer une certaine étape. De plus, l'état du système dépend des tâches qu'on lui a données à exécuter auparavant et qui sont différentes pour chaque passage de notre programme. Cela est particulièrement sensible avec les accès disques qui peuvent être inutiles si une exécution précédente a déjà amené en mémoire rapide l'information qui est demandée.

Les programmes peuvent apprendre

Certains programmes sont capables d'apprendre au fur et à mesure qu'ils résolvent des problèmes. Cela peut se manifester par la modification de paramètres qui régissent leur fonctionnement. Le programme peut aussi réécrire certains des sous-programmes qui le constituent en remédiant ainsi à leurs faiblesses ; il utilise immédiatement ces sous-programmes à la place des anciens. De tels programmes ont des comportements différents quand on leur propose plusieurs fois le même problème, car ils tiennent compte de leurs erreurs passées.

L'irrépétabilité est-elle utile en soi ?

Il y a une quarantaine d'années, certains avaient proposé d'ajouter aux ordinateurs une instruction qui produirait un nombre aléatoire, par exemple à partir du bruit de fond d'un tube à vide. Avec une telle instruction, nous aurions eu des programmes indiscutablement irrépétables ; cela répondrait à une objection de

Giuseppe Longo sur l'impossibilité du jeu de l'imitation. Pourtant on n'a jamais mis en œuvre cette idée. J'avais réfléchi à cette époque [Pitrat 1964] à l'intérêt d'introduire de l'aléatoire dans un programme. En général, cela sert à définir avec peu d'instructions un balayage de plus en plus fin d'un espace de recherche ; mais pour cela les nombres pseudo-aléatoires sont largement suffisants. Par contre, cela aurait rendu très difficile la mise au point des programmes, car les résultats varieraient trop d'un essai à l'autre. On augmenterait ainsi considérablement la difficulté du programmeur sans apporter aucun avantage pratique.

Références

Hsu T. and Liu P., Verification of endgame databases, *ICGA Journal*, 25-3, 2002, 132-144.

Littman M., Keim G. and Shazeer N., A probabilistic approach to solving crossword puzzles, *Artificial Intelligence*, 134-1, 2002, 23-55.

Pitrat J., Utilisation des nombres aléatoires dans les problèmes d'intelligence artificielle, *Revue Française de Traitement de l'Information*, 1964, 25-48.