PD PA3 Global Placement

By R08943094 吳牧庭

Data Structure

I use the data structure given by the sample codes. Nothing new is added.

Algorithm

2 algorithms are implemented in this pa: analytical placement and simulated annealing.

- Analytical placement

  The objective is to minimize the cost function $W + \lambda D$ using the built-in gradient optimizer. LSE is applied for W while sigmoid function is applied for D. Their respective gradients are derived and shown as follows to update the variables.

$$\frac{\partial \ln(\Sigma e^{\frac{x_i}{\gamma}})}{\partial x_i} = \frac{1}{\Sigma e^{\frac{x_i}{\gamma}}} \cdot \frac{\partial \Sigma e^{\frac{x_i}{\gamma}}}{\partial x_i} = \frac{e^{\frac{x_i}{\gamma}}}{\gamma \Sigma e^{\frac{x_i}{\gamma}}}$$

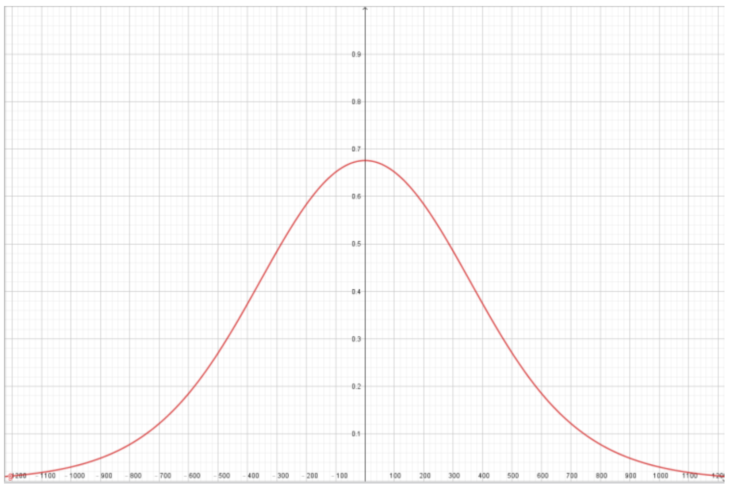
$$\frac{\partial \ln(\Sigma e^{\frac{-x_i}{\gamma}})}{\partial x_i} = \frac{1}{\Sigma e^{\frac{-x_i}{\gamma}}} \cdot \frac{\partial \Sigma e^{\frac{-x_i}{\gamma}}}{\partial x_i} = \frac{-e^{\frac{-x_i}{\gamma}}}{\gamma \Sigma e^{\frac{-x_i}{\gamma}}}$$

$$\begin{aligned} &D(x,w) \\ &= sig(\tfrac{1}{2}w + x) \cdot sig(\tfrac{1}{2}w - x) \\ &= \frac{1}{(1+e^{-\alpha(\frac{1}{2}w+x)})(1+e^{-\alpha(\frac{1}{2}w-x)})} \\ &= \frac{1}{1+e^{-\alpha(\frac{1}{2}w+x)}+e^{-\alpha(\frac{1}{2}w-x)}+e^{-\alpha w}} \end{aligned}$$

$$\begin{aligned} &\frac{\partial D(x,w)}{\partial x} \\ &= (D(x,w))^2 \times \left(-\frac{\partial D^{-1}}{\partial x}\right) \\ &= -D^2 \times (0 + -\alpha e^{-\alpha(\frac{1}{2}w+x)} + \alpha e^{-\alpha(\frac{1}{2}w-x)} + 0) \\ &= \alpha(D(x,w))^2 (e^{-\alpha(\frac{1}{2}w+x)} - e^{-\alpha(\frac{1}{2}w-x)}) \\ &= \alpha(D(x,w))^2 \left(\frac{1}{sig(\frac{1}{2}w+x)} - \frac{1}{sig(\frac{1}{2}w-x)}\right) \end{aligned}$$

  The tricky part for this algorithm is that there are too many parameters to tune. Although the result isn't satisfactory eventually, I have some observation about the parameters.

  α : This parameter is easy to tune. It turns out that setting $\alpha = \frac{1}{\ln(\frac{wb+hb}{2})^k}$ results in a smooth curve. This makes is easy for a movable module to move across bins. The figure below shows the curve of α when $k = 3$.



  γ :This parameter can be decided by the upper bound of $e^{\frac{x}{\gamma}}$. Therefore, γ equals to $\frac{x}{ln(ub)}$, where x can be substitute by $max(chip\_width, chip\_height)$.

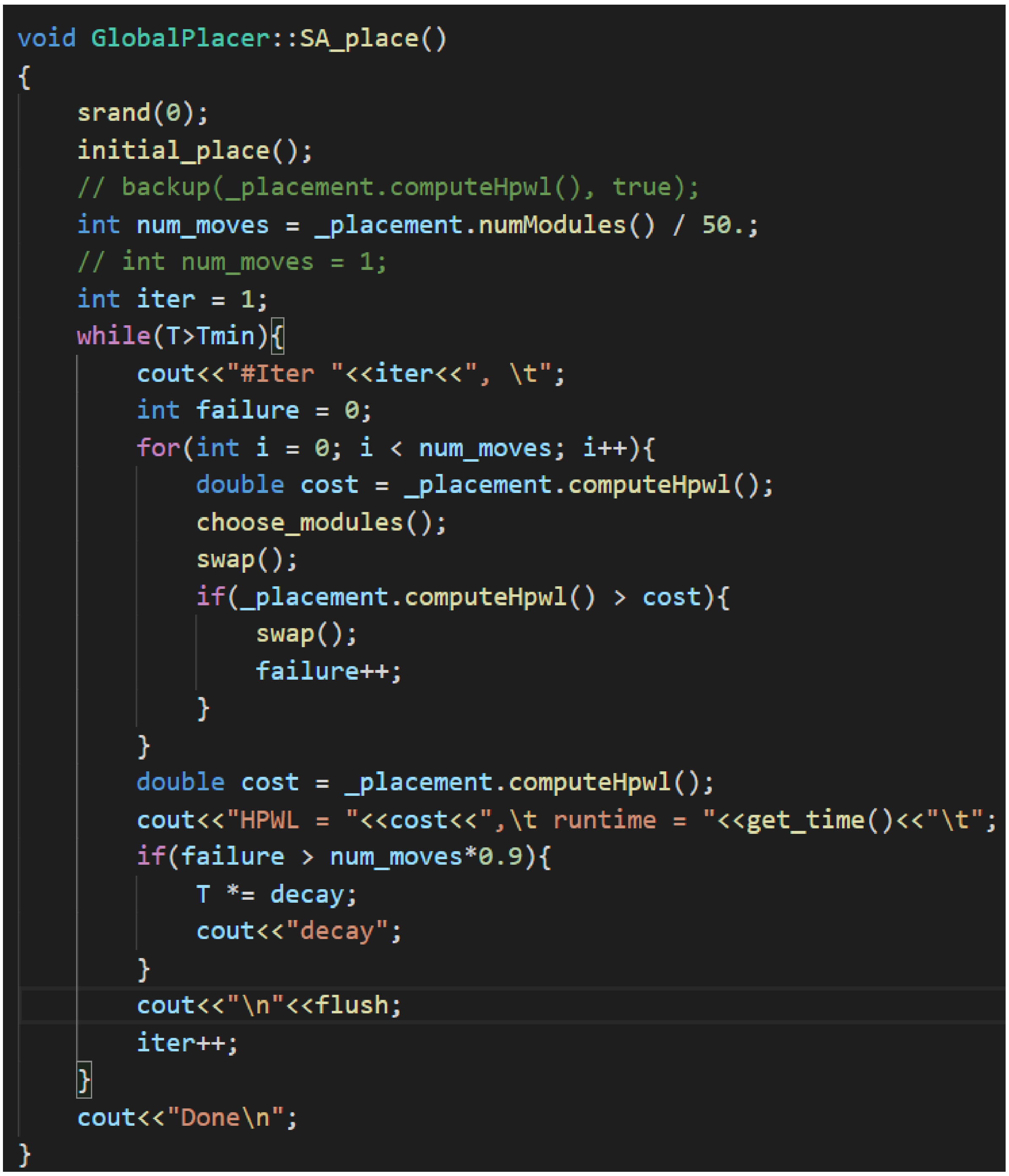
  λ : This parameter is the tricky part. We know that λ has to grow in time. However, as it

grows larger than 1, the optimization process begins to suffer. Therefore, λ has to start from an extremely low value, depending on the number of iterations the optimizer has to be working for.
Mb: This parameter stands for the desired density for each bin. It depends on how we acquire the density value. Ideally, every bin should be assigned the same number of modules.

Therefore, it's set as $\frac{\#modules}{\#bins}$.

#bins: This parameter is the most difficult for me. If the bin size is too large, it will be difficult for modules to move across bins. If it's too small, the modules become sensitive. It also depends on the maximum step size of the optimizer.
In summary, the wire length cost works just fine while the density works odd. When cost is W instead of $W + \lambda D$, modules tends to gather to the center. However, when cost is λD instead of $W + \lambda D$, modules either sticks where they are or move far away from the chip. This makes the algorithm so hard to find a sweet spot where wire length and module overlaps is minimized.

- Simulated Annealing
  Since the analytical placer fails to find a balanced solution, I turn to SA algorithm. Given an initial placement, the neighborhood structure can be found by swapping any 2 modules. The problem of SA on placement is the scalability. Due to the large input size, a lot larger than that of a floor-planning problem, there are so many possible moves that we can never go through the entire solution space. Therefore, due to the low efficiency, SA is modified into a greedy algorithm to optimize a placement with better efficiency. Unlike the conventional SA, where a temperature is given to decide the probability of going up-hill. The T value here is transformed as a minimum ratio to search across the chip. For example, T=0.3 means that only 2 modules m1 and m2 away with a distance of $|x1 - x2| > T * chip\_width$ and $|y1 - y2| > T * chip\_height$ can be regarded as a valid move. Moreover, up-hill moves are prohibited to make the algorithm greedy. Finally, after an iteration of several moves, if the cost succeeds to go down on over 10% of the moves, the T value keeps on. Otherwise, the T value will decay by a factor. The process is shown by the code below.



```cpp
void GlobalPlacer::SA_place()
{
    srand(0);
    initial_place();
    // backup(_placement.computeHpwl(), true);
    int num_moves = _placement.numModules() / 50.;
    // int num_moves = 1;
    int iter = 1;
    while(T>Tmin){
        cout<<"#Iter "<<iter<<", \t";
        int failure = 0;
        for(int i = 0; i < num_moves; i++){
            double cost = _placement.computeHpwl();
            choose_modules();
            swap();
            if(_placement.computeHpwl() > cost){
                swap();
                failure++;
            }
        }
        double cost = _placement.computeHpwl();
        cout<<"HPWL = "<<cost<<",\t runtime = "<<get_time()<<"\t";
        if(failure > num_moves*0.9){
            T *= decay;
            cout<<"decay";
        }
        cout<<"\n"<<flush;
        iter++;
    }
    cout<<"Done\n";
}
```