GP As If You Meant It: Real and Imaginary User Experience

William A. Tozier

Abstract (to be written last)

Key words: keywords to your chapter, these words should also be indexed

1 Why

More than a decade ago, Rick Riolo, Bill Worzel and I were working on a consulting project together that involved genetic programming. As we chatted one day, Rick was asked what he'd most like to see as part of the research program of GP "moving forward". Rick's answer informs this contribution, as well as much of my professional work with GP in the years since.

As I recall, Rick said he'd like work to focus on the "symptoms" we often see in evolutionary search processes: premature convergence, failure to improve, catastrophic lack of diversity, mysterious things like that sense we all get when we look at results that suggests *we made bad choices* of some sort.

The literature abounds with well-written papers describing tips for avoiding local minima, improving on common search operators, and describing "horse races" between Bad Old and Better New search methodologies applied to benchmark problems. But I take Rick's challenge not only to mean that it would be useful to have a catalog which lists situations where search operator X acting under contingency Y tends to produce outcome Z, but also that *the things we identify as symptoms themselves* are poorly understood.

This contribution aims squarely at that sensibility which sees the "symptoms" and "pathologies" as something we as a research community should be trying to stamp out *before* letting users run their own GP systems. Instead, I'll argue that because GP differs qualitatively and philosophically from most other machine learning approaches, and also from the broader lay understanding of the "automated design"

and exploratory research, much of the value GP offers is lost or ignored when we approach it as something to be "cleaned up".

1.1 Challenges

Somewhere during that same project years ago, I remember a Project Manager recounting the story that when domain expert customers were shown a collection of Pareto-optimal solutions to the problem being explored, they were upset and confused: "We don't want a choice, we want *the best*." Of course there could be no *best*, and they supposedly knew that; this response came after *months* of analysis, interviews, technical planning and a collective agreement that the problem was fundamentally multi-objective. Even so, when the decision-makers were faced with the *unquestionably* successful results they had collectively specified and specifically asked for... those results weren't right enough.

I would much rather we faced the fact that this shocked response to *seeing what's happening* is inevitable—not just in the form of unwitting surprise from silly lay "customers" in an application project, but also from ourselves when we undertake any *interesting* GP project, whether the project is "theoretical" or "practical", "small" or "large".

Any *interesting* project will resist our best expectations. Now and then we should question whether our habitual response to that inevitable resistance is in any sense effective.

The field of GP has grown quite a bit in the 15 years since my two anecdotes. We who follow it closely have watched as frameworks, techniques and methodologies have diversified. There are the extraordinary ones we describe to one another in our little workshops, but also an increasing number of newsworthy and admirable public successes.

But this acceleration of innovation brings with it a dilution of the theoretical warrants we use to justify results. Most practical successes today highlight unique domain-specific quirks. New language and framework implementations makes numerous contingent (and often arbitrary) design and architectural choices. Even the fundamental ontological concepts of "individual", "fitness", "behavior", "generation" and "population" start to get troubling when try to fit many modern algorithms into their categories.

This growing divergence between current theory and practice has consequences both outside and within our field. The continued lack of interest in GP among statisticians, planners, designers and fans of traditional mathematical programming models is not merely a matter of disciplinary border wars. As we increasingly lose the ability to say *why* something is happening in a given GP project, we are faced with an audience who stopped paying attention to progress in GP last century. As a result, students and researchers aiming to convince peers outside the field are forced to undertake mismatched experiments and apply unconvincing statistical tests. In response to these constant distractions, much time and attention goes towards un-

helpful work guaranteeing that which cannot be guaranteed: running "replicates" of a process that is *designed* to provide novel answers, measuring "reliability" of a process that *intentionally* skirts dynamical chaos, providing "summary charts" of a process that strives to be as complex as the evolution of living beings, and (possibly worst) setting strict deadlines for an open-ended process to "finish" or "succeed".

I hasten to say this is not a fault in our field, but rather a broader philosophical and cultural problem. But it is nonetheless a misconception that undermines our understanding of what GP is *doing*, of the way it unfolds in theory and in practice, and even what it's *for*.

1.2 A top-down approach

I will make my case here in the form of an exercise for advanced GP users, or *kata*. The habit of pursuing *kata*, "code retreats", "hackathons" and other skill-honing practices is popular among software developers, and especially among the more advanced. Indeed, the title of my exercise ("GP as if you meant it") is taken from a particularly influential exercise designed by Keith Braithwaite, "TDD as if you meant it".

Braithwaite's exercise feels subjectively *harder* for more advanced programmers honing their development form; he suggests that novice programmers haven't learned ingrained but questionable habits, and haven't identified "shortcuts" that "simplify" the practices. In the same way, this exercise will feel *most* artificial and restrictive to those of us with the most experience with GP. But like the martial arts exercises from software *kata* were inspired, it isn't intended to be simple or even pleasant for the participants.

I intend it to expose habits I'll tactlessly call "psueudo-GP" among those of us who have learned through the years to think it's cheap and painless to *just shut it off and start over* when things start to get strange in the course of a GP project. As an important side-effect, it helps surface that philosophical problem I alluded to above. But I have found the most useful result is how strongly and immediately it suggests tools with which one can address Rick's decade-old wish. It forces the participant to formally identify "pathology" and "symptom" before allowing them to attempt a "fix"... and even that must done with an intentionally limited set of tools.

Note, what I'll describe is a "thought-experiment", nor is it a serious suggestion for a new way of working on "real problems", nor as "training" for newcomers to the field. Rather it is designed as a rigorous and formal exercise to be undertaken by those of us already working closely with GP systems. Its intent is to surface the three shortcomings I've identified above:

1. what GP *does*
2. how GP *unfolds*
3. what GP is *for*

2 How we treat GP, and how it treats us in turn

Genetic Programming[1] embodies a very particular *stance* towards the scientific and engineering work of modeling, design, analysis and optimization. I increasingly suspect the resistance we've all recounted towards GP from our prospective technical and lay audience has little to do with our technical results, but rather arises from uncertainty among that audience with GP's very particular "way of working".

There is a tacit assumption, even among GP theorists and practitioners, that science and engineering are only "rigorous" when they proceed through a sequence of ordered phases from planning to implementation. The "scientific method" is most often represented something along the lines of

> 1. conceptualization; 2. planning; 3. design; 4. architecture; 5. implementation; 6. testing; 7. debugging

I am sure very few scientists or engineers of my acquaintance would admit any *real* project (in history) has ever followed this narrative arc in a literal sense. But that story nonetheless informs and constrains much of our work lives, from fund-raising to publishing reports:

> Because of the body of published work, an insight was had. The insight was framed as a formal hypothesis. The hypothesis (and current Best Statistical Practices) suggested an experimental design, one familiar and obvious to any in Our Discipline. The experimental design was undertaken, the data were collected, the hypothesis duly tested, and now we can be confident of its veracity because… well, you just heard me say "Best Statistical Practices", right?

Under trivial term substitutions—"cost–benefit analysis" and "requirements document" for "hypotheses" and "experimental design", for example—the same narrative can be used to describe almost any institutional project management and public policy planning process as well. The flow in every case is essentially from *vision* to *plan*, *plan* to *implementation*, *implementation* to *verification*, and *verification* to *validation*.

Of course, nobody "really believes" this narrative who has ever done any of the work. It is a matter for another day to draw parallels with the social construction of religious belief.[2] There have been numerous philosophical challenges to this artificial narrative of course, from Peirce and Dewey nearly a century ago, to Kuhn and Lakatos in the 1970s, and many more to be found in the Table of Contents in any Philosophy of Science text.

One in particular is my focus today: Andrew Pickering.

---

[1] And not just Genetic Programming as such, but also the broader discipline to which I claim it belongs and which is not obliged to be either "genetic" or "programming". I prefer to call this looser collection of practices "generative processing", and will also abbreviate it "GP"; assume I mean the latter in every case.

[2] Paul Veyne's excellent *Did the Greeks Believe in Their Myths?* might be an interesting starting point, though.

2.1 Pickering's "Mangle of Practice"

Andrew Pickering's *Mangle of Practice* is a decade old, but surprisingly little-known outside the field of Science Studies. His approach is especially noteworthy here because I find it so close to our actual experience using GP. Indeed, most colleagues who hear it for the first time utter an inevitable "didn't we already know this?" I think the distinction Pickering's approach is able to highlight is exactly the problematic one in our work: between the illusory (but publishable) narrative of "scientific method", and the realized experience we have of *performing science*. At the cost of glossing a lot of his well-considered structure, let me summarize.

First, the act of "doing science" is *at no point whatsoever* to be understood as one in which an isolated "researcher" works in an objective and static field of "externalities" and "facts". Rather, research begins only when the researcher undertakes to *makes some artifact*: a block of code, a technical instrument, an equation, a sketch, a maquette or even a thoughtful conversation at a conference. Everything before the researcher begins to make these things is a matter of building "vision" (my term); research proper only occurs when work is done in the real world. I'll call this work product *the thing made*, and keep in mind that it is a proxy for the vague collection we might otherwise call "the project".

We see Pickering's model moving quickly away from more traditional views of science when we realize he has given the inanimate *thing made* an agency of its own. In any real project, we perceive the *thing made* resisting—whether we imagine that it resists "in itself", or as an agent of externalities that impinge on the work in progress to make our driving vision differ from reality. The *thing made* comes to represent, in other words, the facts of the actual world, the cultural assumptions and norms of one's discipline, the raw materials and toolkit available to a practitioner, and so forth. "Resistance" here is not merely a problem with a software bug or lost raw materials, but includes one's sense *on seeing it* that something's not quite right. The *thing made*'s resistance, as perceived by the researcher, leads her to react in turn.

Consider the language we use in the face of this resistance: it "feels wrong"; it "points something out"; it "wants to do X instead of Y"; it's "doing something too complicated for me to understand right now".

And—assuming science, engineering, art or any other creative process is indeed what's being done—the researcher *changes in response to this resistance*. That vision changes, the plan adapts, or in some other way the *thing made* causes a response in the state of the researcher herself. Pickering's Mangle is this emergent dance of inanimate agency: the researcher starting to follow a vision by making (or altering) a *thing*, and the *thing made* in turn acting as a channel for the world itself to steer the researcher in another direction.

Pickering's "mangle"[3] is this emergent dance of agency, undertaken between the researcher and the project: the researcher makes, the *thing made* resists, the researcher is influenced and redirected, *accommodating* the resistance. It may seem glib to say that "no plan survives contact with the enemy," but Pickering's narrative of creative work emphasizes the fact that no *revised* plan survives unchanged, either. In the traditional narrative, we elide the work as it unfolded and re-frame it as a sort of idealized, apersonal Platonic truth: we use the passive voice, we hide the missteps and confusion, we paint a story moving from vision to plan to success.

In a GP setting, the many modes of resistance we perceive are the very "symptoms" Rick mused about. Even though we as researchers know we've written all the code and set all the parameters, we're nonetheless willing to speak of a GP run "doing" things, as opposed to merely unfolding according to our plan. A GP system does not "resist" by, for example, "having the wrong population size". Rather it resists by *causing concern or dissatisfaction in the user*, which in turn sparks in that user a practical (if only explanatory) response, which leads them to change the *thing made*.

2.2 GP as "mangle-ish practice"

The broader field of machine learning seems to take a much more traditional stance towards its subject matter: machine learning frameworks (excepting GP) are each discrete tools aimed at producing standardized and reproducible results to particular statistical questions. The result of training a neural network or even a random forest on a given data set is not expected to be a *surprise* in any real sense, but rather the reliable and robust end-product of applying numerical optimization to a well-specified mathematical programming problem. Whether one describes these machine learning processes as "minimizing out-of-sample error" or "maximizing information gain", the supposed strength of most machine learning approaches is the *unsurprising* nature of their use cases and outputs.

On the other hand, GP has the capacity to *tell us stories*—even in the relatively "simple" domain of symbolic regression. The space under consideration by GP is not merely a vector of numerical constants or a binary mask over a suite of input variables, but the *power-set* of inputs, functions over inputs, and higher-order func-

[3] The word "mangle" he has chosen is itself interesting and insightful:

> . . . I find "mangle" a convenient and suggestive shorthand for the dialectic because, for me, it conjures up the image of the unpredictable transformations worked upon whatever gets fed into the old-fashioned device of the same name used to squeeze the water out of the washing. It draws attention to the emergently intertwined delineation and reconfiguration of machinic captures and human intentions, practices, and so on. The word "mangle" can also be used appropriately in other ways, for instance as a verb. Thus I say that the contours of material and social agency are mangled in practice, meaning emergently transformed and delineated in the dialectic of resistance and accommodation. . . .

tions over those. We who work in the field can be glib about the "open-endedness" of GP systems, but that open-endedness puts them at odds with their supposed relatives in machine learning. While GP *can* be used to explore arbitrarily close to some paradigmatic model, its more typical use case leads to the production of *unexpected* insights—to the degree that a number of us feel justified in treating it as a strong candidate for "real" artificial intelligence.

I argue we have that leeway because of the way GP surfaces Pickering's Mangle. When the methodology "works", it does so by offering *helpful resistance* in our engagement with the problem at hand, whether in the form of surprising answers, or validation of our suspicions, or simply legible suggestions of ways to make subsequent moves. GP *dances* with us, while most other machine learning methods are "mere tools".

2.3 Against replication

Nonetheless, there seems to be a widespread desire, inside and outside our field, to frame GP as a methodology for producing *unsurprising* models from data, more in keeping with the traditional linear of scientific work. That is, an idealized user is expected to proceed something like the users of any other machine learning or mathematical programming framework:

1. frame your problem in the correct formal language
2. "get" a GP system
3. run GP "on your data"
4. (whatever this is, it's not our problem)
5. you have solved your problem

This is not original with the GP community; there is strong pressure from our peers in other disciplines and our users to promote this use case, not least because it is *exactly* the stance expected in any planning or public policy setting, or in any scientific or programming project management setting. That is, we are under tremendous social pressure to treat GP as a *tool* to be invoked in a known, predictable and well-described planning situation.

The resulting resistance from early-adopting users shouldn't be unexpected. *Being surprising* may well be the worst conceivable behavior for any tool to be used in a traditional project setting. And given that pressure, it's no wonder that so much of GP research is focused on constraining tweaks to bring GP "into line". If only GP could be "tamed" or made "adaptive" so that step (4) above *never happens*.

I imagine this is why so many GP research projects strive for rigor in the form of counting replicates which "find the solution": they aim not to convince users of the known strengths of GP, but rather demonstrate to critical peers that GP can be "tamed" into a mere tool.

What would a "replicate" stand for, to a user who sought to exploit GP's strength in a project (whether theoretical or practical) where *search* rather than the algorithm

itself is the primary focus? Projects which authentically "use" GP *must necessarily* be those searching for noteworthy answers—which is to say *surprising* and *interesting* answers—that they could not otherwise obtain. Thus, we should better think of a "replicate" as a sort of proxy for user frustration in step (4) above: that is, it represents a project in which search begins, stalls, and for which the user cannot see a way to move search forward towards more interesting and useful answers.

But I think we would agree: it is a poor researcher who, when faced with a stumbling block in the form of a black box's misbehavior, doesn't attempt to work around that obstacle. Not to *begin the project again from scratch*, but to make changes and continue. Any researcher who is using GP *realistically* will be watching, and adjusting, and engaging and interacting with the process of search itself.

When a GP user is viewed as working within the traditional *linear* narrative of research, we see her run a population of 100 individuals for 100 generations, peer at the results that have been dumped into a CSV file, and find them wanting. Desiring an actual *answer*, she adjusts the GP parameters and begins another run of 100 generations… and repeats as necessary.

But there is *no discernible difference* when we frame this same process of "many runs" as a single project involving initial researcher moves, resistances thrown up by the *thing made*, and subsequent accommodations made by the researcher to the new perceived truths. Except, that is, in the researcher's view of her own response to resistance: if she comes to the project with *plans* for running "ten replicates", we can only assume she has learned from someone that a search algorithm is *supposed* to forget everything it has learned every 100 generations….

I cannot help but be reminded of the fallacy, surprisingly common both in and outside of our field, that "artificial intelligence" must somehow be a self-contained and non-interactive process. That is, that an "AI candidate" loses all authenticity as soon as it is "tweaked" or "adjusted" in the course of operation. It is as if every new-born "AI" must be jammed into an air-tight computational container and isolated *until it learns to reason*, and that without exceeding a finite computational budget. If humans creating *real* intelligences treated them anything like the way computer scientists insist we treat nascent *artificial* ones, murder charges would be forthcoming.

There are several practical reasons for us to try something different.

Consider our hypothetical GP user. In keeping with the Behaviorist standards of GP, she is carefully "not interfering" with any given run of 100 generations; she can only peer at a results file after the fact. During the course of any 100 generations, all sorts of dynamics have happened: crossover, mutation, selection, all the many random choices. Imagine for a moment we were given perfect access to the entire dynamical pedigree of the unsatisfying results she receives at the end, and were able to backtrack to any point in the run and change a single decision. Before that point, it's unclear how badly things will actually turn out at the 100-generation mark; at some point after that juncture, it's obvious to anybody watching that the whole thing's a mess.

If such miraculous insights were available, then surely the correct approach would be to intervene and adjust the situation when the crucial point was reached…

and then continue. Lacking (as we do) this miraculous insight, *why then does it seem reasonable to stop any run arbitrarily at a pre-ordained time point and begin again from scratch?*

It is, I think, because the myths of artificial intelligence and the linear narrative of science are so deeply intertwined. It is frowned upon to admit in a scientific paper, even when no mistakes were made, that the original vision and plan changed over the course of the project; rather we are expected to describe research *results* as the inevitable outcomes of an ahistorical process, and erase all resistance and accommodation actually done by human beings in the context of their projects. Similarly, it feels somehow wrong to admit in a GP project, even if every parameter was set correctly, that the original vision and plan gave way to the inevitable surprises thrown up by GP's inherent tendencies to do just that.

But insofar as GP *surprises us*, and since that is its sole strength over more predictable and manageable frameworks, we must inevitably see a good fraction of those surprises at least in part as *disappointments* rather than encouraging opportunities to change our plans. Let's learn new ways to accommodate those disappointments, and stop trying to make them go away.

3 "TDD as if you meant it"

As far as I can tell, Keith Braithwaite first described his training exercise for software developers in 2009. The target of the exercise is "Pseudo-TDD": the noted habit among software developers who claim to "know" and "do" test-driven development as part of their daily work towards a sort of thoughtless approximation of the technique.

I should note that a number of agile software development practices share informative relations to genetic programming's dynamics[4], but in this work I'll focus on those of TDD. In particular, test-driven development (or more accurately "test-driven design") *when done correctly* can break down the complex design space of a software project into a value-ordered set of incremental test cases, focus the developers' attention on those cases alone, inhibit unnecessary "code bloat" and feature creep, and produce low-complexity understandable and maintainable software.

TDD *as such* is a rigorous process, to the point where it can be described as "painful" (though also "useful") by experienced programmers. The steps are deceptively easy to trivialize and misunderstand, especially for those whose habits of thinking about code are ingrained:

1. Add a little (failing) test which exercises the next behavior you want to build into your codebase

[4] I imagine there is an Engineering Studies thesis in this for some aspiring graduate student: Genetic programming and agile development practices arose in the same period and more or less the same culture, and both informed by the same currents in complex systems and emergent approaches to problem-solving.

2. Run all tests, expecting only the newest to fail
3. Make the minimal change to your codebase that permits the new test to pass
4. Run all tests, expecting them all to succeed
5. Refactor codebase to remove duplication

Each stage offers a stumbling block for an experienced programmer, but the most salient for us now is the iterative flow of implementation (or "design") that it imposes: Each cycle begins with a choice of *which little test should next be added*; each cycle ends with a rigorous process of refactoring, not just of the new code but of the *entire cumulative codebase* produced so far. The middle three steps—implementing a *single* failing test and modifying the codebase *by just enough* so that all tests pass—feel when one is working through them as if they could be automated easily. The *mindfulness* of the process lives in the choice of next steps and (though somewhat less so) of standard refactoring operations.

Braithwaite's exercise does an interesting thing to surface the formal rigor of this approach. In it, the participants (willing, of course, because the exercise is a *kata* or "refresher" for experienced software developers to hone their skills) are asked to implement a nominally simple project like the game of Tic-Tac-Toe, given an *ordered* list of features to implement and the artificial restriction that they must go farther than normal TDD practice asks. Rather than producing a suite of tests and a self-contained codebase, they are forced to use *only* refactoring of code added to tests to produce their eventual "codebase". In other words, no code can be "produced" until the "smell" of duplication in the code added to multiple passing tests *provides a warrant* for refactoring it out.

Further, a facilitator patrols ongoing work and deletes *any and all code not called for by a pre-existing failing test*. Words like "irritating" and "annoying" crop up in participants' accounts of this onerous backtracking deletion the first few times it happens, as one might imagine. But as Gojko Adzik emphasizes in his descriptions of workshops, the resulting designs for even simple algorithms in this artificially amplified setting seems much more *open-ended* than it would if the software were built under the typical norms and habits an experienced programmer uses in normal conditions.

A number of contextually positive benefits are attributed to agile software development practices, and to TDD within that suite of practices. But the one that brings us here today is that aspect surfaced particularly in Adzik's account of Tic-tac-toe:

> By the end of the exercise, almost half the teams were coding towards something that was not a 3×3 char/int grid. We did not have the time to finish the whole thing, but some interesting solutions in making were:
>
> - a bag of fields that are literally taken by players—field objects start in the collection belonging to the game and move to collections belonging to players, which simply avoids edge cases such as taking an already taken field and makes checking for game end criteria very easy.
> - fields that have logic whether they are taken or not and by whom
> - game with a current state field that was recalculated as the actions were performed on it and methods that could set this externally to make it easy to test

In other words: innovative approaches to the problem at hand began to arise, though there wasn't enough time to finish them in the time alloted for the exercise. The analogy to our collective experience to date with genetic programming should start to peek through at this point—though the thoughtful reader will hopefully wonder what utility there is in an analogy drawn between two similarly unsatisfying outcomes.

[more here]

4 GP as if we meant it

In the same way that Braithwaite's onerous coding exercise is intended to drive the attention of its participants toward test-driven design with its obligation to write "real" code *only as a refactoring*, I'd like to be able to demand a *warrant* for every step that moves our changing genetic programming setup away from just plain random guessing. Braithwaite's target of "Pseudo-TDD" suggests an analogous "Pseudo-GP": one in which the fitness function is the only "interface" with the problem itself, and where the representation language, search operators, search objectives and other algorithmic "parameters" are *fixed*.[5]

Not only do traditional search operators like crossover, mutation and [negative] selection not come "for free" in this variant, but in every case we must develop a cogent, data-driven argument in favor of starting them *as part of an ongoing search process*. Similarly, the initial selection criteria will be limited to a minimal subset of the training data, and expansion (and other alterations) of the training set will have to be made in light of measured progress, not assumptions that "more is better" in every case.

The result will be an incremental process of refinement of an ongoing search, carried out not at the level of externally-assigned parameter "tweaks" but rather by *opening* the black boxes we typically demand and demanding we do surgery to correct their "pathologies" (and understand their mysteries) without killing them outright. It is not intended as an "algorithm" to supplant those used today, but rather as a forced re-description of what we actually already do.

[5] Braithwaite's participants often acknowledge they *know* and *use* TDD as it's formally described, but rarely take the time to do so unless "something goes wrong". I imagine many GP users will say they *know* and *use* all the innumerable design and setup options of GP, but treat them as adjustments to be invoked only when "something goes wrong". I offer no particular justification for either anecdote here, but the curious reader is encouraged to poll a sample of participants at any conference (agile or GP).

4.1 A tableau representation

The exercise proceeds as a sort of game, in which the User and the System take alternating turns. During the User's turn, she can see state of the system so far and make any of several *moves* from a limited set, each of which involve changing the settings of a *tableau*, which completely describes the state of the search system. During the System's turn, it will execute a finite number of steps in which it creates new individuals. At the end of the System's turn, control reverts to the User, and vice versa.

The tableau, and the suite of moves available to the User, affect three core aspects of search: `operators`, `answers`, and `rubrics`.

4.1.1 Operators

An `operator` is any function which takes as argument a (possibly empty) collection of `answers` and produces a new collection of `answers`. Operators thus include "random guessing", which in GP systems is often used to build an initial population, any "crossover" and "mutation" functions. No `operator` can refer to any `answer` not part of its argument set, not can any unset values be set within an `operator`.

So for example, if the User wished an `operator` could be constructed which took 30 `answers` as inputs, read their attributes (including `script` and that subset of their `rubric` values which were already set), and produce a single `answer` in its return set. It could *not* take an `answer`, change it, and keep the new one "if it was better", since the new `answer` would have no scores at all.

Note that no process is provided for `answers` to be *removed* from a tableau. The framework is purely cumulative.

All "parents" and other `answer` inputs required by an `operator` are chosen by *lexicase selection* automatically, using the complete suite of `rubrics` in play when invoked. Lexicase selection samples each `rubric` with equal probability.

4.1.2 Answers

An `answer` might traditionally be called an "individual" in GP literature. We can model them programmatically as key–value hash. All new `answers` are born with only a unique `id` and `script` field set. As an `answer` "matures" in the unfolding tableau, various other attributes will be set by the System player through the application of `rubrics`.

In the tableau visualization, we represent the unfolding collection of `answers` as the *rows* of a spreadsheet-like table, and their attributes and scores as the *columns*.

4.1.3 Rubrics

A `rubric` is any function which returns a scalar numerical value, given the instantaneous state of the tableau itself. For the most part these values can be understood as "scores" for various search objectives, though it is likely that a number of "implicit objectives" will also create new `rubrics`.

A `rubric` cannot store intermediate values, but can refer to the values of other `rubric` columns. Thus on specifying a `rubric` like "sum squared error over a training set", one must also create `rubrics` for "measured error when provided input *i*" for every element *i* of the training set. If the training set has 100 elements, then the single SSE `rubric` *implicitly* represents a suite of 101 new columns added to the tableau.

A `rubric` function does however have access to the state of the entire tableau as needed, including "forcing" other `rubric` values to be calculated. So it is entirely possible to specify `rubric` functions which score:

- number of characters in the `answer`'s script
- maximum error measured in any of 35 other `rubrics`
- number of `div0` errors produced when running with a particular set of inputs
- number of stochastic instructions appearing in the `answer`'s script, compared to *all other* `answers`

4.1.4 User moves

If we think of the tableau as a "spreadsheet" with a small list of `operators` and a large sheet of `answers` as rows and `rubrics` as columns, the User can add one new `operator` or one new column.

- add (or activate from a pre-existing list) exactly one `operator`
- add exactly one `rubric`, which may in turn create other *entailed* `rubrics` as described above

4.1.5 System turn

During its turn, the System player adds a specified minimum number of new `answers` to the tableau. If we think of the tableau as a "spreadsheet" the System can only add new `answers`. It follows a single rote cycle to do so:

1. select an `operator` from those in play, with equal probability
2. apply *lexicase selection* to the tableau to select the required number of input `answers`, filling in missing `rubric` scores as needed
3. apply that one `operator` to the inputs to produce new `answers`, and append those new `answers` immediately to the tableau

4. `HALT` if the number of new `answers` added in this turn meets or exceeds the number present when the turn started (in other words, if the number of `answers` has doubled); otherwise, go to step (1)

4.1.6 Initial setup and restrictions

- the only `operator` is "random guess", which creates a new `answer` with an arbitrary script
- no `rubrics` exist (only the `script` and `id` attributes of the `answers`)
- 50 new `answers` will be produced in the first System turn
- No `operator` can *evaluate* any `answer` except in a self-contained local namespace which lacks access to any tableau values or state information
- There is no mechanism for *removing* `answers` from the tableau
- The System player *always* uses lexicase selection, and *always* uses all `rubrics` as the selection features with equal probability.
- A `rubric` can only *run* a single `answer` once; stochastic scripts will only be sampled one time, and no `rubric` score is ever recalculated after the first time

4.1.7 Interface: the Goal of the Exercise

During the User's turn, they can of course interrogate the tableau in any way they want, without changing it. The point of the exercise is to drive the User to explore and create analytics and visualizations which can better inform their decisions over the course of the game.

5 An example session

6 Exploration and exploitation interfaces and affordances

7 Final thoughts: What should it mean to *act intelligently*?

The idea of this exercise came from observing the differences between experienced and "novice" (though with strong technical skills) GP users as they explored new problems in a GP setting.

8 An example session

Placeholder: I am converting the transcript of the session from an awful multipage LaTeX `\longtable` into a much more manageable `\enum` list like this. It will take a while, since the original `\longtable` was more than nine pages long and full of table markup.

8.1 The stage is set

`me` well what have we here?
`system` it does and says something back to me
`me` hey look I can have a little dialog sortof
`system` I can fight you every step of the way, though!

8.2 Progress and Stagnation

`me` Keep going
`system` OK!

8.3 Innovation

`me` Hey!
`system` What?

8.4 Frustrations

(&c)

References

Spector L (2012) Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In: McClymont K, Keedwell E (eds) 1st workshop on Understanding Problems (GECCO-UP), ACM, Philadelphia, Pennsylvania, USA, pp 401–408, DOI doi:

10.1145/2330784.2330846, URL http://hampshire.edu/lspector/pubs/wk09p4-spector.pdf