Multi-task Bayesian Optimisation with batch sampling

Student Name: [redacted]

Supervisor Name: [redacted]

Submitted as part of the degree of M.Sc. MISCADA to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

Context/Background - In recent years, Bayesian optimization (BO) algorithms have attracted attention in the field of machine learning because of their efficiency in finding the optimal value of the black-box functions. Most of the current algorithms still focus on using BO for a single task, while in real life, finding the best value or best decision for multiple tasks is more common. Dealing with multiple tasks at the same time is computationally expensive, so more experts tend to study how to use a batch sampling algorithm in multi-task Bayesian optimisation to speed up the optimisation process.

Aims - This project aims to provide a BO framework that could deal with multiple tasks and use a batch sampling algorithm for reducing the running time.

Method - This paper introduces two batch Bayesian optimisation frameworks to deal with multiple tasks. The first one is based on non-dominated sorting genetic algorithms (NSGA) with a mini-batch K-means algorithm. The second one is based on the q-expected hyper-volume improvement (qEHVI) acquisition function. They are suitable for different situations.

Results - 7 benchmark functions including ZDT1 are used to test the first algorithm. The result shows that the first algorithm tends to find Pareto optimal sets for most benchmark functions in a relatively short time but is only allowed to handle the low-dimension input. And a complex and synthetic Branin-Currin function is used to test the second algorithm. The result shows that this algorithm can deal with data set with a high dimension but blindly increasing batch size will lead to a larger amount of computation.

Conclusions – Although both algorithms have unavoidable shortcomings, for example, the first algorithm cannot handle high-dimension data set and the second algorithm is easy to cause memory intensive, both of them have achieved relatively good performance in dealing with multiple tasks under the constraints of different computing environments.

Keywords — Bayesian optimisation, Multi-task learning, batch sampling, Gaussian process, Pareto frontier

I INTRODUCTION

Optimisation problems are everywhere in life, such as how to choose subway routes to work or how to choose flights and hotels when traveling. Simply put, the optimization problem is the problem of finding the best solution from all feasible solutions. In the field of machine learning, optimisation problems have also received high attention, such as finding the best set of hyper-parameters for neural networks. Black-box functions are expensive or even impossible to evaluate, they have unknown mathematical representations, convexity properties and the choice of hyper-parameters undoubtedly is a black-box optimization problem because we only see the input and output of the model during the tuning process, and cannot obtain the gradient information of the model training process, nor can it be assumed that the model hyper-parameters and final indicators meet the convex optimization conditions, otherwise we can derive the optimal solution through derivation or convex optimization methods. And We know that it takes several hours or even months for hyper-parameters to train a model. So if only rely on the experience of experts or rely on automatic tuning algorithms such as grid search and random search, it is obvious that getting an expected result is very time-consuming. At this time, the Bayesian optimisation (BO) algorithm shows its great advantages in dealing with black-box functions, it attempts to find the global optimum in a minimum number of steps. This is because compared with the grid search without considering the previous parameter information, Bayesian optimisation can effectively use the complete historical information to improve the search efficiency and avoid unnecessary sampling. In general, Bayesian optimisation solves the following problems:

$$x^* = argmax_{x \in \chi} f(x)$$

where χ is the parameter space we are interested in. In the field of machine learning, it can also be understood as a hyper-parameter space. And $f(x)$ is the mapping function from the parameter space to an indicator we are interested in. Usually, this indicator is the problem that we actually care about. When modeling $f(x)$ with the Gaussian process, $f(x)$ is usually assumed to be a continuous function.

Reference to (Brochu et al. 2010), the idea of the BO algorithm to find the extreme value of the objective function $f(x)$ is to first generate an initial solution set D, then find the next candidate that may be an extreme value based on this data set D. After that, add this candidate to the data set D and repeat this step until the iteration ends. Finally, find the extreme point from these points as the solution to the problem. The following algorithm gives the simplest form of Bayesian optimisation.

Algorithm 1 Bayesian Optimisation

```
**Input:**$f(objective\ function), X(search\ space), a(acquisition\ function), M(surrogate\ model)$
$D \leftarrow$ **Initialize data points**$(f, X)$
**for** $i \leftarrow |D|$ **to** $T$ **do**
    $p(y|x, D) \leftarrow$ **Fit data set to the surrogate model**$(M, D)$
    $x_i \leftarrow argmax_{x \in X} a(x, p(y|x, D))$
    $y_i \leftarrow f(x_i)$    $\triangleright$ **Expensive step**
    $D \leftarrow D \cup (x_i, y_i)$
**end for**
```

The key issue here is how to determine the next candidate point based on the data set D. Bayesian optimisation estimates the mean and variance of the true objective function value based on data set D. Then an acquisition function $S(x, p(y|x, D))$can be constructed according to this posterior probability. The acquisition function can trade-off exploitation and exploration and then give us suggestions for finding the next sampling points in the search space. Exploitation means it will try to choose the point close to the known point as the reference point for the next iteration, that is, try to dig out the points around the known points. The distribution of points will appear in a dense area, which will lead to a local maximum. While exploration means it tends to choose a point far away from the known points as the candidate point for the next iteration, that is, try to explore the unknown area, and the distribution of points will be as even as possible. In other words, exploration is to choose places with large variance. This is because according to the Gaussian regression process, the variance is greater in places far from the known points.

And for the costly and complex objective function, we often use a surrogate model such as the Gaussian process instead. There are various kinds of surrogate models and acquisition functions. As two key components for a Bayesian optimisation framework, choosing an appropriate one will influence the performance of optimisation. As for the different types of surrogate models and acquisition functions, they will be introduced in detail in the next section.

Recently, more experts attempt to do an extension for single task Bayesian optimisation. One of the development directions is enabling Bayesian optimisation deal with multiple tasks at the same time instead of just focusing on one single target function. This is a promising development direction because, in real life, it is more likely to encounter situations where multiple objective functions are supposed to be optimized at the same time and these objective functions might restrict each other. Besides, as mentioned in the paper (Swersky et al. 2013), the independent learning of a single task often ignores the experience information from other tasks. In some cases that the target task is lack of information, blindly trains the model for the target task just a waste of learning resources. In order to alleviate these problems, multi-task learning methods belonging to the category of transfer learning have gradually attracted the attention of researchers. Unlike single-task learning that only uses sample information of a single task, multi-task learning assumes that there is a certain similarity between different distributions, and again, based on joint training and optimisation, the connection between tasks is established. This training mode fully promotes the exchange of information between tasks and achieves the purpose of mutual learning, allowing the target task to benefit from it. And by sharing the parameters between different tasks to a certain extent, it may generalize the target task better (Swersky et al. 2013). As long as there is more than one loss function, it can be considered as multi-task learning, therefore, this form of assisting the target task with multiple auxiliary tasks together with joint learning also regarded as multi-task learning. And when dealing with multi-task learning, if the relationship between tasks is assumed in advance, and the objective function is designed according to this hypothetical relationship, once the model assumptions used are unreasonable, it may cause negative transfer between tasks. To overcome this problem and increase the flexibility of the multi-task learning structure, multi-task learning based on Bayesian methods has been studied. This is because Bayesian multi-task learning does not require a pre-defined relational structure.In conclusion, that is why multi-task learning has received attention in the field of Bayesian optimization.

Even though adapt the Bayesian optimisation algorithm to address the multi-task optimisation problems, this is still a difficult problem to solve because the objective functions may not be independent of each other or may even restrict each other. And the contradictions between the goals make the problem impossible to have a single absolute optimal solution, but there is a set of balanced solutions, the so-called Pareto optimal solution set. So it can also be said that the main purpose of the multi-objective optimization technique is to find one or more satisfactory solutions in the Pareto solution set. The definition of a multi-objective optimization problem is: the decision variable $X = (x_1^*, x_2^*, ...x_n^*)^T$ that meets the constraints makes the objective vector function $f(x)$ optimal. Here, The constraint condition is $g_i(x) \leq 0, i = 1, 2, ..., m$. m is the number of constraint conditions. $f_i(x)(1 \leq x \leq k)$ is the vector function of each sub-objective, and k is the number of sub-objectives. So a multi-task optimization problem is composed of n decision variables, m constraints, and k sub-objectives. The sub-objective function is linear or non-linear. The optimization function is to map the decision vector X to the objective vector$f(x)$. The essence of the multi-objective optimization problem is that there are contradictions between each sub-objective. In the process of changing the value of the decision-making variable, the improvement of one sub-objective may cause a decrease in the performance of another sub-objective. That is to say, it might be impossible to optimise all objective functions at the same time, but only multiple Pareto dominant solutions, namely Pareto optimal frontiers (Eckart et al. 2001).

Most of the previous multi-task learning is mainly used in the fields of regression and prediction, and starting from the multi-task Bayesian optimisation (Swersky et al. 2013), more and more scholars have begun to study multi-task learning in the field of Bayesian optimization. Multi-task Bayesian optimisation is an outstanding development that supports automatic machine learning hyper-parameter optimization. The multi-task Gaussian process model is used to adaptively learn the dependencies between tasks, and the overall search of the Bayesian optimisation algorithm is improved, and good results are achieved.

To deal with multi-task optimisation problem, two kinds of algorithms are often used, one of Evolutionary Algorithms (EAs) and the other is Bayesian Global Optimization (BGO). Compared with EAs, BO shows the superiority in finding the optimal solution with as few iterations as possible which reduces the budget of evaluating (Yang et al. 2019) while EAs such as NSGA II prone to convergence stagnation in the later stage of evolution and waste the resource of learning. And BO is restricted by the curse of dimensionality while EAs can deal with relatively high dimensional input. Because these two algorithms for handling multi-task optimal problems are complementary in some aspects, I have an idea of how to add EAs to the BO framework, hoping that this algorithm can converge faster and can be extended to a little higher dimensions. After the experiment, I found there is a little improvement in the convergence speed compared to the traditional EAs, and the K-means batch BO algorithm is used to further optimise the convergence speed. But there is no good progress in terms of improving the ability to handle higher dimensional input. Because the result did not meet my expectation, I tried to find another algorithm that can deal with high dimensional input and at the same time has a relatively high speed of convergence. Unfortunately, the algorithm I found based on hyper-volume improvement is likely to cause memory intensive, so this algorithm is recommended to run on a supercomputer or a computer with a better configuration.

In this paper, I introduce various kind of surrogate model, acquisition function, batch algorithms and an evolution of multi-task learning in section Related Work, which could help the reader get a better understanding of multi-task Bayesian optimisation. Then in section Solution, it gives the details of my algorithms, the corresponding code is also given as a supplement material. In section Results and Evaluation, my algorithms are tested by a variety of benchmark functions and complex problems, showing their good performance in dealing with multiple tasks and get shortcomings by testing their limits. In the last section, it gives the overall evaluation of my algorithms and suggests some directions for improvement which can be seen as future work.

II RELATED WORK

As I mentioned in the last section, BO has two key components, the one is the surrogate model and the other is the acquisition function.

A Surrogate model

In real life, many objective functions tend to be unknown and computationally expensive to evaluate. In this case, a surrogate model that is relatively easier to calculate is used to represent the objective function. According to whether the number of parameters of the model is fixed, surrogate models can be classified into parametric models and non-parametric models.

As for parametric models, in the process of data increase and optimisation, the number of model parameters always remains unchanged. In Bayesian theory, the posterior probability distribution is generally difficult to obtain a closed solution. The usual solution is to select a conjugate prior distribution for the likelihood distribution so that the posterior probability distribution and the prior distribution It has the same expression form and is easy to calculate. One of the simplest surrogate models is the Beta-Bernoulli model. Bernoulli distribution and beta distribution are conjugate, this model can be applied not only to drug design problems, but also A/B test (Scott 2010) and other fields. The linear model is another parametric model that is often used. In many applications, it is usually assumed that the decisions are independent of each other. By establishing a linear model to capture the relationship between variables, the purpose of reducing the number of evaluations can be achieved (Shahriari et al. 2015).

In machine learning, highly flexible models can usually get satisfactory prediction results. This is mainly because these models have high scalability. Generally, there are two ways to extend the flexibility of the model. One way is making the parametric model have more parameters than the data set. Another method is using a non-parametric model. In a non-parametric model, the parameters of the model increase as the amount of data increases, and there is even an infinite number of parameters. Therefore, compared to a parametric model with fixed parameters, the non-parametric model is more flexible and is not easy to be overfitting.

Gaussian processes (GPs) is a commonly used non-parametric model. At present, Gaussian processes have been widely used in regression, classification, and many fields that need to infer black-box functions. This is also the surrogate model I used in my algorithms. The Gaussian process involves two important words Gaussian and process. Gaussian refers to Gaussian distribution and process refers to a random process, . Putting it together means that we have a set of random variables, for any point, it obeys Gaussian distribution. And also due to the property of Gaussian distribution, the prior and posterior of the Gaussian process are very cheap to calculate, which is why the Gaussian process is widely used in Bayesian optimisation (Williams & Rasmussen 2006).Citation to paper (Swersky et al. 2013), Gaussian processes are specified by a mean function $m : XR$ where $X = x_n \in \chi_{n=1}^{N}$, and a co-variance (kernel) function$K : XXR$. The predictive mean and co-variance under a GP can be respectively expressed as:

$$\mu(x; x_n, y_n, \theta) = K(X, x)^T K(X, X)^{-1}(y - m(X)) \tag{1}$$

$$\Sigma(x, x'; x_n, y_n, \theta) = K(x, x') - K(X, x)^T K(X, X)^{-1} K(X, x') \tag{2}$$

Here K(X, x) is the N-dimensional column vector of cross-co-variances between x and the set X and the N N matrix K(X, X) is the Gram matrix for the set X. And following is the kernel function for the multi-task learning:

$$K_{multi}((x, t), (x', t')) = K_t(t, t') \bigotimes K_x(x, x')$$

where t represents the $t{-}th$ task, $\bigotimes$ denotes the Kronecker product, $K_x$measures the relationship between inputs, and K_t measures the relationship between tasks.

Other surrogate model such as student-t process and neural network are also considered as a surrogate model. Student-t distribution can be obtained by superimposing unlimited Gaussian distributions with the same mean and different variances. Compared with the Gaussian distribution, it is more insensitive to outliers and therefore has higher robustness. As for neural network, It is understood as regularization by introducing uncertainty into the weight of the neural network, which is equivalent to ensemble of an infinite number of neural networks on a certain weight distribution for prediction (Martinez-Cantin et al. 2018). Using neural network as the surrogate model might enable the BO deal with higher dimensional input but at the same time, it is really computationally expensive.

B Acquisition Function

Now, we discuss acquisition functions that can trade off exploitation and exploration and then give us suggestions for finding the next sampling points in the search space. Exploration means it tends to choose a sampling point far away from the known point as the reference point for the next iteration, that is, try to explore the unknown area, and the distribution of points will be as even as possible. In other words, exploration is to choose places with large variance. This is because according to the Gaussian regression process, the variance is greater in places far from the known points. While exploitation means it will try to choose the point close to the known point as the reference point for the next iteration, that is, try to dig out the points around the known point. The distribution of points will appear in a dense area, which might lead to a local maximum. Next, I introduce 3 basic acquisition functions.

B.1 Expected Improvement (EI)

EI acquisition function is improved based on the probability of improvement (PI) acquisition function (Brochu et al. 2010). Compare with PI, EI can show the extent of improvement that a point may produce. Suppose the goal of the function is to find the global minimum, EI can be simply defined as

$$\mathbf{EI}(x) = \mathbf{E}max((f(x) - f(x^+), 0),$$

where $f(x)$ is the minimal value found from known observations, x^+ is the point we expect that its corresponding value $f(x^+)$ will be smaller than $f(x)$. The intuition of EI acquisition function is that according to the known observations, we can get that the optimum observation value $f(x)$ at point x, then suppose we have an additional evaluation that could be performed anywhere and get $f(x^+)$ at x^+. Compare this possible optimum value $f(x^+)$ with original optimum value,$f(x^+)$ will replace $f(x)$ as a new optimum value if it is smaller than $f(x)$ and the improvement will be $f(x) - f(x^+)$, otherwise the improvement will be 0.

B.2 Upper Confidence Bound (UCB)

UCB acquisition function is easiest one to understand. It can be defined as

$$\mathbf{UCB}(x) = \mu(x) + \kappa\sigma(x),$$

where κ is a parameter that can be set by users (Brochu et al. 2010), and it is intuitively understood as the upper confidence boundary, for example, when κ =1.96, UCB is the upper bound of the 95% confidence interval.

B.3 Entropy Search (ES)

ES acquisition function proposed by (Hennig & Schuler 2012) tend to find the next sampling point that causes the largest decrease in differential entropy. It can be defined as

$$\mathbf{ES}(x) = \mathbf{H}(P(x^+)) - \mathbf{E}_{f_x}[\mathbf{H}(P(x^+ \mid f(x))],$$

where $H(\Delta)$ represents the entropy of (Δ) and E_{f_x} represents expectation is taken over $f(x)$. However, calculating the entropy is computationally expensive and finding optimum value of ES acquisition function might be difficult. Then predictive entropy search (PES) acquisition function proposed by (Hernández-Lobato et al. 2014) alleviates this problem using the symmetry of mutual information and $H(P_n(f(x)))$ can be calculated in a close form, it can be expressed as

$$\mathbf{PES}(x) = \mathbf{ES}(x) = \mathbf{H}(P_n(f(x))) - \mathbf{E}_{x^*}[\mathbf{H}(P_n(f(x) \mid x^*))].$$

B.4 Portfolio of Acquisition Function

Bayesian optimization algorithm using a single acquisition function cannot show the best performance on all problems. Therefore, to obtain a robust method, (Hoffman et al. 2011) introduced a combination of hedging strategies GP-Hedge for multiple acquisition functions. In each iteration, the GP-Hedge algorithm combines the candidates obtained from each acquisition function and then select evaluation points from the set of candidates according to the hedging strategy. Besides, (Shahriari et al. 2014) proposed an information-based portfolio strategy entropy search portfolio (ESP). Unlike GP-Hedge, ESP can provide the most information for the global optimal solution based on entropy search algorithms. The ESP strategy is also highly robust and compared with GP-Hedge, this algorithm can tolerate the poor performance of the acquisition function in the portfolio.

C Batch Sampling

BO is a sequential design strategy for optimisation. To speed up the efficiency of BO, many experts attempt to apply the batch sampling algorithm to the acquisition function, hoping that the acquisition function will find multiple candidates in one iteration instead of returning only one candidate per iteration. (Ginsbourger et al. 2010) proposed a pluralisation method. The main idea of this method is to combine the observed data set $D_{1:t}$ and the data set $D_{1:p} = ((x_1', y_1'), ..., (x_p', y_p')$, which is being observed and not yet finished to select the next evaluation point. When y_p' is a constant, the method is called a constant liar strategy and when y_p' is the mean value of x_p' at Gaussian process prediction, this method is called Kriging believer strategy.

And (González et al. 2016) proposed a highly effective heuristic approach based on an estimate of the function's Lipschitz constant. Due to the property of Lipschitz-continuity and a local penalizer, the acquisition function has a local exclusion around the new evaluation and then every element in each batch can be distributed as expected. This algorithm can be considered as an extension of the entropy search acquisition function. A very outstanding advantage of this method is that there is no need to re-compute the GPs after every point is selected. Also, it has limitations, for instance, it requires a sample path from the GP measure that needs to be Lipschitz-continuous. Although the idea of this algorithm is interesting and it is not computationally expensive, this algorithm is only allowed to be used in the single-task optimisation. One simple way to apply this algorithm to multi-task Bayesian optimisation is combining multiple objective functions into one target function through the weight method, of course, this method requires that there is no mutual restriction between the objective functions which seems impossible in reality.

In the single-objective setting, a large body of work focuses on practical extensions to BO for supporting parallel evaluation and outcome constraints. Less attention has been given to such extensions in the multiple-objective setting. This is mainly because the acquisition functions in the multiple-objective setting are computationally expensive (Groves & Pyzer-Knapp 2018). In batch Bayesian optimisation algorithms, batch expected improvement (qEI) and batch upper confidence bound (qUCB) has aroused the research interest of scholars. However, qEI and qUCB were just one step look-ahead batch algorithms and mostly used in single task (Wu & Frazier 2019) until Monte-Carlo simulations are used to evaluate acquisition functions, then multiple acquisition functions including qEI and qUCB acquisition functions tend to find multiple candidates in one iteration. (Marmin et al. 2015) showed that acquisition functions estimated via Monte Carlo integration are consistently amenable to gradient-based optimization. And then, in this year a new acquisition function proposed by (Daulton et al. 2020) based on qEI is called q-expected hyper-volume improvement (qEHVI) which shows an impressive performance on multi-task Bayesian optimisation. Apart from applying the batch algorithm for acquisition functions, a novel algorithm proposed by (Groves & Pyzer-Knapp 2018) recently use K-Means to conduct batch Bayesian optimisation. More details of the last two batch algorithms will be introduced in the Solution section.

D Multi-task Bayesian Optimisation

Most of the previous multi-task learning is mainly used in the fields of regression and prediction, and starting from the multi-task Bayesian optimisation (Swersky et al. 2013), more and more scholars have begun to study multi-task learning in the field of Bayesian optimization. Multi-task Bayesian optimisation is an outstanding development that supports automatic machine learning hyper-parameter optimization. The multi-task Gaussian process model is used to adaptively learn the dependencies between tasks and achieve good results. However, BO is usually limited to fairly low or medium dimensional problems. Besides, the application of Bayesian optimisation is mainly limited to the continuous search space and is not directly applied to the combined search space, in which indefinite kernels may be difficult to handle. In contrast, evolutionary algorithms (EAs) have been very promising in filling these gaps. EAs provide great flexibility in adapting to different data representations and can be well extended to higher dimensions. Therefore, many multi-task evolutionary optimisation algorithms have been produced. However, EAs like NSGA algorithms prone to convergence stagnation in the later stage of evolution while BO has better performance in this aspect. Therefore, combining BO and EAs is considered to be a promising way. And a few years after the popularity of EI as an acquisition function in Bayesian optimisation, (Emmerich 2005) promoted EI to EHVI according to the excess index. Similar to EI, EHVI is the expected increment of the excess index, considering the Pareto frontier approximation set and the predicted multivariate Gaussian distribution at the new point but EHVI has a high computational overhead. Inspired by EHVI acquisition function and Monte Carlo integration, (Daulton et al. 2020) proposed a novel formulation of q-Expected Hyper-volume Improvement (qEHVI), an acquisition function that extends EHVI to the parallel, constrained evaluation setting, improving efficiency for optimisation.

III SOLUTION

In this section, I will introduce two kinds of multi-task Bayesian optimisation algorithms with batch sampling. The first one is mainly based on non-dominated sorting genetic algorithms (NSGA) with the K-Means batch sampling algorithm. For this algorithm, there is no need to use a computer with configuration, it can be used by private laptop or a jupyter notebook provided by Durham University. Although it tends to perform well in finding optimal solutions for complex Pareto frontiers, it can only handle data sets with relatively low dimensionality. The second algorithm is based on q-expected hyper-volume improvement acquisition function which can accurately calculate the joint EHVI of q new candidate points. It has a good performance when input is high dimensional, but it needs a highly configured computer or even supercomputers such as COSMA7. Next, I will show the details of these two algorithms.

A MTBO Based on Non-dominated Sorting Genetic Algorithm

Multi-objective evolutionary algorithms (MOEAs) mainly rely on non-dominated solution sorting to promote the population search Pareto front. It has better global optimisation performance then even if the number of iterations increases a lot, there is no improvement in approaching the true Pareto optimal solution. In order to address this problem, (Feliot et al. 2017) proposed an algorithm incorporates the selection method of the non-dominated sorting genetic algorithm-II (NSGA-II) into the Bayesian optimization framework, using extended governing rules to deal with goals and constraints in a uniform manner, and proposed corresponding expected over-improvement sampling criteria. Before introducing NSGAII and NSGA III, It is necessary to briefly explain some basic concepts of Pareto frontier.

Pareto dominance relation: For minimizing multi-objective optimisation problems, for n target components $f_i(x), i = 1, ..., n$, given two decision variables X_a and X_b, if the following two conditions are true,X_a is said to dominate X_b.

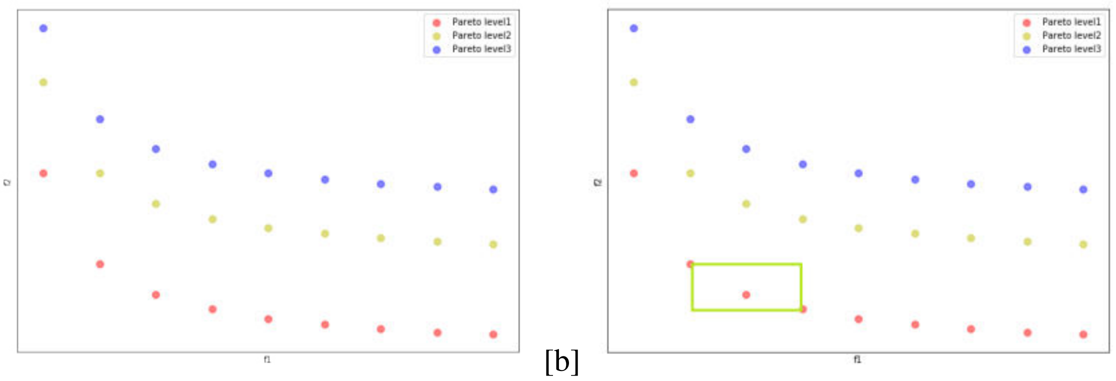
1. For $\forall i \in 1, 2, ..., n$, there exists $f_i(X_a) \leq f_i(X_b)$.

2.$\exists i 1, 2, ..., n$, there exists $f_i(X_a) \leq f_i(X_b)$.

If there is no other decision variable that can dominate this decision variable, then this decision variable is called a non-dominated solution.

Pareto level: In a set of solutions, the Pareto level of the non-dominated solution is defined as level 1. The non-dominated solution will be deleted from the set of solutions and then the Pareto level of the remaining solutions is defined as level 2, and so on. Figure 1 [a] clearly shows an example of Pareto dominance relation and their levels.

Crowd: In order to avoid the solution distribution too uneven in the search space, the crowding degree n_d is introduced. Take a two-objective optimization problem as an example, It is like the sum of the side lengths of the largest rectangle that the individual can generate in the (the rectangle cannot touch other points in the search space). The green rectangle in Figure 1 [b] illustrates this concept.



[a] [b]

Figure 1: Pareto Frontier

Now, I start to introduce NSGA. There are three generations of NSGA. The NSGA II is based on the original NSGA. The NSGA was abandoned because it was too outdated while NSGA II has widely used as the a baseline algorithm in dealing with multi-task problems. And NSGA III again has an improvement based on NSGA II which is still an advanced algorithm.

A.1 NSGA II

Compared with NSGA, NSGA II reduce calculation pressure and speed up convergence. This algorithm proposed by (Deb et al. 2002) merges the parent population with the offspring population so that the next generation population is selected from a larger space. Apart from that, they also introduced elite strategies to ensure certain excellent populations remaining. Finally, the algorithm uses the congestion degree to Prevent the population from being too densely distributed.

A.2 NSGA III

The main idea of NSGA III is to introduce a reference point mechanism based on NSGA II, and retain those population individuals that are not dominated and close to the reference point (Deb & Jain 2013). The following summarizes the steps of the $t-th$ generation of NSGA III. P_t is the parent of the $t-th$ generation, and its population size is N. Its generated offspring is Q_t with the same population size N. The algorithm starts with combining the offspring and the parent $R_t = P_t \cup Q_t$ (population size = $2N$) and then select N individuals from them. As for selection process, we divide R_t into multiple non-dominated layers $(F_1, F_2...)$ through non-dominated sorting firstly. Next, we construct a new population S_t from F_1 until its population size is equal or greater than$N$ and call the last layer the $l-th$ layer. The solution above the $(l+1)-th$ layer will be abandoned.

But, not all the solutions in $l-th$ layer will be accepted. The selection strategy of the key layer ($l-th$ layer) is also the biggest difference between NSGA II and NSGA III. As I mentioned above, NSGA III introduces a series of reference points. For all the solutions in the layer before the key layer, each solution will be associated with a reference point. Which reference point each solution is related to depends on its distance to the reference line (the principle of proximity). After the association, We define the number of solutions for each reference point as ρ_j. If $\rho_j = 0$, choose a solution with the smallest distance to the reference point to join the population from the key layer (if any), otherwise remove the reference point from the current generation. If $\rho_j > 0$, then randomly select a solution related to the reference point from the key layer and add it to the population (if any) (Jain & Deb 2013). Reference to (Deb & Jain 2013), Algorithm 2 shows the main part of algorithm NSGA III:

```
**Algorithm 2** Generation t of NSGA-III procedure
**Input:** reference points $Z^s$ , supplied aspiration points $Z^a$, parent population $P_t$ **Output:** $P_{t+1}$
$S_t = \emptyset, i = 1$
$Q_t = Recombination + Mutation(P_t)$
$R_t = P_t \bigcup Q_t$
$(F_1, F_2, ...) = Non - dominatedsort(R_t)$
**repeat**
    $S_t = S_t \bigcup F_i \ and \ i = i + 1$
**until** $|S_t|N \ Lastfronttobrincluded : F_l = F_i$ **if** $|S_t| = N$ **then**
    $P_{t+1} = S_t, break$
**else**
    $P_{t+1} = \bigcup_{j=1}^{l-1} F_j$
    Points to be chosen from $F_l : K = N - |P_{t-1}|$
    Normalize objectives and create reference set $Z^r$: $Normalize(f^n, S_t, Z^r, Z^s, Z^a)$
    Associate each member s of $S_t$ with reference point: $[\pi(s), d(s)] = Associate(S_t, Z^r)$
    Compute niche count of reference point$j \in Z^r$: $\rho_j = \Sigma_{s \in S_t/F_i}((\pi(s) = j)?1 : 0)$
    Choose K members one at a time from $F_l$ to
    construct $P_{t+1} : NichingK, \rho_j, \pi, d, Z^r, F_l, P_{t+1}$
**end if**
```

Now, The specific steps of my first algorithm are given and the main packages I use are GpyOpt and gpytorch . The algorithm uses the Bayesian optimisation framework, so there is no doubt the first step is to generate the initial data set. Here I use the random version of Latin Hyper-cube Sampling (LHS). And the surrogate model I chose is the exact Gaussian process with kernel Martern32 instead of the approximate Gaussian process model even though computational constraints have limited exact GP to problems with fewer than about ten thousand training points (Wang et al. 2019). This is because the target of this algorithm is not to handle a large number of data set and the accuracy of modeling for this algorithm is more important. As for acquisition functions, both expected improvement and upper confidence bound are working. After fitting the surrogate model with the initial data set, I use adam optimizer for likelihood optimization and get a loss for GPs. Setting a likelihood optimization criteria is also important because stopping the loop when the conditions are met can effectively reduce the amount of calculation and time. And I set all parameter gradients equal to 0, then fill in gradients by calling backward on the loss. As for finding new candidates, I create an acquisition function and used it together with the number of generation and population as the input of a pre-defined NSGA2 or NSGA3 class. NSGA algorithms with acquisition functions will select the promising members in each generation and carries out evaluations for them. In terms of batch sampling algorithm, K-Means Batch Bayesian Optimization proposed by (Groves & Pyzer-Knapp 2018) caught my attention which uses unsupervised learning to efficiently estimate peaks of the model acquisition function. And according to the paper (Sculley 2010), Mini-Batch K-Means has a better performance than classic and full batch K-Means, so I use this Mini-Batch K-Means batch algorithm finally.

B MTBO Based on q-Expected Hyper-volume Improvement

This algorithm is my second algorithm based on a novel acquisition function qEHVI. Besides packages GpyOpt and gpytorch, package botorch is also used frequently. First, I use the exact Gaussian process model as my surrogate model for a similar reason that caring more about the accuracy of the model. To deal with multi-task problems, I use a multi-output single task exact GP model based on the exact GP model which assumes conditional independence of the outputs given the input. More specifically, this model works in batch mode and model the outputs using exact GPs independently (each batch has its hyper-parameters). It requires only a single posterior method that takes in a Tensor X of design points, and returns a posterior object describing the (joint) probability distribution of the model output(s) over the design points in X. As for the acquisition function, I choose q-expected hyper-volume improvement acquisition function (qEHVI), a new acquisition proposed by (Daulton et al. 2020) this year. To better introduce this algorithm, we need to know the concept of hyper-volume and hyper-volume improvement first.

The definition of hyper-volume (HV): Given a reference point $r \in R^M$, the hyper-volume indicator of a finite approximate Pareto set P is the M-dimensional Lebesgue measure λ_M of the space dominated by P and bounded from below by r:

$$HV(P,r) = \lambda_M(\cup_{i=1}^{|P|}|r, y_i|)$$

The definition of hyper-volume improvement (HVI): Given a Pareto set P and reference point r, the hyper-volume improvement of a set of points Y is:

$$HVI(Y,P,r) = HV(P \cup Y, r)HV(P,r).$$

Then, expected hyper-volume improvement is the expected value of HVI over the posterior:

$$\alpha_{EHVI}(\chi_{cand}) = E[HVI(f(\chi_{cand}))]$$

And according to the assumption, objective functions are independent GPs, which means EHVI can also be expressed in closed form.

Reference to the paper (Daulton et al. 2020), (Daulton et al. 2020) gave the specific formula for calculating HVI:

$$HVI(f(x)) = \sum_{k=1}^{K} HVI_k(f(x), l_k, u_k) = \sum_{k=1}^{K}\prod_{m=1}^{M}[z_k^{(m)} = l_k^{(m)}]_+$$

where $u_k^{(m)}$, $l_k^{(m)}$, $f^{(m)}(x)$ and $z_k^{(m)}$ represent the m^{th} component of the corresponding vector and $[\Delta]_+$ denotes the min(·,0) operation. As for q-hyper-volume improvement, given q new points$f(x_i)_{i=1}^q, let A_i := \delta(f(x_i), P, r)\ for\ i, ..., q$ be the space dominated by $f(x_i)$ but not dominated by P, independently of the other $q-1$points. The union of the subsets A_i is the space dominated jointly by the q new points, so we can get:

$$HVI(f(x_i)_{i=1}^q = |\cup_{i=1}^q A_i| = \sum(-1)^{j+1} \sum_{1 \le i_1 \le ... \le i_j \le q} |A_{i_1} \bigcap ... \bigcap A_{i_j}|$$

where S_k is the k^{th} hyper rectangle and expected volume for qHVI over posterior can be represented as following:

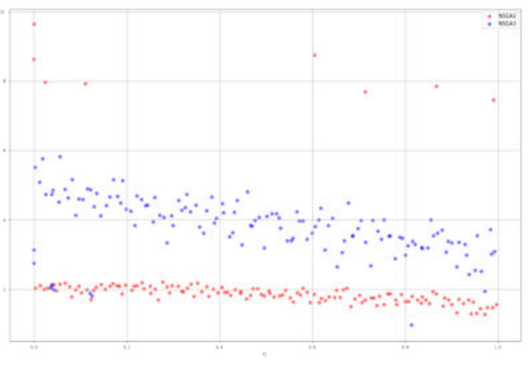
$$\alpha_{qEHVI(x)} = E[HVI(f(x_i)_{i=1}^q)] = \int_{-\infty}^{\infty} HVI(f(x_i)_{i=1}^q) df$$

Considering the expensive computation of qEHVI, Monte Carlo sampling based on for high dimensional cases (d¿6) will be used to calculate EHVI (Luo et al. 2014), where d stands for the dimension in objective space. And like NSGA III, qEHVI also needs a reference point, which can be simply defined as 1 in practice. Next, I define a helper function that performs the essential BO step (initialization, optimisation and returning new candidates) for qEHVI. Because qEHVI uses the inclusion-exclusion principle to calculate the joint HVI of each MC sample at the pending points $x_1, ..., x_{i-1}$ and the new candidate x_i, thereby reducing the amount of calculation by avoiding determining the Pareto set for each sample. In terms of optimizer of acquisition function, I simply choose LBFGS. Finally, using the basic Bayesian optimisation framework to start the loop.

IV RESULTS AND EVALUATION

In Pareto optimization, using less time or less number of evaluations to find an approximate Pareto frontier means that this algorithm has relatively high efficiency. First, I use 7 benchmark functions to test the first algorithms. I chose these 7 multi-objective benchmark functions because they have a variety of complex properties, and it is difficult to converge to the Pareto optimal frontier (Dong et al. 2005). The benchmark functions are ZDT1, ZDT2, ZDT3, schaffer, binh-korn, chakong-haimes , osyczka-kundu.

Taking benchmark function ZDT2 as an example, when increasing the number of iterations, the Pareto frontier estimated by the algorithm will gradually approach the true Pareto distribution and Figure 2 shows this evolution. In order to make data visualization more convenient, I set the number of target tasks equal to 2. In fact, multi-task Bayesian optimisation based on NSGA II cloud deal with at least 3 tasks at the same time, and if choosing NSGA III to help to find the Pareto frontier, it can handle 5 tasks at the same time. Figure 2[a] shows the Pareto frontier found by the algorithm when the number of Bayesian iterations is 1. The red dots represent the method using NSGA III, and the blue dots represent the method using NSGA II. Obviously, the convergence speed of NSGA III is faster. And when the number of iterations increases to 8, as shown in Figure 2[b], both methods have approached the true Pareto frontier.

[a]


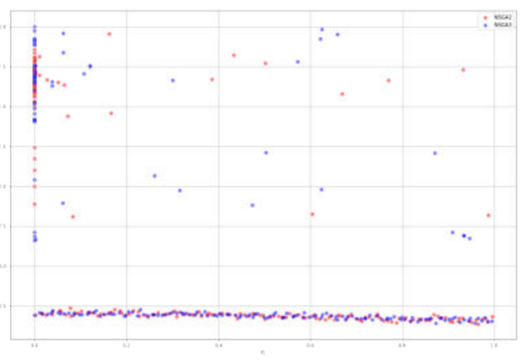
[b]


Figure 2: ZDT2

And in each run, the search path used by the optimization algorithm may be different, leading to some differences in terms of the time required or the number of evaluations. This partly depends on its starting conditions such as the number of the initial data. Take ZDT1 as an example, I only change the amount of initial data, the number of initial data for Figure 3[a] and

Figure 3[b] are 8 and 16. Because the initial data is too small and I do not use a large number of iterations, the algorithm cannot even find the prototype of the Pareto Frontier. And the number of initial data for Figure 3[c] and Figure 3[d] are 64 and 128. It is clear to find that when the number of iterations remains unchanged and the initial data set becomes larger, the arc of the Pareto front becomes clearer.

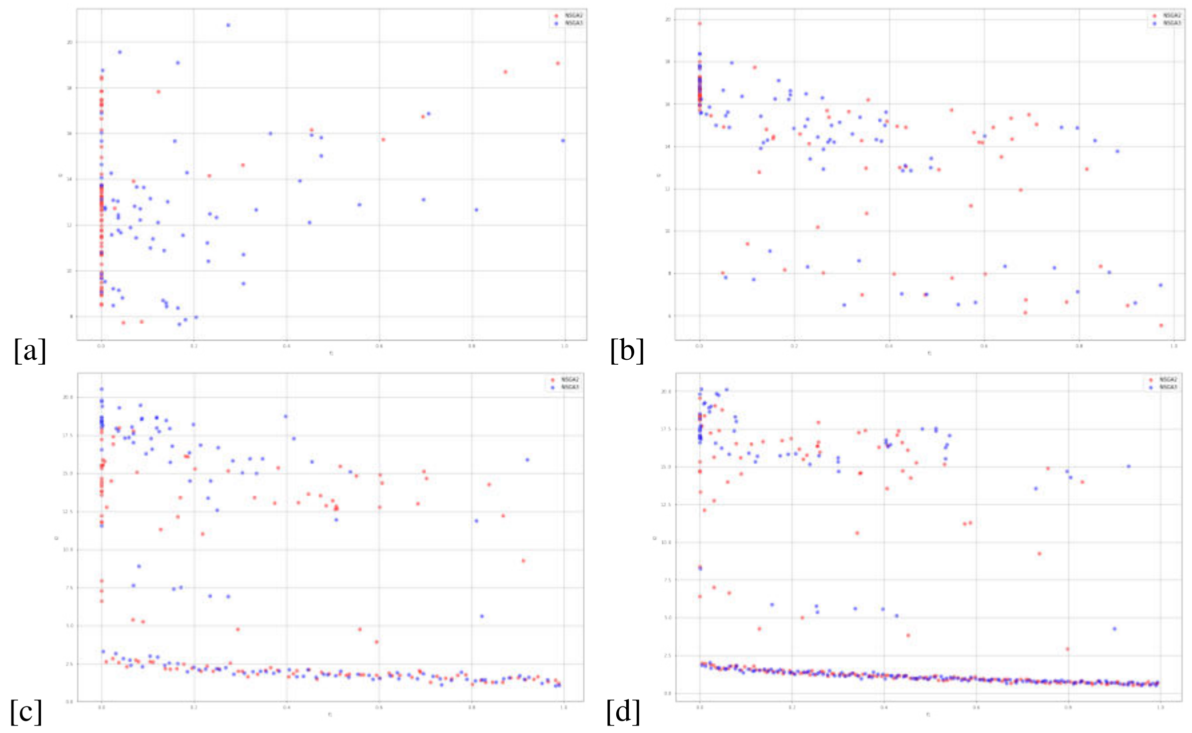


Figure 3: Pareto frontier of ZDT1 with increasing number of initial data

When the algorithm uses small initial data, even if the number of iterations increases, its convergence speed is still very slow. The iterations for Figure 4[a] and Figure 4[b] are 16 and 32 and the corresponding running time reached half an hour and an hour respectively. But as shown in Figure 4, they are still far away from the true Pareto frontier.

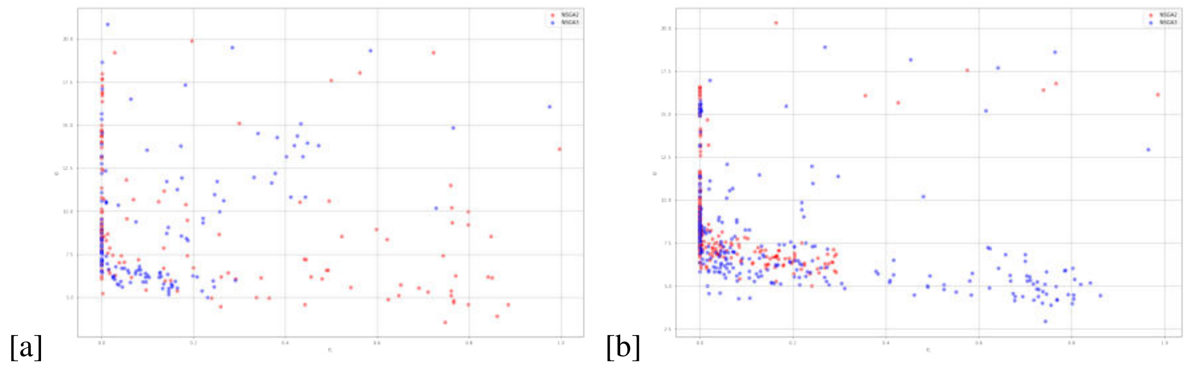
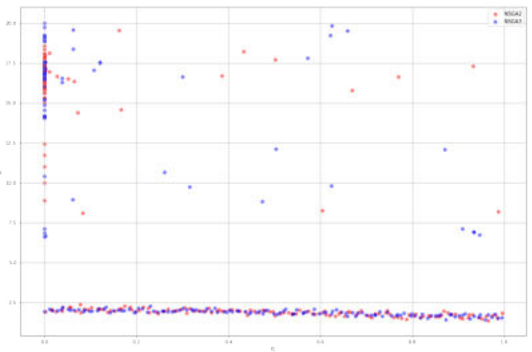
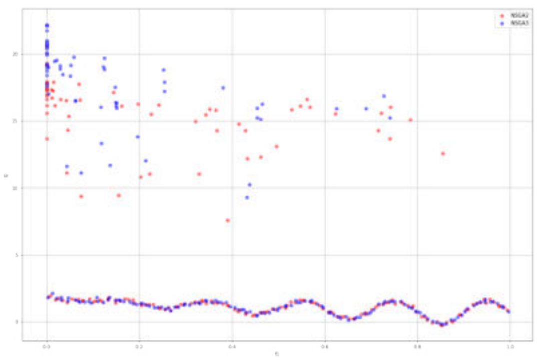
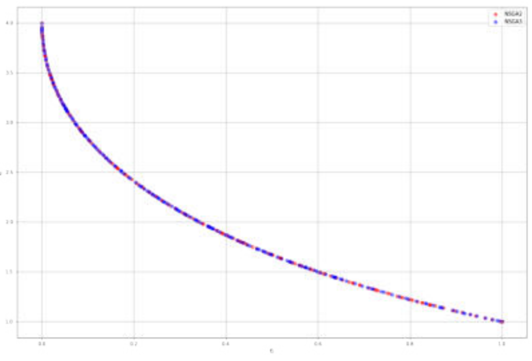


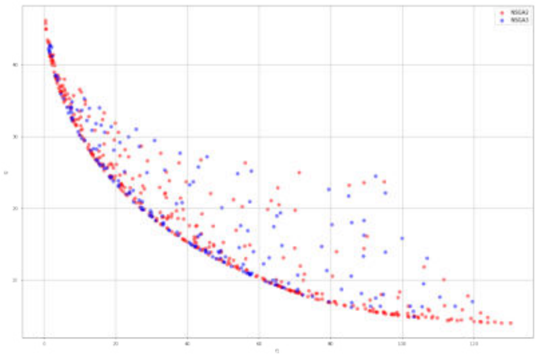
Figure 4: Pareto frontier of ZDT1 with increasing number of iterations

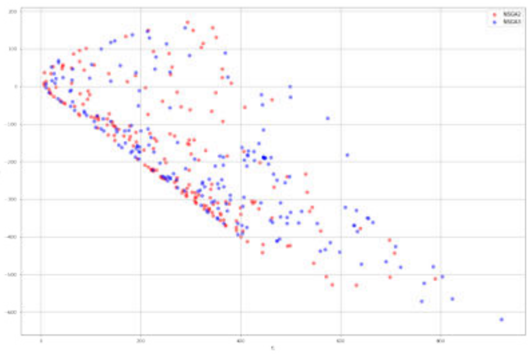
Next, I show the images of the Pareto frontier of 6 benchmark functions except ZDT1 in Figure 5. Here, I fix the maximum number of iterations of finding optimisation of likelihood equals to 5000. This is because most residuals of the loss function can meet or near the requirement (less than 1e-6) and using a larger maximum number of iterations is computationally expensive. And I set the number of initial data equals to 128, population size equals 50, maximum iteration of Bayesian optimisation equals 8. The benchmark functions used in Figure 6 from left to right and top to bottom are ZDT2, ZDT3, schaffer, binh-korn, chakong-haimes, osyczka-kundu. Clearly, the algorithm could find the Pareto frontier successfully using the first 3 benchmark functions, but not good at the last 3 benchmark functions. When the iterations of BO increases, the performance for Figure 5[d] has improved significantly while there is not much improvement for the last two functions. Due to the technical issue of my computer, I was unable to use a large number of iterations, but I could find that the predictive Pareto frontiers are approaching the true Pareto frontier. I am not sure whether my algorithm could get an optimal Pareto set, but I have to admit that my algorithm does not perform well in the last two benchmark functions which are too complex for my algorithm to deal with.


[a]


[b]


[c]


[d]


[e]

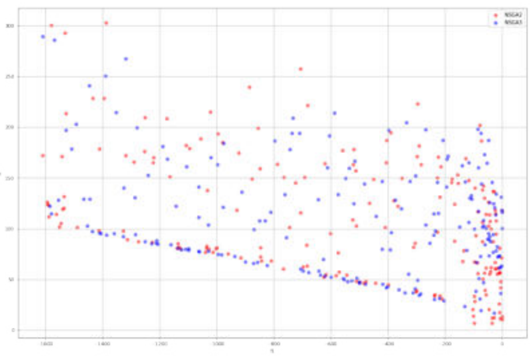

[f]

Figure 5: Pareto frontier of different benchmark functions

To test my second algorithm, I use a complex and synthetic test function Branin-Currin. The log hyper-volume difference is considered as an indicator for algorithm's performance, which measures the log difference between the hyper-volume of the true Pareto front and that of the approximate Pareto front given by the algorithm I use. Figure 6 shows the evolution of log hyper-volume differences with the increasing number of observations. The blue line represents the Sobol baseline and the yellow line represents another new proposed multi-task Bayesian optimisation algorithm qPAREGO which has a smaller workload(Daulton et al. 2020). In this case, my algorithm (blue line) still outperforms the qParEGO and Sobol baselines. And the short vertical bars represent the error bars. As seen from Figure 6, my algorithm has a relatively low variance for trials compared with the rest algorithms.

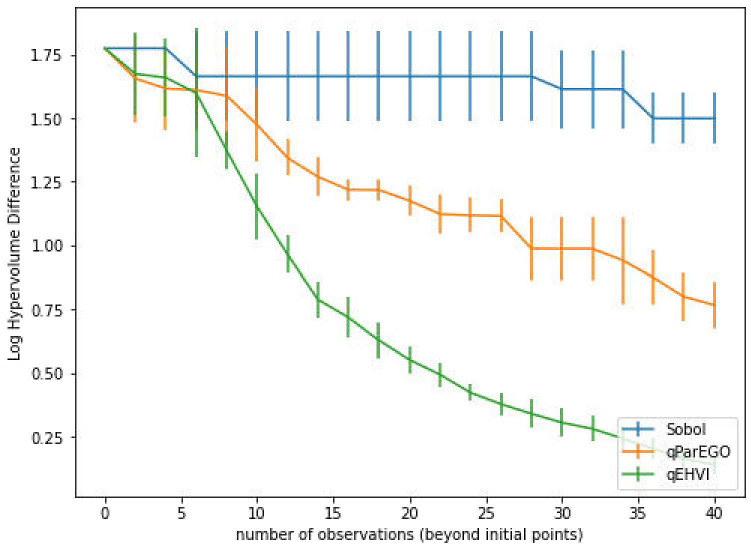


Figure 6: The log hyper-volume difference for different algorithms

Figure 7 plots the collected observations under each algorithm where the color corresponds to the BO iteration at which the point was collected. It is just another way to show the performance of algorithms. The plot on the right for qEHVI shows that the qEHVI quickly identifies the Pareto front and most of its evaluations are very close to the Pareto front. Both Figure 6 and Figure 7 demonstrate that qEHVI has a better performance than that of the qParEGO and Sobol.

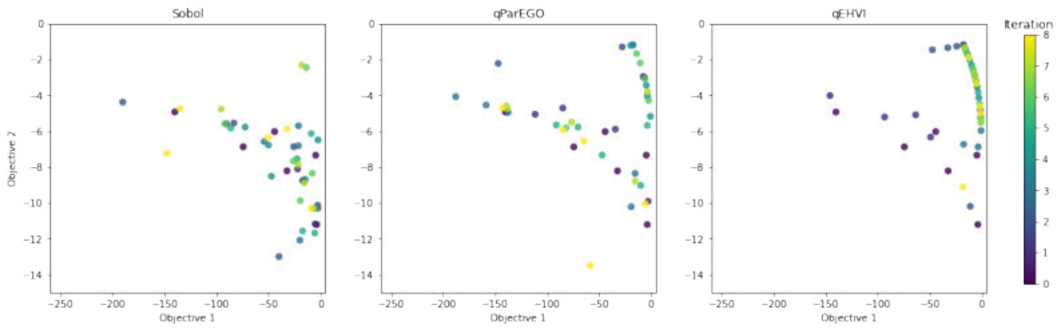


Figure 7: qHEVI

Figure 8 shows that if the batch size is large, it will converge slowly at the beginning, but after several iterations and the model gets more data, it will converge quickly and almost has the same performance compared to the algorithm without batch sampling at the end of the iteration.

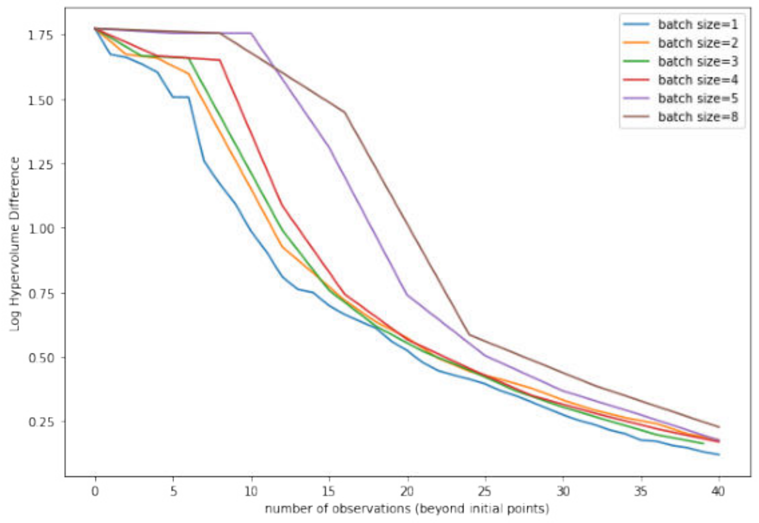


Figure 8: The evolution of log hyper-volume with different batch size

However, blindly increasing the size of batch size is not advisable. It may lead to a substantial increase in the amount of calculation and increasing the running time. This phenomenon is well proved in Figure 9, when the batch size is equal to 5 or larger, the running time increases dramatically. By that I mean, when the selected batch size is unreasonable, the algorithm using the batch algorithm may perform worse. To conduct this experiment, I keep the result of the product of the number of iterations and batch size constant, for example, if I choose batch size equals to 2, then I will choose the number of iterations equals to 20, and if batch size equals to 4, the number of iterations will be 10, in other words, I keep the number of the total observations constant.

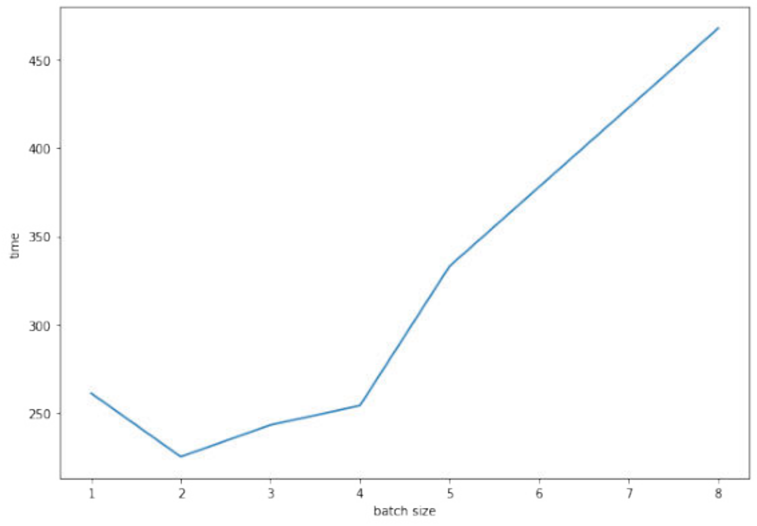


Figure 9: The running time against the batch size

As seen from Figure 9, it seems that choosing batch size equals to 2 or 3 is a better decision. In fact, the best choice of batch size may be affected by the number of cores the computer has. Because of technical issues of my computer, the computer I use to do this experiment only has two cores. Theoretically, if the computer has more cores, increasing batch size could have a better performance.

V CONCLUSIONS

In this paper, I introduce two multi-task Bayesian optimisation with batch sampling algorithms. The first algorithm combining batch Bayesian optimisation and NSGAs, which has relatively low computational complexity but still has good performance in benchmark functions. And it does not have a high requirement for computers, optimising is available even using a private laptop. The second algorithm using a new proposed acquisition function qHEVI, HEVI has high accuracy at the same time suffers from high computational overhead, batch sampling with a parallel approach on a computer with multiple cores tends to reduce the high computational complexity. And the result shows this algorithm indeed performs well in dealing with complex synthetic functions and high dimensional input. However, there are still many aspects that can be improved.

In Bayesian optimization, the selection of the appropriate surrogate model is more critical than the selection of the acquisition function. My algorithm still uses the exact Gaussian process which will meet the problem of the curse of dimensionality. Replacing the Gaussian process by neural networks or using deep learning might be helpful. However, finding an appropriate set of hyper-parameters of neural network or deep learning might be as difficult as finding the global optimization of your target tasks. At present, no surrogate model applies to all cases, and specific analysis is still needed for specific problems. It is best to use the experience and knowledge of experts to guide the choice of models.

When dealing with a large data set, using approximation techniques instead of Gaussian processes might also be a good way. A reduced-rank approximation method SPGP (sparse Gaussian process) proposed by (Snelson & Ghahramani 2006) introduced pseudo-inputs $m < t$, This method makes the rank of the co-variance matrix approximately reduced to m, thereby significantly reducing the amount of calculation to solve the inverse of the co-variance matrix. After that, the Monte Carlo method is used to sample to obtain an approximate co-variance function. But this method is also prone to overfitting when the data set is small. Although this approximate methods can speed up the efficiency of the solution, they might lose solution accuracy.

And there are some problems need to be solved, the first one is the cold start problem. Both two algorithms still need a relatively large number of initial data set. Another problem is for the first algorithm, it still can not deal with more tasks and enable to address the input with high dimensions. As for the limitation for the second algorithm, observations are assumed to be noiseless, which means it might not have an ability to address the real-life problem. And due to technical issues of my computer, I can not do a further experiment using more iterations and a larger data set and batch size. If possible, those will be my future work.

References

Brochu, E., Cora, V. M. & De Freitas, N. (2010), 'A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *arXiv preprint arXiv:1012.2599* .

Daulton, S., Balandat, M. & Bakshy, E. (2020), 'Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization', *arXiv preprint arXiv:2006.05078* .

Deb, K. & Jain, H. (2013), 'An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints', *IEEE transactions on evolutionary computation* **18**(4), 577–601.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002), 'A fast and elitist multiobjective genetic algorithm: Nsga-ii', *IEEE transactions on evolutionary computation* **6**(2), 182–197.

Dong, H., He, J., Huang, H. & Hou, W. (2005), A mixed mutation strategy evolutionary programming combined with species conservation technique, *in* 'Mexican International Conference on Artificial Intelligence', Springer, pp. 593–602.

Eckart, Z., Marco, L. & Lothar, T. (2001), 'Improving the strength pareto evolutionary algorithm for multiobjective optimi-zation', *EUROGEN, Evol. Method Des. Optim. Control Ind. Problem* pp. 1–21.

Emmerich, M. (2005), 'Single-and multi-objective evolutionary design optimization assisted by gaussian random field metamodels', *Dissertation, LS11, FB Informatik, Universität Dortmund, Germany* .

Feliot, P., Bect, J. & Vazquez, E. (2017), 'A bayesian approach to constrained single-and multi-objective optimization', *Journal of Global Optimization* **67**(1-2), 97–133.

Ginsbourger, D., Le Riche, R. & Carraro, L. (2010), Kriging is well-suited to parallelize optimization, *in* 'Computational intelligence in expensive optimization problems', Springer, pp. 131–162.

González, J., Dai, Z., Hennig, P. & Lawrence, N. (2016), Batch bayesian optimization via local penalization, *in* 'Artificial intelligence and statistics', pp. 648–657.

Groves, M. & Pyzer-Knapp, E. O. (2018), 'Efficient and scalable batch bayesian optimization using k-means', *arXiv preprint arXiv:1806.01159* .

Hennig, P. & Schuler, C. J. (2012), 'Entropy search for information-efficient global optimization', *The Journal of Machine Learning Research* **13**(1), 1809–1837.

Hernández-Lobato, J. M., Hoffman, M. W. & Ghahramani, Z. (2014), Predictive entropy search for efficient global optimization of black-box functions, *in* 'Advances in neural information processing systems', pp. 918–926.

Hoffman, M. D., Brochu, E. & de Freitas, N. (2011), Portfolio allocation for bayesian optimization., *in* 'UAI', Citeseer, pp. 327–336.

Jain, H. & Deb, K. (2013), 'An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: handling constraints and extending to an adaptive approach', *IEEE Transactions on evolutionary computation* **18**(4), 602–622.

Luo, C., Shimoyama, K. & Obayashi, S. (2014), Kriging model based many-objective optimization with efficient calculation of expected hypervolume improvement, *in* '2014 IEEE Congress on Evolutionary Computation (CEC)', IEEE, pp. 1187–1194.

Marmin, S., Chevalier, C. & Ginsbourger, D. (2015), Differentiating the multipoint expected improvement for optimal batch design, *in* 'International Workshop on Machine Learning, Optimization and Big Data', Springer, pp. 37–48.

Martinez-Cantin, R., Tee, K. & McCourt, M. (2018), Practical bayesian optimization in the presence of outliers, *in* 'International Conference on Artificial Intelligence and Statistics', PMLR, pp. 1722–1731.

Scott, S. L. (2010), 'A modern bayesian look at the multi-armed bandit', *Applied Stochastic Models in Business and Industry* **26**(6), 639–658.

Sculley, D. (2010), Web-scale k-means clustering, *in* 'Proceedings of the 19th international conference on World wide web', pp. 1177–1178.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & De Freitas, N. (2015), 'Taking the human out of the loop: A review of bayesian optimization', *Proceedings of the IEEE* **104**(1), 148–175.

Shahriari, B., Wang, Z., Hoffman, M. W., Bouchard-Côté, A. & de Freitas, N. (2014), 'An entropy search portfolio for bayesian optimization', *arXiv preprint arXiv:1406.4625* .

Snelson, E. & Ghahramani, Z. (2006), Sparse gaussian processes using pseudo-inputs, *in* 'Advances in neural information processing systems', pp. 1257–1264.

Swersky, K., Snoek, J. & Adams, R. P. (2013), Multi-task bayesian optimization, *in* 'Advances in neural information processing systems', pp. 2004–2012.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q. & Wilson, A. G. (2019), Exact gaussian processes on a million data points, *in* 'Advances in Neural Information Processing Systems', pp. 14648–14659.

Williams, C. K. & Rasmussen, C. E. (2006), *Gaussian processes for machine learning*, Vol. 2, MIT press Cambridge, MA.

Wu, J. & Frazier, P. (2019), Practical two-step lookahead bayesian optimization, *in* 'Advances in Neural Information Processing Systems', pp. 9813–9823.

Yang, K., Emmerich, M., Deutz, A. & Bäck, T. (2019), 'Multi-objective bayesian global optimization using expected hypervolume improvement gradient', *Swarm and evolutionary computation* **44**, 945–956.