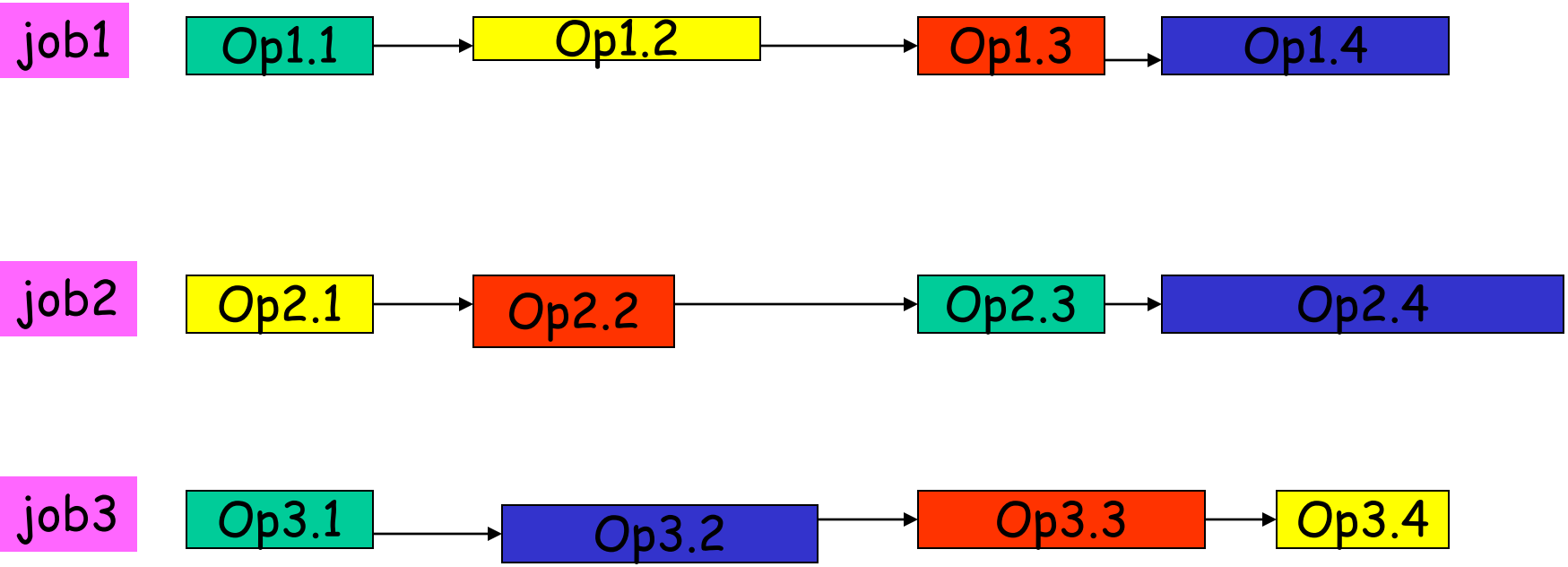
jobshop scheduling

We have

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

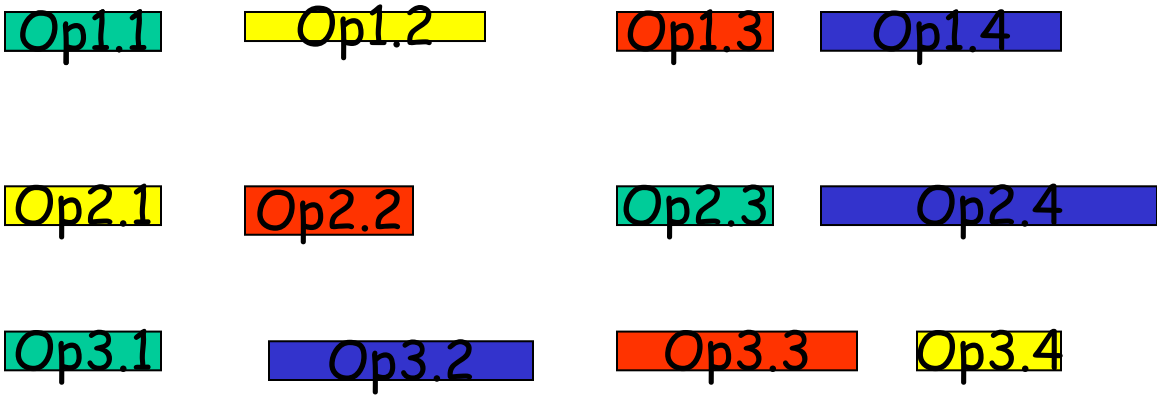
- sequence the activities on the resources

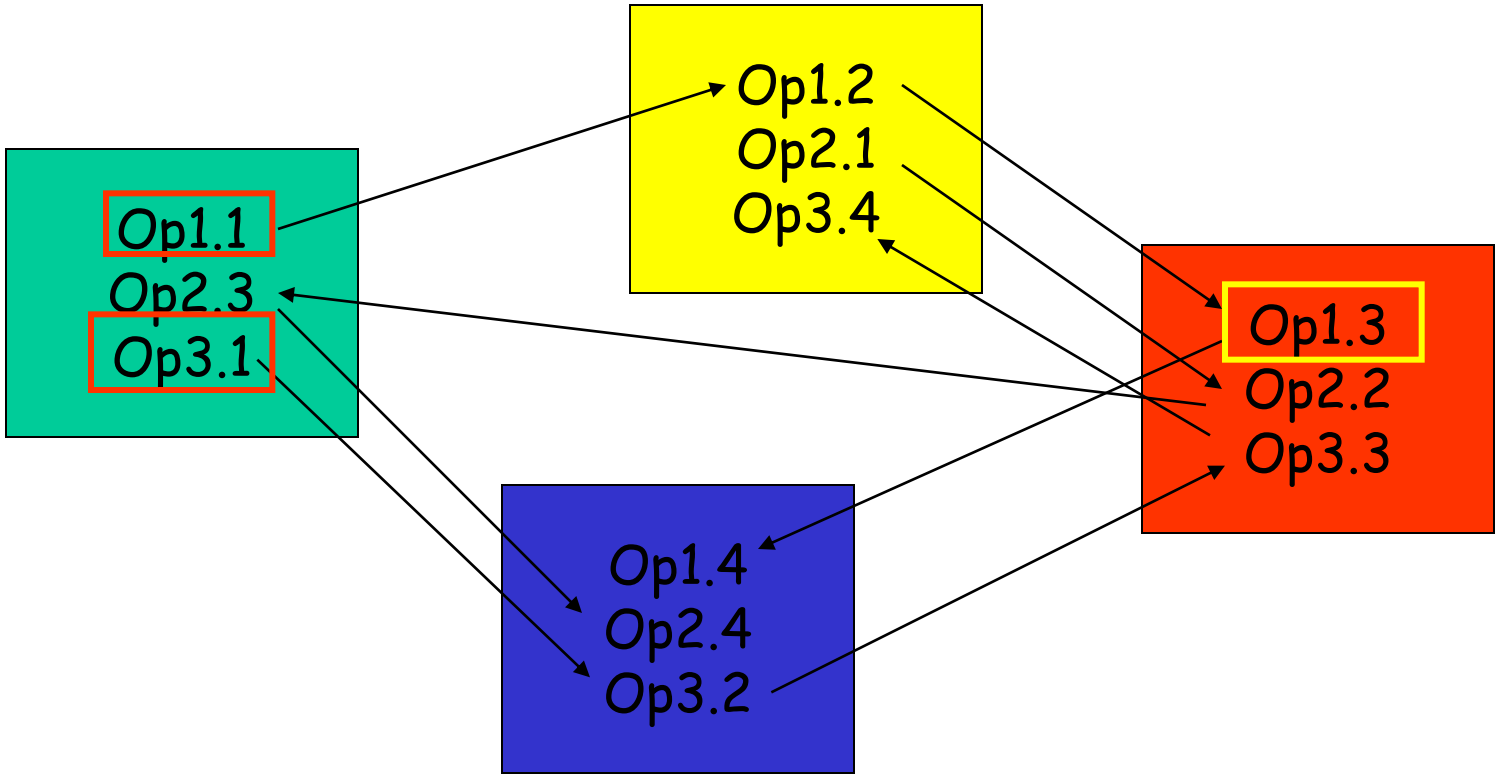
An example: 3 x 4



- 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





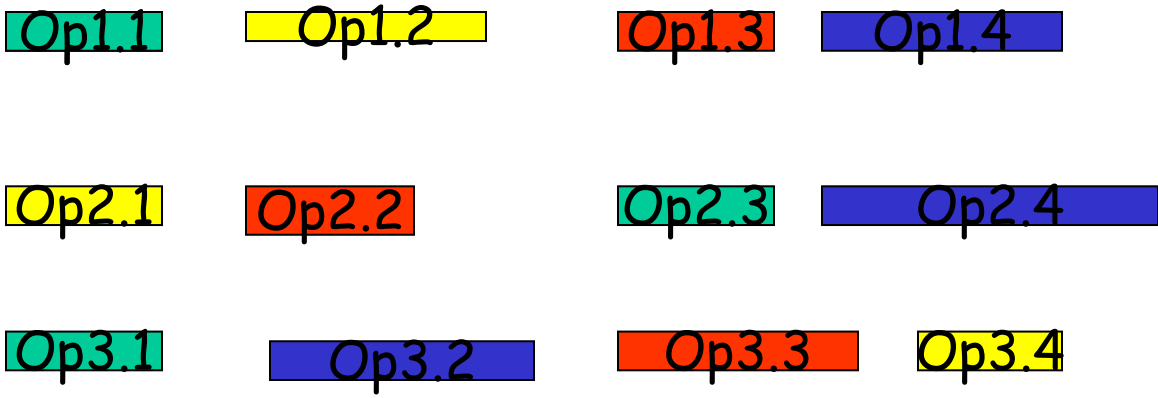
Assign a start time to each operation such that
(a) no two operations are in process on the same machine at the same time and
(b) temporal constraints are respected

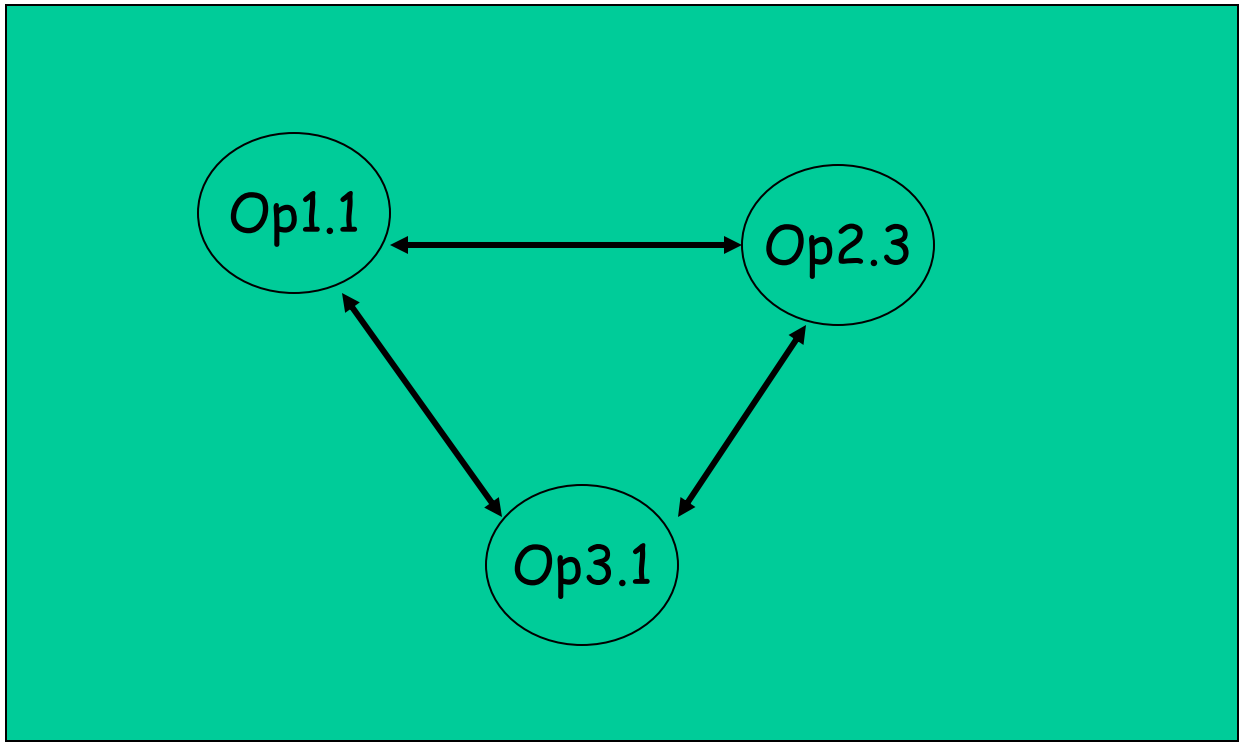
The problem is NP complete

Alternatively … sequence operations on resources

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

An example: 3 x 4

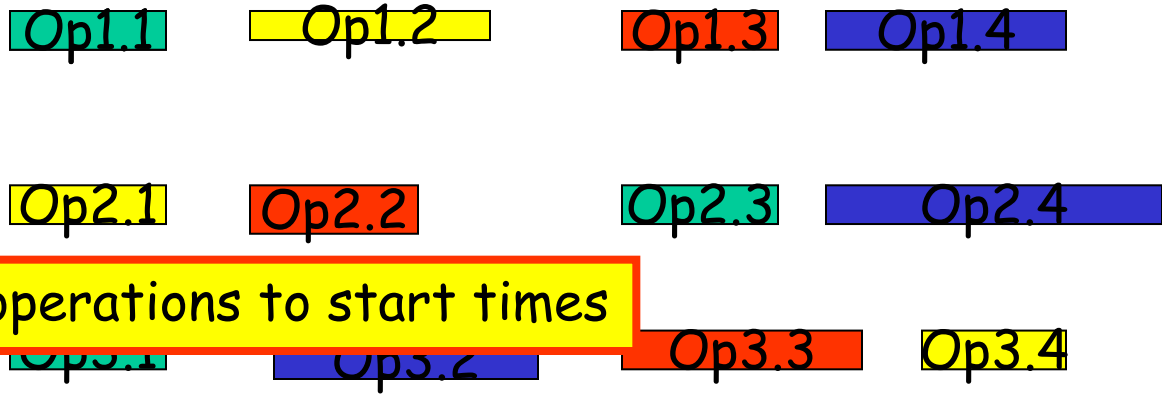




On the "green" resource, put a ***direction*** on the arrows

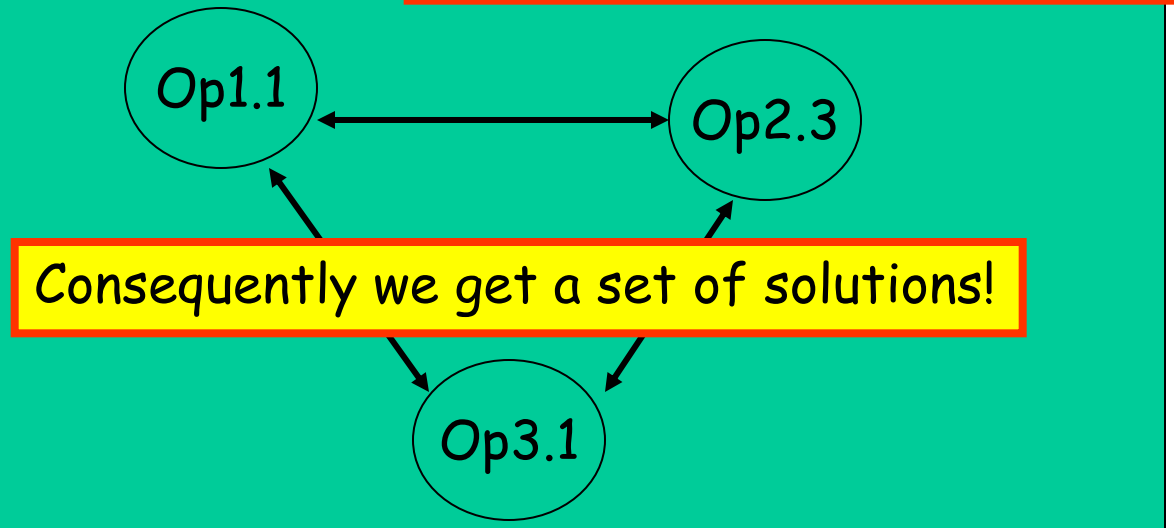
A disjunctive graph

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!

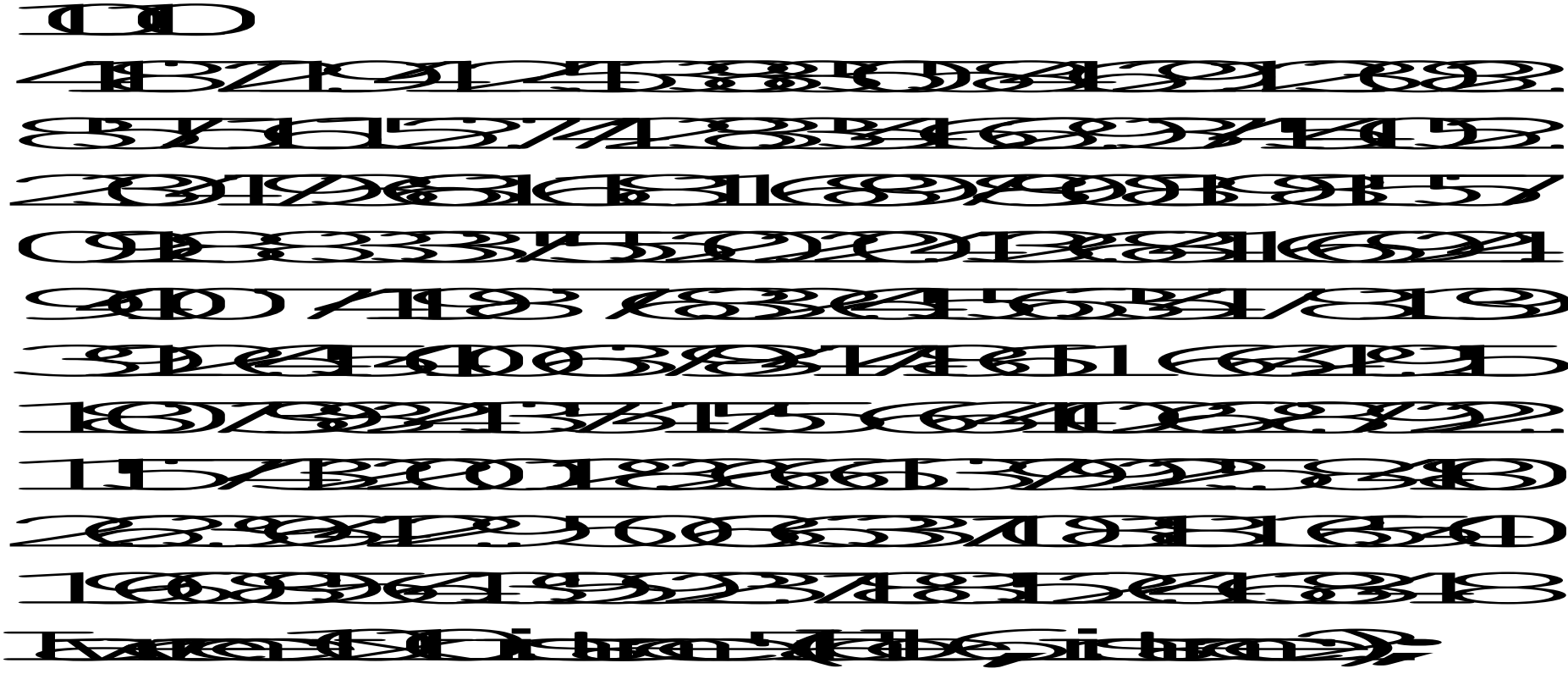
On the "green" resource, put a ***direction*** on the arrows

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
```

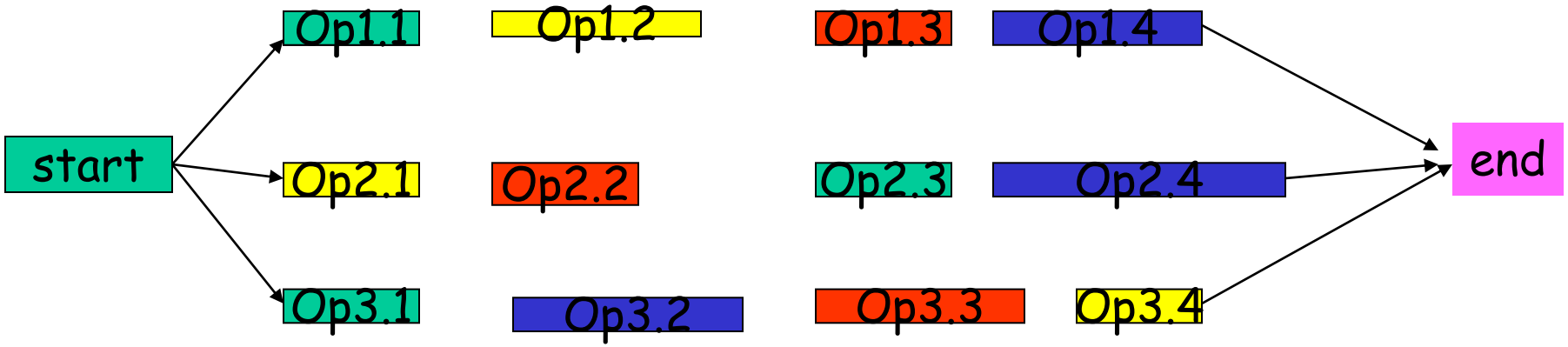
What is makespan!



For a long time, unsolved

Why bother?

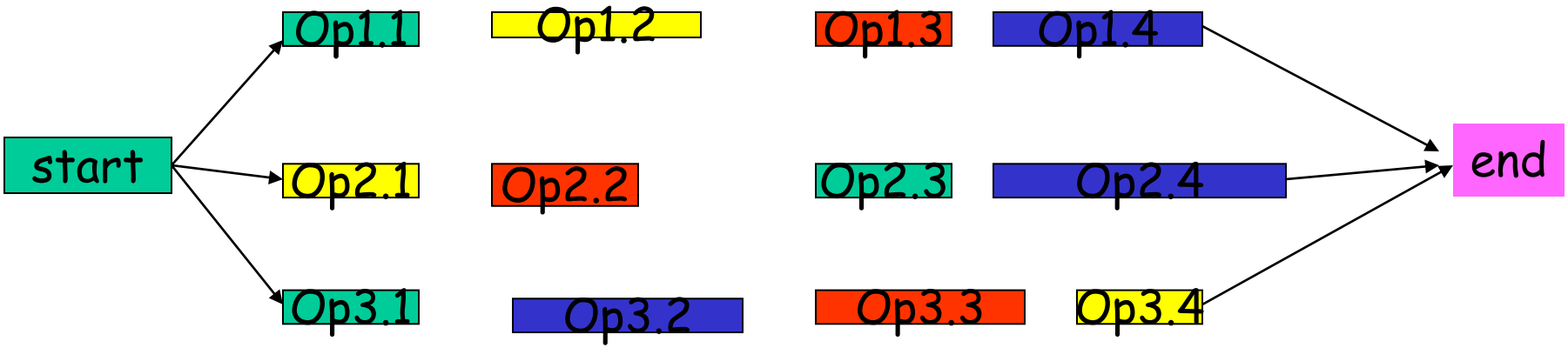
- Minimise makespan
    - what is makespan?
- Maximise start
    - JIT, minimise inventory levels
- minimise idle time on resources
    - maximise ROI
- ...



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***



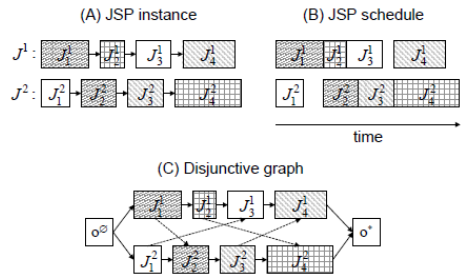
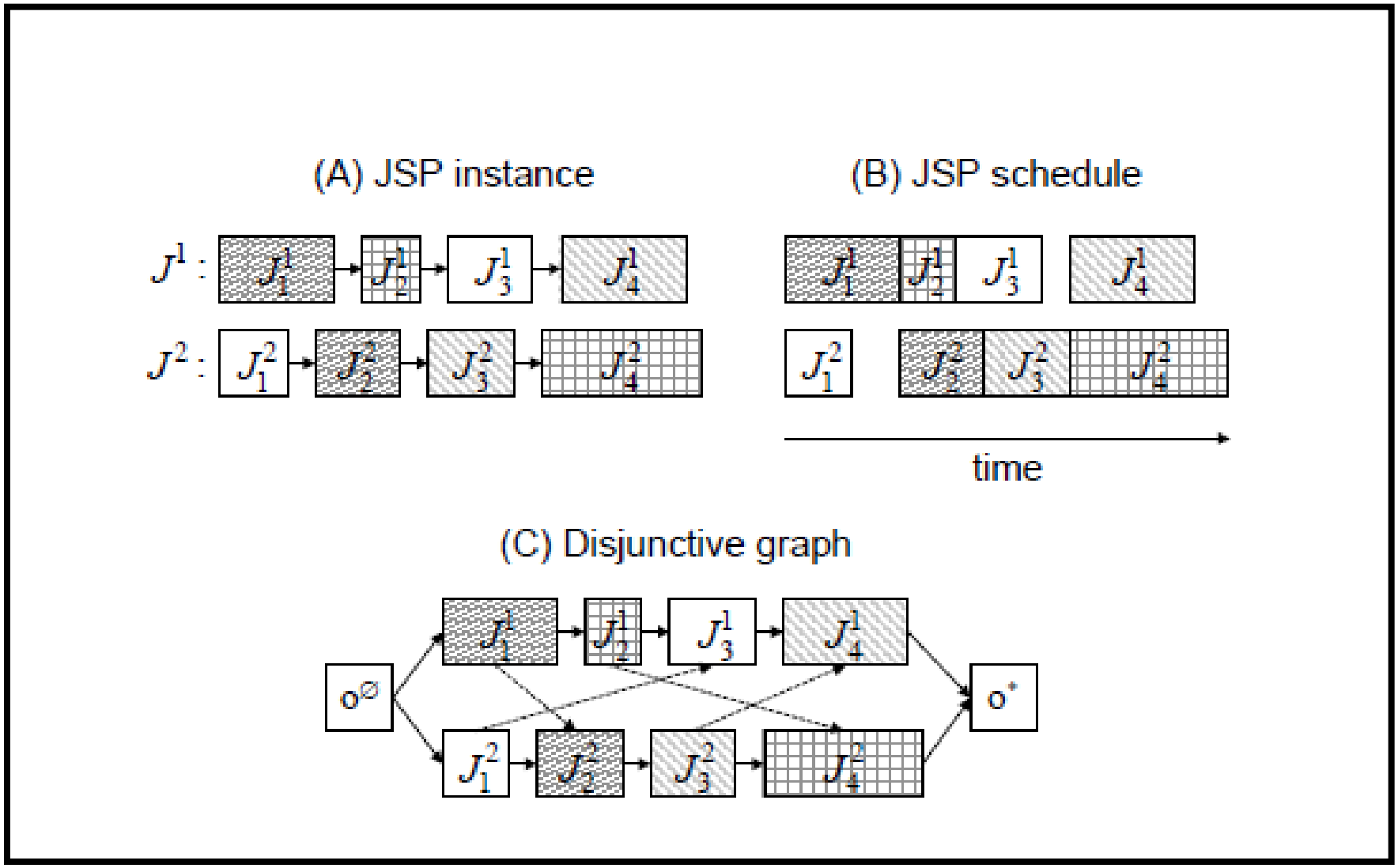


Figure 2: (A) A JSP instance, (B) a feasible schedule for the instance, and (C) the disjunctive graph representation of the schedule. Boxes represent operations; operation durations are proportional to the width of a box; and the machine on which an operation is performed is represented by texture. In (C), solid arrows represent conjunctive arcs and dashed arrows represent disjunctive arcs (arc weights are proportional to the duration of the operation the arc points out of).

(Cheeseman, Kanefsky, & Taylor, 1991; Yokoo, 1997). This phenomenon has been referred to as an "easy-hard-easy" pattern of instance difficulty (Mammen & Hogg, 1997). In §7.4 we discuss evidence of an easy-hard-easy pattern of instance difficulty in the JSP, though (to our knowledge) it is not associated with any phase transition.

The results in §§4-5 and the empirical results in §6 were previously presented in a conference paper (Streeter & Smith, 2005a).

3. The Job Shop Scheduling Problem


(A) JSP instance
(B) JSP schedule
time
(C) Disjunctive graph

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.

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

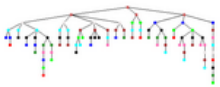
CP4: choco | M. J. Streeter and S. F. Smith (2006)

www.dcs.gla.ac.uk/~pat/cpM/choco.html

Search

Computing Science
University of Glasgow

Constraint Programming M



- Home Page
- News
- Schedule
- Contrib
- Papers
- Choco
- Links
- Exercises
- MiniZinc

- choco4
  - choco web site
  - our files
  - choco user guide
  - choco4 jar file
  - choco4 api
  - some of my files
- First Simple Examples
  - Start 2018
- The Crystal Maze
  - We present a simple encoding of the crystal maze problem
- Small TSP
  - We present a small TSP with the element constraint
- Two Models for Team Building
  - We present two choco models for Ex01
- The N-queens
  - We present the n-queens problem (CP "Hello World!"),
- Jobshop scheduling.
  - An introduction to jobshop scheduling
  - Code etc to minimise makespan jssp
- Number Partitioning
  - Code, data sets, and slides
- Bin Packing
  - jchoco code and data sets binPack
  - Slides presenting 1D bin packing as a case study
- Allocating Employees to Cost Centres
  - jchoco code and data sets teams with budgets
- Graph Colouring
  - Three models for graph colouring

Operation

Operation

Operation - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.Constraint;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Model model;

    Operation(String id,String resId,int duration,int earliestStart,int latestStart,Model model){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        this.model    = model;
        start         = model.intVar(id,earliestStart,latestStart);
    }

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

    Constraint before(Operation op2){
        return model.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.Model;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.Constraint;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Model model;

    Operation(String id,String resId,int duration,int earl
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        this.model    = model;
        start         = model.intVar(id,earliestStart,latestStart);
    }

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

    Constraint before(Operation op2){
        return model.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.Model;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.Constraint;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Model model;

    Operation(String id,String resId,int duration,int earliestStart,int latestStart,Model model){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        this.model    = model;
        start         = model.intVar(id,earliestStart,latestStart);
    }

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

    Constraint before(Operation op2){
        return model.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.Model;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.Constraint;

public class Operation {
    String id;
    String resId;
    int duration;
    IntVar start;
    Model model;

    Operation(String id,String resId,int duration,int earliestStart,int latestStart,Model model){
        this.id       = id;
        this.resId    = resId;  // resource id
        this.duration = duration;
        this.model    = model;
        start         = model.intVar(id,earliestStart,latestStart);
    }

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

    Constraint before(Operation op2){
        return model.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
    //
}
```

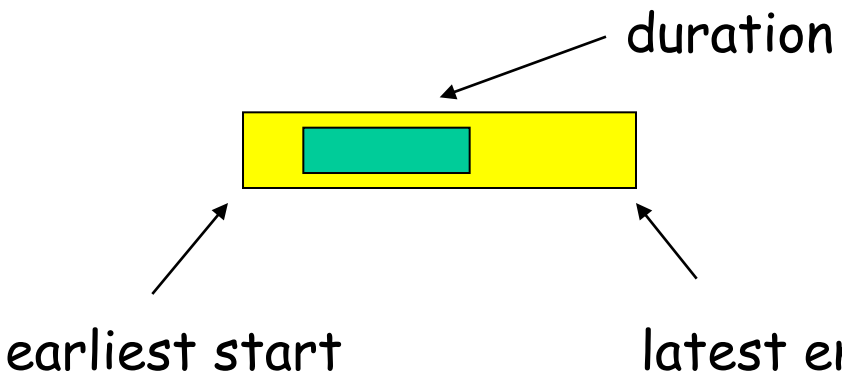
Operation

```
Constraint before(Operation op2){
    return model.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

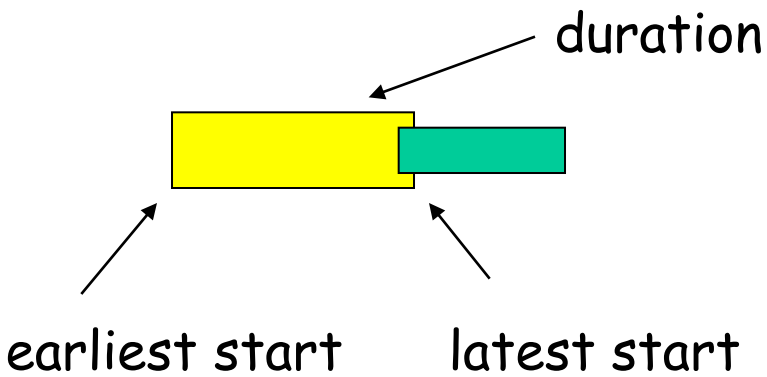
Picture of an operation

op1.before(op2)



Picture of an operation

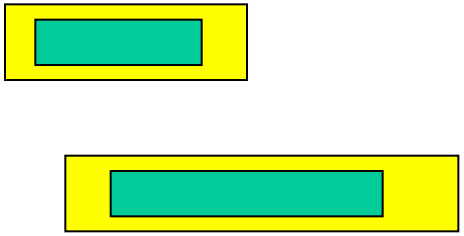
op1.before(op2)



Constrained integer variable represents start time

Picture of an operation

op1.before(op2)



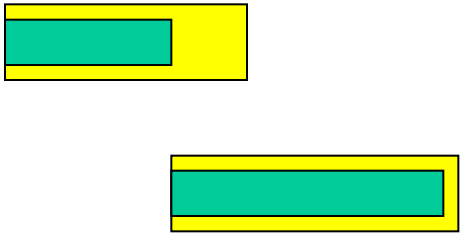
op1

op2

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

Picture of an operation

op1.before(op2)



op1

op2

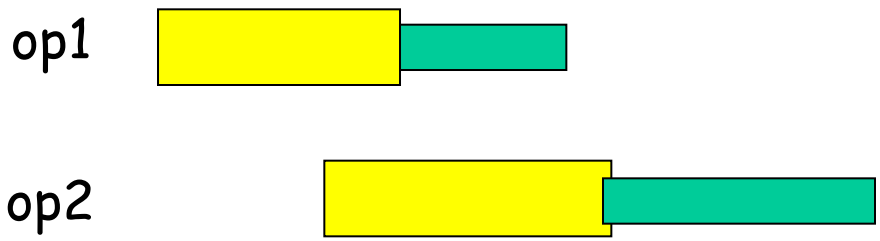
propagate

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

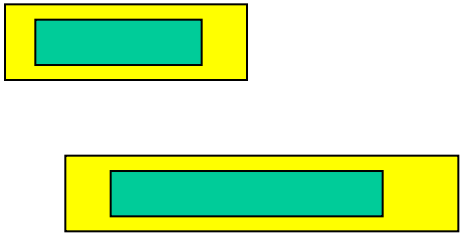
op1.before(op2) $\longrightarrow$ op1.start() + op1.duration() $\leq$ op2.start()

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.

Operation Test

OperationTest - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.IIntConstraintFactory.*;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.search.strategy.strategy.IntStrategy;
import org.chocosolver.solver.search.strategy.selectors.values.IntDomainMax;
import org.chocosolver.solver.exception.ContradictionException;

public class OperationTest {

    private static int slack(IntVar op_i,int d_i,IntVar op_j,int d_j){
        return op_j.getUB() - Math.max(op_i.getLB() + d_i,op_j.getLB());
    }
    //
    // slack if op_i before op_j
    // i.e. slack(op_i -> op_j) in S&C AAAI-93 parlance
    //
    // NOTE: we consider earliest and latest start times whereas S&C
    //       consider earliest start and latest finish, but our calculations
    //       are exactly the same
    //

    public static void main(String args[]) throws ContradictionException {

        Model model   = new Model("Operation Test");
        Solver solver = model.getSolver();
        Operation op1 = new Operation("op1","test",2,1,10,model);
        Operation op2 = new Operation("op2","test",3,1,10,model);

        IntVar decision = model.intVar("decision",0,1);

        model.ifThen(model.arithm(decision,"=",0),op1.before(op2)); // decision = 0 -> op_i before op
        model.ifThen(model.arithm(decision,"=",1),op2.before(op1)); // decision = 1 -> op before op_i

        //
        // initial state
        //
        System.out.println();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println();

        //
        // propagate
        //
        System.out.println();
        solver.propagate();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println("nothing happens");
        //
        // make decision op1 -> op2
        //
        System.out.println();
        solver.getEnvironment().worldPush();
        decision.instantiateTo(0,null);
        solver.propagate();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println("op1 before op2");
        solver.getEnvironment().worldPop();

        //
        // make decision op2 -> op1
        //
        System.out.println();
        solver.getEnvironment().worldPush();
        decision.instantiateTo(1,null);
        solver.propagate();
        System.out.println(decision);
```

Operation Test

OperationTest - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.IIntConstraintFactory.*;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.search.strategy.strategy.IntStrategy;
import org.chocosolver.solver.search.strategy.selectors.values.IntDomainMax;
import org.chocosolver.solver.exception.ContradictionException;

public class OperationTest {

    private static int slack(IntVar op_i,int d_i,IntVar op_j,int d_j){
        return op_j.getUB() - Math.max(op_i.getLB() + d_i,op_j.getLB());
    }
    //
    // slack if op_i before op_j
    // i.e. slack(op_i -> op_j) in S&C
    //
    // NOTE: we consider earliest and l
    //       consider earliest start ar
    //       are exactly the same
    //

    public static void main(String args

        Model model   = new Model("Oper
        Solver solver = model.getSolver
        Operation op1 = new Operation("
        Operation op2 = new Operation("

        IntVar decision = model.intVar(

        model.ifThen(model.arithm(decis
        model.ifThen(model.arithm(decis

        //
        // initial state
        //
        System.out.println();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println();

        //
        // propagate
        //
        System.out.println();
        solver.propagate();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println("nothing hap
        //
        // make decision op1 -> op2
        //
        System.out.println();
        solver.getEnvironment().worldPu
        decision.instantiateTo(0,null);
        solver.propagate();
        System.out.println(decision);
        System.out.println(op1);
        System.out.println(op2);
        System.out.println("op1 before
        solver.getEnvironment().worldPo

        //
        // make decision op2 -> op1
        //
        System.out.println();
        solver.getEnvironment().worldPush();
        decision.instantiateTo(1,null);
        solver.propagate();
        System.out.println(decision);
```

Command Prompt

```
Z:\public_html\cpM\choco4\jsspResearch\code>java OperationTest

decision = [0,1]
{op1 test 2 [1,10]}
{op2 test 3 [1,10]}


decision = [0,1]
{op1 test 2 [1,10]}
{op2 test 3 [1,10]}
nothing happens

decision = 0
{op1 test 2 [1,8]}
{op2 test 3 [3,10]}
op1 before op2

decision = 1
{op1 test 2 [4,10]}
{op2 test 3 [1,7]}
op2 before op1

slack(op1 -> op2) : 7
slack(op2 -> op1) : 6
slack(op1,op2) : 7

Z:\public_html\cpM\choco4\jsspResearch\code>_
```

Job

Job

Job - Notepad

File Edit Format View Help

```java
import java.util.ArrayList;
import org.chocosolver.solver.Model;

public class Job {
    String id;
    ArrayList<Operation> operations;
    int length;
    Model model;

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

    void add(Operation op){
        if (!operations.isEmpty()) operations.get(length-1).before(op).post();
        operations.add(op);
        length++;
    }
    //
    // adding operations in sequence to a job, such that
    // operation op_i is before operation op_i+1
    // i.e. precedence constraints between operations in a job.
    //

    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;
    Model model;

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

Job is a sequence of operations

Job

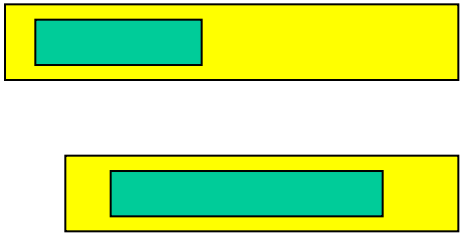
```
void add(Operation op){
    if (!operations.isEmpty()) operations.get(length-1).before(op).post();
    operations.add(op);
    length++;
}
//
// adding operations in sequence to a job, such that
// operation op_i is before operation op_i+1
// i.e. precedence constraints between operations in a job.
//
```

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

Decision

Picture of an operation

op1.before(op2)



op1

op2

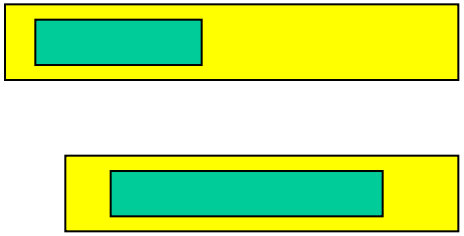
Use a 0/1 decision variable d[i][j] as follows

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

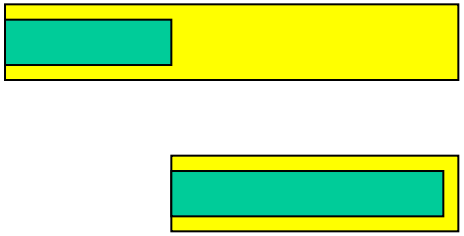
op2

d[i][j] = 0 → op[i]1.before(op[j])

op1 before op2

Picture of an operation

op1.before(op2)



op1

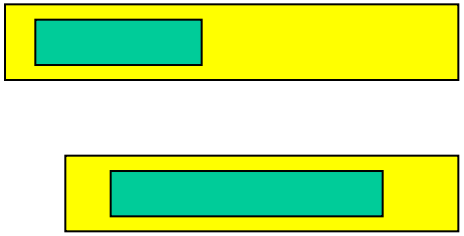
op2

d[i][j] = 0 → op[i]1.before(op[j])

op1 before op2

Picture of an operation

op1.before(op2)



op1

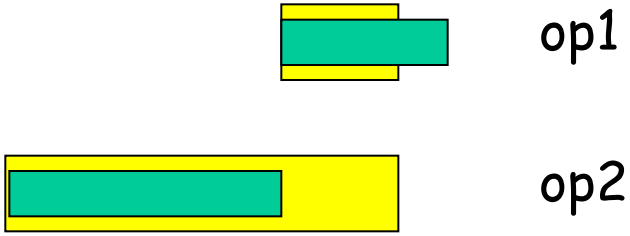
op2

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.impl.BoolVarImpl;
import org.chocosolver.solver.Model;

public class Decision extends BoolVarImpl{
    Operation op_i;
    Operation op_j;

    Decision(Model model,Operation op_i,Operation op_j){
        super("dec-"+ op_i.id +"-"+ op_j.id,model);
        this.op_i = op_i;
        this.op_j = op_j;
    }

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

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

Decision

Decision - Notepad

File Edit Format View Help

```
import org.chocosolver.solver.variables.impl.BoolVarImpl;
import org.chocosolver.solver.Model;

public class Decision extends BoolVarImpl{
    Operation op_i;
    Operation op_j;

    Decision(Model model,Operation op_i,Operation op_j){
        super("dec-"+ op_i.id +"-"+ op_j.id,model);
        this.op_i = op_i;
        this.op_j = op_j;
    }

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

    Operation getOp_i(){return op_i;}
    Operation getOp_j(){return op_j;}
}
```

A decision is essentially a triple:

- a zero/one variable (this)
- an operation op_i
- an operation op_j

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

Resource

Resource

Resource - Notepad

File Edit Format View Help

```
import java.util.*;
import org.chocosolver.solver.Model;

public class Resource {
    String id;
    ArrayList<Operation> operations;
    ArrayList<Decision> decisions;
    Model model;

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

    void add(Operation op){
        int n = operations.size();
        for (int i=0;i<n;i++){
            Operation op_i = operations.get(i);
            Decision decision = new Decision(model,op_i,op);
            decisions.add(decision);
            model.ifThen(model.arithm(decision,"=",0),op_i.before(op)); // decision = 0 -> op_i before op
            model.ifThen(model.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;
    Model model;

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

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);
        Decision decision = new Decision(model,op_i,op);
        decisions.add(decision);
        model.ifThen(model.arithm(decision,"=",0),op_i.before(op)); // decision = 0 -> op_i before op
        model.ifThen(model.arithm(decision,"=",1),op.before(op_i)); // decision = 1 -> op before op_i
    }
    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);
        Decision decision = new Decision(model,op_i,op);
        decisions.add(decision);
        model.ifThen(model.arithm(decision,"=",0),op_i.before(op)); // decision = 0 -> op_i before op
        model.ifThen(model.arithm(decision,"=",1),op.before(op_i)); // decision = 1 -> op before op_i
    }
    operations.add(op);
}
```

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

JSSP

JSSP

JSSP - Notepad

File Edit Format View Help

```java
import java.io.*;
import java.util.*;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.IIntConstraintFactory.*;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.exception.ContradictionException;

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!
    Model model;

    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;
        model             = new Model("id");
        endOp             = new Operation("endOp","nullRes",0,0,dueDate,model);
        int totalDuration = 0;
        for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
        for (int i=0;i<n;i++){
            Job job = new Job("job_"+i,model);
            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,0,dueDate,model);
                resource.add(operation);
                job.add(operation);
            }
            job.add(endOp);
            jobs.add(job);
        }
        model.arithm(endOp.start,"<=",totalDuration).post();
        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;
    }
    //
    // to be used with min-slack dvo
    //

}
```

JSSP - Notepad

File Edit Format View Help

```java
import java.io.*;
import java.util.*;
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.solver.constraints.IIntConstraintFactory.*;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.exception.ContradictionException;

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!
    Model model;

    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;
        model             = new Model("id");
        endOp             = new Operation("endOp","nullRes",0,0,dueDate,model);
        int totalDuration = 0;
        for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
        for (int i=0;i<n;i++){
            Job job = new Job("job_"+i,model);
            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,0,dueDate,model);
                resource.add(operation);
                job.add(operation);
            }
            job.add(endOp);
            jobs.add(job);
        }
        model.arithm(endOp.start,"<=",totalDuration).post();
        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;
    }
    //
    // to be used with min-slack dvo
    //

}
```

JSSP

Ouch!

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!
    Model model;
```

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;
    model              = new Model("id");
    endOp              = new Operation("endOp","nullRes",0,0,dueDate,model);
    int totalDuration = 0;
    for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
    for (int i=0;i<n;i++){
        Job job = new Job("job_"+i,model);
        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,0,dueDate,model);
            resource.add(operation);
            job.add(operation);
        }
        job.add(endOp);
        jobs.add(job);
    }
    model.arithm(endOp.start,"<=",totalDuration).post();
    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;
    model               = new Model("id");
    endOp               = new Operation("endOp","nullRes",0,0,dueDate,model);
```

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;
    model               = new Model("id");
    endOp               = new Operation("endOp","nullRes",0,0,dueDate,model);
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,model);
    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,0,dueDate,model);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
model.arithm(endOp.start,"<=",totalDuration).post();
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,model);
    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,0,dueDate,model);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
model.arithm(endOp.start,"<=",totalDuration).post();
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
for (int i=0;i<n;i++){
    job job = new Job("job_"+i,model);
    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,0,dueDate,model);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
model.arithm(endOp.start,"<=",totalDuration).post();
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,model);
    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,0,dueDate,model);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
model.arithm(endOp.start,"<=",totalDuration).post();
sc.close();
```

JSSP

```
int totalDuration = 0;
for (int i=0;i<m;i++) resources.add(new Resource("r_"+i,model));
for (int i=0;i<n;i++){
    Job job = new Job("job_"+i,model);
    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,0,dueDate,model);
        resource.add(operation);
        job.add(operation);
    }
    job.add(endOp);
    jobs.add(job);
}
model.arithm(endOp.start,"<=",totalDuration).post();
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.model.getSolver();
      Decision[] decisions   = jssp.getDecisions();
      int n                  = decisions.length;
      IntVar makeSpan        = jssp.getMakeSpan();

      //solver.setSearch(Search.inputOrderUBSearch(decisions));
      solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                       new MinSlackHeuristic(),
                                       new MaxSlackValue()));

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

      System.out.println(makeSpan + " ["+ makeSpan.getLB() +","+ makeSpan.getUB() +"]");
      System.out.println("nodes: " + solver.getMeasures().getNodeCount() +
                         "  cpu: " + solver.getMeasures().getTimeCount());
  }
}
//
// is there a legal schedule with a make span of dueDate, or less?
//
```

Optimize

```
public class Optimize {

    public static void main(String[] args) throws ContradictionException, FileNotFoundException, IOException {
        JSSP jssp              = new JSSP(args[0],9999);
        String valueHeuristic = args[1];
        int timeLimit          = Integer.parseInt(args[2]);
        Model model            = jssp.model;
        Solver solver          = model.getSolver();
        Decision[] decisions   = jssp.getDecisions();
        IntVar makeSpan        = jssp.getMakeSpan();
        int lwb                = 0;
        int upb                = 9999;

        solver.limitTime(timeLimit*1000);

        if (valueHeuristic.equals("maxSlack"))
            solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                             new MinSlackHeuristic(),
                                             new MaxSlackValue()));
        else if (valueHeuristic.equals("fuzzyMaxSlack"))
            solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                             new MinSlackHeuristic(),
                                             new FuzzyMaxSlackValue(0.5)));
        else if (valueHeuristic.equals("minSlack"))
            solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                             new MinSlackHeuristic(),
                                             new MinSlackValue()));
        else if (valueHeuristic.equals("fuzzyMinSlack"))
            solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                             new MinSlackHeuristic(),
                                             new FuzzyMinSlackValue(0.5)));
        else if (valueHeuristic.equals("random"))
            solver.setSearch(new IntStrategy(jssp.getDecisions(),
                                             new MinSlackHeuristic(),
                                             new FuzzyMaxSlackValue(-1.0)));
        else solver.setSearch(Search.inputOrderLBSearch(jssp.getDecisions()));
        //
        // attach a variable & value ordering heuristic to solver
        //

        model.setObjective(Model.MINIMIZE,makeSpan);
        while (solver.solve()){
            lwb = makeSpan.getLB();
            upb = makeSpan.getUB();
        }
        System.out.println("makespan: ["+ lwb +","+ upb +"]");
        System.out.println("nodes: "+ solver.getMeasures().getNodeCount() +
                           "   cpu: "+ solver.getMeasures().getTimeCount());
    }
}
```

Wot!? No heuristics!?!!!