Hybrid Ant Colony Algorithms for Path Planning in Sparse Graphs

Kwee Kim Lim [a], Yew-Soon Ong [a], Meng Hiot Lim [b], Xianshun Chen[a], Amit Agarwal[b]

[a] School of Computer Engineering, Nanyang Technological University, Singapore 639798
[b] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798

{S7302871Z, asysong, emhlim, chen0040, PG0212208T}@ntu.edu.sg

Abstract

The general problem of path planning can be modeled as a traveling salesman problem which assumes that a graph is fully connected. Such a scenario of full connectivity is however not always realistic. One such motivating example for us is the application of path planning for unmanned reconnaissance aerial vehicles (URAVs). URAVs are widely deployed for photography or imagery gathering missions of sites of interest. These sites can be targets in a combat zone to be investigated or sites inaccessible by ground transportation, such as those hit by forest fires, earthquake or other forms of natural disasters. The navigation environment is one where the overall configuration of the problem is a sparse graph. Unlike graphs that are fully connected, sparse graphs are not always Hamiltonian. In this paper, we describe hybrid ant colony algorithms (HACAs) proposed for path planning in sparse graphs since existing ant colony solvers designed for solving TSP do not apply to the present context directly. HACAs represent ant inspired algorithms incorporated with a local search procedure and some heuristic techniques for uncovering feasible route(s) or path(s) in a sparse graph within tractable time. Empirical results conducted on a set of generated sparse graphs demonstrate the excellent convergence property and robustness of HACAs in uncovering low risk and Hamiltonian visitation paths. Further, the obtained results also indicate that HACAs converge to secondary closed paths in situations where a Hamiltonian cycle do not exists theoretically or is not attainable within the bounded computational time window.

Keywords: Sparse graph, path planning, URAV, optimization, hybrid ant colony algorithm.

1. Introduction

The ant colony system (ACS) is a recent distributed computational intelligence approach for combinatorial optimization problems and was first used to address the well-known traveling salesman problem [11, 12, 13, 14, 28]. Up to now, ant algorithms have been used extensively to solve a wide range of combinatorial optimization problems such

as quadratic assignment [16, 23], vehicle routing [6, 7, 15], sequential orderings [17], job shop scheduling [9, 29], telecommunication network routing [10, 26] and logic circuit design [8].

The ACS is inspired by the behavior of real ants searching for food. Real ants communicate with each other using a chemical substance called pheromone, which they leave on the paths they traverse [18]. This pheromone trail acts as a form of "shared memory" of path leading to promising food source. New ants joining the search, attracted by the pheromone, further reinforce the path. With time, a series of feasible routes are traversed by teams of ants. As the pheromone evaporates, solutions with no substantial travelers will have difficulty in maintaining their pheromone trails. This way, infeasible routes are gradually eliminated.

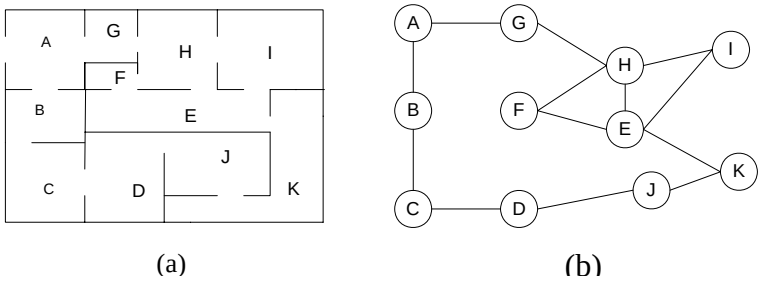
We focus on hybrid ant colony algorithms (HACAs) for path planning involving sparse graphs. The rationale behind this is that full connectivity is not always enforceable in real-world situations. A motivating real-world application for us is path planning for unmanned reconnaissance aerial vehicles (URAVs) [1-5]. Such a problem scenario presents unique challenges in terms of algorithm implementation, particularly in the design of suitable search operators and heuristics that are capable of driving the algorithm towards good quality solutions efficiently. The intention is to perform an offline path planning in the given sparse graph to uncover low risk visitation sequences. We begin with graphs that are unweighted and derived visitation sequences that cover each vertex exactly once. The HACAs are subsequently extended to weighted sparse graphs for uncovering closed visitation sequences that have low flight risks. It is worth noting that while the current study on path planning is demonstrated in the context of URAV, the proposed algorithms are generally applicable to other search problems that may be represented in the form of sparse graphs.

The remaining sections of this paper are organized in the following manner. Section 2 presents the modeling of the URAVs path planning problem as a sparse graph and highlighted the similarities to other existing problems. Section 3 outlines the proposed HACAs with details of the local search strategy, pheromone management policy and path construction rules. Section 4 presents an empirical study of the HACAs applied to URAV path planning, based on a series of generated graphs. Finally, Section 5 concludes this paper with a brief summary.

2. Modeling as Sparse Graph

The general problem of path planning is to uncover feasible route(s) or path(s) in the presence of constraints. In solving it, the scenario of the problem is first modeled as a graph. Graphs can be seen as representation for real world systems such as telecommunication networks, maps of roads, printed circuit boards routing, transportation systems, etc. [8, 19]. A graph is said to be sparse if the total number of edges is small compared to the total number of possible edges.

Or, more formally, in a sparse graph, $M = O(N)$ where N and M are the number of nodes and edges in the graph respectively. For instance, the map of connected cities shown in Figure 1a can be modeled by the sparse graph representation of Figure 1b.



(a) (b)

Figure 1: An 11-city map and its graphical representation. (a) Map of cities (b) Representation of cities as a graph

In the context of URAV path planning, a common objective is to locate low risk paths by avoiding known threats in an area of interest. Based on knowledge of the threats to be avoided, the paths of low risk can be represented using a Voronoi diagram. A simple illustration of a Voronoi diagram is given in Figure 2.

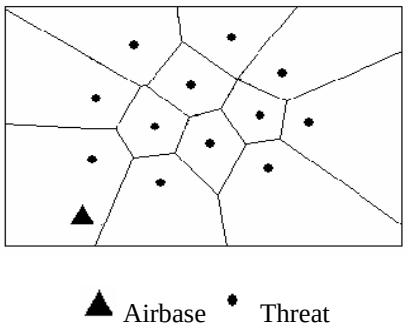


Figure 2: Voronoi diagram

Visually, it may be observed that the safe path corresponding to the Voronoi edges connects to form a sparse graph. The edges represent possible flight paths from one reconnaissance site to another. Weight may be assigned to each edge in Figure 2 to reflect the cost associated to the risk and distance. Hence, the area of interest for reconnaissance purposes can be modeled as a sparse graph and the configuration of minimal risk path for the URAV can be defined as follow:

For a weighted sparse graph G(V,E) where V represents a set of k waypoints or sites of interest {S_i; i=1,2…k-1} and a designated airbase A, find a path of minimal cost starting and ending at A. The path should traverse as many waypoints as possible, subject to each waypoint being visited exactly once.

Here we discuss the similarities and differences on the URAV path planning problem to well-known traveling salesman problems (TSP). While work on ant colony optimization algorithms for traveling salesman problem (ACS-TSP) have demonstrated much success [12], the assumption of a fully connected graph in TSP generally limits the applicability of existing ant colony algorithms for path planning. In the context of URAV path planning problem, the navigation environment is one where the overall configuration of the problem is a sparse graph. Unlike graphs that are fully connected, sparse graphs are not always Hamiltonian. Hence existing ACSs that were designed for solving TSP do not apply to sparse graphs directly. Furthermore, since the Hamiltonian property of the graph is not known *a priori*, the proposed search algorithm requires versatility in locating a Hamiltonian circuit if it exists but, also flexible in forgoing a potentially futile search for such a path. For such a situation, a secondary path with the largest possible visitation coverage is desirable.

A variation of the standard TSP problem in the context of sparse graphs is the generalized TSP (GTSP) [20, 21]. In this class of problems, the set of vertices is usually divided into multiple interconnected clusters. The focus of GTSP is then to locate the minimum cost path that contains at least one vertex in each cluster, since a salesman is expected to visit only one or more vertices in each cluster. In contrast, our work addresses the strict requirements posed by the motivating example of URAV path planning. It is to visit all vertices in the graph exactly once, using a hybrid ant colony algorithm. In the next section, we outline our novel hybrid ant colony algorithm to address the URAV path planning problem, represented as a sparse graph.

3. Hybrid Ant Colony Algorithm (HACA)

Uncovering Hamiltonian paths for sparse graphs is one of the fundamental objectives of the proposed HACA. The second objective of the HACA is to locate the minimum risk Hamiltonian path of weighted sparse graphs. However, in graphs where no Hamiltonian path exists, a secondary path with the largest possible visitation coverage in terms of the number of vertices while avoiding revisiting any of the vertices is desired.

The pseudo codes and corresponding functional flow diagram of the proposed HACA for path planning in sparse graphs is outlined in Figures 4 and 5, respectively. In the HACA search, all ants in the population start by performing a random walk across vertices in the sparse graph until each is trapped or returns back to the starting vertex. Figures 6(a) and (b) depict instances of an ant trapped in an acyclic path and the uncovering of a non-Hamiltonian path, respectively. Subsequently, all solutions or paths discovered by the ants then undergo local refinement [24,25] based on the Tour Improvement Procedure (TIMP).

Procedure::HACA
BEGIN
While (terminating condition is not satisfied)
Begin

- Generate a new population
  If (stagnation occurred)
  Begin
  Apply enforcement scheme (EFS)
  Else
  Apply path construction procedure (PCP)
  End If
- Perform tour improvement procedure (TIMP)
- Evaluate the ant population
- Perform pheromone updating and evaporation

End While
END

Figure 3: Outline of Hybrid Ant Colony Algorithm

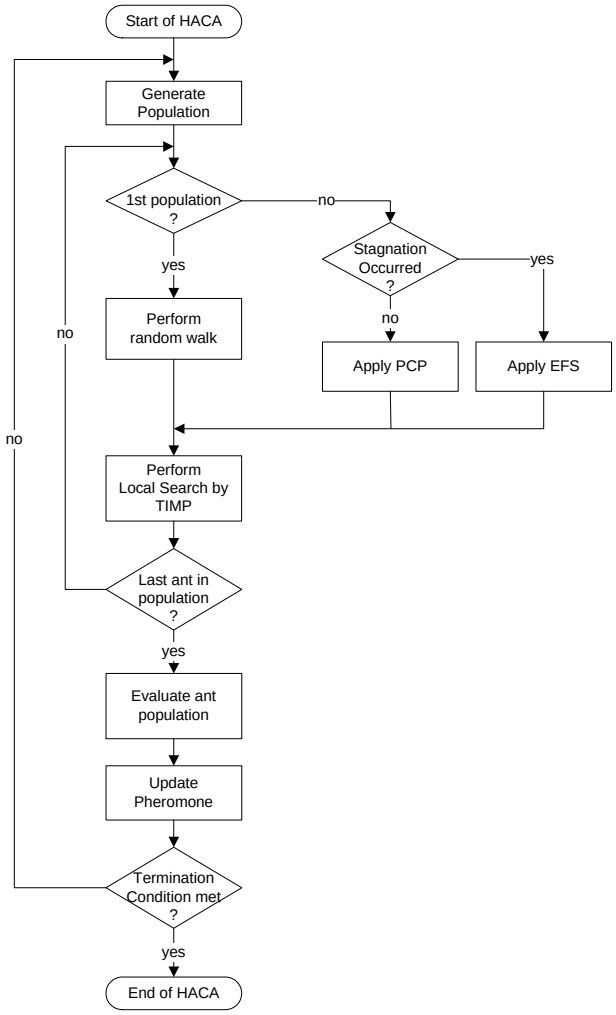


Figure 4: HACA functional flow diagram

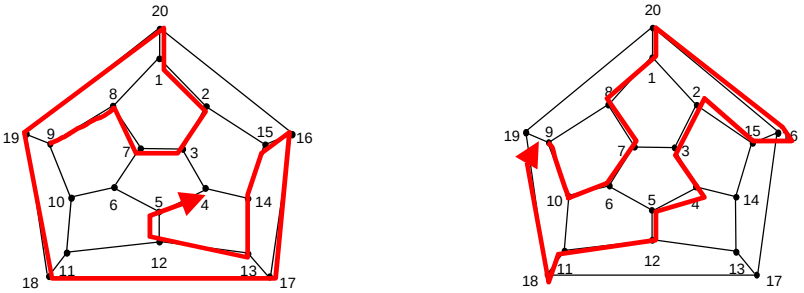


Figure 5: (a) Ant trapped in an acyclic path (b) Ant uncovers a non-Hamiltonian path

3.1 Fitness Evaluations

Using an integer representation, the solution path is encoded as a permutation, Π[*j*]. Consider the graph with 10 vertices given in Figure 6(a). The cyclic path shown in the figure illustrates a potential solution uncovered by an ant. The corresponding integer permutation is depicted in Figure 6(b), where *j* is the visitation order. Hence, the solution path is encoded as a sequence of vertices arranged in the order that an ant visits. From Π[*j*], the order of visited vertices, Π_v, and unvisited vertices, Π_{uv}, can be established. Note that the last vertex of Π_v is also the starting vertex.

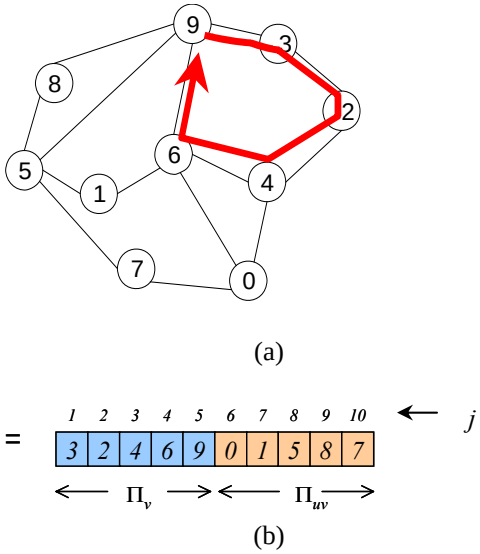


Figure 6: An example illustrating feasible solution path

A. Unweighted Sparse Graphs

For unweighted sparse graphs, the main objective is to uncover Hamiltonian paths. In such cases, the quality or fitness of solution path L_k, that an ant k discovers may be computed using Eq. (1).

$$L_k = \begin{cases} 1/N^2 & ; \text{if Hamiltonian path} \\ (N\text{-}V_i)/(N\text{-}1) & ; \text{if non-Hamiltonian cyclic path} \\ 1 & ; \text{if acyclic path} \end{cases} \tag{1}$$

Here, N refers to the total number of vertices while V_i is the number of vertices that an ant has covered i.e. $|\Pi_v|$. L_k is minimum if a Hamiltonian path is uncovered. Hence the number of possible solutions is equal to the number of Hamiltonian paths in a graph. On the other hand, the quality of a solution path is penalized according to the number of unvisited vertices, N-V_i, for non-Hamiltonian path. To avoid any non-Hamiltonian paths that are acyclic, a heavy penalty is applied to L_k.

B. Weighted Sparse Graphs

For a weighted sparse graph, besides uncovering a Hamiltonian path, the objective is also to locate the minimum risk Hamiltonian path. Here, the minimization fitness function, L_{kopt} is adapted from (1) and defined in Eq. (2):

$$\text{Min } L_{kopt} = L_k \Big/ (1 - C_e)$$

$$\text{where } C_e = \sum_{j=1}^{V_i} \varphi(j) \Big/ N \tag{2}$$

The fitness of a solution path that an ant converges to, is scaled based on the number of visited vertices and C_e, the total cost to cover these vertices. Note that $\varphi(j)$ represents the risk associated to the j^{th} vertex of the solution path in Π_v. Here, C_e and $\varphi(j)$ are normalized to values between 0 and 1. L_{kopt} is defined as a function of L_k and C_e. Hence L_{kopt} is optimum when C_e is minimum among all the possible Hamiltonian paths that exist in a sparse graph.

3.2 Tour Improvement Procedure (TIMP)

TIMP is a local search heuristic procedure adapted from the standard 2-opt method [22, 32]. Like the standard 2-opt method, non-adjacent pair of edges is exchanged. However, unlike fully connected graphs where the standard 2-opt method was designed for, random swapping of edges in the sparse graph could lead to undesirable solution paths,

i.e., a potentially good solution path may be broken into smaller disjoint sets. In effect, TIMP is designed to insert unvisited vertex members obtained from Π_{uv}, into the permutation of visited vertices, Π_v, based on a first improvement strategy.

We further differentiate between the TIMPs for unweighted and weighted sparse graphs as TIMP-E and TIMP-O, respectively. The basic outline of TIMP-E for uncovering Hamiltonian paths is depicted in Figure 7 while Figure 8 outlines the procedure to uncover low risk Hamiltonian paths in TIMP-O. In TIMP, a successful tour improvement is attained when any insertion of unvisited vertex results a lower value of L_k or L_{kopt}.

```
Procedure::TIMP_E
For each element i_uv in Π_uv
Begin
    For each element i_v in Π_v except the last element
    Begin
        If ((i_uv connects to i_v) and (i_v+1 connects to i_uv))
            Π_v ← Π_v + i_uv
            Π_uv ← Π_uv - i_uv
            Π[j] ← Π_v ∪ Π_uv
            break;
        End if
    End
End
```

Figure 7: Outline of tour improvement procedure for unweighted sparse graphs, TIMP-E

```
Procedure::TIMP_O
For each element i_uv in Π_uv
Begin
    For each element i_v in Π_v except the last element
    Begin
        If ((i_uv connects to i_v) and (i_v+1 connects to i_uv))
            Perform fitness evaluation
            If (old fitness > new fitness)
                Π_v ← Π_v + i_uv
                Π_uv ← Π_uv - i_uv
                Π[j] ← Π_v ∪ Π_uv
            End if
            break;
        End if
    End
End
```

Figure 8: Outline of tour improvement procedure for weighted sparse graphs, TIMP-O

3.3 State Transition Rule, Pheromone Deposition and Evaporation

After fitness evaluations, the best solution path uncovered by the ants is used to guide the search in subsequent generations. Here we consider two separate schemes of state transition rule as well as pheromone deposition and evaporation used in the HACA which differentiates between HACA-1 and HACA-2.

In HACA-1, the pseudo-random proportional state transition rule, local and global pheromone updating procedures (i.e,, based on the global best ant) of the standard ACS search scheme proposed by Dorigo et al in [12] is considered. In particular, the pseudo-random proportional state transition rule is defined as:

$$s = s_{HACA-1} = \begin{cases} \arg\max_{u \in J_c(r)} \tau(r,u), & if\ (p \le q) \\ s_{random-proportional}, & if\ (p > q) \\ rand\_assign\{u \mid u \in J_{uc}(r)\}, & otherwise \end{cases} \quad (3)$$

where *r* is the current vertex visited by an ant and *s* is the next vertex the ant will move to, $\tau(r,u)$ is the pheromone level on the edge *E(r, u).* $J_c(r)$ is the set of unvisited vertices connected to *r* whereas $J_c(r)$ is the set of unvisited vertices that are disconnected from *r*. $0 \le q \le 1$ is the exploitation biasing parameter. When the randomly generated variable *p* is smaller than or equal to *q*, *s* is assigned the unvisited vertex *u* whose edge *E(r, u)* has the highest pheromone from among the unvisited edges connected to *r* (i.e., a form of exploitation), otherwise *s* is assigned the vertex $S_{random-proportional}$ which represents the unvisited vertex selected based on the random proportional state transition rule introduced in [11] (i.e., a form of exploration). When $J_c(r)$ is empty (i.e., in cases that there are no connected and unvisited edges to *r*), *s* is randomly assigned a vertex from $J_{uc}(r)$ to derive at a complete solution path.

In addition, the pheromone level of visited edges is updated based on the local updating rule as follows:

$$\tau(r,s) = (1-\theta_l) * \tau(r,s) + \theta_l * \tau_0 \quad (4)$$

where $\tau(r,s)$ is the pheromone level on edge *E(r, s)* and τ_0 is the initial pheromone level. $0 \le \theta_l < 1$ is the user-defined pheromone evaporation.

In contrast to HACA-1, the ants in HACA-2 traverse the vertices based on a state transition rule that is defined by

$$s = s_{HACA-2} = \begin{cases} \arg\max_{u \in J_c(r)} \tau(r,u), & if\ (p \le q) \\ rand\_assign\{u \mid u \in J_c(r)\}, & if\ (p > q) \\ rand\_assign\{u \mid u \in J_{uc}(r)\}, & otherwise \end{cases} \quad (5)$$

To facilitate both exploitation and exploration in the HACA-2 search, an ant proceeds to a connected vertex with the highest pheromone only when $p \in [0,1]$, a randomly generated value, is found to be lower than or equal to the user-specific threshold *q*. What differentiates HACA-2 from HACA-1 lies in that the former choses a next vertex to traverse

from the list of connected yet unvisited vertices when p is larger than q, see Eq(5), while the latter uses the pseudo-random proportional state transition rule in choosing the next move, see Eq(3). In cases where there are no unvisited connected vertices left, any of the unvisited and unconnected vertices may be randomly chosen to derive a complete permutation of the solution path.

The same global updating rule are used by both the HACAs, i.e., HACA-1 and HACA-2. To be precise, the global evaporation of HACA-2 serves to eliminate poor solution paths by evaporating the pheromone level on the edges and is defined by

$$\tau(r,s) = (1-\theta_g) * \tau(r,s) \quad (6)$$

where θ_g denotes the rate of pheromone evaporation.

On the other hand, the global updating rule of the HACAs deposit pheromones only on edges that form the fittest path:

$$\tau(r,s) = \theta_g * \Delta\tau(r,s) \quad (7)$$

where $(1-\theta_g)$ is the pheromone deposition rate and $\Delta\tau(r,s)$ is the amount of pheromone deposited to strengthen the chances of visiting edge $E(r, s)$ in subsequent search while the global pheromone deposited on edge $E(r, s)$ is represented as $\tau(r,s)$. In HACA-1, the local and global evaporation rates, labeled as θ_l and θ_g , respectively in Eq (7) are assigned the same value. On the other hand, the local pheromone updating rule of HACA-2 considers a smaller θ_l such that $\theta_l << \theta_g$.

To uncover the Hamiltonian paths of an unweighted sparse graph, $\Delta\tau(r,s)$ is modeled here as

$$\Delta\tau(r,s) = 1/L_k \quad (8)$$

On the other hand, to uncover low risk Hamiltonian path having minimum path length on weighted sparse graph, $\Delta\tau(r,s)$ becomes

$$\Delta\tau(r,s) = 1/L_{kopt} \quad (9)$$

Upon construction of the solution path, further potential improvements are attained through the tour improvement procedure (TIMP) described in section 3.2. The entire reproduction process then iterates until the termination condition is met.

3.4 Diversity Enforcement Scheme

The traditional evaporation operator of standard ant colony algorithms is designed to prevent the search from getting trapped in local optima. However it is often the case that the evaporation operator may require many search generations to take effect. To circumvent this, we introduced a diversity enforcement scheme in the HACAs. The scheme operates by favoring vertices that have been left unexplored for some time during the search. This is achieved by maintaining a candidate set, $J_m(r)$ of unexplored vertices connected to the current visited vertex *r*. When premature convergence occurs, for instance, after *m* number of generations with no improvement in the path quality, all the ants are forced to navigate according to Eq. (10) in the next generation.

$$s = \begin{cases} enforce\_assign\{u \mid u \in J_m(r)\}, & if\ (stagnation) \\ s_{HACA-1} \quad or \quad s_{HACA-2} & otherwise \end{cases} \tag{10}$$

where $J_m(r)$ is the set of unexplored and connected vertices to *r* while s_{HACA-1} and s_{HACA-2} are defined by Eqns. (3) and (5) for HACA-1 and HACA-2, respectively. Unexplored neighboring vertices are given higher priorities to be chosen as the next connected vertex to visit. To describe the diversity enforcement scheme, we use the example given in Figure 9(a) which illustrates a solution path uncovered by an ant on a sparse graph. It may be observed that vertices 13, 14 and 17 have been left unexplored by the ants for some time. If there is no improvement in solution path quality after *m* number of generations by the ants, the enforcement scheme kicks in to force the ants to use only unexplored connected vertices in the set $J_m(r)$ that contains the unvisited vertices. For instance, ants at vertex 16 are only allowed to proceed to vertex 13, 14 or 17 as depicted in Figure 9(b).

In addition, if the search quality remains to stagnate even after the EFS operation, the pheromone trails are forced to evaporate more rapidly by increasing the global evaporation rate θ_g to θ_e where $\theta_e >> \theta_g$. This represents a form of adaptation in the HACA according to the state of the search and help facilitates a faster transition from search exploitation to greater search exploration.

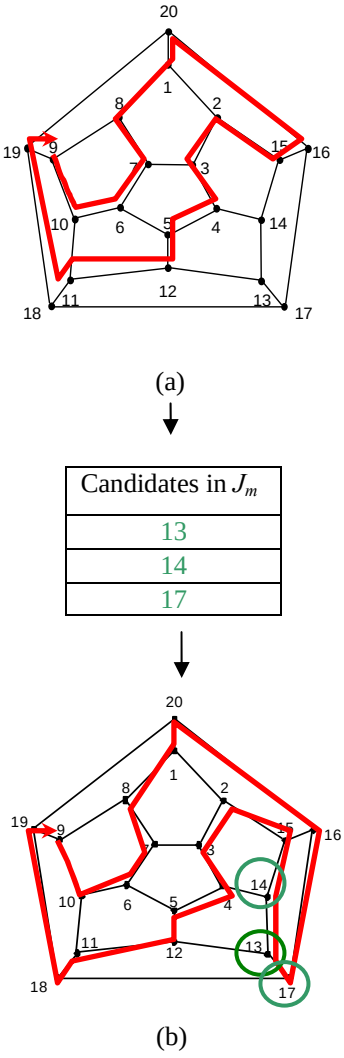


Figure 9: An example to illustrate the operation of diversity enforcement scheme

4. Empirical Study

In this section, we present an empirical study on HACAs, i.e., HACA-1 and HACA-2, for path planning problem using a series of generated sparse graphs of varying complexity that depict the locality and distribution of reconnaissance sites. To illustrate the efficacy of the proposed HACAs, the search performance is compared to that of a standard ant colony system search scheme which is labeled here as ACS in short [11],[12]. The generated sparse graphs used to validate our algorithm are derived from the famous Tutte graph, known to be non-Hamiltonian. In the Tutte graph, the non-Hamiltonic property of the graph is preserved if the connectivity constraints of certain critical nodes are preserved. From the basic Tutte graph, other more complicated graphs can be derived by joining two or more graphs; at least one of it being a Tutte graph if the non-Hamiltonic property of the graph is to be preserved. It is noted that our algorithm does not assume any *a priori* knowledge of the connectivity of the graphs used in testing.

The initial nine sparse graphs considered here vary in terms of 1) Hamiltonian property, 2) number of vertices, $|V|$, 3) number of edges, $|E|$. For the sake of brevity, we classify the nine graphs into 2 categories, namely,

- HC: Hamiltonian Graphs
- NHC: Non-Hamiltonian graphs. These problems were generated to evaluate the ability of HACA to uncover secondary paths.

These nine graphs are depicted in Figures 10 and 11 and is available for download at [30]. HC01 is the graph of a regular solid dodecahedron. The graph is 3-connected. Graph HC02 was obtained from Graph HC01 through a sequence of edge operations – a Hamiltonian cycle was first uncovered in the latter, following which three edges were deleted and ten edges were added while preserving its Hamiltonian property. Graph NHC01 is the Tutte graph – a non-Hamiltonian 3-connected cubic graph. Graphs HC03 through HC07 and, NHC02 and NHC03 are derived by applying a sequence of edge addition or deletion operations to the Tutte graph. In graphs NHC02 and NHC03, the subgraph comprising edges 44-48, 48-46, 46-49, 49-45, 46-47 and 47-43 precludes the presence of Hamiltonian cycle.

In our experimental study, the configurations of the HACAs and ACS control parameters used are summarized in table 1. Several criteria have been defined here to assess the performance of the HACA for path planning on weighted and unweighted sparse graphs. They are listed in tables 2 and 3, respectively. Among these criteria, *CPU time* is used to measure the computational cost of the algorithms in wall-clock time. *BestGen* provide a measure on the convergence rate of the algorithms in terms of the generation size. *Average or Best cycle lengths/path costs, Average cycle lengths/path costs gap* and *Success rate* serve as the criteria for measuring the solution quality of the algorithms considered. On weighted sparse graphs, *BPC* and *AvePC* report the shortest and average path cost found in all the HACA runs, respectively, while *AvePCGap* measures the gap between the best (shortest path cost) and average solution quality (average path cost) attained.

For each problem, the average performance of ten optimization runs is reported. It is worth highlighting that the objectives and termination conditions differ for unweighted and weighted sparse graphs since greater computational efforts are required in the latter. The differences are stated as follows:

4.1 Unweighted Sparse Graphs

For HC problems, HACA is evaluated based on the ability to uncover a Hamiltonian path if it exists. On the other hand, the ability of HACA in locating a secondary path with maximum coverage of vertices is considered for NHC problems. In the experimental study, the search terminates whenever a Hamiltonian cycle is found or when a maximum generation of 50,000 is reached for both HC and NHC problems.

The results obtained from the empirical study on the HACAs for unweighted sparse graphs are tabulated in Tables 4 and 5. Based on the results, both HACAs are observed to be capable of uncovering a Hamiltonian path on all the six unweighted sparse graphs considered. A *success rate* of 100% across all 10 runs is obtained for HACAs on the HC problems, expect for HACA-1 which scores 80% *success rate* on HC03. ACS on the other hand is observed to be less robust since it presents 100% *success rate* on only 3/6 HC graphs considered, i.e., HC01, HC02, HC05, and fares poorly on HC03 with only 30% *success rate* in locating a Hamiltonian path and obtained an average cycle gap, i.e., *AveCLGap* of 1.52%. For NHC problems, HACA-2 and ACS were flexible enough to forgo the search for Hamiltonian by converging on the solution path with maximum coverage of vertices within the computational time budget imposed. HACA-1 on the other hand finds the best-known cycle length on 9 out of the 10 independent runs, i.e., *CLSuccRate*=90%, on NHC01 while always finding the solution path with maximum coverage of vertices on both NH02 and NH03 in all 10 runs.

4.2 Weighted Sparse Graphs

On weighted sparse graphs, the HACAs are evaluated according to its robustness in uncovering the lowest risk Hamiltonian path with minimum path cost for HC problems. Similarly, the efficacy of the HACAs and ACS in uncovering low risk secondary path with the largest coverage of vertices and minimum path cost is reported for NHC problems. For weighted sparse graphs, the search terminates when a maximum of 500,000 generations is reached.

On weighted sparse graphs, HACA-2 was capable of uncovering a Hamiltonian path consistently. This is observed in Tables 6 and 7 where a *success rate* of 100% on the *CLSuccRate* criterion is obtained by HACA-2 across all 10 runs on the HC and NHC weighted sparse graphs. HACA-1 although underperforms HACA-2 in uncovering at least a Hamiltonian path consistently on all problems, it displays a success rate of 100% on 4/6 and 3/3 for the HC and NHC weighted sparse graphs considered respectively, as opposed to 2/6 and 2/3 on the same set of problems in the case of ACS. In Tables 6 and 7, note that the seemingly smaller values of *BPC* and *AvePC* for ACS does not indicate the ability to converge to lower risks solution path. On the contrary, the small values of *BPC* and *AvePC* highlight the failure of ACS in uncovering a Hamiltonian cycle or the longest secondary path but converge to shorter solution paths that do not transverse across all nodes. Hence, *BPC* and *AvePC* only serve as useful metric when the algorithm locates a Hamiltonian cycle or the longest secondary path, i.e., *CLSuccRate is 100%.*

Further, in comparison to ACS, the HACAs display greater efficacy in uncovering a Hamiltonian path or low risk secondary solution path on the sparse graphs. The low percentage in the average path cost gap even on the sparse

graphs, i.e., *AvePCGap* < 5%, suggests that HACAs always converge to or near to the best known solution, see tables 6 and 7.

4.3 Large Sparse Graphs

Here, we consider further the HACAs on more complex problems, i.e., unweighted sparse graphs greater than 100 vertices and various edge densities, i.e., HC07, HC08, HC09, HC10. Interestingly, Judging from *CLSuccRate*, HACA-1 was always capable of converging to the Hamiltonian solution path on all 4 large sparse problems. HACA-2 on the other hand fares badly on the large sparse graphs suggesting that it is not suitable for solving large sparse problems.

To summarize, the present study demonstrate that the capability and robustness of the HACAs over ACS for uncovering Hamiltonian paths or a low risk Hamiltonian path in both unweighted and weighted sparse graphs, respectively. The results obtained also established the ability and flexibility of HACAs in forgoing the search on Hamiltonian path for non-Hamiltonian graphs. In particular, HACAs provide a solution path for URAV having good coverage of vertices with low risks. Further, HACA-1 is shown to be appropriate for all forms of sparse problems, i.e., it is relatively more robust than HACA-2 on small, medium and large sparse graphs. Nevertheless, since the HACA was proposed and developed for the purpose of incorporating into URAV navigation system in which the complexity of the problem only ranges to less than 100 nodes, HACA-1 remains the algorithm of choice to use for handling such problems due to its high robustness and efficiency on both unweighted and weighted sparse graphs.

Table 1 Parameters setting for the HACAs and ACS.

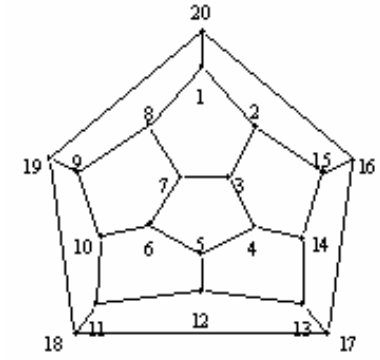
HACA parameters	**Value**
Population Size	30
Global pheromone evaporation Rate, θ_g	0.1
Local pheromone evaporation Rate, θ_l	0.1 (HACA-1) 0.02 (HACA-2)
Probability of Exploitation, *q*	0.8
Maximum Generations	50,000 (unweighted) 500,000 (weighted)
EFS global evaporation rate, θ_e	0.6
Number of no improvement Generations before stagnation happened, *m*	300

Table 2. Performance measure for unweighted sparse graphs

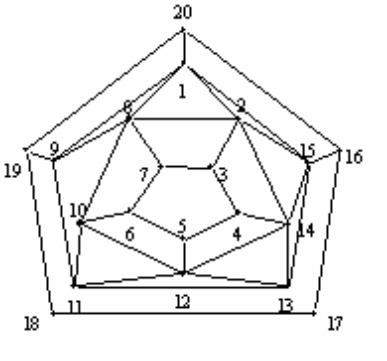
Criterion	Definition
CPU time	Average computation time in seconds upon uncovering of best-known solution
BestGen	Best number of generations elapsed before the occurrence of the best-known solution
BCL	Best cycle length obtained among all the HACA runs.
AveCL	Average cycle length (in term of vertex) obtained for all the HACA runs.
AveCLGap	Difference between the average cycle length (*AverCL)* and the best-known cycle length (bkcl) of the test problem *AveCLGap = (bkcl - AveCL)/bkcl*100%* where *bkcl* is the best-known cycle length of the test problem
CLGap	Difference between the best-found cycle length (BCL) and the best-known cycle length (bkcl) of a test problem. *CLGap = (bkcl - BCL)/bkcl*100%* where bkcl is the best-known cycle length
CLSuccRate	Number of times the algorithm finds the best-known cycle length of all the HACA runs.
bkcl	Best known cycle length

Table 3. Performance measure for weighted sparse graphs

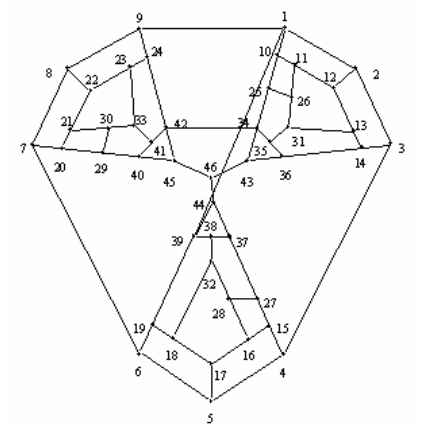
Criterion	Definition
BPC	Best path cost among all the HACA runs
AvePC	Average path cost obtained for all the HACA runs
AvePCGap	Difference between the average path cost found (*AvePC)* and the best found path cost (BPC) of the test problem *AvePCGap = (AvePC - BPC)/BPC*100%*



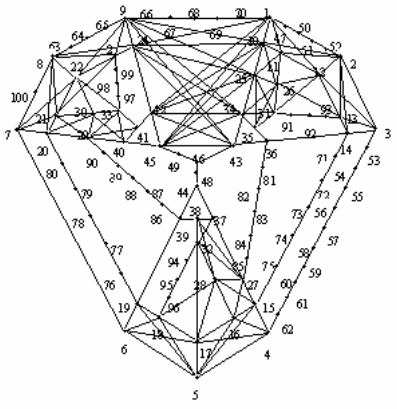
HC01: $|V| = 20$, $|E| = 30$, $E_d \approx 0.1579$



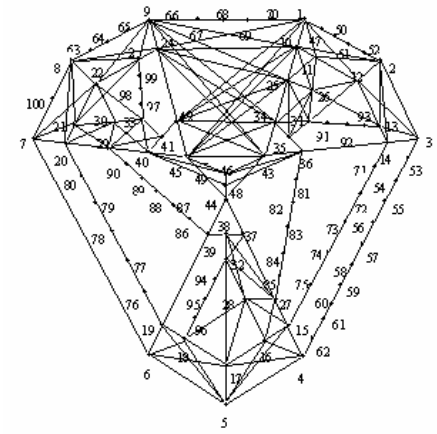
HC02: $|V| = 20$, $|E| = 37$, $E_d \approx 0.1947$



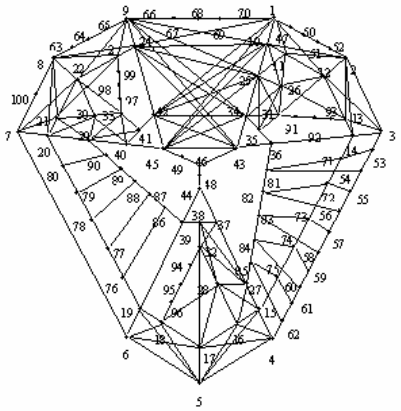
HC03: |V| = 46, $|E| = 71$, $E_d \approx 0.0686$



HC04: |V| = 100, $|E| = 197$, $E_d \approx 0.0398$

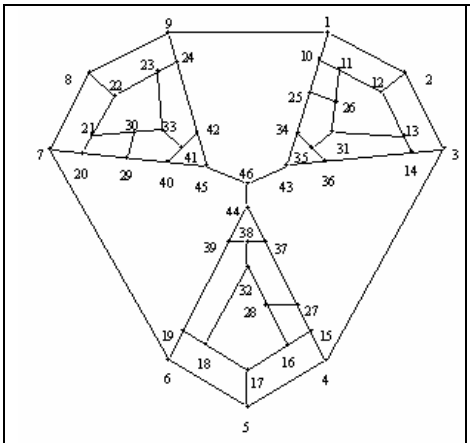


HC05: |V| = 100, $|E| = 203$, $E_d \approx 0.0410$

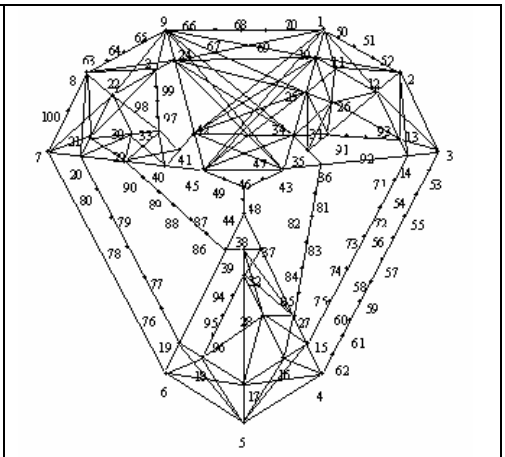


HC06: |V| = 100, $|E| = 217$, $E_d \approx 0.0438$

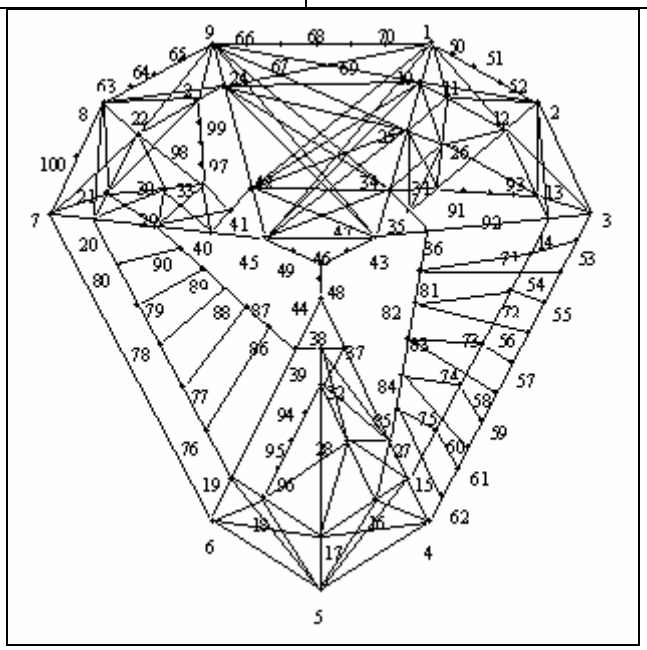
Figure 10: Sparse Hamiltonian graphs



NHC01: |V| = 46, |*E*| = 69, $E_d \approx 0.0667$



NHC02: |V| = 100, |*E*| = 197, $E_d \approx 0.0398$



NHC03: |V| = 100, |E| = 207, $E_d \approx 0.0418$

Figure 11: Sparse Non-Hamiltonian graphs

Table 4. Simulation results of HACA-1, HACA-2 and ACS for HC graphs.

Unweighted Hamiltonian Sparse graphs

	(V,E)	Method	CPU time	BestGen	BCL	AveCL	AveCLGap	CLGap	CLSuccRate
HC01	(20,30)	HACA-1	0.0015	2	20	20.00	0.00%	0.00%	100.00%
bkcl =20		HACA-2	0.0095	1	20	20.00	0.00%	0.00%	100.00%
		ACS	0.0015	2	20	20.00	0.00%	0.00%	100.00%
HC02	(20,37)	HACA-1	0.0015	2	20	20.00	0.00%	0.00%	100.00%
bkcl =20		HACA-2	0.0063	1	20	20.00	0.00%	0.00%	100.00%
		ACS	0.01	4	20	20.00	0.00%	0.00%	100.00%
HC03	(46,71)	HACA-1	22.81	12	46	45.80	0.43%	0.00%	80.00%
bkcl =46		HACA-2	5.6266	226	46	46.00	0.00%	0.00%	100.00%
		ACS	9.12	6	46	45.30	1.52%	0.00%	30.00%
HC04	(100,197)	HACA-1	20.91	114	100	100.00	0.00%	0.00%	100.00%
bkcl =100		HACA-2	4.8249	270	100	100.00	0.00%	0.00%	100.00%
		ACS	52.9	54	100	99.90	0.10%	0.00%	90.00%
HC05	(100,203)	HACA-1	3.53	42	100	100.00	0.00%	0.00%	100.00%
bkcl =100		HACA-2	5.0203	294	100	100.00	0.00%	0.00%	100.00%
		ACS	9.05	70	100	100.00	0.00%	0.00%	100.00%
HC06	(100,217)	HACA-1	20.58	68	100	100.00	0.00%	0.00%	100.00%
bkcl =100		HACA-2	17.4388	205	100	100.00	0.00%	0.00%	100.00%
		ACS	28.09	77	100	99.80	0.20%	0.00%	80.00%

Table 5. Simulation results of HACA-1, HACA-2 and ACS for NHC graphs.

Unweighted Non-Hamiltonian Sparse Graphs

	(V,E)	Method	CPU time	BestGen	BCL	AveCL	AveCLGap	CLGap	CLSuccRate
NHC01	(46,69)	HACA-1	13.16	5	45	45	0.22%	0.00%	90.00%
bkcl =45		HACA-2	33.24	29	45	45	0.00%	0.00%	100.00%
		ACS	1.98	6	45	45	0.00%	0.00%	100.00%
NHC02	(100,197)	HACA-1	21.63	50	99	99	0.00%	0.00%	100.00%
bkcl =99		HACA-2	139.37	382	99	99	0.00%	0.00%	100.00%
		ACS	22.09	390	99	99	0.00%	0.00%	100.00%
NHC03	(100,216)	HACA-1	90.21	48	99	99	0.00%	0.00%	100.00%
bkcl =99		HACA-2	144.6266	274	99	99	0.00%	0.00%	100.00%
		ACS	11.72	98	99	99	0.00%	0.00%	100.00%

Table 6. Simulation results of HACA-1, HACA-2 and ACS for weighted HC graphs.

Weighted Hamiltonian Sparse graphs

	(V,E)	Method	CPU time	BestGen	BCL	AveCL	AveCLGap	CLGap	CLSuccRate	BPC	AvePC	AvePCGap
HC01	(20,30)	HACA-1	0.00	2	20	20.0	0.00%	0.00%	100%	37	37.00	0.00%
bkcl =20		HACA-2	0.12	4	20	20.0	0.00%	0.00%	100%	32	32.00	0.00%
		ACS	0.00	2	20.0	20.00	0.00%	0.00%	100%	36	36.67	1.85%
HC02	(20,37)	HACA-1	0.00	2	20	20.0	0.00%	0.00%	100%	135	136.8	1.33%
bkcl =20		HACA-2	0.11	46	20	20.0	0.00%	0.00%	100%	130	130	0.00%
		ACS	0.01	2	20.0	20.00	0.00%	0.00%	100%	142	158.67	11.74%
HC03	(46,71)	HACA-1	39.19	60	46	44.8	2.61%	0.00%	20%	872	902.20	3.46%
bkcl =46		HACA-2	47.69	3,221	46	46.0	0.00%	0.00%	100%	912	912.00	0.00%
		ACS	296.51	39	46.0	45.17	1.81%	0.00%	50%	873	1007.17	15.37%
HC04	(100,197)	HACA-1	277.45	83	100	100.0	0.00%	0.00%	100%	3,762	3,881.50	3.18%
bkcl =100		HACA-2	300.54	412	100	100.0	0.00%	0.00%	100%	3,808	3,886.20	2.05%
		ACS	231.06	296	100	99.17	0.83%	0.00%	83.33%	3,695	3791.67	2.62%
HC05	(100,203)	HACA-1	49.83	991	100	100.0	0.00%	0.00%	100%	3,153	3,244.30	2.90%
bkcl =100		HACA-2	505.06	1,785	100	100.0	0.00%	0.00%	100%	3,173	3,205.30	1.02%
		ACS	31.22	329	100	99.17	0.83%	0.00%	83%	3,107	3212.00	3.38%
HC06	(100,217)	HACA-1	903.96	169	99	98.9	1.10%	1.00%	0%	2,768	2,886.20	4.27%
bkcl =100		HACA-2	20.95	1,410	100	100.0	0.00%	0.00%	100%	2,646	2,736.10	3.41%
		ACS	54.63	176	99	98.17	1.83%	1.00%	0%	2,662	2759.50	3.66%

Table 7. Simulation results of HACA-1, HACA-2 and ACS for weighted NHC graphs.

Weighted Non-Hamiltonian Sparse graphs

	(V,E)	Method	CPU time	BestGen	BCL	AveCL	AveCLGap	CLGap	CLSuccRate	BPC	AvePC	AvePCGap
NHC01	(46,69)	HACA-1	17.63	110	45	45.0	0.00%	0.00%	100%	2,144	2,164.10	0.94%
bkcl =45		HACA-2	94.078	500	45	45.0	0.00%	0.00%	100%	2,121	2,134.40	0.63%
		ACS	141.64	758	45	45.0	0.00%	0.00%	100%	2,166	2,293.67	5.89%
NHC02	(100,197)	HACA-1	122.57	617	99	99.0	0.00%	0.00%	100%	4,199	4,398.70	4.76%
bkcl =99		HACA-2	319.265	1,174	99	99.0	0.00%	0.00%	100%	4,443	4,501.40	1.31%
		ACS	719.99	237	99	98.2	0.84%	0.00%	83.33%	4,115	4,309.17	4.72%
NHC03	(100,216)	HACA-1	370.65	594	99	99.0	0.00%	0.00%	100%	4,001	4,077.00	1.90%
bkcl =99		HACA-2	863.46	1,474	99	99.0	0.00%	0.00%	100%	4,123	4,304.60	4.40%
		ACS	684.21	2,868	99	99.0	0.00%	0.00%	100%	3,915	4,133.17	5.57%

Table 8. Simulation results of HACA-1 and HACA-2 for unweighted graphs on large sparse graphs, i.e., > 100 nodes.

Unweighted Non-Hamiltonian Sparse Graphs

	(V,E)	Method	CPU time	BestGen	BCL	AveCL	AveCLGap	CLGap	CLSuccRate
HC07	(120, 240)	HACA-1	7.4	254	120	120.00	0.00%	0.00%	100.00%
bkcl=120		HACA-2	196.3203	1	120	118.40	1.33%	0.00%	40.00%
HC08	(146, 289)	HACA-1	109.47	861	146	146.00	0.00%	0.00%	100.00%
bkcl=146		HACA-2	1373.05	16249	138	136.71	6.36%	5.48%	0.00%
HC09	(200, 400)	HACA-1	39.59	413	200	200	0.00%	0.00%	100.00%
bkcl =200		HACA-2	1298.095	440	187	181	9.35%	6.50%	0.00%
HC11	(200, 414)	HACA-1	97.78	644	200	200	0.00%	0.00%	100.00%
bkcl =200		HACA-2	1296.166	700	193	184	8.10%	3.50%	0.00%

5. Conclusion and future work

Ant inspired algorithms incorporating a local search procedure and some heuristic techniques are presented for uncovering feasible optimal route(s) or path(s) in a sparse graph within tractable time. The motivating real world example for us is path planning for unmanned reconnaissance aerial vehicles (URAVs). Experimental study on test problems of various complexity including generated sparse graphs of known optimum shows that the proposed HACAs are highly robust, capable of converging to good quality solution paths having minimal risk efficiently. In addition, the HACAs also converge to the secondary path of minimal risk on non-Hamiltonian sparse graphs.

Acknowledgements

The authors acknowledge the contributions of Intellisys, a centre for research collaboration between NTU and ST Engineering.

References

1. Agarwal A, Lim MH, Er MJ, Nguyen TN (2007), Rectilinear Workspace Partitioning for Parallel Coverage Using Multiple UAVs. Advanced Robotics, 21(1-2): 105-120.
2. Agarwal A, Lim MH, Xu YL, Ong YS (2003), Evolutionary Graph Mining for the Discovery of Site Visitation Sequences for a Single URAV. 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems, 2003.
3. Agarwal A, Lim MH, Er MJ (2004), Model-Solution Framework for Minimal Risk Planning for URAVs. Military and Security Applications of Evolutionary Computation Workshop, GECCO, Seattle, USA.
4. Agarwal A, Lim MH, Chew CY, Poo TK, Er MJ, Leong YK (2004), Solution to the Fixed Airbase Problem for Autonomous URAV Site Visitation Sequencing. In: Proc Genetic and Evolutionary Computation Conference, Seattle, USA 2:850-858.
5. Agarwal A, Lim MH, Er MJ, Maung YWK (2004), Inflight Rerouting for an Unmanned Aerial Vehicle. In: Proc Genetic and Evolutionary Computation Conference, Seattle, USA 2:859-868.
6. B. Bullnheimer, R.F. Hartl, and C. Strauss (1999), Applying the ant system to the vehicle routing problem, In: Voss S., Martello S., Osman I.H., Roucairol C. (eds.) MetaHeuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston.

7. N. Christofides, A. Mingozzi, P. Toth (1979), The Vehicle Routing Problem, in: N. Christofides, A. Mingozzi, P. Toth, C. Sandi (Eds.), Combinatorial Optimization, Wiley, Chichester, pp. 315-338.
8. Coello CAC, Gutiérrez RLZ, García BM, Aguirre AH (2002), Automated Design of Combinational Logic Circuits using the Ant System. Engineering Optimization 34(2): 109-127.
9. Colorni A, Dorigo M, Maniezzo V, Trubian M (1994), Ant system for Job-Shop Scheduling. Belgian Journal of Operations Research, Statistics and Computer Science 34(1):39-53.
10. Di Caro G., Dorigo M. (1998), AntNet: Distributed Stigmergetic Control for Communications Networks, Journal of Artificial Intelligence Research, 9: 317-365.
11. Marco Dorigo, Gianni Di Caro, Luca M. Gambardella (1999), Ant Algorithms for Discrete Optimization. Artificial Life, MIT Press 5 (2): 137–172.
12. Dorigo M, Gambardella LM (1997), Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1): 53-66.
13. Dorigo M, Gambardella LM (1997), Ant Colonies for the Traveling Salesman Problem. BioSystems 43: 73-81.
14. Flood MM (1956), The Traveling Salesman Problem. Operations Research 4: 61-75.
15. Forsyth P, Wren A (1997), An Ant System for Bus Driver Scheduling. 7th Int Workshop on Computer-Aided Scheduling of Public Transport, Boston, USA.
16. Gambardella LM, Taillard E, Dorigo M (1999), Ant Colonies for the Quadratic Assignment Problem. J of the Operational Research Society 50: 167-176.
17. Gambardella LM, Dorigo M (1997), HAS-SOP: An Hybrid Ant system for Sequential Ordering Problem. Tech Rep No IDSIA 97-11, Lugano, Switzerland.
18. Goss S, Beckers R, Deneubourg JL, Aron S, Pasteels JM (1990), How Trail Laying and. Trail Following Can Solve Foraging Problems for Ant Colonies. In: R.N.Hughes (ed.) NATO-ASI Series Behavioral Mechanisms of Food Selection.
19. Gross J, Yellen J, Raton B (1999) Graph Theory and its Applications: CRC Press series on Discrete Mathematics and its Applications, ISBN: 0849339820.
20. Laporte G, Vaziri AA, Srikandarajah C (1996), Some Applications of the Generalized Travelling Salesman Problem. Journals of Operational Research Society 47: 1461-1467.
21. Labordere ALH (1969), The Record Balancing Problem: a Dynamic Programming Solution of a Generalized Traveling Salesman Problem. RIBO B-2, 736-743.

22. Lin S, Kernighan B (1973), An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research, 21(2), pp. 498-516.
23. Maniezzo V, Colorni A (1999), The Ant System Applied to the Quadratic Assignment Problem. IEEE Trans on Knowledge and Data Engineering 11(5): 769-778.
24. Ong YS and Keane AJ (2004), Meta-Lamarckian Learning in Memetic Algorithm. IEEE Transactions On Evolutionary Computation, 8(2): 99-110.
25. Ong YS, Lim MH, Zhu N and Wong KW (2006), Classification of Adaptive Memetic Algorithms: A Comparative Study. IEEE Transactions on Systems, Man and Cybernetics - Part B, 36(1): 141-152.
26. Sandalidis HG, K. Mavromoustakis K, Stavroulakis P (2001), Performance Measures of an Ant based Decentralised Routing Scheme for Circuit Switching Communication Networks. Soft Computing - A Fusion of Foundations, Methodologies and Applications 5 (4): 313-317.
27. Schoonderwoerd R, Holland O, Bruten J, Rothkrantz L (1997) Ant-based Load Balancing in Telecommunications Networks. Adaptive Behavior 5(2): 169-207.
28. Stützle T, Hoos H (1997), The MAX-MIN Ant system and Local Search for the Traveling Salesman Problem. In: Proc 4th IEEE Int Conf on Evolutionary Computation, IEEE Press: 308-313.
29. Ying KC, Liao CJ (2004), An Ant Colony System for Permutation Flow-Shop Sequencing. Source Computers and Operations Research Archive 31(5): 791-801.
30. Sparse Graphs, http://www.ntu.edu.sg/home/asysong/sparse/default.htm
31. University of Heidelberg, Department of Computer Science. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95
32. Lim M H, Yu Yuan and Omatu S (2002), Extensive Testing of A Hybrid Genetic Algorithm for Solving Quadratic Assignment Problems. Computational Optimization and Applications, Kluwer Academic Publishers, 23: 47-64.