Universiteit
Leiden
The Netherlands

Trade-offs between classical and quantum space using spooky pebbling

Quist, A.; Laarman, A.W.

Citation
Quist, A., & Laarman, A. W. (2025). Trade-offs between classical and quantum space using spooky pebbling. *Logical Methods In Computer Science*, *21*(4). doi:10.46298/lmcs-21(4:29)2025

Version:	Publisher's Version
License:	Creative Commons CC BY 4.0 license
Downloaded from:	https://hdl.handle.net/1887/4285544

Note: To cite this publication please use the final published version (if applicable).

Logical Methods in Computer Science
Volume 21, Issue 4, 2025, pp. 29:1–29:26
https://lmcs.episciences.org/

Submitted Aug. 16, 2024
Published Dec. 12, 2025

TRADE-OFFS BETWEEN CLASSICAL AND QUANTUM SPACE USING SPOOKY PEBBLING *

AREND-JAN QUIST AND ALFONS LAARMAN

Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
e-mail address: {a.quist, a.w.laarman}@liacs.leidenuniv.nl

Abstract. Pebble games are used to study space/time trade-offs. Recently, spooky pebble games were introduced to study classical space / quantum space / time trade-offs for simulation of classical circuits on quantum computers. In this paper, the spooky pebble game framework is applied for the first time to general circuits. Using this framework we prove an upper bound for quantum space in the spooky pebble game. We also prove that solving the spooky pebble game is PSPACE-complete. Moreover, we present a solver for the spooky pebble game based on satisfiability solvers combined with heuristic optimizers. This spooky pebble game solver was empirically evaluated by calculating optimal classical space / quantum space / time trade-offs. Within limited runtime, the solver could find a strategy reducing quantum space when classical space is taken into account, showing that the spooky pebble model is useful to reduce quantum space.

1. Introduction

For a long time, scientists have been thinking how specific problems could be calculated within constraints on computational resources, such as size of the memory and runtime of the calculation. To study the memory usage and runtime for a calculation, pebble games were introduced. Pebble games model the use of space and time in the run of a given circuit. Trade-offs between space and time can also be easily studied by a pebble game by studying the maximum number of pebbles and time used. Such trade-offs are a fundamental problem in Computer Science [AHM+00, Ben89, Swa78, Val76][Sav08, Chapter 10].

Pebble games. In a given circuit, a pebble game is played on its underlying directed acyclic graph (DAG) structure. Each gate within the circuit corresponds to a node in this graph, with the direct input/output relationships between gates represented by directed edges. Every node in the graph can be pebbled or unpebbled. Pebbling a node models calculating and storing the output of the corresponding gate in memory, while unpebbling models freeing memory. The primary objective of the pebble game is to pebble the entire circuit, achieved by pebbling the output gates. Analyzing trade-offs between memory usage and runtime entails considering all strategies for pebbling the circuit. The space used by a

Key words and phrases: Pebble game, Spooky pebble game, Quantum computing, Satisfiability.

* This paper is an extension of conference proceedings [QL23]. An overview of contributions with respect to the previous version are in the 'Contents and contributions' paragraph in the introduction. One should be aware of the change of terminology from "spooks" (previous version) to "ghosts" (this work).

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-21(4:29)2025
© A. Quist and A.W. Laarman
Creative Commons

strategy corresponds to the maximum number of pebbles employed at any given time, while the runtime is determined by the number of (un)pebble moves making up the strategy.

There exist variations of the pebble game to model different types of computing, see for example [Cha13, Nor15] for a (non-complete) overview. In this paper, we consider three types of pebble games: irreversible pebble games, reversible pebble games and spooky pebble games. For the *irreversible pebble game* all inputs of a node must be pebbled to pebble that node, but there are no constraints for unpebbling a node. This is a model for classical computing. For the *reversible pebble game*, introduced by Bennett [Ben89], all inputs of a node must be pebbled to either pebble or unpebble that node. This is a model for reversible simulation of irreversible circuits in general and, more specific, for simulation of irreversible circuits on a quantum computer [LTV98].

Spooky pebble game. The *spooky pebble game* is an extension of the reversible pebble game for quantum computing which was introduced recently by Gidney [Gid19] and more extensively worked out by Kornerup et al [KSS21, KSS24]. This game weakens the constraint for unpebbling a node in reversible pebbling at the cost of doing a quantum measurement. In the spooky pebble game, this is represented by a ghost, which replaces a pebble and classically stores the measurement outcome. The (classical) value of the measurement outcome is used later by the quantum computer to restore the state. Thus for the spooky pebble game we can study the trade-off between classical memory, quantum memory and time. As quantumly accessible classical space (represented by ghosts) is assumed to be much cheaper than quantum space (represented by pebbles) (see e.g. [BGB+18, PPR19, Pei20]), studying such a trade-off could be very useful for memory reduction in quantum computing.

The spooky pebble game, similar to reversible pebbling, models running an irreversible (classical) computation on a quantum computer with as input a superposition. Therefore, it is a useful model for a quantum oracle calculation of a classical function. In many famous quantum algorithms like Shor's [Sho94] and Grover's [Gro96] such oracle calculations are essential. Thus, studying the spooky pebble game can be useful for quantum memory management for near term quantum computing on machines with limited space and time.

Contents and contributions. This paper is an extended version of [QL23]. Its main contributions with respect to the previous version are

- a PSPACE-completeness proof for solving the spooky pebbling game,[1]
- optimization algorithms for simplifying spooky pebble game instances, and
- a solver implementation that combines SAT solving with heuristic optimization.

In the previous version of this paper, the spooky pebble game solver consisted of a SAT solver only. The strength of SAT solvers is being very fast at finding *a* solution for the game, which is not necessarily an *optimal* solution. The new spooky pebble game solver is an iterative combination of SAT solving and optimization of their solutions by heuristics in a way that these complementary approaches mutually reinforce each other. The extensive experimental results show improvements to the SAT-only solvers by up to 39 percent.

The remainder of this paper is organized as follows. In Section 2, we explain the theory about the spooky pebble game. Note that we are the first that study the spooky pebble game on general DAGs instead of linegraphs as in [KSS21][2]. In Section 3, we prove a theorem about the maximum memory cost for a quantum oracle computation with respect to the

[1] A similar result was also proven by [KSS24] after the first preprint of our work was published.

[2] The results on general DAGs by [KSS24] were published after the first preprint of our work was published.

equivalent classical computation when there is also classical memory available. Moreover, we show that solving the spooky pebble game is PSPACE-complete. Section 4 describes a solver of the spooky pebble game developed by us. To our knowledge this is the first solver for the spooky pebble game. In Section 5, the details of our open source solver implementation are presented. This section also describes experiments to create optimal quantum space, classical space and time trade-offs with the solver.

2. Background: spooky pebble game

In this section, we first explain measurement-based uncomputation for quantum computing, which is the basis for the spooky pebble game. Then we formally introduce (spooky) pebble games. Finally, we connect measurement-based uncomputation and spooky pebble games. Most of this section was also introduced by [Gid19, KSS21, KSS24].

Any reader that is only interested in pebble games but not familiar with quantum computing can safely skip Sections 2.1 and 2.3. The main message that such skipping reader should be aware of is that in a spooky pebble game every pebble models a qubit, every ghost models a classical bit, and every move models a (semi-)quantum operation. As qubits are expensive with respect to classical bits, we usually want to reduce the number of pebbles in a game.

2.1. Measurement-based uncomputation.

Assume we want to simulate a classical algorithm on a superposition of classical inputs on a quantum computer. In quantum computing, the only operations one can apply are reversible operations and (possibly irreversible) measurements. Therefore, to apply irreversible (classical) operations additional qubits (ancillae) are needed. A quantum algorithm can produce a lot of garbage qubits this way. These ancillae qubits cannot be trivially removed as measuring them can destroy the superposition. Therefore, reversible uncomputation is needed to clean the ancillae.

Measurement-based uncomputation is a technique to do an irreversible uncomputing operation using a measurement such that, under certain conditions, the uncollapsed state can be restored using the outcome of the measurement. This technique can be applied for quantum oracles where the quantum input remains in memory during the entire computation and no interference gates are applied in between the measurement-based uncomputation and the restoring of the state. These conditions are satisfied when we quantumly simulate a classical oracle computation, using only ((multi-)controlled) NOT gates, which we focus on in this paper.

Figure 1 depicts the idea of measurement-based uncomputation in a circuit. Assume that an irreversible function f is computed on some quantum superposition input $\sum_x \alpha_x |x\rangle$. This is done by the reversible operation U_f which writes the outcome in a new register, resulting in the state $\sum_x \alpha_x |x\rangle |f(x)\rangle$. Now, (a part of) the input is changed by an unknown but non-interfering computation, while keeping the function output in memory. If we want to remove the function output from the memory, we cannot simply uncompute the function output by applying U_f again as the input has gone. Instead the function output is measured after applying a Hadamard gate to it. On measurement outcome b, a phase $(-1)^{bf(x)}$ is added to every basis state of the superposition. After computing the semi-classical CNOT dependent on the measurement outcome, the memory of the function output is free and can be reused used as a $|0\rangle$ ancilla by other computations. If we want to correct for the phase $(-1)^{bf(x)}$, we need the input $\sum_x \alpha_x |x\rangle$ again and compute the function f. Now we

can apply a Z-gate on the function output register, dependent on the measurement outcome b. Now, the function output can be uncomputed by applying U_f again, and we are back in the state with input $\sum_x \alpha_x |x\rangle$.

Note that this trick of temporary uncomputing the function output of an irreversible function f when the inputs are not available is not possible in usual reversible computing. Hence, measurement-based uncomputation can be seen as quantum semi-reversible computation.

For a more formal and extensive description of measurement-based uncomputation, we refer to [KSS21].

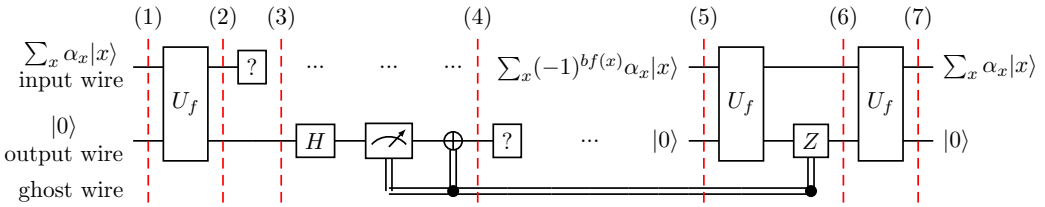


Figure 1: This circuit describes measurement-based uncomputation. The three wires depict the input and output of the function f and the ghost wire to classically store the measurement outcome. The labels (1), (2), etc correspond to the corresponding spooky pebbling configurations as shown in Figure 3.

2.2. **Pebble games.** In this section, we formally define three types of pebble games: irreversible, reversible and spooky.

We first define the irreversible and reversible pebble game. The irreversible pebble game is a natural model for classically computing a circuit. The reversible pebble game is a natural model for reversible computation on irreversible circuits, e.g., quantumly computing a classical circuit.

Definition 2.1 ((Ir)reversible pebbling)**.** Let $G = (V, E)$ be a directed acyclic graph (DAG). The set of *roots* R is defined as $R = \{v \in V \mid \textit{out-degree}(v) = 0\}$. A *(ir)reversible pebbling strategy* on G is a sequence of sets of pebbled vertices $P_0, P_1, ..., P_T \subseteq V$ such that the following conditions hold:

(1) $P_0 = \emptyset$;
(2) $P_T = R$;
(3) for every $t \in \{1, 2, ..., T\}$ there exists one $v \in V$ such that one of the following holds:
- *pebble(v):* $P_t = P_{t-1} \cup \{v\}$ and all direct predecessors (i.e. in-nodes) of v are in P_{t-1};
- *unpebble(v):*
  For *irreversible* pebbling: $P_t = P_{t-1} \backslash \{v\}$;
  For *reversible* pebbling: $P_t = P_{t-1} \backslash \{v\}$ and all direct predecessors (i.e. in-nodes) of v are in P_{t-1}.

Now, we will define the spooky pebble game, a natural model for quantumly computing a classical circuit using measurement-based uncomputation. The definition for this game is mainly adapted from [KSS21]. The spooky pebble game can be viewed as an extended generalization of the (ir)reversible pebble game. The main difference between spooky

pebbling and (ir)reversible pebbling is the addition of ghosts. To regulate the ghosts, the actions *ghost* and *unghost* are added, and for the actions *pebble* and *unpebble* the requirement that no ghost is changed is added.

Definition 2.2 (Spooky pebbling)**.** Let $G = (V, E)$ be a directed acyclic graph (DAG). The set of *roots* R is defined as $R = \{v \in V \mid \textit{out-degree}(v) = 0\}$. A *spooky pebbling strategy* on G is a sequence of pairs of pebbles and ghost pebbles $(P_0, S_0), (P_1, S_1), ..., (P_T, S_T) \subseteq V \times V$ such that the following conditions hold:

(1) $P_0 = \emptyset$ and $S_0 = \emptyset$;
(2) $P_T = R$ and $S_T = \emptyset$;
(3) for every $t \in \{1, 2, ..., T\}$ there exists one $v \in V$ such that one of the following holds:
   - *pebble(v):* $P_t = P_{t-1} \cup \{v\}$ and $S_t = S_{t-1}$ and all direct predecessors (i.e. in-nodes) of v are in P_{t-1};
   - *unpebble(v):* $P_t = P_{t-1} \backslash \{v\}$ and $S_t = S_{t-1}$ and all direct predecessors (i.e. in-nodes) of v are in P_{t-1};
   - *ghost(v):* $P_t = P_{t-1} \backslash \{v\}$ and $S_t = S_{t-1} \cup \{v\}$;
   - *unghost(v):* $P_t = P_{t-1} \cup \{v\}$ and $S_t = S_{t-1} \backslash \{v\}$ and all direct predecessors (i.e. in-nodes) of v are in P_{t-1}.[3]

We define a *spooky pebbling sub-strategy* as a spooky pebbling strategy that does not necessarily obey conditions (1) and (2). We will use spooky pebbling sub-strategies as subroutines of a spooky pebbling strategy.

For a pebbling (sub-)strategy on a DAG G, the following three quantities are useful. The *pebbling time* is the integer T. The *pebbling cost* is $\max_{t \in [T]} |P_t|$ and (for the spooky pebbling game) the *ghosting cost* is $\max_{t \in [T]} |S_t|$.

In Figure 2, we see some examples of moves for the different types of pebble games. For the configuration on the top, the move *unpebble(E)* can be applied for all irreversible, reversible and spooky pebbling. The move *unpebble(D)* can only be applied in the irreversible pebble game, as its input vertex A is not pebbled. The move *ghost(D)* can only be applied in the spooky pebble game, as the (ir)reversible pebble game does not have ghosts.

2.3. **Connection between spooky pebble game and measurement-based uncomputation.** In Figures 1 and 3, the relation between the spooky pebble game and measurement-based uncomputation is depicted. Quantum space is represented by pebbles and ghosts represent classical space. Like in the reversible pebble game, pebbling and unpebbling are just applying classical (irreversible) gates on a quantum computer, storing the gate-outputs in additional space. Placing a ghost can be interpreted as applying measurement-based uncomputation and storing the measurement outcome in memory. Unghosting can be interpreted as restoring the uncomputed state by recomputing the function.

Assume we want to simulate a classical algorithm on a superposition of inputs on a quantum computer, for example the classical circuit in Figure 4. A reversible or spooky pebbling strategy now corresponds directly to a simulation of the classical algorithm on a quantum computer. Figures 5 and 6 show two such examples of quantum circuits that simulate the classical circuit of Figure 4.

Note that for Figure 4 the pebbling cost for the reversible pebble game is at least 4, while the pebbling cost for the spooky pebble game is 3. This shows that measurement-based

[3]Note that a pebble is placed on v when v is unghosted.

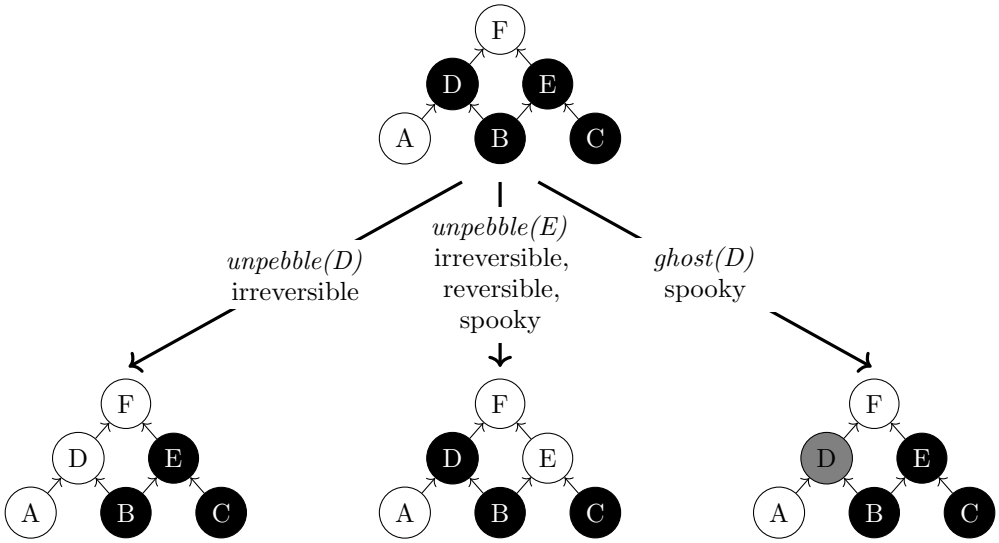


Figure 2: Three examples of steps for different types of pebble games. A white node represents an empty vertex, a black vertex is a pebbled vertex and a grey vertex is a ghosted vertex.

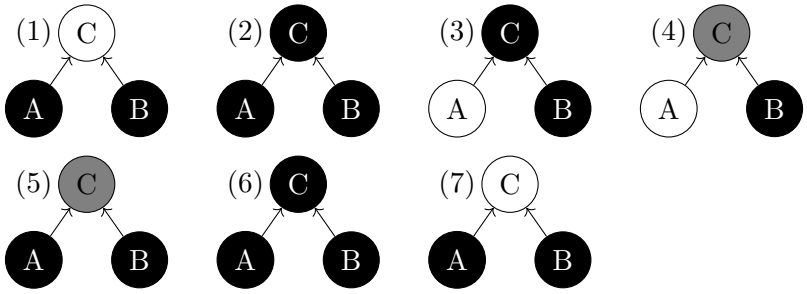


Figure 3: Spooky pebbling configurations for the labels in the circuit of Figure 1. In Definition 2.2, the spooky pebble game is introduced. The function inputs in vertices (A) and (B) are used to compute a function f and store the output in vertex (C). In other words: $f((A),(B)) = (C)$. A white vertex is unpebbled, a black one is pebbled and a grey vertex is ghosted. In this example not the entire input is changed: vertex (B) is pebbled (i.e. hold in memory) all over the computation and only vertex (A) is unpebbled (i.e. removed from memory).

uncomputation (i.e. the spooky pebble game) can reduce the number of qubits (pebbles) that is needed to simulate the circuit of Figure 4 on a quantum superposition.

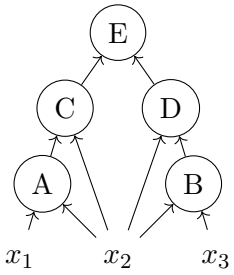


Figure 4: Classical circuit on input bits x_1, x_2, x_3. The gates A,B,C,D,E are classical gates, e.g. AND, OR.

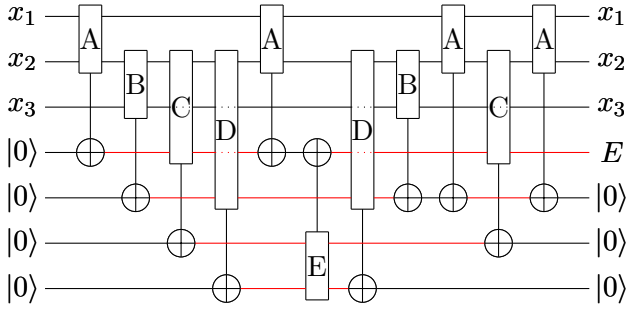


Figure 5: Quantum circuit to compute the circuit from Figure 4 on a superposition on a quantum computer without measurement-based uncomputation.
The reversible pebbling strategy that leads to this circuit consists of the following consecutive moves: pebble(A), pebble(B), pebble(C), pebble(D), unpebble(A), pebble(E), unpebble(D), unpebble(B), pebble(A), unpebble(C), unpebble(A).
This strategy has a pebbling cost of 4.

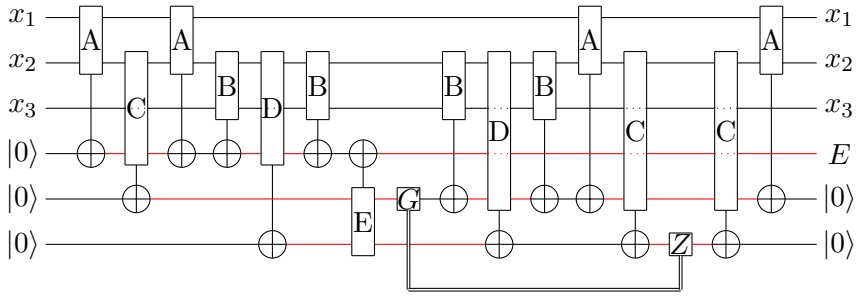


Figure 6: Quantum circuit to compute the circuit from Figure 4 on a superposition on a quantum computer using measurement-based uncomputation. Gate G is a H-gate followed by a measurement in the Z-basis and a classically-controlled correction.
The spooky pebbling strategy that leads to this circuit consists of the following consecutive moves: pebble(A), pebble(C), unpebble(A), pebble(B), pebble(D), unpebble(B), pebble(E), ghost(C), pebble(B), unpebble(D), unpebble(B), pebble(A), unghost(C), unpebble(C), unpebble(A).
This strategy has a pebbling cost of 3 and a ghosting cost of 1.

3. Spooky pebble cost equals irreversible pebble cost Spooky pebble game is PSPACE-complete

In this section, we show that deciding whether a DAG can be spooky pebbled using P pebbles is PSPACE-complete answering the open question of [KSS21]. For the irreversible [GLT79] and reversible [CLNV15] pebble game, this problem is already known to be PSPACE-complete. A reduction from the irreversible pebble game is used to show PSPACE-completeness for the spooky pebble game.

Theorem 3.1. *The decision problem for the existence of a spooky pebbling strategy for a DAG G with P pebbles is PSPACE-complete.*

Before we prove Theorem 3.1, we present results relating the pebbling cost of different types of pebble games presented in the previous section. This is a kind of intermezzo that is not needed to show Theorem 3.1, but the result follows directly from Lemma 3.4 and 3.3 which we use later to prove Theorem 3.1. We show first that if a DAG with m roots (outputs) can be irreversible pebbled with C pebbles, then it can be spooky pebbled with at most $C + m$ pebbles. This result is somewhat surprising as in the irreversible strategy we can unpebble at any time. This shows the power of ghosts with respect to the reversible pebble game where a similar result does not hold.[4] Intuitively, this result holds because the irreversible pebble game can be simulated by the spooky pebble game with additional pebbles at all roots. Gidney [Gid19] already proved a similar result for line graphs where an obvious optimal irreversible pebble strategy with $C = 2$ pebbles can be converted to a spooky pebble strategy with $C + m = 2 + 1 = 3$ pebbles, but we generalize this to arbitrary DAGs and arbitrary irreversible pebbling strategies. The statement can be formalized as follows.

Theorem 3.2. *Let $G = (V, E)$ be a DAG with m roots. Assume there exists an irreversible pebbling strategy to pebble G with irreversible pebbling cost C and pebbling time T. Then there exists a spooky pebbling strategy to pebble G with spooky pebbling cost at most $C + m$ and pebbling time $T + (T + 1)(|V| - m)$.*

Proof. Let $P_0, P_1, ..., P_T$ be an irreversible pebbling strategy with irreversible pebbling cost C. The result follows directly by combining Lemma 3.3 and 3.4, using $R' = R$. □

Lemma 3.3. *Let $P_0, P_1, ..., P_T$ be an irreversible pebbling strategy for graph $G = (V, E)$ with pebbling time T and irreversible pebbling cost C. Then there exists a spooky pebbling sub-strategy from $(P_0, S_0) = (\emptyset, Q)$ to $(P_T, S_T) = (R, V \backslash R)$ with pebbling time T and pebbling cost C, for any subset $Q \subseteq V$.*

Proof. The irreversible pebbling strategy is followed with the following modifications. If a vertex is pebbled in the irreversible game, then so in the spooky game. If a vertex is unpebbled in the irreversible game, then that pebble is ghosted in the spooky game. If a vertex has a ghost when it is pebbled, the vertex is unghosted instead of pebbled. Thus at the end of this procedure, all roots (outputs) are pebbled and all other vertices are ghosted. This takes C pebbles and T moves, because the irreversible pebbling strategy needs C pebbles and T moves. □

[4] This is easy to see, as [LTV98, Theorem 2] shows that reversibly pebbling a line graph needs at least logarithmic pebbles, but, trivially, irreversible pebbling of a line graph needs a constant number of pebbles.

Lemma 3.4. *Let $P_0, P_1, ..., P_T$ be an irreversible pebbling strategy for graph $G = (V, E)$ with pebbling time T and irreversible pebbling cost C. Then for every $R' \subseteq R$ there exists a spooky pebbling sub-strategy from $(P_0, S_0) = (R', V \backslash R')$ to $(P_K, S_K) = (R', \emptyset)$ with pebbling time $K = (T + 1)(|V| - |R'|)$ and pebbling cost $C + |R'|$.*

Proof. Let v be a vertex of G and let $G_v = (V_v, E_v) \subseteq G = (V, E)$ be the largest subgraph of G rooted at v. We can now unpebble v by projecting the irreversible strategy to G_v. For this purpose, let $P'_i = P_i \cap V_v$. Note, however, that all vertices in W are still ghosted. By Lemma 3.3 for graph G' with $S_0 = Q = V_v$ and $R = \{v\}$, v can be pebbled in pebbling time T and pebbling cost C. Hence, unpebbling of v can be done in pebbling time $T + 1$ and pebbling cost C. We remark that, after applying this spooky sub-strategy, the vertices in $V_v \backslash \{v\}$ are again ghosted and v does not contain a pebble or ghost.

The above procedure can be done for every vertex v in $V \backslash R'$. To ensure that unghosting a vertex v does not interfere with other unghostings, unghosting+unpebbling is done in the reverse order as the order we pebbled them (for the first time t) in the irreversible pebbling strategy.

As we need to remove $|V| - |R'|$ pebbles, the total pebbling time is $(T + 1)(|V| - |R'|)$. The pebbling cost is $C + |R'|$, as the $|R'|$ pebbles remain on the roots while C pebbles are used to remove the ghosts. □

Now, we show that the spooky pebble game is PSPACE-complete. To demonstrate PSPACE-hardness, we provide a reduction from QBF, a well-known PSPACE-complete problem [SM73].

First, we restate a theorem which shows a reduction from QBF to the irreversible pebble game for a graph G. This irreversible reduction is extended in Theorem 3.7 by constructing a graph G' related to G for a reduction from QBF to the spooky pebble game.

In [GLT79], for every quantified formula $Q_1 x_1 Q_2 x_2 ... Q_n x_n F$ a graph G is constructed. This graph G has the properties that it is linear sized in terms of the formula length of $Q_1 x_1 Q_2 x_2 ... Q_n x_n F$, that it has more than one edge, and that it contains a single output vertex q_1.

Theorem 3.5 [GLT79, Theorem 1]**.** *The quantified Boolean formula $Q_1 x_1 Q_2 x_2 ... Q_n x_n F$ is true if and only if the graph G with the properties as above has an irreversibly pebbling strategy with pebbling cost $P = 3n + 4$.*

Proof of Theorem 3.5. In [GLT79] another move for the pebble game is allowed called sliding. Therefore, the proof of [GLT79] needs a little addition which we provide below.

The sliding move that is allowed in [GLT79] has the following rule:
"(iii) If all predecessors of an unpebbled vertex v are pebbled, a pebble may be moved from a predecessor of v to v."

By Theorem A from [EBvL78], for every DAG G with at least one edge the pebbling cost differs exactly one between allowing and not allowing the sliding move. As G in this theorem has more than one edge and the original theorem in [GLT79] has $P = 3n + 3$, Theorem 3.5 holds for our definition of irreversible pebble games. □

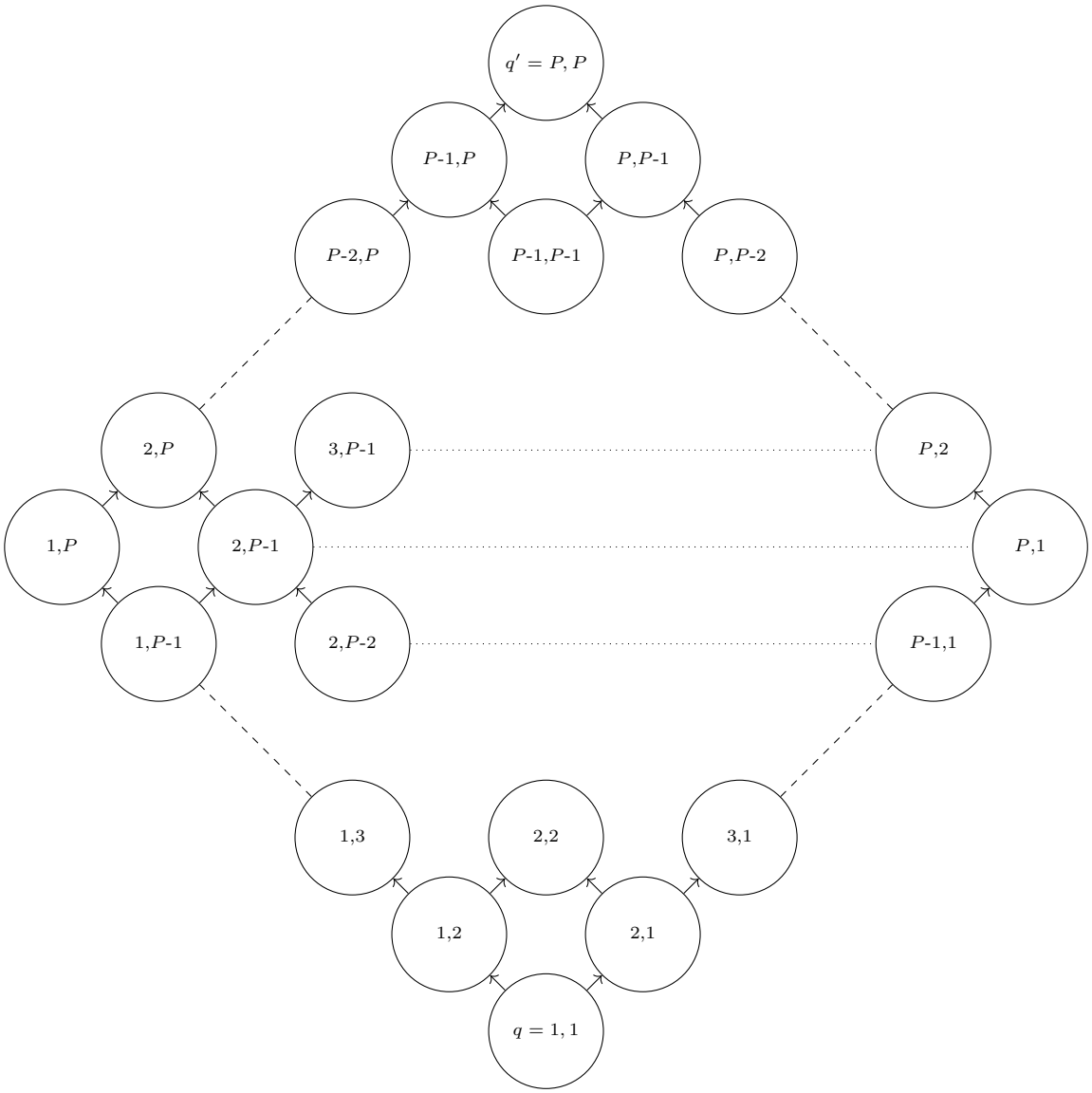
We will first show the existence of a DAG with desired properties that will be used in the proof of Theorem 3.7.

Lemma 3.6. *For every integer P there exists a DAG $G_P^\diamond$ with the following properties:*

- *$G_P^\diamond$ has one input (leaf) vertex and one output (root) vertex*

- $G_P^\diamond$ *has size polynomial in* P
- $G_P^\diamond$ *can be irreversibly pebbled with* $P+1$ *pebbles, pebbling the input vertex only once*
- $G_P^\diamond$ *cannot be irreversibly pebbled with* P *pebbles*
- *Every vertex in* $G_P^\diamond$ *except the output vertex can be irreversibly pebbled with* P *pebbles, pebbling the input vertex only once.*
- $G_P^\diamond$ *can be spooky pebbled with* $P+1$ *pebbles, and in every such spooky pebbling strategy at some point there are* $P+1$ *pebbles on non-input nodes and the input node must be pebbled at a later time*

Proof. Let $G_P^\diamond$ be the following DAG:



Now we will verify that this DAG satisfies the properties from this lemma.
It is easy to check that $G_P^\diamond$ has P^2 nodes, so its size is polynomial in P.

Moreover, $G_P^\diamond$ has a single input and output node.

$G_P^\diamond$ can be irreversibly pebbled with $P+1$ pebbles, pebbling the input $(1,1)$ only once, using the following strategy:

```
for i in 1 to P do
    pebble(i,1)
for j in 2 to P do
    for i in 1 to P do
        pebble(i,j)
        unpebble(i,j − 1)
for i in 1 to P − 1 do
    unpebble(i,P)
```

Every vertex in $G_P^\diamond \backslash \{q'\}$ can be pebbled with P pebbles using the above algorithm with i always in 1 to $P-1$ instead of in 1 to P. The nodes (P,k) with $k \in \{1,...,P-1\}$ cannot be pebbled using this algorithm, but, by symmetry of the graph, we can run the symmetric algorithm with swapped coordinates (i.e. replace pebble(x,y) and unpebble(x,y) with pebble(y,x) and unpebble(y,x)) to pebble them.

Consider an irreversible or spooky pebbling strategy to pebble $G_P^\diamond$. Consider the last time that a node on the middle line $(1,P)$,$(2,P-1)$,...,$(P,1)$ is pebbled before the output q' is pebbled. Say that $(i,P-i+1)$ is pebbled. Then without loss of generality there exists a pebble-free path r in $G_P^\diamond$ from $(i,P-i+1)$ to q', otherwise, the pebbling of $(i,P-i+1)$ could have been omitted. Moreover, after $(i,P-i+1)$ is pebbled, there should be no path without pebbles from the middle line to q'. We have that every left diagonal path
$(P,P)-(P-1,P)-...-(1,P)$;
$(P,P-1)-(P-1,P-1)-...-(1,P-1)$;
...;
$(P,P-i)-(P-1,P-i)-...-(1,P-i)$
and every right diagonal path
$(i+1,P)-(i+1,P-1)-...-(i+1,1)$;
...;
$(P,P)-(P,P-1)-...-(P,1)$
must be blocked by a pebble. This should happen below the path r, as a top part of the path r can be combined with a diagonal path into a new path, which should be blocked. Therefore, at least $P-1$ pebbles are needed to block all these $P-1$ diagonals. As also 1 pebble is needed to pebble $(i,P-i+1)$, and the inputs to $(i,P-i+1)$ should be pebbled, at least $P+1$ pebbles are needed to pebble $G_P^\diamond$.
Moreover, if $G_P^\diamond$ is spooky pebbled with $P+1$ pebbles, all $P+1$ pebbles are on non-input nodes. Moreover, there are no other pebbles left below the middle line, so unpebbling the input of $(i,P-i+1)$ is not possible without pebbling q at a later time. When ghosting the input of $(i,P-i+1)$, this ghost should be removed before ending the game, which also requires to pebble q at a later time. So the input of $G_P^\diamond$ must be pebbled at a later time to finish the game. □

Using the above theorem and lemma, we can prove the reduction from QBF to the spooky pebble game. This is formally stated in the following theorem.

Theorem 3.7. *For every Quantified Boolean Formula F there exists a DAG G' with one output and a number of pebbles P such that F is true if and only if G' can be spooky pebbled with P pebbles.*

Proof. Let F be a Quantified Boolean Formula. Take from Theorem 3.5 DAG G and number of pebbles P corresponding to F. Thus G can be irreversibly pebbled with P pebbles if F is true, and with more than P pebbles if F is false. Let $G_P^\diamond$ be the DAG from Lemma 3.6.

Now, we can construct DAG G' from G and $G_P^\diamond$. We will show that G' can be spooky pebbled with $P+1$ pebbles if F is true, and with strictly more than $P+1$ pebbles if F is false. Recall that G has one output node, which we will name q. Let the input node of $G_P^\diamond$ be q and let q' be the output node of $G_P^\diamond$. See Figure 7 for the construction of G'.

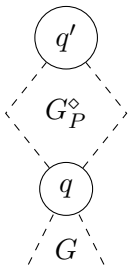


Figure 7: DAG G' which can be spooky pebbled with $P+1$ pebbles if and only if G can be pebbled with P pebbles. See the proof in Theorem 3.7.

We will show that G' can be spooky pebbled with $P+1$ pebbles if and only if G can be irreversibly pebbled with P pebbles. This shows the theorem.

($\Longleftarrow$) Assume that G can be irreversibly pebbled with P pebbles. So q can be irreversibly pebbled with P pebbles. As $G^\diamond$ can be irreversibly pebbled with $P+1$ pebbles pebbling q once, we can irreversibly pebble G' using $P+1$ pebbles. Similarly, every $G'\backslash\{q'\}$ can be irreversibly pebbled using P pebbles. By Lemma 3.3, we can pebble q' using $P+1$ pebbles, leaving ghosts at $G'\backslash\{q'\}$. As every node in $G'\backslash\{q'\}$ can be irreversibly pebbled using P pebbles, every largest subgraph G'_v of G' with single root $v \in G'\backslash\{q'\}$ can be irreversibly pebbled with P pebbles. Applying Lemma 3.4 to these graphs shows that any ghost from $G'\backslash\{q'\}$ can be removed with P pebbles.
Hence, G' can be spooky pebbled with $P+1$ pebbles.

($\Longrightarrow$) Assume G' can be spooky pebbled with $P+1$ pebbles. Recall that we assumed that $G_P^\diamond$ can be spooky pebbled with $P+1$ pebbles, and that in every such spooky pebbling strategy at some point there are $P+1$ pebbles on non-input nodes and the input node must be pebbled at a later time. Hence, when G' is spooky pebbled, at this point q must be pebbled at a later time when at least one pebble is on $G_P^\diamond\backslash\{q\}$: if all pebbles from $G_P^\diamond\backslash\{q\}$ could be removed, then the pebblings on $G_P^\diamond\backslash\{q\}$ could be omitted from the strategy. As q is the output of G, G can be spooky pebbled with P pebbles. □

Note 3.8. *It is easy to see that solving the spooky pebble game is in PSPACE using the following argument (which is similar to the one in [GLT79] for reversible pebbling): checking whether a spooky pebbling strategy using P pebbles is valid can be done in polynomial space, thus solving the spooky pebble game is in NPSPACE. By Savitch's theorem [Sav70], we know that NPSPACE = PSPACE. So solving the spooky pebble game is in PSPACE.*

Proof of Theorem 3.1. Using the fact that QBF is PSPACE-complete [SM73], Theorem 3.7 shows that the spooky pebble game is PSPACE-hard. Note 3.8 shows that solving the spooky pebble game is in PSPACE. Hence, deciding whether there exists a spooky pebbling strategy for a graph G with P pebbles is PSPACE-complete. □

4. Spooky pebble game solver

As shown in the previous section, solving the spooky pebble game is a hard problem. In this section we present a solver algorithm for the spooky pebble game that searches for strategies with a minimal cost. To this end, in Section 4.1, we encode the spooky pebble game as a satisfiability problem, similar to the method of [MSR+19] for the reversible pebble game. The output of the satisfiability solver seems usually not to be an optimal pebbling strategy in number of pebbles and ghosts used or in pebbling time. Therefore, we introduce efficient and short-running heuristics in Section 4.2, to further optimize the pebbling strategy by reducing time, the number of pebbles and the number of ghosts.

4.1. SAT solver.

To encode the spooky pebble game into SAT, we use the Boolean variables $p_{v,i}$ and $s_{v,i}$ with v the vertex and i the time. The variable $p_{v,i}$ indicates whether vertex v is pebbled at time i, and similar for ghosting with variable $s_{v,i}$. We use the shorthand $x_i = \bigcup_{v\in V} p_{v,i} \cup \bigcup_{v\in V} s_{v,i}$ to describe all variables at time i.

We use the clauses as stated below to describe the game. If there is an assignment of variables that satisfies all clauses, i.e. evaluates to true, this assignment describes a solution of the game. The clauses below are for the game with T timesteps.

Initial clauses: For V the set of vertices in DAG G we have initial clauses:

$$I(x_0) \triangleq \bigwedge_{v\in V} \neg p_{v,0} \wedge \neg s_{v,0} \tag{4.1}$$

Final clauses: For R the set of outputs (roots) in DAG G we have final clauses:

$$F(x_T) = \bigwedge_{v\in R} p_{v,T} \wedge \bigwedge_{v\notin R} \neg p_{v,T} \wedge \bigwedge_{v\in V} \neg s_{v,T} \tag{4.2}$$

Move clauses: There are the following move clauses for edgeset E of DAG G, for every time $i = 0, ..., T-1$:

To add a pebble all predecessors must be pebbled and a ghost on this vertex should be removed:[5]

$$M_1(x_i, x_{i+1}) = \bigwedge_{(v,w)\in E} [(\neg p_{v,i} \wedge p_{v,i+1}) \Longrightarrow (p_{w,i} \wedge p_{w,i+1})] \wedge \bigwedge_{v\in V} [(\neg p_{v,i} \wedge p_{v,i+1}) \Longrightarrow \neg s_{v,i+1}] \tag{4.3}$$

To remove a pebble all predecessors must be pebbled or the pebble must be changed into a ghost:

$$M_2(x_i, x_{i+1}) = \bigwedge_{(v,w)\in E} [(p_{v,i} \wedge \neg p_{v,i+1}) \Longrightarrow ((p_{w,i} \wedge p_{w,i+1}) \vee s_{v,i+1})] \tag{4.4}$$

[5]Hereby we prevent a vertex to be pebbled and ghosted at the same time.

To add a ghost the vertex must be unpebbled:

$$M_3(x_i, x_{i+1}) = \bigwedge_{v \in V} [(\neg s_{v,i} \wedge s_{v,i+1}) \implies (p_{v,i} \wedge \neg p_{v,i+1})] \tag{4.5}$$

To remove a ghost all predecessors must be pebbled and the ghost must be changed into a pebble:

$$M_4(x_i, x_{i+1}) = \bigwedge_{(v,w) \in E} [(s_{v,i} \wedge \neg s_{v,i+1}) \implies ((p_{w,i} \wedge p_{w,i+1}) \wedge p_{v,i+1})] \tag{4.6}$$

Cardinality clauses: For $P \in \mathbb{N}$ the maximal number of pebbles and $S \in \mathbb{N}$ the maximal number of ghosts we have for every $i = 0, 1, ..., T$ the clauses:

$$C_{P,S}(x_i) = \left(\sum_{v \in V} p_{v,i} \leq P \right) \wedge \left(\sum_{v \in V} s_{v,i} \leq S \right) \tag{4.7}$$

Using these clauses, we actually encode our problem as in Bounded Model Checking format (BMC) [BHv21, Chapter 18]. Using this formula the problem can be easily defined for arbitrary times T. The following BMC formula is used:

$$I(x_0) \wedge C_{P,S}(x_0) \wedge M(x_0, x_1) \wedge C_{P,S}(x_1) \wedge ... \wedge M(x_{T-1}, x_T) \wedge C_{P,S}(x_T) \wedge F(x_T). \tag{4.8}$$

Here, M is the conjunction $M = M_1 \wedge M_2 \wedge M_3 \wedge M_4$ of the move clauses from Equations (4.3) to (4.6). With this BMC formula, the time T can be gradually unrolled until a solution is found. This formula can be given to a SAT solver to find a solution for the spooky pebble game problem.

Algorithm 1 Spooky pebble game solver using SAT (runtime timeout t_{max}).

```
1: T, T_prev := 0, 0
2: formula = I(x_0) ∧ C_{P,S}(x_0) ∧ F(x_0)          ▷ encode pebble game for T = 0
3: while result ≠ SAT do
4:     formula := formula.pop(F_{x_prev})
5:     formula := formula.push(M(x_{T_prev}, x_{T_prev+1}) ∧ C_{P,S}(x_{T_prev+1}) ∧ ... ∧ M(x_{T-1}, x_T) ∧
       C_{P,S}(x_T) ∧ F(x_T))                          ▷ unroll formula up to time T
6:     T_prev := T
7:     result := SAT(formula) ▷ SAT solver returns (un)sat, or timeout after t_wait seconds
8:     if result = timeout then
9:         T := T + T^(skip)
10:    if result = UNSAT then
11:        T := T + 1
```

In Algorithm 1 we state our heuristic for the spooky pebble game solver. The solver starts with time $T = 0$. If the formula is proven to be *unsatisfiable* (i.e., the graph cannot be pebbled with P pebbles and S ghosts in T moves), T is incremented by 1. If the SAT solver cannot find a solution in runtime t_{wait} but the formula cannot be proven to be unsatisfiable, it returns a *timeout* and T is incremented with $T^{(skip)}$. For a larger time T the solver may find a solution faster, but this solution might not have optimal depth.[6] If the SAT solver

[6]We define $T^{(skip)}$ (which is possibly greater than 1) because every call to the SAT solver takes some 'useless' time to optimize the SAT formula. Thus the pebble game solver usually can find a spooky pebble game solution faster when $T^{(skip)} > 1$.

finds a *satisfiable* solution, the program is stopped as a strategy for the spooky pebble game is found. This entire solving procedure has a timeout of t_{max}.

Using push and pop statements for the final clauses, incremental SAT solving can be used, which updates the SAT formula for every new time T. This incremental extension of the formula speeds up the SAT solver in subsequent runs.

We remark that the above encoding implements parallel pebbling semantics so multiple pebbles and ghosts can be added or removed at the same timestep. For example, in the pebble configuration at the top of Figure 2, the steps *unpebble(B)*, *unpebble(C)* and *pebble(F)* can be applied in one parallel timestep, see Figure 8. In general, the parallel pebbling time for a game is much smaller than the sequential pebbling time as multiple sequential steps can be applied in one parallel step. This highly reduces the number of BMC unrollings to find a solution. As every BMC unrolling duplicates the move and cardinality clauses as well as the variables all the time, the size of the SAT formula can be highly reduced too. This reduction of clauses and variables in the SAT formula speeds up the SAT solver. A disadvantage of this method is that the solver in general does not provide a solution with optimal sequential time.

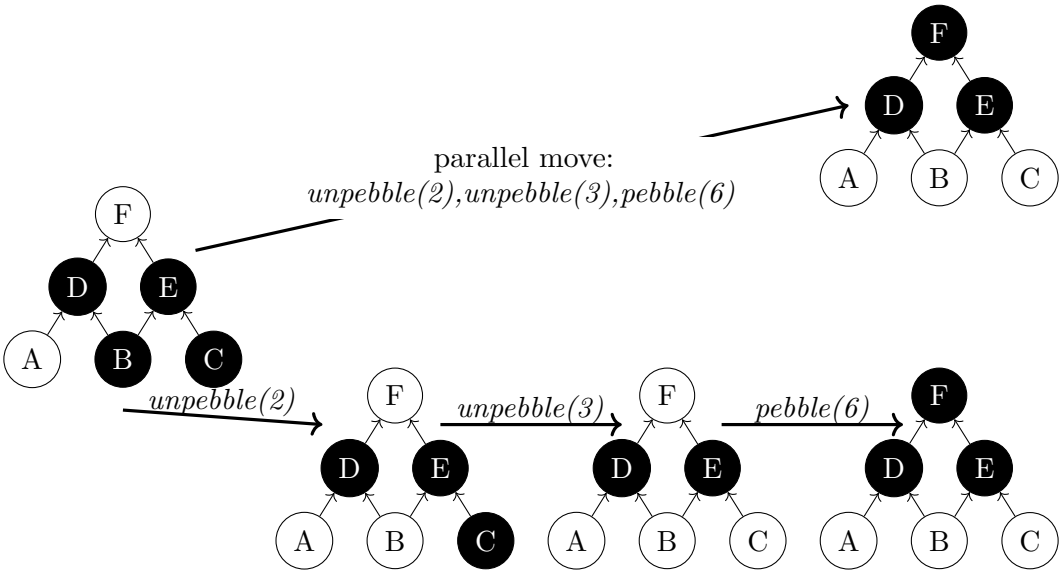


Figure 8: Example of a parallel move which consists of 3 sequential moves. Allowing parallel moves reduces the number of moves and hence reduces the size of the SAT encoding of a pebble game.

By linearizing solutions with parallel pebbling semantics, we can easily obtain a sequential solution, in which one pebble or ghost is added or removed at each time step. These sequential times (which we usually refer to as *pebbling time*, Section 2) are usually reported for pebbling strategies, see e.g. [MSR+19], and these sequential pebbling strategies are used by our optimization heuristics in the subsequent section.

4.2. **Heuristics for Optimization.** The solutions to the spooky pebble game as found by SAT solving described in the previous section are usually not optimal. Often the number of pebbles, ghosts or time steps can be reduced. In this section we will describe how we

optimize these solutions using a combination of heuristic methods. These heuristics reduce the number of pebbles, the number of ghosts and/or the depth of the solution.

The suboptimallity of the spooky pebble game solutions the SAT solver outputs arises from our SAT encoding, because we do not constrain the SAT solver to optimize the number of moves, the number of pebbles or the number of ghosts in a solution. Moreover, with our encoding of the spooky pebble game, the SAT solver searches only for semantically parallel solutions as this highly decreases the SAT solver runtime, but this can give inefficient sequential solutions.

We tried to find solutions with optimal number of pebbles and ghosts by using exact methods, for example MaxSAT.[7] These methods turned out to be very slow: they did not provide results on inputs that the SAT solver could easily deal with. This happens because exact methods only cannot find almost optimized solutions but only exactly optimized solutions, and exactly optimized solutions may be very difficult to find. Optimizing this way thus seemed to be impractical.

Therefore, we propose heuristic methods to optimize the spooky pebble game solutions provided by the SAT solver. These heuristics take as input a spooky pebble game strategy and output an optimized strategy with less pebbles, less ghosts or less (sequential) time. Using these optimizers we can find closer Pareto fronts, such that we can find more precise trade-offs between pebbles, ghosts and (sequential) time. We developed 6 optimization algorithms, which take all a greedy local optimum for one optimization goal such as removing all useless pebbling moves. These optimizers are described in Algorithms 2 to 7. Table 1 presents an overview of all optimizers and links them to their algorithms. Applying these basic optimizers one after the other gives an improved solution with respect to the input.

Table 1: Overview of optimization algorithms.

move type	**optimization type**	**gain**	**possible cost**	**algorithm**
pebble	delay	pebble reduction	-	4
ghost	delay	ghost reduction	pebble	6
unpebble	bring forward	pebble reduction	-	5
unghost	bring forward	ghost reduction	pebble	7
pebble + unpebble	remove moves	pebble reduction	-	2
ghost + unghost	remove moves	ghost reduction	pebble	3

These optimization heuristics all follow the same strategy: search for a type of move (such as pebble, unpebble, ghost or unghost); try to delay the move (for pebble and ghost) or to bring the move forward (for unpebble and unghost). This delay or bringing forward of moves is only possible under some conditions, e.g. for pebbling moves, the inputs must be pebbled at a later time and the pebble is not used to pebble or unghost another vertex.

The use of these optimizers is the following. By delaying pebbling and ghosting moves, useless pebbles or ghosts are removed. For example, if a pebble can be placed at a later time, this pebble can be used to pebble another node or be removed entirely, reducing the number of needed pebbles. Delaying ghost moves gives the same reduction for ghosts. Note that delaying a ghost move may need more pebbles, as a pebble is needed for a longer time if a ghost move is delayed. A similar reduction of pebbles resp. ghosts can be obtained by bringing unpebble and unghost moves forward.

[7] More specific, we tried the method of [NB14] as implemented in Z3 by [BdMNW].

The optimizers that delay pebbling and bring unpebbling forward can also be combined. This results in an optimizer that removes unused pebbles by removing both the pebble and unpebble move. A similar optimizer that combines delay of ghosting and bring forward of unghosting is also defined.

The complexity of these optimizer algorithms is only linear in the time of the solution and linear in the number of nodes and edges of the graph when storing the graph by adjacency lists. If the number of inputs or the number of outputs of a node is unbounded, some optimizer algorithms scale linearly with these as well. We thus see that the optimizers are efficient.

Given a pebbling strategy $((P_0, S_0), ..., (P_T, S_T))$, our algorithms use the following predicates for all vertices $v \in V$ and time steps $t \in [T]$.

$$\begin{aligned} \mathit{pebbled}_{v,t} &\triangleq v \in (P_t \setminus P_{t-1}) \cap (V \setminus S_{t-1}) & \mathit{ghosted}_{v,t} &\triangleq v \in S_t \setminus S_{t-1} \\ \mathit{unpebbled}_{v,t} &\triangleq v \in (P_{t-1} \setminus P_t) \cap (V \setminus S_t) & \mathit{unghosted}_{v,t} &\triangleq v \in S_{t-1} \setminus S_t \\ \mathit{used}_{v,t} &\triangleq \mathit{pebbled}_{v,t} \vee \mathit{unpebbled}_{v,t} \vee \mathit{unghosted}_{v,t} \end{aligned}$$

Algorithm 2 Remove useless pebblings from strategy $((P_0, S_0), ..., (P_T, S_T))$.

```
for t = T, T−1, ..., 1 do
    for v ∈ (P_{t−1} \ P_t) ∩ (V \ S_t) do        ▷ For each unpebbling move at time step t
        for t_0 = t−1, ..., 1 do
            if unghosted_{v,t_0} ∨ ∃(w, v) ∈ E: used_{w,t_0} then
                break                              ▷ v was unghosted or used at time step t_0
            if pebbled_{v,t_0} then
                P_i := P_i \ {v} for i ∈ [t_0, t]   ▷ remove pebble v at time steps t_0 to t
```

Algorithm 3 Remove useless ghostings from strategy $((P_0, S_0), ..., (P_T, S_T))$.

```
for t = T, T−1, ..., 1 do
    for v ∈ S_t \ S_{t−1} do                       ▷ For each ghosting move at time step t
        if ∀(v, w) ∈ E : w ∈ (P_{t−1} ∩ P_t) then  ▷ all inputs of v are pebbled
            t_0 := t
            while v ∈ S_{t_0} do                   ▷ replace ghosting by unpebbling: remove ghost
                S_{t_0} := S_{t_0} \ {v}
                t_0 := t_0 + 1
```

Algorithm 4 Delay pebble placements from strategy $((P_0, S_0), ..., (P_T, S_T))$.

for $t = T, T-1, ..., 1$ **do**
  for $v \in (P_t \setminus P_{t-1}) \cap (V \setminus S_{t-1})$ **do** ▷ For each pebbling move at time step t
    $t_0 := t$
    while $v \in P_{t_0}$ **and** $t_0 < T$ **do**
      if $\exists (w, v) \in E : used_{w,t_0} \vee used_{w,t_0+1}$ **then**
        break ▷ v was used at timestep t_0 or $t_0 + 1$
      if $\forall (v, w) \in E : w \in (P_{t_0-1} \cap P_{t_0})$ **then** ▷ all inputs of v are pebbled
        $P_i := P_i \setminus \{v\}$ **for** $i \in [t, t_0 - 1]$ ▷ shift pebble move on v from t to t_0
      $t_0 := t_0 + 1$

Algorithm 5 Expedite unpebblings from strategy $((P_0, S_0), ..., (P_T, S_T))$.

for $t = 1, 2, ..., T$ **do**
  for $v \in (P_{t-1} \setminus P_t) \cap (V \setminus S_t)$ **do** ▷ For each unpebbling move at time step t
    $t_0 := t - 1$
    while $v \in P_{t_0}$ **and** $v \notin S_{t_0-1}$ **and** $t_0 > 0$ **do**
      if $\exists (w, v) \in E : used_{w,t_0} \vee used_{w,t_0+1}$ **then**
        break ▷ v was used at timestep t_0 or $t_0 + 1$
      if $\forall (v, w) \in E : w \in (P_{t_0-1} \cap P_{t_0})$ **then** ▷ all inputs of v are pebbled
        $P_i := P_i \setminus \{v\}$ **for** $i \in [t_0, t - 1]$ ▷ shift unpebble move on v from t to t_0
      $t_0 := t_0 - 1$

Algorithm 6 Delay ghost placements from strategy $((P_0, S_0), ..., (P_T, S_T))$.

for $t = T, T-1, ..., 1$ **do**
  if $|P_t| < P$ **then** ▷ If not all pebbles are used at time t
    for $v \in S_t \setminus S_{t-1}$ **do** ▷ For each ghosting move at time step t
      $P_t := P_t \cup \{v\}$ ▷ remove ghost and place pebble on v at time t
      $S_t := S_t \setminus \{v\}$
      if $|P_t| \geq P$ **then** ▷ Check current availability of pebbles
        break

Algorithm 7 Replace ghost by pebble as soon as possible from strategy $((P_0, S_0), ..., (P_T, S_T))$.

for $t = 1, 2, ..., T$ **do**
  if $|P_t| < P$ **then**
    for $v \in S_{t-1} \setminus S_t$ **do** ▷ For each unghosting move at time step t
      $t_0 := t - 1$
      while $v \in S_{t_0}$ **and** $t_0 > 0$ **do**
        if $|P_t| \geq P$ **then** ▷ Check current availability of pebbles
          break
        if $\forall (v, w) \in E : w \in (P_{t_0-1} \cap P_{t_0})$ **then** ▷ all inputs of v are pebbled
          for $i \in [t_0, t - 1]$ **do** ▷ shift unghost move on v from t to t_0
            $P_i := P_i \cup \{v\}$
            $S_i := S_i \setminus \{v\}$
        $t_0 := t_0 - 1$

5. Implementation and Experiment

We implemented our spooky game solver algorithm from Section 4 in Python. The code is open source and can be found in [Qui25]. The Z3 solver for satisfiability is used as SAT solver [DMB08]. The optimizer algorithms were implemented in Python.

Table 2: Properties of the DAGs of the ISCAS85 benchmarks used to benchmark the spooky pebble game solver.

name	vertices	roots	edges
c432	172	7	260
c499	177	32	246
c880	276	26	374
c1355	177	32	246
c1908	193	25	257
c2670	401	46	533
c3540	830	22	1395
c5315	1089	96	1503
c6288	979	32	1639
c7552	988	62	1584

To test the performance of our implementation, we use an Intel Core i7-6700 CPU with eight cores clocked at 3.40GHz. We use the well-known ISCAS85 benchmarks circuits [BF85]. Using the open source tool mockturtle, we transformed the circuits via XOR-majority graphs (DAGs) to circuits with common quantum gates [SRWDM17]. Table 2 shows the characteristics of the resulting DAGs.

For all these benchmarks we measure the performance of the pebble game solver. The goal is to find a Pareto front of spooky pebble game solutions with optimal parameters (sequential) time, number of pebbles and number of ghosts. The procedure is described in detail in Algorithms 8 and 10. For various numbers of ghosts a solution is searched with a decreasing number of pebbles. For each constraint, the solver is run three times with different seeds. If at least one solution is found, the number of pebbles is decreased by five. If no solution is found, the number of pebbles (P) is not decreased anymore and the iteration with next number ghosts (S) is started.

Algorithm 8 Optimal solutions search for benchmarks. Function spooky_pebble_game_solver(pebbles, ghosts) is described in Algorithm 1.

```
1: for S = vertices, vertices/5, vertices/10, vertices/20, 0 do
2:     spooky_pebble_game_solver(∞, S)              ▷ run solver five times
3:     pb := minimal #pebbles in found solutions
4:     for P = pb, pb-5, pb-10, ... do
5:         spooky_pebble_game_solver(P, S)          ▷ run solver five times
6:         if no solution is found by solver then
7:             break
```

As runtime parameters for the solver (Algorithm 1), we use $t_{wait} = 15$ seconds and $t_{max} = 2$ minutes. The solver gives little results for these runtimes on large benchmarks.[8]

Thus, we used a longer runtime for the large benchmarks. For the four largest benchmarks, $t_{wait} = 60$ seconds and $t_{max} = 8$ minutes is used. For all benchmarks, $T^{(skip)}$ is set to 5.

Algorithm 9 Transform parallel solution $((P_0, S_0), ..., (P_T, S_T))$ with P pebbles and S ghosts obtained from SAT solver into sequential solution $((P'_0, S'_0), ..., (P'_{T'}, S'_{T'}))$.

$(P'_0, S'_0) := (P_0, S_0)$
$T' := 1$
for $t = 0, 1, ..., T-1$ **do**
  pebbling, unpebbling, ghosting, unghosting = *empty list*
  for $v \in V$ **do**
    if $v \in P_{T+1} \setminus P_T$ **then** ▷ v is pebbled
      pebbling.push(v)
    if $v \in P_T \setminus P_{T+1}$ **then** ▷ v is unpebbled
      unpebbling.push(v)
    if $v \in S_{T+1} \setminus S_T$ **then** ▷ v is ghosted
      ghosting.push(v)
    if $v \in S_T \setminus S_{T+1}$ **then** ▷ v is unghosted
      unghosting.push(v)
  for $v \in$unpebbling **do**
    $(P'_{T'}, S'_{T'}) := (P'_{T'-1} \setminus \{v\}, S'_{T'-1})$
    $T' := T' + 1$
  while |ghosting| > 0 **or** |unghosting| > 0 **do**
    if $|P'_{T'-1}| \geq P$ **and** $|S'_{T'-1}| \geq S$ **then** ▷ ghost and unghost in one timestep
      v :=unghosting.pop()
      w :=ghosting.pop()
      $(P'_{T'}, S'_{T'}) := ((P'_{T'-1} \cup \{v\}) \setminus \{w\}, (S'_{T'-1} \cup \{w\}) \setminus \{v\})$
      $T' := T' + 1$
    if |unghosting| > 0 **and** $|P'_{T'-1}| < P$ **then**
      v :=unghosting.pop()
      $(P'_{T'}, S'_{T'}) := (P'_{T'-1} \cup \{v\}, S'_{T'-1} \setminus \{v\})$
      $T' := T' + 1$
    if |ghosting| > 0 **and** $|S'_{T'-1}| < S$ **then**
      v :=ghosting.pop()
      $(P'_{T'}, S'_{T'}) := (P'_{T'-1} \setminus \{v\}, S'_{T'-1} \cup \{v\})$
      $T' := T' + 1$
  for $v \in$pebbling **do**
    $(P'_{T'}, S'_{T'}) := (P'_{T'-1} \cup \{v\}, S'_{T'-1})$
    $T' := T' + 1$

[8]Large benchmarks need many clauses and variables to encode one BMC iteration, and usually also take a larger pebbling time (more BMC iterations). Hence, their SAT formula is much larger, which makes it a harder job for the SAT solver to find a solution.

Algorithm 10 Optimize solution $((P_0, S_0), ..., (P_T, S_T))$ obtained from SAT solver. Other orders of optimizers are possible, this is random.

```
solution := ((P_0, S_0), ..., (P_T, S_T))
while solution improved do        ▷ time, pebbles or ghosts reduced w.r.t. previous run
    solution := remove_ghostings(solution)
    solution := remove_pebblings(solution)
    solution := delay_pebbling(solution)
    solution := bring_forward_unpebbling(solution)
    solution := delay_ghosting(solution)
    solution := bring_forward_unghosting(solution)
```

For every solution the SAT solver found, we sequentialize the parallel solution from the SAT solver using Algorithm 9. After sequentialization, we run the optimizer algorithm as described in Algorithm 10 on the solution. For every pebbling strategy the SAT solver found, this algorithm is run five times, each time with a different order of optimizations. If the solution is not optimized by the six optimization operations, the optimization algorithm is stopped.

Table 3: Maximal runtimes of pebble game (PG) solver and optimization heuristic per benchmark.

name	PG solver	heuristic
c432	121.3	13.7
c444	111.3	11.5
c880	117.8	44.5
c1355	110.4	13.8
c1908	108.8	19.4
c2670	118.5	97.9
c3540	464.6	814.5
c5315	397.7	802.8
c6288	361.9	1341.9
c7552	46.2	146.1

In Figure 9, the results of the runs of the spooky pebble game solver are depicted. The color of the datapoint indicates the number of ghosts used by the solution. The large dots indicate the solutions found by the SAT solver. The smaller dots indicate the pebbling strategies found by optimizing the SAT solver solutions. Notice that there are many optimized pebbling strategies as all solutions found during Algorithm 10 are depicted.

Table 3 shows the maximal runtimes of the SAT solver and the optimizer to find a solution, respectively to optimize the solution. The maximal runtimes of the SAT solver are indeed bounded by 2 minutes (for the 6 smallest benchmarks) and 8 minutes (for the 4 largest benchmarks).[9] The maximal runtimes of the optimizers are strongly correlated to the maximal sequential time of the found solutions. This is because the time complexity of

[9]The runtime for c432 is marginally longer, as the time for loading the benchmark into the solver is also taken into account. This loading takes some additional seconds.

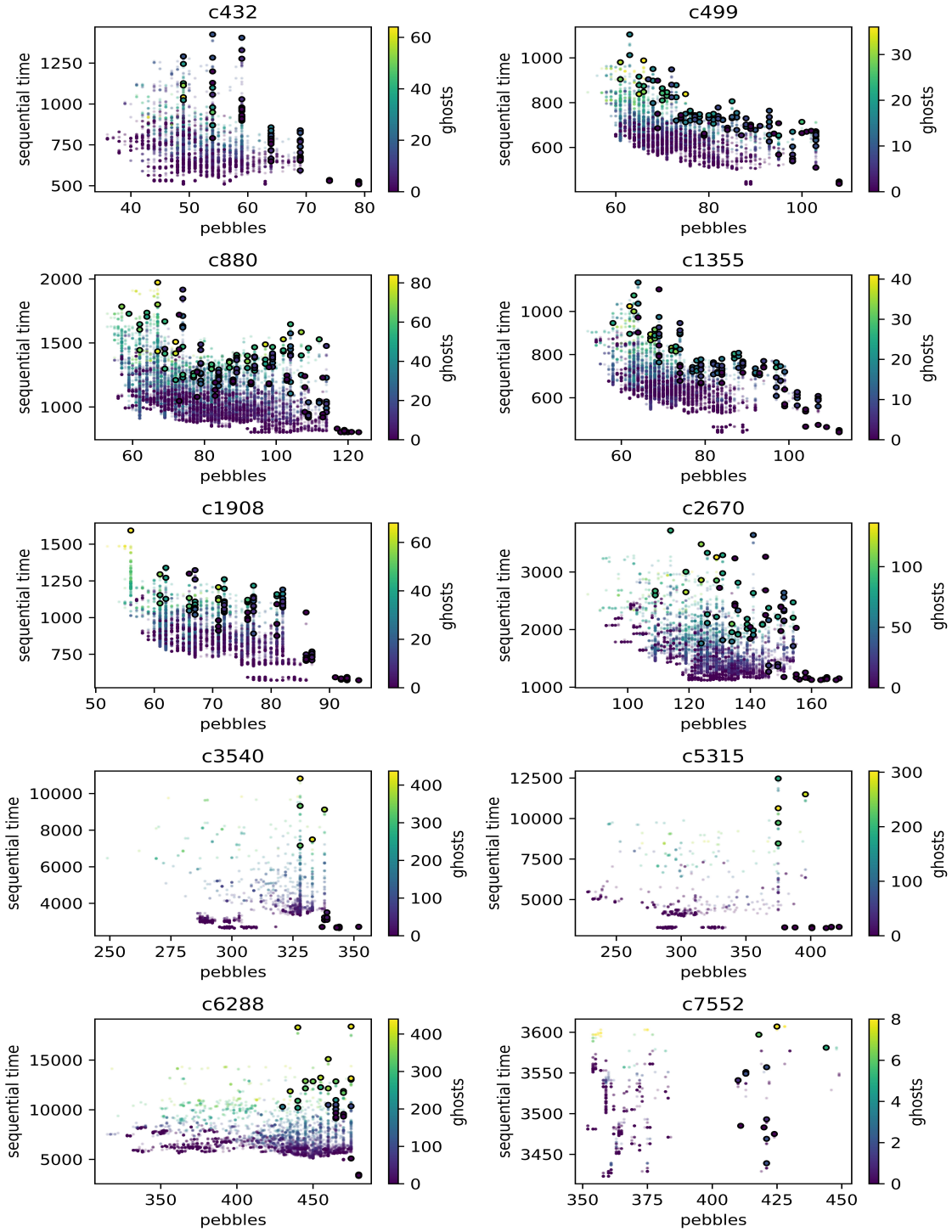


Figure 9: Solutions of the spooky pebble game with sequential time, number of pebbles and number of ghosts found by the SAT solver heuristic of Algorithm 8 (big dots) and their optimized solutions as found by the optimizer heuristics of Algorithm 10 (small dots).

the optimizers goes linear in the sequential time of the solution. The optimizers can be sped up by implementing the optimizers in e.g. C or C++ instead of Python, as their execution involves extensive use of for-loops. Therefore, these long runtimes of the optimizers seem to be no bottleneck or problem for the total runtime of the solver.

The data in Figure 9 shows that in all cases the optimizers reduce the number of ghosts and sequential time of the pebbling strategies with respect to the strategies found by the SAT solver. Moreover, the number of pebbles is reduced by the optimizers. This reduction of pebbles is significant (up to 39%). Thus, the SAT solver is a good starting point to find pebbling strategies, but the optimizers can improve the solution substantially.

These results show a well-shaped front of solutions without unexpected jumps or irregular discontinuities or non-convexity. This improves upon the results of the previous version of this paper [QL23].

Moreover, we observe that many optimized solutions with minimal sequential time use (almost) no ghosts. This ghost reduction is much better than the SAT solver pebbling strategies, as provided in the previous version of this paper [QL23]. Nevertheless, for most benchmarks the pebbling strategy with the least number of pebbles uses ghosts. Thus, optimization of quantum space is still obtained by using classical space.

For the cases where the pebbling strategy for a specific number of pebbles with minimal sequential time uses non-zero ghosts, the sequential time is much less than in similar cases with zero ghosts. See e.g. 62 pebbles in c880 or 67 in c1355. This indicates that not reducing the number of ghosts may reduce the sequential time. But as our optimization algorithms usually reduce the number of ghosts, this behavior is rarely observed.

6. Conclusions and further research

In this paper, we explored the spooky pebble game, a model for the trade-off between quantum space, classical space and time in a quantum computation. Using the spooky pebble game, we proved that a classical calculation that uses space C can be executed by a quantum computer using quantum space $C + m$ and some additional classical space, where m is the space of the output. In addition, we showed that solving the spooky pebble game is PSPACE-complete.

Moreover, we showed the design and implementation of a spooky pebble game solver, which gives a memory management strategy for a quantum computation with a limited amount of quantum space and classical space. This solver uses a SAT solver to initially find pebbling strategies. Using this exact SAT solver method, optimal solutions are very hard to find. The (non-optimal) strategies found by the SAT solver are combined with heuristics to optimize these moderate solutions into significantly better local optima. Thus the SAT solver and the heuristics are complementary: the SAT solver can only find moderate solutions, but the heuristics can optimize this by pebble reductions up to 39%. This is not surprising as solving the spooky pebble game is PSPACE-complete, and SAT solving is used here as an NP-oracle. Although solving a spooky pebble game is PSPACE-complete, this solver runs in practice within limited runtime. This solver provides strategies that reduce quantum space when classical space is added. Moreover, with this solver we can find Pareto fronts for trade-offs between quantum space, classical space and time used to execute a circuit.

Further research. An idea for further research is to build more optimization heuristics for spooky pebble strategies. Currently used optimization strategies emphasize the reduction of ghosts, such that many optimized solutions use (almost) no ghosts. For example optimizers reducing the number of pebbles but increasing the number of ghosts can be developed to better optimize quantum space.

It would also be interesting to study other solving techniques for the spooky pebble game. The recently developed Incremental Property Directed Reachability [BL23] and divide-and-conquer methods [ZLY25] would be worthwhile to apply to spooky pebble game solving, as they have promising results for solving the reversible pebble game.

Another idea for further research is to investigate in which contexts the spooky pebble game and measurement-based uncomputation can be used. Our main motivation in this paper to study these concepts was simulation of classical algorithms on a superposition of inputs on a quantum computer. It would be interesting to investigate the extent to which these techniques can also be used for broader classes of quantum algorithms.

Finally, it would be worthwhile to investigate other quantum space reduction techniques and combine them with spooky pebble games. A promising candidate is Quantum Catalytic Space [BFM+25]. Catalytic space has already had great impact on the classical setting [CM24], so developing a pebble game formulation for the quantum setting and integrating it into the spooky pebbling framework would be a promising next step.

Acknowledgements. This publication is part of the project Divide & Quantum (with project number 1389.20.241) of the research programme NWA-ORC which is partly financed by the Dutch Research Council (NWO).

References

[AHM+00] Mikel Aldaz, Joos Heintz, Guillermo Matera, José Luis Montaña, and Luis Miguel Pardo. Time-space tradeoffs in algebraic complexity theory. *journal of complexity*, 16(1):2–49, 2000. doi:10.1006/jcom.1999.0526.

[BdMNW] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger. Programming z3. URL: https://theory.stanford.edu/~nikolaj/programmingz3.html.

[Ben89] Charles H Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. doi:10.1137/0218053.

[BF85] F Brglez and H Fujiwara. A neutral netlist of 10 combinational benchmark circuits. In *Proc. IEEE Int. Symp. Circuits and Systems, Jun. 1985*, pages 695–698, 1985.

[BFM+25] Harry Buhrman, Marten Folkertsma, Ian Mertz, Florian Speelman, Sergii Strelchuk, Sathyawageeswar Subramanian, and Quinten Tupker. Quantum catalytic space. *arXiv preprint arXiv:2506.16324*, 2025. URL: https://arxiv.org/abs/2506.16324.

[BGB+18] Ryan Babbush, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4):041015, 2018. doi:10.1103/physrevx.8.041015.

[BHv21] A. Biere, M. Heule, and Maaren H. van. *Handbook of satisfiability*. Frontiers in artificial intelligence and applications ; 336. IOS Press, Amsterdam, second edition, 2021. doi:10.3233/faia336.

[BL23] Max Blankestijn and Alfons Laarman. Incremental property directed reachability. In *International Conference on Formal Engineering Methods*, pages 208–227. Springer, 2023. doi:10.1007/978-981-99-7584-6_13.

[Cha13] Siu Man Chan. Just a pebble game. In *2013 IEEE Conference on Computational Complexity*, pages 133–143. IEEE, 2013. doi:10.1109/ccc.2013.22.

[CLNV15] Siu Man Chan, Massimo Lauria, Jakob Nordstrom, and Marc Vinyals. Hardness of approximation in pspace and separation results for pebble games. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 466–485. IEEE, 2015. doi:10.1109/focs.2015.36.

[CM24] James Cook and Ian Mertz. Tree evaluation is in space O(log n· log log n). In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1268–1278, 2024.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

[EBvL78] P Emde Boas and J van Leeuwen. Move rules and trade-offs in the pebble game. Technical Report RUU-CS-78-4, Rijksuniversiteit Utrecht, April/August 1978. doi:10.1007/3-540-09118-1_12.

[Gid19] Craig Gidney. Spooky pebble games and irreversible uncomputation, 2019. Blog post. URL: https://algassert.com/post/1905.

[GLT79] John R Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 237–248, 1979. doi:10.1145/800135.804418.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

[KSS21] Niels Kornerup, Jonathan Sadun, and David Soloveichik. The spooky pebble game. *arXiv preprint arXiv:2110.08973*, 2021. URL: https://arxiv.org/abs/2110.08973v1.

[KSS24] Niels Kornerup, Jonathan Sadun, and David Soloveichik. Tight bounds on the spooky pebble game: Recycling qubits with measurements. *arXiv preprint arXiv:2110.08973*, 2024. URL: https://arxiv.org/abs/2110.08973v2.

[LTV98] Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120(1-2):168–176, 1998. doi:10.1016/s0167-2789(98)00052-9.

[MSR+19] Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjorner, and Giovanni De Micheli. Reversible pebbling game for quantum memory management. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 288–291. IEEE, 2019. doi:10.23919/date.2019.8715092.

[NB14] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014. doi:10.1609/aaai.v28i1.9124.

[Nor15] Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results, 2015. URL: https://www.csc.kth.se/~jakobn/research/PebblingSurveyTMP.pdf.

[Pei20] Chris Peikert. He gives c-sieves on the csidh. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 463–492. Springer, 2020. doi:10.1007/978-3-030-45724-2_16.

[PPR19] Daniel K Park, Francesco Petruccione, and June-Koo Kevin Rhee. Circuit-based quantum random access memory for classical data. *Scientific reports*, 9(1):3949, 2019. doi:10.1038/s41598-019-40439-3.

[QL23] Arend-Jan Quist and Alfons Laarman. Optimizing quantum space using spooky pebble games. In *International Conference on Reversible Computation*, pages 134–149, 2023. doi:10.1007/978-3-031-38100-3_10.

[Qui25] Arend-Jan Quist. Spooky pebble game, October 2025. doi:10.5281/zenodo.17476182.

[Sav70] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. doi:10.1016/s0022-0000(70)80006-x.

[Sav08] John E Savage. *Models of Computation: Exploring the Power of Computing*. The book is released in electronic form under a CC-BY-NC-ND-3.0-US license., 2008. URL: https://cs.brown.edu/people/jsavage/book/pdfs/ModelsOfComputation.pdf.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. IEEE, 1994. doi:10.1109/sfcs.1994.365700.

[SM73] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, 1973. doi:10.1145/800125.804029.

[SRWDM17] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. Design automation and design space exploration for quantum computers. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 470–475. IEEE, 2017. doi:10.23919/date.2017.7927035.

[Swa78] Sowmitri Swamy. *On space-time tradeoffs*. PhD thesis, Brown University, 1978.

[Val76] L. G. Valiant. Space-time tradeoffs in computations. *Asterisque*, 38–39:254–264, 1976.

[ZLY25] Kezhen Zhang, Riling Li, and Mingsheng Ying. A divide-and-conquer pebbling strategy for oracle synthesis in quantum computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025. doi:10.1109/TCAD.2025.3570608.

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany