These functions are used in the article:

Chow, J. Y. J., 2016. Dynamic UAV-based traffic monitoring under uncertainty as a stochastic arc-inventory routing policy. *International Journal of Transportation Science & Technology*, in press.

SDAIRP corresponds to Algorithm 1 in the paper in Section 4.2. SARP uses MATLAB’s commercial IP solver (intlinprog) to solve a selective VRP as formulated in Eq (5) in the paper.

---------------

function [X,L,F,Y0,V0]=SDAIRP(C,H,E,Q,W,K,Rtp,T,O,Yd,S0,mu,sigma,NumBasis,fuel)

%SDAIRP (stochastic dynamic arc-inventory routing problem) is a policy for determining which nodes to visit at time 0

%while accounting for future actions and inventory states up to horizon T

%and dependent on current inventory state. Expectation of future inventory

%states is estimated using polynomial least squares (Hermite). Simulation

%of stochastic demand consumption assumes mean-reverting process.

%INPUTS

%C = NxN drone travel cost matrix

%H = NxN inventory cost matrix

%E = NxN energy use for monitoring

%Q = NxN max inventory allowed

%W = vehicle energy capacity

%K = number of vehicles in fleet

%Rtp = NxNxTxP simulated P sample paths as per LSM approach (Longstaff & Schwartz,

%2001; Chow & Regan, 2011)

%T = time horizon

%O = NxN stock out cost at each link ($/period if S\_t < 0)

%Yd = NxNxT deterministic policy of when the baseline policy is to execute

%a refill at a link

%S0 = NxN initial inventory levels

%sigma = NxN volatility parameter

%NumBasis = number of polynomials to estimate each state from

%fuel = 0 leaves out C in capacity, 1 leaves it in

%OUTPUTS

%X,L,F,Y0 = base time output decision variables

%Ytp = selection decision at each t and p

%Stp = remaining post-decision inventory at each t and p

%V0 = policy value at base time

%Vtp = PxT policy values

%Initiate (Step 1 and 2 done externally)

P=size(Rtp,4);

N=size(C,1);

Stp=zeros(N,N,T,P);

%Create X variables for full inventory

RegXfull=zeros(NumBasis+1,N,N);

for i=1:N

for j=1:N

if Q(i,j)>0

RegXfull(1,i,j)=1;

RegXfull(2,i,j)=Q(i,j);

for m=2:NumBasis

RegXfull(m+1,i,j)=Q(i,j)\*RegXfull(m,i,j)-m\*RegXfull(m-1,i,j);

end

end

end

end

%Step 3 - determine realized inventories

for t=1:T

for p=1:P

for i=1:N

for j=1:N

if S0(i,j)>0

if Yd(i,j,t)==0

if t==1

Stp(i,j,1,p)=S0(i,j)-Rtp(i,j,t,p);

else

Stp(i,j,t,p)=Stp(i,j,t-1,p)-Rtp(i,j,t,p);

end

else

Stp(i,j,t,p)=Q(i,j);

end

end

end

end

end

end

%Step 4 - approximate dynamic programming

failed=zeros(N,N,P); %this is for checking if sample path stocked out

stopt=zeros(N,N,P)+T; %this is for knowing where to update S/check failed to

tau=T;

while tau>0

if tau==T

for p=1:P

Rest=zeros(N,N);

for i=1:N

for j=1:N

if Stp(i,j,T-1,p)-Rtp(i,j,T,p)<mu(i,j)

Rest(i,j)=-O(i,j);

end

if Stp(i,j,T-1,p)-Rtp(i,j,T,p)<0

failed(i,j,p)=1;

end

end

end

[~,~,~,Y,~,~,~]=SARP(C,E,H,Q,Stp(:,:,T-1,p)-Rtp(:,:,T,p),Rest,K,W,fuel);

for i=1:N

for j=1:N

if Y(i,j)==1

Stp(i,j,T,p)=Q(i,j);

end

end

end

end

else

%determine new decisions using approximation

Restfull=zeros(N,N);

Rest=zeros(N,N,P);

for i=1:N

for j=1:N

%estimate coefficients for basis functions

if sigma(i,j)>0

dbasis=zeros(P,1);

RegYd=zeros(P,1);

RegXd=zeros(P,NumBasis+1);

for p=1:P

RegYd(p,1)=failed(i,j,p)\*O(i,j);

if tau>1

dbasis(p,1)=Stp(i,j,tau-1,p)-Rtp(i,j,tau,p);

else

dbasis(p,1)=S0(i,j)-Rtp(i,j,tau,p);

end

%recursive generation of Hermite polynomial basis functions

RegXd(p,1)=1;

RegXd(p,2)=dbasis(p,1);

for m=2:NumBasis

RegXd(p,m+1)=dbasis(p,1)\*RegXd(p,m)-m\*RegXd(p,m-1);

end

end

dCoeff=((RegXd'\*(RegXd+realmin))\(RegXd+realmin)')\*RegYd;

%estimated risk from full inventory

Restfull(i,j)=RegXfull(:,i,j)'\*dCoeff;

%estimated risk from pre-decision inventory state

for p=1:P

if tau>1

if Stp(i,j,tau-1,p)-Rtp(i,j,tau,p)<mu(i,j)

Rest(i,j,p)=-O(i,j);

else

Rest(i,j,p)=Restfull(i,j)-RegXd(p,:)\*dCoeff;

end

else

if S0(i,j)-Rtp(i,j,tau,p)<mu(i,j)

Rest(i,j,p)=-O(i,j);

else

Rest(i,j,p)=Restfull(i,j)-RegXd(p,:)\*dCoeff;

end

end

end

elseif Q(i,j)>0

%determine the need for serving deterministic links

for p=1:P

if tau>1

if Stp(i,j,tau-1,p)-Rtp(i,j,tau,p)<Rtp(i,j,tau+1,p)

Rest(i,j,p)=-O(i,j);

end

else

if S0(i,j)-Rtp(i,j,tau,p)<Rtp(i,j,tau+1,p)

Rest(i,j,p)=-O(i,j);

end

end

end

end

end

end

%update decisions, states

for p=1:P

p

tau

Rest(:,:,p)

if tau>1

[~,~,~,Y,~,~,~]=SARP(C,E,H,Q,Stp(:,:,tau-1,p)-Rtp(:,:,tau,p),Rest(:,:,p),K,W,fuel);

else

[~,~,~,Y,~,~,~]=SARP(C,E,H,Q,S0(:,:)-Rtp(:,:,tau,p),Rest(:,:,p),K,W,fuel);

end

for i=1:N

for j=1:N

if Y(i,j)==1

Stp(i,j,tau,p)=Q(i,j);

stopt(i,j,p)=tau;

if tau>1

if Stp(i,j,tau-1,p)-Rtp(i,j,tau,p)>=0

failed(i,j,p)=0;

else

failed(i,j,p)=1;

end

else

if S0(i,j)-Rtp(i,j,tau,p)>=0

failed(i,j,p)=0;

else

failed(i,j,p)=1;

end

end

else

for sau=tau+1:stopt(i,j,p)

if sau>1

Stp(i,j,sau,p)=Stp(i,j,sau-1,p)-Rtp(i,j,sau,p);

else

Stp(i,j,sau,p)=S0(i,j)-Rtp(i,j,sau,p);

end

if Stp(i,j,sau,p)<0

failed(i,j,p)=1;

end

end

end

end

end

end

end

%update time

tau=tau-1;

end

%update base period decisions and values

Restfull=zeros(N,N);

Rest0=zeros(N,N);

AvgR1=zeros(N,N);

for i=1:N

for j=1:N

AvgR1(i,j)=mean(Rtp(i,j,1,:));

%estimated risk from pre-decision inventory state

Restfull(i,j)=RegXfull(:,i,j)'\*dCoeff;

RegXd=zeros(1,NumBasis+1);

dbasis(1,1)=S0(i,j)-AvgR1(i,j);

RegXd(1,1)=1;

RegXd(1,2)=dbasis(1,1);

for m=2:NumBasis

RegXd(1,m+1)=dbasis(1,1)\*RegXd(1,m)-m\*RegXd(1,m-1);

end

if S0(i,j)-AvgR1(i,j)<mu(i,j)

Rest0(i,j)=-O(i,j);

elseif Q(i,j)>0

Rest0(i,j)=Restfull(i,j)-RegXd(1,:)\*dCoeff;

end

end

end

[X,L,F,Y0,V0,~,~]=SARP(C,E,H,Q,S0-AvgR1,Rest0,K,W,fuel);

function [X,L,F,Y,fval,exitflag,output]=SARP(C,E,H,Q,S,stockout,K,W,fuel)

%SARP is a selective arc routing problem, where a link is monitored in the %current time if the risk of stockout exceeds the cost of allocating a UAV to %the link.

%INPUTS

%C = NxN cost matrix

%E = NxN energy use for monitoring a link

%stockout = NxN benefit (negative) of stockout risk reduction

%K = number of vehicles in fleet

%W = fuel capacity

%H = inventory holding cost

%Q = max inventory

%S = inventory

%fuel = 0 if travel cost is not included in fuel capacity, 1 else --

%original runs with Monroy network were with fuel=1

%OUTPUTS

%X = NxNxK = binary, directional link traversed

%L = NxNxK = binary, link monitored

%F = NxNxK = subtour elimination dummy var

%Y = NxN = binary, link selected for monitoring

N=size(C,1);

NumVar=N\*N\*K+N\*N\*K+N\*N\*K+N\*N;

coeff=zeros(NumVar,1);

for i=1:N

for j=1:N

for p=1:K

coeff((i-1)\*N\*K+(j-1)\*K+p,1)=C(i,j);

end

coeff(N\*N\*K\*3+(i-1)\*N+j,1)=stockout(i,j)+H(i,j)\*(Q(i,j)-S(i,j));

end

end

A=zeros(N\*N\*K+K+N\*N\*K,NumVar);

b=zeros(N\*N\*K+K+N\*N\*K,1);

Aeq=zeros(N\*K+N\*N+N\*K,NumVar);

beq=zeros(N\*K+N\*N+N\*K,1);

ineq=0;

ieq=0;

%flow constraints

for p=1:K

for i=1:N

ieq=ieq+1;

for j=1:N

if C(i,j)>0

Aeq(ieq,(i-1)\*N\*K+(j-1)\*K+p)=-1;

end

if C(j,i)>0

Aeq(ieq,(j-1)\*N\*K+(i-1)\*K+p)=1;

end

end

end

end

%link selection for monitoring

for i=1:N

for j=i+1:N

if C(i,j)>0

ieq=ieq+1;

for p=1:K

Aeq(ieq,N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p)=1;

Aeq(ieq,N\*N\*K+(j-1)\*N\*K+(i-1)\*K+p)=1;

end

Aeq(ieq,N\*N\*K\*3+(i-1)\*N+j)=-1;

end

end

end

%coverage ensured

for p=1:K

for i=1:N

for j=1:N

ineq=ineq+1;

A(ineq,(i-1)\*N\*K+(j-1)\*K+p)=-1;

A(ineq,N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p)=1;

end

end

end

%fuel capacity

for p=1:K

ineq=ineq+1;

for i=1:N

for j=1:N

if C(i,j)>0

if fuel==1

A(ineq,(i-1)\*N\*K+(j-1)\*K+p)=C(i,j);

end

end

if C(i,j)>0

A(ineq,N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p)=E(i,j);

end

end

end

b(ineq,1)=W;

end

%subtour elim 1

for p=1:K

for i=2:N

ieq=ieq+1;

for j=1:N

if C(i,j)>0

Aeq(ieq,N\*N\*K\*2+(i-1)\*N\*K+(j-1)\*K+p)=1;

end

if C(j,i)>0

Aeq(ieq,N\*N\*K\*2+(j-1)\*N\*K+(i-1)\*K+p)=-1;

end

if C(i,j)>0

Aeq(ieq,N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p)=-1;

end

end

end

end

%subtour elim 2

for p=1:K

for i=1:N

for j=1:N

if C(i,j)>0

ineq=ineq+1;

A(ineq,(i-1)\*N\*K+(j-1)\*K+p)=-N^2;

A(ineq,N\*N\*K\*2+(i-1)\*N\*K+(j-1)\*K+p)=1;

end

end

end

end

%trim

Aeq=Aeq(1:ieq,:);

beq=beq(1:ieq,1);

A=A(1:ineq,:);

b=b(1:ineq,1);

intcon=zeros(N\*N\*K\*2+N\*N,1);

for i=1:N\*N\*K\*2

intcon(i,1)=i;

end

for i=1:N\*N

intcon(N\*N\*K\*2+i,1)=i+N\*N\*K\*3;

end

ub=ones(NumVar,1);

for i=1:N

for j=1:N

if C(i,j)==0

for p=1:K

ub((i-1)\*N\*K+(j-1)\*K+p,1)=0;

ub(N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p,1)=0;

ub(N\*N\*K\*2+(i-1)\*N\*K+(j-1)\*K+p,1)=0;

end

end

if Q(i,j)==0

ub(N\*N\*K\*3+(i-1)\*N+j,1)=0;

end

end

end

for i=N\*N\*K\*2+1:N\*N\*K\*3

ub(i,1)=inf;

end

lb=zeros(NumVar,1);

[sol,fval,exitflag,output]=intlinprog(coeff,intcon,A,b,Aeq,beq,lb,ub);

X=zeros(N,N,K);

for p=1:K

for i=1:N

for j=1:N

X(i,j,p)=sol((i-1)\*N\*K+(j-1)\*K+p,1);

end

end

end

L=zeros(N,N,K);

for p=1:K

for i=1:N

for j=1:N

L(i,j,p)=sol(N\*N\*K+(i-1)\*N\*K+(j-1)\*K+p,1);

end

end

end

F=zeros(N,N,K);

for p=1:K

for i=1:N

for j=1:N

F(i,j,p)=sol(N\*N\*K\*2+(i-1)\*N\*K+(j-1)\*K+p,1);

end

end

end

Y=zeros(N,N);

for i=1:N

for j=1:N

Y(i,j)=sol(N\*N\*K\*3+(i-1)\*N+j,1);

end

end