TIG Challenge Design Guide

The Innovation Game
John Fletcher | Aoibheann Murray | Jack Fan

April 25, 2025

1. Introduction

The Innovation Game (TIG) is a protocol designed to accelerate algorithmic innovation by turning difficult computational problems into competitive challenges. It creates a "synthetic market"[1] where **Innovators** contribute new solving methods and **Benchmarkers** compete to solve problem instances using those methods for token rewards. This framework allows problems of high intrinsic scientific importance (even those which may not yet be monetisable in industry) to attract effort and funding. In TIG, the competition among Benchmarkers effectively generates demand for solutions to the chosen problem, providing a way to assess performance and capture value from solution methods. Every TIG challenge must satisfy several key criteria to be suitable. Specifically, a problem proposed as a challenge should be **important**, **asymmetric**, **independent** and allow **random instance generation**. We outline these requirements below:

- **Importance:** The problem should address a significant question in science or technology. Importance can mean practical relevance (e.g. solving it unlocks valuable applications or has major societal impact) or fundamental intellectual merit (e.g. solving it would deepen scientific understanding or link together important theoretical domains). Notably, "important" need not mean immediately profitable or commercially applicable—a purely scientific challenge can qualify if its resolution would be a notable breakthrough. See 2.2, below.
- **Asymmetry:** The challenge must be difficult to solve but easy to verify; in the context of TIG, we call this property "asymmetry". In practice, finding a solution for a given instance should require significantly more computational effort (time, resources) than checking whether a proposed solution is correct. The asymmetry condition is essential for TIG's economic model and is a strong prevention against Sybil attacks. (Importantly, a problem need not necessarily be NP-hard or "formally" hard; it only needs to exhibit a large gap between solution time and verification time for the instance sizes of interest.) See 2.3, below.
- **Independence:** The challenge should be sufficiently independent from other challenges in TIG. Ideally, a proposed challenge should be such that an optimization or algorithm that drastically improves performance on this challenge should *not* automatically yield comparable improvements on other active challenges. See 2.4, below.
- **Random Instance Generation:** It must be possible to generate unlimited pseudorandom instances of the problem, with adjustable parameters that affect difficulty. This allows Benchmarkers to continually compete on new instances and enables the challenge difficulty to scale over time. Each instance should be representative of the real-world version of the problem to the extent possible, so that progress in TIG translates to real progress on the underlying problem. See 2.5, below.

TIG incentivises the development of more efficient algorithms progressively raising the difficulty of challenge instances. Benchmarkers typically start by solving relatively small or easy instances, and as they compete for rewards, they will attempt instances with higher difficulty settings. This creates a built-in pressure to

[1] TIG introduces a market mechanism for algorithms; see the TIG whitepaper (J. Fletcher *et al.*, 2024) for an overview of its value capture strategy, including a hybrid open/commercial licensing model (tig.foundation).

innovate: over time, methods must improve to handle larger or more complex instances efficiently. Any approach that can find solutions at lower cost, such as a method which uses fewer computational steps, will give Benchmarkers an edge and will likely be widely adopted. We note that, even if a challenge begins with simpler instances than those found in industry applications, the competitive dynamic of TIG will drive the state-of-the-art forward until it approaches real-world scales.

2. Challenge Checklist

When proposing a new TIG challenge, the following sections should be included in your challenge description. Each section addresses a critical aspect of the problem formulation and ensures the challenge meets TIG's requirements:

2.1 Problem Description and Formulation

Clearly define the problem being posed as a challenge and outline its objectives and scope. This section provides the foundation for understanding what the challenge is about and what solving it entails.

- **Objectives:** State the goal of the challenge. What constitutes a valid solution or a successful outcome? For example, is the aim to find any solution that satisfies certain conditions, or to find an optimal/minimal solution according to some metric?
- **Constraints:** List any constraints or rules that solutions must obey. These could include input domain restrictions, resource or time limits for solvers, allowable techniques, or other conditions that define the legitimate solution space.
- **Scope:** Clarify the boundaries of the problem. What is included in the challenge, and what is explicitly out of scope (if anything)? It is often useful to mention not only what the challenge covers, but also what it deliberately does not cover. State any assumptions you are making about the problem or the environment (for instance, idealised conditions, glossing over certain complexities, etc.) that are relevant to how the challenge is formulated.
- **Background:** (If applicable) Provide brief context on how this problem has been addressed previously. Mention any existing algorithms, known results, or prior competitions related to the problem. Citing relevant prior work or literature can help justify the challenge and show how it builds on or differs from past approaches.

2.2 Importance: Practical and/or Scientific

Explain why this problem is important and worth tackling. Justify the significance of the challenge either in practical terms, scientific terms, or both.

- **Practical Applications:** If the challenge has direct practical or industrial relevance, describe the potential applications. How could a better solution to this problem benefit society, industry, or other fields? Be specific about any use cases or downstream technologies that would be enabled or improved.
- **Scientific Significance:** If the problem is more on the fundamental or theoretical side (e.g. a pure mathematics or basic science question), explain its intrinsic importance. For example, does solving this challenge fill a crucial gap in knowledge, test a longstanding hypothesis, or connect two areas of research? Emphasise why progress on this problem matters for science.

2.3 Asymmetry Characteristics

Demonstrate that the challenge is "asymmetric": hard to solve, but easy to verify. Discuss the computational complexity (especially time complexity) of solving an instance of the problem versus verification of a given solution. Ideally, provide rough comparisons (for example, solving might be exponential or require heavy computation, whereas verification might be polynomial-time). Make it clear why solving will always take

significantly longer in expectation than checking a solution for the instance sizes of interest. It is also important to outline how a solution would be verified in practice within TIG. Typically, verification means an independent part (e.g. the network or a smart contract) checks if the solution satisfies the challenge instance. For example, in SAT, the verifier checks if the challenge instance evaluates to True with the solution variable assignments. If your challenge requires additional steps for verification (such as recomputing a known baseline result or other auxiliary calculations: see 2.6, below), note what those are. All components of the verification process should be efficient enough that they are small compared to the time it takes to find the solution in the first place. Ensuring this gap in effort is crucial for the challenge to qualify as asymmetric. Problems can satisfy this asymmetry requirement in several ways:

- **Inherently Asymmetric Problems**: Some computational problems naturally possess this structure. For example, Boolean satisfiability (SAT) is classically known for this property.
- **Transformable Problems**: Some problems can be reshaped in order to achieve asymmetry. For instance, many optimisation problems can be turned into decisional variants for which verification is efficient.
- **Inverse Problems**: Many scientific applications require calculation of initial conditions or parameters that produce specific outcomes. While a forward simulation may not be efficient to verify, the inverse problem (e.g. initial conditions that lead to desired outcomes) is typically asymmetric. Often, it is the inverse problem that is ultimately of interest.

2.4 Challenge Independence

Consider the relationship of this challenge to other challenges in TIG. Explain how your challenge is sufficiently independent. The core idea is that progress or optimisations in this challenge should not immediately translate to solving other challenges. If the same algorithmic technique or hardware improvement could be applied to many different challenges, then those challenges are not independent. Describe any connections or similarities to existing TIG challenges or well-known problems, and argue why your chosen problem requires distinct methods or approaches. By demonstrating that excelling at your challenge won't give a competitor an automatic advantage on different challenges, you show that it stands on its own as a valuable addition to TIG.

2.5 Random Instance Generation

Describe how to generate random instances of the problem for benchmarking. A TIG challenge requires a method to produce many varied problem instances at different difficulty levels. Detail the procedure for creating a new instance on demand. The goal is to have a supply of new, unpredictable instances for solvers to tackle, preventing overfitting to any fixed set of examples. Crucially, the instances in TIG should mirror the real-world version of the problem as closely as possible. If there is any way in which the synthetic instances differ from real cases (for example, perhaps the instances are smaller, or assume ideal conditions that real-world problems don't), explain why solving these synthetic instances is still relevant. You might argue, for example, that as the difficulty parameters increase, the random instances will approach the complexity of some real-world applicaiton of interest ; or that there is reason to believe that algorithmic improvements on simplified instances will generalise to those of a real-world problem. If generating an instance requires calculating some known value (like a target or baseline), mention that here (and see the next section for baseline details).

2.6 Baseline Calculation (if applicable)

If the challenge does not inherently have efficient verification, it may be possible to introduce a "baseline" in order to transform it into a decision problem that is asymmetric. In such cases, outline how to compute a baseline solution or threshold for the problem. Not every challenge will need a baseline. There are inherently asymmetric decision problems (e.g. SAT or other problems where any given solution can be checked quickly) so a baseline isn't necessary. However, for many optimisation problems, simply checking a candidate solution's quality could be fast, but finding that solution from scratch is hard. To adapt these for

TIG, you can set a baseline performance that solvers must exceed. Explain what baseline you will use and how to compute it quickly:

- You might have an analytical solution or formula for a simple version of the problem that gives a baseline result.
- Or you could run a fast, greedy or approximate algorithm to produce a baseline solution (one that is not optimal but can be found very quickly).

The baseline should be something that can be computed much faster than the expected time for a completive algorithm (in TIG) to find a high-quality solution. Remember that generating this baseline value will be part of each instance generation (and thus part of the verification process). Keeping the baseline computation lightweight helps maintain the overall asymmetry of the challenge. Also describe how the baseline is used in evaluation. The baseline acts as a reference point to judge solution quality. Please note that this approach works best if the baseline's performance is reasonably consistent relative to the true optimum across different instances. If there is high variance (i.e. sometimes the baseline is close to optimal and other times far off), consider tuning the baseline method to be more stable. The idea is that beating the baseline by a given factor is a reliable indicator of a high quality solution.

2.7 Difficulty Parameters

Define one or more **difficulty parameters** that control the hardness of an instance, and explain how they can be adjusted to make the challenge easier or more difficult. These parameters will be used to scale the challenge over time. Common difficulty parameters include, for example:

- **Problem size:** Increase the size or complexity of the input (e.g. number of variables, number of constraints, dataset size, etc.). Larger problem instances typically require more computation, making them harder to solve.
- **Solution quality threshold:** If the challenge uses a baseline, you could control difficulty be adjusting the standard that solutions must meet (requiring a better objective value relative to the baseline for higher difficulty).

Make sure at least one difficulty parameter is unbounded or can grow without a fixed limit. This ensures there is no inherent maximum difficulty—the challenge can continue to scale up as needed. Other parameters can have upper bounds if appropriate (where these bounds may depend upon other difficulty parameters), as long as the overall configuration space allows for arbitrarily hard instances. It's worth noting that in the TIG competition, Benchmarkers will select the difficulty of the instances they attempt to solve, balancing the risk of solving a non-qualifying difficulty against the risk of failing to solve them. This creates a competitive dynamic where the effective difficulty keeps increasing as long as solvers keep improving. Your challenge's parameters should be designed to support this: there should always be a harder level to push to. (For more on how difficulty ties into TIG's reward mechanism and the concept of a Pareto frontier of difficulty vs. success, see the TIG documentation on OPoW.[2])

2.8 Example

Finally, provide a concrete example to illustrate one instance of your challenge and how it would be solved and verified. Walk through the process step by step: For example, you might describe a specific instance generation: choose particular difficulty parameter values, generate a problem instance (show a sample input), compute the baseline solution for this instance (if applicable), and then discuss what a solver would need to do to produce a valid solution. Then explain how that solution would be checked against the instance's requirements (and baseline, if used). This example helps readers and evaluators understand the challenge in practical terms. It demonstrates that you have thought through the implementation details and that the challenge is well-defined end-to-end (from instance creation to solution verification).

[2]Details on difficulty selection and the OPoW reward mechanism can be found at docs.tig.foundation/opow.

3. Common Issues and Emulators

Due to the nature of the TIG protocol, issues can often come up in challenge design which limit the usefulness of innovation to real-world applications. Some common issues include:

- **In-built structure:** To imitate real-world instances, part of the generation may give Sybil attackers an underlying structure in the challenge instance. This could give Sybil attackers an almost analytical way to produce a "correct" solution for much more cheaply than it would take to verify, thus undermining the necessary asymmetrical aspect of the challenge. Although these would not qualify for rewards as they would not pass the final WASM verification, they could trigger many instances for WASM verification which is computationally expensive, leading to a Sybil attack. A solution for this is to utilize the underlying structure in the baseline algorithm, rather than having the baseline algorithm start from scratch. Legitimate algorithms will drive the *better_than_baseline* difficulty parameter high enough that it would cost much more compute to find a "correct" solution even when given the underlying structure.

- **Difficulty does not scale with difficulty parameter:** As challenge size scales, sometimes it doesn't necessarily result in an increasing difficulty. For example, in the 0/1 single knapsack problem, increasing the number of items was equivalent to cutting up existing items into smaller parts, which could then just be fitted into the knapsack as before. This resulted in the challenge difficulty not increasing as the difficulty parameter increased thus limiting the usefulness of innovation submitted to the challenge. Linear problems often have this issue. This can be fixed by updating the 0/1 single knapsack problem to the quadratic knapsack problem, which doesn't exhibit this issue.

- **Near-optimal baseline algorithms:** In some problems due to the instance parameters, greedy-heuristics that are used commonly for baseline algorithms can generate near-optimal solutions. Along with the *better_than_baseline* difficulty parameter, it can result in a target quality (e.g. travel distance) better than the exact solution, thus making it impossible for the benchmarker to find a qualifying solution. This is unlikely to happen if the challenge difficulty scales correctly with the difficulty parameters (e.g. size complexity), especially when at sizes generally found in real-world applications.

- **Large instance generation takes too long:** If any component of the instance generation (such as baseline algorithm of instance initialization) scales expensively as problem complexity increases, this may result in instance generation becoming too expensive once the challenge scales into higher difficulty parameters. As instance generation is part of the initial verification process, a Sybil attack can be done by submitting random and invalid solutions thus forcing the verifier to run the expensive instance generation.

TIG labs has recently started experimenting with emulators for instance generation. Emulators are useful in TIG challenges as they can generate challenge instances through a blackbox, hiding any underlying structure from benchmarkers that may have been used during instance generation without emulators (e.g. via simulation). Unlike in other applications, emulators in TIG are not necessarily difficult to set up. In most cases, generating training data for emulators are the most time-consuming aspect. In TIG, however, instances generation can be very fast, and a large dataset used to train the emulator can be generated relatively cheaply. If you think emulators may be an efficient way to generate instances, we highly encourage exploring it.