

A FastMap-Based Framework for Black-Box Optimization

Kexin Zheng¹ and T. K. Satish Kumar¹

University of Southern California, Los Angeles, California, USA
kexinzhe@usc.edu, tskwork@gmail.com

Abstract. Complex engineering and design problems often do not have a closed-form expression. Such problems are referred to as Black-Box Optimization (BBO) problems. In BBO problems, finding an optimal solution requires us to efficiently search through a space of complex objects by leveraging only the input-output interface to an expensive evaluation procedure. One existing BBO approach is to use Machine Learning (ML)-based surrogate models that replace expensive evaluation procedures with cheaper high-fidelity approximations. However, in this paper, we propose a novel framework for BBO that not only has complementary strengths compared to ML approaches but can also be synergistically combined with them. In the first stage, our framework generates an artificial neighborhood structure and an optimization landscape on the space of complex objects using an efficient embedding algorithm called FastMap. In the second stage, it is combined with a viable search method, such as Variable Neighborhood Search (VNS), to explore the optimization landscape and find an optimal or near-optimal solution without necessarily calling the evaluation procedure on all proposed solutions. We demonstrate the efficiency and effectiveness of our framework on complex graph design problems that are characterized by multiple constraints and scoring functions derived from various measures of centrality.

Keywords: Black-Box Optimization · FastMap · Sample-Efficient Optimization · Variable Neighborhood Search · Graph Design.

1 Introduction

There are many real-world engineering and design problems that cannot be reduced to an optimization problem with a closed-form expression. Instead, they are characterized by an expensive evaluation procedure, the internal mechanism of which is either unknown or too complex to model analytically. When invoked, the evaluation procedure ingests a proposed solution, performs expensive computations, and outputs a score for the proposed solution. Hence, for such problems, the task of finding an optimal solution is challenging because standard optimization techniques, such as those based on the gradient [20], cannot be applied. In fact, since the optimization task on such problems has access only to the input-output interface of the evaluation procedure, such problems are formally referred to as Black-Box Optimization (BBO) problems [3].

As an example, consider the warehouse layout design problem [28]. When robots are deployed to execute warehouse operations, various competing factors should be considered: the robots should not collide with or block each other; the storage space should be maximally utilized; and the overall throughput of the system should be maximized. A design problem with such complex constraints and objectives is difficult to formulate using a closed-form expression. Consequently, it is challenging not only to find an optimal solution but even to evaluate a proposed solution. A proposed solution is evaluated across many simulated scenarios by running a robot-coordination algorithm on each of them.

Other real-world examples of BBO problems include the design of circuits [19], Carnot engines [5], RF antennas [4], and aerodynamic optimization of aircraft [23]. In these domains, simulation procedures to evaluate proposed solutions are prohibitively expensive. Moreover, in some domains, evaluation may be impossible *in silico*. For example, in automated drug discovery, peptide sequences must be evaluated *in vitro*—not *in silico*—to see how well they steer misfolded proteins back to healthy shapes [25]. Similarly, aircraft aerodynamics may require physical evaluation in wind tunnels if high-fidelity simulations are insufficient.

Two fundamental characteristics define the nature of the foregoing BBO problems. First, a proposed solution is often a complex specification, referred to as a “complex object” or simply an “object”. For instance, in the case of the warehouse layout design, a proposed solution, i.e., an object, is an entire environment map that specifies where the inventory items are stored and which locations the robots are allowed to occupy. Second, the evaluation procedure that runs on an input object is expensive: It can either be *in silico* and require simulations or *in vitro* and require physical experiments. In warehouse layout design, the evaluation procedure is typically a multi-agent path finding algorithm that runs on multiple test scenarios within the same environment map [28]. In aerodynamic design, the evaluation procedure may require physical wind tunnel experiments.

In this paper, we address the general BBO problem of searching a space of complex objects. Typically, finding the optimal object is not amenable to standard optimization techniques. One existing approach to address this is using Machine Learning (ML)-based surrogate models [1]. First, a cheaper high-fidelity surrogate is learned to replace the expensive evaluation procedure. Then, this surrogate model is evaluated on a large set of objects to identify the best solution.

We propose a novel BBO framework that is capable of synergistically harnessing existing ML approaches. In its first stage, the framework uses an efficient embedding algorithm, FastMap, to generate an artificial neighborhood structure and optimization landscape on the space of complex objects. FastMap [9] is a Data Mining algorithm that embeds a given set of objects into a user-specified K -dimensional Euclidean space, approximately preserving a quadratic number of pairwise domain-specific distances. Remarkably, it runs in only linear time.

In the second stage, our framework employs a search algorithm to explore the optimization landscape created by FastMap. The optimization landscape is conceptualized in $K + 1$ dimensions, where the first K dimensions are the FastMap coordinates of an object and the last dimension is the score of that

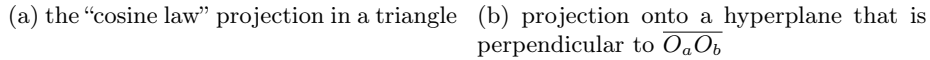


Fig. 1. Illustrates how the Euclidean coordinates are computed and recursion is carried out in FastMap; reproduced from [6].

object as determined by the evaluation procedure. The search algorithm can follow any standard strategy for optimization, such as Simulated Annealing [13], Large Neighborhood Search [22], or Variable Neighborhood Search (VNS) [16].

The search algorithm in the second stage aims to find an optimal or near-optimal solution without necessarily calling the expensive evaluation procedure on all objects. Notably, evaluation access remains unchanged in our FastMap-based framework: It still adheres to the black-box input-output interface, making it well suited for BBO. However, our framework uniquely leverages a domain-specific *distance function* on pairs of objects to conceptualize an optimization landscape that guides the search.

To demonstrate the efficiency and effectiveness of our framework, we choose illustrative Graph Design Problems (GDPs). These problems require searching through a space of graphs to identify a desired graph. The space of graphs can be vast, yet consists of a finite number of proposed candidates. The desired graph should satisfy multiple structural constraints and optimize a score derived from various measures of centrality. On these GDPs, we show that our FastMap-based framework yields (near-)optimal solutions with a speedup of 2–10 $\times$.

2 Background: FastMap

(This section is adapted from [6].) FastMap [9] embeds a set of complex objects in an artificially created Euclidean space. It takes as input a set $\mathcal{O}$ of such objects, where $D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in \mathcal{O}$.¹ It generates a Euclidean embedding that assigns a K -dimensional

¹ $D(\cdot, \cdot)$ is a non-negative symmetric function that yields 0 for identical objects.

point $\mathbf{p}_i \in \mathbb{R}^K$ to each object O_i . A good Euclidean embedding is one in which the Euclidean distance $\|\mathbf{p}_j - \mathbf{p}_i\|_2$ between any two points $\mathbf{p}_i$ and $\mathbf{p}_j$ closely approximates $D(O_i, O_j)$.

In the first iteration, FastMap heuristically identifies the farthest pair of objects O_a and O_b in linear time. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$, and $d_{ib} = D(O_i, O_b)$, as illustrated in Figure 1a. These sides define the triangle’s entire geometry; thus, the projection of O_i onto the line $\overline{O_a O_b}$ is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2) / (2d_{ab}). \quad (1)$$

FastMap sets the first coordinate of $\mathbf{p}_i$, the embedding of O_i , equal to x_i . In the subsequent $K - 1$ iterations, the same procedure is followed for computing the remaining $K - 1$ coordinates of each object; however, the distance function is adapted for each iteration. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab} , respectively. Because these coordinates fully explain the true distance between these two objects, from the second iteration onward, the remaining coordinates of $\mathbf{p}_a$ and $\mathbf{p}_b$ should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line $\overline{O_a O_b}$. Figure 1b illustrates this. Although the hyperplane is never explicitly constructed, it conceptually implies that the distance function for the second iteration should be changed for all i and j in the following way:

$$D_{\text{new}}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2, \quad (2)$$

in which O'_i and O'_j are the projections of O_i and O_j , respectively, onto this hyperplane, and $D_{\text{new}}(\cdot, \cdot)$ is the new distance function. The distance function is recursively updated according to Equation 2 at the beginning of each of the $K - 1$ iterations that follow the first.

In each of the K iterations, FastMap heuristically finds the farthest pair of objects according to the distance function defined for that iteration. These objects, called pivots, are stored as reference objects, yielding at most $2K$ of them. Technically, finding the exact farthest pair in any iteration takes $O(N^2)$ time. However, FastMap employs a linear-time “pivot changing” heuristic [9] to efficiently identify a pair O_a and O_b that is very often the farthest. It does this by initially choosing a random object O_b and selecting O_a to be the object farthest from O_b . It then iteratively reassigns O_b to be farthest from O_a , and O_a to be farthest from O_b , until convergence or a maximum of $C \leq 10$ iterations.

3 The FastMap-Based Framework for BBO

In this section, we first outline the observations that motivate our novel framework before detailing the framework itself.

Although the primary question of finding the optimal object is very challenging, much intuition can be drawn from trying to answer the following ancillary question instead: Given a set of proposed objects, is it possible to find the best

among them without calling the evaluation procedure on all of them? Remarkably, we now show that FastMap answers this question affirmatively because of its unique ability to create a Euclidean embedding of the objects—and hence a neighborhood structure on them—in only linear time.

Revisiting Boolean Satisfiability (SAT) solvers provides key observations. Past solvers like WalkSAT [21] and G2WSAT [15] conduct local search over a high-dimensional optimization landscape to find an optimal solution². Figure 2a illustrates this. The horizontal axis represents the assignment space, where bit-vectors denote all possible complete assignments to the Boolean variables³. The vertical axis counts the unsatisfiable clauses per complete assignment. Critically, these solvers use powerful heuristics to efficiently explore the exponential search space and find a global minimum without brute-force evaluating all assignments. Prominent methods incorporate hill climbing, noise, and tabu lists in the inner loop, and random restarts in the outer loop [10].

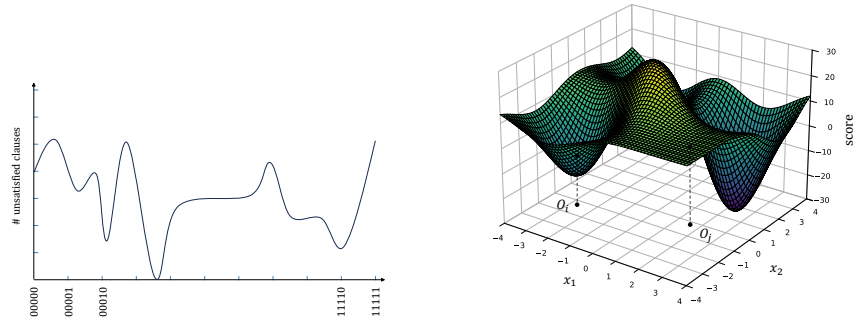
Upon closer inspection, a key feature of SAT solvers is that they interact with the evaluation procedure strictly via an input-output interface. Here, the evaluation procedure is a scoring function that takes a complete assignment, evaluates it against the SAT instance, and returns the number of unsatisfied clauses. Although this procedure is computationally simple, the crucial observation is that a SAT solver’s exploration of the optimization landscape does not rely on it having any access to the internal mechanism of the evaluation procedure. Instead, exploration relies critically on the definition of a “neighborhood” structure on the assignment space. For instance, a SAT solver might deem two complete assignments as neighbors if their Hamming distance is exactly 1.

A SAT solver’s principle of not accessing the evaluation procedure’s internal mechanism is actually a strict requirement in BBO. However, a SAT solver relies on a neighborhood structure on the assignment space, which is generally absent from existing discrete BBO frameworks. Consequently, existing frameworks do not leverage a neighborhood structure for complex proposed objects, rendering them unable to conduct efficient search. Although a “natural” neighborhood structure may not exist on the object space, the main novelty of our FastMap-based framework is to artificially create one, as illustrated in Figure 2b. By leveraging a domain-specific distance function, it embeds the objects in a Euclidean space in linear time while approximately preserving their pairwise distances. This Euclidean embedding yields the necessary neighborhood structure, allowing top- k nearest neighbor queries to be answered efficiently via Locality Sensitive Hashing [7] or other methods.

Drawing on these observations, our FastMap-based framework for BBO is built in two stages. First, it embeds the proposed objects in a user-specified K -dimensional Euclidean space. This linear-time embedding invokes FastMap using a user-supplied, domain-specific distance function. This subsequently establishes a neighborhood structure on the object space: for any k , the top- k nearest neighbors of any object are well-defined and efficiently computable. In the second

² Here, a solution and an “optimal” solution are identical.

³ A *complete assignment* is an assignment of values to all the Boolean variables.



(a) an optimization landscape conceptualized by a SAT solver (b) an optimization landscape conceptualized by our framework for BBO

Fig. 2. Illustrates how an optimization landscape based on a neighborhood structure facilitates efficient search for an optimal solution. In (b), FastMap embeds the objects in a 2-dimensional Euclidean space, as shown for O_i and O_j .

stage, our framework invokes a search algorithm that utilizes this neighborhood structure, much like SAT solvers. Crucially, because not every Euclidean point maps back to a valid object, the algorithm does not search the continuous vector space directly. Instead, it strictly exploits the induced neighborhood structure.

One template search algorithm that is highly suited to our framework is Variable Neighborhood Search (VNS) [16].⁴ It is a generalization of the greedy hill-climbing search. However, in the VNS template, multiple neighborhoods $\mathcal{N}(x) = \{\mathcal{N}_1(x), \mathcal{N}_2(x) \dots \mathcal{N}_\tau(x)\}$ are defined at each point x . In each iteration, x' is chosen randomly from one of the neighborhoods of x , and x'' is computed to be the local optimum in the vicinity of x' . A transition is made from x to x'' , i.e., $x \leftarrow x''$, if x'' has a better score. The VNS template can be instantiated in a variety of ways to yield specific local search algorithms. These variations include diverse strategies for exploring the neighborhoods of each point, the choice between making the neighborhoods static or adaptive, methods for determining the value of τ , and distinct termination criteria [8].

4 An Application to Graph Design Problems

Graphs are widely used to model a multitude of real-world domains, including transportation, social, and communication networks. Typically, the vertices represent the underlying entities, and the edges represent the relationships between them. Given the expressive modeling power of graphs, GDPs have profound implications on how we can structure and optimize these domains to satisfy strict constraints and achieve specific operational objectives.

⁴ The VNS template is applicable to both minimization and maximization problems.

In this section, we introduce hard GDPs that can be used to illustrate the power of our FastMap-based framework for BBO. The GDPs are characterized by combinatorially intricate constraints and objectives. The constraints are complex because they are global, and the objectives are complex because they are derived from globally defined measures of centrality. In essence, the sheer difficulty of these GDPs naturally frames them as BBO problems, in which the evaluation procedure is computationally expensive and defies analytical reasoning.

Our GDPs are grounded in this task: Given a grid world with 64×64 cells, we must mark exactly 2,000 cells as obstacles, ensuring the 2,096 remaining free cells form a single connected component.⁵ A target cell is specified with the objective of maximizing its value under a chosen centrality measure. Naturally, any proposed solution is viewed as an undirected graph with vertices corresponding to free cells and edges to adjacent free cells. Furthermore, the objective function employs an established centrality measure [18]: closeness, harmonic, current-flow closeness, eigenvector, Katz, or betweenness centrality. These GDPs can serve as good benchmarking problems for BBO with or without fixing the number of proposed objects.⁶

The FastMap component of our framework requires a pairwise distance function on the objects. Therefore, to apply our framework on the GDPs, we must define a distance function on graphs (the objects in this domain). Moreover, because graphs are inherently very general, the required distance function must be carefully crafted: A distance function that is too general would fail to sufficiently capture the nuanced combinatorial aspects of the GDPs.

To address this, we define two distance functions, $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, that are tailored to the GDPs. Given a target vertex v and two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $v \in V_1 \cap V_2$ and $|V_1| = |V_2|$, we first compute the shortest path trees rooted at v in both graphs. This yields distances $d_{i_1}^{G_1}$ and $d_{i_2}^{G_2}$ for all $v_{i_1} \in V_1$ and $v_{i_2} \in V_2$, where $d_{i_1}^{G_1}$ and $d_{i_2}^{G_2}$ represent the shortest path distances from v to v_{i_1} in G_1 and to v_{i_2} in G_2 , respectively. Then, $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$ are defined and computed differently. $D_1(G_1, G_2)$ is defined to be the cost of the “maximum matching of minimum cost” formulated over a complete bipartite graph with V_1 and V_2 as the two partitions, where each edge (v_{i_1}, v_{i_2}) is annotated with the cost $|d_{i_1}^{G_1} - d_{i_2}^{G_2}|$. $D_1(\cdot, \cdot)$ can be computed in polynomial time via a min-cost maxflow algorithm [14]. $D_2(G_1, G_2)$ is computable more efficiently and, for a parameter $\alpha \in [0, 1]$, is given by

$$D_2(G_1, G_2) = \left| \sum_{v_{i_1} \in V_1} \alpha^{d_{i_1}^{G_1}} - \sum_{v_{i_2} \in V_2} \alpha^{d_{i_2}^{G_2}} \right|. \quad (3)$$

For solving GDPs within our proposed framework, the VNS template can be instantiated to a relatively simple sequential deterministic form, called Variable Neighborhood Descent (VND) [8]. At each point x , VND sequentially explores larger neighborhoods of x within an increasing radius: $\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \dots \mathcal{N}_\tau(x)$.

⁵ This configuration serves as a proof of concept; other settings yield similar results.

⁶ In this paper, we fix the number of objects for an initial proof of concept.

Specifically, VND tries to find a better point x'' in the neighborhood of x , starting from the smallest one. If it can do so, x'' is assigned to x for the next iteration. If not, x'' is sought in the next larger neighborhood of x . VND terminates if the largest neighborhood $\mathcal{N}_\tau(x)$ is exhausted without finding an x'' better than x .

5 Experimental Results

We now present experimental results to demonstrate the efficacy of our FastMap-based framework on GDPs. We implemented our framework in Python 3 and conducted our experiments on an Apple M2 processor with 16 GB of RAM.

We formulated 18 BBO problems using all combinations of three target vertices (cells) and six measures of centrality. The three target vertices in the 64×64 grid world are: (10, 10), (32, 32), and (40, 50). The six measures of centrality are: closeness, harmonic, current-flow closeness, eigenvector, Katz, and betweenness centrality. Each BBO problem, i.e., each GDP instance, is formulated on a space of 2,000 proposed graphs. As mentioned in the previous section, the task is to find a graph among a fixed number of proposed graphs that maximizes the target vertex’s value with respect to the chosen centrality measure. The difficulty of the optimization task stems mainly from the resource-intensive evaluation of the centrality values.⁷ The theoretical complexity of computing these values for each centrality measure can be found in [18].

In our framework, we implemented the VND algorithm with an outer loop. The outer loop has ‘MaxIter’ iterations, for a user-specified value of ‘MaxIter’. In each iteration, the algorithm starts at a randomly chosen point (proposed graph) and conducts search using radii to define the neighborhoods. We set $\tau = 3$ and, at any point x , define $\mathcal{N}_1(x)$, $\mathcal{N}_2(x)$, and $\mathcal{N}_3(x)$ to be the sets of all points within distances of R , $1.5R$, and $2.25R$ from x , respectively. The initial radius R is a user-specified parameter. Moreover, if the cardinality of $\mathcal{N}_1(x)$, $\mathcal{N}_2(x)$, or $\mathcal{N}_3(x)$ exceeds 150, we retain only 150 randomly chosen elements in that set. An iteration stops when there is no point in the neighborhood of x that is better than x . Hence, the termination of the VND algorithm can be based either on the maximum number of outer-loop iterations, i.e., ‘MaxIter’, or a user-specified maximum number of calls to the expensive evaluation procedure, i.e., ‘MaxEval’. In any case, the algorithm employs memoization to avoid calling the evaluation procedure on the same proposed graph more than once.

We run our framework on the GDP instances using different configurations. These configurations comprise combinations of: (a) the choice of $D_1(\cdot, \cdot)$ or $D_2(\cdot, \cdot)$; (b) the value of $\alpha \in \{0.10, 0.25, 0.50, 0.99\}$ for $D_2(\cdot, \cdot)$; (c) the value of $K \in \{2, 3\}$; and (d) the value of ‘MaxIter’ or ‘MaxEval’. For each configuration, we average the results over 30 runs: Each run is invoked with a different randomness seed. In the remainder of this section, we report our experimental results on two out of the three target vertices—(10, 10) and (32, 32)—and three out of the

⁷ For each GDP instance, the ground-truth optimal graph is obtained via brute-force evaluation of all the 2,000 proposed graphs.

Table 1. Shows the results of our framework on GDPs with three different measures of centrality and two distinct target vertices. Termination is governed by the maximum number of outer-loop iterations: ‘MaxIter’.

Closeness Centrality								
MaxIter	# Evaluations	Rank of Success		Time (s)	# Evaluations	Rank of Success		Time (s)
		Result	Rate			Result	Rate	
Target = (10, 10)	D1: Initial Radius = 0.04; $K = 2$.				D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 2$.			
1	150.8	1.0	100.0	299	212.2	10.0	90.0	417
2	173.9	1.0	100.0	343	319.6	4.0	96.7	624
3	194.7	1.0	100.0	383	368.3	1.0	100.0	719
4	215.3	1.0	100.0	423	417.4	1.0	100.0	813
5	232.4	1.0	100.0	456	450.2	1.0	100.0	876
Brute Force	2000.0	1.0	100.0	3871	2000.0	1.0	100.0	3871
Target = (32, 32)	D1: Initial Radius = 0.03; $K = 2$.				D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 2$.			
1	168.0	86.4	93.3	330	265.4	1.2	96.7	517
2	206.4	1.0	100.0	404	395.9	1.0	100.0	768
3	227.4	1.0	100.0	444	483.8	1.0	100.0	936
4	240.3	1.0	100.0	469	556.0	1.0	100.0	1075
5	251.4	1.0	100.0	490	600.0	1.0	100.0	1160
Brute Force	2000.0	1.0	100.0	3843	2000.0	1.0	100.0	3843
Current-Flow Closeness Centrality								
MaxIter	# Evaluations	Rank of Success		Time (s)	# Evaluations	Rank of Success		Time (s)
		Result	Rate			Result	Rate	
Target = (10, 10)	D1: Initial Radius = 0.07; $K = 3$.				D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 3$.			
1	293.7	1.3	83.3	780	487.2	3.8	73.3	1289
2	403.5	1.1	96.7	1069	680.9	1.3	96.7	1798
3	471.6	1.1	96.7	1248	811.0	1.3	96.7	2141
4	552.6	1.0	100.0	1461	915.8	1.0	100.0	2416
5	592.9	1.0	100.0	1567	978.9	1.0	100.0	2582
Brute Force	2000.0	1.0	100.0	5274	2000.0	1.0	100.0	5274
Target = (32, 32)	D1: Initial Radius = 0.05; $K = 2$.				D2: Initial Radius = 0.15; $\alpha = 0.99$; $K = 2$.			
1	491.6	63.9	80.0	1325	483.8	1.3	93.3	1304
2	696.8	1.0	100.0	1876	729.0	1.0	100.0	1961
3	828.3	1.0	100.0	2229	847.3	1.0	100.0	2278
4	912.7	1.0	100.0	2454	916.9	1.0	100.0	2465
5	959.2	1.0	100.0	2578	997.3	1.0	100.0	2680
Brute Force	2000.0	1.0	100.0	5379	2000.0	1.0	100.0	5379
Katz Centrality								
MaxIter	# Evaluations	Rank of Success		Time (s)	# Evaluations	Rank of Success		Time (s)
		Result	Rate			Result	Rate	
Target = (10, 10)	D1: Initial Radius = 0.09; $K = 3$.				D2: Initial Radius = 0.15; $\alpha = 0.25$; $K = 2$.			
1	558.1	3.7	46.7	69	141.1	1.0	100.0	24
2	812.7	1.9	70.0	97	176.8	1.0	100.0	27
3	1012.8	1.3	83.3	119	210.4	1.0	100.0	31
4	1178.9	1.1	96.7	137	237.8	1.0	100.0	34
5	1248.8	1.1	96.7	144	271.1	1.0	100.0	37
Brute Force	2000.0	1.0	100.0	236	2000.0	1.0	100.0	236
Target = (32, 32)	D1: Initial Radius = 0.09; $K = 3$.				D2: Initial Radius = 0.20; $\alpha = 0.25$; $K = 3$.			
1	431.0	8.6	23.3	53	236.4	1.3	93.3	33
2	761.8	1.9	60.0	88	326.3	1.0	100.0	42
3	973.1	1.3	83.3	111	379.6	1.0	100.0	48
4	1135.7	1.1	90.0	128	417.4	1.0	100.0	52
5	1238.3	1.1	93.3	139	439.8	1.0	100.0	54
Brute Force	2000.0	1.0	100.0	222	2000.0	1.0	100.0	222

six measures of centrality—closeness, current-flow closeness, and Katz—under many different configuration settings.⁸

⁸ Although we are unable to present all of the experimental results here, the omitted results exhibit the same performance patterns and are available at https://github.com/kexinzheng05/FastMap_Based_Framework_for_BB0.

Table 1 shows the experimental results when the termination is governed by ‘MaxIter’. It includes a “Brute Force” baseline algorithm that calls the evaluation procedure on all of the proposed graphs. The three sub-tables correspond to the three measures of centrality: closeness, current-flow closeness, and Katz. Each sub-table is horizontally sectioned by the target vertices (10, 10) and (32, 32), respectively. In turn, each is vertically divided by distance functions $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$. In each case, the best possible values are chosen for R (the initial radius), K , and α .⁹ ‘# Evaluations’ refers to the number of calls to the evaluation procedure. ‘Rank of Result’ is the rank of the reported graph among the proposed graphs sorted by their evaluation scores. ‘# Evaluations’ and ‘Rank of Result’ are averaged over 30 runs, each with a different randomness seed. ‘Success Rate’ is the percentage of these 30 runs in which the optimal graph is found.

From Table 1, we observe that our framework evaluates only a fraction of the 2,000 proposed graphs to find the optimal one. For closeness centrality, a 100% success rate is achieved under all settings in only 1–3 outer-loop iterations. Under $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, this needs an average of fewer than 207 and 396 calls to the evaluation procedure, yielding a speedup of $\approx 10\times$ and $\approx 5\times$, respectively, over the baseline algorithm. For current-flow closeness centrality, the number of outer-loop iterations required for a 100% success rate is 2–4 under all settings. Under $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, this entails an average of fewer than 697 and 916 calls to the evaluation procedure, yielding a speedup of $\approx 3\times$ and $\approx 2\times$, respectively. For Katz centrality, the configuration under $D_2(\cdot, \cdot)$ clearly outperforms that under $D_1(\cdot, \cdot)$. While it takes over 5 outer-loop iterations to achieve a high success rate under $D_1(\cdot, \cdot)$, it takes only 2 outer-loop iterations to achieve a 100% success rate under $D_2(\cdot, \cdot)$. This requires an average of over 1,200 calls to the evaluation procedure under $D_1(\cdot, \cdot)$ but fewer than 327 calls under $D_2(\cdot, \cdot)$, yielding a speedup of $\approx 5\times$.

Table 2 shows the experimental results when the termination is governed by ‘MaxEval’. It includes a “Random Sampling” baseline algorithm that calls the evaluation procedure on ‘MaxEval’ proposed graphs chosen uniformly at random. Its performance metrics are also computed over 30 runs with different randomness seeds. Table 2 retains the semantics of Table 1 and has a similar structure. In fact, although the termination is governed differently in the two tables, the observations made in Table 1 can be validated again in Table 2. Additionally, we can make a few more observations in Table 2. For closeness centrality, the configuration setting under $D_1(\cdot, \cdot)$ requires a budget of only 100 and 150 calls to the evaluation procedure to achieve a 100% success rate for the target vertices (10, 10) and (32, 32), respectively. However, the configuration setting under $D_2(\cdot, \cdot)$ is slightly less effective. The required budget is 400 for a 96.7% success rate. Nonetheless, it is still significantly better than the baseline algorithm, which has a very low success rate even for 500 calls. For current-flow closeness centrality and the target vertex (10, 10), 600 and 900 calls are required under $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, respectively, for a 93.3% success rate. For the target

⁹ The “best” possible values for the hyperparameters were chosen by manual inspection. These could be further optimized through a systematic grid search.

Table 2. Shows the results of our framework on GDPs with three different measures of centrality and two distinct target vertices. Termination is governed by the maximum number of calls to the evaluation procedure: ‘MaxEval’.

Closeness Centrality									
	MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate	
Target = (10, 10)	D1: Initial Radius = 0.04; $K = 2$.			D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 2$.			Random Sampling		
	50	69.0	73.3	200	40.9	76.7	50	43.0	0.0
	100	1.0	100.0	300	17.6	86.7	200	11.3	6.7
	150	1.0	100.0	400	4.0	96.7	350	5.7	10.0
	200	1.0	100.0	500	1.0	100.0	500	4.4	20.0
Target = (32, 32)	D1: Initial Radius = 0.03; $K = 2$.			D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 2$.			Random Sampling		
	100	57.4	76.7	200	58.1	80.0	100	22.3	0.0
	150	1.0	100.0	300	1.8	93.3	250	7.8	0.0
	200	1.0	100.0	400	1.1	96.7	350	5.0	6.7
	250	1.0	100.0	500	1.0	100.0	500	3.8	20.0
Current-Flow Closeness Centrality									
	MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate	
Target = (10, 10)	D1: Initial Radius = 0.07; $K = 3$.			D2: Initial Radius = 0.10; $\alpha = 0.99$; $K = 3$.			Random Sampling		
	200	8.1	53.3	500	6.0	60.0	200	10.8	13.3
	400	1.2	86.7	700	2.0	86.7	500	3.4	16.7
	600	1.1	93.3	900	1.4	93.3	800	2.5	33.3
	800	1.0	100.0	1100	1.0	100.0	1100	2.0	46.7
Target = (32, 32)	D1: Initial Radius = 0.05; $K = 2$.			D2: Initial Radius = 0.15; $\alpha = 0.99$; $K = 2$.			Random Sampling		
	200	89.2	43.3	400	8.8	76.7	200	8.4	6.7
	400	13.0	66.7	600	1.2	93.3	450	3.5	30.0
	600	1.2	93.3	800	1.0	100.0	700	2.4	46.7
	800	1.0	100.0	1000	1.0	100.0	1000	1.6	60.0
Katz Centrality									
	MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate		MaxEval	Rank of Success Result Rate	
Target = (10, 10)	D1: Initial Radius = 0.09; $K = 3$.			D2: Initial Radius = 0.15; $\alpha = 0.25$; $K = 2$.			Random Sampling		
	600	2.2	50.0	100	6.8	56.7	100	22.4	6.7
	800	1.8	56.7	150	2.0	96.7	450	4.8	23.3
	1000	1.4	76.7	200	1.1	96.7	800	2.4	36.7
	1200	1.2	90.0	250	1.0	100.0	1200	1.7	56.7
Target = (32, 32)	D1: Initial Radius = 0.09; $K = 3$.			D2: Initial Radius = 0.20; $\alpha = 0.25$; $K = 3$.			Random Sampling		
	600	3.1	43.3	150	15.5	26.7	150	10.4	10.0
	800	2.2	50.0	200	5.6	70.0	500	4.7	16.7
	1000	1.6	70.0	250	1.6	96.7	850	2.2	46.7
	1200	1.2	86.7	300	1.0	100.0	1200	1.4	66.7

vertex (32, 32), 600 calls are required in both cases for a 93.3% success rate. These numbers are significantly better than those of the baseline algorithm, whose success rate is very low even for 1,100 calls. For Katz centrality, the numbers

are in favor of the configuration setting under $D_2(\cdot, \cdot)$. For the target vertex (10, 10), 1,200 and 150 calls are required under $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, respectively, for a $\geq 90\%$ success rate. For the target vertex (32, 32), $>1,200$ and 250 calls are required under $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$, respectively, for a high success rate. Once again, these numbers are significantly better than those of the baseline algorithm, whose success rate is very low even for 1,200 calls.

Overall, Tables 1 and 2 show that our FastMap-based framework is highly effective. In fact, even when it is unable to find an optimal graph within its allocated computational resources, the ‘Rank of Result’ column shows that it is able to find near-optimal graphs very quickly.

Figure 3 provides insights into our framework’s behavior at an instance level. Figure 3a visualizes the optimization landscape for a specific GDP instance. It exhibits a gradation effectively exploited by an intelligent search algorithm like VND. This gradation demonstrates that the FastMap embedding captures key combinatorial aspects of the GDP via the distance function. Hence, a smart distance function creates a “smooth” landscape, enabling the algorithm to efficiently find an optimal or near-optimal solution. Figure 3b shows how fast our framework converges to the optimal solution for another typical GDP instance. It showcases rapid convergence, notably within the first 300 calls to the evaluation procedure. Thus, an optimal or near-optimal graph is usually found by evaluating only $\approx 15\%$ of the proposed graphs.

6 Related Work

We now discuss important related works and their inapplicability to GDPs in discrete search spaces. We categorize this literature based on two key aspects of BBO that can be synergistically integrated: (a) the optimization landscape, specifically whether the approach employs a cheaper surrogate for expensive evaluations, and (b) the neighborhood structure it defines on the search space.

Direct search approaches for BBO do not seek a cheaper surrogate for evaluations and thus do not conceptualize a surrogate optimization landscape. They assume a continuous vector space where the neighborhood structure is naturally defined by Euclidean distance. Instead of modeling the landscape, they explore the search space via geometric rules and sampling. For instance, the Nelder-Mead approach [17] evaluates the vertices of a simplex and iteratively modifies the worst-scoring vertex. Pattern search [26] and mesh adaptive direct search [2] evaluate the vertices of an adaptive mesh around the current vector, iteratively moving the mesh center to a better-scoring vertex. Direct search approaches are unsuited to explore discrete or categorical search spaces.

Surrogate approaches for BBO seek a cheaper surrogate for evaluations, consequently conceptualizing a surrogate optimization landscape. They can rely on Neural Networks (NNs) or Deep NNs. For instance, the Conservative Objective Model (COM) [27] trains a feed-forward NN as a cheaper proxy during search. To avoid generating a “hallucinated” optimal object, COM uses a regularized training loss function to offset over-optimistic predictions. MATCH-OPT [11] follows

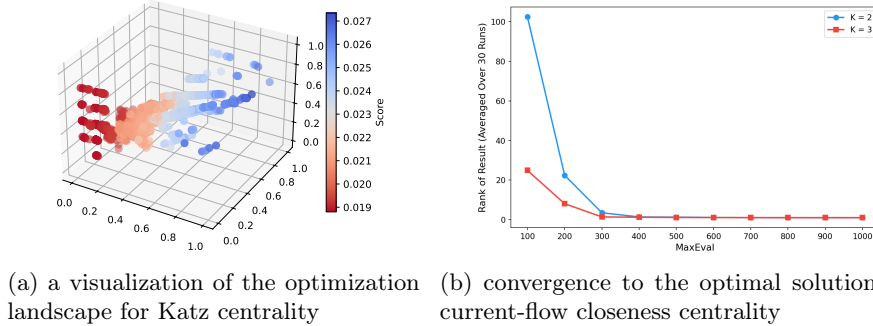


Fig. 3. The left side shows a visualization of the optimization landscape for maximizing the Katz centrality at the target vertex $(32, 32)$. The 2,000 proposed graphs are represented as points in a 3-dimensional Euclidean space generated by FastMap using $D_2(\cdot, \cdot)$ with $\alpha = 0.25$. The evaluation scores are color-coded: reddest (lowest) to bluest (highest). The right side shows the rate at which our framework converges to the optimal solution for maximizing the current-flow closeness centrality at the target vertex $(10, 10)$. The embedding utilizes the distance function $D_1(\cdot, \cdot)$.

a similar paradigm: Its regularized training loss function matches the surrogate gradient to the numerically estimated “true” gradient on training objects.

The Bayesian Optimization approach [24] is a surrogate approach that uses a Gaussian Process (GP) to predict the evaluation score at a given point in continuous space. Rather than outputting a single value, it outputs an entire probability distribution. This distribution has a lower (higher) variance when the point is closer to (farther from) previously evaluated points. Hence, this approach combines a predicted expected evaluation score and a measure of uncertainty to formulate the acquisition function. The acquisition function elegantly balances exploitation against exploration to guide the search. However, a GP scales cubically in the number of evaluated points and can itself become computationally prohibitive as a surrogate. A Sparse GP [12] is used to alleviate this problem by leveraging a fixed number of *inducing points* in the search space.

A major drawback of surrogate approaches, including COM, MATCH-OPT, and Bayesian Optimization, is that they rely on continuous vector search spaces. Although continuous relaxations of discrete variables are sometimes used, surrogate approaches are fundamentally unequipped to conduct search on discrete or categorical variables. More critically, when objects are sufficiently complex, their methodologies fail: Not only is vector representation unwieldy, but the neighboring points of a vector may not even map back to valid, meaningful objects. Additionally, some surrogate approaches suffer from the curse of dimensionality: The models struggle to converge as vector space dimensionality increases.

A major advantage of our BBO framework is its ability to search within complex object spaces. Critically, it leverages a domain-specific distance function to create a Euclidean embedding. Although objects have a vector representation

here, search is not conducted within the continuous vector space. Instead, the embedding is solely used to efficiently answer the top- k nearest neighbor queries.

7 Conclusions and Future Work

In this paper, we presented a novel framework for BBO based on FastMap. While previous BBO approaches require continuous vector search spaces, our framework obviates this necessity. It is applicable to BBO problems on complex objects that are not necessarily amenable to vector representations but are amenable to a pairwise domain-specific distance function. Such complex objects arise as proposed solutions in many real-world engineering and design problems where the task is to identify a highly optimized configuration. The core innovation of our framework is to generate an artificial neighborhood structure and an optimization landscape on the complex object space in only linear time. After doing so, a viable search method, such as VNS, is invoked to explore the landscape and efficiently find an optimal or near-optimal solution. We demonstrated the efficiency and effectiveness of our framework on a collection of GDPs, where the evaluation procedure uses various centrality measures and is computationally expensive. On these complex GDPs, we showed that our framework almost always finds the optimal solution with a speedup of 2–10 $\times$. In future work, we plan to apply our framework in more engineering domains.

Acknowledgments This material is based upon work supported by the National Science Foundation under grant number 2112533.

References

1. Alizadeh, R., Allen, J.K., Mistree, F.: Managing Computational Complexity Using Surrogate Models: A Critical Review. *Research in Engineering Design* (2020)
2. Audet, C., Dennis, J.E.: Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization* (2006)
3. Audet, C., Hare, W.: *Derivative-Free and Blackbox Optimization*. Springer Cham (2017)
4. Borges, L.M., Barroca, N., Saraiva, H.M., Tavares, J., Gouveia, P.T., Velez, F.J., Loss, C., Salvado, R., Pinho, P., Gonçalves, R., Carvalho, N.B., Chavéz-Santiago, R., Balasingham, I.: Design and Evaluation of Multi-Band RF Energy Harvesting Circuits and Antennas for WSNs. In: *Proceedings of the International Conference on Telecommunications* (2014)
5. Chen, L., Zhu, X., Sun, F., Wu, C.: Exergy-Based Ecological Optimization of Linear Phenomenological Heat-Transfer Law Irreversible Carnot-Engines. *Applied Energy* (2006)
6. Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., Kumar, T.K.S.: The FastMap Algorithm for Shortest Path Computations. In: *Proceedings of the International Joint Conference on Artificial Intelligence* (2018)
7. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-Sensitive Hashing Scheme Based on p -Stable Distributions. In: *Proceedings of the Annual Symposium on Computational Geometry* (2004)

8. Duarte, A., Sánchez-Oro, J., Mladenović, N., Todosijević, R.: Variable Neighborhood Descent. Springer International Publishing (2018)
9. Faloutsos, C., Lin, K.I.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (1995)
10. Gent, I.P., Walsh, T.: Towards an Understanding of Hill-Climbing Procedures for SAT. In: Proceedings of the National Conference on Artificial Intelligence (1993)
11. Hoang, M., Fadhel, A., Deshwal, A., Doppa, J.R., Hoang, T.N.: Learning Surrogates for Offline Black-Box Optimization via Gradient Matching. In: Proceedings of the International Conference on Machine Learning (2024)
12. Jin, W., Barzilay, R., Jaakkola, T.: Junction Tree Variational Autoencoder for Molecular Graph Generation. In: Proceedings of the International Conference on Machine Learning (2018)
13. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science (1983)
14. Kumar, T.K.S.: Incremental Computation of Resource-Envelopes in Producer-Consumer Models. In: Proceedings of the Principles and Practice of Constraint Programming (2003)
15. Li, C.M., Huang, W.Q.: Diversification and Determinism in Local Search for Satisfiability. In: Proceedings of the Theory and Applications of Satisfiability Testing (2005)
16. Mladenović, N., Hansen, P.: Variable Neighborhood Search. Computers & Operations Research (1997)
17. Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. The Computer Journal (1965)
18. Newman, M.: Networks. Oxford University Press (2018)
19. Richer, S., DiVincenzo, D.: Circuit Design Implementing Longitudinal Coupling: A Scalable Scheme for Superconducting Qubits. Phys. Rev. B (2016)
20. Ruder, S.: An Overview of Gradient Descent Optimization Algorithms (2017), <http://arxiv.org/abs/1609.04747>
21. Selman, B., Kautz, H.A., Cohen, B.: Noise Strategies for Improving Local Search. In: Proceedings of the National Conference on Artificial Intelligence (1994)
22. Shaw, P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Proceedings of the Principles and Practice of Constraint Programming (1998)
23. Skinner, S.N., Zare-Behtash, H.: State-of-the-Art in Aerodynamic Shape Optimisation Methods. Applied Soft Computing (2018)
24. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian Optimization of Machine Learning Algorithms. In: Proceedings of the Advances in Neural Information Processing Systems (2012)
25. Takahashi, T.T., Austin, R.J., Roberts, R.W.: mRNA Display: Ligand Discovery, Interaction Analysis and Beyond. Trends Biochem Sci. (2003)
26. Torczon, V.: On the Convergence of Pattern Search Algorithms. SIAM Journal on Optimization (1997)
27. Trabucco, B., Kumar, A., Geng, X., Levine, S.: Conservative Objective Models for Effective Offline Model-Based Optimization. In: Proceedings of the International Conference on Machine Learning (2021)
28. Zhang, Y., Fontaine, M.C., Bhatt, V., Nikolaidis, S., Li, J.: Multi-Robot Coordination and Layout Design for Automated Warehousing. In: Proceedings of the International Joint Conference on Artificial Intelligence (2023)