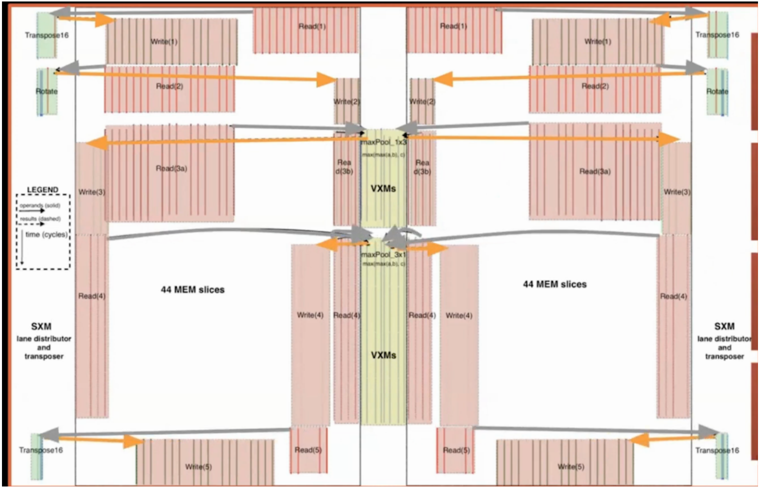
A 10-minute scheduling classic with `glpk-hs`

Valentin Reis
https://github.com/ilpsched/ilpsched

Motivation


Transpose16
Write(1)
Read(1)
Read(1)
Write(1)
Transpose16
Rotate
Read(2)
Write(2)
Write(2)
Read(2)
Rotate
maxPool_1x3
Read(3a)
Rea d(3b)
Rea d(3b)
Read(3a)
LEGEND
time (cycles)
Write(3)
VXMs
Write(3)
maxPool_3x1
44 MEM slices
44 MEM slices
Read(4)
Write(4)
Read(4)
Read(4)
Write(4)
Read(4)
SXM
lane distributor
and
transposer
VXMs
SXM
lane distributor
and
transposer
Transpose16
Read(5)
Read(5)
Transpose16
Write(5)
Write(5)

Integer Linear Programming

$$
\begin{aligned}
\text{minimize} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\
& \mathbf{x} \geq \mathbf{0}, \\
\text{and} \quad & \mathbf{x} \in \mathbb{Z}^n.
\end{aligned}
$$

`glpk-hs`: shallow eDSL, solver bindings

The scheduling problem

Sequential jobs with precedence constraints, heterogeneous resources.

- Each resource is only able to run a subset of the jobs.
- Job runtime is function of resource and job.

Formalization

Statically known set of **jobs** J, **resources** R and **precedences** P.

- $j \in J$ must precede job $k \in J$ if there exists $(j, k) \in P \subset J \times R$ (no cycles).
- Resource-job compatibility parameter $v_{jr} \in \{0, 1\}$ of job $j \in J$ and resource $r \in R$.
- Processing time $p_{jr} \in \mathbb{N}$ for job $j \in J$ on resource $r \in R$.

Formulation

```
newtype Time = Time {unTime :: Int}
class ToILPVar a where
  lpShow :: a -> String -- constructs a variable name for our ILP

formulation ::
  forall job resource.
  (Eq job, ToILPVar job, ToILPVar resource) =>
  Set job ->
  Set resource ->
  Set (job, job) -> -- precedence relations
  ((job, resource) -> Time) -> -- runtimes
  ((job, resource) -> Bool) -> -- job validity
  Time -> -- makespan upper bound
  LP String Int
formulation
  (toList -> _J :: [job])
  (toList -> _R :: [resource])
  (toList -> _P :: [(job, job)])
  ((>>> unTime) -> p_ :: (job, resource) -> Int)
  v_
  (unTime -> u :: Int) = execLPM . sequenceA_ $
    ... -- [LPM] where LPM is the eDSL monad
```

Parametrization

Design variables:

- allocation: $x_{jr} = 1$ if $j \in J$ assigned to $r \in R$.

```
-- cartesian :: [a] -> [b] -> [(a,b)]
-- x_ :: (ToILPVar job, ToILPVar resource) => (job, resource) -> String
[setVarKind (x_ jr) BinVar | jr <- cartesian _J _R, v_ jr]
```

- start times: $s_j \in \mathbb{N}$ start time of $j \in J$.

```
-- s_ :: ToILPVar job => job -> String
[varGeq s_j 0 >> setVarKind s_j IntVar | (s_ -> s_j) <- _J]
```

Objective : $C_{\max} \in \mathbb{N}$.

```
-- cMax :: String
[setVarKind cMax ContVar, setDirection Min, setObjective $ toFun cMax]
```

parameters: `_R::[resource]` `_J::[job]` `_P::[(job,job)]` `p_::(job,resource)->Int`
`v_::(job,resource)->Bool` `cMax :: String` `x_ :: (job,resource) -> String` `s_ :: job -> String`

Constraints 1: Makespan definition.

For any job $j \in J$ and resource $r \in R$ such that $v_{jr} > 0$, one has:

$$C_{\max} \geq s_j + p_{jr} x_{jr}$$

```
[ linCombination [(-1, cMax), (1, s_ j), (p_ jr, x_ jr)] `leqTo` 0
  | jr@(j, _) <- cartesian _J _R, v_ jr ]
```

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Constraints 2: Job assigned to exactly one resource.

For any job $j \in J$, one has:

$$\sum_{r \in R} x_{jr} = 1$$

```
[add [toFun (x_ (j, r)) | r <- _R] `equalTo` 1 | j <- _J]
```

```
_R::[resource]    _J::[job]    _P::[(job,job)]    p_::(job,resource)->Int    v_::(job,resource)->Bool
cMax :: String    x_ :: (job,resource) -> String    s_ :: job -> String
```

Constraints 3: Resource must support job.

For any job $j \in J$ and resource $r \in R$ such that $v_{jr} = 0$, one has:

$$x_{jr} = 0$$

```
[ toFun (x_ jr) `equalTo` 0 | jr <- cartesian _J _R, not (v_ jr) ]
```

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Constraints 4: Precedences.

For any $r \in R$ and $(j, k) \in J$ such that $v_{jr} = 1$ and there exists a precedence relation $(j, k) \in P$, one has:

$$s_k \geq s_j + p_{jr} x_{jr}$$

```
[ linCombination [(1, s_ j), (-1, s_ k), (p, x_ (j, r))]
    `leqTo` 0
  | (j, k) <- _P,
    r <- _R,
    v_ (j, r),
    let p = p_ (j, r)
]
```

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Constraints 5: Resources not over-committed.

```
j    |------------------------------------|
k    .        |-----------------------------------------|
     .        .                           .                .
     .        .          O_jk             .                .
     .       <--(negative direction)------                 .
     .                                                     .
    <-----(negative direction)----------------------------
                   O_kj
```

$O_{kj} < 0$ and $O_{jk} < 0$ must not be satisfied at the same time.

- introduce τ_{jk} binary variables.
- make τ_{jk} correspond to $O_{jk} < 0$ (requires bounds on O_{jk})
- NAND constraint: $x_{jr} + x_{kr} + \tau_{jk} + \tau_{kj} \leq 3$

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

All Done! Example.

Two types of operations,

```
data Op = A | B Int
data Task = Task TaskID Op
newtype TaskID = TaskID Int
```

two types of resources.

```
data ResourceType = RA | RB
data Resource = Resource RID ResourceType
newtype RID = RID Int
```

Example, cont.

Jobs are only valid on a resource of the same type,

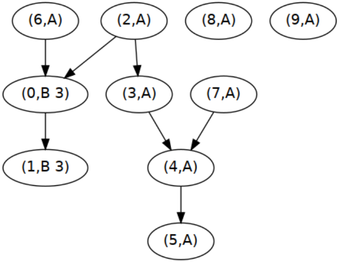
```
validity :: (Task, Resource) -> Bool
validity (Task _ A, Resource _ RA) = True
validity (Task _ (B _), Resource _ RB) = True
validity _ = False
```

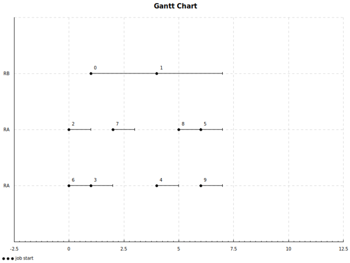
runtimes are only instruction dependent.

```
runtime :: (Task, Resource) -> Time
runtime (Task _ A, Resource _ RA) = Time 1
runtime (Task _ (B x), Resource _ RB) = Time x
runtime _ = Time (panic "runtime query on invalid placement")
```

Example, cont.

```
exampleResources :: Set Resource
exampleResources = [ Resource 1 RA, Resource 2 RA, Resource 3 RB ]
```






```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Thank you for your attention

Christodoulos A Floudas and Xiaoxia Lin.

Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications.

Annals of Operations Research, 139(1):131–162, 2005.

Mark Freeman Tompkins.

Optimization techniques for task allocation and scheduling in distributed multi-agent operations.

PhD thesis, Massachusetts Institute of Technology, 2003.

Backup

Constraints 5: resources are not over-committed, cont.

For any $(j, k) \in J$, j and k overlap iff $O_{kj} < 0$ and $O_{jk} < 0$

```
j    |------------------------------------|
k    .        |--------------------------------------------|
     .        .                           .                 .
     .        .         O_jk              .                 .
     .        <--(negative direction)------                 .
     .                                                      .
     <-----(negative direction)-----------------------------
                    O_kj
```

where $O_{jk} = s_k - \sum_{r \in R} p_{jr} x_{jr} - s_j$

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Constraints 5, cont.

We introduce $\tau_{jk} \in \{0, 1\}$, indicator variable for $O_{jk} + 1 \leq 0$ and for any $(j, k) \in J$ specify that:

$O_{jk} + 1 \leq u(1 - \tau_{jk})$ and $O_{jk} \geq -u\tau_{jk}$

where $u \in \mathbb{N}$ is an upper bound.

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```

Constraints 5, cont.

For any $(j,k) \in J$ such that $j \neq k$, $(j,k) \notin Q$ full precedence graph one has:

$$s_k - \sum_{r \in R} p_{jr} x_{jr} - s_j + 1 \leq u(1 - \tau_{jk})$$

$$s_k - \sum_{r \in R} p_{jr} x_{jr} - s_j \geq -u\tau_{jk}$$

$$\tau_{jk} \in \{0,1\}$$

and for any $r \in R$, $(j,k) \in J$ s.t. $j \neq k$, $v_{jk} = v_{kj} = 1$ one has:

$$x_{jr} + x_{kr} + \tau_{jk} + \tau_{kj} \leq 3$$

```
_R::[resource]    _J::[job]    _P::[(job,job)]    p_::(job,resource)->Int    v_::(job,resource)->Bool
cMax :: String    x_ :: (job,resource) -> String    s_ :: job -> String
```

Constraints 5, cont.

```
[ let eq = toFun (s_ k) - toFun (s_ j) + linCombination [(u, tau_ (j, k))]
            - linCombination
              [ (p_ (j, r), x_ (j, r))
                | r <- _R,
                  v_ (j, r)
              ]
  in do
        eq `geqTo` 0
        eq `leqTo` (u-1)
        setVarKind (tau_ (j, k)) BinVar
  | (j,k) <- cartesian _J _J,
    j /= k,
    (j, k) `notElem` fullPrecs
]
[ let elms = [x_ (j, r), x_ (k, r), tau_ (j, k), tau_ (k, j)]
   in linCombination ((1,) <$> elms) `leqTo` 3
  | (j, k) <- cartesian _J _J,
    r <- _R,
    v_ (j, r),
    v_ (k, r)
]
```

```
_R::[resource]   _J::[job]   _P::[(job,job)]   p_::(job,resource)->Int   v_::(job,resource)->Bool
cMax :: String   x_ :: (job,resource) -> String   s_ :: job -> String
```