CUED - Engineering Tripos Part IIB 2023-2024 **Module Coursework**

Module	**4M17**	Title of report	**Assignment 2: Optimisation Algorithm Performance Comparison Study**

UNDERGRADUATE STUDENTS ONLY	
Candidate number:	

Assessment for this module is ☑ 100% / ☐ 25% coursework of which this assignment forms **50%**

MPHIL STUDENTS ONLY	
Candidate number:	

Date submitted:

OTHER POSTGRADUATE STUDENTS	
Name:	
CRSid:	

Feedback to the student

☐ **See also comments in the text**

		Very good	**Good**	Needs imprvmt
CONTENT	**Completeness, quantity of content:** Has the report covered all aspects of the assignment? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
PRESENTATION	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Marker: Date:

4M17 Coursework #2
Optimisation Algorithm Performance Comparison

Candidate No: 5730E

December 19, 2023

1 Introduction

This report conducts a comparative analysis of two optimisation algorithms applied to minimise Keane's Bump Function, (KBF). In particular, the study focuses on a Continuous Genetic Algorithm, (CGA), as well as an alternative algorithm not covered in the lectures: Parallel Tempering, (PT). The initial sections involve fine-tuning the algorithms on the two-dimensional KBF to enhance their performance and investigate their respective capabilities. Following this, the algorithms are deployed on the eight-dimensional KBF, and a comprehensive performance evaluation is conducted to discern differences and similarities between them.

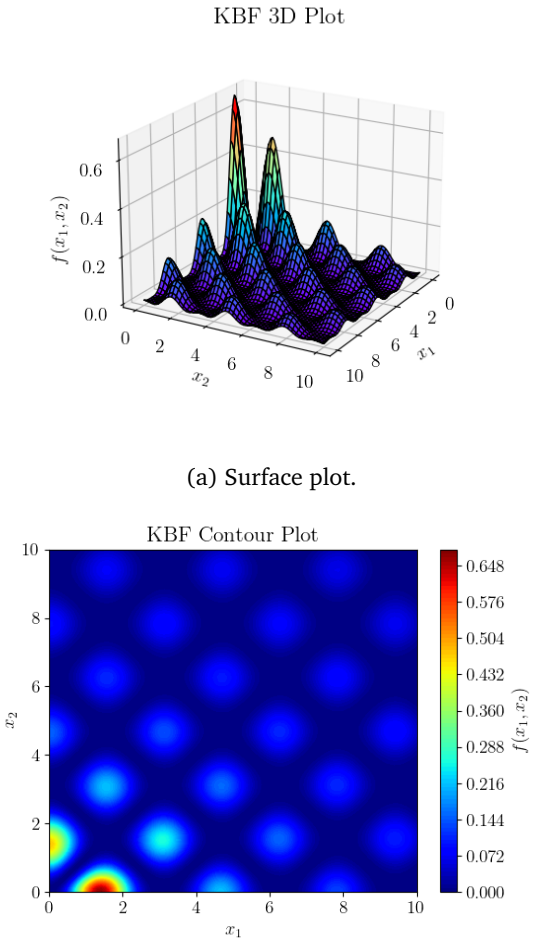
2 Keane's Bump Function

To compare the performances of the two algorithms, the Keane's Bump Function, (KBF), is used as the objective function. In particular, the n-dimensional constrained optimisation problem is defined as the maximisation of:

$$f(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} (\cos(x_i))^4 - 2\prod_{i=1}^{n} (\cos(x_i))^2}{\sqrt{\sum_{i=1}^{n} i \cdot x_i^2}} \right| \tag{1}$$

$$\begin{aligned} \text{subject to} \quad & 0 \leq x_i \leq 10 \quad \forall i \in \{1, \ldots, n\} \\ & \prod_{i=1}^{n} x_i > 0.75 \\ & \sum_{i=1}^{n} x_i < \frac{15n}{2} \end{aligned} \tag{2}$$

The two-dimensional form of the function has been plotted in Figure 1. Some notable properties are as follows:



(a) Surface plot.

(b) Contour plot.

Figure 1: Two-dimensional visualisation of the Keane's Bump Function, (KBF).

- The function is undefined at the origin, (0, 0). This is due to the division by zero in the denominator of Equation 1. Otherwise, the function is continuous and differentiable everywhere.
- The function is highly multi-modal. Its global maximum is located on the constaint boundary $x_n = 0$, where x_n denotes the final variable in the n-dimensional space. However, there are many

local maxima located inside the feasible region, all of which have quite similar amplitudes.

- The function is nearly symmetric about the line $x_1 = x_2$. This stems from its construction in 1, using the sums of squared, symmetric terms, x_i^2, $(\cos(x_i))^2$, and $(\cos(x_i))^4$. This results in some invariance regarding the order of the input variables. Overall, the peaks consistently manifest in pairs, yet there is a notable pattern wherein one peak always surpasses its counterpart in magnitude.

Given the above properties, the KBF is a challenging function to optimise. The presence of multiple, similar-amplitude local maxima makes it difficult for an optimisation algorithm to converge to the global maximum. On the other hand, all control variables share the same nature, (continuous variables), and exhibit identical scales. Additionally, all constraints are of the inequality type, and the feasible space is non-disjoint.

The problem becomes more complicated with the inclusion of the constraints outlined in 2. Figure 2 illustrates the resulting feasible region carved out of the original function space. Notably, the contstraint boundaries are non-linear. The problem complexity is additionally excaberated by the presence of multiple optima along the constraint boundaries, including the global maxima that we seek to identify.

These properties make the KBF a suitable candidate for the comparative analysis of the two optimisation algorithms, as discussed in the previous work of [3].

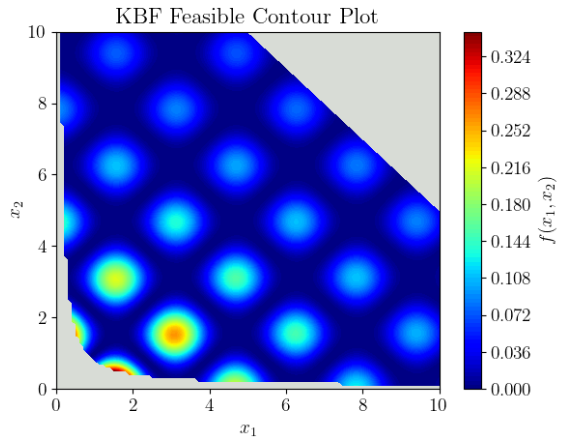


Figure 2: Feasible region carved out from the two-dimensional visualisation of the Keane's Bump Function, (KBF).

3 The Continuous Genetic Algorithm (CGA)

The discrete nature of the GA presented in [9] makes it unsuitable for the optimisation of the KBF. An implementation of a Continuous Genetic Algorithm, (CGA), is used instead, which lends itself better to the problems presented in 1-2.

The CGA, a technique inspired by natural selection and genetics, presents itself as particularly well-suited to tackling challenges associated with multiple local optima. Furthermore, the algorithm lends itself well to parallelisation with low implementation effort, and offers ample opportunities for modifications and adaptations, supported by a rich body of literature on the subject.

The primary difference between the CGA and the GA in [9] is the representation of individuals, (solutions of the state space), within the population. Rather than representing an individual as a vector of binary values or bits, (0s and 1s), the CGA uses a real-valued vector of floating-point numbers to represent each individual, as discussed in [4]. This allows for a direct representation of the problem, and eliminates the need for a decoding function, which reduces overhead in function evaluations.

This adjustment marks a significant departure from conventional GAs, aligning the algorithm more closely with Evolution Strategies (ES), another member of the evolutionary algorithms family presented in [12]. However, the algorithm presented in 3.1 is still classified as a GA in accordance with the differences presented in [5], given that mutation does not serve as the primary search mechanism for exploring the state space. Instead, it functions as a non-adaptive, background operator.

3.1 Implementation

In accordance with the terminology presented in [9], a vector solution of the state space will be referred to as an *individual* or *chromosome*. Correspondingly, a collection of such individuals arranged in a matrix format will be denoted as a *population*. Each individual is delineated as an $n \times 1$ vector of real-valued (floating-point) numbers, where n signifies the number of variables in the state space. The population itself is represented as a $m \times n$ matrix, where m designates the count of individuals in the population. This count is explicitly defined as a hyperparameter within the code, and is referred to as *POPULATION_SIZE*.

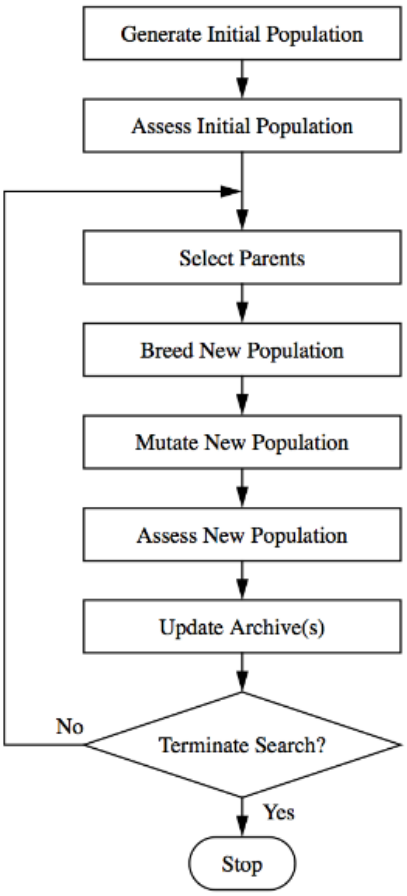


Figure 3: A flowchart depicting the CGA process, taken from [9].

The CGA process is outlined in Figure 3. Notably, three selection strategies and two mutation procedures were deliberately implemented to harness the flexibility inherent in the CGA, tailoring it to the optimisation challenges posed by the KBF. The subsequent section, 3.2, elucidates the selection method and mutation procedure choices that form the basis of the forthcoming comparison.

The CGA can be can be finely tuned for a specific objective function by modifying its fitness function, which assesses the quality of an individual. The management of constraints is woven into the selection process, as outlined later in Section 3.1.2.

3.1.1 Initialisation

The population is initialised with random values uniformly distributed within the range of 0 to 10, which represents the bounds of the state space, encompassing both feasible and infeasible values. This population comprises 250 individuals, as defined by the *POPULATION_SIZE* hyperparameter.

3.1.2 Parent Selection

Three selection methods were implemented: proportional selection, tournament selection, and stochastic remainder selection without replacement (SRS). The hyperparameter governing the quantity of parents chosen, denoted as *NUM_PARENTS*, is established at 25% of the population size, (rounded down to the nearest even number to facilitate the creation of parent pairs).

Another important aspect of the selection process is the handling of constraints. In particular, the selection process is repeated until only feasible individuals are selected. Infeasible individuals are simply not chosen as parents, a strategy advised by [9], which was deemed suitable considering that the feasible space is non-disjoint, and the constraints are of the inequality type.

Proportional Selection

The probability of an individual being selected for mating is proportional to its fitness:

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

Here, f_i denotes the fitness of the ith individual. This is the simplest selection method, and is implemented in accordance with the theory presented in [9].

As noted in [9], this approach is susceptible to high variance in individual selection, primarily because there is no assurance of choosing the optimal individual. Alternatively, the following two procedures show greater potential, incorporating a degree of determinism into the selection process.

However, it would be premature to disregard the proportional selection method. There is a chance that the determinism inherent in the other two methods could negatively impact the KBF optimisation process. A notable degree of stochasticity may improve the algorithm's effectiveness in exploring the search space, which may prove vital given the multi-modal nature of the KBF.

Tournament Selection

As outlined in [9], this strategy involves taking a small subset of the population, and selecting the top two individuals with the highest fitness. This is repeated until the required number of parents is achieved. Selection pressure can then be adjusted by varying the size of the subset, controlled by the *TOURNAMENT_SIZE* hyperparameter.

This is a popular selection method, as it is simple to implement, and is known to perform well in practice. It can be improved by including some of the concepts presented in [7], however, these have been avoided to

reduce the complexity of the algorithm's implementation.

Stochastic Remainder Selection without Replacement (SRS)

This strategy draws inspiration from [9]. Specifically, a group of chosen individuals is curated by generating an expected number of copies for each individual, denoted as:

$$E_i = N * P_i$$

Here, N represents the population size, and P_i is elucidated above in the context of proportional selection. The anticipated number of duplicates is subsequently divided into an integer part, $I_i = \lfloor E_i \rfloor$, and a remainder, $R_i = E_i - I_i$.

The integer part is used for deterministic selection of individuals. The ith indivdual is selected I_i times. Subsequently, the remained, R_i, is then used to stochastically augment the collection of individuals until the required number of parents is achieved. This is done by selecting the ith individual with a probability of R_i.

The discussion in [9] highlights that this approach appears to yield superior performance, ascribed to the inclusion of a degree of determinism in the selection criteria. Nevertheless, it remains worthwhile to evaluate the performance of the other two methods, as they might present distinct advantages within the framework of the KBF.

3.1.3 Mating Procedure

Similarly, two mating procedures have implemented: crossover and heuristic crossover. The entire population is replaced by offspring, bred from two randomly allocated parents from the pool of selected parents.

Crossover

The crossover procedure adheres to the principles detailed in [9]. Initially, a crossover point is randomly chosen. Genes from the first parent are incorporated into the offspring until this point, beyond which genes from the second parent take their place. Specifically, the sequencing of the parents is governed by the probability specified by the hyperparameter *CROSSOVER_PROB*.

Heuristic Crossover

Heuristic crossover is presented in [6]. By this variation, a random number β in the interval $[0, 1]$ is generated. The genes of the offspring are then determined as a blend of the original two parents, p_1 and p_2, as follows:

$$o_i = \beta(p_{1i} - p_{2i}) + p_{2i}$$

With this inspiration in mind, heuristic crossover was implemented in the CGA. Specifically, the sequence of parents in the above formula is determined by the *CROSSOVER_PROB* hyperparameter, aligning with the approach used previously in the original crossover procedure.

A crucial factor to bear in mind is that certain offspring may be produced outside the feasible region. This implementation deviates from the recommendation in [6] by not outright rejecting these offspring during the mating procedure. Rather, they are simply excluded from consideration as parents in the subsequent selection process.

The decision to incorporate this mutation procedure stems from the continuous nature of the state space. While the conventional crossover methods may be well-suited to binary representations, a more intuitive approach emerges when grappling with real-valued variables - using a blend of characteristics from both parents.

Moreover, the use of the heuristic crossover method aims to address previous limitations associated with standard crossover. Unlike conventional crossover, which confines offspring values to those of the parents, blending permits the generation of offspring beyond the parent values, allowing for potentially new information. This feature may prove particularly advantageous in the context of the KBF, enhancing the algorithm's capability to navigate the search space adeptly and thoroughly explore local optima.

An additional noteworthy observation is that the introduction of β serves to bring the CGA into closer alignment with Evolutionary Strategies (ES). This resemblance becomes evident as the formulation above bears some similarity to the intermediate recombination strategy outlined in [12]. Nevertheless, it is essential to emphasise, as mentioned earlier, that mutation does not serve as the primary search mechanism in the CGA. Consequently, the CGA can still be appropriately categorised as a GA, as per the classification seen in [5].

3.1.4 Mutation

The mutation procedure proposed by [4] involves perturbing a gene within a chromosome by a normally distributed random number with a mean of zero and a standard deviation of σ. However, the effectiveness of this procedure is constrained by the selection of σ as an additional hyperparameter, necessitating careful tuning.

Instead, a simpler approach was adopted, where genes within the population have an probability, given by the

mutation rate, of being reset to a random value drawn from a uniform distribution within the confines of the state space, mirroring the initialisation procedure.

3.1.5 Evaluation

The population is then evaluated, and the individuals are ranked in order of fitness. Here, the fitness is defined as the negative of the KBF cost function, outlined in 1. This is done to align the algorithm with the maximisation of the KBF.

3.1.6 Termination

For this section of the report, the algorithm terminates after a maximum number of iterations, (defined as a hyperparameter). A more stringent convergence criterion is adopted for the comparison between the two algorithms in Section 5. The individual with the lowest fitness value is then returned as the optimal solution to the optimisation problem.

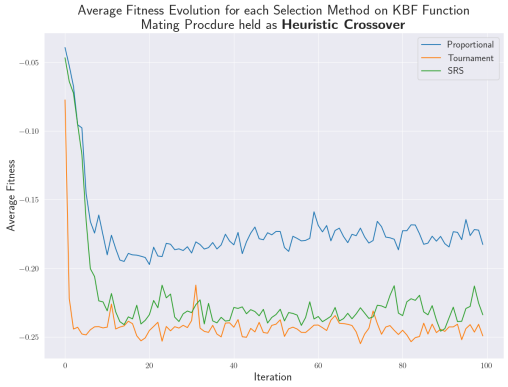
3.2 Tuning the CGA

Given the flexibility of the CGA, multiple selection methods and mutation procedures were implemented within the codebase, as discussed previously. These are then selected as hyperparameters within the script, to take advantage of the flexibility offered by the CGA.

To choose a selection method and mutation procedure going forwards into the main comparison of this report, code was used as a platform for experimented. Each selection method and mutation procedure was evaluated over 10 and 100 iterations, with a constant mutation rate of 0.05, and a crossover probability of 0.7. The number of parents was established at 25% of the population size, (rounded down to the nearest even number to facilitate the creation of parent pairs). The population size itself was set at 250, and the tournament size was similarly defined as 25% of this.

Supplementary results regarding this initial exploration of hyperparameter tuning are detailed in Section 6.1. Specifically, the final fitness values are outlined in Table 2. At a first glance, it would appear that all selection methods and mutation procedures perform similarly, offering convergences to similar fitness values. Minor variations could be attributed to the stochastic nature of the CGA, and the small number of iterations.

(a) Mutation Procedure: Crossover



(b) Mutation Procedure: Heuristic Crossover

Figure 4: Evolution of the average fitness values of the CGA population over 100 iterations for each of the selection methods. The results are presented for both mutation procedures: crossover and heuristic crossover. Here, the mutation rate was set to 0.05, and the crossover probability was set to 0.7.

However, more insightful conclusions can be drawn from the figures presented in 4, which illustrate the evolution of the average fitness values of the CGA population over 100 iterations for each of the selection methods. They illustrate that tournament selection seemed to outperform the other two selection methods when maximising the KBF, which is surprising given the favourable description in [9] regarding SRS.

The experiment was repeated several times, yielding results that were admittedly variable, attributed to the inherent stochastic nature of the CGA. At times, the superiority of SRS over tournament selection was evident, yet proportional selection generally did not prove to perform better. Despite this variability, the outcomes exhibited sufficient consistency to justify the designation of tournament selection as the primary method for the upcoming comparison.

The observed differences in performance may be attributed to the distinct characteristics of the KBF. Tournament selection appears to demonstrate greater efficacy in navigating the search space compared to SRS, perhaps due to its less deterministic nature. This becomes particularly significant in light of the multimodal nature of the KBF, where a certain level of stochasticity may prove essential for thorough exploration of the search space.

The results additionally indicate that when combined with tournament selection, the heuristic crossover procedure generally outperformed the standard crossover method. This is evident in the typically steeper decline in average fitness values, as illustrated in Fig. 4b. Furthermore, the heuristic crossover exhibited greater result consistency across repeated experiments and generally converged to higher maximas compared to the standard crossover. Although the observed differences were note always significant, the overall trend supports the choice of heuristic crossover as the preferred mating procedure for the forthcoming comparison.

This may be attributed to the heuristic crossover's capacity to generate offspring beyond the existing genotypes present within a generation. This attribute is especially beneficial within the framework of the KBF, enhancing the algorithm's ability to navigate the search space adeptly and systematically explore the numerous local optima.

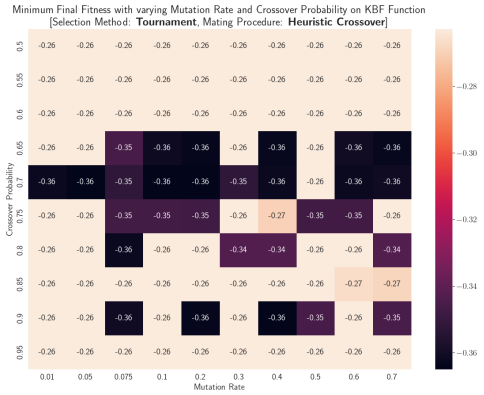
Having cemented the selection method and mutation procedue choices as tournament selection and heuristic crossover respectively, the CGA was further tuned to optimise its performance.

Specifically, choices for the mutation rate and crossover probability were explored. The results are presented in Figure 5, which showcases two heat maps of the final fitness values after 100 iterations for the CGA using the chosen procedures. The plots are presented for both the average final fitness, as well as the final fitness of the best individual.

Notbaly, as depicted in Fig. 5a, it is evident that an increased stochasticity of the CGA, indicated by higher crossover probabilities, leads to improved average final fitness. This aligns with expectations, as a broader and more randomly distributed population enhances the likelihood of individuals being situated in the proximity of local maxima. Consequently, a greater number of individuals within the population receive lower fitness values, a trend reflected in the observed average final fitness.

Fig. 5b offers deeper insights. It is evident that the crossover probability stands out as the most influential hyperparameter, while the fine-tuning of the mutation rate plays less of a role in shaping the performance of the CGA.

(a) Average final fitness across entire population.



(b) Final (minimum) fitness of best individual.

Figure 5: Heat maps of the final fitness values after 100 iterations for the CGA using tournament selection and heuristic crossover. Here, the mutation rates and crossover probabilities were varied to assess their impact on performance.

Overall, a clear optimal set of hyperparameters emerges. Moving forward, the mutation rate is fixed at 0.1, and the crossover probability is set to 0.65. These values have proven to yield a consistently optimal performance for the algorithm, comfortably lying within the darker regions of the heat maps.

To reiterate and summarise this section of the report, the CGA is now constructed as follows:

- **Selection Method:** Tournament Selection
- **Mutation Procedure:** Heuristic Crossover

- **Mutation Rate:** 0.1
- **Crossover Probability:** 0.65
- **Tournament Size:** 62
- **Number of Parents:** 62
- **Population Size:** 250

Fig. 6 depicts the evolution of the fitness values over 100 iterations using the optimal hyperparameters delineated above. Note that the (minimum) fitness attributed to the best individual remains constant, because an individual within the population is stochastically initialised near the local maxima that the algorithm converges to. This is less likely to occur in the 8-dimensional comparison, where the minimum fitness is expected to evolve more noticeably, given that the search space is significantly larger, and initialisation will be handled with more care.

Fig. 7 depicts the convergence of the population to the second largest maxima over 4 and 80 iterations using the optimally-tuned CGA. Note that the horizontally and vertically distributed outliers are attributed to crossover. Regrettably, global maximum convergence remained elusive even after 100 iterations. However, there does seem to be a discernible trend of the algorithm converging towards the second-largest maxima in Fig. 7, offering a promising indication of progress.

The primary challenge lies in the fact that both the first and second largest maxima are situated along the non-linear constraint boundary of the optimisation problem, posing a difficult navigational challenge.

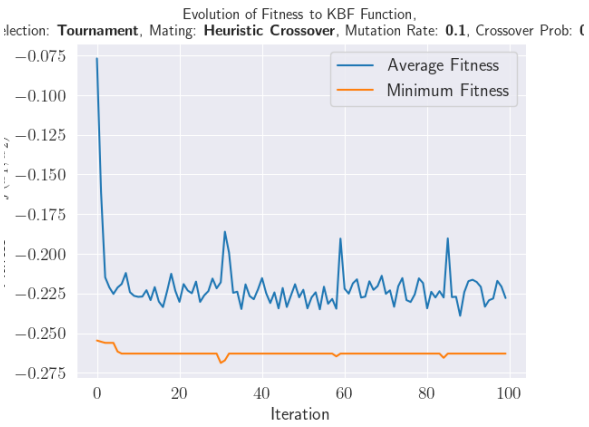
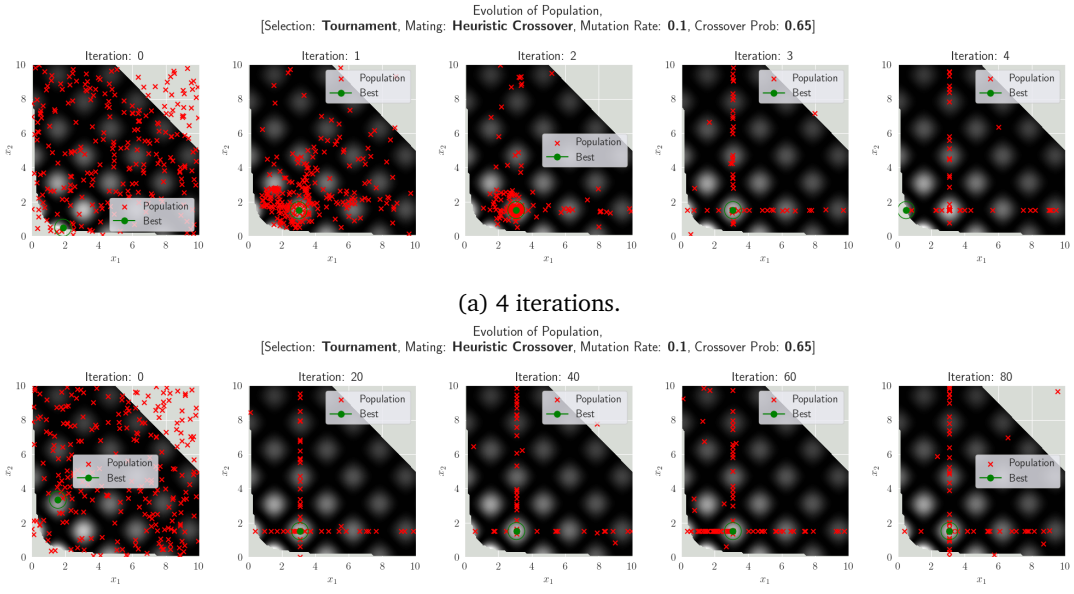


Figure 6: Evolution of the fitness values of the CGA population over 100 iterations for optimally-tuned CGA.



(a) 4 iterations.

(b) 80 iterations.

Figure 7: Evolution of the CGA population over 4 and 80 iterations using the optimal hyperparameters outlined at the end of Section 3.2. The second plot depicts the algorithm converging to a local maxima as a result of an individual being initialised near it.

4 Parallel Tempering (PT)

Having obsererved the CGA's performance and its difficulty in identifying global maxima effectively, an informed decision was made to introduce Parallel Tempering (PT) as the second algorithm for comparison.

This is a Markvo Chain Monte Carlo (MCMC) optimisation approach presented in [2]. Similar to simulated annealing, (SA), it revolves around the idea of smoothening the search space by introducing a temperature parameter, t.

By incorporating a Metropolis-Hasting acceptance probability that is proportional to $\exp(1/t)$, exploration of the solution space is encouraged at high temperatures, while exploitation of the local nature of the function is facilitated at low temperatures. At high temperatures, the acceptance distribution is is smoothened out by this proportionalilty, and the algorithm is then able to explore the search space more thoroughly. As the temperature diminishes, the acceptance distribution becomes more dependent on the fitness of the proposed solution to the KBF, enabling the algorithm to explore the local nature of the function.

However, PT incorporates the novel concept of Replica Exchange Monte Carlo. Unlike SA, which operates along a single tragectory, starting with a high temperature that gradually decreases, PT maintains multiple replicas of the system at different temperatures simultaneously. Each replica explores its own solution space, and exchanges of the solutions can occur between replicas at adjacent temperature levels. This exchange mechanism enhances global exploration by facilitating the movement of solutions between replicas operating at high and low temperature.

This parallel exploration across temperatures improves the algorithm's ability to escape local optima, providing a valuable alternative to traditional single-trajectory methods like SA. Consequently, PT could emerge as a well-suited approach for maximising the highly multi-modal KBF.

Unfortunately, this introduces an elevated computational cost, due to the management of multiple replicas and the periodic exchange of solutions. However, the potential advantages in terms of global exploration of the KBF outweigh the added computational expense. This is especially true, considering the highly parallelisable nature of PT, which may allow for even more efficient utilisation of parallel computing resources than the CGA.

4.1 Implementation

The discourse presented in [2] emphasises a delicate trade-off between optimising MCMC sampling outcomes and minimising computational efforts. Hence, the PT implementation presented here is deliberately crafted for flexibility, offering the ability to fine-tune the algorithm, as shown in Section 4.2.

As per the guidance in [10], constraints are handled by simply rejecting any proposed changes that violate the constraints. This is accomplished within the Metropolis-Hastings criterion regarding the acceptance or rejection of a proposed change. This was determined as a suitable approach, given that the feasible space is non-disjoint, and the constraints are of the inequality type.

The approach can additionally be characterised as population-based, incorporating 10 replicas in accordance with the *NUM_REPLICAS* hyperparameter. Within each replica, there are 25 chains of solutions determined by the *NUM_CHAINS* hyperparameter. Consequently, the total number of solutions being evaluated at any given time is 250, which is equivalent to the population size of the CGA.

4.1.1 Initialisation

The chains are initialised within the range of 0 to 1, as advised by the guidance provided in [11], specifically under the SA-derived solution generation framework presented in Section 4.1.4.

This range persists throughout the algorithm, until function evaluations are necessitated, at which point the *scale_up* lambda function is called to return the solution in the original state space.

Mirroring the CGA implementation, solutions were initialised without consideration for their feasibility. Whilst initialising within the feasible solution space was observed to enhance performance, this approach was deliberately omitted to ensure fair comparisons between the PT and CGA. However, the performance was notably compromised, given that solutions initialised deep within the infeasible zones of the search space faced challenges in escaping under the Metropolis-Hastings acceptance criterion.

4.1.2 Temperature Scheduling

A significant source of flexibility in PT lies in setting and adapting the temperatures of the replicas, as discussed in [2]. Drawing insights from the previous work of [1] on an unrelated, yet similar problem, this was taken advantage of by parameterising the temperature

schedule between 0 and 1, as follows:

$$T_i = \left(\frac{i}{N_{\text{replicas}}}\right)^p \quad \forall i \in \{1, \ldots, N_{\text{replicas}}\} \tag{3}$$

Here, T_i denotes the temperature of the ith replica, and N_{replicas} is the total number of replicas. The exponent p is a hyperparameter referred to as *POWER_TERM* that can be tuned to optimise the performance of the algorithm.

Fig. 8 illustrates the influence of parameter p on the temperature schedule. It intuitively demonstrates how p affects the balance between exploration and exploitation. A higher value of p corresponds to a slower and more gradual increase in temperature, promoting a conducive environment for exploration. A smaller value of p results in a more rapid increase in temperature, encouraging exploitation of the local nature of the function.

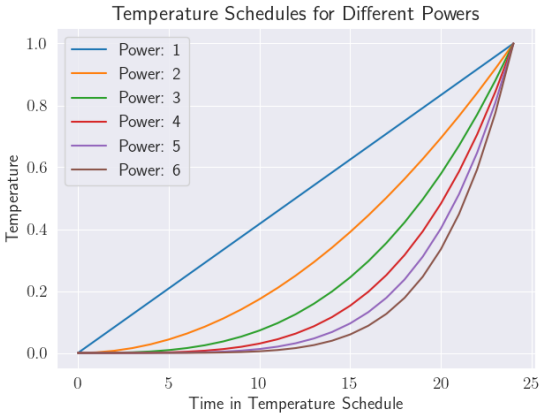


Figure 8: Temperature schedule 3 for the PT algorithm for different values of the *POWER_TERM* hyperparameter.

Tuning this balance is crucial for the algorithm's performance against the KBF, and is conducted in Section 4.2. While alternative approaches, such as geometric progression [2] are recommended in most literature, the above formulation was made to enhance control over the replica temperatures.

4.1.3 Replica Exchange

The most popular approach to replica exchange, as outlined in [2], involves periodic execution. The interchange of solutions between replicas at adjacent temperature levels occurs at regular intervals. In this implementation, periodic exchange is guided by the hyperparameter *EXCHANGE_PARAM*. This hyperparameter assumes values in the range of 0 to 1, representing the proportion of total iterations that must pass before another swap takes place. A value of 0.05 indicates that a swap occurs every 5 iterations when the maximum number of iterations is set to 100.

However, a second approach has also been implemented, which will be referred to as stochastic exchange. In this configuration, the possibility of replica exchange exists in every iteration, and the hyperparameter *EXCHANGE_PARAM* now functions as the probability governing the occurrence of this swap.

Both have been introduced to enhance the flexibilty of the algorithm, allowing for either a more deterministic or stochastic approach to replica exchange. The optimal choice is determined in Section 4.2.

Upon meeting the conditions for a swap, the replicas are sequentially traversed in ascending order of temperature. Each replica undergoes an attempt to exchange all of its solutions with the subsequent replica in the sequence. The likelihood of accepting such a swap is dictated by the adapted Metropolis-Hastings acceptance criterion, as detailed in Section 4.1.5.

4.1.4 Metropolis Update

The inspiration for proposing a new solution stems from [11], which introduces a routine that leverages accumulated experience from prior iterations to generate SA update steps. Given the similarity between SA and PT, this approach was deemed suitable for the PT implementation.

In particular, a new solution is generated by:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \mathbf{D}\mathbf{u} \tag{4}$$

In this context, $\mathbf{u}$ represents a vector of uniformly distributed random numbers within the range of $[-1, 1]$, while $\mathbf{D}$ is a diagonal matrix specifying the maximum permissible step size in each dimension of the state space. In this PT implementation, $\mathbf{D}$ is uniquely defined for each solution across all replicas. Following the acceptance of a new solution, the matrix $\mathbf{D}$ is updated using the formula:

$$\mathbf{D}_{\text{new}} = (1 - \alpha)\mathbf{D}_{\text{old}} + \alpha\omega\mathbf{R}$$

Here, the constants α and ω are set at 0.1 and 2.1 respectively, in accordance with the guidlines presented in [11]. The matrix $\mathbf{R}$ is a diagonal matrix whose elements represent the magnitudes of the successful changes made to each dimension within a specific solution.

4.1.5 Metropolis-Hastings Acceptance Criterion

The probability of accepting a proposed solution is determined by the Metropolis-Hastings acceptance criterion. This has been specifically formulated to work alongside the "adapting" update step, 4, with additional insights drawn from the works of [11] and [8]. The probability of accepting a new solution generated by Eq. 4 is given by:

$$P_{\text{accept}} = \min\left[1, \exp\left(\frac{f(\mathbf{x}_{\text{new}}) - f(\mathbf{x}_{\text{old}})}{k \cdot T_i \cdot \tilde{d}}\right)\right]$$

In the above formulation, the negation of the objective function: $-f(\mathbf{x})$, represents the fitness of a solution $\mathbf{x}$, k is the Boltzmann constant, T_i is the temperature of the $i$tj replica, and $\tilde{d}$ is euclidean normed distance between the old and new solutions.

However, when judging the swap of two solutions through a replica exchange, the acceptance probability is instead adapted to:

$$P_{\text{accept}} = \min\left[1, \exp\left(\frac{f(\mathbf{x}_{\text{new}}) - f(\mathbf{x}_{\text{old}})}{k\tilde{d}}\left(\frac{1}{T_{\text{old}}} - \frac{1}{T_{\text{new}}}\right)\right)\right]$$

in accordance with [8]. This imposes a harsher standard for the acceptance of a swap between replicas operating at vastly different temperatures, which is a desirable feature, especially when the power term, p, is set to a high value.

4.1.6 Termination

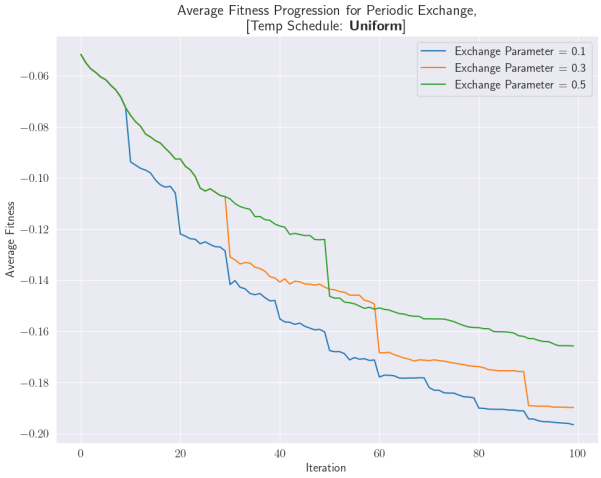
Again, the algorithm terminates after a maximum number of iterations, (defined as a hyperparameter), in this section of the study. A more stringent convergence criterion is adopted for the comparison between the two algorithms in Section 5. The solution with the lowest fitness value across all replicas is then returned as the optimal solution to the optimisation problem.

The total number of iterations is fixed at 100, which generally proved insufficient for the algorithm to achieve complete convergence. Rather than emphasising complete convergence, the assessment centers around tracking the evolution of average fitness values. This approach provides insights into the algorithm's effectiveness and speed, gauging its progress within the limited span of 100 iterations.

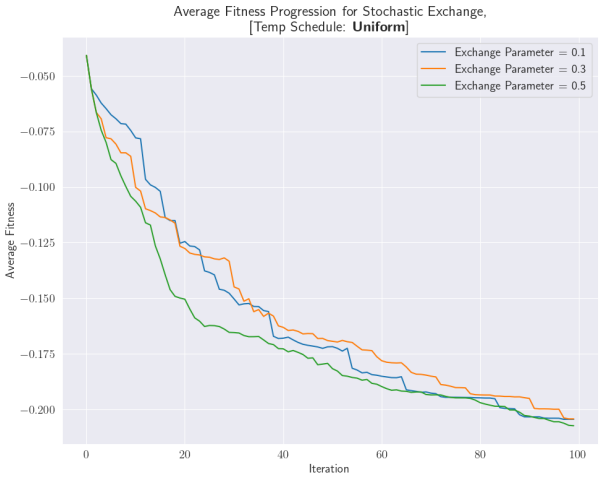
4.2 Tuning the PT

As with Section 3.2, the flexibility offered by PT was first explored using the parallelised experiments presented in code

Unlike the CGA table (Table 2), the preliminary exploratory findings presented for PT (Table 3) imply a subtle correlation between the power term of the temperature schedule and the algorithm's performance. Final fitness values exhibit relative constancy, hinting at no potential connection. However, the variability in algorithm performance appears more pronounced when considering the approach to replica exchange.



(a) Periodic Exchange



(b) Stochastic Exchange

Figure 9: Evolution of average fitness values of the PT solutions over 100 iterations for the two approaches to replica exchange. Temperature scheduling was kept uniform.

Fig. 9 instead illustrates the evolution of the average fitness values across all 100 iterations for both exchange procedures. The result present a clear distinction between the two approaches that remains consistent with repitition.

The periodic exchange approach seems to exhibit a higher dependency on the value assigned to the ex-

change parameter. In contrast, the stochastic exchange approach appears to follow a consistent descent, irrespective of the exchange parameter value. This observation is unexpected, considering that the exchange parameter is anticipated to exert a similarly significant influence on both approaches. In both cases, the exchange parameter affects the frequency of swaps, which is expected to impact the algorithm's ability to escape local optima.

Additionally, the descent depicted in Fig. 9a seems steeper when the exchange parameter assumes smaller values. This aligns with expectations, as a smaller exchange parameter corresponds to a higher frequency of swaps in the periodic case. Here, the exchange parameter is interpreted as the proportion of total iterations that transpire before another swap takes place. When applied to the KBF, PT demonstrates greater efficacy when the algorithm is permitted to exchange solutions more frequently. This implies that increased exploration of the search space is well-suited to optimising the KBF.

Figures 11 and 12 present a final analysis of these hyperparameters. However, the stochastic heatmap, 12b, did not exhibit any consistency or superiority over the periodic heatmap, 11b, leading to no further consideration of stochastic exchange, given its suboptimal and unpredictable behavior.

In contrast, a consistent pattern unveiled itself in Fig. 11b, which persisted across repetitions. The PT algorithm, when applied to the KBF, consistently demonstrated a inclination towards a lower exchange parameter paired with a low power term. This may be rationalised by the fact that a lower exchange parameter corresponds to a heightened frequency of swaps, facilitating dialogue between the replicas and encouraging more exploration. Conversely, a low power term fosters a delicate equilibrium between exploration and exploitation, as evident in Fig. 8. The preference for uniform temperature scheduling likely stems from its enhanced capability to focus on local changes, after arriving at a particular optimum. Any exploratory capabilities stem from replica exchange, after which the algorithm can then focus on exploiting the local nature of the function.

Overall, these results imply that optimal settings for the PT algorithm on the KBF involve a power term of 1 and an exchange parameter of 0.01. These values are associated with a uniform temperature schedule and replica exchange taking place at every iteration. They position the PT algorithm favorably within the lighter regions of Fig. 11b.

In summary and to reiterate, the PT algorithm is now constructed as follows:

- **Number of Replicas:** 10
- **Number of Chains:** 25
- **Power Term:** 1
- **Exchange Procedure:** Always
- **Exchange Parameter:** N/A

Fig. 10 depicts the evolution of the fitness values over 100 iterations using the optimal hyperparameters delineated above. The fitness of the best solution drops noticeably once better solutions are identified and arrived at through the MCMC updating process.

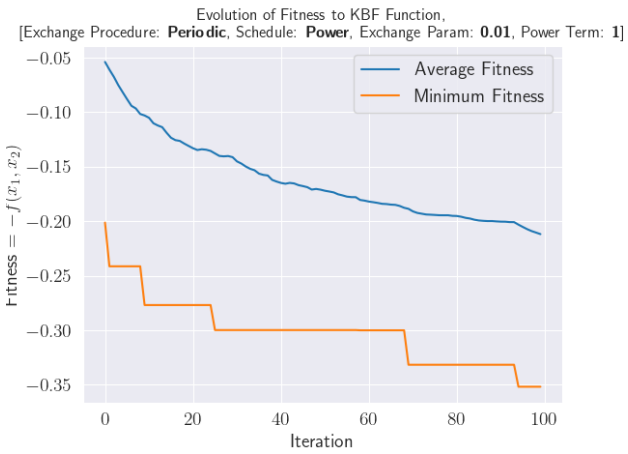


Figure 10: Evolution of the fitness values of the optimally-tuned PT solutions over 100 iterations.

The top row in Fig. 13 depicts the iterative evolution of the solutions. Notably, all solutions consistently converge towards the prominent maxima, and the global maximum is successfully identified in every repetition. The PT algorithm has demonstrated superior capabilities in exploration compared to the CGA.

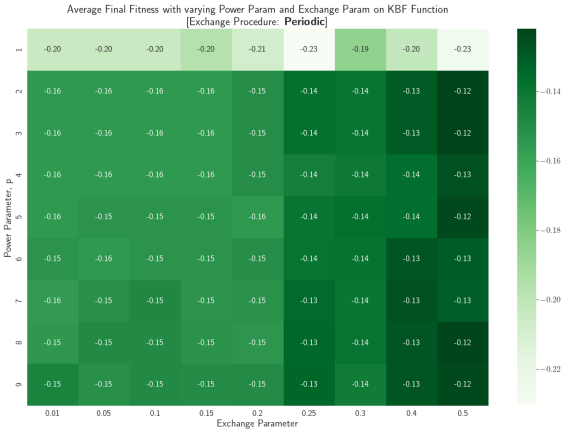
The optimal solution is denoted by a green circle. For a more insightful comprehension of the Markov Chain Monte Carlo (MCMC) updating process, a particular solution within the final replica is emphasised in yellow. This solution is monitored throughout all iterations and it seems to become ensnared in a local optimum during the earlier iterations. However, it then successfully breaks free through the exchange mechanism and begines to converge towards a larger maximum in subsequent iterations.

The bottom row in Fig. 13 showcases the final solutions of each replica, organised by temperature and superimposed over the temperature-smoothed KBF con-

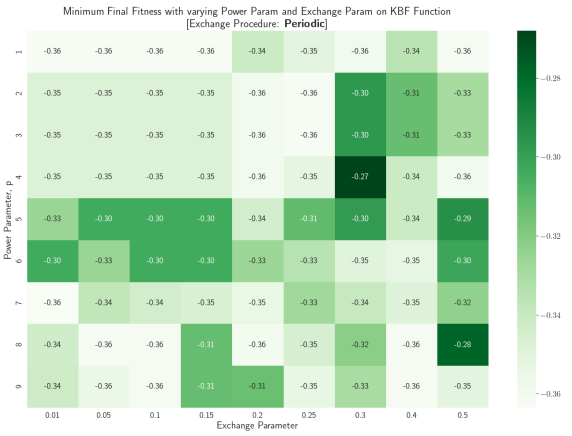
tours. The arrangement follows an ascending temperature order, with the replica at the lowest temperature positioned on the left. These results shed light on the fundamental dynamics of the algorithm.

At lower temperatures, all solutions exhibit a clear inclination toward the higher, global maxima of the KBF. This preference arises from ample exploration of the search space. In contrast, solutions at higher temperatures display a more dispersed distribution, converging around local optima.

Additionally, although these higher temperatures encourage localised exploration, the replica exchange mechanism facilitates the movement of solutions within the final temperature towards the larger maximas. Subsequent iterations are anticipated to drive convergence of solutions at higher temperatures towards the global maxima as well.

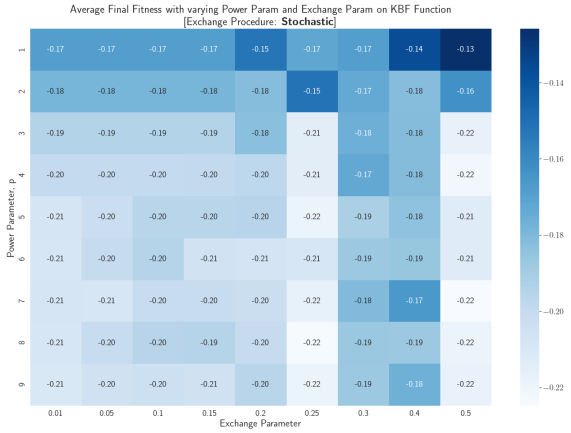


(a) Average Fitness Values

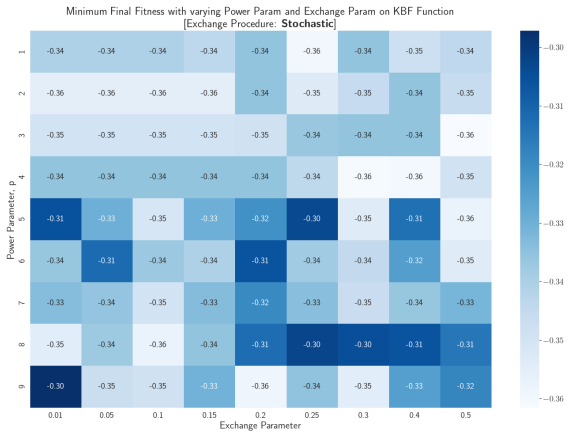


(b) Minimum Fitness Values

Figure 11: Heatmaps of the final fitness values after 100 iterations for the PT algorithm using periodic exchange. Here, the power term and exchange parameter were varied to assess their impact on performance. Despite the somewhat varied outcomes between repitition, a consistent trend emerges - the PT algorithm demonstrates a preference for a lower exchange parameter coupled with a moderate power term.



(a) Average Fitness Values



(b) Minimum Fitness Values

Figure 12: Heatmaps of the final fitness values after 100 iterations for the PT algorithm using stochastic exchange. Here, the power term and exchange parameter were varied to assess their impact on performance. The values for the best (minimum) final fitness varied greatly upon repitition. Generally, performance was outmatched by periodic exchange, which was also more consistent and controllable.

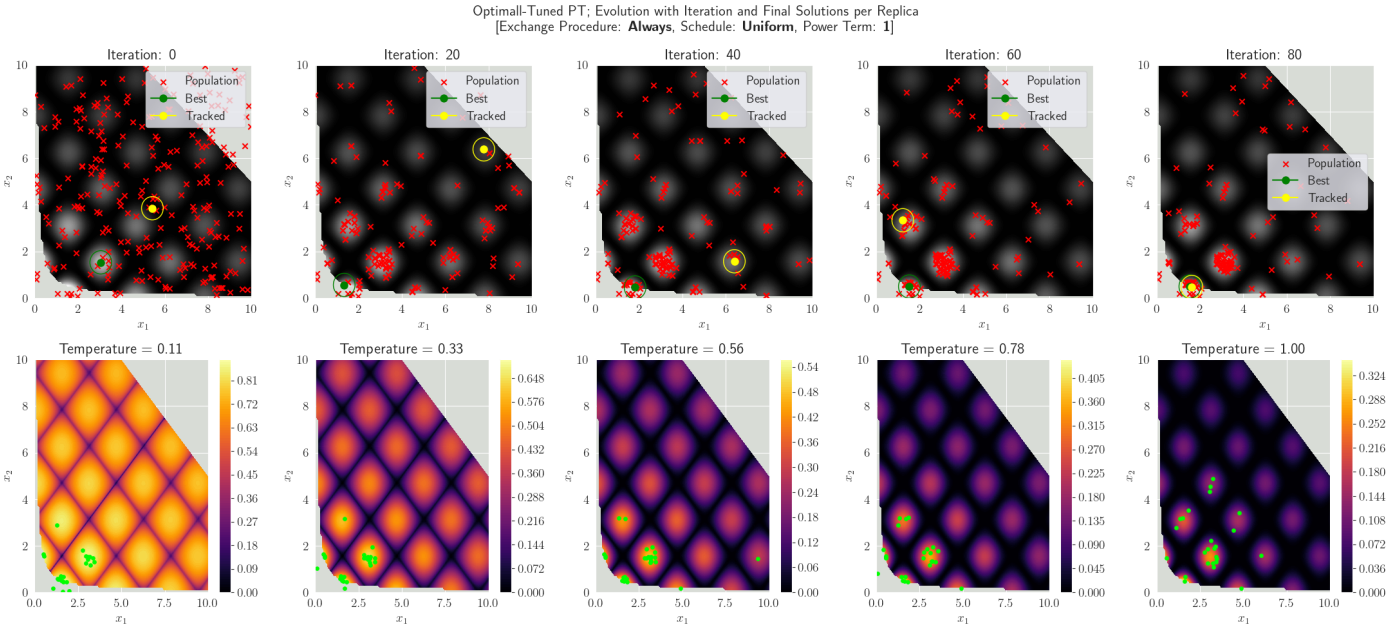


Figure 13: Visualisation of the optimally-tuned PT algorithm. The first row illustrates the evolution of the solution with each iteration. The optimal solution is marked with a green circle. To enhance the understanding of the MCMC updating process, a specific solution within the final replica is highlighted in yellow. The second row displays the final solutions arranged for each replica, overlaid across the smoothened KBF, $f(\mathbf{x})^T$, to help visualise the theory of PT. Lower temperatures facilitate exploration, while higher temperatures encourage exploitation. However, exchange ensures that solutions at higher temperatures are also driven towards the global maxima.

5 Comparison

After fine-tuning both algorithms for the 2-dimensional KBF, a comparative analysis was conducted to assess their performances in optimising the 8-dimensional KBF. 50 optimisation loops were conducted for each algorithm, each with a different initialisation. Collected data included expected values and standard deviations of fitness values, as well as the mean CPU time taken to complete all 10,000 iterations and the average iteration count at which convergence occurred.

To uphold fairness and reproducibility, these initialisations were shared between both algorithms. They were generated using random number seeds within the *generate_initial* function, which is part of the *helper_functions.py* file. The implementation of this function was crucial for ensuring reproducible outcomes and guaranteeing that solutions were initialised away from the global maxima. This was achieved by imposing a constraint that the maximum function value of these initial collections never exceeded 0.3.

Each run was limited to a maximum of 10,000 function evaluations, and convergence was identified as the point where the l_2-normed difference between the minimum fitness values of consecutive iterations stayed within a tolerance of 0.00025 for 1300 consecutive iterations. This was deemed appropriate after the trial run outlined in Section 5.1.

5.1 Trial Run

Prior to the experiment, a trial run of 15,000 funcation evaluations was conducted using one initialisation to assess the suitability of the convergence criterion, and to set expectations.

The outcomes are detailed in Table 1 and illustrated in Fig. 14. These findings underscore a particularly noteworthy result.

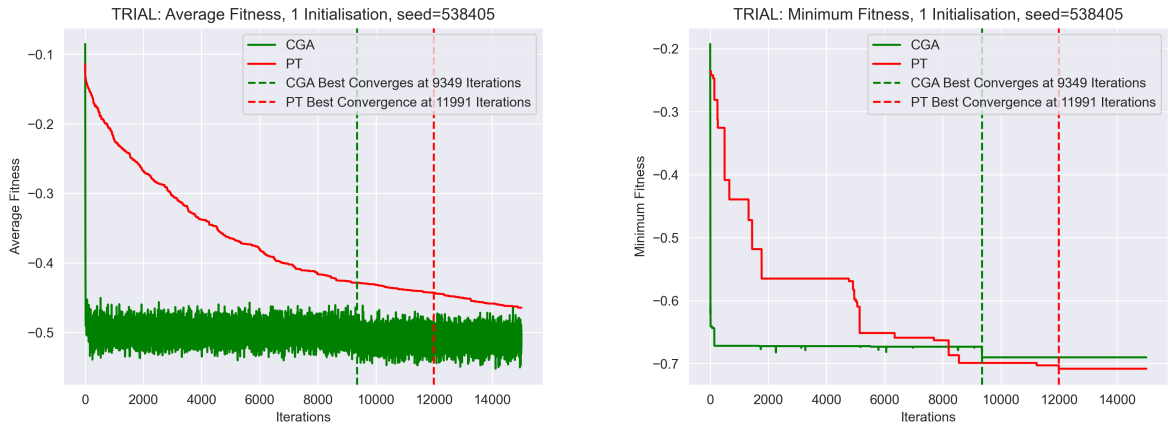
The two algorithms yielded fairly divergent solutions, as described in the caption of Figure 14. The CGA underwent significant change within the initial 1% of iterations, after which it stabilised near its ultimate solution. In contrast, the PT algorithm consistently evolved over the entire span of 15,000 iterations. Consequently, the CGA demonstrated superior performance in the early iterations, but the PT algorithm ultimately surpassed it by converging to a lower minimum fitness value. This observation is a promising indication of the PT's capacity to escape local optima and explore the search space more efficiently.

However, this advantage comes with a trade-off in terms of increased computational demands. Specifically, the PT algorithm required approximately double the time to finish the 15,000 iterations compared to the CGA. Additionally, the PT algorithm required an extra 2551 iterations to converge to its optimal solution, resulting in a greater time investment of approximately 335.9 seconds, with respect to the CGA.

Upon repeated trials, the PT algorithm was unable to consistently overtake the CGA within 10,000 iterations. As a result, it is unlikely that this dynamic will be evident in the final comparison. Nevertheless, the trial run yields valuable insights into the relative performance of the two algorithms and sheds light on potential trade-offs. Additionally, it reinforces the appropriateness of the convergence criterion, which was subsequently employed in the final comparative analysis.

	Final Avg. Fitness	**Final Min. Fitness**	**Total Time Taken**	**Iters to Convergence**
CGA	-0.4971	-0.6901	299.2	9349
PT	-0.4646	-0.7081	592.7	11990

Table 1: Trial comparison between CGA and PT, after 15,000 iterations with one shared initialisation, (random seed = 538405).



(a) Average fitnesses.

(b) Minimum fitnesses.

Figure 14: Evolution of fitness values over 15,000 iterations for the CGA and PT algorithms, using a single initialisation as a trial run, (random seed = 538405).
Best CGA individual: $[3.104, 3.027, 1.624, 0.5872, 0.5363, 0.6130, 0.5294, 0.4808]^T$
Best PT solution: $[3.111, 2.948, 3.071, 1.643, 0.3356, 0.4636, 0.4284, 0.2457]^T$

5.2 Final Comparison

Having validated the experiment with a trial run, the final comparison was conducted using 50 different intialisations, generated with the following random seeds:

References

[1] Ben Calderhead and Mark Girolami. Estimating bayes factors via thermodynamic integration and population mcmc. *Computational Statistics & Data Analysis*, 53(12):4028–4045, 2009.

[2] David J. Earl and Michael W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910, 2005.

[3] M.A. El-Beltagy and A.J. Keane. A comparison of various optimization algorithms on a multilevel problem. *Engineering Applications of Artificial Intelligence*, 12(5):639–654, 1999.

[4] Randy L. Haupt, S. E. Haupt, and Randy L. Haupt. *Practical Genetic Algorithms*, chapter 3: The Continuous Genetic Algorithm, pages 51–66. John Wiley & Sons, Ltd, 2003.

[5] Frank Hoffmeister and Thomas Bäck. Genetic algorithms and evolution strategies: Similarities and differences. In Hans-Paul Schwefel and Reinhard Männer, editors, *Parallel Problem Solving from Nature*, pages 455–469, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[6] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer, 2011.

[7] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9, 1995.

[8] Radford M. Neal. Sampling from multimodal distributions using tempered transitions. *Statistics and Computing*, 6(4):353–366, 1996.

[9] Geoff Parks. Genetic algorithms: Lecture notes. Cambridge University, 4M17 Practical Optimisation Module, 2023. Unpublished.

[10] Geoff Parks. Simulated annealing: Lecture notes. Cambridge University, 4M17 Practical Optimisation Module, 2023. Unpublished.

[11] Geoffrey Thomas Parks. An intelligent stochastic optimization routine for nuclear fuel cycle design. *Nuclear Technology*, 89(2):233–246, 1990.

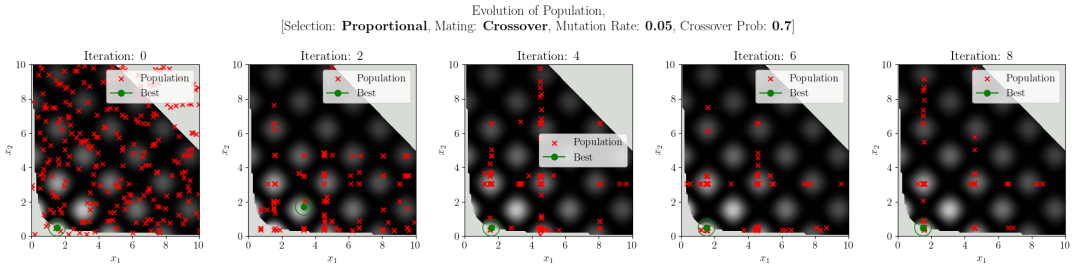
[12] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

6 Appendix

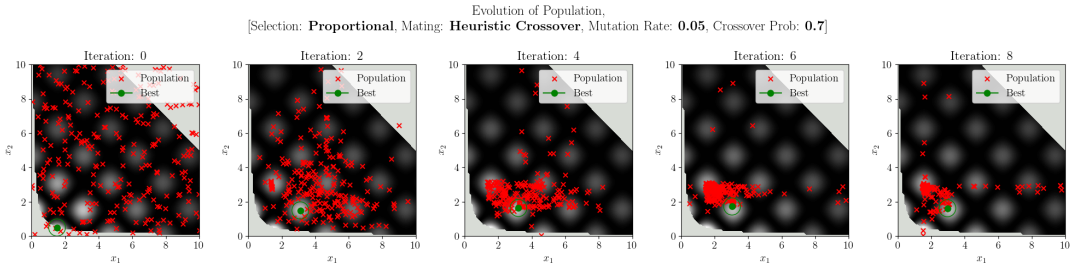
6.1 Supplementary Results regarding CGA Tuning

Selection Method	Mating Procedure	Iterations	Final Avg. Fitness	Final Min. Fitness
Proportional	Crossover	10	-0.19890798	-0.247328018
		100	-0.212384457	-0.24867806
	Heuristic Crossover	10	-0.199296791	-0.258198948
		100	-0.22656183	-0.250977443
Tournament	Crossover	10	-0.309572335	-0.331996331
		100	-0.309235401	-0.342745604
	Heuristic Crossover	10	-0.241261775	-0.262876797
		100	-0.247574635	-0.262876811
SRS	Crossover	10	-0.189986008	-0.208434151
		100	-0.288766718	-0.319667279
	Heuristic Crossover	10	-0.177779432	-0.207148488
		100	-0.182101833	-0.338823558

Table 2: Raw results from an initial exploration of the selection method and mutation procedure hyperparameters within the CGA. Presented as the final fitness values of the CGA population after 10 and 100 iterations. Here, minimum fitness refers to the fitness of the best (feasible) individual within the population. Additionally, the mutation rate was set to 0.05, and the crossover probability was set to 0.7.

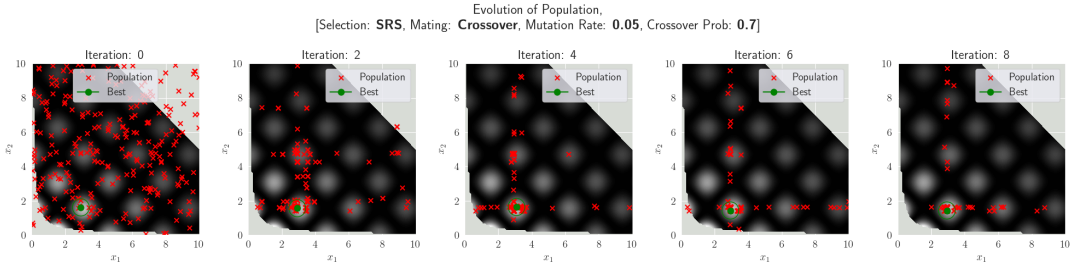


(a) Mutation Procedure: Crossover

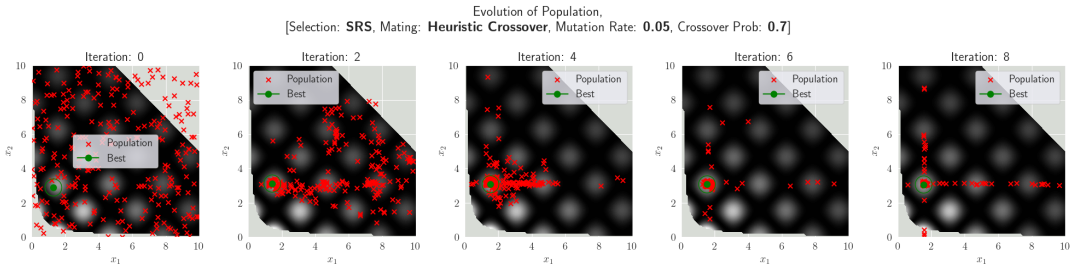


(b) Mutation Procedure: Heuristic Crossover

Figure 15: Evolution of the CGA population over 8 iterations using proportional selection. Proportional selection proved to be a somewhat effective selection method. However, it was not chosen over tournament selection as the primary selection method for the reasons outlined in Section 3.2.



(a) Mutation Procedure: Crossover



(b) Mutation Procedure: Heuristic Crossover

Figure 16: Evolution of the CGA population over 8 iterations using stochastic remainder selection without replacement (SRS). SRS proved to be the second most effective selection method, when compared to Proportional Selection and Tournament Selection, as discussed in 3.2.

6.2 Supplementary Results regarding PT Tuning

Exchange Procedure	Exchange Parameter	Power Term	Final Avg. Fitness	Final Min. Fitness
Periodic	0.1	1	-0.186415811	-0.34081297
		3	-0.186415811	-0.34081297
		5	-0.192142049	-0.34081297
	0.3	1	-0.183591253	-0.335207939
		3	-0.183591253	-0.335207939
		5	-0.178574074	-0.310814711
	0.5	1	-0.166370319	-0.355617525
		3	-0.166370319	-0.355617525
		5	-0.162769374	-0.31189813
Stochastic	0.1	1	-0.182496471	-0.327434397
		3	-0.182496471	-0.327434397
		5	-0.197887075	-0.328011329
	0.3	1	-0.191162383	-0.310872726
		3	-0.191162383	-0.310872726
		5	-0.196042151	-0.314329893
	0.5	1	-0.206621007	-0.347483314
		3	-0.206621007	-0.347483314
		5	-0.211224821	-0.349273739

Table 3: Raw results from an initial exploration of the exchange procedure, exchange parameter, and power term hyperparameters within PT. Presented as the final fitness values of the PT solutions after 100 iterations. Here, minimum fitness refers to the fitness of the best (feasible) solution across all chains.

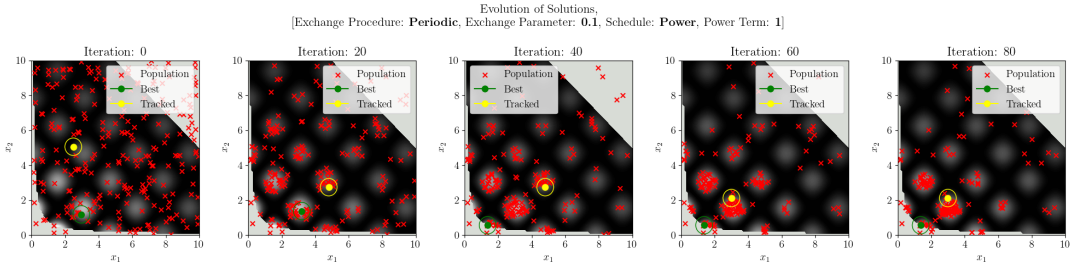


Figure 17: Evolution of the chain across all replicas over 100 iterations using periodic replica exchange and a uniform temperature scheduling, (power term of 1). Here, replica exchange occurs every 10% of the time, (exchange parameter set to 0.1). The 'tracked' solution is distinct from the 'best' solution. It is a random solution within the final replica that is observed to verify the MCMC evolution.

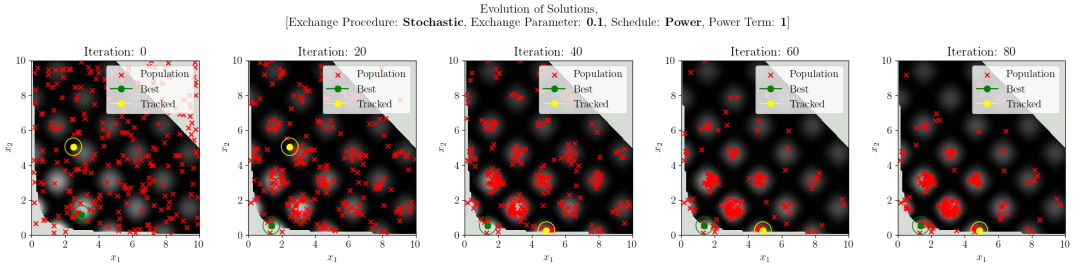


Figure 18: Evolution of the chain across all replicas over 100 iterations using stochastic replica exchange and a uniform temperature scheduling, (power term of 1). Here, replica exchange occurs with a probability of 10%. The 'tracked' solution is distinct from the 'best' solution. It is a random solution within the final replica that is observed to verify the MCMC evolution.

7 Code

To ease development, the codebase has been modularised in the manner presented below:

src/

- *algorithms/*
  - *CGA/*
    - *CGA.py*
    - *mating_functions.py*
    - *selection_functions.py*
  - *PT/*
    - *PT.py*
    - *replica_exchange_functions.py*
    - *temp_prog_functions.py*
- *utils/*
  - *helper_functions.py*
  - *plotting_functions.py*
- *functions.py*

FinalComparison.py

CGA_TuningExperiments.py

PT_TuningExperiments.py

7.1 algorithms

7.1.1 CGA

Listing 1: CGA.py

Listing 2: mating_functions.py

Listing 3: selection_functions.py

7.1.2 PT

Listing 4: PT.py

Listing 5: replica_exchange_functions.py

Listing 6: temp_prog_functions.py

7.2 utils

Listing 7: helper_functions.py

Listing 8: plotting_functions.py

7.3 functions

Listing 9: functions.py

7.4 Experiments

7.4.1 CGA_TuningExperiments

Listing 10: CGA_TuningExperiments.py

7.4.2 PT_TuningExperiments

Listing 11: PT_TuningExperiments.py

7.4.3 FinalComparison

Listing 12: FinalComparison.py