Algorithm 1: 网格划分

```
输入：需要划分网格的数据集 Dataset
begin
    num = Dataset 的长度
    根据 X、Y 极值分别生成间隔数为 r，c 的区间 X_range、Y_range：
    存储数据集的位置信息：
    dis_x = [ ], dis_y = [ ]
    for Dataste 的每一行 do
        for X_range 中的 r 个区间 do
            if Dataset 的 X 介于第 j 个区间 then
                dis_x[i]=j
                （把 Dataset 划分在第 i 行）
            end if
        end for
        for X_range 中的 c 个区间 do
            if Dataset 的 Y 介于第 k 个区间 then
                dis_y[j]=k
                （把 Dataset 划分在第 j 列）
            end if
        end for
    end for
    更新 Dataset 位置信息：
    Dataset['district_x'] = dis_x
    Dataset['district_y'] = dis_y
end
输出：Dataset
```

Algorithm 2: enlarge_gridshape

```
输入：网格扩大次数 iter_num
begin
    if iter_num = 0 then
        return gridshape
    else
        if 所有都被分配完毕 then
            size = (1, 1)
            return size
        else
            确保网格能够平滑扩大：
            r = gridshape[0] - 1 * iter_num
            c = gridshape[1] - 1 * iter_num
            if r ≤ 0 then
                r = 1
            end if
            if c ≤ 0 then
                c = 1
            end if
            size = (r, c), return size
        end if
    end if
end
输出：网格参数 size
```

Algorithm 3: solve

输入： 阿姨数据集 aunt, 订单数据集 order, 当前时刻 t, 求解器模型 solver_mode, 递归深度 n, 求解状态 status, 高质量阿姨标识 hq_aunt

begin

  从求解器里将求解值赋值给参数：**prob** 表示问题参数、**df** 表示解矩阵、k 是分配订单个数

  objective = Maximize(obj), prob = Problem(objective, constrains)

  prob.solve(solver=cp.GLPK_MI, qcp=True), df = pd.DataFrame(x.value),n = k

  if *status* **then**

    如果求解状态开启

    if *prob.status = 'optimal' and n ≥ 1* **then**

      if *n > max(rank)* **then**

        防止无限递归

        return None, 0, 0

      end if

      n = n+1, prob_1, df_1, m_1 = solver(*, n, *)

      if *prob_1 = None* **then**

        return prob, df, sum(x)

      else

        if *prob_1.value > prob.value* **then**

          return prob_1, df_1, m_1

        else

          return prob, df, k

        end if

      end if

    end if

    if *prob.status = 'infeasible' and n > 1* **then**

      当前深度不存在可行解

      return None, 0, 0

    end if

    if *prob.status = 'infeasible' and n = 1* **then**

      第一次求解时无可行解

      warnings.warn, prob_1, df_1, m_1 = solver(*, n, *)

      return prob_1, df_1, m_1

    end if

  else

    return prob, df, k

  end if

end

输出： 问题参数 prob, 解矩阵 df, 分配订单个数 k