Gastropod Mollusc (or Slug) Optimisation Algorithm

Technical Report on Optimization Techniques for Problem Solving in Uncertainty

Ben Kenwright

Abstract

This chapter presents a nature–inspired computing optimisation algorithm. The computational algorithm is based upon the patterns and behaviours of the extraordinary and under–appreciated Gastropod Mollusc (or Slug). The slug which has been around since the ice–age, belongs to a fascinating and complex group of creatures whose biology is every bit as interesting and worthy of admiration as Earth's more loved and head line grabbing species. As we explain in this chapter, slugs are simple creatures but are able to solve complex problems in large groups (one of nature's evolutionary triumphs). These abilities form the underpinnings of the 'slug optimisation algorithm' (SOA) presented in this chapter. What is more, the optimisation algorithm is scalable and can be implemented on massively parallel architectures (like the graphical processing unit). While algorithms, such as, the firefly, cockroach, and bee, have proven themselves as efficient methods for finding optimal solutions to complex problems, we hope after reading this chapter the reader will take a similar view on the slug optimisation algorithm.

Keywords: optimisation, adaptive, automation, nature-inspired, smart, evolutionary, population, gastropod mollusc, slug, algorithm

1 Introduction

Why is optimisation important? Optimisation is the process of making the best or most effective use of a situation or resource and as such is a vital aspect of almost every discipline from construction [19] to animation [13]. However, the search for optimality in many cases is highly challenging due to the complexity of the problems. For instance, some optimisation algorithm do not necessarily guarantee optimality will be reached. One measure of an optimisations 'quality' is its efficiency (e.g., computational cost, search time, and memory overhead). While algorithm, such as, the classical hill–climbing [27, 17] or steepest decent [7] method are efficient solutions for 'local' optimisation problems, they suffer limitations when exploring global landscape. One of the most important measures is 'time' (i.e., some algorithms due to the probabilistic nature may converge on a solution but may require huge amounts of computational power and take long durations). These challenges are truncated by the complexity of the optimisation problems and in many cases may be multi–modal and non–linear which can only be solved using 'global' optimisation methods. Of course, these global problems are very challenging. Research has suggested that one promising solution to these challenging problems is through meta–heuristic algorithms, like the particle swarm optimisation (PSO) and evolutionary differential algorithm [25, 24, 11].

What is a meta–heuristic algorithm? A large number of meta–heuristic algorithms are inspired by nature [25]. For example, the genetic algorithm which follows a survival of the fittest analogy with genes from the strongest individuals leading to the most popular choice [4]. Another popular example, is the particle swarm optimisation (PSO), with variations of this including, the firefly algorithm [24] (inspired by the flashing behaviour of fireflies) and ant–colony algorithm [5]. These meta–heuristic algorithms in one form or another have been applied to almost all areas of optimisation [25, 21, 24] (e.g., timetabling [15], manufacturing, biology and video games [11]). In this chapter, we extend the population–based analogy to create a novel meta–heuristic slug optimisation algorithm. We explain the algorithm and its practical applications for solving global non–linear functions. We provide benchmark performance data for the algorithm and compare our technique against a variety of challenging non–linear problems. In particular, our algorithm has a number of in inherent advantages: parallelizable (i.e., ported to massively parallel architectures, such as, the GPU); simple and intuitive to implement and understand; solve non–linear, multi–modal, multi–objective problems; while being relatively robust and efficient.

How do population and evolutionary based optimisation techniques work? The metaheuristic algorithms presented in this chapter is a non–deterministic optimization technique. Many people do not realize that a stochastic algorithm is nothing else than a random search, with hints chosen through heuristics (or meta–heuristics) functions to guide the solution towards the most probable answer. People who realize this feel uneasy about stochastic algorithms, because there is no guarantee that the algorithm (based on random choices) will find the global optimum [3]. However, as we and others have shown, these 'probabilistic' optimisation algorithms have proven themselves in a number of real–world situations [25, 24].

What does optimisation have to do with 'slugs'? Slugs have been around since the ice–age and are more important and common we know. For instance, a vegetated area (such as, a garden or a field), will possess on average 200 slugs per square metre. Slugs detect food by smell: the four tentacles on their head contain chemoreceptors that pick up chemicals in the air, just as your nose does. Importantly, a slug is able to find odors over large distances, as invertebrates have an amazing sense of smell, plus are often attuned to different odors. [16]). According to horticultural experts, it is estimated that an acre of farmland may support over 250,000 slugs. **Slugs but they are perfectly capable of 'communicating' the whereabouts of food**. Slugs are efficient feeders in addition to having a long evolutionary history. These invertebrates have been around for millions of years and have managed to colonise a huge variety of ecological niches – just about anywhere, in fact, where there is water and something to eat, you will find a type of slug. Although soft and seemingly fragile, slugs are actually quite robust little creatures and are able to survive freezing weather and temporary droughts. While slugs do not have language in any conventional sense, they are able to 'communicate' the whereabouts of food, for instance, through fragrances, taste and slime trails [8]. **An interesting behavioural phenomenon that has been observed is slugs tend to keep company when feeding.** For example, several common garden slugs would feed on one piece of food together, even if there are other food sources available. In addition to navigating to new source of food, slugs possess a "homing instinct" to find their way back to a patch of ground.

Chapter's Contribution The key contributions of this chapter are: (1) a efficient optimisation algorithm based on nature; (2) the innovative algorithm is able to solve a variety of real–world tasks; (3) demonstrate the algorithms scalability on massively parallel architectures (e.g., Graphical Processing Unit); (4) as we show in this chapter the proposed algorithm is capable of revealing an effective performance improvement; (5) the proposed Gastropod Mollusc (or Slug) Optimisation Algorithm (or SOA) can be utilized to provide insight into various engineering problems (6) importantly, this chapter provides benchmarking data on a variety of formidable test cases to show the viability of the proposed bio–inspired technique.

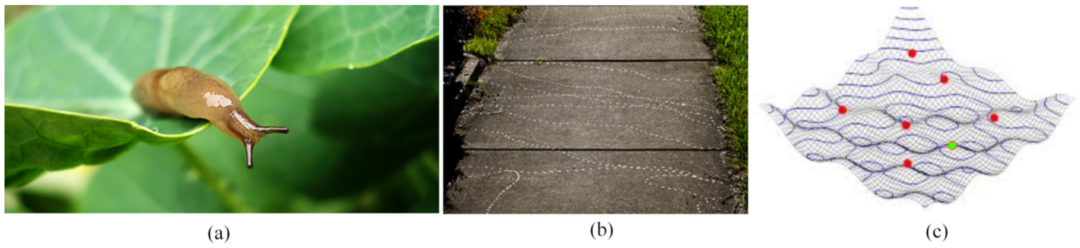

Figure 1: **Patterns -** (a) The common garden 'slug' (and other terrestrial gastropods) use the non–linear viscoelastic properties of excreted trail mucus for transitory attachment, allowing locomotion on inclined surfaces [6]. (b) The behavioural properties play an important role in numerous biological functions, for instance the locomotion strategy used by terrestrial gastropods – which we use in this chapter to develop the optimisation technique. (c) The nature–inspired embodiment of the exploratory properties for searching and finding optimal solutions for complex discontinuous multi–modal problems.

2 Related Work and Background

Slugs do play an important role in ecology. Slugs not only help with breaking down dead vegetation using their rasping tongue (called a radula), they also consume vast quantities of decaying vegetation, such as, the dead leaves after autumn. Slugs are a type of gastropod which forms the second largest class in the animal kingdom, the largest being the insects. Slugs are hermaphrodite, having both male and female reproductive organs. 95% of slugs are underground, nibbling on seeds and roots and breeding.

Optimality is an important and challenging subject in many fields. This includes, factors such as, efficiently (time and complexity), not to mention, computational cost – which form the measure of comparability between algorithms.

Many optimisation algorithms are only effective at finding 'local' optimal solutions (e.g., gradient–based algorithms like, steepest decent and hill–climbing).

Global optimisation problems are often required for 'multi–model' and 'highly non–linear' problems. These optimisation problems are often extremely challenging (especially for problems with large numbers of parameters). Of course, recent research into meta–heuristic algorithms have suggested that population based optimisations are promising for solving these tough optimisation problems [26].

Heuristics vs Meta–heuristics Optimization algorithms can be roughly divided into two categories: exact algorithms and heuristics [18]. The meta–heuristics techniques are problem–independent and as such, do not take advantage of any specificity of the problem – so are ideal for black box methods (meta–heuristic techniques are methods that optimizes a problem by iteratively trying to improve a candidate). Meta–heuristics build upon the ideology of heuristics to formulate approximate search methods for solving complex optimization problems that arise in business, commerce, engineering, industry, and many other areas. For instance, meta–heuristic algorithms guide subordinate heuristic concepts from artificial intelligence, biological, mathematical, natural and physical sciences to improve their performance. As we show in this chapter, most successful meta–heuristics algorithms in use today are nature–inspired.

Global vs Local Optimisation As we have explained, the optimization process attempts to find a solution that minimizes a function. This optimisation process is subdivided into local and global paradigms. For local search solutions the nearby data is used to steer the optimisation solution, while global solutions include all feasible data in the search criteria. Crucially, the distinction between global search and local search can be very subtle. For

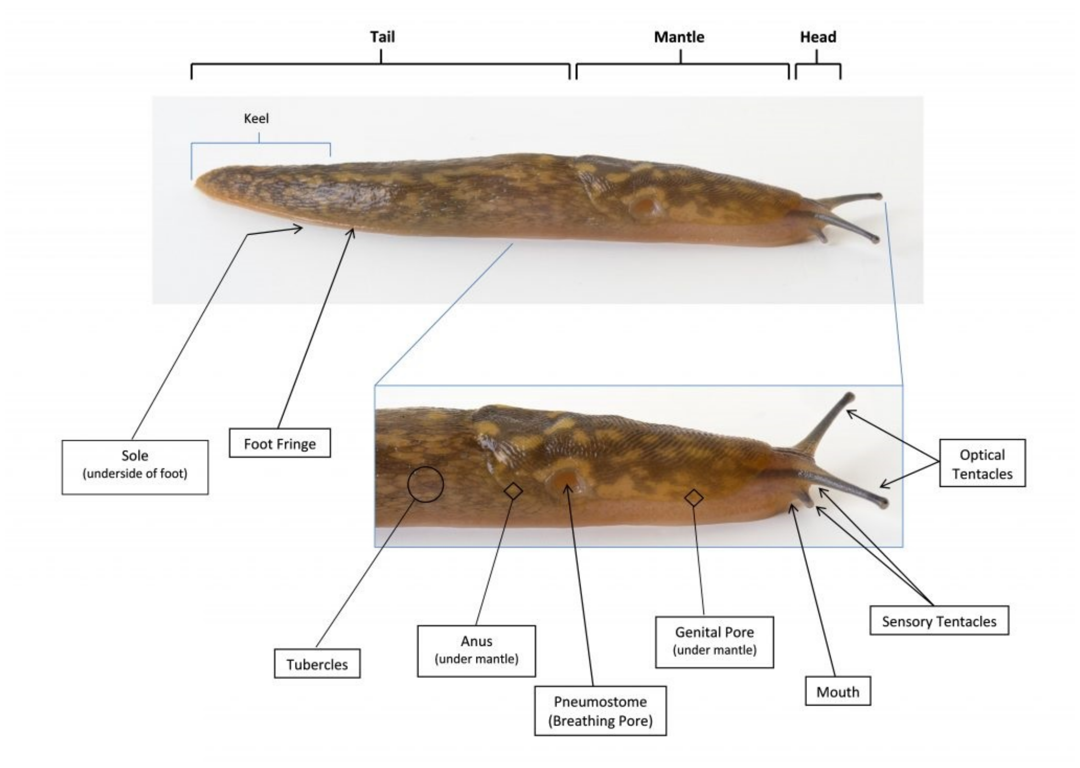

Figure 2: **Slug Anatomy** - A slug anatomy guide to help visualize and identify biological concepts.

example, many global search techniques such as, particle swarm optimisation (PSO) have their parameters adjusted to focus more on local search (e.g. higher constriction to limit the speed and movement of particles). While local search techniques are often thought of as "greedy", we must remember, global search techniques often employ elitism (e.g. genetic algorithm, differential evolution and particle swarm optimisation) which display aspects of greediness.

Deterministic or Probabilistic A search technique which always reaches the same locally optimal solution from the same starting point is probably a local search technique. Equally, the performance of a global search technique should be less dependent on its initial position(s). Whereas a local search technique will target nearby local optima, global search techniques should be able to find (local) optima anywhere in the search space.

3 Slug Optimisation Algorithm

The Slug Optimisation Algorithm (SOA) is based upon the exploratory, behavioural and navigatory patterns of slugs. The SOA algorithm is a population based global optimization algorithm. The slug is a valuable source of inspiration, for example, each slug pursues needs which are influenced by sensory input (smell and swarm state), while at the same time randomly exploring and searching. The slugs' needs are determined by the landscape and their objective function.

The motion of each slug is influenced by the success rate 'shared' by other slugs within the population (e.g., found food, but how many have found food, and is the resource over

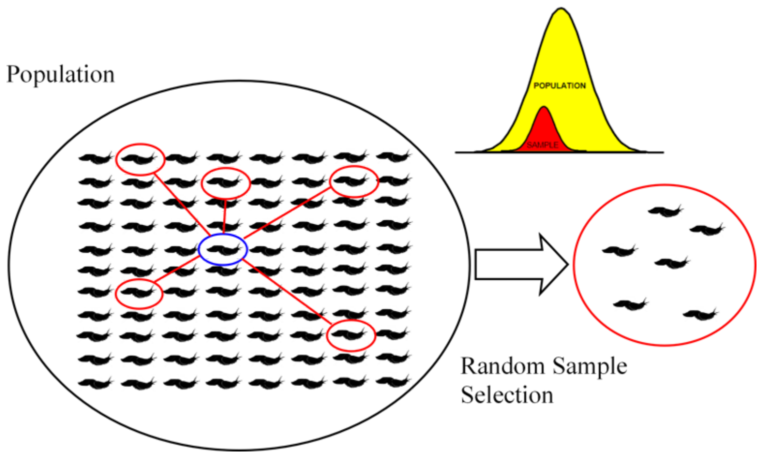

Figure 3: **Sub–Sample -** Each character selects a sub–sample from the population to provide approximate numerical characteristics of the overall population to steer the individuals decision.

subscribed, speed of the change from the current to the new).

$$x_i^{t+1} = x_i^t + v_i^t \tag{1}$$

where x is an individual within the population (identified by i), t is the evolution number, and v is the velocity (motion) by which to change the current individual (i.e., v denotes slugs migration direction based on the slugs state and behavioural pattern).

Slugs interact with peers and respond to their immediate environment and make decisions based on their interaction, such as, selecting shelter, searching for food sources and friends, dispersing when danger is noticed and limiting reproduction when food is scarce. Key challenge in non–linear multi–modal optimisation solutions that use global techniques is the algorithm may converge on small regions or get trapped in local valleys – making it difficult for the algorithm to find the 'true' global minimum. Well established algorithms, like simulated annealing [14], require careful tuning to ensure the algorithm does not 'prematurely' converge on a local optimal while begin able to 'escape' any local modes/valleys.

1. Hunger Behaviour (migrates from its comfortable shelter and friends company to look for food)
2. Explore (Dispersion) behaviour (altruistic behaviours such as cooperative breeding, which restricts the dispersal of individual helpers potentially limit the slugs reproductive success)
3. Other Behaviour (e.g., Sleep, Random and Chase)

The 'three' states of the algorithm help resolve issues. The rational and reasoning as to why the states help with optimisation include:

- Hunger state – explores possible 'optimal' regions (quickly narrow down solutions)
- Explore state – enables individuals to 'escape' local minimums (i.e., move away and explore the larger landscape)
- Other state – introduces chaotic element (e.g., random phenomenon). For example, the randomization increases the diversity of the solution during each iteration

The proposed SOA algorithm is outlined in Algorithm 2:

Algorithm 1: SOA Definitions.

```
Input/Definitions
Global
    Fitness function: $f(x)$ [0 to 1]
    Population size: $N$ [100 to 1000]
    Problem dimensions: $D$ [1 to 100]
    Crossover mutation: $CR$ [0 to 1]
    Sample size: $S$ [3 to 10]
    Epoch count: $t$ [0 to infinity]
Individuals:
    Individual index: $i$ [0 to N–1]
    Hunger counter: $H$ [1 to 10]
    Explore counter: $E$ [1 to 10]
```

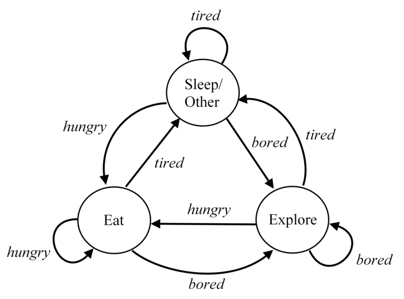

Figure 4: **States** – The behaviour pattern of our virtual slug is represented by three states.

The innovative SOA possesses the enhanced potential for handling both deep and explorative landscapes.

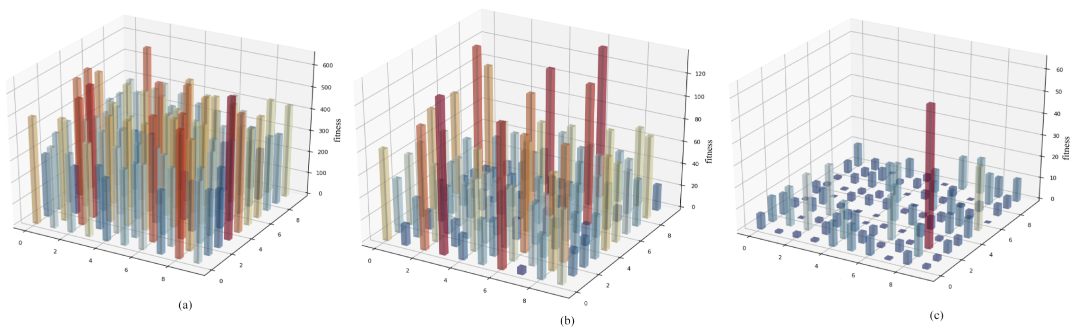

Figure 5: **Population Fitness** - Graphical visualisation of the population fitness over multiple generations (10,100,1000). (Population size = 100, Rastrigin function).

(1) Initialise slug population with uniform distributed random numbers and set all parameters with values (2) For each individual: (3) – Select number of samples from population (not same as each other or the current individual) (4) Perform chase–swarming (5) Perform hunger behaviour (6) Perform dispersion behaviour (7) Repeat the loop until stopping criterion is reached

Cultivation reduces slug numbers slugs pick the best time for a sex change New Scientist: Slugs are able to fine–tune their sex–change to ensure they make best use of the available resources (this also includes which slugs grow the quickest). Incorporate this into the algorithm – where more prosporous areas spawn more children/fertile parents.

Compare against 'random': uniform distribution focused test cases novel test cases

Algorithm: 1. Algorithm begins with a random population 2. Algorithm evolves incrementally (generations) 3. Logical structure for the motion (adaptation) of individuals: (reproduction, mutation, randomness) 4. If the new individual is better – replace the original ('stability') – if the individual is the same – don't modify (often overlooked)

This chapter has presented a novel algorithm based on the bio–inspired observations of how slugs adapt and solve every day problems. The preliminary results from the algorithm show promising possibilities

The following observations may be beneficial to realize why SOA is able to perform so well:

- The underlying behavioural pattern enable the solution to vastly lay emphasis on explo-

Algorithm 2: SOA Listing.

```
Initialize population
Initialize parameters
while not met criteria do [max iterations, fitness tolerance] do
    for every member of population [i to N–1] do
        Select $S$ mutually exclusive individuals from population
        Set $x_\alpha$ as the best slug from $S$
        Set $x_\beta$ as the worst slug from $S$
        Increment hunger counter $h$ for $x_i$
        Increment explore counter $e$ for $x_i$

        ▷ hunger counter triggered
        if ( $x_i[h] > H$ ) then
            $v = (x_\alpha – x_i)$
        end
        ▷ explore counter triggered
        else if ( $x_i[e] > E$ ) then
            $v = (x_\beta – x_i)$
        end
        ▷ otherwise random behaviour (e.g., sleeping/mutating)
        else
            $v$ = random individual
        end

        R = RandomInteger(0, D–1)

        for $d = 0$ to $D–1$ do
            if ( $d == R || RandomReal(0,1) < CR$ ) then
                $x_i(t+1)[d] = x_i(t)[d] + v[d]$
            end
        end

        if ( $x_i[h] > H$ ) then
            $x_i[h]$ = RandomInteger(0,H)
        end
        else if ( $x_i[e] > E$ ) then
            $x_i[e]$ = RandomInteger(0,E)
        end

        ▷ keep best candidate (i.e., new or original)
        if ( $f(x_i(t))$ ) < $f(x_i(t+1))$ ) then
            $x_i(t+1) = x_i(t)$
        end
    end
    Set $t = t + 1$
end
```

ration in the initial steps and exploitation on last iterations
- SOA preserves the key superiorities of population diversity of multiple generations
- When SOA converges to a local optimal, the exploratory behaviour helps it to jump out of these trenches and travel in the direction of new improved locations
- The chaotic element of the algorithm helps increase the diversity of individuals over the duration of the iterative evolution of the population

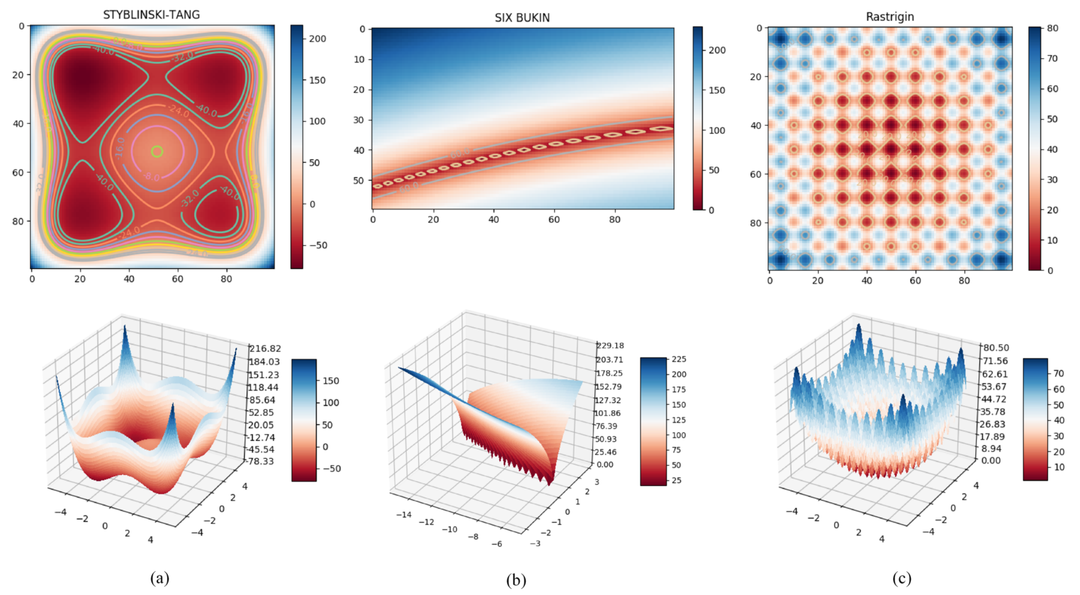

Figure 6: **Optimisation Function** - Optimisation functions shown here in their two–dimensional form to illustrate the challenges (e.g., multiple local minimums and non–linear landscape). Note, functions, such as, the Rastrigin and Styblinski–Tang function allow multiple dimensions so we are able to perform more extreme simulations.

4 Massively Parallel Architectures

The SOA is able to be ported to massively parallel architectures (such as, the Graphical Processing Unit (GPU)). This enables the algorithm to take full advantage of the systems resources and reduce computational times. The algorithm is highly suited to parallel architectures due to the nature of the algorithm (i.e., population based whereby each individual performs the same calculation on different data). The GPU is highly suited to 'single instruction multiple data' (SIMD) operations. To avoid 'bottlenecks' or 'stalls' when updating the population (e.g., reading/writing to the same location) – we double buffer the population for parallel implementation. This means the entire population is able to be updated in parallel without any locking dependencies (i.e., read data from the old population memory location and write the new population to a new memory location). After calculating the new population, we simply 'flip' the memory locations, so the current population becomes the reference data for the next population.

Common challenges when implementing a parallel solution (on the GPU), include:

- random number generation
- complexity of the algorithm (avoid conditional logic due to the SIMD architecture)
- memory access issues (cache/local/global memory)
- 'bubbles' the slowest calculation will stall the entire population evolving

Importantly, the speed–up will depend on the optimisation problem. However, for a large number of problems, the GPU offers huge improvements. As shown in the literature [11, 2, 12] multiple optimisation problems using population based algorithms have successfully been ported to massively parallel architecture.

For example, previous evolutionary algorithms that have exploited massively parallel architectures, include: Wong et al. [23], who was one of the first researchers to develop a fine grained master–slave implementation on the GPU (evolutionary algorithm with fitness, mutation and reproduction on the GPU, while the CPU performed the selection and competition calculations). Later Wong et al. [22], implemented a version of the genetic evolutionary algorithm that run solely on the GPU (i.e,. all except the random number generation). Of course, with the introduction and adoption of CUDA and OpenCL a whole range of applications have been developed for the GPU. From deep learning algorithms, sparse matrix solvers, sorting algorithms, and physics–based simulators [20]. Our parallel implementation was written in Visual Studio (2015) using C++ and the CUDA 8.0 API. The hardware used to carry out the parallel simulations was a NVidia GeForce GT 750M. The GPU kernel would perform the epoch multiple times (i.e., would not need any interference or data from the CPU during the evolving of the population). The algorithm would continue to evolve the population until the some predefined criteria, such as, maximum iterations or the fitness solution was within some specified tolerance. Each thread updated an individual within the population. The updated individual is placed in the secondary memory location to avoid any deadlocks. Each individual is able to see all the population of the 'previous' generation (memory) to provide informed information for the individual in the 'next' generation (memory). Crucially, our algorithm is able to operate completely on the GPU (i.e., no–need to switch the population or data back and forth between the CPU and GPU for sorting/evaluating/randomizing). Similar to Arora et al. [1], we seed the initial population (individuals) with a random value (i.e., the standard 'C' programming language *rand()* function [9]). This uncomplicated number generator produces a random value based upon the seed and a maximum limit (Linear Congruential Generator (LCG)). While there are a number of built in custom CUDA RAND libraries [9], the LCG function enabled full control over sequential–parallel implementations/comparisons. Even if the random number is not cryptographically secure, it provides an efficient solution for the simple task of randomizing our evolving population values.

5 Experimental Results

We apply the optimisation algorithm to a number of popular global optimisation benchmark functions [10] (e.g., Rastrigin and Styblinski–Tang). Figure 6 shows a graphical illustration of the different search spaces for the 2–dimensional variations.

Rastrigin Function The Rastrigin function is a popular (non–linear multi–modal) non–convex function used as a performance test problem for optimization algorithms (see Figure 6(c)). Finding the minimum of this function is a fairly difficult problem due to its large search space and its large number of local minima (see Equation 2).

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right] \tag{2}$$

where $A = 10$ and $x_i \in [-5.12, 5.12]$. The global minimum is at $x = 0$ where $f(x) = 0$. The simulation results for the SOA for solving the Rastrigin function are shown in Figure 7.

Sixth Bukin Function The sixth Bukin function is a two dimensional optimisation problem with multiple local minima, all of which lie in a 'ridge' (see Equation 3).

$$f(x_1, x_2) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10| \tag{3}$$

The function is evaluated in the region: $x_1 \in [-15, 15], x_2 \in [-3, 3]$ and has a global minimum

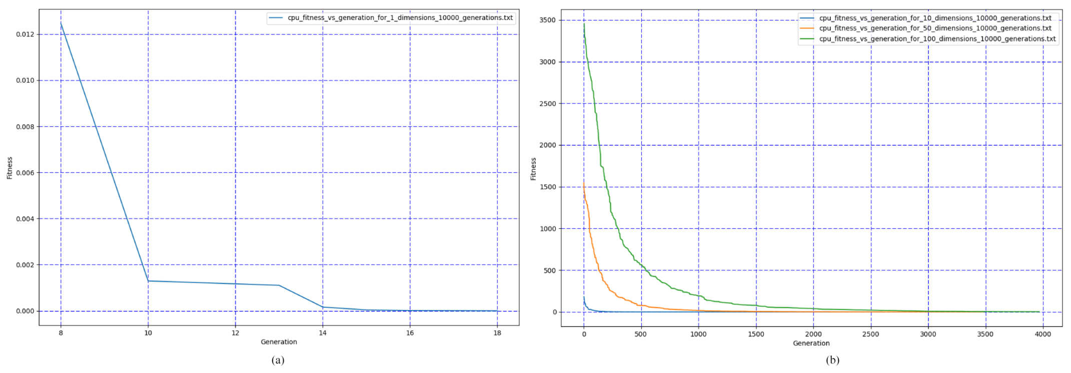

Figure 7: **Rastrigin Function -** SOA simulations for different situations: (a) single dimensional problem and (b) multiple dimensional simulations [10, 50 and 100]. (Population size = 100).

at $f(-10, 1) = 0$ (see Figure 6(b)). Simulation results for the Sixth Bukin Function shown in Figure 8.

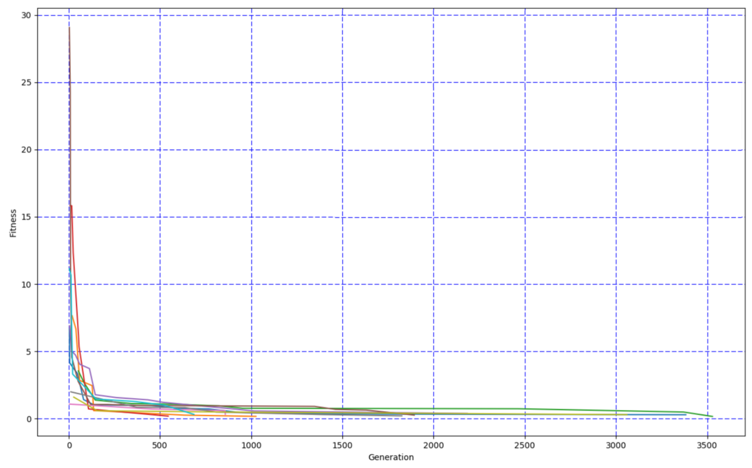

Figure 8: **Sixth Bukin Function -** SOA simulations (multiple–reruns). (Population size = 100).

Styblinski–Tang Function The Styblinski–Tang function (shown in Equation 4) is a multi–dimensional optimisation problem (see Figure 6(a)).

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n}\left(x_i^4 - 16x_i^2 + 5x_i\right) \tag{4}$$

The function is evaluated in the range $x_i \in [-5, 5]$ for $i = 1..n$. The global minimum for the function is: $f(-2.903534, ..., -2.903534) = -39.16599n$. Simulation results using the SOA for the Styblinski–Tang function are shown in Figure 9.

Limitations (Advantages and Disadvantages) The proposed chapter's SOA has a probabilistic algorithm at its heart. The SOA is an efficient optimisation tool for solving a variety of non–linear problems. However, a crucial factor to the success or failure of the algorithm is the fitness function. For instance, given an appropriate fitness function, the SOA described in this chapter would be able to find a desired solution within only small number of

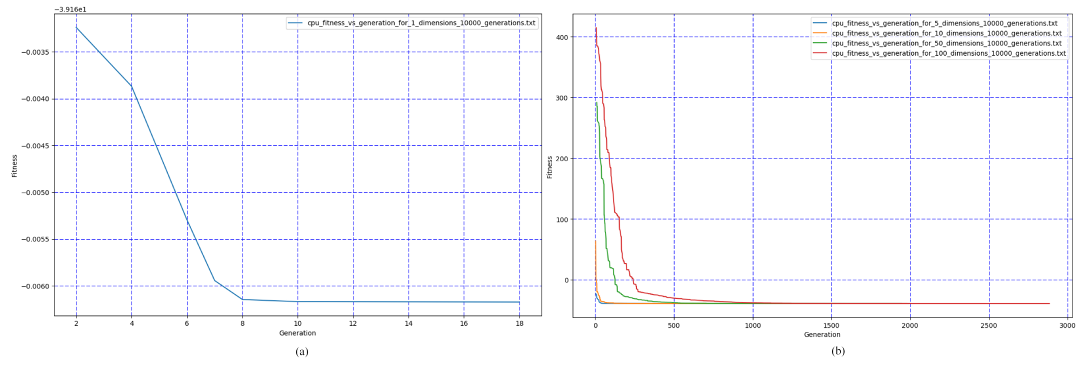

Figure 9: **Styblinski–Tang Function** - SOA simulations for different situations: (a) single dimensional problem and (b) multiple dimensional simulations [5, 10, 50 and 100]. (Population size = 100). Note – since the fitness function is a multiple of the dimensions (i.e., –39.16599n) – we divide the fitness value by n to enable us to plot the different simulation results side–by–side.

iterations. Yet, a major problem lies in the definition and the combination of multiple–fitness functions. Since tuning the algorithm parameters in addition to selecting suitable weighting coefficients might not be intuitive. We highlight the facts in respect to finding a solution based upon an evolving algorithm (i.e., genetic algorithm optimisation as proposed in this chapter):

Advantages:

- handle multiple local optima
- handle non–smooth highly non–linear objective functions
- able to work with a large numbers of parameters
- handle noisy (or stochastic) objective functions
- algorithm is straightforward to understand and implement
- modular in that the optimisation can be separated from the problem
- always get an answer (a best guest solution when non–exists) and will improve over time
- the underlying algorithm is inherently parallelizable (distributed across multiple cores/systems) – exploit processing power of parallel architectures (scalable solution)
- the SOA uses probabilistic selection rules (not deterministic ones)
- less likely to be trapped in a local optimal (compared to traditional gradient based methods)
- importantly, is able to work with problems that have no analytical solution
- simple to represent real–numbers (e.g., compared to other techniques, like the Genetic Algorithm, which requires variables to be encoded/converted to a chromosome representation)

Disadvantages:

- no guarantee of finding a global maxima
- time taken to converge (unpredictable)
- parameter tuning (unintuitive on what the parameters should be, such as, for hunger/explore values)

6 Discussion/Future Work

While the algorithm has demonstrated the ability to efficiently find an optimal solution for a variety of situations – the algorithm still has room for improvement based upon the original bio–inspired concept. Different environments/habitats, influence the population and individuals characteristics (urban, desert, forest or a farmers field). For instance, the urban habitats are areas dominated by human activities and human constructions. These include towns, cities and associated landscapes, such as landfill sites. It can almost be described as a patchwork of other habitats where buildings are artificial cliffs, sewers and drains are waterways, and parks, gardens and brownfield sites provide forests and meadows. Animals which have adapted to the urban environment are tolerant of the light and noise generated by human activity, and take advantage of the heat and the abundant food sources.

- Species of slug
- Fitness functions
- Multi–objective situations
- Shock propagation (enhance technique)
- Population catastrophe (regularly inject diversity)
- Hybrid possibilities

The SOA possesses an underlying chaotic behaviour, which under the right conditions, gives the algorithm and enhanced search efficiency. The ability to search local regions in addition to tunnel (or jump) through local traps helps make the algorithm more versatile and improve the probability of finding the global optimal

7 Conclusion

This chapter has introduced an innovative Gastropod Mollusc (or Slug) Optimisation Algorithm (SOA) for solving non–linear multi–modal problems. The global search capacity of the SOA is boosted by its nature inspired behavioural pattern and underlying chaotic randomness. Experimental results revealed the exploration capacity of SOA for solving challenging problems (including multi–model/multi–variable tasks). In summary, the SOA is an efficient and powerful tool for addressing highly non–linear discontinuous problems.

Acknowledgements

We gratefully acknowledge the support of the NVIDIA Corporation for the donation of hardware used in this research. In addition, we want to thank the reviewers for taking the time to provide insightful and valuable comments to help to improve the quality of this chapter.

References

[1] Ramnik Arora, Rupesh Tulshyan, and Kalyanmoy Deb. Parallelization of binary and real–coded genetic algorithms on gpu using cuda. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1—8. IEEE, 2010. 9

[2] Alexander Choong, Rami Beidas, and Jianwen Zhu. Parallelizing simulated annealing–based placement using gpgpu. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 31—34. IEEE, 2010. 8

[3] Pierre Collet and Jean–Philippe Rennard. Stochastic optimization algorithms. *arXiv preprint arXiv:0704.3780*, 2007. 2

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga–ii. *IEEE transactions on evolutionary computation*, 6(2):182—197, 2002. 2

[5] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28—39, 2006. 2

[6] Randy H Ewoldt, Anette E Hosoi, and Gareth H McKinley. Nonlinear viscoelastic biomaterials: meaningful characterization and engineering inspiration. *Integrative and comparative biology*, 49(1):40—50, 2009. 3

[7] Jörg Fliege and Benar Fux Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479—494, 2000. 1

[8] Alan Gelperin. Olfactory basis of homing behavior in the giant garden slug, limax maximus. *Proceedings of the National Academy of Sciences*, 71(3):966—970, 1974. 2

[9] Lee Howes and David Thomas. Efficient random number generation and application using cuda. *GPU gems*, 3:805—830, 2007. 9

[10] Momin Jamil and Xin–She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150—194, 2013. 9

[11] Ben Kenwright. Inverse kinematic solutions for articulated characters using massively parallel architectures and differential evolutionary algorithms. In *VRIPHYS2017: 13th Workshop on Virtual Reality Interaction and Physical Simulation*. Springer, 2017. 1, 2, 8

[12] Ben Kenwright. Optimisation of articulated holonomic structures using a cuda–based differential algorithm. In *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 2017. 8

[13] Benjamin Kenwright. Planar character animation using genetic algorithms and gpu parallel computing. *Entertainment Computing*, 5(4):285—294, 2014. 1

[14] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671—680, 1983. 5

[15] Rhydian Lewis. A survey of metaheuristic–based techniques for university timetabling problems. *OR spectrum*, 30(1):167—190, 2008. 2

[16] R Menzel, Rapporteur G Bicker M Lindauer, TJ Carew HS Markl, K–F Fischbach WG Quinn, JL Gould CL Sahley, B Heinrich AR Wagner, and MA Heisenberg. Biology of invertebrate learning. In *The biology of learning: report of the Dahlem Workshop on the Biology of Learning, Berlin, 1983, October 23–28*, volume 29, page 249. Springer, 1984. 2

[17] Melanie Mitchell, John H Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing? *Working paper, Santa Fe Institute, Santa Fe, New Mexico*, 1993. 1

[18] Ibrahim H Osman and James P Kelly. Meta–heuristics: an overview. In *Meta–heuristics*, pages 1—21. Springer, 1996. 3

[19] Konstantinos E Parsopoulos. *Particle swarm optimization and intelligence: advances and applications: advances and applications*. IGI global, 2010. 1

[20] Shane Ryoo, Christopher I Rodrigues, Sara S Baghsorkhi, Sam S Stone, David B Kirk, and Wen–mei W Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73—82. ACM, 2008. 9

[21] Yuhui Shi et al. Particle swarm optimization: developments, applications and resources. In *evolutionary computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 81—86. IEEE, 2001. 2

[22] Man–Leung Wong and Tien–Tsin Wong. Parallel hybrid genetic algorithms on consumer–level graphics hardware. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2973—2980. IEEE, 2006. 9

[23] Man–Leung Wong, Tien–Tsin Wong, and Ka–Ling Fok. Parallel evolutionary algorithms on graphics processing unit. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2286—2293. IEEE, 2005. 9

[24] Xin–She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169—178. Springer, 2009. 1, 2

[25] Xin–She Yang. *Nature–inspired metaheuristic algorithms*. Luniver press, 2010. 1, 2

[26] Xin–She Yang. Chaos–enhanced firefly algorithm with automatic parameter tuning. *Int J Swarm Intell Res*, 2(4):125—36, 2012. 3

[27] Deniz Yuret and Michael De La Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 208—212. Citeseer, 1993. 1