Sample-and-Bound for Non-convex Optimization

Yaoguang Zhai*, Zhizhen Qin*, Sicun Gao

University of California, San Diego
yazhai@ucsd.edu, zhizhenqin@ucsd.edu, sicung@ucsd.edu

Abstract

Standard approaches for global optimization of non-convex functions, such as branch-and-bound, maintain partition trees to systematically prune the domain. The tree size grows exponentially in the number of dimensions. We propose new sampling-based methods for non-convex optimization that adapts Monte Carlo Tree Search (MCTS) to improve efficiency. Instead of the standard use of visitation count in Upper Confidence Bounds, we utilize numerical overapproximations of the objective as an uncertainty metric, and also take into account of sampled estimates of first-order and second-order information. The Monte Carlo tree in our approach avoids the usual fixed combinatorial patterns in growing the tree, and aggressively zooms into the promising regions, while still balancing exploration and exploitation. We evaluate the proposed algorithms on high-dimensional non-convex optimization benchmarks against competitive baselines and analyze the effects of the hyper parameters.

Introduction

Non-convex global optimization problems are pervasive in engineering (Mistakidis and Stavroulakis 2013; Campi, Garatti, and Ramponi 2015), computer science (Liu and Lu 2014; Jain, Kar et al. 2017), and economics (Bao and Mordukhovich 2010). The problem is well-known to be NP-hard, and the practical challenge lies in distinguishing the global optimum from exponentially many potential local optima (Jain, Kar et al. 2017; Yang 2019).

Existing approaches to non-convex optimization can be largely categorized into sampling-based methods and tree-search methods. Sampling-based approaches, such as simulated annealing (SA) (Henderson, Jacobson, and Johnson 2003) and cross-entropy (CE) (De Boer et al. 2005), explore the solution space through random sampling and guided search strategies with the minimal assumptions about the objective function. Sampling methods can be designed to asymptotically converge towards the global optimum, but suffer from the curse-of-dimensionality in practice. Tree search and interval-based optimization methods (Gurobi Optimization 2023; Ninin 2016) leverage various branch-and-bound techniques that maintain a partition tree over the domain to systematically prune the space towards global optima. Such algorithms typically employ rigorous techniques (e.g., linear relaxation (Yanover et al. 2006) and interval arithmetic (Hickey, Ju, and Van Emden 2001; Araya and Reyes 2016)) for bounding the functions and systematically explore the solution space. The size of the search tree can quickly become exponential in the number of dimensions and is the major bottleneck for scaling up.

We propose an approach that combines the benefits of sampling-based and tree-based approaches as well as interval bounding and local optimization techniques, by taking advantage of the recent progress in Monte Carlo Tree Search (MCTS) methods. We assume that the analytic form of the objective function is known over a compact domain, so that we can use interval bounding (Araya and Reyes 2016) on the function value and its local first-order and second-order information in different parts of the MCTS design. A key feature of the Monte Carlo trees is that the growth of the tree is driven by samples rather than partitions, and hence the name *Sample-and-Bound*. By associating the analytic and estimated properties of adjustable neighborhoods around each sample, we design the MCTS algorithm to best balance exploration and exploitation based on the important numerical properties of the objective function. We evaluate the proposed algorithms against a wide range of existing algorithms and analyze the importance of various hyper parameters.

Related Work

Some classical approaches to global optimization explore the search space by sampling without explicitly building models of the objective function. Common techniques in this category include stochastic methods such as SA (Henderson, Jacobson, and Johnson 2003) and CE (De Boer et al. 2005), as well as deterministic schemes like Nelder-Mead (NM) (Gao and Han 2012). SA (Henderson, Jacobson, and Johnson 2003) uses a probability-driven search process to escape local minima. CE (De Boer et al. 2005), on the other hand, is a technique that iteratively updates the probability distribution on the search space to look for optimal regions. NM method (Gao and Han 2012) is a deterministic sampling approach, which maintains a simplex within the search space and updates its vertexes based on evaluations at selected points.

Sampling-based approaches have recently been combined

*These authors contributed equally.

with tree search by building a search tree for the state space and pick only the most promising subspace to sample. Existing algorithms include Deterministic Optimistic Optimization (DOO) (Munos 2011), Latent Action Monte Carlo Tree Search (LaMCTS) (Wang, Fonseca, and Tian 2020a), and Monte Carlo Tree Descent (MCTD) (Zhai and Gao 2022). DOO segments the search domain into sections, each represented by a point; and the new sample is carefully selected by choosing the most suitable section. LaMCTS method employs MCTS to manage search space partitioning. It learns latent actions to distinguish good and bad regions in the search space, and samples in the good partitions during its tree's expansion. MCTD assumes the objective function as black-box, utilizes a combination of sampling based approach and learning based approach for local optimization, and employes MCTS to select the best local optimization processes. Although these methods have adeptly laid out a comprehensive framework for navigating the search space, the task of identifying the most promising subspace from the sample data remains a challenge.

Another category of global optimization methods require the access to the formulation and rely on precise anticipation of objective function values within predefined regions. They employ the branch-and-bound algorithms in which they constitutes a robust framework that systematically partitions the solution space into more accessible sub-regions referred to as branches. The evaluation of each branch is made according to its potential to outperform the current optimal solution based on the bounding of objective function intervals specific to that branch. As the algorithm advances, it tactically prunes branches that can be definitively identified as incapable of providing a superior solution. The typical solvers for this category are BARON (Tawarmalani and Sahinidis 2005; Sahinidis 2023) and Gurobi (Gurobi Optimization 2023). BARON (Tawarmalani and Sahinidis 2005; Sahinidis 2023) is explicitly tailored to address non-convex global optimization problems by strategically exploring the solution space. Its purpose is to either uncover globally optimal solutions or provide verified lower bounds for the optimal objective value. It achieves this through accurate bounding of non-linear terms with several exceptions such as trigonometric functions and min/max functions. Gurobi (Gurobi Optimization 2023) is a widely used commercial optimization solver famous for its proficiency in handling quadratic programming problems and various optimization scenarios.

Preliminary

Problem Formulation. We consider the problem of minimizing an objective function $f(x) : \Omega \to \mathbb{R}$, where the domain $\Omega \subseteq \mathbb{R}^n$ is compact. In our approach, we assume that we have access to the analytical form of the function $f(x)$, enabling us to query its first-order derivative vector $G(x) = f'(x)$, the second-order partial derivative Hessian matrix $H(x) = f''(x)$, and evaluate the function value interval $f(B)$ over a specified input box $B \subseteq \Omega$.

Interval Arithmetic. Interval computation is a mathematical and computational approach that operates on quantities and variables represented as intervals (Alefeld and Herzberger 2012). In this context, for a function f defined on an input box domain B, the value of $f(B)$ is expressed as an interval $[lb, ub]$, where for every x within B, the function value satisfies the inequality $lb \leq f(x) \leq ub$.

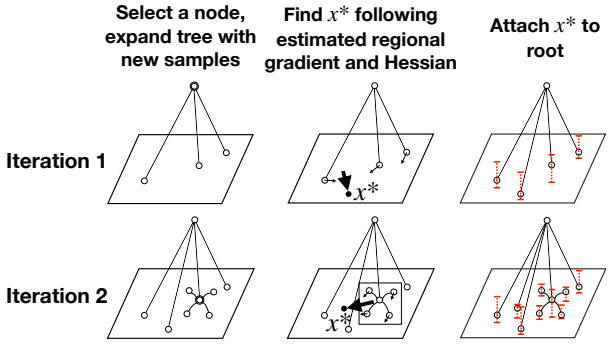


Figure 1: Steps in each iteration of the MCIR algorithm

Monte Carlo Tree Search. MCTS effectively balances exploration and exploitation based on the theory of multi-armed bandits. The MCTS framework consists of four main steps: Selection, Expansion, Simulation, and Backpropagation. During the Selection step, the search tree is traversed from the root node to a leaf node. The Upper Confidence Bound for Trees (UCT) value, defined as Eq. 1, is used to select the best child of a parent node:

$$\nu(n_i) = \frac{R_i}{N_i} + C \cdot \sqrt{\frac{2 \cdot \ln(N_p)}{N_i}} \tag{1}$$

Here, R_i represents the rewards obtained on child node n_i, N_i is the number of visits to n_i, N_p is the number of visits to n_i's parent node n_p, and C is a constant that balances exploration and exploitation. From the root node, the algorithm recursively select the child node with the highest UCT value, until a leaf node is reached. During the Expansion step, a new child node is added to the selected leaf node. In the Simulation step, a random simulation is performed from the newly added child node until a terminal node is reached, and the simulation reward is estimated. Finally, in the Backpropagation step, the simulation reward is propagated backward from the expanded node to the root node, whose statistics are updated accordingly.

Monte Carlo Tree Search with Interval Bounds and Regional Estimation

Overview. The pseudocode of MCIR is provided in Alg. 1, and a high-level visualization is depicted in Fig. 1. Our MCIR algorithm employs a search tree structure constructed based on collected samples of the objective function and follows a systematic searching approach in each iteration. Each node in the tree contains a batch of samples encompassed within a box domain associated with that node. In every iteration, we select a leaf node n_p using a modified UCT formula with function evaluation on the box, and we expand the selected node by adding new child nodes n_{ci}

generated from random sampling (Fig. 1 (a)). We also identify another new child node n^* by leveraging the regional estimation based on gradient and Hessian within the neighborhood of the selected node n_p (Fig. 1 (b)). The node n^* represents a node superior to the selected n_p, and we attach it with the root node. This attachment allows us to prioritize the search on this node, thereby accelerating the search process (Fig. 1 (c)). For each newly created node we run local optimization with limited steps to enhance the quality of the best-found sample on it.

Note that despite the special design of different parts of MCTS for the optimization context, the proposed algorithm ensures non-zero probability of sampling any neighborhood with positive measure in the input space. Consequently, MCIR is complete in the sense that it will eventually find an ε-neighborhood around the optimal value of any continuous function with arbitrarily small positive ε.

Sub-domain Marking. Samples are the primary information in each node of the search tree that our algorithm build. Around each sample, we mark up the subdomain around it that is considered at the node. The subdomain, typically a hyperbox, will be the focus of local search and optimization at the node, for determining the value of a node. We use the notation B_i to denote the box subdomain associated with the node n_i. In the first iteration, the root node n_{root} encompasses the entire search space, $B_{root} = \Omega$, with the function's lower bound on B_{root} denoted as $lb_{root} = lb(f(B_{root}))$, and its box volume V_{root} represented in logarithmic scale. For subsequent iterations, box B_i is assigned to a node n_i, while lb_i and V_i will be updated according to formulas to be described below. To compute the lower bound of the objective function within a specified input box domain efficiently using global interval bounding (Ninin 2016).

Path Selection. The key to our design is a modified UCT formula that considers both exploration and exploitation. The pseudocode of this procedure can be found in the **SELECT** function in Alg. 1. For each child node n_{ci} with $i = 1, ...,$ and its parent node n_p, the UCT value $u(n_{ci})$ is determined by the following equation:

$$u(n_{ci}) = -y^*_{ci} - C_{lb} \cdot lb_{ci} - C_v \cdot V_{ci} + C_x \cdot \sqrt{\frac{\log N_p}{N_{ci}}} \quad (2)$$

In this formula, C_{lb}, C_v and C_x are weights for the function's lower bound, the volume of the box, and visitation-based exploration, respectively. The variables N_p and N_{ci} denote the number of visits to the parent node n_p and the child node n_{ci}. y^*_{ci} indicates the current best function value discovered on the node n_{ci}, and lb_{ci} corresponds to the lower bound of the function's interval value on the node n_{ci}. The term V_{ci} is the volume (in logarithmic scale) of the box where the lower bound is identified. It is worth noting that after the creation of new child nodes, the function lower bound lb_p and the box volume V_p on the parent node n_p can be updated, as detailed in the subsequent section.

This formulation takes into account the following factors to balance exploration and exploitation: (1) the best function value observed within the box domain, (2) the lower bound of the function value within the domain from interval computation, which reflects the potential best function value upon further exploitation, (3) the volume of the box where the lower bound is determined, related to the reliability of the function lower bound prediction, and (4) the frequency of node visitation. While we considered other ingredients - such as upper function value bound, or values from leveraging the function's analytical form - to put into the formula, the design in Eq. 2 turns out to be the most effective.

Utilizing Eq. 2, our algorithm tends to redirect its attention to probe alternative sub-domains when a local optimum is identified. When a box is tight enough, the variance of the objective function in the box is low, so the identified local optimum within the box lb_{ci} is relatively accurate.

If this lb_{ci} is close to the minimum of all other $lb_{ci'}$, it indicates a near-optimal solution has been identified. Conversely, if an $lb_{ci'}$ exists that is substantially lower than the current lb_{ci}, the search scheme leans towards selecting the node with the lower $lb_{ci'}$ value in the subsequent iteration due to the path selection criterion Eq. 2. In summary, Eq. 2 within our algorithm helps strike a balance between exploiting the neighbor region around the current best-found point and exploring other domains that might contain lower function values.

Tree Expansion. In our algorithm, we utilize two steps to expand the tree effectively. The first step involves sampling within the box of the parent node and generating new child nodes based on these chosen samples. The second step emphasizes learning a high-quality sample point by leveraging both global Hessian and local gradients.

After selecting the leaf node, we proceed to exploit the function space by sampling and creating a cluster of child nodes within the corresponding box (**EXPAND** in Alg. 1). To ensure comprehensive coverage, we divide the box B_p into distinct subsets B_{ci} for each child node n_{ci}, satisfying $\cup\{B_{ci}\} = B_p$ and $B_{ci} \cap B_{cj} = \emptyset, i \neq j$. Additionally, local optimization may be applied to each individual child node n_{ci} to improve sample quality. When a child node n_{ci} is created, we ascertain its function lower bound lb_{ci} through interval propagation of the corresponding box B_{ci}: $lb_{ci} = lb(f(B_{ci}))$. Once the cluster of child nodes is created and their boxes fully cover the parent box B_p, we update the lower bound on the parent node $lb_p = \min(lb_{ci}) = lb_{cj}$ and the volume of the associated box $V_p = V_{cj}$, where $i = 1, .., j, ...$ This update will be propagated to the root node.

Next, we learn a representative node n^* using the current set of samples n_{ci} from the selected node n_p, as outlined in Alg. 1 **LEARN**. This step is performed by computing the diagonal of the Hessian matrix, $diag(H)$, for each child node n_{ci}, and estimating the expected value. Furthermore, we collect the gradient information G around the best sample of $n_{ci}, i = 1, ...$ and perform a step of Newton's method (or gradient decent when Newton's method is not conductive to minimization), starting from the best sample. The average Hessian, derived from the broad region within the box, represents the overall curvature characteristics of the box. By integrating locally-averaged gradient information, we can identify a sub-region within the box that is more likely to encompass a minimum. The learned representative node, de-

Algorithm 1: Monte Carlo Tree Search with Interval Bounds and Regional Estimation (MCIR)

function MCIR(objective: f, domain: Ω)
  $n_0 \leftarrow None$ ▷ root without sample
  $B_0 \leftarrow \Omega, lb_0 \leftarrow f(\Omega), V_0 \leftarrow V(B_0)$
  for step = $1, ..., t$ **do**
    $n_p \leftarrow$ Select(n_0)
    EXPAND(n_p)
    LEARN(n_p)
    BACKUP(n_p)
  end for
  return y_0^*
end function

function SELECT(node: n)
  $n_p \leftarrow n$
  while n_p has children **do**
    for $n_{ci} \in n_p$.children **do**
      $u(n_{ci}) \leftarrow u(y_{ci}^*, lb_{ci}, V_{ci})$ by Eq. 2
    end for
    $j \leftarrow \text{argmax}_i\, u(n_{bi})$
    $n_p \leftarrow n_{cj}$
  end while
  return n_p
end function

function BACKUP(node: n_p)
  if n_p has children **then**
    $\{n_{ci}\} \leftarrow n_p$.children
    $y_p^* = \min_i y_{ci}^*$
    $j = \text{argmin}_i\, lb_{ci}$
    $lb_p = lb_{cj}$
    $V_p = V_{cj}$
  end if
  BACKUP(n_p.parent)
end function

function EXPAND(node: n_p)
  $B = B_p$
  while $B \neq \emptyset$ **do**
    $x_{ci} \leftarrow x \in B$
    $x_{ci} \leftarrow$ LocalOpt (x_{ci}, B_{ci})
    $n_{ci} \leftarrow x_{ci} \in B$
    $B_{ci} \leftarrow b \in B, x_{ci} \in b$
    $lb_{ci} \leftarrow f(B_{ci})$
    $V_{ci} \leftarrow V(B_{ci})$
    $B \leftarrow B \backslash B_{ci}$
    n_p.children.append(n_{ci})
  end while
end function

function LEARN(node: n_p)
  $n_{ci} \leftarrow n_p$.children
  $\overline{H} \leftarrow \overline{Hessian(x_{ci})}, i = 1, ...$
  $j \leftarrow \text{argmin}_i\, (y_{bi})$
  $\{x'\} \leftarrow x$, for $|x - x_{cj}| < \delta$
  $\overline{G} = \overline{grad(x')}$
  for $d = 1, ..., dims$ **do**
    if $\overline{H}_{dd} > 0$ **then**
      $x_d^* = x_{cj,d} - \overline{G_d}/\overline{H_{dd}}$
    else
      $x_d^* = x_{cj,d} - \overline{G_d}$
    end if
  end for
  $n^* \leftarrow x^*$
  $B^* \leftarrow B_{cj}$, centered at x^*
  $lb^* \leftarrow f(B^*), V^* \leftarrow V(B^*)$
  $x^* \leftarrow$ LocalOpt (x^*, B^*)
  n_0.children.append(n^*)
end function

noted as n^*, is attached to the root node. Note that this attachment means the root node can have children nodes n_{ci} and n_{cj} where $B_{ci} \cap B_{cj} \neq \emptyset$. This step grants n^* higher priority in subsequent iterations. Such prioritization promises to guide the search toward a favorable region, thereby reducing unnecessary tree expansion and preserving tree manageability. Considering that this step may expand the tree's first level of children in every iteration, an extra step may be taken to evaluate the quality of the newly learned node and prune unnecessary ones.

Local Optimization. Upon creating a child node, we have the option to conduct local optimization steps to improve the quality of the samples on the node. While this step is not obligatory, it offers a beneficial opportunity to refine the samples on each node. To ensure efficient execution, the number of optimization steps is typically kept at a low value, preventing over-exploitation of the immediate local neighborhood. Local optimization can utilize a variety of numerical optimization algorithms. Since the representative node has already been learned using second-order information, we make quasi-Newton methods such as L-BFGS-B (Byrd et al. 1995; Zhu et al. 1997) the default choice for local optimization. To ensure computational efficiency, we limit the number of function evaluations during the local optimization. In most cases we cap the number of iterations at fewer than 50, as we do not want to overemphasize the choice of the local optimizer. It is worth mentioning that alternative local optimization algorithms can be employed based on specific requirements and preferences.

Backward Propagation. After creating and locally optimizing children nodes n_{ci} and n^*, we back propagate three important values upwards as in Alg. 1 **BACKUP**, to enhance efficient exploration and decision-making in the subsequent steps.

Firstly, we update the best function value y_{ci}^* found on the child node n_{ci}, to the parent node n_p with y_p^*. This ensures that the parent node retains the most optimal function value discovered within its subtree. Secondly, we update the lower bound of the function interval value lb_p on the parent node n_p with $lb_p = \min(lb_{ci})$. Given that the newly created child nodes comprehensively cover the box of the parent node, this update provides more precise informa-

tion guiding the search towards the global minimum. Lastly, we propagate the size of the box V_{ci} from which the lower bound of the function value originates: $V_p = V_{cj}$, where $j = \text{argmin}_i\, lb_{ci}$. This box size represents the uncertainty in the input search space concerning the approximated function interval value. The same propagation is applied to the node n^*, even though its parent is the root node.

Experiments

Benchmarks. To evaluate the performance of our algorithms, our benchmark sets include three distinct categories: synthetic functions designed for nonlinear optimization, bound-constrained non-convex global optimization problems derived from real-world scenarios, and neural networks fitted for single valued functions. It is important to note that our approach relies on having access to the symbolic expression of the objective function and do not consider other relational constraints to the variables (e.g., "<="). As a result, benchmark sets that are commonly used for black-box optimization problems and constraint optimization problems are not applicable in our case.

Synthetic functions are widely-used in nonlinear optimization benchmarks (Lavezzi, Guye, and Ciarcià 2022). These functions usually have numerous local minima, valleys, and ridges in their landscapes which is hard for normal optimization algorithms. In our tests, we choose three functions: Levy, Ackley, and Michalewicz, and examine our algorithm's performances on the functions in 50d, 100d, and 200d. For our evaluation of non-convex global optimization problems in various fields, we select bound-constrained problems from the collection presented in (The Optimization Firm 2023; Puranik and Sahinidis 2017) that do not involve any additional inequality or equality constraints. To strike a balance between computational resources and the complexity of the function landscapes, we specifically select functions with input dimensions between 30 and 1000, and ensure that the functions could be evaluated within a reasonable time, considering the computational cost of computing the gradient and Hessian. The chosen functions for our evaluation include Biggsbi1 (1000d), Harkerp (100d), and Watson (31d). It is worth noting that this set of test functions is also utilized in the development of BARON (Tawarmalani and Sahinidis 2005) and continues to be used in the latest version (Sahinidis 2023). In addition to the aforementioned problems, we also explore the application of one-layer neural networks with ReLU activation functions fitted for specific objective functions. The nonlinearity introduced by activation functions and the partitioning of the input space pose challenges in finding the global minimum of neural networks. To assess the performance of our algorithm, we train neural networks with varying numbers of input dimensions and one layer of 16 hidden unite. We translate the entire network into an analytic expression form, enabling us to evaluate the algorithm's effectiveness in optimizing neural network models. We conduct our experiments on a local machine with Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 16G RAM, and NVIDIA GeForce GTX 1080 graphic card.

Baselines. We select various sampling-based global optimization algorithms as baselines for our experiments, including: basinhopping (Olson et al. 2012), differential evolution (Storn and Price 1997; Pant et al. 2020), dual annealing (Xiang et al. 2013), direct (Gablonsky and Kelley 2000), CMA (Hansen et al. 2023), TuRBO (Eriksson et al. 2019), LaMCTS (Wang, Fonseca, and Tian 2020b), and Gurobi (Gurobi Optimization 2023). It should be pointed out that some algorithms, including TuRBO and LaMCTS, are GPU-ready. However, due to the limitations of our computational resources, we refrain from using GPUs for the optimization process, except for tasks related to training and evaluating the neural network model. To compensate for the reduced performance from utilizing the CPU, we extend the timeout for TuRBO and LaMCTS to be five times of other baselines.

It is important to mention that we do not incorporate BARON (Sahinidis 2023) as one of our baseline methods, despite its renowned ability to efficiently bound boxes. The reason behind this decision lies in the fact that BARON can manage the functions present in their test sets during pre-processing, entirely eliminating the need to execute the optimization algorithm. For instances like Biggsbi1, Harkerp, and Watson, BARON can solve them instantly, requiring zero seconds and iterations. Moreover, BARON encounters challenges with certain function types, including but not limited to trigonometric functions and min/max functions (Sahinidis 2023). These types of functions are prevalent in synthetic test function sets as well as function sets based on neural networks.

Another consideration is that Gurobi requires expertise and extra effort to achieve peak performance. While Gurobi stands out as an exceptionally efficient and versatile optimization solver, especially in the context of non-convex optimization problems, it comes with certain prerequisites. Its handling of non-linear terms, for instance, treats them as General Constraints, which necessitates extra manual modification to the objective function expression, as outlined in (Gurobi Optimization 2023). This specific trait might limit our ability to deploy it on entire test sets.

Metrics. For each benchmark function, we conduct experiments using baseline algorithms and our proposed algorithm with five different random seeds. The time limits for the baselines are set to 2 hour. Due to CPU utilization, the limits for TuRBO and LaMCTS are extended to 10 hours. Throughout the experiments, we track the best-found function value until each step and compute the mean and standard deviation across all runs. This allows us to compare the final best-found values as well as the speed at which each algorithm converges to the optimal result.

Overall Performance. Fig. 2 presents performance comparisons between the MCIR algorithm and baseline algorithms on the synthetic function benchmark set. For the Ackley function (Fig. 2 first row) and Levy function (Fig. 2 second row), CMA emerges as the top-performing algorithm, followed by MCIR and dual annealing. For the Michalewicz function (Fig. 2 third row), dual annealing and MCIR delivers similar best performance in terms of final optimization result, while CMA fails to optimize efficiently. Notably,

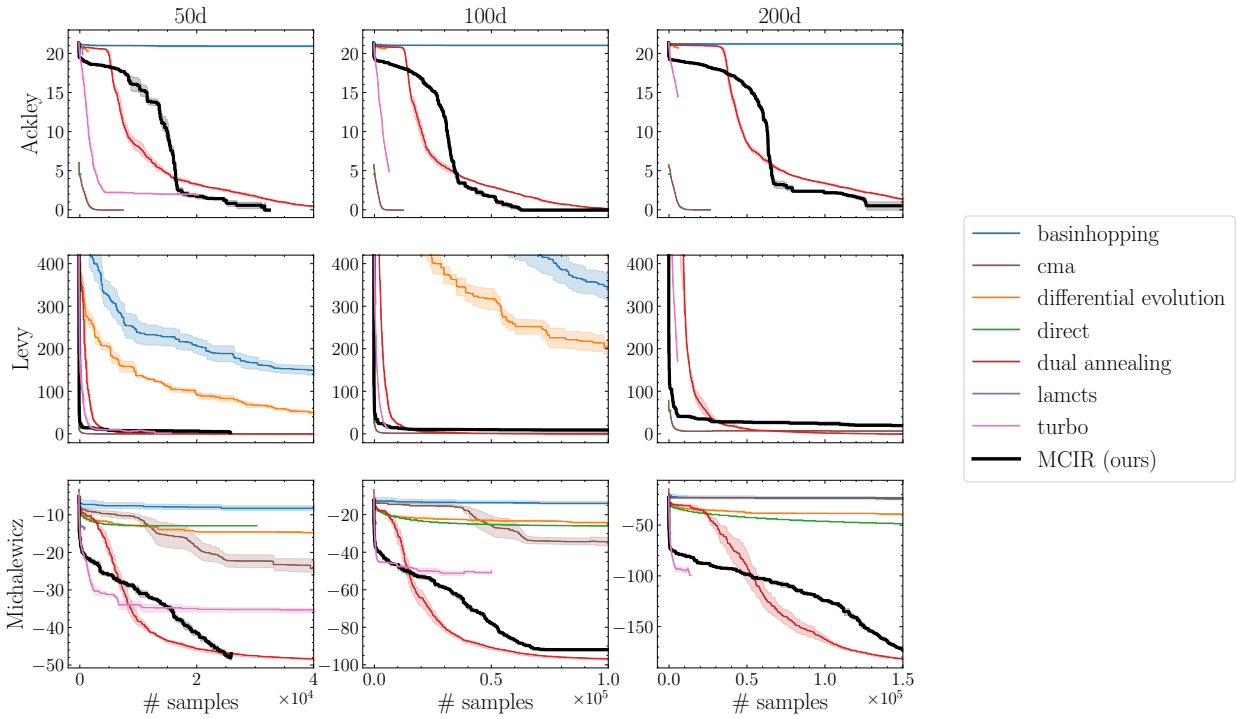


Figure 2: Overall performance of the baselines and MCIR on tested synthetic functions.

Gurobi consistently completes its optimization with just one function evaluation call. It attains a best function value of 2.02 on Ackley (for three dimensions), 0.0 for Levy in both 50d and 100d, and -0.00016 for Levy-200d (an anomalous value, because Levy function is greater or equal to 0.0). It also attains the lowest value amongst all algorithms on the Michalewicz function.

Fig. 3 offers an in-depth comparison between the MCIR algorithm and the baseline algorithms across benchmarks such as Biggsbi1, Harkerp, Watson, and Neural Networks. In the bound-constrained optimization problems (BCP) of Biggsbi1, Harkerp, and Watson (Fig. 3 first row), MCIR exhibits exemplary performance. It adeptly strikes a balance between exploring the search space and executing local optimization, culminating in the precise pinpointing of the global minimum from many suboptimal local minima derived from local optimization. CMA shows a performance comparable to MCIR on Harkerp and Watson, and direct algorithm mirrors MCIR's efficacy on Biggsbi1. Turning our attention to trained neural networks (Fig. 3 second row), CMA shines on Ackley-50d, yet MCIR continues to deliver impressive results. Notably, for Michalewicz-50d and Michalewicz-100d, MCIR outperforms all other baseline algorithms.

Upon a closer examination of the result curves, it becomes evident that the MCIR algorithm's optimization performance is both commendable and in line with our initial expectations.

Ablation Study. We conducted ablation studies to analyze the individual contributions of different components in our algorithm. Specifically, we investigated the influence of the number of random samples placed under each selected node on the Michalewicz-50d function (Fig. 4a), assessed the effectiveness of local optimization on Michalewicz-50d (Fig. 4b), and examined the effectiveness of local optimization on the Watson-31d (Fig. 4c). Furthermore, we tested the hyper parameters used in the UCT formula (Eq. 2) using the Ackley-50d function as depicted in Fig. 4d, 4e, and 4f. From Fig. 4a, we observed that the number of new nodes placed after selecting a leaf node should be kept at a moderate level. Overpopulating the same local region with new nodes does not significantly enhance performance due to the closeness of local optima. The significance of local optimization can be found in Fig. 4b and Fig. 4c. It can be concluded that local optimization can both improve and hinder the performance of the algorithm, as shown Fig. 4c and Fig. 4b, respectively. Therefore, setting a reasonable budget for the local optimizer is important. A local optimizer with stopping criterion such as improvement threshold could be more preferable over one with fixed number of iterations. Fig. 4d, Fig. 4e, and Fig. 4f demonstrate the importance of the lower bound of the function value in determining the best node for searching the global minimum. Additionally, the balance between exploiting nodes excessively and leaving nodes unexplored becomes evident. The size of the box where the lower bound originates is the least sensitive hyper parameter, as a smaller box increases the certainty of the function's lower bound but has less impact compared to the value of the function's lower bound itself.

We have noticed that sometimes the effectiveness of C_v is limited, but as C_v represents the confidence of the predicted objective function interval, the choice of C_v highly

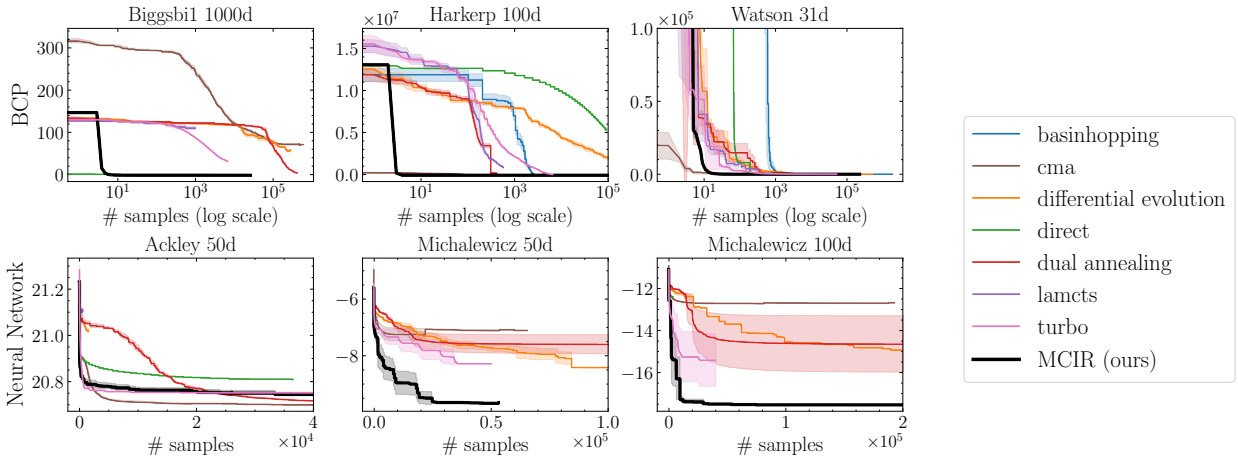
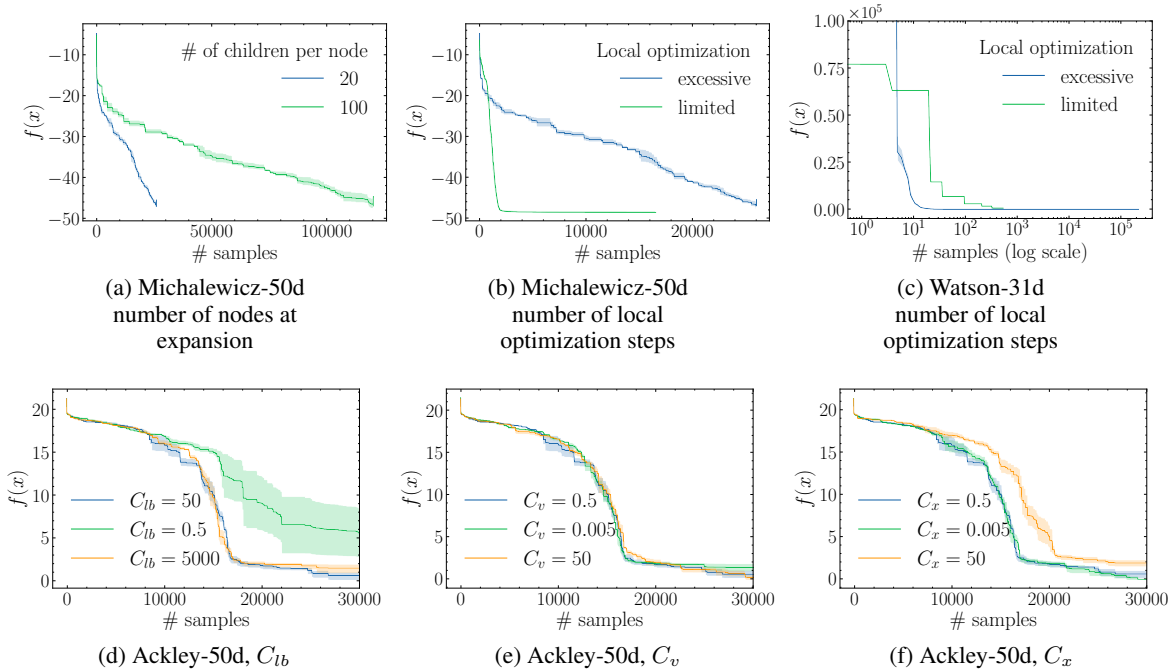


Figure 3: Overall performance of the baselines and MCIR on Biggsbi1, Harkerp, Watson, and Neural Networks.



(a) Michalewicz-50d number of nodes at expansion

(b) Michalewicz-50d number of local optimization steps

(c) Watson-31d number of local optimization steps

(d) Ackley-50d, C_{lb}

(e) Ackley-50d, C_v

(f) Ackley-50d, C_x

Figure 4: Ablation studies on function Michalewicz-50d for the number of children at expansion (a), the effectiveness of local optimization (b), on function Watson-31d for the effectiveness of local optimizaiton (c), and on function Ackley-50d for C_{lb} (d), C_v (e), and C_x (f).

depends on the landscape of the objective function. For functions where the bounds are evaluated through rough approximation, C_v becomes important. In our benchmark tests, we select our hyper parameters C_{lb}, C_v, and C_x that perform consistently well over multiple tests without requiring too much fine-tuning. Most of the parameters are shared across all problems based on the dimension and the type of problems. In practice, one can start with the setup that provides the best performance in this paper, and fine tune to specific tasks based on observed function complexity and landscape.

Conclusion

We introduced a new approach to non-convex optimizations problems by leveraging analytic and sampling-based information in an MCTS framework, enabling efficient exploration and exploitation of the state space. Experiments results on standard benchmark problem sets demonstrated clear benefits of the proposed approach. Future work can focus on reducing the overhead of various numerical computation involved in the proposed algorithm and further optimizing the search tree.

References

Alefeld, G.; and Herzberger, J. 2012. *Introduction to interval computation*. Academic press.

Araya, I.; and Reyes, V. 2016. Interval branch-and-bound algorithms for optimization and constraint satisfaction: a survey and prospects. *Journal of Global Optimization*, 65: 837–866.

Bao, T. Q.; and Mordukhovich, B. S. 2010. Set-valued optimization in welfare economics. *Advances in mathematical economics*, 113–153.

Byrd, R. H.; Lu, P.; Nocedal, J.; and Zhu, C. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5): 1190–1208.

Campi, M. C.; Garatti, S.; and Ramponi, F. A. 2015. Nonconvex scenario optimization with application to system identification. In *2015 54th IEEE Conference on Decision and Control (CDC)*, 4023–4028. IEEE.

De Boer, P.-T.; Kroese, D. P.; Mannor, S.; and Rubinstein, R. Y. 2005. A Tutorial on the Cross-Entropy Method. *Annals of operations research*, 134(1): 19–67.

Eriksson, D.; Pearce, M.; Gardner, J.; Turner, R. D.; and Poloczek, M. 2019. Scalable Global Optimization via Local Bayesian Optimization. In *Advances in Neural Information Processing Systems*, 5496–5507.

Gablonsky, J. M.; and Kelley, C. T. 2000. A locally-biased form of the DIRECT algorithm. Technical report, North Carolina State University. Center for Research in Scientific Computation.

Gao, F.; and Han, L. 2012. Implementing the Nelder-Mead Simplex Algorithm with Adaptive Parameters. *Comput. Optim. Appl.*, 51(1): 259–277.

Gurobi Optimization, L. 2023. *Gurobi Optimizer Reference Manual*.

Hansen, N.; yoshihikoueno; ARF1; Kadlecová, G.; Nozawa, K.; Rolshoven, L.; Chan, M.; Akimoto, Y.; brieglhostis; and Brockhoff, D. 2023. CMA-ES/pycma: r3.3.0.

Henderson, D.; Jacobson, S. H.; and Johnson, A. W. 2003. *The Theory and Practice of Simulated Annealing*, 287–319. Boston, MA: Springer US.

Hickey, T.; Ju, Q.; and Van Emden, M. H. 2001. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5): 1038–1068.

Jain, P.; Kar, P.; et al. 2017. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4): 142–363.

Lavezzi, G.; Guye, K.; and Ciarcià, M. 2022. Nonlinear Programming Solvers for Unconstrained and Constrained Optimization Problems: a Benchmark Analysis.

Liu, X.; and Lu, P. 2014. Solving nonconvex optimal control problems by convex optimization. *Journal of Guidance, Control, and Dynamics*, 37(3): 750–765.

Mistakidis, E. S.; and Stavroulakis, G. E. 2013. *Nonconvex optimization in mechanics: algorithms, heuristics and engineering applications by the FEM*, volume 21. Springer Science & Business Media.

Munos, R. 2011. Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness. In Shawe-Taylor, J.; Zemel, R.; Bartlett, P.; Pereira, F.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Ninin, J. 2016. Global optimization based on contractor programming: An overview of the IBEX library. In *Mathematical Aspects of Computer and Information Sciences: 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers 6*, 555–559. Springer.

Olson, B.; Hashmi, I.; Molloy, K.; and Shehu, A. 2012. Basin hopping as a general and versatile optimization framework for the characterization of biological macromolecules. *Advances in Artificial Intelligence*, 2012: 3–3.

Pant, M.; Zaheer, H.; Garcia-Hernandez, L.; Abraham, A.; et al. 2020. Differential Evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence*, 90: 103479.

Puranik, Y.; and Sahinidis, N. V. 2017. Bounds tightening based on optimality conditions for nonconvex box-constrained optimization. *Journal of Global Optimization*, 67: 59–77.

Sahinidis, N. 2023. *BARON: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual. The Optimization Firm, LLC, niksah@minlp.com.

Storn, R.; and Price, K. 1997. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4): 341.

Tawarmalani, M.; and Sahinidis, N. V. 2005. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103: 225–249.

The Optimization Firm. 2023. NLP and MINLP Test Problems. https://minlp.com/nlp-and-minlp-test-problems. Accessed: Aug.10, 2023.

Wang, L.; Fonseca, R.; and Tian, Y. 2020a. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. *arXiv preprint arXiv:2007.00708*.

Wang, L.; Fonseca, R.; and Tian, Y. 2020b. Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems*, 33: 19511–19522.

Xiang, Y.; Gubian, S.; Suomela, B.; and Hoeng, J. 2013. Generalized simulated annealing for global optimization: the GenSA package. *R J.*, 5(1): 13.

Yang, T. 2019. Advancing non-convex and constrained learning: Challenges and opportunities. *AI Matters*, 5(3): 29–39.

Yanover, C.; Meltzer, T.; Weiss, Y.; Bennett, K. P.; and Parrado-Hernández, E. 2006. Linear Programming Relaxations and Belief Propagation–An Empirical Study. *Journal of Machine Learning Research*, 7(9).

Zhai, Y.; and Gao, S. 2022. Monte Carlo Tree Descent for Black-Box Optimization. In *Advances in Neural Information Processing Systems*.

Zhu, C.; Byrd, R. H.; Lu, P.; and Nocedal, J. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4): 550–560.