GP As If You Meant It: An Exercise for Mindful Practice

William A. Tozier

Abstract In this contribution I present a *kata* called "GP As If You Meant It", aimed at advanced users of genetic programming. Inspired by *code katas* that are popular among software developers, it's an exercise designed to help participants hone their skills through mindful practice. Its intent is to surface certain unquestioned habits common in our field: to make the participants painfully aware of the tacit *justification* for certain GP algorithm design decisions they may otherwise take for granted. In the exercise, the human players are charged with trying to "rescue" an ineffectual but unstoppable GP system (which is the other "player"), which has been set up to only use "random guessing"—but they must do so by *incrementally modifying the search process without interrupting it*. The exercise is a game for two players, plus a Facilitator who acts as a referee. The human "User" player examines the state of the GP run in order to make amendments to its rules, using a very limited toolkit. The other "player" is the automated GP System itself, which adds to a growing population of solutions by applying the search operators and evaluation functions specified by the User player. The User's goal is to convince the System to produce "good enough" answers to a target supervised learning problem chosen by the Facilitator. To further complicate the task, the User must also provide the Facilitator with convincing justifications, or *warrants*, which explain each move she makes. The Facilitator chooses the initial search problem, provides training data, and most importantly is empowered to *disqualify* any of the User's moves if unconvinced by the accompanying warrants. As a result, the User is forced to work around our field's most insidious habit: that of "stopping it and starting over again with different parameters". In the process of working within these constraints, the participants—Facilitator and User—are made mindful of the habits they have already developed, tacitly or explicitly, for coping with "pathologies" and "symptoms" encountered in their more typical work with GP.

Key words: mindful practice, design process, coding kata, praxis, Mangle of Practice

1 Why: An Excuse

More than a decade ago, Rick Riolo, Bill Worzel and I worked together on a genetic programming consulting project. As we chatted one day, one of them—I don't recall which, and accounts vary—was asked what he'd most like to see as GP "moved forward", and said he'd want the field to think more about the "symptoms" we so often see when we apply evolutionary search processes in complex settings. That is: premature convergence, slow and spotty improvement, a catastrophic lack of diversity, and more generally that ineffable feeling all GP professionals experience when we look at results and *know we have chosen unwisely*.

Now the reader will point out that the literature in our field overflows with well-written papers describing tips for avoiding local minima, improving on common search operators, and running "horse races" between Bad Old and Better New search methodologies applied to benchmark problems. But while as a rule these are presented as a sort of *general principle*, most in practice are *case studies* in which it is shown, for example, that search operator X acts under contingency Y and sometimes produces outcome Z.

I am not left with a sense that this catalog addresses my colleague's stated wish. While it is surely necessary to compile such a list of individual observations under particular suites of experimental treatments and benchmarks, it is insufficient until we can achieve the skill of recognizing *when* something noteworthy is happening. In particular, I want to focus on those frequent but contingent situations in which we are willing to say our GP system *resists our expectations*.

A good deal of this chapter will be spent explaining just what I mean by this particular usage of "resistance", but for the moment the idea will be clear enough from this simple mental exercise: What are things GP has "done"—in the context of a particular project—that have left you feeling unsatisfied, confused, or frustrated? Not because you've made a simple mistake setting a parameter, or introduced a bug in a codebase, but because (at that moment) you have no idea why the GP process is doing *that thing* under those particular circumstances? Maybe it's not converging when you know it should; maybe it's exploring unexpectedly complex algorithms rather than obvious simpler ones, despite your reasonable parameter settings; maybe it's failing to solve simple problems but easily solving hard ones. In any case, the system "resists" your careful *and knowledgeable* expectations by inexplicably refusing to follow your plan.

My general point, and the particular point of this chapter and the exercise it describes, is that we should then ask: *When we feel GP is resisting us, what should we do—and why?* Further, after more than twenty years of work in the field, I think we have learned enough for this to be reasonable and productive research program.

I'm sure every reader has at least one story they can tell, in which GP has "resisted" in this sense. I want to draw our attention to these incidents from a conviction that it is exactly the steps we take to *accommodate* GP's resistance which provoke our sudden insights into the structure of our problem, drive us to build a "workaround" that becomes a novel selection algorithm or architecture, and lead us to successfully re-frame our project to use a completely new approach. Almost any chapter in this series of GPTP Workshop Proceedings will contain a story just like this: We create an age-layered population because working without age-layering didn't perform as expected (Hornby, 2009); we invent a new selection mechanism because traditional algorithms worked as described, but failed to capture crucial details of our problems (Spector, 2012); we create an entirely new representation and suite of search operators because our goals aren't met by snipping up and recombining in the traditional way (Ryan and Nicolau, 2003); and so on.

The point in each case is: one gains little *insight* into a problem when GP quickly pops out the "right answer" without a fight. Too often I see papers treating this ubiquitous resistance as something to be eradicated *before* "real users" are allowed run their own GP searches. Instead I'll argue here that "surprises" and "disappointments" are not only inevitable but *are the main source of value in many GP projects*, and as such should be the main focus of our theoretical and practical work.

Every interesting project will resist our plans and expectations. But in our field, whenever we're faced with even a bit of this sort of behavior, our first reaction is generally to act as though something has "gone wrong in the setup", then shut the misbehaving run down so we can start again with "better parameters" next time. I find it unusual for anybody to interrogate the system *in itself* when it resists, or to make an attempt to adapt or accommodate perceived resistance. I hasten to say this is not a fault with our field, but rather a symptom of broader philosophical and cultural problems in our approach to programming and computational research projects, and our understanding of computing more generally.

The exercise I describe in this chapter is intended to bring our attention back on GP as a dynamical process *in itself*, as opposed to a tool to be adjusted "in between" applications. It may be the case that other "traditional" machine learning methodologies are built on better-defined information-theoretic foundations, and come with suites of strong statistical tests for overfitting and robustness; as a result it's perfectly reasonable to treat them as *tools*, and as *malfunctioning* or just being *wrong for a problem* when they act in unexpected ways. But GP is somehow a different sort of animal: when it "resists", we are left with an essentially unlimited choice of how we should *accommodate* that resistance.

1.1 On mindful exercises

The habit of pursuing *kata*, "code retreats", "hackathons" and other skill-honing practices is popular among software developers, and especially among the more advanced. It apparently arose independently among a few groups in the 1980s, but Dave Thomas seems to have first used the Japanese term (Thomas, 2013).

Indeed, the title of my exercise ("GP As If You Meant It") is directly inspired by from an exercise designed by Keith Braithwaite, "TDD as if you meant it" (Braithwaite, 2011). One of the interesting aspects of Braithwaite's exercise is that it feels subjectively "harder" when attempted by 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, the exercise I describe will feel *most* artificial and restrictive to those of us with the most experience with GP. But like the martial arts exercises by which software *kata* first were inspired, it isn't intended to be simple or even pleasant for the participants. Just as Braithwaite's exercise targeted what he calls "Pseudo-TDD", I intend mine to surface habits we can think of as "pseudo-GP": the sense that it's cheap and painless to *just shut it off and start over* when we start sensing a problem is arising.

1.2 Caveats

What I describe here should be considered a "thought-experiment" backed up by my own experiences; there are only a few sketchy manual implementations to date. Some readers have imagined that the "automated" System player must somehow be a self-contained and tested process running on the cloud (and that is an eventual goal), but in all my early tests that role has been played by a handful of simple Ruby scripts, edited between "turns" by the Facilitator to reflect changes made by the User player. In other words, *there need not be an actual autonomous "computer player" in this game*, unless you feel compelled to write one up beforehand, and the game can as easily be played by a Facilitator who *codes* the moves of the System and *plays that role*, plus the User as described.

Further, there is no reason the User player need be a single human being. Indeed, it's common practice in many software development *katas* to work in teams or even as an aggregated crowd of everybody in the room. Many hands make light work, especially on a tight schedule. In other words, these roles are intended to be "notional" rather than definitional.

I should also point out that this game is not a serious suggestion for a new way of working on "real problems", nor as any sort of "training" for newcomers to the field of GP. Rather it is designed as a rigorous and formal exercise in mindful practice, to be undertaken by people already working closely enough with GP systems to recognize the problems when they arise.

2 "TDD as if you meant it"

Keith Braithwaite seems to have first described this training exercise for software developers in 2009 (Braithwaite, 2011). His target was a sense that software developers who thought they were using test-driven development practices (Beck, 2002) 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
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.

[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.

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 Adzic (Adzic, 2009) has said 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.

Adzic passes along some observations in his account of a Tic-Tac-Toe exercise (Adzic, 2009) that are especially interesting for me:

> 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 You 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

[2] In this the sensibility reminds me of the constraint-driven art collective Oulipo (Becker, 2012), who are perhaps most famous for the *lipogram*, a literary work which cannot use a particular letter of the alphabet.

[3] Braithwaite's participants (Braithwaite, 2012) 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).

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 single training case, and expansion of the active training set will have to be made *in response to particular features* of observed progress, not merely on the basis of the assumption that "more will be better".

The result is a painfully incrementalized process, one that focuses on the refinement and eventual correction of an unstoppable search which was *intentionally* "started wrong", and which must be carried out by *doing surgery on the living patient* to correct perceived "pathologies" and "resistance". Along the way, a fraction of the mysteries of "pathology" has the potential to be much clearer and better-defined.

4 Overview

The exercise is structured as though it were a game for two players, plus a Facilitator who establishes the ground rules, provides any needed technical infrastructure, and acts as referee. One player is (or represents) a running GP System, and one player is (or represents) a human User trying to mindfully drive the System's performance in a desirable direction over a series of turns.

While I speak below of the System player "being" a self-contained software process, it is of course a loose role that might more easily be played by another human, potentially the Facilitator herself, writing and running a simple series of scripts on a laptop. Similarly, while I may say that the User player "is" a single human being, it could as easily be a room-full of students or workshop participants, or a mailing list voting over many weeks on strategies for each turn. Indeed, in working out the exercise as it's described here, I've "played" all the roles myself, simultaneously, and still found interesting and unexpected insight.

In preparing the *kata*, the Facilitator selects a *target problem*, which should be a supervised learning task for which plenty of data is available. The target should not be "toy" in the sense of having a simple, well-known answer; rather it should be challenging and open-ended enough to warrant a publication if solved (a problem that has *recently* been solved might do in a pinch, though there is no shortage of open ones). Any good book of mathematical recreations (for example, (Winkler, 2003) or nearly any book by Martin Gardner, Ian Stewart or Ivan Moscovich) will provide numerous abstract problems that have never been attempted with GP.

The Facilitator will also need to choose a representation language, set up an initial Tableau, and provide enough software infrastructure so that the System's turns can be made easily. In each turn of the game, both the System and the User players take actions to modify and extend the initial Tableau: the User's options include adding *rubrics* and *operators* (described below), and in its turns the System acts "mindlessly" by *invoking* the specified rubrics and operators to add a fixed number of new individuals to the growing population.

The User's goal is to drive the System towards producing *sufficiently good solutions* to the target problem, but their decisions are constrained by the obligation to provide a *warrant* for every change made which is convincing to the Facilitator. The particular definition of a "sufficiently good solution" is left for the players and Facilitator to decide in context: it may be impossible to completely "solve" some target problems, given the tools at hand.

It should always be kept in mind that *the point of the exercise is to evoke interesting and useful warrants*, not merely to drive the System in a desired direction. In each turn the User is obliged to produce a convincing *warrant* for every move she makes, which must be reviewed and approved by the Facilitator before play proceeds. These warrants need not be *factually correct*, but they must be *convincing* in the context of the game state at the time they're put forth.

4.1 The Tableau

The game "board" is a *Tableau* with two components: A list of search `operators`, and a two-dimensional spreadsheet-like table which uses `answers` as its row labels and `rubrics` as its column labels. Initially the Tableau is empty, except for a single entry on the list of `operators` labeled "random guess".

The "random guess" operator is set up by the Facilitator before play begins; it produces a single random `answer` for the target problem, with no arguments. Think of it as equivalent to the "initialization" function used to produce a single "random" individual in a traditional GP setting.

During the User's turn, she can examine the state of the Tableau, including any or all of its history, and the algorithms in play, and apply any amount of data analysis or statistical work she wants. On her turn she is permitted no more than *two* moves: She can (optionally) code and append one new `operator` to the list of `operators`, and she can (optionally) code and append one new `rubric` column to that table.

During the System's turn, it will produce a fixed number of new `answers`. It creates each new `answer` by first picking an entry from the list of `operators` with uniform probability. When an `operator` is chosen, it executes the indicated algorithm, selecting parents from the `answers` table as needed. For each required selection of a parent, the scores recorded in the table of `answers` vs `rubrics` are used (see below). Once parents are selected, the `operator` is applied and a new `answer` is immediately appended to that part of the Tableau.

4.1.1 Answers

What I'm calling an `answer` here would probably be called an "individual" in the GP literature. In this case, it is a particular *script* or program in the representation language the Facilitator has chosen for the target problem.

Answers never "die", and cannot be removed from the Tableau by either player.

4.1.2 Operators

Each `operator` is a function which takes as its argument an unordered collection of zero or more `answers`, and which produces a new collection of one or more `answers` as output.

The initial Tableau includes only a single `operator`, which implements a pre-coded “random guessing” algorithm. On the User’s move, she may build and launch other more complex (and familiar) `operators` like crossover or mutation. The only permitted argument is a set of zero or more `answers`; no numerical or other parameters are allowed. A wide variety of `operator` algorithms are still possible, and the details of the code in which they are implemented is left to the Facilitator to specify as part of the game setup.

Except for some unusual edge cases, the *specific* set of `answers` to which an `operator` will be applied cannot be chosen directly by the User. All “parents” defined for each `operator` are chosen independently (and incrementally) by *lexicase selection*, using the suite of `rubrics` in play when the System takes its turn. This lexicase selection process samples every `rubric` in the Tableau with equal probability. It may be possible for the User to design a new “selection” `operator` which takes as its argument “all the answers” and somehow culls that collection down to a subset—but to do so, it must rely on the immediate state of the `answers` it is given; `rubric` scores are “stored” only in the Tableau `answer` table itself, not as gettable attributes of the `answers`, making it very difficult for any new `operator` algorithm to use `rubric` scores in its implementation.

Note again: no mechanism exists which *removes* `answers` from the Tableau.

4.1.3 Rubrics

A `rubric` is a function which returns a scalar value (not necessarily a *number*) for any 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”) for that `answer` in the appropriate cell of the “spreadsheet” portion of the Tableau.

No score is ever changed for an `answer`, once it has been 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 its existence “entails” all of the component `rubrics` from which its score is derived. So, for example, if the User constructs a `rubric` like “maximum absolute error observed over any of these 30 training cases”, on the turn when she adds that `rubric` up to 31 new columns will be added to the spreadsheet portion of the Tableau: one for each of the 30 requisite “sub-`rubrics`”, which produce as their scores the absolute error for a single training case, and also one *aggregate* `rubric` which calculates and reports “maximum of those other 30 scores”. Whenever the User submits such a higher-order `rubric` on her turn, all of the entailed `rubrics` are added automatically if they do not already exist. Note however that there must still be a *warrant* for the `rubric`, which convinces the Facilitator of its potential usefulness.

The User can use any information present in the Tableau in building new `rubric` functions (such as row numbers, string values of `answers`, or scores), and can also examine the detailed state of the interpreter before and after the `script` is run. It should be clear therefore that the User can specify `rubric` functions which score aspects of the problem such as:

- the absolute error for a single training case
- the number of tokens in the `answer`'s script
- maximum error measured in any of 35 other `rubrics`
- number of `div0` errors produced when running a script with a particular set of inputs
- number of stochastic instructions appearing in the `answer`'s script
- rewrite difference between the scored script the most common `answer` in the population
- (and so on)

Various problem-specific aspects are glossed here, and it is left to the Facilitator to be reasonable in the context of the target problem and the representation she has chosen. Suffice to say, the construction of useful `rubrics` in response the System's moves is the core of the User player's game strategy.

4.2 Lexicase Selection

All `operator` inputs are chosen by lexicase selection. Slightly simplifying Spector's original description (Spector, 2012), the following algorithm can be used:

- (beginning with a "set under consideration" which includes all `answers`...)
- for each `rubric` in a random permutation of all the `rubrics` in the Tableau, discard all `answers` from the set under consideration whose score on this `rubric` is sub-optimal (relative to the current set under consideration)
- if multiple `answers` remain after all `rubrics` have been applied, return one `answer` picked randomly, with uniform probability, from the remaining set under consideration

At the beginning of the game when no `rubrics` have been added and no `operators` exist which require "parents", no selection occurs or needs to occur. But note that the algorithm as described would *still* provide a parent, even when no `rubrics` have been specified: an empty set of `rubrics` would be immediately exhausted without eliminating any `answers`, and then a single `answer` would be selected with uniform probability from that complete set.

Lexicase selection has several characteristics which argue for its use here; in particular it is interesting because it seems (anecdotally, but see also Helmuth in this volume) to be a relatively "slow" selection method, permitting a diversity of `answers` to coexist in a population at the same time. However I admit readily that I've specified it as a core part of this *kata* because it is *unfamiliar to most prospective players*, and therefore more likely to produce unexpected behavior of a useful sort.

4.3 The User's turn

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

1. add one `operator` to the list in the Tableau
2. add one `rubric` to the Tableau, which can be "higher-order" and therefore entail others

To support these decisions, the User can examine the Tableau state and history in detail. Before any code is implemented, though, a *warrant* must be written for each one.

4.3.1 Warrants

A *warrant* is a verbal or written argument which spells out the justification for one of the User's moves *in the context of the game as it stands when the move is made*.

Suppose, for example, that the User has not yet added *any* `rubrics`, and would like to add a new one that scores `answers` for a single training case. Since the System is simply making new `answers` with the "random guess" `operator`, it seems perfectly reasonable to warrant this new `rubric` simply by pointing out that selection can't drive search towards better answers without at least one training case being used.

Suppose further that later in the same game the User has added more `rubrics`, such that more than 100 training cases are being used to select parents for the several `operators` in play. In this context, the reasons given for adding a new `rubric` that specifies *just that one particular training case* must surely be very different: perhaps there is a problematic region of the response surface, or a different training case that needs additional "support" to differentiate between two close input states (and so forth).

That said, a warrant does *not* need to be "technical" or even "rigorous", but merely robust enough to be convincing *in the moment*. For example, "I made a crossover `operator` because I think we need to search for new answers 'in between' the parents," sounds to me like a shoddy excuse that invokes received wisdom. On the other hand, "I made crossover so we can drop the variance on this rubric and foster inbreeding of *these* solutions *here*," given a glance at the Tableau and charts on hand, should be more convincing.

Note though that decisions and the warrants that supported them may well prove to have been wrong in hindsight. This in itself is an interesting and useful outcome in the game: when an earlier decision does not actually produce the expected effect in the system, this is a perfect point for the User to be reminded of the apparent inconsistency. Such "failed" warrants shouldn't be rescinded, but it may be appropriate to bring them up as "concerns" in later moves, especially if they begin to accumulate.

There is one particular warrant that the Facilitator should always permit, as long as it isn't abused by the User: "Because I have no idea what will happen when I do this, but I suspect it may be useful in order to [X] later on." In other words, it is perfectly reasonable for the User to state outright they are "exploring" the range of system behaviors. More generally, the basis on which the User, and therefore the Facilitator, decides to make or permit a change should always depend on the history of the game so far. Indeed, reasoning will *always* change dramatically over the course of any interesting game, under the constraint that the Facilitator should never approve a change on the argument that "that's the way we always do it". Every clear and convincing *why* argument will surely be contingent on the immediate state of the system and dependent on all the prior decisions made by the User and Facilitator.

4.4 The Facilitator

The Facilitator is responsible for picking a representation language and target problem, for providing technical infrastructure as required for the *kata*, and for approving and implementing any changes to the System that the User player makes. As noted several times above, *warrants* are the focus of the exercise, and as such great care should be taken that each decision is well-justified.

4.4.1 Choosing a representation 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, there should be multiple paths to success. So for example if the problem's solution could reasonably be expected to involve ordered lists, then it would be best if the language had at least two different ways to "use lists", for example with iterators, recursion, a comprehensive set of second-order functional operators, an explicit `List` type with associated methods, and so forth.

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 thus the 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. Unfamiliarity can be a useful constraint here.

4.4.2 Target problems

As with the choice of language, 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 an 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 solves it quickly without much input from the User, then the "problem" addressed in the exercise can be extended by the Facilitator to one of driving the System to find a second *dissimilar* solution... still without restarting the search process, of course. Keeping in mind that the purpose of the exercise is to provoke insight into the justification for particular decisions, this decision (by the Facilitator) can be justified by being a perfectly reasonable event in a real research project, in which intellectual "stretch goals" are commonplace when resources permit.

4.4.3 Initial setup and restrictions

- the only `operator` is "random guess", which creates one new `answer` with an arbitrary script
- no `rubrics` are present
- 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` in the Tableau
- a `rubric` can only be *run* on a given `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 stochastic `answer` will probably end up with different scores in the same field

4.4.4 The System's turn

During its turn, the System player will add a specified 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 select the required number of input `answers`
3. apply the chosen `operator` to the inputs `answers` 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, and delete any extra `answers` that exceed the limit; otherwise, go to step (1)

The number of `answers` created in each turn should be enough to have a chance of providing new and useful information to the User, and not so much that the System state grows out of control. I would suggest 100 or 500 `answers` per turn; this is about the size of the typical "population" for most GP users, and is on a comfortable scale for them.

5 Why: A Warrant

Genetic Programming[4] embodies a very particular *stance* towards the scientific and engineering work of modeling, design, analysis and optimization. I increasingly suspect that social resistance to GP has little to do with the quality of our technical results. Rather it arises from unfamiliarity with GP's very particular "way of working". We in the field have become used to it—perhaps to the point of taking it for granted—but colleagues in other fields have not.

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 I can pastiche here as something like:

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 nonetheless informs and constrains much of our work lives, from fund-raising to publishing reports, producing narratives of our work that run more or less like this: "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

[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.

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.

That said, it is Andrew Pickering who has provided my immediate inspiration for this project.

5.1 On the Mangle of Practice

Andrew Pickering's monograph *The Mangle of Practice* (Pickering, 1995) 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 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 simply 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

[5] Paul Veyne's excellent *Did the Greeks Believe in Their Myths?* (Veyne, 1988) might be an interesting starting point, I suspect.

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 phrases we utter in the course of our work: the system "is acting up"; 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 (see (Koutalos, 2008) for a particularly good assessment of this from a biologist). 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 Mangle, the degree to which we as researchers can successfully *accommodate* the resistance we encounter is *exactly* the degree to which we can say we have made progress in our projects.

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

---

[6] Pickering's word "mangle" and the way he came to choose it are a recursive example of the framework itself:

> ... 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. ...

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—dotted the Ts and crossed the Is, as it were—we're *still* driven to speak of our GP run "doing" things, rather than 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 figure out how the "winning solution" GP has disgorged actually works.

It is worth saying explicitly now (and then again as many times as necessary) 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 by "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. Rather we should say that a GP system is resisting when it is *raising concern or causing dissatisfaction in its human user*. It is inevitably that *human* observer who is driven to the insight needed to provide an accommodating response.

5.2 GP as "mangle-ish practice"

The broader field of machine learning seems to take a much more "linear" stance towards its subject matter than we do in ours, in the sense that the pastiche of the "scientific method" applies. 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 in a qualitatively different realm from its 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. (unexpected things happen here, but it's not our problem)
5. you have solved your problem

This is of course exactly the stance expected in any planning or public policy setting, or any workplace using waterfall project management. And as we know from those cultures, "being surprising" could be the worst imaginable outcome. Given that pressure, it's no wonder that so much of GP research is focused on the discovery of constraining tweaks aimed at bringing GP "into line" with more predictable machine learning tools. 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.

If humans creating real intelligences (for example, other human beings) treated them anything like the way computer scientists insist we treat nascent artificial intelligences, I have no doubt that the resulting murder convictions would be swift and merciless. It is my hope with this contribution to suggest that we might be able to do better than the virtualized serial murder that is our legacy to date.

Consider the poor "GP user" that most of our research seems aimed at, one who is carefully "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 if she 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 her strategy would be one of *intervention*: even if she could only decide after the fact, she should roll back the system to that crucial turning point, make a change aimed at avoiding the mess, and then continue from there.

Lacking (as we do) this miraculous insight, or the tools for understanding the internal dynamics of any particular GP system, on what grounds does it seem reasonable to stop *any* run arbitrarily at a pre-ordained time point and begin again from scratch? Replication, in the sense we are prohibited from reaching in and affecting outcomes, is no better than dice-rolling.

I would much rather say this: Insofar as GP *surprises us*, and since that is its sole strength over more familiar and manageable machine learning frameworks, we must learn to recognize and accommodate the surprises that arise in its use. Some surprises will always remain disappointments, but the senseless restriction we impose on engaging the systems we build blocks us from seeing others as **encouraging opportunities to improve our plans before it's too late**.

References

Adzic G (2009) Tdd as if you meant it – revisited. URL http://gojko.net/2009/08/02/tdd-as-if-you-meant-it-revisited/

Beck K (2002) Test Driven Development: By Example. Addison-Wesley Professional

Becker DL (2012) Many Subtle Channels: In Praise of Potential Literature. Harvard University Press

Braithwaite K (2011) Tdd as if you meant it. URL http://cumulative-hypotheses.org/2011/08/30/tdd-as-if-you-meant-it/

Braithwaite K (2012) Tdd as if you meant it (workshop recording). URL http://www.infoq.com/presentations/TDD-as-if-You-Meant-It

Hornby GS (2009) A steady-state version of the age-layered population structure EA. In: Riolo RL, O'Reilly UM, McConaghy T (eds) Genetic Programming Theory and Practice VII, Genetic and Evolutionary Computation, Springer, Ann Arbor, chap 6, pp 87–102, DOI doi:10.1007/978-1-4419-1626-6_6

Koutalos Y (2008) "the docile body of the scientist". In: Andrew Pickering KG (ed) The Mangle in Practice: Science, Society, and Becoming, Duke University Press

Pickering A (1995) The Mangle of Practice: Time, Agency, and Science. University of Chicago Press

Ryan C, Nicolau M (2003) Doing genetic algorithms the genetic programming way. In: Riolo RL, Worzel B (eds) Genetic Programming Theory and Practice, Kluwer, chap 12, pp 189–204, DOI doi:10.1007/978-1-4419-8983-3_12, URL http://www.springer.com/computer/ai/book/978-1-4020-7581-0

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

Thomas D (2013) Codekata: How it started. URL http://codekata.com/kata/codekata-how-it-started/

Veyne P (1988) Did the Greeks Believe in Their Myths?: An Essay on the Constitutive Imagination. University of Chicago Press

Winkler P (2003) Mathematical Puzzles: A Connoisseur's Collection. A K Peters/CRC Press