1 Introduction

The core issue of interactive design is to search for a point in a usually large design space. Previous studies have been looking into searching strategies within the interaction. In this report, we consider how previous records of interactions can help future searching.

Recall that an interaction can be considered as a sequence of transitions of states. A state is described by the current sampled points \( X \) and the corresponding labels \( y \). At each state, the searching algorithm determines where to sample next and the user responds by labeling these new points. This completes the transition to a new state. There are rewards associated with transitions: a transition is preferred if it changes the preferred points. Just like we prefer algorithms with high convergence rate in optimization, here we prefer transitions with new best points.

Learning from previous interactions can help to achieve this goal in several possible ways. This study examines the most straightforward approach.

If the searching algorithm is deterministic (SVM-KFF, see previous report) and the initial samples are fixed, then the state space is finite. Further, with different objective functions (preferences) of users, each interaction will generate a path and all paths belong to a tree. Let \( n \) be the sample size in each iteration and \( l \) be the maximum iteration number. Each node of the tree will have \( 2^n \) descendant nodes. The total number of nodes of the tree is approximately \( 2^{nl} \). Inevitably there exists a trade-off: increasing \( n \) and \( l \) helps more accurate searches but will also make the tree untrackable. However, it is reasonable to assume that the optimal points from different people are not spread widely in the design space but instead form clusters. Thus building a tree \textit{en route} helps to reveal the paths that most interactions will cover. With this tree at hand, an Ad Hoc method to increase future convergence is to always skip the states that are recorded and has has only one branch, see Figure 1.

2 Simulations

We demonstrate the proposed method using two simulations: a Gaussian function with its optimum uniformly distributed in \( \{(x, y) \mid x \in [0.25, 0.75], \ y \in [0.25, 0.75]\} \), and a Branin function with its three optima moving in the box of size 1. In each case, 400 interactions are run to build the tree. Figure 2 and 3 show the comparisons between mean errors from 10 experiments of pure interactions (SVM-none) and that of interactions with off-line learning (SVM-learn).
Figure 1: We jump from state $i$ to state $j$ if there is only one branch on any state (node) in between.

3 Technical discussion

3.1 Searching algorithm

The searching algorithm used in this study is SVM-KFF. It samples the kernel farthest away points from the current sampled points within the boundary of the preferred points. This algorithm is theoretically deterministic, but finding the kernel-farthest points may involve an optimization subroutine with stochastic nature. This, however, will not affect the training since for each recorded state, new samples generated from this state are also recorded and can be used directly.

3.2 Other heuristics based on the tree

Like any learning method, the trade-off between exploitation and exploration exists. The heuristic we used in this study is purely exploitation (Interaction without previous knowledge is pure exploration). It is useful when the tree is thoroughly built. If this is not the case, jumping to a conclusion may be risky. Other heuristics can be proposed to have both exploitation and exploration features, e.g. jumping a certain distance rather than straight to the end.
Figure 2: Simulation on Gaussian

Figure 3: Simulation on Branin