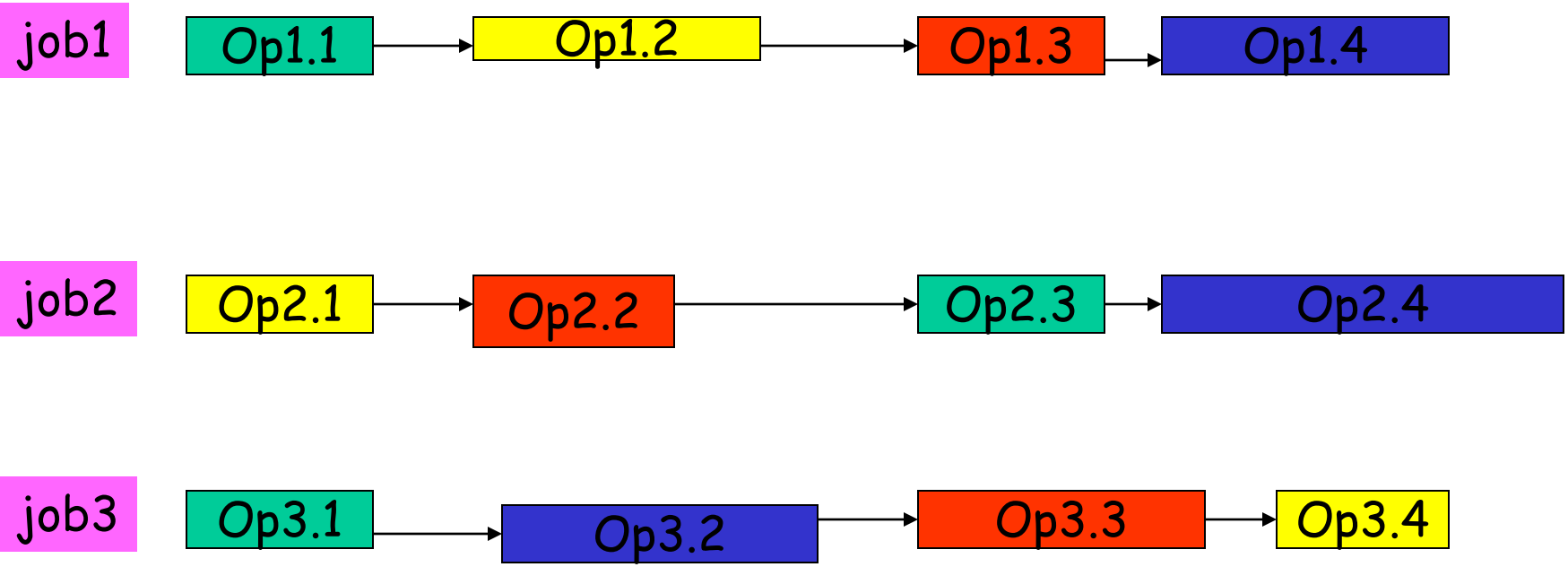
jobshop scheduling

We have

- a set of resources
- a set of jobs
    - a job is a sequence of operations/activities

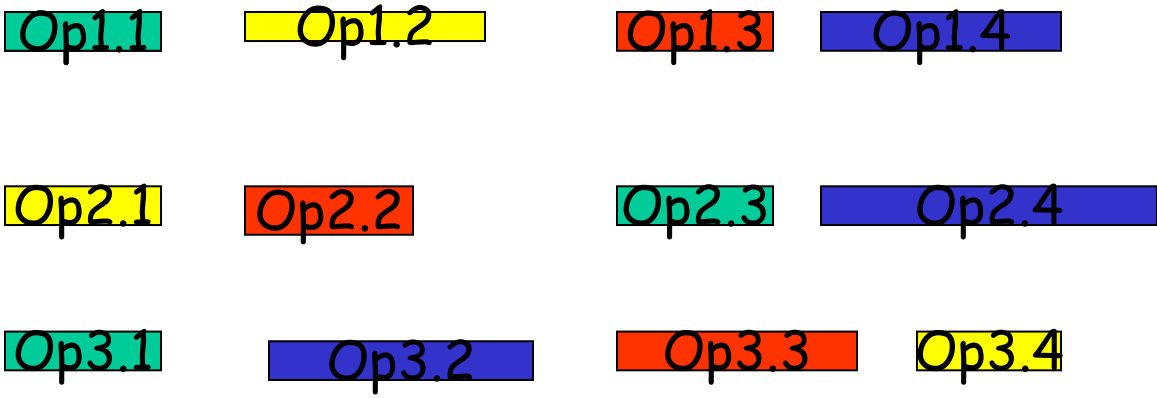
- sequence the activities on the resources

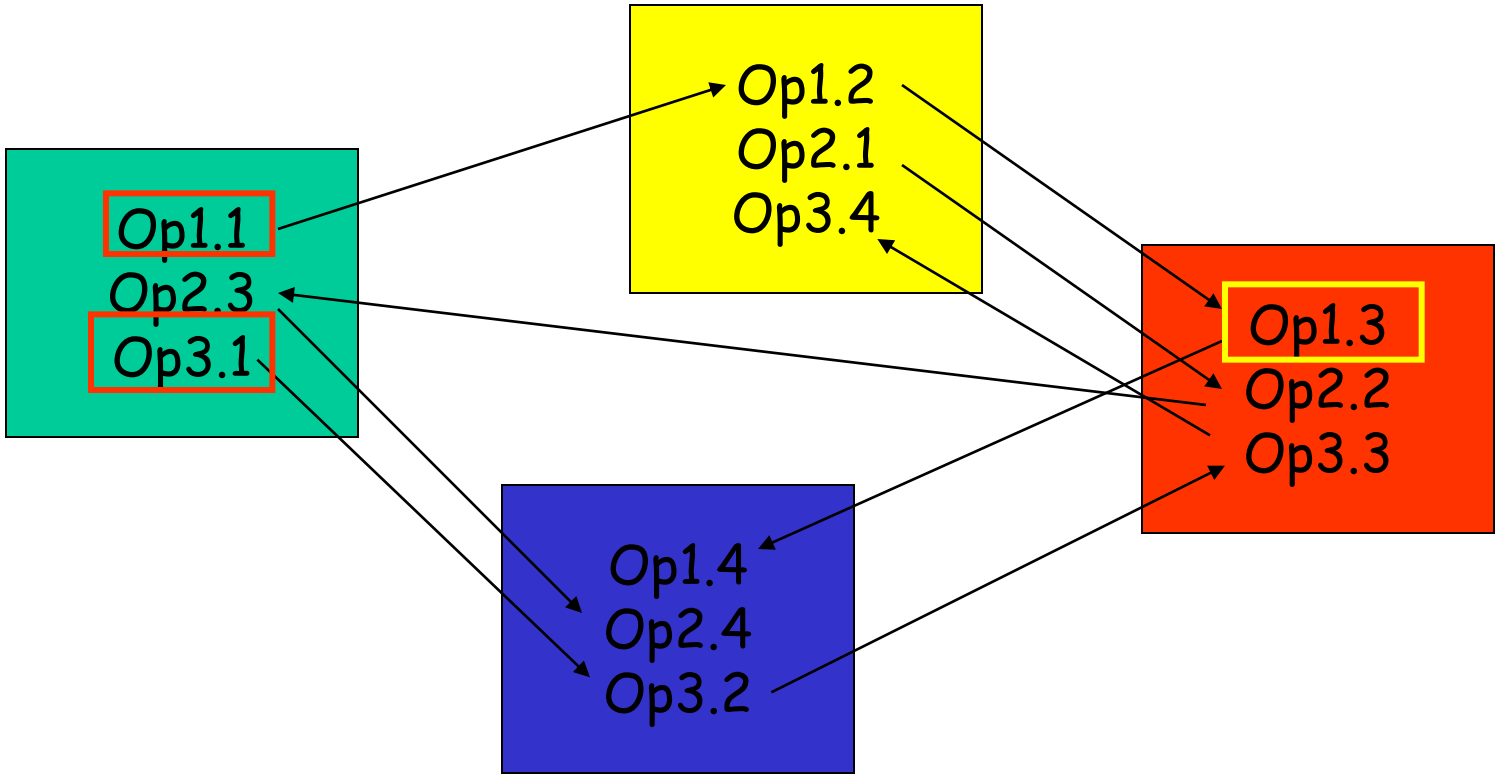
An example: 3 x 4



- We have 4 resources: green, yellow, red and blue
- a job is a sequence of operations (precedence constraints)
- each operation is executed on a resource (resource constraints)
- each resource can do one operation at a time
- the duration of an operation is the length of its box
- we have a due date, giving time windows for operations (time constraints)

An example: 3 x 4





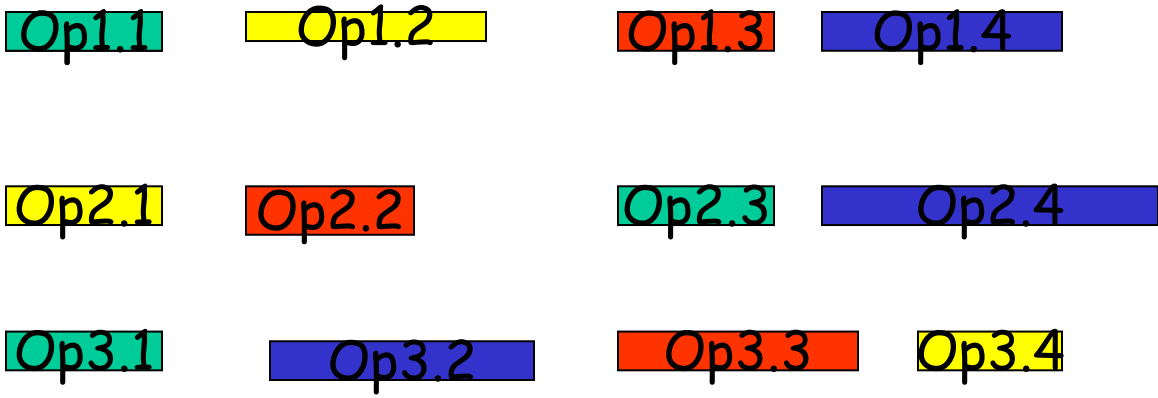
Assign a start time to each operation such that
(a) no two operations are in process on the same machine at the same time and
(b) temporal constraints are respected

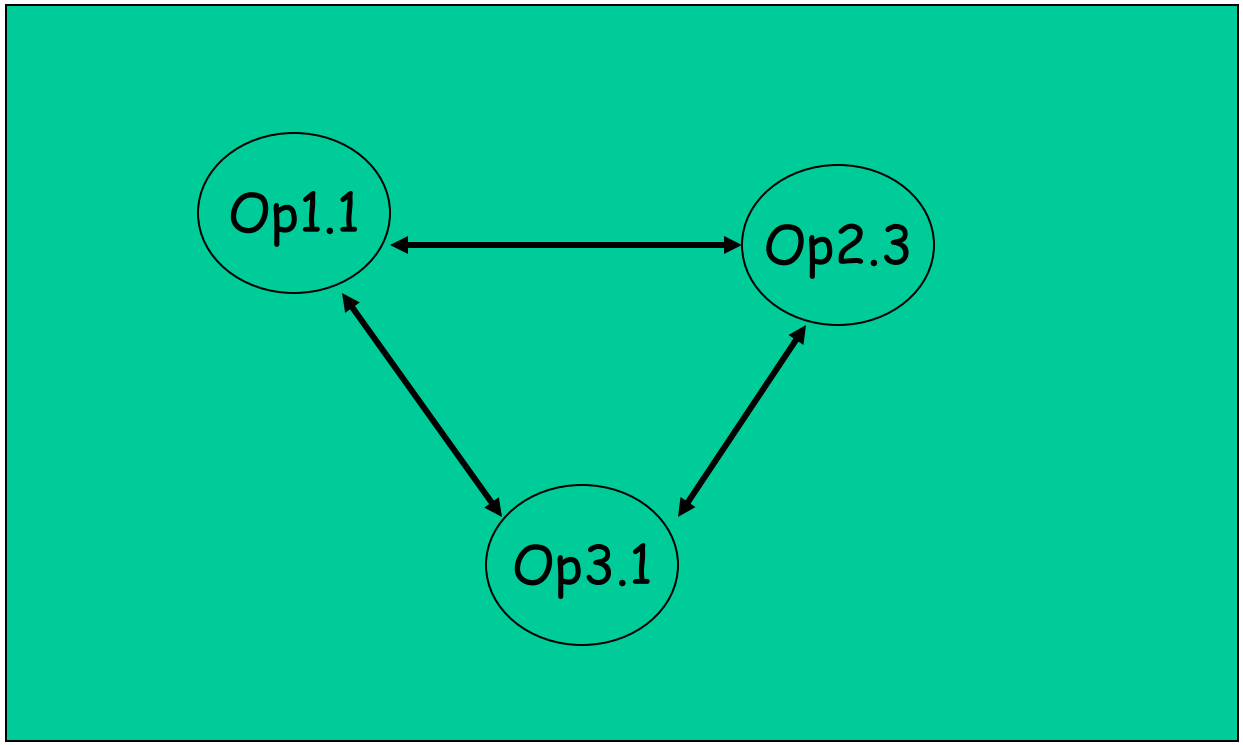
The problem is NP complete

Alternatively … sequence operations on resources

This gives a set of solutions, and might be considered a "least commitment approach"

An example: 3 x 4

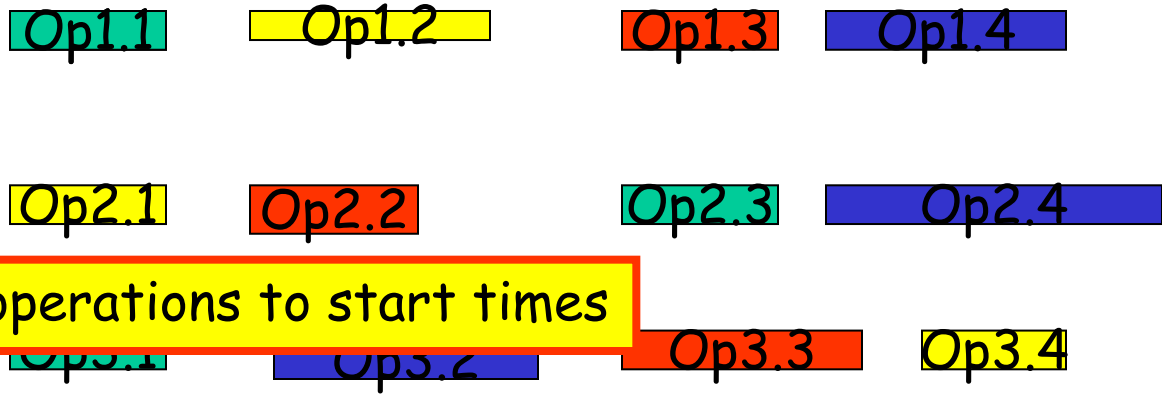




On the "green" resource, put a ***direction*** on the arrows

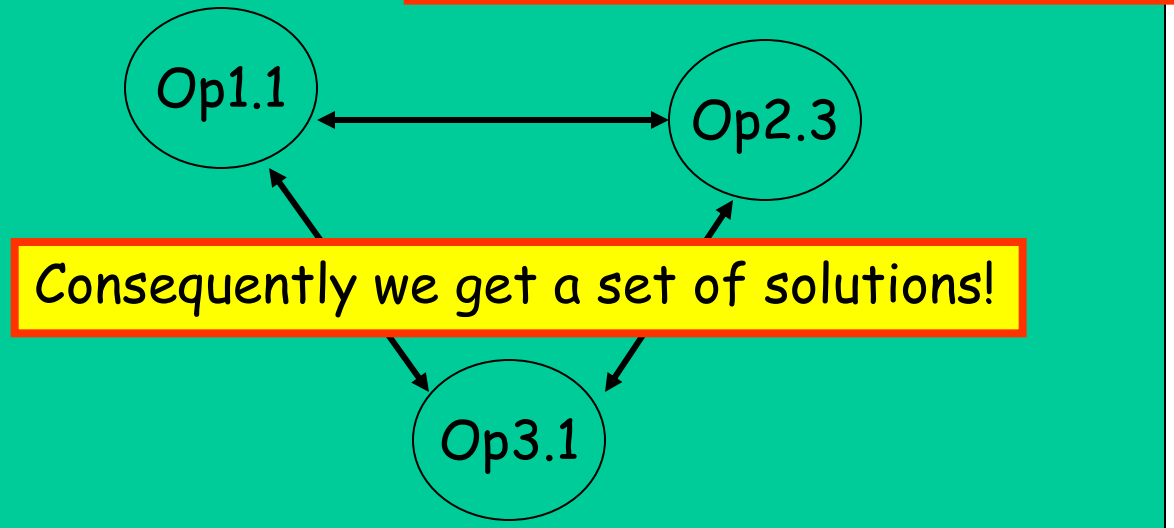
A disjunctive graph

An example: 3 x 4



We do not bind operations to start times

We take a least commitment approach



Consequently we get a set of solutions!

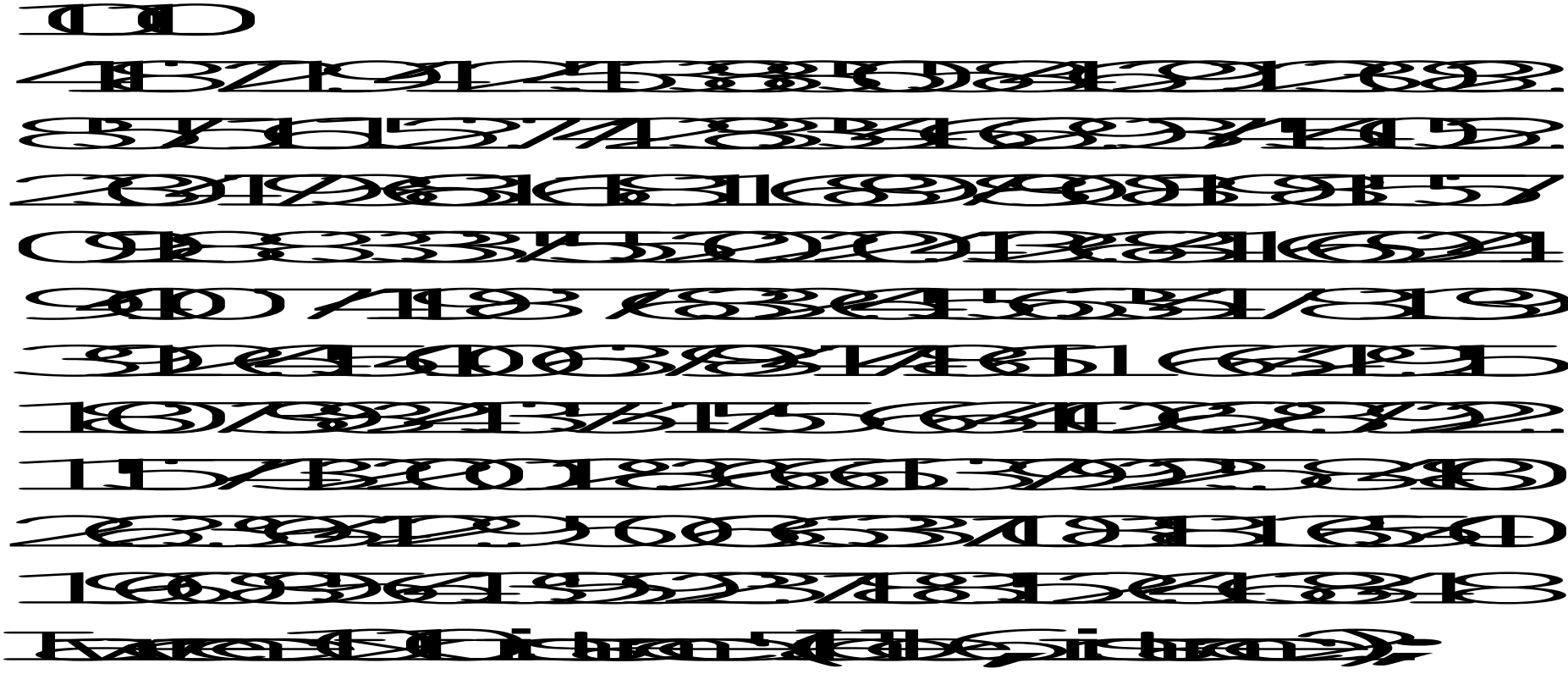
On the "green" resource, put a ***direction*** on the arrows

A disjunctive graph

7x6

```
76
210316375346
182540550034
253458091147
150525334859
291345540331
133359004421
244402375213
//
//a 7x6 jssp problem
//each row (except 7) is a job
//each job has operations (matrix column pair)
//each operation requires:
// — a matrix (0 to 5)
// — a duration (1 to 10)
//
//The problem is to find the shortest makespan
//(makespan is the time required to complete all jobs)
//
//minimum makespan is 57
```

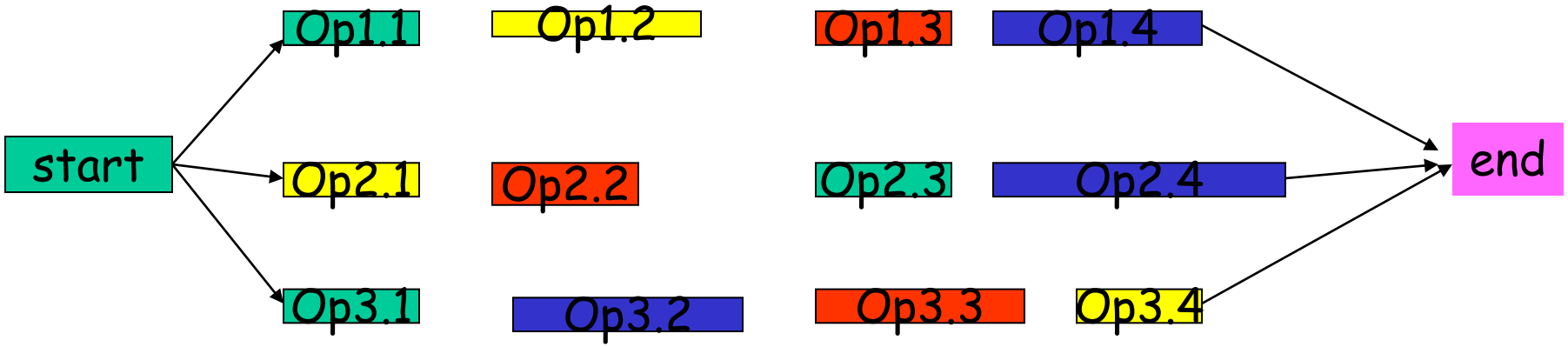
What is makespan!



For a long time, unsolved

Why bother?

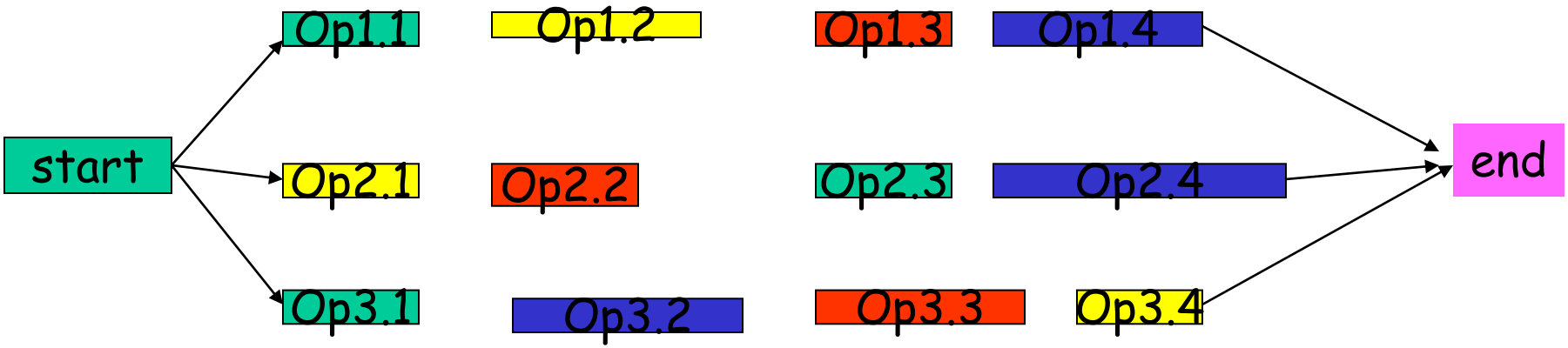
- Minimise makespan
    - what is makespan?
- Maximise start
    - JIT, minimise inventory levels
- minimise idle time on resources
    - maximise ROI
- ...



Find the smallest value for end
minimise makespan

How can we view this as a csp?

Each operation is a variable
domain is set of start times
there are precedence constraints between operation in a job
operations on a resource have ***disjunctive constraints***



Complexity

What is the complexity of this problem?

- Assume we have m resources and n jobs
- on each resource we will have n operations
- we can order these in n! ways
- therefore we have $O(n!^m)$ states to explore

$$O(n!^m)$$

But we want to optimise, not satisfy

How do you optimise with CP?

A sequence of decision problems

- Is there a solution with makespan 395?
  - Yip!
-
-

- Is there a solution with makespan 300?
  - Let me think about that ...
  - Yes
- Is there a solution with makespan 299?
  - Hold on, … , hold on
  - NO!
- Minimum makespan is 300.

When optimising, via a sequence of decision problems, will all decisions be equally difficult to answer?

What does branch and bound (BnB) do ?

Who cares about jobshop scheduling?

Manufacturing inc.

Is CP any good for this class of problem?

- Main competitor is OR technology
- Mark Fox 1980's,
  - ISIS constraint directed search(CMU)
- OPIS
  - Steve Smith, Peng Si Ow (CMU)
- DAS
  - Burke et al (SU)
- MicroBoss
  - Norman Sadeh (CMU)
- Edge-finding
  - a novel technique to infer constraints on a resources
    - J Carlier & E. Pinson 1994
  - CP solves open benchmarks, beats OR
- Texture based heuristics
  - J-C Beck and Mark Fox 1990's
- ILOG-Scheduler
  - Claude Le Pape, Wim Nuijten, Philippe Babtiste, J-Chris Beck
  - and many others
- 2000
  - ILOG buy CPLEX

Is CP any good for this class of problem?

- 2000
  - ILOG buy CPLEX
- …
- 2009
  - P. Vilim, edge-finding filtering
- …

File Edit View History Bookmarks Tools Help

Guy Martin: what it'... | Index of /~pat/cpM... | job shop schedulin... | Limited discrepanc...

cpaior2015.uconn.edu

UCONN | UNIVERSITY OF CONNECTICUT

CPAIOR 2015

Home | Conference Schedule | Conference Details | Program | Local Info

CPAIOR 2015

The Twelfth International Conference on Integration of Artificial Intelligence (AI) and Operations Research

The aim of the conference series is to bring together interested researchers from constraint programming in the intersection of these fields and to provide an opportunity for researchers in one area to learn about the opportunity to show how the integration of techniques from different fields can lead to interesting res approaches from more than one of the areas are especially solicited. High quality papers from a single ar challenging applications or experience reports on such applications are strongly encouraged.

After a successful series of five CPAIOR international workshops in Ferrara (Italy), Paderborn (Germany), A conference. More than 100 participants attended the first meeting held in Nice (France). In the subsequen Pittsburgh (USA), Bologna (Italy), Berlin (Germany), Nantes (France) and Yorktown Heights (USA).

See the official webpage of the CPAIOR conference series for more information.

BARCELONA, SPAIN

File Edit View History Bookmarks Tools Help

Guy Martin: wh... | Index of /~pat/... | job shop sched... | Limited discrep... | JEA_16_1-6.dvi - a1_... | Petr Vilim Homepage | Journal of Sche... | Home | CPAIOR 2015 | 25th Internation...

icaps15.icaps-conference.org

icaps Jerusalem 2015

Jerusalem, Israel
June 7-11, 2015

Home | Technical Program | Workshops | Tutorials | Special Tracks | Demos | Doctoral Consortium | Registration | Accommodation

Conference Info | Co-located Events | Committees

Welcome

ICAPS 2015, the 25th International Conference on Automated Planning and Scheduling will take place in **Jerusalem, Israel, June 7-11, 2015**.

ICAPS 2015 is part of the ICAPS conference series. ICAPS is the premier forum for exchanging news and research results on theory and applications of intelligent planning and scheduling technology.

The conference features a pre-conference program of workshops and tutorials on current research topics. The main technical program consists of invited talks by leading scientists working in the area, presentations of technical papers, as well as system demonstrations. For graduate students the pre-conference program includes a Doctoral Consortium.

The detailed conference schedule is available here.

The program leaflet (pdf) is available here.

The social events include a reception on Monday night at the Tower of David followed by the Night Spectacular sound and lights show, a walking tour of the old city of Jerusalem Tuesday night, and the Banquet at Beit-Shmuel's banquet hall overlooking the old city of Jerusalem.

ICAPS-2015 will be co-located with the 8th Annual Symposium on Combinatorial Search (SOCS-15) including a joint session on Thursday June, 11.

A free shuttle bus will take SOCS participants attending the joint ICAPS-SOCS session to the SOCS venue in Ein Gedi on Thursday afternoon.

An advertising flier (PDF) for the conference is available here.

Important Dates

News and Updates

[01 June 2015] The program leaflet (pdf) is available.

[22 May 2015] All workshops proceedings are now available.

[27 May 2015] The DC schedule is now available.

[22 May 2015] All workshops schedules are now available.

[20 May 2015] The prereception optional walking tours registration is now open. See here the details.

[20 May 2015] Detailed information on the welcome reception and prereception optional walking tours is now available here.

[20 May 2015] The list of posters and demos is now available in the detailed conference schedule. Please note poster board size is 90cm x 150cm.

[13 May 2015] The detailed conference schedule is now available in an online

File Edit View History Bookmarks Tools Help

Guy Martin: what it's lik... | Index of /~pat/cpM/cho... | job shop scheduling co... | Limited discrepancy sea...

link.springer.com/journal/10951

Springer Link

Search

Home • Contact Us

» Browse Volumes & Issues

Journal of Scheduling

ISSN: 1094-6136 (Print) 1099-1425 (Online)

Description

The *Journal of Scheduling* provides a global forum for the publication of all forms of scheduling research. It is the only peer reviewed journal with broad coverage of the techniques and applications of scheduling that spans several distinct disciplines.

Readers facing complex scheduling problems can turn to the journal to find the latest advances in the field. Each issue features new and novel techniques, applications, theore ... show all

Find your Volume or Issue | Browse all Content

Volume | Issue | Browse Volumes & Issues

Latest Articles

OriginalPaper
An efficient algorithm for semi-online multiprocessor scheduling with given total processing time
Hans Kellerer, Vladimir Kotov, Michaël Gabay (December 2015)
» Download PDF (536KB) » View Article

OriginalPaper
Identical coupled task scheduling: polynomial complexity of the cyclic case
Vassilissa Lehoux-Lebacque, Nadia Brauner, Gerd Finke (December 2015)

Look Inside

Impact Factor	Available
1.028	2003 - 2015
Volumes	Issues
13	77
Articles	Open Access
592	24 Articles

Other actions

» Register for Journal Updates

» About This Journal

Share

GEORGE BOOLE* 200

CP 2015 ∧ ICLP 2015

Cork, Ireland

Variants of jsp

- openness:
  - variety of resources can perform an operation
  - processing time dependant on resource used
- set up costs, between jobs (transition cost)
- consumable resources
  - such as gas, oil, etc
- pre-emption
  - can stop and restart an operation
- resource can perform multiple operations simultaneously
  - batch processing
- secondary resources
  - people, tools, cranes, etc
- etc

Chris Beck (2006) "The jssp has never been spotted in the wild."

Why might CP be technology of choice for scheduling?

- can model rich real-world problems
  - addition of side constraints etc
- incorporate domain knowledge
  - in the form of variable and value ordering heuristics
- powerful reasoning/inference allied to novel search techniques

We can get a solution up and running quickly

File Edit View History Bookmarks Tools Help

Index of /~pat/cpM/choco...

www.dcs.gla.ac.uk/~pat/cpM/choco3/cpM/jssp/

Index of /~pat/cpM/choco3/cpM/jssp

Name	Last modified	Size	Description
Parent Directory		-	
code/	29-Oct-2015 17:06	-	
papers/	28-Oct-2015 15:38	-	
problems/	29-Oct-2015 17:01	-	
results/	30-Oct-2015 04:33	-	
slides/	28-Oct-2015 15:38	-	

Apache/2.2.3 (CentOS) Server at www.dcs.gla.ac.uk Port 80

Operation

Operation

Operation - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Solver solver;


    Operation(String id,String resId,int duration,int latestStart,Solver solver){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        start         = VF.bounded(id,0,latestStart,solver);
    }

    public String toString(){
        return "{" + id + " " + resId + " " + duration + " [" + start.getLB() + "," + start.getUB() + "]}";
    }

    Constraint before(Operation op2){
        return ICF.arithm(op2.start,">=",start,"+",duration);
    }
    //
    // this operation is before operation op2
    // Therefore, if this operation starts at time t and has duration d
    // it finishes at time t+d. Therefore operation op2 can start immediately
    // at time t+d or any time after that
    //
}
```

Operation

Operation - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Solver solver;


    Operation(String id,String resId,int duration,int lates
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        start         = VF.bounded(id,0,latestStart,solver);
    }

    public String toString(){
        return "{" + id + " " + resId + " " + duration + " [" + start.getLB() + "," + start.getUB() + "]}";
    }

    Constraint before(Operation op2){
        return ICF.arithm(op2.start,">=",start,"+",duration);
    }
    //
    // this operation is before operation op2
    // Therefore, if this operation starts at time t and has duration d
    // it finishes at time t+d. Therefore operation op2 can start immediately
    // at time t+d or any time after that
    //
}
```

Operation has

- a duration
- a scheduled start time
- is on a resource

Operation

Operation - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Solver solver;

    Operation(String id,String resId,int duration,int latestStart,Solver solver){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        start         = VF.bounded(id,0,latestStart,solver);
    }

    public String toString(){
        return "{" + id + " " + resId + " " + duration + " [" + start.getLB() + "," + start.getUB() + "]}";
    }

    Constraint before(Operation op2){
        return ICF.arithm(op2.start,">=",start,"+",duration);
    }
    //
    // this operation is before operation op2
    // Therefore, if this operation starts at time t and has duration d
    // it finishes at time t+d. Therefore operation op2 can start immediately
    // at time t+d or any time after that
    //
}
```

Operation

Operation - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Solver solver;


    Operation(String id,String resId,int duration,int latestStart,Solver solver){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        start         = VF.bounded(id,0,latestStart,solver);
    }

    public String toString(){
        return "{" + id + " " + resId + " " + duration + " [" + start.getLB() + "," + start.getUB() + "]}";
    }

    Constraint before(Operation op2){
        return ICF.arithm(op2.start,">=",start,"+",duration);
    }
    //
    // this operation is before operation op2
    // Therefore, if this operation starts at time t and has duration d
    // it finishes at time t+d. Therefore operation op2 can start immediately
    // at time t+d or any time after that
    //
}
```

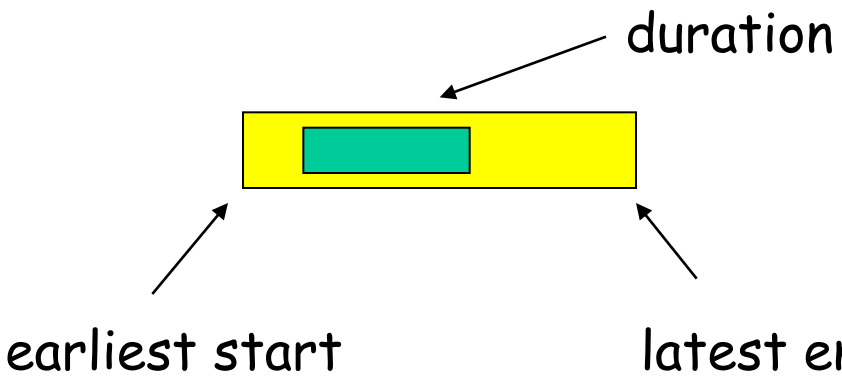
Operation

```
Constraint before(Operation op2){
    return ICF.arithm(op2.start,">=",start,"+",duration);
}
//
// this operation is before operation op2
// Therefore, if this operation starts at time t and has duration d
// it finishes at time t+d. Therefore operation op2 can start immediately
// at time t+d or any time after that
//
```

see next slides

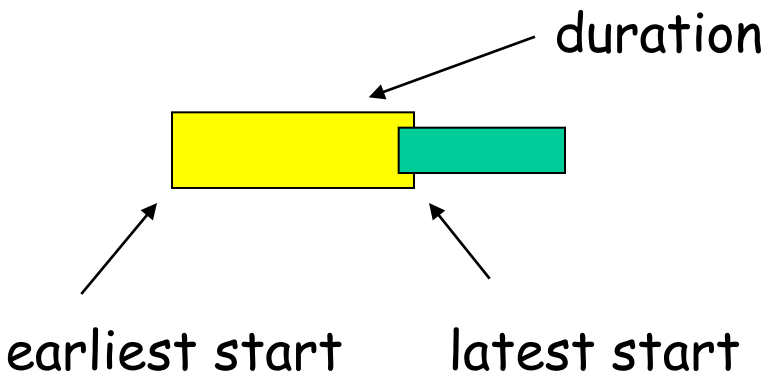
Picture of an operation

op1.before(op2)



Picture of an operation

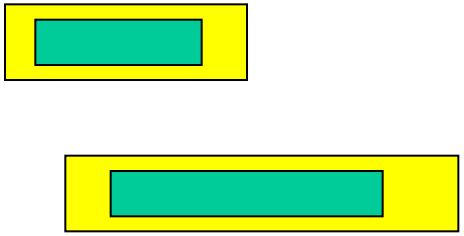
op1.before(op2)



Constrained integer variable represents start time

Picture of an operation

op1.before(op2)



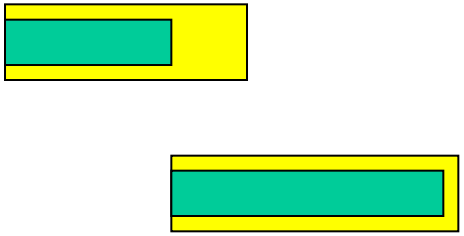
op1

op2

op1.before(op2) $\longrightarrow$ op1.start() + op1.duration() $\leq$ op2.start()

Picture of an operation

op1.before(op2)



op1

op2

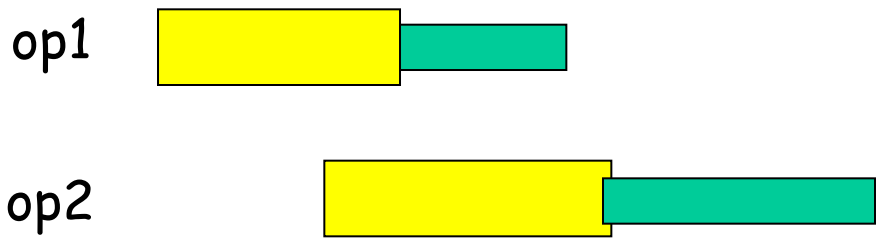
propagate

op1.before(op2) $\longrightarrow$ op1.start() + op1.duration() $\leq$ op2.start()

Update earliest start of operation op2

Picture of an operation

op1.before(op2)



propagate

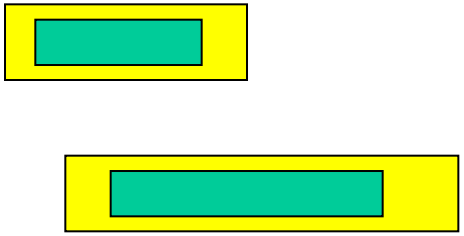
op1.before(op2) $\longrightarrow$ op1.start() + op1.duration() $\leq$ op2.start()

Update latest start of operation op1

No effect on this instance

Picture of an operation

op1.before(op2)



op1

op2

op1 and op2 cannot be in process at same time

→ op1.before(op2) ***OR*** op2.before(op1)

Not easy to propagate until decision made (disjunction broken)

File Edit View History Bookmarks Tools Help

Index of /~pat/cpM/choco... | Allen's interval algebra - W...

https://en.wikipedia.org/wiki/Allen's_interval_algebra

Create account Not logged in Talk Contributions Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Allen's interval algebra

From Wikipedia, the free encyclopedia

For the type of boolean algebra called interval algebra, see Boolean algebra (structure)

Allen's interval algebra is a calculus for temporal reasoning that was introduced by James F. Allen in 1983.

The calculus defines possible relations between time intervals and provides a composition table that can be used as a basis for reasoning about temporal descriptions of events.

Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Français
Edit links

Relations [edit]

The following 13 base relations capture the possible relations between two intervals.

Relation	Illustration	Interpretation
$X < Y$ $Y > X$	X Y	X takes place before Y
$X\ \mathbf{m}\ Y$ $Y\ \mathbf{mi}\ X$	X Y	X meets Y (*i* stands for *inverse*)
$X\ \mathbf{o}\ Y$ $Y\ \mathbf{oi}\ X$	X Y	X overlaps with Y
$X\ \mathbf{s}\ Y$ $Y\ \mathbf{si}\ X$	X Y	X starts Y
$X\ \mathbf{d}\ Y$ $Y\ \mathbf{di}\ X$	X Y	X during Y
$X\ \mathbf{f}\ Y$ $Y\ \mathbf{fi}\ X$	X Y	X finishes Y
$X = Y$	X Y	X is equal to Y

Using this calculus, given facts can be formalized and then used for automatic reasoning. Relations between intervals are formalized as sets of base relations.

The sentence

During dinner, Peter reads the newspaper. Afterwards, he goes to bed.

Job

Job

Job - Notepad

File Edit Format View Help

```
import java.util.ArrayList;
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Job {
    String id;
    ArrayList<Operation> operations;
    int length;
    Solver solver;

    Job(String id,Solver solver){
        this.id     = id;
        operations  = new ArrayList<Operation>();
        length      = 0;
        this.solver = solver;
    }

    void add(Operation op){
        if (!operations.isEmpty())
            solver.post(operations.get(length-1).before(op));
        operations.add(op);
        length++;
    }

    Operation get(int i){return operations.get(i);}

    public String toString(){
        String s = "(" + id + " ";
        for (int i=0;i<operations.size();i++) s = s + ((Operation)operations.get(i)).toString() + " ";
        return s + ")";
    }
}
```

Job

```
public class Job {
    String id;
    ArrayList<Operation> operations;
    int length;
    Solver solver;
```

```
  Job(String id,Solver solver){
      this.id      = id;
      operations   = new ArrayList<Operation>();
      length       = 0;
      this.solver  = solver;
  }
```

Job is a sequence of operations

Job

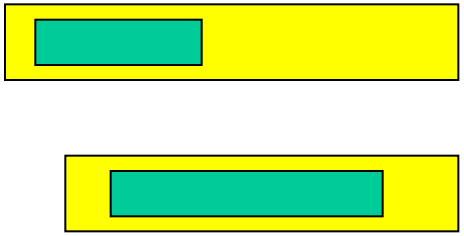
```
void add(Operation op){
    if (!operations.isEmpty())
        solver.post(operations.get(length-1).before(op));
    operations.add(op);
    length++;
}
```

Creating/building a job as a sequence of operations each one ***before*** the other

Decision

Picture of an operation

op1.before(op2)



op1

op2

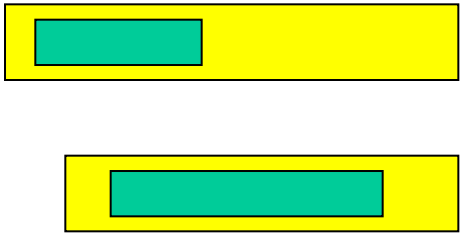
Use a 0/1 decision variable d[i][j] as follows

d[i][j] = 0 → op[i]1.before(op[j])

d[i][j] = 1 → op[j]1.before(op[i])

Picture of an operation

op1.before(op2)



op1

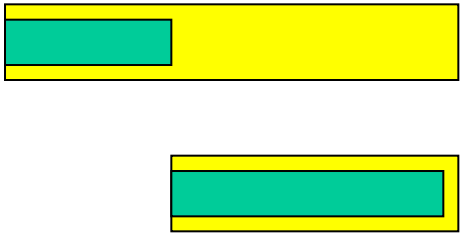
op2

d[i][j] = 0 → op[i]1.before(op[j])

op1 before op2

Picture of an operation

op1.before(op2)



op1

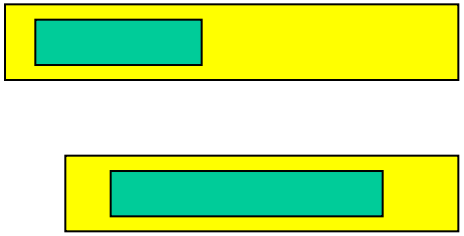
op2

d[i][j] = 0 → op[i]1.before(op[j])

op1 before op2

Picture of an operation

op1.before(op2)



op1

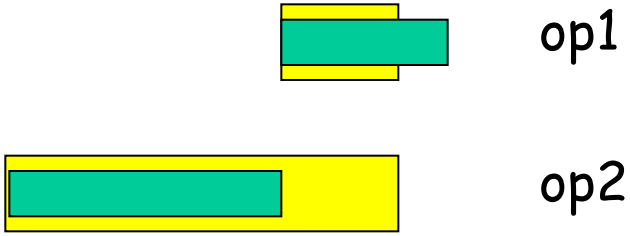
op2

d[i][j] = 1 → op[j]1.before(op[i])

op2 before op1

Picture of an operation

op1.before(op2)



d[i][j] = 1 → op[j]1.before(op[i])

op2 before op1

Decision

Decision - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.variables.*;

public class Decision {
    IntVar d;
    Operation op_i;
    Operation op_j;

    Decision(IntVar d,Operation op_i,Operation op_j){
        this.d    = d;
        this.op_i = op_i;
        this.op_j = op_j;
    }

    public String toString(){return "{" + d + " " + op_i +  " " + op_j + "}";}

    IntVar getD(){return d;}
    Operation getOp_i(){return op_i;}
    Operation getOp_j(){return op_j;}
}
//
// A Decision is a triple where 0/1 variable d decides
//  the ordering between two operations on a resource
//
```

Decision

```
public class Decision {
    IntVar d;
    Operation op_i;
    Operation op_j;

    Decision(IntVar d,Operation op_i,Operation op_j){
        this.d    = d;
        this.op_i = op_i;
        this.op_j = op_j;
    }
```

A decision is a triple:

- a zero/one variable d
- an operation op_i
- an operation op_j

Value of d decides relative order of The two operations (before or after)

Resource

Resource

Resource - Notepad

File Edit Format View Help

```
import java.util.ArrayList;
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class Resource {
    String id;
    ArrayList<Operation> operations;
    ArrayList<Decision> decisions;
    Solver solver;

    Resource(String id,Solver solver){
        this.id     = id;
        operations  = new ArrayList<Operation>();
        decisions   = new ArrayList<Decision>();
        this.solver = solver;
    }

    void add(Operation op){
        int n = operations.size();
        for (int i=0;i<n;i++){
            Operation op_i = operations.get(i);
            IntVar decision = VF.enumerated("dec_"+ i +","+ n,0,1,solver);
            decisions.add(new Decision(decision,op_i,op));
            LCF.ifThen(ICF.arithm(decision,"=",0),op_i.before(op)); // decision = 0 -> op_i before op
            LCF.ifThen(ICF.arithm(decision,"=",1),op.before(op_i)); // decision = 1 -> op before op_i
        }
        operations.add(op);
    }

    public String toString(){return "Res: " + id +
                                    " NoOps: " + operations.size() +
                                    " NoDecVars: " + decisions.size();}

}
```

Resource

```
public class Resource {
    String id;
    ArrayList<Operation> operations;
    ArrayList<Decision> decisions;
    Solver solver;

    Resource(String id,Solver solver){
        this.id      = id;
        operations   = new ArrayList<Operation>();
        decisions    = new ArrayList<Decision>();
        this.solver  = solver;
    }
```

Resource is a collection of operations and decisions that will be made on their ordering/sequencing on this resource

Resource

```
void add(Operation op){
    int n = operations.size();
    for (int i=0;i<n;i++){
        Operation op_i = operations.get(i);
        IntVar decision = VF.enumerated("dec_"+ i +","+ n,0,1,solver);
        decisions.add(new Decision(decision,op_i,op));
        LCF.ifThen(ICF.arithm(decision,"=",0),op_i.before(op)); // decision = 0
        LCF.ifThen(ICF.arithm(decision,"=",1),op.before(op_i)); // decision = 1
    }
    operations.add(op);
}
```

Add an operation to a resource and then constrain it …

Resource

```
void add(Operation op){
    int n = operations.size();
    for (int i=0;i<n;i++){
        Operation op_i = operations.get(i);
        IntVar decision = VF.enumerated("dec_"+ i +","+ n,0,1,solver);
        decisions.add(new Decision(decision,op_i,op));
        LCF.ifThen(ICF.arithm(decision,"=",0),op_i.before(op)); // decision = 0
        LCF.ifThen(ICF.arithm(decision,"=",1),op.before(op_i)); // decision = 1
    }
    operations.add(op);
}
```

decision = 0 implies op_i before op
decision = 1 implies op before op_i

JSSP

JSSP - Notepad

File Edit Format View Help

JSSP

```java
import java.util.*;
import java.io.*;
import org.chocosolver.solver.*;
import org.chocosolver.solver.variables.*;
import org.chocosolver.solver.constraints.*;

public class JSSP {
    String id;                      // file name
    int n;                          // number of jobs
    int m;                          // number of resources
    int dueDate;                    // aka makespan
    ArrayList<Job> jobs;            // jobs to complete
    ArrayList<Resource> resources;  // resources to use
    Operation endOp;                // last operation for ALL jobs!
    Solver solver;

    JSSP(String fname,int dueDate) throws IOException {
        Scanner sc        = new Scanner(new File(fname));
        id                = fname;
        n                 = sc.nextInt(); // number of jobs
        m                 = sc.nextInt(); // number of resources
        jobs              = new ArrayList<Job>();
        resources         = new ArrayList<Resource>();
        this.dueDate      = dueDate;
        solver            = new Solver("id");
        endOp             = new Operation("endOp","nullRes",0,dueDate,solver);
        int totalDuration = 0;
        for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
        for (int i=0;i<n;i++){
            Job job = new Job("job_"+i,solver);
            for (int j=0;j<m;j++){
                Resource resource = resources.get(sc.nextInt());
                int duration      = sc.nextInt();
                totalDuration     = totalDuration + duration;
                Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
                resource.add(operation);
                job.add(operation);
            }
            job.add(endOp);
            jobs.add(job);
        }
        solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
        sc.close();
    }

    public  String toString(){
        String s = "JSSP " + n + "X" + m + "\n";
        for (int i=0;i<jobs.size();i++){
            Job job = (Job)jobs.get(i);
            s = s + job  + "\n";
        }
        return s;
    }

    IntVar getMakeSpan(){return endOp.start;}

    Decision[] getDecisions(){
        Decision[] decisions = new Decision[((n * (n-1))/2) * m];
        for (int i=0,k=0;i<m;i++)
            for (Decision decision : resources.get(i).decisions)
                decisions[k++] = decision;
        return decisions;
    }
    //
```

JSSP

```
public class JSSP {
    String id;                          // file name
    int n;                              // number of jobs
    int m;                              // number of resources
    int dueDate;                        // aka makespan
    ArrayList<Job> jobs;                // jobs to complete
    ArrayList<Resource> resources;      // resources to use
    Operation endOp;                    // last operation for ALL jobs!
    Solver solver;
```

A jssp is a collection of jobs and resources

JSSP

```
JSSP(String fname,int dueDate) throws IOException {
    Scanner sc        = new Scanner(new File(fname));
    id                = fname;
    n                 = sc.nextInt(); // number of jobs
    m                 = sc.nextInt(); // number of resources
    jobs              = new ArrayList<Job>();
    resources         = new ArrayList<Resource>();
    this.dueDate      = dueDate;
    solver            = new Solver("id");
    endOp             = new Operation("endOp","nullRes",0,dueDate,solver);
    int totalDuration = 0;
    for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
    for (int i=0;i<n;i++){
        Job job = new Job("job_"+i,solver);
        for (int j=0;j<m;j++){
            Resource resource = resources.get(sc.nextInt());
            int duration      = sc.nextInt();
            totalDuration     = totalDuration + duration;
            Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
            resource.add(operation);
            job.add(operation);
        }
        job.add(endOp);
        jobs.add(job);
    }
    solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
    sc.close();
}
```

JSSP

```
JSSP(String fname,int dueDate) throws IOException {
    Scanner sc          = new Scanner(new File(fname));
    id                  = fname;
    n                   = sc.nextInt(); // number of jobs
    m                   = sc.nextInt(); // number of resources
    jobs                = new ArrayList<Job>();
    resources           = new ArrayList<Resource>();
    this.dueDate        = dueDate;
    solver              = new Solver("id");
    endOp               = new Operation("endOp","nullRes",0,dueDate,solver);
```

JSSP

```
JSSP(String fname,int dueDate) throws IOException {
    Scanner sc          = new Scanner(new File(fname));
    id                  = fname;
    n                   = sc.nextInt(); // number of jobs
    m                   = sc.nextInt(); // number of resources
    jobs                = new ArrayList<Job>();
    resources           = new ArrayList<Resource>();
    this.dueDate        = dueDate;
    solver              = new Solver("id");
    endOp               = new Operation("endOp","nullRes",0,dueDate,solver);
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

JSSP

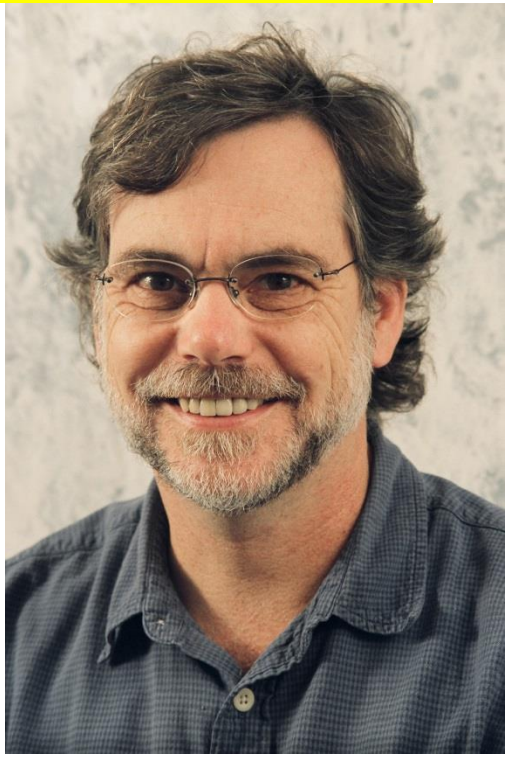
```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,solver));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,solver);
    for (int j=0;j<m;j++){
        Resource resource = resources.get(sc.nextInt());
        int duration      = sc.nextInt();
        totalDuration     = totalDuration + duration;
        Operation operation = new Operation("op_"+i+"_"+j,resource.id,duration,dueDate,solver);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
solver.post(ICF.arithm(endOp.start,"<=",totalDuration));
sc.close();
```

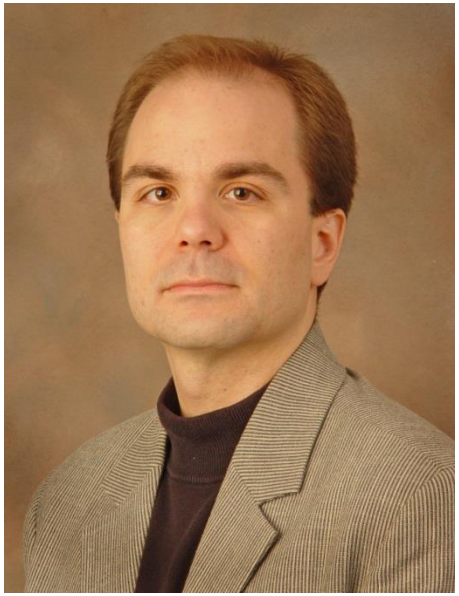
DecisionProblem

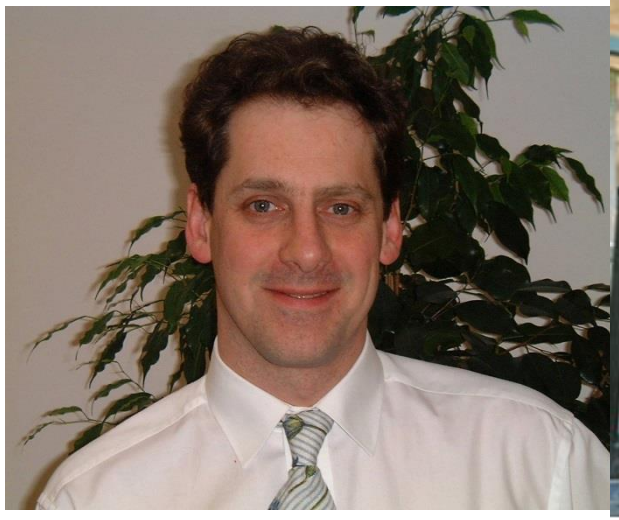
DecisionProblem

```
public class DecisionProblem {

  public static void main(String[] args) throws FileNotFoundException, IOException {
      int dueDate            = Integer.parseInt(args[1]);
      JSSP jssp              = new JSSP(args[0],dueDate);
      Solver solver          = jssp.solver;
      Decision[] decisions   = jssp.getDecisions();
      int n                  = decisions.length;
      IntVar makeSpan        = jssp.getMakeSpan();

      solver.set(ISF.lexico_LB(jssp.getDecisionIntVars()));

      System.out.println("solved: " + solver.findSolution());

      System.out.println(makeSpan + " ["+ makeSpan.getLB() +","+ makeSpan.getUB() +"]");
      System.out.println("nodes: " + solver.getMeasures().getNodeCount() +
                         "  cpu: " + solver.getMeasures().getTimeCount());
  }
}
```

Wot!? No heuristics!?!!!







Maintaining Singleton Arc Consistency

Patrick Prosser[1] and Christophe Lecoutre[2]

[1]Department of Computing Science
University of Glasgow
Scotland
pat@cs.gla.ac.uk

[2]CRIL-CNRS FRE 2499,
Université d'Artois
Lens, France
lecoutre@cril.univ-artois.fr

Abstract. Singleton Arc-Consistency (SAC) is a simple and strong level of consistency, but is expensive to enforce. To date, research has focused on improving the performance of algorithms that achieve SAC. These algorithms tend to be somewhat complex and non-trivial to implement. Furthermore, these algorithms have mostly been tested out as a preprocessing step before actually solving a problem. Here, we show how a basic and simple SAC algorithm can be incorporated into a constraint programming toolkit and then used within search. We then propose limited degrees of SAC, in particular SAC on the upper and lower bounds of variables (Bound-SAC) and SAC on only the first value in domains (First-SAC). We investigate the effects of limiting the amount of SAC maintained whilst solving a problem instance, i.e. by limiting the values in domains that are made SAC and the variables that are made SAC. Our studies show that judicious use of SAC within the search process can indeed pay off.

Instance	MAC	Maintaining		
		B-SAC$_{dn}$	B-SAC$_{st}$	B-SAC
la01	666	666	666	666
la02	655	655	655	655
la03	653	597	603	603
la04	628	598	590	590
la05	593	665	665	665
la06	1245	1146	1233	1237
la07	1214	897	1336	1359
la08	1161	1084	1400	1393
la09	1498	1049	1527	1520
la10	1658	972	1192	1259
la11	1453	1787	–	–
la12	1467	1504	–	–
la13	2899	2310	–	–
la14	1970	1784	–	–
la15	2368	2200	–	–

Table 1. Cost of best solution found for Lawrence job-shop scheduling instances, given 10 minutes CPU

5 A study of Golomb rulers

In [16], experiments were performed on Golomb rulers. In particular, given the length l