Intl. Trans. in Op. Res. 11 (2004) 259–275

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

A Java™ universal vehicle router for routing unmanned aerial vehicles

R.W. Harder[a], R.R. Hill[b] and J.T. Moore[c]

[a]*Air Force Personnel Operations Agency, Crystal City, Virginia,* [b]*Department of Biomedical, Industrial and Human Factors Engineering, Wright State University, Dayton Ohio, and* [c]*Department of Operational Sciences, Air Force Institute of Technology, Dayton, Ohio*

Corresponding author e-mail: ray.hill@wright.edu

Received 3 January 2002; accepted 7 November 2002

Abstract

We consider vehicle routing problems in the context of the Air Force operational problem of routing unmanned aerial vehicles from base locations to various reconnaissance sites. The unmanned aerial vehicle routing problem requires consideration of heterogeneous vehicles, vehicle endurance limits, time windows, and time walls for some of the sites requiring coverage, site priorities, and asymmetric travel distances. We propose a general architecture for operational research problems, specified for vehicle routing problems, that encourages object-oriented programming and code reuse. We create an instance of this architecture for the unmanned aerial vehicle routing problem and describe the components of this architecture to include the general user interface created for the operational users of the system. We employ route building heuristics and tabu search in a symbiotic fashion to provide a user-defined level-of-effort solver interface. Empirical tests of solution algorithms parameterized for solution speed reveal reasonable solution quality is attained.

Keywords: vehicle routing, tabu search, object-oriented programming, empirical analysis

1 Introduction

The traveling salesman problem (TSP) and vehicle routing problem (VRP) are members of a general class of extremely difficult problems. The TSP requires uniquely visiting a number of customers using one or more salesmen or vehicles. The VRP adds vehicle capacities and customer demands to the TSP. Often there are other 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. Various problem extensions have pushed problem complexity closer to real-world problem complexity so that traditional solution techniques, such as branch-and-bound,

become less applicable. There are a variety of non-traditional techniques available to solve TSP and VRP problems, but tabu search appears to be the most effective (Laporte, 1992).

Tabu search is a heuristic developed by Glover (1986) to intelligently search a problem's solution space. Tabu search works on the principle that the intelligent use of memory can help a search find and escape from local optima while fully investigating each local optimum's region. Common features of tabu search include intensification and diversification. Tabu search uses these features to fully investigate regions about a particular solution (intensification) or to move to a different part of the solution space (diversification). Details of tabu search are found in Glover and Laguna (1997). For this work we assume a basic knowledge of tabu search.

A tabu search may save good solutions encountered during the search and produce an élite list of solutions. Tabu search may then use élite list entries as a means to restart the search at those solutions. Tabu search may spend some time building this élite list, but the search intensification returns often outweigh the list-building effort.

Unmanned aerial vehicles (UAVs) serve military forces by flying in dangerous areas primarily for surveillance missions. UAVs have flight times that far exceed those of manned aircraft. Long flight times mean a UAV may visit many sites, or targets, during a mission. Because of the myriad planning considerations, which include visitation time requirements and airspace flight restrictions, finding a good path among the sites that the UAV must visit is a daunting task. Even more daunting is to modify a path to accommodate new sites, or 'pop-up targets', particularly when the path planning must be accomplished quickly, sometimes immediately. Current UAV planning tools do not automate the UAV routing problem.

The routing of unmanned aerial vehicles in military environments is a perfect example of a complex, real-world routing problem. We propose a layered architecture for, and present a prototype application of, a routing tool to support both the preplanning and real-time re-tasking phases of the UAV routing task. The architecture components and prototype application are coded in Java™ and designed to enhance UAV operator planning tasks. The routing algorithm builds on initial work by O'Rourke *et al.* (2001) but adds new UAV considerations — site priorities and restricted geographic operating zones — and new tabu search techniques.

Figure 1 depicts the main components of our layered architecture (shown left to right) and suggests how these components collectively realize the AFIT Router specified for UAV routing problems. The core feature of our architecture and prototype application is a universal vehicle router (UVR). The UVR supervises the solution process (generating and improving routes) by

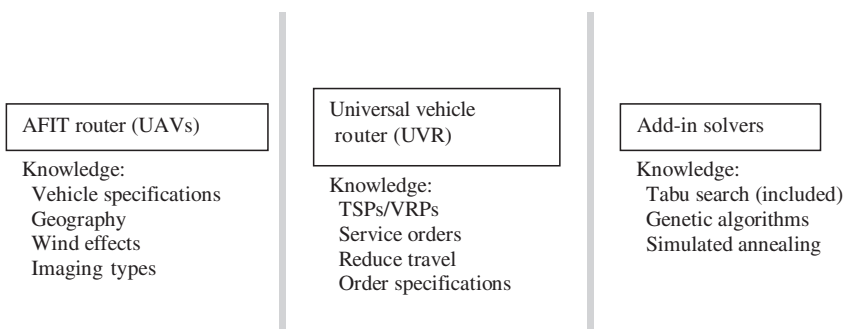


Fig. 1. AFIT router layers.

exploiting object-oriented programming techniques to interface to more general purpose add-in solvers.

2 An architecture for optimization applications

As problems grow in complexity and the time available to obtain solutions diminishes, operational research analysts increasingly need flexible tools to solve classes versus specific instances of problems (e.g. vehicle routing, assignment, scheduling). At an analysis level, we move away from a focus on techniques (e.g. linear programming, integer programming, heuristic libraries) to a focus on layered tools. Creating flexible tools requires an understanding of situations facing analysts and how one can divorce the problem class/solution technique relationship. Figure 2 depicts the analysis situation as a three-layer hierarchy.

The top level in Fig. 2 shows the messes that decision-makers deal with regularly and for which analysis provides insight (Ackoff, 1979). An analyst examines and frames the mess as a particular class of problem, such as a vehicle routing or scheduling problem. Once the problem is classified, there are a variety of solution techniques available and an analyst can apply one or more to the specific problem. The solution technique application should be readily available and decoupled from the specifics of the problem class instance under examination.

2.1 The architecture

The analytic community needs an architecture that focuses analytic and development efforts by facilitating reuse rather than reinvention or re-implementation. With such an architecture, an analyst focused on applications can solve an instance from a problem class without recreating a solution technique. Another analyst, focused on algorithm development, can test a new technique on existing problems nearly in isolation of the specifics of the problem instance. Figure 3 lays out such an architecture. The left side of Fig. 3 maps this architecture back to the analytical hierarchy in Fig. 2. There is inherent flexibility in this architecture; many additional messes, problem classes, and solution techniques are easily added to the architecture.

Developing software at each level requires an analyst identify that level's common elements. For example, at the technique level, setting a variable value is not common among tabu searches, but evaluating a solution neighborhood and selecting a best move is common. At the problem class level, not all TSPs use surface roads as routes, but all TSPs do involve a travel cost. Modules

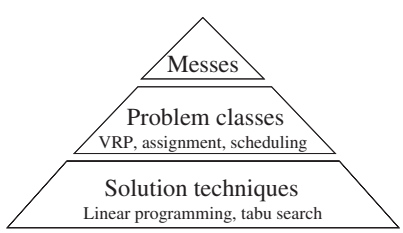


Fig. 2. The situation facing analysts.

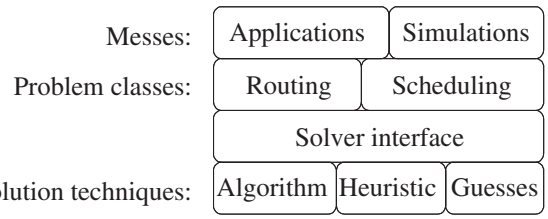


Fig. 3. Architecture for the general operations research software.

at each layer communicate via defined interfaces. For example, a capital budgeting application passes information to a multi-dimensional knapsack problem class model, which then passes information to an available solution technique. The solution technique could be a greedy heuristic, a meta-heuristic, or some branch-and-bound optimization routine; the solution technique choice is transparent to layers above the solver layer in the architecture.

2.2 An architecture instance for VRPs and TSPs

The general architecture proposed in Fig. 3 was instantiated to support the UAV routing problem. More specifically, a UAV routing-specific interface was built on top of the routing components within an instance of the architecture defined above. Although windows and buttons in an interface may change, the elements of UAV routing remain constant: multiple starting locations, wind speed, flying distances between sites, etc. Thus, the UAV routing software application is partitioned into a prototype application and a core component identified as 'core AFIT router'. The core AFIT router is derived from a vehicle routing class and employs a layer called the 'universal vehicle router'. At the lowest level of this architecture, we defined and deployed a general tabu search (general TS) solution technique. The entire application is coded in Java™ for flexibility in distribution (in fact, modifications to this AFIT router application resulted in the OpenTS application available at www.*i*Harder.net). The highlighted sections in Fig. 4 are the components built for this specific instance, each of which is described in the next section. Non-highlighted blocks represent alternative components one could employ within this general architecture.

3 Prototype application

The prototype application was designed for simplicity, practicality, and rapid response to encourage its use in solving actual UAV routing problems. In our case, the application was designed as a pop-up application within a larger software system in use within the UAV operational environment. The UAV operational environment involves multiple targets, heterogeneous vehicles, environmental considerations, and a variety of operational considerations that constrain the routing. The front panel (Fig. 5) contains important summary information and allows quick access to more detailed information concerning the multiple sites and multiple vehicles within a typical UAV routing problem.

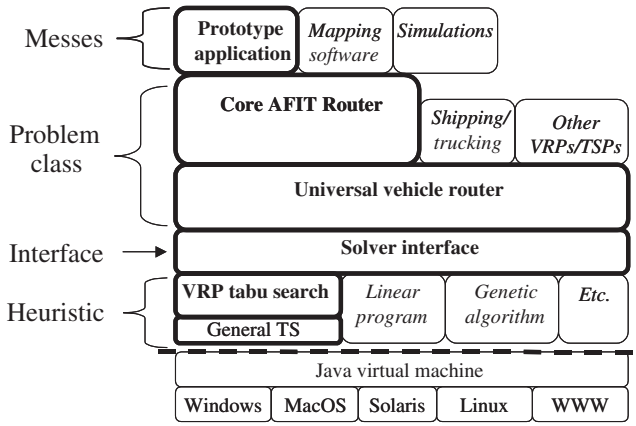


Fig. 4. Architecture for vehicle routing and traveling salesman class of problems.

Fig. 5. Screenshot of main prototype AFIT router panel (MacOS).

Figure 6 is the site definition window providing access to relevant site information. Site information may be loaded from files or copied from spreadsheets. Sites are designated by a name, a latitude, and a longitude. Each site has a service time in minutes (Service…) that includes the amount of time spent loitering at or around the site. Site priorities (Priority) allow the user to specify site priorities to use if all sites cannot be visited. Other windows in the application allow the user to specify priority handling. The requirements field (Require…) provides a matching of operational need at a site (laser designation, synthetic aperture radar, etc.) to any vehicle with matching capability. No requirement specified means that any vehicle may service the site. The time window and time wall fields (in order depicted, earliest arrival (Earliest…), latest departure (Latest…), earliest restricted (Earliest…), and latest restricted (Latest…)) provide a means to specify when a site may or may not be visited. This window is intentionally spreadsheet compliant to enhance ease of use in operational settings where 'target decks' are often provided through receipt of a spreadsheet file.

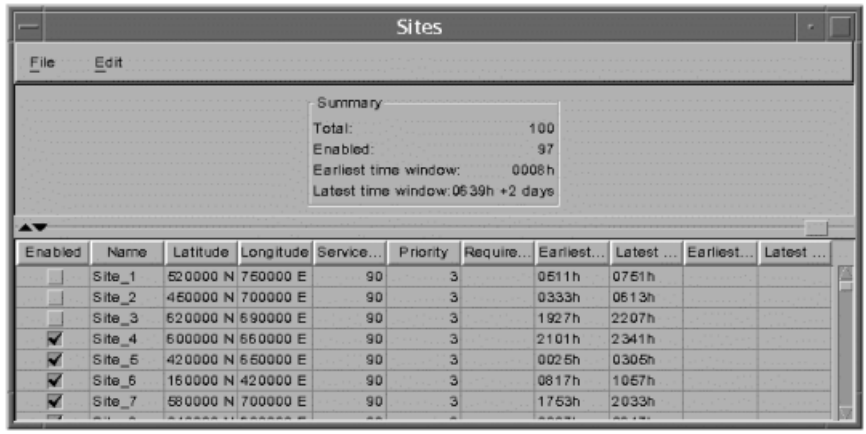


Fig. 6. Screenshot of sites screen (Solaris/CDE).

Figure 7 is the vehicle and base definition window. A base (Home) may be specified for a UAV and by default, the UAV will leave from and return to the specified base. Since an important operational consideration is to dynamically alter defined routes to accommodate real-time re-tasking, we implemented a means to define new or alternate starting locations. Selecting Use Alternate Location (Use Al…) treats the UAV as leaving from the alternate latitude (Alt. Lat…) and longitude (Alt. Lo…) location and returning to the home base. The capabilities field (Capabi…) is used to match a site's requirement to vehicle capabilities. A UAV may only cover a site for which it is operationally equipped. The speed (Speed…) in knots, range (Range…) in hours, and altitude (Altitud…) in feet are used to calculate travel and endurance times. The start-time field (Start Ti…) specifies when the vehicle is available for take-off, in the case of preplanning a mission, or the current time, in the case of real-time re-tasking.

An academic researcher prefers 'best' solutions to a problem. An operator would like best solutions but would accept a 'good' solution received 'in time' to a best solution received late. We adopted a novel approach to straddle the solution quality to solution-time issue.

Upon clicking the solve button on the main panel (Fig. 5), the user is presented with choices (Fig. 8) regarding how to treat site priorities and how much time to spend solving the problem. Three priority schemes are depicted. The Absolute Priorities scheme covers the higher priority sites before including lower priority sites. The Flex Priorities scheme provides substitutability among sites (e.g. five priority-2 sites can replace a priority-1 site). Finally, the Custom Priorities scheme allows the user to combine features of Absolute and Flex priority. The Solve Time slider lets the user decide how quickly the solution must be returned. The Use post-optimization box invokes the longer running, more comprehensive tabu search algorithm to obtain better solutions to the particular routing problem. Prior to commencing activities, when time is available, a user might select Longer and use post-optimization. In a dynamic environment, the user will likely move the slider to Shorter and not use post-optimization.

Figure 9 shows a sample solution summary and a visual display of the routing solution. Clicking the 'Details…' button yields the solution details depicted in Fig. 10. Solution details include estimated arrival and departure times for the sites visited by the vehicles in the solution. Data (solutions) can be saved using the 'copy to clipboard' button.

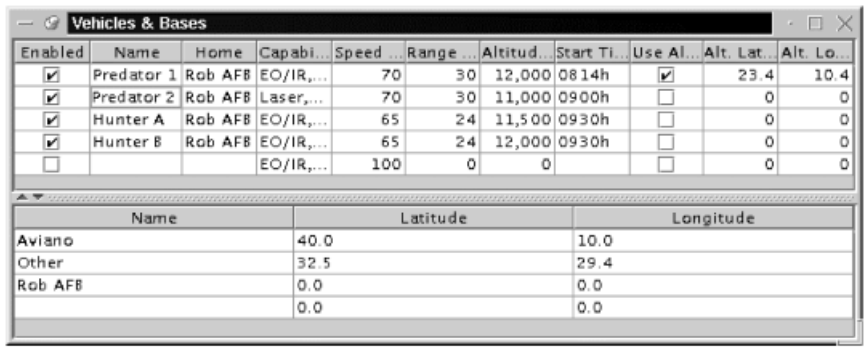


Fig. 7. Screenshot of vehicles and bases screen (Linux/KDE).

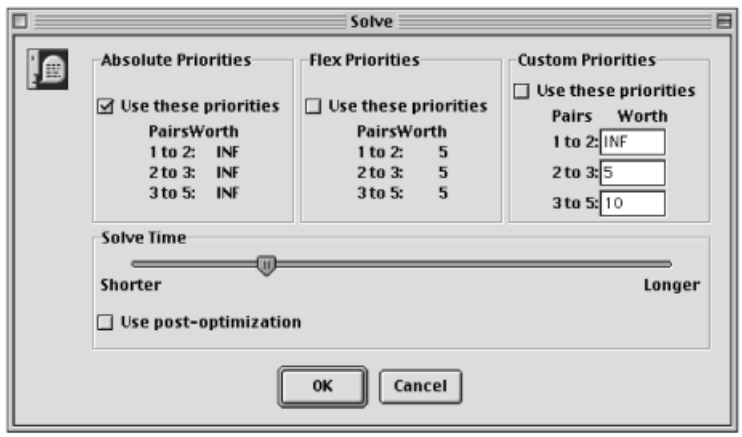


Fig. 8. Solve dialog for prototype AFIT Router (MacOS).

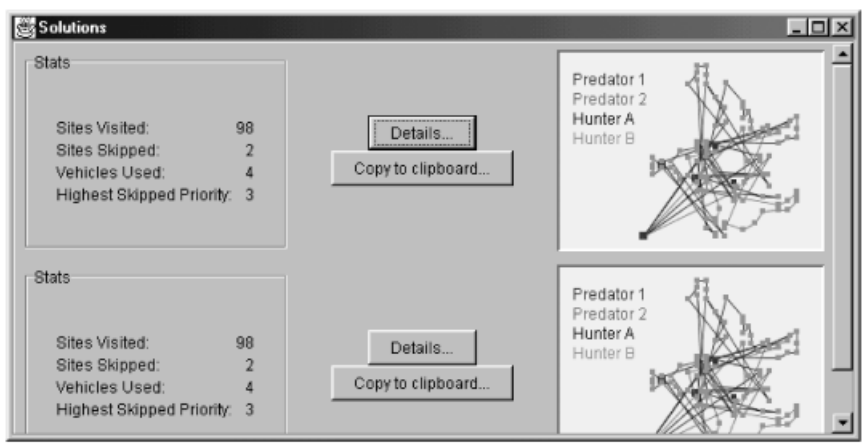
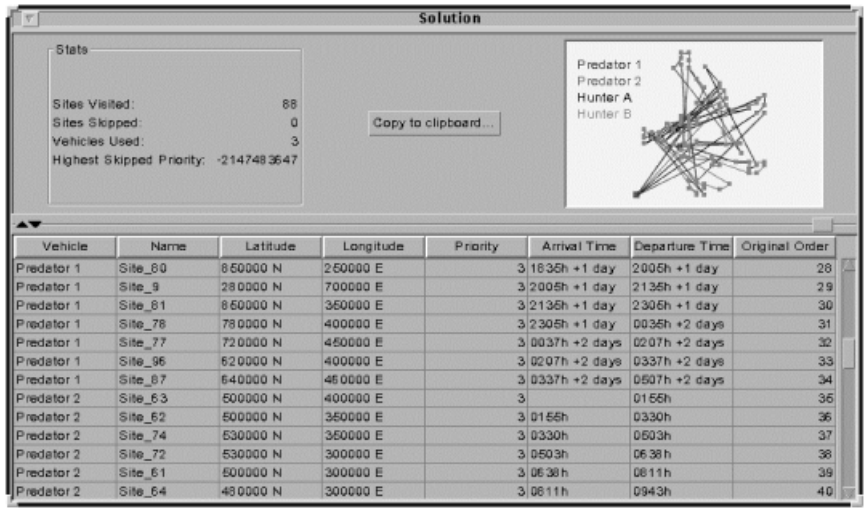


Fig. 9. Screenshot of multiple solutions screen (Windows).



Vehicle	Name	Latitude	Longitude	Priority	Arrival Time	Departure Time	Original Order
Predator 1	Site_60	650000 N	250000 E	3	1835h +1 day	2005h +1 day	28
Predator 1	Site_9	280000 N	700000 E	3	2005h +1 day	2135h +1 day	29
Predator 1	Site_81	650000 N	350000 E	3	2135h +1 day	2305h +1 day	30
Predator 1	Site_78	780000 N	400000 E	3	2305h +1 day	0035h +2 days	31
Predator 1	Site_77	720000 N	450000 E	3	0037h +2 days	0207h +2 days	32
Predator 1	Site_96	620000 N	400000 E	3	0207h +2 days	0337h +2 days	33
Predator 1	Site_87	640000 N	460000 E	3	0337h +2 days	0507h +2 days	34
Predator 2	Site_63	500000 N	400000 E	3		0155h	35
Predator 2	Site_62	500000 N	350000 E	3	0155h	0330h	36
Predator 2	Site_74	530000 N	350000 E	3	0330h	0503h	37
Predator 2	Site_72	530000 N	300000 E	3	0503h	0638h	38
Predator 2	Site_61	500000 N	300000 E	3	0638h	0811h	39
Predator 2	Site_64	480000 N	300000 E	3	0811h	0943h	40

Fig. 10. Screenshot of single solution screen (Solaris/OpenWindows).

Our focus in developing this prototype AFIT Router application was the UAV operators who must route UAVs to sites specified on a tasking order or site list. The site list is typically provided in spreadsheet form so we provided a copy and paste function. Although the prototype is specific to the needs of the Air Force's 11th Reconnaissance Squadron, it does serve as a presentation mechanism for the architecture's use in other areas, most recently with the Navy global command and control system intelligence surveillance and reconnaissance capability (GISRC) system (Burdell, 2001).

4 Core AFIT router

The core AFIT router is concerned with how the data are stored and manipulated. The core AFIT router *kernel* (Fig. 11) links the data structures within the architecture and the application software. Any component needing a route interacts with the AFIT router core kernel to obtain the route

The kernel provides access to lists of vehicles, sites, winds, restricted operating zones, and solutions. Applications using the core AFIT router kernel 'listen' for changes to these lists and when triggered, reflect these changes by updating the summary information presented to the user via the application layer. Table 1 shows the information tracked for the various components in the core AFIT router. Restricted operating zones are used to specify time windows and time walls for particular geographic regions. Travel times between sites are calculated using great circle distance methods and accounting for effects of winds aloft.

5 Universal vehicle router

The core AFIT router kernel employs the services of the universal vehicle router (UVR) for routing. The UVR provides a means to solve a wide variety of VRPs and TSPs. The UVR

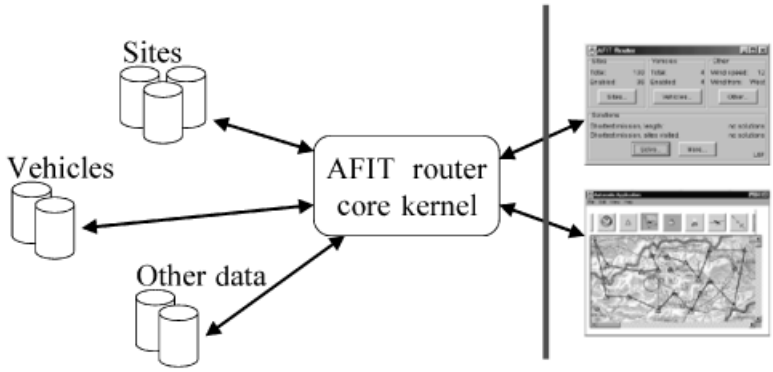


Fig. 11. AFIT router core kernel as a point of contact.

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

interfaces to an application/data structure, abstracts out the common routing features, and passes this 'common' problem to the solution technique, which then solves the problem.

The UVR interface is straightforward. From the higher level, the UVR requests information about vehicles and *orders* (UVR terminology for sites or customers). From the lower levels, it requests solutions that route vehicles to orders. Table 2 shows the information and control requested from higher-level software by the UVR. Priority values are assumed in ascending order where lower values mean higher priority.

The UVR stores solution information concerning each vehicle and its assigned orders as well as any vehicles not employed and orders not visited. Figure 12 provides a graphical depiction of the solution data structure. Note a 'Solution' contains one to many tours and each 'tour' contains tour specific information along with the list of orders assigned the vehicle for that tour. Unvisited orders are maintained in a similar data structure dubbed the 'dummy tour'.

The UVR lets users define an 'evaluator', which is used by the lower-level solvers to determine solution quality. A typical evaluator might specify minimizing the number of exceeded vehicle ranges as a primary goal, the number of skipped orders as a secondary goal, and the total travel time as a tertiary goal. The UVR default 'evaluator' minimizes the following: the number of exceeded vehicle ranges, the number of violated time windows, the number of skipped orders of descending priority (variable number of priorities), travel time penalties, and total wait time. For

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

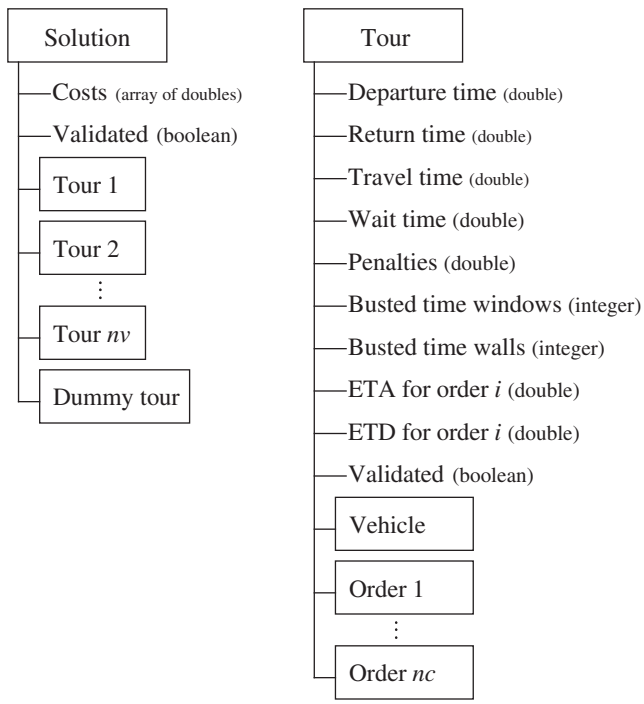


Fig. 12. Representation of a solution in the UVR.

UAV applications, we employ travel time penalties to force routes away from identified threat areas.

6 Solver interface

The solver interface does not preclude any viable solution technique from use by the UVR. Rather, our approach encourages a diverse set by defining a solver interface between the UVR and the actual solution technique. The solver interface receives all orders initially placed in the dummy tour, accesses pertinent vehicle information, formulates the appropriate problem, and requests a solution from one of the lower-level solvers.

As previously discussed, a novel aspect of the application level of the architecture is the solution effort parameter. This is a capability particularly suited to the needs and operational requirements of the UAV routing community. Variable solution effort is enabled using two interconnected solution methods and the architecture's ability to accommodate multiple solution techniques. A tour-building heuristic, detailed by Kinney *et al.* (2001), generates a list of starting solutions ranked by solution quality. Shorter solution time might involve returning the best starting solution found. Additional allotted time is used to start local searches using as many starting solutions as time permits, with the best overall solution returned. Long search time or post-optimization involves a tabu search solver described below.

7 Adaptive tabu search solver

The tabu search solver builds on a general tabu search engine (described below) and features a sorted élite list of starting solutions, a user-specified level of search effort, and an adaptive level of search effort.

Each starting solution generated by the tour-building heuristic is used to initiate a tabu search. When the search stalls, the next-best remaining solution is used to start a new search. The number of solutions used to re-start the search is a function of the user-specified level of search effort: less effort, fewer starting solutions; more effort, more starting solutions.

Each starting solution is evaluated for a minimum number of iterations. The search proceeds if improvements continue and moves to the next starting solution when the search stalls (e.g. at a local minimum). The entire search stops if (1) there are no further starting solutions, (2) the user defined level of effort is reached, or (3) the overall search has stalled and further effort is likely unproductive. Figure 13 outlines the steps in this adaptive tabu search solver.

The tabu search implements four move types: relocate orders within a tour, relocate orders to another tour, relocate orders to the dummy tour, and relocate orders from the dummy tour. The first two move types insert orders a maximum number of places as defined by

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

where n_t is the number of orders currently in the tour at iteration t. These two move types are generated in alternate iterations. This reduces the size of the search neighborhood thereby reducing the computational burden. The last two move types, moving orders in and out of the dummy tour, are generated each iteration. Each order in a tour is moved to the dummy tour, and each order in the dummy tour is moved to each tour. Since this tabu search builds on a list of good starting solutions, there are generally not many orders in the dummy tour, so this phase is not computationally expensive.

8 General tabu search

The adaptive tabu search employed by the UVR was built on a general tabu search engine. This general tabu search engine abstracts out the common elements in a tabu search and provides a

Initialize:

Set n = number of orders or sites

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{ n*s, 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 on starting solution c to $g = m$
2. Perform g iterations
3. If a better solution has been found within R iterations, set $g = E$ and go to 2
4. If a new global best solution has been found set $b = c$.
5. If $c - b >= B$, quit.
6. Set $c = c + 1$. If $c > s$, quit, else go to step 1

Fig. 13. Steps for the adaptive tabu search.

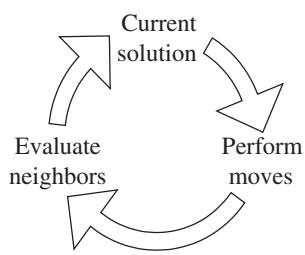


Fig. 14. General cycle of a tabu search.

framework to build specific tabu searches. We found tabu searches follow the pattern shown in Fig. 14: a given solution is altered and evaluated before a new current solution is chosen.

The general tabu search engine ('Engine' in Fig. 15) allows analysts to concentrate on defining the specific of the tabu search. Figure 15 depicts how an analyst can create objects that provide specific capabilities to the engine. These objects, written in Java™, 'listen' for key events to trigger specific tabu search strategies such as intensification, diversification, and strategic oscillation.

To create a specific tabu search, an analyst defines each of the required objects for the engine. The engine then performs the search as defined by the analyst's inputs.

The objective function and penalty function objects provide a means to evaluate a solution, as defined by the solution definition objects. The move manager object determines which moves to

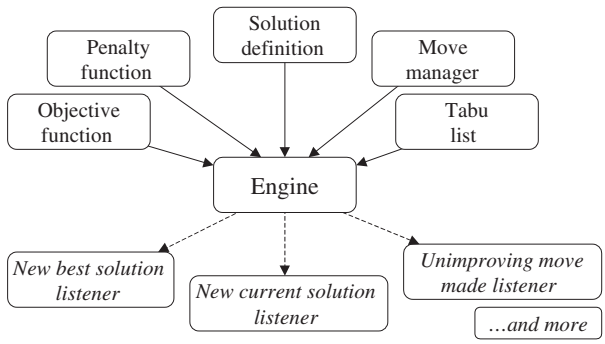


Fig. 15. General tabu search engine.

generate and evaluate with respect to the current solution. The best moves are then executed by the engine for that iteration, subject to the defined tabu list object restrictions. Moves selected are registered by the engine in the tabu list. A default aspiration criteria is implemented but more sophisticated aspiration criteria are easily implemented.

The listener capability exploited by the engine is a perfect vehicle for building specific tabu searches of varied complexity and applicability. Early examples include a large-scale goal programming problem (Cullenbine, 2000), a TSP (Hall, 2000), and a project scheduling application (Calhoun, 2000). A free version, complete with developer specifications, can be found at www.*i*Harder.net.

9 Empirical analysis

Any tabu search requires parameterization. An empirical analysis led to our tabu search parameterization and a general impression of algorithm performance when solution speed is favored over better solution quality. The Solomon data set was used for the testing (Solomon, 1987). As an aside, all that was required to enable the testing with the UAV router was a Solomon application in place of the UAV application since our architecture decouples application and algorithm efforts.

Two tabu list structures were tested: a static or fixed tabu list length and a reactive tabu list length. Our unreported results suggest use of a static tabu list length. The quality improvement obtained using the reactive list scheme was insufficient to warrant the extra processing time required. With an operational focus, like that in a UAV routing environment, processing speed is paramount. Thus, a static tabu list length is the default setting in the UVR.

Three tabu search parameters were tested:

- the default number of iterations for each starting solution;
- the number of consecutive starting solutions not improving on the best so far before quitting; and
- the maximum number of positions to consider when inserting an order in a tour.

A full, two-level design in all three parameters was examined using the Solomon data set (Solomon, 1987). Table 3 summarizes the parameter settings. Our testing revealed two insights.

Table 3
Parameter settings for adaptive tabu search test design

Parameter	Low setting (L)	High setting (H)
Default (minimum) iterations	1/4 Number of customers	Number of customers
Non-improving starts before quitting	10% Number of starting solutions	50% Number of starting solutions
Maximum places to insert	10% Tour length	50% Tour length

First, the level of solution effort (as measured by solve time) increased significantly at the higher parameter setting levels. Second, the additional processing effort did not significantly improve the solution quality. We therefore settled on low default settings for each parameter to focus on quickly returning good routing solutions. The immediate question is how much does solution quality suffer when an algorithm is focused and parameterized for processing speed.

Table 4 presents the solution results on the Solomon data set using the default parameter settings. The Best columns in Table 4 come from the compilation in Kinney, Hill and Moore (2001). Figure 16 depicts the gap between the general purpose UVR results (light area) and the best known results (dark area). The question is how large is this area.

Table 5 summarizes the average and range of our results versus the best for each of the six types of problem sets in the Solomon data set. The first two problem sets (C1xx and C2xx) contained clustered sites and are considered the easiest problems to solve. The randomly placed (R1xx and R2xx) and the randomly placed and clustered (RC1xx and RC2xx) problem sets are more difficult to solve. On average, the UVR does quite well on the clustered problems, which are more typical of UAV operational problems. Across all problems, the UVR and adaptive tabu search were approximately 16–17% above the best known solution. Again, our focus is on processing speed, so 16–17% above the best known solutions to academic problem sets can realistically be construed as a good solution for operational problems when returned very quickly. Improved solution performance is attainable at the expense of processing speed.

10 Conclusion and recommendations for further work

This research proposes a generic architecture for analytic applications. The architecture decouples the specifics of particular classes of problems from the techniques and mechanisms used to obtain solutions to these problems. The layers proposed and defined in this architecture provide specific functionality used by other layers in the architecture.

A prototype application based on this architecture was built using routing problems as the focus. Each layer of the architecture was populated with components appropriate to the layer. The application's focus was the routing of UAVs, an important operational issue within the Department of Defense. Such operational applications need quality solutions obtained very quickly; thus, we parameterized the solution technique to focus on solution speed. The graphical user interface built for the UAV operator provides a simple, Windows-like, spreadsheet-based environment. As demonstrated in our own empirical testing, interfaces are easily replaced with other, possibly more sophisticated interfaces. We claim our architecture approach decouples

Table 4
Performance of UVR versus best known solutions

	Number of vehicles		Distance traveled		Solve time
	UVR	Best	UVR	Best	
C101	10	10	852	827	69
C102	10	10	960	827	18
C103	10	10	923	826	188
C104	10	10	913	823	263
C105	10	10	860	827	80
C106	10	10	877	827	141
C107	10	10	894	827	113
C108	10	10	853	827	151
C109	10	10	854	827	240
C201	3	3	591	592	83
C202	3	3	676	592	179
C203	3	3	683	591	204
C204	3	3	656	591	259
C205	3	3	588	589	141
C206	3	3	633	588	172
C207	3	3	601	588	159
C208	3	3	629	588	163
R101	20	18	1805	1608	207
R102	19	17	1661	1434	251
R103	14	13	1587	1207	272
R104	11	9	1156	1007	243
R105	14	14	1517	1377	228
R106	13	12	1344	1252	213
R107	12	10	1247	1105	228
R108	10	9	1112	964	245
R109	13	11	1334	1206	251
R110	12	10	1248	1135	248
R111	11	10	1242	1097	223
R112	10	10	1148	954	232
R201	4	4	1544	1254	197
R202	4	3	1378	1214	254
R203	3	3	1210	949	268
R204	3	2	946	867	372
R205	3	3	1208	999	234
R206	3	3	1094	833	279
R207	3	3	1078	815	326
R208	2	2	989	739	407
R209	3	3	1157	855	293
R210	3	3	1232	963	258
R211	3	2	980	924	364
RC101	16	14	1802	1669	223
RC102	14	12	1698	1555	269
RC103	13	11	1502	1110	322
RC104	13	10	1502	1135	327
RC105	+16	13	1706	1643	285

Table 4. (Contd.)

	Number of vehicles		Distance traveled		Solve time
	UVR	Best	UVR	Best	
RC106	13	11	1478	1448	355
RC107	12	11	1434	1230	286
RC108	11	10	1228	1140	261
RC201	4	4	1810	1407	176
RC202	4	4	1542	1153	312
RC203	3	3	1484	1068	258
RC204	3	3	1113	804	336
RC205	4	4	1758	1302	228
RC206	4	3	1421	1156	214
RC207	4	3	1362	1075	239
RC208	3	3	1099	834	356

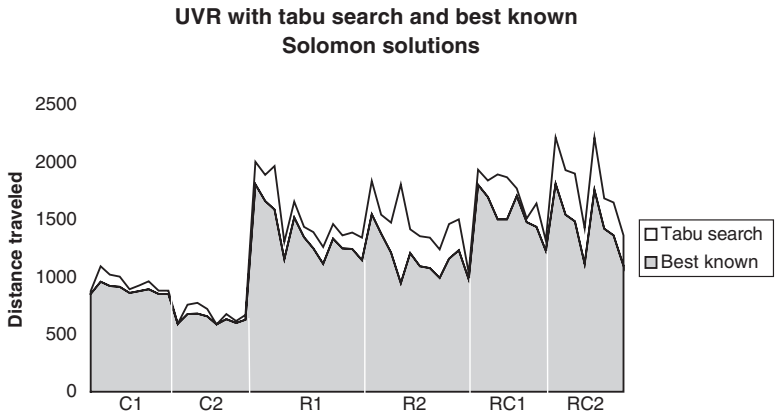


Fig. 16. Comparing UVR using tabu search and best known Solomon solutions.

Table 5
Statistics of percentage over best known solution by problem set

Problem Set	Minimum	Average	Maximum
C1xx	3.02	7.37	16.08
C2xx	—	7.16	15.57
R1xx	7.35	14.52	31.48
R2xx	6.06	23.72	35.32
RC1xx	2.07	14.38	35.32
RC2xx	22.92	32.02	38.95
All problems		Average 16–17%	

application and algorithm development. We realized this capability in building two applications, the UAV router application depicted in the paper and a Solomon data set interface not shown, while building a solver (the general tabu search) and implementing another (Kinney's heuristic).

Avenues for future work, both applied and theoretical, abound because of 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 a customer's existing suite of software. Theoretical advances can expand the suite of solvers for routing or refining the general tabu search objects to enhance specific tabu search implementations.

Acknowledgements

This research was funded by the United States Air Force Unmanned Aerial Vehicle Battlelab, under the able guidance of Dr Mark O'Hair. The views expressed in this article are those of the authors and do not reflect the official policy of the United States Air Force, Department of Defense, or the US Government.

References

Ackoff, R.L., 1979. The future of operational research is past. *Journal of the Operational Research Society*, 30.

Burdell, J., 2001. Personal communication and e-mail. February–July.

Calhoun, K., 2000. 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.

Cullenbine, C., 2000. 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.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13, 533–549.

Glover, F., Laguna, M., 1997. *Tabu search*. Kluwer Academic Publishers, Boston.

Hall, S., 2000. 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.

Kinney, G., Hill, R., Moore, J.T., 2001. A hybrid tabu search heuristic for the unmanned aerial vehicle (UAV) routing problem. Working Paper Series, WP01-03, Department of Operational Sciences, School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, August.

Laporte, G., 1992. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 345–358.

O'Rourke, K.P., Bailey, T.G., Hill, R.R., Carlton, W.B., 2001. Dynamic routing of unmanned aerial vehicles using reactive tabu search. *Military Operations Research*, 6, 1, 5–30.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35, 2, 254–265.