Crowdsourcing is a new computing paradigm where humans are actively enrolled to participate in the procedure of computing, especially for tasks that are easier for humans than for computers.
The popularity of mobile computing and sharing economy has extended conventional webbased crowdsourcing to spatial crowdsourcing (SC), where spatial data such as location, mobility and the associated contextual information, plays a central role.

Famous SC platforms include taxi travel service platform (e.g. DiDi uber), food carryout service platform like meituanwaimai event-participant service (e.g. Meetup4), and so on.

A core issue of SC platforms is to assign tasks to suitable crowd workers
Existing studies usually study the assignment problem between workers and tasks.
They model this problem as a maximum unweighted/weighted bipartite graph matching (shorten as 2D matching) problem

Profit paper Aim to maximize the profit of the platform
establish a task reward pricing model with tasks' temporal constraints (i.e., expected completion time and deadline)
In this model, the workers travel to the users' locations to provide services. In other words, the service place is the same with the location of users.

However, some new O2O applications provide such services that require users and workers go to a third place together.

Station ride-sharing. The DiDi2 platform provides a station ride-sharing service, in which several users go to a station (i.e., workplace) on foot, and a car/taxi (i.e., worker) drive to that station to pick these users.

Personalized haircut service. On the Nanguache1 platform, each user would like to require a hairdresser (i.e., worker) and book a barber shop (i.e., workplace) to have a haircut. The user (e.g. u2) and hairdresser (e.g. w2) can depart from their living places, and go to a barber shop (e.g. p1) together to make the haircut.

There are some studies focus on the 3d matching problem Song et al in [25] propose a 3D matching strategy, in which the platform makes plan by maximizing
the total utility score of users and workers. This strategy just
maximized the satisfaction of the platform, but failed to comprehensively
consider the request of both users and workers.

So this paper propose a stable matching strategy which consider the request of both users and workers. Before introduce the problem statement we can see a simple example to feel

the difference between this two paper.

讲 figure1

we treat the reciprocal value of distance as the utility. Using the strategy in [25], we can obtain the global plan shown as the red lines. In this plan, u3 and w1 are matched together, and assigned to place p2. However, there obviously exists another place p3 which is nearer to both u3 and w1. Then neither u3 nor w1 will be happy: "Clearly, there is a workplace closer to us. Why the platform let us travel further?!"

We propose a new matching over the SC platforms for such 3D matching services, which considers the distance between workplaces to both users and workers comprehensively.
We name this problem as 3-Dimensional Stable Spatial Matching (3D-SSM). Under our 3D-SSM, the global plan in Fig. 1 is shown as the blue lines,

table1
W　the worker set
DW　The ordered distance set of workers to workpla
To convenient,　these sets are stored as a decreased order of the distance
Capacity is an attribute of worker,means the number of the users that the worker can matched

Algorithm-1
From the proof of Theorem 1, we can find that the special case of 3DSSM can be converted to maximum independent set problem [27].
It stimulates us that we can solve 3D-SSM by constructing a graph and conducting theMaximumWeight Independent Set (MWIS) [14]

The main idea is as follows. Firstly, we construct vertices using stable triples in 3D-SSM. When the capacity of each worker $c_w = 1$, a graph GM can be equivalently constructed for 3D-SSM. Each stable pair is a vertex of GM. When $c_{w_i} > 1$, the vertex will become a super vertex containing multiple stable triples. In the vertexes, worker and workplace are the same, but users are different. The number of stable triples in a vertex becomes its weight. Secondly, we link edges between vertices who contain the same workers, users, or workplaces. Finally, we conduct the algorithms solving the Maximum Weight Independent Set (MWIS) problem to calculate the result [14] and [15].

The pseudocode of the baseline algorithm is shown in Algorithm 1. We initialize an empty graph GM as a pair of empty vertex set VM and edge set EM (Line 1). Then, for each worker $w \in W$ and each workplace $p \in P$, we initialize an empty set S. We enumerate all possible stable triples and put them into S (Lines 2-7). We state a counter i from 1 to $\min(c_w, |S|)$, and $\binom{|S|}{i}$ vertices with weight i would be constructed (Line 9). After constructing all vertices (Lines 2-9), we link the edges between vertices whose w or u or p are the same (Lines 10-12). Finally, we conduct algorithms for MWIS problem to solve the problem (Lines 13-14).

3.2 GSM Algorithm

As discussed above, the MWIS-based algorithm can provide exact answers, but its computational complexity is too high. Thus, we consider another approximate algorithm called Greedy Stable Matching (GSM), which is more efficient. We propose an approximate algorithm, Greedy Stable Matching (GSM) algorithm. It directly finds the stable matching for users one by one according to the distance.

The main idea of GSM is to optimize the workplace and worker for a given user. Given a user u, we greedily check the nearest workplace p and the nearest worker w to p, until (w, u, p) is stable. Recursively repeat this method until no stable matches can be found.

The details of GSM are illustrated in Algorithm 2. We initialize an unmatched worker (resp. workplace) set AW (resp. AP) (Line 1). For each $u \in U$, it enumerates workplaces ordered by the distances from the smallest to largest. If $p \in AP$, it first enumerates workers in AW to find a stable triple, then deletes the worker (resp. workplace) from AW (resp. AP) and puts the triple into matching results (Lines 5-9). If $p \notin AP$, there is a worker w matched with p (Line 11). If (w, u, p) is stable and the capacity of w is satisfied, the triple is a matching result of u (Lines 10-14). If a stable triple is found for the user, the algorithm starts to process the next user (Lines 15-16).

Shortcoming
This is because in GSM,
the matching result cannot be changed once a worker is matched
with a workplace, even though he may match more users in other

workplaces. The capacity of workers may be wasted and leads to a bad effectiveness.

The details of DSM are illustrated in Algorithm 3. We initialize an unmatched sets AW, AU and AP, and initialize a termination flag (Line 1). The algorithm executes until $Flag = 0$, which means it cannot find any other new stable triples (Lines 2-18). For each user $u \in AU$ (Line 4), we consider three different cases. In the first case, if we find a stable triple (w, u, p) where $w \in AW$ and $p \in AP$, we update the matching results, and set $Flag$ equals to 1 (Lines 5-7). Next, we find whether other users can be matched with w and p (Lines 8-10). If u is not matched in the first case (Line 11), the algorithm tries to find a stable triple where $w \in AW$ and $p \notin AP$. If w can match more users with p than the current worker matched with p, the current triples with p are deleted from M and new stable triples are put into M (Lines 12-14). If u is still not matched (Line 15), the last case is to find a stable triple where $w \notin AW$ and $p \in AP$. The matching result M will be updated if w can match more users with p than current workplace where he matches (Lines 16-18).

Example 3. Back to the running example in Table 2. At the beginning, we initialize AW, AU and AP. For $u_1 \in AU$, we find p_4 and w_4, where p_4 is the closest workplace to u_1 and w_4 is closest worker to p_4, and (w_4, u_1, p_4) is a stable triple and put into M. The capacity of w_4 is 3, so u_2 and u_3 can be matched. The matching results are shown in Fig. 2(a). Fig. 2(b)-2(d) show the matching results of u_4, u_7, u_8. For $u_{11} \in AU$, the only elements in AW and AP are w_3 and p_1, but (w_3, u_{11}, p_1) is not stable. For $p_5 \notin AP$, (w_3, u_{11}, p_5) is stable and (w_3, u_8, p_5) is also stable. One more user will be matched if w_3 is matched with p_5. Therefore, (w_5, u_8, p_5) is deleted, (w_3, u_{11}, p_5) and (w_3, u_8, p_5) are put into M. The result is shown in Fig. 2(e). Finally, (w_5, u_{12}, p_1) is matched in Fig. 2(f), and DSM terminates as AU is empty.

we propose Path Swap algorithm. We build a data structure storing the stable triples that are either already in the result set M or has possibility to be put in M. We call this data structure "Swap Path". The calculation of 3D-SSM can be accelerated by constructing and updating the Swap Paths.

Example 4. Take Fig. 3(a) as an example, there are two fixed units (w1,p2) and (w4,p4). {u1,u2,u3} are users matched with (w4,p4). The strong candidates of both (w1,p2) and (w4,p4) are {u7,u8,u9,u10, u11,u12}, these users are not matched in any triples. They have no weak candidates.

After introducing the swap path, we are ready to describe how to update the swap paths. For each triple (w_i, u_i, p_i) in $\mathcal{P}$, where $i \in [1, |\mathcal{P}|)]$, u_i is matched with (w_{i+1}, p_{i+1}) in M, we break the triple (w_{i+1}, u_i, p_{i+1}) and put the new triple $(w_i, u_i.p_i)$ into M. If $u_{|\mathcal{P}|}$ is matched with a unit (w', p') and there are no users matched with it after $(w', u_{|\mathcal{P}|}, p')$ is broken, we delete the unit from M. If $u_{|\mathcal{P}|}$ is not matched with any units, we just put $(w_{|\mathcal{P}|}, u_{|\mathcal{P}|}, p_{|\mathcal{P}|})$ into M.

Example 5. Take Fig. 3(b) as an example, we can find three paths for (w_2, p_1), $Path1 = \{(w_2, u_{10}, p_1)\}$, $Path2 = \{(w_2, u_{11}, p_1)\}$ *and* $Path3 = \{(w_2, u_5, p_1), (w_4, u_7, p_4)\}$. *And then, we start to update the matching result based on the paths. For* $Path1$, u_{10} *is not matched and we can put* (w_2, u_{10}, p_1) *into* M. $Path2$ *is updated in the same way.* $Path3$ *is different from the formers. We firstly break* (w_4, u_5, p_4) *and put* (w_2, u_5, p_1) *into* M, *and then match* u_7 *with* (w_4, p_4), *where* u_7 *is a strong candidate of* (w_4, p_4).

We conduct our algorithms over two real-life datasets, Meetup [20] and DiDi Open Dataset5. Meetup is an event-based social network platform.We extract data over different cities and set the capacity of each worker from 1 to 3. DiDi is a popular taxi travel service platform in recent years. We draw some areas of different sizes according to latitude and longitude, and set the upper limit of capacity as 4, according to the actual passenger load situation of the taxies.
we conduct experiments using the two baseline algorithms (MWIS and GSM), DSM algorithm and Path Swap algorithm on real datasets, and test the effectiveness and efficiency of the algorithms.

Figure a and b shows the unstable percent in TOM algorithm ,which focuses on minimizing minimize total distance. Many matches are unstable in the real datasets, even more than 30% in Singapore and Region D

The matching results of our algorithms are shown in Fig. 4(c) and Fig. 4(d). The MWIS

algorithm can find an optimal solution that all the users can be matched on dataset of Beijing and Region A, but cannot output solutions within an acceptable period of time in other datasets. The matching result of GSM is the worst. The matching results of Path Swap is better than DSM in Meetup dataset, and both the algorithms reach the optimal solutions in Beijing. In DiDi dataset, DSM can match more triples than Path Swap.

Efficiency on Real Datasets. The running time of MWIS is unacceptable except in Beijing and Region A. GSM runs much faster than the other algorithm, however, the matching result is too bad as mentioned above. Path Swap runs faster than DSM in Auckland, Singapore and Region D, shown in Fig. 4(e) and Fig. 4(f). What is more, MWIS costs much more time and memory than DSM and Path Swap on Beijing dataset. The memory of GSM and DSM is similar. GSM, DSM and Path Swap cost more time and memory as the size of datasets increases on both the datasets. Path Swap costs more memory than two greedy algorithms, because it needs to index the candidates for each matched workplace, shown in Fig. 4(g) and Fig. 4(h).

In this section, we test the scalability of the two approximate algorithms. We conduct the algorithms on a synthetic dataset and test the running time, matching results and memory cost.

|W |

1.Fig. 5(a) shows the running time of the algorithms with different size of |W |.

As |W | increases, the running time of Path Swap gets longer. as |W | increase that it costs more time to enumerating stable triples.

2. When |W | = 2500, there are many fixed units after the initialization step, more users are matched so that the algorithms can terminate faster. Therefore, the time starts to decrease.

3.The running time of DSM shows a different trend. DSM costs more time than Path Swap when |W | is small, and the time reduces when |W | = 2000. The reason is that when |W | is small, workers are soon matched, but there are still many unmatched users and workplaces, DSM spends much time in the third case in order to matching more users. When the workers are enough to match all the users, DSM terminates quickly because most of users are matched in the first case. When |W | is large enough, the running time increase again due to the time of enumerating workers increases.

4.The matching result of DSM and Path Swap is similar, as shown in Fig. 5(e). When workers are not enough to match all the users, the size of matching set |M| is close to the theoretical maximum number of matches, which is min{|W |, |P |}×c. Fig. 5(i) shows the memory cost of the algorithms,

5.both the two algorithms cost more memory as |W | increase, and Path Swap costs more memory than DSM because it needs to index the candidate information.

|P |
The result is similar to the experiment of scalability w.r.t |W |.
A little difference In Fig. 5(f), both of the algorithms still show a good performance and the Path Swap is better than DSM when |P | = 1500.

|U |
1.As |U | increases, the running time of Path Swap increases and then decreases when |U | is greater than 8000, while the running time of DSM increases, as shown in Fig. 5(c). The reason is that the time of enumerating stable triples of both the algorithmsand the cost of find and update paths of Path Swap increases as |U | increases. But when |U | is greater than 8000, there are many unmatched users while finding a path, it will save much time to find stable triples in these users than finding a long path and update.
2.The value of |M|/|U | decrease due to |U | gradually surpasses the theoretical maximum number of matches. The memory cost of Path Swap gets larger due to the size of candidate set increases as there are more users, as shown in Fig. 5(k).

C
1.As the capacity increases, workers can match more users, and the size of unmatched users decreases more quickly. When the capacity is greater than 4, the theoretical maximum number of matches is greater than |U |, therefore the time increases gently.
2.As capacity increase, the path update costs more times because the update operator will be called more times in each matched unit.
3.The matching result gets better as the theoretical maximum number of matches gets larger.
4.The memory cost of Path Swap decreases linearly, as the size of candidate set decreases.

The MWIS algorithm can output optimal solutions within an acceptable time on small datasets. As the size of dataset increases, the running time is unbearable compared with the other three algorithms.
GSM shows a good efficiency but the matching result is not well.
DSM and Path Swap show a good performance of the number of users that are matched, and Path Swap is better than DSM on real datasets. The running time of the algorithms are different in different data distribution.
According to the experiment results, DSM is more suitable in the scenario that |U | is close to the theoretical maximum number of matches and |W | is close to |P |.
In other scenarios, where |U | is larger than the theoretical maximum number of matches and |W | differs greatly to |P |, Path Swap is more efficient.