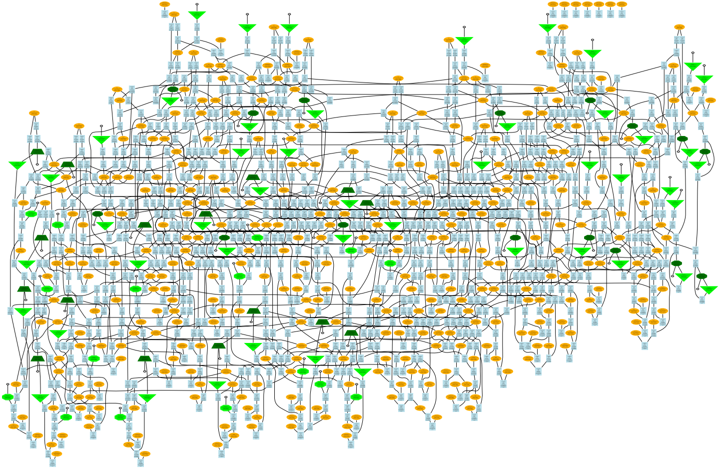
# GGen: Random Graph Generation for Scheduling Simulations

**Swann Perarnau**, Denis Trystram, Jean-Marc Vincent

MOAIS/MESCAL Inria Teams
IUF,Grenoble University

ROADEF 2011

# MOAIS/MESCAL Research Domain: HPC

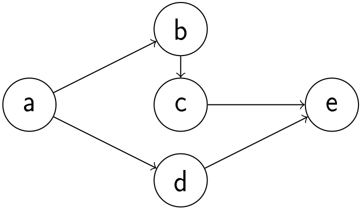


Molecular Dynamics Simulation: Data/Communications Graph.

# MOAIS/MESCAL Research Domain: HPC

Part of the Grid5000 Experimental TestBed.

# Motivation: Simulation of Scheduling Algorithms




## Input Characteristics: Directed Acyclic Graph

- Vertices are tasks to execute.
- Edges are precedence constraints or communications.
- Additional annotations for costs.

# Workload Characterization

### Uniform Generation of Random Graphs

Combinatorial Approach.

### Specific Classes of Random Graphs

Graphs respecting a set of well known properties.

### Traces / Collected Workloads

Identified instances from real/academic environments.

# Workload Characterization

### Uniform Generation of Random Graphs

Impractical.

### Specific Classes of Random Graphs

Graphs respecting a set of well known properties.

### Traces / Collected Workloads

Identified instances from real/academic environments.

# Workload Characterization

**Uniform Generation of Random Graphs**

Impractical.

**Specific Classes of Random Graphs**

Graphs respecting a set of well known properties.

**Traces / Collected Workloads**

Hard to generalize results.

# Workload Characterization

**Uniform Generation of Random Graphs**

Impractical.

**Specific Classes of Random Graphs**

Our focus.

**Traces / Collected Workloads**

Hard to generalize results.

# GGen Objectives

### Challenges

- Implementations are rarely provided.
- Analysis of each classical method.

### State of the project

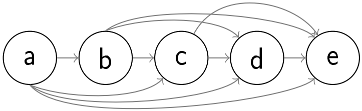
- A framework to generate and analyze DAGs.
- In-depth analysis of generation methods and their influence on schedulers.

# Outline

# Outline

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | - | - | - | - |
| b | X | X | - | - | - |
| c | X | X | X | - | - |
| d | X | X | X | X | - |
| e | X | X | X | X | X |

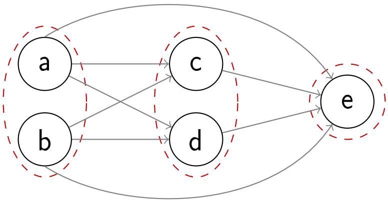



## Parameters

$n$ : Number of nodes.
$l$ : Number of layers.
$p$ : Probability to choose any possible edge.

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | - | - | - | - |
| b | X | X | - | - | - |
| c | X | X | X | - | - |
| d | X | X | X | X | - |
| e | X | X | X | X | X |




## Parameters

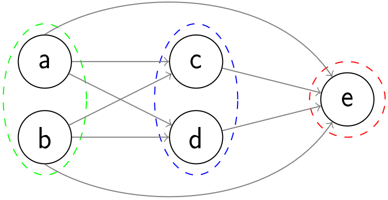
$n$ : Number of nodes.

$l$ : Number of layers.

$p$ : Probability to choose any possible edge.

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | X | - | - | - |
| b | X | X | - | - | - |
| c | X | X | X | X | - |
| d | X | X | X | X | - |
| e | X | X | X | X | X |




## Parameters

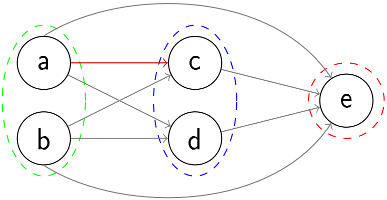
$n$ : Number of nodes.

$l$ : Number of layers.

$p$ : Probability to choose any possible edge.

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | X | - | - | - |
| b | X | X | - | - | - |
| c | X | X | X | X | - |
| d | X | X | X | X | - |
| e | X | X | X | X | X |




## Parameters

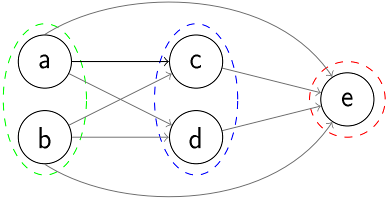
$n$ : Number of nodes.

$l$ : Number of layers.

$p$ : Probability to choose any possible edge.

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | X | 1 | - | - |
| b | X | X | - | - | - |
| c | X | X | X | X | - |
| d | X | X | X | X | - |
| e | X | X | X | X | X |




## Parameters

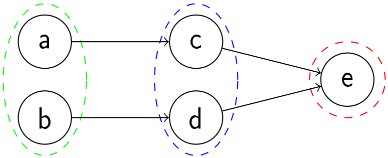
*n* : Number of nodes.

*l* : Number of layers.

*p* : Probability to choose any possible edge.

# Layer-by-Layer [Kasahara *et al.*, 2002]

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | X | X | 1 | 0 | 0 |
| b | X | X | 0 | 1 | 0 |
| c | X | X | X | X | 1 |
| d | X | X | X | X | 1 |
| e | X | X | X | X | X |



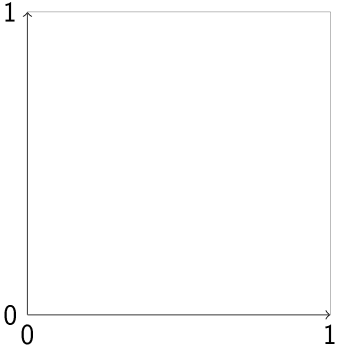

## Parameters

$n$ : Number of nodes.

$l$ : Number of layers.

$p$ : Probability to choose any possible edge.

# Random Orders [Winkler, 1985]

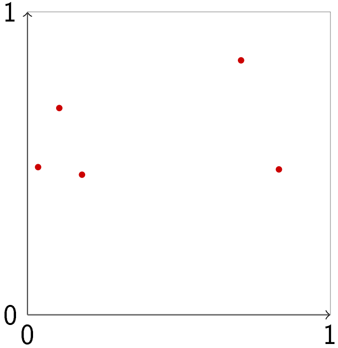



## Parameters

$n$ : Number of nodes.

$k$ : Number of total orders to intersect.

# Random Orders [Winkler, 1985]

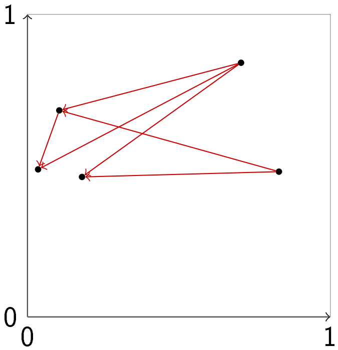



## Parameters

$n$ : Number of nodes.

$k$ : Number of total orders to intersect.

# Random Orders [Winkler, 1985]

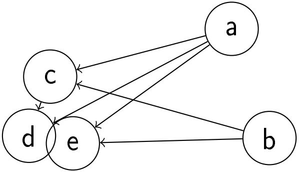



## Parameters

$n$ : Number of nodes.

$k$ : Number of total orders to intersect.

# Random Orders [Winkler, 1985]

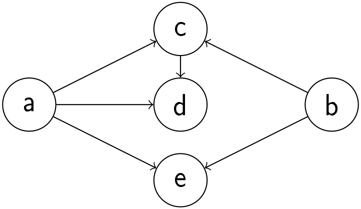



**Parameters**

$n$ : Number of nodes.

$k$ : Number of total orders to intersect.

# Random Orders [Winkler, 1985]




## Parameters

$n$ : Number of nodes.

$k$ : Number of total orders to intersect.

# Summary

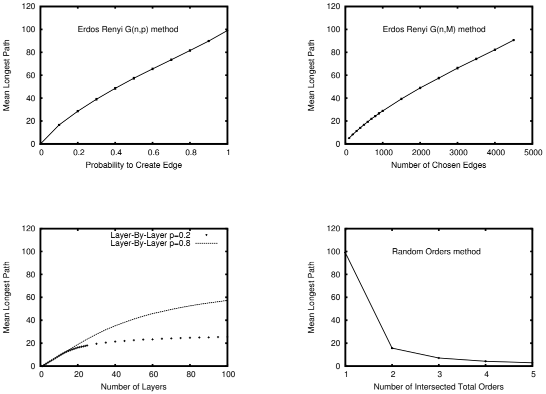
## Needs

- Many generation methods and variants.
- Analyze each method characteristics.

## GGen first steps

- Provide a reference implementation
- Use Grid'5000 for big analysis campaigns

# Critical Path Analysis




| Experimental Design | |
|---|---|
| Sample Size: | 1,000 |
| Number of nodes: | 100 |
| Confidence Intervals: | 95% |

# Outline

# Why generation parameters matter

### Scheduler Sensibility

Small variations in the input lead to big differences in performance.

### Experimental Design

- Select various schedulers,
- Measure their performance on a reference data set,
- Change the data set in subtle ways,
- Measure the variation in performance.

# Input Characteristics

| Data Set | Vertices | Edges | | Comput. Cost | | Comm. Cost | | CCR |
|---|---|---|---|---|---|---|---|---|
| | | mean | sd | mean | sd | mean | sd | |
| $T_{small}$ | 500 | 746 | 27 | 9,98 | 5.7 | 0.5 | 0.2 | $\approx 20$ |
| $T_{moy}$ | | | | | | 5.1 | 2.5 | $\approx 2$ |
| $T_{big}$ | | | | | | 10.2 | 5.1 | $\approx 1$ |

## Simulation Parameters

- Varying number of processors.
- Network is a complete graph.
- 1000 simulations per data point.

# Base performance of schedulers

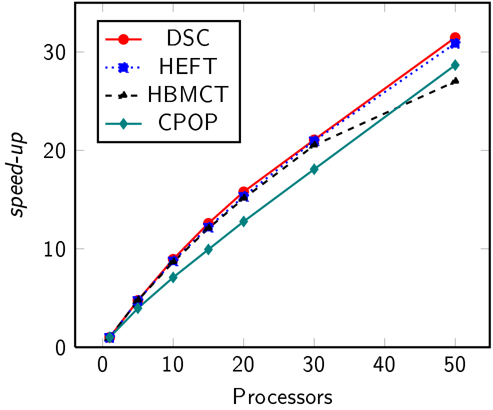



Figure: *speed-up* on a varying number of processor (on $T_{small}$).

# Performance variation

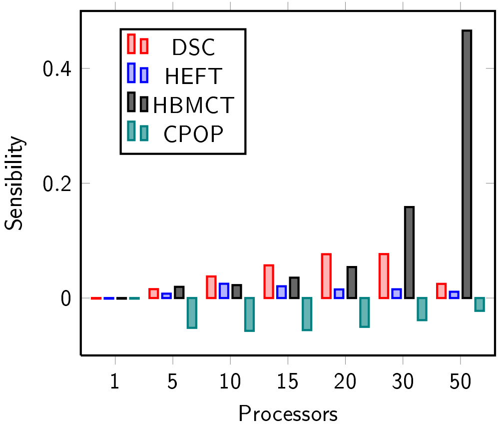



Figure: Sensibility to computational costs, on $T_{small}$ modified with an exponential distribution.

# Outline

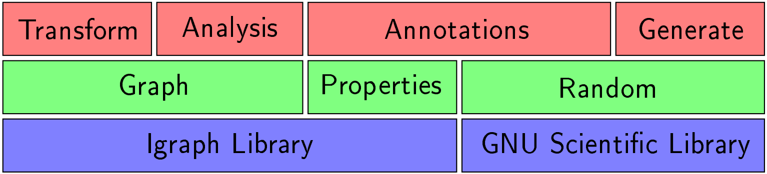
# Classical Performance Evaluation Process

1. Choose a scheduling algorithm.
2. Characterize the input data.
3. Choose a generation method.
4. Generate workload.
5. Check the quality of the input.
6. Simulate the scheduling algorithm.
7. Analyse results.

# Classical Performance Evaluation Process

1. Choose a scheduling algorithm.
2. Characterize the input data.
3. Choose a generation method.
4. Generate workload.
5. Check the quality of the input.
6. Simulate the scheduling algorithm.
7. Analyse results.

# GGen: Software Architecture



# GGen: Technical Info

### Random Graph Generator

Contains most classical methods.
Easily extensible code.
Standard output format: Graphviz DOT.

### Technical Info

C Code under GPL compatible license.
Both a library and binaries utilities.
Publicly available at `http://ggen.ligforge.imag.fr/`

### Demo

Available on demand during the conference.

# Outline

# Ongoing and Future Works

## On Graph Generation

- More statistical studies.
- More graph classes.
- Integration of new methods as they appear.

## On Simulations

- More distributions tested.
- Influence of generation parameters.
- Automated analysis.

# Thanks

Thank you for your attention.

Demo available on demand.

GGen is publicly available at `http://ggen.ligforge.imag.fr/`