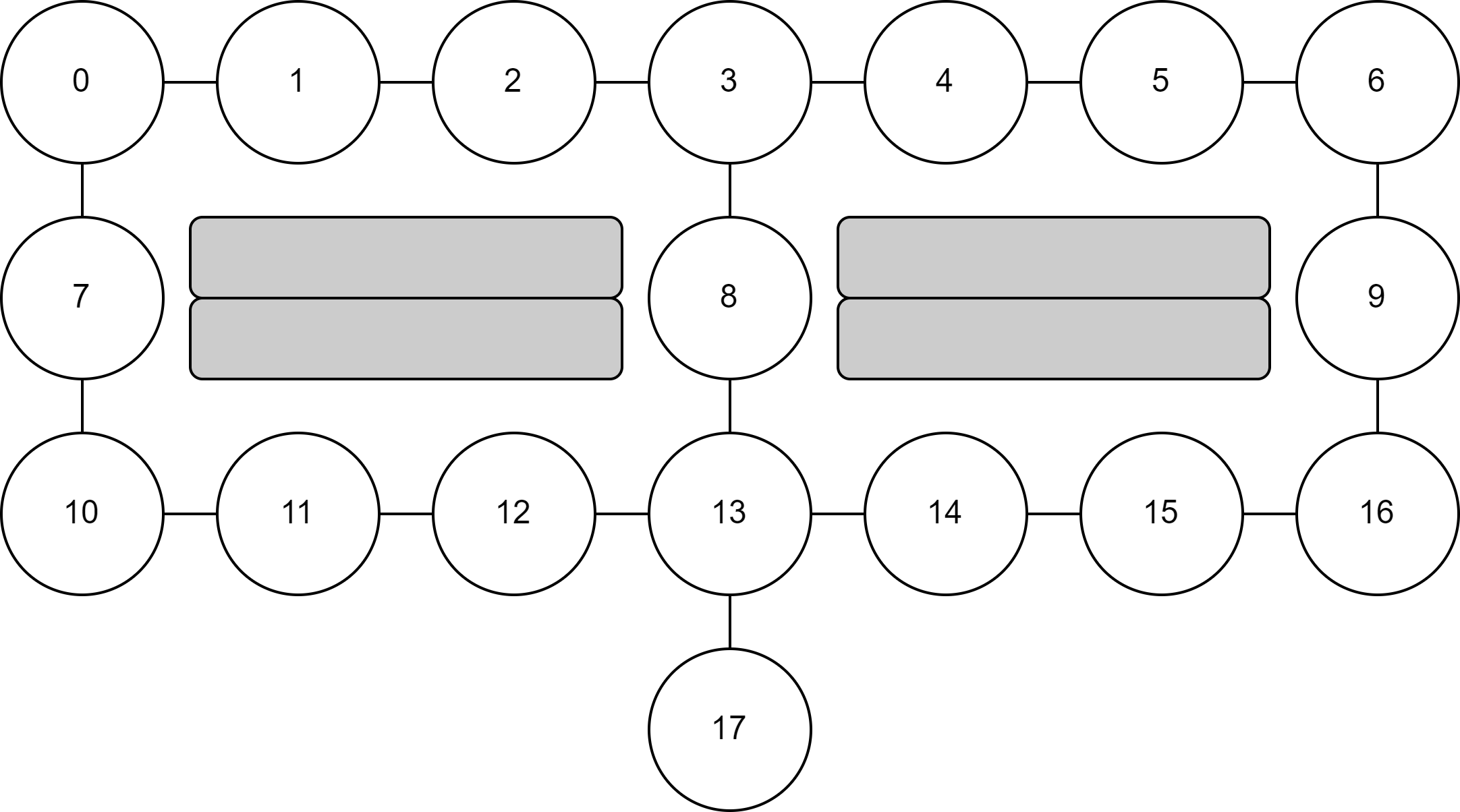
X.揀貨演算法

在現有的自動倉儲系統中已經提出了很多種揀貨的演算法[1,2,3]，其大多都是旅行推銷員問題，基於本研究中的研究環境下設計了一套揀貨演算法，由於可以確保研究環境下是由多個員工同時揀貨，因此對每一個員工的工單都可以規劃出一條揀貨路徑，本研究演算法以Floyd-Warshall演算法和回朔法與部分優化實作。

X.1 倉儲系統環境



圖X.研究倉儲系統環境

圖X表示了本研究演算法中的系統環境，其中圓形代表節點，圓形連接的黑線代表邊，就如同常見的矩形倉庫，分成多個走道，圖中節點17為門禁的位置，人員的出入口都在節點17，因此揀貨完畢會回到節點17而在揀貨的過程則會經過不同指定的節點，路徑的規劃則是依靠本研究所提出的揀貨演算法。

X.2 演算法設計

Algorithm 1: DFS-BACKTRACKING(S,d,sum,u,end,minCost,depth)

1. **if** sum > minCost **then**

2. return

3. **if** depth == S.size **then**

4. **if** sum + d[u][end] < minCost **then**

5. minCost = sum + d[u][end]

6. return

7. # all possible permutations of the nodes in the set S

8. **for** each vertex v ∈ S **do**

9. **if** v.visit == 0 **then**

10. v.visit = 1

11. DFS-BACKTRACKING(S,d,sum+d[u][v],v,end,minCos,depth+1)

12. # Pass minCost by reference

13. v.visit = 0

圖X與旅行推銷員問題不盡相同，但是解決問題的方法是一致的，先使用Floyd Warshall演算法估算出任兩點的最短路徑，再透過回朔法求最佳解(如演算法1)，對於回朔法的優化在此可以假設一個例子，工單A揀貨須經過0,1,2的節點，先透過深度優先搜尋(DFS)的方式求出一個答案0⇝1⇝2，得到一組最少的花費，之後在每一層遞迴時去比較如果由起點至現在的總和已經超過此答案則可以直接捨去，減少遞迴次數，在後續的實驗結果中也會展示未優化和優化後的具體時間差異。演算法2為本研究所使用的揀貨演算法之虛擬碼。

Algorithm 2: 揀貨演算法(S,G,start,end)

1. n <- the number of nodes in G

2. pi <- n\*n matrix

3. d <- n\*n matrix

4. **for** k <- 1 to n **do**

5. **for** i <- 1 to n **do**

6. **for** j <- 1 to n **do**

7. **if** d[i][k]+d[k][j] < d[i][j] **then**

8. d[i][j] = d[i][k]+d[k][j]

9. pi[i][j] = pi[k][j]

10. minCost = INT\_MAX

11. DFS-BACKTRACKING(S,d,0,start,end,minCost,0)

12. # Pass minCost by reference

13. return minCost