EURISKO: A Program That Learns New Heuristics and Domain Concepts (1982)

Tom Rochette <tom.rochette@coreteks.org>

July 24, 2025 — daae079c

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- How are functions/programs represented within EURISKO?

1 Overview

- EURISKO makes use of IsA/Examples/AllIsA (generalization and reciprocal relations)

1.1 Limitations of EURISKO

- EURISKO requires a good amount of bootstrapping code/configuration
- Algorithms used within slots are not open/available for editing by EURISKO

2 Quotes

> My take on the Eurisko project is that it was a brilliant and early effort to adapt evolutionary algorithms to a complex, adaptive ecosystem in the form of software. I say complex, adaptive because the agenda mechanism implemented feedback and decay; there was a threshold below which Eurisko would leave one agenda and start messing on another, but any task from any agenda could just as easily place rewards back to any agenda, meaning the entire system was dynamic.
>
> Like AM, Eurisko was about messing with slots in frames. AM was written in Lisp, which turned out to be problematic in the sense that claims made for AM's performance were difficult to assess; Lenat did a post-doc with Herbert Simon, from which RLL – a representation language language was hatched to isolate Eurisko from Lisp; a kind of early DSL for discovery systems, perhaps.

Jack Park (2015-02-20)

Source: https://www.quora.com/Has-Douglas-Lenats-EURISKO-research-ever-been-reproduced

3 Notes

- Automated Mathematician (AM) was an automatic programming system, whose primitive actions were modifications to pieces of LISP code, code which represented the characteristic functions of various math concepts

3.1 1 Design Decisions in Constructing the EURISKO Program

- The EURISKO project was first conceived in 1976
- Over 6 years, there has been an accumulation of "design ideas" which have been tested

3.2 1.1 Ideas about representing concepts

3.2.1 Rules need not distinguish "slots" from "functions"

- EURISKO basic representation employs frames (units) with slots
- Each slot can be viewed as a unary function which is handed a unit-name and returns a value
- Other unary functions exist and can be defined in terms of these more primitive slots
- The decision about which functions are implemented as primitive (slots), and which are computed dynamically from others, is invisible to the rules, and may change from time to time (e.g., after a great amount of experience is accumulated in some domain, it may be apparent that a certain function is requested so often that it should be stored primitively)
- All a "concept" is is a legal argument for a list of functions (mostly unary ones)

3.2.2 "GET" knows why it's being called, "PUT" knows how the value is justified

- GET Cf: get the value stored in slot f of concept C
- f(C): GET Cf
- Different reasons to query for f(C): is there any value, what is the length of the set of values, what are some of the values
- Due to the limitation on the amount of available resources to expend (time or space or number of queries to the human user), an extra argument was added to each call on GET
- This argument would specify the reason behind the call (Existence, Length, Some, Up-to-date) and how much resources can be spent (Time, Cells, Queries)
- Calls on PUT are more standard; they may trigger some flurry of re-writing, but the only extra argument one wishes to supply is an indication of the justification of the value being changed

3.2.3 "The size of ships" can mean different things, and there should be a place for each

- Each meaning should be unambiguously representable

3.2.4 Each kind of slot has a unit describing it

- Each kind of slot should "know" what it means to have a value stored on itself
- Should we redundantly cache (store) this value, or just assume we'll recompute it whenever we need it?
- The optimal answer will depend on how the knowledge base grows, changes and is used
- When an entry is added or removed from an IsA slot, we expect the "inverse link" to be likewise added or removed

3.3 1.2 Ideas about control (agendae, reasons, and heuristic rules)

3.3.1 The control structure of the system is represented as part of the knowledge base

- While an AM-like agenda mechanism has been retained, the precise control algorithm is represented within EURISKO as a set of concepts, so the system can modify it itself
- Basically, there is a Select-Execute-PostMortem loop represented as a unit
- Specializations of this unit form the three nested loops that characterize the EURISKO program:
  - select and work on a topic
    - given a topic, select and work on a promising task
      - given a task, select and obey a relevant individual heuristic rule
- Representing the control structure explicitly had three benefits so far
  - It facilitates explanation; EURISKO can more coherently explain what it is doing at any given moment, when asked by the user

  - It allows enforced semantics
  - It is possible for EURISKO to notice when it's in danger of being in an infinite loop, even a subtle one in which there is no obvious infinite recursion or circular list structure involved
- The original motivation for explicitly representing control was to enable the program to meaningfully modify its own control code, but this has always resulted in bugs (due to an inadequate mastery of programming, of models of learning, and so on)

3.3.2 Multiple agendae

- The human research sticks with a topc for an extended period of time
- Party this is due to the difficulty of "swapping in" a whole new set of concepts, heuristics, etc.
- How does EURISKO stay focused on a single topic for a nontrivial period of time?
- Some (initially eight) of EURISKO's concepts (e.g., Games, DevicePhysics, NumberTheory) represent "topics"
- Each topic has a slot called Agenda, which contains its own agenda of tasks dealing with that topic (concept) and/or with one or more of its specializations
- There is no longer one central agenda; rather, there is a "current topic", and its agenda is the one being used for a while

3.3.3 Dynamic creation and elimination of agendae (topics)

- If agenda A contains more than four times as many tasks as the average agenda, then (try to) split A into about three pieces
- If the number of units called on per task, when working on tasks of agenda A, is more than ten times the rate at which other agendae inspect units, then (try to) split A into two pieces
- When an agenda shrinks too small, rules cause it to be merged into all appropriate immediate generalizations' agendae

3.3.4 Selecting a task: the half-frame problem

- Selecting a task is done as follows
  - The top task's reasons are evaluated carefully, and its rating is updated
  - If, after reevaluation, the top task's rating falls below that of task number 2, we merge it back into the agenda, and repeat this step
  - Some task will stay at the top of the agenda and be elected for execution

3.3.5 Executing a task: dynamically assembling a rule interpreter

- The first activity is to locate a set of potentially relevant heuristic rules, rules whose execution may (help to) satisfy the chosen task
- Space and time bounds are computed (and may be updated as the rules fire)
- There are several ways that the pieces of the relevant rules can be run as executable code - i.e., several possible rule interpreters

3.3.6 Post-mortem of a task: non-blind "suspend and resume"

- After the second phase ends (execution), a careful analysis is performed upon that activity
  - What happened?
  - How many rules succeeded?
  - How long did the task take?
  - How much space?
  - Is the user still interested in this topic, or is it time to (possibly) switch to a new one?
- The task is re-examined in light of its reasons: is it now worth putting back on the agenda? With what reason?
- If a task failed, it will usually be placed back on the agenda along with some new tasks which (if they succeed) might enable this one to run successfully

- This task's failure serves as one reason for those new tasks, and when they succeed their post-mortem should boost the priority of this task
- What EURISKO's control structure allows here is a sort of best-first knowledge-guided generalization of that mechanism

3.3.7 Each heuristic rule is itself a concept; we do not distinguish metarules from rules

- As part of what we get from representing each rule as a full-fledged unit, the rules are automatically now organized into an enormous generalization/specialization hierarchy
- The so-called Weak Methods (generate and test, hill climbing, etc.) lie at the top (most general), and there are many hundreds of entries near the bottom (specific judgmental rules which mentioned particular terms like "n-doped", "nuclear dampers", and "perfect numbers")
- But what of the structure in between? In particular what are the next hundred or so nodes below the five weak methods? What is the average depth of the tree, the average branching factor, and so on?
- Once a task is chosen, say working on concept C, a rule interpreter is chosen or synthesized
- This is run on the set of potentially relevant rules, namely the rules pointed to by C or by one of its generalizations
- The organization of rules into a tree enables this set to be small (on the order of the log of the total number of rules in the system)
- The interpreter will evaluate If parts of rules and execute Then parts in some fashion (perhaps dealing wtih rules one at a time, perhaps running all their If's and then picking a rule at a time to carry out its Then's, perhaps running all the Then-Conjecture slots of all truly relevant rules immediately, etc.)
- The post-mortem of an individual rule is necessarily simple: bookkeeping information about time and space used, new units created, etc. are recorded

3.4 1.3 Ideas about communication

3.4.1 As EURISKO matures, it interacts less as a pupil, more as a co-researcher

- Besides models of users and user-groups, EURISKO should have models of dialogue-modes (tutoring the system, solving problem, being taught by the system, etc.)

3.4.2 EURISKO must quickly notice when new concepts are related to existing ones

- EURISKO generates new concepts frequently
- One result we've noted is the high frequency with which these "new" concepts are in fact equivalent to an already-existing one
- EURISKO should have a fast way of checking each new concept, to see if it's genuinely new or not
- We call this "the recognition problem"
- EURISKO currently employs the following strategy to deal with this problem
  - Each unit knows which slots are criterial, i.e., define it
  - Each such criterial slot s knows the way in which it makes sense to do matching

3.4.3 EURISKO always has the initiative; the user can request but never demand

- When the user types in some message indicating that he wishes to define or modify a concept, that request is placed as a very (but not infinitely-) high priority task on the agenda

3.4.4 Modelling the user enables the creation of a good first impression

- Creating a good "first impression" is important
- EURISKO solves this problem by building up and using models of its users
- When a user logs in, the program attempts to quickly guess as much as possible his profession, his interests, his notations
- When a few things are observed, EURISKO can tentatively assign (as defaults, as it were) all the other known co-ocurring "symptoms"

- To support expectation-filtering by user models, a massive data base must exist, dealing with people in general, broken into groups, and even some data about specific known individual users

3.5 2 Results of EURISKO Applied to Naval Fleet Design

- In AM, there was always the possibility that while each heuristic seemed intuitively obvious and general, its true nature was merely an encoding of some of known mathematics, and that was in fact why it appealed to our intuition (that our intuition has been shaped to reflect a rough image of mathematics that exists already)
- We strongly believed this not to be the case however
- EURISKO has been a good test of the hypothesis that a large but general set of judgmental rules for manipulating concepts (and for discovering new rules) can be found and operationalized
- Any program claiming to be a "discovery program" should aspire to two goals:
  - use the same methods to discover concepts, conjectures, and heuristics in several domains
  - make at least a few genuinely new (to mankind) useful discoveries
- EURISKO designed a fleet of ships suitable for entry in the 1981 and 1982
- Each participant has a budget of a trillion "credits" (roughly equal to dollars) to spend in designing and building a fleet of futuristic ships
- There are over one hundred pages of rules which detail various costs, constraints, and tradeoffs, but basically there are two levels of variability in the design process:
  - Design an individual ship: worry about tradeoffs between types of weapons carried, amount of armor on the hull, agility of the vessel, groupings of weapons into batteries, amount of fuel carried, which systems will have backups, etc.
  - After designing many distinct kinds of individual ships, group them together into a fleet. The fleet must meet several design constraints (e.g., some ships in the fleet, having a total fuel tonnage of at least 10% of the total fleet fuel tonnage, must be capable of refueling and processing fuel), and in addition must function as a coherent unit
- To handle this task, 146 units were added, by hand, to EURISKO
- A couple of slots of interest:
  - MyWorth: The value of the concept to EURISKO - i.e., how compact it's been, how little CPU time it's wasted, how many interesting analogies were built using it, how many of the structural modifications to it were fruitful, etc.
  - MyInitialWorth: The value of MyWorth at the time it was created
  - Worth: How useful it the concept to the goal (e.g., EnergyGun Worth specifies how useful energy guns are to have on ships)
- What happened to lower the MyWorth of EnergyGun?
  - At one time, it was selected as a candidate for modification
  - EURISKO spent some time trying to analogize between it and other types of weapons, and nothing much came out of that
  - As a result, its MyWorth was dropped from 500 to 400
  - Through many tens of simulations, it became clear that one could buy enough armor plating to make a ship invulnerable to attacks by these types of weapons, and from then on almost all ships were so armored
  - Thus, any ships having energy weapons were at a serious disadvantage, and gradually - as they lost - the Worth of EnergyGun declined
- The slot called "Rarity" reflects the fact that, during a recent run, EURISKO examined nine objects, known to be weapons, to see if they were energy guns; one of them was
- This is the kind of bookkeeping record which heuristic rules might want to access; e.g., rules which say "If C is a specialization of G, and (empirically) very few G's turn out to be C's, then . . . "
- Many of EURISKO's slots have inverses
- Of the 146 added concepts, two represented new types of activities: playing a game, and running a simulation
- Managing a simulation caused us to augment EURISKO with three new heuristics; these check for termination, try to project the ultimate outcome of the simulation, and check for infinite loops during

simulation
- How did we get EURISKO to play the game?
- The unit Games is a topic, and as such can have an agenda
- One of the new units, PlayTravellerFleetBattle, had no examples, so a general heuristic added a new task to the Games agenda:
  - Find examples of PlayTravellerFleetBattle
- The Games topic was selected as the current topic (by pointing to it with a cursor, though it could have been selected indirectly by supplying a user model that claimed the user is very interested in games)
- Once Games was the chosen topic, there were only a few tasks with high priority
- The first one EURISKO ran defined the difference between Games and TwoPersonGames, and made a note that sometime EURISKO should look into defining some of those NonTwoPersonGames
- To win the TCS, EURISKO used many "mutation" operators (genetic programming?)
  - To keep a mutation, it had to beat its predecessors (ancestors)
- What EURISKO found were not fundamental rules for fleet and ship design; rather, it uncovered anomalies, fortuitous interactions among rules, unrealistic loopholes that hadn't been forseen by the designers of the TCS simulation system
- This notion of a large, unexplored search space, not necessarily well-matched to our everyday common-sense intuitions appears to characterize those domains for which automated discovery (of both concepts and heuristics) is currently most viable

3.6 3 Results of EURISKO Applied to Other Tasks

3.7 3.1 EURISKO applied to elementary mathematics

- The first domain we added concepts about was mathematics, specifically the same starting collection of finite set theory concepts AM began with
- Fifty heuristics were added, which subsumed most of AM's old set of 243
- EURISKO duplicated many of the results of AM: finding elementary set theory theorems, extreme properties of set operations, and defining useful new objects and operations about 50% of the time
- The other 50% of its time was spent about half in generating awful concepts, and half in attempting to produce new heuristics and types of slots
- About 200 math concepts were present in the system, to work in set theory and number theory
- After about 500 hours of running, another thousand concepts had been considered, and 200 of them had proven interesting (empirically, in the program's judgment, and later confirmed by human inspection)
- Of these new concepts, 11 were valuable, specific new heuristics, and 7 were useful new types of slots
- Of the 7 new slots types, four were slots that only heuristics could possess
  - IfConstant
  - IfIdentity
  - IfUnchanged
  - ThenConjecture
- The three If slots were needed because of the high frequency with which new functions turned out to be closely related to
  - a constant function
  - the identity function
  - the same function they were synthesized from

3.8 3.2 EURISKO applied to LISP programming

- 200 of the most common INTERLISP functions have been represented as units within EURISKO
- A very general heuristic EURISKO possessed said: "If f can often be used in place of g, and f uses less resources, then replace g by f wherever possible"
- An interesting LISP heuristic was found: "Sometimes 'AND' means 'do in sequence', and sometimes it means 'doable simultaneously', and only the latter case is likely to yield good results if you're considering generalizing a piece of code by removing conjuncts

- Large programs are carefully engineered artifacts, complex constructs with thousands of pieces in a kind of unstable equilibrium
- Any sort of random perturbation is likely to produce an error rather than a novel mutant
- EURISKO had successes in automatic programming only when it modified functions which had been coded as units. Why was this?
  - It's easier to modify small programs than large, opaque lump of LISP code about which nothing is known

3.9 3.3 EURISKO applied to other tasks

3.9.1 Evolution

- The simulation of organisms competing, followed by the most fit ones reproducing mutated offspring for the next simulated generation
- There is no doubt that the simulated evolution progressed almost not at all when mutation was random, and quite rapidly when mutation was under control of a body of heuristic rules

3.9.2 Games

- Several general Games concepts were added to the knowledge base: material, position, tactic, two-person game, fairness, player, opponent, etc.
- A few heuristics were inserted, as very general stategies: simultaneous action, feint, pin, trap
- These heuristics were derived using Chess and Bridge (by a system builder) and were successfully applied to Go
- One area of current research is getting EURISKO to discover new games; that is, make up a set of rules, simulate the game, and evaluate it according to various criteria

3.9.3 Heuretics

- The study of heuristics
- In operational terms, EURISKO spent time forming and testing heuristics about learning new heuristics
- One of the first heuristics that EURISKO synthesized (H59) would put its own name down as one of the discoverers of a new conjecture
- This required the implementation of a small "meta-level" of protected code that the rest of the system could not modify
- A heuristic arose which said that all machine-synthesized heuristics were terrible and should be eliminated
- EURISKO chose this very heuristic as one of the first to eliminate, and the problem solved itself

3.9.4 Representation

- EURISKO's task is quite constrained, actually: look for useful new slots which are specific to the various domains you are working in
- This type of activity - formulating new domain-specific slots - happened rarely, but we believe it to be one of the most important long-range activities EURISKO can do

3.10 4 Conclusions about Mechanizing the Process of Discovery

- The domain should be as little explored as possible
- There must be a way to simulate - or directly carry out - experiments
- The "search space" should be too immense for other methods to work
- There should be many objects, operators, kinds of objects, and kinds of operators. They should be related hierarchically and in other ways
- The task domain must be rich in heuristic structure
- There must be ways to generate, to prune, and to evaluate

- The "language" one uses to represent the concepts must be a natural one, given the set of objects and operators
  - Even though the discovery of new heuristics is important, the presence (and maintenance) of an appropriate representation for knowledge is even more necessary
- Criteria which make a domain suitable for AM-like exploration (discovery of new concepts and conjectures) are - taken to extremes - the same criteria which make a domain suitable for EURISKO-like exploration (discovery of new heuristics)

4 See also

- AM: An artificial intelligence approach to discovery in mathematics as heuristic search
- From AM to CYRANO

5 References

- http://www.cs.northwestern.edu/~mek802/papers/not-mine/Lenat_EURISKO.pdf
- http://aliciapatterson.org/stories/eurisko-computer-mind-its-own