这星期阅读了 Hongzhi Guo，Jiajia Liu 等人的 **Mobile-Edge Computation Offloading for Ultra-Dense IoT Networks**，这篇文章主要研究了超密集物联网中 MECO 的问题，并提出了一个双层博弈论贪婪卸载方案，大量的数值结果证实了在超密集的物联网网络中，在多个边缘服务器之间进行计算卸载的优越性能。

作者的主要贡献

1.在多用户超密集边缘服务器场景中，不仅考虑了 MDs 随机请求各种计算任务的情况，而且考虑了任务随机到达边缘服务器，边缘服务器上的计算资源动态变化的情况。

2.根据物联网 MDs 的能量消耗和计算任务的处理延迟提出了系统模型，并将超密集物联网中的最优 MECO 问题定义为一个约束优化问题，其目标是在满足给定的无线信道约束的情况下，使总体计算开销最小化。

3.作者首先给出了一个最优枚举卸载方案，该方案可以实现问题的最优解。然后考虑到它的指数计算复杂度，进一步提出了一个两层博弈理论贪婪近似卸载方案，具有更高的计算效率。

算法伪代码：

Procedure 1 a greedy approximation offloading procedure (GAOP).

for all $j \in \mathcal{M}$ **do**
  compute $\psi_{i,j}^{S}$;
  choose the SC k with the minimum $\psi_{i,j}^{S}$;
end for
associate MD i to the SC k;
for all $j \in \mathcal{M}$ **do**
  compute $\psi_{i,j}^{L}$, $\psi_{i,j}^{M}$, $\psi_{i,j}^{S}$;
  if $min\{\psi_{i,j}^{M}, \psi_{i,j}^{S}\} \geq \psi_{i,j}^{L}$ **then**
    $\lambda_{i,j} = 0$, $\mathcal{M} = \mathcal{M} \backslash \{j\}$;
  end if
end for
for all $l \in \mathcal{M}$ **do**
  compute $r_{i,l}^{M}$ and $r_{i,l}^{k}$;
  compute $\psi_{i,l}^{L}$, $\psi_{i,l}^{M}$, $\psi_{i,l}^{S}$;
  if $min\{\psi_{i,l}^{M}, \psi_{i,l}^{S}\} \leq \psi_{i,l}^{L}$ **then**
    $\mathcal{O} = \mathcal{O} \cup \{l\}$;
    execute the offloading Procedure 2;
  else
    $\lambda_{i,l} = 0$, $\mathcal{M} = \mathcal{M} \backslash \{l\}$;
  end if
end for

Procedure 2 an offloading decision procedure.

for all $m \in \mathcal{O}$ **do**
  compute $\psi_{i,m}^{M}$ and $\psi_{i,m}^{S}$;
  if $\psi_{i,m}^{M} \leq \psi_{i,m}^{S}$ **then**
    $\lambda_{i,m} = -1$;
  else
    $\lambda_{i,m} = k$;
  end if
end for

Algorithm 2 a game-theoretic greedy approximation offloading algorithm (GT-GAOA).

Input: $\mathcal{N}$, $\mathcal{M}$, $\mathcal{S}$, $T_{i,j}$, $P_{i,j}$, f_i, δ_i, f^M, f_k^S, ω^M, ω^k, σ, p_i, c, $\alpha_{i,j}^t$, $\alpha_{i,j}^e$, $\forall\, i \in \mathcal{N}$, $j \in \mathcal{M}$, $k \in \mathcal{S}$.

Output: an optimal computation offloading decision profile S, and the overall minimum computation overhead Ψ_{min}; otherwise, NIL.

Initialize the offloading decision profile for MD i $\Lambda_i = \{0, 0, ..., 0\}$, $\forall\, i \in \mathcal{N}$, the set of MDs, whose offloading decision profiles should be updated, is denoted as $U(t)$, and $U(t) = \emptyset$, the set of types of tasks offloading to the MEC servers is set to be empty, i.e., $\mathcal{O} = \emptyset$;
for each decision slot t **do**
  for all $i \in \mathcal{N}$ **do**
    call Procedure 1 to compute MD i's local optimal offloading decision profile $\Lambda_i^*(t+1)$ in next time slot $t+1$, which contains both its optimal associated SC and all its tasks' offloading decision list with the minimum computation offloading overhead;
    compute $\Psi_i(t+1)$ based on $\Lambda_i^*(t+1)$;
    if $\Psi_i(t+1) < \Psi_i(t)$ **then**
      store MD i into the set of $U(t)$;
    end if
  end for
  while $U(t) \neq \emptyset$ **do**
    each MD in $U(t)$ contends for the offloading decision profile update opportunity;
    if MD j wins the update opportunity **then**
      $\Lambda_j = \Lambda_j^*(t+1)$;
      update Λ_j in S;
      broadcast the update message to other MDs;
    else
      $\Lambda_j = \Lambda_j(t)$;
    end if
  end while
end for
compute the overall minimum computation overhead Ψ_{min} based on the calculated offloading decision profile S;
return S and Ψ_{min}.

输出：最优计算卸载决策 S，总的最小计算开销 ψ min;
参数说明：

$\lambda_{i,j} \in \{0, -1, 1, 2, ..., S\}$ MDi 设备中任务 j 的卸载决策

$\Lambda_i = \{\lambda_{i,1}, \lambda_{i,2}, ..., \lambda_{i,M}\}$ MDi 设备的全部卸载决策

$$\Psi_{i,j}(\lambda_{i,j}) = \begin{cases} \psi_{i,j}^{L}, & if\ \lambda_{i,j} = 0, \\ \psi_{i,j}^{M}, & if\ \lambda_{i,j} = -1, \\ \psi_{i,j}^{S}, & if\ \lambda_{i,j} = k, \quad 1 \le k \le S. \end{cases} \tag{14}$$

其中 λi,j = -1，将任务卸载到与 MBS 相连的 MEC 服务器。
当 λi,j = 0，将任务卸载到本地。
当 λi,j = k，卸载到与 SCk 相连的 MEC 服务器。
其中 ψij 为计算开销。

代码分析：

(1) Algorithm2:

1.初始化：初始化每个终端设备 MD 上的一个工作流的所有任务（假设任务数为 taskNum 个）设置为不卸载{0，0，…，0},并且设置需要更新卸载决策文件的 MDs 集合为空，和卸载到 MEC 服务器的任务类型设置为空

2.对于每一个 MD i 上的一个工作流：按照 Procedure1 计算 MD i 的局部最优卸载决策（这个卸载决策具有最小的优化目标值），如果新计算出来的 MD i 的优化目标值小于之前的优化目标值，则将该 MD i 添加到博弈更新圈。

3.只要博弈更新圈不为空：从博弈更新圈中随机挑选一个 MD i ，赋予 i 更新卸载决策的机会，i 进行更新，并广播给其他 MD（因为该更新的 MD 会占用公共资源），保存 i 更新后的卸载策略。博弈圈中的其他 MD 暂不更新。

4.基于计算卸载决策 S 计算整体最小的优化目标值（eg:Time、Energy）。返回 S 与最小优化目标值。

(2) Procedure1:

1 对于 MDi 工作流中的每一个卸载任务，先计算在云服务器上的开销，选择开销最小的云服务器 SC k 和 MDi 相匹配。

2 对于 MDi 工作流中的每一个卸载任务，分别计算在在本地、边缘服务器、云服务器上的开销 a，b，c。

3 找出不卸载的任务：如果当前 task 在本地执行优化目标值最小,则设置它的卸载策略为 0（不卸载），卸载的 task 放在一个集合 M 中。

Procedure2：

4 再遍历 M 集合，分别计算该任务 task 卸载到云服务器、边缘服务器的优化目标值 c,d。

5 若 c < d 卸载到云服务器，反之，卸载到边缘服务器。