CSCE 686 Complex Optimization Algorithm Design Project

1Lt Chip Van Patten*
Air Force Institute of Technology
Dayton, Ohio 45433
Email: *donald.vanpatten@afit.edu

***Abstract*—This project addresses a problem in the field of Unmanned Aircraft Systems dealing with assigning UAS orbits. Given a limited number of UAVs, each based in specific locations – each with specific ranges of flight and operating costs – and a list of targets to surveil, how do we assign surveillance orbits to the UASs such that all targets are surveilled while keeping the UASs close enough to their base of operations that they may safely return, either when their mission finishes or they've exhausted their fuel, at minimum cost? This project attempts to integrate this application domain with the set cover problem domain and a number of algorithm domains to solve the problem efficiently.**

I. Introduction

This project addresses a problem in the field of Unmanned Aircraft Systems (UASs) dealing with assigning UAS orbits to cover a list of targets. Given a limited number of UAVs, each assigned to specific units based in specific locations – each with specific ranges of flight and operating costs – and a list of targets to surveil, how do we assign surveillance orbits to the UASs such that all targets are surveilled while keeping the UASs close enough to their base of operations that they may safely return, either when their mission finishes or they've exhausted their fuel, at minimum cost? This project attempts to integrate this application domain with the set cover problem domain and a number of algorithm domains to solve the problem efficiently.

In this paper, we implement a deterministic algorithm and a stochastic algorithm to solve the UAS problem. According to the No Free Lunch (NFL) Theorem, "no search algorithm is better than a random search on the space of all possible problems - in other words, if a particular algorithm does better than a random search on a particular type of problem, it will not perform as well on another type of problem, so that all in all, its global performance on the space of all possible problems is equivalent to a random search." [6], [28] Stated another way, the "NFL results show that if there is no assumption regarding the relation between visited and unseen search points, efficient search and optimization is impossible." [7] Therefore, while the implemented algorithms solve this hyper-constrained UAS problem, there's no guarantee of their effectiveness or efficiency in solving other related UAS problems.

The rest of this paper is organized as follows. Section II, discusses the problem domain and UAS application problem domain discussed briefly at the beginning of this section. In Section III, the Algorithm domain is discussed and the disciplined, top-down algorithm design process from [20], [18], [17] is used to integrate the problem, application, and algorithm domains ending with a coded implementation in Section III-C. Results and analysis of the implementation, tested in accordance with the guidelines put forth in [2], can be found in Section IV. Finally, Section V concludes with a summary of the project.

II. CSCE 686 Project Algorithmic Design Process

This section details the algorithmic design process using the top down disciplined design approach specified in [20], [18]. The problem domain is discussed in Section II-A. Integration of the problem domain with the application domain is detailed in Section II-B. The algorithm domain is discussed Section III where the initial integration of the algorithm and problem domains begins, data structures from III are refined and integrated into the program design. Pseudocode for the deterministic and stochastic algorithms are presented in Sections III-A4 and III-B4 respectively. This section concludes with the coded implementations, based on the aforementioned pseudocode, found in Section III-C.

A. Problem Domain

1) Problem Domain Discussion: The set cover problem is a class of problems in which one seeks to select a minimum number of sets, at minimum cost, so that the union of all selected sets contains all elements in the input [24]. More formally, given a universe, or ground set of elements, $\mathcal{R} = \{r_1, \ldots, r_m\}$ and a family of sets $\mathcal{S} = \{S_1, \ldots, S_N\}$, a set covering of $\mathcal{R}$ exists if

$$\bigcup_{i=1}^{k} S_{j_i} = \mathcal{R},$$

with S_{j_i} being the covering sets [5, pp. 39-40]. Christofides further state that if, for each $S_j \in \mathcal{S}$, "there is associated a (positive) cost c_j" [5, p. 39], then the goal of the set cover problem is to find a set of sets $\mathcal{S}'$ which includes all m elements of $\mathcal{R}$ while minimizing $\sum_{i=1}^{k} c_j$ [5], [24], [26], [33].

In the classic, or un-weighted, set cover problem, all $c_j = 1$ [24]. In this paper we deal with a weighted variant of the set cover problem, consequently, for the purposes of this paper, the term *set cover problem* is synonymous with *weighted set cover problem*, but the former is be used exclusively.

2) Symbolic Set Development and Discussion: We now develop a symbolic set (adapted from [5], [18]) which formalize the notation used throughout this paper and, hopefully, aid in understanding and clarity as we progress in our development to the algorithm and application domains.

Domains:

D_i : $(\mathcal{R}, \mathcal{S})$

$\mathcal{R}$: set of m elements r

$\mathcal{S}$: set of N sets with costs c that may cover $\mathcal{R}$

set cover : $\bigcup_{i=1}^{k} S_{j_i} = \mathcal{R}$ [5, Eq. 3.12]

– union of sets cover $\mathcal{R}$ elements

D_o : set of set-covers $\mathcal{S}'$ of $\mathcal{R}$ with additive cost

3) $0/1$ Problem Description: An integer linear program is a special case of integer programming "in which unknowns are binary, and only the restrictions must be satisfied" [31]. According to [5], the set cover problem "can be formulated as a $[0/1]$ linear program" as follows:

$$\text{minimize} \sum_{j=1}^{N} c_j \xi_j$$

$$\text{subject to} \sum_{j=1}^{N} t_{ij} \xi_j \geqslant 1 \qquad \forall r_i \in \mathcal{R}$$

where $c_j \geqslant 0$, $\xi_j \in \{0, 1\}$, if $S_j \notin \mathcal{S}'$ or $S_j \in \mathcal{S}'$ respectively, and $t_{ij} \in \{0, 1\}$, if $r_i \notin S_j$ or $r_i \in S_j$ respectively [5, p. 39-40].

This is different from the set cover problem defined above in that we encode, using binary representation, whether every set is in the set cover or not. We assign a 1 if the set is in the set cover, and a 0 otherwise. This provides a $\log n$-approximation algorithm for the minimum set cover problem [33].

4) Problem Domain Complexity: In 1972, Karp, et al. proved that the set cover problem is NP-Complete [10], while the optimization version is NP-Hard [33]. According to [33], [18], the complexity of the set cover problem is the powerset of $\mathcal{S}$, that is $\mathcal{O}(2^{|\mathcal{S}|}) \rightarrow \mathcal{O}(2^N)$, for both the search and solution spaces. This is due to the fact that a solution is found, if it exists, only by searching the space of all possible combinations of $\mathcal{S}$.

5) Problem Domain Specification: Using the symbolic notation developed in Section II-A2 we define the input and output domains and conditions for the set cover problem domain.

D_i : $(\mathcal{R}, \mathcal{S})$

$\mathcal{R}$: set of m elements r

$\mathcal{S}$: set of N sets with costs c that may cover $\mathcal{R}$

set cover : $\bigcup_{i=1}^{k} S_{j_i} = \mathcal{R}$ [5, Eq. 3.12]

– union of sets cover $\mathcal{R}$ elements

D_o : set of set-covers $\mathcal{S}'$ of $\mathcal{R}$ with additive cost

Solution:

Minimize additive cover costs subject to following constraints:

- set cover:
  $\sum_{j=1}^{N} t_{ij} \xi_j \geqslant 1, \quad i = 1, 2, \ldots, M$ [5, Eq. 3.14]
  cover all elements at least once

B. Application Problem Domain

1) Application Problem Domain Discussion: The UAS problem deals with assigning UAS orbits to cover a list of targets. Given a limited number of UAVs, each assigned to specific units based in specific locations – each with specific ranges of distance able to travel and operating costs – and a list of targets to surveil, how do we assign surveillance orbits to the UASs such that all targets are surveilled while keeping the UASs close enough to their base of operations that they may safely return, either when their mission finishes or they've exhausted their fuel, at minimum cost? The UAS problem relates to the set cover problem in that we are trying to cover a set of targets with a set of UAVs based on their locations, all while keeping them within a specified distance of their home stations and minimizing the associated operating costs.

Formally, there exists a universe of targets that we want to surveil $\mathcal{T} = \{t_1, \ldots, t_m\}$, each with a specific hideout location t_{i_h}, a set of UASs $\mathcal{U} = \{u_1, \ldots, u_n\}$, each with an assigned squadron and operating cost, a set of squadrons $\mathcal{F} = \{f_1, \ldots, f_q\}$ with specific geographic locations f_{j_ℓ}, and, finally, a set of schedules $\mathcal{A} = \{a_1, \ldots, a_r\}$ for each target which denote time windows in which the target is able to be surveilled. The objective is to surveil each target t during their time window a using a minimum number of UASs $u \in \mathcal{U}$ at minimum cost c such that each target is surveilled.

A review of the available literature reveals that, while much research has been performed on routing UASs and target coverage, none of the available sources provide insight into the constrained problem defined in this section. The most similar research can be found in [9], which uses a Max-Min Ant System algorithm, a member of the ant colony optimization algorithm family [30]. Other research with related, but not similar, goals to this project are discussed in [1], [3], [4], [22], [23], [25].

2) Application Problem Domain Complexity: Similar to the problem domain complexity described in Section II-A4, the complexity of the application domain is the powerset of each search space $\mathcal{O}(2^{|\mathcal{T}|} \times 2^{|\mathcal{U}|} \times 2^{|\mathcal{F}|} \times 2^{|\mathcal{A}|}) \rightarrow \mathcal{O}(2^{m+n+q+r})$, or every possible combination of targets, UASs, squadrons, and surveillance time windows.

We now prove that the application domain, the UAS problem, is NP-Complete. This can be shown using the technique described in [11] in which the constraints described in Section II-B1 are relaxed to facilitate a polynomial-time reduction [32] into the form of a known NP-Complete problem. This would allow a polynomial-time black-box solver for the known NP-Complete problem, if one existed, to solve the relaxed-constraint UAS problem. The UAS problem with relaxed constraints could then be said to be at least as hard as NP-Complete.

Relaxing the constraints of the UAS problem results in a problem as follows: given a universe, or set of targets, $\mathcal{R}$ a set of UASs $\mathcal{U}$, and a family of sets $\mathcal{S}$, containing UASs available to surveil specific targets. Assign to each $s \in \mathcal{S}$ an operating cost c. The goal is then to surveil each target in $\mathcal{R}$ at minimum cost. This transformation can be done in polynomial time.

The solution returned is easily verified by iterating through $\mathcal{R}$ and checking that there is in fact a UAS $\in \mathcal{U}$ assigned to surveil that target. The complexity of the verification is $\mathcal{O}(|\mathcal{R}| \times |\mathcal{U}|)$

Therefore, since we have shown that, in polynomial-time, the UAS problem can be transformed to the set cover problem (a known NP-Complete problem), $UAS \leqslant_p Set\ Cover$, and we can verify in polynomial-time the solution, the UAS problem is NP-Complete.

III. Algorithm and Problem Domain Initial Integrations

A. Selection of Deterministic Algorithm

The deterministic algorithm used for this project is the A* search algorithm. The heuristic function is defined the same way the classic greedy approach to this problem is as per [33], with the exception that the heuristic is inverted, as done in [8], due to the fact that A* chooses the next search node which has the $h(n)$ closest to 0. The heuristic function developed for this project is defined as: $h(n) = -\frac{\text{\# of targets surveilled}}{\text{cost}}$. The function for deciding which node to visit next is:

$$\text{minimize } f(n) = g(n) + h(n),$$

where $g(n) = g(n') + c(n', n)$.

The following sections document the disciplined design approach found in [18], [20], [17] – starting from the point of choosing data structures (since the problem domain has already been covered in Sections II-A and II-B) and finishing with a coded implementation (see Section III-C).

1) A Algorithm Domain Discussion:* The A* algorithm is a best first search algorithm [15], [21] which is "widely used in pathfinding and graph traversal" [27] which makes it ideal for our problem. The star "refers to always finding the optimal solution" [15]; it can be modified to find the optim*um* solution if termination is delayed. While it "is generally outperformed by algorithms which can pre-process the graph to attain better performance" [27], it is still suitable for our purposes, especially when using an admissable heuristic to assist the search process [21] as is done in this paper, and is easily adaptable to work with set cover problem instantiations.

What differentiates A* from other greedy best first search algorithms is that it not only takes into account the distance to the goal (the $h(n)$ part of the algorithm), but it also takes the distance already traveled from the start node to the current node as well in the "$g(n)$ part of the heuristic" [15].

2) Data Structures and Operations: For the A* search algorithm, we use the following data structures:

Solution : a structure that holds the current partial solution
Target : a structure that represents a target to surveil, with a geographic location
UAS : a structure that represents an available Unmanned Aircraft System with an assigned squadron and operating cost
Squadron : a squadron that "owns" one or more UASs with a geographic location
Schedule : a structure that represents one or more time windows, per target, for which the target is "available" to be surveilled

Operations on the A* algorithm include the following, adapted from [14]:

next-state-generator : the set of nodes explored (those in *OPEN*) and the set of nodes expanded (those in *CLOSED*)
set of candidates : nodes in the frontier ((OPEN) list) sorted by $h(n) = -\frac{\text{\# of targets surveilled}}{\text{cost}}$
selection function : choose node n' with the smallest $f(n')$ and place it on the *CLOSED* list. Place its descendents on the *OPEN* list
feasibility function : all nodes are feasible because of the tableau construction [5]
solution function : a node is a solution if the sets it contains cover the set of targets

This concludes the data structures and operations discussion of the A* algorithm. In the next section, we refine the data structures and algorithm towards a complete, *creative !* solution.

3) Refinement: Refining the A* algorithm towards a complete solution is the topic of this section. We now refine the data structures using abstract data types and provide more insight into the final form.

Data Structure Refinement:

Solution : a structure that holds the current partial solution
Target : a structure that represents a target to surveil, with a geographic location
UAS : a structure that represents an available Unmanned Aircraft System with an assigned squadron and operating cost
Squadron : a squadron that "owns" one or more UASs with a geographic location
Schedule : a structure that represents one or more time windows, per target, for which the target is "available" to be surveilled
OPEN : a priority queue which holds the set of candidates
CLOSED : a set containing nodes which have already been expanded

4) Pseudocode: This section presents pseudocode for the deterministic A* search algorithm describe in the preceding sections. The pseudocode can be seen in Algorithm 1 and is based on the A* algorithm detailed in [21]. It has been modified to delay termination as to find a complete solution

and to include the heuristic detailed in Section III-A. It is adapted from the form found in [8].

Algorithm 1 A* Search Algorithm from [21] modified with a heuristic and delayed termination to solve SCP/SPP

```
    /*                    Initialization                    */
 1: generate tree
 2: OPEN : priority queue to hold unvisited nodes
 3: CLOSED : set to hold visited nodes
 4: n : a set of sets with properties id, cost, and vector of
    children

 5: Step 1: Put the start node s on OPEN
 6: while true do
    /*                    Step 2                            */
    /*                 Solution Function                    */
    /*                Delayed termination                   */
 7:     if all elements covered && OPEN is empty then
 8:         return solution obtained by tracing back the
                pointers from n to s
 9:     end if
    /*                    Step 3                            */
    /*                Selection Function                    */
10:     Remove from OPEN and place on CLOSED a set
        for which f(n) = h(n) + c(n, n') is minimum
    /*                    Step 4                            */
    /*            Next-State-Generator Function             */
11:     Otherwise expand n, generating all its successors, and
        attach to them pointers back to n.
    /*                    Step 5                            */
    /*                Feasibility Function                  */
12:     for each successor n' of n do
    /*                Objective Function                    */
13:         if n' is not already on OPEN or CLOSED then
14:             Estimate h(n) = -(# of elements in the set / cost)
15:             Calculate f(n') = h(n') + c(n, n')
16:         else if n' is already on OPEN or CLOSED then
17:             Direct its pointers along the path yielding the
                lowest h(n')
18:         end if
19:         if n' required pointer adjustment && was on
              CLOSED then
20:             Put n' on OPEN
21:         end if
22:     end for
23: end while
```

B. Selection of Stochastic Algorithm

Simulated annealing is the stochastic algorithm used in this project. A brief introduction and review of simulated annealing follows in Section III-B1, with the remaining subsections in this section devoted to documenting the disciplined, top-down design approach found in [17], [20], [18] – starting with data structure choice (as the problem domain has already been discussed in Sections II-A and II-B) and finishing with a coded implementation (see Section III-C).

1) Simulated Annealing Algorithm Domain Discussion: Simulated annealing, according to [34], "is a probablistic technique for approximating the global optimum of a given function [and a] metaheuristic to approximate global optimization in a large search space." The algorithm takes its cue from a process known as annealing in metallurgy [19], modeling the cooling of metals as the atoms of superheated metals gradually progress towards an equilibrium as the metal cools [26], [29]. Consequently, the key to the simulated annealing algorithm is the temperature, which is controlled via a cooling schedule. The cooling schedule is responsible for allowing the search to escape from local optimals and gradually decreases the temperature to slow the exploration of the search space to the global optimum [19], [26]. A generic simulated annealing model with standard search constructs is presented in Algorithm 2.

Algorithm 2 Generic Simulated Annealing Algorithm Model with Standard Search Constructs taken from [26]

Input: Cooling schedule.

```
    /*                Step 1: Initialization                */
 1: s = s_0;          // Generation of the initial solution
 2: T = T_max;                          // Starting temperature
 3: repeat
 4:     repeat                          // At a fixed temperature
    /*        Step 2: Next-State-Generator Function         */
 5:         Generate a random neighbor s';
    /*              Step 3: Feasibility Function            */
 6:         ΔE = f(s') − f(s) ;
    /*                Step 4: Solution Function             */
 7:         if ΔE ⩽ 0 then
 8:             s = s'          // Accept the neighbor solution
 9:         else
10:             Accept s' with a probability e^(−ΔE/T);
11:         end if
12:     until Equilibrium condition
    /*   e.g. a given # of iterations executed at each T    */
    /*                Step 5: Selection Function            */
13:     T = g(T);                      // Temperature update
    /*                Delayed Termination                   */
14: until Stopping criteria satisfied          // e.g. T < T_min
```

Output: Best solution found.

A difficulty encountered when implementing the simulated annealing algorithm is the concept of neighborhood selection. Considered an art-form by some [16], selecting the neighborhood provides the local search aspect of the algorithm while the cooling schedule allows it to search globally as well, leading to the global optim*um* [34].

2) Data Structures and Operations: For the simulated annealing algorithm we use the following data structures:

Solution : a structure that holds the current partial solution

Target : a structure that represents a target to surveil, with a geographic location
UAS : a structure that represents an available Unmanned Aircraft System with an assigned squadron and operating cost
Squadron : a squadron that "owns" one or more UASs with a geographic location
Schedule : a structure that represents one or more time windows, per target, for which the target is "available" to be surveilled

Operations on the simulated annealing algorithm include the following, adapted from [12]:

next-state-generator : generation uses mutation operating on current individual according to some probability
set of candidates : a solution
selection function : select from current solution for next generation solution
feasibility function : check new population child for feasibility (done by construction)
solution function : population members are feasible solutions by construction
fitness function : reflects symbolic problem domain optimization goal

This concludes the data structures and operations discussion of the simulated annealing algorithm. Once again, in the next section we refine the data structures and algorithm towards a complete, *creative !* solution.

3) Simulated Annealing Algorithm Refinement: We refine the simulated annealing algorithm by specifying more specific data structures defined in Section III-B2. The data structure refinement follows:

Solution : a set that holds the current partial solution
Target : a structure that represents a target to surveil, with a geographic location
UAS : a structure that represents an available Unmanned Aircraft System with an assigned squadron and operating cost
Squadron : a squadron that "owns" one or more UASs with a geographic location
Schedule : a structure that represents one or more time windows, per target, for which the target is "available" to be surveilled
Temperature : an integer representation of the temperature used in the cooling schedule function

This refinement enables us to now provide a pseudocoded implementation, which follows in Section III-B4.

4) Pseudocode: The pseudocode implementation presented in Algorithm 3 is adapted from [12] to solve the UAS problem described in this project.

Algorithm 3 Simulated Annealing Algorithm

```
   /*                Step 1: Initialization                */
 1: initialize starting Temperature value
 2: generate feasible solution : place first available point in
    each neighborhood in Solution

 3: while true do
    /*                Step 2                                */
    /*            Solution Function                         */
    /*            Delayed termination                       */
 4:     if temp == 0 OR iterations > MAX_IT then
 5:         return solution
 6:     end if
    /*      Step 3: Next-State-Generator Function           */
 7:     mutate current solution by assigning a target to a UAS

    /*            Step 4: Selection Function                */
 8:
 9:     if value of item in Solution.pop() < current solution
    then
10:         replace current solution with Solution.pop()

    /*            Step 5: Feasibility Function              */
11:     else if temp(Solution.pop()) == true then
    /*            worse solution found                      */
12:         replace current solution with Solution.pop()
13:     end if
14: end while
```

C. Program Implementation

The A* Search and Simulated Annealing algorithm implementations, the input files used in the experiments detailed in Section IV, and the LaTeX files for this document can be seen in the GitHub repository[1] created for this project.

IV. Design of Experiments

This section reports the results from testing the designed algorithms on small and medium input data. The data used in these experiments was derived from unclassified Air Force UAS mission sets. The input files can be seen in the GitHub repository created for this project.

Experiments for this project were designed in accordance with the guidelines set forth in [2]. Each algorithm was run 100 times on each of the three input files. Times reported in Tables I and II are averaged over all 100 program executions.

An Apple MacBook Pro equipped with a 2.8 GHz Intel Core i7 processor with 16 GB of 1600 MHz DDR3 RAM running Mac OS X version 10.11.4 was used to run all experiments.

Tabular results from each experiment are presented in Section IV-A, Table I and Section IV-B, Table II.

A. A* Search Algorithm Results and Analysis

Results for the A* Search Algorithm are presented in Table I. These times are averages over 100 executions on three input files.

[1] https://github.com/chipvp/686Project

TABLE I
AVERAGE RUN TIMES OF 100 EXECUTIONS OF THE A* SEARCH ALGORITHM DESIGNED FOR THIS PROJECT USING AS INPUT THREE FILES DERIVED FROM UNCLASSIFIED UAS MISSION DATA.

Size	A*
Small	3.882
Medium	10.310
Large	43.654

B. Simulated Annealing Algorithm Results and Analysis

Results for the Simulated Annealing Algorithm are presented in Table II. These times are averages over 100 executions on three input files using three different cooling (temperature decay) schedules: SA 1, SA 2, and SA 3.

TABLE II
AVERAGE RUN TIMES OF 100 EXECUTIONS OF THE SIMULATED ANNEALING ALGORITHM DESIGNED FOR THIS PROJECT UTILIZING THREE DIFFERENT COOLING SCHEDULES USING AS INPUT THREE FILES DERIVED FROM UNCLASSIFIED UAS MISSION DATA.

Size	SA 1	SA 2	SA 3
Small	2.317	2.197	2.767
Medium	11.585	10.733	20.551
Large	33.905	36.418	37.264

V. CONCLUSIONS

The experience gained in performing this top-down program design relates very well to the educational goals outlined in [13]. Performing the work for this paper provided an understanding of the SCP problem domain, concept, classification and design integration. I have gained a better understanding of deterministic and stochastic algorithmic search techniques and theability to describe and use the A* deterministic and simulated annealing stochastic search techniques. This paper also provided an opportunity to use heuristic global and local search methods. Finally, through this paper and the assigned class homeworks, I gained an understanding and ability to refine general search algorithms to solve specific problem types, all while integrating proper data structures along with appropriate heuristics.

The experimental design results show that the chosen algorithm implementations solve the UAS problem presented. Further optimizations to improve the efficiency and effectiveness of the algorithms are left to future work, but remembering the NFL Theorem [6], [28], [7].

Overall, this project cemented many of the educational goals layed out in [13] and provided an opportunity to put the ideas and concepts of algorithm design and metaheuristics to practical use.

REFERENCES

[1] G. Avellar, G. Pereira, L. Pimenta, and P. Iscold. Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time. *Sensors*, 15(11):27783–27803, 2015.

[2] R. Barr, B. Golden, J. Kelly, W. Steward, and M. Resende. Guidelines for designing and reporting on computational experiments with heuristic methods. In *Proceedings of International Conference on Metaheuristics for Optimization, Kluwer Publishing*, pages 1–17, 2001.

[3] C. Caillouet. Energy efficient point coverage problem using UAVs.

[4] W. Carlyle, J. Royset, and R. Wood. Routing military aircraft with a constrained shortest-path algorithm. Technical report, DTIC Document, 2007.

[5] N. Christofides. *Graph Theory: An Algorithmic Approach*. Computer Science and Applied Mathematics. Academic Press, Incorporated, 1975.

[6] P. Collet and J.P. Rennard. Stochastic optimization algorithms. *arXiv preprint arXiv:0704.3780*, 2007.

[7] C. Igel. No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics. In Y. Borenstein and A. Moraglio, editors, *Theory and Principled Methods for the Design of Metaheuristics*, chapter 1, pages 1–23. Springer-Verlag, Copenhagen, Denmark, 2014.

[8] P. Jordan and C. Van Patten. CSCE 686: Homework 8. (Unpublished class homework). Air Force Institute of Technology, May 2016.

[9] M. Karakaya. UAV Route Planning For Maximum Target Coverage. *CoRR*, abs/1403.2906, 2014.

[10] Richard M. Karp. *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, chapter Reducibility among Combinatorial Problems, pages 85–103. Springer US, Boston, MA, 1972.

[11] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[12] G. Lamont. CSCE 686 - Advanced Algorithm Design: Simulated Annealing (SA) Global Search. 2003. rev b. (Unpublished class notes). Air Force Institute of Technology.

[13] G. Lamont. CSCE 686 - Advanced Algorithm Design: "A Course in Deterministic & Stochastic Heristic/Metaheuristic Optimization", Spring Quarter, 2016. 2016. (Unpublished class syllabus). Air Force Institute of Technology.

[14] G. Lamont. CSCE 686 - Advanced Algorithm Design: A* Example. 2016. (Unpublished class notes). Air Force Institute of Technology.

[15] G. Lamont. CSCE 686 - Advanced Algorithm Design: Best First Search with gs_bfs. 2016. (Unpublished class notes). Air Force Institute of Technology.

[16] G. Lamont. CSCE 686 - Advanced Algorithm Design: Homework 9. 2016. (Unpublished class notes). Air Force Institute of Technology.

[17] G. Lamont. CSCE 686 - Advanced Algorithm Design: MIS Global Depth-First-Search Back-Tracking Design. 2016. (Unpublished class notes). Air Force Institute of Technology.

[18] G. Lamont. CSCE 686 - Advanced Algorithm Design: SCP/SPP Global Depth-First-Search Back-Tracking Design. 2016. (Unpublished class notes). Air Force Institute of Technology.

[19] G. Lamont. CSCE 686 - Advanced Algorithm Design: Simulated Annealing. 2016. (Unpublished class slides). Air Force Institute of Technology.

[20] G. Lamont. CSCE 686 - Advanced Algorithm Design: Specific Disciplined Design Approach. 2016. (Unpublished class notes). Air Force Institute of Technology.

[21] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984.

[22] S. Quintero, F. Papi, D. Klein, L. Chisci, and J. Hespanha. Optimal UAV coordination for target tracking using dynamic programming. In *49th IEEE Conference on Decision and Control (CDC)*, pages 4541–4546, December 2010.

[23] E. Semsch, M. Jakob, D. Pavlíček, and M. Pěchouček. Autonomous UAV surveillance in complex urban environments. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE/WIC/ACM International Joint Conferences on*, volume 2, pages 82–85. IET, 2009.

[24] T. Stern. Set Cover Problem (Chapter 2.1, 12). (Unpublished class notes). Massachussetts Institute of Technology: Seminar in Theoretical Computer Science, February 2006.
[25] K. Sundar and S. Rathinam. Algorithms for Routing an Unmanned Aerial Vehicle in the presence of Refueling Depots. *CoRR*, abs/1304.0494, 2013.
[26] E. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.
[27] Wikipedia. A* search algorithm — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 14-May-2016].
[28] Wikipedia. No free lunch theorem — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 20-April-2016].
[29] Wikipedia. Annealing (metallurgy) — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 18-May-2016].
[30] Wikipedia. Ant colony optimization algorithms — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 2-May-2016].
[31] Wikipedia. Integer programming — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 2-May-2016].
[32] Wikipedia. Polynomial-time reduction — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 14-May-2016].
[33] Wikipedia. Set Cover Problem — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 25-April-2016].
[34] Wikipedia. Simulate annealing — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 18-May-2016].