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

William A. Tozier

Abstract In this contribution I describe a *kata*, or exercise intended for advanced GP users, called "GP as if you meant it". In the exercise, the human participant(s) are charged with trying to "rescue" an ineffectual but unstoppable GP system which is set up initially to only use "random guessing". The exercise is in the form of a game of alternating turns, in which the human "User" player is provided complete information but only a very limited toolkit, and the (automated) GP System "player" can only be modified and "adjusted" by adding new search operators and objectives. The crux of the exercise is the human players' development of cogent *warrants* for every design decision they make when building and using GP systems—decisions which otherwise can be habitual or arbitrary. By explicitly prohibiting the most common response of "stop it and restart with different parameters", the participants are mindful of ways to *accommodate* the noted "pathologies" and "symptoms". The computational participant tends to just get warmer.

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

[version of 2015/03/29 at 18:52:59]

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, he said he'd like work to focus on the "symptoms" we often see in evolutionary search processes: premature convergence, slow and spotty improvement, the catastrophic lack of diversity,

and that ineffable feeling all GP professionals experience when they see results and *know* that somewhere, somebody has Chosen Unwisely.

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 an unfortunate sensibility I've detected in our field, which sees the many "symptoms" and "pathologies" as bugs to be eradicated *before* we let "real users" run their own GP systems. Instead I'll argue (belatedly following Rick's lead) that these "surprises" and "disappointments" are not only inevitable but should be the main focus of our theoretical and practical work. Indeed, much of the value GP can offer can be lost if we approach these phenomena as something to be "cleaned up" rather than being *accommodated*.

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.

In hindsight, I've come to think this shocked response was inevitable. It wasn't an unwitting surprise from some silly lay "customers" in an application project. Indeed, it's analogous to the way many in our field treat *interesting* GP projects, whether they're "theoretical" or "practical", "small" or "large". *Any interesting project will resist our initial plans and expectations.* When we're faced with even a little resistance, our first reaction is to imagine something has "gone wrong in the setup", shut the thing off and start again with "better" parameters.

It's unusual for anybody to consider interrogating the system *in itself* when it resists, and adapt or accommodate that resistance. Within our field, but also in a surprisingly broad layer of technical society.

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 unhelpful 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 which software *kata* first 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 "pseudo-GP" among those of us who have come 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 years-ago 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 "TDD as if you meant it"

As far as I can tell, Keith Braithwaite first described his training exercise for software developers in 2009. His target was a sense that software developers who thought they were using test-driven development practices were in fact doing something more like "Pseudo-TDD", a sort of slapdash and habitual approximation lacking many of the benefits of mindful practice.

While I've noted elsewhere that several agile software development practices share useful overlaps with the problems of GP science and engineering[1], in this work I'll focus on those of TDD. In particular, the observation that 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 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 very constraining and rigorous process—to the point where it can easily be described as "painful" (though also "useful") by experienced programmers. The steps are deceptively easy to gloss, misunderstand or miscommunicate, especially for those whose coding habits are ingrained. To paraphrase Beck and Braithwaite:

1. Add a little (failing) test which exercises the next behavior you want to build into your codebase
2. Run all tests, expecting *only the newest* to fail

[1] 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.

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 the codebase to remove duplication

Even though a single cycle through this iterative process can take less than a minute, each step can throw itself up as a stumbling block for an experienced programmer. But the most salient for us here is the iterative flow of implementation (or "design") that the cycle imposes: it begins with a choice of *which little test should next be added*, and 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 as though 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 those decisions, by making them *harder* rather than easier. In "TDD as if you meant it", the participants (willing, of course) are asked to implement a nominally simple project like the game of Tic-Tac-Toe using TDD, and are given a list of requisite features and an extra constraint. Rather than using "normal" TDD and producing a suite of tests to exercise a separate and self-contained codebase, they are forced to add code *only* to the tests themselves (to make them pass), and can only produce a separate "codebase" when duplication or other "code smells" drive them to refactor the code already added to tests. In other words, the *demand for a warrant* for writing code is much more stringent.

Throughout the exercise, a facilitator patrols teams of participants 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 descriptions of workshops he's facilitated, the resulting designs even for well-known algorithms in this artificially amplified setting become much more "open-ended" than would be expected if the code were written under the offhand attention of an experienced programmer without painful constraints.

Adzik mentions something in his account of a Tic-tac-toe exercise that's especially interesting:

> 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.

3 GP as if we meant it

In the same way that Braithwaite's coding exercise uses an onerous extra constraint[2] to drive participants towards more mindful and insightful decision-making, in this exercise I will demand a *warrant* for each implementation decision that moves a running GP setup away from random guessing. Braithwaite's target of "Pseudo-TDD" suggests an analogous "Pseudo-GP": one in which the fitness function and *post hoc* analysis is the only "interface" with the problem itself, and where the representation language, search operators, search objectives and other algorithmic "parameters" are *fixed* in the course of the run.[3]

Not only will 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 argument in favor of starting them *as part of an ongoing search process*. Similarly, the initial selection criteria will be limited to a minimal single element of the training data, and expansion of the training set in use will have to be made in light of measured progress, not the assumption that "more is better".

The result is a painfully incremental process of refinement of an unstoppable search which has *intentionally* "started wrong", which must be carried out not by stopping the whole thing and launching it again with parameter "tweaks" but rather by *doing surgery on the living patient* to correct any perceived "pathologies". Along the way, a fraction of the mysteries of "pathology" become much clearer and well-defined, even when they cannot be solved easily.

Again, let me emphasize that this is not intended as an "algorithm" in its own right, but rather as a suite of constraints imposed on what we do without paying attention already.

3.1 The game tableau

The exercise is structured as a sort of game with two players, one of which is a running GP System, and one which is a human User (or team of Users collaborating on making a single move at a time). The two players make alternating turns.

The game "setup" can be imagined as a *tableau* with two components: A single list of search `operators`, and a large (and growing) two-dimensional spreadsheet-like table with `answers` as its row labels and `rubrics` as its column labels.

---

[2] In this the sensibility reminds me of the constraint-driven art collective Oulipo, who are perhaps most famous for the *lipogram*, a literary work which cannot use a particular letter of the alphabet.

[3] 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).

During the User's turn, she can examine the state of the tableau, including all of its history, and can do any amount of data analysis or statistical consideration she wants. She is however permitted no more than *two* moves: in each turn she can code and launch one new `operator`, and can code and launch one new `rubric`.

Finally, the User is obliged to provide a convincing *warrant* for every move she make, to be reviewed by an external Facilitator. In the event this warrant is seen by the Facilitator to justify the particular move, he has the right to delete all `operators` and `rubrics` added by the User from that point forward *without resetting the System's moves*.

Facilitator buy-in for every move is therefore *strongly* encouraged.

During the System's turn, it will execute a finite number of steps in which it creates new individuals using the specified search `operators` and `rubrics`. At the end of the System's turn, control reverts to the User, and vice versa.

3.1.1 Answers

An `answer` might traditionally be called an "individual" in the GP literature. It is modeled here as key–value hash (or a simple prototypal object). Every new `answer` is born with only a unique `id` and its `script` field set. As they "mature" during the System's turns, various other attributes ("scores") will be set by `rubrics`.

Answers never "die", and cannot be removed from the tableau by either player as search progresses. The User can only reduce or increase the chance that certain types of `answer` contribute to later generations by constructing and launching `rubrics` to adjust selective forces.

3.1.2 Operators

An `operator` is a coded function which takes as its argument an unordered collection of `answers`, and which produces a new collection of `answers` as output.

The initial tableau includes only a single `operator`, which implements a precoded "random guessing" algorithm. On the User's move, she may build and launch other more complex (and familiar) `operators` like "crossover" or "mutation" functions. A wide variety of `operator` structures are possible in the domain-specific language provided.

The specific set of `answers` to which an `operator` is applied cannot be chosen directly by the User. All "parents" defined for each `operator` are chosen by *lexicase selection* automatically during the System's turns, using the complete suite of `rubrics` in play when invoked. This selection process samples each `rubric` in the tableau with equal probability.

It is feasible for the User to design a new "selection" `operator`, but it must rely on the immediate state of the `answers` it is given; in other words, no `operator` can explicitly invoke a particular `rubric` (or vice versa).

Note again that no mechanism exists which removes `answers` from the tableau.

3.1.3 Rubrics

A `rubric` is a function which returns a scalar value (not necessarily a *number*) for a given `answer`, conditioned as needed on the instantaneous tableau state. When a `rubric` is applied to an `answer`, it sets a new value (or "score") in that `answer`.

No "score" is ever updated in an `anwer` once set by a `rubric`. However, if multiple `answers` exist which have the same `script`, each may be scored at different times or in different contexts, and the resulting values may differ.

An aggregate or *higher-order* `rubric` can be created, but as a consequence it "entails" all of its component `rubrics`. So if the User constructs a higher-order `rubric` like "max error over 30 training cases", when she launches that `rubric` 31 in total will be launched, with the first 30 being in the form "absolute error for training case *i*".

If a User launches such a higher-order `rubric` on her turn, all its entailed `rubrics` will also be added before the System takes its turn.

When one of these "input" `rubrics` is applied during the System's turns, it does *not* automatically cause the higher-order aggregate to be applied. However, when the higher-order `rubric` is applied, it will force all of its "input" `rubrics` to be calculated and applied first.

In addition to "entailed" `rubrics` for aggregate scoring, the User can use any of the other extant `rubrics` present in the tableau in her definition, or the state of the interpreter before and after the `script` is run. It is therefore entirely possible to specify functions which score:

- number of characters in the `answer`'s script (since the `script` is an attribute of every `answer` with its own column in the tableau, and therefore its value is exactly equivalent to any other `rubric`-created score)
- 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`

Various problem-specific aspects are glossed here. Suffice to say, the construction of reasonable `rubrics` in response the System's moves is the core of the User player's game strategy.

3.1.4 User turn

In her turn, the User can do either or both of the following:

1. create and launch one `operator`
2. create and launch one `rubric` (which can be "higher-order" and entail others)

To support these decisions, the User can examine the tableau state and history in detail. Before the decisions are implemented, though, a *warrant* must be written for

each one. Whether the Facilitator is asked to review this is entirely up to the User player, subject to the warnings mentioned above.

3.1.5 System turn

During its turn, the System player adds a specified minimum number of new `answers` to the tableau, one at a time, using a simple form of lexicase selection. Until it reaches its halting state it iterates this cycle:

1. select one `operator` from those in the tableau, 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 the `operator` to the inputs to produce one or more new `answers`, and append those new `answers` immediately to the tableau
4. `HALT` if the number of new `answers` meets or exceeds the limit; otherwise, go to step (1)

3.1.6 Choosing a language

No particular constraints apply to the language or representation chosen for the exercise, except that it should be sophisticated enough to support *redundant capacity*. That is, for any given algorithmic goal (or stage), there should be multiple paths to success. So for example if the problem could reasonably be expected to involve ordered lists, then it would be wise if the language had at least two different ways to "deal with lists": iterators, recursion, a comprehensive set of second-order functional operators, an explicit `List` type with associated methods, that sort of thing.

If the language is sophisticated enough to have "libraries" of instructions and types intended for specialized domains, I'd strongly encourage that *all of these should be used*. That is, the facilitator should err on the side of "winnowing complexity", rather than forcing the User and System to *invent basic data structures* or the idea of floating-point numbers at the same time they're trying to solve the "real" problem under consideration.

Finally, if there is a choice between a familiar language and an odd one, then the odd one should be chosen, all other things being equal.

3.1.7 Problems

In keeping with my notes on languages above, in the choice of problems the planner should aim for something that would strike any experienced GP person as "ambitious". Which is to say: a reasonably good programmer would be able to hand-code the answer with a day's thought and work, but only in a familiar language; in the oddball GP language, it should feel "practically" impossible to hand-code. That

said, it should *also* be clear to any programmer familiar with the language specification that the needed *components* are all there, and that there are enough "parts" to approach any sub-task that crops up along the way from more than one angle.

Colleagues have pointed out that even if the chosen problem turns out to be "too simple", in the sense that the System player solves it quickly without much input from the User, then the "problem" addressed in the exercise immediately becomes one of driving the System to find a second *dissimilar* solution... and without restarting the search process, of course.

3.1.8 The Facilitator

There need not be a formal "Facilitator" if the User is willing and able to honor the spirit of the exercise. But especially in more formal and public situations, it makes perfect sense for the Facilitator to be the one to choose the problem and language, and also to provide the necessary technical infrastructure on which the computational part of the exercise will be run.

Any warrant that relies on "best practices", habit, tradition, or anything considered or described as "obvious" deserves to be invalidated as soon as it's noticed. Warrants should rather be judged valid only when the state of the tableau (or its history) is invoked to make a cogent argument for the submitted moves.

That said, a warrant does *not* need to be "technical" or even "rigorous", but merely robust enough to be convincing. For example, "I made crossover because I think it needs to *search in between the parents*," sounds to me like a shoddy excuse that invokes received wisdom. "I made crossover to drop the variance on this rubric and foster inbreeding of *these* solutions *here*," given a glance at the tableau and charts on hand, sounds good to me.

3.1.9 Initial setup and restrictions

- the only `operator` is "random guess", which creates one new `answer` with an arbitrary script
- the only `rubrics` present are the `script` and unique `id` fields of the `answers`
- the System player will add 250 new `answers` (or slightly more, depending on the `operators` chosen) in each of its turns
- the System player moves first
- there is no mechanism for *removing* `answers` from the tableau
- the System player *always* uses lexicase selection, *always* choses `operators` with equal probability from the current list, 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, though multiple copies of the same stochatic `answer` will probably end up with different scores in the same field

3.2 Interface

[this entire section is still being transcribed from an ill-advised experiment with LaTeX table typesetting; it'll be inserted as soon as possible]

3.2.1 The `operator` language

3.2.2 The `rubric` language

3.2.3 Visualization

4 An example session

[this entire section is still being transcribed from an ill-advised experiment with LaTeX table typesetting; it'll be inserted as soon as possible]

5 Summing up: How we treat GP, and how it treats us in turn

Genetic Programming[4] embodies a very particular *stance* towards the scientific and engineering work of modeling, design, analysis and optimization. I increasingly suspect the resistance from colleagues we've all encountered towards GP has little to do with our technical results as such. Rather it arises from unfamiliarity with GP's very particular "way of working". We are used to it—perhaps to the point of taking it for granted—but the mismatch and resulting response from those less familiar with the system has I think led us to *mask* our sense of it in an effort to fit in.

Briefly, the systemic fault lies in the awful "scientific method" that permeates our cultural dialog about the practice of science and engineering. You know the one, which is often something along the lines of:

vision → planning → design → architecture → implementation → testing → debugging

I'm sure very few scientists or engineers of my acquaintance would admit any *real* project has *ever* followed this narrative in a literal sense. But that story nonethe-

[4] 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.

less informs and constrains much of our work lives, from fund-raising to publishing reports:

> Based on the body of published work, an insight was had. The insight was framed as a formal hypothesis. The hypothesis (shaped by current Best Statistical Practices) immediately suggested an experimental design, which design is obvious to anyone familiar with Our Discipline. That 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 Practices", right?

"Nothing surprising happened while we were working on this project," in other words.

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 or 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 the work. It is a matter for another day to draw parallels with the social construction of religious belief.[5] And I am not the first to point it out; the history of Philosophy of Science is built primarily from the numerous philosophical challenges to this artificial narrative, from Peirce and Dewey nearly a century ago, to Kuhn and Lakatos and Feyerabend in the 1970s, and with many more to be found in the Table of Contents in any Philosophy of Science text.

Andrew Pickering is the inspiration for this project, though.

5.1 On the Mangle of Practice

Andrew Pickering's monograph *The Mangle of Practice* is a decade old, but surprisingly little-known outside his discipline of Science Studies. His approach is especially useful here, because I find it captures a surprising amount of our *actual experience* of building and using GP systems. Indeed, most colleagues who hear it for the first time utter an inevitable "didn't we already know this?"

Pickering's approach focuses on that problematic division I've sketched above, between the illusory (but publishable) linear narrative of the "scientific method", and the realized experience we all have had of *performing science* (or Mathematics, or Engineering, or for that matter Art). At the cost of glossing too much of his well-considered structure, let me summarize.

First, I should remind us all that the *performance of science* is just that: not an isolated but perceptive mind standing apart from the world, working in an objective and static field of "externalities" and "facts", but a *performance* done by a human

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

being present in that world. In Pickering's framework, we can say that research proper begins only when the researcher makes some artifact or formal "machine" in the world: writes a block of code, designs a technical instrument, considers a particular equation, draws a pencil sketch, or even a has a thoughtful conversation at a conference. Let me call this artifact *the thing made*. This is not the scientific paper that results in the end of the project, but rather the sum of all the sketchy notes, the cloud of more-or-less coherent ideas, the code and instruments, the collected observations, the plan and the community of colleagues helping with that plan: everything done in the world, mentally or physically, towards the goals of the project.

Pickering's model jumps quickly away from more traditional "scientific methods" when he treats this mechanism as capable of *agency in its own right*, and is willing to say that it can and does *resist* our intentions. In the context of the Mangle, the *thing made* is the conduit of the facts of the actual world to the researcher (and also of the cultural assumptions and norms of one's discipline, of the inherent tendencies of the raw materials and the practitioner's toolkit). "Resistance" here is not merely a reference to a software bug, a mathematical mistake or a shortage of crucial raw materials, but specifically denote one's sense *on seeing it* that "something's not quite right". In other words, it is the *thing made*'s resistance "on behalf of" the real world which forces the researcher to reconsider, change or adapt her plans, or otherwise *accommodate* that resistance.

The inspiration for granting agency to machinic abstractions (or even concrete dynamics) is obvious whenever we hear the language that crops out in the course of our work: the code "feels wrong"; the mathematics is "pointing something out"; the machine "wants to do X instead of Y". Projects in science, engineering and the arts do not proceed from a stage of planning to a stage of implementation, except in the ahistorical mythology of our published papers. Pickering's Mangle[6] does much better at capturing our first-hand experience of the work as an emergent dance of human and machinic agency *with one another*. The researcher starts to follow her vision by making (and altering) some artificial thing, that *thing made* acts as a channel for the world itself to resist, and as a result the researcher *accommodates* that resistance by moving in some different direction. In the traditional linear 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, after the fact paint a story which flows from vision to plan to success. But within the dance of Pickering's

[6] 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. . . .

Mangle, the degree to which we as researchers can successfully *accommodate* the inevitable resistance

In a GP setting, the notion of "machinic agency" seems much closer to our experience; after all, we are obliged not only a pick or write a specialized formal language to represent the space of solutions, but in every project we must also cobble together some framework of search operators, fitness operators, algorithms and instrumentation. But even when we've written all the code and set all the parameters personally, we're *still* willing to speak of a GP run "doing" things, as opposed to merely unfolding according to our plan. Indeed, if it happens by chance that GP "runs according to our plan" then arguably the problem was too *boring* to be worth mentioning....

I will argue below that GP's power (and difference from other machine learning approaches) lies in the very particular form of resistance it can offer us as its users. This is not merely "resistance" of the frustrating kind: we use GP most effectively when we want it to surprise us. The "surprise" is certainly something we are forced to accommodate with just as much attention and concentration as any more annoying resistance which might be thrown up, for example when we are forced to piece together "how the hell this thing actually works".

It is worth saying explicitly now, and again later, that by granting the *thing made* a machinic agency of its own, we can frame the problem of "pathology" and "symptoms" in GP more constructively. A GP system does not resist by "having the wrong population size" or "having too high a mutation rate"; those are not *behaviors*, but tiny facets of a complex plan instantiated (to some extent) in a complex dynamical system that is the GP system. Rather I would say a GP system resists by *raising concern or causing dissatisfaction in a human user*. It is inevitably that *human* observer who produces a productive and accommodating response.

5.2 GP as "mangle-ish practice"

The broader field of machine learning seems to take a much more traditional stance towards its subject matter. 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. Indeed, the supposed strength of most machine learning approaches is the very *unsurprising* nature of their use cases and outputs.

On the other hand, we all know that GP embodies a capacity to *tell us stories*, even in the relatively "simple" domain of symbolic regression. The space under consideration by GP is not some 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 functions over those. We who work in the field can be glib about the "open-endedness" of GP systems, but that open-endedness puts GP at odds with its supposed machine learning cousins. While GP can be used to explore arbitrarily

close to some parametric model, its more common use case is exactly the production of *unexpected* insights.

When the GP approach "works", it does so by offering *helpful resistance* in our engagement with the problem at hand, whether in the form of surprising answers, validation of our suspicions, or simply as a set of legible suggestions of ways to make subsequent moves. GP *dances* with us, while most other machine learning methods are exactly the "mere tools" they have been designed to be.

5.3 Against replication

Nonetheless, there seems to be a widespread desire inside and outside our field to frame GP as a way of exploring *unsurprising* models from data. As with neural networks or decision trees, the machine learning tool-user is expected to proceed something like this:

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

This does not arise from within the GP community; it is *exactly* the stance expected in any planning or public policy setting, or in any waterfall management or programming project management setting. Twenty years and more of skepticism even from early-adopting users shouldn't be unexpected. *Being surprising* might be the worst conceivable outcome for any tool used in a traditional project management setting.

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 a solution": they aim not to convince *users*, but rather to demonstrate to critical peers that GP can be "tamed" into another mere tool.

Think about "replicates" for a moment. What might a "replicate" be for a user who wants to exploit GP's strength of discovering new solutions? If one is searching for noteworthy answers—which is to say *surprising* and *interesting* answers—then a "replicate" must be some sort of proxy for user frustration in step (4) above. That is, a "replicate" stands in for a project in which search begins, stalls, and where the user cannot see a way to accommodate the resistance in context... and just gives up trying.

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 authenticity as

soon as it's "tweaked" or "adjusted" in the course of operation. It is as if every newborn "AI" must be quickly jammed into an air-tight computational container and isolated *until it learns to reason by itself*, and for that matter without exceeding a finite computational budget.[7]

Maybe the responsibility for this failure lies neither with the stupid user or the shoddily-made GP system, but in our unwillingness to let them give more useful feedback to one another.

Consider our typical GP user who is "not interfering" with her running GP system; she can only peer at a results file after the fact, and can't fiddle with the "settings" while the thing is actually working. But of course during any given run of 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?*

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".

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`

[7] If humans creating real intelligences treated them anything like the way computer scientists insist we treat nascent artificial ones, murder charges would be forthcoming.