# Genetic Programming Hyper-Heuristic for Emergency Medical Dispatching

Isaac Young

***Abstract*—Emergency Medical Dispatch (EMD) defines the healthcare task concerning the assignment of paramedic resources (equipment and people) to emergencies in the community. Due to the random and unpredictable nature of such tasks, EMD is highly dynamic. Research exists using Machine Learning to automatically learn heuristics for EMD via the Genetic Programming Hyper Heuristic (GPHH) technique. This project seeks to improve upon such existing research, reducing training time through engineering optimisations and implementing multi-fidelity techniques, in particular by training GPHH models on city networks (graphs) of varying fidelity. We produce lower-fidelity graphs by subdividing a source graph into boxes, and quantising locations within each box to a single location. The fidelity of graphs is then dictated by the size of the subdivided boxes. We evaluate our multi-fidelity models against pre-existing models on two real-world graphs, the Wellington and Christchurch ambulance coverage areas to investigate the impacts of applying multi-fidelity techniques to EMD. We find that the computation savings afforded by multi-fidelity techniques decrease model training time on graphs generated from real-world ambulance coverage areas by up to 51.3%, whilst not significantly impacting training or test performance. In addition, we find that per-generation training times are reduced by over 90% at the lowest fidelity levels, and so further speedup may be achieved through sustaining lower fidelity levels throughout training.**

***Index Terms*—Emergency Medical Dispatch, Genetic Programming, Hyper-Heuristic, Caching.**

## I. Introduction

In the execution of the emergency response services, a critical concern is the decision-making process by which medical resources (such as ambulances and medical personnel) are dispatched to emergencies in a community. Emergency Medical Dispatch (EMD) encapsulates this problem, efficiently dispatching medical resources to randomly occurring emergencies within a geographical coverage area in realtime. The dispatch decisions made in EMD can have a significant impact on the wait time of patients in emergency situations. Research has shown that the wait time of patients is positively correlated with patient mortality, and so it is imperative that wait times are minimised to maximise patient health outcomes [1], [2]. A common heuristic used in many real-world ambulance services is the "closest-idle" heuristic, where emergent emergencies are immediately paired with the closest idle ambulance (one not already attending to an emergency). However, intuitive approaches to EMD, such as the aforementioned "closest-idle" heuristic have been shown to be suboptimal [3]. Therefore, more sophisticated decision-making processes and heuristics stand to significantly improve dispatch performance and thus health outcomes if effective processes and heuristics can be found. Since EMD is a highly dynamic problem in which new information can invalidate prior solutions, *a priori* scheduling via traditional optimisation techniques is infeasible. Instead, the decision-making process must be able to make dispatch decisions in a dynamic environment in realtime.

This research project extends upon existing research conducted by MacLachlan et al. on using Genetic Programming (GP) to train heuristics for key decisions made in a typical EMD execution, known as Genetic Programming Hyper Heuristic (GPHH) [4], [5]. GPHH is well suited to EMD due to its inherent model interpretability [4]. Model interpretability (the ability to reason about the behaviour of the produced model) is critical for audit and liability purposes, and we expect that a GPHH approach is more feasible to implement in real-world EMD than other contemporary approaches such as Reinforcement Learning (RL) for this reason. In GPHH, an initial population of individuals representing heuristics are evolved through genetic operators (such as crossover and mutation) and evaluated to ascertain their fitness, dictating their likelihood for reproduction into the next generation. In the GPHH for EMD implementation by MacLachlan et al., fitness evaluation is achieved by simulating the individual on a "day" of randomly occurring emergencies, spread out over a geographical coverage area represented by a graph. The graph operations used in the fitness evaluation of individuals constitutes the majority of training time, and as such, our research aims to reduce model training time through engineering optimisations such as caching and improved multithreading, as well as implementing *multi-fidelity* training techniques adapted from research on GPHH for Dynamic Flexible Job Shop Scheduling (DFJSS). This is achieved by simplifying the graph to a lower "fidelity" in order to reduce the computational cost of route-finding queries. Since simplifying the graph to a lower fidelity will ultimately result in less precise simulations, the fidelity level is changed throughout training to vary the balance between accuracy and speed, resulting in a *multi-fidelity* approach.

### A. Motivations

As a service that directly protects and saves the lives of a municipality, ambulance services play a crucial role in society. Ambulances service demand has steadily increased for a variety of reasons such as population growth and ageing, pricing and availability, and reduced access to traditional healthcare services [6]. In addition, recent extreme events such

This project was supervised by Yi Mei. Please indicate affiliation of any external (non-ECS) supervisors.

as the COVID-19 pandemic have placed yet more pressure on ambulance services [7]. Our research aims to decrease the training time of the GPHH for EMD ambulance services through multi-fidelity techniques, making more effective utilization of limited computing resources. Doing so reduces the technical computing resource requirements, reducing costs and increasing the feasibility of an implementation at scale, with the ultimate goal of real-life adoption and thus improvement in real world EMD.

### B. Key Findings

We find that our application of multi-fidelity techniques to the baseline GPHH for EMD implementation reduced mean training time by between $41\%$ and $51.3\%$, while not significantly impacting final train and test performance. Learning effectiveness (fitness reduction per generation) is reduced slightly at low fidelity levels, but per-generation training time is reduced by up to $93\%$ at the lowest fidelity levels. However, fidelity levels above $250$ did not provide any significant per-generation training time reduction. Our multi-fidelity implementation does not impact the generalisability of the learned heuristics from one geographical coverage area to another – no significant difference in test performance was found when policies learned through the baseline were compared to policies learned with our multi-fidelity implementation.

## II. Related Work

### A. Learning Emergency Medical Dispatch Policies via Genetic Programming

As discussed in Section I, this project builds upon research conducted by MacLachlan et al. on using GP to train heuristics for decisions made in EMD [4]. The GP process described by Koza et al. [8] involves: *(1) Randomly creating an initial population of individuals consisting of functions and terminals, (2) Iteratively performing the sub-steps to create each subsequent **generation**: (a) Executing each program (individual) in the population to ascertain its fitness, (b) Performing genetic operations on units or pairs of individuals to form the next **generation**. Finally, when some termination criterion is satisfied, (3) The single best program in the population produced during the run is chosen as the result of the run.* In the existing GPHH for EMD implementation by MacLachlan et al., the initial population is built using the ramped half-and-half initialisation scheme [4]. In ramped half-and-half initialisation, trees are built to a maximum depth using, with equal probability, the *GROW* and *FULL* algorithms [9]. To ascertain their fitness, individuals are tested on $n$ simulated "days", known as *instances*, where a simulated day is a number of randomly occurring emergencies within a coverage area on a particular 24-hour day. Individuals within this context consist of five policies (implemented as GP trees, where internal nodes are functions and leaf nodes are terminals) that determine a set of key decisions in the execution of EMD:

- **D1:** Dispatch now (to emergency)? – Choose whether to dispatch an ambulance to a new emergency
- **D2:** Choose Ambulance – Choose the ambulance to dispatch to the emergency
- **D3:** Dispatch (idle ambulance) now? – Choose whether to dispatch a newly idle ambulance to an emergency in the emergency queue
- **D4:** Choose emergency – If a newly idle ambulance is to be dispatched, choose which emergency to attend to
- **D5:** Choose facility – If a new idle ambulance is *not* to be dispatched, choose which facility to return to

The training fitness of an individual on a particular instance is an average of response times, weighted by an urgency level (Level 1, 2, 3, or 4, in decreasing order of urgency). The final calculated training fitness of an individual is then the average fitness of the individual over the $n$ training instances.

The geographical coverage area is represented by a graph, which defines the road network and associated geometry, and facilitates route-finding queries via the OpenRouteService library, which uses highly optimised Contraction Hierarchies algorithms based on work in [10] for route finding. The heuristics learned by MacLachlan et al. already perform significantly better than the baseline model designed alongside Wellington Free Ambulance (WFA) that approximates their human decision-making logic. However, this work has been tested exclusively on *synthetic* node-based graphs, which do not replicate the scale of real-life coverage areas. Lately, further work has been completed to enable the use of coordinate-based graphs that fully replicate real-world geographical areas using the OpenRouteService library. However, these graphs are significantly larger and more complex than the synthetic graphs used prior, and as graphs get larger and more complex, training time increases exponentially. As a result, while training on graphs of small cities (such as those used in this project, *Wellington (WLG)* and *Christchurch (CHC)*) remains feasible, larger cities such as London or Chicago would pose a significant challenge for training time. In addition, since each training run starts with a randomised initial population, the first few generations are spent to simply bring the best fitness values below that of the manually-designed rules. While this is not a significant issue on small graphs where generations may only take minutes, it may be troublesome on large graphs where generations can take many hours.

### B. Multi-Fidelity Genetic Programming

While Evolutionary Computation (EC) techniques such as GP and GPHH are powerful tools for global optimisation with a wide variety of applications, it is also computationally expensive, due to the large number of fitness evaluations required in the training process to produce an acceptable solution [11], [12]. As such, a number of fitness approximation techniques have been developed to facilitate learning with reduced computational cost (resources/time). Usually, these approximation techniques trade increased computational efficiency for decreased accuracy relative to the true fitness [13], [14].

Fitness approximation techniques for EC can be broadly categorised into three categories: problem approximation, function approximation, and evolutionary approximation [11]. Problem approximation, replaces the original problem with a surrogate problem that is easier to solve. Approaches to

problem approximation vary, from using simplified equations in simulation to reducing the size of the problem solved [11]. Functional approximation avoids the expensive evaluation altogether, instead using an alternate approximated expression for the fitness function based on a set of sample points on the original problem [15]. Evolutionary approximation estimates fitness for *similar* individuals, such as an individuals parents or other individuals deemed similar as a result of some clustering algorithm. In [16], Esparcia-Alcázar et al. combine problem approximation and evolutionary approximation to evolve bots in the game *Unreal Tournament 2004*™, running individuals on a simplified test versions of the game and estimating an individuals' fitness based on individuals with similar outputs in the test games. This phenotypic characterisation, in which the behaviour of an individual is encoded to compare similar individuals is also used in [17], in which a nearest-neighbour regression is used to predict the fitness of an individual based on a vector of numerical values representing the behaviour of the individual. Hildebrandt et al. also use this vector to remove individuals with duplicate vectors, reducing the number of fitness evaluations required [17].

While fitness approximation reduces the computational cost of obtaining fitness values, a single surrogate means that training is highly sensitive to the accuracy of that surrogate model [12]. Instead, a multi-fidelity approach means that a number of surrogate models are used in training, reducing the sensitivity of training on any particular surrogate's accuracy. Multi-fidelity techniques have been applied to a number of problems, including wind farm layout design [18], antenna design [19], as well as Dynamic Job Shop Scheduling (DJSS) and DFJSS. At present, there are no applications of surrogate fitness approximation or multi-fidelity training for EMD. However, there is existing work on single and multi-fidelity surrogates for DJSS and DFJSS.

In DJSS, a number of *jobs* must be assigned to a set of heterogenous *machines*, where each job can only be completed by a particular machine of the set. Because jobs arrive in the "shop" over time without prior information, jobs must be assigned dynamically rather than following an *a priori* schedule. DFJSS expands upon DJSS by loosening the constraint that each job can only be completed by a particular machine, instead allowing each job to be completed by some subset of the machines. In both DJSS and DFJSS, the goal is to optimise the machine resources to achieve some objective, such as the minimising the overall makespan or max-flowtime [20]. Since the DFJSS problem shares similarities with EMD, such as the heterogeneity of workers and the highly dynamic nature of the problem, it is likely that learnings from single and multi-fidelity for DFJSS can be applied to EMD. There are various single and multi-fidelity surrogate approaches that have been applied to DJSS.

In [21], Nguyen et al. use problem approximation to evolve dispatching rules for DJSS, scaling the original problem by reducing the number of machines and operations arriving at the shop to produce a fitness estimate $f'(\Delta_i)$. These fitness estimates are then used to build the population of the next generation from a candidate pool $P'$, the result of applying genetic operations to the prior generations' population $P$. Nguyen et al. also reduce the number of replications (equivalent to instances in our usage) to produce a fitness $f_g(\Delta_i)$, used to decide the best performing individual $\Delta_g^*$ of a population $P$ at generation $g$. Finally, full evaluation is performed on $\Delta_g^*$ to find the true training fitness $f\left(\Delta_g^*\right)$, allowing it to be compared to the best performing individual of other generations. An equivalent problem approximation for producing an estimate $f'(\Delta_i)$ in our GPHH for EMD implementation would be to reduce the number of agents and the size of the coverage area. However, there are various complications when reducing coverage area size, such as route finding complications if arterial roads are cut off in the process.

### C. Surrogate-assisted GP For Dynamic Flexible Job Shop Scheduling

Our multi-fidelity approach is inspired by research conducted on surrogate-assisted GP for DFJSS by Zhang et al. [20], as the fidelity strategy aligns well with our aim of simplifying the original coverage area graph to a variety of fidelity levels to vary the balance between approximation accuracy and training speed. To train GP models for the DFJSS problem, Zhang et al. use an approach described by Yska et al. [22], in which both *routing* and *sequencing* rules are evolved simultaneously. Similarly, in the GPHH for EMD research [4] which forms the basis of this research, MacLachlan et al. evolve all five key EMD rules **D1-5** simultaneously with GP.

In [20], Zhang et al. use problem approximation, where an approximated, simplified surrogate problem is used in place of the original problem. Then, the fitness value obtained through evaluation on the simplified problem is used as an estimator for the fitness value of an individual on the original problem [20]. In [20], the problem is simplified by reducing the number of jobs in the DFJSS problem. Thus, a low-fidelity surrogate has fewer jobs than a high-fidelity surrogate. An equivalent problem approximation method for EMD would be to reduce the number of occurring emergencies and the length of simulation time. However, we did not use this method of problem approximation, as our problem approximation design (discussed in Section IV) targets more directly the performance bottleneck – the number of times that unique route requests are made.

Naturally, low-fidelity surrogates will have some fitness approximation error, but Zhang et al. assert that this does not necessarily harm training performance. Instead, Zhang et al. hypothesise that these lower fidelity surrogates may positively contribute to the evolutionary search than the original problem, as the problem simplification is capable of smoothing the multimodal/noisy landscape of a complex problem [20]. Based on the assumption that the surrogate fidelity and the performance of the surrogate fitness values as estimators for the true fitness values are positively correlated, Zhang et al. use simple (low-fidelity) surrogates early in training, and increase the surrogate fidelity as training progresses. Zhang et al. propose two multi-fidelity surrogate strategies in [20]: Adaptive Surrogate Genetic Programming (ASGP) and Generation-Range-Based Surrogate Genetic Programming (GSGP).

In ASGP, the surrogate fidelity is increased continually throughout training. In [20], the number of jobs increases linearly: $N_{\text{job},i} = N_{\text{job}} \times \frac{\max(i,1)}{\text{maxGen}-1}$, where $N_{\text{job},i}$ is the number of jobs for the surrogate at a particular generation $i$, maxGen is the maximum (last) generation, and $N_{\text{job}}$ is the number of jobs in the original problem.

In GSGP, the generations are split into ranges, each of which is assigned a particular fidelity level. The fidelity levels are non-decreasing as the generation $i$ increases towards maxGen. GSGP is well suited to our problem approximation design (Section IV), as graphs can be built with varying fidelity levels at runtime. In testing, Zhang et al. found that ASGP and GSGP reduced computation time by 25.7% and 34.4% respectively, and achieved similar or better test performance compared to the full-fidelity training. ASGP is not feasible for our method of problem simplification – since the first generation of training/evaluation performed at each fidelity level must be performed at full-fidelity for data collection (discussed in Section V-B), fidelity cannot be increased continuously. As such, we use a multi-fidelity strategy based on GSGP.

## III. Improvements to Original GPHH for EMD Implementation

As implemented by MacLachlan et al., the original GPHH for EMD implementation is a multithreaded application written in Java 8 using the ECJ library. ECJ is a popular and well-established EC toolkit first released in 1999, offering optimised performance and support for parallel computation in a unified framework [23]. Whilst our multi-fidelity design aims to improve execution speed by reducing the accuracy of estimates within the simulation, we first seek to exhaust optimisation efforts that do not have a tangible impact on training. As such, a number of engineering improvements have been made to the original GPHH for EMD implementation to form a baseline from which the multi-fidelity implementation can be compared to.

### A. Maven & Java Version Update

The original implementation was built using the IntelliJ build system using local .JAR dependencies on a particular machine, making it difficult to transfer to other development environments. In order to ease onboarding for new contributors using local development environments, we replaced this with the Maven build system, enabling access to updated dependencies through the Maven Central Repository.

The original implementation was run on OpenJDK 8. Since non-commercial support for Java 8 ended in March 2022, we upgraded to Java 17, the latest LTS version of Java in March 2023, the start date of this project [24]. Java 17 has premier support until "at least" 2026. We also changed runtime to Azul 17, which we found to reduce training time by approximately 50% in local testing.

### B. Multithreading Improvements with ForkJoinPool

Within a single generation of GP training, we evaluate 1000 individuals to ascertain their fitness. The GP training evaluation process is described in Section II-A. Because evaluation is independent between individuals, the workload is split across multiple cores using multithreading. We use a compute cluster with 256 cores, and the multi-threading implementation built into ECJ yields a significant speedup over single-threaded execution. However, the ECJ implementation splits the workload before execution. Since we do not have *a priori* knowledge of how long evaluating any given individual (and evaluation times between individuals vary significantly) will take, threads are inevitably assigned unequal workloads. No further processing can be completed until all individuals are fully evaluated, and thus a few long-running threads assigned particularly heavy workloads will prevent the next stage (building the population for the next generation) from executing.

To minimise workload inequality, we integrate an updated multi-threading implementation into the existing ECJ evaluation framework, using the Java `ForkJoinPool`, a thread pool that implements "work stealing" [25]. Using the `ForkJoinPool`, each individual is considered a "task", distributed to the thread pool. Threads in the pool then attempt to find and execute tasks submitted to the pool when underworked, thereby allowing the computational load to be distributed in a dynamic (rather than a purely static) manner. Under profiling, we have found the `ForkJoinPool` to distribute workload more evenly, reducing the variance in workload between physical threads reducing the overall time to evaluate all individuals within a generation by 7.5% on average.

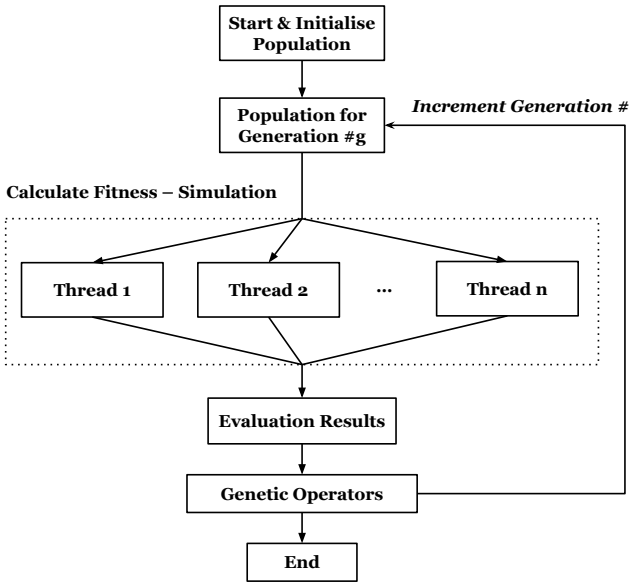


Fig. 1: Flow diagram showing the fan-out, fan-in nature of multithreaded computation within training.

Aside from evaluation, all other computations are done in a single-threaded context on the main thread, though a number of multi-fidelity computations are partially executed during evaluation and thus in a multithreaded context. These multi-fidelity computations are discussed in Section V-B. Figure 1 shows the overall training flow and multithreaded execution.

### C. Caching & `hashCode` Optimisations

Because the routes returned from route queries must be stored in order for quantised routes to prevent recalculation for similar endpoints, caching is a pre-requisite for multi-fidelity. However, caching can also improve the performance of the original implementation. Since route queries with identical endpoints will always return the same route, stored routes can be returned for future queries with the same endpoints.

As discussed in Section III-B, evaluation is executed in parallel, the cache must support multithreaded operation with minimal locking. We considered three implementations of a parallel key-value cache in Java:

1) `java.util.concurrent.ConcurrentHashMap<K, V>` from the Java Standard Library,
2) `com.google.common.cache.Cache<K, V>` from the Google Guava Library, and
3) `com.github.benmanes.cache.Cache<K, V>` from the Caffeine caching library, a successor to the Guava's caching API [26]–[28]

Despite its simplicity and ease of use, we found that the `ConcurrentHashMap` implementation in the Java Standard Library is unsuitable for our cache use, due to its lack of an eviction policy – memory usage could grow to an unsustainable size given the number of possible points in a graph. Guava uses the common Least Recently Used (LRU) cache replacement policy, which discards the item used furthest in the past. Caffeine uses a modern "Window Tiny Least Frequently Used" policy, which has shown in benchmarks to perform significantly better than LRU [29]. When testing on the existing GPHH for EMD system, we found that Caffeine resulted in an average of 3% lower training time over Guava (which itself produced training times under half that of the original implementation), and so Caffeine is the chosen caching implementation for our simulation. We also optimised `hashCode` implementations, eliminating `equals` and `hashCode` contract violations, and minimising the boxing of Java primitives where possible.

Ultimately, this optimisation effort resulted in a baseline implementation that is approximately 6x faster than the original implementation, reducing mean training time from >24 hours to 4 hours.

## IV. Multi-Fidelity Design

Simulation graphs are at the scale of, and represent real-world coverage areas, such as the *Wellington* and *Christchurch* graphs used in this research project. The execution of these individuals on the simulated days is computationally heavy, and is the most time-consuming process when training. Therefore, our research aims to reduce the computational load of simulations while producing comparable results in terms of resultant fitness, ultimately reducing model training time. The advantages of reduced model training time are twofold, both reducing feedback cycle time of the GPHH for EMD implementation in its research & development phase, and facilitating a larger number of users (emergency dispatch services) for a given compute capacity in production.

The computational cost of route-finding queries within a graph is the primary contributor to the complexity of the simulation. Route-finding queries are used throughout the simulation to implement the various terminals that make up an individual, as well as to calculate locations of agents in transit. We use the OpenRouteService library, which supports a simultaneous query for a $\mathit{Path}_{i\to j} = \{p_i, \ldots, p_j\}$ and $\mathit{Cost}_{i\to j} \in \mathbb{R}^+$ from a *source* point $p_i = \langle \mathrm{lat}_i, \mathrm{lon}_i \rangle$ to a *destination* point $p_j = \langle \mathrm{lat}_j, \mathrm{lon}_j \rangle$. We then denote the tuple $\mathit{Route}_{i\to j} = \langle \mathit{Path}_{i\to j}, \mathit{Cost}_{i\to j} \rangle$ as the *route* between two points on a graph, such as the route from an agent to a new emergency.

These route-finding queries are performed throughout the training process in excess of 10,000 times per individual, per training instance, per generation, per training seed, per graph. In a full training run, we train 1000 individuals on 5 instances each for 50 generations, repeated over 30 Pseudorandom Number Generator (PRNG) seeds. As a result, route-finding queries are made well in excess of 75 billion times per training run, and so optimisations made to this critical operation can have a significant impact on the resulting training time. Given this observation, the principal aim of our research is to reduce the computational cost of route-finding queries.

In real-world EMD, operators must make dispatch decisions with estimates for route costs, and indeed the costs returned from OpenRouteService throughout the simulation can only be considered estimates. Our approach to optimising route-finding requests is to decrease the accuracy of these estimates in order to increase the speed of the requests over many invocations. The effective accuracy of these estimates constitute a level of *fidelity*, where a low-fidelity graph will produce low-accuracy routes, and a high-fidelity graph will produce high-accuracy routes. The fidelity level is then inversely proportional to the amortised speed of request invocations, and thus we implement *multi-fidelity* techniques, varying the balance between accuracy and speed throughout training.

### A. Caching and Route-Point Quantisation

An initial optimisation over the original GPHH for EMD implementation is the usage of query caching. Since queries can be considered pure functions, the result $\mathit{Route}_{i_1\to j_1}$ of a given route query can be reused for query $p_{i_2} \to p_{j_2}$, provided that $p_{i_1} = p_{i_2}$ and $p_{j_1} = p_{j_2}$. This can provide significant speedup for queries between static locations, such as the route between an agent at a facility to a repeat emergency, eliminating the need for recalculation when start and end points are identical.

However, if $p_{i_1}$ and $p_{i_2}$ or $p_{j_1}$ and $p_{j_2}$ differ, even by a small amount (e.g., within 15 meters), the route must be calculated anew. Our graph simplification approach is to loosen this constraint through the quantisation of endpoints in a route query, such that route query results can be re-used when $p_{i_1} \simeq p_{i_2}$ and $p_{j_1} \simeq p_{j_2}$, with lower fidelity graphs quantising more aggresively, allowing for reuse between pairs of points with greater deviations than higher fidelity graphs. An intuitive design for route-point quantisation would be to store the result $\mathit{Route}_{i\to j}$ for each pair $p_i \to p_j$, and return $\mathit{Route}_{i\to j}$ for some

$p_{i_2} \to p_{j_2}$ if $p_i \simeq p_{i_2}$ and $p_j \simeq p_{j_2}$. As fidelity increases, we would decrease the maximum allowable distance for endpoints to be quantised to endpoints for which a route has already been computed. However, with this design, the resulting endpoints used in the route query for are dictated solely by the points used in the first query with points in their respective areas, since subsequent queries with endpoints within a radius of the endpoints in the initial query will return the same route. If the first query had endpoints in a difficult-to-access location (e.g., down a narrow, windy road), it may be a poor representative of endpoints in the vicinity. In addition, within a multithreaded simulation context, the first query may differ from run-to-run, resulting in non-deterministic behaviour for the resulting quantised routes.

For these reasons, we use a quantisation design that is less sensitive to initial route queries, and can be implemented in a deterministic manner. For a fidelity level $l$, we divide the graph into $l^2$ equally sized areas called *boxes*. Figure 2 shows the *Wellington (WLG)* graph divided into boxes at $l = 10$ and $l = 50$. We then elect (through some election algorithm) a *delegate point* $p_{\mathbf{B}_{\text{del}}}$ for each box $\mathbf{B}$ in the graph that is used as the quantised point when a point $p$ is used as an endpoint for a route query: $p \rightsquigarrow p_{\mathbf{B}_{\text{del}}} \iff p \in \mathbf{B}$. We use $\rightsquigarrow$ to indicate replacement with an approximation: $x \rightsquigarrow \hat{x}$ denotes $x$ being replaced with $\hat{x}$. Then, a route query between endpoints $p_i \to p_j$ are substituted with a route query between the delegate points for the boxes in which $p_i$ and $p_j$ reside respectively: $\textit{Route}_{i\to j} \rightsquigarrow \textit{Route}_{p_{\mathbf{B}^{\mathbf{x}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{y}}_{\text{del}}}} \iff p_i \in \mathbf{B}^{\mathbf{x}} \wedge p_j \in \mathbf{B}^{\mathbf{y}}$. This route is cached, and future route queries from $p_{i_1} \to p_{i_2}$ can return the same route, provided that $p_{i_2} \in \mathbf{B}^{\mathbf{x}} \wedge p_{j_2} \in \mathbf{B}^{\mathbf{y}}$. As fidelity increases, we divide the graph into a greater number of boxes, resulting in smaller boxes and thus more accurate delegate points.

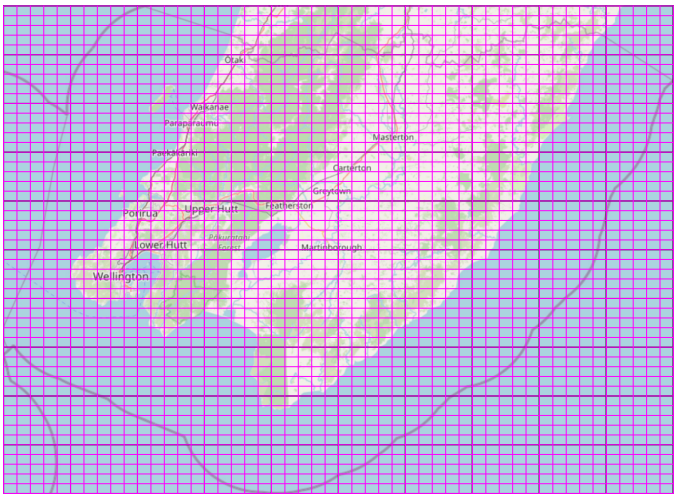


Fig. 2: *Wellington (WLG)* graph divided into $10^2$ boxes at fidelity 10 (thick black lines) and $50^2$ boxes at fidelity 50 (magenta lines).

### B. Multi-Fidelity Strategy

We use a *Multi-Fidelity Strategy* adapted from the GSGP strategy described by Zhang et al. [20]. In a fidelity stage a particular fidelity level is maintained for a range of generations, after which the fidelity level is updated for the next stage. Multiple fidelity stages are sequenced together to comprise a fidelity sequence. The fidelity sequences used in our evaluation and analysis are described in Section VI-B.

### C. In-Transit Simulation Simplification (ITSS)

In addition to the route-point quantisation described in Section IV-A, we also simplify the simulation to facilitate the use of route-point quantisation. When an individual is evaluated on the simulated "days" of occuring emergencies, agents (such as ambulances carrying medical personnel) travel to emergencies, and optionally return to a facility so patients can receive further care. When an agent is en-route to a destination (either a facility or emergency), its location must be updated so that it can be used in the decision-making process. For example, an ambulance with capacity en-route to a facility may become the closest ambulance to a newly occuring emergency, resulting in its dispatch to the emergency. Due to limitations with the OpenRouteService graph implementation, we are unable to obtain time estimates within a route. Therefore, the current implementation uses a binary search of route queries to points within the agent's current route are used to locate agents that have been in-transit on a particular route for some period of time.

However, since route-point quantisation effectively limits the resolution of such requests, this binary search method is infeasible, as time estimates for points further along a route may not necessarily be longer. As a result, we opt to simplify the method by which agent locations are calculated. Consider an agent on a $\textit{Route}_{i\to j} = \langle \textit{Path}_{i\to j}, \textit{Cost}_{i\to j} \in \mathbb{R}^+ \rangle$, where $\textit{Path}_{i\to j} = \{p_i = p_0, \ldots, p_j = p_n\}$ has $n$ points. If $m$ time has passed since the agent left $p_i$, we linearly interpolate the point based on the proportion between the elapsed and total estimated time, taking point $p_{\lfloor n \times \frac{m}{\textit{Cost}_{i\to j}} \rfloor}$ to be the calculated location of the agent.

### D. Intra-Box Routing

As discussed in Section IV-A, route queries between endpoints $p_i \to p_j$ are substituted: $\textit{Route}_{i\to j} \rightsquigarrow \textit{Route}_{p_{\mathbf{B}^{\mathbf{x}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{y}}_{\text{del}}}} \iff p_i \in \mathbf{B}^{\mathbf{x}} \wedge p_j \in \mathbf{B}^{\mathbf{y}}$. However, when both the source point $p_i$ and destination point $p_j$ reside within the same box $\mathbf{B}$, then $\mathbf{B}^{\mathbf{x}} = \mathbf{B}^{\mathbf{y}}$, and thus $p_{\mathbf{B}^{\mathbf{x}}_{\text{del}}} = p_{\mathbf{B}^{\mathbf{y}}_{\text{del}}}$. Then, the calculated route would be: $\textit{Route}_{p_{\mathbf{B}^{\mathbf{x}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{y}}_{\text{del}}}} = \langle \{p_{\mathbf{B}^{\mathbf{x}}_{\text{del}}}\}, 0 \rangle$. A $\textit{Cost}_{i\to j}$ of 0 is not a representative cost value for routes within a box, as costs are always positive. Such a cost could encourage the model to prefer the dispatch of agents to emergencies in the same box, since it would achieve this at no time cost. However, this would not generalise well to testing (nor in real-world application), since testing is always performed at perfect fidelity and actual costs are positive. Instead, for each box $\mathbf{B}$, we calculate a cost value $c$ that is used to approximate the mean cost of routes with both endpoints within $\mathbf{B}$, resulting in a quantised intra-box route of $\textit{Route}_{p_{\mathbf{B}_{\text{del}}} \to p_{\mathbf{B}_{\text{del}}}} = \langle \{p_{\mathbf{B}_{\text{del}}}\}, c \rangle$.

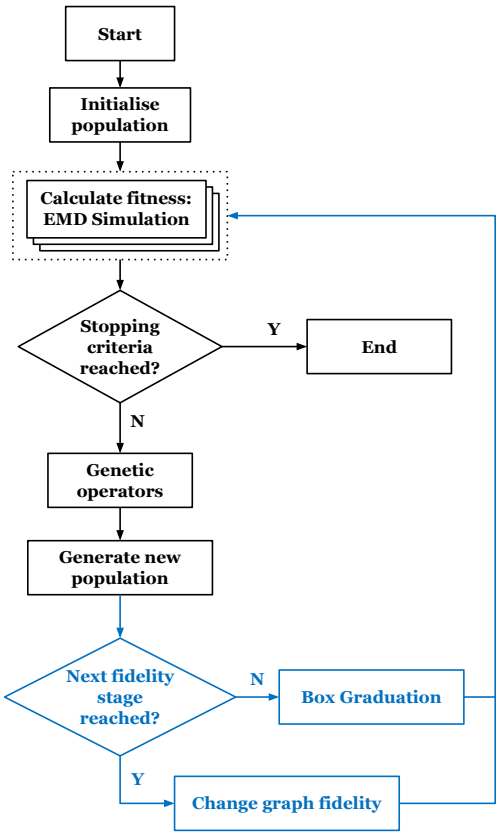


Fig. 3: Overall flow diagram of Multi-Fidelity GPHH for EMD, with new multi-fidelity flow highlighted. Dotted outline with layered boxes show multithreaded execution.

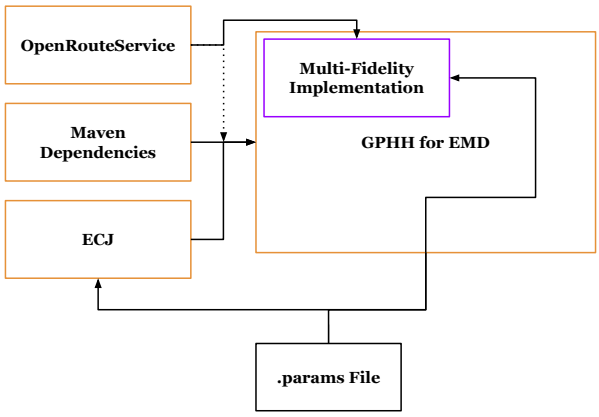


Fig. 4: Architecture of Multi-Fidelity GPHH for EMD. Orange components are written in Java, while the purple multi-fidelity component is written in Kotlin. Dotted line shows default connection from OpenRouteService to GPHH for EMD system when Multi-Fidelity is disabled.

## V. Multi-Fidelity Implementation

As discussed in Section III, the improved baseline GPHH for EMD implementation is a Java 17 application using the ECJ library. It uses the OpenRouteService Java library for graph operations such as route finding, as well as a number of other Java dependencies managed by Maven. We use the Kotlin programming language for the multi-fidelity implementation due to its ease of use facilitating faster prototyping, full interoperability with Java, and similar runtime performance [30]. Figure 4 shows the architecture of the Multi-Fidelity GPHH for EMD system.

### A. Multi-Fidelity Strategy

To facilitate the simple configuration of the multi-fidelity strategy, we integrate the multi-fidelity parameters, such as fidelity staging (fidelity at each generation range) into the plaintext `.params` configuration file used by ECJ.

### B. Box Lifecycle

As discussed in Section IV-A, we elect a delegate point $p_{\mathbf{B}_{\text{del}}}$ (used for both inter and intra-box routes) and intra-box cost value $c$ for each box $\mathbf{B}$. An intuitive implementation for delegate point election would be to elect the central point of the box. However, the central point may not be a good representative for query points in that box. For example, for a box with an arterial road on its west side, a delegate point situated towards the west would be more representative of the typical point than the central point of the box. Similarly, for intra-box routes, every box will have a different typical cost (compare an inner-city box to a rural box with a highway roads). Therefore, for each box $B$, we observe a number of route queries made during simulation/evaluation to elect the delegate point $p_{\mathbf{B}_{\text{del}}}$ and cost value $c$. There are two distinct cases for route queries:

1) Inter-box queries: the endpoints reside within different boxes: $p_i \in \mathbf{B^1} \wedge p_j \in \mathbf{B^2}, \mathbf{B^1} \neq \mathbf{B^2}$
2) Intra-box queries: the endpoints reside within the same box: $p_i \in \mathbf{B} \wedge p_j \in \mathbf{B}$

For case (1), we add $p_i$ and $p_j$ as delegate point "votes" for $\mathbf{B^1}$ and $\mathbf{B^2}$ respectively. Once the *Graduation Eligibility Criteria* (Section V-B1) for a box $\mathbf{B}$ is met for inter-box queries, the delegate point $p_{\mathbf{B}_{\text{del}}}$ is calculated via the *Delegate Point Election Algorithm* (Section V-B2). Before $p_{\mathbf{B}_{\text{del}}}$ is elected, $\mathbf{B}$ is *externally inactive*, and all inter-box queries with an endpoint residing in $\mathbf{B}$ are calculated at perfect fidelity with the underlying OpenRouteService implementation, though the *In-Transit Simulation Simplification (Section IV-C)* still applies. After $p_{\mathbf{B^x}_{\text{del}}}$ for a box $\mathbf{B^x}$ is elected, the box "graduates" to become *externally active*, and inter-box queries with an endpoint residing in $\mathbf{B^x}$ would be quantised – including queries in which the box the other endpoint resides in has *not* elected a delegate point.

In case (2), we record the $Cost_{i \to j}$ for the enclosing box $\mathbf{B}$. Once the *Graduation Eligibility Criteria* (Section V-B1) for a box $\mathbf{B}$ is met for intra-box queries, the cost value $c$ is calculated by averaging the recorded costs. Before $c$ is calculated, $\mathbf{B}$ is *internally inactive*, and all intra-box queries with endpoints residing in box $\mathbf{B}$ are calculated at perfect fidelity with the underlying OpenRouteService implementation, though the *In-Transit Simulation Simplification* still applies. Once $c$ is calculated, provided that a $p_{\mathbf{B}_{\text{del}}}$ has previously been elected, the box "graduates" to become *internally active*,

and intra-box queries within endpoints residing in box $\mathbf{B}$ use quantised routes.

As a result, there are three possible states for each box, shown in Table I.

| State Symbol | Internal State | External State | Possible Next States |
|---|---|---|---|
| $\mathbf{B}$ | Inactive | Inactive | $\{\mathbf{B}+\}$ |
| $\mathbf{B}+$ | Inactive | Active | $\{\mathbf{B}\pm\}$ |
| $\mathbf{B}\pm$ | Active | Active | $\emptyset$ |

TABLE I: Box states with state symbols and possible next states.

When fidelity is changed, the graph is divided into a new number of boxes, and all boxes are initialised to an internally and externally inactive state.

*1) Graduation Eligibility Criteria:* To calculate representative values for $p_{\mathbf{B}_{\text{del}}}$ and $c$, we first observe full-fidelity route queries and collect point and cost data, calculating values after the number of data observations meet or exceed a particular threshold $k$. We use a value of $k = 50$, which we have found in testing to produce relatively stable values (with different PRNG seeds) while not significantly impacting performance. However, if boxes were permitted to graduate immediately after meeting the threshold $k$ and begin returning quantised routes, this would occur during evaluation, thereby changing the behaviour of the simulation and resulting in an unequal evaluation of individuals within a generation: Consider a graph with fidelity $n = 3$, such that the boxes are in state $\mathbb{B}_t$:

$$\mathbb{B}_t = \begin{bmatrix} \mathbf{B^{11}} & \mathbf{B^{12}} & \mathbf{B^{13}} \\ \mathbf{B^{21}} & \mathbf{B^{22}}\pm & \mathbf{B^{23}} \\ \mathbf{B^{31}} & \mathbf{B^{32}} & \mathbf{B^{33}} \end{bmatrix}$$

Where a $+$ suffix indicates that a box is externally active only, and a $\pm$ suffix indicates a box is both internally and externally active.

If an individual $d_1$ is evaluated given $\mathbb{B}_t$, when we request a route from $p_{i_1} \in \mathbf{B^{23}}$ to $p_{j_1} \in \mathbf{B^{22}}$, the route request would be quantised to:

$$\mathit{Route}_{i_1 \to j_1} \rightsquigarrow \mathit{Route}_{p_{i_1} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} \tag{1}$$

The returned route would then be:

$$\mathit{Route}_{p_{i_1} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} = \langle \{p_{i_1}, \ldots, p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}\}, \mathit{Cost}_{p_{i_1} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} \rangle \tag{2}$$

If another individual $d_2$ is then evaluated and performs a route request from $p_{i_2} \in \mathbf{B^{23}}$ to $p_{j_2} \notin \mathbf{B^{23}}$, the number of external requests with a point within $\mathbf{B^{23}}$ may reach the threshold $k$. Then, if boxes were permitted to graduate immediately after reaching the threshold, then $\mathbf{B^{23}}$ would graduate, and the boxes would then be in state $\mathbb{B}_{t+1}$:

$$\mathbb{B}_{t+1} = \begin{bmatrix} \mathbf{B^{11}} & \mathbf{B^{12}} & \mathbf{B^{13}} \\ \mathbf{B^{21}} & \mathbf{B^{22}}\pm & \mathbf{B^{23}}+ \\ \mathbf{B^{31}} & \mathbf{B^{32}} & \mathbf{B^{33}} \end{bmatrix}$$

Any subsequent requests from the same $p_{i_1} \in \mathbf{B^{23}}$ to $p_{j_1} \in \mathbf{B^{22}}$ as in (1) would then be quantised to:

$$\mathit{Route}_{i_1 \to j_1} \rightsquigarrow \mathit{Route}_{p_{\mathbf{B}^{\mathbf{23}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} \tag{3}$$

Returning a route not equal to (2):

$$\mathit{Route}_{p_{\mathbf{B}^{\mathbf{23}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} = \langle \{p_{\mathbf{B}^{\mathbf{23}}_{\text{del}}}, \ldots, p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}\}, \mathit{Cost}_{p_{\mathbf{B}^{\mathbf{23}}_{\text{del}}} \to p_{\mathbf{B}^{\mathbf{22}}_{\text{del}}}} \rangle$$

As a result, the evaluation on individuals would differ between states $\mathbb{B}_t$ and $\mathbb{B}_{t+1}$. This is undesired, as the fitness values for each individual used to determine the likelihood for reproduction will not be comparable. In addition, such an implementation would exhibit non-deterministic behaviour in a multithreaded context, as the first $k$ queries (from any threads) would vary from run to run.

Therefore, we limit box graduation to occur only when the generation is incremented, when all evaluations for that generation have completed and the system is running in a single-threaded context (see Figure 1). This ensures that all individuals in a particular generation are evaluated under identical circumstances. In addition, we do not limit the number of route request observations to the *first* $k$, instead graduating any boxes that have collected *at least* $k$ observations. As a result, the order in which routes are requested before graduation does not affect the resultant $p_{\mathbf{B}_{\text{del}}}$ and $c$ values for each box, so $p_{\mathbf{B}_{\text{del}}}$ and $c$ are deterministic across runs (provided PRNG seeds are identical). However, the number of observations is only guaranteed to be $\geq k$, rather than $= k$, since boxes continue to collect observations until they graduate.

*2) Delegate Point Election Algorithm:* To calculate the delegate point $p_{\mathbf{B}_{\text{del}}}$ for a box, a number of alternatives were considered:

- **MFE**: Average *every* ($\geq k$) observed point equally
- **MFD**: Average $\geq k$ *distinct* observed points
- **MFL**: Compute a weighted average of every point, weighting the *log* ($\log_{10}$) of the number of points observed at a distinct location

| | MFE | MFL | **MFD** |
|---|---|---|---|
| Average Test Fitness ($\sigma$) | 47.25 (0.96) | 46.67 (1.14) | **46.59 (1.10)** |

TABLE II: Average test fitness across 30 runs (PRNG seeds) on WLG graph after 25 generations for delegate point election algorithm alternatives.

When using **MFE**, we found that the elected delegate points for particular boxes were more varied between runs with different PRNG seeds. Since training instances (which contain emergencies and their locations) are used repeatedly, many routes had the same endpoints, and thus the delegate points were highly sensitive to the first few training instances used. Therefore, we implemented **MFL**, ensuring that repeated points had only a logarithmic (rather than linear) influence on the calculated average. This ultimately produced models with lower (better) average fitness ($46.67$ for **MFL** vs $47.25$ for **MFE**). However, in testing, we found that **MFD** produced models with the lowest (best) fitness of the three alternatives, and so **MFD** is the chosen *Delegate Point Election* algorithm. Figure 5 shows how **MFD** elects a delegate point, compared to simply using the central point of a box.

In total, the multi-fidelity techniques employed in our implementation encompass the following additions to the existing GPHH for EMD implementation:

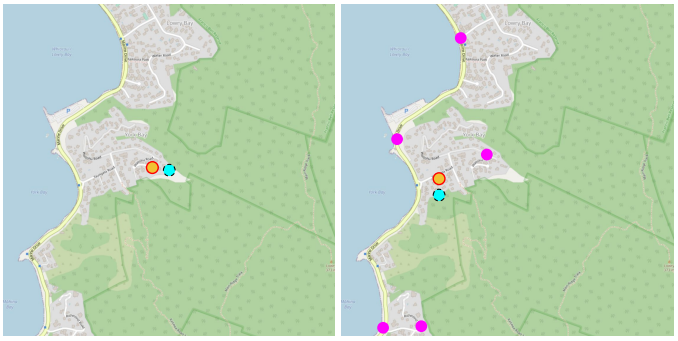

Fig. 5: **Left**: Delegate point in a grid square chosen by center point method, and automatically snapped point by ORS to nearest street. **Right**: Delegate point in a grid square elected using **MFD**. The magenta points denote the previously requested points, the dashed outlined point denotes the average point chosen as the delegate, and the ringed point denotes the automatically snapped point by ORS to nearest street from the delegate.

- Subdivision of the original graph (Section IV-A) during initialisation
- Change of graph fidelity through re-subdivision in accordance with the staging strategy (Section IV-B),
- Observation and collection of full-fidelity route queries during evaluation (Section V-B)
- Graduating boxes from inactive to active states between generations (Section V-B1), and
- Fulfillment of route requests between points with routes between quantised delegate points during evaluation.

## VI. Evaluation and Analysis

To evaluate the effects of employing multi-fidelity techniques in GPHH for EMD, we compare a number of models trained with multi-fidelity training against the updated baseline implementation described in Section III.

The full evaluation process involves three stages: *Training*, *Validation*, and *Testing*.

*1) Training:* As discussed in Section IV, during a full training run, we train 1000 individuals on $\mathbf{5}$ instances each for 50 generations, repeated over 30 PRNG seeds. However, due to computing resource constraints discussed in Section VI-A and following an observation that the model tended to converge well before the 25th generation (Generation 24), this was halved to 25 generations. The training fitness of each individual at each generation is ascertained by evaluating the individual on 5 instances using a graph at the prescribed fidelity level for that generation. During training, for each generation, the individual with the lowest training fitness out of the population (1000 individuals) is retained as the candidate individual for that generation. The result of training is the 25 candidate individuals and their training fitness, one for each of the 25 generations.

*2) Validation:* During validation, the 25 candidate individuals produced during training are validated on $\mathbf{250}$ instances each to ascertain their validation fitness. The validation fitness of each individual is ascertained by evaluating the individual on instances using the baseline graph without fidelity simplification. The individual with the lowest validation fitness is then the chosen individual for that run.

*3) Testing:* In testing, we test the 25 candidate individuals (which includes the chosen individual from validation) on $\mathbf{500}$ each to ascertain their test fitness. The test fitness of each individual is ascertained by evaluating the individual on instances using the baseline graph without fidelity simplification.

The result of the evaluation process for each run is the 25 individuals for a run, alongside their *train*, *validation*, and *test* fitness values. This process is repeated over 30 runs (with different PRNG seeds) to account for variations from run to run. All summary statistics are averaged across these 30 runs, and are shown alongside the standard deviation.

### A. Computing Resource Constraints

The computational load of running the GPHH for EMD implementation requires the use of a High Performance Computing (HPC) cluster. Because of the performance and optimisation focused nature of this project, it is critical that timing runs are completed on the actual hardware that the implementation will run on – a number of performance bottlenecks only manifested themselves when running on the HPC cluster, such as lock contention. The majority of timing runs when improving the original GPHH for EMD implementation to form the new baseline model were run on the *Rāpoi* HPC computing resource. However, partway through the completion of this project, the *Rāpoi* resource became unavailable to us, and so we did not have any HPC resources to complete timing runs. Eventually, we gained limited access to *NESI*, another HPC cluster, which we were able to run the final evaluations on. We would like to thank Jordan MacLachlan (of MacLachlan et al.) for his efforts in finding and gaining access to *NESI*. However, the temporary lack of computing resource which lasted over a month restricted progress on this project. In particular, we were unable to iterate on our solution based on results, as we were only able to complete the final train, validation, and test runs in the remaining time. Instead, we iterated on the multi-fidelity implementation with local testing only. Because of the change from *Rāpoi* to *NESI*, training times are not comparable between Section III and Section VI, though we estimate that *NESI* is approximately 2x faster than *Rāpoi*.

### B. Fidelity Sequences

We evaluate three separate fidelity sequences to investigate how different sequences affect training. All three sequences use the same generation ranges, starting at fidelity 50 and ending at fidelity 3000, but have differing intermediate fidelity levels. We expect that the sequences with lower intermediate fidelity levels will complete training in a reduced time, since a lower fidelity facilitates increased use of cached routes. However, the effect of lower fidelity on final test fitness may be more nuanced, since lower fidelity graphs may be worse predictors of the test fitness of individuals than higher fidelity graphs, but may also smooth the search space facilitating more effective learning.

| Generations | FS0 Fidelity | FS1 Fidelity | FS2 Fidelity |
|---|---|---|---|
| $[0, 5]$ | 50 | 50 | 50 |
| $[6, 10]$ | 75 | 100 | 250 |
| $[11, 15]$ | 250 | 500 | 1000 |
| $[16, 20]$ | 1000 | 1500 | 2000 |
| $[21, 25]$ | 3000 | 3000 | 3000 |

TABLE III: Fidelity levels for FS0, FS1, and FS2.

| Fidelity ($l$) | Geographical Box Size on WLG ($w \times h$) |
|---|---|
| 50 | 3.6km × 2.6km |
| 75 | 2.4km × 1.7$\overline{3}$km |
| 100 | 1.8km × 1.3km |
| 250 | 0.72km × 0.52km |
| 500 | 360m × 260m |
| 1000 | 180m × 130m |
| 1500 | 120m × 86.$\overline{6}$m |
| 2000 | 90m × 65m |
| 3000 | 60m × 43.$\overline{3}$m |

TABLE IV: Geographical box size on Wellington (WLG) graph at each fidelity level.

The models using these fidelity sequences are then called **MFD-FS0**, **MFD-FS1**, and **MFD-FS2** respectively.

Training time confirms our hypothesis that lower fidelity sequences complete training in a reduced time than higher fidelity sequences. Figure 6 shows that MFD-FS2, MFD-FS1, and MFD-FS0 all have considerably lower training times than the baseline, with the average training times 41%, 48.4%, and 51.3% faster than the baseline respectively. After removing extreme values [1], mean test fitness does not differ significantly between the four tested models. However, standard deviation does differ – the multi-fidelity models exhibit a higher standard deviation, and Figure 7 shows that fitness values for MFD-FS2, MFD-FS1, and MFD-FS0 are spread out across a larger range than the baseline. We attribute this increased variation in multi-fidelity models to an increased variance in the produced rules/individuals. Figure 8 shows that the aggregate tree size of the individuals vary more across the 30 runs of each multi-fidelity model than the baseline.

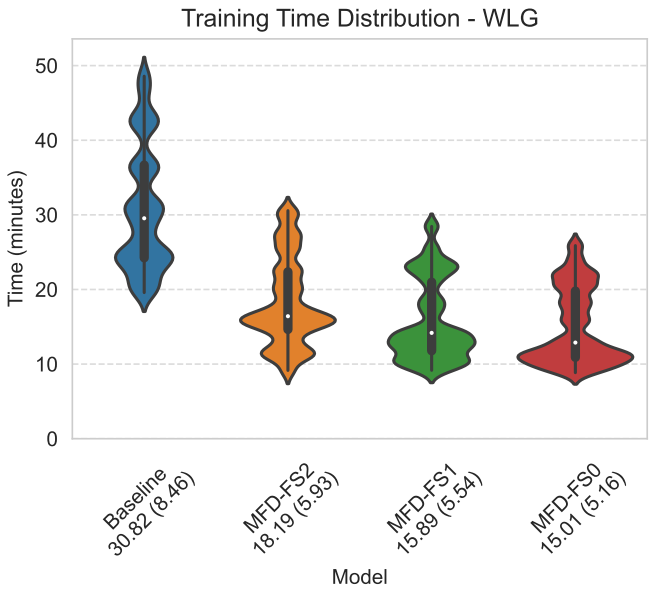


Fig. 6: Violin plot showing the distribution of training times over 30 runs. $x$ axis labels show the model, the mean training time, and the standard deviation in brackets. An outlier time of 48 minutes for MFD-FS1 has been removed, as the scheduler-measured job time was not consistent with the internally measured running time. Thus, MFD-FS1 is aggregated across 29 runs.

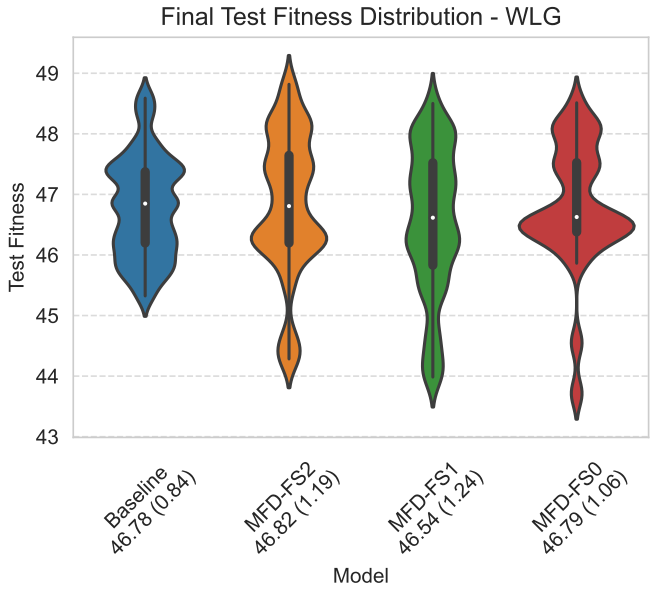


Fig. 7: Violin plot showing the distribution of test fitness over 30 runs [2]. $x$ axis labels show the model, the mean test fitness, and the standard deviation in brackets.

To show the learning effectiveness throughout the training generations, we plot the average test fitness of the candidate individual at each generation across the 30 training runs. Figure 9 shows multi-fidelity models starting at a higher test fitness than the baseline in the first generation, before the average test fitness drops to become equivalent to the baseline model by generation 8. In generations $[0, 5]$, while the baseline has consistently lower test fitness than the multi-fidelity models, the multi-fidelity models are still able to learn effectively, as the test fitness reduces in tandem with the baseline. As discussed in Section V-B, the first generation of each fidelity stage is evaluated at perfect fidelity. However, since the *In-Transit Simulation Simplification (Section IV-C)* still applies, the graph is still a surrogate replacement of the original graph, and thus there remains some increase in

[1] Three extreme fitness values above 60 (those which perform worse than *closest-idle*) have been removed, two from MFD-FS0 and one from the baseline, in order to more accurately reflect the comparative test performance between the baseline and the shown models. Extreme test fitness values have been equally observed in both baseline and multi-fidelity models, and thus we believe that this is an issue unrelated to the implementation of multi-fidelity learning. The cause of individuals exhibiting high test fitnesses despite low validation fitnesses is unclear at present.

[2] See footnote 1.

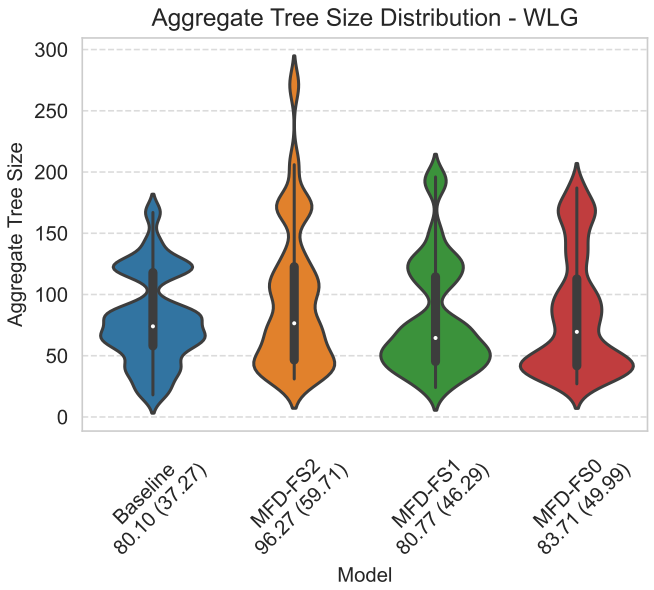


Fig. 8: Violin plot showing the distribution of aggregate tree size over 30 runs. Aggregate tree size is defined as the number of nodes in all 5 GP trees for EMD decisions D1-5. $x$ axis labels show the model, the mean aggregate tree size, and the standard deviation in brackets.

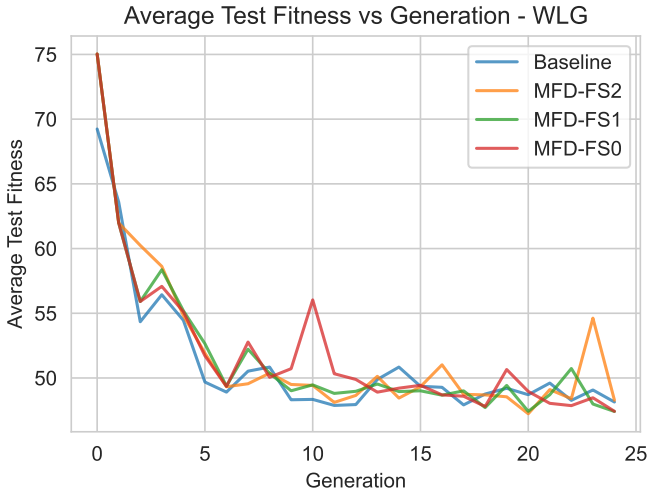


Fig. 9: Line graph showing mean test fitness for candidate individuals at each generation.

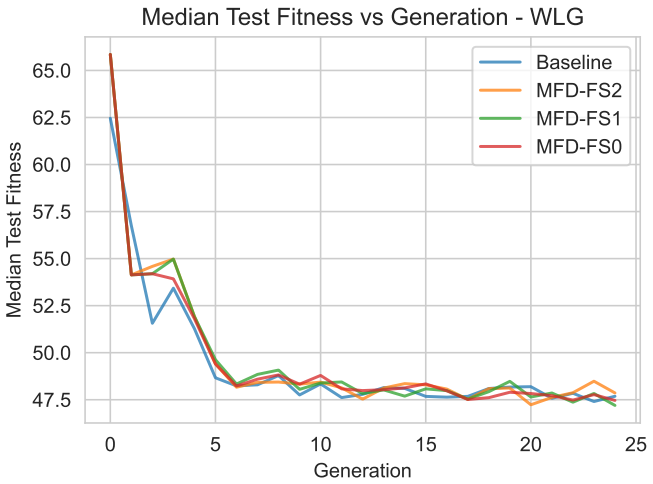


Fig. 10: Line graph showing median test fitness for candidate individuals at each generation.

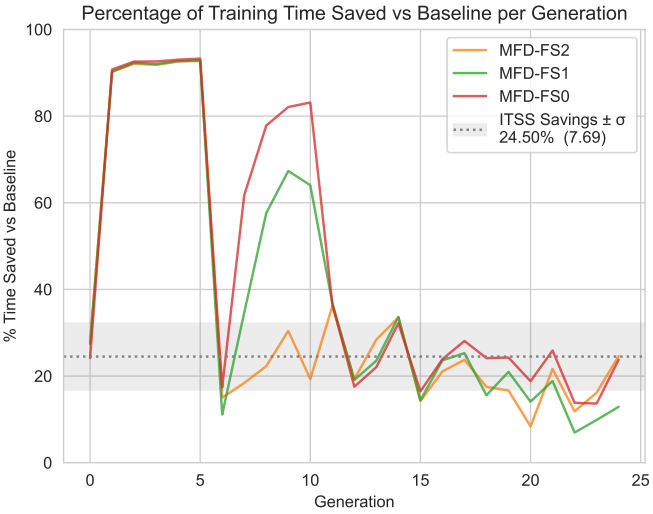


Fig. 11: Line graph showing the average percentage of time saved per generation, relative to baseline: $\frac{t_b - t_{\text{mf}}}{t_b}$, where $t_b$ is the time taken to train the baseline at that generation, and $t_{\text{mf}}$ is the time taken to train the multi-fidelity model at that generation. "ITSS Savings" line and shaded area shows mean $\pm$ $\sigma$ percentage time saved through ITSS, obtained by averaging percentage of time saved across generations at the start of each fidelity stage, across all multi-fidelity models.

approximation error. Therefore, the evaluation is less effective at assigning the lowest training fitness to the best individual, a limitation further exacerbated by the fact that the first generation is randomly initialised. After generation 8, training progress is similar for all models. However, MFD-FS0 and MFD-FS2 show peaks in average test fitness at generation 10 and 23 respectively. Further analysis shows that these peaks are the result of a small number of extreme values that impact the mean. Figure 10 shows that median test fitness does not peak in this manner.

Overall, we conclude that the multi-fidelity models are able to learn as effectively as the baseline, but in a significantly reduced training time. However, the majority of these time savings occur within the first 10 generations. Figure 11 shows that for the first 6 generations (generations $[0, 5]$), average training time for that generation is over $90\%$ (up to a maximum of $93\%$) lower in the latter 5 generations compared to the baseline. The stable percentage time saved for generations $[0, 5]$ suggests that almost all boxes that graduated in the first fidelity stage for all multi-fidelity models (since all have the same first fidelity stage) did so in the graduation step between generation 0 and 1. By contrast, the ramp up in percentage

time saved for the second fidelity stage $[6, 10]$ indicate that boxes continued to graduate as the generations increased. In particular, for MFD-FS0, the percentage time saved appears to increase logarithmically between $[6, 10]$ from $15\%$ to $83\%$, suggesting that a consistent proportion of inactive boxes graduated to active states between each generation. A similar pattern is observed for MFD-FS1, but the percentage time saved is significantly reduced for MFD-FS2, within the range achieved through ITSS alone. Past generation 10, percentage time saved for all multi-fidelity models falls to around $20\%$, with savings dropping below those achieved through ITSS past generation 20. From this, we conclude that fidelity levels greater than 250 do not contribute positively to a reduced training time. However, ITSS remains an effective method to optimise training speed, reducing training time by an average of $24.5\%$ whilst not making a measurable impact on test performance during training.

Combined with the observation that training is effective even at the lowest fidelity levels, because the majority of percentage time saved occurs during fidelity stages with fidelity levels well below 250, we believe that a greater emphasis should be placed on low fidelity surrogates in future fidelity sequences. It is clear that, particularly in early generations, there is sufficient accuracy in low fidelity graphs to facilitate learning, though it is unclear whether this holds true for later generations.

Since MFD-FS1 has the lowest test fitness $(46.54)$ and yet maintains a significant training time reduction $(48.4\%)$ over the baseline, we choose it as the candidate multi-fidelity model for further analysis. To see the speedup and learning progress MFD-FS1 compared to the baseline, we plot the average test fitness in realtime, showing the velocity of training at each generation. Figure 12 shows the effectiveness of low-fidelity training – by the time that the baseline has trained to generation 5, MFD-SF1 has trained generation 15, and MFD-SF1 completes training before the baseline has reached generation 10.

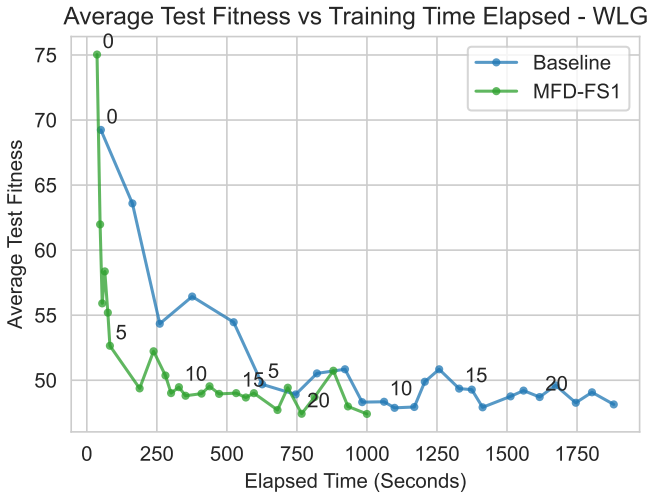


Fig. 12: Line graph showing average test performance of candidate individuals at each generation, placed on the $x$ axis according to the elapsed time when training for that generation has concluded. Line markers indicate each generation, and every 5th generation, marker labels (displayed to the top right of the marker) indicate the generation number.

We also plot the 25th, 50th, and 75th percentiles of test fitness across the 30 runs for each generation to investigate how the fitness values converge as training progresses. We use percentiles rather than mean $\pm\, \sigma$ due to the right skewed distribution of fitness, particularly at early generations. Figure 13 shows that both the median fitness and the Interquartile Range (IQR) remain consistent between the baseline and MFD-FS1, with both models converging to a stable IQR by generation 10. Past generation 10, mean and median fitness values reduce very little, suggesting that the individuals with low fitness values found during these generations is not the result of continued/cumulative learning, but rather random chance. However, the probability of finding exceptional individuals increasing as more trials/generations occur, and as such, many of the individuals chosen during validation are candidate individuals from later generations.

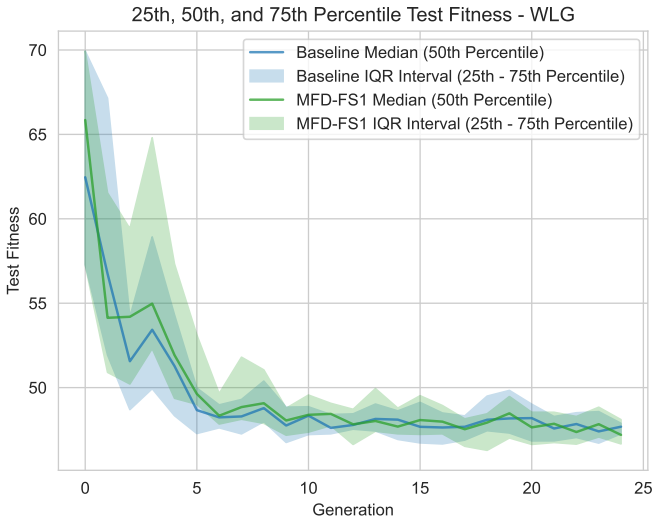


Fig. 13: Line graph showing 25th, 50th, and 75th percentiles of test fitness across 30 runs for each generation of training, for the baseline model and MFD-SF1 on WLG.

### *C. Model Generalisability – Christchurch (CHC)*

As discussed in Section II-A, a limitation of the existing implementation is that training starts with a randomised initial population, meaning that even rudimentary heuristics can take multiple generations to emerge through training. As a result, we aimed to apply transfer learning techniques to GPHH for EMD, building a baseline population that could be used as the seed population for training on novel geographical areas. Such a baseline population should not be specialised to any particular graph, instead containing general dispatch heuristics that could be refined during further (transfer) training. We hypothesised that by applying our multi-fidelity techniques to training, the resulting individuals would be more generalisable since the lack of precision/detail in a low-fidelity graph could

limit specialisation to a particular geograph. More generalisable individuals would thus be more suitable for transfer learning. To test this, we ran the *validation* and *test* stages with the Christchurch (CHC) graph, using both the baseline and MFD-FS1 trained on the WLG graph. This combination of training on the WLG graph, then validating and testing on the CHC graph is denoted as "WLG-CHC". Figure 14 shows

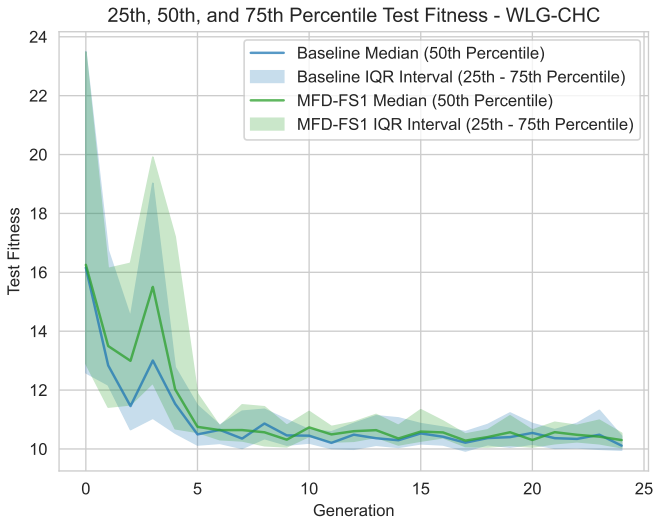


Fig. 14: Line graph showing 25th, 50th, and 75th percentiles of test fitness across 30 runs for each generation of training, for the baseline model and MFD-SF1. The models are trained on WLG, while test fitness is evaluated on the CHC graph.

the 25th, 50th, and 75th percentiles of test fitness for each generation. We see no significant difference in IQR between the two models, with IQR reducing to a consistent level by generation 10, mirroring the behaviour when tested on the WLG graph. MFD-FS1 performs worse in test fitness for the first 5 generations, but since this also occurs when testing on WLG, this is likely due to a slightly reduced overall learning effectiveness at low fidelities rather than reduced generalisability.

Figure 15 shows that none of the multi-fidelity models perform significantly better or worse on average than the baseline on CHC, when trained on WLG. Overall, we do not observe any significant impact of multi-fidelity on the generalisability of the produced models.

Accordingly, we propose that to reduce the time used to build rudimentary heuristics on novel geographical areas, early generations should use a randomised initial population as standard, but run at lower fidelities, reducing the time taken to complete training for each generation while not biasing the population from prior learning on a baseline graph.

## VII. Conclusions and Future Work

This project aims to reduce the training time of the GPHH for EMD implementation through both engineering improvements and the implementation of multi-fidelity training techniques. We make a number of engineering improvements to the original GPHH for EMD implementation to improve source

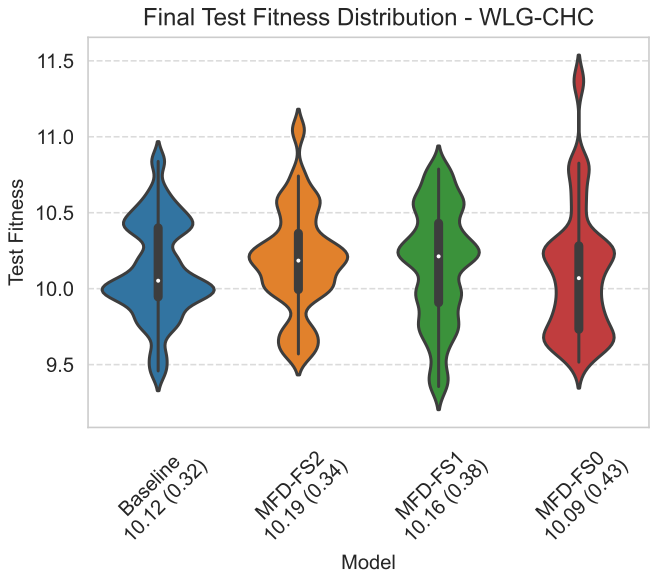


Fig. 15: Violin plot showing the distribution of WLG-CHC test fitness over 30 runs. $x$ axis labels show the model, the mean test fitness, and the standard deviation in brackets.

code portability and execution performance, forming a new baseline implementation that is 6x faster compared to the original implementation. We then propose a set of multi-fidelity algorithms that enable the approximation of route-finding in a geographical, coordinate based graph by subdividing the geography into boxes, and quantising endpoints of route requests into elected delegate points within those boxes. We integrate an implementation of these multi-fidelity algorithms into the baseline GPHH for EMD implementation, and compare training execution time and test fitness values throughout training. We find that multi-fidelity techniques do not significantly impact test fitness nor model generalisability compared to the baseline model, but achieve comparable fitness levels between $41\%$ to $51.3\%$ faster. In addition, we find that the speedup gained from these multi-fidelity techniques is concentrated at fidelity stages with fidelity levels below $250$, with per-generation training time reduced by over $90\%$ at the lowest fidelity level.

These results indicate that multi-fidelity techniques in GPHH for EMD are feasible and provide various opportunities for future work to explore in greater depth:

1) **Extended training at low fidelity**
   Our results indicate that the time savings are concentrated at the lowest fidelity levels (those below $250$), and that training effectiveness is not significantly impacted by the loss in route accuracy in early generations.
   However, further work could explore whether these low fidelity levels continue to facilitate training at later generations, or if fidelity must necessarily increase throughout the training process.
2) **Multi-fidelity testing**
   At present, multi-fidelity is only used during training, reducing training time by up to $51.3\%$. As discussed in Section I, fitness evaluation constitutes the majority of

training time, a process also completed during testing.
Testing also takes a significant amount of time to complete, since 500 evaluation instances are used in testing rather than 5 during training, and it is completed using the original graph at full fidelity.
Further work could explore the feasibility of completing testing at a reduced fidelity, in particular the correlation between test fitness at various fidelity levels compared to full fidelity. If feasible, multi or reduced fidelity testing could then further reduce iteration cycle time via a reduction in testing time.

3) **Mixed-fidelity graphs**
In our multi-fidelity implementation, we subdivide a graph into a fixed number of boxes, regardless of the usage of endpoints within those boxes.
Our implementation could be extended to further subdivide boxes so as to improve accuracy for oft-queried boxes (such as those residing in cities) whilst exploiting the speedup from large boxes in less dense areas, such as the countryside. Of particular value would be an exploration into whether this recursive subdivision results in bias, such as improved response times in urban areas and worse response times in rural areas.
4) **Affects of multi-fidelity training on diversity**
In our testing, multi-fidelity models exhibited slightly higher test fitness standard deviation from run to run.
Further exploration into the effects of surrogate-based multi-fidelity training on population diversity would help us to further enhance our multi-fidelity techniques.

## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Yi Mei for his unwavering support and guidance throughout this project. I would also like to thank Jordan MacLachlan, whose research this project extends, for his support and encouragement, and for the countless discussions that were pivotal in shaping this project. I am also deeply grateful to my girlfriend Harley for her enduring motivation and encouragement throughout university, which have been instrumental in my academic achievements. Finally, I owe a debt of gratitude to my parents for their love, belief in my abilities, and extraordinary support throughout my education.

## REFERENCES

[1] J. holmén, J. Herlitz, S. Ricksten, A. Strömsöe, E. Hagberg, C. Axelsson, and A. Rawshani, "Shortening ambulance response time increases survival in out-of-hospital cardiac arrest," *Journal of the American Heart Association*, vol. 9, 05 2020.

[2] W. J. Brady, A. Mattu, and C. M. Slovis, "Lay responder care for an adult with out-of-hospital cardiac arrest." *The New England journal of medicine*, vol. 381 23, pp. 2242–2251, 2019.

[3] C. Jagtenberg, S. Bhulai, and R. Mei, "Dynamic ambulance dispatching: is the closest-idle policy always optimal?" *Health Care Management Science*, vol. 20, 12 2017.

[4] J. MacLachlan, Y. Mei, F. Zhang, M. Zhang, and J. Signal, "Learning emergency medical dispatch policies via genetic programming," in *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO)*. ACM, jul 2023.

[5] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, *Exploring Hyper-heuristic Methodologies with Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 177–201. [Online]. Available: https://doi.org/10.1007/978-3-642-01799-5_6

[6] J. Lowthian, P. Cameron, J. Stoelwinder, A. Curtis, A. Currell, M. Cooke, and J. Mcneil, "Increasing utilisation of emergency ambulances," *Australian health review : a publication of the Australian Hospital Association*, vol. 35, pp. 63–9, 02 2011.

[7] E. Mahase, "Covid-19: High prevalence and lack of hospital beds putting "intense pressure" on ambulances," *BMJ*, vol. 378, 2022. [Online]. Available: https://www.bmj.com/content/378/bmj.o1763

[8] J. Koza and R. Poli, *Genetic Programming*, 01 2005, pp. 127–164.

[9] S. Luke and L. Panait, "A survey and comparison of tree generation algorithms," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 81–88.

[10] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, "Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm," *ACM Journal of Experimental Algorithmics*, vol. 15, 01 2010.

[11] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, pp. 3–12, 10 2005.

[12] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling: An Evolutionary Learning Approach*, 01 2021.

[13] N. Javed, F. Gobet, and P. Lane, "Simplification of genetic programs: A literature survey," *Data Min. Knowl. Discov.*, vol. 36, no. 4, p. 1279–1300, jul 2022. [Online]. Available: https://doi.org/10.1007/s10618-022-00830-7

[14] L. Wang, Y. Yao, T. Zhang, C. D. Adenutsi, G. Zhao, and F. Lai, "A novel self-adaptive multi-fidelity surrogate-assisted multi-objective evolutionary algorithm for simulation-based production optimization," *Journal of Petroleum Science and Engineering*, vol. 211, p. 110111, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920410522000080

[15] Y. Tenne and C.-K. Goh, *Computational Intelligence in Expensive Optimization Problems*, 01 2010.

[16] A. I. Esparcia-Alcázar and J. Moravec, "Fitness approximation for bot evolution in genetic programming," *Soft Computing*, vol. 17, no. 8, pp. 1479–1487, Aug 2013. [Online]. Available: https://doi.org/10.1007/s00500-012-0965-7

[17] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary computation*, vol. 23, 06 2014.

[18] P.-E. Réthoré, P. Fuglsang, G. Larsen, T. Buhl, T. Larsen, and H. Madsen, "Topfarm: Multi-fidelity optimization of wind farms," vol. 17, 12 2014.

[19] Y. Song, Q. Cheng, and S. Koziel, "Multi-fidelity local surrogate model for computationally efficient microwave component design optimization," *Sensors*, vol. 19, p. 3023, 07 2019.

[20] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651–665, 2021.

[21] S. Nguyen, M. Zhang, and K. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Transactions on Cybernetics*, vol. X, 04 2016.

[22] D. Yska, Y. Mei, and M. Zhang, *Genetic Programming Hyper-Heuristic with Cooperative Coevolution for Dynamic Flexible Job Shop Scheduling*, 01 2018, pp. 306–321.

[23] E. O. Scott and S. Luke, "Ecj at 20: Toward a general metaheuristics toolkit," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1391–1398. [Online]. Available: https://doi.org/10.1145/3319619.3326865

[24] "Oracle java se support roadmap — oracle australia," https://www.oracle.com/au/java/technologies/java-se-support-roadmap.html.

[25] "Forkjoinpool (java se 17 & jdk 17)," https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/concurrent/ForkJoinPool.html.

[26] "Overview (java se 17 & jdk 17)," https://tinyurl.com/yc8xezxy, Apr 2023.

[27] "Guava," https://guava.dev/.

[28] Ben-Manes, "Caffeine: A high performance caching library for java," https://github.com/ben-manes/caffeine.

[29] G. Einziger, R. Friedman, and B. Manes, "Tinylfu: A highly efficient cache admission policy," *ACM Trans. Storage*, vol. 13, no. 4, nov 2017. [Online]. Available: https://doi.org/10.1145/3149371

[30] S. Gakis and N. Everlönn, "Java and kotlin, a performance comparison," 2020. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:hkr:diva-20721