TestGreedyInputNodeSelection

March 2, 2025

[1]:
```
from qiskit import QuantumCircuit, QuantumRegister

from rustworkx.visualization import graphviz_draw
from rustworkx.rustworkx import simple_cycles
from collections import Counter
```

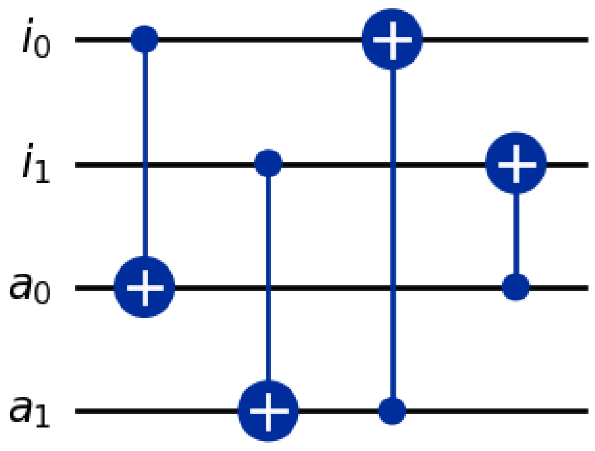
[2]:
```
import os, sys

sys.path.append(os.path.abspath("../"))

num_i = 2
num_a = 2
```

[3]:
```
from helperfunctions.reversecircuitgraph import reverse_all_operations
from helperfunctions.circuitgraphfunctions import get_computation_graph,␣
  ↪get_uncomp_circuit
from helperfunctions.graphhelper import breakdown_qubit, edge_attr, node_attr
from helperfunctions.uncompfunctions import add_uncomputation

from helperfunctions.constants import StringConstants
```

[4]:
```
def small_example_circuit():
    i = QuantumRegister(num_i, 'i')
    # o = QuantumRegister(num_o, 'o')
    a = QuantumRegister(num_a, 'a')

    circ = QuantumCircuit(i,a)

    circ.cx(i[0],a[0])
    circ.cx(i[1],a[1])

    circ.cx(a[1],i[0])
    circ.cx(a[0],i[1])

    return circ
```

[5]:
```
circc = small_example_circuit()


circ.draw('mpl')
```

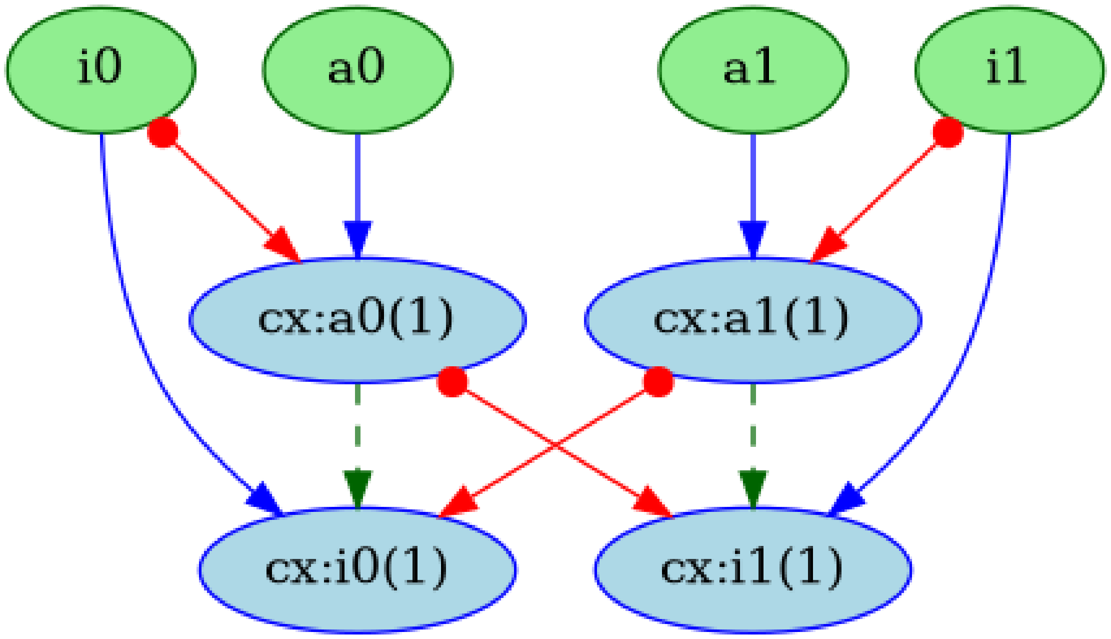
[5]:



[6]:
```
qubits =  [breakdown_qubit(q)['label'] for q in circ.qubits]

# ancillas_list = qubits[:num_i] + qubits[-num_a:]
ancillas_list = qubits[-num_a:]

# output_list = qubits[num_i:num_i+num_o]
print(ancillas_list)
# print(output_list)

cg = get_computation_graph(circ, ancillas_list, outputs=None)

print(cg.nodes())

graphviz_draw(cg,
                     node_attr_fn=node_attr,
                     edge_attr_fn=edge_attr)
```

```
['a0', 'a1']

Adding Nodes for Circuit: 100%|      | 4/4 [00:00<00:00, 27639.57it/s]

[CGNode: Labeled i0 @ index: 0 of type input is a initialize node.
, CGNode: Labeled i1 @ index: 1 of type input is a initialize node.
, CGNode: Labeled a0 @ index: 2 of type ancilla is a initialize node.
, CGNode: Labeled a1 @ index: 3 of type ancilla is a initialize node.
, CGNode: Labeled a0 @ index: 4 of type ancilla is a computation node.
, CGNode: Labeled a1 @ index: 5 of type ancilla is a computation node.
, CGNode: Labeled i0 @ index: 6 of type input is a computation node.
, CGNode: Labeled i1 @ index: 7 of type input is a computation node.
]
```

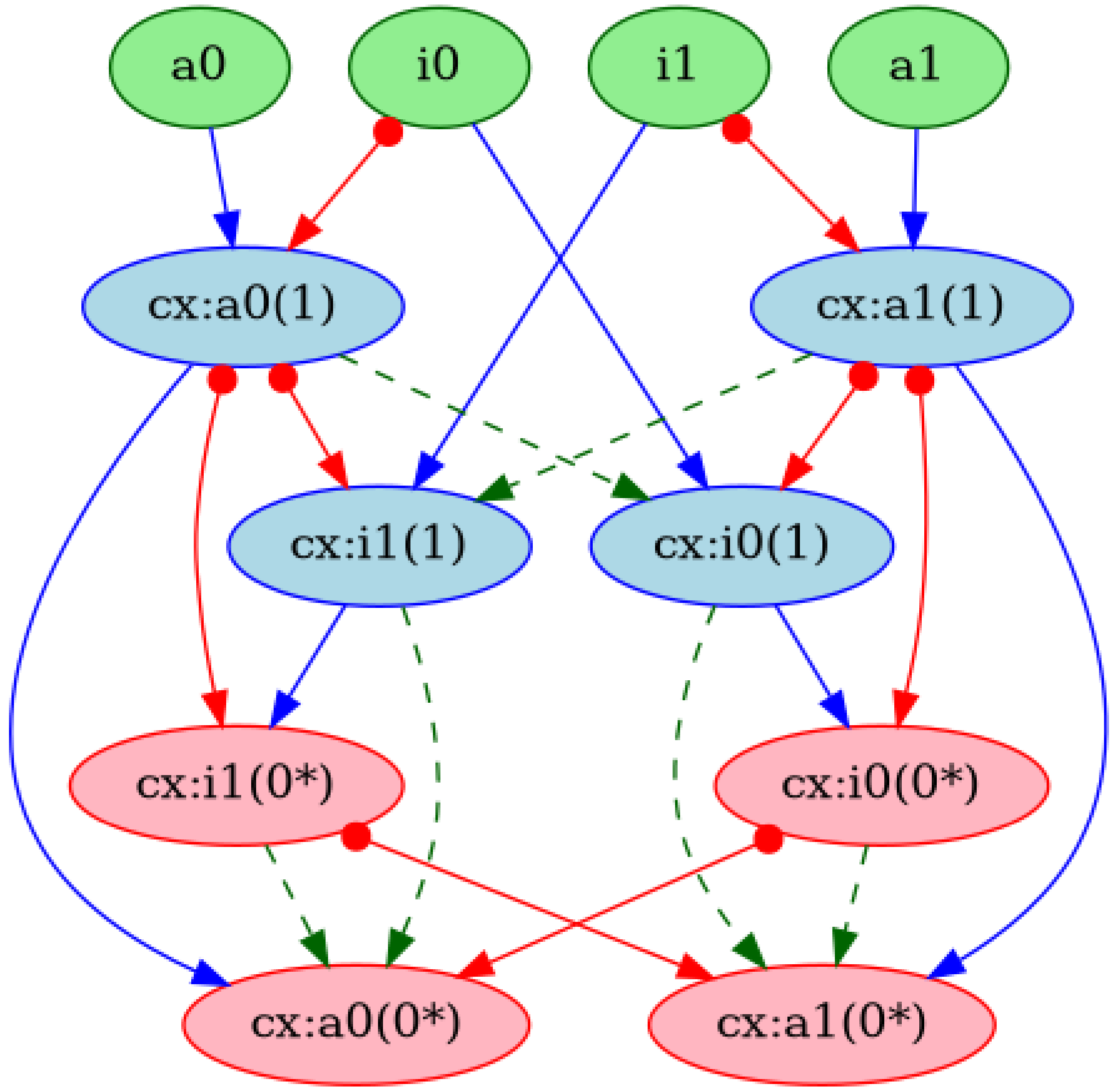
[6]:



[7]:
```
all_uncomp_cg = reverse_all_operations(cg)
print(all_uncomp_cg.nodes())
graphviz_draw(all_uncomp_cg,
                        node_attr_fn=node_attr,
                        edge_attr_fn=edge_attr)
```

```
[CGNode: Labeled i0 @ index: 0 of type input is a initialize node.
, CGNode: Labeled i1 @ index: 1 of type input is a initialize node.
, CGNode: Labeled a0 @ index: 2 of type ancilla is a initialize node.
, CGNode: Labeled a1 @ index: 3 of type ancilla is a initialize node.
, CGNode: Labeled a0 @ index: 4 of type ancilla is a computation node.
, CGNode: Labeled a1 @ index: 5 of type ancilla is a computation node.
, CGNode: Labeled i0 @ index: 6 of type input is a computation node.
, CGNode: Labeled i1 @ index: 7 of type input is a computation node.
, CGNode: Labeled i0 @ index: 8 of type input is a uncomputation node.
, CGNode: Labeled i1 @ index: 9 of type input is a uncomputation node.
, CGNode: Labeled a0 @ index: 10 of type ancilla is a uncomputation node.
, CGNode: Labeled a1 @ index: 11 of type ancilla is a uncomputation node.
]
```

[7]:



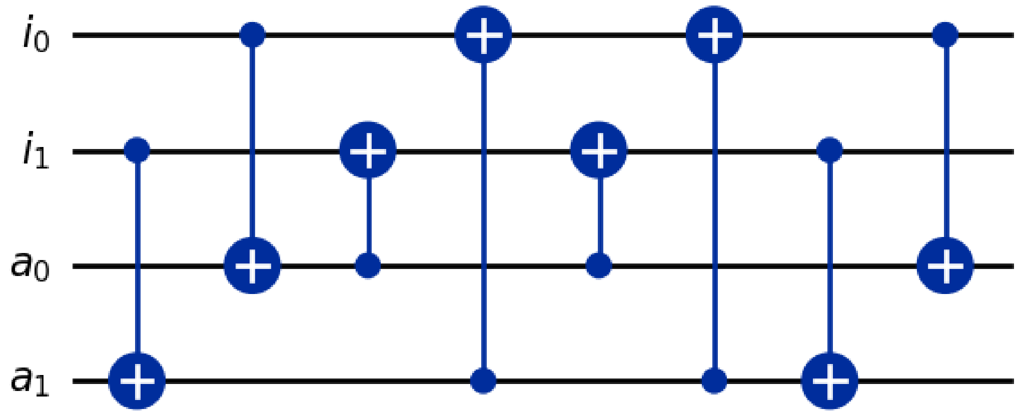
[8]:

```
uncomp_circ = get_uncomp_circuit(all_uncomp_cg)
print(sum(uncomp_circ.count_ops().values()))
uncomp_circ.draw('mpl')
```

```
Building uncomp circuit from circuit graph: 100%|      | 12/12 [00:00<00:00,
63872.65it/s]
```

```
8
```

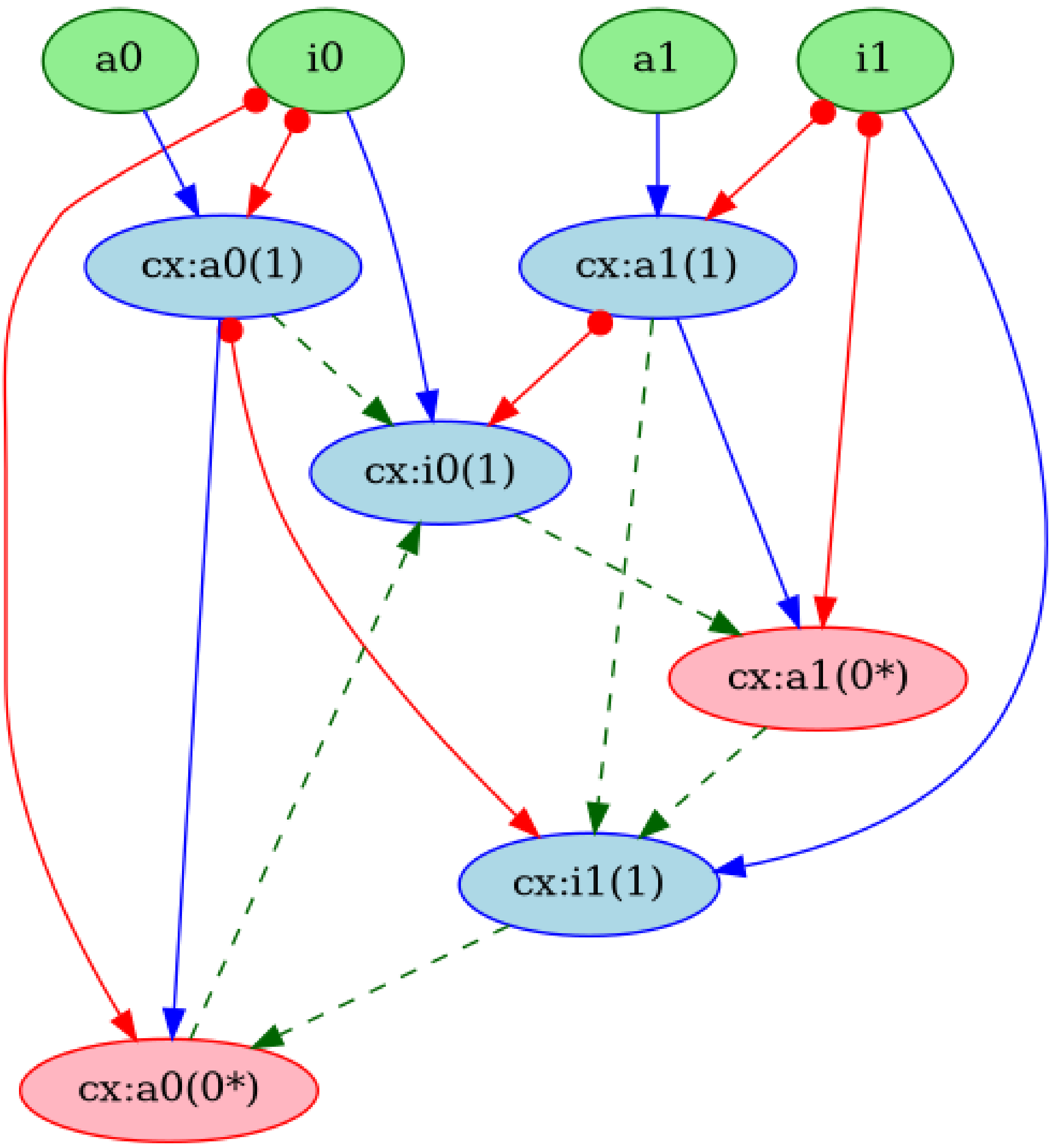
[8]:



[9]:
```
pldi_cyclic_uncomp_cg, has_cycles = add_uncomputation(cg, ancillas_list,␣
  ↪allow_cycle=True)
print(has_cycles)
print(pldi_cyclic_uncomp_cg.nodes())
graphviz_draw(pldi_cyclic_uncomp_cg,
                          node_attr_fn=node_attr,
                          edge_attr_fn=edge_attr)
```

```
False
[CGNode: Labeled i0 @ index: 0 of type input is a initialize node.
, CGNode: Labeled i1 @ index: 1 of type input is a initialize node.
, CGNode: Labeled a0 @ index: 2 of type ancilla is a initialize node.
, CGNode: Labeled a1 @ index: 3 of type ancilla is a initialize node.
, CGNode: Labeled a0 @ index: 4 of type ancilla is a computation node.
, CGNode: Labeled a1 @ index: 5 of type ancilla is a computation node.
, CGNode: Labeled i0 @ index: 6 of type input is a computation node.
, CGNode: Labeled i1 @ index: 7 of type input is a computation node.
, CGNode: Labeled a0 @ index: 8 of type ancilla is a uncomputation node.
, CGNode: Labeled a1 @ index: 9 of type ancilla is a uncomputation node.
]
```

[9]:



[10]:
```
ANCILLA = StringConstants.ANCILLA.value

pldi_simple_cycles = simple_cycles(pldi_cyclic_uncomp_cg)
ancilla_counter = Counter(ancillas_list)
ancilla_counter.subtract(ancillas_list)
print(ancilla_counter)

for cycle in pldi_simple_cycles:
    print([pldi_cyclic_uncomp_cg.get_node_data(i).simple_graph_label() for i in␣
 ↪cycle])
```

```
    for i in cycle:
        node = pldi_cyclic_uncomp_cg.get_node_data(i)
        if node.qubit_type is ANCILLA:
            ancilla_counter[f'{node.qubit_name}{node.qubit_wire}'] += 1

print(ancilla_counter)
```

```
Counter({'a0': 0, 'a1': 0})
['cx:a0(0*)', 'cx:i0(1)', 'cx:a1(0*)', 'cx:i1(1)']
Counter({'a0': 1, 'a1': 1})
```

[11]:
```
from helperfunctions.reversecircuitgraph import reverse_input_qubits

reverse_input_qubits(pldi_cyclic_uncomp_cg)
```

```
CGNode: Labeled i0 @ index: 6 of type input is a computation node.
CGNode: Labeled i1 @ index: 7 of type input is a computation node.
{'i0': [CGNode: Labeled i0 @ index: 6 of type input is a computation node.
], 'i1': [CGNode: Labeled i1 @ index: 7 of type input is a computation node.
]}
```