Problem E:
Why is this cable not longer?

Von Ala Eddine Ben Yahya, Julius Schmück

The Problem

Adam just moved into his new apartment and simply placed everything into it at random. This means in particular that he did not put any effort into placing his electronics in a way that each one can have its own electric socket.

Since the cables of his devices have limited reach, not every device can be plugged into every socket without moving it first. As he wants to use as many electronic devices as possible right away without moving stuff around, he now tries to figure out which device to plug into which socket. Luckily the previous owner left behind a plugbar which turns one electric socket into 3.

Can you help Adam figure out how many devices he can power in total?

Input

```
3 6 8
1 1
1 2
1 3
2 3
2 4
3 4
3 5
3 6
```

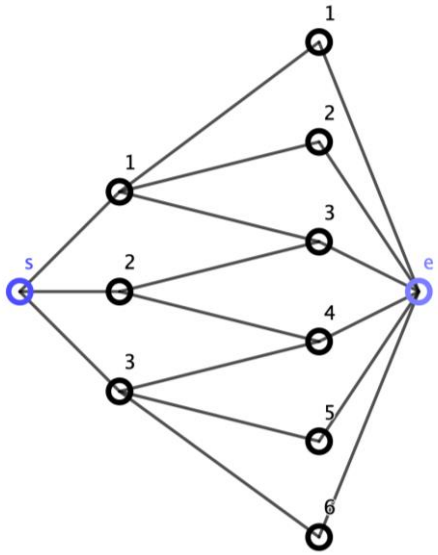
Die erste Zeile gibt die Anzahl an

- Steckdosen m ($1 \leq m \leq 1\,500$)
- Geräten n ($1 \leq n \leq 1\,500$)
- möglichen Verbindungen k ($1 \leq k \leq 75\,000$)

Die weiteren k Zeilen geben die möglichen Verbindungen an,
wobei die erste Zahl für die Nummer der Steckdose
und die zweite für die des Gerätes steht.
Jeweils mit 1 angefangen durchzunummerieren

Lösungsansatz

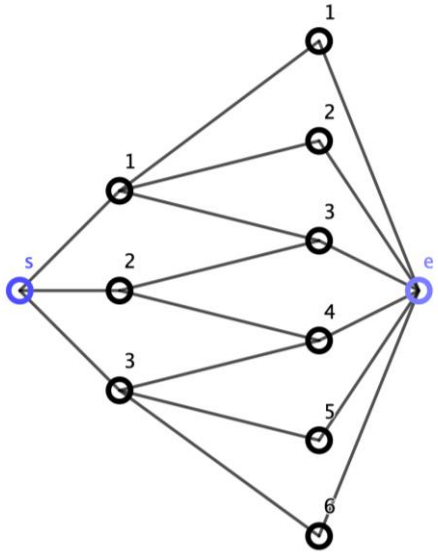
3 6 8
1 1
1 2
1 3
2 3
2 4
3 4
3 5
3 6



- Input in Graphen G umwandeln
- Startknoten s der mit einem Knoten für jede Steckdose verbunden ist
- Endknoten e der mit einem Knoten für jedes Gerät verbunden ist
- Kanten zwischen Steckdosenknoten und Geräteknoten wenn eine passende Verbindung in der Eingabe gegeben ist
- Die Kanten sind ungewichtet und in Richtung e gerichtet

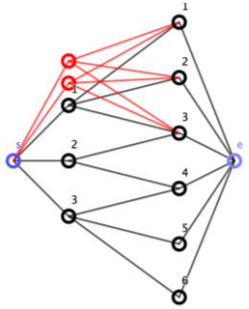
Lösungsansatz

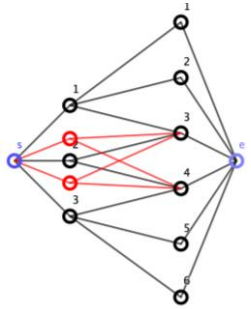
3 6 8
1 1
1 2
1 3
2 3
2 4
3 4
3 5
3 6

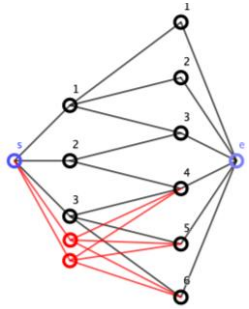


Die maximale Anzahl gleichzeitig eingesteckter Geräte = Max Flow von s nach e

Realisierung des Mehrfachsteckers indem man alle Steckdosenknoten durchgeht und für je einen zwei Duplikate einfügt und dann den Max Flow berechnet und am Ende das Maximum dieser zurückgibt







Max Flow

Viele Möglichkeiten:

- Ford–Fulkerson Algorithmus
- Edmonds–Karp Algorithmus
- Dinic’s Algorithmus

Da der Graph der sich aus dem Problem ergiebt, aber bipartite und ungewichtet ist kann man die Lösung leichter finden.

Max Flow (Option 1)

1. Nutze DFS oder BFS um einen s-e-Weg zu finden
2. Sobald ein s-e-Weg gefunden ist werden alle seine Kanten von diesem umgedreht
3. Wiederhole 1 und 2 bis es keine Verbindung mehr zwischen s und e gibt
4. Der Graph aus den umgekehrten Kanten ergibt das maximale Matching zwischen den Steckdosenknoten und den Geräteknoten
5. Aus dem maximalen Matching erhält man den Max Flow

Beispiel (Option 1)

Input

5 4 8

1 1

2 1

2 2

3 3

3 4

4 2

5 1

5 4

Beispiel (Option 1)

Input

5 4 8

1 1

2 1

2 2

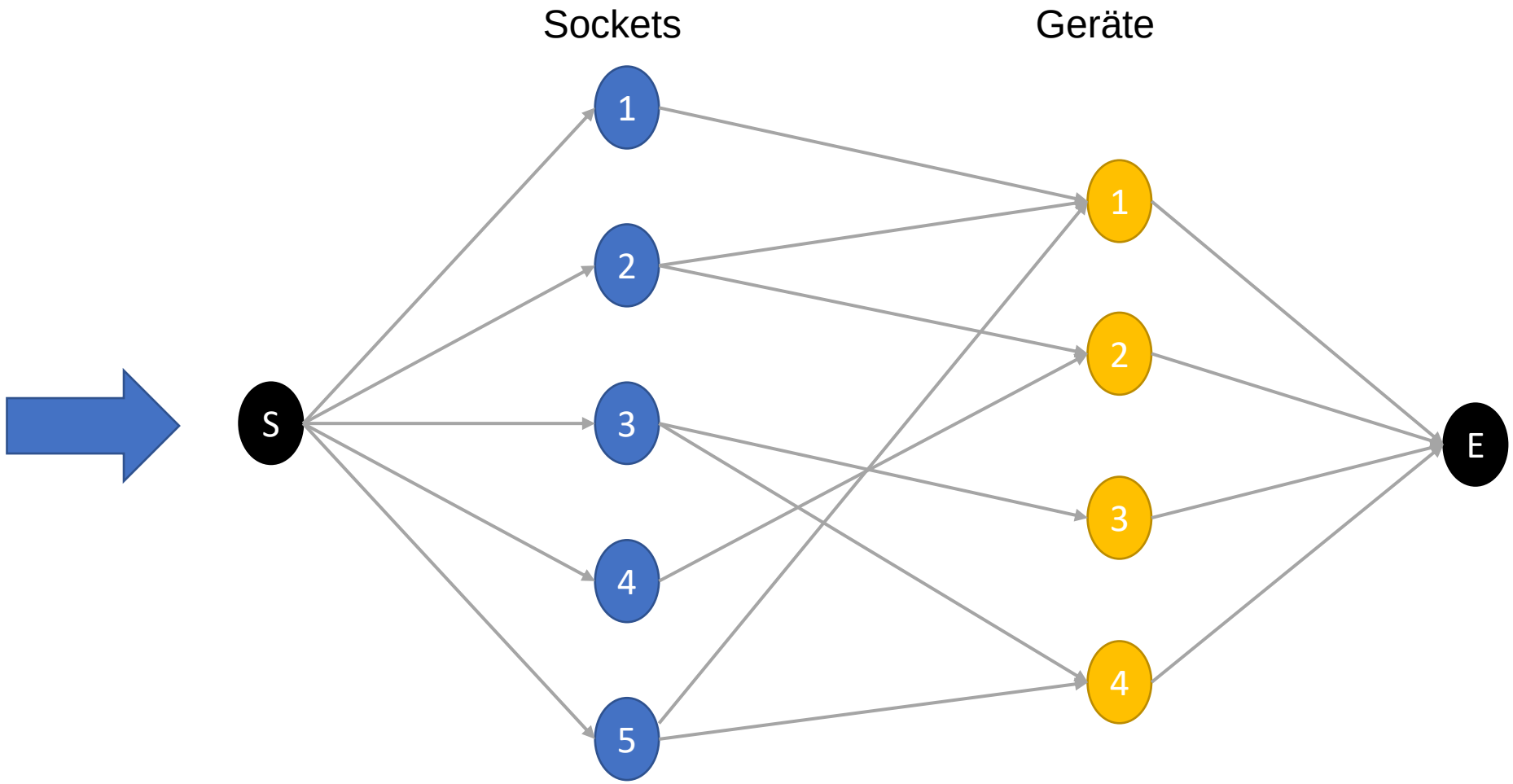
3 3

3 4

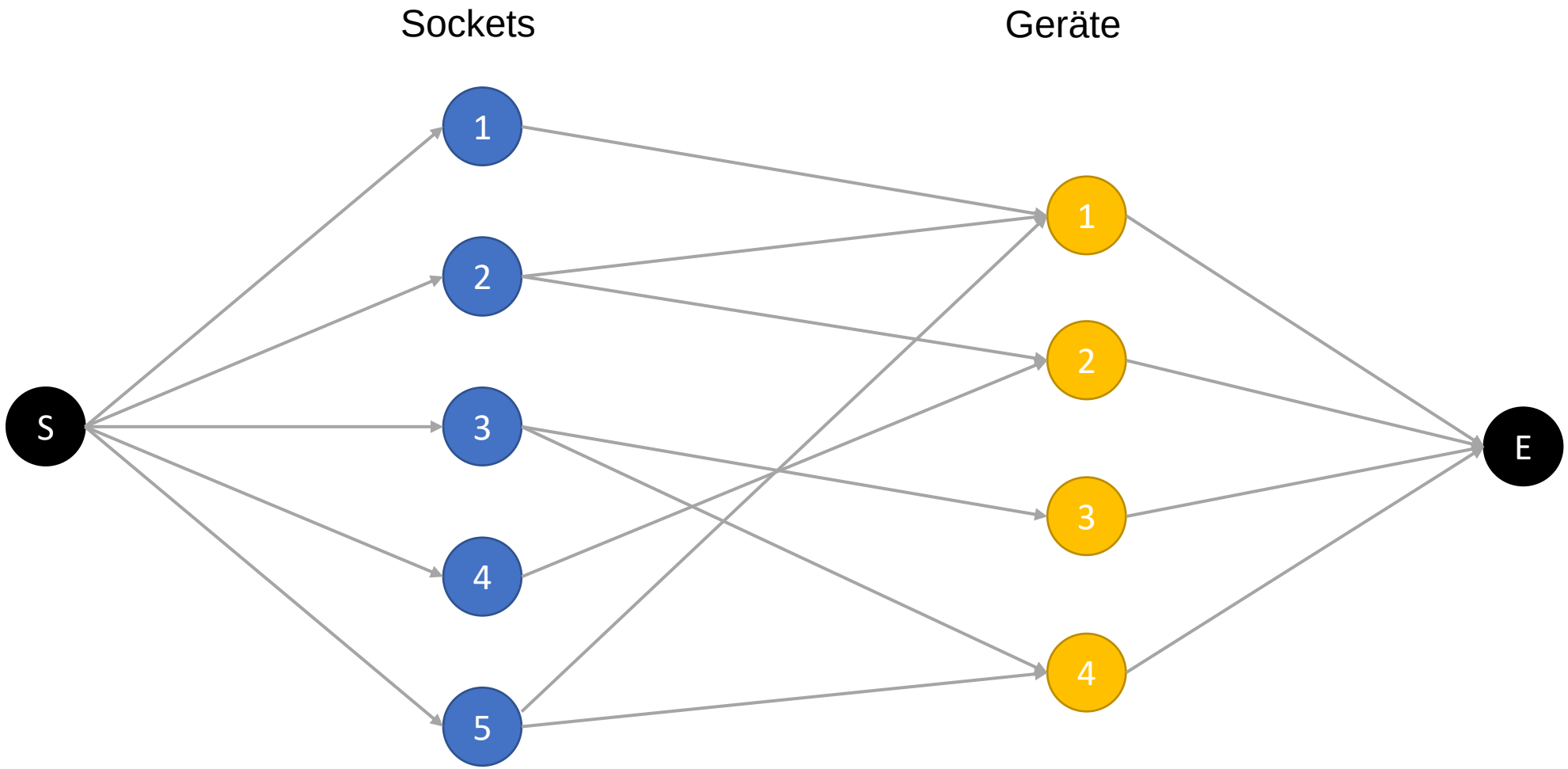
4 2

5 1

5 4

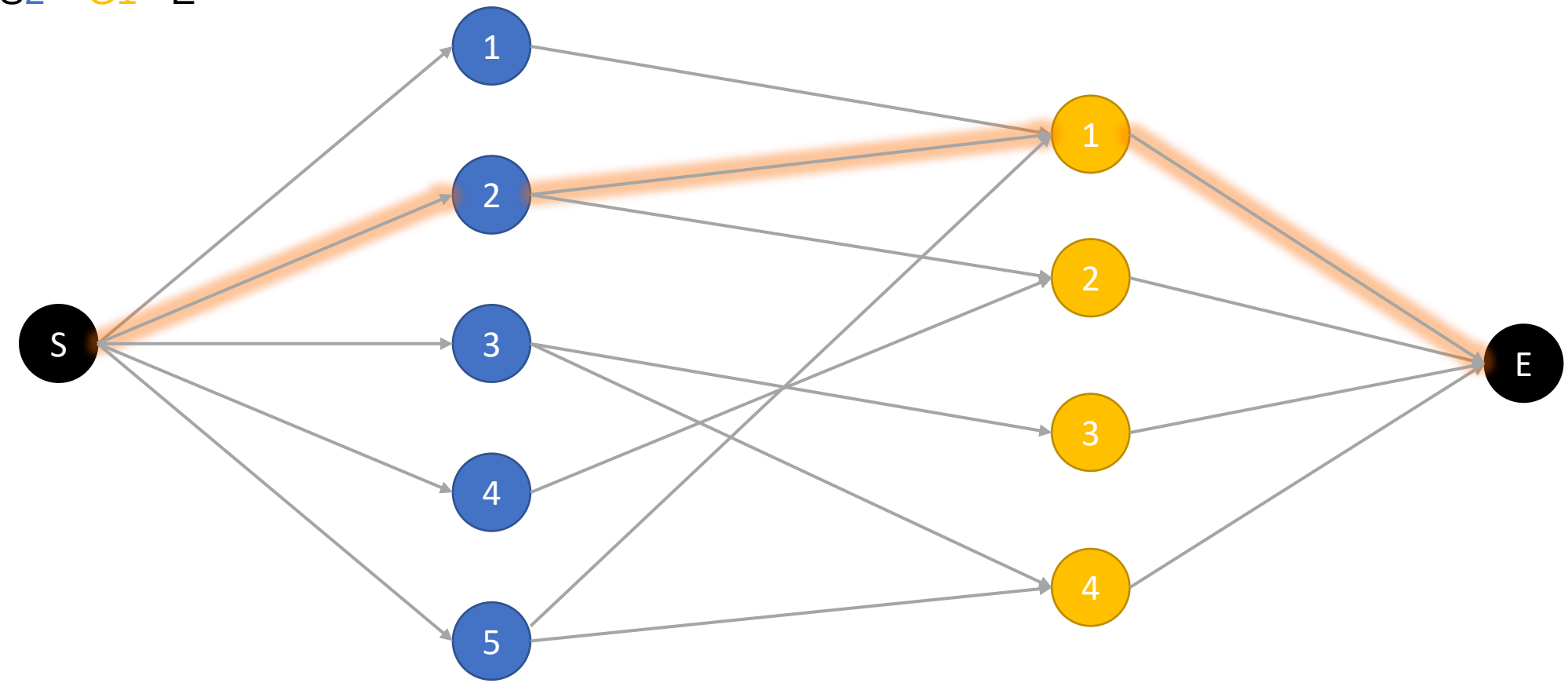


Beispiel (Option 1)



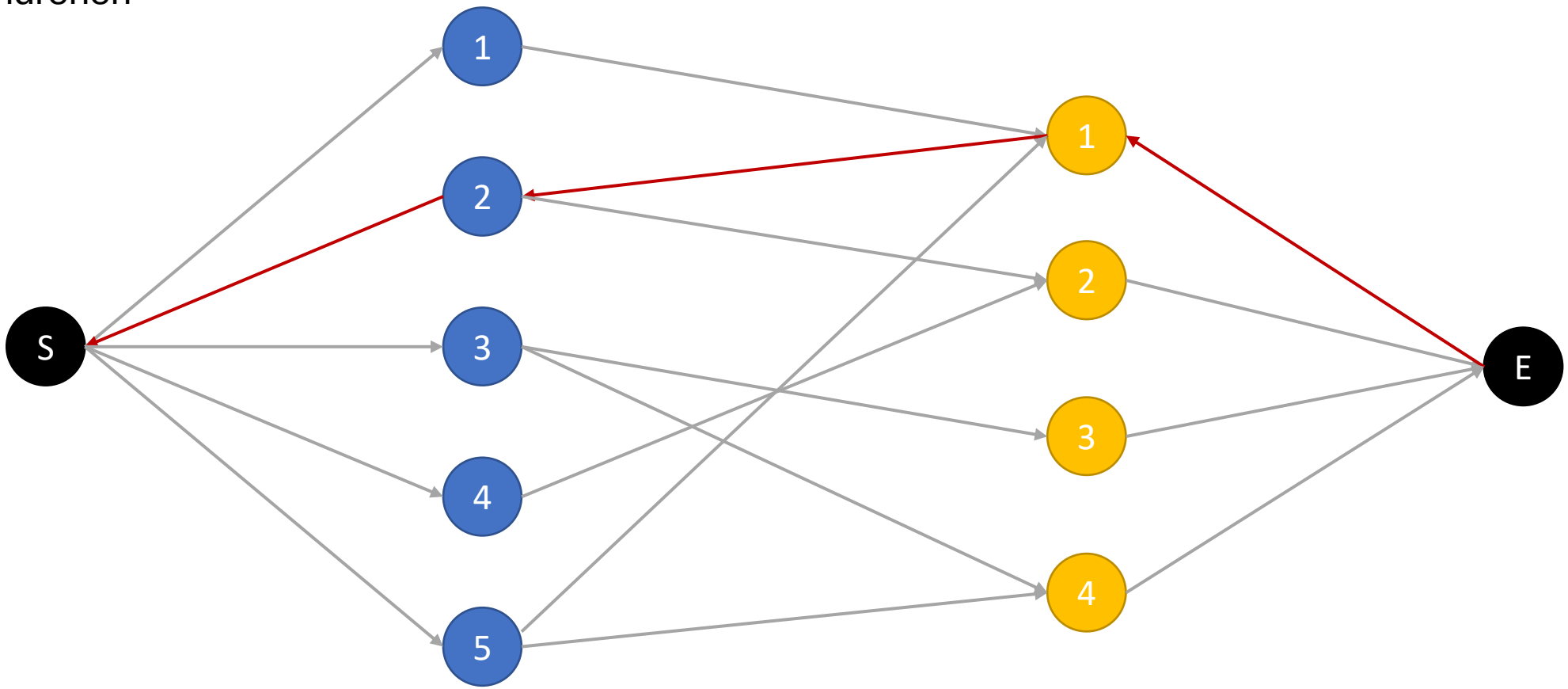
Beispiel (Option 1)

Weg S –S2 – G1 - E



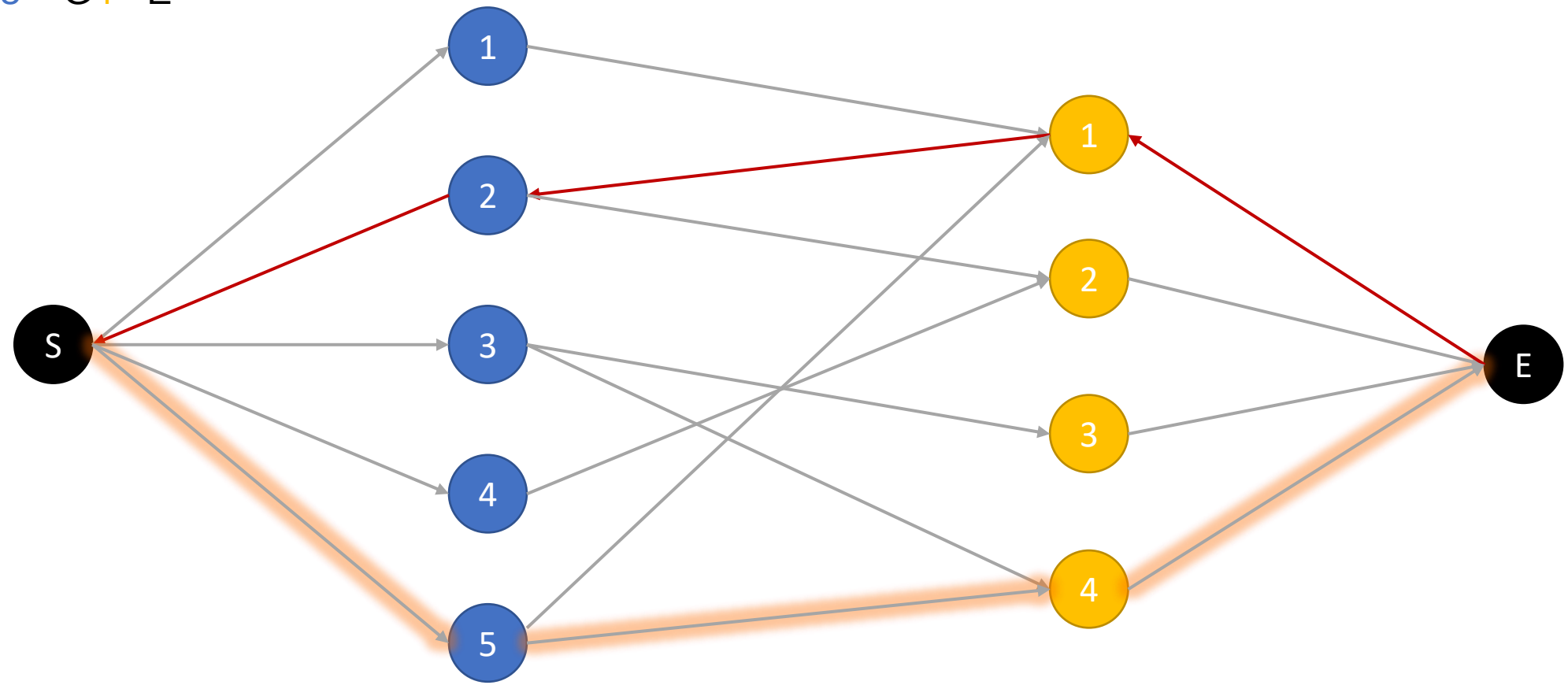
Beispiel (Option 1)

Kanten umdrehen



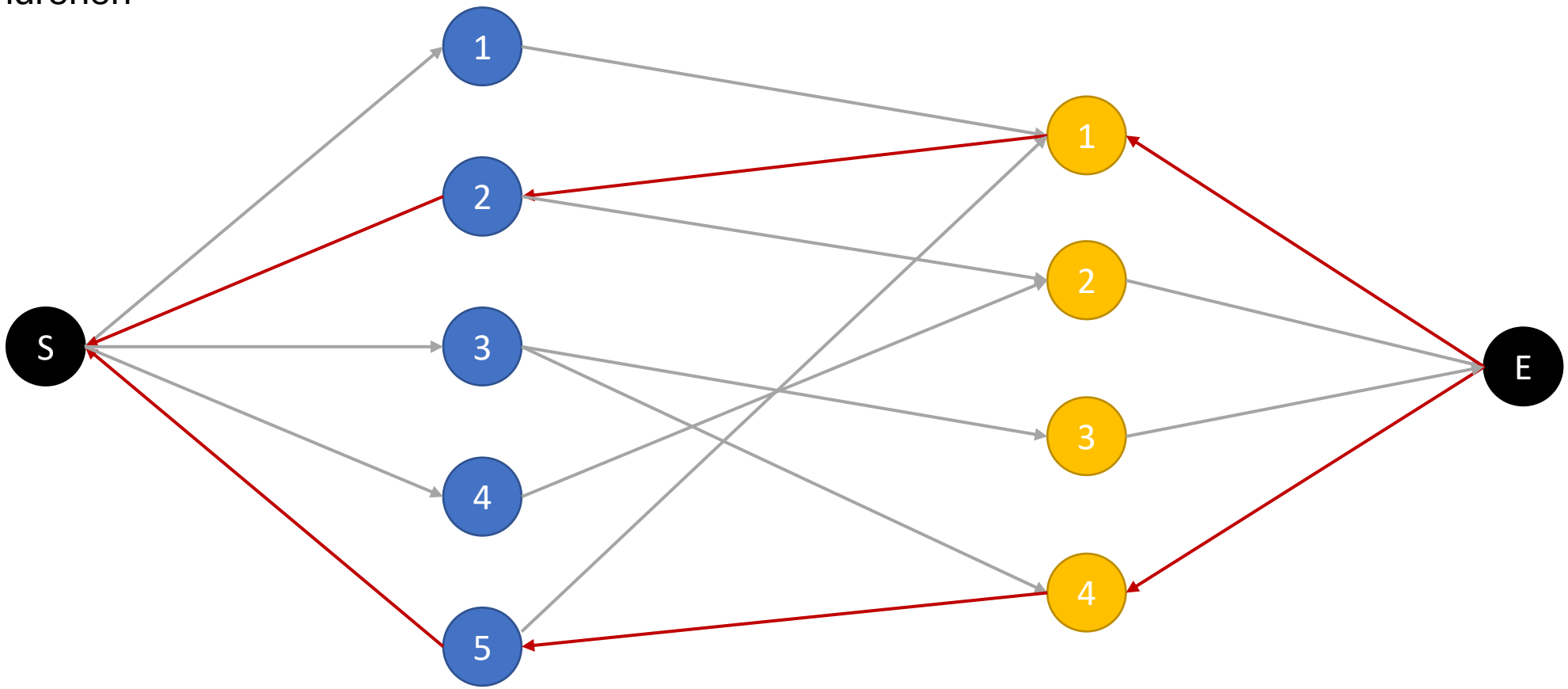
Beispiel (Option 1)

Weg S- S5 – G4 - E



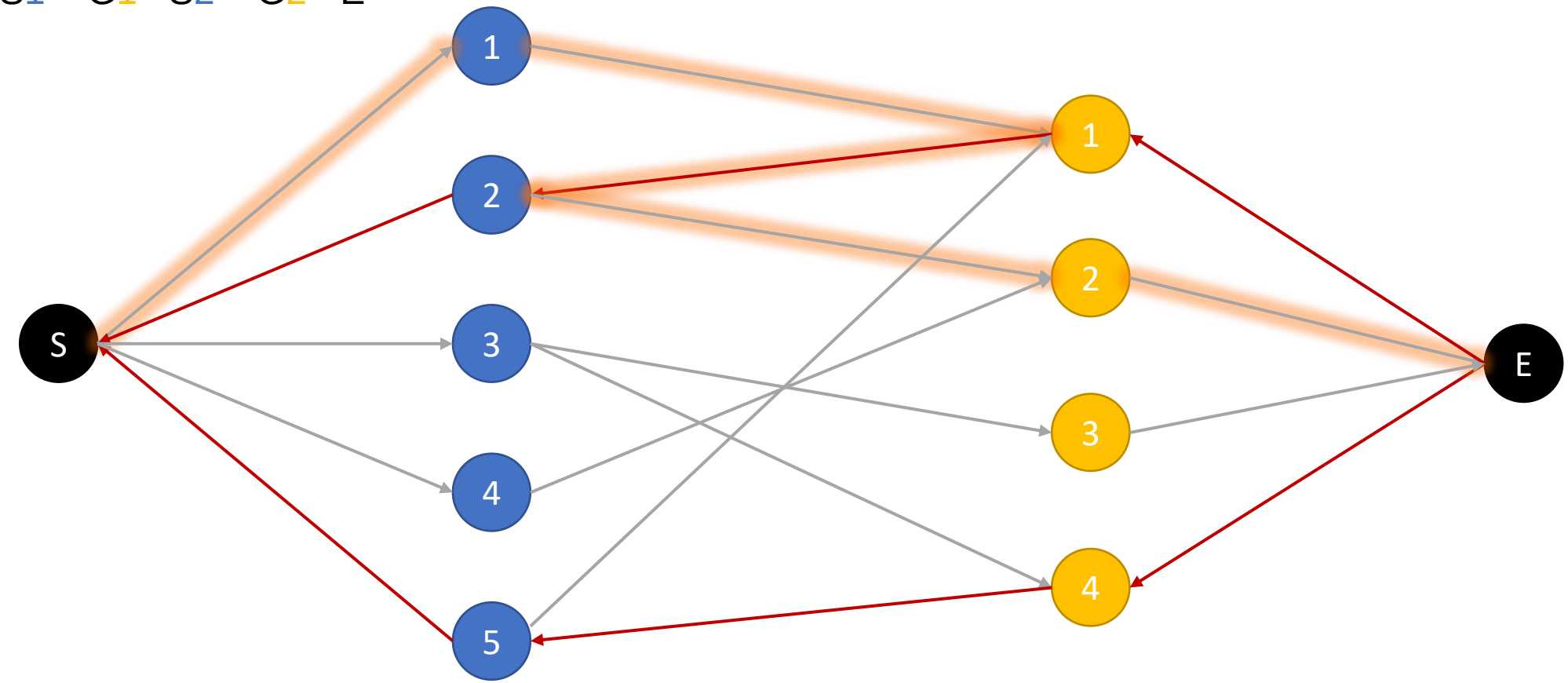
Beispiel (Option 1)

Kanten umdrehen



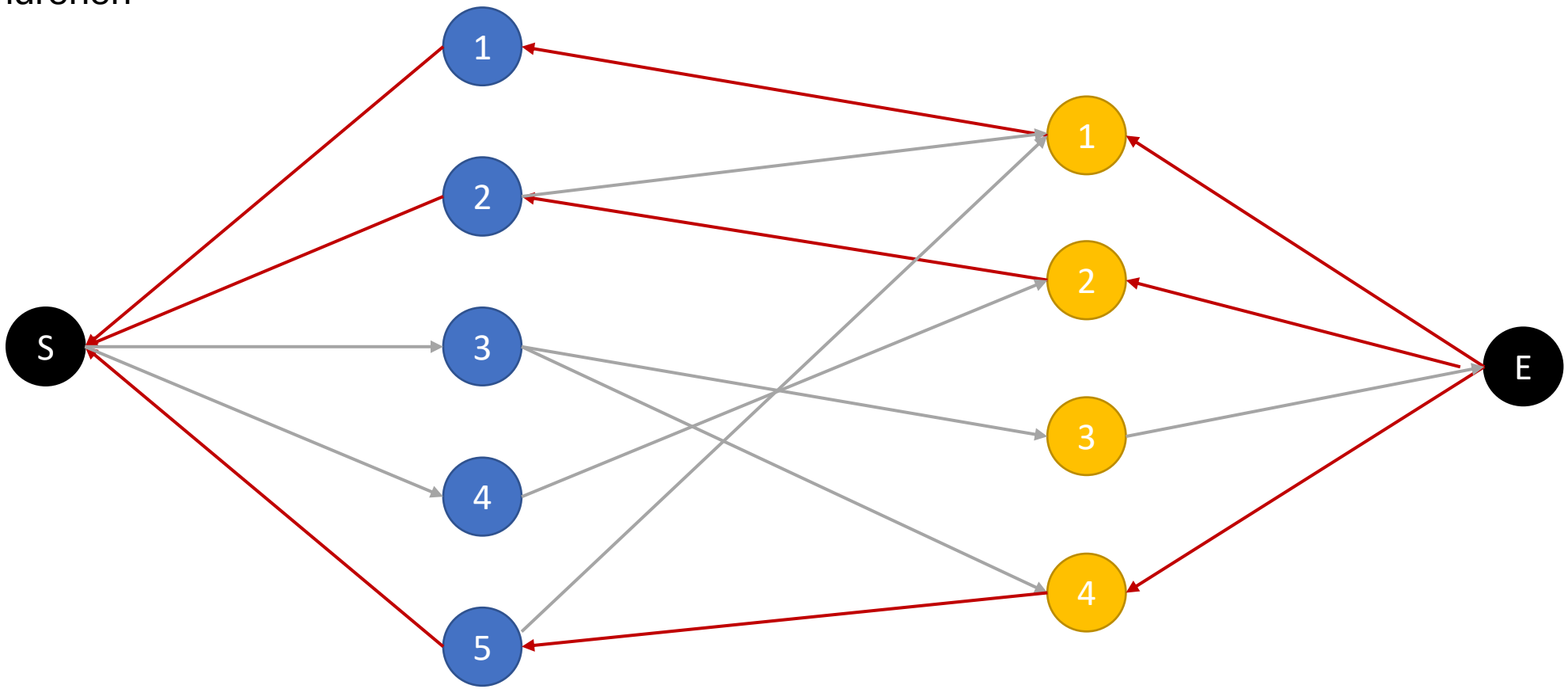
Beispiel (Option 1)

Weg S – S1 – G1 –S2 – G2 - E



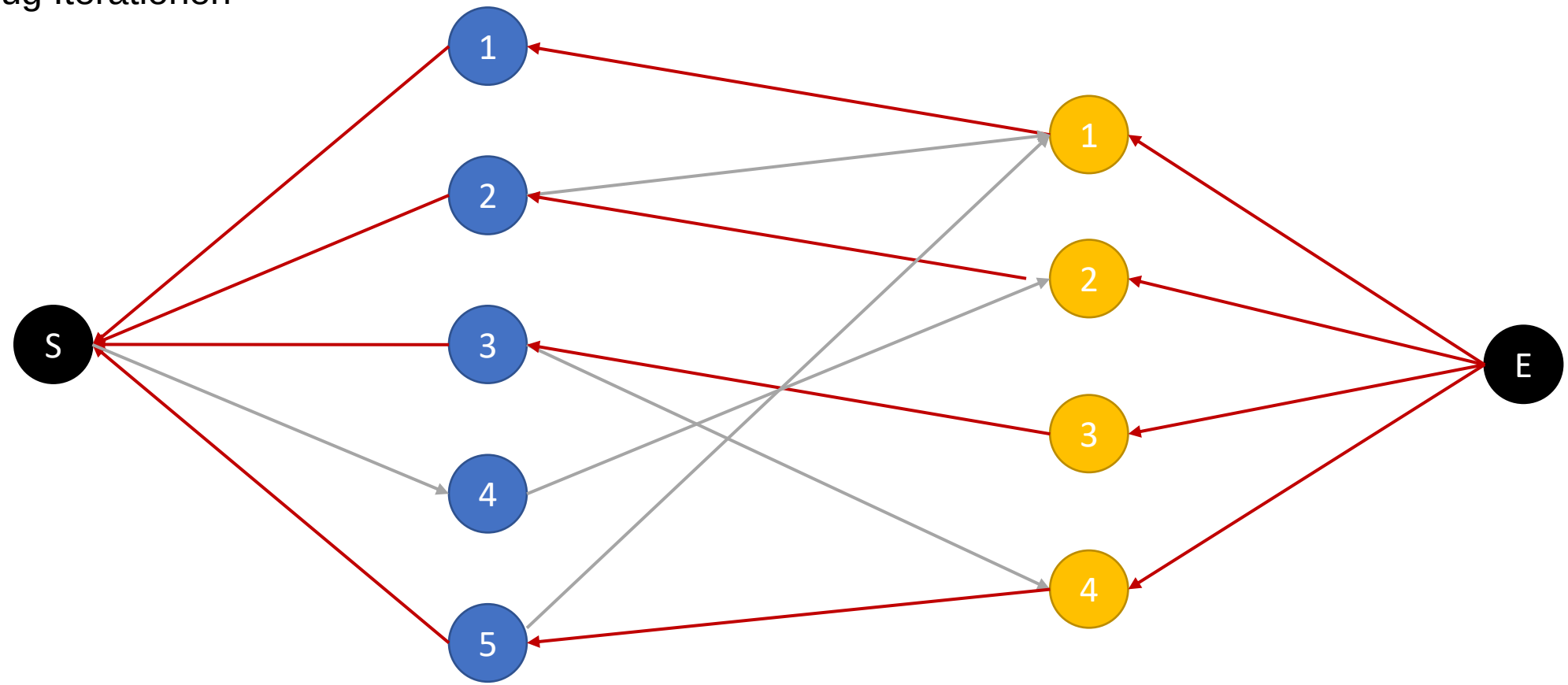
Beispiel (Option 1)

Kanten umdrehen



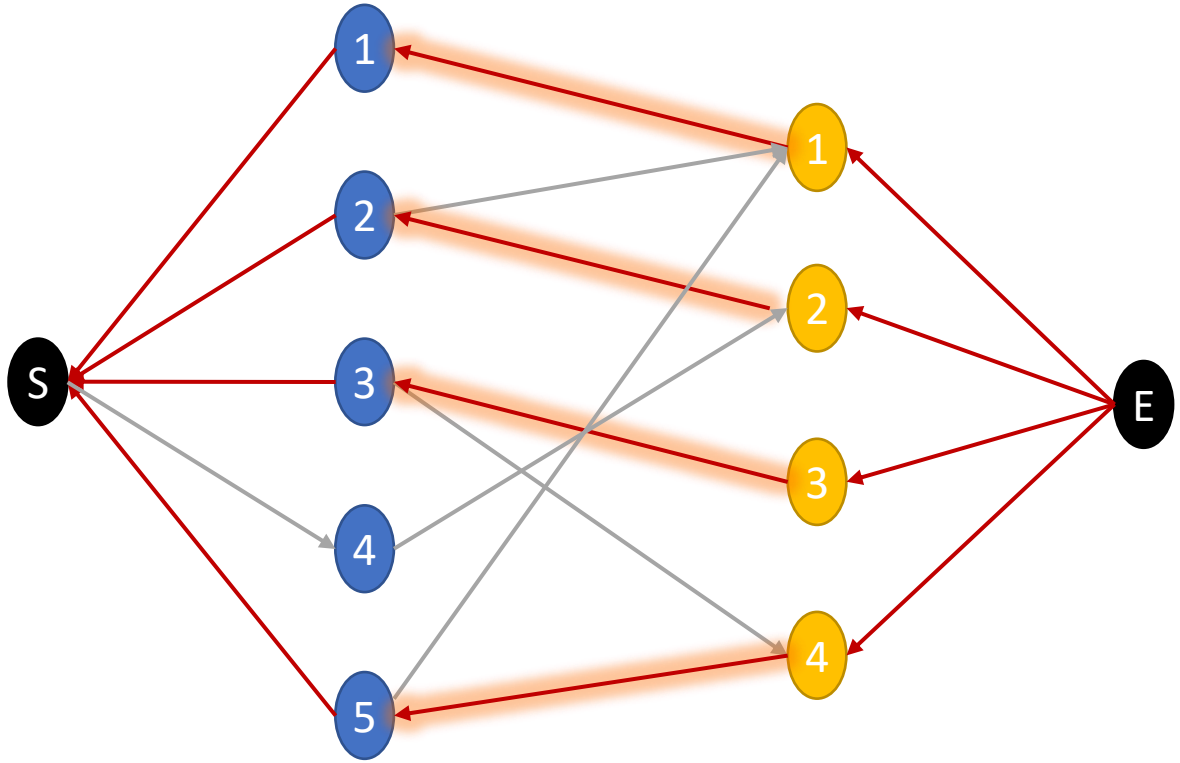
Beispiel (Option 1)

Nach genug Iterationen



Beispiel (Option 1)

- Keine Wege mehr möglich
- größtes Matching
- Max Flow ist die Anzahl an umgekehrten Kanten zwischen Steckdosenknoten und Geräteknoten



Laufzeit (Option 1)

- Einen s-e-Weg mit DFS oder BFS zu finden dauert $O(k)$ Zeit.
- Für jeden gefundenen Weg wird eine von s ausgehende Kante umgekehrt.
- Da s nur $O(m)$ -Kanten enthält, werden höchstens $O(m)$ -Pfade gefunden.
- Daher benötigt Option 1 $O(mk)$ Zeit.

Max Flow (Option 2)

Versuche für jeden Knoten i der Steckdosenknoten ihn mit allen Geräteknoten j, für die es möglich ist mit i verbunden zu werden, mit i zu verbinden.

- Ist j noch mit keinen Steckdosenknoten verbunden verbinde ihn mit i und mach weiter mit dem nächsten Steckdosenknoten i+1.
- Ist j schon mit einem Steckdosenknoten k verbunden muss rekursiv geprüft werden ob es möglich ist k mit einem anderen Knoten zu verbinden.
- Um sicherzustellen das k nicht wieder mit j verbunden wird markieren wir j als gesehen.

Beispiel (Option 2)

Input

4 3 7

1 1

1 2

2 2

2 3

3 1

4 2

4 3

Beispiel (Option 2)

Input

4 3 7

1 1

1 2

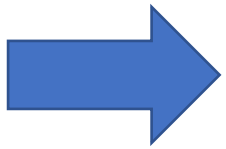
2 2

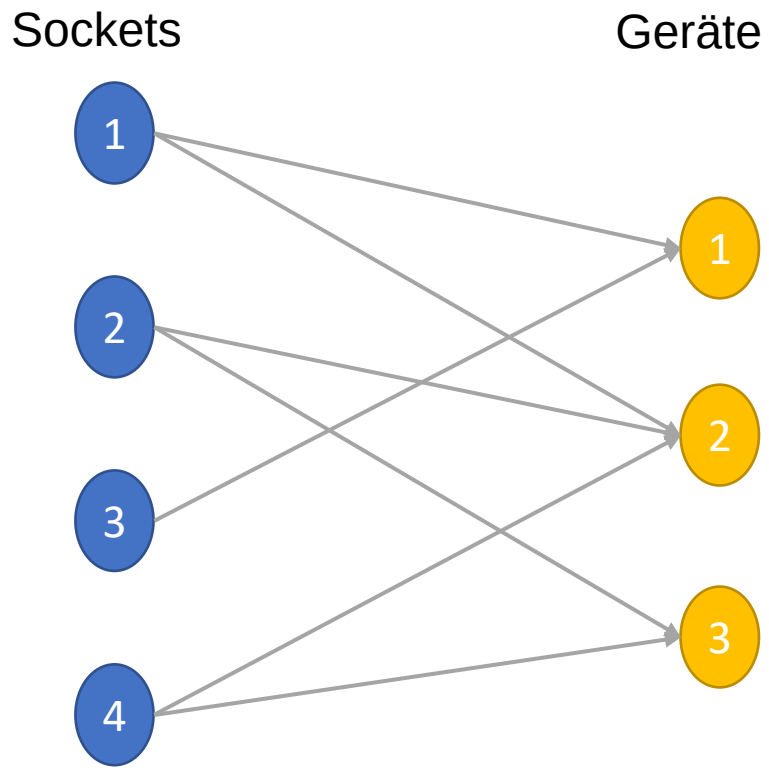
2 3

3 1

4 2

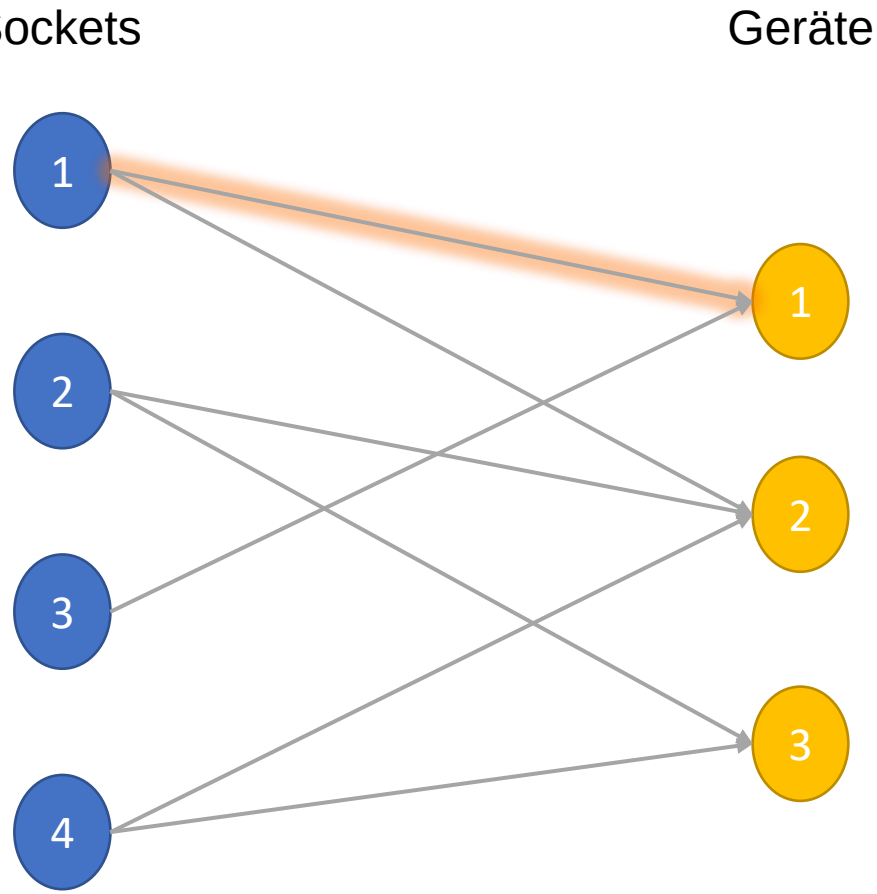
4 3





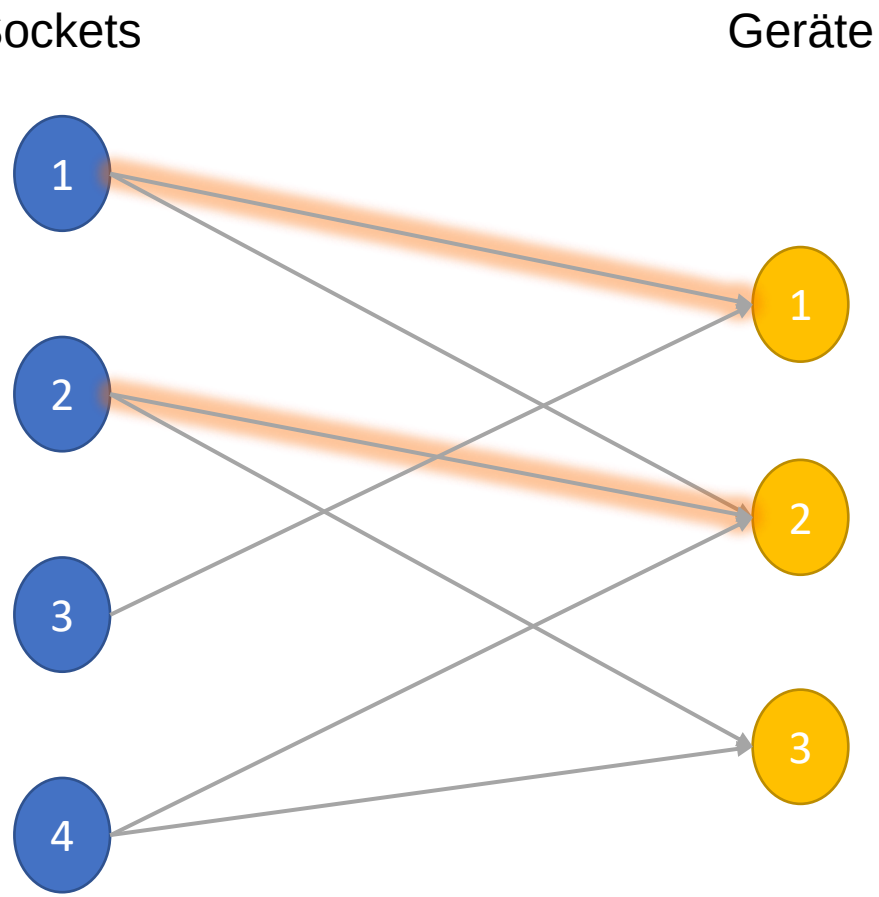
Beispiel (Option 2)

- Knoten S1 :
  - verbinde (S1,G1)



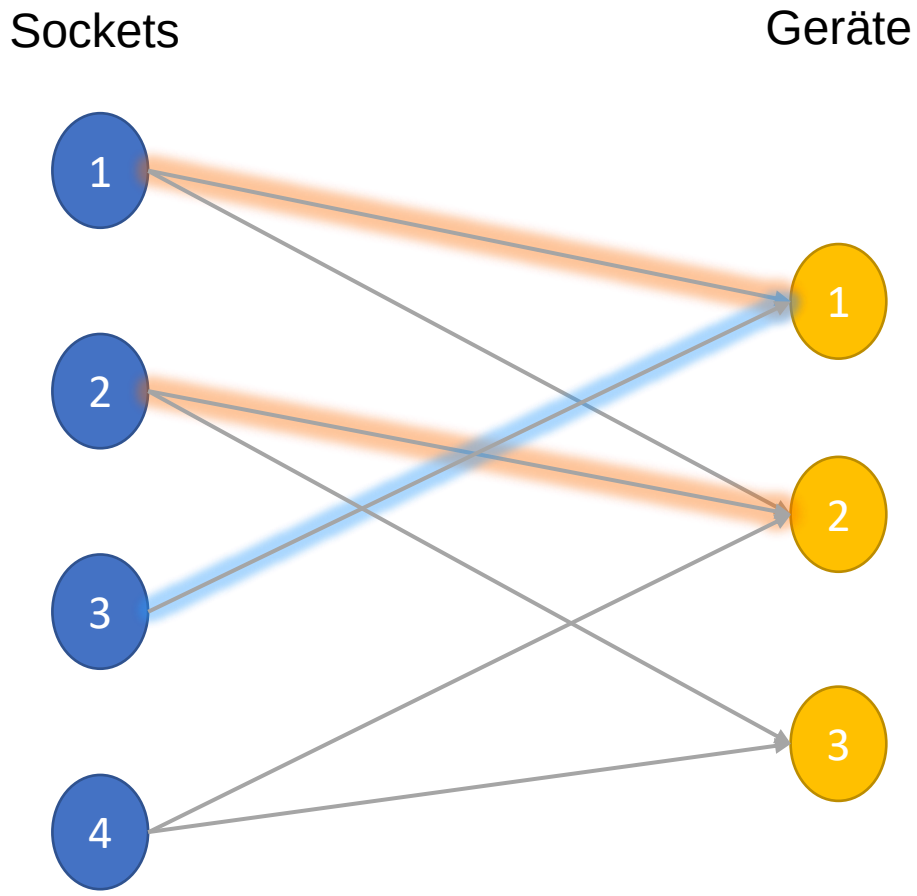
Beispiel (Option 2)

- Knoten S2:
  - verbinde (S2,G2)



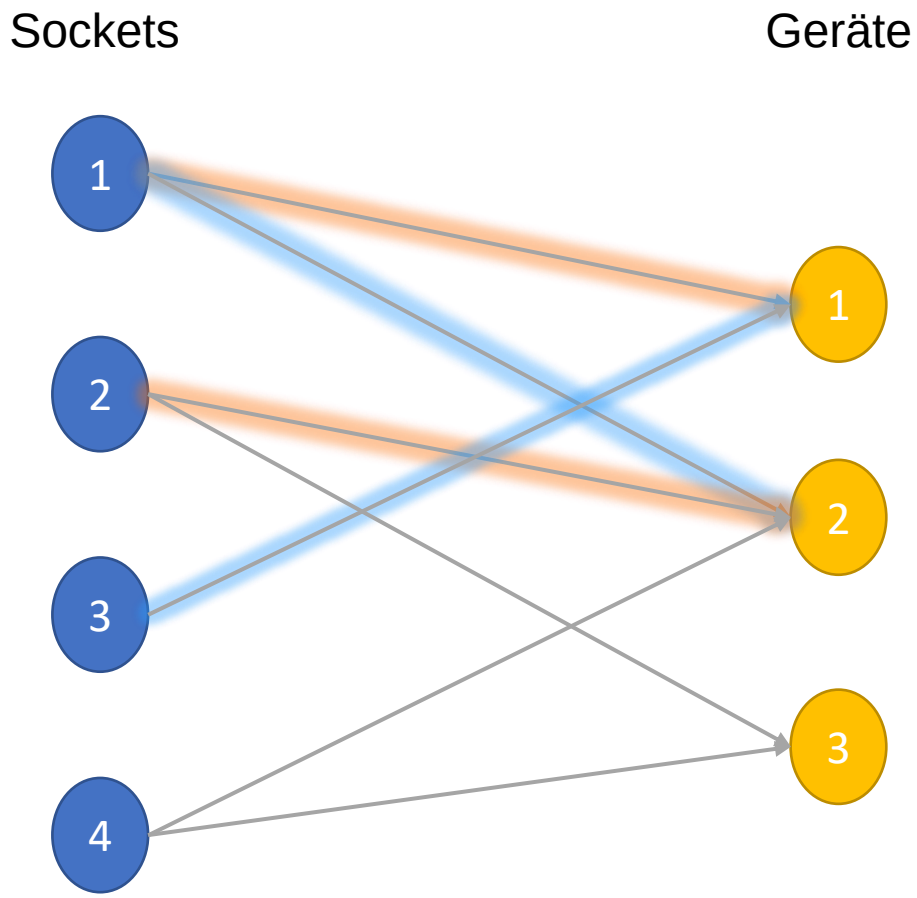
Beispiel (Option 2)

- Knoten S3:
  (1) G1 ist verbunden
  (2) Suche rekursiv alternatives G für S1



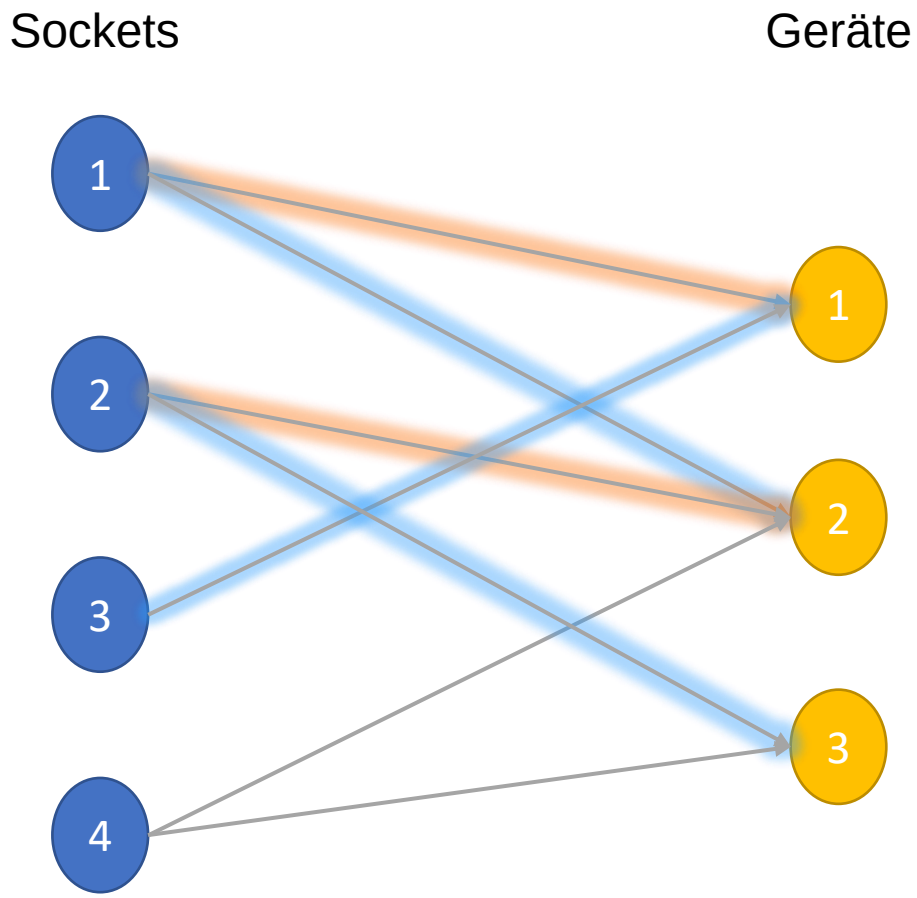
Beispiel (Option 2)

- Knoten S3:
  - (1) G1 ist verbunden
  - (2) Suche rekursiv alternatives G für S1
  - (3) G2 ist verbunden
  - (4) Suche rekursiv alternatives G für S2



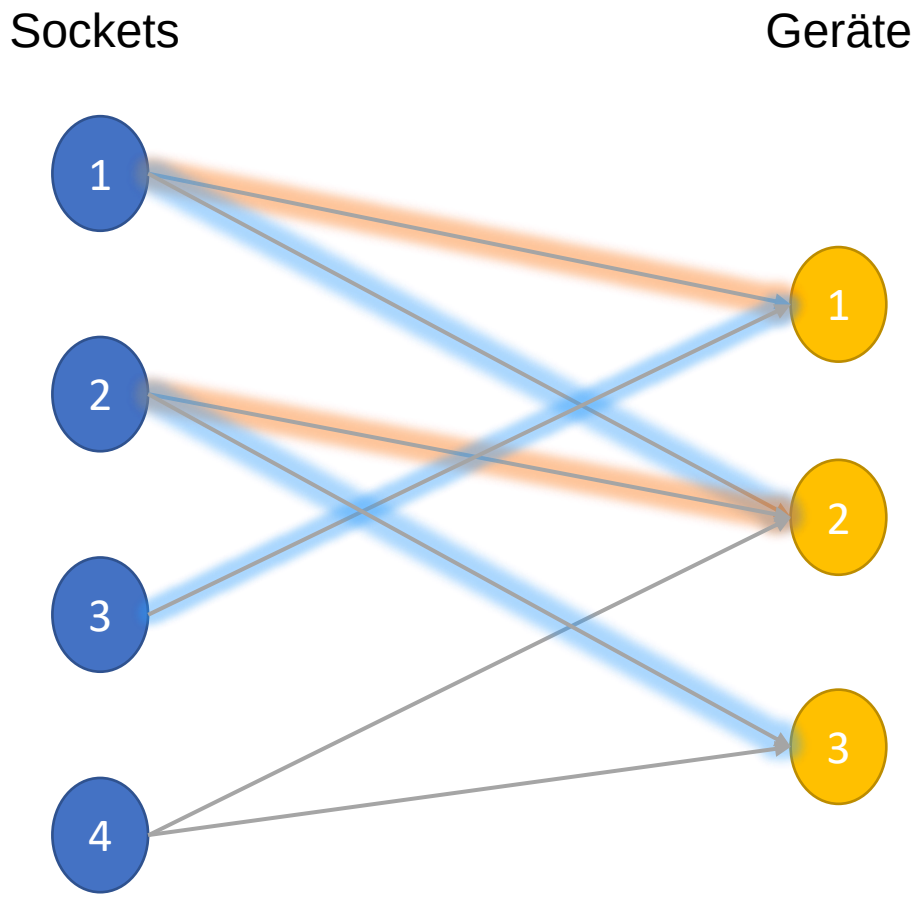
Beispiel (Option 2)

- Knoten S3:
  - (1) G1 ist verbunden
  - (2) Suche rekursiv alternatives G für S1
  - (3) G2 ist verbunden
  - (4) Suche rekursiv alternatives G für S2



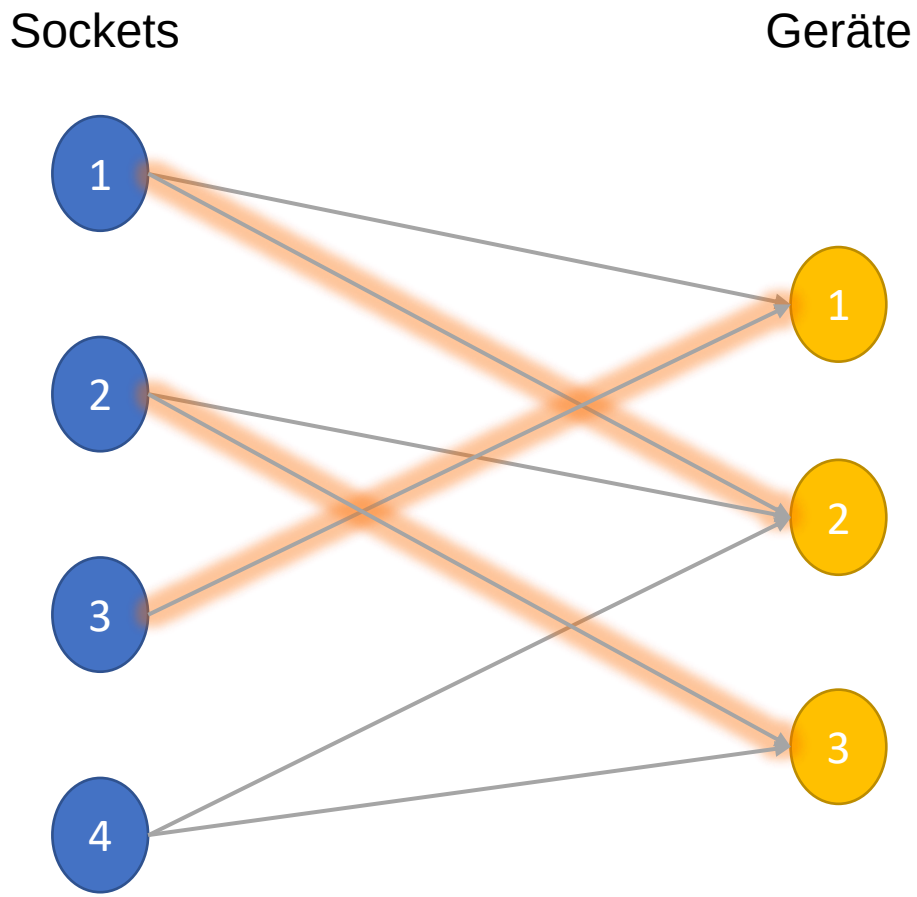
Beispiel (Option 2)

- Knoten S3:
  - (1) G1 ist verbunden
  - (2) Suche rekursiv alternatives G für S1
  - (3) G2 ist verbunden
  - (4) Suche rekursiv alternatives G für S2
  - (5) G3 ist verfügbar



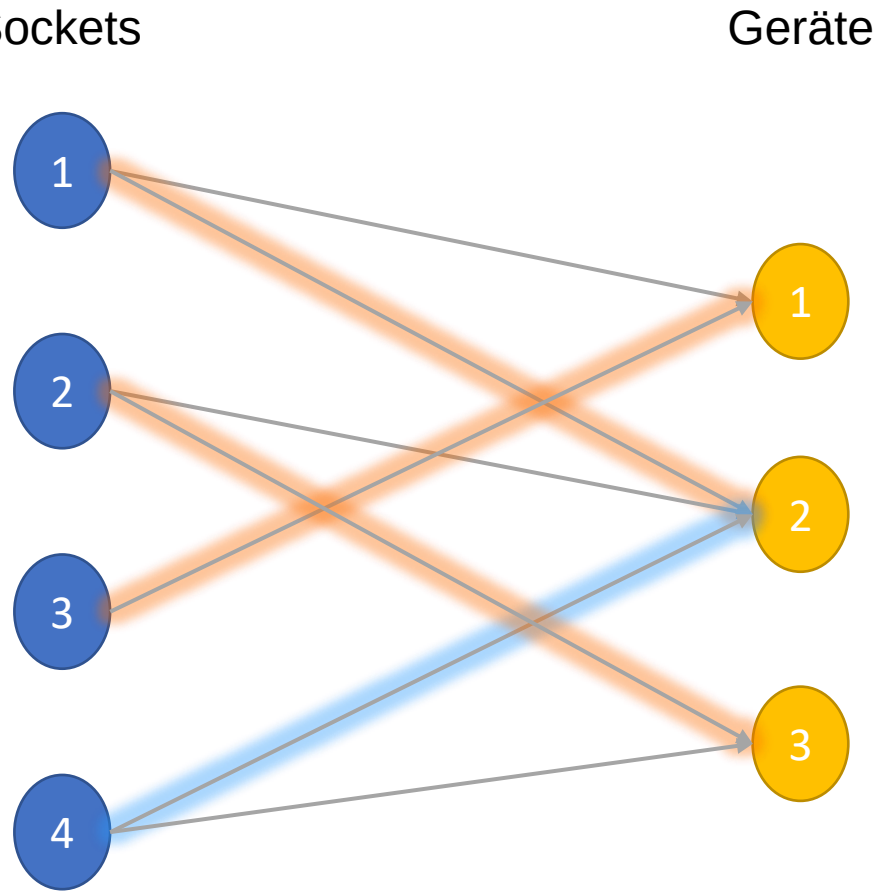
Beispiel (Option 2)

- Knoten G3:
  - Verbinde (S1,G2) , (S2,G3), (S3,G1)



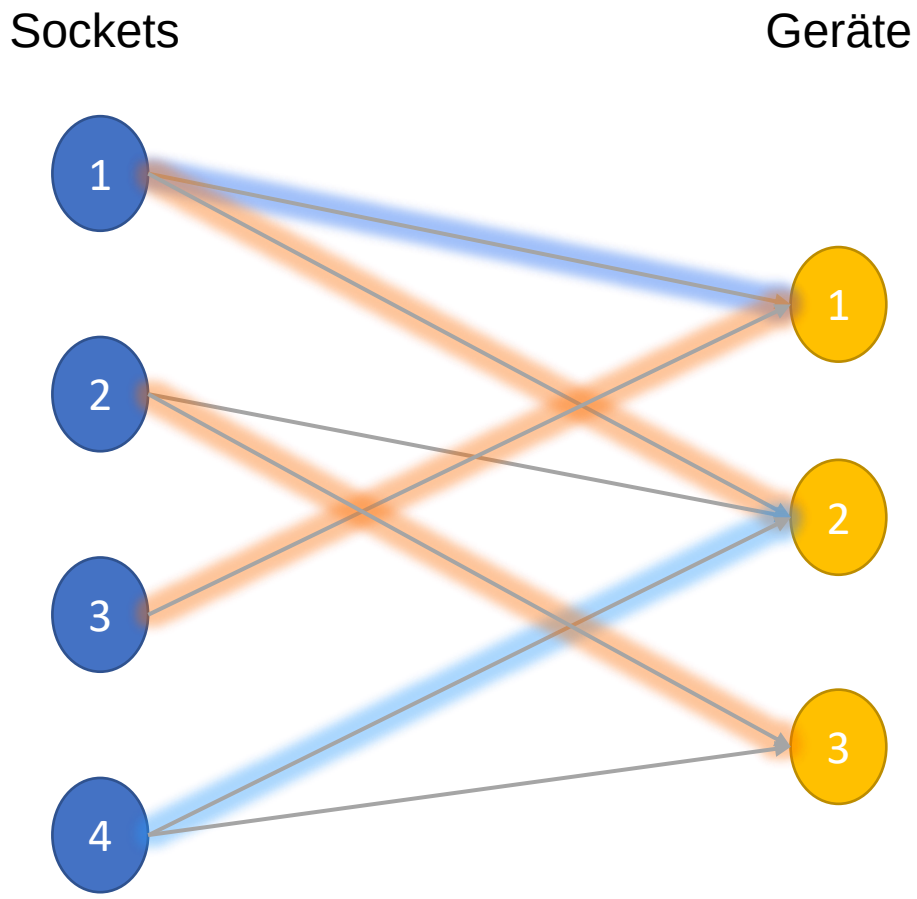
Beispiel (Option 2)

- Knoten S4:
  1) G2 ist verbunden
  2) Suche rekursiv alternatives G für S1



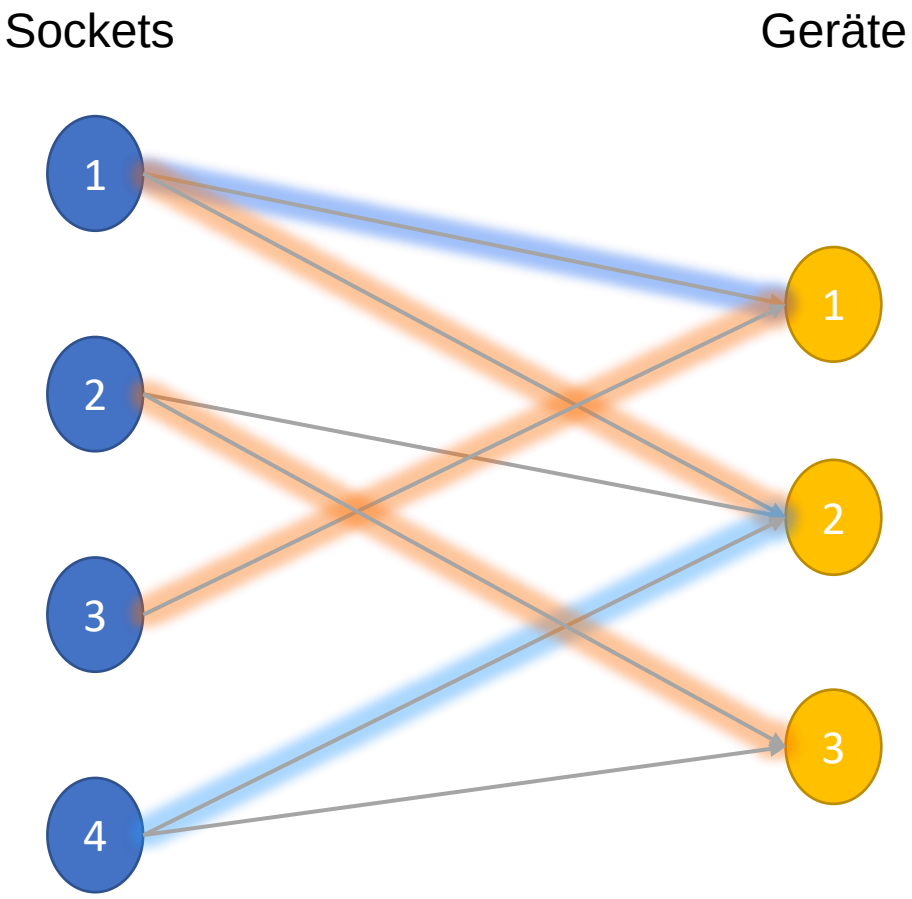
Beispiel (Option 2)

- Knoten S4:
  1) G2 ist verbunden
  2) Suche rekursiv alternatives G für S1
  3) G1 ist verbunden
  4) Suche rekursiv alternatives G für S3



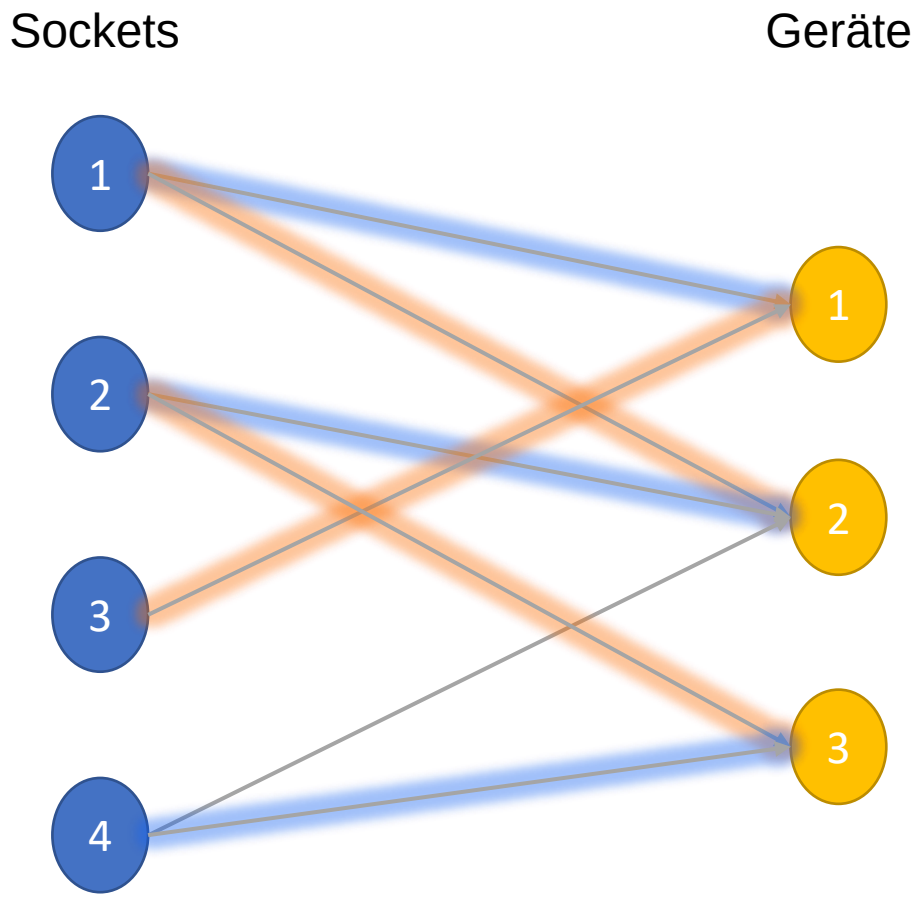
Beispiel (Option 2)

- Knoten S4:
  1) G2 ist verbunden
  2) Suche rekursiv alternatives G für S1
  3) G1 ist verbunden
  4) Suche rekursiv alternatives G für S3
  5) Für S3, kein verfügbares G
  6) Beende Suche



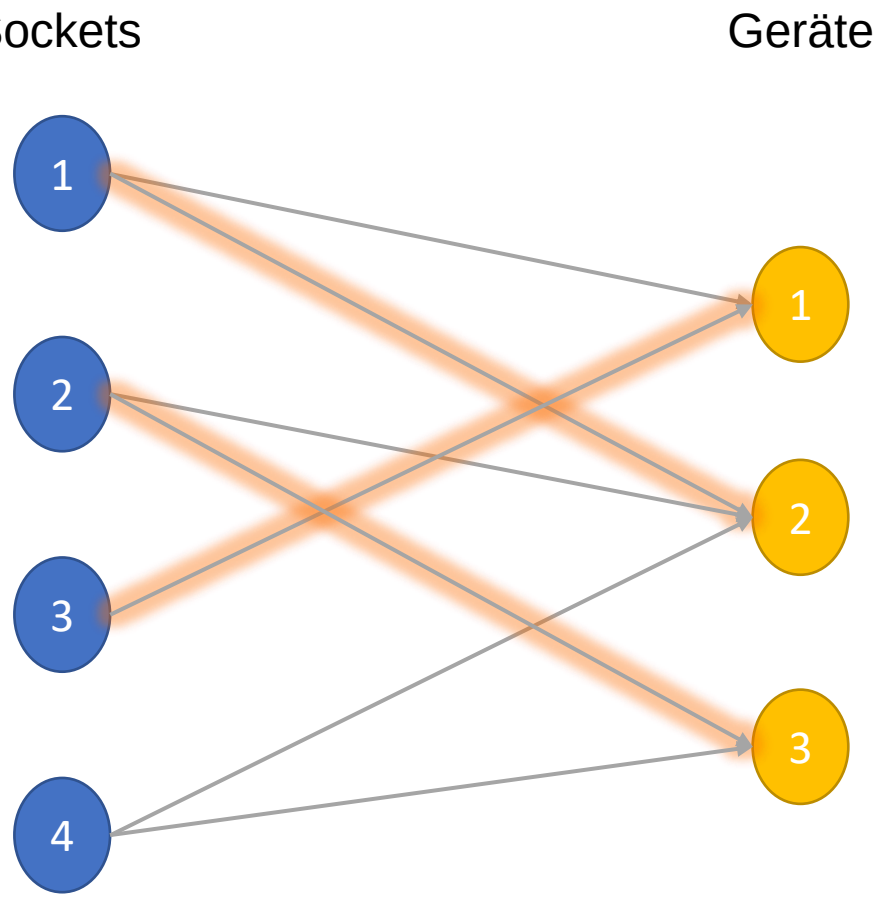
Beispiel (Option 2)

- Knoten S4:
  1) G3 ist verbunden
  2) Suche rekursiv alternatives G für S2
  3) G2 ist verbunden
  4) Suche rekursiv alternatives G für S1
  5) S1 ist verbunden
  6) Suche rekursiv alternatives G für S3
  7) Für S3, kein verfügbares G
  8) Beende Suche



Beispiel (Option 2)

Suche beendet



Max Flow (Option 2)

```
def FindMatch(i):
  for all neighbours j of i:
    if not seen[j]:
      seen[j] = True
        if match[j] < 0 or FindMatch(match[j]):
          match[j] = i        ## Pair up (i, j)
          return True
  return False

def BipartiteMatching():
  numMatches = 0
  for all i in |U|:
    seen[] = (False)*|V|
    if FindMatch(i):  numMatches++
```

Laufzeit (Option 2)

- Für jeden Steckdosenknoten i könnten wir rekursiv alle Kanten durchlaufen und versuchen, eine Übereinstimmung zu finden, wobei wir $O(k)$ Zeit benötigen.
- Wenn wir dies für jeden Steckdosenknoten tun, erhalten Sie eine Gesamtlaufzeit von $O(mk)$.

- Damit ist Option 2 zumindest theoretisch so schnell wie Option 1

Laufzeit

Aufbau Graph:

$O\,(m+n+k) = O\,(m+k)$ da $O(m) = O(n)$

Max Flow:

$O(mk)$

Algo:

$O(m^2k)$ da $O(m)$ Positionen des Mehrfachsteckers betrachtet werden müssen

Möglichkeiten zur Optimierung

Berechnen Max Flow ohne Mehrfachstecker -> maxflowohne
Füge für jeden Steckdosenknoten 2 Duplikate hinzu.
Iteriere über die Möglichkeiten mit Duplikaten und breche ab wenn das Ergebnis einer Iteration gleich

- maxflowohne + 2 (maximal mögliche Verbesserung durch Mehrfachstecker)
- n (alle Geräte angeschlossen)
- $m+2$ (alle Steckdosen belegt)

Nur Iterationen durchführen die Sinn machen. Wenn es für eine Steckdose nur ein mögliches Gerät gibt, verbessert sich das Ergebnis nicht, wenn diese den Mehrfachstecker bekommt.