A JAVA UNIVERSAL VEHICLE ROUTER IN SUPPORT OF ROUTING UNMANNED AERIAL VEHICLES

Robert W. Harder, 2nd Lieutenant, USAF

AFIT/GOR/ENS/00M-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

20000613 102

DTIC QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 2000

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
A JAVA UNIVERSAL VEHICLE ROUTER IN SUPPORT OF ROUTING UNMANNED AERIAL VEHICLES

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert W. Harder, 2Lt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2750 P Street
Wright-Patterson AFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/00M-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Mark O'Hair, Lt Col, USAF
UAV Battlelab — mark.ohair@eglin.af.mil
1003 Nomad Way, Suite 107 — Comm: (850) 882-5940 x208
Eglin AFB, FL 32542-6867 — DSN: 872-5940 x208

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Advisor: Maj Ray Hill, AFIT/ENS,
Comm: 937-255-6565 x4327, DSN: 785-6565 x4327
ray.hill@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Unmanned Aerial Vehicles (UAVs) help the military gather information in times of peace and war. During a mission, typically 100 sites or more, a UAV will frequently be re-tasked to visit a pop-up threat, leaving the operator to determine the best way to finish the day's list of sites after the re-tasking. I develop a prototype application to serve the needs of a specific customer, the 11th Reconnaissance Squadron, by helping them preplan missions and dynamically re-task UAVs. This prototype application is built on a reusable airframe router called the core AFIT Router, which can later be added to more sophisticated mapping and planning software for other customers. The core AFIT Router is built on a new architecture, defined and implemented in this research, which calls for tools that solve entire classes of problems. To support the UAV routing problem, I develop such an architecture for Vehicle Routing Problems (VRPs) and Traveling Salesman Problems (TSPs) and call it the Universal Vehicle Router (UVR). The UVR allows for many solving techniques to be plugged in, and two sample solvers are included, one a tour-building heuristic by Gary Kinney and the other an adaptive tabu search developed in this research.

14. SUBJECT TERMS
Vehicle Routing, Unmanned Aerial Vehicles, UAV, Java, architecture, tabu search

15. NUMBER OF PAGES
73

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

AFIT/GOR/ENS/00M-16

A JAVA UNIVERSAL VEHICLE ROUTER IN SUPPORT OF ROUTING UNMANNED AERIAL VEHICLES

THESIS

Presented to the Faculty of the Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Robert W. Harder, B.S.

2nd Lieutenant, USAF

March 2000

Approved for public release; distribution unlimited

AFIT/GOR/ENS/00M-16

A JAVA UNIVERSAL VEHICLE ROUTER IN SUPPORT OF ROUTING UNMANNED AERIAL VEHICLES

Robert W. Harder, B.S.

2nd Lieutenant, USAF

Approved:

	Date
Ray Hill, Major, USAF Advisor	29 Feb 00
Dr. James Moore Reader	29 Feb 00

Acknowledgements

Thank you to my loving wife Gabrielle for whom this 18 month stint at AFIT was her honeymoon. *I'll give you a real one, honey.* She encouraged me and cared for me while I tarried away. Thank you to my friends at Fairhaven Church who encouraged me in growing spiritually as well as academically. Thank you to those at AFIT who befriended me and shared my joys and trials. Thank you to Tony and Beth Snodgrass for the holiday meals. Thank you to Chris Cullenbine who helped debug my tabu search software and Shay Capehart who always helped me think through my philosophical questions.

Thank you to my thesis advisors, Maj. Ray Hill and Dr. James Moore, as well as my original advisor Dr. Glenn Bailey who retired on me. Thank you to my thesis sponsor Lt. Col. Mark O'Hair who encouraged us, funded us, and sent us to Europe. Thank you to my colleague Capt. Gary Kinney for showing me where to eat in Germany and to our contact at the CAOC Lt. Col. Raplh Park for showing us where to eat in Italy.

Table of Contents

Acknowledgements iii

Table of Contents iv

List of Figures vi

List of Tables vii

Abstract viii

Chapter One: Introduction 1
- Setting 1
- Purpose 2
- Existing Software 4
- Conclusion 6

Chapter Two: Literature Review 7
- Tabu Search 7
- Advances in Tabu Search 8
  - Reactive Tabu Search 9
  - Hashing Techniques 10
  - Elite List 11
  - Jump Search 11
- The Traveling Salesman Problem (TSP) 12
  - 1-TSP 14
  - Multiple-TSP 14
  - mTSP-Time Windows 16
- The Vehicle Routing Problem (VRP) 17
  - Simple VRP 17
  - VRP-Time Windows 18
  - Heterogeneous Vehicles VRP(TW) 19
  - Multiple Depot (Het)VRP(TW) 21
  - Other VRP Issues 21
- Summary 22

Chapter Three: An Architecture for Optimization Applications 23
- Introduction 23
- Proposed Architecture 24
- Instance of Architecture for VRPs and TSPs 25
  - Prototype Application 26
  - Core AFIT Router 31
  - Universal Vehicle Router 36
  - Solver Interface 39
  - Sample Adaptive Tabu Search Solver 40
  - General Tabu Search 41
- Conclusion 45

Chapter Four: Empirical Analysis 46
Introduction 46
Selecting a Tabu List Style in the UVR's Included Tabu Search 46
Setting the Parameters in the UVR's Included Tabu Search 47
The Universal Vehicle Router, Sample Tabu Search, and Solomon Data Sets 50
Conclusion 53

Chapter Five: Conclusion 54
Specific Contributions 54
Recommendations for Future Work 55

Bibliography 56

Additional References 60

List of Figures

Figure 1: Predator UAV in the air 2
Figure 2: AFIT Router layers 4
Figure 3: Sample screen shot of O'Rourke and Flood's software 5
Figure 4: Example of a forward insertion move 14
Figure 5: The situation facing an analyst 23
Figure 6: Architecture for the general operations research software 24
Figure 7: Architecture for Vehicle Routing and Traveling Salesman class of problems 26
Figure 8: Screenshot of main prototype AFIT Router panel (MacOS) 27
Figure 9: Screenshot of sites screen (Solaris/CDE) 28
Figure 10: Screenshot of vehicles and bases screen (Linux/KDE) 29
Figure 11: Solve dialog for prototype AFIT Router (MacOS) 29
Figure 12: Screenshot of multiple solutions screen (Windows) 30
Figure 13: Screenshot of single solution screen (Solaris/OpenWindows) 31
Figure 14: AFIT Router Core Kernel as a point of contact 32
Figure 15: Distance and heading geometry on a spherical triangle 34
Figure 16: Headwind and tailwind ground speed adjustment 36
Figure 17: Representation of a solution in the UVR 38
Figure 18: Steps for the adaptive tabu search 40
Figure 19: General cycle of a tabu search 42
Figure 20: General tabu search engine 43
Figure 21: Visual summary of parameter settings tests 49
Figure 22: Comparing UVR with tabu search and best known Solomon solutions 51

List of Tables

Table 1: Information tracked by the core AFIT Router kernel 33
Table 2: Information and control requested by the UVR of higher level software..... 37
Table 3: Comparing performances of two tabu list styles..... 47
Table 4: Parameter settings for adaptive tabu search..... 48
Table 5: Performance of UVR against best known solutions 51

AFIT/GOR/ENS/00M-16

Abstract

Unmanned Aerial Vehicles (UAVs) help the military gather information in times of peace and war. During a mission, typically 100 sites or more, a UAV will frequently be re-tasked to visit a pop-up threat, leaving the operator to determine the best way to finish the day's list of sites after the re-tasking. I develop a prototype application to serve the needs of a specific customer, the 11th Reconnaissance Squadron, by helping them preplan missions and dynamically re-task UAVs. This prototype application is built on a reusable airframe router called the core **AFIT Router**, which can later be added to more sophisticated mapping and planning software for other customers. The core AFIT Router is built on a new architecture, defined and implemented in this research, which calls for tools that solve entire classes of problems. To support the UAV routing problem, I develop such an architecture for Vehicle Routing Problems (VRPs) and Traveling Salesman Problems (TSPs) and call it the **Universal Vehicle Router** (UVR). The UVR allows for many solving techniques to be plugged in, and two **sample solvers** are included, one a tour-building heuristic by Gary Kinney and the other an adaptive tabu search developed in this research.

A JAVA UNIVERSAL VEHICLE ROUTER

IN SUPPORT OF ROUTING UNMANNED

AERIAL VEHICLES

Chapter One: Introduction

Setting

Unmanned Aerial Vehicles (UAVs) serve America's military forces by flying in dangerous areas in primarily surveillance missions. Because they are unmanned, UAVs have endurance times that far exceed any mission normally flown by manned aircraft. Long endurance times mean a UAV may visit many targets, or sites, during a mission. Due to myriad planning considerations, finding a good path among the sites that the UAV must visit is a daunting task. UAV operators have software to help plan a mission for each UAV, but the tools lack a way to suggest a sequence in which to visit these sites. A good sequence or route is one that dramatically reduces mission times and exposure to threats while satisfying the critical needs placed on the overall mission.

The Predator UAV, in use by both the US Air Force and US Army, can stay aloft for more than 24 hours while cruising at 70 knots at altitudes up to 25,000 feet. With a variety of intelligence-gathering payloads, the Predator transmits high resolution video and synthetic aperture radar images via a satellite link or line of sight communications. This information is useful for preplanning missions, executing missions, and assessing battle damage. Figure 1 shows a Predator in flight.

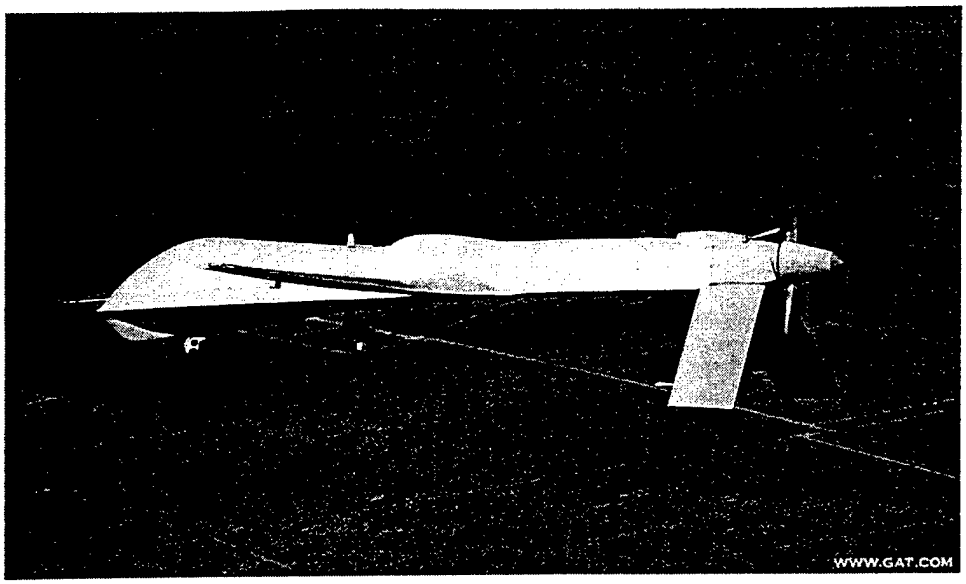


Figure 1: Predator UAV in the air

In wartime scenarios, UAVs may be responsible for imaging hundreds of sites every day. Although a list of sites to visit is available at the beginning of the day, past conflicts have shown that a hundred or more targets may interrupt that list throughout the day, requiring quick-turn re-tasking of intelligence assets. The daunting task of re-routing a UAV to accommodate these pop-up targets that interrupt the pre-determined target list falls squarely on the shoulders of the UAV operators. Existing mission planning tools provide little support in the re-routing efforts of the UAV operators.

Purpose

This thesis presents an architecture for and prototype application of a routing tool to help UAV operators reduce mission times in both the preplanning and real-time re-tasking phases. All code, to include the sample search algorithm within the router, was coded in Java

and designed to work with the existing mission planning tools in use by UAV operators. The algorithm extends O'Rourke's (1999) work and adds new UAV considerations—site priorities and restricted geographic operating zones—and new tabu search techniques, some found in the literature and some developed in this research.

The architecture and prototype application is built around a Universal Vehicle Router (UVR) specified and developed in this research. Figure 2 suggests how the various components in the architecture work together in layers to form the AFIT Router, used for the specific UAV routing problem. The UVR supervises the solving (generating and improving routes) by exploiting modern object oriented techniques to interface to add-in Vehicle Routing Problem (VRP) solvers. In general this approach looks not to include a specific set of features but instead seeks *not to prohibit* both common and obscure features. For example, in a general routing application, accounting for multiple operating locations, or depots, can get complicated with traditional approaches. However, within the VRP solver paradigm, each vehicle dictates its own starting position and so could have its own depot. An application, such as the UAV algorithm in this research, does not need to know about routing algorithms when using the UVR. It simply describes its components in terms of performance and capabilities.

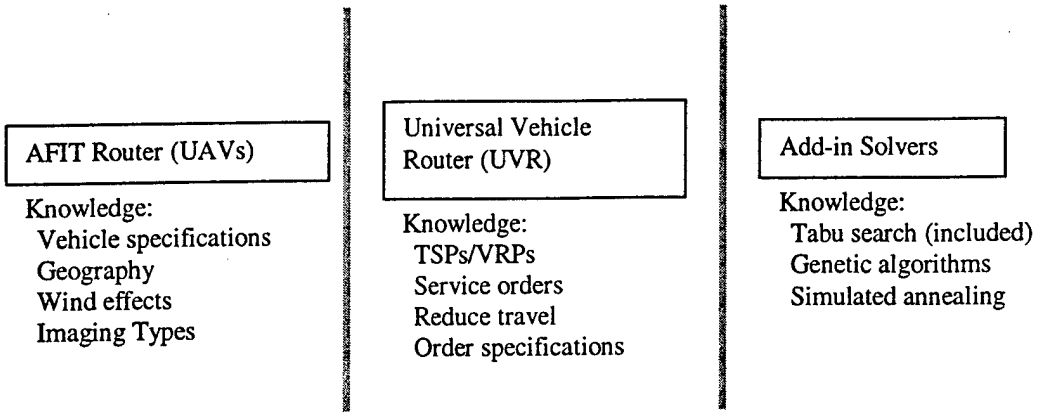


Figure 2: AFIT Router layers

This thesis effort is part of a larger effort sponsored by the UAV Battlelab directly supporting the 11th Reconnaissance Squadron. This research defined and built an architecture for routing problems and used the architecture to prototype the *AFIT Router*, a UAV routing tool. Kinney (2000) developed and empirically tested solution heuristics incorporated in the Universal Vehicle Router. External to these research efforts were efforts to interface the prototype AFIT Router application with the existing DEMPC software (see below) and train the 11th Reconnaissance Squadron personnel on the use of the AFIT Router.

Existing Software

UAV Battlelab-sponsored research by O'Rourke (1999) extended previous work (Ryan 1998 and Carlton 1995a) and included software (Flood 1999) that provided a graphical user interface (GUI) to tie maps and targets to routes. This software, called Auto-Router, uses a tabu search meta-heuristic to find good, possibly-optimal, routes for UAVs. Figure 3

shows a sample screenshot of this Auto-Router software. In Figure 3, triangles represent targets visited along the UAV route represented by the connected directed arcs. The single operating location (depot) is at the upper left corner represented by the 11th Reconnaissance Squadron patch.

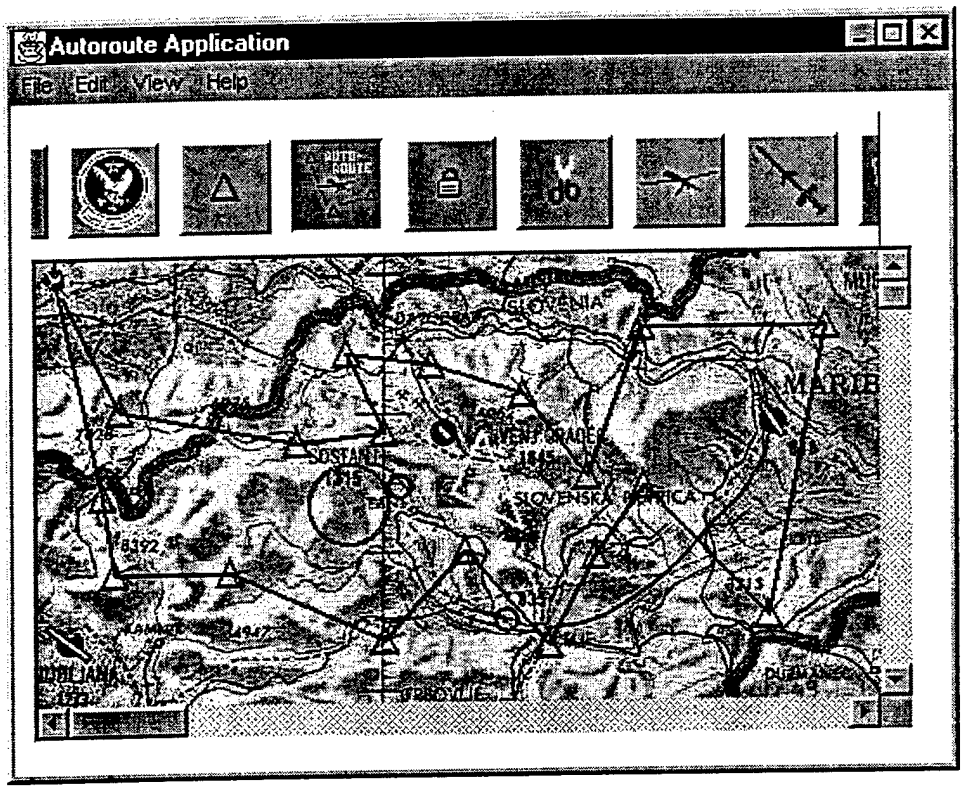


Figure 3: Sample screen shot of O'Rourke and Flood's software

This software was a good first step in combining routing software with a usable Graphical User Interface (GUI), but with a limited interface, the Auto-Router best served as a proof of concept for future work.

A mission planning tool called DEMPC (Data Exploitation and Mission Planning Console) was developed by Boeing for the Predator UAV Operators. DEMPC allows

operators to both preprogram a mission and follow an ongoing mission. The DEMPC has its own useful GUI. Since the DEMPC has this GUI, our prototype application does not need any map overlay features like O'Rourke and Flood's Auto-Router. The DEMPC has tools that help the operators with *how* the Predator is to fly: waypoint-to-waypoint, wings level, record imagery, stay in assigned air space, avoid threats, and avoid terrain. The DEMPC does not address *why* the Predator flies where it does. There is no auto- routing feature to help the operators know *why* they should fly a particular route.

When a UAV is re-tasked to visit a new site, the UAV operators must replan the mission manually. Sometimes after visiting a new site, returning to where they left the planned route is sufficient, but that detour could add enough extra travel time to adversely affect the rest of the mission. If the UAV operators could generate a new plan while on their way to a new site, they might not have to skip sites for fear of running out of fuel or missing time over target requirements (referred to as time windows).

Conclusion

This thesis presents the specification and design of a flexible architecture built around a Universal Vehicle Router (UVR) for solving Vehicle Routing Problems. The UVR was applied to the UAV routing problem in support of the 11th Reconnaissance Squadron and applied to the academic problems contained in the Solomon data sets for testing.

Chapter Two: Literature Review

The Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) are members of a general class of problems where one or more vehicles visit customers and may or may not deliver some product along the way. Often there are extra considerations such as visiting a particular customer within a certain time window or specifying that certain customers must be visited before or after other customers.

There are a variety of techniques to solve these problems but none have been as effective overall as tabu search (Laporte 1992). A survey of these problems, starting with the simplest Traveling Salesman Problem and working up to complex Vehicle Routing Problems, follows an overview of tabu search.

Tabu Search

Tabu search is a technique developed by Glover (1986) that intelligently searches the solution space of a complex problem. It works on the principle that the use of memory can help a search avoid getting trapped in local optima and can help it look more closely at optima that may be sensitive to small changes. Tabu search explores the solution space in small steps and typically halts arbitrarily after some number of iterations, typically 1,000 to 5,000. The tabu search process is defined by an operation called a *move*. A move is an operation on a solution that changes that solution. That changed solution is called the *neighbor* of the current solution since the changed solution is only one move away from the current solution. A move can be as simple as toggling a binary variable on or off or swapping

two items in a sequence. In each iteration the tabu search evaluates all, or some portion of, neighboring solutions. All such neighboring solutions comprise the *neighborhood* of the current solution. The tabu search picks the neighbor with the best objective function value, and that neighbor becomes the current solution for the next iteration.

As moves are executed, their particular attributes are registered on a *tabu list*, remaining there for a specified number of iterations, the *tabu tenure*. The tabu list size and the tabu tenure are closely related and vary based on problem specifics. While tabu, the move cannot be chosen unless the tabu restriction is overridden by some useful goal, called an *aspiration criterion*. One common aspiration criteria is achieving the best known solution. The basic purpose of the tabu list is to avoid revisiting previous solutions and force the search into unexplored areas of the search space. Altering the size of the tabu list helps control search features.

Common enhancements to tabu search include *intensification* and *diversification*. The tabu search uses these features to look more closely at a particular solution (intensification) or to move to a different part of the solution space (diversification). A long tabu list can force the search to visit new locations (diversification) while a short tabu list can force the search to spend more time around the current solution (intensification).

Advances in Tabu Search

Tabu search (TS) has been applied to more than just TSPs and VRPs, and for those problems where TS may not seem to be a first choice, it can be a meta-heuristic, guiding other search techniques. A discussion of several aspects of TS follows.

Reactive Tabu Search

There are several parameters used to specify a particular tabu search. For example: tabu list length and neighborhood size. Properly setting these parameters can dramatically improve tabu search performance. A tabu search that tunes these parameters as it searches the solution space is called a reactive (or adaptive) tabu search. Battiti (1996) points out a drawback to tuning these parameters prior to the search process: you need either a lot of prior experience and knowledge about the problem or you need to do a lot of trial and error to find the parameter settings. He further points out that a "crucial issue" in heuristics is finding the balance between a detailed search in a local optimum and diversifying the search to include other portions of the solution space. Indeed there may be no optimal parameter settings to satisfy this balance. Thus a self-tuning tabu search may be necessary to both explore and escape local optima.

Making the tabu list tenure (*i.e.,* how long a move remains tabu) reactive is a common and very effective way to increase the quality of a tabu search. Battiti (1996) defines a hamming distance as a way to measure how far away one solution is from another. By tying the tabu list tenure to this hamming distance, Battiti intensifies and diversifies the search as needed. Battiti suggests using the following function, where $T = 1$ (T is an attribute of the tabu list length) at the beginning and L is defined as the size of the neighborhood, to change the tabu list tenure (T-REACT) when a local optima is repeated.

$$\text{T-REACT}(T) = \min\{ \max\{ T \times 1.1, T + 1 \}, L - 2 \} \tag{1}$$

The upper bound of $L - 2$ ensures that the tabu search will always make a decision based on the objective function and never just default to a new solution. Similar schemes for

modifying the tabu list tenure have proved successful (Toulouse *et al.* 1998 and Talbi *et al.* 1998) including O'Rourke's (1999) tabu search in the Auto-Router software.

Hashing Techniques

Original tabu search methods determine tabu moves by some attribute of the move such as a binary variable *i* being toggled on or off. Woodruff and Zemel (1993) suggest using a hashing function to characterize the entire solution and place that characterization on the tabu list, thus making previously visited solutions tabu. This tabu list is less restrictive and thus needs a longer tenure to match the characteristics of a traditional tabu list. Woodruff and Zemel (1993) state three goals for a hashing function: computation should be easy, storage and comparison efforts should be reasonable, and probability of *collision* should be low. A collision occurs when two dissimilar solutions yield the same hash value and thus appear to be the same solution.

A hashing function generates an integer in some predefined range mapping a larger domain to a smaller domain. The size of this predefined range directly affects the probability of a collision, and smaller ranges have a higher chance of collisions (imagine trying to map basketball scores to a number between 1 and 10). It might appear that the probability of a collision is the inverse of the maximum range (1/MAXRANGE), but Carlton and Barnes (1996) point out that this is an overly optimistic estimate of collision rates. Recalling the birthday problem, they note that there is a better than 50-50 chance of two people in a room of 25 having the same birthday, not the 1/365 = .00274 chance one might expect. Extending this principle to the hashing problem, they show that with a two-byte number (up to 65,535) there is an 85% chance of a collision after just 500 iterations when all solutions are hashed and recorded.

Woodruff and Zemel (1993) present two hashing functions that use pre-computed vectors or matrices of random integers, z_i and where x_i is some value for element i in the solution. The first

$$h_1 = \sum_{i=1}^{n} z_i x_i \tag{2}$$

uses as many random integers as there are decision variables. The second creates an $n \times n$ matrix, Z, of random integers and a hash value based on

$$h_2 = \sum_{i=1}^{n} Z(x_i, x_{i+1}) \tag{3}$$

The second hash function provides a lower probability of collision at the expense of taking up more memory with its $n \times n$ matrix.

Elite List

As it continues its iterations, a tabu search may save good solutions that meet certain criteria This list of good solutions is called an *elite list*, and the tabu search may go back to these solutions and perform a more intense search in that part of the solution space. Although the tabu search may spend a lot of time building the initial elite list, the focused intensification it later performs often outweighs the effort expended.

Jump Search

Many good tabu searches build an elite list of good solutions and then revisit these solutions with various intensification or path-relinking techniques. Tsubakitani and Evans (1998a) coined a technique called *jump search* where solution-construction heuristics

effectively give the tabu search an elite list at the first iteration. Having an elite list from the start saves valuable tabu search iterations. The Universal Vehicle Router in this thesis uses jump search solution-construction techniques developed by Kinney (2000).

The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP) presents a case where a number of customers must be visited exactly once by one or more salesmen or vehicles. The simplest TSP with one vehicle can be represented as follows. Let nc be the number of customers. Let x_{ij} equal 1 if customer j is immediately preceded by i and zero otherwise. Let c_{ij} be the cost of traveling from customer i to customer j. Let t_{ij} be the time it takes to travel from i to j. Let s_j be the service time at j. Let R be the limiting range in time of the vehicle. Although the limiting range R is not technically included in the simplest possible TSP, it is included here, skipping the more trivial case when the vehicle's route could go on forever..

$$\min Z = \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij} c_{ij} \tag{4}$$

{Minimize total cost—usually travel time}

subject to

$$\sum_{i=0}^{nc} x_{ij} = 1 \quad \forall j = 1..nc \tag{5}$$

{Arrive at each customer once}

$$\sum_{j=1}^{nc} x_{ij} = 1 \quad \forall i = 0..nc \tag{6}$$

{Leave each customer once}

$$\sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij} t_{ij} + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij} s_j \le R \tag{7}$$

{Range of vehicle}

$$x_{ij} \in \{0,1\}$$

$$i = 0..nc, \quad j = 1..nc \tag{8}$$

{Domain}

$$\sum_{i=1}^{nc}\sum_{\substack{j=1 \\ j \ne i}}^{nc} x_{ij} \le nc - 1 \tag{9}$$

{Subtour-breaking constraints}

Calrton (1995) presents a very effective method for the TSP that avoids some of the difficulties associated with the previous notation. It involves listing all vehicles and customers in a vector where solution states can be changed by removing an item and inserting it before or after its current position. Figure 4 shows an example of what the vector might look like as well as some possible moves for customer number two. The move shown is called a forward insertion move.

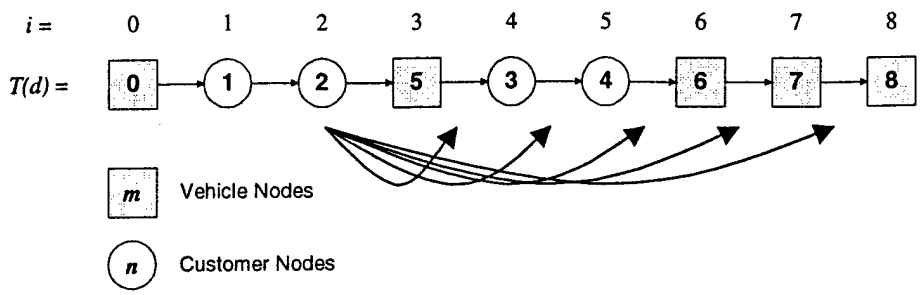


Figure 4: Example of a forward insertion move

1-TSP

The simplest case of a TSP involves one vehicle that must visit each customer. Because this is the simplest case, it is a good test platform for experimental techniques that may prove useful in more complicated problems. Since tabu list tenure greatly influences the success of a TSP solved by tabu search, Tsubakitani and Evans (1998b) use a 1-TSP with a symmetric cost matrix to find a good tabu list tenure based on the number of customers in the problem. They suggest a tenure between $nc/16$ and $nc/4$ depending on the complexity of the moves used in the tabu search. Gendreau *et al.* (1992) develop an insertion and post-optimization technique called GENIUS (Generalized Insertion and Unstringing/Stringing) that can help heuristics solve TSPs. The insertion technique (GENI) can be used in more complicated problems. The post-optimization technique (US) can be modified for more complex problems.

Multiple-TSP

With Multiple-TSP (mTSP) problems, there is more than one vehicle capable of visiting the customers. This requires a slight modification to the decision variable x. Let v

index nv vehicles and let x_{ij}^{v} equal one if vehicle v goes from i to j and zero otherwise. The objective is

$$\min \; Z = \sum_{v=1}^{nv} \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} c_{ij} \tag{10}$$

{Minimize total cost}

and the rest of (5) through (9) becomes

$$\sum_{v=1}^{nv} \sum_{i=0}^{nc} x_{ij}^{v} = 1 \quad \forall j = 1..nc \tag{11}$$

{Arrive at each customer once}

$$\sum_{v=1}^{nv} \sum_{j=1}^{nc} x_{ij}^{v} = 1 \quad \forall i = 0..nc \tag{12}$$

{Leave each customer once}

$$\sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} t_{ij} + \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} s_{j} \le R \quad \forall v = 1..nv \tag{13}$$

{Range of vehicle}

$$x_{ij}^{v} \in \{0,1\}$$

$$i = 0..nc, \quad j = 1..nc, \quad v = 1..nv \tag{14}$$

{Domain}

$$\sum_{i=1}^{nc} \sum_{\substack{j=1 \\ j \ne i}}^{nc} x_{ij}^{v} \le nc - 1 \quad \forall v = 1..nv \tag{15}$$

{Subtour-breaking constraints}

mTSP-Time Windows

The mTSP with Time Windows (mTSPTW) adds the restriction that customers must be visited within a certain time frame. Generally, the vehicle may arrive early but must then wait until the beginning of the time window. Let e_i be the earliest arrival time, l_i be the latest arrival time, and s_i be the service time for customer i. Let t_{ij} be the travel time between customers i and j. Let A_i and T_i be the time a vehicle arrives at customer i and the time it begins servicing customer i, respectively. Let W_i be the time spent waiting for service to begin at customer i. To account for time windows, the following two equations are appended after (15) above.

$$if\ x_{ij}^{v} = 1\ then\ T_i + s_i + t_{ij} + W_j \le T_j \tag{16}$$

{Time precedence}

$$e_i \le T_i \le l_i \quad \forall i = 1..nc \tag{17}$$

{Time windows}

and the vehicle range constraint (13) is rewritten as

$$\sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} t_{ij} + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} s_j + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} W_j \le R \quad \forall v = 1..nv \tag{18}$$

{Range of vehicle}

The Vehicle Routing Problem (VRP)

The vehicle routing problem adds vehicle capacities and customer demands to the traveling salesman problem. The literature is rich on VRPs since VRPs closely resemble the majority of problems in the real world. A progression from simple to complex VRPs follows.

Simple VRP

In a VRP, one or more vehicles deliver products to customers. Nearly all VRPs assume multiple vehicles, so the M is commonly dropped from MVRP. Just as the 1-TSP is useful for experimental TSP techniques, the single VRP is used to test new VRP techniques. Gendreau *et al.* (1994) develop a heuristic called TABUROUTE that works within a tabu search and allows infeasible solutions during the search. This heuristic performs well and outperforms many of the top performing heuristics. Gendreau *et al.* (1996) extend TABUROUTE to solve VRPs with stochastic demands and customers to test the robustness of a heuristic. Traditional VRPs assume only one kind of product or service is supplied by all vehicles.

To account for the vehicles' capacities to deliver products, let d_i be the demand of customer i and D be the capacity of each of the vehicles. This assumes the vehicles are identical, or homogeneous. Letting V^v be the set of customers visited by vehicle v. The objective is

$$\min \; Z = \sum_{v=1}^{nv} \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} c_{ij} \tag{19}$$

{Minimize total cost}

subject to

$$\sum_{v=1}^{nv}\sum_{i=0}^{nc} x_{ij}^{v} = 1 \quad \forall\, j = 1..nc \tag{20}$$

{Arrive at each customer once}

$$\sum_{v=1}^{nv}\sum_{j=1}^{nc} x_{ij}^{v} = 1 \quad \forall\, i = 0..nc \tag{21}$$

{Leave each customer once}

$$\sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} t_{ij} + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} s_{j} \le R \quad \forall\, v = 1..nv \tag{22}$$

{Range of vehicle}

$$x_{ij}^{v} \in \{0,1\}$$

$$i = 0..nc, \quad j = 1..nc, \quad v = 1..nv \tag{23}$$

{Domain}

$$\sum_{i=1}^{nc}\sum_{\substack{j=1 \\ j \ne i}}^{nc} x_{ij}^{v} \le nc - 1 \quad \forall\, v = 1..nv \tag{24}$$

{Subtour-breaking constraints}

$$\sum_{i \in V^{v}} d_{i} \le D \quad \forall\, v = 1..nv \tag{25}$$

{Demand and capacity}

VRP-Time Windows

Reflecting real-world concerns, many people require that some customers be visited within a certain time window. Desrochers *et al.* (1992) use a column generation technique to solve 100 customer problems to optimality while determining the ideal number of vehicles to

have in the fleet. Garcia *et al.* (1994) use a parallel tabu search to explore large regions of the solution space, resulting in good solutions very quickly. Formulating time windows in a VRP is similar to formulating time windows in a TSP. The following equations are appended after (25).

$$if\ x_{ij}^{v} = 1\ then\ T_i + s_i + t_{ij} + W_j \le T_j \tag{26}$$

{Time precedence}

$$e_i \le T_i \le l_i \quad \forall i = 1..nc \tag{27}$$

{Time windows}

and the vehicle range constraint (22) is rewritten as

$$\sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} t_{ij} + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} s_j + \sum_{i=0}^{nc}\sum_{j=1}^{nc} x_{ij}^{v} W_j \le R \quad \forall v = 1..nv \tag{28}$$

{Range of vehicle}

Heterogeneous Vehicles VRP(TW)

Heterogeneous vehicles do not share the same capacity or range, though they typically still carry only one type of service or product. Gendreau *et al.* (1999) adapt the GENIUS heuristics to account for different capacities and ranges and experiment with several parameters to suggest a robust set of values. In their tests on smaller problems (fewer than 75 customers), other techniques outperform GENIUS, but on larger problems, GENIUS produces the best results. To account for heterogeneous vehicles, many of the variables need to be indexed by the vehicle v: c_{ij} becomes c_{ij}^{v}, s_j becomes s_j^{v}, R becomes R^{v}, and D becomes D^{v}. The objective is:

$$\min \; Z = \sum_{v=1}^{nv} \sum_{i=1}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} c_{ij}^{v} \tag{29}$$

{Minimize total cost}

subject to

$$\sum_{v=1}^{nv} \sum_{i=0}^{nc} x_{ij}^{v} = 1 \quad \forall j = 1..nc \tag{30}$$

{Arrive at each customer once}

$$\sum_{v=1}^{nv} \sum_{j=1}^{nc} x_{ij}^{v} = 1 \quad \forall i = 0..nc \tag{31}$$

{Leave each customer once}

$$\sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} t_{ij}^{v} + \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} s_{j}^{v} + \sum_{i=0}^{nc} \sum_{j=1}^{nc} x_{ij}^{v} W_{j} \le R^{v} \quad \forall v = 1..nv \tag{32}$$

{Range of vehicles}

$$x_{ij}^{v} \in \{0,1\}$$

$$i = 0..nc, \quad j = 1..nc, \quad v = 1..nv \tag{33}$$

{Domain}

$$\sum_{i=1}^{nc} \sum_{\substack{j=1 \\ j \ne i}}^{nc} x_{ij}^{v} \le nc - 1 \quad \forall v = 1..nv \tag{34}$$

{Subtour-breaking constraints}

$$\sum_{i \in V^{v}} d_{i} \le D^{v} \quad \forall v = 1..nv \tag{35}$$

{Demand and capacity}

$$if \; x_{ij}^{v} = 1 \; then \; T_{i} + s_{j}^{v} + t_{ij}^{v} + W_{j} \le T_{j} \tag{36}$$

{Time precedence}

$$e_i \le T_i \le l_i \quad \forall i = 1..nc \tag{37}$$

{Time windows}

Multiple Depot (Het)VRP(TW)

Stationing vehicles at different depots requires a closer look at which vehicles are assigned to service which customers. Renaud *et al.* (1996) present a tabu search algorithm called FIND—Fast improvement, Intensification, and Diversification. The three phases in the FIND algorithm address the three key stages in tabu search. Published results suggest FIND outperforms existing algorithms on benchmark tests. An effort by Laporte *et al.* (1988) uses graph transformations with a modified branch and bound procedure to successfully solve multiple depot VRPs. One way to account for multiple depots extends the notion of heterogeneous vehicles. For example, the unique cost matrix c_{ij}^{v} would differ from another $c_{ij}^{v'}$ not only because of vehicle performance but because of depot location.

Other VRP Issues

VRPs can grow in complexity to reflect real-world concerns until the formulation becomes almost too unwieldy. Several efforts have expanded VRPs to include these real-world complexities. Golden *et al.* (1997) address the possibility that a vehicle may be reused during the day. Ryan (1998) adds the possibility of threat circles and no fly zones for Unmanned Aerial Vehicles (UAV). O'Rourke (1999) considers wind speeds and directions at different altitudes for UAVs. Laporte *et al.* (1988) determines where multiple depots should be located and assigns vehicles to customers.

Summary

Many people have solved the TSP, VRP and their variants. This chapter surveyed the TSP and VRP problems and their various extensions. These problem extensions have pushed problem complexity (and thus problem size) closer to real-world problem complexity. As these mathematical problems get more complicated, encompassing more of the real-world aspects, traditional solution techniques become harder to apply. In fact some formulations get to be so complex that being able to implement and represent them is as difficult a task as solving them. Thus, real-world problems require real-world structures, and object-oriented techniques can provide those real-world structures. An architecture for optimization applications designed and realized in this research offers such a real-world structure.

Chapter Three: An Architecture for Optimization Applications

Introduction

As problems grow in complexity, and the time available to obtain solutions diminishes, analysts need flexible tools that solve classes of problems (vehicle routing, assignment, scheduling) versus just techniques (linear programming, integer programming, heuristic libraries). Creating such flexible tools requires an understanding of the situations facing analysts. Figure 5 depicts the situation.

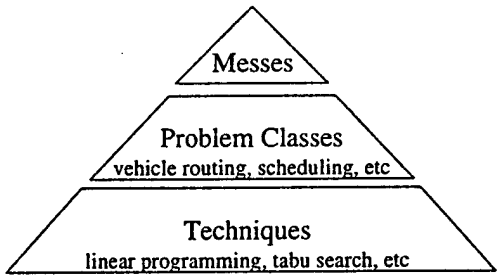


Figure 5: The situation facing an analyst

At the top level are the messes which managers and leaders must deal with regularly and for which models are built to provide insight (Ackoff 1979). An analyst examines and frames the mess as a particular class of problem, such as vehicle routing or scheduling problem. Once the problem is classified, a set of techniques are available. The analyst then chooses a technique best suited for the problem at hand.

One of two difficulties often arise once the technique is selected: 1) software implementing the technique must be written from scratch or 2) existing software requires analysts to translate their problem directly from some other solution formulation or data. The modern analyst needs to be able to plug into reusable, general purpose techniques. Reusable techniques mean analysts only develop those components particular to their problem. Such an architecture is available

Proposed Architecture

The analytic community needs an architecture that facilitates reuse rather than reinvention. Such an architecture allows one analyst to solve a problem class without recreating a technique, while another analyst could test a new technique on real, existing problems. Figure 6 proposes such an architecture.

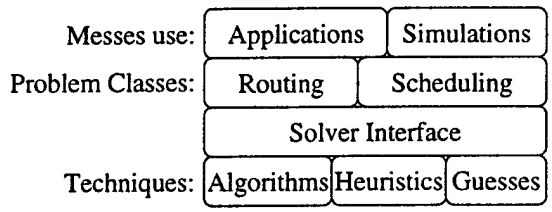


Figure 6: Architecture for the general operations research software

Developing software at each level requires the analyst to identify the level's common elements. For an example at the technique level, setting some variable x_i to 1 is not common to tabu searches, but evaluating a neighborhood and selecting an appropriate move is. For an example at the problem class level, not all traveling salesman problems (TSPs) use surface

roads to get from *A* to *B*, but all TSPs are concerned with some kind of cost for traveling from *A* to *B*. Modules developed for each layer communicate via defined interfaces. Thus an application in capital budgeting passes information to a multi-dimensional knapsack problem class model which in turn passes information to an available solver which may be some heuristic.

Instance of Architecture for VRPs and TSPs

For routing and rerouting Unmanned Aerial Vehicles, we need an instance of this architecture that supports the vehicle routing and traveling salesman class of problems. This research does not need a fully functional mapping application, only a prototype application. Fortunately the problem class layer does not care about the specifics of these applications. Although the windows and buttons may change with a more polished piece of software, the elements of UAV routing will remain the same: multiple starting locations, wind speed, "Great Circle" distances across the globe, etc. Thus the UAV routing software is broken into a prototype application and a core component which will be identified as *AFIT Router*. The AFIT Router is from the vehicle routing class and requires a layer boldly identified as *Universal Vehicle Router*. To complete this first instance of the architecture, an example tabu search solver is included. The software is coded in Java for flexibility in distribution. Figure 7 shows this specific instance of the general architecture. Each of the highlighted components are described next.

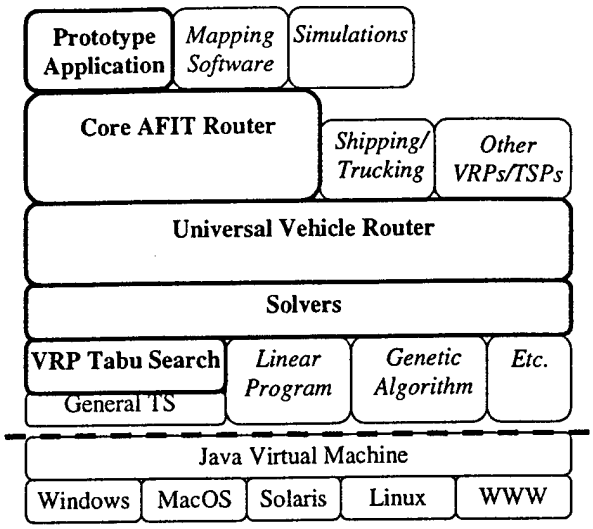


Figure 7: Architecture for Vehicle Routing and Traveling Salesman class of problems

Prototype Application

Acceptance of new software by any user community is sometimes difficult. In general, users want software that is easy to use and intuitive in application. They want the software to assist their efforts, not control them. The prototype AFIT Router software was designed for simplicity and practicality to encourage users to use it to solve their routing problems. The front panel (Figure 8) contains important summary information and allows quick access to more detailed information.

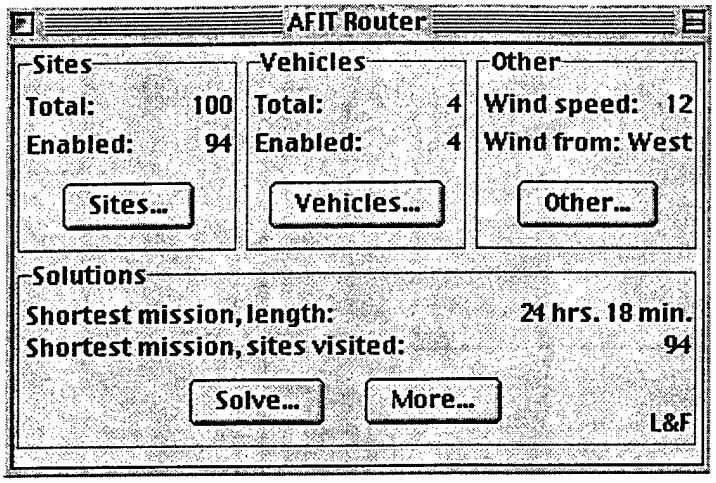


Figure 8: Screenshot of main prototype AFIT Router panel (MacOS)

Figure 9 shows a screenshot of the sites window that provides access to relevant information about the sites that the UAVs should visit. Sites may be loaded from files or copied and pasted from sources such as spreadsheets. Sites are designated by a name and a latitude and longitude. They have a service time (in minutes) that estimates the amount of time spent loitering at or around the site. The priorities allow the user to determine which sites are preferred over other sites if not all sites can be visited. At the user's discretion, this could mean that a priority one site is more important than any number of priority two sites or that a certain number of less important sites outweighs a single high priority site. The requirements field allows the option of matching a particular need at a site (laser designation, synthetic aperture radar, etc) to a vehicle with a matching capability. Having no requirement specified means that any vehicle may visit the site. The time window and time wall fields (earliest arrival, latest departure, earliest restricted, and latest restricted) allow the option of specifying when a site must or must not be visited.

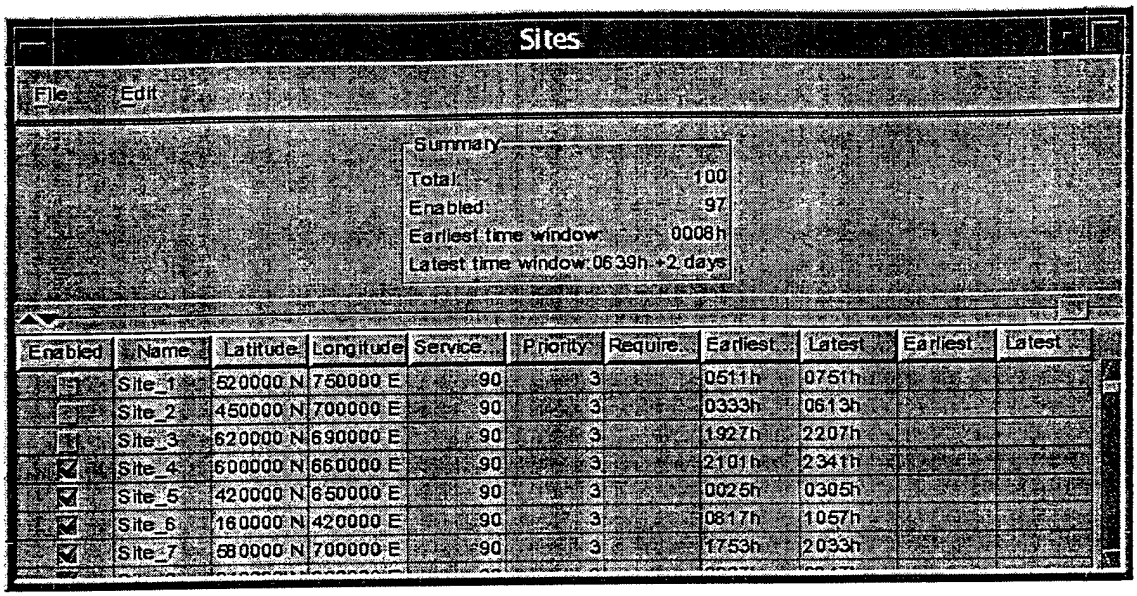
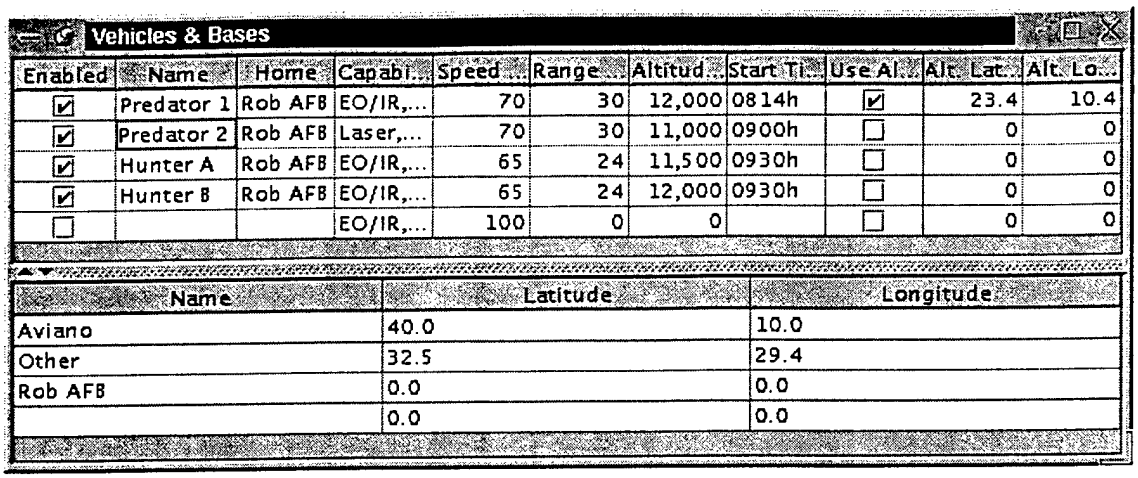


Figure 9: Screenshot of sites screen (Solaris/CDE)

Figure 10 shows a screenshot of the vehicles and bases window providing access to relevant UAV information. A base may be specified for a UAV. The UAV defaults to leaving from and returning to the latitude and longitude of that base. If the UAV is in the air, selecting *Use Alternate Location* treats the UAV as leaving from the alternate latitude and longitude and returning to the home base. The capabilities field allows the matching of a site's requirement to a specific vehicle or set of vehicles. All vehicles may visit sites with no specified requirement. The speed (knots), range (hours), and altitude (feet) are used in calculating travel and endurance times. The start time field specifies when the vehicle will be available for takeoff, in the case of preplanning a mission, or the current time, in the case of real-time re-tasking.



Vehicles & Bases

Enabled	Name	Home	Capabi...	Speed ...	Range ...	Altitud...	Start Ti...	Use Al...	Alt. Lat...	Alt. Lo...
☑	Predator 1	Rob AFB	EO/IR,...	70	30	12,000	0814h	☑	23.4	10.4
☑	Predator 2	Rob AFB	Laser,...	70	30	11,000	0900h	☐	0	0
☑	Hunter A	Rob AFB	EO/IR,...	65	24	11,500	0930h	☐	0	0
☑	Hunter B	Rob AFB	EO/IR,...	65	24	12,000	0930h	☐	0	0
☐			EO/IR,...	100	0	0		☐	0	0

Name	Latitude	Longitude
Aviano	40.0	10.0
Other	32.5	29.4
Rob AFB	0.0	0.0
	0.0	0.0

Figure 10: Screenshot of vehicles and bases screen (Linux/KDE)

Upon clicking the solve button on the main panel, the user is presented with some choices (Figure 11) regarding how to treat site priorities and how much time to spend solving the problem. The "Use post-optimization" checkbox below the solve time slider allows the user to request extra optimization at the expense of a longer solve time. Checking this box activates the sample tabu search discussed later in this chapter.

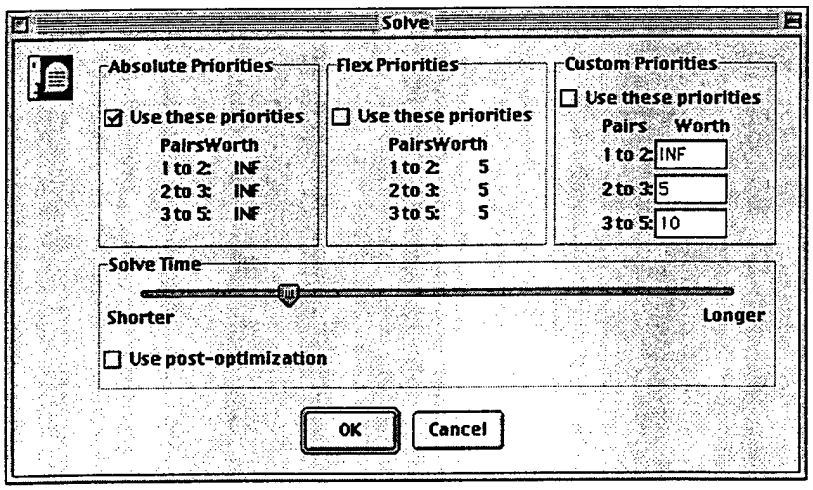


Figure 11: Solve dialog for prototype AFIT Router (MacOS)

Figure 12 and Figure 13 provide summary and detailed information, respectively, for solutions that have been found. A rough visual display of the solution is available as are details regarding estimated arrival and departure times at the sites. Solutions may be saved to disk, and sites may be selected and copied to the computer's clipboard directly or indirectly with the *copy to clipboard* button which is necessary in the case of running this software as a world wide web applet.

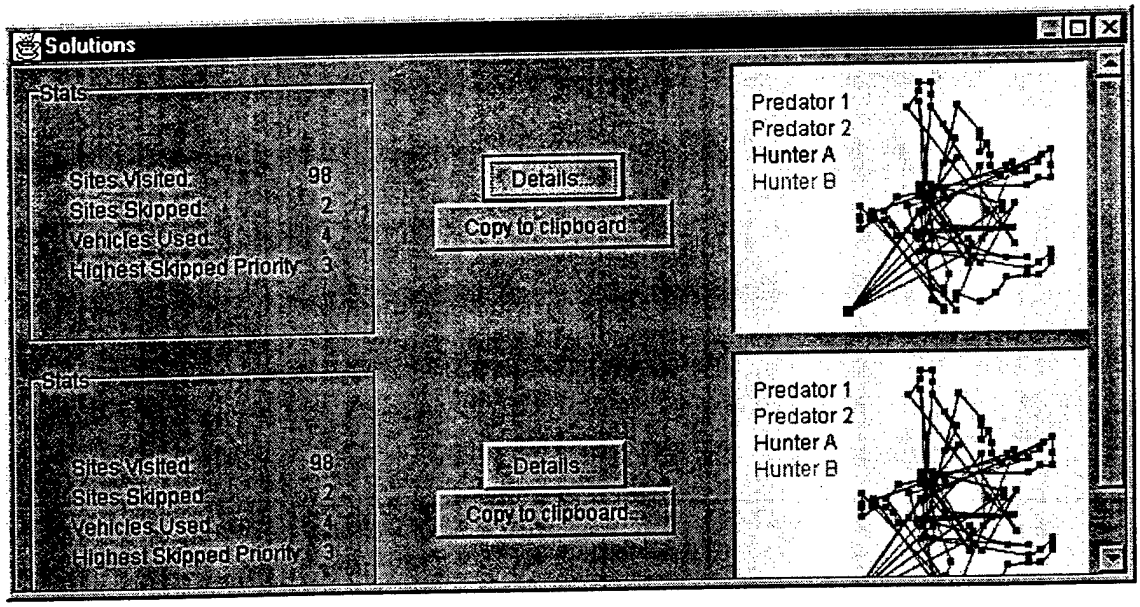


Figure 12: Screenshot of multiple solutions screen (Windows)

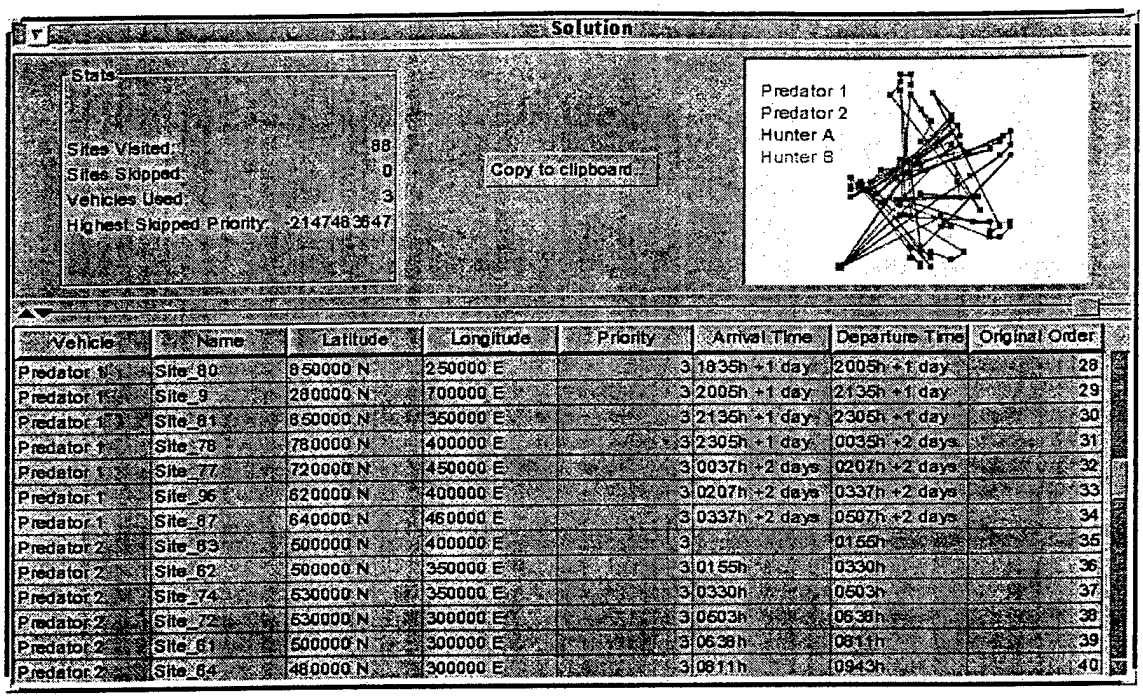


Figure 13: Screenshot of single solution screen (Solaris/OpenWindows)

This prototype AFIT Router application should be of immediate use to the UAV operators who must determine a route to sites they are tasked to visit. The site list is typically provided in spreadsheet form, and it can be copied and pasted into this application. This prototype is specific to the needs of the 11th Reconnaissance Squadron but serves as a presentation mechanism for how it could be used for other needs. The reader might also note the inherent portability of the architecture as evidenced by the previous screenshots taken from various computer platforms.

Core AFIT Router

The next layer of software is not concerned with how the data is presented to the user, but how the data about sites and vehicles is stored and manipulated. The core AFIT Router *kernel* (Figure 14) serves as a point of contact between the data structures and

application software. Not all features of the core AFIT Router kernel are used in the prototype AFIT Router application. Existing or future software that wishes to route UAVs (or other airframes) only need to know how to interact with the core AFIT Router kernel.

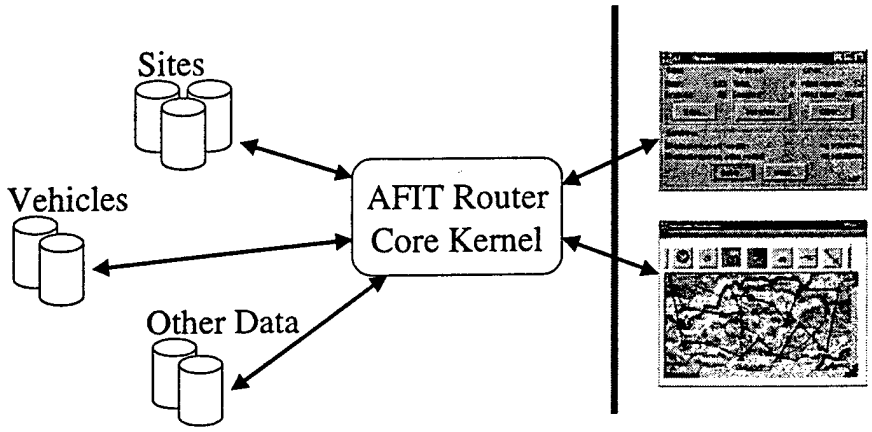


Figure 14: AFIT Router Core Kernel as a point of contact

The kernel makes available lists of vehicles, sites, winds, restricted operating zones, and solutions. Applications using the core AFIT Router kernel can listen for changes to these lists and reflect these changes by updating a table of summary information presented to the user. Table 1 shows the information tracked for the various components in the core AFIT Router.

Table 1: Information tracked by the core AFIT Router kernel

Component	*Information tracked*
Site	Name, latitude, longitude, priority, requirement, enabled status, service time, earliest arrival time, latest departure time, earliest restricted time, latest restricted time
Vehicle	Name, home base, capabilities, speed, range, altitude, enabled status, earliest starting time, at home status, alternate latitude, alternate longitude
Wind	Speed, bearing, lower altitude, upper altitude
Restricted Operating Zone	Name, earliest arrival time, latest departure time, earliest restricted time, latest restricted time, list of latitudes and longitudes defining its geographic region

Restricted operating zones are not used in the prototype AFIT Router application, but they aid in specifying time windows and time walls for entire geographic regions. Since the data is pertinent to more than one potential airframe routing application, the kernel maintains the data.

The core AFIT Router kernel also handles the difficult calculations for determining travel times between two points on a windy globe. The estimated time for traveling from one point to another is found by $time = {distance}/{rate}$. The distance calculation considers the great circle effect of traveling on a sphere. These calculations make the estimated times more accurate than a simple Pythagorean Theorem calculation. The following illustrations were taken from O'Rourke (1999). The calculations are from AFR 51-40, Air Navigation (Departments of the Air Force and Navy 1983).

The distance in nautical miles between two points is given by

$$d = 60 \cdot \cos^{-1}\left[\sin L_1 \cdot \sin L_2 + \cos L_1 \cdot \cos L_2 \cdot \cos(\lambda_2 - \lambda_1)\right] \tag{38}$$

where L_1 and L_2 are the starting and ending latitudes, respectively, and λ_1 and λ_2 are the starting and ending longitudes, respectively.

In order to account for the wind, a heading is needed. The intermediate angle H_{ij} in degrees clockwise from true north is given by

$$H_{ij} = \cos^{-1}\left[\frac{\sin L_2 - \sin L_1 \cdot \cos\left(\frac{d}{60}\right)}{\sin\left(\frac{d}{60}\right) \cdot \cos L_1}\right] \tag{39}$$

This intermediate heading is used to calculate the initial true heading Θ_{ij} also measured in degrees clockwise from true north and given by

$$\Theta_{ij} = \begin{cases} H_{ij}, & \sin(\lambda_2 - \lambda_1) < 0 \\ 360^\circ - H_{ij}, & \sin(\lambda_2 - \lambda_1) \geq 0 \end{cases} \tag{40}$$

Below is a graphical representation of this spherical triangle.

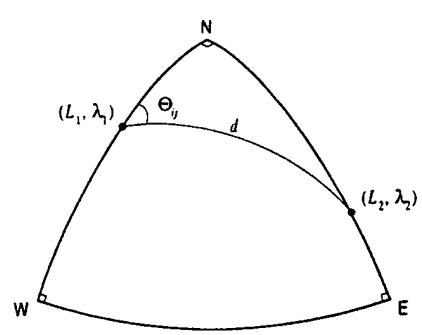


Figure 15: Distance and heading geometry on a spherical triangle

Arriving at a final ground speed GS requires several intermediate calculations. In the following calculations, WS represents wind speed in knots from a bearing of Θ_{WS} measured in degrees clockwise from true north, and δ is the difference between Θ_{ij} and Θ_{WS}. A, B, and C are intermediate values used to make the formulas easier to read. The ground speed GS is calculated by

$$\delta = \Theta_{ij} - \Theta_{WS} \tag{41}$$

$$A = WS \cdot \cos(180 - \delta) \tag{42}$$

$$C = WS \cdot \sin(180 - \delta) \tag{43}$$

$$B = \sqrt{AS^2 - C^2} \tag{44}$$

$$GS = A + B = WS \cdot \cos(180 - \delta) + \sqrt{AS^2 - WS^2 \cdot \sin^2(180 - \delta)} \tag{45}$$

and the final travel time is simply $t_{ij} = d_{ij} / GS$. Below is a graphical representation of these calculations.

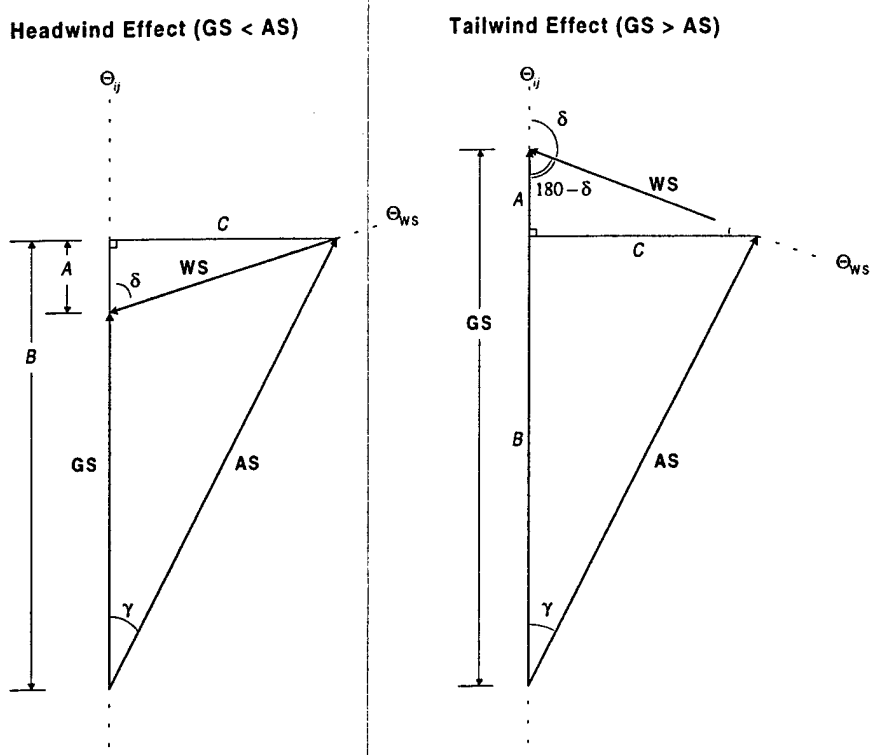


Figure 16: Headwind and tailwind ground speed adjustment

Universal Vehicle Router

The core AFIT Router kernel does nothing to route vehicles to sites but instead uses the Universal Vehicle Router (UVR). The UVR is so named because of its ability to solve a wide variety of VRPs and TSPs. It identifies the elements common to many VRPs and TSPs and defines a way for higher level software, like the core AFIT Router kernel, to interact with lower level software like a tabu search solver. If interested, the user can specify a solver from a suite of solvers.

From the higher level software, the UVR requests information about vehicles and *orders* (UVR terminology for sites or customers). From the lower level solvers it requests solutions for routing vehicles to orders. Table 2 shows the information and control

requested of higher level software using the UVR. Priority values are assumed to be in ascending order where lower values mean higher priority.

Table 2: Information and control requested by the UVR of higher level software

Component	*Information and control requested*
Order	Earliest arrival time, latest departure time, earliest restricted time, latest restricted time, priority, order type, amount needed
Vehicle	Range, earliest departure time, time to service order *A*, time to travel *A* to *B*, penalty to travel *A* to*B*, supports order type *C*, current amount available for order type *C*, remove product for order type *C*, replace product for order type *C*, reset products for all order types

The UVR stores solution information in a logical way that enables every instance of a solution to contain information about every vehicle and order including vehicles not used and orders not visited. Figure 17 visually represents how the data is stored in a solution. A dummy tour is used to store orders that are not visited. Each tour also has a data structure as depicted in Figure 17.

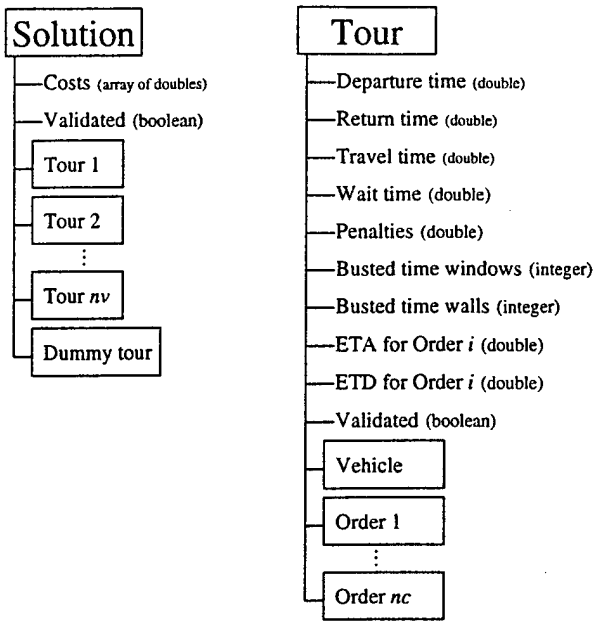


Figure 17: Representation of a solution in the UVR

The data stored in the tour is factual and can be calculated without assigning any importance to the values. The solution costs are different. They can change depending on how the user wishes to define a good solution. The UVR lets users define an *Evaluator* used by the solvers to determine solution quality. Evaluators may list any number of goals to minimize, and each goal is considered infinitely more important than the one before it. A typical evaluator might specify the number of exceeded vehicle ranges as the number one goal, the number of skipped orders as the number two goal, and the total travel time as the number three goal. A default evaluator minimizes the following: the number of exceeded vehicle ranges, the number of busted time windows, the number of skipped orders of priority HIGHEST_PRIORITY, the number of skipped orders of priority NEXT_HIGHEST_PRIORITY,..., the number of skipped orders of priority LOWEST_PRIORITY, travel time plus penalties, and wait time. Note that penalizing

certain legs of a route with values on a scale similar to the time units can discourage those legs. In the case of the core AFIT Router kernel, for example, legs that cross threat areas could be penalized more than safer legs.

Solver Interface

There are a variety of techniques to solve a VRP or TSP. The UVR provides an interface to allow different techniques to solve the same problem. Solvers receive a solution with all of the orders in the dummy tour (a placeholder). If they prefer, a list of good starting solutions may be generated using a tour-building heuristic. The solver can then access all the vehicle and order information discussed above, formulate the problem as it sees fit, and solve its problem. A solver is asked to scale its projected solve time according to an *effort* parameter (from zero to one) that is passed from the user through the UVR.

If a solver requests a list of good starting solutions, a tour-building heuristic written by Kinney (2000) generates a list of starting solutions and ranks them according to the values returned by the Evaluator. The heuristic is based on a Solomon insertion heuristic and varies several parameters to generate up to 176 unique starting solutions depending on the effort requested and the number of duplicate solutions generated. For a more detailed discussion on the nature of the heuristic and its proven performance, see Kinney's thesis *A Hybrid Jump Search and Tabu Search Metaheuristic for the Unmanned Aerial Vehicle (UAV) Routing Problem* (2000). The heuristic returns very good results and may itself act as a sufficient solver. One of the two sample solvers (the other being a tabu search) simply requests this list of starting solutions and immediately returns the best one. The UVR will use this solver if another is not specified. A user may use this solver to get a good answer very quickly.

Sample Adaptive Tabu Search Solver

To demonstrate the feasibility of adding solvers, a sample adaptive tabu search solver is included with the UVR. The tabu search requests starting solutions from the tour-building heuristic and searches this pseudo-Elite List according to a level of effort defined by the user. Each starting solution is evaluated for a minimum number of iterations. The search continues if improvements continue and moves to the next starting solution if the search stalls. The entire search stops when it runs out of starting solutions or has searched a certain number of consecutive starting solutions without generating a new global best solution. Figure 18 shows the steps taken by this tabu search solver.

Initialize:
Set n = number of orders
Set s = number of starting solutions
Set effort as requested by user, $e \in [0,1]$
Set minimum number of iterations per starting solution, $M = \max\{ 5, n * e / 2 \}$
Set extra iterations to give solutions, $E = \max\{5, 0.3 * m \}$
Set recency of last best solution required for extra iterations to be given,
  $R = \max\{ 5, 0.3 * m \}$
Set number of bad consecutive starting solutions before quitting,
  $B = \min\{ ns, \max\{ 3, s * e \}\}$
Set tabu tenure, $T = 3\ n$
Set current starting solution, $c = 1$
Set starting solution yielding last global best solution, $b = 0$

Steps:
1. Set iterations left to perform, $g = M$
2. Perform g iterations on starting solution c
3. If a solution better than starting solution c has been found within R iterations, set $g = E$ and go to step 2
4. If a new global best solution has been found set $b = c$.
5. If $c - b >= B$, quit.
6. Set $c = c + 1$
7. If $c > s$, quit.
8. Go to step 2

Figure 18: Steps for the adaptive tabu search

This tabu search defines four types of moves: relocate within tour, relocate to other tour, relocate to dummy tour, relocate from dummy tour. The first two move types insert orders a maximum number of places as defined by

$$\min\{\, n_t - 1, \max\{\, 5, 0.3 * n_t \,\} \,\} \tag{46}$$

where n_t is the number of orders in the tour at that iteration. These two move types are not generated at each iteration. They alternate such that one type is generated on odd-numbered iterations, and the other type is generated on even-numbered iterations. This reduces the size of the neighborhood at each iteration. The last two move types, moving orders in and out of the dummy tour, are generated at every iteration. Each order in a real tour is moved to the dummy tour, and each order in the dummy tour is moved to each location in each real tour. Since this tabu search builds on a list of good starting solutions, there are generally not many orders in the dummy tour.

This tabu search adapts to the supplied starting solutions provided by the UVR. More sophisticated tabu searches written for the UVR may also benefit by following this model as a way to exploit the available pseudo-Elite List.

General Tabu Search

The sample adaptive tabu search provided with the UVR is of course specific to vehicle routing but was built from a more general tabu search code. This general tabu search package continues the layered software approach by identifying the elements common to all tabu searches and providing a framework on which to build specific tabu searches. It has already appeared in three other research efforts including a weapons assignment model

(Cullenbine 2000), an abstract algebra approach to the Traveling Salesman Problem (Hall 2000), and a force allocation model (Calhoun 2000).

The design recognizes that although specific tabu searches have their own solution definitions, move types, and strategies, each one generally follows the pattern shown in Figure 19 where a given solution is altered and evaluated before a new current solution is chosen.

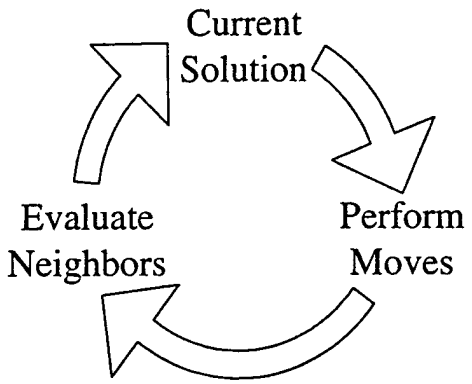


Figure 19: General cycle of a tabu search

The general tabu search package provides an engine that frees the analyst from writing the controlling code and allows the analyst to concentrate on defining the specifics of the search. Figure 20 suggests how an analyst can feed specifics into the engine and then listen for key events that may trigger specific strategies such as intensification, diversification, and strategic oscillation.

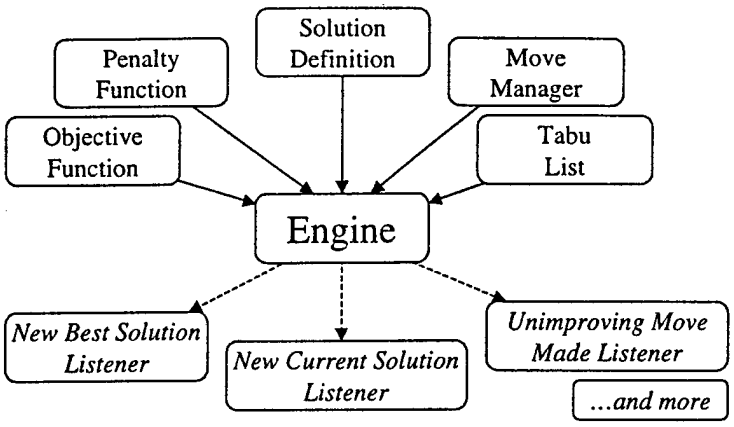


Figure 20: General tabu search engine

To create a new tabu search, an analyst would define each of the required objects and pass them to the Engine. The engine then performs the search as defined by the analyst's inputs.

The engine makes no assumptions about the structure of an analyst's solution. The analyst is free to define the solution variables in any manner. The engine only asks that the analyst's solution be able to duplicate itself when copies of a current solution are needed.

The objective function and penalty function evaluate a given solution however the analyst sees fit. The two values are added together. Thus one could evaluate the solution in one function and return zeroes in the other or logically divide the functions into contributions and penalties. The analyst may specify more than one number as the value for the solution in which case the list of values are compared lexicographically to determine which of two solutions is better.

The move manager determines at each iteration which moves (not shown in Figure 20) should be generated. These moves, each in turn, operate on the current solution. The new solution is evaluated, and the move undoes its operation. If the user specifies, the

engine will divide the moves up among available computer processors and evaluate the neighborhoods simultaneously. The best moves are then be executed for that iteration. Two moves may specify that their contributions to the total costs of the solution are independent in which case more than one move may be accepted at each iteration. If this is true the new solution costs are not reevaluated by the objective and penalty functions but instead are given by

$$\boldsymbol{Costs}_{new} = \boldsymbol{Costs}_{old} + (\boldsymbol{Costs}_{after\ move\ i} - \boldsymbol{Costs}_{old}) + (\boldsymbol{Costs}_{after\ move\ j} - \boldsymbol{Costs}_{old}) \qquad (47)$$

where the vectors $\boldsymbol{Costs}_{old}$, $\boldsymbol{Costs}_{after\ move\ i}$, and $\boldsymbol{Costs}_{after\ move\ j}$ have already been calculated.

When the engine considers a move, it checks with the analyst's tabu list. If the move is declared tabu, it is not executed unless it results in the best known solution. Moves selected at each iteration are registered with the tabu list.

The listeners provide a way for the analyst to perform certain actions when events of interest occur. Each time a new best solution is found or unimproving move is made, for example, an event is triggered which the analyst could intercept and use to begin intensification or diversification.

Specifications for interested developers are available in the software development kit available at the general tabu search package's web site http://www.crosswinds.net/~rharder/tabusearch/. For examples of others using this engine, see the theses by Cullenbine (2000), Hall (2000), and Calhoun (2000).

Conclusion

A general architecture for solving specific problems based on general components is proposed. This architecture is realized for routing problems and the AFIT Router exploits this architecture. The core component of the architecture, the Universal Vehicle Router (UVR), allows rapid development of routing problem solution tools with the existing suite of two solvers and the ability to plug in new solvers in the future.

Chapter Four: Empirical Analysis

Introduction

The development of the Universal Vehicle Router (UVR), its adaptive tabu search solver component, and the resulting AFIT Router software required testing to ensure that the software performed quickly and could solve the appropriate problems. Two types of tabu lists were considered and tested. Additionally, certain parameters in the adaptive tabu search have a critical effect on performance and were examined. The UVR is a general purpose tool so it must be able to handle the well-known problems in VRP and TSP research venues. The Solomon set of standard problems was used. To solve the Solomon problems a simple application was designed at the same level as the core AFIT Router. Thus, two examples of using the UVR to solve specific Vehicle Routing Problems already exist.

Selecting a Tabu List Style in the UVR's Included Tabu Search

Two tabu list styles were tried and tested. One tabu list had a fixed length and recorded a hash value of the solution. The other tabu list was reactive, increasing and decreasing tenure as a function of move quality.

Both tabu list styles were applied to all 56 of Solomon problems. The static tabu list based on the solution performed better more often and remained in the UVR's tabu search. Table 3 shows how each tabu list style performed, relative to each other.

Table 3: Comparing performances of two tabu list styles

Tabu List Style	*Number of Vehicles*		*Distance*		*CPU Seconds*	
	Count	*Avg Beating*	*Count*	*Avg Beating*	*Count*	*Avg Beating*
Order Attribute/ Reactive	0	0	4	10.65	16	5.1
Solution Hash/ static	0	0	8	8.46	36	10.2

The three values measured were the number of vehicles needed to visit all the customers, the total distance traveled by the vehicles, and the time in seconds that a computer[1] took to solve the problems. The *count* is the number of times that one tabu list beat the other tabu list. The *Avg Beating* is the average improvement seen by one tabu list over the other when an improvement existed.

Setting the Parameters in the UVR's Included Tabu Search

In designing and testing the adaptive tabu search in the UVR, certain parameters demonstrated a drastic effect on solution time. These parameters were

- the default number of iterations for each starting solution,
- the number of consecutive bad starting solutions to allow before quitting, and
- the maximum number of places to insert a site in a tour.

These three parameters were set at a low and high setting and run against each of the problems in the Solomon data sets. All runs were made at the maximum *effort* level with the

[1] The computer was a 266Mhz Pentium II running Windows 98 with 128Mb of RAM. The Java Virtual Machine used was Sun's prerelease of Java 2 v1.3.

objective of minimizing the number of vehicles used and the distance traveled. Table 4 shows the settings tested for each parameter.

Table 4: Parameter settings for adaptive tabu search

Parameter	*Low Setting (L)*	*High Setting (H)*
Default (minimum) iterations	¼ Number of customers	Number of customers
Bad consecutive starts before quitting	10% Number of starting solutions	50% Number of starting solutions
Maximum places to insert	10% Tour length	50% Tour length

The measured results were the time taken for the tabu search to solve the problem and the total distance traveled for the vehicles to visit all the customers. The visual summary of the results, shown in Figure 21, provides insight into the behavior of the parameters. The results are averaged across each of the six classes of Solomon problems. See Kinney (2000) for a more thorough description of these problem sets. The *LLH* nomenclature identifies a *L*ow and *H*igh setting for each of the three parameters in order: *default iterations, bad consecutive starts before quitting,* and *maximum places to insert.*

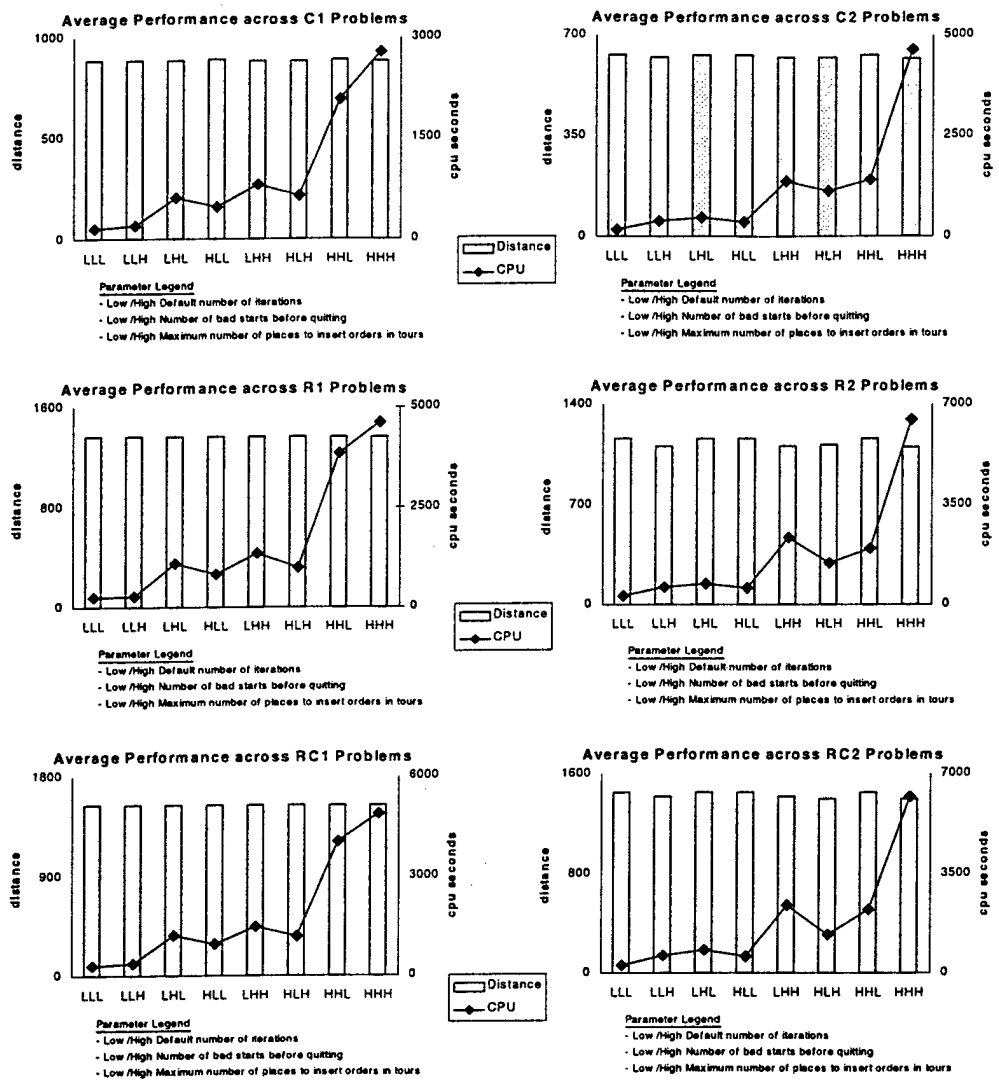


Figure 21: Visual summary of parameter settings tests

The line representing the solve time grows very quickly—almost a full order of magnitude—as the parameter setting levels increase. This suggests one should keep the parameters at the low settings. The small amount of variation in the solution qualities (the bars) provides no reason to increase the parameter settings.

These simple tests confirmed what experience in building the tabu search suggested: that the low settings for the three parameters was sufficient to adequately search the solution space and arrive at good solutions.

The Universal Vehicle Router, Sample Tabu Search, and Solomon Data Sets

Since the UVR and sample tabu search is a general-purpose tool, it is not expected to be the fastest solver or come as close to the optimal solution as other more specific solvers. The question is whether or not the ease with which the UVR and sample tabu search could be implemented outweighs its solution quality gap between the best known solutions and the quick-turn solutions.

Figure 22 shows the results of the runs (at parameter setting LLL) along with the best known solutions, as reported in the summary of best known solutions to the Solomon problems published by Kinney (2000). Values are rounded to the nearest whole number.

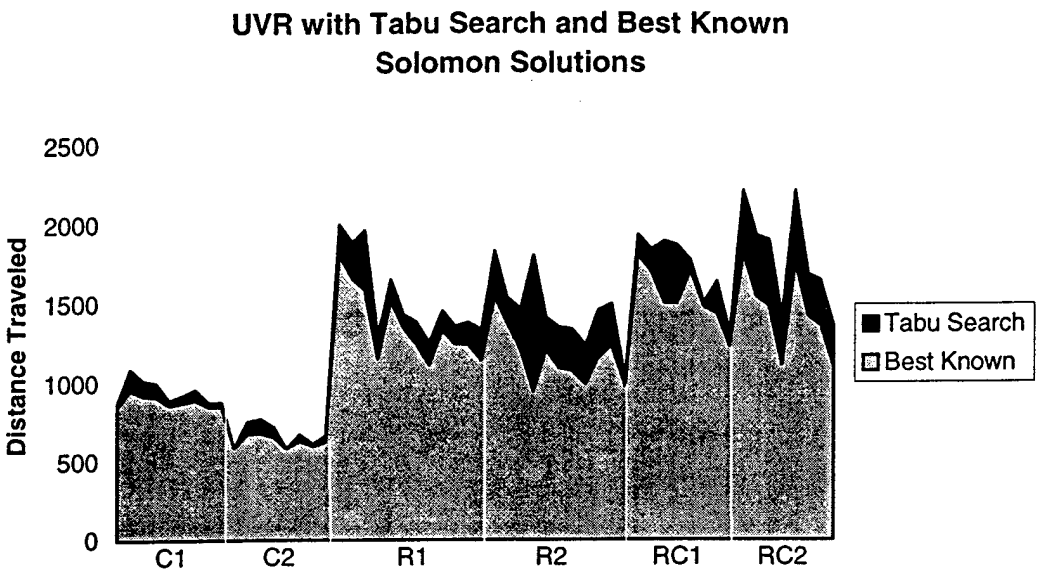


Figure 22: Comparing UVR with tabu search and best known Solomon solutions

Although the distance data represents 56 discrete problems, the chart represents the data as continuous which is easier to read than 56 bars. The lighter area represents the best known solutions for each Solomon problem. The darker area on top represents the gap between the best known solutions and the solutions found by the UVR and sample tabu search solver. Table 5 shows all of the data. The sample tabu search and UVR, in addition to being easy to use, fare well on the standard problems.

Table 5: Performance of UVR against best known solutions

	Number of Vehicles		*Distance Traveled*		*Solve Time (sec)*		*Source of*
	UVR	*Best*	*UVR*	*Best*	*UVR*	*Best*[2]	*best known*
C101	10	10	852	827	69		*1*
C102	10	10	960	827	18		*1*
C103	10	10	923	826	188		*2*
C104	10	10	913	823	263		*2*
C105	10	10	860	827	80		*2*

[2] The solve times for the best known solutions are not readily available since they are not often published. Often the best known solutions are discovered after many runs taking many hours which makes the interpretation of *solve time* difficult to publish.

	Number of Vehicles		Distance Traveled		Solve Time (sec)		Source of
	UVR	Best	UVR	Best	UVR	Best[2]	best known
C106	10	10	877	827	141		1
C107	10	10	894	827	113		1
C108	10	10	853	827	151		1
C109	10	10	854	827	240		2
C201	3	3	591	592	83		3
C202	3	3	676	592	179		3
C203	3	3	683	591	204		4
C204	3	3	656	591	259		3
C205	3	3	588	589	141		3
C206	3	3	633	588	172		3
C207	3	3	601	588	159		4
C208	3	3	629	588	163		4
R101	20	18	1805	1608	207		1
R102	19	17	1661	1434	251		1
R103	14	13	1587	1207	272		5
R104	11	9	1156	1007	243		6
R105	14	14	1517	1377	228		4
R106	13	12	1344	1252	213		4
R107	12	10	1247	1105	228		6
R108	10	9	1112	964	245		6
R109	13	11	1334	1206	251		6
R110	12	10	1248	1135	248		6
R111	11	10	1242	1097	223		6
R112	10	10	1148	954	232		4
R201	4	4	1544	1254	197		7
R202	4	3	1378	1214	254		8
R203	3	3	1210	949	268		4
R204	3	2	946	867	372		7
R205	3	3	1208	999	234		7
R206	3	3	1094	833	279		5
R207	3	3	1078	815	326		4
R208	2	2	989	739	407		4
R209	3	3	1157	855	293		5
R210	3	3	1232	963	258		7
R211	3	2	980	924	364		8
RC101	16	14	1802	1669	223		5
RC102	14	12	1698	1555	269		8
RC103	13	11	1502	1110	322		5
RC104	13	10	1502	1135	327		6
RC105	16	13	1706	1643	285		8
RC106	13	11	1478	1448	355		8
RC107	12	11	1434	1230	286		6
RC108	11	10	1228	1140	261		8
RC201	4	4	1810	1407	176		7
RC202	4	4	1542	1153	312		7

	Number of Vehicles		*Distance Traveled*		*Solve Time (sec)*		*Source of*
	UVR	*Best*	*UVR*	*Best*	*UVR*	*Best*[2]	*best known*
RC203	3	3	1484	1068	258		*7*
RC204	3	3	1113	804	336		*7*
RC205	4	4	1758	1302	228		*7*
RC206	4	3	1421	1156	214		*7*
RC207	4	3	1362	1075	239		*7*
RC208	3	3	1099	834	356		*4*

1. Desrochers *et al.* 1992
2. Kohl and Madsen 1997
3. Potvin and Bengio 1996
4. Rochat and Taillard 1995
5. Thangiah *et al.* 1994
6. Shaw 1997
7. Kilby *et al.* 1997
8. Taillard *et al.* 1997

Conclusion

Tabu search parameters were examined against the Solomon standard problem set. To solve the Solomon problems required a simple interface to the UVR. By design no changes to the UVR were necessary to accommodate the Solomon application. The sample tabu search found good solutions to the problems

Chapter Five: Conclusion

Specific Contributions

This thesis presents a proposed generic architecture for analytic support. This architectures uncouples the specifics of particular problems from the techniques and mechanisms used to obtain solutions to the class of problems. The layers proposed and defined in this architecture provide specific functionality used by other layers in the architecture.

The architecture was built using routing problems as the focus. Each layer of the architecture was populated with components appropriate to that layer. A prototype application for routing UAVs was built on this architecture to meet the specific needs of the 11th Reconnaissance Squadron.

The heart of the architecture, the Universal Vehicle Router (UVR), builds upon an initial taxonomy for routing problems to provide a reusable component applicable to routing applications. Module reuse is demonstrated by building a Solomon test set solution application on top of the UVR.

A general tabu search engine was used to provide the base functionality employed by the UVR and other research efforts. This general tabu search is enhanced by an adaptive tabu search providing quality answers to vehicle routing problems.

The prototype AFIT Router application, made possible by the supporting architecture, has a simple-to-use yet data-rich interface. By maintaining the data and interface

to the UVR in the core AFIT Router kernel, the router frees developers to include general airframe routing without reaching further down into the architecture.

Recommendations for Future Work

Avenues for future work, both applied and theoretical, abound due to the modularity of the architecture. Applied research can focus on increasing realism in data representation such as maps, routing concerns such as weather and threats, or application interface issues such as tighter integration with the DEMPC. Already underway is follow-on work to define and include pertinent weather concerns in the core AFIT Router. Additionally, post-thesis discussions are underway to operationalize the AFIT Router by including it in both the DEMPC by Boeing and mapping software by a different contractor.

Efforts in theoretical advances can focus on expanding the suite of general solvers for the routing architecture. Another extension is to define and populate similar architectures for other problem classes.

A combined applied and theoretical effort could examine over-target routing tools where concerns specific to target-types and platform capabilities determine the specific flight path characteristics such as approach angle and standoff distance. Such tools can exploit lower levels of the routing architecture while defining their specific solver interfaces. This would provide the UAV operator a true end-to-end routing capability.

Bibliography

Ackoff, Russell L. "The Future of Operational Research is Past," *Journal of the Operational Research Society*, 30 (1979).

Battiti, Roberto. "Reactive Search: Toward Self-Tuning Heuristics," *Modern Heuristic Search Methods,* Rayward-Smith (ed.), John Wiley and Sons Ltd: 61-83, (1996)

Calhoun, Kevin. *Tabu Search for Combat Aircraft Scheduling and Rescheduling.* MS thesis, AFIT/GOR/ENS/00M-6. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Carlton, William B. *A Tabu Search to the General Vehicle Routing Problem.* Ph.D. dissertation. University of Texas, Austin, TX, (1995).

Carlton, William B. and J. Wesley Barnes. "A Note on Hashing Functions and Tabu Search Algorithms," *European Journal of Operational Research, 95:* 237-239 (1996).

Cullenbine, Christopher. *Tabu Search Approach to the Weapons Assignment Model (WAM).* MS thesis, AFIT/GOR/ENS/00M-8. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Desrochers, Martin, Jacques Desrosiers, and Marius Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows," *Operations Research, 40*: 342-354 (March-April 1992).

Flood, R. *A Java Human Computer Interface for Displaying Maps in Support of a UAV Decision Support Tool.* MS thesis, AFIT/GCS/ENS/99M-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

Garcia, Bruno-Laurent, Jean-Yves Potvin, and Jean-Marc Rousseau. "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints," *Computers and Operations Research, 21*: 1025-1033 (1994).

Gendreau, Michel, Alain Hertz, and Gilbert Laporte. "A Tabu Search Heuristic for the

Vehicle Routing Problem," *Management Science, 40*: 1276-1289 (October 1994).

Gendreau, Michel, Alain Hertz, and Gilbert Laporte. "New Insertion and Postoptimization Procedures for the Traveling Salesman Problem," *Operations Research, Vol 40, No. 6:* 1086-1094 (1992).

Gendreau, Michel, Gilbert Laporte, and René Séguin. "A Tabu Search Heuristic for the Vehicle Routing Problem with Stochastic Demands and Customers," *Operations Research, 44*: 469-477 (May-June 1996).

Gendreau, Michel, Gilbert Laporte, Christophe Musaraganyi, and Éric D. Taillard. "A Tabu Search Heuristic for the Heterogeneous Fleet Vehicle Routing Problem," *Computers and Operations Research, 26:* 1153-1173 (1999).

Glove, Fred. "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research, 13*: 533-549 (1986).

Golden, Bruce L., Gilbert Laporte, and Éric D. Taillard. "An Adaptive Memory Heuristic for a Class of Vehicle Routing Problems with MinMax Objective," *Computers and Operations Research, Vol 24, No. 5:* 445-452 (1997).

Hall, Shane. *A Group Theoretic Tabu Search Approach to the Traveling Salesman Problem.* MS Thesis, AFIT/GOR/ENS/00M-14. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Kilby P., P. Prosser, and P. Shaw. "Guided Local Search for the Vehicle Routing Problem," In *Proceedings of the 2nd International Conference on Meta-heuristics,* (1997).

Kinney, Gary. *A Hybrid Jump Search and Tabu Search Metaheuristic for the Unmanned Aerial Vehicle (UAV) Routing Problem.* MS thesis, AFIT/GOA/ENS/00M-5. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Kohl, N. and O.B.G. Madsen. "An Optimization Algorithm for the Vehicle Routing Problem with Time Windows Based on Lagrangian Relaxation," *Operations Research,* 45(3): 395 (1997).

Laporte, Gilbert, Yves Nobert, and Serge Taillefer. "Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems," *Transportation Science: Vol 22, No. 3*: 161-172 (1988).

Laporte, Gilbert. "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms," *European Journal of Operational Research, 59:* 345-358 (1992).

O'Rourke, K. *Dynamic Unmanned Aerial Vehicle Routing with a Java-encoded Reactive Tabu Search Metaheuristic.* MS thesis, AFIT/GOA/ENS/99M-06. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

Potvin, J.-Y. and S. Bengio. "The Vehicle Routing Problem with Time Windows—Part II: Genetic Search," *ORSA Journal on Computing,* 8(2): 165 (1996).

Renaud, Jacques, Gilbert Laporte, and Fayez F. Boctor. "A Tabu Search Heuristic for the Multi-Depot Vehicle Routing Problem," *Computers and Operations Research, 23*: 229-235 (1996).

Rochat, Y. and E. Taillard. "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics*, 1(1): 147-167 (1995).

Ryan, Joel L. *Embedding a Reactive Tabu Search Heuristic in Unmanned Aerial Vehicle Simulations.* MS thesis, AFIT/GOR/ENS/98M. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February 1998.

Shaw, P. "A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems," APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK. (June 1997).

Taillard, E., P. Badeau, M. Gendreau, F. Guertain, and J.-Y. Potvin. "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, 32(2) (1997).

Talbi, E.G., Z. Hafidi, and J-M. Geib. "A Parallel Adaptive Tabu Search," *Parallel Computing, 24*: 2003-2019 (1998).

Thangiah, S.R., I.H. Osman, and T. Sun. *Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows.* Technical Report UKC/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury, UK. (1994).

Toulouse, Michel, Teodor G. Crainic, and Michel Gendreau. "Communication Issues in Designing Cooperative Multi-Thread Parallel Searches," International Conference. Boston: Kluwer Academic Press, 1996.

Tsubakitani, Shigeru and James R. Evans. "An Empirical Study of a New Metaheuristic for the Traveling Salesman Problem," *European Journal of Operational Research, 104*: 113-128 (1998).

Woodruff, David L. and Eitan Zemel. "Hashing Vectors for Tabu Search," *Annals of Operations Research, 41:* 123-137 (1993).

Additional References

Armentano, Viníécius A. and Débora P. Ronconi. "Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems," *Computers and Operations Research, 26*: 219-235 (1999).

Barbarosoglu, Gulay and Demet Ozgur. "A Tabu Search Algorithm for the Vehicle Routing Problem," *Computers and Operations Research, 26*: 255-270 (1999).

Crainic, Teodor Gabriel and Michel Gendreau. "Towards an Evolutionary Method—Cooperating Multi-Thread Parallel Tabu Search Hybrid," *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Boston: Kluwer Academic Publishers, 1997.

Cullenbine, Christopher. *Tabu Search Approach to the Weapons Assignment Model (WAM).* MS thesis, AFIT/GOR/ENS/00M-8. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Glover, Fred and Manuel Laguna. *Tabu Search.* Boston: Kluwer Academic Publishers, 1997.

Glover, Fred. "Heuristic for Integer Programming Using Surrogate Constraints," *Decision Sciences, 8*: 156-166 (1977).

Hall, Shane. *A Group Theoretic Tabu Search Approach to the Traveling Salesman Problem.* MS Thesis, AFIT/GOR/ENS/00M-14. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

Liaw, Ching-Fang. "A Tabu Search Algorithm for the Open Shop Scheduling Problem," *Computers and Operations Research, 26*: 109-126 (1999).

Løkketangen, Arne and Fred Glover. "Solving Zero-One Mixed Integer Programming Problems Using Tabu Search," *European Journal of Operational Research, 106*: 624-658 (1998).

Nowicki, Eugeniusz and Czeslaw Smutnicki. "A Fast Taboo Search Algorithm for the Job Shop Problem," *Management Science, 42*: 797-813 (June 1996).

Toulouse, Michel, Teodor Gabriel Crainic, Brunilde Sansó, and K. Thulasiraman. "Self-Organization in Cooperative Tabu Search Algorithms," *IEEE International Conference on Systems, Man, and Cybernetics, 3*: 2379-2384 (1998).

Tsubakitani, Shigeru, and James R. Evans. "Optimizing Tabu List Size for the Traveling Salesman Problem," *Computers and Operations Research, Vol 25, No. 2:* 91-97 (1998).