Graph algorithmic approach

Sebastian Schenker

Feasibility and optimality

Simple instance

```
4            (time slots)
2            (types)
0 1 0 0      (demand)
0 0 1 1      (demand)
3            (costs)
0 4          (costs)
5 0          (costs)
```

- Feasible schedule:
  - demand of each type is fulfilled
  - at each time slot at most one type is produced
- Optimal schedule:
  - Lowest cost (inventory + changeover) among all feasible schedules

Feasible schedules

```
0 1 0 0   (demand type 0)
0 0 1 1   (demand type 1)
```

- 6 feasible schedules:
  a) [-1 0 1 1]
  b) [0 -1 1 1]
  c) [0 1 -1 1]
  d) [1 0 -1 1]
  e) [0 1 1 -1]
  f) [1 0 1 -1]

Variables

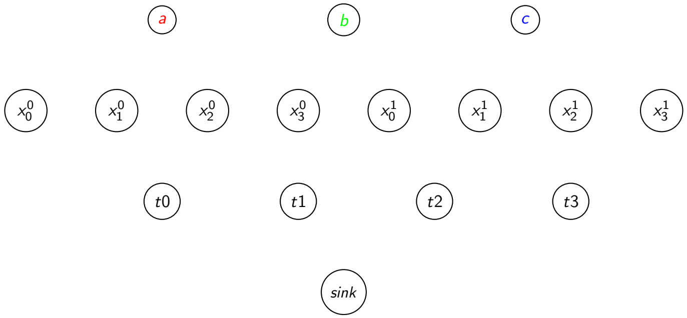
- consider $x_j^i \in \{0,1\}$ with

  $x_j^i = 1 \;\Leftrightarrow$ type *i* is produced at time slot *j*

Graph instance

0 1 0 0 (demand type 0)

0 0 1 1 (demand type 1)

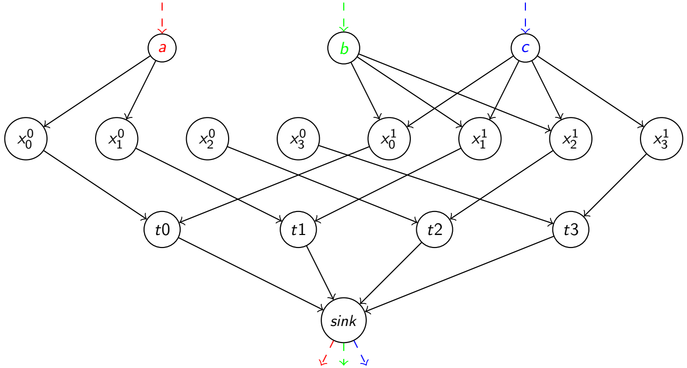


- Number of vertices: $|\text{items}| + (|\text{time\_slots}| + 1) \cdot |\text{types}| + 1$

Graph instance

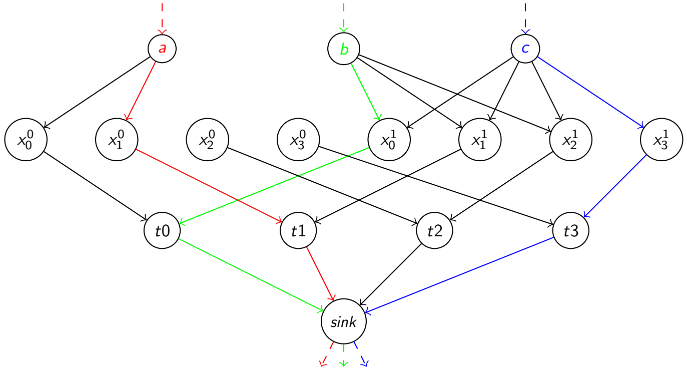
0 1 0 0 (demand type 0)

0 0 1 1 (demand type 1)



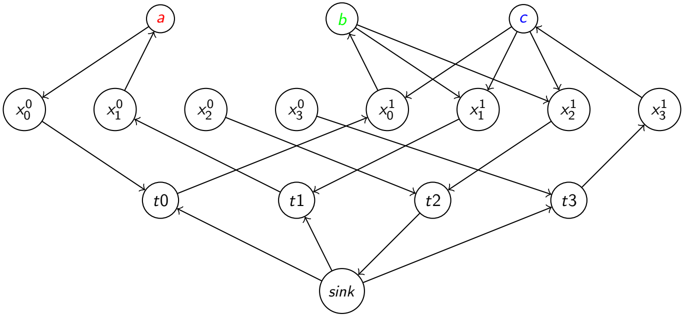
- Unit capacity on each edge

Feasible flow corresponds to schedule

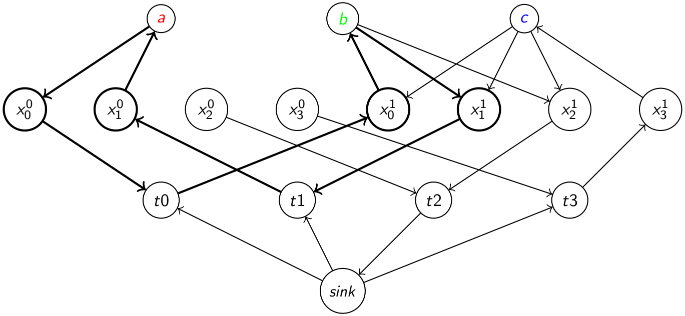


- corresponding schedule: [1 0 -1 1]

Consider directed cycles in residual graph

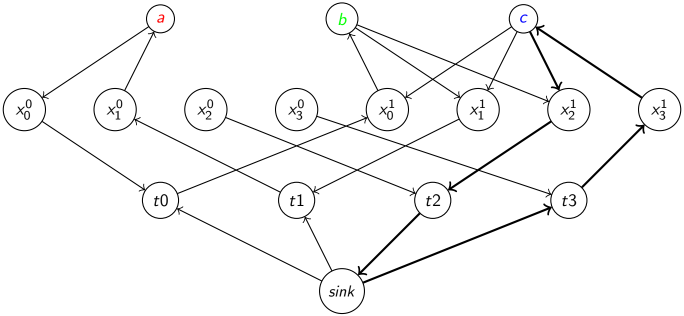


Simple directed cycle yielding different schedule



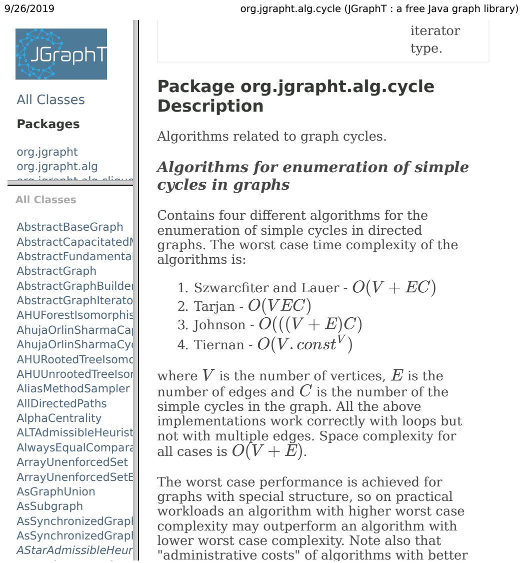
- schedule: [1 0 -1 1] $\Rightarrow$ [0 1 -1 1]

Another simple directed cycle yielding another different schedule



- schedule: [1 0 -1 1] $\Rightarrow$ [1 0 1 -1]

Algorithms for finding cycles



9/26/2019 org.jgrapht.alg.cycle (JGraphT : a free Java graph library)

All Classes

Packages

org.jgrapht
org.jgrapht.alg

All Classes

AbstractBaseGraph
AbstractCapacitatedM
AbstractFundamental
AbstractGraph
AbstractGraphBuilder
AbstractGraphIterato
AHUForestIsomorphis
AhujaOrlinSharmaCap
AhujaOrlinSharmaCyc
AHURootedTreeIsomo
AHUUnrootedTreeIsor
AliasMethodSampler
AllDirectedPaths
AlphaCentrality
ALTAdmissibleHeurist
AlwaysEqualCompara
ArrayUnenforcedSet
ArrayUnenforcedSetE
AsGraphUnion
AsSubgraph
AsSynchronizedGrapl
AsSynchronizedGrapl
AStarAdmissibleHeur

iterator type.

Package org.jgrapht.alg.cycle Description

Algorithms related to graph cycles.

Algorithms for enumeration of simple cycles in graphs

Contains four different algorithms for the enumeration of simple cycles in directed graphs. The worst case time complexity of the algorithms is:

1. Szwarcfiter and Lauer - $O(V + EC)$
2. Tarjan - $O(VEC)$
3. Johnson - $O(((V + E)C)$
4. Tiernan - $O(V.const^V)$

where V is the number of vertices, E is the number of edges and C is the number of the simple cycles in the graph. All the above implementations work correctly with loops but not with multiple edges. Space complexity for all cases is $O(V + E)$.

The worst case performance is achieved for graphs with special structure, so on practical workloads an algorithm with higher worst case complexity may outperform an algorithm with lower worst case complexity. Note also that "administrative costs" of algorithms with better

Number of cycles grows exponentially in instance size

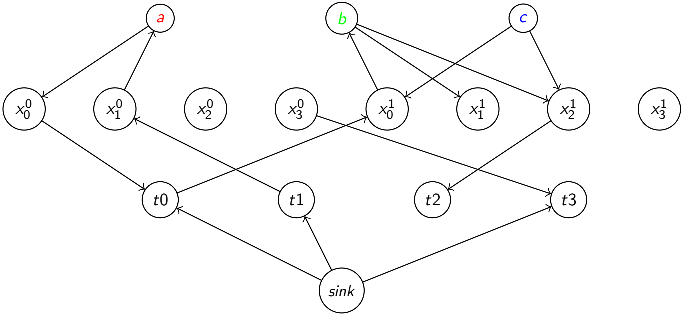
```
15
5
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
10
0 78 86 93 120
165 0 193 213 178
214 170 0 190 185
178 177 185 0 196
201 199 215 190 0
```

- 59118 cycles found in 0.359 seconds

```
15
6
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
10
0 78 86 93 120 12
165 0 193 213 178 12
214 170 0 190 185 12
178 177 185 0 196 12
201 199 215 190 0 12
201 100 88 190 14 0
```

- 2646487 cycles found in 8.463 seconds

Limit number of cycles by considering (different) subgraph(s)



Algorithmic idea

```
1: Compute initial feasible schedule $s$
2: done $\leftarrow$ false
3: while !done do
4:     Find set $C$ of directed cycles in (set of) residual subgraph(s)
5:     Compute best schedule $\bar{s}$ for $(C, s)$
6:     if cost($\bar{s}$) $<$ cost($s$) then
7:         $s \leftarrow \bar{s}$
8:     else
9:         done $\leftarrow$ true
```