APPENDIX A:
Hybrid Automata Codes:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Brute Force Optimization Routine    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
clc

    N=40;               % Number of Nodes
    PackCost = [];
    %%%%%% WHAT DO YOU WANT TO OPTIMIZE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    OT = 2; %1 = time, 2 = Priority Cost, 3 = both
    if OT == 1,
        kiti = 500;
        error_tol = 1e-5;
    elseif OT == 2
        kiti = 1000;
        error_tol = 0.5;
    elseif OT == 3
        kiti = 1000;
        error_tol = 0.5;
    end


    %%% for 1 k =500, 2 k =700;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%% PACKAGES 1st row Pickup locations, 2nd row Delivery node, 3 rd
    %%%%% Priority rating [0 inf)
    Pickup0 = [26 22 35 19;40 5 13 12;50 10 50 3]; %Packages to Pickup
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%% UAV SPECS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Fuel0      = [500,600];
    CruiseV    = [700,600];
    ClimbV     = [500,600];
    fuel_burn  = [50,60];
    UAV0 =       [29,15];      % UAV Start Locations
    NUAV =    size(Fuel0,2);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%% OPTIMIZATION %%%%%%%%
for kkkk =1:10
    for kkk = 1:10
        for k=1:kiti;
            Tr=[];
            for pp=1:NUAV-1
                Tr=[Tr rand_gen(size(Pickup0,2))];
            end
            Tr=sort(Tr);
            Tr=[1 Tr];
            track =[];
            for p =1:NUAV
                if p ~= NUAV
```

```
                    Pickup01 = Pickup0(:,Tr(p):Tr(p+1)-1);
                else
                    Pickup01 = Pickup0(:,Tr(p):end);
                end
                pm = rand_gen(NUAV);
                if size(Pickup01,2) ~= 0
                    [x0,PackCost,Pickup] = Gen_Transistions(Pickup01);
                    [costi, UAV0(pm)] =
Hybrid_Auto(x0,N,PackCost,Pickup,UAV0(pm),Fuel0(pm),CruiseV(pm),ClimbV(pm),fu
el_burn(pm));
                    track = [track 0 pm 0 x0];
                else
                    costi = zeros(1,4);
                end
                if OT == 1
                    cost(p) = costi(OT);
                elseif OT == 2
                    cost(p) = costi(4);
                else
                    cost(p) = costi(1) + costi(4);
                end

            end
                cost_total(k) = max(cost);
            if k ==1
                costopt = cost_total(k);
                TrackOpt = track;
            else
                if costopt > cost_total(k),
                    costopt= cost_total(k);
                    TrackOpt = track;
                end
            end

        end
        costopt_stats(kkk) = costopt;
    end

    if std(costopt_stats) < error_tol,
        fprintf('Optimal Solution Found')
        disp(TrackOpt);
        disp(costopt);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%   HYBRID AUTOMATION   %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Cost,UAV_node] =
Hybrid_Auto(Route,N,PackCost,Pickup,UAV_Node,Fuel0,CruiseV,ClimbV,fuel_burn)

%%% Round estimations


%ASSUMPTIONS
        % Goal is to drive PackCost to zero in minimum time / Min Fuel, etc
```

```
        % Aircraft cannot refuel and unload at the same time


    %%% Loc = Locations of Nodes 1:N, [x;y]
    xx = [-86.9311530000000,-111.496583000000,-92.1606450000000,-
119.714356000000,-105.695802000000,-81.6137700000000,-83.5473640000000,-
114.968263000000,-89.2163090000000,-86.1401380000000,-93.6071780000000,-
98.3752450000000,-84.2028810000000,-92.3767090000000,-69.1955570000000,-
84.6589360000000,-94.4421390000000,-89.5642090000000,-92.3767090000000,-
109.449463000000,-99.5837410000000,-117.008057000000,-105.823975000000,-
74.5788580000000,-79.0832520000000,-100.352784000000,-82.5329590000000,-
97.5402840000000,-120.281983000000,-77.6770020000000,-81.0388190000000,-
100.220948000000,-86.1364750000000,-98.3093270000000,-111.514893000000,-
78.2702640000000,-120.172120000000,-80.8410650000000,-89.6740730000000,-
107.471924000000];
    yy =
[32.4838170000000,34.7885440000000,34.8606940000000,36.3257480000000,38.93548
50000000,28.1998630000000,32.7429300000000,44.2231590000000,40.2560540000000,
39.8524080000000,42.1031130000000,38.6563460000000,37.5846360000000,30.983261
0000000,45.2949840000000,43.4607510000000,46.3363090000000,32.7373860000000,3
8.5017520000000,47.0746110000000,41.6450050000000,39.8979370000000,34.5843800
000000,42.8646920000000,35.5197090000000,47.4919670000000,40.3013140000000,35
.6626520000000,43.9659350000000,40.9682850000000,34.0399150000000,44.42279600
00000,35.9833400000000,31.4155380000000,39.0669670000000,37.5498040000000,47.
4028110000000,38.6048520000000,44.5482030000000,43.0737030000000];
    loc = [xx; yy];
    %Refuel nodes first row is node name, second row is refuel rate
    %gallons per hour
    FuelNodes = [500*ones(1,N)];
    %Unload/Load rate hours/package
    UnloadRates = [ones(1,N)*10];

    %% Route order (to be Optimized)
    %Fuel0      = [500];
    %CruiseV    = [700];
    %ClimbV     = [500];
    %fuel_burn  = [50];      %g/hour5
    %UAV_Node   = UAV0;

    %% Time Varible
    t = 0;
    dist = 0;
    Pcost =0;
    x = [t,Fuel0,dist];

    % UAV FLIGHT POLCIY
    alpha = 10;         %degrees
    cruise_alt = 5;  %km
    Fuel_Safe  = 1.10;    %Fuel Safety margin.
    wind = 0;

%%%%%%%%%%%%%%% RUN MISSION %%%%%%%%%%%%%
 jj=1;
 flag = 1;
 while(flag ~= 0)
        %%% FOR UAV 1 %%%
        UAV = 1;
```

```
        dest = Route(jj);
        %%% COMPUTE Great Circle Distance to Target %%%
        lat1  =  deg2rad(loc(2,UAV_Node(UAV)));
        lat2  =  deg2rad(loc(2,dest));
        dlong =  deg2rad(loc(1,dest) - loc(1,UAV_Node(UAV)));
        distr =  2*6373*asin(sqrt(sin((lat1 - lat2)/2)^2 +
cos(lat1)*cos(lat2)*sin(dlong/2)^2));

        %%%% Check for required fuel estimate flight time.
            Estimate_t = (cruise_alt/tan(deg2rad(alpha)))*2/ClimbV + (distr -
(cruise_alt/tan(deg2rad(alpha)))*2)/CruiseV;  %%% use for ASTAR %%
            Estimate_Fuel = Estimate_t*fuel_burn*Fuel_Safe;

            %%% ADD code for case when not all nodes are fuel nodes
            if Estimate_Fuel >  x(2),
                [x,flag] = refuel(x,UAV_Node(UAV),Fuel0,UAV,FuelNodes);
            end

            if Estimate_Fuel >  x(2);
                fprintf('Fuel Tank Not Large Enough to Reach Destination');
                fail = 1;
                return
            end
            if Estimate_Fuel <=  x(2);

                [x,distr,flag] =
Climb(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn,wind);
                [x,distr,flag] =
Cruise(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn, wind, CruiseV);
                [x,distr,UAV_node(UAV),flag] =
Descend(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn, wind,dest);
            end
            [x,PackCost,flag,Pcost] =
unload(x,dest,PackCost,UnloadRates,Pcost);
            [x,PackCost,flag,Pickup] =
loadup(x,dest,PackCost,Pickup,UnloadRates);

            if (size(PackCost,2)) < 1 && (size(Pickup,2)) < 1
                %fprintf('All Packages Delivered\n');
                flag =0;
            end
            jj=jj+1;
end

Cost = [x,Pcost];



function [x,distr,flag] =
Climb(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn, wind);
        x(1) = x(1) + (cruise_alt/tan(deg2rad(alpha)))/ClimbV;
        x(3) = x(3) + (cruise_alt/tan(deg2rad(alpha)));
        x(2) = x(2) - fuel_burn*(cruise_alt/tan(deg2rad(alpha)))/ClimbV;
        distr = distr - (cruise_alt/tan(deg2rad(alpha)));
        if x(2) < 0
            fprintf('You ran out of fuel\n');
            flag =0;
```

```
            return
        end
        flag =1;
    end


function [x,distr,flag] =
Cruise(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn, wind, CruiseV);
        x(1) = x(1) + (distr)/CruiseV -
(cruise_alt/tan(deg2rad(alpha)))/ClimbV;
        x(3) = x(3) + (distr-(cruise_alt/tan(deg2rad(alpha))));

        x(2) = x(2) - fuel_burn*((distr)/CruiseV -
(cruise_alt/tan(deg2rad(alpha)))/ClimbV);
        distr = (cruise_alt/tan(deg2rad(alpha)));
        if x(2) < 0
            fprintf('You ran out of fuel\n');
            flag =0;
            return
       end
       flag =1;
    end

function [x,distr,UAV_node,flag] =
Descend(x,distr,UAV,cruise_alt,ClimbV,alpha,fuel_burn, wind, node);
         x(1) = x(1) + (cruise_alt/tan(deg2rad(alpha)))/ClimbV;
         x(3) = x(3) + (cruise_alt/tan(deg2rad(alpha)));
         x(2) = x(2) - fuel_burn*((cruise_alt/tan(deg2rad(alpha)))/ClimbV);
         distr = 0;
         UAV_node(UAV) = node;
         if x(2) < 0
            fprintf('You ran out of fuel\n');
            flag = 0;
            return
         end
        flag =1;
    end

    function [x,PackCost,flag,Pickup] =
loadup(x,node,PackCost,Pickup,UnloadRates)
        nopack=0;
        for i = 1:size(Pickup,2)
            if Pickup(1,i) == node,
                j = i;
                nopack =0;
                break
            else
                nopack =1;
            end
        end
        if (nopack ==1) || (size(Pickup,2) == 0)
            %fprintf('No Package to Pickup \n');
            flag =1;
            return
        end
        %fprintf('Package at %1.0f Picked up at node
%1.0f\n',Pickup(1,j),node);
```

```
        PackCost(1,end+1) = Pickup(2,j);
        PackCost(2,end) = Pickup(3,j);
        Pickup(:,j) = [];
        x(1) = x(1) + UnloadRates(j);
       flag =1;
    end



    function [x,PackCost,flag,Pcost] =
unload(x,node,PackCost,UnloadRates,Pcost)
        nopack = 0;
        for i = 1:size(PackCost,2)
            if PackCost(1,i) == node, j = i;
                nopack = 0;
                break
            else
                nopack = 1;
            end
        end

        if (nopack ==1) || (size(PackCost,2)==0)
            %fprintf('No package to Deliver \n');
            flag =1;
            return
        end
        %fprintf('Package for %1.0f delivered to node
%1.0f\n',PackCost(1,j),node);
        Pcost = Pcost + x(1)*PackCost(2,j);
        PackCost(:,j) = [];
        x(1) = x(1) + UnloadRates(j);
       flag =1;
    end


function [x,flag] = refuel(x,node,Fuel0,UAV,FuelNodes)
        delta = Fuel0(UAV) -  x(2);
        x(2)  = Fuel0(UAV);
        x(1)= x(1) + 1/FuelNodes(node)* delta;
        %fprintf('UAV %1.0f Filled with %1.0f gallons of Jet A-
1\n',UAV,delta);
        flag =1;
    end


function [x0,PackCost,Pickup] = Gen_Transistions(Pickup)


      PackCost = [];
      trans = [Pickup(1,:)];
      mm=size(trans,2)*2;
      x0=[];
      for iii =1:mm
        m = rand_gen(size(trans,2));
        x0 = [x0 trans(m)];
        for kk =1:size(Pickup,2)
```

```
            if trans(m) == Pickup(1,kk)
                trans=[trans Pickup(2,kk)];
            end
        end
        trans(m) = [];
    end


function m = rand_gen(mm);

m = round(rand(1)*(mm-1)+1);
```

APPENDIX B:
Search and Planning C code:

```
/*
This file is a somewhat altered combination of what used to be mysearch.c,
mysearch.h and mydomain.c all writen by Prof. Atkins at University of Michigan,
Aerospace Engineering Department. Used here as a part of final project for
AE 740.

The coordinates and transitions (the grid) is hardcoded into this file, so
every executable compiled from this file is "domain specific." In other words,
look into map_us.h to see the definition of transition_set[] and coords[] and
pick the one you need (comment/ uncomment).

Main changes to original file are found in main method and print_goal (as well
as some new global variables defined on lines 78-84.
*/

#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>


#define MYSEARCH_MAIN
//#include "mysearch.h"

/* Uncomment the next line to see additional search information */
//#define DEBUG


/* Generic main program */
/* This file contains generic search code information */

#ifndef  MYSEARCH
#define MYSEARCH

/* Domain-specific structure to define world state */
struct state {
  char location[20];  /* Map location */
  struct state *prev;        /* Links along state path */
};

/* Domain-specific structure to define transitions  */
struct transition {
  char start[20];
  char end[20];
  float distance;
};
```

```
/* New domain-specific structure for heuristic (h) computation */
struct latlon {
  char  city[20];
  float lat;
  float lon;
};
/* General structure to define each queue element */
struct node {
  struct state s;       /* Domain-specific state description */
  float g, h;           /* Path cost/heuristic values for this node (if used) */
  struct node *prev;
  struct node *next;
};



/* Global variables:  "Closed list" / list of visited nodes */
#ifdef MYSEARCH_MAIN
  struct node *visit_list=NULL;
#else
  extern struct node *visit_list;
#endif


//////////////////////////New definitions///////////////////////////////////
#include "map_us.h"
char strr[100];    //string used to store the itenerary (maybe need longer than 100?
double doubc;      //double to store the cost
char START_CITY[4];//string to store the departure node
char GOAL_CITY[4]; //string to store the goal node
////////////////////////////////////////////////////////////////////////////

/* Domain-specific function prototypes */

void initialize_problem(struct node *n);
int goal_test(struct state s);
float goal_distance(struct node *n);
void print_node(struct node *n);
void print_goal(struct node *n);
int visited(struct node *n);
struct node *expand(struct node **n);

/* General search function prototypes */

int general_search(char *stype, struct node *queue);
struct node *depth_first_queue(struct node *old_queue, struct node *neww);
struct node *breadth_first_queue(struct node *old_queue, struct node *neww);
struct node *uniform_cost_queue(struct node *old_queue, struct node *neww);
struct node *greedy_queue(struct node *old_queue, struct node *neww);
struct node *astar_queue(struct node *old_queue, struct node *neww);

#endif
```

```
void initialize_problem(struct node *n)
{
  strcpy(n->s.location,START_CITY);    /* Initial state */
  n->g = 0.;
  n->h = goal_distance(n);
  n->s.prev = NULL;
  n->next = NULL;
  n->prev = NULL;
}

int goal_test(struct state s)
{
  if (!strncmp(s.location,GOAL_CITY,3))   /* Goal destination found */
    return(1);
  return(0);
}

/* Function to compute straight-line distance to the goal (h)
  from current node *n (complete for Problem 2a*/
#define R 6373.0  /* Radius of the Earth in miles */
float goal_distance(struct node *n)
{

float lat1;
float lon1;
float lat2;
float lon2;
float c;
float dist;
int i;

for (i=0;i<NUMCITIES;i++) //Look up the coordinates of the goal city
{
        if (!strncmp(coords[i].city,GOAL_CITY,3))
        {
        lat1=coords[i].lat*M_PI/180;
        lon1=coords[i].lon*M_PI/180;
        }
}

for (i=0;i<NUMCITIES;i++) //Look up the coordinates of the current city
{
        if (!strncmp(coords[i].city,n->s.location,3))
        {
        lat2=coords[i].lat*M_PI/180;
        lon2=coords[i].lon*M_PI/180;
        }
}

c=acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*cos(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
//Calculate the angle between two vectors (OA and OB) as shown at
//http://mathforum.org/library/drmath/view/51722.html

dist=R*c; //Great circle distance (arc-length) is just radius times angle (in rad)

  return(dist);
```

```
}

void print_node(struct node *n)
{
   printf(" %s (g = %5.1f; h = %5.1f)\n", n->s.location, n->g, n->h);
}

void print_goal(struct node *n)
{
  struct state *s;
  s = &(n->s);
  //printf("Total path cost (g) = %5.1f\n", n->g);
  //printf("Goal ");

    doubc= n->g;
  while (s) {

    printf(s->location);//" <-- %s\n",  "%s,",

   //Save the route for later output to a tect file
    strcat(strr,s->location);

    s = s->prev;
  }
}

int visited(struct node *n)
{
  struct node *v;
  v = visit_list;
  while (v) {
    if (!strcmp(v->s.location, n->s.location))
      return(1);  /* 1 = visited */
    v = v->next;
  }
  return(0);  /* 0 = not visited */
}

/* Generate linked list of "child" states created when expanding parent */

struct node *expand(struct node **q)
{
  int i, match;
  struct node *n, *parent, *first_child=NULL, *child;

  /* Remove parent from queue; place on "visit_list" list */
  parent = *q;
  *q = (*q)->next;
  if (*q) (*q)->prev=NULL;
  if (visit_list) {
    visit_list->prev = parent; parent->next = visit_list; visit_list = parent;
  } else {
    visit_list = parent; visit_list->next = NULL; visit_list->prev = NULL;
  }

  for (i=0;i<NTRANSITIONS;i++) {
```

```
    /* Note:  For code simplicity, left out malloc error checking... */
    if ((!strcmp(transition_set[i].start,parent->s.location)) ||
          (!strcmp(transition_set[i].end,parent->s.location))) {

      n = (struct node *) malloc(sizeof(struct node));
      n->s.prev = &(parent->s);
      if (!strcmp(transition_set[i].start,parent->s.location))
          strcpy(n->s.location,transition_set[i].end);
      else
          strcpy(n->s.location,transition_set[i].start);

      if (visited(n))
          free(n);
      else {
          if (!first_child) {
            first_child = n;
            child = first_child;
            child->prev = NULL;
          } else {
            child->next = n;
            child->next->prev = child;
            child = child->next;
          }
          child->g = parent->g + transition_set[i].distance;  /* Path cost g */
          child->h = goal_distance(child); /* Heuristic function h */
          child->next = NULL;
      }
    }
  }
  return(first_child);
}


int main()
{

  //Read the start node from the text file
  ifstream start_file ( "start.txt" );
  start_file>> START_CITY;
  start_file.close();

  //Read the goal node from the text file
  ifstream finish_file ( "finish.txt" );
  finish_file>> GOAL_CITY;
  finish_file.close();

  //char s[80];
  struct node top_node;
  initialize_problem(&top_node);

  /* Pass in string corresponding to search choice */
  //printf("Enter search choice (DEPTH, BREADTH, UCOST, GREEDY, ASTAR):\n");
  //scanf("%s",s);
  //Always use A* - maybe will read it from a text file in the future
  if (general_search("ASTAR", &top_node))
  {
```

```
    printf("No solution found.\n");
}
  // Output the sequence of nodes the UAV should follow to a txt file
  ofstream it_file ( "itenerary.txt", ios::trunc);
  it_file<<strr;
  it_file.close();

  // Also, output the cost inquired
  ofstream c_file ( "cost.txt", ios::trunc);
  c_file<<doubc;
  c_file.close();


}

int general_search(char *stype, struct node *queue)
{
  struct node *children, *new_queue, *q;
  if (!queue) return(-1);  /* No more nodes in queue; goal not found */

  if (goal_test(queue->s)) {  /* Goal found! */
    print_goal(queue);
    return(0);
  }

#ifdef DEBUG
  printf("Currently expanding:\n");
  print_node(queue);
#endif

  children = expand(&queue);

#ifdef DEBUG
  printf("Queue:  \n");
  q=queue;
  while (q) {
    print_node(q);
    q = q->next;
  }
  printf("Child nodes of current:\n");
  q=children;
  while (q) {
    print_node(q);
    q = q->next;
  }
#endif

  if (!strncmp(stype,"DEPTH",5))
    new_queue = depth_first_queue(queue, children);

  else if (!strncmp(stype,"BREADTH",7))
    new_queue = breadth_first_queue(queue, children);

  else if (!strncmp(stype,"UCOST",5))
    new_queue = uniform_cost_queue(queue, children);
```

```
  else if (!strncmp(stype,"GREEDY",6))
    new_queue = greedy_queue(queue, children);

  else if (!strncmp(stype,"ASTAR",5))
    new_queue = astar_queue(queue, children);

#ifdef DEBUG
  printf("New queue:\n");
  q=new_queue;
  while(q) {
    print_node(q);
    q = q->next;
  }
#endif

  if (general_search(stype,new_queue) == -1) return(-1);
  return(0);
}

struct node *depth_first_queue(struct node *queue, struct node *children)
{
  struct node *qptr, *cptr;
  qptr = queue; cptr = children;
  if (!qptr) return(cptr);

  /* Add new child nodes to head of queue */
  while (cptr) {
    if (qptr)
      qptr->prev = cptr;
    cptr = cptr->next;
    qptr->prev->next = qptr;
    qptr = qptr->prev;
  }
  qptr->prev = NULL;
  return(qptr);
}

struct node *breadth_first_queue(struct node *queue, struct node *children)
{
  struct node *qptr, *cptr;
  qptr = queue; cptr = children;
  if (!qptr) return(cptr);

  /* Add new child nodes to tail of queue */
  while (qptr->next) qptr = qptr->next;
  while (cptr) {
    qptr->next = cptr;
    cptr = cptr->next;
    qptr->next->prev = qptr;
    qptr = qptr->next;
  }
  qptr->next = NULL;
  return(queue);
}

struct node *uniform_cost_queue(struct node *queue, struct node *children)
```

```
{
  float g;
  struct node *next_child, *qptr, *cptr;

  if ((!queue) && (!children)) return(NULL);
  if ((!queue) && children) {  /* Insert first child onto an empty queue */
    queue = children;
    children = children->next;
    queue->next = NULL;
  }
  cptr = children;

  /* Insert each new child node at its proper position */
  while (cptr) {
    g = cptr->g;
    qptr = queue;
    while (qptr->next && (g > qptr->g)) qptr = qptr->next;
    if (g > qptr->g) {  /* Child is inserted at end of queue */
      qptr->next = cptr;
      cptr->prev = qptr;
      cptr = cptr->next;
      qptr->next->next = NULL;
    } else { /* Child is inserted at beginning or in middle of queue */
      next_child = cptr->next;
      if (qptr->prev) qptr->prev->next = cptr;
      cptr->prev = qptr->prev;
      qptr->prev = cptr;
      cptr->next = qptr;
      cptr = next_child;
      if (queue == qptr) queue = qptr->prev;  /* Reset queue beginning */
    }
  }
  return(queue);
}

struct node *greedy_queue(struct node *queue, struct node *children)
{
  float h;
  struct node *next_child, *qptr, *cptr;

  if ((!queue) && (!children)) return(NULL);
  if ((!queue) && children) {  /* Insert first child onto an empty queue */
    queue = children;
    children = children->next;
    queue->next = NULL;
  }
  cptr = children;

  /* Insert each new child node at its proper position */
  while (cptr) {
    h = cptr->h;
    qptr = queue;
    while (qptr->next && (h > qptr->h)) qptr = qptr->next;
    if (h > qptr->h) {  /* Child is inserted at end of queue */
      qptr->next = cptr;
      cptr->prev = qptr;
```

```
      cptr = cptr->next;
      qptr->next->next = NULL;
    } else { /* Child is inserted at beginning or in middle of queue */
      next_child = cptr->next;
      if (qptr->prev) qptr->prev->next = cptr;
      cptr->prev = qptr->prev;
      qptr->prev = cptr;
      cptr->next = qptr;
      cptr = next_child;
      if (queue == qptr) queue = qptr->prev;  /* Reset queue beginning */
    }
  }
  return(queue);
}

/* Complete astar_queue function for homework #6 */
struct node *astar_queue(struct node *queue, struct node *children)
{
  float h;
  float g;
  struct node *next_child, *qptr, *cptr;

  if ((!queue) && (!children)) return(NULL);
  if ((!queue) && children) {  /* Insert first child onto an empty queue */
    queue = children;
    children = children->next;
    queue->next = NULL;
  }
  cptr = children;

  /* Insert each new child node at its proper position */
  while (cptr) {
    g = cptr->g;
    h = cptr->h;
    qptr = queue;
    while (qptr->next && (h+g > qptr->h + qptr->g)) qptr = qptr->next;
    if (h+g > qptr->h + qptr->g) {  /* Child is inserted at end of queue */
      qptr->next = cptr;
      cptr->prev = qptr;
      cptr = cptr->next;
      qptr->next->next = NULL;
    } else { /* Child is inserted at beginning or in middle of queue */
      next_child = cptr->next;
      if (qptr->prev) qptr->prev->next = cptr;
      cptr->prev = qptr->prev;
      qptr->prev = cptr;
      cptr->next = qptr;
      cptr = next_child;
      if (queue == qptr) queue = qptr->prev;  /* Reset queue beginning */
    }
  }
  return(queue);
}
```

Search and Planning MATLAB code:

```
%% UAV package delivery
%AE 740, Final Project, Fall 2009
%Alexey Morozov, 12/14/09, morozova [at] umich [dot] edu

%An extension of the rooms.m file (single uav search on square grid)

%Files required for this file to run properly (all should be in the same
%directory):
%   -astar.m (needs mysearch.exe - precompiled from mysearch.c)
%   -rmvalue.m
%   -iten2line.m
%   -substr.m (I didn't write this one, but I hope it's ok if I use it
%   here. Taken from pja:
%   http://home.online.no/~pjacklam/matlab/software/util/strutil/substr.m)


%% Setup the problem
clear
clc

%city=struct('name',{'AL','AZ','AR','CA','CO','FL','GA','ID','IL','IN','IA','
KS','KY','LA','ME','MI','MN','MS','MO','MT','NE','NV','NM','NY','NC','ND','OH
','OK','OR','PA','SC','SD','TN','TX','UT','VA','WA','WV','WI','WY'},...
%    'x',{5,7,1,1,1,1,3,3,3,3},'y',{3.5,3.5,5.5,4.5,3,1,5.5,4.5,3,1});
nn_c=['AL';'AZ';'AR';'CA';'CO';'FL';'GA';'ID';'IL';'IN';'IA';'KS';'KY';'LA';'
ME';'MI';'MN';'MS';'MO';'MT';'NE';'NV';'NM';'NY';'NC';'ND';'OH';'OK';'OR';'PA
';'SC';'SD';'TN';'TX';'UT';'VA';'WA';'WV';'WI';'WY'];
nn=['01';'02';'03';'04';'05';'06';'07';'08';'09';'10';'11';'12';'13';'14';'15
';'16';'17';'18';'19';'20';'21';'22';'23';'24';'25';'26';'27';'28';'29';'30';
'31';'32';'33';'34';'35';'36';'37';'38';'39';'40'];
xx=[-86.931153  -111.496583 -92.160645  -119.714356 -105.695802 -81.61377   -
83.547364  -114.968263 -89.216309  -86.140138  -93.607178  -98.375245  -
84.202881  -92.376709  -69.195557  -84.658936  -94.442139  -89.564209  -
92.376709  -109.449463 -99.583741  -117.008057 -105.823975 -74.578858  -
79.083252  -100.352784 -82.532959  -97.540284  -120.281983 -77.677002  -
81.038819  -100.220948 -86.136475  -98.309327  -111.514893 -78.270264  -
120.17212  -80.841065  -89.674073  -107.471924];
%longitude

yy=[32.483817   34.788544   34.860694   36.325748   38.935485   28.199863
32.74293    44.223159   40.256054   39.852408   42.103113   38.656346
37.584636   30.983261   45.294984   43.460751   46.336309   32.737386
38.501752   47.074611   41.645005   39.897937   34.58438    42.864692
35.519709   47.491967   40.301314   35.662652   43.965935   40.968285
34.039915   44.422796   35.98334    31.415538   39.066967   37.549804
47.402811   38.604852   44.548203   43.073703];
%latitude

uav_s='29';
%package=struct('name',{'P1','P2','P3'},'pickup_l',{'40','34','15'},'deliver_
l',{'23','25','16'});
```

```
package=struct('name',{'P1','P2','P3','P4'},'pickup_l',{'22','35','19','23'},
'deliver_l',{'05','13','12','34'},'ps',{0},'ds',{0},'pc',{0},'dc',{0});
package_cp=package; %just a copy of package for later

coords_m=[xx' yy'];

for i=1:length(xx)
    %if i<10; nn(i,:)=strcat('a0',num2str(i));  else
nn(i,:)=strcat('a',num2str(i)); end;
    city(i).name=nn(i,:);
    city(i).name_c=nn_c(i,:);
    city(i).x=xx(i);
    city(i).y=yy(i);
end
num_cities=length(city);



%
package=struct('name',{'P1','P2','P3'},'pickup_l',{'01','05','06'},'deliver_l
',{'10','08','05'});
% package_cp=package; %just a copy of package for later
% uav_s='03';



%city=struct('name',{'R01','R02','R03','R04','R05','R06','R07','R08','R09','R
10'},...
%    'x',{5,7,1,1,1,1,3,3,3,3},'y',{3.5,3.5,5.5,4.5,3,1,5.5,4.5,3,1});

% city=struct('name',{'r01','r02','r03','r04','r05'},'x',{0,1,2,3,4},'y',{0
% ,0,0,0,0});

cc=zeros(num_cities,num_cities);
cc2=[1,1,1,1,1,1,2,2,2,3,3,3,3,3,4,4,5,5,5,5,5,5,6,6,7,8,8,8,8,8,9,9,9,9,9,9,
10,10,10,10,10,11,11,11,11,11,12,12,12,13,13,13,13,13,14,14,15,15,16,16,16,16
,17,17,17,18,18,19,19,20,20,20,20,21,21,22,22,22,23,23,23,24,25,25,25,26,27,2
7,28,29,30,30,31,32,35,36;6,7,14,18,31,33,4,23,35,18,19,28,33,34,22,35,12,21,
23,28,35,40,7,14,31,20,22,29,37,40,10,11,12,19,16,39,19,13,16,27,38,12,21,32,
17,39,21,19,28,19,38,36,25,33,18,34,16,24,27,39,30,24,26,32,39,33,34,28,33,37
,40,32,26,32,40,29,35,40,28,34,35,30,31,36,33,32,30,38,34,37,38,36,33,40,40,3
8];
for i=1:length(cc2)
    cc(cc2(1,i),cc2(2,i))=cc(cc2(1,i),cc2(2,i))+ 1;
end
cc=cc+cc';
max(cc(:))
%% service routines for STATES problem
%{

%     cc(1,6)=1;%connected cities
%     cc(1,7)=1;
%     cc(1,14)=1;
%     cc(1,18)=1;
%     cc(1,31)=1;
```

```
%     cc(1,33)=1;
%     cc(2,4)=1;
%     cc(2,23)=1;
%     cc(2,35)=1;
%     cc(3,18)=1;
%     cc(3,19)=1;
%     cc(3,28)=1;
%     cc(3,33)=1;
%     cc(3,34)=1;
%     cc(4,22)=1;
%     cc(4,35)=1;
%     cc(5,12)=1;
%     cc(5,21)=1;
%     cc(5,23)=1;
%     cc(5,28)=1;
%     cc(5,35)=1;
%     cc(5,40)=1;
%     cc(6,7)=1;
%     cc(6,14)=1;
%     cc(7,31)=1;
%     cc(8,20)=1;
%     cc(8,22)=1;
%     cc(8,29)=1;
%     cc(8,37)=1;
%     cc(8,40)=1;
%     cc(9,10)=1;
%     cc(9,11)=1;
%     cc(9,12)=1;
%     cc(9,19)=1;
%     cc(9,16)=1;
%     cc(9,39)=1;
%     cc(10,19)=1;
%     cc(10,13)=1;
%     cc(10,16)=1;
%     cc(10,27)=1;
%     cc(10,38)=1;
%     cc(11,12)=1;
%     cc(11,21)=1;
%     cc(11,32)=1;
%     cc(11,17)=1;
%     cc(11,39)=1;
%     cc(12,21)=1;
%     cc(12,19)=1;
%     cc(12,28)=1;
%     cc(13,19)=1;
%     cc(13,38)=1;
%     cc(13,36)=1;
%     cc(13,25)=1;
%     cc(13,33)=1;
%     cc(14,18)=1;
%     cc(14,34)=1;
%     cc(15,16)=1;
%     cc(15,24)=1;
%     cc(16,27)=1;
%     cc(16,39)=1;
%     cc(16,30)=1;
%     cc(16,24)=1;
```

```
%     cc(17,26)=1;
%     cc(17,32)=1;
%     cc(17,39)=1;
%     cc(18,33)=1;
%     cc(18,34)=1;
%     cc(19,28)=1;
%     cc(19,33)=1;
%     cc(20,37)=1;
%     cc(20,40)=1;
%     cc(20,32)=1;
%     cc(20,26)=1;
%     cc(21,32)=1;
%     cc(21,40)=1;
%     cc(22,29)=1;
%     cc(22,35)=1;
%     cc(22,40)=1;
%     cc(23,28)=1;
%     cc(23,34)=1;
%     cc(23,35)=1;
%     cc(24,30)=1;
%     cc(25,31)=1;
%     cc(25,36)=1;
%     cc(25,33)=1;
%     cc(26,32)=1;
%     cc(27,30)=1;
%     cc(27,38)=1;
%     cc(28,34)=1;
%     cc(29,37)=1;
%     cc(30,38)=1;
%     cc(30,36)=1;
%     cc(31,33)=1;
%     cc(32,40)=1;
%     cc(35,40)=1;
%     cc(36,38)=1;

trans='{';%h(n)=great circle dist
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;


dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
        end
    end
end
```

```
trans = substr(trans,0,-1);
trans = strcat(trans,'};');
q=0;
trans_g='{';%g(n)=great circle dist + 20
for i=1:(num_cities)
    for j=(1):num_cities
        if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;


dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            dist=dist+20;
            trans_g =
strcat(trans_g,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',
');
        q=q+1;
        end
    end
end
trans_g = substr(trans_g,0,-1);
trans_g = strcat(trans_g,'};');
q
%coords_m=zeros(num_cities,2);
coords='{';%Just coordinates
for i=1:num_cities
    %coords_m(i,:)=[city(i).x city(i).y];
        coords =
strcat(coords,'{"',city(i).name,'",',num2str(city(i).y),',',num2str(city(i).x
),'}',',');
end
coords = substr(coords,0,-1);
coords = strcat(coords,'};');
%}


%% assuming all cities are connected
%{
q=0;
trans='{';%h(n)=great circle dist
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        if cc(i,j)
%             lat1=city(i).y*pi/180;
%             lon1=city(i).x*pi/180;
%
%             lat2=city(j).y*pi/180;
%             lon2=city(j).x*pi/180;
%
```

```
%
dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
%             %Great circle distance (arc-length) is just radius times angle
(in rad)
%             trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
        q=q+1;
        end

    end
end
trans = substr(trans,0,-1);
trans = strcat(trans,'};');
q
q=0;
trans_g='{';%g(n)=great circle dist + 20
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        %if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;


dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            dist=dist+20;
            trans_g =
strcat(trans_g,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',
');
       q=q+1;
        %end
    end
end
trans_g = substr(trans_g,0,-1);
trans_g = strcat(trans_g,'};');

q
%}
%% service routines for room problem
%{
trans='{';%straight-line dist
for i=1:(num_cities-1)

    for j=(i+1):num_cities
        dist=sqrt((city(i).x-city(j).x)^2+(city(i).y-city(j).y)^2);
        trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
    end
```

```
end
trans=substr(trans,0,-1);
trans = strcat(trans,'};');

transM='{';%Manhattan dist
for i=1:(num_cities-1)

    for j=(i+1):num_cities
        dist=abs(city(i).x-city(j).x)+abs(city(i).y-city(j).y);
        transM =
strcat(transM,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',','
);
    end
end
transM=substr(transM,0,-1);
transM = strcat(transM,'};');

coords_m=zeros(num_cities,2);
coords='{';%Just coordinates
for i=1:num_cities
    coords_m(i,:)=[city(i).x city(i).y];
        coords =
strcat(coords,'{"',city(i).name,'",',num2str(city(i).x),',',num2str(city(i).y
),'}',',');
end
coords=substr(coords,0,-1);
coords = strcat(coords,'};');
%}

%% Find the solution using "pick up and deliver cheapest package" method
%{
% mov = avifile('sim.avi','compression','Cinepak');
% for o=1:9
% iten_m=astar('03',strcat('0',num2str(o)));

%iten_m=astar('03','09');
    %[iten_m1,cost_p]=astar(uav_s,package(1).pickup_l);
    %[iten_m2,cost_d]=astar(package(1).pickup_l,package(1).deliver_l);
%
uav_c=uav_s;    %initially uav is located at uav_s (Start node). uav_c -
Current uav node
k=1;            %counter for # of delivered packages
iten_p=struct('iten',{});
iten_d=struct('iten',{});

while ~isempty(package)    %while packages are not delivered
                                                %deliver packages

    cost_p=[];  %the pick up cost row (cost to pick up each package in
current queue)
    cost_d=[];  %the deliver cost row (cost to deliver each package)
    cost=[];    %the cost to pick up and deliver the current package

    for i=1:length(package) %iterate through all packages and find p+d costs
            [iten_pg,cost_p(i)]=astar(uav_c,package(i).pickup_l);
```

```
[iten_dg,cost_d(i)]=astar(package(i).pickup_l,package(i).deliver_l);
    end

    cost=cost_p+cost_d;
    fst_p=find(cost==min(cost),1,'first');
    package_c=package(fst_p);

    [iten_p(k).iten,cost_pp(k)]=astar(uav_c,package_c.pickup_l);
    uav_c=package_c.pickup_l;
    [iten_d(k).iten,cost_dd(k)]=astar(uav_c,package_c.deliver_l);
    uav_c=package_c.deliver_l;

    package=rmvalue(package,fst_p);%if package has been delivered, take it
    %out of the que (rmvalue is a custom Mfile that gets rid of values in a
    %struct

    k=k+1; %one more package was delivered
end
%}
%% Find the solution using the "blind optimal method"
% mov = avifile('sim.avi','compression','Cinepak');
% for o=1:9
% iten_m=astar('03',strcat('0',num2str(o)));
%
uav_c=uav_s;    %initially uav is located at uav_s (Start node). uav_c -
Current uav node
iten_p=struct('iten',{});
iten_d=struct('iten',{});

k1=1;
k2=1;

flag=0;
while ~flag    %while packages are not delivered
    for i=1:length(package)     %calculate all the current p and d costs
        [iten_pg,package(i).pc]=astar(uav_c,package(i).pickup_l);
        [iten_dg,package(i).dc]=astar(uav_c,package(i).deliver_l);
    end

    cost=[];
    actions=[];
    for i=1:length(package) %got through all the packages
      if (~package(i).ps && ~package(i).ds)     %if package has not been
delivered yet
        cost=[cost package(i).pc];
        actions=[actions 0];
      elseif (package(i).ps && ~package(i).ds)  %if package has not been
picked up yet
        cost=[cost package(i).dc];
        actions=[actions 1];
      else %the package was already picked up and delivered - so don't go
there!
          cost=[cost 10^7];
          actions=[actions -1];
```

```
      end
    end

    fst=find(cost==min(cost),1,'first');
    package_c=package(fst);
    if (actions(fst)==0)
        [iten_p(k1).iten,cost_pp(k1)]=astar(uav_c,package_c.pickup_l);
        uav_c=package_c.pickup_l;
        k1=k1+1;
            package(fst).ps=1;
    elseif (actions(fst)==1)
        [iten_d(k2).iten,cost_dd(k2)]=astar(uav_c,package_c.deliver_l);
        uav_c=package_c.deliver_l;
        k2=k2+1;
            package(fst).ds=1;
    else
        printf('Hello! Something really bad just happened!')  %This should
never happen
    end

    sum=0;
    for i=1:length(package)
        sum=sum+package(i).ds;
        if sum==length(package);
            flag=1;
        end
    end

end

%}
%% Plot the map and solution using the "some nodes are connected assumption

%path_m=[3 4 5 6 10 9 1 2 1 8 7 8 4];
%[trans_x,trans_y]=iten2line(path_m,coords_m);

trip_xp=struct('coord',{});
trip_yp=struct('coord',{});
trip_xd=struct('coord',{});
trip_yd=struct('coord',{});

%almost me_ex
%  iten_p=struct('iten',{[29,22],[22,35],[05,12,19]});
%  iten_d=struct('iten',{[35,05],[19,12],[12,19,13]});
%almost_all
%  iten_p=struct('iten',{[29,22],[22,35],[05,19]});
%  iten_d=struct('iten',{[35,05],[19,12],[12,13]});
%opt
% iten_p=struct('iten',{[29,22],[22,19],[19,35]});
% iten_d=struct('iten',{[35,13],[13,12],[12,05]});

for k=1:length(iten_p)

    [trip_xp(k).coord, trip_yp(k).coord]= iten2line(iten_p(k).iten,coords_m);
    [trip_xd(k).coord, trip_yd(k).coord]= iten2line(iten_d(k).iten,coords_m);
```

```
end

sca=6;

h=figure(1);clf
plot(coords_m(:,1),coords_m(:,2),'bo')
hold on;
for i=1:length(city); text(city(i).x+.3,city(i).y+.3,city(i).name_c); end
%.15 for room
%line(trans_x,trans_y,'LineStyle',':');



for k=1:length(iten_p)
    line(trip_xp(k).coord+k/sca, trip_yp(k).coord+k/sca, 'color','g'
,'LineWidth',2);
    line(trip_xd(k).coord+(k+1)/sca, trip_yd(k).coord+(k+1)/sca, 'color','r'
,'LineWidth',2);
end

legend('states','pickup','deliver')%,'pickup 2','deliver 2','pickup
3','deliver 3')

for i=1:(length(city)-1)
    for j=(i+1):length(city)
        if cc(i,j)
            line([city(i).x city(j).x], [city(i).y city(j).y],
'LineStyle',':');
        end
    end
end

for i=1:length(package_cp); text(city(str2double(package_cp(i).pickup_l)).x-
.5, city(str2double(package_cp(i).pickup_l)).y-.5, package_cp(i).name,
'color', 'm', 'FontSize', 15); end %.15 for room
for i=1:length(package_cp); text(city(str2double(package_cp(i).deliver_l)).x-
.5, city(str2double(package_cp(i).deliver_l)).y-.5, strcat('D',num2str(i)),
'color', 'b', 'FontSize', 15); end %.15 for room


hold off
xlabel('x, longitude (deg)');
ylabel('y, latitude (deg)');

% for i=1:10
% mov = addframe(mov,getframe);
%   %F(o)=getframe(h);
% end
% end
% mov = close(mov);
%imwrite(A,'sim','gif','DelayTime',.01)
%movie(F)

%% Plot the map and solution using the "all-nodes connected" assumption
```

```
%{
trip_xp=struct('coord',{});
trip_yp=struct('coord',{});
trip_xd=struct('coord',{});
trip_yd=struct('coord',{});

for k=1:length(iten_p)
    [trip_xp(k).coord, trip_yp(k).coord]= iten2line(iten_p(k).iten,coords_m);
    [trip_xd(k).coord, trip_yd(k).coord]= iten2line(iten_d(k).iten,coords_m);
end

sca=6;

h=figure(1);clf
plot(coords_m(:,1),coords_m(:,2),'bo')
hold on;
for i=1:length(city); text(city(i).x+.3,city(i).y+.3,city(i).name_c); end
%.15 for room
for i=1:length(package_cp); text(city(str2double(package_cp(i).pickup_l)).x-
.5, city(str2double(package_cp(i).pickup_l)).y-.5, package_cp(i).name,
'color', 'm', 'FontSize', 15); end %.15 for room
for i=1:length(package_cp); text(city(str2double(package_cp(i).deliver_l)).x-
.5, city(str2double(package_cp(i).deliver_l)).y-.5, strcat('D',num2str(i)),
'color', 'b', 'FontSize', 15); end %.15 for room

for k=1:length(iten_p)
    line(trip_xp(k).coord, trip_yp(k).coord, 'color','g' ,'LineWidth',2);
    line(trip_xd(k).coord, trip_yd(k).coord, 'color','r' ,'LineWidth',2);
end
legend('cities','pickup','deliver')%,'pickup','deliver','pickup','deliver')

hold off

% for i=1:10
% mov = addframe(mov,getframe);
%   %F(o)=getframe(h);
% end
% end
% mov = close(mov);
%imwrite(A,'sim','gif','DelayTime',.01)
%movie(F)
%}
%% route vector
clc
p=[];
p2=[];
for i=1:length(iten_p)
    p=[p iten_p(i).iten iten_d(i).iten];
end

k=1;
for i=1:(length(p)-1)
    if p(i)~=p(i+1)
        p2(k)=p(i);
        k=k+1;
```

```
    end
end
p
p2

%% 2 UAVs Example
%AE 740, Final Project, Fall 2009
%Alexey Morozov, 12/14/09, morozova [at] umich [dot] edu

%An extension of the main.m file (single uav search) to two uavs

%Files required for this file to run properly (all should be in the same
%directory):
%   -astar.m (needs mysearch.exe - precompiled from mysearch.c)
%   -rmvalue.m
%   -iten2line.m
%   -substr.m (I didn't write this one, but I hope it's ok if I use it
%   here. Taken from pja:
%   http://home.online.no/~pjacklam/matlab/software/util/strutil/substr.m)

%% Setup the problem
clear
clc

nn_c=['AL';'AZ';'AR';'CA';'CO';'FL';'GA';'ID';'IL';'IN';'IA';'KS';'KY';'LA';'
ME';'MI';'MN';'MS';'MO';'MT';'NE';'NV';'NM';'NY';'NC';'ND';'OH';'OK';'OR';'PA
';'SC';'SD';'TN';'TX';'UT';'VA';'WA';'WV';'WI';'WY'];
nn=['01';'02';'03';'04';'05';'06';'07';'08';'09';'10';'11';'12';'13';'14';'15
';'16';'17';'18';'19';'20';'21';'22';'23';'24';'25';'26';'27';'28';'29';'30';
'31';'32';'33';'34';'35';'36';'37';'38';'39';'40'];
xx=[-86.931153  -111.496583 -92.160645  -119.714356 -105.695802 -81.61377   -
83.547364  -114.968263 -89.216309  -86.140138  -93.607178  -98.375245  -
84.202881  -92.376709  -69.195557  -84.658936  -94.442139  -89.564209  -
92.376709  -109.449463 -99.583741  -117.008057 -105.823975 -74.578858  -
79.083252  -100.352784 -82.532959  -97.540284  -120.281983 -77.677002  -
81.038819  -100.220948 -86.136475  -98.309327  -111.514893 -78.270264  -
120.17212  -80.841065  -89.674073  -107.471924];
%longitude

yy=[32.483817   34.788544   34.860694   36.325748   38.935485   28.199863
32.74293    44.223159   40.256054   39.852408   42.103113   38.656346
37.584636   30.983261   45.294984   43.460751   46.336309   32.737386
38.501752   47.074611   41.645005   39.897937   34.58438    42.864692
35.519709   47.491967   40.301314   35.662652   43.965935   40.968285
34.039915   44.422796   35.98334    31.415538   39.066967   37.549804
47.402811   38.604852   44.548203   43.073703];
%latitude

uav=struct('name',{1,2},'c',{'37','34'},'iten_p',{[]},'iten_d',{[]},'k1',{1},
'k2',{1},'trip_xp',{[]},'trip_yp',{[]},'trip_xd',{[]},'trip_yd',{[]});

%package=struct('name',{'P1','P2','P3'},'pickup_l',{'40','34','15'},'deliver_
l',{'23','25','16'});
%package=struct('name',{'P1','P2','P3','P4'},'pickup_l',{'22','35','19','23'}
,'deliver_l',{'05','13','12','34'},'ps',{0},'ds',{0},'pc',{0},'dc',{0});
```

```
%package=struct('name',{'P1','P2','P3','P4','P5','P6'},'pickup_l',{'29','35',
'21','03','07','25'},'deliver_l',{'22','05','11','33','31','36'},'ps',{0},'ds
',{0},'pc',{0},'dc',{0});
%package=struct('name',{'P1','P2'},'pickup_l',{'29','03'},'deliver_l',{'22','
33'},'ps',{0},'ds',{0},'pc',{0},'dc',{0});
%package=struct('name',{'P1','P2','P3','P4'},'pickup_l',{'29','35','03','25'}
,'deliver_l',{'22','05','33','36'},'ps',{0},'ds',{0},'pc',{0},'dc',{0});
%package=struct('name',{'P1','P2','P3','P4','P5','P6'},'pickup_l',{'29','35',
'12','03','25','30'},'deliver_l',{'22','05','11','33','36','24'},'ps',{0},'ds
',{0},'pc',{0},'dc',{0});
package=struct('name',{'P1','P2','P3','P4','P5','P6'},'pickup_l',{'29','35','
02','03','25','30'},'deliver_l',{'22','05','04','33','36','24'},'ps',{0},'ds'
,{0},'pc',{0},'dc',{0});
package_cp=package; %just a copy of package for later

coords_m=[xx' yy'];

for i=1:length(xx)
    %if i<10; nn(i,:)=strcat('a0',num2str(i));  else
nn(i,:)=strcat('a',num2str(i)); end;
    city(i).name=nn(i,:);
    city(i).name_c=nn_c(i,:);
    city(i).x=xx(i);
    city(i).y=yy(i);
end
num_cities=length(city);

cc=zeros(num_cities,num_cities);
cc2=[1,1,1,1,1,1,2,2,2,3,3,3,3,3,4,4,5,5,5,5,5,5,6,6,7,8,8,8,8,8,9,9,9,9,9,9,
10,10,10,10,10,11,11,11,11,11,12,12,12,13,13,13,13,13,14,14,15,15,16,16,16,16
,17,17,17,18,18,19,19,20,20,20,20,21,21,22,22,22,23,23,23,24,25,25,25,26,27,2
7,28,29,30,30,31,32,35,36;6,7,14,18,31,33,4,23,35,18,19,28,33,34,22,35,12,21,
23,28,35,40,7,14,31,20,22,29,37,40,10,11,12,19,16,39,19,13,16,27,38,12,21,32,
17,39,21,19,28,19,38,36,25,33,18,34,16,24,27,39,30,24,26,32,39,33,34,28,33,37
,40,32,26,32,40,29,35,40,28,34,35,30,31,36,33,32,30,38,34,37,38,36,33,40,40,3
8];
for i=1:length(cc2)
    cc(cc2(1,i),cc2(2,i))=cc(cc2(1,i),cc2(2,i))+ 1;
end
%cc=cc+cc';
%max(cc(:))
%% Service routines for creating map_us.h
% assuming some states are connected
%{
trans='{';%h(n)=great circle dist
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;
```

```
dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
        end
    end
end
trans = substr(trans,0,-1);
trans = strcat(trans,'};');

trans_g='{';%g(n)=great circle dist + 20
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;


dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            dist=dist+20;
            trans_g =
strcat(trans_g,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',
');
        end
    end
end
trans_g = substr(trans_g,0,-1);
trans_g = strcat(trans_g,'};');

%coords_m=zeros(num_cities,2);
coords='{';%Just coordinates
for i=1:num_cities
    %coords_m(i,:)=[city(i).x city(i).y];
        coords =
strcat(coords,'{"',city(i).name,'",',num2str(city(i).y),',',num2str(city(i).x
),'}',',');
end
coords = substr(coords,0,-1);
coords = strcat(coords,'};');
%}


%% Service routines for creating map_us.h
% assuming all states are connected
%{
q=0;
```

```
trans='{';%h(n)=great circle dist
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        if cc(i,j)
%             lat1=city(i).y*pi/180;
%             lon1=city(i).x*pi/180;
%
%             lat2=city(j).y*pi/180;
%             lon2=city(j).x*pi/180;
%
%
dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
%             %Great circle distance (arc-length) is just radius times angle
(in rad)
%             trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
        q=q+1;
        end

    end
end
trans = substr(trans,0,-1);
trans = strcat(trans,'};');
q
q=0;
trans_g='{';%g(n)=great circle dist + 20
for i=1:(num_cities-1)
    for j=(i+1):num_cities
        %if cc(i,j)
            lat1=city(i).y*pi/180;
            lon1=city(i).x*pi/180;

            lat2=city(j).y*pi/180;
            lon2=city(j).x*pi/180;


dist=6373*acos(cos(lat1)*cos(lon1)*cos(lat2)*cos(lon2)+cos(lat1)*sin(lon1)*co
s(lat2)*sin(lon2)+sin(lat1)*sin(lat2));
            %Great circle distance (arc-length) is just radius times angle
(in rad)
            dist=dist+20;
            trans_g =
strcat(trans_g,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',
');
       q=q+1;
        %end
    end
end
trans_g = substr(trans_g,0,-1);
trans_g = strcat(trans_g,'};');

q
%}
```

```
%% Find the solution using "pick up and deliver cheapest package" method
%{
uav_c=uav_s;    %initially uav is located at uav_s (Start node). uav_c -
Current uav node
k=1;            %counter for # of delivered packages
iten_p=struct('iten',{});
iten_d=struct('iten',{});

while ~isempty(package)    %while packages are not delivered
                                                %deliver packages

    cost_p=[];  %the pick up cost row (cost to pick up each package in
current queue)
    cost_d=[];  %the deliver cost row (cost to deliver each package)
    cost=[];    %the cost to pick up and deliver the current package

    for i=1:length(package) %iterate through all packages and find p+d costs
            [iten_pg,cost_p(i)]=astar(uav_c,package(i).pickup_l);

[iten_dg,cost_d(i)]=astar(package(i).pickup_l,package(i).deliver_l);
    end

    cost=cost_p+cost_d;
    fst_p=find(cost==min(cost),1,'first');
    package_c=package(fst_p);

    [iten_p(k).iten,cost_pp(k)]=astar(uav_c,package_c.pickup_l);
    uav_c=package_c.pickup_l;
    [iten_d(k).iten,cost_dd(k)]=astar(uav_c,package_c.deliver_l);
    uav_c=package_c.deliver_l;

    package=rmvalue(package,fst_p);%if package has been delivered, take it
    %out of the que (rmvalue is a custom Mfile that gets rid of values in a
    %struct

    k=k+1; %one more package was delivered
end
%}


%% Find the solution using the "blind optimal method"
flag=0;
while ~flag    %while packages are not delivered
    for m=1:length(uav)

        for i=1:length(package)     %calculate all the current p and d costs
            [iten_pg,package(i).pc]=astar(package(i).pickup_l,uav(m).c);
            [iten_dg,package(i).dc]=astar(package(i).deliver_l,uav(m).c);
        end

        cost=[];
        actions=[];
        for i=1:length(package) %got through all the packages
```

```
          if (~package(i).ps && ~package(i).ds)     %if package has not been
delivered yet
            cost=[cost package(i).pc];
            actions=[actions 0];
          elseif (package(i).ps && ~package(i).ds)  %if package has not been
picked up yet
            cost=[cost package(i).dc];
            actions=[actions 1];
          else %the package was already picked up and delivered - so don't go
there!
              cost=[cost 10^9];
              actions=[actions -1];
          end
        end

        fst=find(cost==min(cost),1,'first');
        package_c=package(fst);
        if (actions(fst)==0)
            [uav(m).iten_p(uav(m).k1).iten,
cost_pp]=astar(package_c.pickup_l,uav(m).c);
            uav(m).c=package_c.pickup_l;
%             k1=k1+1;
              uav(m).k1=uav(m).k1+1;
                package(fst).ps=1;
        elseif (actions(fst)==1)
            [uav(m).iten_d(uav(m).k2).iten,
cost_dd]=astar(package_c.deliver_l,uav(m).c);
            uav(m).c=package_c.deliver_l;
%             k2=k2+1;
              uav(m).k2=uav(m).k2+1;
                package(fst).ds=1;
        else
            printf('Hello! Something really bad just happened!')  %This
should never happen
        end

        sum=0;
        for i=1:length(package)
            sum=sum+package(i).ds;
            if sum==length(package);
                flag=1;
            end
        end
    end

end


%% Plot the map and solution using the "some nodes are connected assumption
%almost me_ex
%  iten_p=struct('iten',{[29,22],[22,35],[05,12,19]});
%  iten_d=struct('iten',{[35,05],[19,12],[12,19,13]});
%almost_all
%  iten_p=struct('iten',{[29,22],[22,35],[05,19]});
%  iten_d=struct('iten',{[35,05],[19,12],[12,13]});
%opt
```

```
% iten_p=struct('iten',{[29,22],[22,19],[19,35]});
% iten_d=struct('iten',{[35,13],[13,12],[12,05]});
for m=1:length(uav)
    for k=1:(length(uav(m).iten_p))
        [uav(m).trip_xp(k).coord, uav(m).trip_yp(k).coord]=
iten2line(uav(m).iten_p(k).iten,coords_m);
        [uav(m).trip_xd(k).coord, uav(m).trip_yd(k).coord]=
iten2line(uav(m).iten_d(k).iten,coords_m);
    end
end

sca=1000;

h=figure(1);clf
plot(coords_m(:,1),coords_m(:,2),'bo')
hold on;
for i=1:length(city); text(city(i).x+.3,city(i).y+.3,city(i).name_c); end
%.15 for room

for m=1:length(uav)
    for k=1:(length(uav(m).iten_p))
        line(uav(m).trip_xp(k).coord+k/sca, uav(m).trip_yp(k).coord+k/sca,
'color','g' ,'LineWidth',2);
        line(uav(m).trip_xd(k).coord+(k+1)/sca,
uav(m).trip_yd(k).coord+(k+1)/sca, 'color','r' ,'LineWidth',2);
    end
end

legend('states','pickup','deliver')%,'pickup 2','deliver 2','pickup
3','deliver 3')

for i=1:(length(city)-1)
    for j=(i+1):length(city)
        if cc(i,j)
            line([city(i).x city(j).x], [city(i).y city(j).y],
'LineStyle',':');
        end
    end
end

for m=1:length(uav)
    for k=1:(length(uav(m).iten_p))
        line(uav(m).trip_xp(k).coord+k/sca, uav(m).trip_yp(k).coord+k/sca,
'color','g' ,'LineWidth',2);
        line(uav(m).trip_xd(k).coord+(k+1)/sca,
uav(m).trip_yd(k).coord+(k+1)/sca, 'color','r' ,'LineWidth',2);
    end
end


for i=1:length(package_cp); text(city(str2double(package_cp(i).pickup_l)).x-
.5, city(str2double(package_cp(i).pickup_l)).y-.5, package_cp(i).name,
'color', 'm', 'FontSize', 15); end %.15 for room
```

```
for i=1:length(package_cp); text(city(str2double(package_cp(i).deliver_l)).x-
.5, city(str2double(package_cp(i).deliver_l)).y-.5, strcat('D',num2str(i)),
'color', 'b', 'FontSize', 15); end %.15 for room


hold off
xlabel('x, longitude (deg)');
ylabel('y, latitude (deg)');


%% Plot the map and solution using the "all-nodes connected" assumption
%{
trip_xp=struct('coord',{});
trip_yp=struct('coord',{});
trip_xd=struct('coord',{});
trip_yd=struct('coord',{});

for k=1:length(iten_p)
    [trip_xp(k).coord, trip_yp(k).coord]= iten2line(iten_p(k).iten,coords_m);
    [trip_xd(k).coord, trip_yd(k).coord]= iten2line(iten_d(k).iten,coords_m);
end

sca=6;

h=figure(1);clf
plot(coords_m(:,1),coords_m(:,2),'bo')
hold on;
for i=1:length(city); text(city(i).x+.3,city(i).y+.3,city(i).name_c); end
%.15 for room
for i=1:length(package_cp); text(city(str2double(package_cp(i).pickup_l)).x-
.5, city(str2double(package_cp(i).pickup_l)).y-.5, package_cp(i).name,
'color', 'm', 'FontSize', 15); end %.15 for room

for k=1:length(iten_p)
    line(trip_xp(k).coord, trip_yp(k).coord, 'color','g' ,'LineWidth',2);
    line(trip_xd(k).coord, trip_yd(k).coord, 'color','r' ,'LineWidth',2);
end
legend('cities','pickup','deliver')%,'pickup','deliver','pickup','deliver')

hold off
%}


%% Rooms Example for Search Algorithms
%AE 740, Final Project, Fall 2009
%Alexey Morozov, 12/14/09, morozova [at] umich [dot] edu

%A description that even a casual observer will understand (I hope):
%An autonomous UAV can go from one waypoint to another only along
%vertical and horizontal lines (NOT along diagonal). Also it cannot go from
%certain waypoints to certain other waypoints because there are obstacles
%in between. Once you look at the plot this file produces, the map should
%become more apperent.
```

```
%The goal is to pick up a package from waypoint 1 (called 'R01' because this
%problem was originally about rooms) and deliver it to waypoint 10 ('R10').
%The UAV starts in waypoint 3 ('R03').

%Files required for this file to run properly (all should be in the same
%directory):
%   -astar.m (needs mysearch_rooms.exe - precompiled from mysearch.c)
%NOTE: if you just got these files, rename mysearch.exe to
%mysearch_uavs.exe and then mysearch_rooms.exe to mysearch.exe (because
%these files are precompiled with domain-specific maps).

%   -rmvalue.m
%   -iten2line.m
%   -substr.m (I didn't write this one, but I hope it's ok if I use it
%   here. Taken from pja:
%   http://home.online.no/~pjacklam/matlab/software/util/strutil/substr.m)

%% Setup the problem
clear
clc

%the grid map
city=struct('name',{'R01','R02','R03','R04','R05','R06','R07','R08','R09','R1
0'},...
    'x',{5,7,1,1,1,1,3,3,3,3},'y',{3.5,3.5,5.5,4.5,3,1,5.5,4.5,3,1});
num_cities=length(city);

%coords in matrix form
for i=1:num_cities
    coords_m(i,:)=[city(i).x city(i).y];
end

%packages to be delivered, pickup_l=pickup location ...
package=struct('name',{'P1'},'pickup_l',{'01'},'deliver_l',{'10'});

%where uav starts
uav_s='03';


%% Subroutines used to create the map_us.h entries: transition_set[] and
coords[]

%{
trans='{';              %straight-line dist (not needed even for the C-file)
for i=1:(num_cities-1)

    for j=(i+1):num_cities
        dist=sqrt((city(i).x-city(j).x)^2+(city(i).y-city(j).y)^2);
        trans =
strcat(trans,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',')
;
    end
end
trans=substr(trans,0,-1);
trans = strcat(trans,'};');
```

```
transM='{';             %Manhattan dist
for i=1:(num_cities-1)

    for j=(i+1):num_cities
        dist=abs(city(i).x-city(j).x)+abs(city(i).y-city(j).y);
        transM =
strcat(transM,'{"',city(i).name,'","',city(j).name,'",',num2str(dist),'}',',','
);
    end
end
transM=substr(transM,0,-1);
transM = strcat(transM,'};');

coords_m=zeros(num_cities,2);
coords='{';             %Just coordinates
for i=1:num_cities
   coords =
strcat(coords,'{"',city(i).name,'",',num2str(city(i).x),',',num2str(city(i).y
),'}',',');
end
coords=substr(coords,0,-1);
coords = strcat(coords,'};');
%}

%% Find the solution

%how to call the search method: [iten,cost]=astar('03','09')

% since there is only one package, could just do this:
%[iten_m1,cost_p]=astar(uav_s,package(1).pickup_l);
%[iten_m2,cost_d]=astar(package(1).pickup_l,package(1).deliver_l);


% More complicated solver (used in early versions of main.m)

uav_c=uav_s;    %initially uav is located at uav_s (Start node). uav_c -
Current uav node
k=1;            %counter for # of delivered packages
iten_p=struct('iten',{});
iten_d=struct('iten',{});

while ~isempty(package)    %while packages are not delivered
                                            %deliver packages

    cost_p=[];  %the pick up cost row (cost to pick up each package in
current queue)
    cost_d=[];  %the deliver cost row (cost to deliver each package)
    cost=[];    %the cost to pick up and deliver the current package
    j=1;        %continuous counter - allows packages to be deleted from
queue (see below)

    for i=1:length(package) %iterate through all packages and find p+d costs
```

```
            [iten_pg,cost_p(j)]=astar(uav_c,package(i).pickup_l);

[iten_dg,cost_d(j)]=astar(package(i).pickup_l,package(i).deliver_l);

            j=j+1;
    end
    cost=cost_p+cost_d;
    fst_p=find(cost==min(cost),1,'first'); %find the cheapest package to pick
up and deliver
    package_c=package(fst_p);

    [iten_p(k).iten,cost_p(k)]=astar(uav_c,package_c.pickup_l);%pick it up
    uav_c=package_c.pickup_l;
    [iten_d(k).iten,cost_d(k)]=astar(uav_c,package_c.deliver_l);%deliver it
    uav_c=package_c.deliver_l;

    package=rmvalue(package,fst_p);%if package has been delivered, take it
    %out of the que (rmvalue is a custom Mfile that gets rid of values in a
    %struct

end




%% Plot the map and solution

path_m=[3 4 5 6 10 9 1 2 1 8 7 8 4];

[trans_x,trans_y]=iten2line(path_m,coords_m);

trip_xp=struct('coord',{});
trip_yp=struct('coord',{});
trip_xd=struct('coord',{});
trip_yd=struct('coord',{});

for k=1:length(iten_p)
    [trip_xp(k).coord,trip_yp(k).coord]=iten2line(iten_p(k).iten,coords_m);
    [trip_xd(k).coord,trip_yd(k).coord]=iten2line(iten_d(k).iten,coords_m);
end


h=figure(1);clf
plot(coords_m(:,1),coords_m(:,2),'bo')
hold on;
for i=1:10; text(city(i).x+.15,city(i).y+.15,city(i).name); end
line(trans_x,trans_y,'LineStyle',':');
for k=1:length(iten_p)
    line(trip_xp(k).coord,trip_yp(k).coord,'color','g','LineWidth',2);
    line(trip_xd(k).coord,trip_yd(k).coord,'color','r','LineWidth',2);
end
hold off
xlim([0 8]);
ylim([0 6.5]);
```

```
legend('cities','roads','pickup','deliver')
```