APPENDIX A

Extra Information

A.1 Ion Routing Algorithm Pass Description

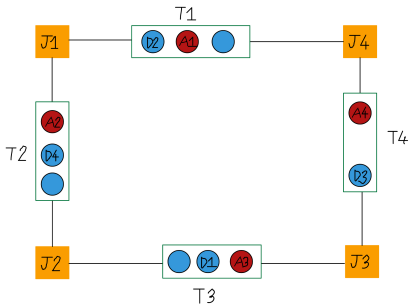

Figure A.1: Setup of the ion routing problem: 4 traps (T1 to T4, each with capacity 4), 4 junctions (J1 to J4 as orange squares), and connecting segments.

During each pass, the algorithm first identifies which gates can be executed without requiring ion movement and are not blocked by gates needing ion routing. These gates are added to the output schedule.

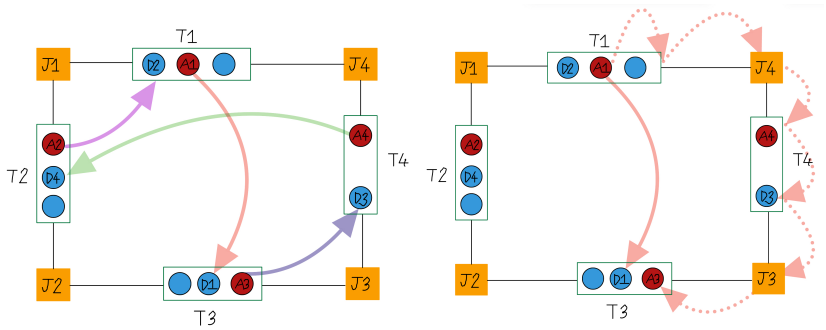

Figure A.2: (Left) Routing destinations for ancilla qubits marked for their next operation. (Right) The routing algorithm finds a path for ancilla A1 to destination D1, where a dotted red arrow denotes each edge. In the path, an edge corresponds to a sequence of ion reconfiguration primitives. For example, the first edge in the path, marked with red dotted arrows from A1 to D1, represents a high-level SWAP operation within trap T1 to move A1 to the trap's edge (requiring 3 MS gates), followed by a split operation to move A1 into segment T1 to J4.

Next, determine the routing destinations for each ancilla qubit based on their next operation. Figure A.2 (left) shows the destinations of the four ancilla qubits, with their

routes marked. The algorithm sequentially allocates paths for as many ancilla qubits as possible , prioritising ancillas required earlier in the input sequence. For instance, ancilla A1, which must move to destination D1 (marked with a red arrow), is given the highest priority. Figure A.2 (right) illustrates the algorithm finding a path for A1 to D1 in the graph described in Section **??**.

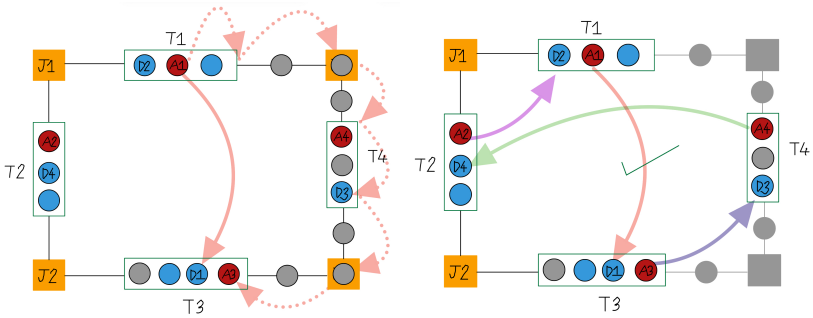

Figure A.3: (Left) Allocation of one extra qubit, denoted by grey circles, to each junction, segment, and trap in the routing path from A1 to D1, excluding the source trap. (Right) Grey shading indicates components removed from the QCCD graph due to capacity constraints or disconnection.

Next, we assume each component in the chosen routing path gains an additional qubit for the remainder of the pass. Figure A.3 (left) shows this allocation along the routing path between A1 and D1. Note that the source trap is excluded, as it does not gain a qubit during the routing sequence. If a component reaches its capacity with the added qubit, it is removed from the QCCD graph. Since segments and junctions have a capacity of 1, all segments and junctions in the routing path are removed. Figure A.3 (right) depicts the updated graph, with unavailable components shaded grey. While trap T4 is not at capacity, it is disconnected from the rest of the architecture and is therefore removed from the graph, along with the green arrow (A4 cannot be routed to D4 in T2) and the purple arrow (A1 cannot be routed to D3 in T4).

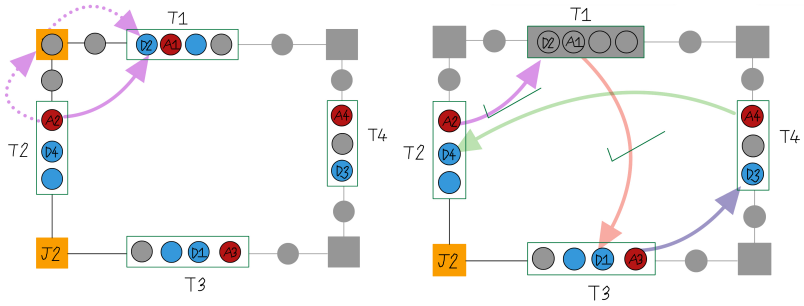

Figure A.4: (Left) Routing path for A2 to D2 with grey circles indicating allocated extra qubits along the path. (Right) Components removed from the graph due to capacity constraints, including T1.

Repeating the steps from Section A.1, ancilla A2 is served as it is the only ancilla able to reach its data qubit D2 in the remaining architecture graph. Figure A.4 (left) shows the routing path selected for A2 to D2, with grey circles representing extra qubits allocated along the path. Figure A.4 (right) illustrates the components removed from the graph for

this pass, including T1. Since no additional ancilla qubits can be served in this pass, the algorithm proceeds to the next phase.

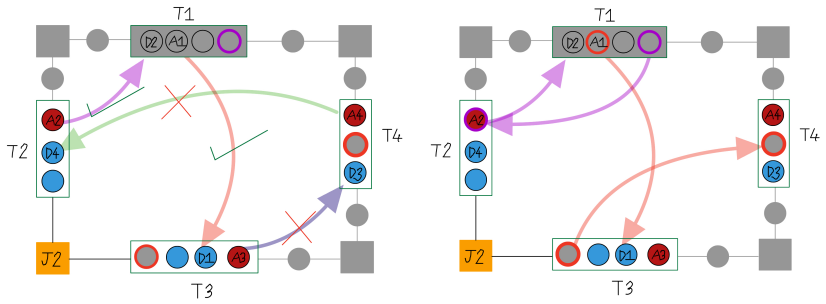

Figure A.5: (Left) T3 violates the invariant by exceeding its capacity of 4 ions. (Right) A1 is re-routed back to T4 along its original path to restore the invariant, while D2 is routed back to T2.

To preserve the invariants described in Section **??** at the end of the pass, ion reconfigurations are necessary. Figure A.5 (left) highlights that T3 violates the first invariant by containing four ions, including A1, which equals its capacity 4. Therefore, A1 must be re-routed from T3 to another trap within the routing path to restore the invariant. Since A1 has already reserved the traps along its original path, A1 is re-routed back to T1 until it reaches either a trap below capacity or the source trap. If T4 were full, A1 would reach its source trap T1, yet T1 is at capacity. However, A1 could still reside there because D2 will be re-routed back to T2, as indicated by the purple arrows in Figure A.5 (right).

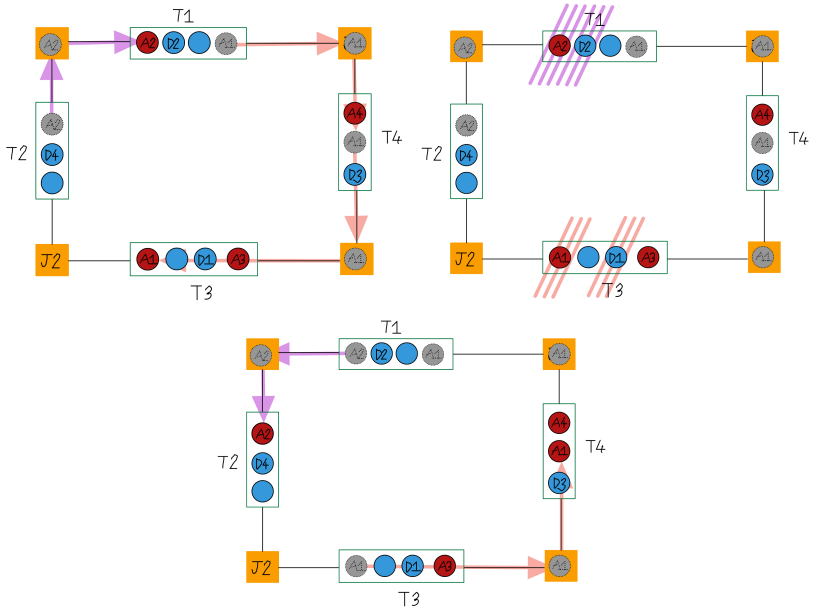

Figure A.6: Routing ancillas to their data qubits for entanglement, followed by the entanglement operation, and finally routing ancillas back to traps while preserving invariants at the end of the pass.

Once routing paths for the current pass are established (Figure A.6), ancillas are dynamically forwarded along their paths. Deadlocks are avoided because two ancillas

cannot share the same segments or junctions in a routing path. Consequently, ancillas in a junction can move to the next segment in their path without being blocked by another ion. Additionally, ions can move into traps or junctions on their paths at any time during the pass, as they reserve a spot in these components for the entire pass.

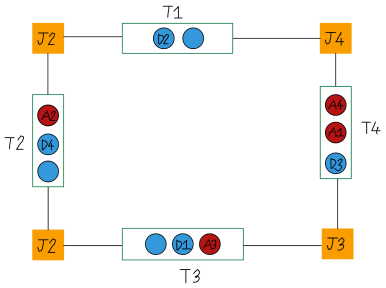

Figure A.7: Final state of the first pass showing both invariants preserved.

Figure A.7 illustrates the final state at the end of the pass. After adding the operations from the pass into the output sequence, a barrier is inserted to prevent operations in the next pass from overlapping with those in the current pass, ensuring correctness.

A.2 Generating Compact Subsets of Traps

Figure A.8 illustrates how centroid traps are determined for two different QCCD hardware configurations. Figure A.9 shows the process of exploring all possible compact subsets centred around a given centroid trap. The strategy uses a square as the expanding curve for exploration, which aligns with the square cellulation of the QCCD grid topology used in this project. This approach can be easily generalised to other hardware topologies by modifying the shape of the expanding curve.

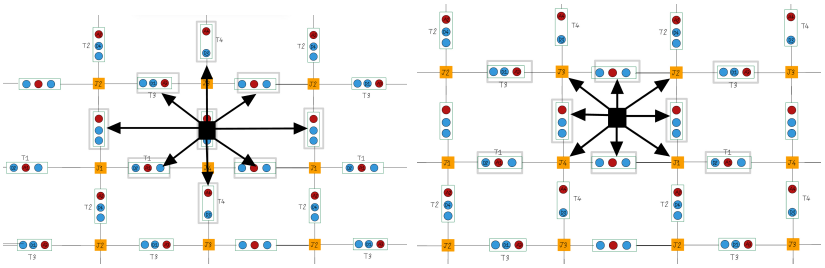

Figure A.8: Both diagrams illustrate how a centroid trap is selected, differing only in whether the hardware centre is positioned on a trap (left) or off a trap (right). Up to 9 centroid traps are considered for exploration, including the trap at the hardware centre (0,0) and traps adjacent to it (e.g., (1,0), (-1,0), (1,1), etc.). A black square represents the hardware centre, while black arrows indicate the centroid traps. The centroid traps chosen in each diagram have a grey-highlighted border. For each centroid trap, compact subsets of traps loosely centred around it are explored.

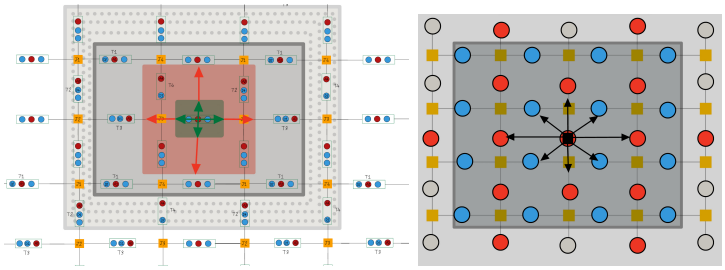

Figure A.9: The first diagram demonstrates the expanding curve strategy for determining compact subsets of traps loosely centred around a centroid trap, highlighted by a green box. Traps guaranteed to be part of any compact subset are found by starting with all traps within a square of length 1 centred at the centroid trap (green box), then expanding the square to length 2 (red box), and continuing until the square's expansion would exceed the number of qubit clusters. The dark grey region represents guaranteed traps, while the light grey dotted region represents candidate traps on the outer edges of the expanding square. All possible compact subsets are explored by including all guaranteed traps and some candidate traps, ensuring the subset cardinality equals the number of qubit clusters. This method is efficient because the number of compact subsets is polynomial in the number of candidate traps, rather than the total number of traps in the topology. The second diagram highlights the guaranteed traps (dark grey) and candidate traps (light grey) for mapping single qubit clusters to traps with capacity 2. The first diagram uses traps with capacity 4, applicable to qubit clusters with a maximum cluster size of 3.