Speeding-Up Parallel Computation of Large Smooth-Degree Isogeny Using Precedence-Constrained Scheduling

Kittiphon Phalakarn[1(✉)], Vorapong Suppakitpaisarn[2], and M. Anwar Hasan[1]

[1] University of Waterloo, Ontario, Canada
{kphalakarn,ahasan}@uwaterloo.ca
[2] The University of Tokyo, Tokyo, Japan
vorapong@is.s.u-tokyo.ac.jp

Abstract. Although the supersingular isogeny Diffie-Hellman (SIDH) protocol is one of the most promising post-quantum cryptosystems, it is significantly slower than its main counterparts due to the underlying large smooth-degree isogeny computation. In this work, we address the problem of evaluating and constructing a *strategy* for computing the large smooth-degree isogeny in the *multi-processor setting* by formulating them as scheduling problems with dependencies. The contribution of this work is two-fold. For the strategy evaluation, we transform strategies into task dependency graphs and apply precedence-constrained scheduling algorithms to them in order to find their costs. For the strategy construction, we construct strategies from smaller parts that are optimal solutions of integer programming representing the problem. We show via experiments that the proposed two techniques together offer more than 13% reduction in the strategy costs compared to the best current results by Hutchinson and Karabina presented at Indocrypt 2018.

Keywords: SIDH · Isogeny-based cryptography · Parallel computing · Precedence-constrained scheduling

1 Introduction

The supersingular isogeny Diffie-Hellman (SIDH) protocol is a post-quantum key exchange protocol introduced by De Feo, Jao, and Plût in 2011 [13], where its security is based on the hardness of supersingular isogeny problems. SIDH was parameterized as the supersingular isogeny key encapsulation (SIKE) protocol [5] and was submitted to the NIST post-quantum cryptography standardization project in 2017 [2]. As announced in 2020, SIKE was selected as one of the alternate candidates [1].

SIDH requires relatively smaller public keys but takes more computation time compared to other schemes [4]. This is because SIDH requires both parties to perform large smooth-degree (i.e., all factors of the degree are small primes) isogeny computations, which are the bottleneck of the protocol. To reduce the computation time of SIDH, an abstraction of large smooth-degree isogeny computation called *strategy* was proposed in [13]. In that paper, the authors gave

K. Nguyen et al. (Eds.): ACISP 2022, LNCS 13494, pp. 309–331, 2022.
https://doi.org/10.1007/978-3-031-22301-3_16

a method to compute the *cost* of a strategy, an abstraction for the computation time. Intuitively, a low-cost strategy will lead to a fast implementation of SIDH. The paper also presented how to construct an *optimal strategy*, a strategy giving the lowest cost among all possible strategies. These optimal strategies are then utilized to implement SIDH and SIKE in order to reduce computation time. Apart from this, several other works were proposed towards lowering the computation time of SIDH [12,14,22,25].

The aforementioned techniques, however, do not consider parallelism and are targeted towards the *single-processor setting*. When we can utilize more than one core or processor, which is the case in many situations these days, we have *multi-processor setting*. Since multiple operations can be performed simultaneously in this setting, we can finish the computation faster. For example, under the multi-processor setting, the SIDH hardware architecture in [21] performed up to 42% faster than previous works. Recently, another fast parallel architecture was introduced in [20].

Apart from strategy construction, another important aspects of the isogeny computation in the multi-processor setting is strategy *evaluation*: the cost of a strategy now depends on how it is evaluated. And to achieve the least cost, both strategy construction and evaluation have to be designed specifically for the number of processors provided. In the works of [20] and [21] mentioned earlier, both implementations evaluate strategies designed for the single-processor setting. Hence, those computation times are not necessarily optimal for the multi-processor setting.

To the best of our knowledge, the only works that construct and evaluate strategies specifically for the multi-processor setting are the work of Hutchinson and Karabina [18], and that of Cervantes-Vázquez et al. [9], where the latter is a software implementation of the former. The two evaluation techniques for the multi-processor setting proposed in [18] are *per-curve parallel (PCP)* and *consecutive curve parallel (CCP)* (see Subsect. 2.4). The results of [18] show that, in the multi-processor setting, strategies constructed specifically for the multi-processor setting lead to lower costs than strategies constructed for the single-processor setting. Moreover, under SIKEp751 parameters with eight processors, their multi-processor setting based approach can achieve more than 50% reduction in the strategy cost compared to the single-processor setting. For two, three, and four processors, the reductions in strategy costs are 30%, 40%, and 46%, respectively. And, the maximum cost reduction achieved when we have arbitrary-many processors is 74%. When utilizing strategies and evaluations from [18] in the implementation of SIKEp751 with three processors, together with other optimizations, [9] could achieve more than 30% speedups in the computation time compared to the single-processor setting.

Nonetheless, the strategy costs reported in [18] are not the least we can achieve as their evaluations do not fully utilize available processors. (We will discuss this in Sect. 3.) In this work, we follow a different approach and formulate this problem as precedence-constrained scheduling problems. To our knowledge, our work is the first for this approach to be applied in reducing the strategy

cost for the computation of large smooth-degree isogenies. Our contribution is two-fold and consists of a novel strategy evaluation and construction techniques leading to lower strategy costs:

1. For the strategy evaluation (Sect. 3), we transform strategies into task dependency graphs and then apply two precedence-constrained scheduling algorithms, Hu's [17] and Coffman-Graham's [10] algorithms, to them in order to calculate the cost of strategies.
2. For the strategy construction (Sect. 4), we formalize the problem as an integer linear program (ILP) and then construct efficient strategies as a combination of optimal solutions to the ILP by structures of PCP.

We list techniques for strategy evaluation and construction of related works in Table 1. Our experimental results show that the application of our proposed techniques leads to more than 13% reduction in strategy cost compared to those reported by [18] under the same parameter sets.

Table 1. Strategy evaluation and construction techniques used in various works.

Works	Strategy evaluation	Strategy construction
[13]	Single operation at a time	Optimal for single-processor setting
[20,21]	PCP	Optimal for single-processor setting
[9,18]	PCP and CCP	Optimal under PCP (multi-processor)
Ours	Precedence-constrained scheduling	Using ILP and PCP

2 Preliminaries

In this section, we review some preliminaries on SIDH, strategies for computing large smooth-degree isogeny, how strategies can be evaluated in the single-processor and multi-processor settings, and precedence-constrained scheduling algorithms.

2.1 SIDH

Let E and E' be elliptic curves over a field F where their identity elements are ∞ and ∞', respectively. An isogeny from E to E' is a morphism $\phi : E \to E'$ satisfying $\phi(\infty) = \infty'$. When specifying an elliptic curve E and a point $R \in E(F)$, one can compute the unique isogeny $\phi : E \to E' = E/\langle R \rangle$ satisfying $\ker \phi = \langle R \rangle$ using Vélu's [27] or $\sqrt{}$élu's [6] formulas. The degree of ϕ is equal to the order of R. Using these notations, SIDH can be described as follows.

Setup: Alice and Bob agree on the following set of public parameters:

- a prime p of the form $\ell_A^{e_A} \ell_B^{e_B} \cdot f \pm 1$ where ℓ_A, ℓ_B are small primes, e_A, e_B are exponents giving $\ell_A^{e_A} \approx \ell_B^{e_B}$, and f is a positive integer,
- a supersingular elliptic curve E_0 over $\mathbb{F}_{p^2}$ with $\#E_0(\mathbb{F}_{p^2}) = (\ell_A^{e_A} \ell_B^{e_B} \cdot f)^2$,
- bases $\{P_A, Q_A\}$ of $E_0[\ell_A^{e_A}]$ and $\{P_B, Q_B\}$ of $E_0[\ell_B^{e_B}]$.

Key Exchange:

1. Alice randomly chooses $m_A \in \mathbb{Z}_{\ell_A^{e_A}}$. She computes an isogeny $\phi_A : E_0 \to E_A$ with kernel $\langle R_A \rangle$ where $R_A = P_A + [m_A]Q_A$, and then sends E_A, $\phi_A(P_B)$, $\phi_A(Q_B)$ to Bob.
2. Similarly, Bob randomly chooses $m_B \in \mathbb{Z}_{\ell_B^{e_B}}$. He computes $\phi_B : E_0 \to E_B$ with kernel $\langle R_B \rangle$ where $R_B = P_B + [m_B]Q_B$, and sends E_B, $\phi_B(P_A)$, $\phi_B(Q_A)$ to Alice.
3. Upon receiving $E_B, \phi_B(P_A), \phi_B(Q_A)$ from Bob, Alice computes an isogeny $\phi'_A : E_B \to E_{AB}$ with kernel $\langle R'_A \rangle$ where $R'_A = \phi_B(P_A) + [m_A]\phi_B(Q_A)$.
4. Similarly, upon receiving $E_A, \phi_A(P_B), \phi_A(Q_B)$ from Alice, Bob computes $\phi'_B : E_A \to E_{BA}$ with kernel $\langle R'_B \rangle$ where $R'_B = \phi_A(P_B) + [m_B]\phi_A(Q_B)$.
5. The shared secret is the j-invariant of the resulting elliptic curves: $j(E_{AB}) = j(E_{BA})$, where $j(E) = 1728\frac{4a^3}{4a^3+27b^2}$ for $E : y^2 = x^3 + ax + b$.

2.2 Large Smooth-Degree Isogeny Computation and Strategies

SIDH requires several computations of isogenies of the form $\phi : E \to E'$ with kernel $\langle R \rangle$ and degree ℓ^e. Theoretically, the degree of isogenies in SIDH can be any sufficiently large integer, but we have not found an efficient way to compute them using Vélu's or $\sqrt{}$élu's formulas. For large smooth-degree (e.g., degree-ℓ^e for small ℓ and large e) isogenies, an efficient way exists, which is to decompose ϕ as a chain of degree-ℓ isogenies [13]:

$$\phi : E = E_0 \xrightarrow{\phi_0} E_1 \xrightarrow{\phi_1} E_2 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_{e-2}} E_{e-1} \xrightarrow{\phi_{e-1}} E_e = E/\langle R \rangle$$

where, for $0 \leq i < e$, $E_{i+1} = E_i/\langle [\ell^{e-i-1}]R_i \rangle$, $R_{i+1} = \phi_i(R_i)$, and $R_0 = R$. We note that $R'_i = [\ell^{e-i-1}]R_i$ is required in order to compute ϕ_i and E_{i+1}. This suggests the following procedure given in Algorithm 1 for computing $\phi_0, \ldots, \phi_{e-1}$.

We can describe Algorithm 1 using a graph with $\frac{e(e+1)}{2}$ vertices arranged in e columns and e rows as shown in Fig. 1(a). Each vertex represents a point where points in each column are on the same elliptic curve. The vertex at the upper left corner represents the point R_0 and the leftmost column are points on E_0. The top-to-bottom arrows depict point multiplications by $[\ell]$ in Line 4 of the algorithm and the left-to-right arrows depict isogeny evaluations in Line 6. Here, ϕ_{e-1}, E_e, and R_e are omitted as they are not relevant for analysis.

In Fig. 1(a), one might notice that R'_1 can also be computed by $R'_1 = \phi_0([\ell^{e-2}]R_0)$. This gives other possible ways of computing large smooth-degree isogenies. By considering how each point in the graph can be computed from other points, we define the graph T_e following [13] which shows all possible point multiplications by $[\ell]$ and isogeny evaluations among all vertices. For simplicity, vertices are referred to by pairs of their column and row numbers, i.e., vertex (i, j) refers to the point $[\ell^j]R_i$ in column i and row j. Vertices representing R'_i, i.e., vertices $(i, e-i-1)$ for $0 \leq i < e$, are called *leaves*.

Definition 1. *The graph showing all possible operations for computing degree-ℓ^e isogeny is defined as a directed graph $T_e = (V_e, E_e)$ where*

Algorithm 1: An algorithm for computing degree-ℓ^e isogeny.

Input : A supersingular elliptic curve E and a point R of order ℓ^e
Output: $\phi_0, \ldots, \phi_{e-1}$ and $E/\langle R \rangle$

$E_0 \leftarrow E$, $R_0 \leftarrow R$
for $i = 0$ **to** $e - 1$ **do**
  $R'_i \leftarrow R_i$
  for $j = 1$ **to** $e - i - 1$ **do** $R'_i \leftarrow [\ell]R'_i$
  Use Vélu's or √élu's formulas to compute ϕ_i and E_{i+1} from E_i and $\langle R'_i \rangle$
  $R_{i+1} \leftarrow \phi_i(R_i)$
return $\phi_0, \ldots, \phi_{e-1}, E_e$

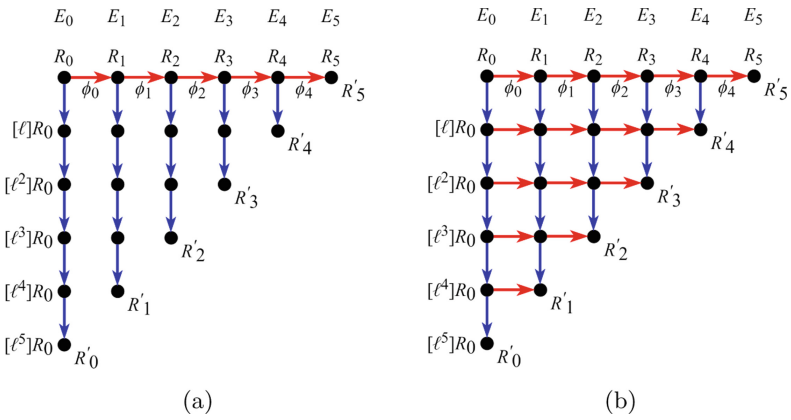

Fig. 1. (a) The graph representing Algorithm 1 and (b) the graph T_e when $e = 6$.

- *the set of vertices* $V_e = \{(i, j) : 0 \leq i, j < e;\ i + j < e\}$,
- *the set of directed edges* $E_e = \downarrow_e \cup \rightarrow_e$,
- *the set of point multiplication edges* $\downarrow_e = \{\langle (i, j), (i, j + 1) \rangle : i + j < e - 1\}$,
- *the set of isogeny evaluation edges* $\rightarrow_e = \{\langle (i, j), (i + 1, j) \rangle : i + j < e - 1\}$.

Next, we define a *strategy* for computing degree-ℓ^e isogeny as follows.

Definition 2. *A* strategy *S for computing degree-ℓ^e isogeny is a subgraph of T_e containing the vertex $(0, 0)$ and all leaves where there are paths from the vertex $(0, 0)$ to each leaf. A strategy S is* well-formed *if removing any edge from S results in a graph that is not a strategy.*

An example of a strategy is the graph in Fig. 1(a). By the definition, one can use a strategy to compute a degree-ℓ^e isogeny by first performing operations along a path from $(0, 0)$ to R'_0, then a path from $(0, 0)$ to R'_1, and so on. Since strategies that are not well-formed have some unnecessary edges, we will consider only well-formed strategies in order to find an efficient strategy.

Now we look at how a strategy can be evaluated which defines the cost of a strategy. We will define the strategy cost using the cost of a single point multiplication by $[\ell]$: $Q \leftarrow [\ell]P$, and the cost of a single degree-ℓ isogeny evaluation: $Q \leftarrow \phi(P)$. We denote their costs as $c_{\downarrow}$ and $c_{\rightarrow}$, respectively.

2.3 Single-Processor Setting

When only a single processor is provided, we have to perform all operations sequentially. Formally, given a strategy S, one can compute a degree-ℓ^e isogeny using Algorithm 2.

Algorithm 2: Strategy evaluation in the single-processor setting.

Input : A strategy $S = (V_S, E_S)$, a curve E, and a point R
Output: $\phi_0, \ldots, \phi_{e-1}$ and $E/\langle R \rangle$

1 $E_0 \leftarrow E$, $R_{(0,0)} \leftarrow R$
2 **for** $i = 0$ **to** $e - 1$ **do**
3 **for** $j = 0$ **to** $e - i - 2$ **do**
4 **if** $\langle (i, j), (i, j + 1) \rangle \in E_S$ **then** $R_{(i,j+1)} \leftarrow [\ell]R_{(i,j)}$
5 $R' \leftarrow R_{(i,e-i-1)}$
6 Use Vélu's or $\sqrt{}$élu's formulas to compute ϕ_i and E_{i+1} from E_i and $\langle R' \rangle$
7 **for** $j = 0$ **to** $e - i - 2$ **do**
8 **if** $\langle (i, j), (i + 1, j) \rangle \in E_S$ **then** $R_{(i+1,j)} \leftarrow \phi_i(R_{(i,j)})$
9 **return** $\phi_0, \ldots, \phi_{e-1}, E_e$

From the above algorithm, we can define the cost of a strategy in the single-processor setting. For a strategy S, let $\#{\downarrow}_S$ denote the number of point multiplication edges in S and $\#{\rightarrow}_S$ denote the number of isogeny evaluation edges. Then, the cost of a strategy S in the single-processor setting, denoted by $C_1(S)$, is computed by

$$C_1(S) = \#{\downarrow}_S \cdot c_{\downarrow} + \#{\rightarrow}_S \cdot c_{\rightarrow}.$$

We emphasize that the strategy cost is only an abstraction for the SIDH computation time since we do not account for the cost of Vélu's or $\sqrt{}$élu's formulas (Line 6 of Algorithm 2) nor other operations required in SIDH (e.g., the cost of Alice computing $R_A \leftarrow P_A + [m_A]Q_A$, etc.). Nevertheless, the strategy cost is a useful measure in order to reduce the computation time of an implementation.

The problem of constructing a least-cost strategy given e, $c_{\downarrow}$, and $c_{\rightarrow}$ has been extensively studied in [13]. That work analyzed a particular type of strategies called *canonical* strategies and proved that a least-cost strategy in the single-processor setting must be in this form. A canonical strategy is defined below.

Definition 3. *A* canonical *strategy for computing degree-ℓ^e isogeny is defined recursively as follows:*

- *If $e = 1$, then T_1 is canonical.*

– *Otherwise, let* S_n*, where* $1 \leq n < e$*, be a canonical strategy for computing degree-*ℓ^n *isogeny. If* $S = (V_S, E_S)$ *is constructed from* $S_n = (V_{S_n}, E_{S_n})$ *and* $S_{e-n} = (V_{S_{e-n}}, E_{S_{e-n}})$ *by the following steps, then* S *is canonical.*
  1. *Rename all vertices* (i, j) *in* S_n *to* $(i, j + (e - n))$.
  2. *Rename all vertices* (i, j) *in* S_{e-n} *to* $(i + n, j)$.
  3. *Construct* $V_S = V_{S_n} \cup V_{S_{e-n}} \cup \{(0, j) : 0 \leq j < e - n\} \cup \{(i, 0) : 0 \leq i < n\}$ *and* $E_S = E_{S_n} \cup E_{S_{e-n}} \cup \{\langle (0, j), (0, j + 1)\rangle : 0 \leq j < e - n\} \cup \{\langle (i, 0), (i + 1, 0)\rangle : 0 \leq i < n\}$.

In brief, a canonical strategy with e leaves can be split into two canonical strategies with n leaves and $e - n$ leaves. Figure 2 depicts the process explained in Definition 3.

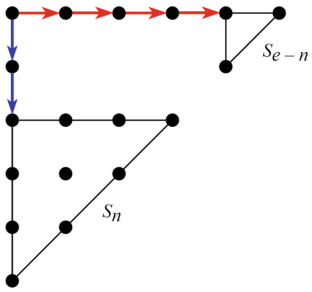

Fig. 2. A canonical strategy.

By exploiting the optimal substructure of the problem, the cost of a least-cost strategy for computing degree-ℓ^e isogeny in the single-processor setting can be calculated by the following recurrence [13]. We abuse the notation C_1 by defining $C_1(e)$ as the cost of a least-cost strategy with e leaves.

$$C_1(e) = \min_{1 \leq n < e} \{C_1(n) + C_1(e - n) + (e - n) \cdot c_{\downarrow} + n \cdot c_{\rightarrow}\}, \quad C_1(1) = 0.$$

2.4 Multi-processor Setting

In this setting, we are provided with $K \geq 2$ processors. At first, a K-time improvement from the single-processor setting might be expected. However, since we need to compute R'_i in order to continue to the next column, the computation is quite restricted and we are not able to fully utilize all processors at all times during the computation. Nevertheless, having multiple processors helps us reduce the cost as discussed next.

Before getting into the strategy cost, we review the implicit restrictions of the degree-ℓ^e isogeny computation. Unlike the single-processor setting, timing plays a crucial role here. Because now we can perform more than one operations at the same time, we have to be careful of which operations are performed first

and when they are finished, as they depend closely on each other. This is very important for achieving the least cost in this setting. In this work, we consider two restrictions of how a strategy is evaluated in parallel:

1. To perform a point multiplication by $[\ell]$ corresponding to a directed edge $\langle (i,j),(i,j+1)\rangle$, the vertex (i,j) corresponding to the point $[\ell^j]R_i$ must have been computed.
2. To perform an isogeny evaluation corresponding to a directed edge $\langle (i,j),(i+1,j)\rangle$, two vertices (i,j) and $(i,e-i-1)$ corresponding to the point $[\ell^j]R_i$ and R'_i, respectively, must have been computed. Here, the latter vertex R'_i is required to construct ϕ_i.

Even though the computation is restricted, there are still several ways of evaluating a strategy in parallel. To have a clearer picture of the problem, we consider the following example of how a strategy is evaluated. In order to specify which operations are performed at which time, each edge is labeled with its finish time. The cost of evaluating a strategy is then labeled on the edge $\langle (e-2,0),(e-1,0)\rangle$, which must be performed as the last operation.

Example 1. Suppose $K = 2$ and $c_\downarrow = c_\rightarrow = 1$. At time 0, although we have two processors, the only operation we are able to perform is the edge $\langle (0,0),(0,1)\rangle$. Again, at time 1, we can only take the edge $\langle (0,1),(0,2)\rangle$. We continue until the edge $\langle (0,3),(0,4)\rangle$ is done at time 4. This part is illustrated in Fig. 3(a).

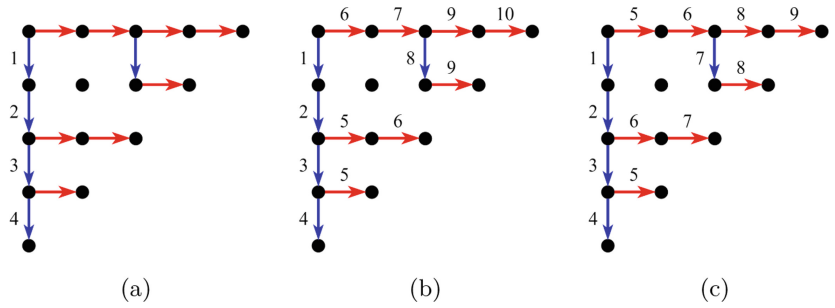

Fig. 3. Examples of parallel evaluations of a strategy with $K = 2$.

At time 5, we now have three options: $\langle (0,0),(1,0)\rangle$, $\langle (0,2),(1,2)\rangle$, and $\langle (0,3),(1,3)\rangle$. Because we have two processors, we can choose up to two operations. Figure 3(b) chooses the last two. After performing the remaining operations, the last operation is done at time 10. Thus, the cost of the evaluation in Fig. 3(b) is 10. On the other hand, Fig. 3(c) chooses operations $\langle (0,0),(1,0)\rangle$ and $\langle (0,3),(1,3)\rangle$. By this evaluation, its cost is only 9. We point out that this is the least possible cost from any strategy we can achieve when $e = 5$, $K = 2$, $c_\downarrow = c_\rightarrow = 1$. (Thus, a strategy giving the least cost in the multi-processor setting does not have to be canonical.)

From the above example, the multi-processor setting is much more complicated compared to the single-processor setting. In the rest of this subsection, we present the result of Hutchinson and Karabina [18] on constructing low-cost strategies and evaluations under some constraints called *per-curve parallel (PCP)* and *consecutive-curve parallel (CCP)*.

Per-curve Parallel. Hutchinson and Karabina started with a simple evaluation of a strategy called *per-curve parallel (PCP)*. Under PCP, two rules apply:

1. only operations of the form $\langle (i,j),(i+1,j)\rangle$ and $\langle (i,j'),(i+1,j')\rangle$ (i.e., isogeny evaluations from the same elliptic curve E_i) can be performed in parallel, and
2. point multiplications cannot be done in parallel. In other words, if one processor performs the edge $\langle (i,j),(i,j+1)\rangle$, other processors must be left idle.

The algorithm representing the strategy evaluation under PCP is similar to Algorithm 2, except that we can simultaneously perform up to K isogeny evaluations in Lines 7–8, i.e., when there are n isogeny evaluations from E_i, the cost of performing these isogeny evaluations is $\lceil \frac{n}{K} \rceil \cdot c_{\rightarrow}$. Let $\#{\rightarrow}_{S,i}$ denote the number of isogeny evaluation edges from E_i in a strategy S, the cost of evaluating S under PCP having K processors is

$$C_K^{\mathrm{PCP}}(S) = \#{\downarrow}_S \cdot c_{\downarrow} + \sum_{i=0}^{e-2} \left\lceil \frac{\#{\rightarrow}_{S,i}}{K} \right\rceil \cdot c_{\rightarrow}.$$

Even though the evaluation under PCP does not provide the least cost in the multi-processor setting, it allows an extensive analysis to construct a strategy with smallest $C_K^{\mathrm{PCP}}(S)$. While not stated in [18], the lemma below can be proved.

Lemma 1. *There exists a canonical strategy providing the least cost under PCP.*

Proof (sketch). Suppose we have a least-cost strategy under PCP that is not canonical, we can modify it to have a least-cost canonical strategy. First, we consider the leftmost leaf $(i', e-i'-1)$ connecting to $(0,0)$ via the edge $\langle (0,0),(1,0)\rangle$. If it is not connected to $(0,0)$ via the vertex $(i',0)$, we can remove the existing path and change it to the path $(0,0) \rightarrow (i',0) \rightarrow (i', e-i'-1)$. By this modification, $\#{\downarrow}_S$ and $\#{\rightarrow}_{S,i}$ for $0 \le i < i'$ do not increase. We perform similar actions with the rightmost leaf connecting to $(0,0)$ via the edge $\langle (0,0),(0,1)\rangle$. The modified strategy now has the same structure as a canonical strategy (Fig. 2) except that two smaller strategies might not be canonical. We can apply the same technique recursively to those smaller strategies to convert them into canonical strategies without increasing the cost. Therefore, we have a least-cost canonical strategy under PCP.

By the above lemma, we can construct a least-cost strategy under PCP by finding a least-cost canonical strategy. The optimal substructure of the problem allowed [18] to present a recurrence describing the least cost under PCP. Let $C_K^{\mathrm{PCP}}(e,k)$ denote the least cost of a strategy with e leaves where, in the first iteration of executing Lines 7–8 of Algorithm 2 in parallel for each curve, we can perform isogeny evaluations of only up to k points (instead of K points). Also, let $n' = e - n$. The recurrence for $C_K^{\mathrm{PCP}}(e,k)$ can be described as

$$C_K^{\mathrm{PCP}}(e,k) = \begin{cases} 0 & \text{if } e = 1, \\ C_K^{\mathrm{PCP}}(e,K) + (e-1)\cdot c_{\rightarrow} & \text{if } e > 1 \text{ and } k = 0, \\ \min\limits_{1 \le n < e} \{C_K^{\mathrm{PCP}}(n,k-1) + C_K^{\mathrm{PCP}}(n',k) + n'\cdot c_{\downarrow} + c_{\rightarrow}\} & \text{otherwise.} \end{cases}$$

The following theorem describes the least possible cost of a strategy under PCP.

Theorem 1 ([18]). *Let K, $c_{\downarrow}$, and $c_{\rightarrow}$ be fixed. The least cost of a strategy under PCP for computing degree-ℓ^e isogeny with K processors is $C_K^{\mathrm{PCP}}(e,K)$ (i.e., evaluating the above recurrence at $k = K$).*

We refer the interested readers to [18] for the detailed proof and explanation of the theorem.

Consecutive-Curve Parallel. Under PCP, we cannot perform any operation in different columns, even though it is allowed to do so and some processors are idle. By this observation, [18] considered another constraint called *consecutive-curve parallel (CCP)*. Let $\downarrow_{S,i}$ denote the set of point multiplication by $[\ell]$ edges for points in E_i in a strategy S and $\rightarrow_{S,i}$ denote that of isogeny evaluation edges. Under CCP, while performing operations in $\rightarrow_{S,i}$, we are allowed to perform operations in $\rightarrow_{S,i+1}$ and $\downarrow_{S,i+1}$ if they are ready to be done.

Because it is more flexible to perform operations in parallel under CCP, it is thus harder to analyze a strategy under this constraint. For this reason, [18] decided to consider only canonical strategies under CCP. As discussed before, operations in $\rightarrow_{S,i+1}$ can be performed after R'_{i+1} is computed. In the case that R'_{i+1} is computed by point multiplication edges in $\downarrow_{S,i+1}$, all operations in $\downarrow_{S,i+1}$ must be done first to obtain R'_{i+1}. By this, CCP uses a greedy heuristic to choose which operations will be performed as described in the following rules:

1. Operations in $\rightarrow_{S,i}$ are performed from bottom to top.
2. If an operation in $\downarrow_{S,i+1}$ is available, then perform one operation in $\downarrow_{S,i+1}$ and $K-1$ operations in $\rightarrow_{S,i}$.
3. If operations in $\downarrow_{S,i+1}$ are all done or there is no operation in $\downarrow_{S,i+1}$, start performing operations in $\rightarrow_{S,i+1}$ as soon as all in $\rightarrow_{S,i}$ is finished.
4. If operations in $\rightarrow_{S,i}$ are all done before $\downarrow_{S,i+1}$ is exhausted, then perform the remaining operations in $\downarrow_{S,i+1}$ before starting $\rightarrow_{S,i+1}$.

The algorithm for computing the cost under CCP of a canonical strategy S, denoted by $C_K^{\mathrm{CCP}}(S)$, is given in [18, Algorithm 1]. Nonetheless, Hutchinson and Karabina stated that, under CCP, they could find no algorithm for constructing least-cost strategies and no formula for the cost of a least-cost strategy.

We must note that, although performing operations in consecutive columns are allowed, performing operations in other columns are not. Thus, this heuristic could make the cost under CCP larger than the least possible.

2.5 Precedence-Constrained Scheduling Algorithms

The problem of scheduling a set of tasks to processors has been studied for a long time and has many applications in various fields. For a given set of tasks, we need to specify which processor performs which task and the goal is to minimize the time that the last task is finished. In this work, we are interested in the problem of *precedence-constrained* scheduling: we are given, for each task, a list of tasks need to be completed in order to start that task. Thus, we also have to specify the order in which tasks are performed by each processor. The dependency between tasks for this problem is usually specified using a task dependency graph defined as follows.

Definition 4. *Given a set of tasks* $T = \{t_1, ..., t_n\}$*, the* task dependency graph *for* T *is a directed acyclic graph (DAG)* $D_T = (V_{D_T}, E_{D_T})$ *where* $V_{D_T} = T$ *and* $\langle t_i, t_j \rangle \in E_{D_T}$ *if* t_i *must be performed and finished before* t_j *can begin.*

There are several variants of this problem, but we restrict ourselves to the case of the graphs D_T with all tasks are of unit-length (i.e., all tasks take the same amount of time to be performed), the number of processors is constant throughout the scheduling, all processors are identical (i.e., no processor performs tasks faster or slower than others) and preemption is not allowed (i.e., tasks cannot be paused and then resumed later). We formally give the definitions of a schedule and the precedence-constrained scheduling problem as follows.

Definition 5. *Let* $D_T = (V_{D_T}, E_{D_T})$ *be a task dependency graph, and let* K *be a positive integer. Suppose that all tasks require one unit of time to complete. A* scheduling of D_T using K processors *is a sequence* $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$ *of non-empty sets of tasks where* s_i *is a set of tasks executed at time* i *such that (i)* $s_1, \ldots, s_n$ *form a partition of* V_{D_T}*, (ii)* $|s_i| \leq K$*, and (iii) for all* $\langle t, t' \rangle \in E_{D_T}$*, if* $t \in s_i$ *and* $t' \in s_j$ *then* $i < j$*. The finished time of* $\mathcal{S}$ *is* n*, the size of* $\mathcal{S}$*, and is denoted by* $t(\mathcal{S})$*.*

A scheduling $\mathcal{S}$ *is* optimal *if* $t(\mathcal{S}) \leq t(\mathcal{S}')$ *for any possible scheduling* $\mathcal{S}'$ *of* D_T *using* K *processors. The* (precedence-constrained) scheduling problem *is to find an optimal scheduling for given* D_T *and* K*.*

For general DAGs, Ullman [26] proved that the problem is NP-complete, and Garey and Johnson [16] mentioned that complexity remains open when the number of processors $K \geq 3$ is fixed.

In the rest of this subsection, we look at two algorithms. The first algorithm by Hu [17] outputs an optimal scheduling for $K \geq 1$ when the task dependency graph is *tree-like*. The second algorithm by Coffman and Graham [10] produces an optimal scheduling when $K = 2$. When $K \geq 3$, no efficient algorithm has been proposed. Nonetheless, there are many approximation algorithms solving this problem with various approximation ratios [15,24].

Hu's Algorithm. The first algorithm applies with a task dependency graph which is tree-like, i.e., all vertices has out-degrees of at most one. For $u \in V_{D_T}$, let $\ell(u)$ denote the length of a longest path started at u. In a tree-like graph, the longest path started from each vertex is unique since all vertices has at most one out-going edge.

Hu's algorithm can be described as Algorithm 3. In short, the algorithm chooses up to K available tasks with largest $\ell(\cdot)$ in each iteration until all tasks are performed. The chosen tasks and their edges are then removed from the graph in order to show new available tasks.

Algorithm 3: Hu's algorithm [17].

Input : A tree-like task dependency graph $D_T = (V_{D_T}, E_{D_T})$ and the number of provided processors K
Output: An optimal scheduling $\mathcal{S} = \langle s_1, \ldots, s_t \rangle$

1 Compute $\ell(u)$ for all $u \in V_{D_T}$
2 $t \leftarrow 0$
3 **while** $V_{D_T} \neq \emptyset$ **do**
4 $\quad t \leftarrow t + 1$
5 $\quad V' \leftarrow \{u \in V_{D_T} : \text{in-degree of } u = 0\}$
6 $\quad$ Sort V' by $\ell(u)$ in an decreasing order, break ties arbitrarily
7 $\quad$ **if** $|V'| \leq K$ **then** $s_t \leftarrow V'$
8 $\quad$ **else** $s_t \leftarrow \{\text{the first } K \text{ vertices in } V'\}$
9 $\quad$ Remove all vertices in s_t and their associated edges from D_T
10 **return** $\mathcal{S} = \langle s_1, \ldots, s_t \rangle$

Coffman-Graham's Algorithm. Instead of using $\ell(\cdot)$, Coffman and Graham [10] presented another way to label vertices for DAGs of any structure without *transitive edges* defined as follows. After all vertices are labeled, the same technique as in Hu's algorithm is then applied, starting at Line 2.

Definition 6. *Given a directed graph $G = (V, E)$, an edge $e = \langle u, v \rangle \in E$ is* transitive *if there exists a vertex $w \notin \{u, v\}$ in V such that u reaches w and w reaches v.*

The labeling process of Coffman and Graham is described as Algorithm 4. We give an example of the function $c(\cdot)$ in Lines 7–8 as follows: Suppose u has

three children v_1, v_2, v_3 and all are labeled with $\ell_{\mathrm{CG}}(v_1) = 4$, $\ell_{\mathrm{CG}}(v_2) = 3$, and $\ell_{\mathrm{CG}}(v_3) = 8$. Then, $c(u)$ is the list $[8, 4, 3]$ as it is sorted in decreasing order. In Line 8, lists are compared lexicographically, e.g., $[4, 2, 1] < [4, 3]$, $[5, 4, 2] < [5, 4, 2, 1]$, and $[\,] < [3, 2]$.

At first, one vertex with no out-going edge is assigned a label of 1. In each iteration, one vertex is labeled. V'' in Line 5 is the set of unlabeled vertices with all children labeled. By the definition of V'', $c(\cdot)$ is well-defined for all vertices in V''. The next vertex to be assigned a label is $u \in V''$ with smallest $c(u)$. The label is assigned from 1 up to $|V_{D_T}|$.

Coffman and Graham proved that, by using $\ell_{\mathrm{CG}}(u)$ instead of $\ell(u)$ in Algorithm 3, the output scheduling is optimal when $K = 2$ for a task dependency graph of any structure. Few years later, Lam and Sethi [23] showed that, when applying Coffman-Graham's algorithm with $K \geq 2$, the algorithm is $(2 - \frac{2}{K})$-approximation. When K is small, the approximation ratio is close to 1.

Algorithm 4: Coffman-Graham's labeling algorithm [10].

Input : A task dependency graph $D_T = (V_{D_T}, E_{D_T})$
Output: Coffman-Graham's label $\ell_{\mathrm{CG}}(u)$ for all $u \in V_{D_T}$

1 Choose any vertex u with out-degree of 0 and assign $\ell_{\mathrm{CG}}(u) \leftarrow 1$
2 $idx \leftarrow 1$
3 **while** there is a vertex without a label **do**
4 $idx \leftarrow idx + 1$
5 $V'' \leftarrow \{u \in V_{D_T} : u$ is not labeled and all its children are labeled$\}$
6 **for** $u \in V''$ **do**
7 $c(u) \leftarrow$ the list of all labels of u's children, sorted in decreasing order
8 Choose $u \in V''$ with smallest $c(u)$ in lexicographical order, break ties arbitrarily
9 $\ell_{\mathrm{CG}}(u) \leftarrow idx$

3 Proposed Strategy Evaluation Technique

To the best of our knowledge, the evaluation of a canonical strategy under CCP gives the least cost among all existing techniques. In this section, we take a closer look at the problem and propose a new approach to evaluate strategies that gives lower costs. To this end, first we give an example showing that the cost under CCP of a canonical strategy is not the least cost we can achieve.

Example 2. Let $e = 9$, $K = 3$, and $c_{\downarrow} = c_{\rightarrow} = 1$. Below shows a canonical strategy which gives the least cost under PCP. When calculating its cost using [18, Algorithm 1], the cost under CCP is 20. The times at which each operation is finished are shown on the corresponding edges as in Fig. 4(a).

Consider another way of evaluating this strategy in Fig. 4(b). Here, the computation is not restricted by CCP. For instance, three isogeny evaluations ϕ_0, ϕ_1,

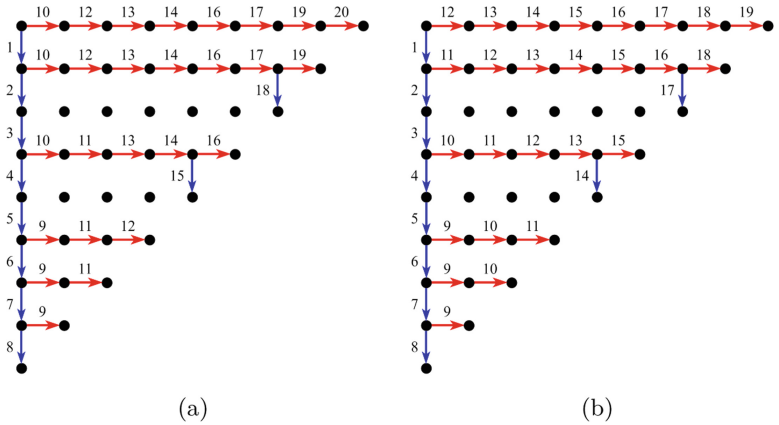

Fig. 4. A strategy giving a cost of 20 under CCP and 19 under another evaluation.

and ϕ_2 are performed in parallel at time 11. As another example, during time 14, two isogeny evaluations ϕ_2, ϕ_3, and a point multiplication on E_4 are done at the same time. These are not permitted under CCP or PCP. As a result, we achieve a lower cost of 19 for this strategy and evaluation.

It is important to note that, unlike the single-processor setting, a strategy in the multi-processor setting does not uniquely correspond to how it is evaluated. This does mean that, in order to obtain the least cost possible, we need to search for a strategy and its evaluation that give the least cost as a pair. Evaluating a good strategy in a wrong way might not give us a low cost. On the other hand, starting with a bad strategy will not give us a low cost under any evaluation. This makes it a challenging problem. Moreover, since it is possible that the least-cost strategy may not be canonical, we might not be able to utilize the recursive structure of canonical strategies to solve the problem.

In this section, we propose a new technique to evaluate strategies. The first part of the technique is to construct the task dependency graph of a strategy, and the second part is to evaluate a strategy by using its task dependency graph and precedence-constrained scheduling algorithms. We tackle the problem of constructing efficient strategies in Sect. 4.

3.1 Task Dependency Graphs of Strategies

Without loss of generality, we assume that for a given strategy $S = (V_S, E_S)$, all vertices in V_S that are unreachable from $(0, 0)$ are removed since they are not related to the cost computation. For any well-formed strategy, there is a unique path from $(0, 0)$ to any vertices in a strategy. This implies that every vertex in a well-formed strategy that can be reached from $(0, 0)$, except for $(0, 0)$, must have only one incoming edge. Thus, for a point (i, j) to be available, the operation representing the incoming edge to the point (i, j) must be completed. Therefore,

in a strategy, a point and its incoming edge represent the same thing. This concept is important in constructing the task dependency graphs of a strategy.

The task dependency graph of a strategy is defined as follows.

Definition 7. *The task dependency graph of a strategy* $S = (V_S, E_S = \downarrow_S \cup \rightarrow_S)$ *is a directed acyclic graph* $D_S = (V_{D_S}, E_{D_S})$ *where* $V_{D_S} = V_S \setminus \{(0,0)\}$ *and*

$$E_{D_S} = (E_S \cup \{\langle (i, e-i-1), (i+1, j)\rangle : \langle (i,j), (i+1,j)\rangle \in \rightarrow_S\}) \setminus \{\langle (0,0), (0,1)\rangle, \langle (0,0), (1,0)\rangle\}.$$

A vertex $(i, j) \in V_{D_S}$ should be thought as a "task" of computing the point (i, j), but it can be thought as the point as well following our discussion earlier. For each isogeny evaluation edge $\langle (i, j), (i + 1, j)\rangle$ in S, we add an edge $\langle (i, e - i - 1), (i + 1, j)\rangle$ to D_S to explicitly specify the dependency that we need to have R'_i before we can evaluate ϕ_i. We also remove $(0, 0)$, since $(0, 0)$ is available from the start and we do not have to perform any task to produce it. The next example depicts this process.

Example 3. Consider a strategy from Example 1 as in Fig. 5(a). The first step of constructing the task dependency graph of a strategy is to add a diagonal directed edge for each isogeny evaluation edge to show the dependency described above. The result of the first step is in Fig. 5(b). The second step is to remove the point $(0, 0)$ and two edges from it. The task dependency graph D_S is shown in Fig. 5(c).

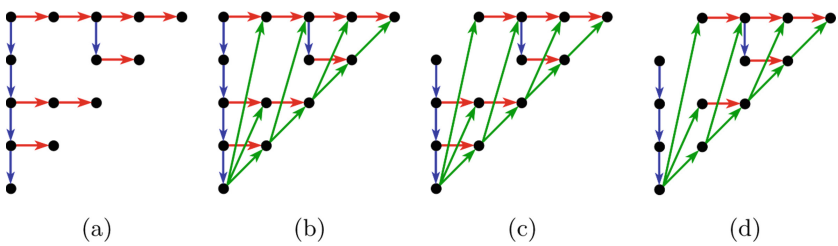

Fig. 5. Constructing the task dependency graph of a strategy.

3.2 Efficient Algorithm for Removing Transitive Edges

In the second part of the technique, we require that task dependency graphs must not have any transitive edge (Definition 6). We give an example below describing transitive edges in the task dependency graph of a strategy.

Example 4. In Fig. 5(c), the edge $\langle (0, 2), (1, 2)\rangle$ is transitive as $(0, 2)$ reaches $(0, 4)$ and $(0, 4)$ reaches $(1, 2)$. The edge $\langle (0, 3), (1, 3)\rangle$ is also transitive. These two edges are the only transitive edges in the graph. Figure 5(d) shows the graph with all transitive edges removed.

Aho, Garey, and Ullman [3] presented that, for a general directed graph, the task of removing all transitive edges from a graph, called *transitive reduction*, can be done in $O(|V|^{\log_2 7})$ steps. For D_S, it can be done in a more efficient way using the following lemma.

Lemma 2. *All transitive edges in a graph D_S must be of the form $\langle(i,j),(i+1,j)\rangle$. Also, the edge $\langle(i,j),(i+1,j)\rangle$ is transitive if and only if (i,j) reaches $(i,e-i-1)$.*

Proof. For an edge $\langle u,v\rangle$ to be transitive in a directed acyclic graph, the out-degree of u and the in-degree of v must be more than 1. Therefore, all point multiplication edges of the form $\langle(i,j),(i,j+1)\rangle$ cannot be transitive.

Next, consider a diagonal edge of the form $\langle(i,e-i-1),(i+1,j)\rangle$. If there exists another diagonal edge coming out of $(i,e-i-1)$, its end point must be $(i+1,j')$ with $j'\neq j$. If $j'>j$, it is impossible that $(i+1,j')$ reaches $(i+1,j)$. If $j'<j$, $(i+1,j')$ can reach $(i+1,j)$ by going through a sequence of point multiplication edges. However, $(i+1,j)$ is the end point of the diagonal edge implies that it is the end point of the isogeny evaluation edge $\langle(i,j),(i+1,j)\rangle$. Thus, there is no point multiplication edges coming to $(i+1,j)$. By both cases, all diagonal edges cannot be transitive.

By Definition 7, for an isogeny evaluation edge of the form $\langle(i,j),(i+1,j)\rangle$, there must exist the diagonal edge $\langle(i,e-i-1),(i+1,j)\rangle$. These are only incoming edges to $(i+1,j)$. Therefore, if this isogeny evaluation edge is transitive, (i,j) must reach $(i,e-i-1)$. This concludes the proof.

In order to remove all transitive edges from D_S, Lemma 2 suggests that we can only go through all isogeny evaluation edges once and remove $\langle(i,j),(i+1,j)\rangle$ if (i,j) reaches $(i,e-i-1)$. Verifying that there is a path from (i,j) to $(i,e-i-1)$ can be simply done by checking if all edges $\langle(i,j),(i,j+1)\rangle,\langle(i,j+1),(i,j+2)\rangle,\ldots,\langle(i,e-i-2),(i,e-i-1)\rangle$ exist, since both points are in the same column. When implemented as in Algorithm 5, the transitive reduction of D_S can be performed in $O(|V|)$ steps since each vertex (i,j) is visited at most once.

Algorithm 5: Transitive reduction algorithm for D_S.

Input : The task dependency graph $D_S=(V_{D_S},E_{D_S})$ of a strategy S
Output: D_S with all transitive edges removed

```
for i = 0 to e − 2 do
    for j = e − i − 2 down to 0 do
        if ⟨(i, j), (i, j + 1)⟩ ∉ E_{D_S} then break
        if ⟨(i, j), (i + 1, j)⟩ ∈ E_{D_S} then E_{D_S} ← E_{D_S} \ {⟨(i, j), (i + 1, j)⟩}
return D_S
```

3.3 Proposed Strategy Evaluation Technique

After we construct the task dependency graph from a strategy and remove all transitive edges, precedence-constrained scheduling algorithms (Hu's and Coffman-Graham's algorithms) described in Subsect. 2.5 can be applied to obtain a scheduling. Although both algorithms assume that all tasks are of unit-length when scheduling, which is not the case for SIDH since $c_{\downarrow} \neq c_{\rightarrow}$, they can be used as an approximation algorithm in our settings. And even though our task dependency graphs D_S are not tree-like, we get interesting results when evaluating (or scheduling) a strategy using Hu's algorithm, where in our technique $\ell(u)$ is the length of a longest path starting at u. We describe our experiments in Sect. 5.

Because both scheduling algorithms are designed for unit-length tasks, we calculate the cost of a strategy evaluation from a scheduling as shown in Algorithm 6: for each $1 \leq i \leq t(\mathcal{S})$, if all tasks in s_i are point multiplications, the cost of s_i is $c_{\downarrow}$. If all tasks in s_i are isogeny evaluations, its cost is $c_{\rightarrow}$. Otherwise, its cost is $\max\{c_{\downarrow}, c_{\rightarrow}\}$. The costs of a strategy S when using Hu's and Coffman-Graham's algorithms with K processors are denoted by $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$, respectively.

Algorithm 6: Computing $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$ of a strategy S.

Input : A strategy $S = (V_S, E_S)$ for computing degree-ℓ^e isogeny and the number of provided processors K
Output: The cost $C_K^{\mathrm{Hu}}(S)$ or $C_K^{\mathrm{CG}}(S)$

Construct D_S from S following Definition 7
Remove all transitive edges from D_S following Algorithm 5
Label all vertices with $\ell(\cdot)$ or $\ell_{\mathrm{CG}}(\cdot)$ (Algorithm 4)
Construct a scheduling $\mathcal{S}$ from (D_S, K) using Algorithm 3
$cost \leftarrow 0$
for $k = 1$ **to** $t(\mathcal{S})$ **do**
    $cost_k \leftarrow 0$
    for $(i, j) \in s_k$ **do**
        if $\langle (i, j-1), (i, j) \rangle \in E_S$ **then** $cost_k \leftarrow \max\{cost_k, c_{\downarrow}\}$
        else $cost_k \leftarrow \max\{cost_k, c_{\rightarrow}\}$
    $cost \leftarrow cost + cost_k$
return $cost$

Example 5. We explain how $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$ are computed for the strategy shown in Fig. 5(a). First, its D_S with all transitive edges removed is as Fig. 5(d). Next, vertices in D_S are labeled. The values of $\ell(\cdot)$ and $\ell_{\mathrm{CG}}(\cdot)$ are provided in Figs. 6(a) and 6(b), respectively. Let $K = 2$, both Hu's and Coffman-Graham's algorithms give the same scheduling $\mathcal{S} = \langle s_1, \ldots, s_9 \rangle$ where $s_1 = \{(0,1)\}$, $\ldots$, $s_4 = \{(0,4)\}$, $s_5 = \{(1,3),(1,0)\}$, $s_6 = \{(1,2),(2,0)\}$,

$s_7 = \{(2,2),(2,1)\}$, $s_8 = \{(3,1),(3,0)\}$, and $s_9 = \{(4,0)\}$. In s_5, $(1,3)$ and $(1,0)$ are computed by isogeny evaluations, thus $cost_5 = c_{\rightarrow}$. In s_7, $(2,2)$ is computed by isogeny evaluation and $(2,1)$ is computed by point multiplication, hence $cost_7 = \max\{c_{\downarrow}, c_{\rightarrow}\}$. The cost $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$ is thus $4c_{\downarrow}+4c_{\rightarrow}+\max\{c_{\downarrow}, c_{\rightarrow}\}$. The evaluation when $c_{\downarrow} = c_{\rightarrow} = 1$ is shown in Fig. 6(c).

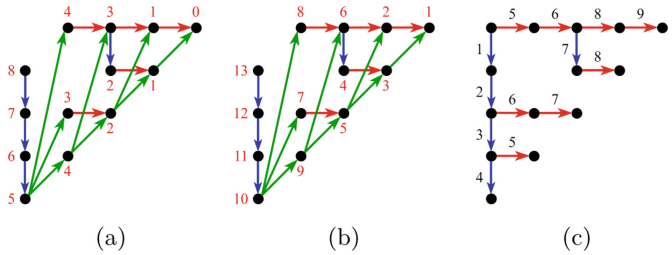

Fig. 6. The process of computing $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$ of a strategy S: (a) the values of $\ell(\cdot)$, (b) the values of $\ell_{\mathrm{CG}}(\cdot)$, and (c) the evaluation when $K = 2$ and $c_{\downarrow} = c_{\rightarrow} = 1$.

4 Proposed Strategy Construction Technique

In addition to an evaluation technique that gives us a low cost from a strategy, we also need efficient strategies that would provide low costs. As discussed earlier, a strategy for the multi-processor setting have to be carefully constructed specifically for the parameter set $(e, c_{\downarrow}, c_{\rightarrow}, K)$. To construct those efficient strategies, we first formalize the problem mathematically as an integer linear program (ILP) and then use optimal solutions of the ILP to generate strategies.

4.1 Optimal Strategies and Evaluations

The problem of constructing a strategy and its evaluation is clearly an optimization problem. We call a pair of a strategy and its evaluation that provides the least cost as *optimal*. In this subsection, we will construct an optimal strategy and evaluation in the simplest case of $c_{\downarrow} = c_{\rightarrow} = 1$, which can be generalized to the case that $c_{\downarrow} = c_{\rightarrow}$.

Let $x_{i,j,t} \in \{0,1\}$ be a decision variable such that $x_{i,j,t} = 1$ if the point represented by the vertex (i,j) is computed and is available no later than time t and 0 otherwise. A discrete optimization problem of finding an optimal strategy

and its evaluation can be formalized as an ILP as follows:

$$
\begin{aligned}
\underset{x_{i,j,t}}{\text{minimize}} \quad & T + 1 - \sum_{t'=0}^{T} x_{e-1,0,t'} \\
\text{subject to} \quad & x_{0,0,0} = 1 \\
& x_{i,j,0} = 0 \qquad\qquad (i,j) \neq (0,0) \\
& x_{i,j,t} \geq x_{i,j,t-1} \\
& x_{i,j,t} \leq x_{i,j-1,t-c_{\downarrow}} + \frac{x_{i-1,j,t-c_{\rightarrow}} + x_{i-1,e-i,t-c_{\rightarrow}}}{2} \\
& \sum_{i,j} (x_{i,j,t+1} - x_{i,j,t}) \leq K \\
& x_{i,j,t} \in \{0,1\}
\end{aligned}
$$

The initial conditions for $x_{i,j,0}$ are $x_{0,0,0} = 1$, since it is available at the start of the isogeny computation, and $x_{i,j,0} = 0$ for $(i,j) \neq (0,0)$. If (i,j) is available no later than time $t-1$, then it is also available no later than time t. Hence, we have the constraint $x_{i,j,t} \geq x_{i,j,t-1}$. Our objective is thus to minimize t' such that $x_{e-1,0,t'} = 1$, the time that $(e-1,0)$ is finished. However, we cannot straightforwardly use this as an objective function because t' is not a decision variable. We instead consider the sum of $x_{e-1,0,t'}$ for $0 \leq t' \leq T$ for some sufficiently large T. The earliest time t' at which $x_{e-1,0,t'} = 1$ can now be expressed by $T + 1 - \sum_{0 \leq t' \leq T} x_{e-1,0,t'}$, which is our objective function.

The fourth constraint comes from two restrictions of the isogeny computation discussed in Subsect. 2.4: $x_{i,j,t}$ can become 1 by one of these two cases: (i) $(i,j-1)$ is ready at time $t - c_{\downarrow}$ and (i,j) is computed by a point multiplication, or (ii) $(i-1,j)$ and $(i-1,e-i)$ are available at time $t - c_{\rightarrow}$ and (i,j) is computed by an isogeny evaluation. The first case is possible only if $x_{i,j-1,t-c_{\downarrow}} = 1$. For the second case, both $x_{i-1,j,t-c_{\rightarrow}}$ and $x_{i-1,e-i,t-c_{\rightarrow}}$ must be 1. Hence, we can perform the second case only if $\frac{1}{2}(x_{i-1,j,t-c_{\rightarrow}} + x_{i-1,e-i,t-c_{\rightarrow}}) = 1$. Because $x_{i,j,t}$ can become 1 by either of the two cases, the value of $x_{i,j,t}$ is restricted to

$$x_{i,j,t} \leq x_{i,j-1,t-c_{\downarrow}} + \frac{x_{i-1,j,t-c_{\rightarrow}} + x_{i-1,e-i,t-c_{\rightarrow}}}{2}.$$

The fifth constraint is the number of processors given. Since we are interested in the case that $c_{\downarrow} = c_{\rightarrow} = 1$, there can be up to K decision variables that change from 0 to 1 at each time, those represent points computed at that time. Therefore, we have $\sum_{i,j}(x_{i,j,t+1} - x_{i,j,t}) \leq K$.

Given the integer linear program of the problem, we can use a solver to find an optimal strategy and its evaluation. However, even in the case of small $e < 15$, the solver can take more than 30 min to produce a solution. This is expected due to the nature of integer linear programming which is NP-complete [19]. Although it is not practical to construct optimal strategies and evaluations for large e directly using ILP, we will use solutions for small e to construct a low-cost strategy for large e in the next subsection.

4.2 Proposed Strategy Construction Technique

In Subsect. 2.4, we state Theorem 1 from [18] for computing the cost of a least-cost (canonical) strategy under PCP. The theorem implicitly describes how this least-cost canonical strategy is constructed: a strategy with e leaves is divided into two smaller strategies with n and $e-n$ leaves, and the construction performs recursively until the base case $e = 1$ is reached. With the ILP we obtain in the previous subsection, we propose a new way of constructing a strategy which is by precomputing optimal strategies and evaluations for some e and then using them as base cases. We need to slightly modify the ILP in order to find optimal strategies and evaluations corresponding to $C_K^{\mathrm{PCP}}(e, k)$, but the main idea is the same. Strategies constructed by our proposed technique can then be viewed as a mixture of a canonical part when e is larger than the base case and a possibly non-canonical part when e is one of the base cases.

Similar to the proposed strategy evaluation technique in the previous section, we assume that $c_{\downarrow} = c_{\rightarrow}$ when we formulate the ILP, which is not the case for SIDH. Also, we only solve the ILP for up to some value of e and combine them for large e. Hence, strategies resulted from our construction technique are considered as approximations of a least-cost strategy.

5 Experiments and Results

For each parameter set $(e, c_{\downarrow}, c_{\rightarrow}, K)$, we conduct two experiments using our proposed strategy evaluation (Sect. 3) and construction (Sect. 4) techniques as follows:

- Experiment A: We use Theorem 1 to construct least-cost canonical strategies under PCP. Since there are many such strategies, we randomly sampled 100,000 of them for evaluation. The cost of strategy S is then computed as $\min\{C_K^{\mathrm{Hu}}(S), C_K^{\mathrm{CG}}(S)\}$.
- Experiment B: We randomly constructed 100,000 strategies using our proposed strategy construction technique described in Sect. 4, where we precomputed solutions for ILP for all $e \leq 14$. The cost of strategy S is also computed as $\min\{C_K^{\mathrm{Hu}}(S), C_K^{\mathrm{CG}}(S)\}$.

We conduct experiments under two sets of parameters from [21], which are also used by [18], for the purpose of comparison. Table 2 compares costs obtained by [18] and our experiments under the parameter set $(e, c_{\downarrow}, c_{\rightarrow}) = (186, 25.8, 22.8)$. Rows 3 and 5 show the smallest $\min\{C_K^{\mathrm{Hu}}(S), C_K^{\mathrm{CG}}(S)\}$ among all randomly sampled strategies in Experiment A and B, respectively. Table 3 reports the results under the parameter set $(e, c_{\downarrow}, c_{\rightarrow}) = (239, 27.8, 17)$. The cost reductions in both tables are compared to the costs under CCP.

The experimental results show the reductions of more than 10% in several cases, which is significant due to the fact that CCP has already improved the cost of PCP and the single-processor setting. Our strategy construction technique (Experiment B) works very well when $c_{\downarrow} \approx c_{\rightarrow}$ as seen in Table 2. We expect greater reductions when we precompute solutions of ILP for more values of e.

Table 2. The cost of best strategies under PCP, CCP, and in our experiments under the parameter set $(e, c_{\downarrow}, c_{\rightarrow}) = (186, 25.8, 22.8)$. The cost $C_1(e)$ for $K = 1$ is 34256.4.

	K	2	3	4	5	6	7	8
PCP	Cost	25942.2	22521.6	20373.0	19197.0	17941.2	16978.8	16617.0
CCP	Cost	23890.2	20515.2	18252.6	17555.4	16482.0	16021.2	15294.6
Exp.	Cost	22203.0	18622.8	16337.4	15708.6	15091.2	14949.6	14063.4
A	% reduction	7.06	9.22	**10.49**	10.52	8.44	**6.69**	**8.05**
Exp.	Cost	22081.2	18340.2	16400.4	15269.4	14973.6	14999.4	14184.0
B	% reduction	**7.57**	**10.60**	10.15	**13.02**	**9.15**	6.38	7.26

Table 3. The cost of best strategies under PCP, CCP, and in our experiments under the parameter set $(e, c_{\downarrow}, c_{\rightarrow}) = (239, 27.8, 17)$. The cost $C_1(e)$ for $K = 1$ is 41653.8.

	K	2	3	4	7	8
PCP	Cost	31886.0	27858.0	25328.8	21572.6	20851.2
CCP	Cost	29931.0	25835.0	23390.8	20399.6	19814.2
Exp.	Cost	28265.0	23625.0	21282.8	19073.6	18641.2
A	% reduction	**5.57**	**8.55**	**9.01**	**6.50**	**5.92**
Exp.	Cost	28574.6	23731.0	21337.8	19319.0	18900.4
B	% reduction	4.53	8.14	8.78	5.30	4.61

In addition, we point out that $C_K^{\mathrm{Hu}}(S)$ and $C_K^{\mathrm{CG}}(S)$ of the same strategy S are equal for all (canonical) strategies sampled in Experiment A, but these costs can be slightly different for some (possibly non-canonical) strategies sampled in Experiment B. When both costs are not equal, $C_K^{\mathrm{Hu}}(S)$ are smaller for some K and strategies, while $C_K^{\mathrm{CG}}(S)$ are smaller for some others. This shows that none of the algorithms provides the least cost for strategy evaluation.

6 Conclusion

We have studied the problem of constructing a strategy for computing degree-ℓ^e isogeny and evaluating it to achieve the least cost possible in the multi-processor setting. The proposed strategy evaluation technique transforms a strategy into a task dependency graph, where we apply precedence-constrained scheduling algorithms to it. Moreover, we have proposed a strategy construction technique which utilizes solutions of ILP for small e. Via experimental results, we have been able to obtain costs that are lower than those under PCP and CCP [18], which already improve the cost of an optimal strategy under the single-processor setting [13]. The improvements can get up to 13.02% under some specific parameter sets.

Although our results outperform those that currently exist in the literature, we note that the proposed strategy evaluation and construction techniques are yet to produce optimal strategy and evaluation. This is because there are several

layers of approximation in our techniques: (i) the ILP is formulated only for $c_{\downarrow} = c_{\rightarrow}$, (ii) we combine optimal solutions of small e to have strategies for large e, and (iii) the scheduling algorithms also assume $c_{\downarrow} = c_{\rightarrow}$. One may need to remove these approximation layers to further reduce the cost.

It is also interesting to apply our techniques to other isogeny-based cryptosystems such as B-SIDH [11], CSIDH [7], and eSIDH [8]. We expect our techniques to be applicable to all schemes but with different degrees of reduction in costs: the techniques may work well with eSIDH as the isogeny degree is quite smooth, but might not work well with B-SIDH and CSIDH as the isogeny degrees are less smooth and they involve several primes with different costs of point multiplications and isogeny evaluations. Moreover, we are yet to implement or benchmark our techniques in hardware or software. These are parts of our future work.

Acknowledgement. The authors would like to thank Jason LeGrow and the reviewers for their constructive comments on improving the manuscript. The first author would like to thank Francisco Rodríguez-Henríquez and Kittiphop Phalakarn for their valuable feedback. The first author is supported by the Ripple Impact Fund through a Ripple Graduate Fellowship. The second author is supported by JSPS Grant-in-Aid for Transformative Research Areas A grant number JP21H05845.

References

1. NIST PQC standardization process: third round candidate announcement. https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement. Accessed 2 Feb 2022
2. NIST PQC standardization project. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization. Accessed 2 Feb 2022
3. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. **1**(2), 131–137 (1972)
4. Alagic, G., et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST (2020)
5. Azarderakhsh, R., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project, vol. 152, pp. 154–155 (2017)
6. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. Open Book Ser. **4**(1), 39–55 (2020)
7. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
8. Cervantes-Vázquez, D., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: Extended supersingular isogeny Diffie-Hellman key exchange protocol: revenge of the SIDH. IET Inf. Secur. **15**, 364–374 (2021)
9. Cervantes-Vazquez, D., Ochoa-Jimenez, E., Rodriguez-Henriquez, F.: Parallel strategies for SIDH: towards computing SIDH twice as fast. IEEE Trans. Comput. **71**, 1249–1260 (2021)
10. Coffman, E.G., Graham, R.L.: Optimal scheduling for two-processor systems. Acta Informatica **1**(3), 200–213 (1972)

11. Costello, C.: B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 440–463. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_15
12. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
13. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**(3), 209–247 (2014)
14. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. IEEE Trans. Comput. **67**(11), 1622–1636 (2017)
15. Gangal, D., Ranade, A.: Precedence constrained scheduling in $(2 - \frac{7}{3p+1})$-optimal. J. Comput. Syst. Sci. **74**(7), 1139–1146 (2008)
16. Garey, M.R., Johnson, D.S.: Computers and Intractability, vol. 174. Freeman San Francisco (1979)
17. Hu, T.C.: Parallel sequencing and assembly line problems. Oper. Res. **9**(6), 841–848 (1961)
18. Hutchinson, A., Karabina, K.: Constructing canonical strategies for parallel implementation of isogeny based cryptography. In: Chakraborty, D., Iwata, T. (eds.) INDOCRYPT 2018. LNCS, vol. 11356, pp. 169–189. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05378-9_10
19. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
20. Koziel, B., Ackie, A.B., Khatib, R.E., Azarderakhsh, R., Kermani, M.M.: SIKE'd up: fast hardware architectures for supersingular isogeny key encapsulation. IEEE Trans. Circ. Syst. I Regul. Pap. **67**(12), 4842–4854 (2020). https://doi.org/10.1109/TCSI.2020.2992747
21. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In: Dunkelman, O., Sanadhya, S.K. (eds.) INDOCRYPT 2016. LNCS, vol. 10095, pp. 191–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49890-4_11
22. Koziel, B., Jalali, A., Azarderakhsh, R., Jao, D., Mozaffari-Kermani, M.: NEON-SIDH: efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 88–103. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_6
23. Lam, S., Sethi, R.: Worst case analysis of two scheduling algorithms. SIAM J. Comput. **6**(3), 518–536 (1977)
24. Levey, E., Rothvoss, T.: A $(1 + \epsilon)$-approximation for makespan scheduling with precedence constraints using LP hierarchies. SIAM J. Comput. **50**(3), 201–217 (2019)
25. Seo, H., Liu, Z., Longa, P., Hu, Z.: SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange. IACR Trans. Crypto. Hardware Embed. Syst. **2018**, 1–20 (2018)
26. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
27. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris Sér. AB **273**, A238–A241 (1971)