Introduction to Parameterized Complexity with Single Machine Scheduling with Rejection Problem

Yuan Wei and Maxim Shelopugin

Abstract

Parameterized complexity is an area of computational complexity theory focused on classifying intractable computational problems in a fine-grained way, and providing guidance of designing efficient algorithms with respect to their parameters. In this summary paper we present an introduction to the parameterized complexity, discuss the classes of parameterized complexity algorithms (XP, FPT, W Hierarchy), and provide examples of classifying a single machine scheduling with rejection problem by various parameters.

1 Introduction

Parameterized complexity was introduced by Rod Downey and Michael Fellows in the 90's[1]. It is a part of the computational complexity theory focused on classifying the intractable computational problems under standard complexity analysis with respect to their parameters. It is a useful method to analyze the complexity of NP-hard problems and reason about their possible solutions. In this paper we give a brief introduction to the parameterized complexity and illustrate the way it is used to analyze the complexity of a single machine scheduling with rejection problem when parameterized by the number of accepted jobs $|A|$, the number of different processing times v_p, and the maximal processing time p_{max} by Hermelin et al.[2].

2 Parameterized Complexity

The goal of parameterized complexity is to analyze the tractability of NP-hard problems with respect to some of their parameters not related to the size of the description of the problem. The parameter measures some aspect of the problem and when the influence of such a parameter to overall complexity is small the problem becomes tractable. In other words, the problem can be solved efficiently by a computer algorithm in pseudo-polynomial time. The pseudo-polynomial time algorithm is an algorithm which running time is a polynomial in the numeric value of the input (i.e. the largest integer present in the input), but not necessarily in the length of the input (i.e. the number of bits required to represent it). Formally, if we have a NP-hard problem which can be parameterized, we define it as a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is the problem space, and $\mathbb{N}$ is the parameter space. Hermelin et al.[2] bring up various parameterized complexity classes in their paper: Fixed Parameter Tractable (FPT), XP, and classes in W hierarchy which we are going to discuss next.

2.1 Class FPT

A problem P is Fixed Parameter Tractable (FPT) with respect to some parameter k if there exists an algorithm which solves any instance $p \in P$ (or a decision problem $(p, k) \in L$?) in $O(f(k)|p|^{O(1)})$, where $f(k)$ is computable and $O(1)$ is a constant. Notable example of FPT problem is satisfiability: given a formula of size $|p|$ and k variables, it can be solved in $2^k|p|$ by brute force. The motivation behind the FPT algorithm is to separate the factors of problem size and the parameter k. In this manner, the runtime depends on the $f(k)$ or $|p|$ which solely contribute to the pseudo-polynomial complexity and the overall complexity is dictated by a dominant value.

2.2 Class XP

A problem P is XP with respect to some parameter k if there exists an algorithm, which solves any instance $p \in P$ (or a decision problem $(p, k) \in L$?) in $O(|p|^{f(k)})$, where $f(k)$ is computable. XP involves all FPT independent parameters of the problem as part of the complexity analysis. Thus it makes XP as a superset of FTP. If k is fixed, the problem is solved in "slice-wise" polynomial time. In contrast to FPT, XP makes the parameter an exponent of the instance size (hence the name). Now, the function which determines the complexity of the problem with respect to the parameter k has a much greater impact on time complexity of the problem in asymptotic sense, and both the parameter and the problem size have an impact on the overall runtime complexity.

2.3 Classes in W Hierarchy

The W hierarchy is a collection of computational complexity classes. The notation W stands for the weft of the boolean circuit which solves an instance of t-weft k-satisfiability (t-weft k-SAT) problem, and the hardness of the class is tied to the level of the difficulty of such a problem.

Boolean circuit is a directed acyclic graph which contains n input gates, and a single output gate. The gates can be AND, OR, or NOT. The values of input or output gates can only be 0 or 1 (True or False). In the circuit, the depth is defined as d, which is the maximum length of a path from an input gate to the output gate. The gate in a boolean circuit is considered to be large if it has more than two inputs. The weft is defined as t, which is the maximum number of large gates on any path from an input gate to the output gate. The t-weft k-SAT problem is a known NP-complete problem: given a boolean circuit, decide if there is an assignment of inputs which assigns exactly k number of 1 (or True)'s to the input gates, that makes the output gate be 1 (or True).

Based on the t-weft k-SAT problem, the complexity class in W hierarchy is defined as $W[t]$ (t is the weft of the circuit): a problem P is in $W[t]$ class, if every instance (p, k) can be reduced in parameterized time to the correlated t-weft k-SAT problem that has a constant depth d and a weft at most t, such that $(p, k) \in L$ if and only if there is a satisfying assignment of k ones to the inputs. In other words, the classes in W hierarchy describe all the problems that are related to each other by the amount of large gates in a boolean circuit. It is believed that FPT $= W[t] \quad (t = 0)$ and FPT $\neq W[t] \quad (t \geq 1)$.

3 Parameterized Complexity Analysis Method

Though there are various approaches to classify parameterized problems, the most common method used in the analysis of parameterized complexity is reduction. To perform a parameterized reduction from problem P_2 to P_1, we need to show there is an algorithm or expression that maps an instance I_2 of P_2 with parameter k_2 to an instance I_1 of P_1 with parameter k_1, where I_1 is a yes-instance of P_1 if and only if I_2 is a yes-instance of P_2, and such algorithm runs in $O(f(k_2)|I_2|^{O(1)})$ time, and $k_1 \leq f(k_2)$.

4 Single Machine Scheduling with Rejection Problem

With the major parameterized complexity classes and method introduced previously, we show the examples of parameterized complexity analysis that Hermelin et al. perform on the single machine scheduling with rejection problem with respect to the number of accepted jobs $|A|$, the number of different processing times v_p and the maximum processing time p_{max}[2].

The offline deterministic scheduling problem with job rejection constraint is a NP-hard problem[3]. The goal of the problem is to create a schedule from the set of jobs, such that a specific constraint is satisfied. Hermelin et al. focus on a single machine scheduling problem with the ability of rejecting jobs that the machine does not like with the related costs. The goal of this problem is to find a solution minimizing the total completion time of all accepted jobs, with the total rejection cost not exceeding a predefined threshold. More formally, the problem has a set of n jobs: $\mathcal{J} = \{J_1, J_2, ..., J_n\}$. Each job $J_j \quad (j = 1, 2, ..., n)$ is associated with two parameters: processing time p_j and rejection cost e_j. Jobs are independent and non-preemptive, meaning jobs have no precedence relationship among each other, and a job cannot be removed from the machine before it is completed. C_j is the completion time of a job J_j, and C_j equals to the processing time p_j plus the waiting time. In their analysis, they partition the set $\mathcal{J}$ into two subsets: the set of accepted jobs A and the set of rejected jobs $\overline{A}$. The goal is to build a set of accepted jobs $A \subseteq \mathcal{J}$ with constraints $F_1 = \sum_{J_j \in A} C_j$ subject to $F_2 = \sum_{J_j \in \overline{A}} e_j \leq R$.

Hermelin et al. want to find out whether the problem becomes tractable when some specific parameters are fixed. We show three parameters they investigate: the cardinality of the set of accepted jobs, the number of different processing times and the maximal processing time, for the single machine scheduling with rejection problem. The results of the analysis of the tractability of three cases of the parameterized problem they find out is that the problem with respect to the number of accepted jobs $|A|$ is W[1]-hard, the problem with respect to the number of different processing times v_p is FPT, and the problem with respect to the maximal processing time p_{max} is FPT as well.

5 Analysis of Scheduling Problem Parameterized by $|A|$

Hermelin et al. use parameterized reduction to show that the single machine scheduling with rejection problem with respect to the number of accepted jobs $|A|$ is W[1]-hard[2] by reducing it to an instance of k-SUM problem. k-SUM is a subset sum problem where given a set of integers $X = \{x_1, x_2, ..., x_h\}$ (where $|X| = h$), an integer k, and a goal integer B, we need to determine whether there exists a set S (where $S \subseteq X$) such that $|S| = k$ and $\sum_{x_i \in S} x_i = B$.

Hermelin et al. propose a way to partition and map the scheduling problem to the k-SUM problem. They construct an instance for the decision version of the scheduling problem by splitting the set of n jobs: $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ into k subsets (i.e. $\{\mathcal{J}_1, \mathcal{J}_2, ..., \mathcal{J}_k\}$), with all sets having a total of h jobs. They note J_{ij} be the jth job in the set $\mathcal{J}_i$. Then they set its rejection cost $e_{ij} = (M_i + x_j)$ and its processing time $p_{ij} = (M_i + x_j)/(k - i + 1)$, where $M_i = (kA - B)h^{i-1}$ and $A = \sum_{x_i \in S} x_i$. In this decision version of the scheduling problem, they ask if there is a solution with constraints satisfied: $\sum_{J_{ij} \in A} C_j \leq K = (kA - B) \sum_{i=1}^{k} h^{i-1} + B$ and $\sum_{J_{ij} \in \overline{A}} e_j \leq R = (h-1)(kA - B) \sum_{i=1}^{k} h^{i-1} + kA - B$. With this setup, they are able to merge two problems together. They use induction to prove that these two constraints hold for $i = k$, and show the proofs bidirectionally. Since there exists a yes-instance for the k-SUM problem, if and only if there exists a yes-instance (or a feasible schedule) for the job scheduling problem with respect to the parameter $|A|$, and the constraints are computable, they successfully reduce the scheduling problem with respect to the parameter $|A|$ to the k-SUM problem with parameter k. Since the k-SUM problem is proved to be W[1]-hard[4], they claim that the single machine scheduling with rejection problem with respect to the number of accepted jobs $|A|$ is W[1]-hard as well.

6 Analysis of Scheduling Problem Parameterized by v_p

Hermelin et al. create a Mixed Integer Convex Programming (MICP) formulation of the scheduling problem to show that it is FPT with respect to the number of different processing times v_p[2]. They transform the scheduling problem with respect to parameter v_p into a MICP formulation. MICP is a mathematical optimization method to minimize a convex function over a convex domain, where variables in such domain are not constrained to be only integers (i.e. some variables can be non-integers).

Hermelin et al. propose a way to partition the scheduling problem to formulate it as the MICP: they partition the set of n jobs: $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ into $k = v_p$ subsets, where all jobs in $\mathcal{J}_i \quad (i = 1, 2, ..., k)$ have the same processing time of p_i. By assuming $p_1 \leq p_2 \leq ... \leq p_k$ and setting $x_i \quad (i = 1, 2, ..., k)$ as the accepted jobs from set $\mathcal{J}_i$, setting $y_i \quad (i = 1, 2, ..., k)$ as the rejected jobs from set $\mathcal{J}_i$, and sorting jobs in non-decreasing order of rejection costs in each set $\mathcal{J}_i$, they are able to show that both constraint expressions $\sum_{i=1}^{k} \sum_{j=1}^{y_i} e_{ij}$ and $\sum_{J_{ij} \in A} C_{ij} = \frac{1}{2}(\sum_{i=1}^{k} p_i x_i + \sum_{i=1}^{k} (\sum_{l=1}^{i} p_l x_l)^2 (\frac{1}{p_i} - \frac{1}{p_{i+1}}))$ are convex functions, where e_{ij} is the rejection cost of job J_{ij}, C_{ij} is the completion time of job J_{ij}, and $\frac{1}{p_{k+1}} = 0$, for $i = 1, 2, ..., k$. The proof of the correctness of the formulation of the constraints above is provided by Knop & Koutecký[5]. The MICP formulation for the problem with a parameterized number of integer variables is proved to be FPT[6]. Thus, the problem of solving such MICP formulation, the single machine Scheduling with Rejection Problem with respect to the number of different processing times v_p, is FPT as well.

7 Analysis of Scheduling Problem Parameterized by p_{max}

Hermelin et al. use dynamic programming technique to show that the single machine scheduling with rejection problem with respect to the maximal processing time p_{max} is FPT[2]. To start the derivation they define the domination rule: given two partial schedules S_1 and S_2, S_1 dominates S_2 if l_1 (the number of accepted jobs in S_1), C_1 (the minimum total completion time of accepted jobs in S_1), and E_1 (the total rejection cost of all rejected jobs in S_1) are less than or equal to the proportional metrics in S_2. Considering two such partial schedules S_1 and S_2: each set contains some amount of jobs $\{J_1, J_2, ..., J_i\}$. Without loss of generality they sort all the jobs in sets by processing times in non-descending order. Given the formulation above, they define the function $F_j(C, l)$ as the minimal total rejection cost over l accepted jobs in some partial schedule j, with a total completion cost C. To solve the problem they define the recursion form as $F_j(C, l) = min(F_{j-1}(C, l) + e_j, F_{j-1}(C - lp_j, l - 1))$, which states that the point $F_j(C, l)$ is reachable either by rejecting a job at stage $j - 1$ and hence not change the internals but increase the cost by e_j, or by accepting the job at stage $j-1$ from state $(C - lp_j, l-1)$ due to the fact that acceptance increases the amount of accepted jobs l and increases the total completion cost C by increasing each of l previously accepted jobs by p_j. Then they initialize the recursion with $F_0(0,0) = 0$ and ∞ everywhere else. Once the computation is done, the optimal solutions is given by: $\hat{C} = min(c \in C)|F_n(c,l) \leq R$. Given the fact that $j \in 1,2,...,n,$ and $l \in 1,2,...,j$, and C is bounded by $\sum_{i=1}^{l} ip_i$, the solution of the single machine scheduling with rejection problem with respect to the maximal processing time p_{max} requires $O(n^2 \sum_{i=1}^{l} ip_i)$, which is equivalent to $O(n^4 p_{max})$ (i.e. FPT).

8 Conclusion and Future Work

In this paper we introduce and discuss parameterized complexity with its subclasses followed by the examples of an application on a single machine scheduling with rejection problem with respect to three different parameters. Under the assumption of $P \neq NP$, the analysis of parameterized complexity is a useful tool to identify how hard a problem is from different perspectives as well as to find the efficient algorithms for those NP-hard problems with the ability to tune solutions for specific instances of the problem in respect to some desirable parameter. If the parameter's impact on the complexity is low enough for it to be practical, it is efficient to use fixed-parameterized tractable algorithm instead of brute-force algorithm to solve it in pseudo-polynomial time, which is not always plausible, but often more desirable than exponential.

One of the potential extension is to tune the analysis of the parameterized complexity into a tool which can process multiple parameters of a problem and verify its tractability in practice. In this situation we consider the inputs are equivalent as the parameters of the problem. This is useful when the problem involves more than one dimension of the data whose amounts are on the approximate same degree. If we are able to know if the sizes of the parameters (or inputs) are tractable or not, by applying the analysis of *the multi-parameterized complexity*, we can customize our algorithm specifically for these parameters (or inputs), in order to maximize the efficiency of the algorithm. A simple example of a problem with only two parameters (or inputs) is finding the optimal sequence alignments from two sequences having different sizes in biology. Myers and Miller propose an optimal alignments algorithm in linear space[7]. The algorithm is designed to reduce the computer space usage due to the massive size of the DNA sequence. The algorithm has $O(NM)$ time complexity and $O(N + logM)$ space complexity, where N and M are the sizes of two sequences to be aligned. Assume we apply the analysis of *the multi-parameterized complexity* to this algorithm, we should be able to at least identify the tractability of the algorithm, and thus find out the correct setup of the algorithm. For example, if we find out $M > N$, and $N + logM$ is feasible to be processed in the given computer memory (but $M + logN$ is not), the setup of the algorithm with $O(N + logM)$ space complexity is the applicable one. In other cases, we may find out neither $N + logM$ or $M + logN$ is tractable, then the algorithm proposed is not applicable regardless of its setup for the given sequences. Though this example is trivial, we can see the potential strength of this extension: the analysis of *the multi-parameterized complexity* provides us a method to analyze the parameters (or inputs) with close degrees, to help us tune up the algorithm and find out if there exists a version of the algorithm is tractable for the problem with the given parameters or inputs.

9 Questions to the Audience

- What class of parameterized complexity does Knapsack 0-1 problem belong to and why?
- What is the innate difference between XP and FPT classes and what are the relationships between them in parameterized complexity overall?

References

[1] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[2] Danny Hermelin, Michael Pinedo, Dvir Shabtay, and Nimrod Talmon. On the parameterized tractability of single machine scheduling with rejection. *European Journal of Operational Research*, 273(1):67 – 73, 2019.

[3] Liqi Zhang, Lingfa Lu, and Jinjiang Yuan. Single-machine scheduling under the job rejection constraint. *Theoretical Computer Science*, 411(16):1877–1882, 2010.

[4] R. G. Downey and M. R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 36–49, June 1992.

[5] Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, Oct 2018.

[6] D. Dadush, C. Peikert, and S. Vempala. Enumerative lattice algorithms in any norm via m-ellipsoid coverings. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 580–589, Oct 2011.

[7] Eugene W. Myers and Webb Miller. Optimal alignments in linear space. *Bioinformatics*, 4(1):11–17, 03 1988.