Ckt name	Argument	Chip Width	Chip Height	HPWL	Runtime(ms)
n10	-a	550	430	19954	915044
n10	-w	351	1163	6726	765858
n10	-c	431	556	14650	793275
n30	-a	418	614	38980	7620038
n30	-w	224	2131	14384	8234123
n30	-c	404	742	38884	8097716
n50	-a	548	435	122686	22559411
n50	-w	219	2635	17277	28050472
n50	-c	237	899	339822	23096276
n100	-a	445	654	180959	103127851
n100	-w	267	2885	41910	106104267
n100	-c	237	1236	79110	84183557
n200	-a	423	1215	513945	432412600
n200	-w	370	3929	88905	443665575
n200	-c	337	1374	268423	444452134
n300	-a	1014	542	1.18578e+06	857427786
n300	-w	645	5597	275242	1083667833

n300	-c	292	1931	314114	84183557

Strategy to optimize the annealing engine:

1. I generate a random initial polish expression by shuffle and check until it is valid and then normalize it. So that it can result in more random distributed movement in the future, not constrained by the move option in the beginning.

```
while(true){
   srand(time(0));
```

```
   random_shuffle(polish.begin(),polish.end()-2,randomfunc); //(no need to
change last 2 operators)

   //print_polish(polish);
   if(isvalid(polish)){
       break;
   }
   }
    normalized(polish,-1);
```

2. 2. I initialize k based on average absolute cost from the first 50 steps according to different scales of the circuit. And then set Boltze to be 0.98 acception rate for the first 50 moves.

```
       for(int i=0;i<50;i++){
           Move(polish);
           double temp=generate_polish(polish).first;
           diffsum+=max(temp-curent_T,curent_T-temp);
           curent_T=temp;
       }
       double avgcost=diffsum/50;
       k=avgcost/(T*log(1.0/0.98));
       cout<<"value of k"<<k<<" "<<curent_T;
```

3. I try a number of different cooling schedule rate and finally find 0.98 fits best to find an optimal solution while not cost too much time.
4. I tried to adjust the probability of different move options but the result is not promising. What I find is although I give 3 options equal probability, but the second options probability generates an invalid move so that the total movement times calculated in the end is approximately 2 : 1 : 2 for three different movement options. And the total accept moves generated by these steps are 4:1:2. Maybe in the future I would implement this.
5. Parameters I used:

```
    int num_of_move_per_timestamp=num_of_Block;
    int timestamp=0;
    int k=40;
    double diffsum=0;
    double T=40000;
    double T_frezz=0.098;
    double coolingfactor=0.98;
    vector<int>moves(3,0);
```