Details regarding the construction and execution of protocols recipes

Sébastian de Bone[1,2] and David Elkouss[1,3]

[1] QuTech, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, The Netherlands
[2] QuSoft, CWI, Science Park 123, 1098 XG Amsterdam, The Netherlands
[3] Networked Quantum Devices Unit, Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan

1 Binary tree

Here, we present a method for converting GHZ creation and distillation protocols into a set of ordered instructions. We consider GHZ creation protocols over N parties in a network: the "network nodes" $\{\nu^{(i)}\}_{i=1}^{N}$. The protocols are generated by the dynamic program presented in the main paper and in Ref. [2], and have the form of a directed *binary tree*. This means that each node in the tree is either created from zero or two direct *children* of the graph. At the top of the binary tree, we find the operation that creates the final GHZ state. At the end leaves of each of the branches of the tree, we find the "elementary links" $\{e_i\}_{i=1}^{k}$. These are the Bell pairs created between two of the network nodes. The elementary links do not have children. An example of a binary tree can be found in Fig. 1a.

The fusion operations $\{f_i\}_{i=1}^{F}$ and distillation operations $\{d_i\}_{i=1}^{D}$ in the tree each have two children. A fusion operation creates a state $|\mathrm{GHZ}_{n_1+n_2-1}\rangle$ out of two children $|\mathrm{GHZ}_{n_1}\rangle$ and $|\mathrm{GHZ}_{n_2}\rangle$, where $|\mathrm{GHZ}_n\rangle$ is a weight-n GHZ state. This is possible if there is one network node that holds a qubit from both $|\mathrm{GHZ}_{n_1}\rangle$ and $|\mathrm{GHZ}_{n_2}\rangle$. A distillation operation, on the other hand, uses an entangled state of the form $|\mathrm{GHZ}_{n_1}\rangle$ to non-locally measure a *stabilizer operator* of weight n_1 from $\mathcal{S}_{n_2}$ on a target state $|\mathrm{GHZ}_{n_2}\rangle$ [5, 1, 4]. Here, the stabilizer operators of $|\mathrm{GHZ}_{n_2}\rangle$ are all operators

$$\mathcal{S}_{n_2} = \langle X_1 X_2 \dots X_{n_2}, Z_1 Z_2, Z_2 Z_3, \dots, Z_{n_2-1} Z_{n_2} \rangle. \tag{1}$$

We consider distillation as a probabilistic operation that only succeeds if the measurement outcome is $(-1)^m = +1$. In case of an outcome $(-1)^m = -1$, we reapply all operations in sub-tree of the operation $d \in \{d_i\}_{i=1}^{D}$—*i.e.*, d itself and all operations below it. More information about the specific operations applied for fusion and distillation can be found in the main paper, or in Ref. [2].

Each binary tree satisfies $D + F = k - 1$, where D, F and k are the number of distillation operations, fusion operations, and elementary links, respectively. Since creating a weight-n GHZ state requires a minimum of $n - 2$ fusion operations working on $n - 1$ elementary links, all additional fusion operations create entangled states used for distillation purposes. Therefore, we use k (the number

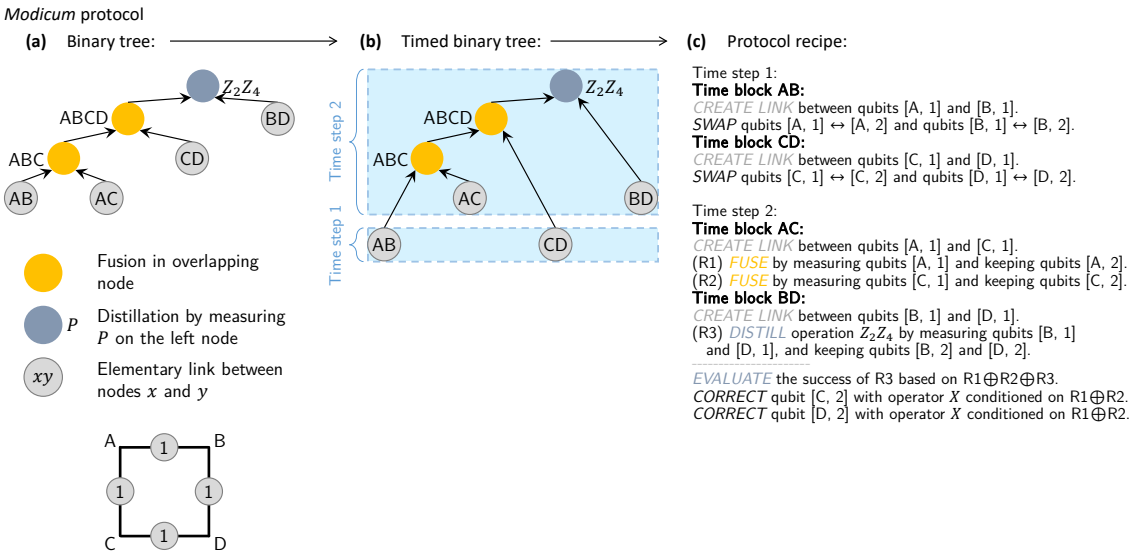

Fig. 1. (a) Binary tree with $k = 4$ found with the dynamic program of Ref. [2]. (b) Identified time steps for the operations in this binary tree. (c) Identified protocol recipe for this binary tree.

of elementary links in a binary tree) as a proxy for the amount of distillation that takes place in a GHZ protocol.

2 Protocol recipe construction

In the presence of decoherence, the order in which the operations are applied can have a strong impact on performance. Therefore, we convert the binary tree of a GHZ creation protocol to a *protocol recipe*. This is a set of instructions that describe what specific operations have to be applied on what qubits of the network nodes. In this section, we discuss the algorithm we use to create a protocol recipe. Both in this section and in the next section, we present a general description of the algorithm in the main text and add specific details in shaded text boxes.

The first step of the protocol recipe construction consists of ordering the elementary links of the binary tree. For this, we follow a recursive approach. We start at the top of the tree. At each step, we select the sub-tree with the largest size, choosing the left one in case of ties. When we reach an elementary link at one of the leaves of the tree, we add this link to an ordered list with elementary links, together with all other non-overlapping elementary links that can be carried out simultaneously.

For each elementary link e that we add to the list, we check if the parent operation p of e can be applied. A parent can be applied if both of its children

are contained in the list of operations. If p can be applied, we add p to the list after e. Then we check if the parent of p can be applied; if it can be added, we add it after p, *etc.*

For each operation in the list, we associate a set of instructions that are physically possible with the target hardware. For example, we assume a single "communication" qubit $c^{(j)}$ per node that can be used to perform operations, and a fixed number of memory qubits Q_j "memory" qubits $\{m_q^{(j)}\}_{q=1}^{Q_j}$ that can be used to store states. We also assume that a node can only be involved in one entanglement operation at a time, and entanglement can only be created between two communication qubits. For this reason, SWAP gates are necessary to free up the communication qubits. For example, for a typical 2-to-1 Bell pair distillation step in the style of the *DEJMPS* distilllation protocol [3], we first create a Bell pair between two communication qubits $c^{(1)}$ and $c^{(2)}$, we swap them to the memory qubits $m_1^{(1)}$ and $m_1^{(2)}$, create a new Bell pair between communication qubits $c^{(1)}$ and $c^{(2)}$, and then carry out the distillation circuit. If distillation succeeds, this leads to a distilled Bell pair on memory qubits $m_1^{(1)}$ and $m_1^{(2)}$. If we now want to further use this distilled Bell pair as an ancillary state in a later operation, we typically have to swap it back to the communication qubits.

We define $N_o \subseteq \{\nu^{(i)}\}_{i=1}^{n}$ as the subset of network nodes in which an operation o takes place. For each elementary link operation $e \in \{e_i\}_{i=1}^{k}$ considered, we identify the set $E_e \subset \{e_i\}_{i=1}^{k}$, that for $e' \in E_e$ fulfills $N_{e'} \cap N_e = \varnothing$, where $\varnothing$ is the empty set. The operations in E_e are all elementary link operations that can be carried out simultaneously with e. We store e and the operations in E_e together as a set s_e in the list l_{link}, where the elementary links e are ordered according to the recursive binary tree approach discussed in the main text above.

Consequently, we identify a list l'_{link} with subsets of link generation operations that can be carried out simultaneously. For each element $o \in l_{\text{link}}$, we identify a subset s_o of link generation operations that can be carried out simultaneously with o. This subset can only contain one operation per node in the network—*i.e.*, there can, *e.g.*, be only one link generation operation in s_o that involves network node A, one link generation operation that involves network node B, *etc.*. The operation o is, itself, also part of s_o. The identified subset s_o is added to l'_{link}. Each operation in l_{link} is only added to one subset in l'_{link}.

Next, we create a list of operations l_{ops}. This list is created by looping over the subsets $s_o \in l'_{\text{link}}$ and by keeping track of the nodes' qubit occupancy during the operations that we add to l_{ops} (to identify the qubits on which the operations should take place). For each subset $s_o \in l'_{\text{link}}$, the link generation operations $o \in s_o$ are added to l_{ops}.

Subsequently, for each $o \in s_o$, we check if we can perform its direct parent operation p in the binary tree. This operation p is either a distillation operation or a fusion operation. We check if, at this point, both direct children of p are contained in l_{ops}. If this is the case, we add p to l_{ops}, and move on to the direct parent of p and do the same—until we stumble upon a parent node that does not have its two direct children contained in l_{ops} yet. For each operation added to l_{ops}, we first check if it is necessary to free up the communication qubits of the involved nodes, or swap back a state to the nodes' communication qubits. If one of those steps is necessary, we add SWAP gate operations to l_{ops} before the actual operation. For each fusion operation added to l_{ops}, we add information about what correction operations we need to perform in case of an odd measurement result. By default, this is a Pauli-X operator on all qubits of the right child in the tree that are not part of the network node in which the fusion operation takes place—so, *e.g.*, for a fusion operation between qubits in nodes ABC and CDE, this will be a Pauli-X operator on the qubits in nodes D and E.

Once all operations are listed, we schedule them. The operation schedule consists of a series of time steps, with each time step divided into different time blocks. The time blocks are created such that they can be executed in parallel in different, non-overlapping parts of the network. We schedule the operations by looping over the list of operations, and, for each operation o, checking if o can be placed in an existing time block. This is possible if the network nodes in which o operates overlap with the network nodes in which the other operations of the time block operate. If it is not possible to add o to an existing time block, we create a new one. At the end of each time step, we add all required fusion corrections, as well as distillation operations that need to be evaluated at the end of the time step. The result is a structure that we refer to as a protocol recipe. An example of a protocol recipe can be found Fig. 1c and in the supplementary document "Binary trees and protocol recipes best-performing protocols.pdf".

We group the operations in l_{ops} into different time steps $\{s^{(\tau)}\}_{\tau=1}^{T}$ and time blocks $\{b_i^{(\tau)}\}_{i=1}^{B_\tau}$ within these time steps. Every time block $b \in \{b_i^{(\tau)}\}_{i=1}^{B_\tau}$ has an associated subset N_b of network nodes in which its operations are carried out. The time blocks are created by looping over the operations $o \in l_{\text{ops}}$, for which wee identify the subset N_o of network nodes in which operations of o take place:

1. We start at the time step $s^{(\tau')} = s^{(T')}$, where $s^{(T')}$ corresponds to the last time step at this stage of the algorithm—*i.e.*, the current size $T' = |\{s^{(\tau)}\}_\tau|$. If $T' = 0$, we create $s^{(1)}$ and set $\tau' = 1$.

2. We check if there is a time block $b \in \{b_i^{(\tau')}\}_{i=1}^{B_{\tau'}}$ that fulfills $N_o \subseteq N_b$.
   a. If this is the case, we add o to this time block b and move on with the next operation in l_{ops}.
   b. If that is not the case, we check if there is at least one time block $b \in \{b_i^{(\tau')}\}_{i=1}^{B_{\tau'}}$ that fulfills $N_o \cap N_b \neq \varnothing$.
      i. If this is the case, we create or move to the next time step $s^{(\tau'+1)}$ and create a new time block b'' in time step $s^{(\tau'+1)}$. We set $n_{b''} = n_o$, add o to b'', and move on with the next operation in l_{ops}.
      ii. If that is not the case, we must have that $N_o \cap N_b = \varnothing$ holds for all $b \in \{b_i^{(\tau')}\}_{i=1}^{B_{\tau'}}$. If $\tau' > 1$ holds, we decrease τ' by one and move back to the beginning of step 2. If $\tau' = 1$ holds, we create a new time block b' in time step $s^{(1)}$, set $N_{b'} = N_o$, add o to b', and move on with the next operation in l_{ops}.

While adding the operations to the time blocks, we keep track of which operation nodes of our original binary tree are successfully created at each stage of the protocol. We do this by keeping track of a dictionary δ_{qubits} that contains information on what binary tree nodes sit on which qubits: after we add an operation with binary tree identification number i_{op} to the protocol recipe, we connect i_{op} to the qubits that hold the state after the operation.

To every time step $s^{(\tau)}$, we add a list $l_{\text{corr}}^{(\tau)}$ of fusion corrections and a list $l_{\text{eval}}^{(\tau)}$ of distillation operations that need to be applied and evaluated after the time step. As mentioned earlier, fusion operators need corrections when their measurement outcome is odd. Typically, the corrections are applied in other nodes as where the fusion operation(s) take place. Because these nodes are typically part of a different time block of the time step, we collect all fusion correction and delay them until the end of a time step. Sometimes, fusion corrections influence the outcome of distillation measurements. This can occur if the fusion correction in a different time block does not commute with one of the measurements of the distillation operation. If this is the case, the success or failure of a distillation operation cannot be determined in the time blocks itself. In that case, we also delay the decision on whether or not distillation was successful to the end of a time step, where all fusion measurement outcomes are available. Similarly, it can occur that a fusion correction is altered by operations applied between its associated fusion operation and the end of the time step. We use standard commutation rules to modify these corrections (and their dependencies) in $l_{\text{corr}}^{(\tau)}$.

Specifically, this is achieved by adding the correction operators of fusion operations that are newly added to time step $s^{(\tau)}$ to $l_{\text{corr}}^{(\tau)}$ without

alteration—since, by default, they are added to the end of a time block. However, every time we now add a new operation o to one of the time blocks of a time steps $s^{(\tau')}$, we have to evaluate their action on the existing fusion corrections in $l_{\text{corr}}^{(\tau'')}$ for $\tau' \leq \tau'' \leq |\{s^{(\tau)}\}_\tau|$. If o is a distillation operation that does not commute with at least one fusion correction in $l_{\text{corr}}^{(\tau'')}$ for $\tau' \leq \tau'' \leq |\{s^{(\tau)}\}_\tau|$, o is added to $l_{\text{eval}}^{(\tau''')}$ and the non-commuting fusion operation(s) are added to the outcomes required to determine the success of o. Here, $s^{(\tau''')}$ corresponds to latest time step in the range $\tau' \leq \tau''' \leq |\{s^{(\tau)}\}_\tau|$ with fusion corrections that do not commute with the distillation operation o.

3 Protocol recipe execution

In this section, we sketch the logic for the efficient simulation of a protocol recipe in the form of Sec. 2.

The GHZ generation protocols considered are probabilistic. To avoid situations in which they do not finish within the coherence time, we impose a *GHZ cycle time* after which the GHZ generation protocol is aborted. To be able to deal with coherence times and the GHZ cycle time, we assign an internal time variable to each network node. At the end of each time step, we synchronize the time in all nodes and add memory decoherence associated with waiting times. This corresponds to the protocol only moving to the next time step when all operations in the current time step are finished. On top of that, if the GHZ state is successfully created before the GHZ cycle time, we add memory decoherence until the GHZ cycle time is reached.

During normal execution of a protocol recipe, we say that we are in "execution mode". If, in execution mode, a distillation operation is unsuccessful, the target state of the distillation operation has to be recreated from scratch. In that case, we have to track back to an earlier stage in the protocol. If we started creating the state in the same time block as where the distillation failure occurred, we can simply try to recreate the state without notifying the nodes outside the time block. If this is not the case, and we have to move back to an earlier time step, all network nodes are notified, and the protocol resets to the earlier time step. This process also involves removing all states that "sit in the way" for recreating the failed state—*i.e.*, that use memory qubits required for regenerating this state. On the other hand, all states on memory qubits that are not needed are kept, so that, when we return to the point where the distillation failure occurred, these states are still there and can be used in the remainder of the protocol. This also includes distillation operations that are only partially carried out.

If it is necessary to move back to an earlier time step after a distillation failure, we store the time t_{fail} at which the measurement result was known, and enter "reconstruction mode". In reconstruction mode, we (re)apply all operations

in this time step until t_{fail} is reached—*i.e.*, if they were already simulated before the distillation operation failed, we make sure all probabilistic operations get the same outcome as in the original execution. In this mode, also operations in other time blocks are carried until t_{fail}. We then move back to an earlier time step of the protocol recipe in execution mode in order to reconstruct the failed state(s)—*i.e.*, we move back to the "failure-reset-level" associated with this distillation operation.

As soon as we have successfully reconstructed the state(s) associated with the failed distillation operation(s), we proceed with the rest of the protocol. This means that, in execution mode, we always have to make use of a data structure that indicates which operations need to be carried and which operations need to be skipped. This is necessary to correctly deal with the recreation of states that suffered from failed distillation earlier in the execution process. For example, after moving back to a failure-reset-level, only operations needed for recreating the state of the failed distillation step have to be reapplied.

Specifically, we make use of the objects l_{recreate} and l_{clear} to keep track of which operations need to be applied in execution mode. We only apply operations contained in the last object of l_{recreate}, and skip all other operations that we encounter. The structure l_{clear} contains information on when the last object in l_{recreate} can be removed—*e.g.*, if we have successfully recreated a state with a distillation operation that failed at an earlier stage of the protocol, we remove the associated object from l_{recreate}. The second-to-last object in l_{recreate} then becomes the last object and determines which operations are executed and skipped. The structure l_{recreate} starts out empty—for an empty l_{recreate} all operations are applied in execution mode.

In reconstruction mode, we deterministically reapply all time blocks $\{b_i^{(\tau')}\}_{i=1}^{B_{\tau'}}$ in this time step τ' until they reach time t_{fail}. Note that, because a distillation operation contains a set of operations that need to be carried out locally in the network nodes—and not necessarily at the same times in each node—it could occur that only part of a full distillation operation is executed before t_{fail}. If this is the case, we typically finish the rest of the operation when we reach this stage in the protocol again after fully recreating the failed state. Every time we enter reconstruction mode, an empty list l_{skip} is initialized. In this list, we collect all operations in the time step that are skipped because they fall outside the time t_{fail}.

When, in reconstruction mode, the end of the time step is reached, we calculate a list l_{reset} with all operations that have to be reapplied because of failed operation(s) in the last time step. This list is calculated

using Alg. 1. It contains all operations in the sub-trees of the failed distillation operation(s). On top of that, it can occur that one or more of their parents are also executed at this point. In that case, these parents also need to be reapplied to recover the state at the end of this time step. To evaluate what parent operations need to be reapplied, we keep track of a "real-time" dictionary δ'_{qubits} that describes what binary tree states currently sit on which network node qubits. On top of that, we also add all operations in l_{skip} to l_{reset}, but we exclude SWAP operations that act on states not contained in l_{reset}. Lastly, we also make sure to recreate states that are present on network node qubits that need to be empty to reapply the operations in l_{reset}: these states—including their sub-trees and (possibly) parents—are also added to l_{reset}. We define the failure-reset-level as the operation $o_{\text{frl}} \in l_{\text{reset}}$ that appears the first in the protocol recipe. We add l_{reset} to the end of l_{recreate} and store information in l_{clear} that tells us we can remove this list from l_{recreate} as soon as we reach the end of this time step again. After that, we tell the algorithm to go back to operation o_{frl} in the protocol recipe, and continue from there.

Below, we present a more precise description of the execution of a protocol recipe in the presence of a finite GHZ cycle time t_{GHZ}:

1. We set $t_\nu = 0$ in all nodes $\nu \in \{\nu^{(i)}\}_{i=1}^n$. We create empty structures l_{recreate} and l_{clear}. We select execution mode.
2. We select $\tau = 1$ and select the first time step $s^{(\tau)}$. We set $i = 1$ and select the first time block $b_i^{(\tau)}$ of time step $s^{(\tau)}$.
3. We select the first operation o in time block $b_i^{(\tau)}$.
4. The operation o is only applied if it is contained in the last list of l_{recreate} or if l_{recreate} is empty. Without loss of generality, we assume that o performs operations in network nodes $N_o = \{\mu_j^{(o)}\}_j$ with $t_{\mu_1^{(o)}} \leq t_{\mu_2^{(o)}} \leq \ldots$, for $\mu_j^{(o)} \in \{\nu^{(i)}\}_{i=1}^n$.
   a. If, for any of the involved nodes $\mu \in N_o$, $t_\mu \geq t_{\text{GHZ}}$ holds *after* o would have taken place, the operations of o in node μ are not executed, and we "switch off" node μ, as it now has reached the GHZ cycle time. If all nodes $\nu \in \{\nu^{(i)}\}_{i=1}^n$ have reached the GHZ cycle time, the full protocol is aborted.
   b. If the operation creates an entangled link between network nodes $\mu = \mu_1^{(o)}$ and $\mu' = \mu_2^{(o)}$, we set $t_\mu \leftarrow t_{\mu'}$.
   c. If we are in reconstruction mode, we check if $t_\mu \geq t_{\text{fail}}$ holds for all of the involved nodes $\mu \in N_o$. If that is the case, we skip (the part of) operation o in node μ and add o to l_{skip}.
   d. We carry out o in the nodes $\mu \in N_o$ that are not yet switched off and (in case we are in reconstruction mode) have not yet reached

t_{fail}. We add the time it takes to perform this operation to t_μ of the involved network node μ.

e. As long as o is the last element of l_{clear}, we remove the last elements of l_{clear} and l_{recreate}.

f. If o is a distillation operation and its success can be evaluated here, we do so:

   i. If the distillation operation succeeds, we proceed as if nothing happened.

   ii. If the distillation operation fails, we calculate l_{reset} and o_{frl} for just the operation o using the version of δ'_{qubits} at the current stage of the protocol and Alg. 1.

   iii. In case of a failed distillation operation, if o_{frl} is an operation in the same time block $b_i^{(\tau)}$ as o, we add o at the end of l_{clear} and add l_{reset} at the end of l_{recreate}. We then set o to o_{flr} and move back to the beginning of step 4.

   iv. In case of a failed distillation operation, if o_{frl} is in a different time block as $b_i^{(\tau)}$, and we are not in reconstruction mode, we set t_{fail} to the time when the full measurement result of o was known. We create a list l_{fail} and add o to this list. The list l_{fail} contains failed distillation operations in the current instance of reconstruction mode. We set l_{skip} to the empty list. We reset l_{recreate}, l_{clear} and δ'_{qubits} to their values at the start of this time step τ, set $i = 1$ to select the first time block $b_i^{(\tau)}$ of this time step τ. We enter reconstruction mode and move back to step 3.

   v. In case of a failed distillation operation, if o_{frl} is in a different time block as $b_i^{(\tau)}$, and we are already in reconstruction mode, we check if o is in l_{fail}. If that is the case, we proceed as if nothing happened. If that is not the case and the time at which the full measurement result of o was known is smaller then t_{fail}, we reset t_{fail} to this earlier time, add o to l_{fail}, reset l_{recreate}, l_{clear} and δ'_{qubits} to their values at the start of this time step τ, set i back to $i = 1$ to select the first time block $b_i^{(\tau)}$ of this time step τ, and move back to step 3. If o is not in l_{fail}, but the time at which the full measurement result of o was known is bigger than t_{fail}, we add o to l_{fail} and proceed as if nothing happened.

If o is not the last operation in $b_i^{(\tau)}$, we select the next operation in $b_i^{(\tau)}$ as the new operation o and move back to beginning of step 4. If o is the last operation in $b_i^{(\tau)}$ and $i < B_\tau$, we increase i by one and move back to step 3. If o is the last operation in $b_i^{(\tau)}$ and $i = B_\tau$ holds, we move to step 5.

5. We have reached the end of the time step. As long as this time step is the last element of l_{clear}, we remove the last elements of l_{clear} and l_{recreate}. Then, if we are not in reconstruction mode, we move to step 6. Otherwise—*i.e.*, if we *are* in reconstruction mode—we add this time step at the end of l_{clear}. On top of that, we calculate l_{reset} and o_{frl} with Alg. 1, using l_{fail} as the list of operations in Alg. 1. Lastly, we add each operation $o'' \in l_{\text{skip}}$ to l_{reset} as well. Here, we exclude operations $o'' \in l_{\text{skip}}$ that are SWAP gates swapping a state that is not contained in l_{reset}. We enter execution mode, set o to o_{frl} and move back to step 4.
6. We evaluate the distillation operations $o'' \in l_{\text{eval}}^{(\tau)}$. Here, if l_{recreate} is not empty, we skip distillation operations that are not part of the last list in l_{recreate}.
   a. As soon as one distillation operation o'' fails, we add this location in the protocol recipe at the end of l_{clear}, calculate l_{reset} and o_{frl} with Alg. 1 for just operation o'', and add l_{reset} at the end of l_{recreate}. We set o to o_{frl} and move back to step 4.
   b. If all distillation evaluations succeed, we move to step 7.
7. We evaluate and carry out the fusion corrections in $l_{\text{corr}}^{(\tau)}$.
8. If $\tau < T$ holds, we increase τ by one, set $i = 1$, set t_ν to $\max_{\nu'} t_{\nu'}$ for all $\nu, \nu' \in \{\nu^{(i)}\}_{i=1}^n$, and move to step 3. If $\tau = T$ holds, we set t_ν to $\max_{\nu'} t_{\nu'}$ for all $\nu, \nu' \in \{\nu^{(i)}\}_{i=1}^n$ and stop the protocol.

References

1. Aschauer, H., Dür, W., Briegel, H.J.: Multiparticle entanglement purification for two-colorable graph states. Physical Review A **71**(1), 012319 (2005)
2. de Bone, S., Ouyang, R., Goodenough, K., Elkouss, D.: Protocols for Creating and Distilling Multipartite GHZ States With Bell Pairs. IEEE Transactions on Quantum Engineering **1**, 1–10 (2020). https://doi.org/10.1109/TQE.2020.3044179
3. Deutsch, D., Ekert, A., Jozsa, R., Macchiavello, C., Popescu, S., Sanpera, A.: Quantum privacy amplification and the security of quantum cryptography over noisy channels. Physical Review Letters **77**(13), 2818–2821 (1996). https://doi.org/10.1103/PhysRevLett.77.2818
4. Goyal, K., McCauley, A., Raussendorf, R.: Purification of large bi-colorable graph states. Phys. Rev. A **74**(3), 032318 (Sep 2006). https://doi.org/10.1103/PhysRevA.74.032318
5. Maneva, E.N., Smolin, J.A.: Improved two-party and multi-party purification protocols. Contemporary Mathematics **305**, 203–212 (2002)

Algorithm 1: Pseudo-code used to identify the failure-reset-level and a list of operations that need to be re-applied in case of a failed distillation attempt.

Data: l_{d}: list of operations that need to be re-applied
$\delta'_{\mathrm{qubits}}$: dictionary with states currently stored on qubits
Binary tree of the protocol
Protocol recipe
Result: Failure-reset-level of operations in l_{d}
List of operations that have to be reapplied

$l_{\mathrm{reset}} \leftarrow \varnothing$
while $l_{\mathrm{d}} \neq \varnothing$ **do**
  for $o \in l_{\mathrm{d}}$ **do**
    Add o to l_{reset}.
    Add all binary tree children of o to l_{reset}.
    Add all binary tree parents of o with identification number i_{op} stored in $\delta'_{\mathrm{qubits}}$ to l_{reset} (and all their children).
  Identify all operations l_{occ} that sit on qubits in $\delta'_{\mathrm{qubits}}$ that need to be empty if we want to reapply the operations in l_{reset}.
  $l_{\mathrm{d}} \leftarrow l_{\mathrm{occ}}$
Identify the first operation $o_{\mathrm{frl}} \in l_{\mathrm{reset}}$ in the protocol recipe.
return o_{frl}, l_{reset}.