Search methods

I. Mathematical Models

Given a poset $P = \{\Omega, C_{parallel}, C_{partial}\}$, where Ω is an anchor function in order to give each task a sequence number. Its domain of definition is $1, \cdots, n.$ and $C_{parallel} = (i, j)$ where $\Omega(i)$ and $\Omega(j)$ is independent; $C_{partial} = (i < j)$, where $\Omega(i)$ must be executed before $\Omega(j)$. The poset P can be regarded as the temporal order of tasks. And we give N as the agents number with difference action, and l as the place we interested. We want to find out the minimum execution time of the P with agents N.

To go further, we give the table of definition for these question models.

Variable	Variable definition	
Name	definition	range
P	partial order set	
m	number of agents	$\mathcal{N}$
n	number of tasks	$\mathcal{N}$
o	number of serves type agent can provide	$\mathcal{N}$
Ω	anchor function	
$r_{i,j,k,l}$	agent i execute task j providing serve l in the order k	$\{0, 1\}$
t_j	begin time of task j	$t_j > 0$
p_j	continue time of task j	$p_j > 0$
$a_{j,l}$	number of servey l task j needed.	$\mathcal{N}$
$b_{i,l}$	type of servey l that agent i can provide	$\{0, 1\}$
v_i	velocity of agent i	$v_i > 0$
$e_{i,j_1,j_2,k}$	agent i go from the place of j_1 to j_2 at kth task	$\{0, 1\}$
dis_{j_1,j_2}	distance from task j_1 to task j_2	$dis_{j_1,j_2} > 0$
$dis_{i,j}$	distance from initial i to task j	$dis_{i,j} > 0$

TABLE I
VARIABLE DEFINITION

$$\min_{r_{i,j,k,l}, t_j} \quad \max(t_j + p_j)$$

s.t.

Temporal constraint of tasks:

$$t_{j_1} + p_{j_1} \leqslant t_{j_2} \quad \forall (j_1, j_2) \in C_{partial} \tag{1}$$

provide enough serves for the task j

$$\sum_{i=1}^{m} \sum_{k=1}^{n} r_{i,j,k,l} b_{i,l} = a_{j,l} \quad \forall j, l \tag{2}$$

one agent can only provide the serve it has:

$$r_{i,j,k,l} \leqslant b_{i,l} \quad \forall i, j, k, l \tag{3}$$

One agent can execute one task no more than once:

$$\sum_{k=1}^{n} \sum_{l=1}^{o} r_{i,j,k,l} \leqslant 1 \forall i, j \tag{4}$$

One agent at any time can execute no more than one task:

$$\sum_{j=1}^{m} \sum_{l=1}^{o} r_{i,j,k,l} \leqslant 1 \forall i, k \tag{5}$$

One agent can execute $k+1th$ task only if it execute kth task.

$$\sum_{j=1}^{m} \sum_{l=1}^{o} r_{i,j,k,l} - \sum_{j=1}^{m} \sum_{l=1}^{o} r_{i,j,k+1,l} \leqslant 0 \qquad \forall i, k < m - 1 \tag{6}$$

Even agent need to obey the motion constrain.

$$t_{i_2} - t_{i_1} - M \sum_{l=1}^{o} r_{i,j_1,k,l} - M \sum_{l=1}^{o} r_{i,j_2,k+1,l} \geqslant dis_{j_1,j_2}/v + p_{i_1} - 2M \forall i, j \tag{7}$$

$$t_i - M \sum_{l=1}^{o} r_{i,j,1,l} \geqslant dis_{i,j}/v - M \qquad \forall i, j \tag{8}$$

Remark 1: here I don't consider the conflict of area that we allow different tasks execute in one place at same time.

Remark 2: Here the function is to solve the prefix, and circle need a further consideration that they need to go back to initial place and the initial place is also variables to consider.

II. Baseline method for MILP

Firstly, we use MILP to solve the optimial question as the baseline, and the complexity analysis will be given here(laaaaaater).

III. Branch and bound method

Branch and bound method performs well in integer programming (but not 01 programming). Branch and bound method is an accelerated methods to recursively split the search into smaller spaces, then minimizing the goal function on these smaller spaces. When checking the lower bounds on small search space, we use the linear programming method which computing complex is P-hard. Thus, we can get the optimal value much faster than MILP method. See algorithm 1.

However, due to the constraint of equation 11 4, the feasible solution can be really sparse that means most of branch is unfeasible. And search these branches may cause a waste of time.

A. lower bound

We raise several celerity methods to get the lower bound of the branches. To a branch N fetched by node routing methods from the search tree, there are a set of tasks T_s has been already assigned and another set of tasks T_l waiting for assignment. For the assigned tasks T_s, we calculate the time without any simplification as boundary conditions. For the unassigned tasks T_l, we consider the simplified situation by only consider part of information from agent serial number i, task serial number j, task order k, sub-task type l:

1) $i + j + k$

Algorithm 1: Branch and bound

Input : The Optimal question $\mathcal{Q}$
Output: Optimal value , answer

Generate relaxation Linear programming problem $\mathcal{Q}'_0$
$Z := +\infty$ %set the best cost so far
$S := \{\mathcal{Q}'_0\}$ % generate the first branch
while $S \neq \phi$ **do**
  remove $\mathcal{Q}'_i$ from S
  solve $\mathcal{Q}'_i$
  if $\mathcal{Q}'_i$ *is feasible* **then**
    let β be optimal basic solution of $\mathcal{Q}'_i$
    if β *satisfies integrality constraints* **then**
      if $cost(\beta) < Z$ **then**
        store β and update Z
    else
      if $cost(\mathcal{Q}'_i) < Z$ **then**
        let x_j be part of binary variables satisfied
        $S := S \cup \{\mathcal{Q}'_0 \ \wedge x_j = 0, \mathcal{Q}'_0 \ \wedge x_j = 1\}$
Return Z, β

Algorithm 2: Branch and bound

Input : The Optimal question $\mathcal{Q}$
Output: Optimal value , answer

Generate relaxation Linear programming problem $\mathcal{Q}'_0$
$Z := +\infty$ %set the best cost so far
$S := \{\mathcal{Q}'_0\}$ % generate the first branch
while $S \neq \phi$ **do**
  remove $\mathcal{Q}'_i$ from S
  solve $\mathcal{Q}'_i$
  if $\mathcal{Q}'_i$ *is feasible* **then**
    let β be optimal basic solution of $\mathcal{Q}'_i$
    if β *satisfies integrality constraints* **then**
      if $cost(\beta) < Z$ **then**
        store β and update Z
    else
      if $cost(\mathcal{Q}'_i) < Z$ **then**
        let x_j be part of binary variables satisfied
        $S := S \cup \{\mathcal{Q}'_0 \ \wedge x_j = 0, \mathcal{Q}'_0 \ \wedge x_j = 1\}$
Return Z, β

2) $i + j + l$
3) $i + j$
4) $j + k$

For situation 1, we ignore the sub-task constrains. That means the answer may exist that we ask the agents without required functionality to execute one task. So the optimal value can be less than the real value as the lower bound:

$$\min_{r_{i,j,k},t_j} \quad \max(t_j + p_j)$$

s.t.

Temporal constraint of tasks:

$$t_{j_1} + p_{j_1} \leqslant t_{j_2} \quad \forall (j_1, j_2) \in C_{partial} \tag{9}$$

provide enough serves for the task j

$$\sum_{i=1}^{m}\sum_{k=1}^{n} r_{i,j,k} b_j = a_j \quad \forall j \tag{10}$$

one agent can only provide the serve it has:

$$r_{i,j,k} \leqslant b_i \quad \forall i, j, k \tag{11}$$

One agent can execute one task no more than once:

$$\sum_{k=1}^{n} r_{i,j,k} \leqslant 1 \forall i, j \tag{12}$$

One agent at any time can execute no more than one task:

$$\sum_{j=1}^{m} r_{i,j,k} \leqslant 1 \forall i, k \tag{13}$$

One agent can execute $k+1th$ task only if it execute kth task.

$$\sum_{j=1}^{m} r_{i,j,k} - \sum_{j=1}^{m} r_{i,j,k+1} \leqslant 0 \qquad \forall i, k < m-1 \tag{14}$$

Even agent need to obey the motion constrain.

$$t_{i_2} - t_{i_1} - M r_{i,j_1,k} - M r_{i,j_2,k+1} \geqslant dis_{j_1,j_2}/v + p_{i_1} - 2M \forall i, j \tag{15}$$

$$t_i - M r_{i,j,1} \geqslant dis_{i,j}/v - M \qquad \forall i, j \tag{16}$$

For situation 2:

$$\min_{r_{i,j,l}} \quad \max(\sum_{j=1}^{m}\sum_{l=1}^{o} r_{i,j,l} p'_j)$$

s.t.
provide enough serves for the task j

$$\sum_{i=1}^{m} r_{i,j,l} b_{i,l} = a_{j,l} \quad \forall j, l \tag{17}$$

one agent can only provide the serve it has:

$$r_{i,j,l} \leqslant b_{i,l} \quad \forall i, j, l \tag{18}$$

One agent can execute one task no more than once:

$$\sum_{l=1}^{o} r_{i,j,l} \leqslant 1 \forall i, j \tag{19}$$

The motion constrains are embodied in execution time that we add the minimum motion time to the task execution time.

$$p'_j = p_j + \min\{dis_{j,j'}/v_m ax, dis_{i,j}/v_m ax\} \tag{20}$$

For situation 3:

$$\min_{r_{i,j}} \quad \max(\sum_{j}^{m} r_{i,j} p_j)$$

Serves constraints:

$$\sum_{i=1}^{n} r_{i,j} = a_j \tag{21}$$

We add the execution time to Simplified motion constraints, the estimate execute time is

$$p_j = \min_{a_{j,l}} \{dis_{j',j}, dis_{i_l,j}\} + p_j \tag{22}$$

For situation 4:

$$\min_{r_{j,k},t_j} \quad \max(t_j + p_j)$$

s.t.

Temporal constraint of tasks:

$$t_{j_1} + p_{j_1} \leqslant t_{j_2} \quad \forall (j_1, j_2) \in C_{partial} \tag{23}$$

We add the execution time to Simplified motion constraints, the estimate execute time is

$$p_j = \min_{a_{j,l}} \{dis_{j',j}, dis_{i_l,j}\} + p_j \tag{24}$$

In my half-baked assumption, I think these low bound method can be used with combination. We can use it as the in mechine learning do that use more accurate algorithm first to cut down the branch faster and use the less accurate algorithm later to explore branches faster.(Even the opposite is true but it should be faster than only use one method)

IV. LOCAL SEARCH METHOD

Local search is the best way to deal with 01 programming. Local search is a sequential search method. The search path forms a track, aiming at the current point (solution), and tries to find a better solution from its neighborhood to replace the current solution. If no solution is found, the search should be stopped. There are two type methods of local search method. The first searching space is composed of feasible solutions and we begin from one node and choose its neighborhoods (other solutions) by checking the value of solutions. Another method focus on a contract sub-question called VNDS (variable neighborhood decomposition search). When it gets an initial feasible solution, it defined a sub-question by remaining k variables invariant, and $n-k$ variable are alterable. Let local search method search $n-k$ free variables. Any local search can get a local optimum, but if an algorithm doesn't have the ability to jump out the local optimum, we will call it Hill Climbing method. (I guess the local optimum in this question is also global). Usually, the distribution of feasible solutions in search space is sparse, so local search method can found a better solution in shorter time than other method.

As algorithm 2 showed, the core of a Hill Climbing local search method is the definition of neighbors. I put forward three kinds of neighbors definitions here and plan to try it in code later.

1) Given a feasible solution, we defined a small neighbor area and a lager neighbor area. The small neighbor area relation is we can exchange any two task in one agent if feasible. The large neighbor area is fetching one task from an agent and put it to another(the small neighbor area will not lead to exchange in the large neighbor area). This method can find the optimal solution of one combination and change to another combination.
2) Given a feasible solution, fetch one action and put it in any other feasible place.
3) Given a feasible solution, defined a sub-question: k variables in this question are invariable, other $n-k$ variables are alterable. And we use local search these $n-k$ variables.

Algorithm 3: Basic local search

Input : The Optimal question $\mathcal{Q}$,round m
Output: Optimal value , answer

```
Generate initial Solution s_0
k=0
while k < m do
    s' = FindBestNeighbor(s) if Q(s') < Q(s)) then
        s = s'
        k = k + 1
Return Q(s), s
```

V. PARALLEL LOCAL BOUND METHOD

During the search progress, the method usually works serially which speed are only limited by the magnitude of CPU frequency, we can use more core to computing the answers which may increase computing speed by several times. However, due to the difficulty of code, I plan to realize it at last.