EURISKO (D.B.Lenat)

*Dominique P*ASTRE

1. Introduction

Des difficultés rencontrées par AM, dues à l'impossibilité de créer de nouvelles heuristiques ou de nouvelles facettes, Lenat en a déduit la nécessité de traiter les heuristiques comme les concepts et a développé un nouveau système : EURISKO [Lenat 78, 82a, 82b, 83a, 83b].

EURISKO manipule des *unités* qui peuvent être, entre autres, des concepts ou des heuristiques et qui sont composés de facettes ou a attributs.
EURISKO sera appliqué au domaine mathématique de AM et à d'autres domaines : wargame, VLSI (circuits intégrés), programmation.

"Our original 1976 assumption was that heuristics could be treated just like math concepts, and we would apply the same methods (heuristic search) to discover new ones. But we were fortunate in choosing elementary mathematics as the test domain for AM; heuristics are like most other domains, it's mathematics that's special and (thanks to LISP) particularly easy." [Lenat 83b page 97]

2. Représentation des heuristiques

Chaque heuristique est une unité particulière, qui peut être dans l'attribut *heuristiques* d'un concept. Il y a aussi l'unité *heuristiques*, et les attributs sont aussi des unités, soit, pour une heuristique h, un concept C et un attribut a, les unités suivantes :

heuristiques	h	C	a
...	...	...	...
Exemples : h	Est-un : heuristique	heuristiques : h	
		a : ...	

Une heuristique peut manipuler (créer) une unité quelconque, donc en particulier créer de nouvelles heuristiques et de nouveaux attributs.
La différence entre une heuristique et une métaheuristique est seulement que la première est appliquée à une unité qui est un concept et la seconde à une unité qui est une heuristique.
D'où pas de différence, mais … EURISKO a séparé les heuristiques des métaheuristiques car, quand une unité a trop d'exemples, elle est éclatée en cherchant des ressemblances entre les exemples.

Au début, il y a deux attributs : si : <fonction LISP>
alors : <fonction LISP>

Puis elle subit des transformations, en particulier des créations d'attributs :

si_identité	alors_conjecture
si_constante	alors_écrire
si_inchangé	...
…	

en fonction des fréquences avec lesquelles on les rencontre.

L'intérêt est que l'on va pouvoir travailler sur ces attributs comme sur les attributs des autres unités par des règles du genre

si pour beaucoup d'unités les attributs r et s …
alors créer une nouvelle unité (concept, heuristique, attribut, …)

3. Un exemple d'heuristique et sa représentation

Généraliser un prédicat rarement vrai

[R8 dans Lenat 83a page 43, Lenat 82 page 230, Pitrat 85 page 55]

L'application de cette heuristique à l'égalité des multi-ensembles conduit à la création du concept même-longueur (qui conduira au concept de nombre).
Son application à l'égalité d'ensembles conduit à la création des concepts d'inclusion, de même-cardinal et de même-premier-élément (ce dernier concept étant sans intérêt).

NAME: Generalize-rare-predicate
ABBREVIATION: GPR
STATEMENT
English: If a predicate is rarely true, Then create generalizations of it
IF-just-finished-a task-dealing with: a predicate P
IF-about-to-work-on-task-dealing-with: an agenda A
IF-truly-relevant : P returns true less than 5% of Average Predicate
IF-resources-available: at least 10 cpu seconds, at least 300 cells
THEN-add-task-to-agenda: Fill in entries for Generalizations slot of P
THEN-conjecture: P is less interesting than expected
Generalizations of P may be better than P
Specializations of P may be very bad
THEN-modify-slots: Reduce Worth of P by 10%
Reduce Worth of Specializations(P) by 50%
Increase Worth of Generalizations(P) by 20%
THEN-print-to-user: English (GPR) with "a predicate" replaced by P
THEN-define-new-concepts:
CODE-IF-PART: λ(P) <LISP function>
CODE-THEN-PART: λ(P) <LISP function>
SPECIALISATIONS: Generalize-rare-set-predicat
Boundary-specializations: Enlarge-domain-of-predicate
GENERALIZATIONS: Modify-predicate, Generalize-concept
Immediate-Generalizations: Generalize-rare-contingent-piece-of-knowledge

Siblings: generalize-rare-heuristic
IS-A: Heuristique
EXAMPLES:
Good-Examples: Generalize Set-Equality into Same-Length
Bad-Examples: Generalize Set-Equality into Same-First-Element
CONJECTURES: Special cases of this are more powerful than Generalizations
Good-Conjec-Units: Specialize, Generalize
ANALOGIES: Weaken-overconstrained-problem
WORTH : 600
ORIGIN: Specialization of Modify-predicate via empirical induction
Defined-using: Specialize
Ceation-date: 6/1/78
HISTORY:
NGoodExamples: 1 NBadExamples: 1
NGoodConjectures: 3 NBadConjectures: 1
NGoodTasks-added: 2 NBadTasks-added: 0
AvgCpuTime: 9.4 seconds AvgListCells: 200

4. Autres exemples heuristiques et applications

4.1. Faire coïncider des éléments

Si f est une fonction intéressante qui a deux arguments de même type alors définir et étudier la fonction les fusionnant g(a)=f(a,a)

[R7 dans Lenat 83a page 40, Pitrat 85 page 61]

Applications aux mathématiques :

EURISKO s'intéresse à	l'*intersection*	et conjecture	$a \cap a = a$
	la *réunion*	"	$a \cup a = a$
	l'*appartenance*	"	$a \notin a$
	la *soustraction*	"	$x - x = 0$
	le *quotient*	"	$x / x = 1$
	l'*addition*	et introduit	le *double*
	la *multiplication*	"	le *carré*

D'une manière générale EURISKO définit aussi de nouvelles propriétés de f selon que l'on a f(a,a) toujours vrai (par exemple a=a, x divise x), toujours faux (par exemple $a \notin a$), égal à a (par exemple $a \cap a=a$) ou constant (par exemple a/a=1)

Application à la bataille navale ("wargame") :
EURISKO a participé à une compétition annuelle de bataille navale basée sur un grand nombre de règles et un simulateur gérant des batailles entre deux flottes. EURISKO a créé des bateaux *à la fois offensifs et transporteurs, et équipés pour la mise à l'eau*. Ces bateaux pouvaient donc se transporter eux-même vers la zone de combat. Cette tactique, bizarre mais possible avec les règles imposées a été très puissante. Les organisateurs du jeu ont alors modifié les règles, interdisant ainsi de tels bateaux.
Les règles imposaient qu'une partie de la flotte reste à l'arrière, loin de la bataille, pour servir de

bateaux ravitailleurs. EURISKO a alors créé les bateaux *à la fois offensifs et ravitailleurs*. Quand ces bateaux étaient modérément (mais non totalement) endommagés au front, ils prenaient place en arrière avec les ravitailleurs. Cette manœuvre était explicitement permise dans les règles, mais personne ne l'utiliserait, sauf en cas de situation désespérée à la fin d'une bataille presque nulle. En raison de la puissance de cette manœuvre, les règles furent de nouveau changées de façon à ce que ce passage vers l'arrière ne puisse pas être aussi rapide.
Les modifications ultérieures de règles ont introduit plus de créneaux qu'elles n'en n'éliminaient, permettant par exemple, de créer un bateau qui, endommagé, se sabordait et coulait pour ne pas diminuer l'agilité globale de la flotte. EURISKO s'est ainsi montré un très bon détecteur de failles.

Application de cette heuristique à des domaines moins techniques :
- l'utilisation des préfixes *auto-* et *self-* dans le langage courant ;
- dans l'intrigue d'un scénario policier, la victime peut être le criminel (escroquerie à l'assurance), ou bien le policier peut être le criminel ou la victime.

Lenat donne aussi des applications à l'informatique (compilation) et aux circuits intégrés (VLSI).

4.2. Inverser les cas extrêmes

Si f est une fonction connue et intéressante, et b un sous-ensemble connu, intéressant et extrême de l'ensemble image, alors définir et étudier $f^{-1}(b)$

[R9 dans Lenat 83a page 44, Pitrat 85 page 65]

Applications aux mathématiques :
- f = *intersection* et b = *ensemble vide* conduisent au concept d'*ensembles disjoints*
  b = *singletons* conduisent aux paires d'ensembles qui ont exactement un élément commun, concept utile dans certaines constructions mathématiques
- f = *diviseurs-de* et b = *ensemble vide* ne donnent rien
  b = *singletons* donnent uniquement 1, sans intérêt
  b = *paires* conduisent aux *nombres premiers*
  b = *ensembles à 3 éléments* conduisent aux carrés, déjà découverts par ailleurs
  b = *ensembles dont le nombre d'éléments est premier* conduisent aux nombres de la forme p^{q-1} avec p et q premiers

Applications à la bataille navale :
EURISKO a créé un minuscule bateau non offensif mais incroyablement agile, donc incoulable. Les règles du jeu ont alors été modifiées pour rendre ceci impossible.
EURISKO a alors créé un autre type de bateau extrême avec une capacité offensive maximale, sans capacité défensive.

Application à des domaines moins techniques :
Si f est l'application qui à un individu associe l'ensemble de ses emplois, on obtient d'une part les chômeurs (qui n'ont pas d'emploi), et d'autre part les personnes ayant plus d'un emploi, puisque la plupart des individus ont exactement un emploi.

Cette heuristique a aussi été utilisée dans la conception des circuits intégrés et en programmation.

EURISKO crée aussi des concepts sans intérêt, mais qui seront éliminés par la suite.

4.3. Analogie et multi-définition

[R17 et R18 dans Lenat 83a page 52]

S'il y a une forte analogie entre des concepts A et B, mais il y a une conjecture "Pour tout b dans B …" dont l'analogue dans A est fausse alors définir le sous-ensemble de A pour lequel on a la conjecture analogue.

Si un concept a un complément (négation) qui est beaucoup plus petit (plus rare), alors définir explicitement et nommer ce complément.

Applications

Il y a une grande analogie entre l'addition et la multiplication (loi de composition interne et commutative avec élément neutre). Tout nombre plus grand que 1 peut être exprimé comme la somme de deux nombres plus petits, mais il n'est pas vrai que tout nombre (même plus grand que 2 par exemple) puisse être exprimé comme le produit de deux nombres plus petits. R17 conduit alors à définir le sous-ensemble des nombres pour lesquels c'est vrai c'est-à-dire les nombres composés.

Comme les nombres composés sont beaucoup plus nombreux que ceux qui ne le sont pas, R18 conduit à définir ceux qui ne le sont pas, et on s'aperçoit que ce sont les mêmes que les nombres premiers définis grâce à R9 vue plus haut. C'est donc une deuxième façon de définir les nombres premiers.

5. Créations de nouvelles heuristiques

L'application d'une heuristique générale à un domaine particulier donne des heuristiques pour ce domaine.

5.1. Co-identification

Si deux attributs s1 et s2 d'une structure F peuvent avoir le même type alors définir une nouvelle heuristique : *si f est un F intéressant et ses attributs s1 et s2 sont de même type* *alors définir et étudier la situation dans laquelle les valeurs des attributs s1 et s2 de f sont égales.*

[R22 dans Lenat 83a page 54, Pitrat 85 page 66]

Applications aux mathématiques :
- avec F = fonctions
  - $s1$ = 1er argument
  - $s2$ = 2ème argument

on obtient l'heuristique R7 vue ci-dessus (4.1) :
Faire coïncider des éléments

- avec F = fonctions
  s1 = argument
  s2 = valeur
on obtient l'heuristique suivante :
  Les points fixes d'une fonction sont intéressants.

- avec F = fonctions
  s1 = ensemble de départ
  s2 = ensemble d'arrivée
on obtient l'heuristique suivante :
  Une fonction d'un ensemble dans lui-même est intéressante.

5.2. Extrêmes

Si f(Exemples(A), Exemples(B)) est presque extrême alors combiner les définitions de A et B pour *obtenir une nouvelle définition*

[R25 dans Lenat 83a page 56]

Applications aux mathématiques :

- avec f = différence symétrique
  extrême = petit
  combiner = prendre l'intersection
on obtient l'heuristique :
  Si quelques uns (mais non la plupart) des exemples de A sont aussi des exemples de B
  et quelques uns (mais non la plupart) des exemples de B sont aussi des exemples de A
  alors définir et étudier l'intersection de A et B, ce nouveau concept est une spécialisation de A et de B et est défini en combinant leurs définitions.

- avec f = différence
  extrême = petit
  combiner = x, y $\rightarrow$ x $\wedge$ $\neg$ y
on obtient l'heuristique :
  Si seulement quelques exemples de A ne sont pas exemples de B
  alors définir et étudier le concept être A et non B.

6. Un exemple d'heuristique qui peut s'appliquer à elle-même

[R26 dans Lenat 83a page 56]

H : si s1 et s2 sont des attributs remplis par des valeurs de même type, et si s1(A) est plus intéressant que s2(A) alors qu'habituellement les valeurs de s1 sont moins intéressantes que celles de s2 alors définir et étudier une nouveau concept A' similaire à A sauf que s2(A')=s1(A)

- avec A = un concept
  s1 = non exemples
  s2 = exemples

on obtient l' heuristique :

Si les non-exemples de A sont plus intéressants que ses exemples
alors définir et étudier le concept ont les exemples sont les non(exemples de A).

C'est la deuxième règle R18 vue au paragraphe 4.3 qui conduit à une nouvelle définition des nombres premiers.

- avec A = cette heuristique H elle-même
  s1 = si assez de temps
  s2 = si assez intéressant

on obtient l'heuristique H' obtenue en ajoutant à l'attribut *si-intéressant* la clause *il y a assez de temps*
H' est plus utile que H car moins coûteuse (mais en général l'intérêt est plus important que le temps) et l'intérêt de H' va augmenter tandis que celui de H va diminuer.

7. EURISKO peut découvrir une heuristique qui aurait été bien utile à AM.

[Lenat 82 page 233]

L'exemple suivant montre comment EURISKO peut découvrir une heuristique qui aurait été bien utile à AM, à propos des nombres premiers, à savoir que

La factorisation unique en nombres premiers (conjecture A) est plus intéressante que la conjecture de Goldbach (conjecture B).

La raison, en gros, en est que A a à voir avec la multiplication et la division, que B a à voir avec l'addition et la soustraction, et que le concept *nombre-premier* est lié à la multiplication et la division.

Sans rentrer dans les détails, on va voir que cette découverte est accélérée par la représentation des heuristiques et des attributs sous forme d'unités, et par la possibilité de créer de nouveaux attributs.

L'attribut *défini-par* du concept *nombre-premier* contient le concept *diviseurs*, car les nombres premiers ont été définis comme les nombres ayant exactement deux diviseurs.
On va s'apercevoir d'autre part que l'attribut *défini-par* est un sous-attribut de l'attribut *unités-pour-bonnes-conjectures*, en appliquant, avec r = *défini-par*, s = *unités-pour-bonnes-conjectures*, l'heuristique suivante :

si pour beaucoup d'unités, la plupart des éléments de l'attribut r sont aussi sur l'attribut s, alors affirmer que r est un sous-attribut de s

[H19 dans Lenat 82 page 235]

Cette relation, *r sous-attribut de s* ($r \subset s$), est représentée de la façon suivante : l'attribut *super-attribut* de l'unité *r*, (ici *défini-par*) contient *s* (ici *unités-pour-bonnes-conjectures*)·
Ceci signifie alors :

si un concept X est défini en termes de Y (c'est-à-dire l'attribut défini-par *de X contient Y),*
alors on peut s'attendre à ce qu'il y ait une bonne conjecture entre X et Y
(c'est-à-dire l'attribut unités-pour-bonnes-conjectures *de X contient Y).*

On en déduit alors que l'attribut *unités-pour-bonnes-conjectures* du concept *nombre-premier* contient le concept *diviseurs*.
Puis, pour beaucoup de concepts, on va créer l'attribut *relations-entre-éléments* de unités-pour-bonnes-

conjectures, qui contient *inverse*, *répétition*, *composition*, en appliquant, avec s = *unités pour bonnes conjectures*, l'heuristique suivante :

> *si un attribut s est important, et tous ses éléments sont des unités,*
> *alors créer un nouvel attribut qui contient toutes les relations entre éléments de s.*

[H22 dans Lenat 82 page 238]

Ensuite, l'heuristique suivante:

> *si pour beaucoup d'unités, l'attribut s contient les mêmes valeurs,*
> *alors ajouter ces valeurs à l'attribut* éléments-attendus *de l'unité* attribut-typique

[H26 dans Lenat 82 page 239]

appliquée avec s = *relations-entre-éléments* de unités-pour-bonnes-conjectures, permet d'essayer, entre autres, *inverse* comme élément de l'attribut *relations-entre-éléments* de unités-pour-bonnes-conjectures du concept *nombre-premier*.

Alors, puisque *diviseurs* est dans 1'attribut *unités-pour-bonnes- conjectures* de *nombre-premier*, son *inverse*, c'est-à-dire la *multiplication*, est aussi dans l'attribut *unités-pour-bonnes-conjectures* de *nombre-premier*, ce qui signifie qu'on va plutôt s'intéresser aux conjectures entre *nombre-premier* et la *multiplication*, c'est-à-dire à la *factorisation-unique-en-nombres-premiers*.

8. Quelques gags

La règle qui indique qu'elle est l'auteur des concepts intéressants a vu son intérêt devenir anormalement élevé.

La règle destructrice qui dit que toutes les règles sont mauvaises, et les élimine, s'est heureusement éliminée.

9. Résultats

9.1. En **mathématiques** EURISKO a eu les mêmes résultats que AM, en démarrant avec les conceps de AM + 50 heuristiques.

AM/EURISKO a découvert des concepts mathématiques. Qu'est-ce que cela signifie ? Lenat reconnaît lui-même qu'AM est très lié à LISP, "AM was actually not walking around in the space of mathematical concepts, it was walking around in the space of 'small LISP predicates' ", ce qui explique ses succès tant que les *codes* LISP sont courts.

A titre d'exemple, essayons de résumer la découverte du concept *nombre*.

AM/EURISKO connait initialement des notions ensemblistes élémentaires, en particulier le concept d'*ensemble* et 1'*ensemble vide* qui sert à le définir *récursivement* : un *ensemble* est soit 1'ensemble vide ({} ou Φ), soit une *structure non ordonnée* (concept initial également) telle que, si on lui retire un élément, ce qui reste est un *ensemble*.

De là, AM/EURISKO construit des exemples d'ensembles, soient {Φ} renommé 1 par D.B.Lenat, {Φ,1} renommé 2, {Φ,1,2} renommé 3, ... C'est la construction des ordinaux finis.

Des concepts *égalité*, puis *même longueur*, on découvre qu'il y a isomorphisme entre ces ensembles et tous les autres ensembles, d'où la définition de l'ensemble des structures canoniques (notion de cardinal) renommé *ensemble des nombres*. Plus que la notion de nombre qui sert à compter, c'est cet isomorphisme qui a été découvert.

EURISKO a passé la moitié de son temps à retrouver les résultats de AM, le reste du temps découvrir des concepts inintéressants, mais aussi à produire de nouveaux attributs et de nouvelles heuristiques [Lenat 83b page 83]. Par exemple, les nouveaux attributs *si-constant*, *si-identité*, *si-inchangé* et *alors-conjecture* et la nouvelle heuristique :

Si une fonction inverse doit être utilisée, même une seule fois
alors cela vaut la peine de chercher un algorithme rapide pour la calculer.

Cette heuristique sera appliquée, par exemple pour la factorisation qui est l'inverse du produit généralisé.

9.2. Dans le domaine des **circuits intégrés** (VLSI), EURISKO a eu de très bons résultats. Il faut remarquer que c'est un domaine nouveau pour l'homme. [Lenat 82 page 243, Lenat 83b page 88].

9.3. Dans les compétitions de "**wargame**", EURISKO a eu de très bons résultats, il a gagné le tournoi en 1981 et 1982, ce qui a amené à chaque fois les organisateurs à changer des règles du jeu. En effet, les règles n'étaient pas toujours conformes au monde réel, ce qui ne gênait pas EURISKO contrairement aux êtres humains.

A partir des règles imposées pour le tournoi, les concurrents doivent concevoir une flotte qui jouera contre les autres concurrents. EURISKO a d'abord joué des batailles contre lui-même et analysé les résultats. Il augmentait alors les poids des caractéristiques conduisant au succès. Par exemple, au cours d'une bataille, tous les bateaux ont été coulés sauf un bateau de sauvetage. Cette situation a montré l'intérêt d'avoir un très petit bateau très agile. La machine tournait toute seule toute la nuit. Lenat regardait ce qu'elle avait fait et sélectionnait les concepts qu'il jugeait utiles. Lenat estime que la réussite est due à 60% pour lui-même et à 40% pour EURISKO, mais que ces 40% ont été indispensables. Lenat souligne que ni lui ni EURISKO n'aurait gagné tout seul. [Lenat 82 page 241, 83a page 42, 83b page 73].

Bibliographie

[Lenat 76] - D.B.Lenat, An artificial intelligence approach to discovery in mathematics as heuristic search, SAIL AIM-286. A.I.Lab. Stanford University (1976)

[Lenat 77] - D.B.Lenat, Automated theory formation in mathematics, IJCAI, Cambridge (1977), 833-842

[Lenat 78] -. D.B.Lenat, The ubiquity of discovery, Artificial intelligence 9(1978), 257-285

[Lenat 82a] - D.B.Lenat, The nature of heuristics, Artificial intelligence 19(1982), 189-249

[Lenat 82b] - R.Davis, D.B.Lenat, Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, 1982

[Lenat 83a] – D.B.Lenat, Theory formation by heuristic search: the nature of heuristics II, background and examples, Artificial intelligence 21(1983), 31-59

[Lenat 83b] - D.B.Lenat, EURISKO: a program that learns new heuristics and domain concepts, Artificial Intelligence 21(1983), 61-98

[Pitrat 85] - J.Pitrat, Utilisation de connaissances déclaratives, Cours fait à l'Ecole Internationale d'Informatique de l'AFCET, Aix-en-Povence, Juillet 1985, Publication du GR 22 n° 56