Supplementary Materials

DATA

In order to schedule the stations of the region in dispatching to fire and accidents, the PSO method was written using the following:

1. The fatigue coefficient of firefighters during operation is considered equal to 0.5.
2. RT_{ijk} is the duration of operation in each mission (fire or incident) by the station K. In the current problem, the solution vector has m+n-1 dimensions.
3. The values of i and j respectively specify the counter of fire and accident and k the counter of fire stations.
4. r= represents work shifts.
5. S= the start time of operation for each relief unit.
6. $ST_{k[r]}$= the start time of the relief operation by the k^{th} station in the r^{th} work shift.
7. FT_{kji}= the completion time of the operation to the i^{th} incident or the j^{th} fire by the k^{th} station.
8. $RT_{k[r]}$= the actual duration of operations in fire (j) and accidents (i) according to the application of the fatigue factor (α).
9. TT_{ijk}= the travel time from the i^{th} event to the j^{th} event for the k^{th} station.
10. TT_{ijk}= the travel time from the i^{th} event to the j^{th} event for the k^{th} station.
11. $TT_{k[r]}$= the travel time of the k^{th} station in the r^{th} work shift.
12. The chi coefficient for particle speed update is calculated based on the following relations:

```
phi=C1+C2;
chi=2/(phi-2+sqrt(phi^2-4*phi));
w=chi; % Inertia Weight
```

13. RT = the final output of the updated vector at the end of the iteration count.
14. DL_i = the intensity of the effect of each mission or incident.
15. X_{ijk}= the problem solving variable for all missions, which is considered as 0 and 1. (If X_{ijk}=1, then the j^{th} fire or the i^{th} incident is assigned to the k^{th} station).
16. The Cap_{ijk} = the capacity and ability of the k^{th} station according to the i^{th} type of accident or the j^{th} type of fire (its variable value is considered 0 and 1). If Cap_{ijk}=1, then each fire station (k) is only capable of dispatching 1 mission. And if Cap_{ijk} =0, then fire (j) or accident (i) cannot be assigned to station (k), so another station must be sent to the location.
17. The generation of the initial response vector in the form of random numbers between 0 and 1 is used to create the nVar parameter.
18. N = the number of incidents and M= the number of available stations.
19. The problem in small dimensions and 10 problems in large dimensions are defined to check the methods. The amount of time of the total operation in seconds will be reported as the answer.

The steps of the algorithm of the problem are written according to the instructions below, and its output is given in the fourth chapter (research findings) in the form of mathematical relations to solve the problem.

Problem Solving Algorithm

```
function
model= Create Model()
Fatigue= ;
PT=[]
PT=model.PT;
Fatigue= Fatigue model;
[~,x]= sort(q);
Majazi= find(x>I);      Virtual
```

```
I=size(PT,1);
K=size(PT,2);
Sijk= [];
Sdpk= []
Wi=[]
;
model.I=I;
model.K=K;
model.PT=PT;
model.Sij
k=Sijk;
model.Sdpk=Sdpk;
model.Wi=Wi;
model.Fatigue=Fatigue;
model.nVar= I+K-1;
end



function
q=CreateRandomSolution(model)
nVar=model.nVar;
q=unifrnd
(0,1,[1 nVar]);
End



function
ANSWER ANALYSIS ALGORITHM
sol=TahlileJavab(q,model)
I=model.I;
K=model.K;
From= [0 Majazi]+1;  Virtual
To= [Majazi I+K]-1; Virtual
L=cell(K,1);
for
k=1:K
L{k}= x(From(k):To(k));
end
ST=zeros(I,1);
FT=zeros(I,1);
p=zeros(I,1);
MCT=zeros(K,1);
for
k=1:K
for
i=L{k}
r= find(L{k}==i);
if r==1
ST(i)= Sdpk(i,k);
else
PreviousJob= L{k}(r-1);
ST(i)= FT(PreviousJob)+Sijk
(PreviousJob,i,k);
end
for
h=1:r
p(i)= PT(i,k)*((1+Fatigue)^(h-1));
end
FT(i)=ST(i)+p(i);
end
if ~ is empty(L{k})MCT(k)= FT(L{k}(end));
end
end
sol.x=x;
sol.L=L;
sol.ST=ST;
sol.p=p;
sol.FT=FT;
sol.MCT=MCT;
end
function
[z sol]=MyCost(q,model)
sol=TahlileJavab(q,model);   ANSWER   AALYSIS
ALGORITHM
```

S_{ijk}=model.S_{ijk}
;
Sdpk=model.Sdpk
;
function
xnew=mutate(q)[~, x]=sort(q);
M=randi([1 3]);
switchM
case1
Newx=DoSwap(x);
case2
Newx=DoReversion(x);
case3
Newx=DoInsertion(x);
end
xnew=zeros(size(q));
xnew(Newx)=q(x);
end

Function 'TRANSACTIONS
Newx=DoSwap(x)
n=numel(x);
i=randsample(n,2);
i1=i(1);
i2=i(2);
Newx=x;
Newx([i1 i2])=x([i2 i1]);
End

functionNewx=DoReversion(x)
n=numel(x);
i=randsample(n,2);
Wi=model.Wi;
FT=sol.FT;
z=sum(Wi.*FT);
end
functionNewx=DoInsertion(x)
n=numel(x);
i=randsample(n,2);
i1=i(1);
i2=i(2);
ifi1<i2
Newx=[x(1:i1) x(i2) x(i1+1:i2-1) x(i2+1:end)];
else
Newx=[x(1:i2-1) x(i2+1:i1) x(i2) x(i1+1:end)];
end
end
clc
;
clear
;
close
all;
formatshort;
%% Problem Definition
model= CreateModel();
CostFunction=@(x) MyCost(x,model); % Cost Function
nVar=model.nVar; % Number of Decision Variables
VarSize=[1 nVar]; % Size of Decision Variables Matrix
VarMin=; % Lower Bound of Variables
VarMax= ; % Upper Bound of Variables
%% PSO ParametersMaxIt=; % Maximum Number of Iterations
nPop=; % Population Size (Swarm Size)% % Constriction Coefficients
C1=
;
C2=
;
phi=C1+C2; Calculating the speed of particles (movement of forces)
chi=2/(phi-2+sqrt(phi^2-4*phi));

```
i1=min(i);
i2=max(i);
Newx=x;
Newx(i1:i2)=x(i2:-1:i1);
End
```

```
% Initialize Velocity
particle(i).Velocity=zeros(VarSize) The velocity of the particle (i) is zero
;
% Evaluation
[particle(i).Cost
particle(i).Sol]=CostFunction(particle(i).Position);
% Update Personal Best
particle(i).Best.Position=particle(i).Position;
particle(i).Best.Cost=particle(i).Cost;
particle(i).Best.Sol=particle(i).Sol;
% Update Global Best
if
particle(i).Best.Cost<GlobalBest.Cost
GlobalBest=particle(i).Best;
end
end
BestCost=zeros(MaxIt,1);
%% PSO Main Loop
Tic
;
for
it=1:MaxIt
for
```

```
w=chi; % Inertia Weight
% Velocity Limits
VelMax=0.1*(VarMax-VarMin);VelMin=-
VelMax;T=;alpha= ;
%% Initialization
empty_particle.Position=[];
empty_particle.Cost=[];
empty_particle.Sol=[];
empty_particle.Velocity=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.Best.Sol=[];
particle=repmat
(empty_particle,nPop,1);
GlobalBest.Cost=inf;
for
i=1:nPop
% Initialize Position
particle(i).Position=CreateRandomSolution(model);
% Update Velocity
particle(i).Velocity = w*(particle(i).Velocity...i
+c1*rand(VarSize).*(particle(i).Best.Position-
particle(i).Position)...
+c2*rand(VarSize).*(GlobalBest.Position-
particle(i).Position));
% Apply Velocity Limits
particle(i).Velocity =
max(particle(i).Velocity,VelMin);
particle(i).Velocity =
min(particle(i).Velocity,VelMax);
% Update Position
particle(i).Position = particle(i).Position +
particle(i).Velocity;
% Velocity Mirror Effect
IsOutside=(particle(i).Position<VarMin |
particle(i).Position>VarMax);
particle(i).Velocity(IsOutside)=-
particle(i).Velocity(IsOutside);
% Apply Position Limits
particle(i).Position =
max(particle(i).Position,VarMin);particle(i).Posit
ion = min(particle(i).Position,VarMax);
% Evaluation
```

```
i=1:nPop
[particle(i).Cost particle(i).Sol] =
CostFunction(particle(i).Position);
% mutate
NewSol.Position= mutate(particle(i).Position);
[NewSol.Cost NewSol.Sol]=
CostFunction(NewSol.Position);
if
NewSol.Cost<= particle(i).Cost
particle(i).Position= NewSol.Position;
particle(i).Cost= NewSol.Cost;
particle(i).Sol= NewSol.Sol;
end
% Update Personal Best
if
particle(i).Cost<particle(i).Best.Cost
particle(i).Best.Position=particle(i).Position;
particle(i).Best.Cost=particle(i).Cost;
particle(i).Best.Sol=particle(i).Sol;
else
delta= particle(i).Cost
particle(i).Best.Cost;
p= exp(-delta/T);
if
rand<=p
particle(i).Best.Position=particle(i).Position;
particle(i).Best.Cost=particle(i).Cost;
particle(i).Best.Sol=particle(i).Sol;
end
% Update Global Best
if
particle(i).Best.Cost<GlobalBest.Cost
GlobalBest=particle(i).Best;
end
end
end
% mutate
NewSol.Position= mutate(GlobalBest.Position);
[NewSol.Cost NewSol.Sol]=
CostFunction(NewSol.Position);
if
NewSol.Cost<= GlobalBest.Cost
GlobalBest= NewSol;
end
```

```
BestCost(it)=GlobalBest.Cost;
disp(['Iteration ' num2str(it) ', Best Cost = ' num2str(BestCost(it))]);
T= alpha*T;
end
toc
;
```

The mathematical model of the problem can be formulated as a non-linear complex integer programming model as follows:

1. $Min\sum_{i}\sum_{k} w_i.C_i^k$

2. $\sum_{k=1}^{m}\sum_{r=1}^{R} X_{i[r]}^k = 1$ i=1, ….,n

3. $\sum_{i=1}^{n} X_{i[r]}^k \le 1$ k=1, ….,m r=1,…,R

4. $X_{0[0]}^k = 1$ k=1, ….,m

5. $\sum_{i=1}^{n} X_{i[r+1]}^k \le \sum_{i=1}^{n} X_{i[r]}^k$ k=1, ….,m r=1,…,R-1

6. $PT_{[r]}^k = \sum_{i=0}^{n} p_i^k (1+\alpha)^{r-1} X_{i[r]}^k$ k=1, ….,m r=1,…,R

7. $S_{[r]}^k \ge s_{ij}^k - BigM(2 - X_{i[r]}^k - X_{j[r+1]}^k)$ $\forall_{\{(i,j)|i\neq j\}}; \forall_{k,r=1,2,...,R-1}$

8. $S_{[r]}^k \le s_{ij}^k + BigM(2 - X_{i[r]}^k - X_{j[r+1]}^k)$ $\forall_{\{(i,j)|i\neq j\}}; \forall_{k,r=1,2,...,R-1}$

9. $S_{[r]}^k \le BigM\,(\sum_{i=1}^{n} X_{i[r]}^k)$ k=1, ….,m r=1,…,R

10. $C_i^k \ge (ST_{[r]}^k + PT_{[r]}^k) X_{i[r]}^k$ i=1, ….,n k=1, ….,m r=1,…,R

11. $ST_{[r]}^k = (ST_{[r-1]}^k + PT_{[r-1]}^k + S_{[r-1]}^k (\sum_{i=1}^{n} X_{i[r]}^k)$ k=1, ….,m r=1,…,R

12. $\sum_{r} X_{i[r]}^{k} \le Cap_{i}^{k}$ — i=1, ….,n; k=1, ….,m

13. $ST_{[0]}^{k} = 0$ — k=1, ….,m

14. $X_{i[r]}^{k} \in \{0,1\}$ — i=1, ….,n; k=1, ….,m; r=1,…,R

15. $ST_{[r]}^{k}, PT_{[r]}^{k}, S_{[r]}^{k}, C_{i}^{k} \ge 0$ — i=1, ….,n; k=1, ….,m; r=1,…,R

16. $Z \le X_2$

17. $Z \le MX_1$

18. $Z \ge X_2 - M(1 - X_1)$