Date:2022-01-04

Continue with the interim report, finish the inner loop algorithm section:

4.2.5 Inner loop Algorithm

An inner loop algorithm will be proposed to solve $C3$. As the power segment of (4) will be set in ascending order, the maximum number of power segment of EH model for each SWIPT-Supported D2D link i will be determined by the maximum received power at the receiver i. According to equation (3), the maximum received power can be calculated when the power splitting ratio is 1 and the transmission power is maximum:

$$P_{i,max}^{R} = P_{max} h_i^D + P_k^c h_{k,i} + N_0 \qquad (16)$$

As shown in Algorithm 2, this algorithm will start to traverse all the SWIPT-Supported D2D links from group $EnaD$ and each sub-partner selection PS_i^D from PS based on the results of the pre-matching algorithm. For each SWIPT-Supported D2D link i paired with a CUE k, the maximum received power will be obtained using equation (16) to calculate the maximum number of power segment N_{max}. Then it will start to traverse all the power segment of the piecewise linear model, and for each power segment j, first initialize the iteration step t as 0 and initialize the maximum number of iterations for simulation. Then, before any iteration, there will be an initialization for G_i^D which is usually very small and set as random positive number before any iteration in this project, the initial value of θ_i can be obtained by calculating (13). Based on the result of [18], $C2$ can be thought of as a function like: $F(p) = \max\{N(x,y) - pD(x,y)\}$ where p is G_i^D, x and y are θ_i and P_i^D respectively, $N(x)$ and $D(x)$ can be replaced with T_i^D and EC_i^D respectively in the optimization problem. And the target is to find a p_n such that $F(p_n) < \phi$ where ϕ is an established boundary which will be initialized as a certain positive number before any iteration. According to the method mentioned in [18], to solve nonlinear programming problems, for each iteration of this algorithm, first, use the value of θ_i at t_{th} step of iteration to calculate P_i^D at $(t+1)_{th}$ iteration step , then use the obtained P_i^D to calculate θ_i at $(t+1)_{th}$ iteration step. Then if $F(p_n) < \phi$, then the iteration should stop, the transmission power, power splitting ratio and EE for each SWIPT-Supported D2D i at j_{th} power segment of EH can be obtained. Otherwise, the iteration process continues, for each iteration, update G_i^D and all the Lagrange multipliers.

ALGORITHM 2: INNER LOOP ALGORITHM

Algorithm 2 Inner loop algorithm		
Input	:	$EnaD, PS$
Output	:	$\theta_{i,j}, P_{i,j}^{D}, EE_{i,j}^{D}$
Step 1	:	$for\ i \in EnaD\ do$
Step 2	:	$for\ k \in PS_i^P\ do$
Step 3	:	Calculate $P_{i,max}^{R}$ using (5), and obtain N_{max} using function A
Step 4	:	$for\ j = 1{:}\ N_{max}\ do$
Step 5	:	Initialize P_i^D as $P_i^D(0), G_i^D$ as $G_i^D(0)$
Step 6	:	Initialize t=0, I, ϕ
Step 7	:	obtain $\theta_i(t)$ by calculating (13) using $P_i^D(0)$
Step 8	:	$while\ t < I$
Step 9	:	Obtain $P_i^D(t+1)$ using $\theta_i(t)$ to calculate (14)
Step 10	:	Obtain $\theta_i(t+1)$ using $P_i^D(t+1)$ to calculate (13)
Step 11	:	$if\ T_i^D\left(\theta_i(t+1), P_i^D(t+1)\right) - G_i^D(t) EC_i^D\left(\theta_i(t+1), P_i^D(t+1)\right) <$ $\phi\ then$
Step 12	:	$\theta_{i,j} = \theta_i(t+1), P_{i,j}^D = P_i^D(t+1), EE_{i,j}^D = G_i^D(t)$
Step 13	:	$break\ iteration$
Step 14	:	$else$
Step 15	:	Update all the Lagrange multipliers using (15)
Step 16	:	$G_i^D(t+1) = \frac{T_i^D\left(\theta_i(t+1), P_i^D(t+1)\right)}{EC_i^D\left(\theta_i(t+1), P_i^D(t+1)\right)}$ $continue$
Step 17	:	$endif$
Step 18	:	$t = t+1$
Step 19	:	$end\ while$
Step 20	:	$end\ for$
Step 21	:	$end\ for$
Step 22	:	$end\ for$

Each SWIPT-Supported D2D link i at different power segment j of the piecewise linear EH model will have the transmission power $P_{i,j}^D$, power splitting ratio $\theta_{i,j}$ and $EE_{i,j}^D$ after using the inner algorithm.

The pseudocode of the inner loop algorithm based on my understanding has been implemented:

Algorithm 2 Inner loop algorithm		
Input	:	$EnaD, PS$
Output	:	$\theta_{i,j}, P_{i,j}^{D}, EE_{i,j}^{D}$
Step 1	:	$for\ i \in EnaD\ do$
Step 2	:	$for\ k \in PS_i^P\ do$
Step 3	:	Calculate $P_{i,max}^{R}$ using (5), and obtain N_{max} using function A
Step 4	:	$for\ j = 1{:}\ N_{max}\ do$
Step 5	:	Initialize P_i^D as $P_i^D(0), G_i^D$ as $G_i^D(0)$
Step 6	:	Initialize t=0, I, ϕ
Step 7	:	obtain $\theta_i(t)$ by calculating (13) using $P_i^D(0)$
Step 8	:	$while\ t < I$
Step 9	:	Obtain $P_i^D(t+1)$ using $\theta_i(t)$ to calculate (14)
Step 10	:	Obtain $\theta_i(t+1)$ using $P_i^D(t+1)$ to calculate (13)
Step 11	:	$if\ T_i^D\left(\theta_i(t+1), P_i^D(t+1)\right) - G_i^D(t) EC_i^D\left(\theta_i(t+1), P_i^D(t+1)\right) <$ $\phi\ then$
Step 12	:	$\theta_{i,j} = \theta_i(t+1), P_{i,j}^D = P_i^D(t+1), EE_{i,j}^D = G_i^D(t)$
Step 13	:	$break\ iteration$
Step 14	:	$else$
Step 15	:	Update all the Lagrange multipliers using (15)
Step 16	:	$G_i^D(t+1) = \frac{T_i^D\left(\theta_i(t+1), P_i^D(t+1)\right)}{EC_i^D\left(\theta_i(t+1), P_i^D(t+1)\right)}$ $continue$
Step 17	:	$endif$
Step 18	:	$t = t+1$
Step 19	:	$end\ while$
Step 20	:	$end\ for$
Step 21	:	$end\ for$
Step 22	:	$end\ for$

The implemented code is also given as following:

```
function [lambda,P_iD,EE]=inner(D2D,CUE,EhaD,Sid,I,phi,hiD,hki,hiB,hkc)
%established parameter
Pmax=0.1995262315;
Pkc=0.1995262315;
Pth=[10 100 230.06 57368]*10^(-6);
kj=[ 0 0.3899 0.6967 0.1427];
b=[0 -1.6613 -19.1737 108.2778]*10^(-6);
N0=1*10^(-13);
N1=1*10^(-13);
pass_loss=3;
Tmin_D=2;
Tmin_C=1;
lambda=[];
PiD=[];
EE=[];

s1=1*10^(-5);
s2=1*10^(-5);
s3=1*10^(-5);
s4=1*10^(-5);
s5=1*10^(-5);
for i=1:size(EhaD,2)
    PS_iD=Sid{i,1};
    hki_sub=hki{i,1};
    hD=hiD(EhaD(i));
    hDB=hiB(EhaD(i),1);
    for k=1:size(PS_iD,1)
        CUE_point=PS_iD(k,:);
        CUE_location=location_CUE(CUE,CUE_point);
        hkc_=hkc(CUE_location,1);
        hki_=hki_sub{k};
        PiR_max=Pmax*hD+Pkc*hki_+N0;
        [EH,Nmax]=EH_model(PiR_max);
        for j=1:Nmax
            t=1;
            PiD_iteration=[];
            QiD_iteration=[];
            lambda_iteration=[];
            %initialize PiD(0)
            PiD_iteration(t)=rand(1)*0.2;

            %initialize Q
            QiD_iteration(t)=rand(1);

            %initialize all the lagrange multipliers
            alpha=0.1;
            beta=0.1;
            gamma=0.1;
            delta=0.1;
            in=0.1;

            %EH coefficient initialization

            kj_=kj(j);
            %calculate the initial power splitting ratio
```

```
lambda_iteration(t)=lambda_fix_PiD(PiD_iteration(t),alpha,beta,gamma,delta,in,hD
,hki_,kj_,QiD_iteration(t));
            a=lambda_iteration(t)
            while t<=I

 PiD_iteration(t+1)=PiD_fix_lambda(lambda_iteration(t),alpha,beta,gamma,delta,in
,hD,hki_,hkc_,hDB,kj_,QiD_iteration(t));

 lambda_iteration(t+1)=lambda_fix_PiD(PiD_iteration(t+1),alpha,beta,gamma,delta,
in,hD,hki_,kj_,QiD_iteration(t));
                t=t+1;
            end
        end
                PiR=received_power(lambda_iteration(t+1),
PiD_iteration(t+1),hD,hki_);
                EH=EH_model(PiR);

 T_D=Throughput_D(lambda_iteration(t+1),PiD_iteration(t+1),hD,hki_);
                EC_iD=Energy_Consumption(PiD_iteration(t+1),EH);
                T_C=Throughput_C(hkc,hiB,PiD_iteration(t+1));
                if T_D-EC_iD*QiD_iteration(t)<phi
                    lambda(i,j)=lambda_iteration(t+1);
                    PiD(i,j)=PiD_iteration(t+1);
                    EE(i,j)=QiD_iteration(t);
                    break;
                else
                    %update all the lagrange multipliers
                    alpha_=alpha-s1*(PiD_iteration(t+1)-Pmax);
                    alpha=max([0 alpha_]);

                    beta_=beta-s2*(lambda_iteration(t+1)-1);
                    beta=max([0 beta_]);

                    gamma_=gamma-s3*(T_D-Tmin_D);
                    gamma=max([0 gamma_]);

                    delta_=delta-s4*(T_C-Tmin_C);
                    delta=max([0 delta_]);

                    in_=in-s5*(PiR-Pth(1));
                    in=max([0 in_]);

                    %update Q
                    QiD_iteration(t+1)=T_D/EC_iD;
                    continue;
                end
                t=t+1;
            end
        end
    end
end
```

However there are some parameters initialization problems which need to be solved:

TABLE I

SIMULATION PARAMETERS

Simulation parameter	Value
Cell radius R	200 m
Number of D2D links N	10∼30
Number of CUEs M	10∼50
D2D communication distance range r	10∼60 m
Pathloss exponent α	3 [47]
Receiver power segment $[P_{th}^0, P_{th}^1, P_{th}^2, P_{th}^3]$	[10, 57368, 230.06, 100] uw
Coefficient $[k_0, k_1, k_2, k_3, k_4]$	[0, 0.3899, 0.6967, 0.1427]
Intercept $[b_0, b_1, b_2, b_3, b_4]$	[0,-1.6613,-19.1737,108.2778]
Maximum harvestable power P_{max}^{EH}	250 uw
Max transmission power for any user $\mathrm{P}_{\mathrm{max}}$	23 dBm
CUE transmission power P_k^C	23dBm
Noise power N_0, N_1	-100 dBm [11], [13], [28]
Circuit power consumption $\mathrm{P}_{\mathrm{cir}}$	20 dBm
Throughput requirement for D2D link $\mathrm{T}_{\mathrm{min}}^{\mathrm{D}}$	2 bit/s/Hz
Throughput requirement for cellular link $\mathrm{T}_{\mathrm{min}}^{\mathrm{C}}$	1 bit/s/Hz

The coefficients are not correct, they didn't quite relate to each other. So i contact the the authors, and the corresponding reference paper needs to be reviewed.

For now, using the initialized parameters, the result will not make sense: ***complex numbers***