2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Multi-Goal Path Planning based on the Generalized Traveling Salesman Problem with Neighborhoods

Kevin Vicencio, Brian Davis, and Iacopo Gentilini

***Abstract*— Often times in mobile robotics, optimizing a sequence of tasks and the paths between those destinations is an essential factor. In simple cases, this problem can be modeled by the well-researched Traveling Salesman Problem (TSP). In more complex situations however, the TSP is not a suitable model. In redundant robotic systems, a robot can assume infinitely many configurations while performing each given task. In these cases, not only is it necessary to optimize the sequence of tasks, but an optimal configuration must be defined for each task as well. This optimization problem can be better modeled by the Traveling Salesman Problem with Neighborhoods (TSPN) in which nodes can move in given domains called neighborhoods. However, one of the limiting factors to the TSPN is that it cannot efficiently solve realistic instances with non-connected neighborhoods. This research proposes an approach to this problem in which the TSPN is extended into a Generalized Traveling Salesman Problem with Neighborhoods (GTSPN) where each node can be located in multiple regions, called neighborhoodsets. An heuristic procedure is proposed to find a near-optimal tour for GTSPN instances using a genetic algorithm approach. Numerical simulations performed on randomly generated instances with up to 300 neighborhoods show that the proposed procedure determines tours for a given instance within a 1% standard deviation error from the best-known tour.**

I. Introduction

The Traveling Salesman Problem (TSP) entails minimizing a path, or *tour*, given a set of nodes, that visits each node once. It is a popular problem in the optimization community and widely researched [1]. Although applicable to many different fields, the model is limited by the fact that each node is a fixed point.

The Generalized Traveling Salesman Problem (GTSP) is an extension of the TSP and is useful for solving problems involving decisions of selection and sequence. The problem entails minimizing a tour, given a set of *nodesets*, that visits each nodeset once, in which only one node from the respective nodeset is visited [2].

The Traveling Salesman Problem with Neighborhoods (TSPN) is an extension of the TSP. Given a set of regions or neighborhoods, the problem seeks to minimize a tour that visits each neighborhood once [3]. This problem involves not only finding an optimal sequence, but also an optimal node within the neighborhood. To emphasize this additional degree of freedom not modeled in the standard TSP, nodes will be referred to as *configurations* hereafter. However, one of the limiting factors of TSPN is that it cannot account for non-connected or clustered neighborhoods, i.e., neighborhoods that are defined as sets of neighborhoods as illustrated in Figure 1.

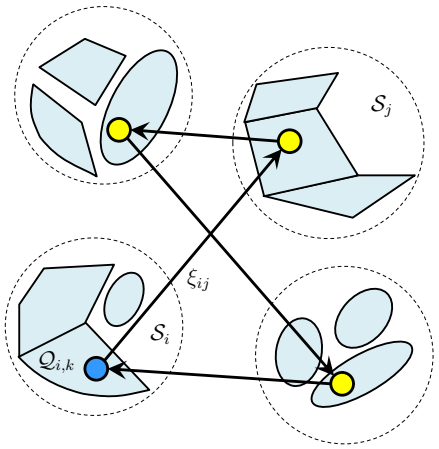

Fig. 1. A GTSPN instance with four neighborhoodsets defined by non-connected or non-convex domains.

The authors propose a novel formulation, the Generalized Traveling Salesman Problem with Neighborhoods (GTSPN), a derivative of the TSPN and the GTSP, to account for the limitations in the TSPN. Given a set of *neighborhoodsets*, each set with a fixed number of neighborhoods, the GTSPN model seeks to optimize a tour that visits each neighborhoodset only once, i.e., in which only one neighborhood from the respective neighborhoodset is visited. Using these parameters, the GTSPN can account for non-connected or clustered neighborhoods. Moreover, the GTSPN can model non-convex neighborhoods by splitting non-convex neighborhoods into separate convex regions. These separated convex neighborhoods can be treated as neighborhoodsets and utilized in the GTSPN.

In robotics, clustered or non-convex neighborhoods can be used to model a variety of multi-goal path planning problems. For redundant robotic manipulators, a TSP formulation can provide an initial approximation [4] or discrete samples for each neighborhood can be extracted and the resulting GTSP can be solved [5]. A TSPN formulation can be also used [6] but with the limitation of considering only one among multiple figures of the manipulator inverse kinematics [7]. For Unmanned Aerial Vehicles (UAV) a flight path can be calculated using a TSPN formulation such that the aircraft

K. Vicencio, and I. Gentilini are with the Department of Aerospace and Mechanical Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301 vicencik@my.erau.edu, gentilii@erau.edu.

B. Davis is with the Department of Electrical Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301 davisb22@erau.edu.

flies within a certain distance from the center of several inspection sites, while minimizing the flying time or energy consumption [8], [9]. However, more realistic scenarios such as obstacles that hinder the field of view of the cameras require a more flexible definition of the target regions. The GTSPN represents a viable approach to overcome the limitations of the previous formulations.

In order to solve GTSPN instances, the Hybrid Random-Key Genetic Algorithm (HRKGA) proposed in [8] to solve TSPN instances is adapted using an ad-hoc *chromosome* coding procedure and *crossover* operators. Numerical simulation performed on randomly generated instances show that the adapted HRKGA retrieve tours that on average are within a 1% standard deviation error from the best-known tour.

The rest of the paper is organized as follows. The proposed GTSPN formulation and the GA are presented in Sections II and III. The numerical results are illustrated in Section IV. Finally, Section V contains conclusions and discusses potential future work.

II. GTSPN Formulation

A feasible tour for a GTSPN instance is defined when the tour visits one configuration in one neighborhood within each neighborhoodset once.

Input: a set $\mathrm{V} = \{1, \ldots, n\}$ of the indices of the neighborhoodsets $\mathcal{S}_i$, and n sets $\mathrm{W}_i = \{1, \ldots, m_i\}$ of the indices of the neighborhoods $\mathcal{Q}_{i,k}$, i.e., $\mathcal{S}_i = \mathcal{Q}_{i,1} \cup \ldots \cup \mathcal{Q}_{i,m_i}$.

Goal: find an optimal sequence of configurations $\boldsymbol{q}_i \in \mathbb{R}^l$ each one within neighborhood $\mathcal{Q}_{i,k} \subseteq \mathcal{S}_i$ with $i = 1, \ldots, n$ and $k = 1, \ldots, m_i$.

Output: a minimal-cost cyclic tour $T = \{\boldsymbol{q}_{\pi(1)}, \ldots, \boldsymbol{q}_{\pi(n)}\}$.

The following formulation is an extension of the Mixed Integer Non-Linear Programming (MINLP) formulation of the symmetric TSPN found in [8]. The MINLP formulation was adapted to account for the new concept of neighborhood sets in the GTSPN.

If $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j)$ represents a symmetric cost function for the robot to move from a configuration $\boldsymbol{q}_i \in \mathbb{R}^l$ to a configuration $\boldsymbol{q}_j \in \mathbb{R}^l$, i.e., $\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) = \mathrm{d}(\boldsymbol{q}_j, \boldsymbol{q}_i)$, the following Mixed Integer Disjunctive Programming formulation of the symmetric GTSPN can be defined:

$$\text{minimize}: \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \xi_{ij}\, \mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_j) \ , \tag{1}$$

$$\text{subject to}: \quad \sum_{j=1}^{i-1} \xi_{ji} + \sum_{j=i+1}^{n} \xi_{ij} = 2 \qquad \forall\, i \in \mathrm{V} \ , \tag{2}$$

$$\sum_{i \in \mathrm{S}} \left(\sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j<i}} \xi_{ji} + \sum_{\substack{j \in \mathrm{V} \setminus \mathrm{S} \\ j>i}} \xi_{ij} \right) \geq 2 \qquad \forall\, \mathrm{S} \subset \mathrm{V} \setminus \{1\},\ |\mathrm{S}| \geq 3 \ , \tag{3}$$

$$\bigvee_{k \in \mathrm{W}_i} \left[\boldsymbol{q}_i \in \mathcal{Q}_{i,k} \subseteq \mathbb{R}^l \right] \qquad \forall\, i \in \mathrm{V} \ , \tag{4}$$

$$\xi_{ij} \in \{0, 1\} \qquad \forall\, i, j \in \mathrm{V},\ j > i \ , \tag{5}$$

$$\boldsymbol{q}_i \in \mathbb{R}^l \qquad \forall\, i \in \mathrm{V} \ . \tag{6}$$

The $n(n-1)/2$ binary variables ξ_{ij} for all $i, j \in V$ with $j > i$ are defined such that $\xi_{ij} = 1$ only if neighborhoodset j is visited just after the neighborhoodset i or viceversa in the tour, otherwise it is zero. The assignment problem constraints (2) ensure that each neighborhood set is visited exactly once. The subtour elimination constraints (3) ensure that no subtour is present in a solution. The new constraints (4) force each configuration to be within only one of the neighborhoods in each neighborhoodset. Finally the constraints (5) and (6) define the domain of the instance.

III. GTSPN Solution Procedure

In [10] an optimal solution to the TSPN was developed using Mixed Integer Non-Linear Programming (MINLP). It was shown that this method is too computationally expensive to be utilized for instances with more than 20 neighborhoods. As the GTSPN is an extension of the TSPN, the necessary computation to find an optimal solution would increase drastically making it implausible. Therefore we propose to use a genetic algorithm to obtain near-optimal results for real-world applications.

Genetic algorithms mimic natural selection by randomly generating a *population* of feasible solutions, called *chromosomes*. Chromosomes undergo selection in which they are evaluated against a given cost function. A GA then utilizes natural selection techniques to evolve the population and create better chromosomes. The genetic algorithm used in this work is an extension of the Hybrid Random-Key Genetic Algorithm (HRKGA) developed for the TSPN in [8].

A. Chromosome Coding

Each chromosome contains n *genes*. In the original HRKGA chromosomes, each gene was comprised of a vector part and a uniformly distributed random number in the interval [0,1]. The vector part corresponds to the configuration of the node within the neighborhood $\boldsymbol{q}_i$, while the fractional part represents the configuration position in the sequence.

Since the GA for the GTSPN needs to determine an optimal neighborhood within a neighborhoodset, each gene will also need to contain an index for the neighborhood within the neighborhood set. Using this method, each gene will contain an index k, a vector part $\boldsymbol{q}_i$, and a uniformly, distributed random number in the interval [0,1]. k is an index corresponding to the neighborhood of the respective neighborhhodset. $\boldsymbol{q}_i$ corresponds to the configuration of the node within the respective neighborhood. The fractional number corresponds to the which element in the sequence the node is. For example, in the case of $\mathrm{V} = \{1, \ldots, 4\}$ and $\mathrm{W}_i = \{1, \ldots, 5\}$ the following chromosome:

$$1\boldsymbol{q}_1.22 \quad 4\boldsymbol{q}_2.72 \quad 2\boldsymbol{q}_3.45 \quad 3\boldsymbol{q}_4.02$$

is decoded into the following tour:

$$\boldsymbol{q}_4 \in \mathcal{Q}_{4,3} \rightarrow \boldsymbol{q}_1 \in \mathcal{Q}_{1,1} \rightarrow \boldsymbol{q}_3 \in \mathcal{Q}_{3,2} \rightarrow \boldsymbol{q}_2 \in \mathcal{Q}_{2,4}$$

When a new chromosome is being generated, the index is determined by randomly choosing among the available neighborhoods in the respective neighborhoodset. The configuration within the respective neighborhood is also randomly generated.

B. Crossover Operations

Genetic Algorithms make use of crossover operators. Given a pool of data points, or chromosomes, for a given cost function, crossover operators mimic natural selection by selecting two chromosomes as *parents* and producing *offspring* chromosomes in the mindset that the offspring will be more cost effective. Many different types of crossover operators have been researched and analyzed. It is found in [11], that the uniform crossover operator appears to be one of the more capable operators.

1) Uniform Crossover Operator: The formal development of the Uniform Crossover Operator is available in [12]. A uniformly distributed set of n random numbers in the interval [0,1] is generated. If the i-th element of the sequence is less than a given threshold, then the offspring will inherit the i-th gene of the first parent, otherwise it will inherit the respective gene of the second.

Given: 2 chromosome parents determined through a *tournament* style selection in which the most cost effective chromosomes are chosen [13].

Operator: selected genes from each parent are uniformly distributed to the offspring.

In this operator, neither the indices, random-key, nor the configuration of the node are manipulated. In turn, this operator will always generate feasible chromosome offspring.

2) Arithmetic Average Crossover Operator: The following crossover proposed is developed by determining the average of two parent chromosomes. This operator attempts to take the benefits of both parents and pass them on to the offspring. This method modifies the indices, sequence, and configurations of the genes rather than simply inheriting individual genes from the parents.

Given: 2 chromosome parents determined through a *tournament* style selection in which the most cost effective chromosomes are chosen.

Operator: the offspring chromosome is equivalent to the sum of the parents divided by two. The indices of the chromosome are rounded to the nearest integer.

Since genes contain the index to a neighborhood within a respective neighborhood set, and the point within the respective neighborhood, when the same genes of the two parent chromosomes are averaged, it is possible that each gene may have a different index. In this situation, a point may be generated outside either neighborhoods. This infeasible situation is addressed by the improvement heuristics. Since the gene retains the index of the neighborhood in which the point should belong to, the heuristics will return the point to the respective neighborhood.

Algorithm 1 A Hybrid Random-Key Genetic Algorithm for solving GTSPN Instances.

Input: a set $\mathrm{V} = \{1, \ldots, n\}$ of the indices of the neighborhoodsets $\mathcal{S}_i$, the neighborhoodsets $\mathcal{S}_i$, n sets $\mathrm{W}_i = \{1, \ldots, m_i\}$ of the indices of the neighborhoods $\mathcal{Q}_{i,k}$, and the neighborhoods $\mathcal{Q}_{i,k}$
Output: a minimal-cost cyclic tour $T = \{\boldsymbol{q}_{\pi(1)}, \ldots, \boldsymbol{q}_{\pi(n)}\}$ with objective value L

```
Generate an initial population of chromosomes
Let T be the best chromosome in the initial population
Let L be the objective value of T
g_C ← 1, g_I ← 1
repeat
    Selection and Crossover of chromosomes
    Mutation of index with randomly generated point
    Immigration
    Improvement Heuristics
    g_I ← g_I + 1
    Let T_g be the best chromosome in the generation
    Let L_g be the objective value of T_g
    if L_g < L then
        T ← T_g
        L ← L_g
    else
        g_C ← g_C + 1
until g_I > g_MAX or g_C > g_CMAX
return T and L
```

This operator is applicable to the original HRKGA as well, and since individual genes in the original HRKGA contain only one neighborhood, the crossover will not produce infeasible offspring. However, let it be noted that this operator may not work on all genetic algorithms and is intended mainly for TSPN or GTSPN instances.

C. Mutation

Genetic Algorithms imitate natural selection by *mutating* selected chromosomes in order to speed up the evolution rate of the population.

During each generation of the algorithm, a random population of chromosomes are chosen to have their neighborhood set indexes mutated. In this case, a random neighborhood set within the chromosome is selected. The index is then erased, and randomly chosen from the available neighborhoods of the respective neighborhood set. Since the index has changed, it is then necessary to initialize the configuration for the respective neighborhoodset. This is done by randomly generating a configuration within the respective neighborhood.

D. Algorithm

Let it be noted that a Lin-Keringhan heuristic and Touring heuristic as described in [8] are used in order enhance the performance of the GA by determining a near-optimal sequence of fixed configurations and an optimal set of configurations for a fixed sequence of convex neighborhoods. In addition, an immigration technique is implemented to maintain the variety of the population. The implemented algorithm is illustrated in Algorithm 1.

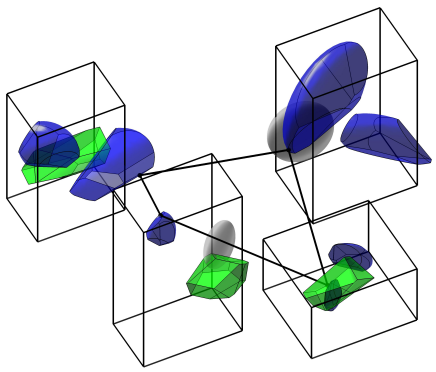

Fig. 2. A GTSPN instance ($l = 3, n = 4, m_i = 3$) with polyhedra (green), ellipsoids (gray), and hybrid neighborhoods (blue). The near optimal tour calculated using Euclidean distance is represented by the black line. Bounds of neighborhoodset are depicted with black solid lines.

The termination criteria are as follows: maximum number of generations, g_{MAX}, and maximum number of generations without improving the best-known chromosome, $g_{C\text{MAX}}$.

IV. Simulation Results

Since to the best knowledge of the authors no benchmark GTSPN instances are available in the literature, the proposed approach is tested using randomly generated instances with varying $n \in \{30, 35, 40, 45, 50\}$, constant $m_i = 6$, and $l = 3$ or $l = 7$ to simulate actual UAV or redundant manipulation applications [8]. All test instances are available from [14], and an example in $\mathbb{R}^3$ is shown in Figure 2.

A. Generation of the Instances

The following explains the procedure of generating the instances for testing.

An *effective radius* r is first defined. The effective radius defines the radius of a volume in which each neighborhoodset can be generated. n uniformly distributed random vectors $\boldsymbol{q}_{r,i} \in [0, r]^l$ are then generated where $i \in \text{V}$. These vectors define configurations within the overall domain of the GTSPN instance.

An *upper bound* $\boldsymbol{ub}_i$ and a *lower bound* $\boldsymbol{lb}_i$ are then assigned to each configuration. The upper and lower bounds define the region for a neighborhoodset in which neighborhoods are to be generated. The upper and lower bounds are functions of the average distance between the configurations d_{AVG} and a user defined parameter h. h is a number on the interval $[0, 1]$ that defines the percentage of the average distance between all configurations to be used in calculating the upper and lower bounds of the neighborhoodset. The upper bound is then calculated by summing $\boldsymbol{q}_{r,i}$ and an uniformly distributed random vector in $[0, h \cdot d_{\text{AVG}}]^l$. The lower bound is calculated by summing $\boldsymbol{q}_{r,i}$ and a uniformly distributed random vector in $[-h \cdot d_{\text{AVG}}, 0]^l$. The center of each neighborhoodset is then determined by taking the average of the upper bound and lower bound vectors.

The next step is to generate neighborhoods for the neighborhoodset. For this purpose, an effective radius $\boldsymbol{r}_i$ must be determined where $i \in \text{V}$. In this situation, $\boldsymbol{r}_i$ defines the volume in which each neighborhood pertaining to a given neighborhoodset can be generated. $\boldsymbol{r}_i$ is defined as the amplitude of the respective neighborhoodset or $(\boldsymbol{ub}_i - \boldsymbol{lb}_i)/2$. m_i uniformly distributed random vectors $\boldsymbol{q}_{r,ik} \in [-\boldsymbol{r}_i, \boldsymbol{r}_i]^l$ are then generated for each neighborhoodset where $i \in \text{V}$ and $k \in \text{W}_i$. The vectors define configurations within the respective neighborhoodset.

An *upper bound* $\boldsymbol{ub}_{ik}$ and a *lower bound* $\boldsymbol{lb}_{ik}$ are then assigned to each configuration. If h_i is a number on the interval [0,1], the upper bound is calculated by summing $\boldsymbol{q}_{r,ik}$ and a uniformly distributed random vector in $[0, h_i \cdot d_{\text{AVG},i}]^l$. The lower bound is calculated by summing $\boldsymbol{q}_{r,ki}$ and a uniformly distributed random vector in $[-h_i \cdot d_{\text{AVG},i}, 0]^l$. The center of each neighborhood is then determined by taking the average of the upper bound and the lower bound.

In real-world robotics applications neighborhoods can have complex shapes in the configuration space. However, an approximation can be achieved using linear constraints, i.e., polyhedra, quadratic constraints, i.e., ellipsoids, or a combination of the the two, i.e., *hybrid* neighborhoods. Polyhedra and ellipsoids are defined by:

$$\mathbf{A}_{ik}\, \boldsymbol{q}_i + \mathbf{b}_{ik} \leq 0 \qquad \forall\, k \in \text{W}_i\ \forall\, i \in \text{V}\ , \quad (7)$$

$$(\boldsymbol{q}_i - \mathbf{c}_{ik})^T\ \mathbf{P}_{ik}^{-1}\ (\boldsymbol{q}_i - \mathbf{c}_{ik}) \leq 1 \quad \forall\, k \in \text{W}_i\ \forall\, i \in \text{V}\ , \quad (8)$$

where $\mathbf{A}_{ik}$ is $(p_{ik} \times l)$ matrix, $\mathbf{b}_{ik}$ is a $(p_{ik} \times 1)$ vector with p_{ik} the number of halfspaces of the ik-th polyhedron, $\mathbf{P}_{ik}$ is a $(l \times l)$ symmetric positive definite matrix, and $\mathbf{c}_{ik}$ is a $(l \times 1)$ vector, center of the ik-th ellipsoid.

For the proposed GTSPN instances, the geometry of each neighborhood is determined randomly. First an ellipsoid is generated by aligning its major axes with the coordinate frame. The lengths of the major axes are defined by the bounding box $[\boldsymbol{lb}_{ik}, \boldsymbol{ub}_{ik}]$. Then a random rotation is applied. If a polyhedron has to be generated, then the $2l$ rotated facets of the bounding box and 6 additional facets tangent to the ellipsoid at randomly selected locations are used. Finally, if a hybrid neighborhood has to be generated, some facets are shifted in the interior of the rotated ellipsoid.

B. Testing Procedure

A first comparison is performed between the Arithmetic Average Crossover Operator and the Uniform Crossover Operator. 10 instances are generated, and for each instance the HRKGA is executed 60 times using the Arithmetic Average Crossover Operator and 60 times using the Uniform Crossover Operator. Then, the overall testing of the GTSPN is performed using the Uniform Crossover Operator. For $l = 3$, 35 instances are generated, and for each instance the HRKGA is executed 30 times. For $l = 7$, five instances are generated, and for each instance the HRKGA is executed 30 times. For testing purposes, $r = 500$, $h = .1$, $h_i = .5$, $g_{\text{MAX}} = 60$, $g_{C\text{MAX}} = 20$, and the used cost function $\mathrm{d}\,(\boldsymbol{q}_i, \boldsymbol{q}_j)$

TABLE I

COMPARISON BETWEEN ARITHMETIC CROSSOVER AND UNIFORM CROSSOVER

Instance	n	m_i	n_p	n_e	n_h	Arithmetic Crossover				Uniform Crossover			
						Avg CPU [s]	Best L	Avg L	STD L	Avg CPU [s]	Best L	Avg L	STD L
c30_6_a	30	6	59	62	59	32.09	3879.83	3879.83	0.00	46.11	3724.66	3751.41	16.10
c35_6_a	35	6	71	68	71	80.49	4184.44	4194.17	8.59	69.07	4033.67	4075.77	25.34
c35_6_b	35	6	74	64	72	73.00	3822.36	3822.37	0.01	62.48	3702.89	3728.07	12.95
c40_6_a	40	6	80	76	84	93.44	4530.00	4579.23	14.41	74.91	4388.65	4426.75	20.04
c40_6_b	40	6	76	69	95	57.06	4525.25	4550.29	5.79	71.52	4363.82	4407.54	26.76
c45_6_a	45	6	92	75	103	95.02	4759.88	4810.92	18.56	79.61	4599.45	4655.16	25.00
c45_6_b	45	6	79	99	92	86.02	4602.78	4613.00	14.58	82.51	4459.35	4490.39	21.63
c50_6_a	50	6	95	110	95	106.69	4989.22	4989.22	0.00	71.30	4756.08	4797.70	25.00
c50_6_b	50	6	97	99	104	101.02	5271.62	5299.56	11.20	79.15	5113.75	5161.59	20.02
c50_6_c	50	6	96	97	107	105.41	4745.64	4752.70	6.50	78.05	4539.94	4585.61	32.16

is the Euclidean distance. The simulations are performed on an `Intel Xeon CPU @3.20 GHz` processor with 12GB of RAM running `Windows 7 64bit`.

C. Evaluation of the Crossover Operators

In this section the Arithmetic Average Crossover is compared to the Uniform Crossover. Table I displays the results of the comparison. Columns labeled n_p, n_e, and n_h report the total number of polyhedra, ellipsoids, and hybrid neighborhood for each instance. The column labeled "Avg CPU" reports the average CPU time in seconds. Finally, columns labeled "Best L", "Avg L", and "STD L" report the best-known tour length, the average tour length, and the tour length standard deviation.

The ratio between the standard deviation of the tour distances and the most cost effective tour developed by the HRKGA with the Arithmetic Average Crossover Operator is on average 0.001708. Therefore the HRKGA with the Arithmetic Average Crossover Operator is consistent within $\pm 0.17\%$ when determining a tour. The ratio between the standard deviation of the tour distances and the most cost effective tour developed by the HRKGA with the Uniform Crossover Operator is on average 0.005134. Therefore the HRKGA with the Arithmetic Average Crossover Operator is consistent within $\pm 0.51\%$ when determining a tour. It can be determined through this comparison that the Arithmetic Average Crossover Operator provides more consistency than the Uniform Crossover Operator.

On average, the HRKGA using the Uniform Crossover Operator is able to determine a tour that is 3.193% more cost effective than the tour developed by the HRKGA using the Arithmetic Average Crossover Operator. In addition, on average, the HRKGA using the Uniform Crossover Operator is able to determine a tour in 14.34% less CPU time than the HRKGA using the Arithmetic Average Crossover Operator. On average, the Uniform Crossover Operator appears to perform slightly better than the Arithmetic Average Crossover Operator.

The HRKGA using the Uniform Crossover Operator is able to determine a most cost-effective overall tour that is on average 3.73% better than the most cost-effective overall tour produced by the HRGKA with the Arithmetic Average Crossover Operator.

From these observations, it has been concluded that the Uniform Crossover Operator is slightly more cost-efficient and computation-efficient. Therefore, the Uniform Crossover Operator is utilized in the main GTSPN evaluation.

D. Evaluation of the HRGKA

In this section an evalutaion of the HRGKA's ability to solve for a near-optimal tour is presented by analyzing the results of the simulations. Table II displays the results of the HRGKA simulations. Column labels are the same as in Table I. An additional column labelled "70% Impr. [s]" is included. This column reports the average CPU time in seconds for the HRKGA to reach at least 70% of the total overall tour improvement.

The ratio between the standard deviation of the tour distances and the most cost effective tour developed by the HRKGA is on average $.00445$ for instances in $\mathbb{R}^3$ and $.00340$ for instances in $\mathbb{R}^7$. Therefore the HRKGA is consistent within $\pm 0.45\%$ and $\pm 0.34\%$ when determining a tour for the GTSPN in $\mathbb{R}^3$ and $\mathbb{R}^7$, respectively. It can be also observed that the difference between the average and the best-know tour length is on average 1.87 times the standard deviation in $\mathbb{R}^3$ and 1.70 times the standard deviation in $\mathbb{R}^7$. This property, investigated for TSP distributions in [15], can provide valuable information about a statistical lower bound for GSTPN instances.

The ratio between the CPU time to reach 70% tour improvement and the total CPU time is on average $.246$ for instances in $\mathbb{R}^3$ and $.373$ for instances in $\mathbb{R}^7$. Therefore the HRKGA improves a tour by at least 70% in 24.6% and 37.3% of the total CPU time when determining a tour for the GTSPN in $\mathbb{R}^3$ and $\mathbb{R}^7$, respectively. This property can provide valuable information for realistic scenarios where CPU time is emphasized over tour length.

V. CONCLUSION AND FUTURE WORK

In redundant, robotic systems, it is often necessary to optimize a sequence of tasks. In the past, these applications often rely on the TSP and the GTSP to model the problem. Recently, approaches have been proposed to utilize the TSPN to create better models of this complex optimization problem. Unfortunately, due to the TSPN's inability to account for non-connected neighborhoods, its applicability to real-world scenarios is diminished.

In this research, a Generalized Traveling Salesman Problem with Neighborhoods is formulated to account for non-connected neighborhoods. The problem entails defining mul-

TABLE II

HRGKA PERFORMANCE EVALUATION ON RANDOMLY GENERATED GTSPN INSTANCES IN $\mathbb{R}^3$ AND $\mathbb{R}^7$.

Instance	l	n	m_i	n_p	n_e	n_h	Avg CPU [s]	70% Impr. [s]	Best L	Avg L	STD L
3D_30_6_a	3	30	6	55	66	59	52.71	10.37	3560.90	3578.10	9.96
3D_30_6_b	3	30	6	51	49	80	51.33	9.29	3879.62	3900.15	10.26
3D_30_6_c	3	30	6	66	52	62	55.70	20.62	3714.36	3740.86	13.68
3D_30_6_d	3	30	6	63	62	55	40.64	4.84	3607.69	3625.71	10.14
3D_30_6_e	3	30	6	62	57	61	52.22	10.12	3593.31	3612.66	11.68
3D_30_6_f	3	30	6	67	65	48	53.26	9.76	3625.18	3650.82	16.75
3D_35_6_a	3	35	6	79	69	62	65.55	16.65	4318.65	4336.62	9.84
3D_35_6_b	3	35	6	60	84	66	52.96	12.38	4214.62	4257.93	16.82
3D_35_6_c	3	35	6	74	64	72	61.03	16.57	3700.23	3728.00	16.75
3D_35_6_d	3	35	6	67	72	71	61.72	24.18	4303.18	4343.53	34.04
3D_35_6_e	3	35	6	67	61	82	60.65	12.81	3899.11	3940.44	19.03
3D_35_6_f	3	35	6	75	73	62	59.74	15.07	4057.73	4117.04	22.04
3D_40_6_a	3	40	6	76	86	78	56.85	14.20	4489.47	4531.18	20.68
3D_40_6_b	3	40	6	86	85	69	80.19	15.87	4176.58	4207.48	11.60
3D_40_6_c	3	40	6	94	65	81	65.71	27.85	4379.74	4413.40	21.83
3D_40_6_d	3	40	6	85	80	75	71.40	13.14	4550.32	4576.56	15.41
3D_40_6_e	3	40	6	82	87	71	68.66	11.43	4118.23	4141.17	11.46
3D_40_6_f	3	40	6	83	85	72	67.13	17.01	4036.65	4064.10	17.38
3D_45_6_a	3	45	6	82	98	90	69.13	12.90	4501.49	4549.29	23.57
3D_45_6_b	3	45	6	100	88	82	90.52	21.07	4919.89	4946.69	24.86
3D_45_6_c	3	45	6	89	102	79	80.96	16.46	4814.32	4849.22	19.51
3D_45_6_d	3	45	6	90	95	85	73.74	24.66	4876.59	4910.04	20.37
3D_45_6_e	3	45	6	89	85	96	81.59	24.51	4939.20	4974.62	20.26
3D_45_6_f	3	45	6	91	89	90	90.18	16.38	4754.97	4788.61	14.56
3D_50_6_a	3	50	6	105	110	85	84.60	27.99	4972.85	5036.79	34.36
3D_50_6_b	3	50	6	104	93	103	84.26	30.56	5079.52	5158.33	47.10
3D_50_6_c	3	50	6	116	90	94	93.29	23.81	4939.75	4984.51	30.05
3D_50_6_d	3	50	6	108	85	107	83.84	19.92	4569.16	4633.16	20.32
3D_50_6_e	3	50	6	99	109	92	84.02	15.33	5367.97	5407.29	22.57
3D_50_6_f	3	50	6	104	99	97	83.64	20.95	4999.46	5032.69	25.51
7D_30_6_a	7	30	6	51	56	73	82.67	33.13	9263.10	9300.74	28.16
7D_35_6_a	7	35	6	70	69	71	108.53	33.47	10704.00	10774.47	31.34
7D_40_6_a	7	40	6	84	87	69	113.17	26.81	10871.40	10936.50	46.94
7D_45_6_a	7	45	6	92	89	89	144.83	71.04	12142.20	12212.10	35.64
7D_50_6_a	7	50	6	98	99	103	160.47	68.76	13868.40	13951.27	52.55

tiple sets of multiple regions in which a configuration can be located. A method for obtaining near-optimal tours of the GTSPN is also proposed through use of a Hybrid Random-Key Genetic Algorithm. Numerical results show the HRKGA is consistent and efficient when solving for near-optimal tours with an average standard deviation error less than 1%.

The proposed method can be further extended to incorporate dynamic constraints into the formulation for motion planning applications. Another possible direction for the research will be to integrate the HRKGA onto actual UAV or redundant mobile manipulators. Real-world neighborhood scenarios will need to be developed, and a real-time collision-avoidance system will need to be integrated into the HRKGA.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton University Press, 2011.

[2] S. Srivastava, S. Kumar, R. Garg, and P. Sen, "Generalized traveling salesman problem through n sets of nodes," *CORS Journal*, vol. 7, pp. 97–101, 1969.

[3] K. Elbassioni, A. Fishkin, and R. Sitters, "Approximation Algorithms for Euclidean Traveling Salesman Problem with Discrete and Continuous Neighborhoods," *International Journal of Computational Geometry and Applications*, vol. 19, no. 2, pp. 173–193, 2009.

[4] L. Gueta, R. Chiba, J. Ota, T. Ueyama, and T. Arai, "Coordinated Motion Control of a Robot Arm and a Positioning Table With Arrangement of Multiple Goals," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 2252–2258.

[5] M. Saha, T. Roughgarden, J. Latombe, and G. Sánchez-Ante, "Planning Tours of Robotic Arms Among Partitioned Goals," *The International Journal of Robotics Research*, vol. 25, no. 3, pp. 207–223, 2006.

[6] S. Alatartsev, V. Mersheeva, M. Augustine, and F. Ortmeier, "On optimizing a sequence of robotic tasks," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 217–223.

[7] I. Gentilini, K. Nagamatsu, and K. Shimada, "Cycle time based multi-goal path optimization for redundant robotic systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1786–1792.

[8] I. Gentilini, "Multi-goal path optimization for robotic systems with redundancy based on the traveling salesman problem with neighborhoods," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, 2012.

[9] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 5321–5326.

[10] I. Gentilini, F. Margot, and K. Shimada, "The Traveling Salesman Problem with Neighborhoods: MINLP Solution," *Optimization Methods and Software*, vol. 28, no. 2, pp. 364–378, 2013.

[11] W. M. Spears and V. Anand, "A study of crossover operators in genetic programming," in *Methodologies for Intelligent Systems*, ser. Lecture Notes in Computer Science, 1991, vol. 542, pp. 409–418.

[12] L. Snyder and M. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006.

[13] R. Haupt and S. Haupt, *Practical genetic algorithms, Second Edition*. Wiley, 2004.

[14] K. Vicencio. (2014) Randomly generated GTSPN instances. [Online]. Available: http://robotics.pr.erau.edu/data/gtspn.zip

[15] I. Basel, John and T. Willemain, "Random Tours in the Traveling Salesman Problem: Analysis and Application," *Computational Optimization and Applications*, vol. 20, no. 2, pp. 211–217, 2001.