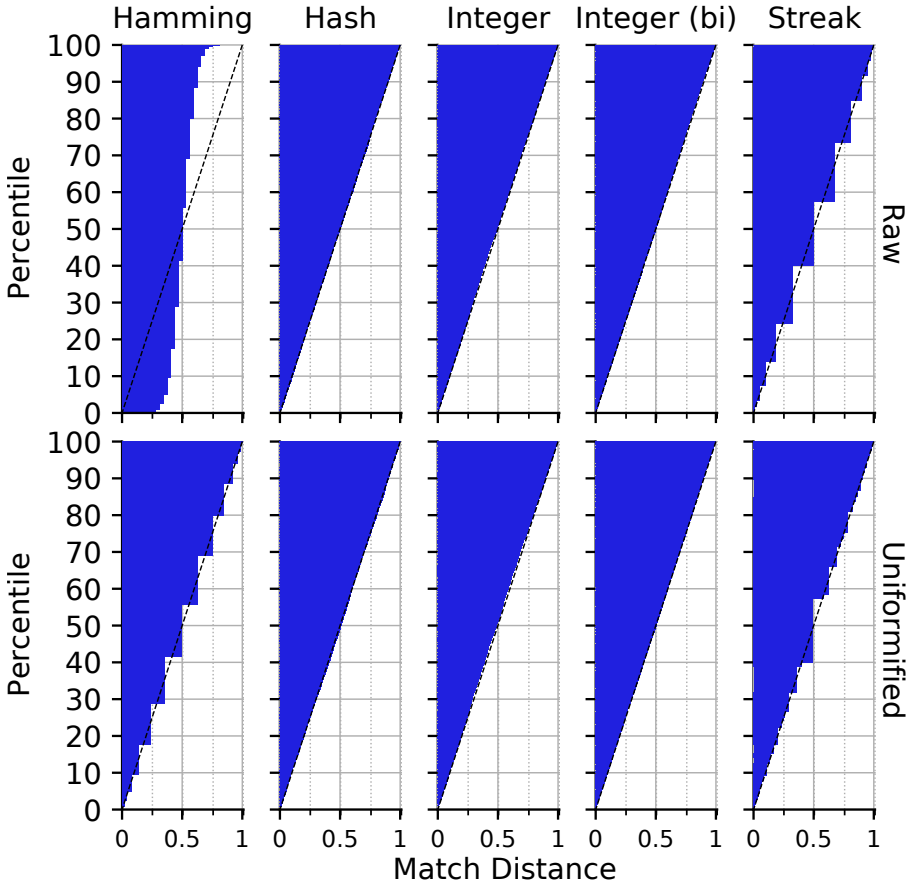


Fig. 14: Distance distributions of metrics before and after uniformification. Each visualization arranges individually sampled observations (thin horizontal bars) vertically in descending order. The y axis can be interpreted as ranging from the 0^{th} percentile of outcomes (bottom) to 100^{th} percentile (top) with width of horizontal bars showing match distance at a certain percentile. Dashed lines trace an ideal uniform distribution.

A Notes on Tag Alphabet

For tractability and consistency, this work exclusively considers strings composed from the binary alphabet $\{0, 1\}$. However, we expect that most geometric, variational, and evolutionary properties of the metrics studied are not fundamentally tied to the particular use of the binary alphabet.

We suspect that the surveyed integer metrics under the existing bitstring representation should behave effectively indistinguishably from a continuous-valued (i.e., floating point) representation. Due to the uniformification process performed, both would be effectively

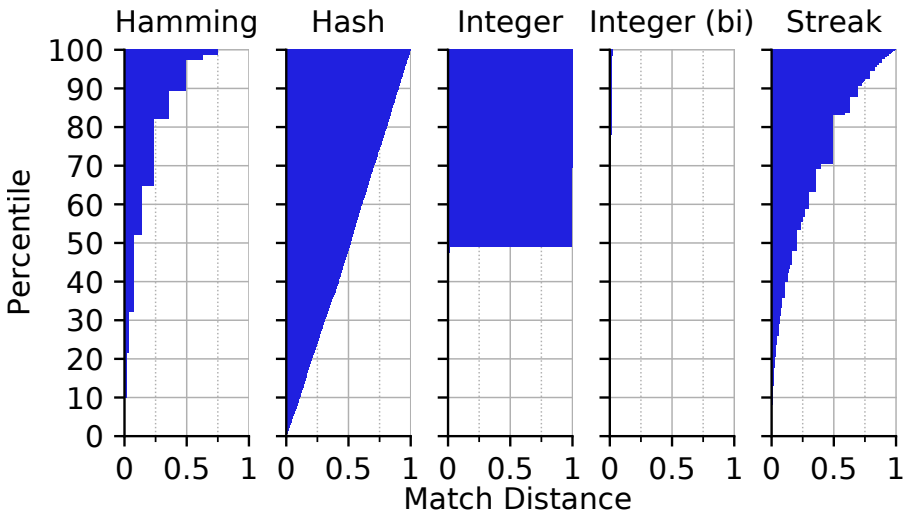


Fig. 15: Cumulative distributions of sampled similarity constraint values. Each visualization arranges individually sampled observations (thin horizontal bars) vertically in descending order. The y axis can be interpreted as ranging from the 0^{th} percentile of outcomes (bottom) to 100^{th} percentile (top) with horizontal bar width showing similarity constraint at a certain percentile.

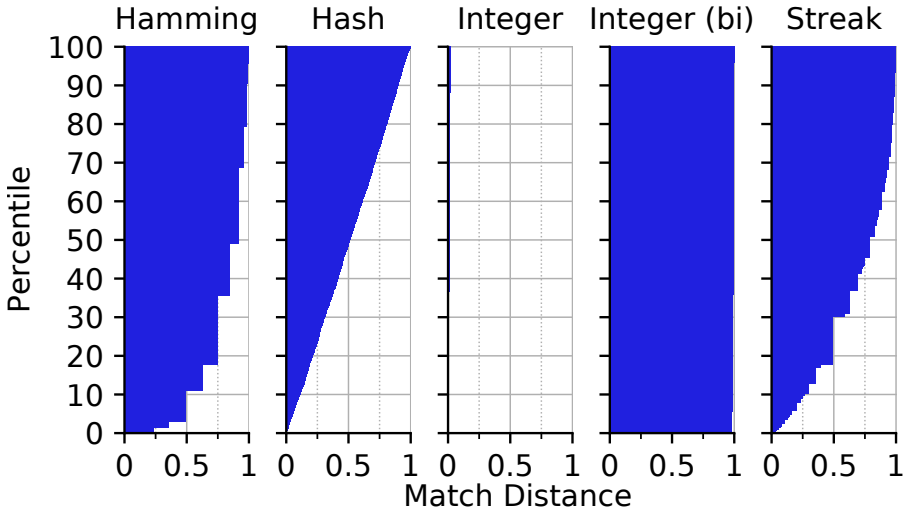


Fig. 16: Cumulative distributions of sampled dissimilarity constraint values. Each visualization arranges individually sampled observations (thin horizontal bars) vertically in descending order. The y axis can be interpreted as ranging from the 0^{th} percentile of outcomes (bottom) to 100^{th} percentile (top) with horizontal bar width showing similarity constraint at a certain percentile.

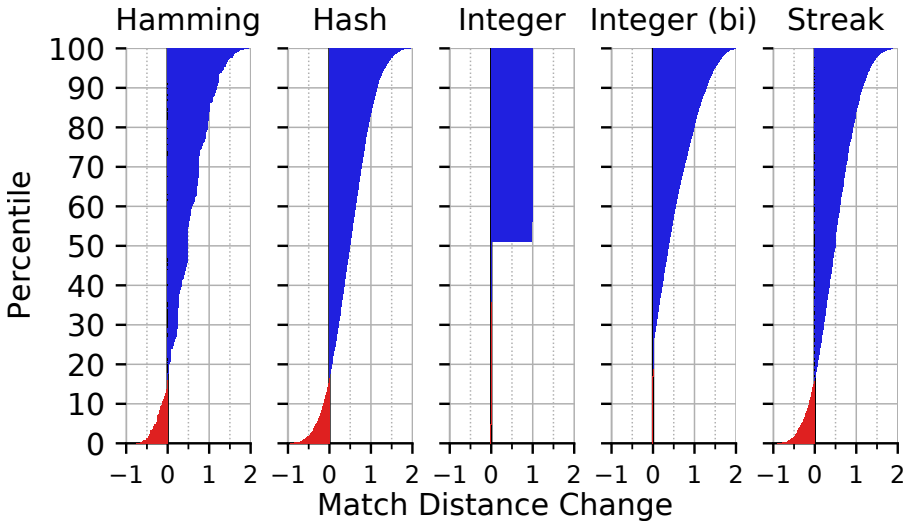


Fig. 17: Cumulative distributions of sampled detour difference values. Each distribution visualization arranges individually sampled observations (thin horizontal bars) vertically in descending order. The y axis can be interpreted as ranging from the 0^{th} percentile of outcomes (bottom) to 100^{th} percentile (top) with horizontal bar width showing the detour difference at a certain percentile. A positive value (colored blue) indicates that total distance increased with the addition of an intermediate stop. A value of exactly 0 indicates an intermediate stop had no effect on total distance. A negative value (colored red) indicates violation of the triangle inequality: taking an intermediate stop reduced the total distance traveled.

rescaled to the range $[0, 1]$. With a precision of $1/2^{32}$ — tighter than 10^{-9} — the 32-bit tags used should exhibit near-undetectable granularity, especially given the relatively small pools of query and operand tags used in our experiments.

However, it is important to note that the bit flip mutation operator used in our experiments induces a roughly exponential distribution of mutational effect size, which might otherwise be an unusual choice when working with a continuous-valued tag system. We unpack this issue in greater detail in Section 4.

Alternate alphabet choice would have a minimal effect on the streak metric. Imagine, for example, using a four-valued alphabet instead of the existing two-valued binary alphabet. Any character in that four-valued alphabet could be encoded by a pair of binary digits. So, the existing bitstring representation for tags could be preserved and adjustment instead made to the match distance metric to count only entirely-matching (or mismatching) pairs of bits as contributing to a streak. The significance of this effect would depend on typical streak length and, of course, for large alphabets this truncation effect would eventually become overwhelming.

Increased alphabet size might have a more nuanced effect on the Hamming metric. Under the binary alphabet, every mutation affects a tag's match distances to all other tags — no mutation is neutral. However, with a larger alphabet size this would no longer be the case. As with the streak metric, increased alphabet size would introduce effects from coarsened granularity, with the magnitude of these effects eventually becoming overwhelming under large alphabets.

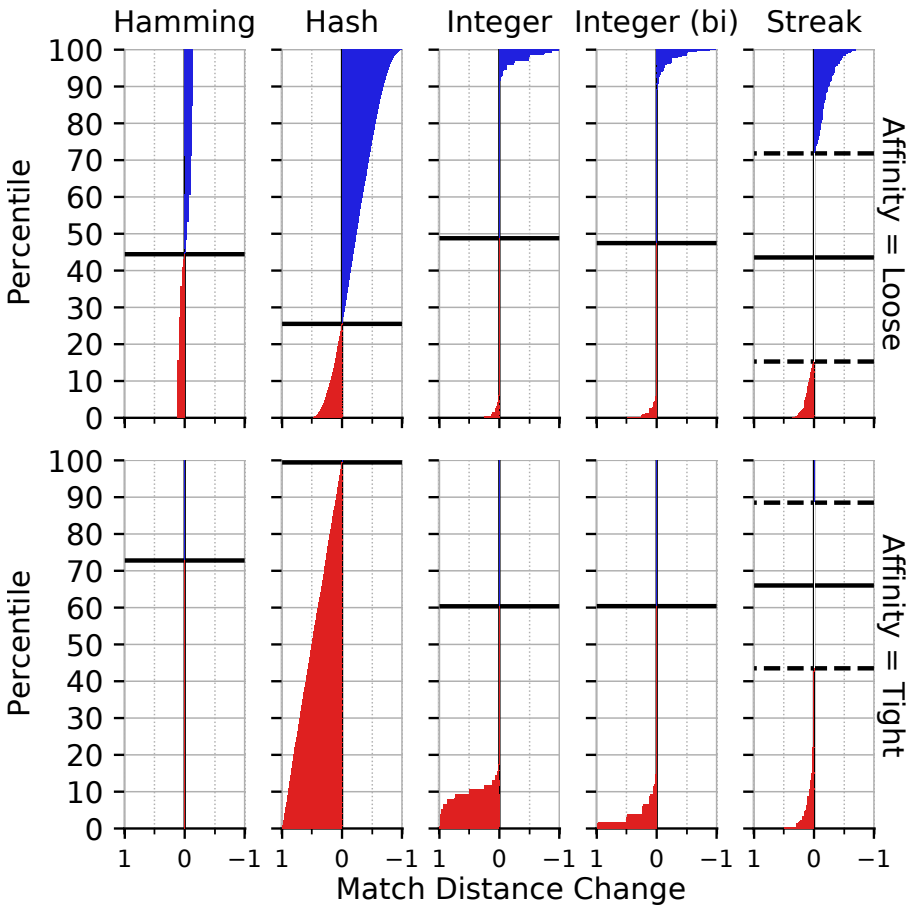


Fig. 18: Distributions of mutation effects on match distance for loosely matched (pre-mutation match distance > 0.5) and tightly matched (pre-mutation match distance < 0.01) tag pairs. Each distribution visualization arranges individually sampled observations of mutation outcome from an independently sampled tag pair (thin horizontal bars) vertically in descending order. The y axis can be interpreted as ranging from the 0^{th} percentile of outcomes (bottom) to 100^{th} percentile (top) with horizontal bar width showing the mutation effect size at a certain percentile. Mutations that increase affinity are colored blue and mutations that decrease affinity are colored red. Solid lines indicate the median between mutations that increase match distance and mutations that decrease match distance. Dashed lines demarcate the boundaries between non-neutral and perfectly-neutral mutations. Note that the x axis is inverted so mutations increasing affinity fall to the right and mutations decreasing affinity fall to the left.

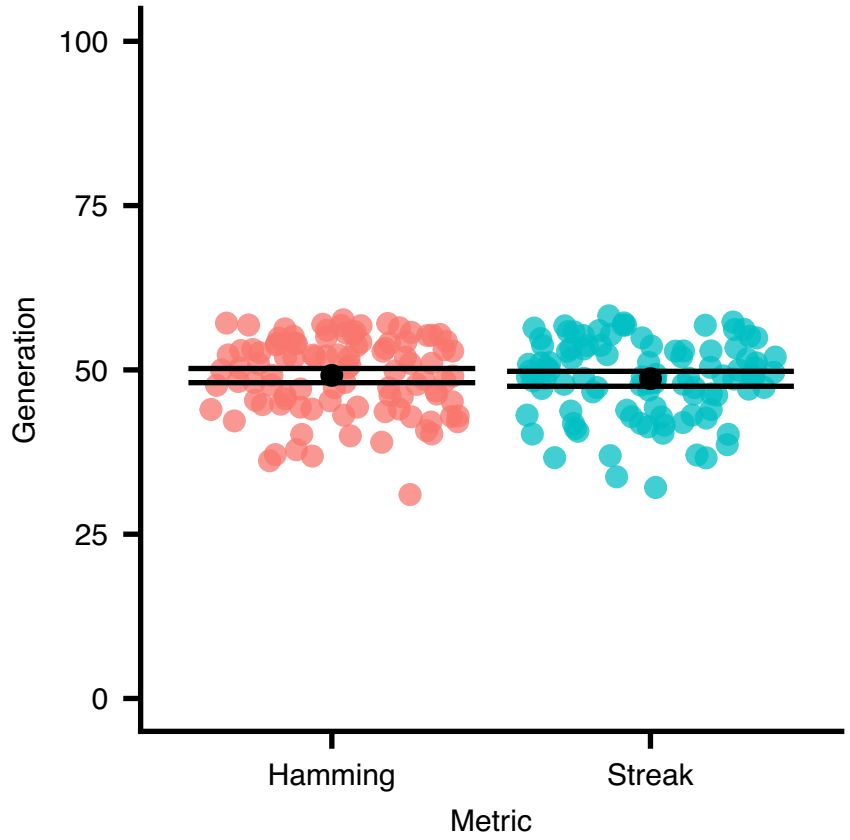


Fig. 19: Generations to solution for the first 100 replicates out of 200 to produce a complete solution to the changing-signal task. Error bars indicate bootstrapped 95% confidence intervals.

We do not fully explore the possibilities introduced by alternate tag-matching representations in this work, so a detailed and rigorous understanding of this topic remains an avenue for future research.

B Graph Matching with Normally-distributed Mutation Operator

In order to contextualize our use of bitwise mutation with integer metrics, we replicated the 32-vertex target graph matching experiment with randomly-initialized genomes reported in Section 5.1 using a normally-distributed mutation operator.

Under the normally-distributed mutational operator, each tag had an integer amount drawn from a normal distribution added to its integer representation every generation. Specifically, the amount added was drawn from $\mathcal{N}(0, m \times c)$, where m was a variable mutation rate parameter and c was a fixed scale parameter $2^{32} - 1$. (Recall that $2^{32} - 1$ is the largest possible Overflow values after addition were wrapped around.

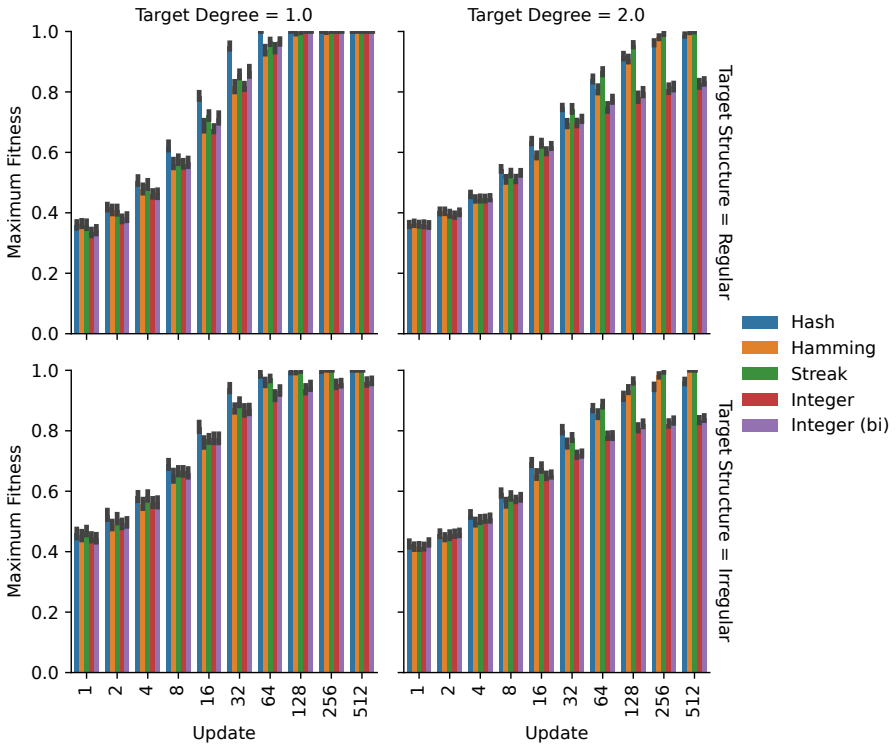


Fig. 20: Trajectories of adaptive evolution for each tag-matching metric on the 32-vertex graph-matching task with randomly-initialized initial genomes. Maximum fitness represents the best fitness value for any individual within a population. Reported results use each metric's best-performing per-bit mutation rate. (See Supplementary Figure 21 for survey of how mutation rate affects adaptive evolution under each metric.) Note logarithmic x axis. Error bars represent bootstrapped 95% confidence intervals across 50 replicate observations.

A swath of 15 standard deviations m between 0.000976562 and 0.125 were surveyed. All metrics had a best-performing mutation rate within the range of surveyed rates for each problem configuration tested. Supplementary Figure 26 summarizes performance across surveyed mutation rates. We used the best-performing mutation rate for each metric on each problem configuration for further analysis. Supplementary Table 4 provides the best-performing mutation rates used.

Supplementary Figure 24 shows adaptation over generations under this mutational operator for regular/irregular target graphs with mean degree 1/2. The normally-distributed mutation operator performs comparably to the bitwise mutation operator or worse in all instances.

It may be possible to achieve better performance with a mutation operator that combines a per-tag mutation probability with a normally distributed mutational effect. Further work will be required to explore such possibilities.

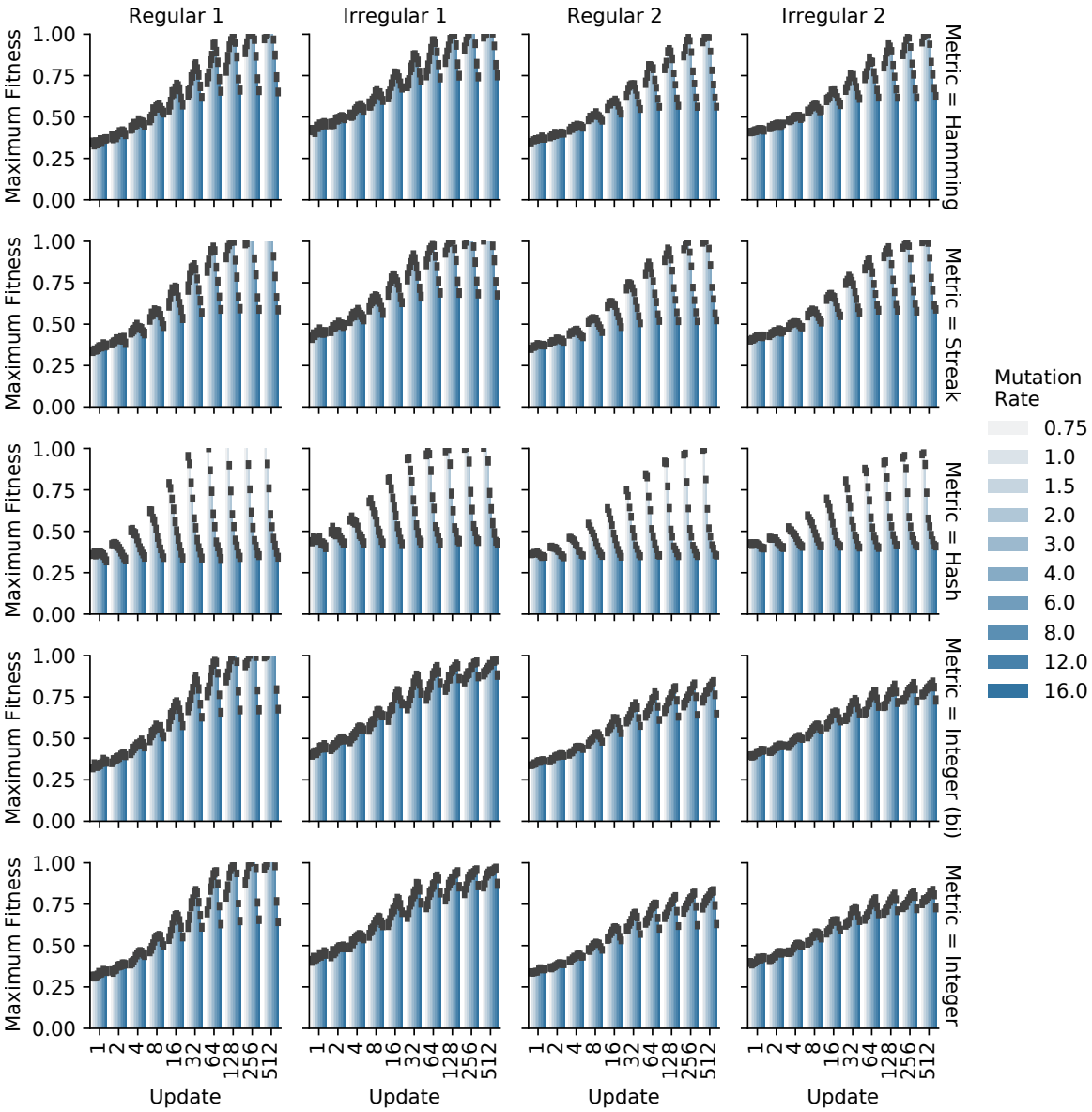


Fig. 21: Survey of adaptive evolution rates across mutation rates for 32-vertex graph-matching task with randomly-initialized initial genomes. Metrics exhibited fastest adaptive evolution within the range of mutation rates surveyed, except the hash metric which exhibited fastest adaptive evolution at at the lowest mutation rate surveyed. Maximum fitness is the best fitness value for any individual within a population. Maximum fitness at each update is presented across the range of surveyed mutation rates. Error bars represent bootstrap 95% confidence intervals across 50 replicate populations.

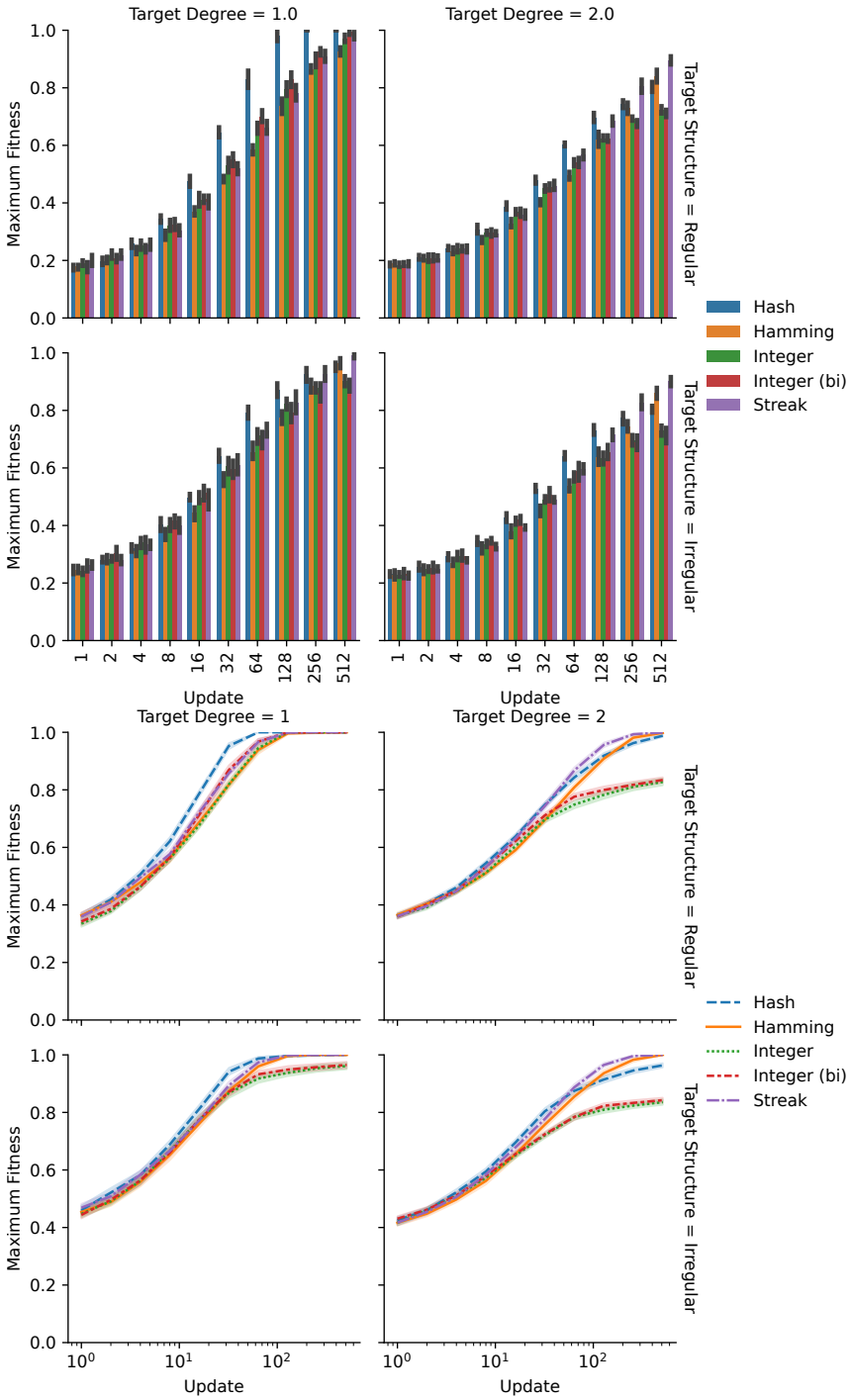


Fig. 22: Trajectories of adaptive evolution for each tag-matching metric on the 64-vertex graph-matching task. Maximum fitness is the best fitness value for any individual within a population. Reported results use each metric's best-performing per-bit mutation rate. (See Supplementary Figure 23 for survey of how mutation rate affects adaptive evolution under each metric.) Note logarithmic x-axes. Shaded area and error bars represent bootstrapped 95% confidence intervals across 10 replicate observations.

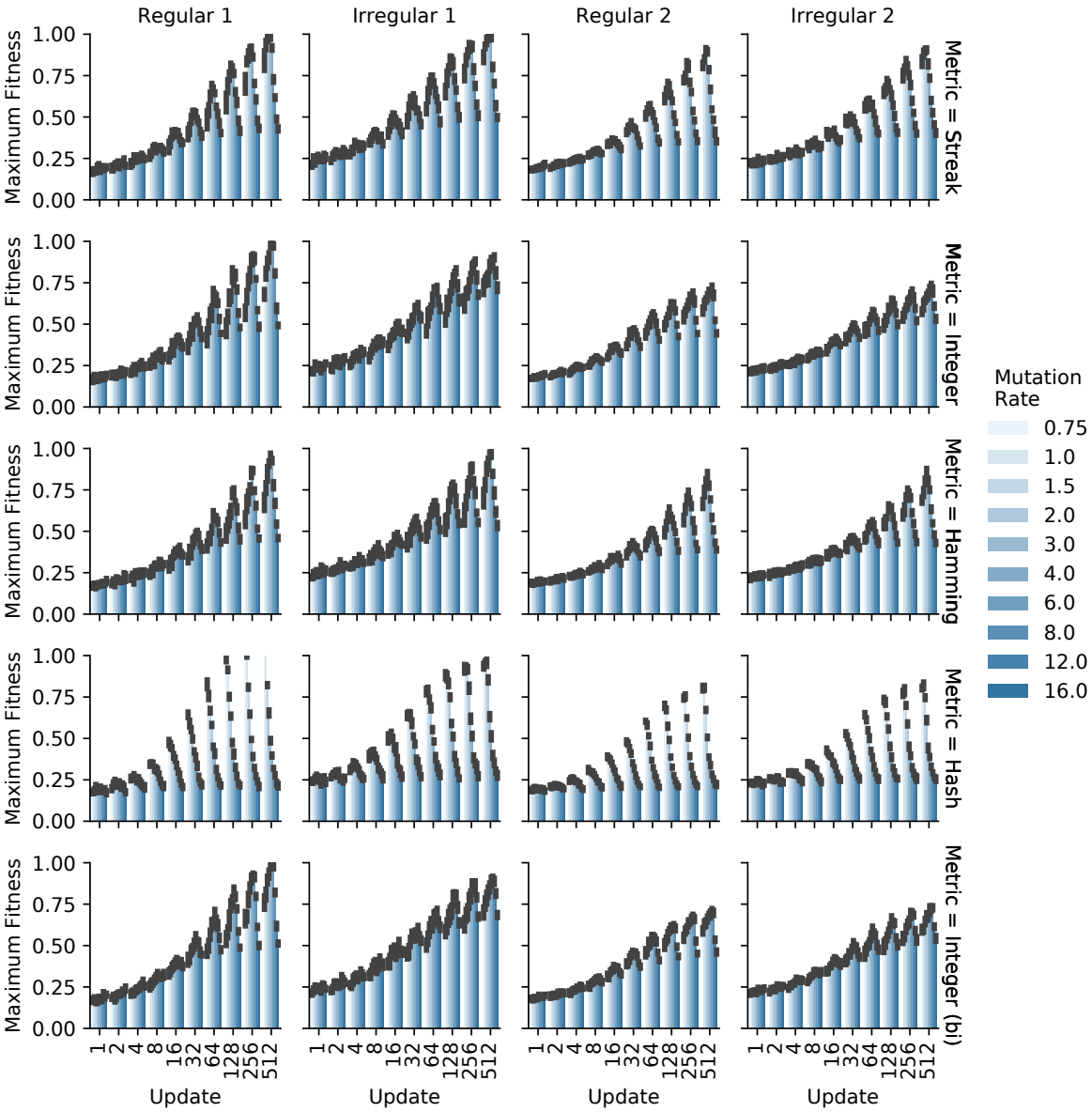


Fig. 23: 64-vertex graph-matching task mutation rate sensitivity analysis. Metrics exhibited fastest adaptive evolution within the range of mutation rates surveyed, except the hash metric which exhibited fastest adaptive evolution at at the lowest mutation rate surveyed. Maximum fitness is the best fitness value for any individual within a population. Maximum fitness at each update is presented across the range of surveyed mutation rates. Error bars represent bootstrap 95% confidence intervals across 10 replicate populations.

Metric	Target Structure	Target Degree	Best-Performing Per-Genome Bit Mutation Rate
Hash	Regular	1	0.75
Hash	Regular	2	0.75
Hash	Irregular	1	1.0
Hash	Irregular	2	0.75
Hamming	Regular	1	4.0
Hamming	Regular	2	2.0
Hamming	Irregular	1	4.0
Hamming	Irregular	2	2.0
Integer	Regular	1	6.0
Integer	Regular	2	6.0
Integer	Irregular	1	8.0
Integer	Irregular	2	6.0
Bidirectional Integer	Regular	1	4.0
Bidirectional Integer	Regular	2	6.0
Bidirectional Integer	Irregular	1	8.0
Bidirectional Integer	Irregular	2	4.0
Streak	Regular	1	3.0
Streak	Regular	2	2.0
Streak	Irregular	1	3.0
Streak	Irregular	2	1.5

Table 3: Best-performing per-bit mutation rates for 64-vertex graph matching tasks. See Supplementary Figure 23 for performance across surveyed mutation rates.

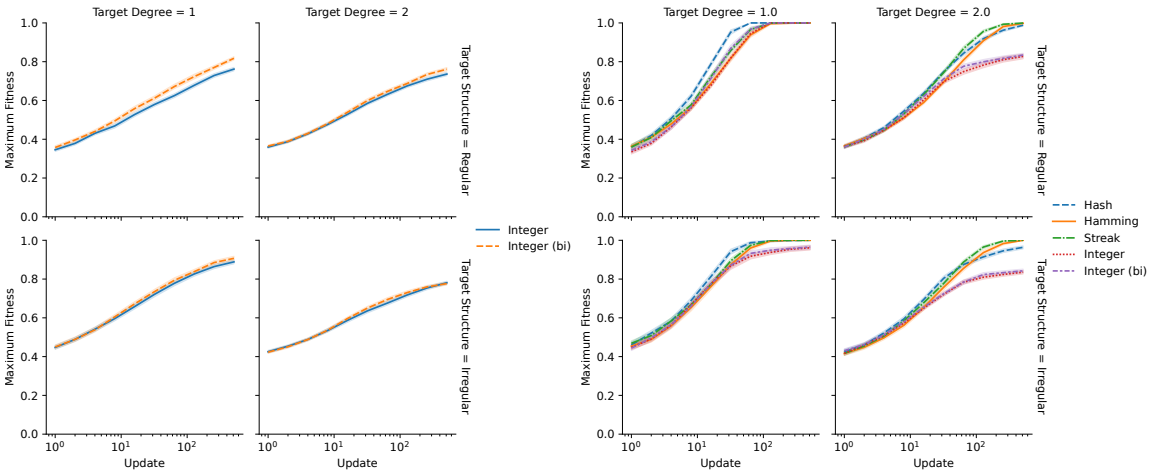
Metric	Target Structure	Target Degree	Best-Performing Mutation Standard Deviation
Integer	Regular	1	0.0117188
Integer	Regular	2	0.0117188
Integer	Irregular	1	0.015625
Integer	Irregular	2	0.0117188
Bidirectional Integer	Regular	1	0.0117188
Bidirectional Integer	Regular	2	0.0117188
Bidirectional Integer	Irregular	1	0.015625
Bidirectional Integer	Irregular	2	0.015625

Table 4: Best-performing mutation rates for 32-vertex graph matching task with normally-distributed mutations. See Supplementary Figure 25 for performance across surveyed mutation rates.

C Genetic Programming Experiments

C.1 SignalGP

SignalGP (Signal-driven Genetic Programs) is a GP representation that enables signal-driven (*i.e.*, event-driven) program execution. In SignalGP, programs are segmented into modules (functions) that may be automatically triggered by exogenously- or endogenously-generated signals. Each module in SignalGP associates a tag with a linear sequence of instructions. SignalGP makes explicit the concept of signals (events), which comprise a tag and, optionally, signal-specific data. Signals trigger the module with the closest matching



(a) Normally-distributed Mutation (b) Bitwise mutation

Fig. 24: Trajectories of adaptive evolution for each tag-matching metric on the 32-vertex graph-matching task with normally-distributed mutation (24a) and bitwise mutation (24b). Maximum fitness is the best fitness value for any individual within a population. Reported results use each metric's best-performing standard deviation mutation rate. (See Supplementary Figure 25 for survey of how mutation rate affects adaptive evolution under each metric.) Note logarithmic x-axes. Shaded area represents bootstrapped 95% confidence intervals across 100 replicate observations.

tag (according to a given tag-matching scheme), using any signal-associated data as input to the triggered module. SignalGP can handle many signals simultaneously, processing each in parallel.

The SignalGP instruction set, in addition to including traditional GP operations, allows programs to generate internal signals, broadcast external signals, and otherwise work in a tag-based context. Instructions contain arguments, including an evolvable tag, that may modify the instruction's effect, often specifying memory locations or fixed values. Instructions may refer to program modules using tag-based referencing; for example, an instruction may trigger the execution of a program module using the instruction's tag to specify which module to trigger. SignalGP also supports genetic regulation with promoter and repressor instructions that, when executed, allow programs to adjust how well subsequent signals match with a target function (specified with tag-based referencing).

See (Lalejini and Ofria, 2018) for a more detailed description of the SignalGP representation. Additionally, see the GitHub repository for the SignalGP implementation used in this work (Lalejini, 2020).

C.2 Changing-signal Task Description

The changing-signal task requires programs to express a distinct response to each of K environmental signal (each signal has a unique tag). Programs express a response by executing one of K response instructions. Successful programs can 'hardcode' each response with the appropriate environmental signal, ensuring that each environmental signal's tag best matches the function containing the correct response. We expect the particular metric used to match tags to influence how well programs adapt to changing-signal task.

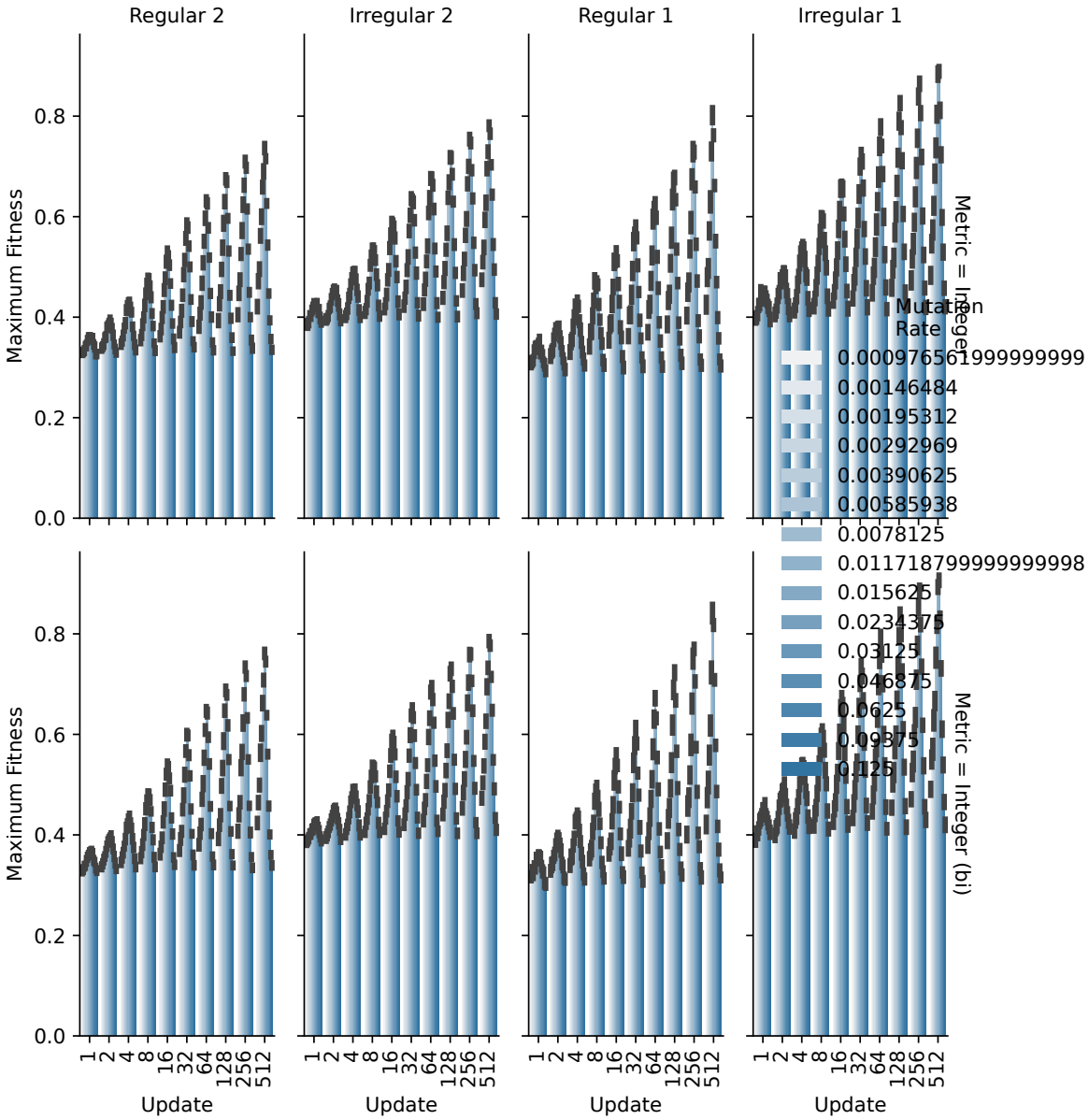


Fig. 25: Survey of adaptive evolution rates across mutation rates for 32-vertex graph-matching task with normally-distributed mutation. Maximum fitness is the best fitness value for any individual within a population. Maximum fitness at each update is presented across the range of surveyed mutation rates. Mutation rate is the standard deviation of mutational difference applied to tags each generation. Error bars represent bootstrap 95% confidence intervals across 100 replicate populations.

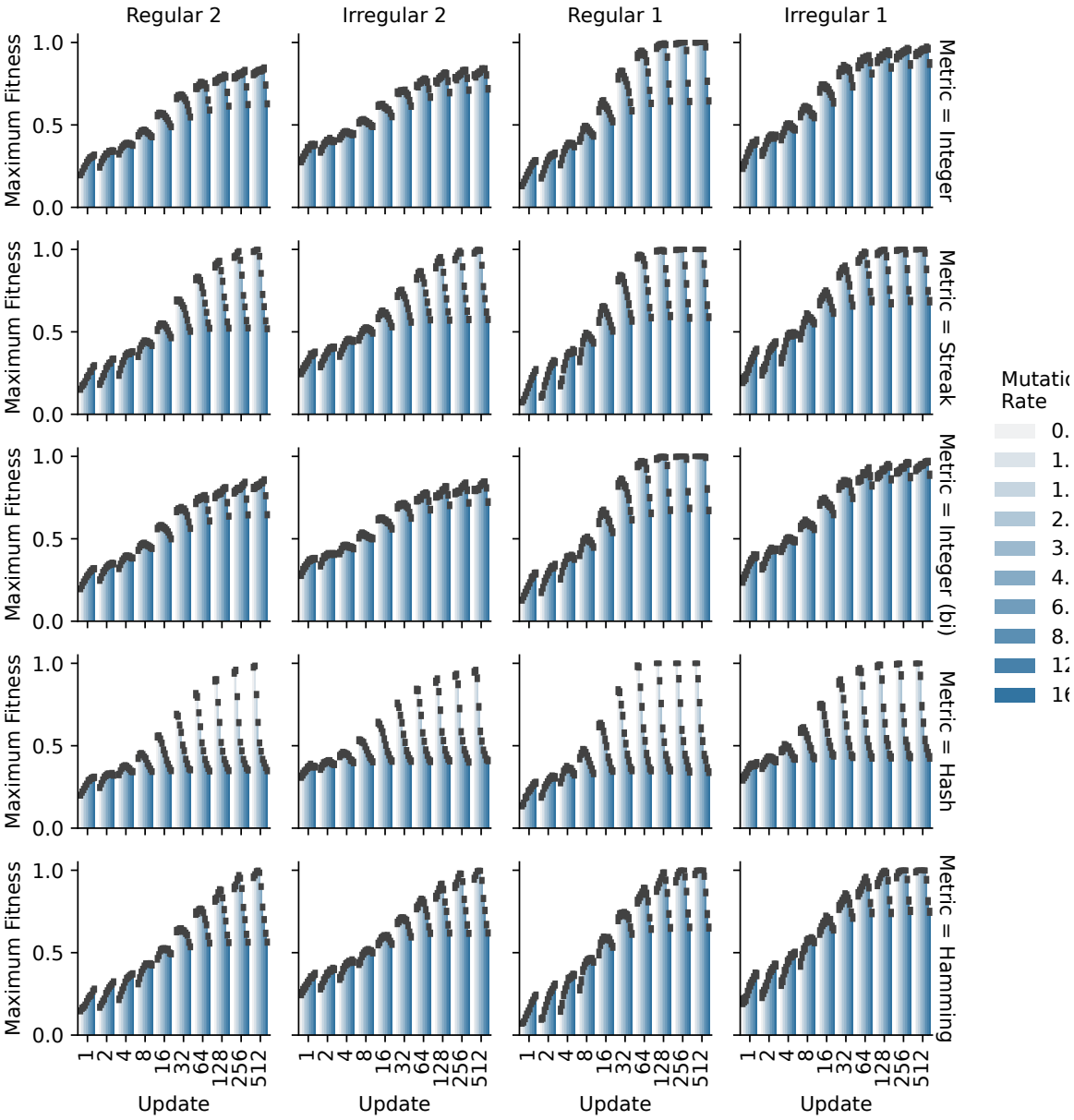


Fig. 26: Survey of adaptive evolution rates across mutation rates for 32-vertex graph-matching task with uniformly-initialized initial genomes. Metrics exhibited fastest adaptive evolution within the range of mutation rates surveyed, except the hash metric which exhibited fastest adaptive evolution at at the lowest mutation rate surveyed. Maximum fitness is the best fitness value for any individual within a population. Maximum fitness at each update is presented across the range of surveyed mutation rates. Error bars represent bootstrap 95% confidence intervals across 10 replicate populations.

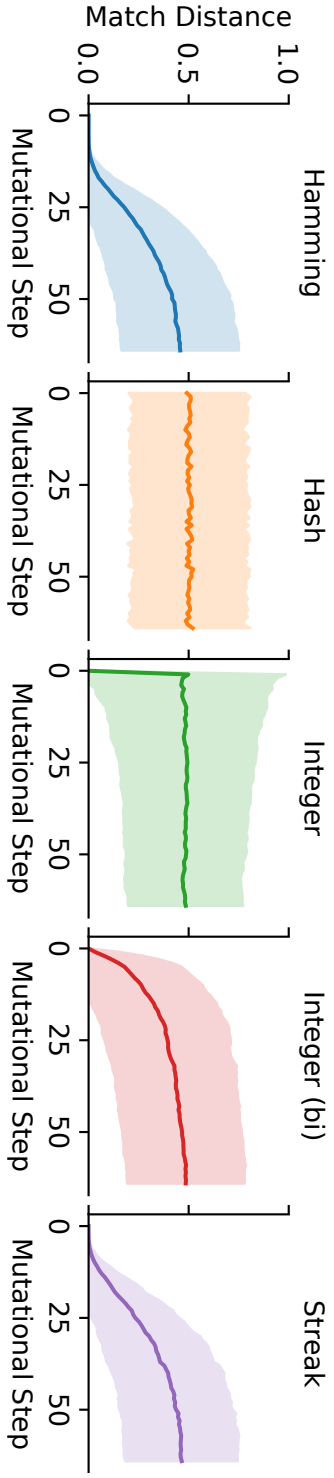
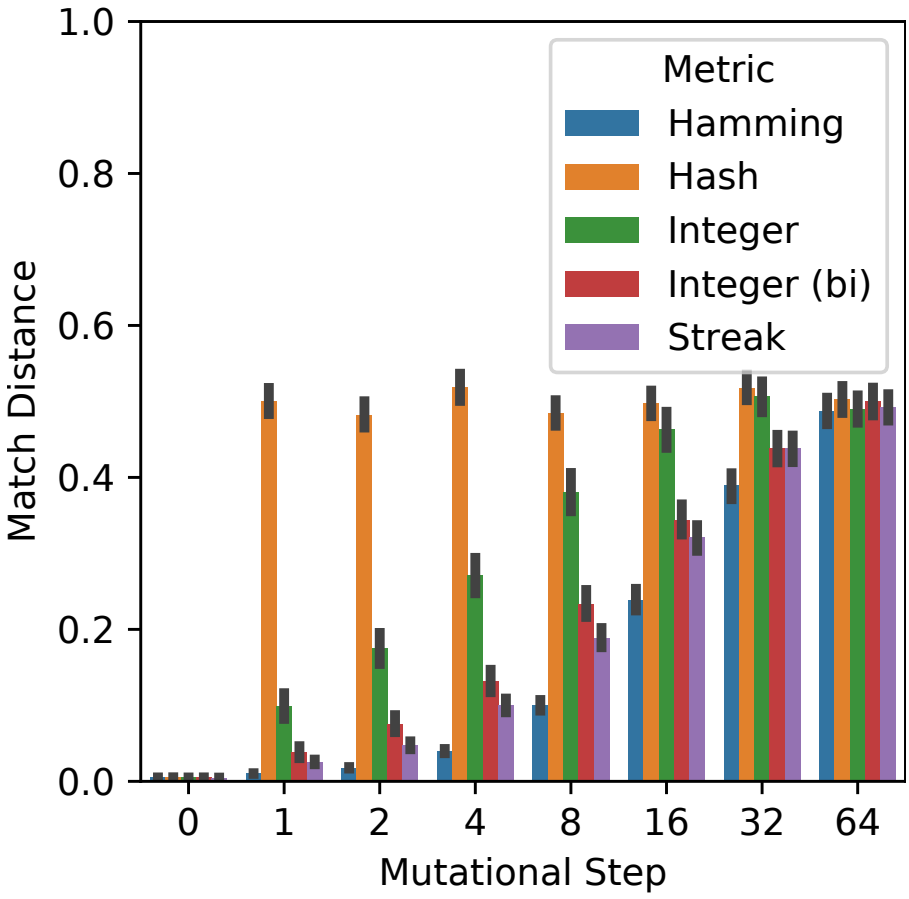
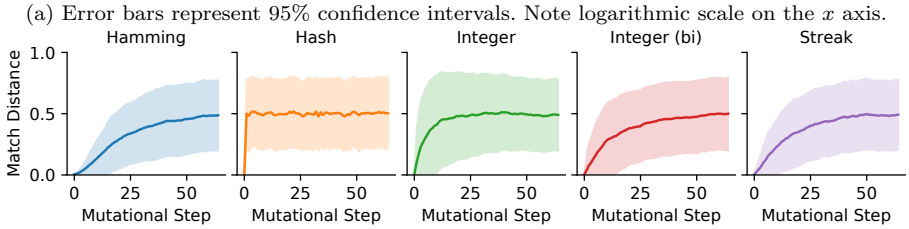


Fig. 27: Match distance over mutational walks from identical tags. Shaded area represents standard deviation.

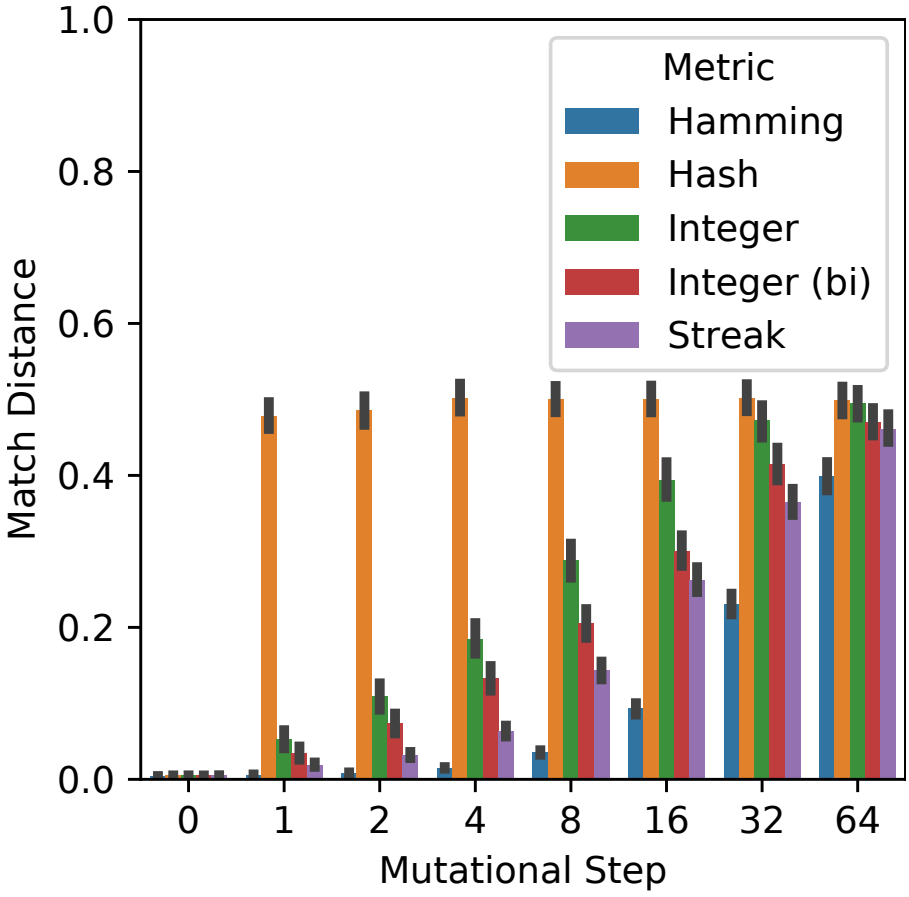


(a) Error bars represent 95% confidence intervals. Note logarithmic scale on the x axis.

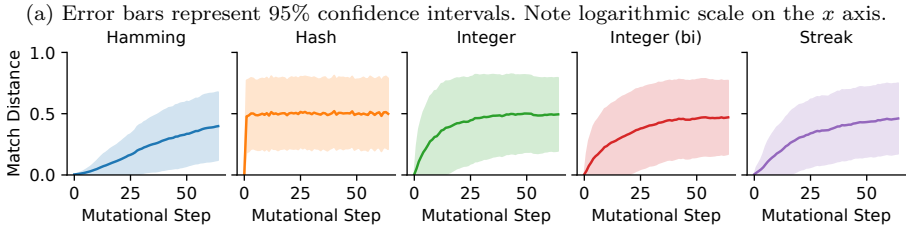


(b) Alternate visualization, shaded area represents standard deviation.

Fig. 28: Match distance along mutational walks from 32-bit tags sampled for initial match distance < 0.01.



(a) Error bars represent 95% confidence intervals. Note logarithmic scale on the x axis.



(b) Alternate visualization, shaded area represents standard deviation.

Fig. 29: Match distance along mutational walks from 64-bit tags sampled for initial match distance < 0.01.

Metric	Target Structure	Target Degree	Best-Performing Per-Genome Bit Mutation Rate
Hash	Regular	1	1.0
Hash	Regular	2	1.0
Hash	Irregular	1	1.0
Hash	Irregular	2	1.0
Hamming	Regular	1	4.0
Hamming	Regular	2	3.0
Hamming	Irregular	1	4.0
Hamming	Irregular	2	3.0
Streak	Regular	1	3.0
Streak	Regular	2	2.0
Streak	Irregular	1	4.0
Streak	Irregular	2	2.0
Integer	Regular	1	4.0
Integer	Regular	2	6.0
Integer	Irregular	1	6.0
Integer	Irregular	2	4.0
Bidirectional Integer	Regular	1	4.0
Bidirectional Integer	Regular	2	6.0
Bidirectional Integer	Irregular	1	8.0
Bidirectional Integer	Irregular	2	6.0

Table 5: Best-performing per-bit mutation rates for 32-vertex graph matching tasks with identically-initialized genomes. See Supplementary Figure 26 for performance across surveyed mutation rates.

During evaluation, we afford programs 64 time steps to express the appropriate response after receiving a signal. Once a program expresses a response or the allotted time expires, we reset the program's virtual hardware (resetting all executing threads and thread-local memory), and the environment produces the next signal. Evaluation continues until the program correctly responds to each of the K environmental signals or until the program expresses an incorrect response. During each evaluation, programs experience environmental signals in a random order; thus, the correct *order* of responses will vary and cannot be hardcoded.

For each metric, we evolved 200 replicate populations (each with a unique random number seed) of 500 asexually reproducing programs in an eight-signal environment ($K = 8$) for 100 generations. We identified the most performant tag mutation rate (from a range of possible mutation rates) for each metric to use in our experiment. These data (and analyses) are available online in the GitHub repository that houses these experiments (Lalejini, 2020). We used the following per-bit tag mutation rates for the changing-signal task: 0.01 for the Hamming and Streak metrics, 0.002 for the Hash metric, and 0.02 for the Integer and Bidirectional Integer metrics. Aside from tag mutation rate, the overall configuration used for each metric was identical. We limited tag variation in offspring to tag mutation (bit flips) by initializing populations with a common ancestor program in which all tags are identical and by disallowing mutations that would insert instructions with random tags.

The full configuration details for the changing-signal task (including a guide to running these experiments on your local machine) can be found in the associated GitHub repository (Lalejini, 2020).

C.3 Directional-signal Task Description

As in the changing-signal task, the directional-signal task requires that programs respond to a sequence of environmental cues; in the directional-signal task, however, the correct response depends on previously experienced signals. In the directional-signal task, there are two possible environmental signals — a 'forward-signal' and a 'backward-signal' (each with a distinct tag) — and four possible responses. If a program receives a forward-signal, it should express the next response, and if the program receives, a backward-signal, it should express the previous response. For example, if response-three is currently required, then a subsequent forward-signal indicates that response-four is required next, while a backward-signal would instead indicate that response-two is required next. Because the appropriate response to both the backward- and forward-signals change over time, successful programs must regulate which functions these signals trigger (rather than hardcode each response to a particular signal).

We evaluate programs on all possible four-signal sequences of forward- and backward-signals (sixteen total). For each program, we evaluate each sequence of signals independently, and a program's fitness is equal to its aggregate performance. Otherwise, evaluation on a single sequence of signals mirrors that of the changing-signal task.

We used an identical experimental design for the directional-signal task as in the changing-signal task. However, we evolved programs for 5000 generations (instead of 100) and re-parameterized each metric's tag mutation rate (these data are available in the associated GitHub repository (Lalejini, 2020)): 0.001 for the Hamming and Hash metrics, 0.002 for the Integer and Streak metrics, and 0.0001 for the Bidirectional Integer Metric.

The full configuration details for the directional-signal task (including a guide to running these experiments on your local machine) can be found in the associated GitHub repository (Lalejini, 2020) .

C.4 Data analysis and Implementation

The source code for our GP experiments can be found in the following GitHub repository: (Lalejini, 2020) . This repository additionally includes all data analysis and visualization scripts, experiment configuration details, and a guide for running our experiments locally.

Supplementary References

Lalejini, A. (2020). Exploring tag-matching metrics in SignalGP GitHub repository. https://doi.org/10.5281/zenodo.3781295.

Lalejini, A. and Ofria, C. (2018). Evolving event-driven programs with signalgp. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1135–1142.