2019 IEEE 35th International Conference on Data Engineering (ICDE)

Procrastination-aware Scheduling: A Bipartite Graph Perspective

Libin Wang†, Yongxin Tong†, Chunming Hu†, Lei Chen‡, Yiming Li†
†BDBC, SKLSDE Lab and IRI, Beihang University, Beijing, China
‡The Hong Kong University of Science and Technology, Hong Kong SAR, China
†{lbwang, yxtong, hucm, ymli}@buaa.edu.cn, ‡leichen@cse.ust.hk

***Abstract*—Procrastination is a prevalent form of self-control failure. As it often concerns with the individual's ability to meet the deadline, an efficient time management is crucial for overcoming it. Though a considerable amount of work in behavioral economics provides useful insights, there is not a computational way to guide us how to obtain an appropriate schedule for all the things to be done, especially when the relationship of the deadlines is intrinsic. In this paper, we first propose the Procrastination-aware Scheduling Problem (PSP) to model an appropriate schedule. A bipartite graph formulation is then developed to further illustrate the concepts. We find the PSP is NP-hard in the strong sense and design an approximation algorithm. In addition, we note the significance of the PSP under the online scenario (called OnlinePSP). Finally, we verify the effectiveness and efficiency of the proposed algorithms through extensive experiments on real datasets.**

I. Introduction

Procrastination, the tendency to put off things and dally with some distractions, is all familiar to most people. It is extremely prevalent and pernicious. For example, students may delay their assignments until the deadlines and get poor grades. Employees who postpone their unpleasant tasks affect the progress of a project. Some patients even suffer from unexpected diseases because of not sticking to the medical check-ups.

These bad consequences are always related to inefficient time management. It is observed that chronic procrastinators spend less time on projects which are more likely to succeed [1], start the tasks intended to be finished earlier at the last day before the deadline [2], or just have difficulties in scheduling too many things [3], [4].

An appropriate schedule will help the individual improve the utility of finished jobs (or tasks) [5]. This is also demonstrated by the well-known MIT experiment [6]: there are three papers to be written over a 12-week-long course. Two groups of students are tested with different schedules, where the first one is required to submit the paper by the end of the last week, while the second one has evenly spaced deadlines, *i.e.*, the fourth, eighth, and twelfth weeks. As we expect, the final results indicate that most students in the first group start to write all three papers in the last four weeks, and they get worse grades than those in the second group.

However, it is hard to obtain such a schedule. For one thing, the individual often encounters many jobs with different release times and deadlines, such as the assignments required by many courses. The intrinsic relationship among the available time periods of jobs makes it hard to say some heuristic ideas (like evenly spaced deadlines in the MIT experiment) can be used to generate the schedule. For another, since we assign jobs to a human, the utility of scheduled jobs will definitely be affected by a heavy workload. As shown in the experiment about proofreading papers [6], the number of errors detected by people who do the jobs in a rush are larger than the average. Therefore, naive approaches such as scheduling as many jobs as we can may not always improve the overall utility.

In real scenario, it may be the case that we do not know the details of all the jobs beforehand; we may be only aware of some of them in any period and they appear in an online fashion, which further hinders producing an appropriate schedule. Recent studies consider different constraints [7], [8], [9], [10], [11], [12], [13]. It is worth noting that in their assignment process, the utility values of new scheduled jobs are not affected by the current assignment, in contrast to our dynamic values.

II. Problem Statement

In this section, we define the *Procrastination-aware Scheduling Problem* (PSP), provide the bipartite graph formulation, and discuss its hardness.

A. Problem Definitions

There are T time periods in total. In the beginning, we are notified of a set of jobs J.

Definition 1 (Job): Each job $j \in J$ has its release time $r_j \in \{1, \ldots, T\}$ and deadline $d_j \in \{r_j, \ldots, T\}$. It requires some individual to take cost c_j to finish it and we can gain a share of utility u_j if it is not done in a rush.

We assume large jobs have been decomposed into small ones such that each job j can be done in any one period $t \in \{r_j, \ldots, d_j\}$ and incurs the one-time cost c_j. As an example, we can imagine the cost takes several hours, and the time period may last three days or a week or longer. For the utility, it can represent the credits of an assignment in a course, or the payments of a task if it is well done by the individual.

There exists one individual who suffers from certain degree of procrastination, and she may delay these jobs until the deadlines. We have to schedule these jobs to overcome her laziness and maximize the utility.

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00164

Definition 2 (Schedule): For each time period t, we force the individual to finish our schedule $S^t \subseteq J$ but assume an upper limit C^t specifying the maximum total cost of jobs we can assign to her (*i.e.*, $\sum_{j \in S^t} c_j \leq C^t$). To carry out the enforcement, we may adopt some coercive measures like not presenting the real deadline.

Definition 3 (Utility function): Since the total utility may be affected as we schedule more jobs in a single period, we assume a utility function $U(S^t)$ on any schedule $S^t \subseteq J$ which is of a wide class called submodular function; for any $S, S' \subseteq J$ such that $S \subseteq S'$ and any $j \in J \backslash S'$, it satisfies $U(S \cup \{j\}) - U(S) \geq U(S' \cup \{j\}) - U(S')$, indicating that with more jobs crowded in a single period, the *marginal* utility should be smaller, also known as the diminishing returns. Assume also it is a monotonically non-decreasing function; for any $S \subseteq S'$, $U(S) \leq U(S')$.

Note that we support any monotone submodular functions, including, but not limited to, the linear sum or the budget-additive functions. The specifications of the functions depend on the individual. The discussion is deferred and for now we just assume there exists a function oracle which outputs the value if we input a set of jobs.

Now we formally propose the *Procrastination-aware Scheduling Problem* (PSP).

Definition 4 (PSP): Given a set of jobs J, we need to make an assignment for jobs $A = (S^1, \ldots, S^T)$ (*i.e.*, the schedule for each t) so as to maximize the total utility value $U(A) = \sum_t U(S^t)$, while satisfying the following constraints.

- Each job j can only be done once; if $j \in S^t$ for some t, then $j \notin S^{t'}$ for any $t' \neq t$.
- Each job j can only be done when it is available; if $j \in S^t$ for some t, then $r_j \leq t \leq d_j$.
- For schedule S^t, the costs of assigned jobs should not exceed the upper limit; $\sum_{j \in S^t} c_j \leq C^t$.
- The utility functions $U(S^t)$ are monotone submodular functions on any set $S^t \subseteq J$.

Graph formulation. It is helpful if we imagine all these concepts on a bipartite graph $G = (J, \{1, \ldots, T\}, E)$, where jobs and time periods are represented by nodes. For each edge $(j, t) \in E$, it indicates that job j can be done in period t. Therefore, for each job j, there are edges $(j, r_j), \ldots, (j, d_j) \in E$. Let $\delta(j)$ and $\delta(t)$ denote the set of edges incident to j and t, respectively. Let $N(j)$ and $N(t)$ denote their corresponding neighbors. Each job j is also associated with its cost c_j and utility u_j.

A feasible bipartite matching M is a subset of edges such that no pair of edges in M share a node $j \in J$, and for any node t, $\sum_{(j,t) \in \delta(t) \cap M} c_j \leq C^t$. We can rewrite $S^t = \{j \in J : (j, t) \in M\}$, and the problem is to find a feasible matching which maximizes the total utility $\sum_t U(S^t)$.

Note that our bipartite graph is dynamic; that is, the utility of adding a new job into some schedule S^t depends on the current matching. However, in traditional bipartite matching, the utility values of all the edges are fixed and the objective function can be represented by the sum of utility values of edges.

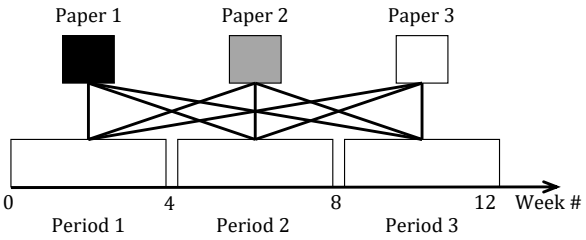


Fig. 1: Bipartite graph of the MIT experiment.

Hardness. Since the submodular functions include the linear ones (*e.g.*, $U(S) = \sum_{j \in S} u_j$), our problem generalizes the Multiple Knapsack problem with Assignment Restrictions (MKAR) [14], which uses the linear utility function. Besides, it requires that for any j, each u_j equals c_j. As it is proved NP-hard in the strong sense, we have the following theorem.

Theorem 1: PSP is strongly NP-hard.

III. Solutions for PSP

In this section, we first present a heuristic idea with poor performance, then propose our OffPSP algorithm.

Abusing notations slightly, in the following we may use $c(j) = c_j$, $c(S) = \sum_{j \in S} c_j$ for $S \subseteq J$, and $c(A) = \sum_t c(S^t)$ for any assignment.

A. Heuristic

Maybe the most natural idea is the greedy algorithm: in each round, we schedule the "best" available job to some period t. A heuristic for the best one is to choose the job-period pair that gives us the most bang for the buck; that is, the pair (j, t) that maximizes the ratio of the increase for the objective function U to the cost c_j: $\arg\max_{(j,t) \in E'} \frac{U(S^t \cup \{j\}) - U(S^t)}{c_j}$. If the current schedule is S^t, the increase would be $U(S^t \cup \{j\}) - U(S^t)$. And all the constraints are satisfied if we maintain a set of feasible edges, denoted by E'. Ties are broken by choosing the pair (j, t) with the largest $C^t - c(S^t)$, since this will gives us a larger room for the rest of the jobs. The scheduling process stops when there is no edges in E' which can improve the utility.

However, we claim that in certain circumstances the performance of this heuristic can be arbitrarily bad. The idea is based on [15]. Imagine two jobs can be assigned to just one period.The first one has the cost 40 and utility 40, while the second one has the cost ϵ and utility 2ϵ. If ϵ is infinitesimal, we gain a small utility value 2ϵ since only the second job can be scheduled. However, the optimal solution is 40, and the ratio can be arbitrarily small.

B. OffPSP Algorithm

The heuristic may mislead us to a locally optimal solution without any performance guarantee. This motivates us to propose the OffPSP algorithm. As the example suggests, a job-period pair (j, t) which maximizes the ratio can be the one that gives very little increase to the utility function if the

Algorithm 1: OffPSP

input : Bipartite graph $G = (J, \{1, \ldots, T\}, E)$, Utility functions U
output: Assignment $A = (S^1, \ldots, S^T)$

1 $E' \leftarrow E$;
2 $i \leftarrow 1$;
3 **foreach** $t \in \{1, \ldots, T\}$ **do**
4 $S_0^t \leftarrow \emptyset$;
5 $A_0 \leftarrow (S_0^1, \ldots, S_0^T)$;
6 **while** $E' \neq \emptyset$ **do**
7 $(j_i, t) \leftarrow \arg\max_{(j,t) \in E'} \frac{U(S_{i-1}^t \cup \{j\}) - U(S_{i-1}^t)}{c_j}$;
8 $S_i^t \leftarrow S_{i-1}^t \cup \{j_i\}$;
9 $E' \leftarrow E' \backslash \delta(j_i)$;
10 **if** $c(S_i^t) > C^t$ **then**
11 $E' \leftarrow E' \backslash \delta(t)$;
12 $i \leftarrow i + 1$;
13 **foreach** $t \in \{1, \ldots, T\}$ **do**
14 **if** $c(S_i^t) > C^t$ **then**
15 Let j be the last job scheduled into period t;
16 **if** $U(\{j\}) > U(S_t^i \backslash \{j\})$ **then**
17 $S_t^i \leftarrow \{j\}$;
18 **else**
19 $S_t^i \leftarrow S_t^i \backslash \{j\}$;
20 **return** $A \leftarrow (S_i^1, \ldots, S_i^T)$;

cost c_j is small. An intuition is to ensure that we schedule jobs with enough costs.

The basic idea is as follows. We still choose the pair which has the most bang for the buck. However, this time we will first make a pseudo-assignment which may be infeasible for the upper limit; we allow adding job j to S^t if it is the first time we exceed the upper limit, *i.e.*, $c(S^t \cup \{j\}) > C^t$. Note that the pseudo-assignment guarantees that the sum of costs in the pseudo-schedule is greater than C^t and hence a larger increase for the utility values. In the final step, we make the pseudo-assignment feasible while not losing much utility. We check each infeasible schedule: we compare the last job with the rest and retain the one with larger utility.

Algorithm 1 illustrates the main procedure. In lines 1-5, we initialize the schedule and the set of feasible edges E'. Note that the subscript i is used to denote the ith iteration, which is convenient for the analysis. In lines 6-12, we iteratively find the pair with the largest ratio among all feasible edges. The difference is that we make a pseudo-assignment where we do not consider period t any more only the first time when $c(S^t)$ exceed the upper limit C^t (in lines 10-11). In lines 13-19, we adjust each infeasible schedule to satisfy the constraints. If the utility of the last job is greater than that of the rest of jobs, we retain the last job (in lines 16-17), otherwise we discard it (in lines 18-19).

Implementation. In practice, we first find for each t its best job among all its available neighbors, then compare those best jobs to get the best pair (j, t). For the second step, we can always maintain a max-heap to speed up our comparison. Note that there is no need to use such heap for the first step, because we have to update all the increases for all t's neighbors once a job is added to S^t. After a job-period pair (j_i, t) has been scheduled, we mark the job j_i and ignore all the pairs incident to j_i in the heap.

Complexity analysis. In practice, we first find for each t its best job among all its available neighbors, then compare those best jobs to get the best pair (j, t). For the second step, we can always maintain a max-heap to speed up our comparison. For the initialization of the max-heap, we need to compute the increase for each edge in $\mathcal{O}(|E|)$ and add those best jobs to the max-heap in $\mathcal{O}(T \log T)$. The maximum size of the heap can be $|E|$ since we never remove the elements. Each time we choose from the max-heap the best pair in $\mathcal{O}(\log |E|)$ and there are at most $|E|$ iterations, the loop in lines 6-12 needs $\mathcal{O}(|E| \log |E|)$ time. Checking the pseudo-assignment in lines 13-19 takes $\mathcal{O}(T)$. Hence, the total time cost is $\mathcal{O}(|E| \log |E|)$.

TABLE I: Real datasets

Factor	Setting
$\#courses$	2, 4, **6**, 8, 10
C^t	2, 4, **6**, 8, 10

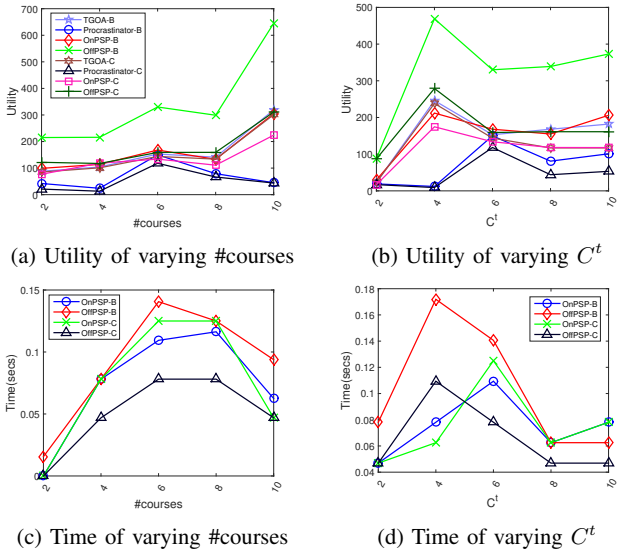
IV. Experimental Study

A. Real datasets.

We use the data published in KDD CUP 2015 from XuetangX, one of the largest MOOC platforms in China. It contains the records of courses from October 30 2013 to August 1 2014 and the students who enrolled in them. As our algorithms are designed for one individual, we extract for each student its own enrollment. We observe that the maximum number of courses someone enrolls in reaches 27, verifying the necessity of an elaborate schedule. In our settings, one time period lasts one day, so their numbers of periods may be different. Each course consists of many modules, *e.g.*, video, problem, discussion. We regard them as jobs and associate them with corresponding costs and utility values. We vary the number of courses $\#courses$ that somebody enrolls in and the upper limit C^t, the settings of which are showed in Table I. We randomly select the student who satisfies the setting and extract the corresponding dataset. As the other parameters vary with the datasets, we do not show them due to limited space. We also test the algorithms on the datasets of the MOOC platform of Beihang University, which opens to the undergraduates. As the results show a similar pattern to those on the XuetangX datasets, we omit them here.

B. Compared algorithms.

Apart from our proposed approaches, we compare the state-of-art algorithm for task assignment as a baseline [7], called TGOA. We also test the performance of the "Procrastinator".



(a) Utility of varying #courses (b) Utility of varying C^t

(c) Time of varying #courses (d) Time of varying C^t

Fig. 2: Results on real data

For the synthetic data, we simulate a procrastinator who does the job j with higher probability as t is close to the deadline d_j. For the real case, we directly examine the log of the student, since it includes all the records of jobs (modules) done by her in each day.

For the submodular utility functions, we consider two types. The first one is budget-additive: $U_S(j) = u_j$ if $\sum_{j \in S} u_j < B$ and 0 otherwise. where the utility of the jobs are all zeros if the total sum has reached the budget B. The second one is more continuous, indicating that the new job will be affected more as the number of scheduled jobs becomes greater: $U_S(j) = u_j - \alpha|S|$, where α represents the degree of how the number of currently scheduled jobs S affects utility. The algorithm which uses these two functions are suffixed with "-B" and "-C". We also test a variant of the algorithm called OnPSP under online scenario, where tasks are revealed in an online fashion. As TGOA runs 20x slower and basically consumes two times memory than the other algorithms, we omit its time costs.

C. Experiment Results

As stated in the setup of real datasets, we vary the number of courses which some student enrolls in and the upper limit of each time period, the results of which are showed in the last two column of Fig. 2. For the number of courses, it is worth noting that a large number does not indicate more jobs, because some courses have many jobs (or modules), whereas others have few, and the students may enroll in different courses. For both two types of functions, OffPSP achieves the largest utility values, and OnPSP performs better than the students' own schedules, which are affected by certain degree of procrastination. Though TGOA achieves moderate values, it incurs large time and space costs and hence inapplicable. For the running time, since the upper limit is small, OnPSP with its complexity $\mathcal{O}(|J^t|C^t)$ may run faster than OffPSP. Similar results can be observed in varying the upper limit C^t,

ACKNOWLEDGEMENT

Libin Wang and Yongxin Tong's works are partially supported by National Science Foundation of China (NSFC) under Grant No. 61822201 and U1811463, the Fundamental Research Funds for the Central Universities under Grant No. YWF-18-XGB-004, Beijing Moral Cultivation Research Project under Grant No. BJSZ2019ZD08, and Didi Gaia Collaborative Research Funds for Young Scholars. Lei Chen's work is partially supported by the Hong Kong RGC GRF Project 16207617, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC ITF grants ITS/391/15FX and ITS/212/16FP, Didi-HKUST joint research lab project, Microsoft Research Asia Collaborative Research Grant and Wechat Research Grant. Chunming Hu is the corresponding author in this paper.

REFERENCES

[1] C. H. Lay, "Working to schedule on personal projects: An assessment of person-project characteristics and trait procrastination," *Journal of Social Behavior and Personality*, vol. 5, no. 3, pp. 91–103, 1990.

[2] C. H. Lay and P. Burns, "Intentions and behavior in studying for an examination: The role of trait procrastination and its interaction with optimism," *Journal of Social Behavior and Personality*, vol. 6, no. 3, pp. 605–617, 1991.

[3] J. R. Ferrari and J. F. Díaz-Morales, "Procrastination: Different time orientations reflect different motives," *Journal of Research in Personality*, vol. 41, no. 3, pp. 707–714, 2007.

[4] S. J. Vodanovich and H. M. Seib, "Relationship between time structure and procrastination," *Psychological Reports*, vol. 80, no. 1, pp. 211–215, 1997.

[5] P. Steel, *The procrastination equation: How to stop putting things off and start getting stuff done*. Random House Canada, 2010.

[6] D. Ariely and K. Wertenbroch, "Procrastination, deadlines, and performance: Self-control by precommitment," *Psychological Science*, vol. 13, no. 3, pp. 219–224, 2002.

[7] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*, 2016, pp. 49–60.

[8] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic online matching in real-time spatial crowdsourcing," in *ICDE*, 2017, pp. 1009–1020.

[9] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *PVLDB*, vol. 10, no. 11, pp. 1334–1345, 2017.

[10] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou, "Latency-oriented task completion via spatial crowdsourcing," in *ICDE*, 2018, pp. 317–328.

[11] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *PVLDB*, vol. 9, no. 12, pp. 1053–1064, 2016.

[12] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *SIGMOD*, 2018, pp. 773–788.

[13] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "SLADE: A smart large-scale task decomposer in crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1588–1601, 2018.

[14] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of Combinatorial Optimization*, vol. 4, no. 2, pp. 171–186, 2000.

[15] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *SIGKDD*, 2007, pp. 420–429.