阅读前请了解

Kaspa 共识算法《Phantom GHOSTDAG》缺乏简易的中文译本，对于许多想要研究 Kaspa 相关技术的学者造成了不小麻烦，特尝试翻译。

翻译工作借助 Google Translate、ChatGPT 等工具，根据中文语境调整得来，可能与原文有一定出入。

受限于本人英语水平以及相关专业知识储备，翻译、解释、语境存在大量歧义、错误，相关格式、排版也不能让人满意，请谅解。

欢迎提出任意形式的建议、批评、指正。

阅读说明

翻译文章分为两栏：左边是原文，右边是对应的翻译以及个人注释

其中注释部分的颜色有如下含义：

绿色部分表示想法来源有依据，可信度较高。

蓝色部分表示对原文的一些分析和补充，可信度中等。

红色部分表示对于文章晦涩部分没有充分理解并给出通俗的翻译或者找到明确的、广知的资料，抑或是单纯吐槽。

反馈

如果您在阅读过程中发现了任何错误，抑或是想要进行学术交流，您可以通过以下方式联系我：

翻译论文极其耗费精力和时间，如果您觉得本翻译对您有帮助，请考虑捐赠任意数量的Kas。

kaspa:qqqn774xzqkkuv4gs07dj5m9mesgt286nnmn9ys52pc2jf2xraft70mawq58p

PHANTOM GHOSTDAG

A Scalable Generalization of Nakamoto Consensus

幽灵协议

中本聪共识的可扩展推广

Abstract

In 2008 Satoshi Nakamoto invented the basis for blockchain-based distributed ledgers. The core concept of this system is an open and anonymous network of nodes, or miners, which together maintain a public ledger of transactions.

The ledger takes the form of a chain of blocks, the blockchain, where each block is a batch of new transactions collected from users.

One primary problem with Satoshi's blockchain is its highly limited scalability.

The security of Satoshi's longest chain rule, more generally known as the Bitcoin protocol, requires that all honest nodes be aware of each other's blocks very soon after the block's creation. To this end, the throughput of the system is artificially suppressed so that each block fully propagates before the next one is created, and that very few "orphan blocks" that fork the chain be created spontaneously.

摘要

2008 年，中本聪完成了基于区块链的分布式账本的基础算法工作。该系统的核心概念是一个开放和匿名的节点或矿工网络，这些节点或网络共同维护着一个公共交易账本。

分布式账本：一种由多个节点组成的网络系统，用于记录和验证数字交易。在一个分布式账本系统中，每个节点都有一个完整的账本副本，并且可以通过网络与其他节点交换和同步数据。

当有新的交易发生时，所有节点都会进行验证并将其添加到账本中，因此所有节点都具有相同的最新账本版本，而且数据的一致性得到了保障。

交易采用区块链的形式，区块链中的每个区块都是由用户生成的新交易。

中本聪区块链的一个主要问题是其可扩展性非常有限。

中本聪最长链原则（通常称为比特币协议）的安全性要求所有诚实节点在区块创建后立即广播到相邻的区块。为此，系统的吞吐量被人为地约束，以便每个块在创建下一个块之前能够完全传播到其他块，但是不可避免地创建具有很少的分叉链的“孤儿块”。

最长链：节点仅接受处于最长链上的交易块，这也称为比特币最长链原则。

孤儿块：指不是最长区块链的一部分，虽然它是有效的。造成这种情况的原因一般是两个矿机同时开出了一个有效块，然后同时广播造成的。矿工无法从孤儿块中得到收益。

In this paper we present PHANTOM, a proof-of-work based protocol for a permissionless ledger that generalizes Nakamoto's blockchain to a direct acyclic graph of blocks (block DAG).

在本文中，我们介绍 PHANTOM，这是一种基于工作量证明（POW）的无许可分类账协议，它将中本聪的区块链概括为块的有向无环图(block DAG)。

无许可分类账协议：可简单理解为公链账本

PHANTOM includes a parameter k that controls the level of tolerance of the protocol to blocks that were created concurrently, which can be set to accommodate higher throughput. It thus avoids the security-scalability tradeoff which Satoshi's protocol suffers from.

PHANTOM 包含一个参数 k，它控制协议对同时创建的块数量的上限，可以通过设置 k 的大小以适应更高的吞吐量。因此，它解决了了中本聪协议所遭受的安全性与可扩展性的权衡问题。

PHANTOM solves an optimization problem over the blockDAG to distinguish between blocks mined properly by honest nodes and those created by non-cooperating nodes who chose to deviate from the mining protocol.

PHANTOM 解决了 blockDAG 的优化问题，以区分由 诚实节点正确开采 的区块和由 选择偏离挖掘协议 的非合作节点创建的区块。

诚实节点正确开采：矿工遵循 POW 协议进行挖矿。

选择偏离挖掘协议：泛指那些攻击者，即未遵循 POW 协议的参与者。

Using this distinction, PHANTOM provides a robust total order on the blockDAG in a way that is eventually agreed upon by all honest nodes.

使用该策略，PHANTOM 通过将所有诚实节点达成共识的方式，实现了在 blockDAG 上完成一个稳健的总交易。

Implementing PHANTOM requires solving an NP-hard problem, and to avoid this prohibitive computation, we devised an efficient greedy algorithm GHOSTDAG that captures the essence of PHANTOM.

实施 PHANTOM 需要解决一个 NP-hard 问题，为了避免这种令人望而却步的计算，我们设计了一种高效的贪婪算法 GHOSTDAG，它代表了 PHANTOM 的本质。

NP 难问题（NP-hard 问题）：指在多项式时间内无法求解的问题。

由于 NP 难问题的求解时间增长速度非常快，随着问题规模的增加，求解所需的时间呈指数级别增长，因此通常被认为是不可行的。

此处的 NP 难是指在一个 DAG 中找出一个 K 完全子图，或者说，是指分团问题。

The GHOSTDAG protocol has been implemented as the underlying technology of the Kaspa cryptocurrency.

加密货币 Kaspa 已将 GHOSTDAG 协议作为底层技术实现。

The Kaspa network allows us to produce statistics about the performance of GHOSTDAG in real world scenarios.

在现实世界场景中，Kaspa 网络使我们能够生成基于 GHOSTDAG 算法的统计数据的性能表现。

We provide an analysis of confirmation times obtained by observing the Kaspa network.

We discuss the properties of GHOSTDAG and how it compares to other DAG based protocols.

We provide a formal proof of the security of GHOSTDAG, namely, that its ordering of blocks is irreversible up to an exponentially negligible factor.

通过观察 Kaspa 网络，我们获得了交易确认时间的分析结果。

我们将讨论 GHOSTDAG 的特性以及它与其他基于 DAG 的协议的比较。

我们将提供 GHOSTDAG 安全性的正式证明，即它的块排序在一定误差内是不可逆的，而误差是指数级可忽略的。

1 INTRODUCTION

The security of the Bitcoin protocol relies on blocks propagating quickly to all miners in the network.

Block creation itself is slowed down via the requirement that each block contain a proof-of-work.

For the Bitcoin protocol to be secure, block propagation must be faster than the typical time it takes the network together to create the next block.

In order to guarantee this property, the creation of blocks in Bitcoin is regulated by the protocol to occur only once every 10 minutes, and the block size itself is limited to allow for fast transmission.

As a result, Bitcoin suffers from a highly restrictive throughput on the order of 3-7 transactions per second (tps)

The PHANTOM protocol. In this paper we present PHANTOM, a protocol that generalizes Nakamoto's longest chain protocol.

1 简介

比特币协议的安全性依赖于将区块快速传播给网络中的所有矿工。

由于要求每个块都包含工作量证明，因此块创建会变慢。

为了维持比特币协议的安全性，只有当上一个区块成功传播后，网络才能创建下一个区块。

为了保证这一特性，协议规定比特币的区块创建为每 10 分钟一次，并且为了支持快速传输，区块本身的大小也受到了限制。

因此，比特币的吞吐量受到极大限制，约为每秒 3-7 笔交易（tps）。

PHANTOM 协议 在本文中，我们介绍了 PHANTOM，这是一种概括了中本聪最长链协议的协议。

In Bitcoin, blocks reference a single predecessor in the chain, hence form a tree; in contrast, PHANTOM blocks reference multiple predecessors, thus forming a Directed Acyclic Graph, a blockDAG.

在比特币中，区块引用链中的单个前继，因此形成一棵树；相比之下，PHANTOM 块引用多个前继，从而形成有向无环图，即 blockDAG。

Each block can thus include several hash references to predecessors. PHANTOM then provides a total ordering over all blocks and transactions, and outputs a consistent set of accepted transactions.

因此，每个块都可以包含几个对前继的哈希加密引用。然后 PHANTOM 将所有区块和交易的进行排序，并输出一组一致的可信赖交易。

哈希加密引用:针对一组数据进行 hash 加密，这种加密在虚拟货币中很常见。

排序：根据时间对交易块进行排序，输出一组关于时间线性的交易数据。

Unlike the Bitcoin protocol, where blocks that are not on the main chain are discarded, PHANTOM incorporates all blocks in the blockDAG into the ledger, but places blocks that were created by attackers later in the order.

比特币协议会把所有不在主链上的块丢弃，与 BTC 协议不同的是，PHANTOM 会将 blockDAG 中的所有块合并到交易块中，但将被攻击者后来创建的区块排在更后面的顺序。

In rough terms, PHANTOM consists of a three-step procedure:

粗略而言，PHANTOM 可以表示为以下三个步骤：

(1) Using the structure of the blockDAG, we recognize a set of well-connected blocks (we later refer to these as blue blocks); this procedure is used to exclude blocks created by misbehaving nodes and is the heart of the protocol: Blocks that either reference only old blocks from the DAG, or are withheld by their creator for some time, will be excluded from the set of blue blocks with high probability.

(1). 使用 blockDAG 的结构特性，我们定位一组连接良好的块（稍后我们将它们称为蓝色块）；此过程用于排除由行为不当的节点创建的块。

该协议的核心思想：对于拥有以下特征的块将会有很高的概率被踢出蓝色块集合：

a) 仅引用 DAG 中的旧块；

b) 被其创建者保留一段时间的块

(2) We complete the DAG's naturally induced partial order to a full topological one (i.e., an order which respects the topology) in a way that favours blocks inside the selected cluster and penalizes those outside it.

(2). 针对 DAG 的自然诱导偏序,我们将其调整为完整拓扑排序（即尊重拓扑的顺序），这种做法有利于所选集群内的块，同时可以惩罚集群外的块。

自然诱导偏序：在块的创建过程中，新块必须在前继的后面，这本身就构成了一种顺序，即自然偏序。

拓扑排序：将一个有向无环图进行排序进而得到一个有序的线性序列

(3) The order over blocks induces an order over transactions; transactions in the same block are ordered according to the order of their appearance in the block.

(3). 有序的区块引发有序的交易；针对同一个区块中的交易，我们按照它们在区块中出现的先后顺序进行排序。

We iterate over all transactions in this order, and accept each one that is consistent (according to the underlying consistency notion) with all transactions approved so far.

我们按此顺序遍历所有交易，对于每一交易，如果能与之前已接受的所有交易保持一致性（就是底层一致性概念），我们便接受它。

Propagation delay. The first step above involves assuming an upper bound on the network's delay diameter D, and parameterizing the protocol accordingly; k denotes this parameter.

传播延迟 第一步，我们假设网络延迟上限为 D，并相应地参数化协议；我们用 k 来表示这个参数。

Such an assumption is made in Nakamoto Consensus as well. In fact, if PHANTOM is set to process low throughput, we can set $k = 0$, in which case PHANTOM coincides with Nakamoto Consensus.

中本聪共识中也有这样的假设。事实上，如果想把 PHANTOM 设置为处理低吞吐量，我们可以设置$k = 0$，此时 PHANTOM 与中本聪共识不谋而合。

However, while Nakamoto Consensus suppresses the throughput and sets the block creation rate λ such that $D \cdot \lambda \ll 1$, PHANTOM does not impose an a priori constraint over λ.

然而，虽然 Nakamoto 共识抑制了吞吐量并设置块创建速度为 λ 使得$D \cdot \lambda \ll 1$，但 PHANTOM 没有对 λ 施加先验约束。

Instead, the throughput (in terms of λ and the block size) can be set to approach the network's capacity, and then k can be set after the fact to ensure the safety of the protocol.

相反，可以将吞吐量（根据 λ 和块大小）设置为接近网络的容量，然后可以在事后设置 k 以确保协议的安全性。

This alleviates the security-scalability tradeoff that Nakamoto Consensus suffers.

这减轻了中本聪共识所遭受的安全性与可扩展性的权衡。

Still, increasing k does not come without cost, as we will discuss shortly.

尽管如此，增加 k 并非没有代价，我们将在稍后讨论。

简单地说，BTC 为了保证安全性，必须约束块的创建速度，但 PHANTOM 没有约束块的创建速度，而是根据网络容量自定义最大创建速度，而后，安全性由参数 k 控制，参数 k 是根据条件给出的。

GHOSTDAG. In its vanilla form, PHANTOM requires solving an NP-hard problem, and is therefore unsuitable for practical applications.

幽灵协议 PHANTOM 的数学实现形式需要解决 NP-hard 问题，因此不适合实际应用。

Instead, we use the intuition behind PHANTOM to devise a greedy algorithm, GHOSTDAG, which can be implemented efficiently.

相反，我们根据 PHANTOM 背后的直觉来设计贪婪算法 GHOSTDAG，它可以高效地实现我们的目标。

We prove formally that GHOSTDAG is secure, in the sense that its ordering of blocks becomes exponentially difficult to reverse as time develops.

我们将正式证明 GHOSTDAG 是安全的，它的块顺序随着时间的推移变得难以逆转。

The main achievement of GHOSTDAG can be summarized as follows:

GHOSTDAG 的主要成果可以概括为如下内容：

Theorem (Informal). Given two transactions tx_1, tx_2 that were published and embedded in the blockDAG at some point in time, the probability that GHOSTDAG's order between tx_1 and tx_2 changes over time decreases exponentially as time grows, even under a high block creation rate that is non-negligible relative to the network's propagation delay, assuming that a majority of the computational power is held by honest nodes.

定理（非正式） 假设大部分计算能力由诚实节点持有，对于在某个时间点给定的，并在 block DAG 上发布两个交易tx_1，tx_2，即使在显著的高块创建率的情况下，面对网络的传播延迟，对于交易tx_1，tx_2之间的顺序，GHOSTDAG 算法能做到让顺序变化的概率随着时间的线性增长呈现指数级下降。

简单来说，GHOSTDAG 能够在可接受时间内完成块的收敛工作，块的顺序被固定，即它们已经被主网确认了。

We will reformalize this theorem in Section 3, and provide a formal proof in Appendix A.

我们将在第 3 节中重新形式化这个定理，在附录 A 中，我们给出了正式证明。

We now proceed to describe the PHANTOM and GHOSTDAG protocols more formally.

我们现在开始更正式地描述 PHANTOM 和 GHOSTDAG 协议。

2 THE PHANTOM PROTOCOL

2.1 Preliminaries

The following terminology is used extensively throughout this paper. A DAG of blocks is denoted $G = (C, E)$, where C represents blocks and E represents the hash references to previous blocks.

We frequently write $B \in G$ instead of $B \in C$. $past(B, G) \subset C$ denotes the subset of blocks reachable from B (excluding B), and similarly $future(B, G) \subset C$ denotes the subset of blocks from which B is reachable (excluding B); these are blocks that were provably created before and after B, correspondingly.

An edge in the DAG points back in time, from the new block to previously created blocks which it extends. We denote by $anticone(B, G)$ the set of blocks outside $past(B, G)$ and $future(B, G)$ (excluding B itself); this is the set of blocks in the DAG which did not reference B (directly or indirectly via their predecessors) and were not referenced by B (directly or indirectly via B's predecessors).

Finally, $tips(G)$ is the set of blocks with in-degree 0 (usually, the most recent blocks).

This terminology is demonstrated in Figure 1.

2 PHANTOM 协议

2.1 前言

以下术语将在本文中广泛使用。块的有向无环图（下面统称为 block DAG）称为$G = (C, E)$，其中，C 表示块，E 表示对先前块的哈希引用。

我们经常使用$B \in G$代替$B \in C$。$past(B, G) \subset C$表示从 B（不包括 B）可到达的块的集合，类似地，$future(B, G) \subset C$表示从它可到达 B 的块子集（不包括 B）;显然，它们分别是在 B 之前和之后创建的块。

$past(B, G)$：一个集合，指 B 从箭头出发可以达到的块的集合。

$future(B, G)$：一个集合，如果一个块能通过箭头方向到达 B，那么它就属于$future(B, G)$。

DAG 中的边指向过去，从新块指向它扩展的先前块。我们用$anticone(B, G)$表示 G 中除$past(B, G)$和$future(B, G)$（不包括 B 本身）之外的块集合；这是 DAG 中未引用 B（直接或间接通过其前继）并且 B 的前继未引用的块集合

边：有向无环图的边，通俗地说，是一个方向。

最后，$tips(G)$是入度为 0 的块集合（通常是最新的块）。

该术语在图 1 中演示。

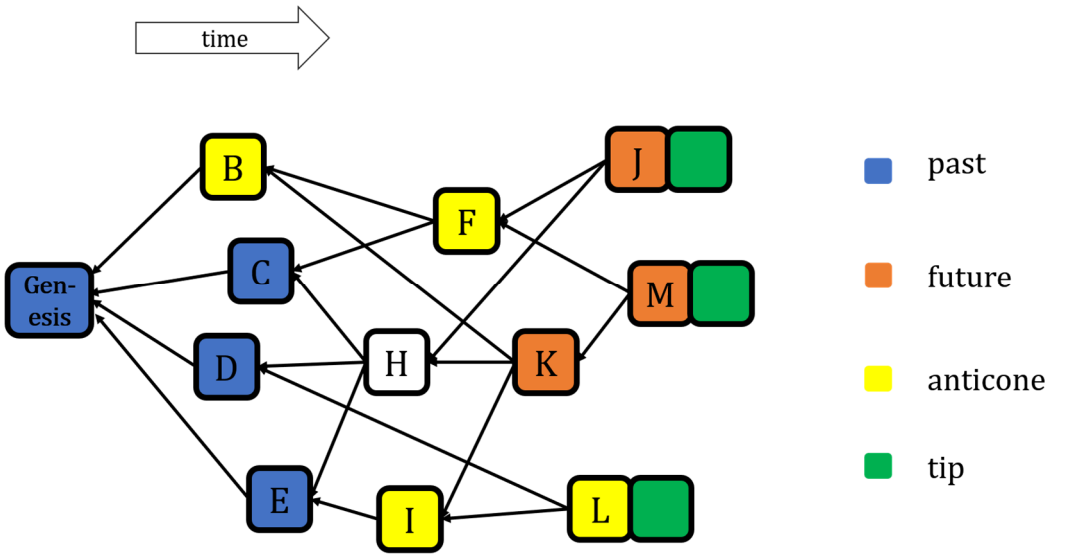


Figure 1: An example of a block DAG G. Each block references all blocks that were known to its miner at the time it was created. The DAG terminology applies to H as follows:

$past(H) = \{Genesis, C, D, E\}$ –blocks which H references directly or indirectly, and which were provably created before H;

$future(H) = \{J, K, M\}$ –blocks which reference H directly or indirectly, and which were provably created after H;

$anticone(H) = \{B, F, I, L\}$ –the order between these blocks and H is ambiguous. Reaching consensus on the order between blocks and other blocks in their anticone is the main challenge that we face.

$tips(G) = \{J, L, M\}$– leaf-blocks, namely, blocks with in-degree 0; these will be referenced in the header of the next block.

图 1：一个块 DAG G 的示例。每个块引用其矿工在其创建时所知道的所有块。我们以节点 H 为例子，如下所示：

$past(H) = \{Genesis, C, D, E\}$——H 直接或间接引用且在 H 之前创建的块；

图 1 中蓝色块。

$future(H) = \{J, K, M\}$——直接或间接引用 H 且在 H 之后创建的块；

图 1 中棕色块。

$anticone(H) = \{B, F, I, L\}$——这些块与 H 之间的顺序是不确定的。

让 H 与$anticone(H)$达成共识是我们面临的主要挑战。

图 1 中黄色块。

$tips(G) = \{J, L, M\}$——叶块，即入度为 0 的块；这些块将作为候选者，新块将选取它们中的某一个作为引用。

图 1 中绿色块。

入度：简单说，有几个箭头指向该块，这个块的入度就为几。

例如，节点 H 的入度为 2，节点 K 的入度为 1.

2.2 The DAG mining protocol

Rather than extending a single chain, a miner in PHANTOM references in its new block all blocks in tips(G), where G is the DAG that the miner observes locally at the time when the new block is created.

Additionally, the miner should broadcast its new block as fast as possible. These two rules together constitute the DAG mining protocol in PHANTOM.

2.3 The DAG ordering protocol

The aforementioned DAG mining protocol implies in particular that even when two blocks contain conflicting transactions, both blocks are incorporated into the blockDAG and referenced by all (honest) miners.

The core challenge is then how to recover the consistency of the blockDAG.

This is done in our framework by ordering all blocks-and by extension, all transactions – and accepting transactions one by one, eliminating individual transactions that are inconsistent with those approved before them.

PHANTOM achieves consensus on the order of blocks, and this guarantees agreement on the set of accepted transactions as well.

2.2 DAG 挖矿协议

在 PHANTOM 中，矿工不再是延伸单一的链，而是在新块中引用 tips(G)中的所有块，其中 G 是矿工在创建新块时，本地观察到的 DAG。

所谓“本地观察”是指矿工自己所连接的块和其他节点所广播的块，这些块可能不同于其他节点观察到的块。

此外，矿工应尽快广播其新块。这两个规则共同构成了 PHANTOM 中的 DAG 挖矿协议。

规则：

- *新块引用 tips(G)中的块；*
- *新块创建后尽快广播；*

2.3 DAG 排序协议

上述的 DAG 挖矿协议意味着即使两个块包含冲突的交易，所有的矿工（诚实的）都会将两个块都纳入 block DAG 中并引用它们。

冲突：指两个块中包含了相同的交易，但是这些交易在两个块中被分别放置在了不同的位置，也就是交易在两个块中的顺序不同。

因此，如何恢复 block DAG 的一致性是核心任务。

我们的框架通过对所有块（以及所有交易）进行排序，并逐个接受交易，消除与之前批准的交易不一致的单个交易，从而完成此操作。

在对 block DAG 进行排序的过程中，算法会逐步验证交易是否正确，这将确保所有节点的交易顺序是一致的，从而避免了交易的双重花费和其他共识问题。

在 PHANTOM 协议中，所有矿工都需要遵守相同的规则来创建和连接块，这样最终会达成全网的共识。

Essentially, Bitcoin can be seen as an ordering protocol as well, according to which transactions embedded in the longest chain of blocks precede those off the longest chain.

Unfortunately, Bitcoin's protocol is known to be secure only under slow block rates (see Section 4).

The ordering rule of PHANTOM has two stages: First, we divide the blocks to Blues and Reds; the Blue set represents blocks that appear to have been mined by cooperating nodes, whereas blocks in the Red set are outliers that were most likely mined by malicious or strategic nodes.

Then, we order the DAG in a way that favours blue blocks and penalizes red ones.

The latter step is rather immediate, and the novelty of PHANTOM lies mainly in the first colouring procedure.

2.3.1 *The intuition behind PHANTOM.* Just like Bitcoin, PHANTOM relies on the ability of honest nodes to communicate to their peers recent blocks in a timely manner, and on the assumption that honest nodes possess more than 50% of the hashrate.

The block rate in Bitcoin is suppressed so as to ensure block creation is slower than the time it takes to communicate them.

In PHANTOM, on the other hand, we notice that the set of honest blocks can be recognized even when the block rate is high and many forks appear spontaneously: Due to the communication and cooperation of honest miners, we should expect to see in the DAG a "well-connected" cluster of blocks.

从本质上讲，比特币也可以看作是一种排序协议，根据这种协议，位于最长的区块链中的交易块比那些不在最长链上的交易块优先级更高。

不幸的是，比特币的协议只在慢速块率下被认为是安全的（见第 4 节）。

PHANTOM 的排序规则有两个阶段：首先，我们将块分为蓝色和红色；

蓝色集合表示由诚实节点挖掘的块，而红色集合中的块是由恶意或不怀好意的节点挖掘的异常块。

然后，我们以倾向于蓝色块并惩罚红色块的方式对 DAG 进行排序。

后者是相当直接的，PHANTOM 的创新主要在于第一个着色过程。

两阶段，染色和排序，PHANTOM 的核心在于染色。

2.3.1 *PHANTOM 背后的直觉。*就像比特币一样，在诚实节点占有超过 50%的算力的前提下，PHANTOM 依赖于诚实节点及时向其他块广播最新块的能力。

比特币的块创建速率被抑制，以确保块的创建速度比通信所需的时间慢。

另一方面，在 PHANTOM 中，我们注意到一个现象，即使块创建速率很高并且出现许多分叉，诚实块的集合也可以被协议识别出来：由于诚实矿工的通信和合作，我们应该可以在 DAG 中找到一个"良好连接"的块集群。

Indeed, let D be an upper bound on the network's propagation delay. If block B was mined by an honest miner at time t, then any block published before time $t-D$ necessarily arrived at its miner before time t, and is therefore included in $past(B)$.

Similarly, the honest miner will publish B immediately, and so B will be included in the past set of any block mined after time $t+D$.

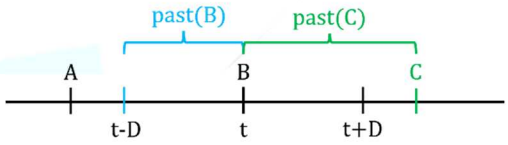
As a result, the set of honest blocks in B's anticone is typically small, and consists only of blocks created in the interval $[t-D, t+D]$.

事实上，假设 D 是网络传播延迟的上界。如果在时间 t，诚实的矿工挖掘出了块 B，则在时间$t-D$之前发布的任何块必然在时间 t 之前到达该矿工，因此被包含在$past(B)$中。

这些块都会被该矿工包含在它挖掘的块 B 的过去集合$past(B)$中。

同样，诚实的矿工将立即发布 B，因此 B 将包含在那些在时间$t+D$之后挖掘的任何块的过去集合中。

因此，B 的 anticone 中的诚实块集通常很小，并且仅由在区间$[t-D, t+D]$中创建的块组成。



如图所示，A 在 $t-D$之前创建，则 $A \subseteq past(B)$，C 在 $t+D$之后创建，则$B \subseteq past(C)$。

The proof-of-work mechanism guarantees that the number of blocks created in an interval of length $2 \cdot D$ is typically below some $k > 0$.

工作量证明机制保证，在长度为$2 \cdot D$的时间间隔内创建的块的数量通常低于某个$k > 0$。

In short, the set of blocks created by honest nodes is well-connected. The following definition captures "well-connectedness":

简而言之，由诚实节点创建的块集合是"良好连接"的。

Definition 1. Given a $DAG\ G = (C, E)$, a subset $S \subseteq C$ is called a k-cluster, if $\forall B \in S: |anticone(B) \cap S| \leq k$.

定义 1 给定一个$DAG\ G = (C, E)$，我们把子集$S \subseteq C$称为一个 k-cluster，当且仅当$\forall B \in S: |anticone(B) \cap S| \leq k$。

$|A|$表示集合 A 中的元素个数。

Attacker nodes may deviate arbitrarily from the mining rules, have large anticones, and even artificially increase the anticone of honest blocks.

攻击节点可能会随意偏离挖矿规则，因此具有较大的 anticones，甚至可以人为地增加诚实块的 anticones。

Nonetheless, since honest miners possess more proof-of-work power, it is usually impossible for malicious miners to create a well-connected set of blocks that is larger than that created by honest nodes.

尽管如此，由于诚实矿工拥有更多的工作证明能力，恶意矿工通常无法创建比诚实节点创建的更大的良好连接块集合。

依赖于 51%算力是诚实的假设前提下，恶意矿工无法颠覆 DAG 网络。

PHANTOM utilizes this fact and selects the largest well-connected set within the DAG, by solving the following optimization problem:

Maximum k-cluster SubDAG (MCS_k)
Input: DAG $G = (C, E)$
Output: A subset $S^* \subset C$ of maximum size, $s.t. \lvert anticone(B) \cap S^* \rvert \leq k$ for all $B \in S^*$.

In this formulation, the parameter k is predetermined; see Section 4 for more details. An example of a maximum k-cluster appears in Figure 2.

2.3.2 *The PHANTOM protocol.* Following the above intuition, the ordering protocol of PHANTOM comprises the following two steps:

(1) Given a block G, solve $MCS_k(G)$; let's refer to its output as the Blue set and to its complement set as the Red one.

(2) Determine the order between Blue blocks according to some topological sort. Then, for any Blue block B, add to the order just before B all of the Red blocks in $past(B)$ that weren't added to the order yet; these Red blocks too should be added in a topological manner.

PHANTOM 利用了这一事实，并通过解决以下优化问题来选择 DAG 中最大的良好连接集合：

Maximum k-cluster SubDAG (MCS_k)
Input: DAG $G = (C, E)$
Output: A subset $S^* \subset C$ of maximum size, $s.t. \lvert anticone(B) \cap S^* \rvert \leq k$ for all $B \in S^*$.

现在我们可以对 k-cluster 下定义了：DAG 中最大的良好连接集合。当然，受参数 k 约束。

在这个公式中，参数 k 是预先确定的；更多细节请参见第 4 节。最大 k-cluster 的示例显示在图 2 中。

2.3.2 *PHANTOM 协议*。根据上述直觉，PHANTOM 的排序协议包括以下两个步骤：

给定一个块 G，解决$MCS_k(G)$问题；我们称其输出为蓝集合，输出的补集为红集合。

$MCS_k(G)$输出是一个 k-cluster。

此处的补集应该是指绝对补集。

假设 U 是全集，则 A 在 U 中的补集A^C（或U/A）定义为：$\{x \mid x \in U, x \notin A\}$。

根据某种拓扑排序确定蓝块之间的顺序。

然后，对于任何蓝块 B，在其前面按拓扑顺序添加所有还未添加到顺序中的$past(B)$中的红块。

上面这句话特别绕。

将蓝块按照某种拓扑排序排列，然后对于每一个蓝块 B，我们执行如下操作：

a) 找到 $past(B)$ 中的所有红块（称为 X）；

b) 找到 X 中所有未添加到排序的红块（称为 Y）；

c) 按照拓扑排序的方式，将 Y 中的块依次添加到 B 的前面。

这里的“前面”是指添加后的区块应该位于 $past(B)$ 中。

An example of the output of the PHANTOM procedure on the small blockDAG from Figure 2 is: $(A, D, C, G, B, F, I, E, J, H, K)$.

Unfortunately, the Maximum k-cluster SubDAG problem is NP hard, and PHANTOM is therefore of less practical use for an ever-growing blockDAG.

We thus introduce a greedy algorithm that is more suitable for implementation. We call this greedy variant GHOSTDAG.

PHANTOM 协议在图 2 的小型 block DAG 上的输出示例为：$(A, D, C, G, B, F, I, E, J, H, K)$。

不幸的是，最大 k-cluster Sub DAG 问题是 NP 难问题，因此对于不断增长的 block DAG，PHANTOM 的实用性较差。

因此，我们引入了一种更适合实现的贪心算法，称为 GHOSTDAG。

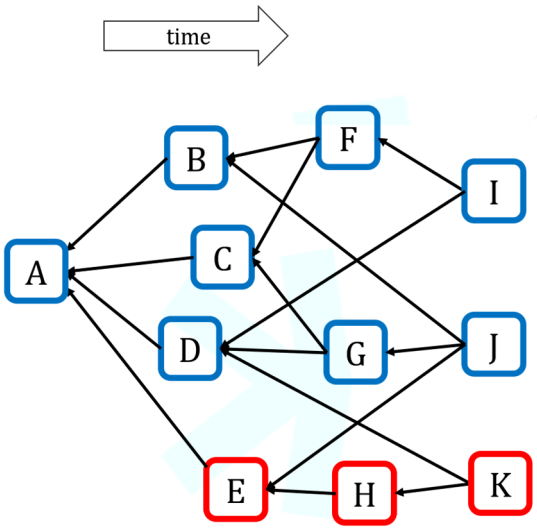


Figure 2: An example of the largest 3-cluster of blocks within a given DAG: A, B, C, D, F, G, I, J (coloured blue).

It is easy to verify that each of these blue blocks has at most 3 blue blocks in its anticone, and (a bit less easy) that this is the largest set with this property.

Setting PHANTOM's inter-connectivity parameter with $k = 3$ means that at most 4 blocks are assumed to be created within each unit of delay, so that typical anticone sizes should not exceed 3.

Blocks outside the largest 3-cluster, E, H, K (coloured red), belong to the attacker (w.h.p.).

For instance, block E has 6 blue blocks in its anticone (B, C, D, F, G, I);

图 2：给定 DAG 中最大的 3-cluster 的示例：A, B, C, D, F, G, I, J（以蓝色标记）。

很容易验证的是，在这些蓝色块中，每一个块的 anticone 集至多有 3 个蓝色块，并且这是具有此属性的最大集合（这并不容易验证）。

将 PHANTOM 的互连参数设置为$k = 3$意味着在每个延迟单位内最多假定创建 4 个块，因此一般来说 anticone 的大小不应超过 3。

最大的 3-cluster 之外的块，E, H, K（以红色标记），属于攻击者（w.h.p.）。

例如，块 E 在其 anticone 中有 6 个蓝色块（B, C, D, F, G, I）；这些块没有引用 E，

these blocks didn't reference E, presumably because E was withheld from their miners.

Similarly, block K admits 6 blue blocks in its anticone (B, C, G, F, I, J); presumably, its malicious miner received already some of these blocks but violated the mining protocol by not referencing them.

这可能是因为 E 被它们的矿工拒绝了。

类似地，块 K 在其 anticone 中有 6 个蓝色块（B, C, G, F, I, J）；这可能是由于 K 的恶意矿工已经收到了其中一些块，但由于违反了挖矿协议，没有引用它们。

2.4 The GHOSTDAG protocol

Similar to PHANTOM, the GHOSTDAG protocol selects a k-cluster, which induces a colouring of the blocks as Blues (blocks in the selected cluster) and Reds (blocks outside the cluster).

However, instead of searching for the largest k-cluster, GHOSTDAG finds a k-cluster using a greedy algorithm.

The algorithm constructs the Blue set of the DAG by first inheriting the Blue set of the best tip B_{max}, i.e., the tip with the largest Blue set in its past, and then adds to the Blue set blocks outside B_{max}'s past in a way that preserves the k-cluster property.

2.4 GHOSTDAG 协议

类似于 PHANTOM，GHOSTDAG 协议也会选择一个 k-cluster，该 cluster 引导块的颜色为蓝色（选择的 cluster 内的块）和红色（cluster 外的块）。

这一点与 2.3.2 中的定义是类似的。

然而，GHOSTDAG 不同于 PHANTOM 的是，它会使用一种贪心算法来寻找 k-cluster。

该算法通过首先继承最佳 tip（称为 B_{max}）的蓝色块集合来构建 DAG 的蓝色块集合，所谓最佳 tip 是指在所有 tip 节点中，过去集合蓝色块数最多的 tip，随后，在保持 k-cluster 属性的前提下，将B_{max}过去之外的块添加到蓝色块集合中。

该算法流程：

1) *找到当前 DAG 中的所有 tip 节点；*
2) *对所有 tip 节点，我们关注$past(tip)$，每一个$past(tip)$都有一个蓝色块集合。*
3) *找到其中最大的蓝色块集合，我们称其对应的 tip 为B_{max}；*
4) *关注$past(B_{max})$之外的块(即红色块)；*
5) *在维持 k-cluster 属性的前提下，将这些红色块加入到蓝色块集合中。*

所谓维持 k-cluster 属性就是保证加入红色块后，每个块的 anticone 集与蓝色块集合的交集大小不大于 k。

Observe that this greedy inheritance rule induces a chain: The last block of the chain is the selected tip of G, B_{max}; the next block in the chain is the selected tip of the DAG $past(B_{max})$; and so on down to the $genesis$.

We denote this chain by $Chn(G) = (genesis = Chn_0(G), Chn_1(G), \ldots, Chn_h(G))$.

The final order over all blocks, in GHOSTDAG, follows a similar path as the colouring procedure: We order the blockDAG by first inheriting the order of B_{max} on blocks in past (B_{max}), then adding B_{max} itself to the order, and finally adding blocks outside past (B_{max}) according to some topological ordering.

Thus, essentially, the order over blocks becomes robust as the colouring procedure.

注意到这种贪心继承规则会产生一个链：链的最后一个块是 G 选中的 tip，即B_{max}；链中的下一个块是$past(B_{max})$选中的 tip；这种递归关系一直到$genesis$。

我们将此链表示为 $Chn(G) = (genesis = Chn_0(G), Chn_1(G), \ldots, Chn_h(G))$。

在 GHOSTDAG 中，对所有块的最终排序遵循类似于着色过程的方法：首先继承B_{max}在$past(B_{max})$中的块的顺序，然后将B_{max}本身添加到顺序中，最后根据某种拓扑排序方法添加$past(B_{max})$之外的块。

通过参考着色的流程，我们可以总结出 GHOSTDAG 的思想。

从 $past(B_{max})$ 出发，算法会继承 $past(B_{max})$ 的拓扑顺序，然后把B_{max}加入该排序列表的最后。随后，通过一种拓扑排序，我们遍历$past(B_{max})$之外的块（按照染色来说就是红色块），将这些块逐步添加到排序列表中，在此过程中我们需要保证每个块要在其父块之后被访问。

因此，基本上，块的顺序在着色过程中变得稳健。

Algorithm 1 Ordering the DAG

Input: G – a block DAG, k – the propagation parameter

Output: $BLUE_k(G)$ – the Blue set of G; ord – an ordered list containing all blocks in G

1. **function** $O_{RDER}DAG(G,k)$
2. **if** $G == \{genesis\}$ **then**
3. **return** $[\{genesis\},\{genesis\}]$
4. **for** $B \in tips(G)$ **do**
5. $[BlueSet_B, OrderedList_B] \leftarrow OrderDAG(past(B),k)$
6. $B_{max} \leftarrow argmax\{|BlueSet_B|: B \in tips(G)\}$ (break ties according to lowest hash)
7. $BlueSet_G \leftarrow BlueSet_{B_{max}}$
8. $OrderedList_G \leftarrow OrderedList_{B_{max}}$
9. add B_{max} to $BlueSet_G$
10. add B_{max} to the end of $OrderedList_G$
11. **for** $B \in anticone(B_{max}, G)$ **do** (in some topological ordering)
12. **if** $BlueSet_G \cup \{B\}$ is a k-cluster **then**
13. add B to $BlueSet_G$
14. add B to the end of $OrderedList_G$
15. **return** $[BlueSet_G, OrderedList_G]$

算法 1 DAG 排序

输入：G（一个 block DAG）；k（传播参数）。

输出：$BLUE_k(G)$（G 中的蓝色块集合）；ord（一个列表，关于 G 中所有区块的排序结果）

算法思想：

1. 首先检查 DAG 是否仅包含$genesis$，若是，返回包含$genesis$的蓝色集合和排序后的块列表。
2. 对于 G 中的所有 tip 块，对于每个 tip 块（称为 B），递归调用该算法来计算$past(B)$的蓝色集合和排序后的块列表。
3. 从所有 tip 块的蓝色集合中选择一个最大的蓝色集合（称为$BlueSet_{B_{max}}$），选中这个最大蓝色集合对应的 tip（称为B_{max}）。如果遇到多个最大值，按 hash 值最低的标准来选择。
4. 把B_{max}对应的蓝色集合添加到当前的 DAG 的蓝色集合$BlueSet_G$中。
5. 把B_{max}对应的排序后的块列表添加到当前 DAG 的排序块列表$OrderedList_G$中。
6. 把B_{max}添加到$BlueSet_G$中。
7. 把B_{max}添加到$OrderedList_G$的队尾。
8. 遍历B_{max}的$anticone$集（遍历对象称为 B），执行如下操作（以一种拓扑排序的方式）：
   - 如果将 B 加入集合$BlueSet_G$后是一个 k-cluster，则把 B 加入集合$BlueSet_G$中。
   - 把 B 加入$OrderedList_G$的队尾。
9. 返回$BlueSet_G$与$OrderedList_G$，即我们想要的$BLUE_k(G)$与ord。

2.4.1 Formal algorithm. The procedures described above are formalized in Algorithm 1 below.

The algorithm begins with the base case where the DAG consists of the $genesis$ block only (lines 2-3).

Next, it performs a recursive call to compute the Blue sets and ordering of the past of each of the DAG's tips (lines 4-5), and inherits those of the best tip (lines 6-8).

Then, the selected tip is added to the Blue set $BlueSet_G$ and to the last position in the current ordered list $OrderedList_G$ (lines 9-10).

Then we iterate over $anticone(B_{max}, G)$ in some topological way which guarantees that a block is visited only after its predecessors are (lines 11-14).

For every block we visit, we check if adding B to the Blue set will preserve the k-cluster property, and if this condition is satisfied, we add B to the Blue set (lines 9-13); either way, we add B to the current last position in the list (line 14).

Finally, we return the Blue set and the ordered Note that the recursion in the algorithm (line 5) halts, because for any block $B \in G: |past(B)| < |G|$.

We demonstrate the operation of this algorithm in Figure 3.

We have an efficient implementation of Algorithm 1, and we tested it under high block rates ($\lambda = 10$, 25 blocks per second, for blocks of size 0.1-1 MB).

2.4.1 正式算法。算法 1 正式描述了相关处理过程。

算法从仅包含$genesis$的初始情况开始（第 2-3 行）。

接下来，它递归调用本身以计算 DAG 的每个 tip 的过去的 Blue 集合和排序（第 4-5 行），并继承最佳 tip 的集合（第 6-8 行）。

最佳 tip：B_{max}；

最佳 tip 的集合：$BlueSet_G$。

然后，将选定的 tip 添加到 Blue 集合$BlueSet_G$和当前有序列表$OrderedList_G$的最后一个位置（第 9-10 行）。

然后，我们按某种拓扑方式遍历$anticone(B_{max}, G)$，这将保证在访问某一个块之前能够先访问其前置块（第 11-14 行）。

对于我们访问的每个块，如果把 B 添加到蓝色集合中后，新的集合是否拥有 k-cluster 的属性（蓝色集合本身是具有 k-cluster 属性的），如果满足此条件，则将 B 添加到蓝色集合中（第 9-13 行）；

无需条件，我们都将 B 添加到当前列表的队尾（第 14 行）。

最后，我们返回蓝色集合和有序列表。请注意，算法中的递归（第 5 行）会停止，这是因为对于任何块，都有$B \in G: |past(B)| < |G|$。

我们在图 3 中演示了此算法的操作。

我们实现了 **Algorithm 1** 的高效实现，并在高块创建速率下（$\lambda = 10$、每秒 25 个块，大小为 0.1-1 MB）进行了测试。

λ：每秒钟生成的区块数量。

读者可能对此有点困惑，既然指定了$\lambda = 10$，这与后文的“每秒 25 个块”是否冲突？

在这种情况下，参数λ是指每秒钟期望接收到的块数，而不是实际的块速率。

而每秒 25 个块是实际的块速率，表示在测试期间每秒生成 25 个块。因此，这两

个参数并不矛盾，它们分别表示了不同的概念。

We will make the implementation available in the full version of this paper.

我们将在本文的完整版本中完成实现。

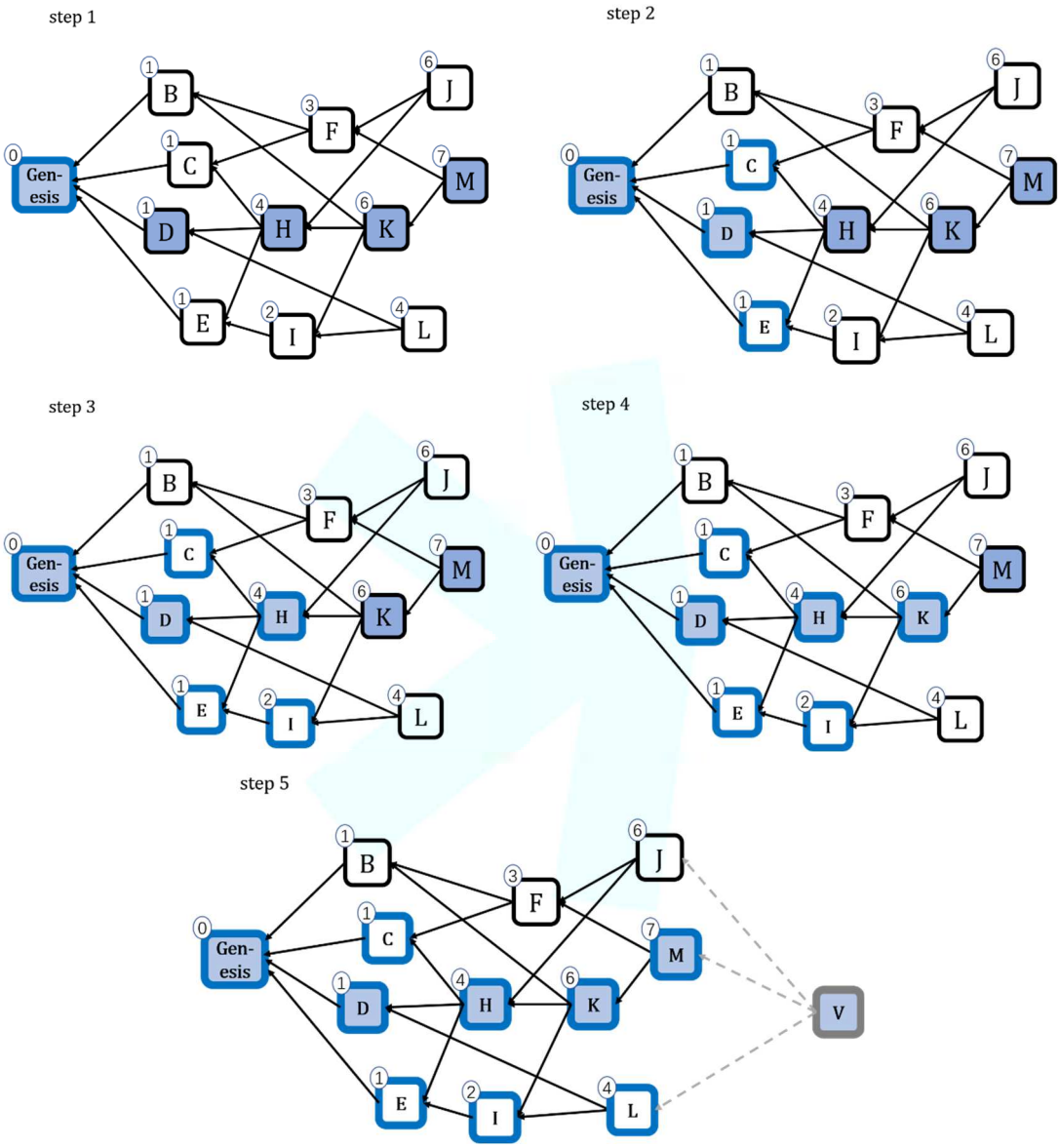


Figure 3: An example of a blockDAG G and the operation of GHOSTDAG to construct its blue set $BLUE_k(G)$ set, under the parameter $k = 3$.

The small circle near each block X represents its score, namely, the number of blue blocks in the DAG $past(X)$.

The algorithm selects the chain greedily, starting from the highest

图 3： 一个 block DAG G 的例子，展示了 GHOSTDAG 构建其蓝色集合$BLUE_k(G)$的操作，其中参数$k = 3$。

每个块 X 旁边的小圆圈代表其分数，分数就是 DAG $past(X)$中蓝块的数量。

该算法使用贪心算法选择链，从最高得分的 tip M 开始，然后选择其前驱 K

scoring tip M, then selecting its predecessor K (the highest scoring tip in $past(M)$), then H, D (breaking the C, D, E tie arbitrarily), and finally $Genesis$.

（$past(M)$中得分最高的 tip），然后是H, D（在C, D, E中脱颖而出），最后是$Genesis$。

For methodological reasons, we add to this chain a hypothetical "virtual" block V – a block whose past equals the entire current DAG.

为了更好地说明，我们在这条链中添加了一个假想的"虚拟"块V——一个过去等于当前整个 DAG 的块。

即$past(V) = DAG$。

Blocks in the chain ($genesis, D, H, K, M, V$) are marked with a light-blue shade.

链中的块（$genesis, D, H, K, M, V$）用浅蓝色阴影标记。

Using this chain, we construct the DAG's set of blue blocks, $BLUE_k(G)$.

使用这条链，我们构建了 DAG 的蓝色块集合$BLUE_k(G)$。

The set is constructed recursively, starting with an empty one, as follows: In step 1 we visit D and add $genesis$ to the blue set, as the only block in $past(D)$.

这个集合是递归构建的，从一个空集合开始，步骤如下：

第 1 步：访问D，将$genesis$添加到蓝色集合中，$genesis$是$past(D)$中唯一的块。

Next, in step 2, we visit H and add to $BLUE_k(G)$ blocks that are blue in $past(H)$, namely, C, D, E.

在第 2 步：访问H，并把$past(H)$中的蓝块（即C, D, E）添加到$BLUE_k(G)$中。

In step 3 we visit K and add H, I; note that block B is in $past(K)$ but was not added to the blue set, since it has 4 blue blocks in its anticone.

第 3 步：访问K，并添加H, I；请注意，虽然块B在$past(K)$中，但没有被添加到蓝色集合中，因为它的 anticone 中有 4 个蓝块。

我们一开始就设定了参数$k = 3$，所以不行。

In step 4 we visit M and add K to the blue set; again, note that $F \in past(M)$ could not be added to the blue set due to its large blue anticone.

第 4 步：访问M，并将K添加到蓝色集合中；同样，请注意，虽然$F \in past(M)$，但是由于它的蓝色 anticone 集很大，所以不能被添加到蓝色集合中。

Finally, in step 5, we visit the block $virtual(G) = V$, and add M and to $BLUE_k(G)$, leaving L away due to its large blue anticone, and leaving J away because adding it would cause I to suffer too large a blue anticone (it already has C, D, and H in it).

第 5 步：访问块$virtual(G) = V$，并将M添加到$BLUE_k(G)$中，由于L的蓝色 anticone 较大，L被舍弃，J也会被舍弃，这是因为如果添加了J，会导致I拥有过大的蓝色 anticone 集（而且它已经有C, D和H了）。

正在施工中

Under construction