BRICKS*

Modélisation probabiliste relationnelle

P. Munteanu - M. Rèche
BAYESIA

* Bayesian Representation and Inference for Complex Knowledge Structuring

Les réseaux bayésiens

Formalisme probabiliste rigoureux pour modéliser la connaissance comportant des incertitudes

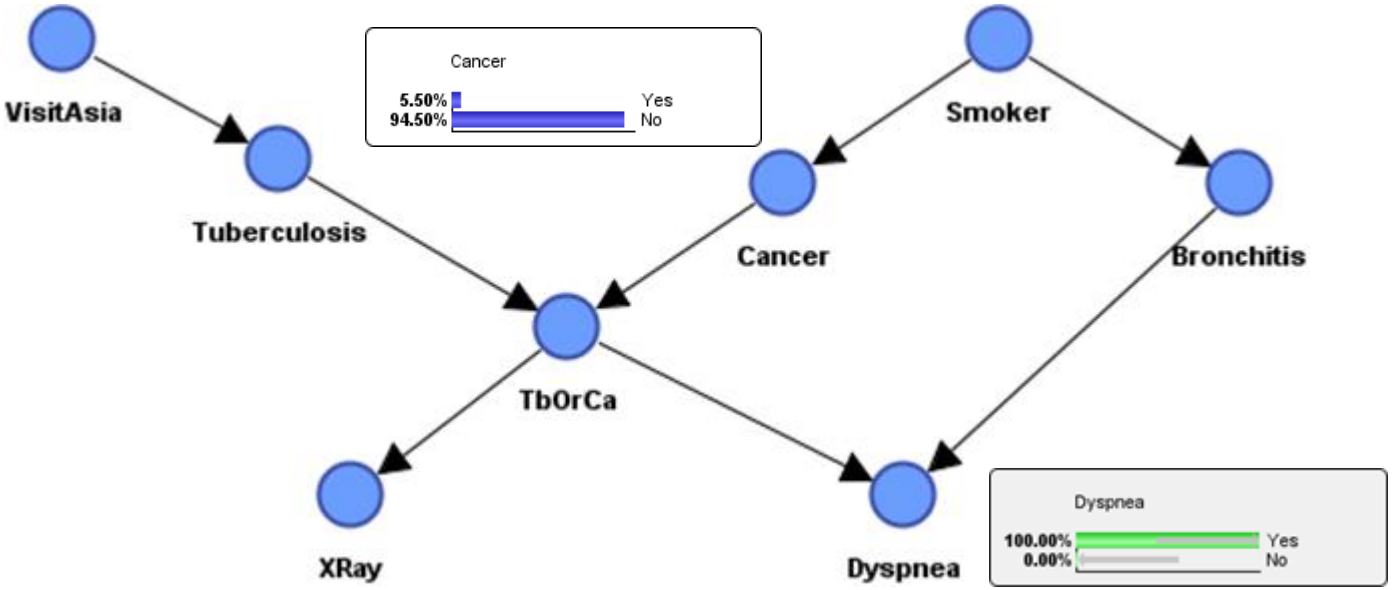
Représentation graphique intuitive

Algorithmes efficaces d'apprentissage et d'inférence

Repères chronologiques

- 1763 : Publication posthume du Théorème de Bayes P(A|B) = P(B|A)P(A)/P(B)
- 1988 : Publication des travaux séminaux de Judea Pearl, « Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference »
- 1996 : Bill Gates déclare au Los Angeles Times que « l'avantage concurrentiel de Microsoft repose sur son expertise en réseaux bayésiens »
- 2004 : L'apprentissage des réseaux bayésiens figure en quatrième position des « dix technologies émergeantes qui vont changer le monde » dans MIT Technology Review
- 2012 : Le prix A.M. Turing (le « Nobel » de l'informatique) est décerné à Judea Pearl

Les réseaux bayésiens



Tuberculosis	Cancer	Value
Yes	Yes	Yes
	No	Yes
No	Yes	Yes
	No	No

TbOrCa	Bronchitis	Yes	No
Yes	Oui	90.000	10.000
	Non	70.000	30.000
No	Oui	80.000	20.000
	Non	10.000	90.000

Exemple introductif

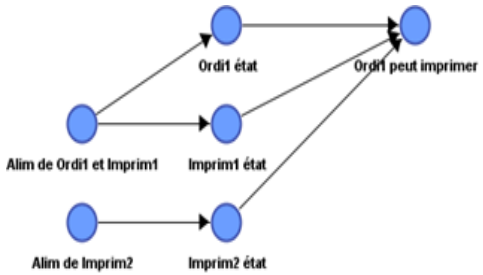
Le problème réel à l'origine de l'exemple

- Modéliser le système d'information d'une grande entreprise (depuis l'infrastructure matérielle jusqu'au processus métier) pour réaliser des études de disponibilité, d'impact, de diagnostic, etc.
- Problèmes similaires : réseaux de transport ou distribution, flottes de véhicules, etc.

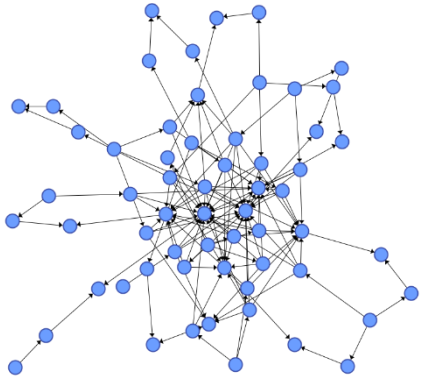
Le problème simplifié

- Estimer la disponibilité du service d'impression dans un immeuble comportant des ordinateurs et imprimantes interconnectés, répartis dans plusieurs salles desservies par des alimentations différentes

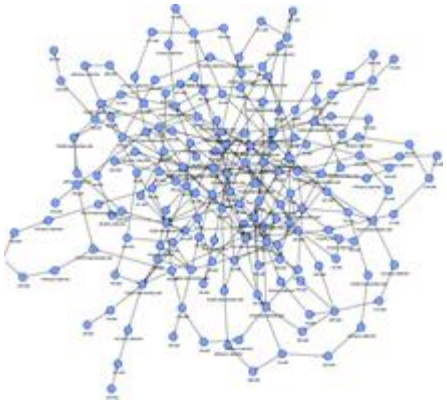
L'approche « classique » par réseaux bayésiens



Un ordinateur
Deux imprimantes



Une trentaine d'équipements



Une centaine d'équipements

Aux origines : le projet ANR SKOOB

Problématique

- modélisation probabiliste des systèmes complexes
- pour les applications de maîtrise des risques et sûreté de fonctionnement

Besoins

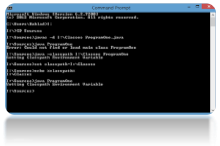
- Ingénierie des modèles
  - gestion de la complexité
  - réutilisation des composants
  - travail collaboratif
- Pouvoir expressif
  - description générique de systèmes comportant des composants
  - configurations
  - horizons de temps variables
- Exploitation en présence de ressources de calcul limitées

Plate-forme technologique de modélisation et d'analyse

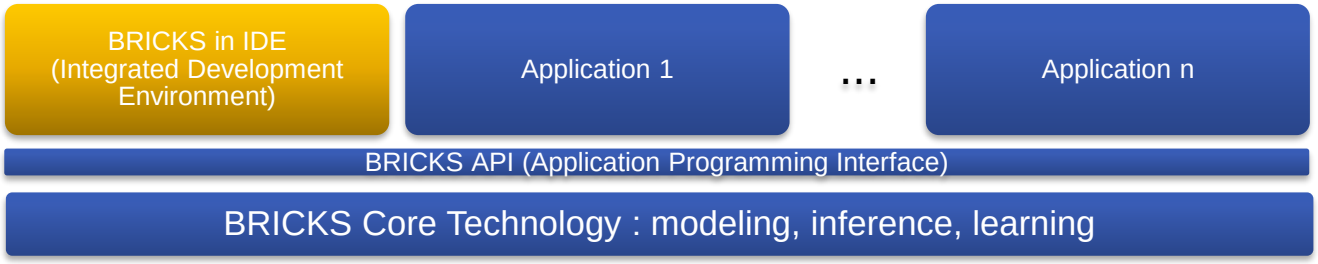
- Edition assistée de modèles probabilistes relationnels
- API d'instanciation - générateur de réseaux bayésiens « classiques » à partir de ces modèles
- API d'inférence probabiliste optimisée

Basée sur Java et donc multi-plateformes

N'impose pas d'environnement de développement



S'intègre dans la logique métier



Le langage BRICKS : base de connaissances (1/2)

Des classes Java instrumentées BRICKS

- via des objets classiques
  - Attribute<? extends AttributeType>
- et surtout des annotations
  - @TabularModel(…)
  - dont la syntaxe peut être vérifiée de façon interactive

```
2
3  import com.bayesia.bricks.prm.annotation.TabularModel;
4  import com.bayesia.bricks.prm.attribute.Attribute;
5  import com.bayesia.bricks.prm.attribute.type.BricksBoolean;
6
7  public class PowerSupply {
8
9    private Attribute<BricksBoolean> state;
10
11   @TabularModel(table = { 0.1, 0.9 })
12   public Attribute<BricksBoolean> getState() {
13     return state;
14   }
15
16   public void setState(Attribute<BricksBoolean> state) {
17     this.state = state;
18   }
19 }
20
```

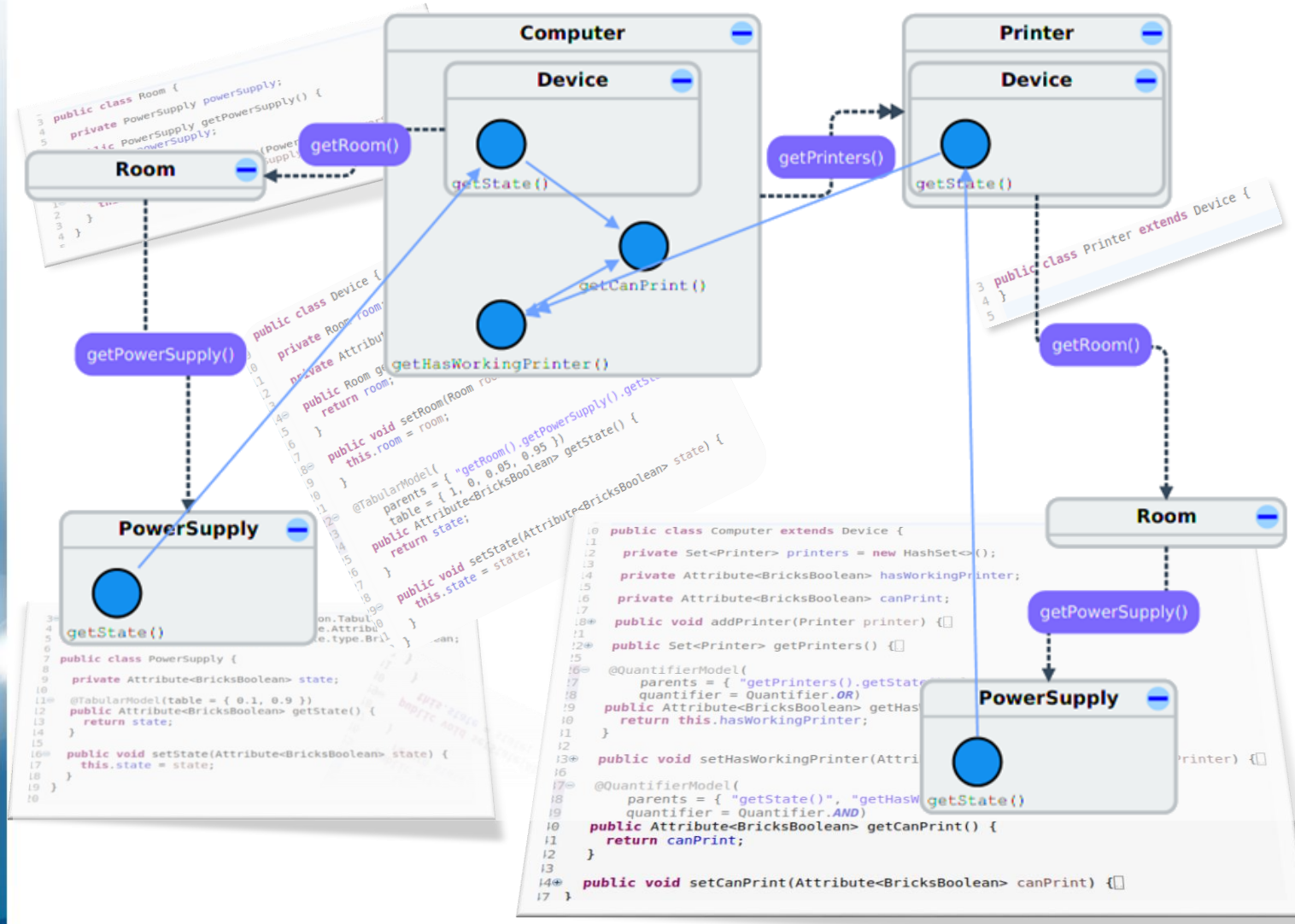
```
2
3  import com.bayesia.bricks.example.printers.classes.Room;
7
8  public class Device {
9
10   private Room room;
11
12   private Attribute<BricksBoolean> state;
13
14   public Room getRoom() {
15     return room;
16   }
17
18   public void setRoom(Room room) {
19     this.room = room;
20   }
21
22   @TabularModel(
23       parents = { "getRoom().getPowerSupply().getState()" },
24       table = { 1, 0, 0.05 })
25   publ                                                    e() {
26     re
27   }
28
29   public void setState(Attribute<BricksBoolean> state) {
30     this.state = state;
31   }
32 }
33
```

Mauvaise taille (attendue 4, trouvée 3)

Press 'F2' for focus

Le langage BRICKS : base de connaissances (2/2)

Définissant un modèle de classes

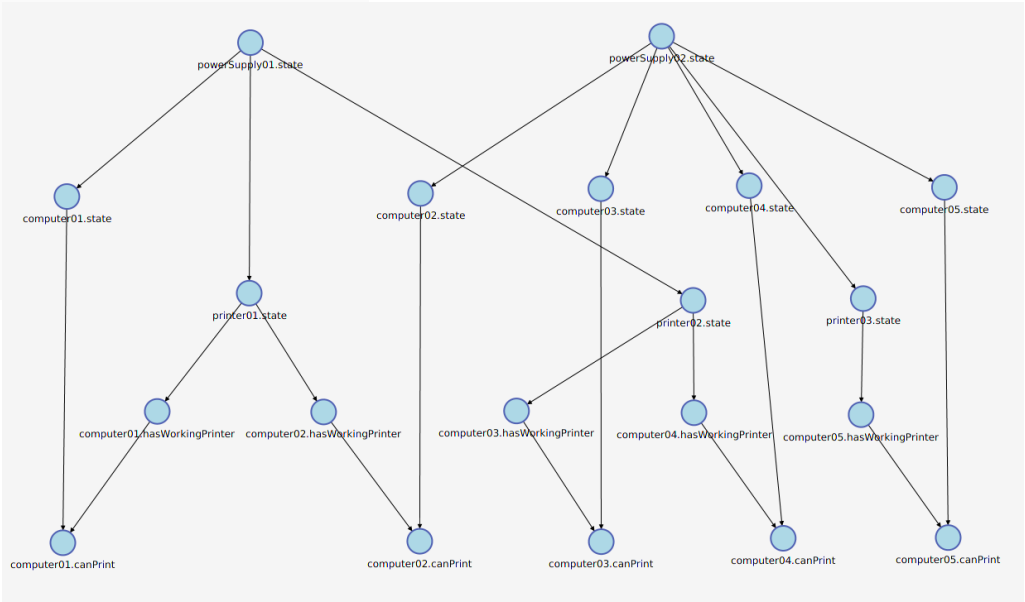


Le langage BRICKS : instanciation (1/2)

Grâce à une Application Programming Interface…

- par injection de dépendance

```
19
20    PowerSupply powerSupply1 = modeler.createInstance(PowerSupply.class, "powerSupply01");
21    PowerSupply powerSupply2 = modeler.createInstance(PowerSupply.class, "powerSupply02");
22
23
24    Room room1 = modeler.createInstance(Room.class, "room01");
25    room1.setPowerSupply(powerSupply1);
26    Room room2 = modeler.createInstance(Room.class, "room02");
27    room2.setPowerSupply(powerSupply2);
28
29    for (int i = 0; i < 5; i++) {
30      Computer computer = modeler.createInstance(Computer.class, "computer0" + (i + 1));
31      computers.add(computer);
32      computer.setRoom(i == 0 ? room1 : room2);
33    }
34
35    for (int i = 0; i < 3; i++) {
36      Printer printer = modeler.createInstance(Printer.class, "printer0" + (i + 1));
37      printer.setRoom(i <= 1 ? room1 : room2);
38
39      switch (i) {
40        case 0:
41          computers.get(0).addPrinter(printer);
42          computers.get(1).addPrinter(printer);
43          break;
44        case 1:
45          computers.get(2).addPrinter(printer);
46          computers.get(3).addPrinter(printer);
47          break;
48        case 2:
49          computers.get(4).addPrinter(printer);
50          break;
51        default:
52      }
53    }
```



Le langage BRICKS : instanciation (2/2)

…compatible Hibernate

- automatique depuis la base de données
- paresseuse pour limiter les besoins de ressources

```
20
21 @Entity
22 @Table(name = "computer")
23 public class Computer extends Device {
24
25   private int id;
26
27   @Id
28   public int getId() {
29     return id;
30   }
31
32   public void setId(int id) {
35
36   private Set<Printer> printers = new HashSet<>();
37
38   private Attribute<BricksBoolean> hasWorkingPrinter;
39
40   public void setHasWorkingPrinter(Attribute<BricksBoolean> hasWorkingPrinter) {
41     this.hasWorkingPrinter = hasWorkingPrinter;
42   }
43
44   @QuantifierModel(parents = { "getPrinters().getState()" }, quantifier = Quantifier.OR)
45   @Transient
46   public Attribute<BricksBoolean> getHasWorkingPrinter() {
47     return this.hasWorkingPrinter;
48   }
49
50   private Attribute<BricksBoolean> canPrint;
51
52   @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
53   @JoinTable(name = "computer_printer", joinColumns = { @JoinColumn(name = "computer_id") },
54       inverseJoinColumns = { @JoinColumn(name = "printer_id") })
55   public Set<Printer> getPrinters() {
56     return printers;
57   }
```

Le défi algorithmique

Problématique

- ☺ Modélisation facilitée des systèmes complexes
- ☹ Taille des RB équivalents prohibitive pour l'inférence probabiliste

Solution : maîtriser cette complexité en combinant de manière innovante

- ❖ L'exploitation de la requête : calculs circonscrits à la seule partie du modèle nécessaire pour répondre à la requête
- ❖ L'exploitation de la microstructure probabiliste pour éviter la croissance exponentielle des TPC impliquant un grand nombre de variables
- ❖ L'exploitation de la structure relationnelle pour l'identification des symétries et des motifs récurrents permettant d'éviter les calculs répétitifs
- ❖ L'utilisation de séparateurs plus compacts que ceux traditionnellement utilisés pour propager l'information probabiliste entre les différentes parties du modèle
- ❖ Les approximations, dans les situations où l'inférence exacte s'avère infaisable malgré ces optimisations

Résultat : on peut gérer des modèles probabilistes largement hors d'atteinte des techniques à base de réseaux bayésiens « classiques »

Toujours via l'API

```
BricksQuery query = new BricksQuery(printerSystem.getModeler());
query.add(printerSystem.powerSupply1.getState());
query.add(printerSystem.powerSupply2.getState());

BricksReasoner reasoner = new BricksReasoner();
Map<Attribute<?>, double[]> results = reasoner.runQuery(query);

for (Entry<Attribute<?>, double[]> entry : results.entrySet()) {
  System.out.println(entry.getKey().getName());
  System.out.println(Arrays.toString(entry.getValue()));
}

System.out.println("\n-----------\n");
query.setObservation(computers.get(0).getCanPrint(), false);
query.add(computers.get(0).getCanPrint());

results = reasoner.runQuery(query);

for (Entry<Attribute<?>, double[]> entry : results.entrySet()) {
  System.out.println(entry.getKey().getName());
  System.out.println(Arrays.toString(entry.getValue()));
}
```

```
powerSupply02.state
[0.1, 0.9]
powerSupply01.state
[0.1, 0.9]

-----------

powerSupply02.state
[0.1, 0.9]
powerSupply01.state
[0.5326231691078561, 0.46737683089214377]
```

Le langage BRICKS : configuration

Plusieurs algorithmes d'inférence…

mais aussi plusieurs moteurs de calcul

- Java natif ou routines optimisées (de type BLAS)
- Sur CPU standard ou en GPU (CUDA)

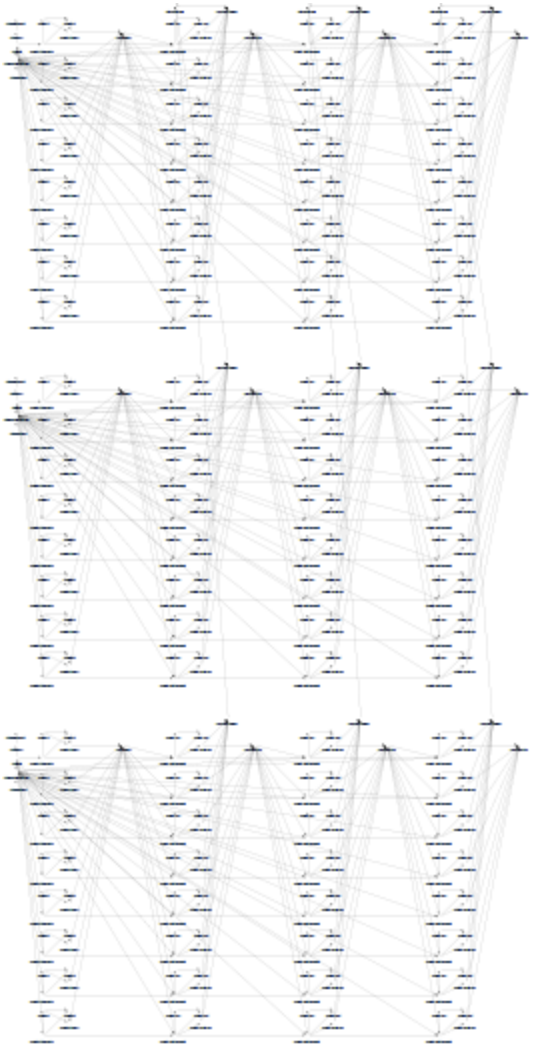
Une large bibliothèque de

- types prédéfinis
- distributions de probabilité
- fonctions déterministes pour certaines très fortement optimisées

Exemples d'applications industrielles de BRICKS

Analyse du risque d'effondrement d'une plateforme pétrolière

- Réseaux bayésiens dynamiques
- Quelques lignes de code « BRICKS » pour modéliser un système très complexe



Exemples d'applications industrielles de BRICKS

Analyse des risques opérationnels (aéronautique)

- Au niveau d'une flotte entière
- Mode « what-if » d'aide à la décision en temps réel

Analyse des risques intégrant des aspects techniques, environnementaux, humains et organisationnels (nucléaire)

- Base de connaissances et IHM client WEB
- Instanciation et exploitation des modèles par les entités métiers

Conclusion

BRICKS

=

les réseaux bayésiens

passés à l'échelle des applications industrielles

Expressivité

Ingénierie des modèles

Performances en exploitation