fb arc set algorithm

Algorithm

Minimal feedback arc set

We want the smallest set consisting of edges that should be removed from a directed graph in order for it to become acyclic.

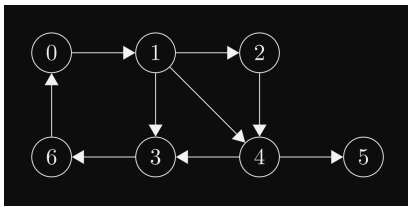
NP-complete problem, would take to long to be solved precicely. Instead we a randomized algorithm: Monte-Carlo algorithm for randomized minimal feedback arc set.

Randomized algorithm

$G =< V, E >$

$V = \{0, 1, 2, 3, 4, 5, 6\}$

$E = \{(0, 1), (1, 2), (2, 4), (1, 4), (1, 3), (4, 3), (4, 5), (3, 6), (6, 0)\}$



1. Create a randomized permutation of V:

   $\text{random} = (3, 6, 1, 5, 0, 2, 4)$

2. Create a set of $(a, b) \in E$ by only selecting (a, b) where $a > b$ in the order of the randomized permutation of V.

   For the random permutation above this would be:

   $\text{fb-arc-set} = \{(0, 1), (1, 3), (4, 3), (4, 5)\}$

   $|\text{fb-arc-set}| = 4$

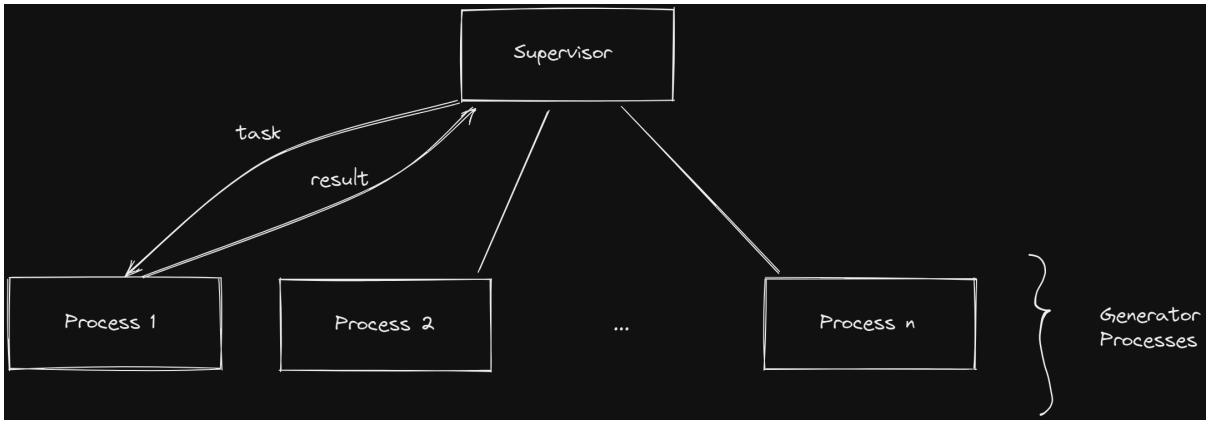
3. Compare the size of the generated set with all other generated solutions.

   Select solution with the smallest number of sets.

   Then repeat this process.

How it works: By removing all edges where $a > b$ only edges where $a < b$ are left, which creates a topological ordering of our graph which is acyclic.

Parallelization



Supervisor

Takes no arguments.

Sets up shared memory and semaphores. Initializes circular buffer.

Waits for generators to write solutions to circular buffer.

Then reads every solution and remembers solution with smallest size so far.

Every time a better solution is found, the supervisor writes that solution to `stdout`.

Either terminates if it receives `SIGINT` / `SIGTERM` signal or if one generator finds a solution of size 0 (then the graph is acyclic).

Before termination:

Tells the generators to terminate aswell → can be done by setting a variable in the shared memory, which is checked by each generator process before writing to the buffer.

Closes all shared recources, exits.

Generator

▼ Takes graph as argument.

```
SYNOPSIS
    generator EDGE1...
EXAMPLE
    generator 0-1 1-2 1-3 1-4 2-4 3-6 4-3 4-5 6-0
    # these are the edges from the example above
```

The given edges must be syntactically correct.

At least one edge must be given (argument is not optional).

Nodes can be derived from edges.

Repeatedly generates random solutions, writes to circular buffer one at a time (until notified by supervisor to terminate).

Circular Buffer (concurrent data structure)

Generators report their solutions to supervisor through a circular buffer (built using shared semaphores, shared memory).

Generators write their solutions to the write end and the supervisor reads from the read end.

FIFO buffer with fixed size (array).

Reading and writing should be able to happen at the same time.

It should be avoided overwriting data which has not been read and reading data which has not been written yet.

Use 1 semaphore for the free space, 1 semaphore for the used space and 1 semaphore for mutual exclusion for writing into the buffer (since we have multiple generators).