DP+DME

输入：抽象的时钟树

输出：完成buffer插入和布线的时钟树

算法思路：利用*动态规划（DP）*的思路，把时钟树buffer的插入问题分解为类似的子问题，即每两个节点的父节点合并问题。同时，这个过程是自底向上的，只有完成了当前父节点的合并，才能进行上一级的父节点合并。而对于每一个子问题，使用*DME算法*来完成节点合并。

问题建模

子问题描述：定义该子问题的解集合Γ_p，其中每个解γ_p={ S,M,D_{min},D_{max},C,P }，代表一种合并方案。S是合并节点p的slew，M是子节点的合并style，D_{min},D_{max}分别是合并节点p到其对应sink的最小和最大延迟，C是节点p的负载电容，P是合并方案的功耗开销。

子问题模型：

该子问题的解方案可以分成两类，有buffer插入的情况和没有buffer插入的情况

考虑如下情况，合并u和v得到p。其中用γ_u和γ_v表示u和v的解，用$\gamma_{u\to p}$和$\gamma_{v\to p}$表示临时解。以下s代表自定义的输入slew，$s\in\{s_0,s_1,...,s_n\}$，$\{s_0,s_1,...,s_n\}$是等差数列

1.对于没有buffer插入的情况（用$u\to p$说明）：

$$
\begin{gathered}
S(\gamma_{u\to p})=S(\gamma_u)\\
D_{min}(\gamma_{u\to p})=D_{min}(\gamma_u)\\
D_{max}(\gamma_{u\to p})=D_{max}(\gamma_u)\\
C(\gamma_{u\to p})=C(\gamma_u)+c\times l_{pu}\\
P(\gamma_{u\to p})=P(\gamma_u)
\end{gathered}
$$

2.对于有buffer插入的情况（用$u\to p$说明）：

$$
\begin{gathered}
S(\gamma_{u\to p})=s\\
C_b(\gamma_{u\to p})=CBuf(S(\gamma_{u\to p}),S(\gamma_u))\\
D_{min}(\gamma_{u\to p})=DBuf(S(\gamma_{u\to p}),C_b(\gamma_{u\to p}))+D_{min}(\gamma_u)\\
D_{max}(\gamma_{u\to p})=DBuf(S(\gamma_{u\to p}),C_b(\gamma_{u\to p}))+D_{max}(\gamma_u)\\
C(\gamma_{u\to p})=C_{in}+c\times d_{pb}\\
P(\gamma_{u\to p})=P(\gamma_u)+PBuf(S(\gamma_{u\to p}),C_b(\gamma_{u\to p}))
\end{gathered}
$$

对于以上两种情况，当把$\gamma_{u\to p}$和$\gamma_{v\to p}$合并时，有以下关系：

$$
\begin{gathered}
D_{min}(\gamma_p)=min(D_{min}(\gamma_{u\to p}),D_{min}(\gamma_{v\to p}))\\
D_{max}(\gamma_p)=max(D_{max}(\gamma_{u\to p}),D_{max}(\gamma_{v\to p}))\\
C(\gamma_p)=C(\gamma_{u\to p})+C(\gamma_{v\to p})
\end{gathered}
$$

可能解的可行性检查：

1.对于没有buffer插入的情况：

$$
\begin{gathered}
l_{pu} = d_{pu} \\
S(\gamma_{u\to p}) = S(\gamma_{v\to p}) = s \\
Skew(\gamma_p) = D_{max}(\gamma_p) - D_{min}(\gamma_p) \leq skewBnd
\end{gathered}
$$

2.对于有buffer插入的情况：

$$
\begin{gathered}
S(\gamma_{u\to p}) = S(\gamma_{v\to p}) = s \\
Skew(\gamma_p) = D_{max}(\gamma_p) - D_{min}(\gamma_p) \leq skewBnd \\
C_b(\gamma_{u\to p}) \geq C(\gamma_u) \\
C_b(\gamma_{v\to p}) \geq C(\gamma_v)
\end{gathered}
$$

如果可能的解通过可行性检查，更新解方案γ_p中的所有信息，并添加到解集合Γ_p中。

其中γ_p剩下的的S、P、M计算如下：

S、P:

$$
\begin{gathered}
S(\gamma_p) = s \\
P(\gamma_p) = P(\gamma_{u\to p}) + P(\gamma_{v\to p})
\end{gathered}
$$

M:

对于没有buffer插入的情况，M包含d_{pu}和d_{pv},计算如下：

$$
d_{pu} = d_{pv} = L/2
$$

其中L是u和v之间的最短曼哈顿距离

对于有buffer插入的情况，M包含d_{bu}、d_{bv}、d_{pu}和d_{pv}，计算如下：

$$
\begin{aligned}
d_{bu} &= \frac{C_b(\gamma_{u\to p}) - C(\gamma_u)}{c} \\
d_{bv} &= \frac{C_b(\gamma_{v\to p}) - C(\gamma_v)}{c} \\
d_{pu} &= max(\frac{L - d_{bu} - d_{bv}}{2}, 0) + d_{bu} \\
d_{pu} &= max(\frac{L - d_{bu} - d_{bv}}{2}, 0) + d_{bv}
\end{aligned}
$$

问题求解

子问题求解思路：

对于每一个s，控制对应的可行解的数量，记这个数量为K

在求解过程中调整等差数列的公差和K,实验并观察不同情况下时钟树的结果。

复现代码

```python
# -*- coding: utf-8 -*-
# @Author: mac
# @Date:   2019-10-14 15:57:10
# @Last Modified by:   mac
# @Last Modified time: 2019-10-16 10:26:39
# 此版本中的查找表是自己定义的，没有采用DME来确定合并节点
import math
from scipy import interpolate
import copy
import numpy as np
import matplotlib.pyplot as plt

# 论文中的一些自定义参数
capacitance_per_unit = 2e-6 #fF/nm
buffer_input_capacitance = 10 #fF
skew_bound = 200 #ps
K = 6
c_max = 60 #fF #最大负载电容约束。这个论文中没提到，但是需要加上

# 定义solution类
class solution:
  def __init__(self,attributes,location,lvl=0,solution_type=0):
    self.S = attributes[0]
    self.M = attributes[1]
    self.D_min = attributes[2]
    self.D_max = attributes[3]
    self.C = attributes[4]
    self.P = attributes[5]

    self.location = location

    # type = 0 represents un-buffered solution
    # type = 1 represents buffered solution
    self.type = solution_type

    self.level = lvl
    self.queue = []
  def add_to_queue(self,sub_solution):
    self.queue.append(sub_solution)

# 读入ispd中的sink信息
def readSinkLocations(sink_num,slew_list=list(range(50,60,2)),
file_path="s1r1.txt"):
    f = open(file_path, "r")
    bounds = f.readline().split(" ")
    sink_solutions = []

    # skip second Line
    f.readline()
```

```python

    num_sinks_in_file = int(f.readline().split(" ")[2])

    for sink in range(sink_num):
        data = f.readline().split(" ")
        location = [int(data[1]), int(data[2])]
        a_solution = []
        for s in slew_list:
          a_solution.append(solution(attributes=[s,[],0,0,10,0],location=[location[0], location[1]]))
        sink_solutions.append(a_solution)
    f.close()
    return sink_solutions

#初始化CBuf查找表和插值函数
def init_cbuf(x,y):
  z = [[60,65,70,75,80],[55,60,65,70,75],[50,55,60,65,70],[45,50,55,60,65],[40,45,50,55,60]]
  f = interpolate.interp2d(x, y, z,kind='cubic')
  return f
#初始DBuf查找表和插值函数
def init_dbuf(x,y):
  z = [[500,550,600,650,700],[550,600,650,700,750],[600,650,700,750,800],[650,700,750,800,850],[700,750,800,850,900]]
  f = interpolate.interp2d(x, y, z,kind='cubic')
  return f
#初始化PBuf查找表和插值函数
def init_pbuf(x,y):
  z = [[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7],[4,5,6,7,8],[5,6,7,8,9]]
  f = interpolate.interp2d(x, y, z,kind='cubic')
  return f
#确定输入slew和负载电容的范围和步长，并初始化CBuf, DBuf, PBuf
def initialize():
  slew_bd = np.arange(50,100,10) #ps
  cap_bd = np.arange(50,100,10) #fF
  cbuf = init_cbuf(slew_bd,slew_bd)
  dbuf = init_dbuf(slew_bd,cap_bd) #ps
  pbuf = init_pbuf(slew_bd,cap_bd) #uW
  return cbuf,dbuf,pbuf

#根据有buffer插入的方式生成父节点
def get_with_buffer_solution(solution_u,solution_v,slew,level,CBuf,DBuf,PBuf):
  length = math.sqrt((solution_u.location[0] - solution_v.location[0])**2 + (solution_u.location[1] - solution_v.location[1])**2)
  x = (solution_u.location[0] + solution_v.location[0])/2
  y = (solution_u.location[1] + solution_v.location[1])/2
  x_delta = (solution_u.location[0] - solution_v.location[0])
```

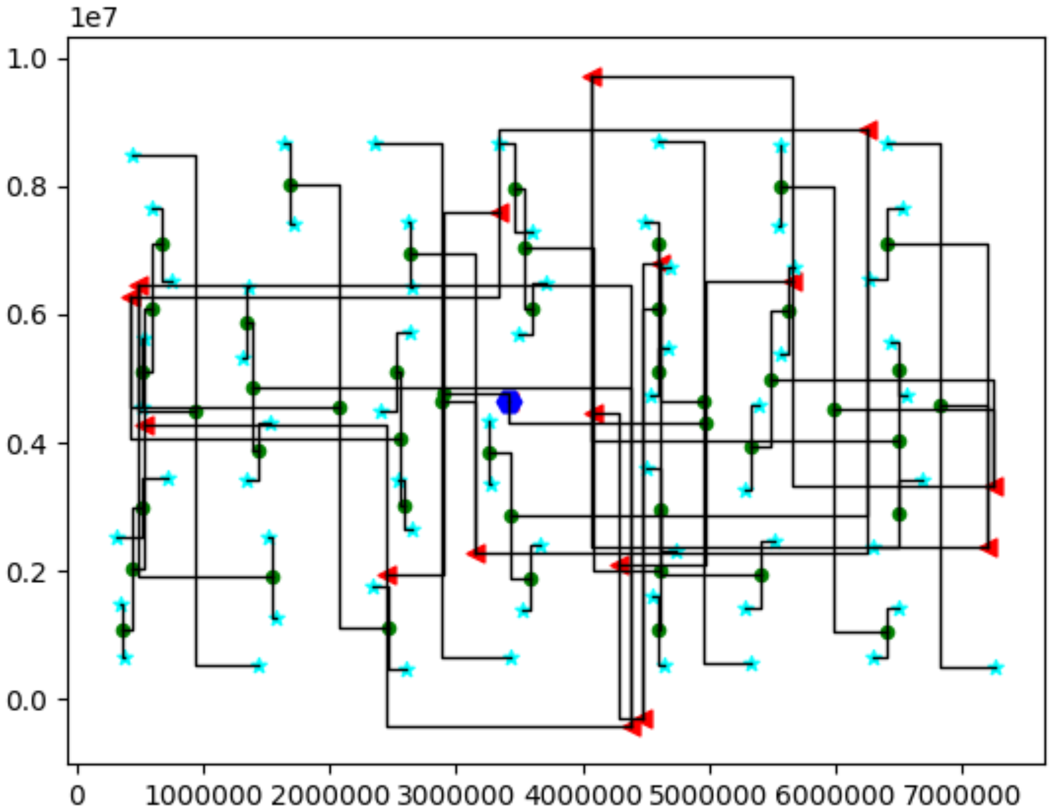
```
  y_delta = (solution_u.location[1] - solution_v.location[1])
  c_bu = CBuf(slew,solution_u.S)
  c_bv = CBuf(slew, solution_v.S)
  d_bu = (c_bu-solution_u.C)/capacitance_per_unit
  d_bv = (c_bv-solution_v.C)/capacitance_per_unit
  d_pb = max((length-d_bu-d_bv)/2,0)
  p_m = [d_bu,d_bv,d_bu+d_pb,d_bv+d_pb]
  p_location = [x-(d_bu-d_bv)*x_delta/(2*length), y-(d_bu-d_bv)*y_delta/(2*length)]
  D_bu = DBuf(slew, c_bu)
  D_bv = DBuf(slew, c_bv)
  d_min_u = D_bu + solution_u.D_min
  d_min_v = D_bv + solution_v.D_min
  d_max_u = D_bu + solution_u.D_max
  d_max_v = D_bv + solution_v.D_max
  p_D_min = min(d_min_u + d_min_v)
  p_D_max = max(d_max_u + d_max_v)
  p_C = (buffer_input_capacitance + capacitance_per_unit*d_pb) + (buffer_input_capacitance + capacitance_per_unit*d_pb)
  p_P = (PBuf(slew, c_bu) + solution_u.P) + (PBuf(slew, c_bv) + solution_v.P)

  attributes = [slew,p_m,p_D_min,p_D_max,p_C,p_P]
  solution_p = solution(attributes=attributes,location=p_location,lvl=level,solution_type=1)

  # check feasibility of solution
  if (solution_u.S == slew) and (solution_v.S ==slew) and ((solution_p.D_max - solution_p.D_min) <= skew_bound) and (solution_u.C <= c_bu) and (solution_v.C <= c_bv) and solution_p.C < c_max:
    return solution_p,True
  else:
    return solution_p,False
# 根据没有buffer插入的方式生成父节点
def get_without_buffer_solution(solution_u,solution_v,slew,level):
  d_bu = 0
  d_bv = 0
  x = (solution_u.location[0] + solution_v.location[0])/2
  y = (solution_u.location[1] + solution_v.location[1])/2
  p_location = [x,y]
  half_length = math.sqrt((solution_u.location[0] - solution_v.location[0])**2 + (solution_u.location[1] - solution_v.location[1])**2)/2
  p_m = [d_bu,d_bv,half_length,half_length]
  p_D_min = min(solution_u.D_min,solution_v.D_min)
  p_D_max = max(solution_u.D_max,solution_v.D_max)
  p_C = (solution_u.C + capacitance_per_unit*half_length) + (solution_v.C + capacitance_per_unit*half_length)
```

```python
  p_P = solution_u.P + solution_v.P

  attributes = [slew,p_m,p_D_min,p_D_max,p_C,p_P]
  solution_p = solution(attributes=attributes,location=p_location,lvl=level,solution_type=0)

  # check feasibility of solution
  if (solution_u.S == slew) and (solution_v.S == slew) and ((solution_p.D_max - solution_p.D_min) <= skew_bound) and solution_p.C < c_max:
    return solution_p,True
  else:
    return solution_p,False

# 根据不同slew产生所有可能的父节点solution，并挑选出top K个solution
def get_father_solutions(solutions1,solutions2,level,cbuf,dbuf,pbuf):
  father_solutions = []
  slew_list = list(range(50,80,2))
  for solution1 in solutions1:
    for solution2 in solutions2:
      for slew in slew_list:
        father_solution,status = get_with_buffer_solution(solution1,solution2,slew,level,cbuf,dbuf,pbuf)
        if status == True:
          father_solution.add_to_queue([solution1,solution2])
          father_solutions.append(father_solution)
  for solution1 in solutions1:
    for solution2 in solutions2:
      for slew in slew_list:
        father_solution,status = get_without_buffer_solution(solution1,solution2,slew,level)
        if status == True:
          father_solution.add_to_queue([solution1,solution2])
          father_solutions.append(father_solution)

  father_solutions.sort(key=lambda x:x.P)
  return father_solutions[0:K]

#画父节点和其子节点的连接关系
def draw_connection(solution,level):
  root_loc = solution.location
  root_type = solution.type
  child1_loc = solution.queue[0][0].location
  child2_loc = solution.queue[0][1].location
  plt.plot([root_loc[0],root_loc[0],child1_loc[0]],[root_loc[1],child1_loc[1],child1_loc[1]],linewidth=1,c='k')
  plt.plot([root_loc[0],root_loc[0],child2_loc[0]],[root_loc[1],child2_loc[1],child2_loc[1]],linewidth=1,c='k')
```

```python
  if root_type == 1:
    plt.scatter(root_loc[0],root_loc[1],marker="<",c='r',s=40)
  else:
    plt.scatter(root_loc[0],root_loc[1],marker="o",c='g',s=20)
  if (level==1):
    plt.scatter(root_loc[0],root_loc[1],marker="H",c='b',s=80)

# 画所有连接关系和solution的类型及位置
def draw(solution,level):
  total_level = 6
  if level != total_level:
    level = level + 1
    draw_connection(solution,level)
    draw(solution.queue[0][0],level)
    draw(solution.queue[0][1],level)

# 主函数
def main():

  sink_num = 64
  # initialize lut
  cbuf,dbuf,pbuf=initialize()

  # initialize sink solution
  solution_sinks = readSinkLocations(sink_num=64)

  # final solutions
  sub_solution = copy.deepcopy(solution_sinks)
  root_solution = []
  next_solutions = []

  total_level = int(math.log2(sink_num))
  print("initialize done")

  # generate all solutions
  for level in range(1,total_level+1):
    father_num = int(sink_num/2**level)
    next_solutions = []

    if level != 1:
      for i in range(father_num):
        # get top K solutions for each pairs
        father_solution = get_father_solutions(sub_solution[2*i], sub_solution[2*i+1], level,cbuf,dbuf,pbuf)
        next_solutions.append(father_solution)
    else:
      for i in range(father_num):
        # get top K solutions for each pairs of sinks
```

```python
        father_solution = get_father_solutions(solution_sinks[2*i], solution_sinks[2*i+1], level,cbuf,dbuf,pbuf)
        next_solutions.append(father_solution)
    sub_solution = next_solutions

    if len(next_solutions) == 1:
      root_solution = next_solutions[0]

    print("generate {}*{} solutions at {}th level".format(len(next_solutions),K,total_level - level + 1))

  # select best solution combination in Top K
  roots = root_solution[0]
  print("start plotting")
  # create plot
  fig = plt.figure()
  ax = fig.add_subplot(1,1,1)

  # draw connections and solution
  draw(roots,level=0)

  # plot sinks
  for sink in solution_sinks:
    plt.scatter(sink[0].location[0],sink[0].location[1],marker='*',s=30,c='cyan')

  plt.show()

if __name__ == '__main__':
  main()
```

复现结果


1e7
1.0
0.8
0.6
0.4
0.2
0.0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000