An Improved Genetic Algorithm With Euclidean Geometry and Hybrid Heuristic Crossover for Traveling Salesman Problem

Bryan Zhu
Bellevue High School
Bellevue, Washington, USA
bzhu1945@gmail.com

Wei-Fang Xie
Department of Mechanical, Industrial & Aerospace, Concordia University,
Montreal, Canada
wfxie@encs.concordia.ca

2023 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-1458-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/CEC53210.2023.10254179

***Abstract*—The traveling salesman problem (TSP) is a classic NP-hard combinatorial optimization problem that has drawn extensive research endeavor due to its theoretical significance and a wide variety of applications. This paper proposes an improved genetic algorithm with two novel strategies. The first strategy uses a Euclidean geometry-based approach to construct an initial population of high-quality solutions through a greedy convex hull algorithm. This strategy helps to effectively reduce the search space and focus on promising regions. The second strategy is a hybrid heuristic crossover operator that combines a new recombination-insertion crossover with a well-known order crossover (OX). Our test results show that either strategy significantly enhances the performance of the classic GA. Moreover, combining both approaches demonstrates a strong synergistic effect for further improvements by outperforming several GA variants and a simulated annealing approach in terms of error rates and consistency of the results.**

Keywords—traveling salesman problem, convex hull, Euclidean geometry, recombination insertion crossover, genetic algorithm, metaheuristics

I. INTRODUCTION

The traveling salesman problem (TSP) aims to optimize the shortest cyclic route that traverses through a set of *N* cities and visits each city exactly once, given by:

$$F(S) = min\{\sum_{i,j\in S} dist[P_i][P_j] + dist[P_N][P_1]\} \quad (1)$$

where $[P_1, P_2, \ldots P_N]$ is a permutation of the cities and *dist*[][] is a matrix of distances or costs between pairs of cities. The cost here can have a variety of definitions, depending on the application. In this study we may use the terms points and cities interchangeably.

TSP and its variants have been studied broadly due to both its theoretical and practical value. As a well-known NP-hard problem, TSP does not have a polynomial solution. The approaches for TSP can be divided into two major categories: exact approaches and approximation approaches. Exact algorithms can find the absolute optima, but its time complexity increases dramatically with the input size, making large-scale problems infeasible.

Approximation algorithms typically employ heuristics to guide the search on specific sub-areas instead of exploring the entire search space. These algorithms do not guarantee global optimality, but they usually provide sufficiently good solutions within a reasonable time frame. Among a multitude of approximation algorithms, the most popular ones are genetic algorithms (GA) [4], simulated annealing (SA) [5] and ant colony [6]. Also called metaheuristics, these algorithms generate one or more initial solutions and iteratively refine them in an attempt to improve their quality.

In this study, we propose an improved GA for TSP with two contributions below:

1) the population initialization: we use a greedy convex hull approach to effectively build solutions with good quality.

2) the crossover: we combine a popular OX operator [11] with our new recombination insertion crossover (RIC). Instead of randomly seleting crossover points, RIC builds offspring deterministicly from parents. This new hybrid heuristic crossover demonstrates an excelling performance compared to OX alone.

The remainder of this paper is organized as follows. Section II introduces the idea of using convex hull for TSP. Section III reviews a GA and an adaptive SA algorithm, which is compared with our GA. Section IV delves into our GA with detailed explanations. Section V presents the experimental results and analysis. Section VI concludes the paper with a summary of the experimental data.

II. CONVEX HULL AND TSP

A convex hull is the minimum set of points that form a convex shape and enclose all other points of the set in a Euclidean space. The well-known Graham's algorithm [7] constructs the convex hull from *N* points with a rotational sweep technique of complexity O(*NlogN*). This algorithm sorts *N*-1 points by their polar angles in counterclockwise order around the bottom-left-most point *P*. Initialized with *P* and the first 2 points in the sorted list, a hull is progressively updated by removing interior points or appending new points that do not introduce concavity.

For a Euclidean TSP problem, an optimum route does not intersect itself [8]. As such, the vertices on the convex hull should be visited following the cyclic order because any other sequence not following this order would inevitably result in one or more intersections.

Takenaka and Funabiki [4] proposed a randomized convex hull approach to generate an initial route. They identified all cities on the boundary of the hull, marked them with 0 placeholders, and shuffled them along with the interior cities. The placeholders were then replaced with the convex hull cities. This approach partially orders the cities on the hull, while leaving all other interior cities fully randomized.

Meeran and Shafie [9] propose a method to group cities into circular neighborhoods using the edges of a convex hull as the circles' diameters. A hierarchical convex family tree is recursively built from the cities not belonging to any neighborhood or belonging to multiple neighborhoods. Each neighborhood is optimized locally. Adjacent neighborhoods are linked in the order of the initial convex hull.

Kai and Mingrui [10] propose a method to partition cities into nested non-intersecting convex hulls and less than 3 residue points, all of which are then joined to form a complete tour.

III. Metaheuristic Algorithms

A. Genetic Algorithms (GA)

Inspired by Charles Darwin's 'survival of the fittest' theory, a GA is a computational technique that simulates the process of natural selection using biological operators such as elitism, crossover, and mutation. A GA typically starts from an initial population of chromosomes, each of which has a fitness function that expresses the quality of the chromosome. In the context of solving the TSP, most researchers use a list of permuted cities as a chromosome, for which the length of the tour is the fitness.

The high-level algorithm is described with the following notations:

- M: population size
- P_i: chromosome i of the population
- G_m: max number of generations
- P_e: elitism rate
- P_c: crossover rate
- P_m: mutation rate

In GAs, crossover is a critical genetic operator used to create new offspring by combining the genes of two parents. One of the most commonly used operators for TSP is the order crossover (OX) [11]. An OX works by copying a random segment of one parent into the offspring and filling the remaining spots while maintaining their relative order in the other parent, as illustrated in Fig. 1. Empirical studies show that OX outperforms other crossover operators [12].

Fig. 1 Order crossover (OX)

Algorithm 1 High Level GA Pseudocode

Input: List of cities with 2D coordinates,
distance matrix between cities
Output: A tour with the best fitness

1 Initialize population $\{P_1, P_2, \ldots, P_M\}$
2 $gen = 0$
3 **While** $gen < G_m$:
4 Sort the population by their fitness values
5 Copy top $M \times P_e$ chromosomes into the next generation
6 **Repeat** $M \times P_c$ times:
7 Create offspring with two random parents
8 **End Repeat**
9 **Repeat** $M \times P_m$ times:
10 Randomly select a chromosome i and mutate
11 **End Repeat**
12 **End While**

13 Return the best chromosome in the population

The Alternating Edges crossover (AEX) on the other hand creates an offspring by concatenating edges from both parents alternately. If there is a conflict, a random unvisited city is added to the tour and the same process is repeated until a complete route is built. Some researchers believe that AEX works best as a single operator [13].

A recent Generalized Partition Crossover (GPX2) [14] is a deterministic operator that decomposes the union graph of two parents' tours into multiple candidate components. Some neighboring infeasible components are merged into recombining components. An offspring is then generated by independently selecting the shortest route inside each recombining component. GPX2 doesn't generate any new edges, but its offspring inherits edges from either parent. As such both the solution quality and the run time heavily depend on the parents' quality and similarity.

An empirical study of a variety of crossover operators demonstrates that combining multiple crossover operators may generate better results than a single operator [15]. Our hybrid heuristic crossover is inspired by this idea.

Mutation is another essential genetic operator used to introduce diversity into the population. The most popular mutation operator for TSP is 3-opt [16]. 3-opt randomly splits the original route into 3 segments and re-connects them. In our study, we use 3-opt for all of our GA variants.

B. Simulated Anealing (SA)

In a traditional Simulated Annealing (SA) approach, inferior solutions are either immediately discarded or accepted with a small probability. However, Geng et. al [5] presented an adaptive approach that retains a pool of those solutions. When the pool size reaches a certain threshold, the current solution X is replaced with the best solution Y from the pool with a probability P_r,. The value of P_r is given by:

$$P_r = e^{-(f(Y)-f(X))/T_{current}\times(10\times N/f(best))} \quad (2)$$

where the function $f(X)$ is the cost of the tour X. $f(best)$ is the cost of the best tour found so far.

The authors used three local search operators to mutate the current solution, a random vertex insertion, a block insertion, and a block reversion.

IV. GA With Euclidean Geometry and Hybrid Heuristic Crossover

A. The Greedy Convex Hull Approach

Fig. 2 illustrates how we create a tour of 10 points from a convex hull, which consists of 5 points and hence an initial partial route S from P_1 to P_5. There are 5 interior points from P_6 to P_{10}. Unlike the work in [4] which randomly shuffles these points and merges them with S, we use a greedy approach to insert each interior point P_j into the partial tour at the best possible position. The incremental cost caused by the insertion between an arbitrary pair of cities P_i and P_{i+1} along the tour is minimized by equation (3) below.

$$\Delta = min\big(dist[P_i][P_j] + dist[P_j][P_{i+1}] - dist[P_i][P_{i+1}]\big) \quad (3)$$

$$where\ i,j \in [1,N], P_i \in S,\ P_j \notin S,\ S\ is\ \text{a partial tour.}$$

The pseudo code for Algorithm 3 shows how this process works.

Algorithm 2 A Greedy Convex Hull Approach for Population Initialization

Input: a set of N cities $\{1, 2, 3 \ldots N\}$ and their coordinates in the plane
Output: an initial tour of the cities

1 Generate a convex hull C of size K $\{P_1, P_2, \ldots P_k\}$
2 Fitness of the partial loop tour S:

$$f = \sum_{i=1}^{k-1} dist[P_i][P_{i+1}] + dist[P_k][P_1] \quad (4)$$

3 Shuffle all the other interior points P_j
4 **For each** interior points P_j
5 Insert P_j into S with the best cost Δ given by equation (3)
6 $S = S + P_j$
7 $f \mathrel{+}= \Delta$
8 **End For**

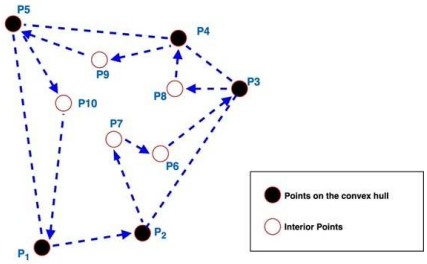


Fig. 2 A complete tour built on top of a convex hull

B. The Hybrid Heuristic Crossover

The hybrid heuristic crossover consists of two operators: a popular order crossover (OX) [11], and a new recombination insertion crossover (RIC). RIC works in two phases. Phase I recombines the edges of both parents appropriately, while phase II inserts missing points into the partial routes obtained from Phase I. Let's assume the two parents A and B have the tours $\{a_1, a_2, a_3, \ldots a_N\}$ and $\{b_1, b_2, b_3, \ldots b_N\}$, which are permutations of cities 1 through N. Without loss of generality, we can assume that $a_1 = b_1$. Note that if they are different, we can always rotate one of the parents to start from the same city as the other parent. We observe that if the tours start from different cities, the crossover may have a destructive impact on good building blocks or sub-tours.

The offspring P and Q are initialized with a_1 and b_1, respectively. Then the next cities a_i and b_i from both parents are processed. Let P be more aggressive and always pick the better option from either a_i or b_i, denoted as X, if the last city in P has a smaller distance to X, and X is not used in P yet. Similarly, the other city not selected by P is appended to Q if it is not used in Q. After that, P and Q may still be incomplete, so we use the same insertion operation as described in the previous section to complete the tours.

It is worth noting that although Q seems to accept the worse candidate at each step, its fitness is not necessarily worse than P because P is aggressively built with local preferences. We retain both P and Q in order to decrease the probability of getting trapped in local optima due to the greedy nature of the process. Another consideration is the efficiency of the overall algorithm. The worst-case complexity of our approach is $O(N^2)$. To ensure efficiency, we generate two offspring at each crossover operation. If we only generated one offspring, we would need to double the quantity of crossover operations to obtain the same number of offspring given the same crossover rate, which could be unnecessarily expensive. P, Q and their parents are ranked by their fitness values. The best two are put into the new population. If a parent is already in the new population, it is skipped for ranking.

Fig. 3 gives an example of this crossover. It can be seen that P misses city 4 and Q misses city 3 after phase I. Both can be inserted into their best positions in the partial routes using the criterion defined in equation (3).

For the worst case, RIC has an $O(N^2)$ complexity, slower than OX with an $O(N)$ complexity. We performed an empirical study to determine the appropriate mix ratio. The results are presented in Section V.

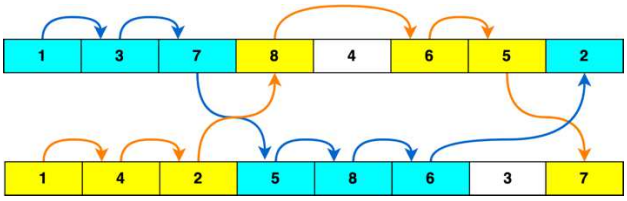


Fig. 3 An example of the recombination insertion crossover

Algorithm 3 Recombination Insertion Crossover

Input: two selected parents A and B
Output: two new offspring P and Q

1 $P = \{a_1\}$, $Q = \{b_1\}$
2 $f_P = 0$, $f_Q = 0$
3 **For** $i = 2 \ldots N$
4 // a_i always holds the better candidate locally.
5 **If** $dist[p_{i-1}][a_i] > dist[p_{i-1}][b_i]$
6 swap a_i and b_i
7 **End If**
8 **If** $a_i \notin P$
9 $P = \{P \leftarrow a_i\}$
10 f_P += $dist[p_{i-1}][a_i]$
11 **End If**
12 **If** $b_i \notin Q$
13 $Q = \{Q \leftarrow b_i\}$
14 f_Q += $dist[q_{i-1}][b_i]$
15 **End If**
16 **End For**

17 **For** $i = 1..N$
18 **If** $i \notin P$
19 $P = \{P + i\}$ with minimum cost Δ given by (3)
20 $f_P = f_P + \Delta$
21 **End If**
22 **If** $i \notin Q$
23 $Q = \{Q + i\}$ with minimum cost Δ given by (3)
24 $f_Q = f_Q + \Delta$
25 **End If**
26 **End For**

The pseudocode for an RIC is shown along with the following notations:

- a_i and b_i: cities of parents A and B, respectively.
- p_i and q_i: cities of offspring P and Q, respectively.
- f_P and f_Q: fitness of P and Q, respectively.
- $\{S + i\}$: a city i is inserted into a partial tour S at best position.
- $\{S \leftarrow i\}$: a city i is appended to the end of the partial tour S.

V. Experimental Results

We ran three sets of experiments. The 1st experiment determines an appropriate ratio of OX and RIC. The 2nd experiment compares 5 GA variants. The 3rd experiment compares 4 better GA variants with an SA approach [5].

The GA variants differ mainly by their population initialization and crossover approaches with the following abbreviations:

- Population initialization:
  - RP: simple random permutation
  - RCH: randomized convex hull approach [4]
  - ICH: our improved convex hull approach
- Crossover:
  - OX: order crossover
  - HHC: our hybrid heuristic crossover

The GA variants are:

- Classic GA with OX (CGA)
- GA with ICH (IGA)
- GA with HHC (HGA)
- GA with both strategies above (IHGA)
- GA with RCH and HHC (RHGA)

IGA and HGA aim to independently explore the improvements introduced by ICH or HHC from CGA. RHGA and IHGA are compared to contrast the performance discrepancy brought by different population initialization approaches - RCH and ICH. For convenience, Table I summarizes the differences of these GAs.

We used the same elitism rate (5%), crossover rate (75%), and mutation rate (20%) for all GAs. We used a mutation rate higher than the traditional rate of less than 10% because our convex hull approach and hybrid crossover strategy amplify the effect of shrinking the search space, so a higher mutation rate is expected to boost the diversity of the population.

The error rate in this study is given by:

$$Error\_rate\% = (actual - optimum)/optimum \times 100\% \quad (5)$$

TABLE I. Population Initialization and Crossover of GA Variants

Algorithm	Population Initialization			Crossover	
	RP	RCH	ICH	OX	HHC
CGA	√	×	×	√	×
IGA	×	×	√	√	×
HGA	√	×	×	×	√
RHGA	×	√	×	×	√
IHGA	×	×	√	×	√

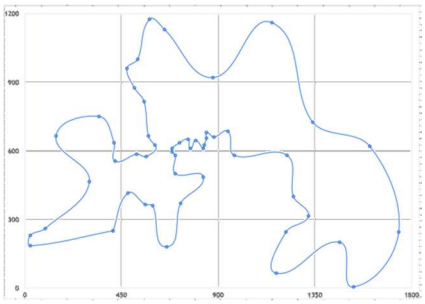


Fig. 4 A perfect tour found for berlin52

All the test data comes from TSPLIB [17], a well-known library for TSP benchmark cases. Fig. 4 shows a perfect tour found by our algorithm for a tour of 52 cities from the dataset berlin52.

Different test cases may have different population sizes and maximum generations. The experiments ran on a mac PC with a 2.3 GHz Dual-Core Intel Core i5 CPU and 8 GB RAM. Each test case runs 20 times and an average is obtained from these runs. The code was written in C++.

A. What makes a good composite crossover

The overall effectiveness of RIC is demonstrated by the improved error rates along with increased portion of RIC in Table II. The group of 40%:60% has the second-best error rate but its runtime is close to that of 30%:70% in Fig. 5. Although the groups of 50%:50% and 70%:30% have slightly better error rates, their runtimes are elevated quite a bit. Hence we used the ratio of 40%:60% for subsequent experiments.

B. Comparison of GA variants

Table III shows the same population size and max generations used by all the GAs except for CGA-7500, which ran for 7500 generations.

Table IV shows the error rates and the CPU runtime in seconds for 5 GA variants, respectively. It turns out that with the same population size and maximum generations, CGA performs poorly compared to other GA variants as its error rate is not even of the same magnitude as others.

CGA running longer with 7500 generations improves the error rates a lot. Nevertheless, the results are still inferior to the results of other variants, and the runtime is much longer. Clearly all 4 other GA variants (IGA, HGA, RHGA, IHGA) have significant improvements from CGA because they use either the convex hull based approach or HHC, or both.

Since both population size and max generations play important roles in the quality of the solutions, we ran another set of experiments exploring the impact of these two parameters. Fig. 6 shows the error rates of IGA, HGA, RHGA, and IHGA for 3 test cases - rat195, gil262, and d493 with different population sizes and max generations.

At lower generations IHGA and IGA generate significantly better results than HGA and RHGA. This fact suggests that our population initialization approach ICH is much more effective than RP (in HGA) or RCH (in RHGA).

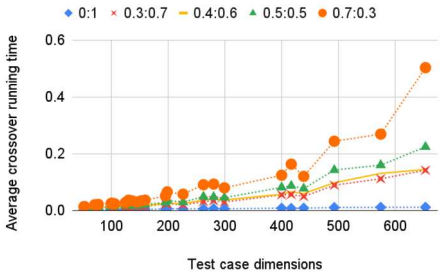


Fig. 5 Average runtime per crossover for different mix ratio of RIC:OX

TABLE II. ERROR RATE FOR DIFFERENT COMBINATIONS OF RIC:OX

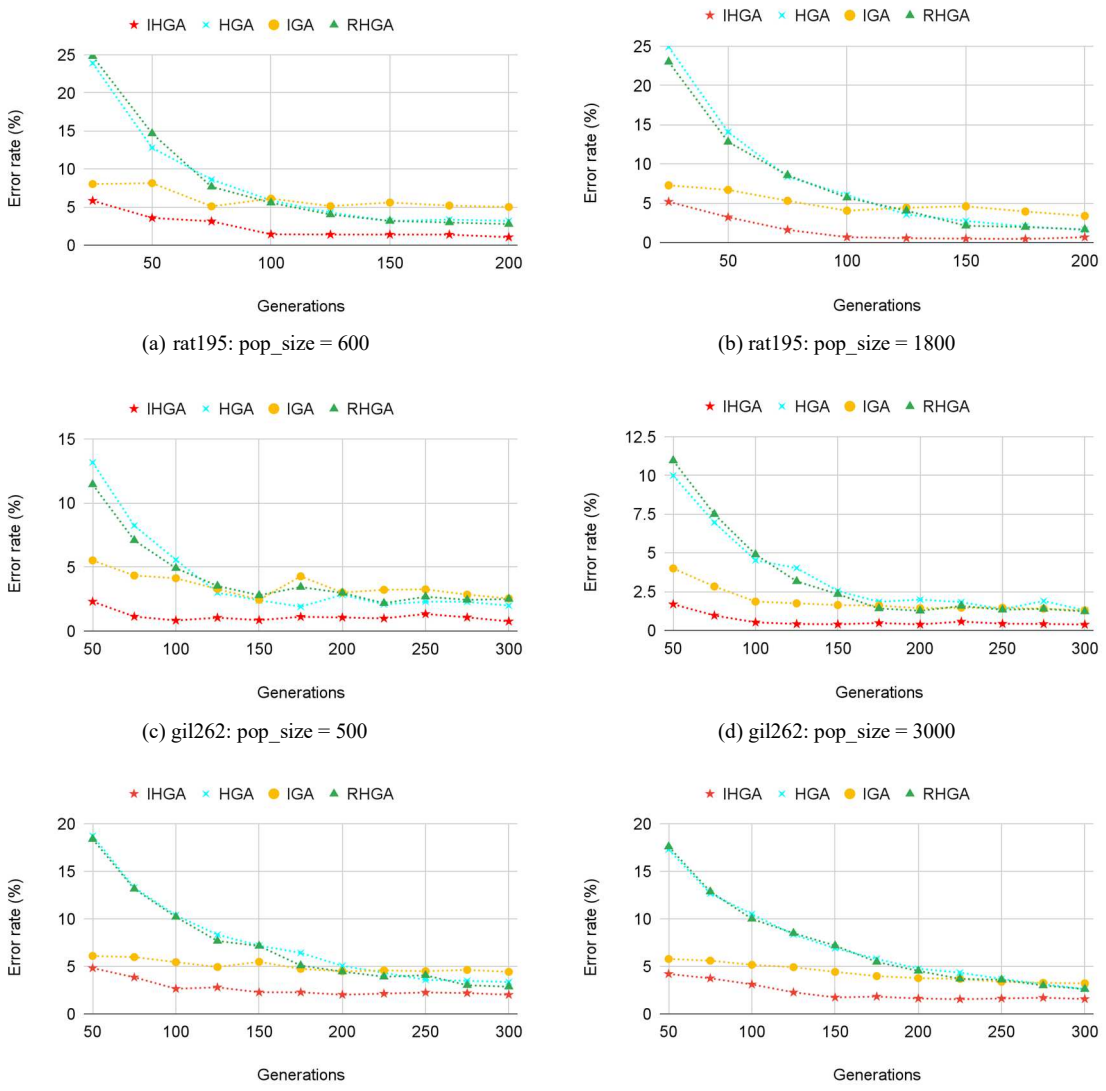
RIC:OX	Error rate (%)
0:100%	1.41
30%:70%	0.59
40%:60%	0.53
50%:50%	0.49
70%:30%	0.49

TABLE III. POPULATION SIZE AND MAX GENERATIONS FOR DIFFERENT TEST CASE SIZE *N*

N	Population Size	Max Generations
<=100	1000	100
<=200	1800	200
<=400	3000	300
<=600	5000	300
>600	6000	300

TABLE IV. PERFORMACE OF 5 GA VARIANTS

Metrics	Test case / Algorithm	berlin52	pr76	kroA150	pr299	pr439
Error rates (%)	CGA	43.33	119.30	271.72	553.96	782.05
	CGA-7500	1.45	1.93	2.84	6.26	7.18
	IGA	0.00	0.90	0.55	0.52	1.77
	HGA	0.00	0.58	0.28	0.96	0.81
	RHGA	0.00	0.46	0.39	0.71	0.97
	IHGA	0.00	0.00	0.00	0.10	0.57
CPU time(s)	CGA	0.08	0.12	0.55	2.20	5.05
	CGA-7500	8.31	10.62	23.20	54.07	137.62
	IGA	0.10	0.18	0.62	2.68	6.88
	HGA	0.21	0.33	2.16	13.17	42.60
	RHGA	0.21	0.32	2.16	13.89	44.79
	IHGA	0.21	0.30	2.06	10.35	28.32



(a) rat195: pop_size = 600

(b) rat195: pop_size = 1800

(c) gil262: pop_size = 500

(d) gil262: pop_size = 3000

(e) d493: pop_size = 1000

(f) d493: pop_size = 5000

Fig. 6 Error rates of 4 GA variants with different population size and max generations for test cases rat195, gil262 and d493

The curves of HGA and RHGA look close, but RHGA is slightly better than HGA, suggesting that RCH approach provides a marginal boost to the solution quality.

Under all circumstances, IHGA consistently shows an outstanding performance against the 4 other GA variants. The relatively constant gap between the error rates of IHGA and IGA demonstrates the additional advantage introduced by HHC. It is interesting to observe that the curve of IHGA looks much flatter than the curve of other GAs because it always initiates from good starting points and tends to stabilize earlier, followed by minor fluctuations.

C. Comparison of GA variants and SA

The SA algorithm [5] relies on 4 parameters: the initial temperature $T_{initial}$, the cooling coefficient T_{cool}, the number of greedy searches (pool size) T_{greedy}, and the end temperature T_{end}. These parameters are given as follows:

$$T_{initial} = 1000$$

$$T_{cool} = \left(\alpha\sqrt{N} - 1\right)/\alpha\sqrt{N} \quad (6)$$

$$T_{greedy} = \beta \times N$$

$$T_{end} = 0.05$$

where N is the number of the cities, and α and β are two empirical parameters. The algorithm runs in 2 stages. In stage 1, $\alpha = 1.6 \times 10^5$ and $\beta = 0.33$. In stage 2, $\alpha = 1.2 \times 10^5$ and $\beta = 1.5$. We observed that a bigger β means a larger pool and generates better results, so we used $\beta = 3$ and $\beta = 5$ instead.

Note that, in equation (2), the authors in [5] used the optimum value of the tour, instead of $f(best)$ as the denominator of the exponent. In this study, we use $f(best)$ for an objective comparison with other algorithms, assuming no prior knowledge is given.

Table V shows the test results of IHGA for 16 additional test cases. For each test case, we have its size, optimum value,

best/worst/average results, standard deviation, and average CPU runtime in seconds. It can be seen that IHGA is able to consistently find an optimum route for 8/16 test cases (underlined). 12/16 of the error rates are less than 0.1%.

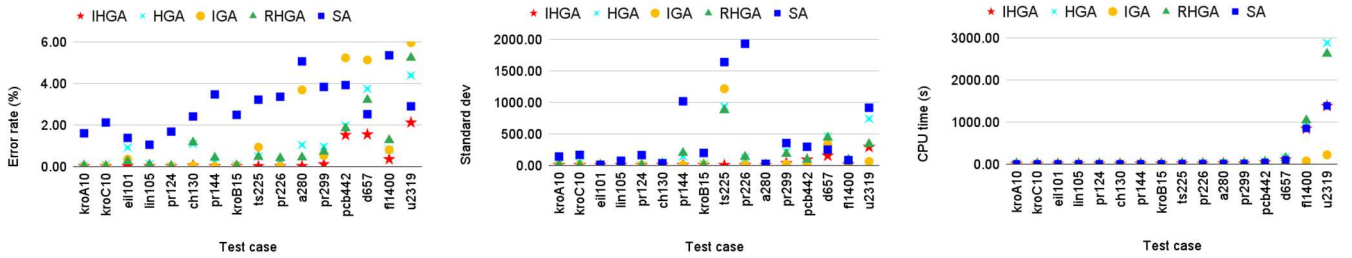
Fig. 7 shows a comparison of the average error rates, standard deviation, and CPU runtime for IHGA, HGA, IGA, RHGA, and SA, respectively, for these 16 test cases. The overall statistics are summarized in Table VI. Note that it's tricky to directly compare the runtime between a GA and an SA because each could have arbitrary termination conditions, and inappropriate conditions could make the program run indefinitely. Therefore we run SA with the approximately the same runtime as IHGA, such that the comparison of two other dimensions is commensurable. Here we use the standard deviation as one of the metrics because it represents the consistency of the algorithm. More varied results indicate that the algorithm is less reliable.

From previous results, we already see the fact that HGA and IGA have much better results than CGA, which means that either ICH or HHC alone plays a crucial role in the improvement of results. HHC has a more impactful contribution on solution quality than ICH, since HGA has a better error rate than IGA. But this improvement comes at the expense of runtime.

Similar to previous results, RHGA performs close to HGA, with a slightly better result. This again indicates that in terms of population initialization, RCH helps with the improvement at a limited scale relative to ICH.

TABLE V. RESULTS OF IHGA FOR 14 ADDITIONAL TEST CASES

Test data	Size	Optimum	Best	Worst	Average	Error rate (%)	Standard dev	Time (s)
kroA100	100	21282	21282	21282	21282	0.00	0.00	0.36
kroC100	100	20749	20749	20749	20749	0.00	0.00	0.36
eil101	101	629	629	629	629	0.00	0.00	1.39
lin105	105	14379	14379	14379	14379	0.00	0.00	1.48
pr124	124	59030	59030	59030	59030	0.00	0.00	1.59
ch130	130	6110	6110	6124	6111	0.03	4.20	1.67
pr144	144	58537	58537	58537	58537	0.00	0.00	2.22
kroB150	150	26130	26132	26143	26133	0.01	3.38	1.90
ts225	225	126643	126643	126643	126643	0.00	0.00	7.72
pr226	226	80369	80369	80369	80369	0.00	0.00	8.13
a280	280	2579	2579	2581	2579	0.01	0.60	11.36
pr299	299	48191	48191	48304	48237	0.10	32.36	10.35
pcb442	442	50778	51380	51667	51543	1.51	87.75	34.41
d657	657	48912	49458	49807	49663	1.54	148.87	102.12
fl1400	1400	20127	20187	20205	20196	0.34	9.58	837.95
u2319	2319	234256	240703	241317	241030	2.89	287.16	1377.18



(a) Error rates (%) (b) Standard deviation (c) CPU time (s)

Fig. 7 Performances of IHGA, HGA, IGA, RHGA and SA for 16 test cases in Table V

TABLE VI. RESULTS OF 4 GA VARIANTS AND SA FOR 16 ADDITIONAL TEST CASES

	IHGA	HGA	IGA	RHGA	SA
Error rate (%)	0.35	0.99	1.41	0.97	2.91
Standard deviation	35.87	191.30	104.48	150.88	452.42
CPU runtime (s)	150.01	243.40	22.72	244.10	152.42

A combination of HHC and ICH in IHGA demonstrates a strong synergistic effect for additional improvements, reflected by its excelling overall average error rate and standard deviation among all these algorithms.

As for the runtime, IGA is the best among the 4 variants. This is expected because although ICH runs at an $O(N^2)$ complexity for the worst case, it only applies to the population initialization, with negligible impact on the overall runtime. In addition, IGA uses OX, which has a linear complexity – faster than HHC (used by all 3 variants: HGA, RHGA, and IHGA).

Although IHGA still runs longer than IGA, it decreases about 38% from HGA and RHGA. This speed-up stems from two aspects. Firstly, unlike HGA, which generates initial solutions randomly, IHGA builds initial solutions on top of a convex hull, which effectively shrinks the search space. Secondly, unlike RHGA, which inserts all interior cities randomly into an initial partial route made of the convex hull, IHGA inserts each interior city deterministically at its best position. Therefore the solutions in IHGA are relatively closer to the optima. Also, two parents tend to "agree" more on each other during crossover, so fewer costly insertions are expected.

Note that we cannot replicate the results of SA in [5]. In this study SA's overall performance is worse than all GA variants. The fact that SA starts from a single random solution and attempts to improve it by local search or mutation can explain the relative low quality and high variance of its solutions.

Based on the results from Table VI, these algorithms can be ranked as follows (the smaller, the better):

- Average error rates: IHGA < RHGA < HGA < IGA < SA
- Average standard deviation: IHGA < IGA < RHGA < HGA < SA
- CPU runtime: IGA < IHGA ≈ SA < RHGA ≈ HGA

VI. CONCLUSION

This paper introduces an improved GA with two novel techniques for the TSP. The first technique is to initialize the population with a convex hull. A partial route is established with cities on the boundary of the convex hull, followed by a greedy insertion of the interior cities with a minimum possible incremental cost. The second technique is a hybrid heuristic crossover which effectively combines a popular order crossover (OX) and a new recombination insertion crossover (RIC). The latter makes sure that two parents start from the same city and picks the next city with a better cost, without discarding the worse option. RIC performs much better than OX but runs slower. A combination of these two operators with a ratio 40%:60% achieves the best trade-off between solution quality and efficiency.

A GA with either the improved convex hull approach or the hybrid heuristic crossover leads to major improvements from a classic GA. The hybrid heuristic crossover introduces more boost than the convex hull approach, at a cost of increased run time. A mix of both approaches demonstrates a strong synergistic effect by outperforming several GA variants and an SA approach with better and more consistent results as well as improved runtime.

REFERENCES

[1] Y. Takenaka and N. Funabiki, "An improved genetic algorithm using the convex hull for traveling salesman problem", IEEE International Conference on Systems, Man, and Cybernetics vol.3, pp. 2279-2284, 1998.

[2] X. Geng, Z. Chen, W. Yang, D. Shi & K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search", Applied Soft Computing, 11(4), 3680-3689, 2011.

[3] P. Du, N. Liu, H. Zhang, & J. Lu, "An improved ant colony optimization based on an adaptive heuristic factor for the traveling salesman problem", Journal of Advanced Transportation, 1-16, 2021.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", Cambridge, MA, USA: The MIT Press, 2009.

[5] R. C. Larson, A. R. Odoni, "Urban Operations Research", Charlestown, MA, USA: Dynamic Ideas, 2007.

[6] S. Meeran, A. Shafie, "Optimum path planning using convex hull and local search heuristic algorithms", Mechatronics, Volume 7, Issue 8, Pages 737-756, 1997.

[7] A. Kai and X. Mingrui, "A Simple Algorithm for Solving Travelling Salesman Problem," Second International Conference on Instrumentation, Measurement, Computer, Communication and Control, 2012.

[8] Kusum Deep & Hadush Adane, "New variations of order crossover for traveling salesman problem", International Journal of Combinatorial Optimization Problems and Informatics, 2011.

[9] Abdoun Otman, Abouchabaka Jaafar, "A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem", International Journal of Computer Applications Volume 31– No.11, 2011.

[10] Wafaa Hameed, "A comparative study of crossover operators for genetic algorithms to solve traveling salesman problem", International Journal of Research - GRANTHAALAYAH. 5, 2018.

[11] R. Tinós, D. Whitley, G. Ochoa, "A new generalized partition crossover for the traveling salesman problem: tunneling between local optima", Evolutionary Computation, 28(2):255-288, 2020.

[12] Hyun-Sook Yoon and Byung-Ro Moon, "An empirical study on the synergy of multiple crossover operators", IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 212-223, April 2002.

[13] M. Mahi, Ö. K. Baykan & H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony optimization and 3-Opt algorithms for traveling salesman problem", Applied Soft Computing, 30, 484–490, 2015.

[14] Gerhard Reinelt, "TSPLIB—A traveling salesman problem library", INFORMS Journal on Computing, INFORMS, vol. 3(4), pages 376-384, 1991.