La simulation de la pensée

In L'année psychologique
Paul Fraisse

Presses Universitaires de France
N°67, 1
1967, pp.135-151

Lien internet permanent pour l'article :
https://www.gerard-vergnaud.org/GVergnaud_1967_Simulation-Pensee_Annee-Psychologique-67-1

l'année psychologique

DIRECTEUR : PAUL FRAISSE

67e ANNÉE - 1967 - FASCICULE 1
PUBLIÉE AVEC LE CONCOURS DU C.N.R.S.

MÉMOIRES ORIGINAUX

PRESSES UNIVERSITAIRES DE FRANCE

NOTE

Laboratoire de Psychologie de l'École Pratique des Hautes Études

LA SIMULATION DE LA PENSÉE

par Gérard VERGNAUD

Au cours des dix dernières années, la simulation sur ordinateur a été promue au rang de méthode scientifique. Lorsque l'évolution d'un système donné est peu prévisible, le meilleur moyen de se renseigner sur l'avenir de ce système, c'est de le simuler par un autre système possédant certaines caractéristiques et relations du premier, mais pas toutes (auquel cas il ne s'agit plus de simulation mais de reproduction). La simulation sur maquette (conditions de vol d'un avion, barrage) rentre dans ce cadre général puisque effectivement la maquette possède certaines caractéristiques seulement du système simulé. Toutefois il est clair qu'il s'agit là d'une simulation privilégiée puisque la validité des conséquences repose sur l'identité physique de certaines composantes du simulateur avec les composantes correspondantes du simulé.

Dans le cas de la simulation sur ordinateur, c'est une tout autre entreprise : les composantes du système simulé sont traduites en informations abstraites et c'est ce système formel d'informations qui sert de simulateur. D'autre part il s'agit d'un modèle qu'on ne sait pas en général exprimer de façon analytique (Guittet, 1966 ; Renard et Renault, 1966) et dans lequel les calculs sont d'une complexité et d'une longueur telles que seul un ordinateur moderne peut les mener jusqu'au bout avec quelque chance de succès.

La simulation permet ainsi d'étendre la notion de modèle à des systèmes beaucoup plus complexes que ceux classiquement utilisés et dans lesquels les calculs pouvaient s'effectuer à la main.

Ce caractère abstrait de la simulation sur ordinateur a pour premier avantage qu'on peut simuler n'importe quoi : il suffit, mais ce n'est pas là une mince condition, d'avoir des informations assez sûres sur le système qu'on veut simuler. Pour s'en tenir à la psychologie, on peut simuler aussi bien l'évolution d'une névrose (Colby, 1963) que le jeu d'un joueur d'échecs (Newell, Shaw, Simon, 1963 *b*) ou un apprentissage par cœur (Feigenbaum et Simon, 1963). Tout cela relève de la

pensée, à un titre ou à un autre, mais nous nous en tiendrons à ce qui relève de la solution de problème au sens restreint car c'est de loin le domaine où la simulation a été le plus pratiquée, et son examen suffit à poser les problèmes les plus importants.

On distingue à juste titre entre les travaux sur l'intelligence artificielle (résoudre au mieux et pas nécessairement comme l'homme une certaine catégorie de problèmes) et les travaux sur la simulation proprement dite (reproduire au mieux les processus de la pensée humaine, y compris les errements). Cependant la distinction n'est pas si simple car on se sert du modèle humain pour mettre au point les programmes d'intelligence artificielle, et les travaux de pure simulation, qui sont plus rares, s'en inspirent considérablement. C'est pourquoi, dans la suite, nous ne ferons pas toujours la distinction.

I. — Que simule-t-on ?

Tout processus de solution de problème peut être simulé, encore faut-il qu'il soit mis sous une forme adéquate. C'est cette forme qui détermine la catégorie des processus simulables. Elle comporte deux contraintes fondamentales :

1° On doit pouvoir formuler sans ambiguïté et sans contradiction les données du problème ;

2° On doit pouvoir écrire le but à atteindre dans des termes qui ont un sens pour la machine.

L'une des tâches essentielles des chercheurs dans ce domaine consiste à trouver dans chaque cas de simulation le codage qui permette de satisfaire ces deux conditions. Les premiers travaux ont été faits dans des cas où elles sont satisfaites sans trop de mal.

La machine à démontrer des théorèmes (logique, géométrie, etc.)

Les axiomes qu'on donne à la machine sont ceux-là mêmes (mais écrits avec soin) que devrait utiliser l'étudiant en mathématiques à qui on soumettrait le problème. Le but à atteindre est simplement le (ou les) théorème(s) à démontrer. C'est ainsi que Newell, Shaw et Simon (1963 *c*) ont construit un programme, le *Logic Theorist*, capable de démontrer, à partir de l'axiomatique du chapitre Ier des *Principia Mathematica*, tous les théorèmes du chapitre II. Hao Wang (1960) a pu démontrer en 8 minutes sur IBM 704 tous les énoncés des 9 chapitres des *Principia Mathematica* de la forme

$$(x1)\ (x2)\ \ldots\ (xn)\ (Ey1)\ (Ey2)\ \ldots\ (Eym)\ P\ (x1, \ldots, xn, y1, \ldots, ym)$$

où (x) et (Ey) sont respectivement les quantificateurs universel et existentiel. Cela donne une idée suffisante de la puissance des ordinateurs modernes.

Gelernter (1963) a écrit un programme qui démontre des théorèmes de géométrie plane.

Voici l'un des exemples qu'il donne (fig. 1) :

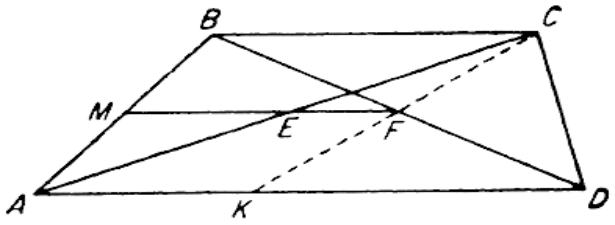

Fig. 1

Prémisses :

Soit le quadrilatère ABCD,
BC parallèle à AD,
E, milieu de AC,
F, milieu de BD,
MEF colinéaires dans cet ordre,
AMB colinéaires dans cet ordre.

Démontrer :

MB = MA.

Pas de symétries syntactiques :

1er essai : échec au bout de 8,12 minutes.

Informations supplémentaires :

— construire CF et le prolonger jusqu'à son intersection avec AD en K ;
— ajouter aux prémisses :
CFK colinéaires dans cet ordre,
AKD colinéaires.

2e essai : démonstration :

BC parallèle à AD (prémisse),
AKD colinéaires (prémisse),
KD parallèle à BC (segments colinéaires à des segments parallèles sont parallèles),
K et C de part et d'autre de la droite DB (voir diagramme), considérons DB (voir diagramme),
l'angle KDB est égal à l'angle CBD (alternes-internes),
CFK sont colinéaires dans cet ordre (prémisse),
DFB sont colinéaires dans cet ordre (par définition du milieu),
l'angle KFD est égal à l'angle CFB (opposés par le sommet),
DF égale FB (par définition du milieu),
considérons le triangle FDK (voir diagramme),
et le triangle FBC (voir diagramme),
FDK et FBC sont égaux (deux triangles sont égaux s'ils ont un côté égal compris entre deux angles égaux),
KF égale CF (côtés correspondants de deux triangles égaux),
CE égale EA (par définition du milieu),
considérons le triangle AKC (voir diagramme),
CEA sont colinéaires dans cet ordre (par définition du milieu),
EF parallèle à AK (le segment joignant les milieux de deux côtés d'un triangle est parallèle au troisième côté),
EF parallèle à KD (segments colinéaires à des segments parallèles),
FE parallèle à BC (segments parallèles à un même segment sont parallèles),
MEF sont colinéaires dans cet ordre (prémisse),
MEF colinéaires *(a fortiori)*,
FM parallèle à BC (segments colinéaires à des segments parallèles),

FM parallèle à DA (segments parallèles à un même segment),
considérons le triangle DBA (voir diagramme),
AMB sont colinéaires dans cet ordre (prémisse),
MB égale MA (une droite parallèle à la base d'un triangle et coupant l'un des côtés en son milieu, coupe l'autre en son milieu).

Temps total écoulé = 30, 68 minutes.

Son programme a même trouvé plusieurs démonstrations originales : par exemple, pour un triangle isocèle ABC (AB = AC), il a démontré l'égalité des angles à la base B et C en écrivant l'égalité suivante : triangle ABC = triangle ACB.

Il est clair toutefois que ce type de simulation ne se rapporte qu'à une partie seulement de l'activité mathématique et que le jour n'est pas encore arrivé où un programme sera capable d'inventer les concepts utiles et de dégager les axiomes puissants. Pitrat (1966), qui a construit une machine logique, fait remarquer que le travail du mathématicien comporte trois types de problèmes :

1° Passer d'une axiomatique définie à un théorème énoncé à l'avance ;
2° Démontrer à partir d'une axiomatique définie des théorèmes intéressants ;
3° Trouver une axiomatique intéressante.

Les simulations pratiquées jusqu'à ce jour ne relèvent que du premier type et l'on ne sait guère comment définir, dans un langage compréhensible pour la machine, les deux derniers types.

Dans le cas du joueur d'échecs, les axiomes sont les règles du jeu et la situation initiale de l'échiquier. Le but à atteindre n'est pas une certaine disposition de l'échiquier mais des relations entre les pièces (échec et mat). Là encore, Newell, Shaw et Simon (1963 *b*) ont joué le rôle de pionniers. A l'heure actuelle, on a des programmes qui battent parfois les meilleurs joueurs. Newell et Simon (1963) ont également développé un programme, le *General Problem Solver* (G.P.S.), qui serait applicable à tous les problèmes. Encore faut-il, là encore, pouvoir coder dans le langage du G.P.S. les problèmes qu'on veut lui soumettre.

Du fait que l'ordinateur ne fait pas autre chose que manier des chaînes de symboles, la philosophie des ordinateurs tient à peu près toute dans celle des systèmes formels, que les logiciens ont développé bien avant l'invention des machines à calculer. C'est pourquoi il est nécessaire de se familiariser d'abord avec les systèmes formels logiques, la théorie de la déduction, les algorithmes et les fonctions calculables (Trahtenbrot (1963) pour une bonne introduction). Par exemple, les théorèmes de limitation en logique se sont révélés d'une importance fondamentale pour la théorie des machines.

La deuxième condition dont nous avons parlé plus haut (but compréhensible par la machine) vient de ce qu'on ne saurait engendrer avec un système formel autre chose que des expressions bien écrites (respectant les règles d'écriture du système).

La solution du problème consiste, formellement, en une suite d'énoncés tirés valablement des axiomes et dont le dernier énoncé consiste dans l'énoncé qu'il fallait engendrer (théorème à démontrer par exemple). Ce qu'on peut tirer valablement des axiomes est régi par certaines règles d'inférence :

— règles de substitution de variables : on peut remplacer toute occurrence d'une variable par la même expression bien formée ;
— règles de remplacement des expressions par d'autres, équivalentes, par substitution de connectifs ;
— règles de détachement A, $A \supset B \Rightarrow B$ *(modus ponens)* ;
— règles d'enchaînement $A \supset B$, $B \supset C \Rightarrow A \supset C$ (syllogisme).

Les règles d'inférence disent ce qui est permis, non ce qu'il faut faire. Le problème fondamental est alors le suivant : quelles règles de production doit-on mettre dans le programme pour qu'il engendre de telles suites ?

II. — Algorithmes et méthodes heuristiques

Une réponse immédiate à la question précédente consiste à dire : combinons tous les axiomes entre eux et de toutes les façons permises ; parmi les combinaisons ainsi engendrées, se trouveront à coup sûr les combinaisons que l'on cherche. Une telle procédure est impraticable dès les premiers pas car le nombre de combinaisons croît à une allure exponentielle. Il faut donc recourir à d'autres méthodes. On utilise des algorithmes et des méthodes heuristiques.

Un algorithme est une procédure qui permet de résoudre en un nombre fini de pas (qui peut être très grand) tout problème d'une classe donnée à l'avance. C'est ainsi qu'il existe un algorithme, en logique des propositions, pour savoir si une proposition est un théorème, c'est la méthode des tables de vérité. Par exemple, pour savoir si l'expression $(P \rightarrow Q) \leftrightarrow (P \mid \bar{Q})$ est un théorème de la logique des propositions, il faut et il suffit qu'elle soit vraie quelle que soit la valeur de vérité de P ou de Q. Pour le savoir, on écrit le tableau suivant, avec les significations habituelles :

$\rightarrow$ implication
$\leftrightarrow$ équivalence
$\mid$ incompatibilité
$\bar{Q}$ négation de Q
V vrai
F faux

P	Q	$\bar{Q}$	$P \rightarrow Q$	$P \mid \bar{Q}$	$(P \rightarrow Q) \leftrightarrow (P \mid \bar{Q})$
V	V	F	V	V	V
V	F	V	F	F	V
F	V	F	V	V	V
F	F	V	V	V	V

l'expression est bien un théorème.

Malheureusement, pour la plupart des théories, il n'existe pas d'algorithme qui permette de décider si une expression est ou non un théorème, et pour la plupart des problèmes, il n'existe pas non plus d'algorithme de solution. Pire, certains algorithmes sont tels que le nombre des opérations à faire dépasse les capacités des machines actuelles les plus puissantes. Il faut donc recourir à d'autres méthodes : on utilise des heuristiques.

Une heuristique est une méthode qui, sans engendrer dans tous les cas la solution d'un problème, réussit le plus souvent, grâce aux règles judicieuses qu'elle donne sur le choix des opérations à faire. Ces méthodes ont été objet de réflexion en psychologie avant de l'être en théorie des machines (Polya, 1954). Il suffit de donner quelques exemples pour voir à quel point elles sont choses de la vie quotidienne (nous empruntons ces exemples à Pitrat, 1966).

Heuristiques générales :

— quand une situation nouvelle a des analogies avec une situation déjà étudiée, essayer en priorité les méthodes qui ont déjà donné de bons résultats ;
— peut-on poser le problème sous une autre forme ?, etc.

Heuristiques particulières :

En géométrie : si une figure a un axe de symétrie, le tracer.
Au bridge : honneur sur honneur ;
avant le mort, jouer dans la forte du mort, etc.

Les chercheurs en matière d'intelligence artificielle ont dû s'intéresser tout de suite aux méthodes heuristiques. Leurs programmes sont en général constitués d'algorithmes partiels et de méthodes heuristiques.

On peut dire, dans un langage naïf, qu'une méthode heuristique essentielle consiste à découper un problème en sous-problèmes et à rechercher la solution des sous-problèmes. Si on veut par exemple produire c à partir de a et que l'on sait qu'on peut produire c à partir de b, alors il peut être intéressant d'essayer de produire b à partir de a. Mais cette procédure contient en germe la théorie des langages et des métalangages dont il est nécessaire que le programmeur soit averti, s'il veut rendre son programme efficient. Par exemple, si l'on appelle *langue* l'ensemble des axiomes et des énoncés qu'on peut tirer valablement des axiomes, les règles d'inférence vues plus haut (règles de substitution, de détachement, d'enchaînement) appartiennent à la *métalangue*, puisqu'elles énoncent les opérations qu'on peut faire sur les énoncés de la langue. Il en va de même pour les règles de production : comme la machine ne fait pas ce qu'on ne lui dit pas de faire, il est nécessaire de la charger avec les métathéorèmes nécessaires aux productions qu'on lui demande.

Dans l'exemple précédent, supposons que a soit un axiome, b et c des théorèmes de la langue : ceux-ci contiennent éventuellement le signe de l'implication que l'on notera $\supset$.

Dire qu'on peut produire c à partir de b, c'est déjà se situer dans la métalangue, on notera $\Rightarrow$ l'implication de la métalangue. Le théorème de transitivité de l'implication auquel nous faisons appel dans la procédure heuristique citée est en fait un métamétathéorème ou encore M^2 théorème : on notera $\rightarrow$ l'implication de la M^2 langue et on écrira $a \Rightarrow b,\ b \Rightarrow c \rightarrow a \Rightarrow c$.

Le programme qu'on charge sur la machine doit donc comporter non seulement des axiomes de la langue mais des axiomes des métalangues. Les M théorèmes d'ordre 2, 3, voire au-delà, sont des instruments puissants qui contribuent activement à l'efficacité du programme (Pitrat, 1966).

La méthode heuristique que nous venons d'examiner comporte une marche à rebours : trouver b qui produit c, puis chercher à produire b à partir de a. Newell, Shaw et Simon (1963 *a*) explicitent de la façon suivante les avantages de la marche à rebours : une preuve comporte une suite d'énoncés dont les premiers sont les théorèmes connus A, le dernier le théorème à démontrer C, et les théorèmes intermédiaires sont les B.

— considérer A et C laisse un champ de possibles très important ;
— considérer A et B conduit à des procédures de type combinatoire dont on a vu les inconvénients ;
— considérer B et C, ou encore partir à rebours, présente des avantages certains qui tiennent à ce qu'il y a plus de théorèmes initiaux que de théorèmes finaux.

A ces règles heuristiques très générales, il faut ajouter des règles heuristiques particulières. Dans le jeu d'échecs, par exemple, les coups joués sont subordonnés à un système d'évaluation du jeu et à une anticipation plus ou moins grande des coups à venir qui permettent de jouer le coup le meilleur, ou plus exactement le moins mauvais (stratégie minimax). Dans le système d'évaluation on tient compte de plusieurs critères (sécurité du roi, qualité des pièces en jeu, contrôle du centre, etc.), ce qui donne des valeurs et des utilités vectorielles. Comme le jeu est à somme nulle (ce que perd l'un des joueurs est gagné par l'autre), la valeur de la position du jeu pour un joueur est l'inverse de celle de son partenaire et l'évaluation des coups peut facilement être faite en considérant le jeu des deux points de vue. Les règles heuristiques sont pleines de « coups de pouce » empiriques dictés par les experts. Un problème essentiel est celui de l'apprentissage par le programme de règles heuristiques nouvelles et plus puissantes. Samuel (1963) avait déjà réalisé un programme à jouer aux dames doué de la faculté d'apprendre (il apprenait à associer un nombre à chaque position du jeu), mais on écrit maintenant des programmes susceptibles d'apprentissage au deuxième degré (apprendre à apprendre, par modification des règles heuristiques) : c'est le cas par exemple du programme GAKU de A. Hormann (1965) pour le jeu de la tour d'Hanoi.

III. — Le problème de l'apprentissage

C'est surtout parmi les chercheurs qui s'intéressent à la reconnaissance des formes *(patterns)* qu'on se préoccupe le plus de doter les programmes de l'aptitude à apprendre. On connaît le schéma de ce problème. Une forme est projetée sur une matrice, par exemple de 250 000 cases (500 × 500), et une exploration systématique de la matrice révèle les cases marquées, ce qui donne une suite de symboles 0 et 1. Il s'agit de classer ces chaînes de symboles de telle sorte que les classes correspondent aux classes dans lesquelles un sujet humain rangerait les patterns perceptifs, par exemple les lettres de l'alphabet.

Voici un exemple (fig. 2), donné par Uhr et Vossler (1963), avec une matrice de 20 × 20 : chaque case où figure une trace est marquée 1, les autres sont marquées 0.

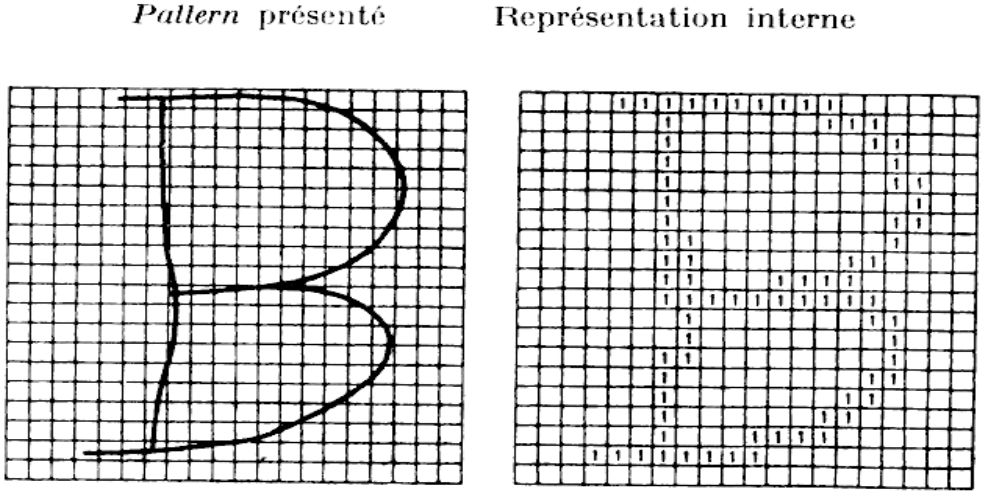

Fig. 2

Le problème très général consiste donc à classer des chaînes de symboles en imposant certaines contraintes aux classes, et à améliorer, en fonction de l'expérience, la façon de classer.

Les premières tentatives s'inspiraient du principe du renforcement (Rosenblatt, 1958) et l'on se contentait de renforcer positivement la machine si elle mettait le *pattern* dans la bonne classe et négativement si elle le mettait dans une mauvaise, ce qui déclenchait la recherche aléatoire d'une nouvelle classe. Par le nombre d'opérations qu'elle suppose, une telle procédure tombe dans le même défaut que la procédure combinatoire dont nous avons signalé plus haut le caractère « intraitable ».

Il a donc fallu très vite doter le programme de règles heuristiques qui permettent d'économiser massivement le nombre des opérations et qui exploitent les indices perceptifs. Ainsi que le font remarquer Gyr, Brown, Willey et Zivian (1966), ces règles devraient s'inspirer de ce que l'on sait sur les activités perceptives.

Feigenbaum et Simon (1963) décrivent un programme, EPAM, qui accomplit quatre tâches :

A) Reconnaître un stimulus sur lequel la machine possède déjà quelques informations ;
B) Mettre en mémoire de nouveaux stimuli en les discriminant des anciens ;
C) Associer deux stimuli x et y en mémoire, ce qui veut dire qu'on va stocker avec x, de l'information concernant y ;
D) Répondre à X par Y en retrouvant y à partir de l'information sur y associée à x.

Ce programme comprend de l'apprentissage à deux titres : sous B où s'élabore la structure des tests de discrimination qu'il applique aux stimuli et sous C où les éléments de réponse sont associés aux stimuli. EPAM est capable d'apprendre à lire des noms d'objets, d'apprendre des paires (au sens classique, en psychologie, de l'apprentissage par paires). Voici par exemple une expérience très simple d'association par paires, où le stimulus initial est présenté oralement (d'où l'écriture) et où la machine répond en désignant l'un de quatre objets (car, ball, cat, dog).

Essais	Stimulus	Réponse d'EPAM
1	KAHR	...
	DAWG	car
	KAT	car
	BAWL	car
2	KAHR	car
	DAWG	ball
	KAT	...
	BAWL	ball
3	BAWL	ball
	KAHR	car
	KAT	cat
	DAWG	...
4	BAWL	ball
	KAHR	car
	DAWG	dog
	KAT	cat

Vossler et Uhr (1962) ont écrit un programme de reconnaissance des formes capable d'engendrer et de modifier les opérateurs par lesquels il doit transformer les chaînes de symboles de l'entrée en chaînes de symboles à la sortie : cela consiste à développer des opérateurs capables de s'attacher aux caractéristiques intéressantes, puis à abandonner les mauvais opérateurs, améliorer les bons, les combiner et les décomposer, bref, à raffiner les méthodes d'évaluation et de décision.

C'est sans doute dans cette direction que le psychologue doit trouver les éléments les plus suggestifs. L'idée sous-jacente est celle d'une hiérarchie de transformations, les premières transformant les entrées en sorties, les secondes opérant sur les premières et ainsi de suite.

IV. — Généralité de la méthode

Nous avons signalé plus haut que Newell et Simon (1963) avaient écrit un *General Problem Solver* pouvant s'attaquer à tout problème bien défini. C'est ainsi que Simon et Kotovski (1963) ont écrit un programme capable de compléter des séries (test de Thurstone).

Exemple : trouver le chiffre qu'il faut écrire à la place du point

1 1 4 7 3 3 4 6 5 5 4.

La réussite à de telles tâches suppose évidemment la découverte de la loi et le programme doit avoir les moyens de découvrir la période et les lois de série pour chaque position. Les auteurs ont mis à l'épreuve quatre programmes différents dont l'un a donné des résultats proches de ce que font des sujets humains, au moins quant à l'ordre de difficulté des séries.

Hunt et Hovland (1963) ont simulé la formation de concept, Feldman (1963) a écrit un programme très intéressant pour les choix binaires dans le cas d'une série totalement contingente, c'est-à-dire dans le type de situations où se sont illustrés les modèles stochastiques d'apprentissage. Pour cela il a fait penser à voix haute chacun de ses sujets, méthode redevenue classique avec les expériences de simulation, et il a pu ainsi se renseigner directement sur les hypothèses faites par le sujet dans chaque circonstance, c'est-à-dire selon les événements juste antérieurs et selon leur accord avec les prédictions. Il a dégagé ainsi les patterns (hypothèses) privilégiés par le sujet, par exemple :

une suite de A	AAAA
une suite de B	BBBB
alternance simple	ABAB
alternance double	AABBAA
2 A et 1 B	AABAAB
2 B et 1 A	ABBABB
3 A et 3 B	AAABBB
etc.	

et fait un sort particulier à l'attitude qui consiste à prédire le contraire de ce que donnerait le *pattern* (illusion du joueur). En écrivant dans le programme les règles pour changer de pattern en fonction des issues, il a pu simuler avec une bonne approximation la suite des prédictions d'un sujet donné (un programme par sujet). Il a notamment utilisé une procédure de « remise sur les rails » pour les cas où le programme ne donne pas la même prédiction que le sujet : c'est alors la prédiction vraie du sujet simulé qui est réinjectée dans la machine à la place de la prédiction donnée par le programme. Flood (1962) a également utilisé cette procédure de réinjection.

Sans vouloir citer toutes les idées originales en matière de simulation, nous citerons encore le programme Argus, écrit par Reitman, Grove et Shoup (1964) qui est composé d'éléments sémantiques analogues aux *cell-assemblies* de Hebb. L'avantage d'Argus sur la plupart des autres programmes est qu'il n'est pas polarisé par sa tâche mais capable de travailler en parallèle sur plusieurs tâches à la fois. Les auteurs ont voulu explorer les implications d'un certain nombre de suppositions sur l'organisation cognitive et l'activité : ils ont donc établi leur réseau d'éléments sémantiques, et défini pour chaque élément divers paramètres (activation, inhibition, seuil, grandeur de l'élément sémantique) variables dans le temps. Un système d'allocation des ressources répartit les possibilités de la machine, et des coalitions entre éléments sont possibles. Le modèle semble bien adapté pour les phénomènes de sélection des messages.

Enfin, il faut signaler que les problèmes de traduction automatique sont étroitement liés à ce que nous avons examiné dans cet article. En effet, la traduction consiste à transformer des *patterns* d'entrée organisés entre eux selon une certaine syntaxe, et reliés à un objet selon une certaine sémantique, en des *patterns* de sortie organisés également par une syntaxe et une sémantique (même objet). Comme on ne peut associer terme à terme les énoncés d'entrée et les énoncés de sortie, à cause du caractère infini du langage, il est évidemment nécessaire d'inclure, dans le programme de traduction, des règles de type heuristique et des algorithmes partiels, comme pour la solution de problème.

V. — Les langages

Un ordinateur est fondamentalement une machine à manipuler des symboles. Mac Carthy (1966) décrit ainsi les trois niveaux auxquels s'effectuent ces manipulations :

1° Les *bits* structurés par une algèbre booléenne :

2 opérations : la négation $\quad -0 = 1 \qquad -1 = 0$;
la conjonction $0,0 = 0,1 = 1,0 = 0$;
$1,1 = 1$;

1 test : savoir si un *bit* est égal à 0 ou 1.

2° Les nombres :

4 opérations : addition, soustraction, multiplication et division ;
2 tests : savoir si deux nombres sont égaux ;
savoir si un nombre est positif ou nul.

3° Les chaînes de caractères :

3 opérations : concaténation ABC * ABA = ABCABA ;
first (ABC) = A ;
Rest (ABC) = BC ;

2 tests : savoir si une chaîne est nulle ou non ;
savoir si deux caractères sont identiques ou non.

Les nombres et les caractères sont faits de *bits*, les chaînes sont faites de caractères. Comme la machine ne connaît que les *bits*, les opérations et les tests sur les nombres et les chaînes de caractères doivent être interprétées en termes d'opérations et de tests sur les *bits*. Cette interprétation se fait automatiquement et les codes de programmation (Fortran, Algol, etc.) sont interprétés en langage-machine par un programme inscrit à demeure dans la machine.

Malheureusement, ces codes de programmation comportent une limitation essentielle, celle de ne pouvoir exprimer les fonctions récursives, lesquelles figurent fréquemment dans les procédures de solution de problèmes dont nous avons parlé. Une fonction récursive est une fonction telle que l'exécution de chaque calcul est subordonnée à l'exécution de calculs du même type, et cela de proche en proche jusqu'à ce qu'on arrive à un cas où la fonction est définie par un paramètre. Or, dans ces codes de programmation usuels, on doit mentionner les adresses où sont enregistrées les données. Il a donc fallu inventer d'autres langages, appelés langages de listes, et dans lesquels l'adresse d'un symbole se trouve inscrite à l'adresse de son prédécesseur sur la liste, ce qui permet les procédures récursives. Les plus connus de ces langages sont IPL V (NEWELL), FLPL (Gelernter), LISP (Mac Carthy). La structure des listes de symboles est telle qu'on peut toutes les engendrer à partir de la liste d'ordre le plus élevé. D'autre part, les opérations suivantes sont définies : lire un symbole, écrire un symbole, copier un symbole, créer un symbole, effacer un symbole, comparer deux symboles. Ces langages ne sont pas utilisés directement sur les machines, ce sont des pseudo-codes qu'il faut traduire dans les codes usuels, ce qui est un facteur considérable de ralentissement.

VI. — Conclusions et perspectives

Dans l'enthousiasme des premiers travaux de simulation de la pensée, certains auteurs ont considéré que les programmes étaient de véritables théories psychologiques. Newell et Simon (1963) affirment que le langage IPL V est à lui seul une théorie psychologique faible, parce que générale, mais complète. Dans un autre article (1958) ils se plaisent à montrer de façon assez convaincante les analogies entre certains aspects du comportement du programme et les phénomènes psychologiques d'insight, d'attitude, de structure hiérarchique. Reitman (1966) souligne que les programmes comportent beaucoup trop de règles *ad hoc* pour mériter le nom de théories, qu'on connaît peu ou pas d'invariants de ces programmes et notamment quant au développement du savoir et à l'utilisation du savoir antérieur. Ajoutons à cela que les programmes ne simulent qu'une partie infime des processus de la pensée, ceux qui sont mis en œuvre devant des problèmes bien définis. Quant aux autres (lorsque le but est mal connu, ou les règles, ou les conditions initiales) ils sont restés pour l'instant en dehors de l'entreprise, nous en avons exposé quelques raisons. D'autres entreprises ne sont pas sans analogie

avec ce que nous avons traité dans cet article, ce sont toutes les recherches dans lesquelles on s'attache à formaliser les opérations du sujet, qu'il s'agisse des travaux genevois, de ceux de Frey (1964), de ceux de Klix (1966) ou de certaines recherches soviétiques, notamment les travaux de Landa (1966) sur l'enseignement programmé.

La simulation est sans doute un outil extrêmement puissant avec lequel les psychologues doivent se familiariser. Il n'est pas dit pour autant que la psychologie soit suffisamment avancée pour en tirer tout le profit possible.

BIBLIOGRAPHIE[1]

Étant donné que le sujet de cette note est abordé pour la première fois dans l'*Année psychologique*, nous avons jugé utile de donner en référence d'autres titres que les seuls travaux cités. La meilleure introduction est sans doute le livre collectif *Computers and thought*, édité par E. A. FEIGENBAUM et J. FELDMAN, New York, Mac Graw Hill, 1963, qui contient des articles essentiels et une bibliographie de 900 titres établie par M. L. Minsky.

ABELSON (R. P.). — Computer Simulation of « hot » cognition, *in* TOMKINS (S. S.), MESSICK (S.) (édit.), *Computer simulation of personnality*, New York, Wiley, 1963, 277-298.

ABELSON (R. P.). — Heuristic processes in the human application of verbal structures in new situations, *Symposium n° 25*, p. 5-14, Moscou, XVIII[e] Congrès de Psychologie, 1966.

ALLEN (M.). — A concept attainment program that simulates a simultaneous scanning strategy, *Behavioral Science*, 1962, 7, *n° 2*, 247-252.

AMAREL (S.). — On the automatic formation of a computer program which represents a theory, *in* YOVITS (M.), JACOBI, GOLDSTEIN (édit.), *Self-organising systems*, Spartan, 1962, 107-175.

ARMER (P.). — Attitudes towards intelligent machines, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 389-405.

BALZER (R. M.). — A mathematical model for performing a complex task in a card game, *Behavioral Science*, 1966, 9, *n° 3*, 219-226.

BANARJI (R. B.). — A formal model for concept description and manipulation, in *Symposium n° 18*, p. 66-81, XVIII[e] Congrès de Psychologie, Moscou, 1966.

BAYLOR (G. W.). — A computer model of checkmating behavior in chess, *Symposium n° 25*, p. 59-68, XVIII[e] Congrès de Psychologie, Moscou, 1966.

BOBROW (D.). — *Natural language input for a computer problem-solving system*, Cambridge, M.I.T. Press, 1964.

COLBY (K. M.). — Computer simulation of a neurotic process, *in* TOMKINS (S. S.), MESSICKS (S.), *Computer simulation of personnality*, New York, Wiley, 1963, 165-179.

ELSHOUT (J. J.) et FRIJDA (N. N.). — The origin of problem-solving methods, *Symposium n° 25*, XVIII[e] Congrès de Psychologie, Moscou, 1966.

FEIGENBAUM (E. A.). — Soviet cybernetics and computer sciences, *Communication A.C.M.*, 1961, *n° 4*, 566-579.

FEIGENBAUM (E. A.) et SIMON (M. A.). — Performance of a reading task by an elementary perceiving and memorizing program, *Behavioral Science*, 1963, 8, *n° 1*, p. 72-76.

1. Nous remercions vivement Marie-Claude Vallet-Gardelle, du Service de Documentation du Laboratoire de Psychologie sociale, ainsi que le Service de Documentation de l'Institut Blaise-Pascal, pour l'aide qu'ils ont bien voulu nous apporter dans nos recherches bibliographiques.

FEIGENBAUM (E. A.). — The simulation of verbal learning behavior, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 297-309.

FEIGENBAUM (E. A.) et FELDMAN (J.). — *Computers and thought*, New York, Mac Graw Hill, 1963.

FELDMAN (J.). — Computer simulation of cognitive processes, *in* BORKO (H.), *Computer applications in the behavioral sciences*, Prentice Hall, Englewood Cliffs, 1962.

FELDMAN (J.). — Simulation of behavior in the binary choice experiment, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 329-346.

FLOOD (M. M.). — Stochastic learning theory applied to choice experiments with rats, dogs and men, *Behavioral Science*, 1962, 7, *n° 3*, 289-314.

FLOOD (M. M.). — What future is there for intelligent machines ?, *A. V. Communication Review*, 1963, 11, *n° 6*, 260-270.

FREY (L.). — Sériation et transitivité, *Cahiers de Psychologie*, 1964, 7, *n° 4*, 143-157.

GAINES (R. S.). — On the translations of machine language programs, *Com. A.C.M.*, 1965, 8, *n° 12*, 736-741.

GELERNTER (H.). — Realization of a geometry-theorem proving machine, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 134-152.

GELERNTER (H.), HANSEN (J. R.) et LOVELAND (D. W.). — Empirical explorations of the geometry-theorem proving machine, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 153-163.

GREEN (B. F. Jr). — Computer models of cognitive processes, *Psychometrika*, 1961, 26, *n° 1*, 85-91.

GREEN (B. F. Jr), WOLF (A. K.), CHOMSKY (C.) et LAUGHERY (K.). — Baseball : an automatic question answerer, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 207-216.

GREENE (P. M.). — An approach to computers that perceive, learn and reason, *Proceedings Western Joint Conference*, 1959, San Francisco, 181-186.

GREENE (P. M.). — A suggested model for information representation in a computer that perceives, learns and reasons, *Proceedings Western Joint Conference*, 1960, San Francisco, 151-164.

GUITTET (J.). — Physionomie et classement des techniques de simulation, *Metra*, 1966, 5, *n° 2*, 191-200.

GYR (J. W.), BROWN (J. S.), WILLEY (R.) et ZIVIAN (A.). — Computer simulation and psychological theories of perception, *Psychological Bulletin*, 1966, 65, *n° 3*, 174-192.

HORMANN (A.). — Gaku, An artificial student, *Behavioral Science*, 1965, 10, *n° 1*, 88-107.

HOVLAND (C. I.). — Computer simulation of thinking, *Amer. Psychologist*, 1960, 15, *n° 11*, 704-712.

HUNT (E. B.). — *Concept learning : an information processing problem*, New York, Wiley, 1962.

HUNT (E. B.) et HOVLAND (C. I.). — Programming a model of human concept formulation, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 310-325.

JOHNSON (E. S.). — An information-processing model of one kind of problem-solving, *Psychol. Monograph*, 1964, 78, *n° 4*, 31 p.

KLIX (F.) et SYDOW (M.). — Information processing in problem-solving behavior, in *Symposium n° 18*, p. 53-55, XVIII[e] Congrès de Psychologie, Moscou, 1966.

KOCHEN (M.). — Experimental study of « hypothesis formation » by computer, *in* CHERRY (C.), *Proceedings of the 4th London Symposium on information theory*, Londres, Butterworth, 1961, 21-32.

KRULEE (G. K.), KUCK (D. J.), LANDI (D. M.) et MANELSKI (D. M.). — Natural language inputs for a problem-solving system, *Behavioral Science*, 1964, 9, *n° 3*.

LANDA (L. N.). — A logico-mathematical approach to the logical structure of some mental processes, *Symposium n° 12*, p. 40-50, XVIII^e Congrès de Psychologie, Moscou, 1966.

LAUGHERY (K. R.) et GREGG (L. W.). — Simulation of human problem-solving behavior, *Psychometrika*, 1962, *27*, 265-282.

LINDSAY (R. K.). — Inferential memory as a basis of machines which understand natural language, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 217-233.

MCCARTHY (J.). — Programs with common sense, in *Mechanisation of thought processes*, Londres, Her Majesty's stationery office, 1959, vol. I, 75-84.

MCCARTHY (J.), ABRAHAM (P. W.), EDWARD (D. J.), HART (T. P.) et LEVIN (M. I.). — *LISP 1.5 Programmer's manual*, Cambridge, M.I.T. Press, 1962.

MCCARTHY (J.). — Information, *Scientific American*, sept. 1966, 215, *n° 3*, 64-73.

MACKAY (D. M.). — Operational aspects of intellect, in *Mechanisation of thought processes*, Londres, Her Majesty's stationery office, 1959, vol. I, 37-54.

MILLER (G. A.), GALANTER (E.) et PRIBRAM (K.). — *Plans and the structure of behavior*, New York, Holt, 1960.

MINSKY (M. L.). — Some methods of artificial intelligence and heuristic programming, in *Mechanisation of thought processes*, Londres, Her Majesty's stationery office, 1959, vol. I, 3-27.

MINSKY (M. L.). — Steps toward artificial intelligence, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 406-450.

MINSKY (M. L.). — A selected descriptor-indexed bibliography to the literature on artificial intelligence, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 453-523.

MINSKY (M. L.). — Artificial intelligence, *Scientific American*, sept. 1966, 215, *n° 3*, 247-260.

NAPALKOV (A. V.). — Algorithmic analysis of complex mental activity, in *Symposium n° 18*, p. 61-65, XVIII^e Congrès de Psychologie, Moscou, 1966.

NEISSER (V.). — The imitation of man by machine, *Science*, 1963, *139*, 193-197.

NEWELL (A.), SHAW (J. C.) et SIMON (H. A.). — Elements of a theory of human problem solving, *Psychological Review*, 1958, 65, *n° 3*, 151-166.

NEWELL (A.), SHAW (J. C.) et SIMON (H. A.). — *(a)* The processes of creative thinking, *in* GRUBER (H. E.), TERRELL (G.), WERTHEIMER (M.), *Contemporary approaches to creative thinking*, New York, Atherton Press, 1963, 63-119.

NEWELL (A.), SHAW (J. C.) et SIMON (H. A.). — *(b)* Chess-playing programs and the problem of complexity, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 39-70.

NEWELL (A.), SHAW (J. C.) et SIMON (H. A.). — *(c)* Empirical explorations with the logic theory machine : a case study in heuristics, *in* FEIGENBAUM (E. A.), FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 109-133.

NEWELL (A.) et SIMON (H. A.). — Computers in psychology, *in* LUCE (R. D.), BUSH (R. R.) et GALANTER (E.), *Handbook of mathematical psychology*, 1963, vol. I, New York, Wiley, 361-428.

NEWELL (A.) et SIMON (H. A.). — GPS, a program that simulates human thought, *in* FEIGENBAUM (E. A.) et FELDMAN (J.), *Computers and thought*, New York, Mac Graw Hill, 1963, 279-293.

NEWELL (A.). — *Information processing language V manual*, Prentice Hall, Englewood Cliffs, 1961.

NEWELL (A.). — Some problems of basic organization in problem-solving programs, *in* YOVITS (M.), JACOBI et GOLDSTEIN (édit.), *Self-organizing systems*, Spartan, 1962, 393-423.

NEWELL (A.). — Human problem-solving, in *Psychological applications of decision theory*, La Haye, Nuffic, 1964, 179-200.

PITRAT (J.). — *Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques*, thèse de la Faculté des Sciences, Université de Paris, 1966.

POLYA (G.). — *How to solve it*, Princeton, Princeton N.J., 1954.

PUSHKIN (V. N.) et ZAVALISHINA (D. N.). — The psychology of human heuristic activity and some problems of the theory of automations, *Symposium n° 25*, p. 73-76, XVIII^e Congrès de Psychologie, Moscou, 1966.

REITMAN (W. R.). — Some soviet investigation of thinking, problem-solving and related areas, *in* BAUER (R. A.), *Some views on soviet psychology*, New York, American Psychological Association, 1962, 29-61.

REITMAN (W. R.). — Personality as a problem-solving coalition, *in* TOMKINS (S. S.) et MESSICK (S.) (édit.), *Computer simulation of personnality*, New York, Wiley, 1963, 69-99.

REITMAN (W. R.), GROVE (R. B.) et SHOUP (R. G.). — Argus : an information processing model of thinking, *Behavioral Science*, 1964, 9, *n° 3*, 270-281.

REITMAN (W. R.). — Information processing models in psychology, *Science*, 1964, *144*, 1192-1198.

REITMAN (W. R.). — *Cognition and computer simulation*, New York, Wiley, 1965.

REITMAN (W. R.). — The study of heuristics, *Symposium n° 25*, p. 44-53, XVIII^e Congrès de Psychologie, Moscou, 1966.

RENARD (B.) et RENAULT (J.-P.). — La simulation et les calculateurs, *Metra*, 1966, 5, *n° 2*, 201-205.

ROCHESTER (N.), HOLLAND (J. M.), HAIBT (L. H.) et DUDA (W. L.). — Test on a cell assembly theory of the action of the brain, using a large digital computer, *I.R.E. transaction on information theory*, 1956, *n° 2*, 80-93.

ROSENBLATT (F.). — The perception : a probabilistic model for information storage and organization in the brain, *Psychological Review*, 1958, *65*, 386-407.

SAMUEL (A. L.). — Some studies in machine learning using the game of checkers, *in* FEIGENBAUM (E. A.) et FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, p. 71-105.

SELFRIDGE (O. G.). — Pandemonium : a paradigm for learning, in *Mechanisation of thought processes*, Londres, Her Majesty's stationery office, 1959, vol. I, 511-526.

SELFRIDGE (O. G.) et NEISSER (V.). — Pattern recognition by machine, *in* FEIGENBAUM (E. A.) et FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, 237-250.

SIMON (H. A.) et KOTOVSKY (K.). — Human acquisition of concepts for sequential patterns, *Psychological Review*, 1963, 70, *n° 6*, 534-546.

SLAGLE (J. R.). — A heuristic program that solves symbolic integration problems in freshman calculus, *in* FEIGENBAUM (E. A.) et FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, 191-203.

SLAGLE (J. R.). — Experiments with a deductive question-answering program, *Com. A.C.M.*, 1965, 8, *n° 12*, 792-798.

STEINBUCH (K.). — Technical models of human information processing, in *Symposium n° 18*, p. 35-52, XVIII^e Congrès de Psychologie, Moscou, 1966.

STRACHEY (C.). — System analysis and programming, *Scientific American*, sept. 1966, 215, *n° 3*, 112-124.

SUPPES (P.). — The uses of computers in education, *Scientific American*, sept. 1966, 215, *n° 3*, 206-220.

TOMKINS (S. S.) et MESSICK (S.). — *Computer simulation of personality*, New York, Wiley, 1963.

TONGE (F. M.). — Summary of a heuristic line balancing procedure, *in* FEIGENBAUM (E. A.), FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, p. 168-190.

TRAHTENBROT (B. A.). — *Algorithmes et machines à calculer*, Paris, Dunod, 1963.

TRAVIS (L. E.). — The value of introspection to the designer of mechanical problem-solvers, *Behavioral Science*, 1963, 8, *n° 3*, 227-233.

TURING (A. M.). — Computing machinery and intelligence, *in* FEIGENBAUM (E. A.) et FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, p. 11-35.

UHR (L.). — The development of perception and language. Simulated models, *in* TOMKINS (S. S.) et MESSICK (S.), *Computer simulation of personality*, New York, Wiley, 1963, 231-266.

UHR (L.). — « Pattern-recognition » computers as models for form-perception, *Psychological Bulletin*, 1963, n° 1, 40-73.

UHR (L.) et VOSSLER (C.). — A pattern-recognition program that generates, evaluates and adjusts its own operators, *in* FEIGENBAUM (E. A.) et FELDMAN (J.) (édit.), *Computers and thought*, New York, Mac Graw Hill, 1963, p. 251-268.

UHR (L.). — Pattern-string learning programs, *Behavioral Science*, 1964, 9, *n° 3*, 258-270.

Von NEUMAN. — *The computer and the brain*, New Haven, Yale University Press, 1958.

VOSSLER (C.) et UHR (L.). — Computer simulation of a perceptual learning model for sensory pattern recognition, concept formation and symbol transformation, *Information Processing 1962*, Proceedings of I.F.I.P. Congress 62, North Holland, 1962.

WANG (H.). — Toward Mechanical mathematics, *I.B.M. Journal of Research and Development*, 1960, 4, *n° 1*, 2-22.