An Annealing-Inspired Gradient-Descent Based Suboptimal Solver for Combinatorial Problems

Shu-Ping Chang[1], Cheng-Che Lee[2], Hsin-Jung Lee[2], Chieh-Hsiung Kuan[1,2]
Jason Gemsun Young[3], Chia-Yu Yao[4], and Jian-Jiun Ding*
[1]Graduate Institute of Biomedical Electronics and Bioinformatics, National Taiwan University
[2]Graduate Institute of Electronics Engineering, National Taiwan University
[3]Electronic and Optoelectronic System Research Laboratories, Industrial Technology Research Institute
[4]Department of Electrical Engineering, National Taiwan University of Science and Technology
*Graduate Institute of Communication Engineering, National Taiwan University
E-mail: jjding@ntu.edu.tw Tel: +886-2-33669652

Abstract—**Combinatorial optimization problems, such as IC layout and industrial scheduling, have significant industrial applications but are challenging due to exponential time complexity. In this work, we propose a novel annealing-inspired heuristic algorithm that treats combinatorial problems as function optimization problems using nonlinear programming. The proposed gradient-descent-based solver significantly improves the convergence rate and includes a new regularization constraint to escape local minima by increasing convexity. Applied to the Traveling Salesman Problem (TSP) with various city counts, the proposed algorithm demonstrates polynomial time complexity. It much reduces the complexity from $(n-1)!/2$ to n^4 and has a marked improvement in computation efficiency. Notably, for a 50-city TSP, the relative error is just around 5%, indicating the accuracy and efficiency of the proposed algorithm in solving high-dimensional instances.**

I. Introduction

In nature, many processes seek stable states, such as the evolution and changes in ecological environments or macroscopic phenomena resulting from the balance of microphysical laws. These processes can be considered natural optimization mechanisms. Over the past 50 years, various global optimization algorithms have been developed to simulate these processes, including Simulated Annealing, Neural Networks, and Genetic Algorithms [1]. Combinatorial optimization problems are crucial due to their wide range of applications, such as IC layout, industrial scheduling, and isomer analysis. The Traveling Salesman Problem (TSP) is a prominent example frequently studied in mathematics and computer science. Traditional methods like dynamic programming and branch-and-bound aim to solve these problems but often require significant computational resources.

However, many combinatorial optimization problems are NP-hard or NP-complete, making exhaustive search impractical within a reasonable time frame. The exponential growth in computational complexity remains a significant challenge, especially for high-dimensional instances.

In this study, we develop an annealing-inspired algorithm to address the exponential growth in computational complexity in combinatorial optimization problems. Our method treats these problems as function optimization tasks using nonlinear programming, with the TSP as a key example. Inspired by Finnila's Quantum Annealing in 1994 [2], which utilizes quantum tunneling, we propose leveraging pathological system characteristics to achieve states close to multiple solutions. This approach, followed by continuous unconstrained optimization, transforms the problem into a universal combinatorial solver, applicable to TSP, IC layout, industrial scheduling, and heterogeneous structure analysis.

Our approach employs gradient descent to solve ill-conditioned systems, freely choosing methods like the conjugate gradient to improve convergence speed. Inspired by the framework in [3], we introduce a regularization term similar to thermal annealing, enabling the solver to escape local minima by gradually increasing cost function convexity. We validate the feasibility of our algorithm through TSP of different dimensions, a standard test model in mathematics and computer science. This paper emphasizes the algorithm's uniqueness and generality, presenting it as an effective strategy for improving combinatorial optimization solutions.

II. Basic Principles Of Our Approach

The Traveling Salesman Problem (TSP) can be transformed into a quadratic polynomial problem through nonlinear programming, representing it as a binary variable optimization problem [4]. Let TSP be defined as n cities in the set $N \equiv \{1,2,\dots n\}$. Without loss of generality, let the nth city be both the starting point and the endpoint, referred to as the "Hometown," since the degree of freedom for an n-city TSP is effectively only $n-1$.

A. *Using Route Length as Cost Function in Unconstrained Optimization*

After excluding the hometown city, the remaining number of cities is denoted as $m = n-1$, forming a city set $C \equiv \{1,2,\dots m\}$, and $T \equiv \{1,2,\dots m\}$ as the ordinal set representing travel time. These correspond to the order of visiting cities in the set C. Through these two sets, we generate a set of binary variables x_{tj} $(t \in T, i \in C)$, defined as follows:

$$x_{ti} \equiv \begin{cases} 1, & \text{if city } i \text{ is visited at time order } t \\ 0, & \text{otherwise} \end{cases}, 1 \le t, i \le m \tag{1}$$

x_{ti} has k variables, where $k = m^2$, capable of expressing one possible path among all permutations, representing the city i at time t. Finally, all variables are placed into a $k \times 1$ matrix, $\mathbf{x} \in \{0,1\}^k$. The optimized cost function $J_1(\mathbf{x})$ can then be expressed as:

$$\mathbf{x} \equiv [x_{11}\ x_{12}\ \cdots\ x_{1m}\ \cdots\ x_{m1}\ x_{m2}\ \cdots\ x_{mm}]^T \tag{2}$$

$$J_1(\mathbf{x}) \equiv \sum_{t\in T\backslash\{m\}} \sum_{i\in C} \sum_{j\in C\backslash\{i\}} d_{ij} x_{ti} x_{(t+1)j} + \sum_{t\in\{1,m\}} \sum_{i\in C} c_i x_{ti} \tag{3}$$

The set of paths between two cities is denoted as $E = C^2$. Among them, d_{ij} $((i,j) \in E)$ is the distance between city i and city j, and c_i $(i \in C)$ is the distance from city i to the hometown. This way, the optimized cost function can fully express the distance of one possible path.

Since the number of distances d_{ij} between cities is $m(m-1)/2$, and each route contains n distances between cities, the average route length L_{avg} is approximately:

$$L_{\text{avg}} \equiv \frac{2n}{m(m-1)} \left(\sum_{i\in C} \sum_{j\in C\{i\}} d_{ij} + \sum_{i\in C} c_i \right) \tag{4}$$

The new cost function $J_2(\mathbf{x})$ is the normalized $J_1(\mathbf{x})$:

$$J_2(\mathbf{x}) \equiv J_1(\mathbf{x}) / L_{\text{avg}} \tag{5}$$

B. Using Regularization to Incorporating Constraints

We utilize regularization to maintain our problem as unconstrained optimization by incorporating regularization terms to incorporate constraints. In Fig. 1, each row is associated with the condition that only one city is visited at a time, meaning that only one binary variable in each row equals 1, and the others are 0. Similarly, each column is associated with the condition that each city is visited only once, meaning that only one binary variable in each column equals 1, and the others are 0. Therefore, we first transform the TSP into a constrained optimization problem:

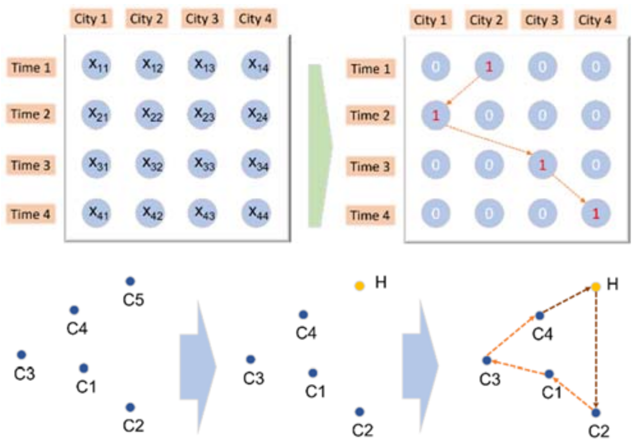

Fig. 1. Traveling Salesman Problem (n = 5) variable presentation form. The arrangement of the remaining cities is expressed using an $n \times n$ matrix, where columns represent the cities and rows represent different time sequences.

Problem TSP:

Minimize

$$J_{\text{TSP}}(\mathbf{x}) \equiv \frac{1}{L_{\text{avg}}} \left(\sum_{t\in T\backslash\{m\}} \sum_{i\in C} \sum_{j\in C\backslash\{i\}} d_{ij} x_{ti} x_{(t+1)j} + \sum_{t\in\{1,m\}} \sum_{i\in C} c_i x_{ti} \right) \tag{6}$$

subject to

$$\sum_{j\in C} x_{tj} = 1, t \in T, \quad \sum_{t\in T} x_{tj} = 1, j \in C,$$

$$x_{tj} \in \{0,1\}, t \in T, j \in C. \tag{7}$$

The above two types of constraints are transformed into the following minimization regularization terms, which require the sum of each row and each column to be close to 1. Therefore, the number of added regularization terms will be the sum of the number of rows and columns, 2m. However, currently, it is not required for all solution elements to be binary values; solutions may be floating-point numbers between 0 and 1:

$$\min_{\mathbf{x}} \left(\sum_{j\in C} x_{tj} - 1 \right)^2, \quad \min_{\mathbf{x}} \left(\sum_{t\in T} x_{tj} - 1 \right)^2. \tag{8}$$

According to the regularization method, by adding the above regularization terms, a new cost function $J_3(\mathbf{x})$ is generated:

$$J_3(\mathbf{x}) \equiv J_2(\mathbf{x}) + \lambda_1 \left[\sum_{t\in T} \left(\sum_{j\in C} x_{tj} - 1 \right)^2 + \sum_{j\in C} \left(\sum_{t\in T} x_{tj} - 1 \right)^2 \right] \tag{9}$$

Since $J_3(\mathbf{x})$ is a quadratic polynomial function with respect to $\mathbf{x}$, we can transform $J_3(\mathbf{x})$ into matrix form:

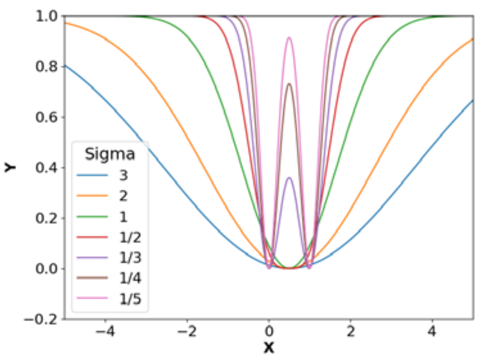

Fig. 2. The normalized regularization term $g_\sigma(x)$ for several σ.

$$J_3(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x} + r \tag{10}$$

where $\mathbf{P}$ is a symmetric $n \times n$ matrix with each element expressed in d_{ij} and λ_1, $\mathbf{q}$ is an $n \times 1$ column matrix with each element expressed in λ_1, and r is a scalar / constant expressed in λ_1.

C. Formally Treating Binary-valued Variables as Continuous Variables

Heading toward possible adoption of the gradient-descent (GD) based methods, such as the Newton-Conjugate-Gradient method, for searching the minimum, we formally treat the binary-valued variables $\mathbf{x}$ as continuous variables such that $\mathbf{x} \in \mathbb{R}^n$. Since the cost function $J_3(\mathbf{x})$ is quadratic in form, $J_3(\mathbf{x})$ is infinitely differentiable, which guarantees the existence of the gradient matrix and Hessian matrix, that are usually required by GD based methods.

In the continuous domain, an $\mathbf{x} \in \mathbb{R}^n$ does not necessarily represent a feasible route. Only a binary-valued $\mathbf{x} \in \{0,1\}^n$ can represent a feasible route. As $\{0,1\}^n$ is a very sparse subset of $\mathbb{R}^n$, we are interested in heuristics that tend to allow many candidates $\mathbf{x}$ in the continuous domain $\mathbb{R}^n$, so that at later stages we can narrow down to those candidates close enough to the binary-valued domain $\{0,1\}^n$. One such "many-candidate" heuristic for picking λ_1 is described in the following subsection.

D. Picking λ_1 by Heuristics Favoring Near-Singular Systems

The gradient of the cost function $J_3(\mathbf{x})$ is

$$\nabla J_3(\mathbf{x}) = 2\mathbf{P}\mathbf{x} + \mathbf{q}. \tag{11}$$

A necessary condition for the minimum of $J_3(\mathbf{x})$ to happen at $\mathbf{x} = \hat{\mathbf{x}}$ is "zero gradient"

$$\mathbf{P}\hat{\mathbf{x}} = -\mathbf{q}/2. \tag{12}$$

Since the above is a linear equation, if $\mathbf{P}$ is near-singular / ill-conditioned, many $\hat{\mathbf{x}}$ exist such that $J_3(\hat{\mathbf{x}})$ falls within an ϵ-neighborhood of a local minimum of $J_3(\mathbf{x})$. Because $\mathbf{P}$ is expressed in terms of η_1, the following "many-candidate" heuristic (as mentioned in the last subsection) is devised.

Heuristic: Picking η_1 such that $\mathbf{P}$ is near-singular / ill-conditioned, or equivalently, the condition number of $\mathbf{P}$ is large.

E. Cost Functions for Square of Variables

The cost function $J_3(\mathbf{x})$ includes the distance form of $d_{ij}x_{ti}x_{(t+1)j}$, which can turn negative when the variables are in the continuous domain $\mathbf{x} \in \mathbb{R}^n$. We simply substitute each variable by its square to arrive at $d_{ij}x_{ti}^2x_{(t+1)j}^2$ to guarantee positive values. Let

$$\mathbf{x^2} \equiv [x_{11}^2\ x_{12}^2\ \ldots\ x_{1c}^2\ \ldots\ x_{c1}^2\ x_{c2}^2\ \ldots\ x_{cc}^2]^T. \tag{13}$$

The new cost function $J_4(\mathbf{x})$ is chosen as

$$J_4(\mathbf{x}) \equiv J_3(\mathbf{x^2}) = \mathbf{x^2}^T\mathbf{P}\mathbf{x^2} + \mathbf{q}^T\mathbf{x^2} + r \tag{14}$$

F. Using Regularization to Incorporate Constraints Favoring Binary Values

The general framework in reference [1] has inspired the following regularization term $f_\sigma(x)$ for the constraints favoring binary values $\{0, 1\}$.

$$f_\sigma(x) \equiv 1 - \exp\left(\frac{-x^2}{2\sigma^2}\right) - \exp\left[\frac{-(x-1)^2}{2\sigma^2}\right], \tag{15}$$

$$\lim_{\sigma\to 0} f_\sigma(x) = \begin{cases} 0, & \text{if } x = 0 \text{ or } 1 \\ 1, & \text{otherwise} \end{cases}. \tag{16}$$

Note that $f_\sigma(x)$ is symmetric about $x = 0.5$, and thus $f(0) = f(1)$. For larger σ, $f_\sigma(x)$ has one minimum at $x = 0.5$. For smaller σ, $f_\sigma(x)$ has two minima at $x = 0$ and $x = 1$.

However, the range of $f_\sigma(x)$ varies with different values of σ. When σ increases to the point where there is only one minimum, the minimum value of the function can be far smaller than 0, as shown in Fig. 2. To rescale the range of $f_\sigma(x)$ for all possible σ values to lie between 0 and 1, we normalize the regularization term function $f_\sigma(x)$. The following is the formula for normalizing the function:

$$f_{norm} = (f - f_{min})/(f_{max} - f_{min}) \tag{17}$$

where f_{max} remains unchanged with respect to σ, fixed at 1, and f_{min} varies. Therefore, to define the scaling scalar s_σ for the normalized function $f_\sigma^*(x)$, we generate the normalized function. The function plot is depicted in Fig. 2, where the curve represents the results for several different σ values.

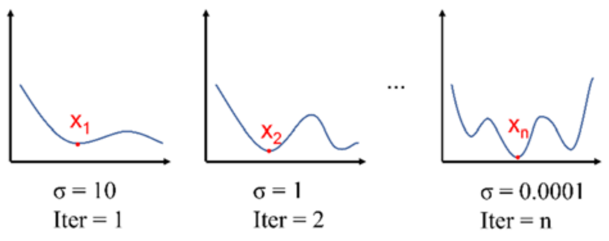

Fig. 3. Visualization of Annealing-Inspired method.

$$s_\sigma \equiv 1 - \min_{x \in \mathbb{R}} f_\sigma(x)$$

$$= \&\begin{cases} 1 - f_\sigma(0) \ , & f_\sigma(0.5) > f_\sigma(0) \\ 1 - f_\sigma(0.5), & \text{otherwise} \end{cases}, \tag{18}$$

$$f_\sigma^*(x) \equiv (f_\sigma(x) + s_\sigma - 1)/s_\sigma. \tag{19}$$

G. *Annealing-Inspired: Slowly Decreasing σ in Attempts to Escape Local Minima*

The cost function $J_\sigma(\mathbf{x})$ may have many local minima. In Fig. 3, the regularization term $g_\sigma(x)$ is more like a line for larger σ. It is flatter, has lower convexity, less favors binary $\{0, 1\}$, and renders fewer local minima for $J_\sigma(\mathbf{x})$. For smaller σ, $g_\sigma(x)$ has larger convexity, much favors binary $\{0,1\}$, and renders more local minima for $J_\sigma(\mathbf{x})$. In attempts to escape local minima, σ is slowly decreased in each optimization iteration, in which the cost function $J_\sigma(\mathbf{x})$ is optimized using a GD based method.

III. Experimental and Results

This chapter is divided into four sections. The first section illustrates the differences between the algorithm proposed in this paper and existing algorithms. The second section tests the algorithm model with parameter adjustments for the Traveling Salesman Problem (TSP) across various dimensions. The third section presents the results and comparisons of finding the shortest path in a randomly distributed city model. The final section uses the renowned TSPLIB dataset as the benchmark to showcase the algorithm's results and advantages over other algorithms [5]. The algorithm is implemented in Python 3.11 without GPU acceleration. The hardware specifications are an Intel Core i5-7200U with 12GB RAM.

A. *Evaluation of Time Complexity*

The Traveling Salesman Problem is an NP-hard problem, meaning it cannot be solved in polynomial time. Its time complexity increases at least exponentially. For instance, for the Traveling Salesman Problem with n cities, there are $(n-1)!/2$ possible routes. Finding the shortest route among them requires a time complexity of $O(n!)$, known as factorial time. Algorithms of this nature are often referred to as exhaustive or brute-force methods. As the input n increases, the execution time increases dramatically. Just for 11 cities, there are 1,814,400 routes. While cases up to 12 cities can be handled conventionally, 17 cities require the most powerful computational resources available today. However, problems involving 21 cities are nearly impossible to solve.

To mitigate the excessively large computational complexity, dynamic programming can be employed. This approach breaks down the entire Traveling Salesman Problem into several subproblems, decomposing the entire route into multiple subpaths. Trading space complexity for reduced time complexity is highly effective, reducing the time complexity to $O(n^2 2^n)$. We can observe that beyond 10 cities, dynamic programming's complexity becomes lower than brute-force methods. By the time 20 cities are reached, there is an eight-order-of-magnitude difference between the two approaches. As shown in Fig. 4, this difference accelerates as the dimensionality increases. However, dynamic programming still exhibits exponential complexity and cannot achieve the polynomial time complexity required by modern integrated circuit computers.

The algorithm we developed is based on the matrix P for optimizing computations, with a dimensionality of n^4. The theoretical time complexity is $O(n^4)$. However, during the search for the minimum value, we utilize the Newton conjugate gradient method. Rather than delving into the details of algorithm analysis, we chose a more efficient approach by the program execution time calculations to assess our algorithm's time complexity. This way systematically analyzed the potential factors affecting time complexity and observed the impact as the number of cities increases.

In Fig. 4, we observed that our ACUO algorithm's computation time grows with the dimensionality, roughly matching $O(n^4)$. We also analyzed the computation times of two exact solving algorithms, exhaustive search (the brute-force method), and dynamic programming, using results from the package of Python-tsp. They exhibit a trend like the estimated time complexity. Hence, these results can serve as the evaluation standard for our algorithm's time complexity. Despite not being outstanding in terms of computational speed, the time complexity remains within polynomial time complexity. Thus, the problem with high city number can find a solution in finite time for our algorithm.

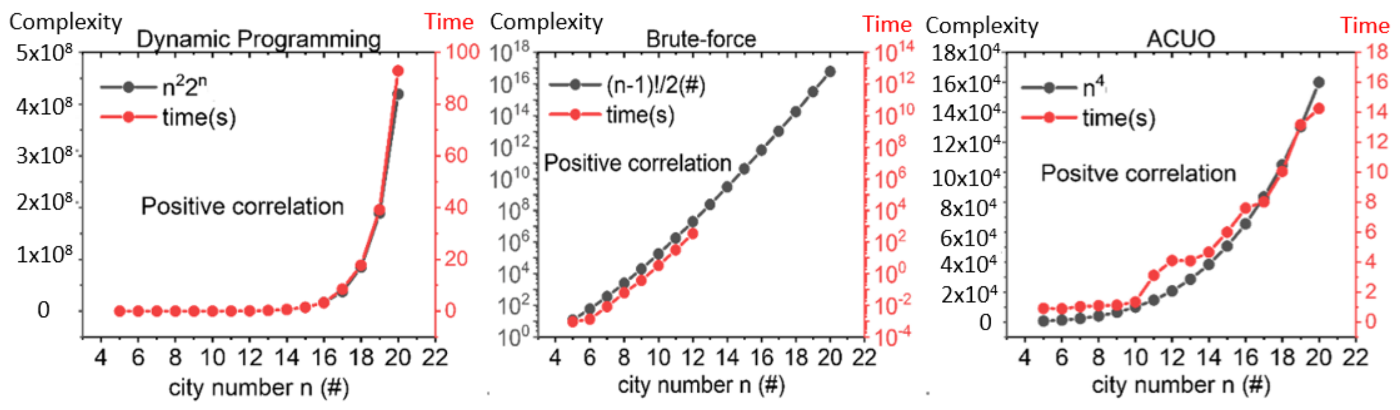

Fig. 4. Comparison of the computation time (red lines) and the complexity (black line) of the (Left) in Dynamic Programming, (Middle) Brute-force, and (Right) proposed ACUO methods. One can see that the proposed algorithm can much reduce the computation time and complexity.

TABLE I. Comparison of ACUO Results with Standard Solutions for Randomly Distributed Cities.

N	ACCU(%)	Brute Force	ACUO	DIFF	DIFF%
5	94	0.8169	0.8186	0.00171	0.2093
6	89	0.7451	0.7465	0.00144	0.1935
7	87	0.6816	0.6833	0.00168	0.2458
8	83	0.6359	0.6381	0.00226	0.3549
9	78	0.5879	0.5910	0.00307	0.5225
10	77	0.5531	0.5562	0.00309	0.5579
11	70	0.5251	0.5285	0.00344	0.6558
12	55	0.5027	0.5100	0.00732	1.4553
13	42	0.4806	0.4902	0.00957	1.9918
14	36	0.4561	0.4676	0.01154	2.5306
15	37	0.4369	0.4511	0.01414	3.2366

B. *Results of ACUO for Random Cities Distribution*

We extensively tested the optimization algorithm using randomly distributed city data, comparing the optimized paths obtained by our algorithm with the exact solutions obtained by dynamic programming. We examined the accuracy and error of the ACUO in low dimensions. We tested city numbers ranging from $n = 5$ to $n = 15$ in various scenarios, generating 100 sets of test data for each scenario using random distributions. The parameters were selected based on the optimal combinations obtained in the previous section. The results are presented in Table I and Fig. 5.

TABLE II. Optimization path length comparison using the TSPLIB data set.

Dataset	*n*	ACUO	SA	MIN	ERR (%)
gr24	24	1340	1350	1272	5.35
bays29	29	2070	2036	2020	2.48
berlin58	58	8006	8135	7542	6.15
eil51	51	450	452	426	5.63

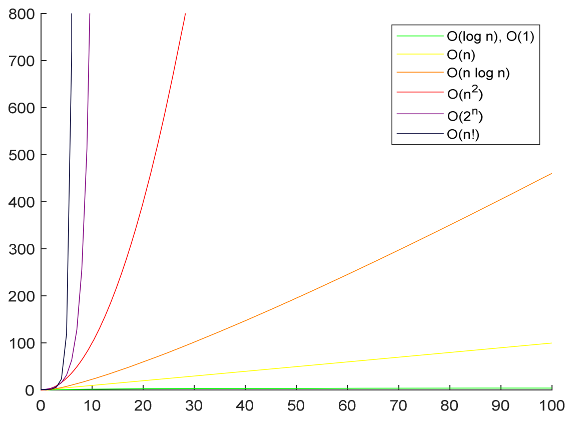

Fig. 5. The time complexity function curve graph.

Considering the error in path length, the difference between the optimized path length obtained by our algorithm and the shortest path is minimal. Regardless of the dimension, the error remains within 6.15%. From the results in Table II and Fig. 6, one can see that there are no obvious undesirable paths with unnecessary overlap. Therefore, the paths selected by our algorithm are considered to be satisfactory choices.

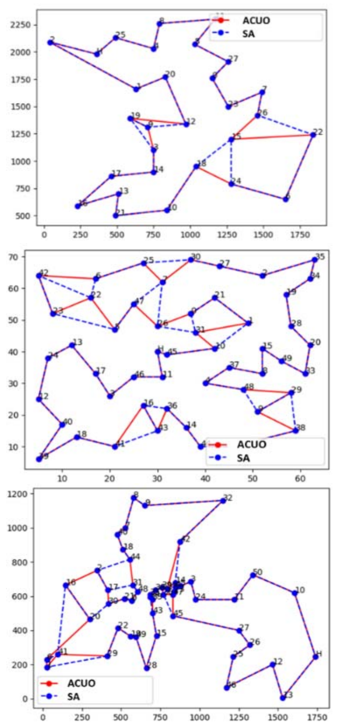

Fig. 6. TSPLIB data set bay29(top), berlin58(middle) and eil51(bottom) optimization path comparison chart

C. Results and Comparison of TSPLIB Test Sets

In order to verify whether good results can be achieved even with a higher number of cities and not limited to randomly distributed cities, we obtained five test datasets from TSPLIB for optimization. We compared the results with the simulated annealing (SA) algorithm in Python, as well as with the known shortest path lengths provided by the database, calculating the errors. The results, as shown in TABLE II, indicate that the path lengths obtained by ACUO are close to those of the simulated annealing method, with some parts performing better and others slightly worse. For datasets with city coordinates provided, the relative positions of cities can be plotted to visualize the differences between the paths generated by ACUO and SA in Fig. 6. Compared to the ground truth, the errors in path length compared to the provided shortest path lengths are also within 6.15%.

IV. Conclusions

We introduce the Annealing-inspired Continuous Unconstrained Optimization (ACUO) algorithm, designed to tackle combinatorial optimization problems by transforming them into polynomial function optimization in the continuous domain. This structural innovation sets ACUO apart from traditional discrete iterative methods, enabling the integration of both discrete and continuous constraints, thereby addressing system stability and facilitating the escape from local minima. Using the Traveling Salesman Problem (TSP) as a case study, we demonstrate that ACUO achieves polynomial growth in time complexity, effectively mitigating the curse of dimensionality. While the algorithm maintains high accuracy for low-dimensional problems, the accuracy diminishes as the number of cities increases. Nonetheless, the maximum error is within 6.15%, and path visualization confirms the validity of the solutions. ACUO exhibits broad applicability, a unique model architecture, and the capability to solve combinatorial problems within a specific margin of error.

References

[1] Larranaga, P., et al., Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial intelligence review, 1999. 13: p. 129-170.

[2] Finnila, A.B., et al., Quantum annealing: A new method for minimizing multidimensional functions. Chemical Physics Letters, 1994. 219(5): p. 343-348.

[3] Mohimani, H., M. Babaie-Zadeh, and C. Jutten, A fast approach for overcomplete sparse decomposition based on smoothed L0 norm. IEEE Transactions on Signal Processing, 2008. 57(1): p. 289-301.

[4] Diaby, M., The traveling salesman problem: a linear programming formulation. arXiv preprint cs/0609005, 2006.

[5] Reinelt, G., TSPLIB—A traveling salesman problem library. ORSA journal on computing, 1991. 3(4): p. 376-384.

[6] Margalit, D., J. Rabinoff, and L. Rolen, Interactive linear algebra. Georgia Institute of Technology, 2017.

[7] Hoerl, A.E. and R.W. Kennard, Ridge Regression: Applications to Nonorthogonal Problems. Technometrics, 1970. 12(1): p. 69-82.

[8] Louizos, C., M. Welling, and D.P. Kingma, Learning Sparse Neural Networks through L0 Regularization. ArXiv, 2017. abs/1712.01312.

[9] Papadimitriou, C.H. and K. Steiglitz, Combinatorial optimization: algorithms and complexity. 1998: Courier Corporation.

[10] Jünger, M., G. Reinelt, and G. Rinaldi, *The traveling salesman problem.* Handbooks in operations research and management science, 1995. **7**: p. 225-330.

[11] Applegate, D.L., et al., The traveling salesman problem. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2006: p. 1-5.