1 Analysis of Particle Swarm Optimisation

1.1 The velocity update

1.1.1 Initialisation

We ran PSO with $N = 80$ random particles at zero initial velocity on $d = 8$ dimensional $f_1 := \textit{Sphere}$ and $f_2 := \textit{Rastrigin}$ objective functions. The velocity update rule was $v_i \leftarrow \omega v_i + \alpha_1 r_1 * (p_i - x_i) + \alpha_2 r_2 * (g - x_i)$; $x_i \leftarrow x_i + v_i$ with standard notation. A particle's position would reset upon exiting the search range $[-5.12, 5, 12]^d$, in order to avoid overflow. A run would stop when either the maximum amount of iterations $T = 150$ was reached, or when the global best fitness value f_g would drop below the goal ϵ, or when f_g would stagnate for $T/2 = 75$ iterations. These termination conditions capture both PSO's ability to make small adjustments on unimodal functions like f_1 and its tendancy to get stuck on multimodal functions like f_2.

1.1.2 Finding α, ω, α_1, α_2

We began by searching over (ω, α), as theory suggests that $\alpha = \alpha_1 + \alpha_2$ is critical for convergence [1], followed by a search over α_1 with $\alpha_2 = \alpha - \alpha_1$. We scanned progressively smaller grids of parameters centered on the previous best parameter pair, with resolution 20×20, beginning with $\omega \in [-1, 1]$ and $\alpha \in [-4, 4]$, motivated by theoretical considerations outlined in the course [2].

f_1	$\epsilon = 0.01$	$\omega = 0.304$	$\alpha = 3.789$	$\alpha_1 = 1.989$	$\alpha_2 = 1.799$	Avg. iters = 23.44
f_2	$\epsilon = 10$	ω = -0.342	$\alpha = 2.131$	$\alpha_1 = 1.250$	$\alpha_2 = 0.881$	Avg. iters = 34.84

For f_2, PSO performed best when the particles had negative inertia. This goes against standard guidelines for parameter values, but agrees with later work on meta-optimization [3][4]. The resulting zig-zagging motion may be more suited towards f_2's multimodal landscape.

1.2 Varying N and d

As d increases, the value of N required in order for PSO to converge within T iterations follows an exponential law (fig.1). We claim that this is counteracting the decreasing size of the goal region G. We believe that the 2^d-scaled ratio of remaining PSO iterations for fixed N (fig.2) is specific to the volume of G because when the swarm samples the search space, it performs a form of Monte-Carlo estimation of $|G|$. Here, G, is revealed to be a d-sphere. The volume of a d-sphere follows a pattern of exponential decrease after a local maximum, and this explains the need for exponentially many particles [6].

1.3 Adaptive Inertial Weight Particle Swarm Optimisation (AIW-PSO)

In PSO, ω is used to balance exploration (large ω) and exploitation (small ω) of the entire swarm [7]. AIW-PSO is finer-grained, allowing each particle i to choose its weights $\omega_{ij}(t) = 1 - \frac{\beta}{1+e-ISA_{ij}}$, where $ISA_{ij} = \frac{|p_{ij} - x_{ij}|}{|p_{ij} - g_j| + \varepsilon}$ is the *individual search ability*, $\varepsilon > 0$, for $j = 1...d$. The weights adapt to the following two scenarios. When p_i is closer to x_i than g, then particle i may be trapped in a local optimum, in which case ISA_{ij} is small and w_{ij} is large, leading to dynamically increased exploration along dimension j. The opposite takes place when p_i is closer to g than x_i. The rate of transfer from ISA to w is controlled by the parameter β [8]. We observe accuracy and convergence speed improvements for both f_1 and f_2 (fig.3).

f_1	$\beta = 1.05$	$\alpha_1 = 1.99$	$\alpha_2 = 0.90$
f_2	$\beta = 0.75$	$\alpha_1 = 1.25$	$\alpha_2 = 0.44$

2 Genetic Programming

Having added a number of features to GP in hopes of encouraging the symbolic regression of f_1 and f_2, we compare the performance of base GP with all combinations of added features on the basis of convergence speed for f_1. For f_2 we study the population diversity and fitness.

A: Hill-climbing	**B**: Vertical shift fitness term	**C**: Elitism

First, **A** allows the generation's single best program to refine its leaves, keeping improvements with probability 0.5 to avoid early convergence. Secondly, **B** attempts to determine whether there exists k such that $gp(x) + k = f(x)$, by minimizing the standard deviation of $gp(x) - f(x)$. Such programs are usually selected against because they don't hit the dataset. However, this may be hacked by programs outputting enormous values, so its weight is set to 0.1. Finally, **C** seeds half the population of sequential restarts with the previous best program.

2.1 Sphere symbolic regression

We define the following problem setup for GP with selection by tournaments of size 5.

Terminals	Goal	p_{cross}	p_{mgrow}	p_{mpoint}	Init. type
$x, 2, 10, \pi$	0.01	0.9	0.1	0.02	Ramped half-and-half
Nonterminals	Function	Restarts	Gens	Data	Fitness
$+, \times, -, \exp_x, \cos, \sum$	f_1	10	30	40	Sum of absolute error

Any added feature alone causes the search to slow down for population sizes ≤ 200 (fig.4). This is quite countertintuitive, because constrained search usually implies smaller search spaces. However, GP relies on the behavioural diversity of its population, and these constraints may cause its collapse [10]. Larger population sizes likely act as a remedy for this issue, where all GP+ variations have similar performance. Combining the added features also seems to restore the performance to the level of base GP.

2.2 Rastrigin symbolic regression

We evaluate the performance of GP+ABC for a population size of 600, and the same problem setup besides the following changes.

p_{mgrow}	Function	Restarts	Max depth
1/64	f_2	5	9

The implementation of the growth mutation is such that the depth of a program tree follows a geometric distribution, and averages around $\log_2(1/p_{mgrow})$, so $p_{mgrow} = 1/64$ encourages deeper trees. For a given run, the best program discovered was $gp(x) = (\sum x^2) + 10 + \sum\sum(2\pi - \cos(x))$, which contains a sum of a constant. An additional feature **D** was previously considered that would ban such *introns*, to limit bloat, however, results did not improve. Overall, performance was poor, perhaps because there is no explicit mechanism to maintain diversity other than the periodic elitist restarts (fig.5). A variant of GP that may perform better at this task is *neat-GP*, which allows the population to be split up into species according to gene history, and performs incremental improvements from very small initial trees to control bloat [9].

3 GP versus m-HGPPSO

3.1 Definition

We implemented a slightly modified HGPPSO (m-HPPSO) and compared it to base GP on convergence speed and fitness [11]. HGPPSO imagines programs taking a position x_i in the space of all possible programs, where each particle is aware of its own personal best position p_i as well as the global best g. This means that particles are encoded as trees instead of real vectors. Therefore, the position update rule in PSO, which looks like

$$x_i \leftarrow f(x_i, p_i - x_i, g - x_i) \tag{1}$$

in the abstract, must be adapted for tree encodings. Since GP uses evolutionary operators, the following update rule is proposed instead.

$$x_i \leftarrow RWS(x_i, MAX(OPC(x_i, p_i)), MAX(OPC(x_i, g))), \tag{2}$$

where

1. RWS is roulette-wheel selection
2. MAX is max-fitness selection
3. OPC is one-point crossover

To express distance, PSO uses vector subtraction, while HGPPSO uses one-point crossover. We further modify HGPPSO by introducing the *grow* and *point* mutation operators into the update rule, inspired by PSO's component-wise multiplication of $p_i - x_i$ and $g - x_i$ by random vectors.

$$x_i \leftarrow RWS(x_i, PNT(GRW(MAX(OPC(x_i, p_i)))), PNT(GRW(MAX(OPC(x_i, g))))) \tag{3}$$

3.2 Comparison

We observe clear improvement of m-HGPPSO over GP for the following standard problem setup (fig. 6).

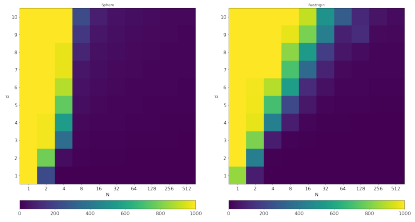
Terminals	Goal	p_{cross}	p_{mgrow}	p_{mpoint}	Init. type
x	0.01	0.9	0.1	0.02	Ramped half-and-half
Nonterminals	Function	Restarts	Gens	Data	Fitness
$+, \times$	$x^4 + x^3 + x^2 + x$	50	40	40	Sum of absolute difference

We argue that these results are analagous to the relative performances of PSO and FIPSO. In GP, programs can be seen as particles that are fully informed, in the sense that each program may cross over with any other, whereas in HGPPSO, only the best program may affect the rest. If this analogy holds, we expect that allowing a small number of best-performing programs to cross over with the rest of the population may prove to be even better than m-HGPPSO, as seen in LIPSO [12]. By limiting communication between particles, we promote global exploration and maintain population behavioural diversity.

4 Symbolic Regression

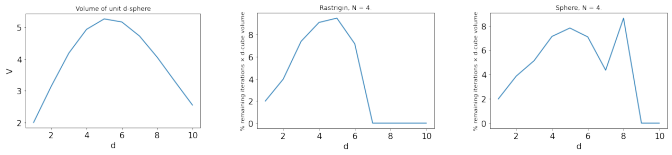
(see code).

Figure 1: Iterations until N-particle PSO convergence on f_1 (a) and f_2 (b) with dimension d. T = 1000.



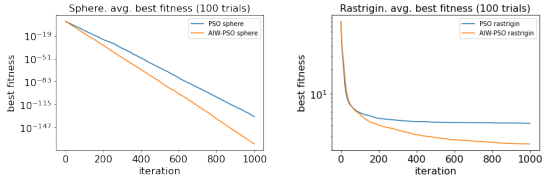
(a) (b)

Figure 2: (a) Volume of unit d-sphere. (b) Percentage of remaining iterations $\times\ 2^d$. $N = 4$ and $T = 1000$, averaged over 50 trials. Peaks in (b) reach higher than in (a) because PSO performs better than random Monte-Carlo sampling and is thus left with more iterations on average.



(a) (b) (c)

Figure 3: Comparison of average best fitness between PSO and AIW-PSO, averaged over 100 trials, for f_1 (a) and f_2 (b).



(a) (b)

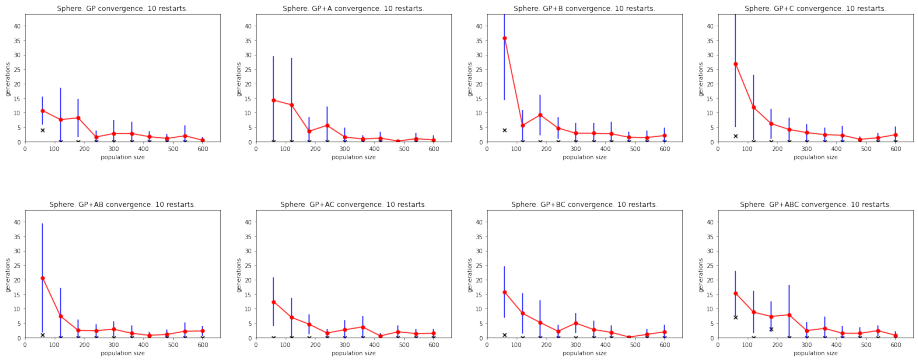
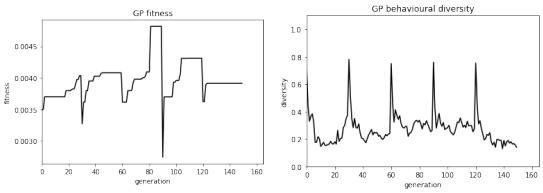


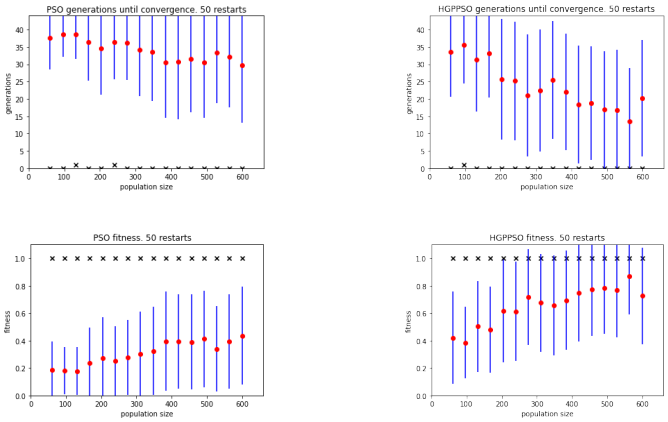
Figure 4: Comparison of convergence speed of f_1 for different combinations of added features.

Figure 5: Fitness and behavioural diversity of GP+ABC on f_2 for a run over 150 generations.



(a) (b)

Figure 6: Average, standard deviation, and best number of generations and fitness, for GP and HGPSO over 50 restarts.



References

[1] M. Clerc and J. Kennedy. The Particle Swam– Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE transactions on Evolutionary Computation*, Volume 6, Pages 58-73, 2002.

[2] M. Herrmann. Natural Computing Tutorial 2: Particle Swarm Optimisation. INFR1161, University of Edinburgh, 2021.

[3] R.C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. *Proceedings of the Congress on Evolutionary Computation*, Volume 1, Pages 81 – 86, 2001.

[4] Pedersen, Magnus Erik Hvass. Tuning & simplifying heuristical optimization. Diss. University of Southampton, 2010.

[5] Kennedy, J. (2003, April). Bare bones particle swarms. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium.* SIS'03 (Cat. No. 03EX706), Pages 80-87. IEEE.

[6] Weisstein, Eric W. "Hypersphere." From MathWorld–A Wolfram Web Resource. `https://mathworld.wolfram.com/Hypersphere.html`

[7] R.C. Eberhart and Y. Shi. A modified particle swarm optimizer. *Proceedings of the Congress on Evolutionary Computation*, Pages 69-73, 1998.

[8] Qin, Z., Yu, F., Shi, Z., and Wang, Y. Adaptive inertia weight particle swarm optimization. *International conference on Artificial Intelligence and Soft Computing*, Pages 450-459. Springer, Berlin, Heidelberg. 2006.

[9] Trujillo, Leonardo, et al. neat genetic programming: Controlling bloat naturally. *Information Sciences* Volume 333, 21-43, 2016.

[10] Gustafson, Steven Matt. An analysis of diversity in genetic programming. Diss. University of Nottingham, 2004.

[11] Qi, F., Ma, Y., Liu, X., and Ji, G. A hybrid genetic programming with particle swarm optimization. *International Conference in Swarm Intelligence*, Pages 11-18. Springer, Berlin, Heidelberg. 2013.

[12] Wen-Bo Du, Yang Gao, Chen Liu, Zheng Zheng, Zhen Wang, Adequate is better: particle swarm optimization with limited-information. *Applied Mathematics and Computation*, Volume 268, Pages 832-838. 2015.