A Meta Heuristic Genetic Algorithm for Multi-Depot Routing in Autonomous Bridge Inspection

Bryan Dedeurwaerder
Dept. of Computer Science
University of Nevada, Reno Reno, NV, USA
bdedeurwaerder@nevada.unr.edu

Sushil J. Louis
Dept. of Computer Science
University of Nevada, Reno Reno, NV, USA
sushil@unr.edu

***Abstract*—We attack the problem of routing autonomous climbing robots for steel truss bridge inspection using a meta-heuristic genetic algorithm. These robots, deployed from four depots at the four corners of the bridge, must traverse every member of the bridge truss while minimizing distance traveled and balancing tours among robots. This problem maps to the well known NP-Hard Min-Max Multi-Depot k (robot) Chinese Postman Problem. We constructed 20 benchmark bridge instances of four different types of truss configurations using realistic dimensions as a testbed for comparison. Compared to the best known direct encoded genetic algorithm approach, our meta-heuristic genetic algorithm produces routes that are on average 25% better quality, and does so $22x$ faster. On the four depot version of the problem, the MetaGA on average performs 42% better. These results on our benchmarks show evidence our meta-heuristic genetic algorithm provides high-quality tours in realistic time for real-world robot bridge inspection scenarios and has the potential to generalize to vehicle routing and the broader class of arc-routing problems.**

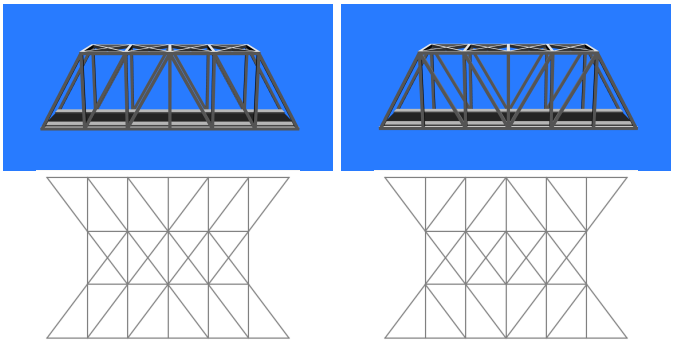
***Index Terms*—MM k-CPP, Genetic Algorithms, Arc Routing, Metaheuristic, Bridge Inspection**

I. Introduction

Bridge inspection is critical to bridge maintenance and public safety. The high cost of bridge inspection in terms of time, resources, and traffic disruption has over time resulted in a significant number of bridges being deemed functionally obsolete in the US [25]. Recent research in climbing and flying autonomous robots for bridge inspection has resulted in a number of designs that can climb bridge trusses in order to inspect truss members through visual, ultrasonic, eddy current and other sensors [23]. Multiple such robots working in parallel can significantly reduce bridge inspection times and costs. Routing such inspection robots to traverse every member of a bridge truss maps to the *NP hard* Multi-Depot Min Max k robot Chinese Postman Problem (MDMM k-CPP). Although there is prior work on single depot min max k-CPPs, this research provides new results on the four depot MM k-CPP. Specifically, this paper compares two genetic algorithm based approaches to attack the MDMM k-CPP problem and shows that a new Meta Heuristic Genetic Algorithm (MetaGA) significantly outperforms a prior Direct Encoded Genetic Algorithm (DEGA) on this problem.

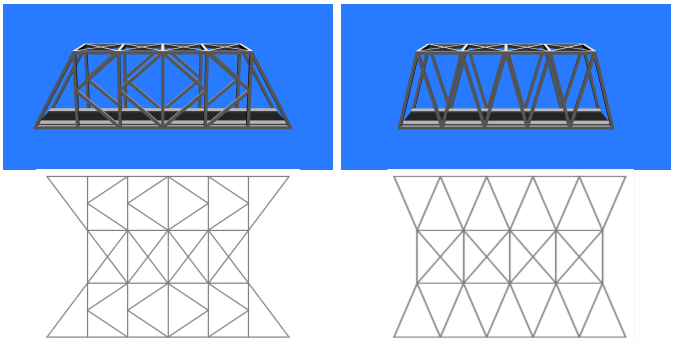
Work on inspection robots has been ongoing for some time [20], [23] and several climbing, rolling, and flying robots and sensors have been developed. Based on this, given a team of climbing robots, this paper addresses the question of routing the robots for efficient bridge inspection. Specifically, *given a team of k bridge inspecting robots, can we generate optimal and balanced routing for each robot, starting at different locations, to minimize total inspection time.* The underlying optimization problem of generating such routes for a team of robots belongs to the family of Arc Routing Problems (ARPs). Although arc routing problems are not as well known as the related traveling salesperson problem, a good approach to arc routing has many applications in transportation and logistics. Specifically, given k robots, generating optimal routes for bridge inspection maps well to the multi-depot NP-hard Min-Max k-Chinese Postman Problem (MM k-CPP). Being NP-hard, finding the globally optimal solution is intractable in reasonable time for large problem instances [6]. The MDMM k-CPP is a well know variant of the MM k-CPP.

This paper therefore describes a new meta-heuristic genetic algorithm approach for attacking arc-routing problems in order to find near-optimal solutions in reasonable time. Genetic and other evolutionary computing algorithm have had much success with NP-hard combinatorial optimization problems, in general, and have been used for arc-routing problems in the past [9], [18], [19]. Unlike other evolutionary computing approaches to the MM k-CPP, MDMM k-CPP, or ogther related routing problems, the MetaGA uses just four simple heuristics encoded in its chromosomes to search through the space of heuristic application sequences to generate optimal tours for k robots inspecting a bridge truss. We also assume that robots start at the four corners (four depots) of a bridge truss corresponding to the most easily accessible locations on the two sides of the bridge span on either side of the road. We generated a set of twenty (20) different truss structures over four (4) different truss types and five (5) sizes each to act as benchmarks. Figure 1 shows sample trusses used in many bridge designs and their mapping to the corresponding MDMM k-CPP graph on which the MetaGA runs. The figure shows how the four extreme vertices correspond to the four



(a) Howe Truss

(b) Pratt Truss



(c) K-Truss

(d) Warren Truss

Fig. 1: Four truss types and their corresponding MDMM k-CPP graphs.

corners of the truss and map to more easily accessible locations on the two sides of the span.

Results using $k = 2, 4$, and 8 robots show that the MetaGA generates routes that are well balanced among the k robots and 25% better in quality compared to DEGA. Furthermore the MetaGA produces these results $22x$ faster than the DEGA, making a significant difference in real-world applications. Furthermore, the MetaGA's performs, on average, 41.72% better than the DEGA on the 4 Depot MM k-CPP (4DMM k-CPP). Finally, the performance difference against the DEGA increases for larger values of k on the 4DMM k-CPP with the difference reaching 61% for $k = 8$. These results show the potential for a meta heuristic genetic algorithm approach for attacking MDMM k-CPPs and other arc routing problems.

The paper contributes four simple new meta-heuristics for arc the MM k-CPP, MDMM k-CPP, and other arc routing problems. In addition, it contributes a set of new benchmarks from bridge routing to the field [8].

The rest of this paper is structured as follows. The next section provides background and prior work in bridge inspection and arc-routing problems. Section III describes our representation and the heuristics used by our meta heuristic genetic algorithm. Subsequently we provide results and discuss possible improvements in heuristics and techniques for improving running time. The last section summarizes our conclusions and gives directions for possible future work.

II. Background and Prior Work

Arc Routing Problems (ARPs) include a broad range of problems with differing constraints. The Chinese Postman Problem (CPP) is polynomially solvable but many of its variants are NP-Hard. These include the Mixed CPP, the Windy Postman Problem, the Rural Postman Problem (RPP), the MinMax k-CPP, the Capacitated Arc Routing Problem (CARP) and many other variants [1], [3], [5], [14], [17], [21], [24]. ARPs are related to the more famous Traveling Salesman Problem (TSP) and the more general Vehicle Routing Problem (VRP), where a set of vertices or nodes of a graph must be traversed in the most efficient manner. In ARPs, we need to traverse instead, the set of *edges* of a graph that may not be fully connected. This class of problems and their solutions have been shown to have multiple practical applications in the real world [3], [14], [17], [21], [24]. We map inspection route traversal by multiple robots to the Multi Depot Min Max k-CPP (MDMM k-CPP) where the objective is to minimize the length of the longest route in order to generate balanced routes staring at multiple depots.

Common approaches to these problems typically use heuristic search. Lacomme et al. examined the use of a Genetic Algorithm (GA) in a bi-objective Capacitated Arc Routing Problem (CARP) [21]. In CARPs, as in our problems, the covering of the graph is split into multiple routes, and the GA was made to minimize both the total cost of the routes and the length of the longest route simultaneously using the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [7]. The GA proved to be a competitive approach to even the single-objective problem, often finding the optimal solution. Additionally, other evolutionary algorithms have seen use in stochastic arc routing problems [12]. For example, memetic algorithms, which combine a GA with local search have produced good results [11], [13], [22].

Closer to the work in this paper, Ahr combined complex domain specific heuristics with tabu search [1], [2]. Unlike that work which assumes a single starting and ending vertex, bridge inspection routing has multiple depots and we distribute our robots equally among depots when the number of robots exceeds the number of depots. More specifically, we assume four depots for the four vertices denoting the four bridge truss anchors on the two sides to be spanned. Moreover, Ahr's heuristics are complex and well suited for arc routing and lead to near-optimal routing for single depot MM k-CPPs. The MetaGA, in contrast, uses four simple heuristics to generate near-optimal routes for the 4-Depot MM k-CPPs considered in this paper.

To the best of our knowledge, the most closely related prior research is work by Harris and Liu on using a direct encoded genetic algorithm to attack MM k-CPPs where the start and end vertices are not pre-specified. Robots may start and end at any graph vertex but must collectively traverse every edge of the graph [16]. The authors used a direct sequential encoding where the chromosome listed edges to be traversed and non-adjacent edges were connected by shortest

routes using Dijsktra's algorithm. They used order crossover and swap mutation to obtain good results on this problem but their approach required impractical run times. We compare our MetaGA performance against this Direct Encoded Genetic Algorithm (DEGA) using their code available on Github [15]. The MetaGA shows significant quality and speed advantages compared to DEGA.

We represent a steel truss bridge as a graph and describe this mapping in the next section. In addition, we specify the heuristics used by our genetic algorithm and their simple binary encoding.

III. Methodology and Heuristics

We have written a bridge truss generator as part of a larger effort in developing simulation training software for future bridge inspection operators. We used this bridge generator to generate the 3D bridges in Figure 1 and can generate four different types of bridge trusses with varying numbers of sections to span distances of varying lengths. The bridge truss to graph mapping is straightforward and we simply map each truss joint to a graph vertex and each truss member to an edge between mapped vertices. The bridge truss generator is available on github and may be used as a problem generator by our research community [4]. Figure 2 shows a better view of the full 3D bridge and its mapping to a problem graph for a small truss with 37 edges and 18 vertices. The mapping is generated automatically so that we may run the MetaGA to generate routes during operator training on inspection scenarios. Vertices and edges are automatically labelled making it easy to feed to an optimizer like the MetaGA and DEGA in order to quickly generate comparable results. We next describe the heuristics used by the MetaGA to generate tours on a mapped bridge truss.

We use four simple heuristics to specify the next edge chosen to be added to a tour. Since two bits can specify one of four heuristics and given a graph with e edges, the chromosome consists of $2e$ bits specifying the sequence of heuristics to apply for choosing the next edge to be added to a tour. The chromosome encodes tours for all k robots and we add the heuristic chosen edge to the current shortest length (or cost) tour. If multiple robots' tours have the same length, we select the first available robot tour.

The decoder algorithm keeps track of edges already in existing tours. Thus, more specifically, the heuristic specifies the next *unvisited* edge (e_i) to be added to the current shortest tour (T_k). Since we cannot guarantee that e_i is adjacent to the last edge of T_k, we add the shortest path to e_i as found by Dijkstra's algorithm to T_k. Additionally, since there are two paths from the end vertex of T_k to the two vertices of e_i, we pick the shorter of these two paths.

The full decoder works by first finding all Dijsktra paths to unvisited edges. These paths are sorted by length (or cost) with shortest paths first. If there exists only one such shortest path, the current heuristic is discarded and this shortest path is added to the tour and we move on to decode the next heuristic in the chromosome. If there are multiple such shortest paths to multiple unvisited edges denoted by the set (E_u), the heuristic picks the path to one of these edges to be added to the tour. The four heuristics pick the edge in E_u with

- The lowest cost
- The highest cost
- The median cost
- A random edge in E_u

If there are no unvisited edges but we have not reached the end of the chromosome, we simply find the shortest Dijkstra paths for each tour back to the depot from which we started to complete the path and then discard the rest of the chromosome. Figure 3 shows the decoding for one allele in the chromosome.

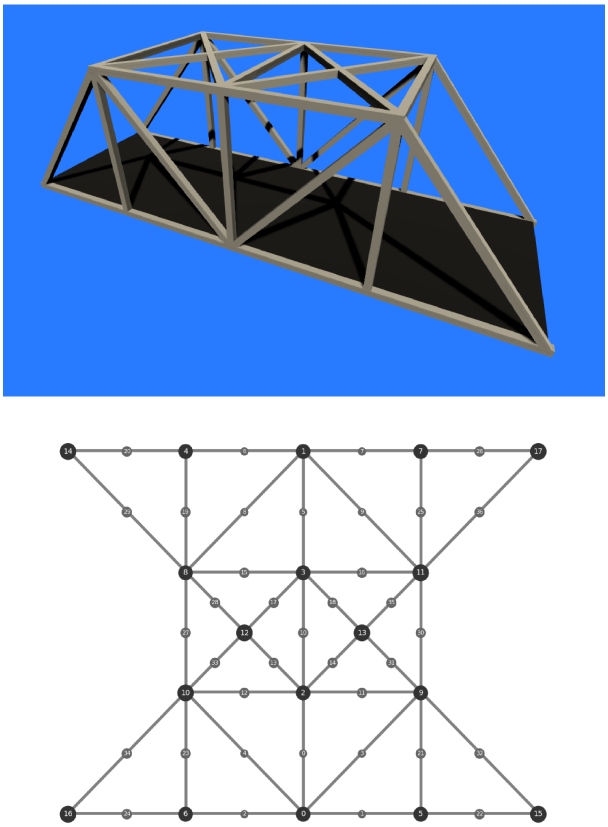


Fig. 2: 3D render of a procedural generated Pratt truss bridge mapped to the corresponding 2D graph.

In the figure, each pair of bits in the chromosome at the top represents a heuristic. The decoded string below the chromosome indicates that heuristic 1 was chosen earlier to build a partial tour. There are currently two partial tours under construction and both tours start at the filled black vertex on the lower left. Tour0 consists of edge 0 followed by edge 1 while tour1 consists of edge 3. The current decoded chromosome heuristic (highlighted) is heuristic 2 which will be used to expand the shorter of tour0 or tour 1. Tour1 is the shorter tour and so heuristic 2 is used to find the next set of edges for tour1. Let us suppose that heuristic 2 picks edge 10.

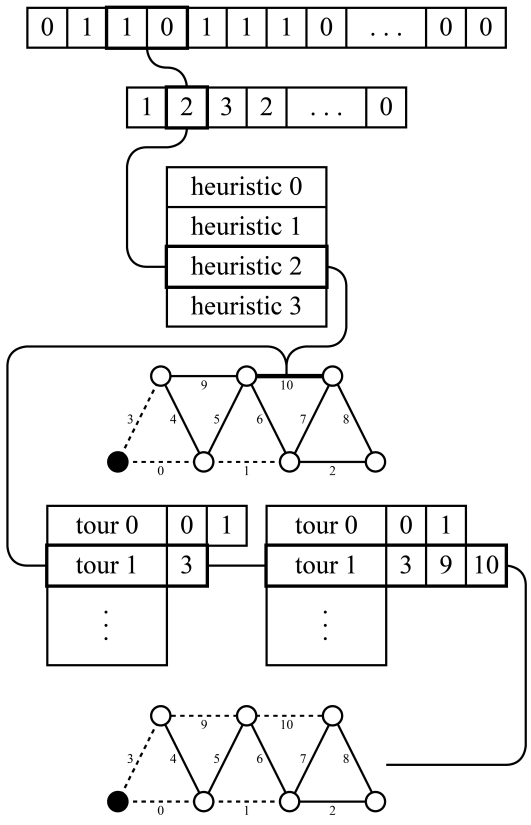


Fig. 3: Decoding a MetaGA chromosome.

Note however that we cannot directly connect edge 3 and 10 since edge 3 and edge 10 do not form a continuous path. To grow tour1 with edge 10 we use Dijkstra's algorithm (cached results) to find a path between the two closest vertices of edges 3 and 10 and connect edge 3 to 10 using this path. In this case, the shortest Dijkstra path is the single edge 9. Tour1 therefore now comprises the edges $3, 9$ and 10 as shown in the figure.

This representation and decoding algorithm has three noteworthy properties. First, it guarantees that every edge will be traversed at least once but does not preclude an edge being used multiple times in one or more tours. Second, the decoding algorithm produces highly greedy tours as it discards chromosome information in two cases; when there is one shortest path to an unvisited edge and when there are no unvisited edges left. And finally, the heuristics are simple to understand and easy to specify.

Although Dijsktra's algorithm can run in $O(E + V \log(V))$ it tends to dominate the time needed to construct and evaluate a tour from a chromosome. This was shown by [16], who used caching to reduce this burden. We thus pre-process the graph data and store the paths between all pairs of vertices in a memory cached table making Dijsktra path finding an $O(1)$ lookup.

With this representation, the objective function is to minimize the length of the longest tour. Mathematically, given an undirected graph $G = \{V, E\}$ where V is the set of vertices and E the set of edges, we want to minimize the length of the longest tour. Denoting the k^{th} robot's (R_k) tour as the sequence of edges $E_k = E_k^1, E_k^2, ...E_k^{k_n}$ where k_n is the number of edges in the tour, we want to minimize the length of the longest tour. Our objective function is therefore

$$Obj = \text{Minimize}\left(Argmax_k\left\{k \mid f(k) = \sum_{i=0}^{k_n} cost(E_k^i)\right\}\right)$$

where $cost(E_k^i)$ is the cost, in this case length, of the i^{th} edge in R_k's tour and $Argmax_k$ returns the maximum cost over the k robots.

We conducted a series of experiments to compare a direct encoded genetic algorithm approach against the MetaGA on the MM k-CPP and the 4DMM k-CPP. The next section provides results.

IV. Results and Discussion

We compared a DEGA with a population size of 1000 run for 1000 generations against a MetaGA with a population of 100 run for 150 generations. This population size and number of generations as well as other GA parameters for the DEGA were set from the DEGA paper [15]. Although the MetaGA uses much smaller population sizes, we get better performance than the DEGA and, of course, the algorithm runs significantly faster although each MetaGA evaluation takes longer. All reported results are from 30 runs of each genetic algorithm over 30 different random seeds. The MetaGA uses linear fitness scaling and both use strong selection pressure where N parents produce N offspring and we choose the best individuals from the combined $2N$ parent offspring pool.

We generated 5 different sizes of trusses of 4 types for a total of 20 problems to serve as our basis for comparison. The first three columns in Table I specify the type of truss and the size of the problem in terms of the number of edges and number of vertices. In addition to comparing against the DEGA, we also compare solution quality against a theoretical lower bound for the MM k-CPP. The theoretical lower bound uses the optimal CPP solution length divided by the number of robots, k to compute this bound. Since the optimal CPP solution can be obtained in polynomial time using the Edmunds algorithm [10], the theoretical lower bound can be easily computed for all our problems. Note however that this lower bound can never be achieved in practice because some edges must be traversed more than once. For example, when r robots start at the same graph node with degree less than r.

Column four in Table I compares MetaGA performance on the 4-depot MM k-CPP against the 1-depot MM k-CPP averaging over $k = 2, 4$ and 8 robots. In these experiments the four depots version of the problem has the four depots located at the four extreme (corners) vertices of the bridge graph, while in the one depot version, the start vertex is the lower left corner vertex.

Columns $4, 5$, and 6 in the table reports performance differences computed as $100 \times (x - \text{4D MetaGA})/\text{4D MetaGA}$,

Instance	$\|E\|$	$\|V\|$	1D Meta vs 4D Meta	4D DEGA vs 4D Meta	CPP/k-LB vs 4D Meta
howe1	37	18	8.84%	19.63%	-12.69%
howe2	63	28	-0.79%	30.09%	-13.22%
howe3	89	38	-1.93%	37.52%	-12.98%
howe4	115	48	-2.17%	41.24%	-12.98%
howe5	141	58	-2.12%	51.63%	-12.65%
ktruss1	45	22	4.92%	25.42%	-13.54%
ktruss2	79	36	-2.14%	36.44%	-13.30%
ktruss3	113	50	-2.82%	43.17%	-11.98%
ktruss4	147	64	-2.27%	56.26%	-11.85%
ktruss5	181	78	-1.84%	73.98%	-11.91%
pratt1	37	18	4.22%	21.13%	-14.05%
pratt2	63	28	-2.44%	30.14%	-12.27%
pratt3	89	38	-2.09%	37.84%	-11.13%
pratt4	115	48	-1.43%	42.37%	-10.88%
pratt5	141	58	-1.28%	50.49%	-11.29%
warren1	59	26	6.08%	31.84%	-8.59%
warren2	85	36	3.66%	39.64%	-8.66%
warren3	111	46	2.19%	45.84%	-9.17%
warren4	137	56	1.65%	54.43%	-9.48%
warren5	163	66	1.23%	65.29%	-9.86%
Averages	**100.5**	**43**	**0.47%**	**41.72%**	**-11.62%**

TABLE I: All truss instance information and MetaGA 4 corner depot percent improvement against MetaGA 1 depot, DEGA 4 corner deployment and CPP/k lower bound.

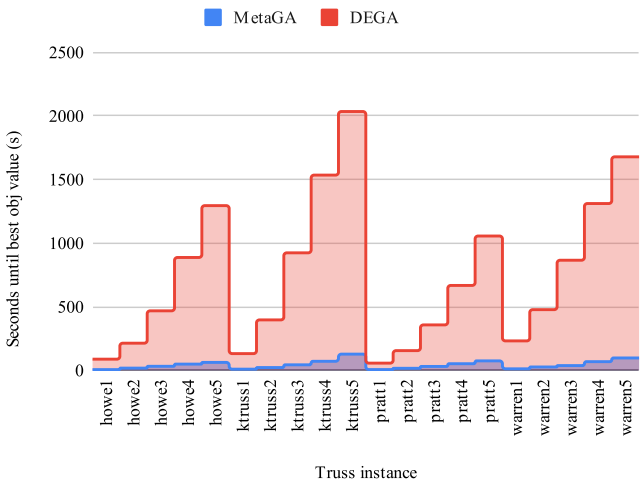


Fig. 4: Average k=2,4,8 average per seed time in seconds to reach best objective

the percentage difference in objective function values, where x is the either the 1 depot MetaGA performance, DEGA performance, or the theoretical lower bound described above. Here the GA performance metric is the average best objective function value over 30 runs. Thus, positive numbers indicate better performance for the 4D MetaGA.

In column four, we use the 1-depot MM k-CPP to compare against in order to better study the gains provided by increasing the number of depots. Does using more depots in our bridge inspection application and therefore having to deploy robots in more locations, lead to gains in inspection time? The fourth column indicates that the total distance traveled is not significantly different and we thus expect to see significantly less (by a factor of k) inspection time as the distance gets distributed across the k robots. The fifth column compares performance against the DEGA again averaging over $2, 4$ and 8 robots. Here we see a significant 42% quality performance improvement in terms of objective function value corresponding to tour length. We break this out for different values of k in subsequent tables. The last column compares MetaGA performance against the lower bound and we see that we are within 12% of the unachievable theoretical lower bound averaged over k.

Figure 4 compares run times for different numbers of robots (different values of k) across the problem instances. The x-axis plots problem instance and y-axis plots runtime in seconds needed to find the best solution. Although runtimes increase as problem size increases for both DEGA and MetaGA, the difference is significant. The figure shows that MetaGA is significantly faster and that the difference in runtime is larger for larger problem sizes.

The growth in runtimes is better depicted in Figure 5. The figure shows how runtimes grow with problems size over our problem set for different values of k. Runtimes increase slowly and linearly for the MetaGA with a significantly smaller slope than for DEGA. Since all three subplots share the same y-axis, time to best solution, the figure indicates that runtimes are dominated by problem size in terms of the number of edges (x-axis). The value of k, the number of robots seems to make little difference. These results provide evidence that the

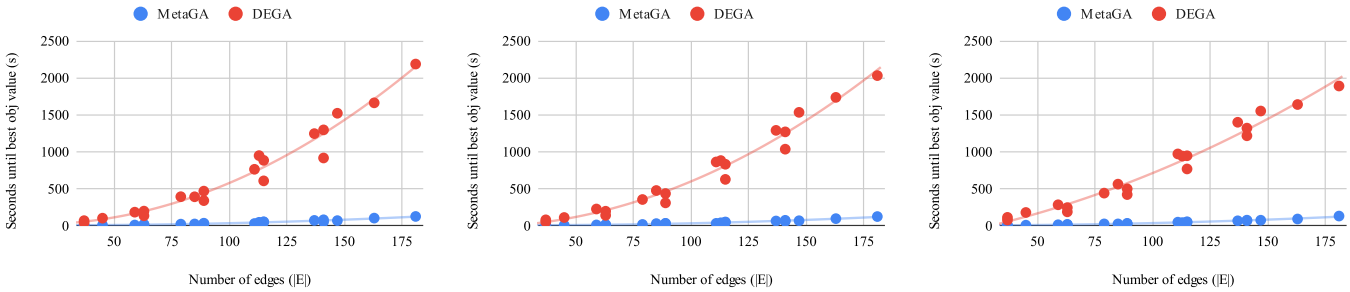
	Ave Var (σ^2)	Ave SD (σ)
MetaGA	3.63	1.73
DEGA	119.28	9.82

TABLE II: Variation in convergence time averaged across k = 2, 4, 8 robtos and averaged over all truss instances.

MetaGA performs significantly better in quality and in speed over the DEGA. Furthermore, when we compute the variance in reliability we see that the MetaGA again does better. Table II shows much lower MetaGA variance and standard deviation for our reliability metric compared to DEGA.

Tables III, IV, and V break out performance differences for different numbers of robots. Table III compares MetaGA performance on the 4-depot MM k-CPP against MetaGA found solutions on the 1-depot MM k-CPP and against DEGA across all 20 problem instances for $k = 2$. As expected, there is no significant difference in tour length as the last row shows that tour-lengths seem to be statistically indistinguishable from those on the 1-depot problem. It is also difficult to see a strong correlation with problem size, However, there is a significant 21.8% improvement over DEGA. Finally, with small k, we do not expect to see a large deviation from the theoretical lower bound and we see this reflected in the insignificant average difference of -1.98% in the last row and column.

Table IV follows the same trend with no significant difference in objective function value between the 1 and 4 depot versions for $k = 4$ robots. MetaGA still performs better



(a) Average 30 seed $k = 2$ results (b) Average 30 seed $k = 4$ results (c) Average 30 seed $k = 8$ results

Fig. 5: $k = 2, 4, 8$ average time to reach best objective value vs number of edges on all truss instances.

Instance	1D MetaGA	4D DEGA	CPP/k-LB
howe1	-0.17%	7.24%	-0.18%
howe2	-0.88%	13.52%	-1.33%
howe3	-0.86%	18.79%	-2.06%
howe4	-1.04%	20.96%	-2.95%
howe5	-0.86%	28.51%	-3.46%
ktruss1	-0.18%	12.28%	-0.20%
ktruss2	-0.72%	17.28%	-2.39%
ktruss3	-0.58%	21.77%	-1.93%
ktruss4	-0.56%	31.04%	-2.94%
ktruss5	-0.53%	48.48%	-3.56%
pratt1	-2.33%	7.25%	-2.35%
pratt2	-0.62%	14.64%	-1.06%
pratt3	-0.05%	17.14%	-1.29%
pratt4	-0.15%	19.51%	-1.99%
pratt5	-0.20%	27.28%	-2.86%
warren1	-1.38%	15.24%	-1.53%
warren2	-0.44%	21.49%	-1.45%
warren3	0.24%	27.24%	-1.57%
warren4	0.19%	30.00%	-2.06%
warren5	0.12%	36.49%	-2.51%
Averages	**-0.55%**	**21.81%**	**-1.98%**

TABLE III: $k = 2$ MetaGA 4 corner depot best result percent improvement against MetaGA 1 depot, DEGA 4 corner deployment and CPP/k lower bound.

Instance	1D MetaGA	4D DEGA	CPP/k-LB
howe1	-1.21%	20.21%	-10.67%
howe2	-3.61%	28.91%	-12.66%
howe3	-3.69%	36.15%	-11.95%
howe4	-4.74%	39.48%	-12.63%
howe5	-3.69%	50.14%	-11.21%
ktruss1	-4.48%	24.19%	-11.47%
ktruss2	-4.70%	37.54%	-11.54%
ktruss3	-3.61%	44.70%	-10.32%
ktruss4	-2.58%	56.67%	-10.05%
ktruss5	-2.26%	72.54%	-10.08%
pratt1	-9.00%	20.14%	-12.63%
pratt2	-4.53%	32.84%	-10.05%
pratt3	-2.75%	39.48%	-8.94%
pratt4	-1.51%	43.06%	-8.71%
pratt5	-1.67%	48.32%	-9.44%
warren1	4.51%	34.04%	-5.47%
warren2	2.90%	39.72%	-5.93%
warren3	1.92%	46.92%	-6.53%
warren4	1.32%	55.53%	-6.79%
warren5	0.87%	67.02%	-7.10%
Averages	**-2.13%**	**41.88%**	**-9.71%**

TABLE IV: $k = 4$ MetaGA 4 corner depot best result percent improvement against MetaGA 1 depot, DEGA 4 corner deployment and CPP/k lower bound.

(41.88% better) than DEGA with four robots but somewhat worse against the theoretical lower bound. Finally, Table V, shows the same trend although the 4D MetaGA seems to do better on Warren trusses compared to the 1D MetaGA. This warrants more investigation.

Figure 6 shows tour differences between the 1-depot and 4-depot tours with $k = 8$ robots on the `howe2` problem instance. Tours start at the lower left for the 1 depot version, while tours start at the four corners on the 4-depot MM k-CPP. The top row shows 4-depot tours generated by the MetaGA with 2 robots starting from each depot. Thicker and redder lines indicate that the member is traversed more times and may appear in more tours than in thinner and blue lines. The tours are different although they share some commonality with edges connected to depots in generally heavier use compared to interior edges. The bottom row of figures shows 1-depot tours for $k = 8$ robots all starting at the bottom left vertex. We can clearly see that the edges connected to this vertex are heavily traversed while edges further away are less heavily used. In these solutions, we note that the sum of the tours travelled by the k robots is approximately 10% shorter for the 4D MM k-CPP compared to the 1D MM k-CPP. This is in contrast to the insignificant difference in objective function value (which is the length of the longest tour) between 1D and 4D tours. We plan to investigate this issue further on larger problems and with larger k and with different sets of heuristics.

V. Conclusion and Future Work

We described a new meta heuristic genetic algorithm approach to the Multi-Depot Min Max k-Chinese Postman and similar arc routing problems. Using four simple heuristics and sorting Djikstra paths to unvisited edges, the MetaGA described in the paper, produces 41.72% better tours $22x$ faster than a directly encoded genetic algorithm approach on 4D MM k-CPPs. Results indicate that with $k = 8$ robots, the MetaGA on 4 depots on a specific type of truss (the Warren trusses) produced tours (routes) that were, on average, better than MetaGA tours with 1 depot. In practical terms, we

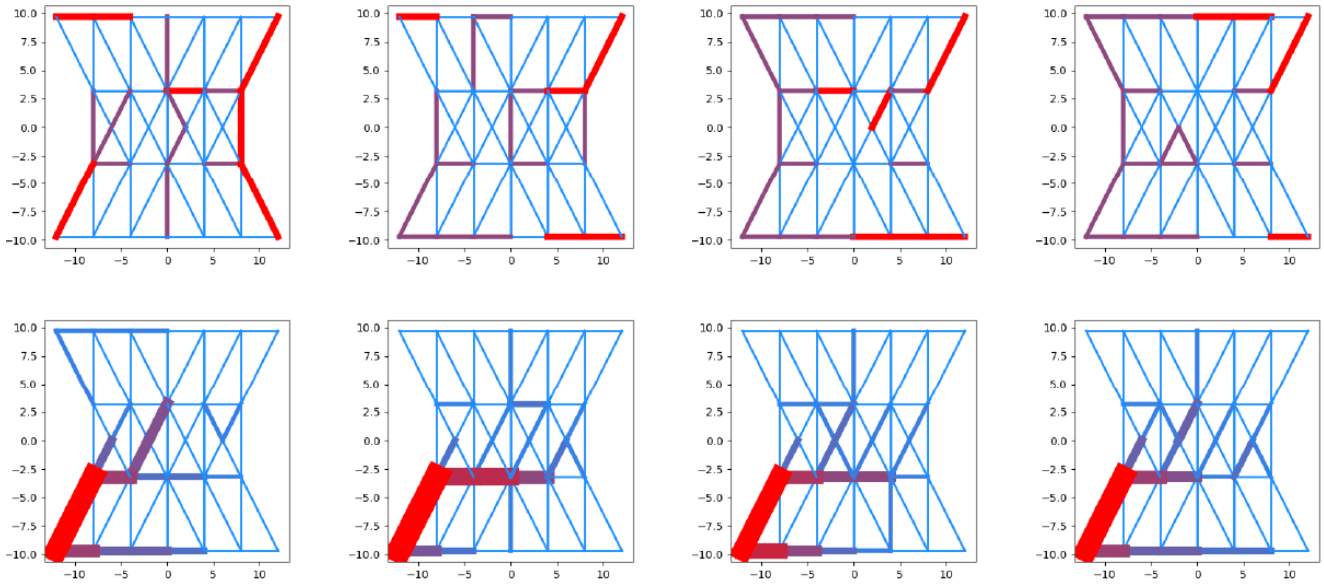


Fig. 6: MetaGA per seed best solution overlap comparison between 1 depot and 4 depots on Howe2 with 8 robots (tours). The top row of images is 4 depot overlap and the bottom row is 1 depot overlap. The overlap is indicated in two ways, line width which represents a edge's sum of visits from all tours and edge color which highlights edges of importance. Color is determined by a gradient blue to red based on the relative amount of visits compared to all other edges visits. Thin blue lines indicate at least one tour includes the edge while thick red lines indicate one or multiple tours visit the edge, one or multiple times.

Instance	1D MetaGA	DEGA	CPP/k-LB
howe1	27.90%	31.43%	-27.21%
howe2	2.12%	47.86%	-25.68%
howe3	-1.25%	57.62%	-24.93%
howe4	-0.74%	63.28%	-23.36%
howe5	-1.80%	76.24%	-23.28%
ktruss1	19.44%	39.81%	-28.95%
ktruss2	-0.99%	54.51%	-25.98%
ktruss3	-4.28%	63.05%	-23.70%
ktruss4	-3.68%	81.08%	-22.56%
ktruss5	-2.73%	100.91%	-22.08%
pratt1	23.98%	36.00%	-27.16%
pratt2	-2.17%	42.96%	-25.71%
pratt3	-3.47%	56.89%	-23.15%
pratt4	-2.64%	64.54%	-21.93%
pratt5	-1.98%	75.86%	-21.58%
warren1	15.10%	46.23%	-18.77%
warren2	8.52%	57.70%	-18.60%
warren3	4.42%	63.35%	-19.41%
warren4	3.44%	77.76%	-19.60%
warren5	2.69%	92.35%	-19.96%
Averages	**4.09%**	**61.47%**	**-23.18%**

TABLE V: $k = 8$ MetaGA 4 corner depot best result percent improvement against MetaGA 1 depot, DEGA 4 corner deployment and CPP/k lower bound.

expect less congestion when starting from multiple depots and thus even a small difference in average performance seems to indicate some utility for multiple robots starting at multiple locations for our bridge inspection problem - at least on some types of trusses. Closer inspection of the results indicate that the largest gains were on smaller problem instances and Warren trusses seem to be the one truss type that exhibited the most consistent performance gains with $k = 8$ robots. We thus plan to investigate different heuristics for different truss types to develop truss specific heuristics that lead to consistent performance gains.

Since bridge trusses often have highly repeatable structures, in future work, we also plan to explore scaling up to large bridge instances using this modularity. That is, solving a smaller instance of a repeated (modular) truss structure may indicate the tours for a much larger problem based on tours produced for a smaller problem. Finally, we plan to investigate whether these simple heuristics generalize to other types of less structured graphs and to investigate new heuristics.

REFERENCES

[1] Dino Ahr and Gerhard Reinelt. New heuristics and lower bounds for the min-max k-chinese postman problem. In Rolf Möhring and Rajeev Raman, editors, *Algorithms — ESA 2002*, pages 64–74, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[2] Dino Ahr and Gerhard Reinelt. A tabu search algorithm for the min–max k-chinese postman problem. *Computers & Operations Research*, 33(12):3403 – 3422, 2006. Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems.

[3] S.K. Amponsah and S. Salhi. The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries. *Waste Management*, 24(7):711–721, 2004.

[4] Sushil J Louis Bryan Dedeurwaerder. A meta heuristic genetic algorithm for multi-depot routing code, 2021. http://github.com/sushillouis/Stacs.

[5] Corberán and Prins. Recent results on arc routing problems: an annotated bibliography. *Networks*, 56(1):50–69, 2010.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[7] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858, 2000.

[8] Bryan Dedeurwaerder. metaga-for-mmmdcpp, January 2022. https://github.com/BryanDedeur/metaga-for-mmmdcpp.

[9] Aditya Dixit, Apoorva Mishra, and Anupam Shukla. Vehicle routing problem with time windows using meta-heuristic algorithms: a survey. In *Harmony search and nature inspired optimization algorithms*, pages 539–546. Springer, 2019.

[10] Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, Dec 1973.

[11] Liang Feng, Yew-Soon Ong, Meng-Hiot Lim, and Ivor W Tsang. Memetic search with interdomain learning: A realization between cvrp and carp. *IEEE Transactions on Evolutionary Computation*, 19(5):644–658, 2014.

[12] G. Fleury, P. Lacomme, and C. Prins. Evolutionary algorithms for stochastic arc routing problems. *Lecture Notes in Computer Science*, 3005:501–512, 2002.

[13] Gérard Fleury, Philippe Lacomme, Christian Prins, and Marc Sevaux. A memetic algorithm for a bi-objective and stochastic carp. *Proceeding of the MIC*, pages 22–26, 2005.

[14] Gianpaolo Ghiani, Francesca Guerriero, and Gennaro Improta. Waste collection in southern italy: Solution of a real-life arc routing problem. *Int Trans Oper Res*, 12(2):135–144, 2005.

[15] Nicholas Harris. Direct Encoded Genetic Algorithm, using CUDA and caching improvements, May 2020. https://github.com/nicholasharris/MMkCPP-GA-research.

[16] Nicholas Harris, Siming Liu, Sushil J. Louis, and Jim Hung La. A genetic algorithm for multi-robot routing in automated bridge inspection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 369–370, Prague Czech Republic, July 2019. ACM.

[17] Geir Hasle. *Chapter 16: Arc Routing Applications in Newspaper Delivery*, pages 371–395. Siam, 10 2013.

[18] Yantao Huang and Kara M Kockelman. Electric vehicle charging station locations: Elastic demand, station congestion, and network equilibrium. *Transportation Research Part D: Transport and Environment*, 78:102179, 2020.

[19] Sašo Karakatič. Optimizing nonlinear charging times of electric vehicle routing with genetic algorithm. *Expert Systems with Applications*, 164:114039, 2021.

[20] Hung Manh La, Tran Hiep Dinh, Nhan Huu Pham, Quang Phuc Ha, and Anh Quyen Pham. Automated robotic monitoring and inspection of steel structures and bridges. *Robotica*, 37(5):947–967, 2019.

[21] Lacomme, Prins, and Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers & Operations Research*, 33(12):3473–3493, 2006.

[22] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1-4):159–185, 2004.

[23] David Lattanzi and Gregory Miller. Review of robotic infrastructure inspection systems. *Journal of Infrastructure Systems*, 23(3):04017004, 2017.

[24] L. Muyldermans, D. Cattrysse, D. Van Oudheusen, and Lotan T. Districting for salt spreading operations. *European Journal of Operational Research*, 139(3):521–532, 2002.

[25] United States Department of Transportation. National bridge inventory - bridge inspection - safety - bridges & structrues, 2019. http://www.fhwa.dot.gov/bridge/nbi.cfm Accessed: 12/19/16.