Learning any memory-less discrete semantics for dynamical systems represented by logic programs

Learning dynamics from any semantics

Tony Ribeiro[1,2,3], Maxime Folschette[4], Morgan Magnin[2,3], Katsumi Inoue[3]

1. Independant Researcher
2. Université de Nantes, Centrale Nantes, CNRS, LS2N, F-44000 Nantes, France
3. National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
4. Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

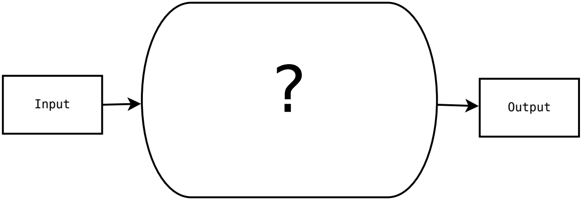
25th October 2021, ILP, Online

Outline

Outline

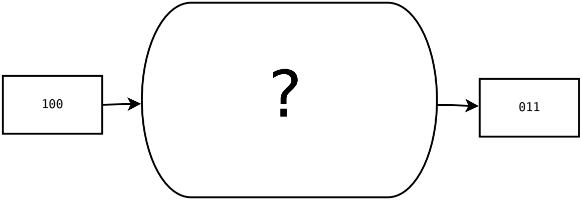
Research area

Idea: given a set of input/output states of a black-box system, learn its internal mechanics.



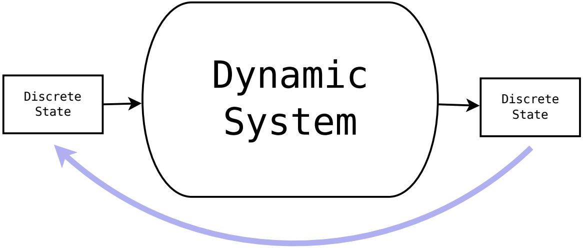
Research area

Discrete system: input/output are vectors of same size which contain discrete values.



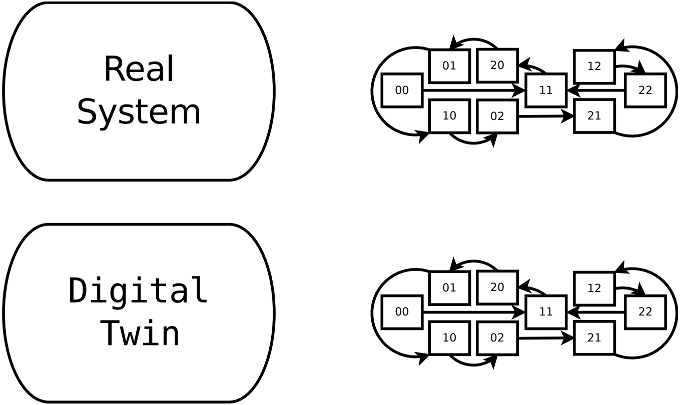
Research area

Dynamic system: input/output are states of the system and output becomes the next input.



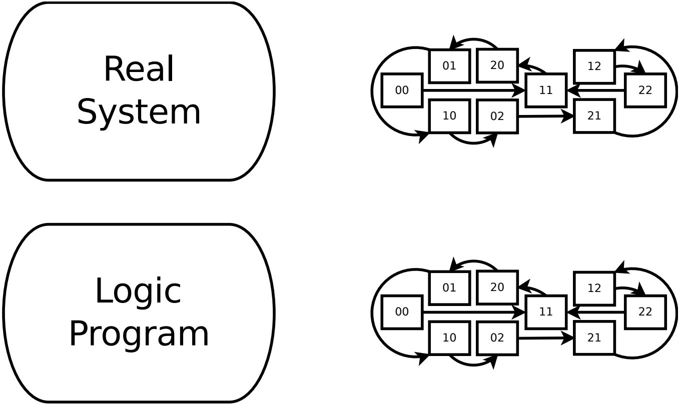
Research area

Goal: produce an artificial system with the same behavior as the one observed, i.e., a digital twin.



Research area

Representation: propositional logic programs with annotated atoms encoding multi-valued variables.

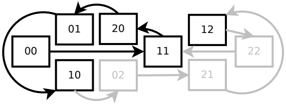


Research area

Method: learn the dynamics of systems from the observations of some of its state transitions.

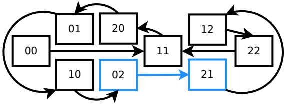
DATA

?



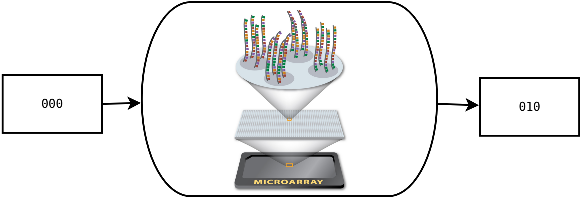
RESULTS

a(0,T) :- a(2,T-1)
a(1,T) :- a(0,T-1), b(0,T-1).
a(2,T) :- a(1,T-1)
● a(2,T) :- a(0,T-1), b(2,T-1).
...
b(0,T) :- a(1,T-1).
● b(1,T) :- b(2,T-1).
b(2,T):- b(2,T-1).
...



Motivation

Data: time series of gene expression levels in a organic cell.
Goal: model gene interactions to understand their influences.

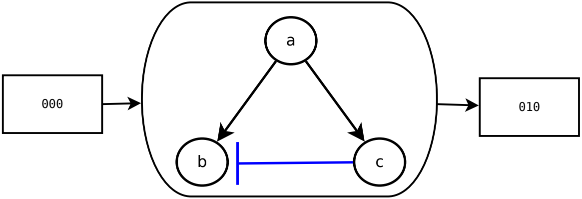


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of gene expression levels in a organic cell.
Goal: model gene interactions to understand their influences.

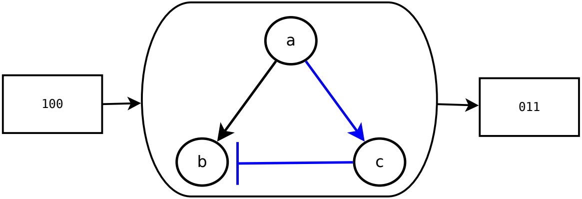


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of gene expression levels in a organic cell.
Goal: model gene interactions to understand their influences.

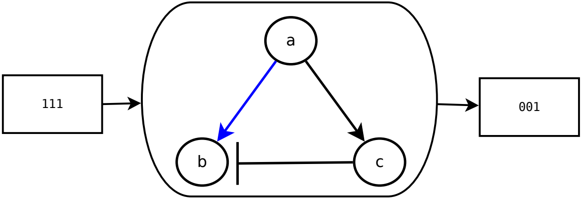


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: time series of gene expression levels in a organic cell.
Goal: model gene interactions to understand their influences.

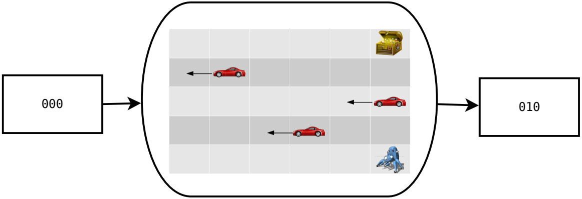


Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

Motivation

Data: observations of environment evolution according to a robot actions.
Goal: produce a predictive model of the environment for action planning.



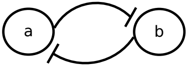
Example (Possible Applications)

- **Bioinformatics**: Construct gene regulatory networks.
- **Robotics**: Learn action models from robot observations.

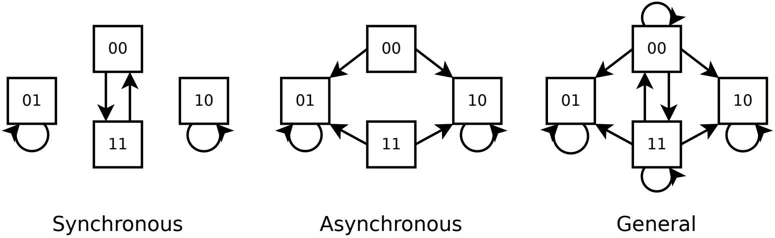
Outline

Dynamical Semantics

Boolean network transitions differ according to the update semantics used.



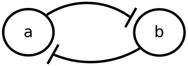
f(a) := not b.
f(b) := not a.



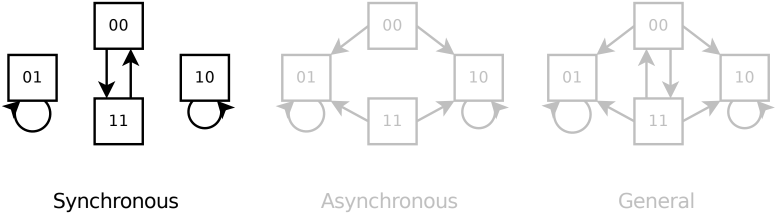
- Synchronous: all variables are updated
- Asynchronous: only one variable is updated
- General: any number of variables can be updated

Dynamical Semantics

Boolean network transitions differ according to the update semantics used.



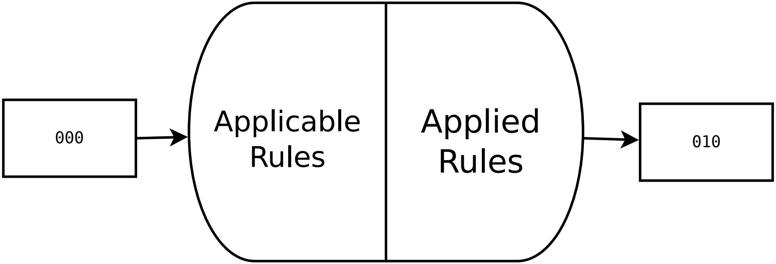
f(a) := not b.
f(b) := not a.



- Synchronous: all variables are updated
- Asynchronous: only one variable is updated
- General: any number of variables can be updated

What is a semantics?

For those three semantics at least, it is about computing the next state by selecting among applicable local rules the ones that will be applied.

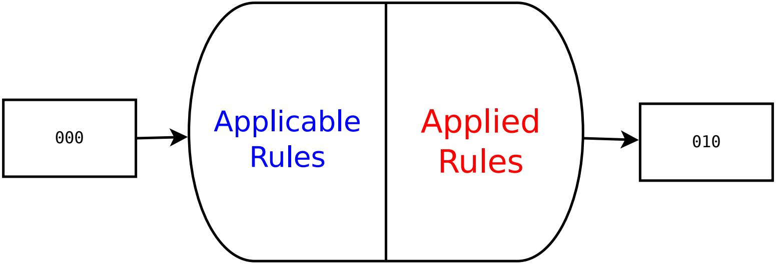


Semantics: what is an applicable rule and what is a valid set of applied rule.

The three semantics that are considered here differ on the selection but share the same definition of what is an applicable rule.

What is a semantics?

For those three semantics at least, it is about computing the next state by selecting among applicable local rules the ones that will be applied.



Semantics: what is an applicable rule and what is a valid set of applied rule.

The three semantics that are considered here differ on the selection but share the same definition of what is an applicable rule.

Learning algorithm intuition: classification problem

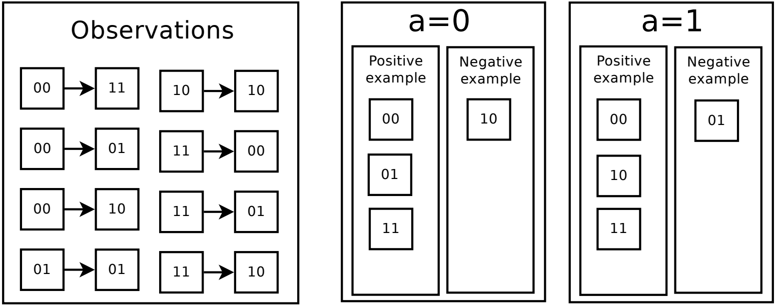
What is an applicable rule?

Learning algorithm intuition: classification problem

What is an applicable rule? The conditions so that a variable can take a certain value in next state.

Learning algorithm intuition: classification problem

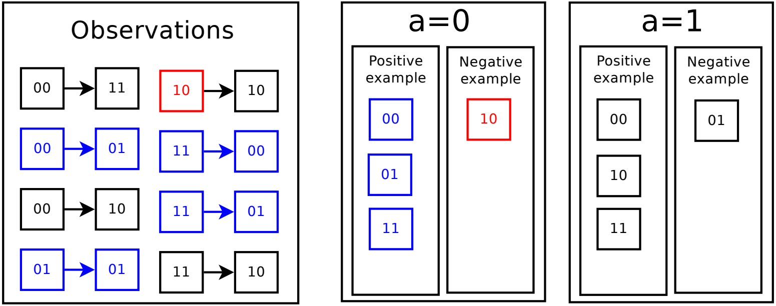
What is an applicable rule? The conditions so that a variable can take a certain value in next state.



Equivalent to a classification problem: for each value of a variable, what is a typical state where the variable can takes this value in the next state ?

Learning algorithm intuition: classification problem

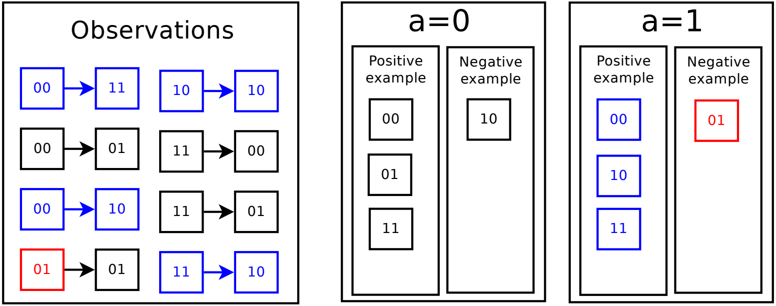
What is an applicable rule? The conditions so that a variable can take a certain value in next state.



Equivalent to a classification problem: for each value of a variable, what is a typical state where the variable can takes this value in the next state ?

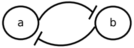
Learning algorithm intuition: classification problem

What is an applicable rule? The conditions so that a variable can take a certain value in next state.

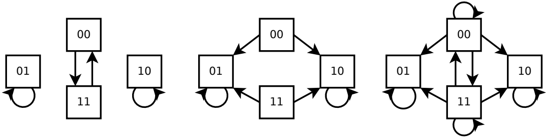


Equivalent to a classification problem: for each value of a variable, what is a typical state where the variable can takes this value in the next state ?

General Usage LFIT Algorithm (**GULA**) ouptut



f(a) := not b.
f(b) := not a.



Synchronous

// f(a) := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// f(b) := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

Asynchronous

// f(a) := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// f(b) := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

// Default rules
$a_t^0 \leftarrow a_{t-1}^0$
$a_t^1 \leftarrow a_{t-1}^1$
$b_t^0 \leftarrow b_{t-1}^0$
$b_t^1 \leftarrow b_{t-1}^1$

General

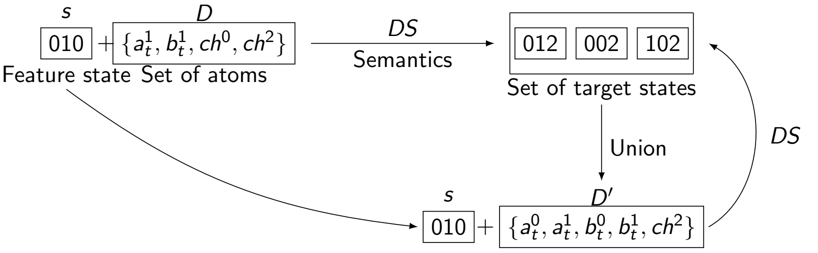
// f(a) := not b
$a_t^0 \leftarrow b_{t-1}^1$
$a_t^1 \leftarrow b_{t-1}^0$

// f(b) := not a
$b_t^0 \leftarrow a_{t-1}^1$
$b_t^1 \leftarrow a_{t-1}^0$

// Default rules
$a_t^0 \leftarrow a_{t-1}^0$
$a_t^1 \leftarrow a_{t-1}^1$
$b_t^0 \leftarrow b_{t-1}^0$
$b_t^1 \leftarrow b_{t-1}^1$

Pseudo-idempotent semantics

GULA can model observations from any pseudo-idempotent semantics.



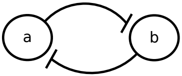
$$\longrightarrow DS(s, D) = DS\Big(s, \bigcup_{s' \in DS(s,D)} s'\Big)$$

where DS is the dynamical semantics, and D is the head of rules of a multi-valued logic program that match the sate s.

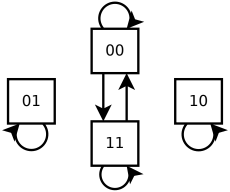
Outline

What about others semantics?

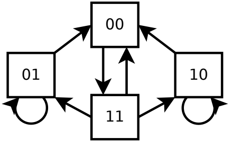
Three examples of arbitrary semantics.



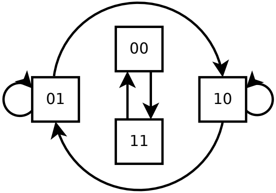
f(a) := not b.
f(b) := not a.



All or nothing change



Degradation

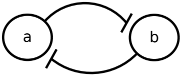


Inverse all values

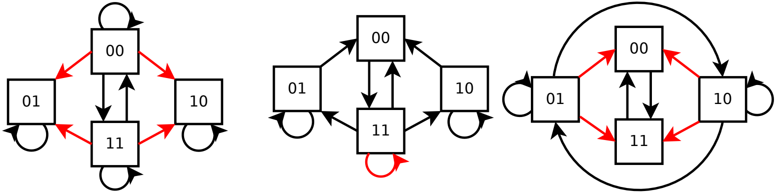
How can we learn a program able to reproduce these behaviors?

What is impossible?

Problem: If GULA learns a program from those transitions and we apply the synchronous semantics, this is what happens:



f(a) := not b.
f(b) := not a.

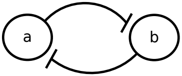


All or nothing change   Degradation   Inverse all values

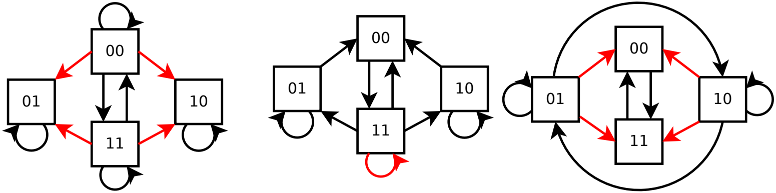
Can we prevent impossible transitions?

What is impossible?

Problem: If GULA learns a program from those transitions and we apply the synchronous semantics, this is what happens:



f(a) := not b.
f(b) := not a.

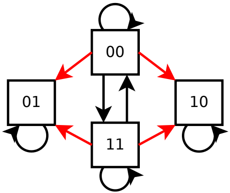


All or nothing change　　Degradation　　Inverse all values

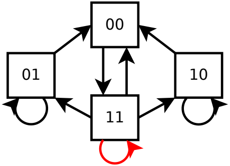
Can we prevent impossible transitions? Yes: with constraints!

Classification modeling of impossibility

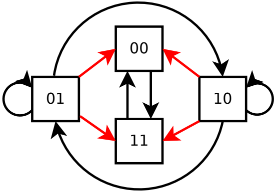
Idea: **GULA** can learn constraints using observations as negative examples.



All or nothing change



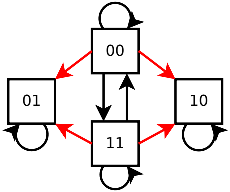
Degradation



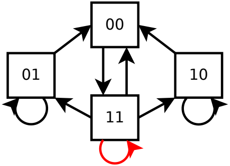
Inverse all values

Classification modeling of impossibility

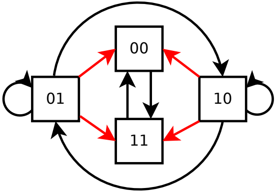
Idea: **GULA** can learn constraints using observations as negative examples.



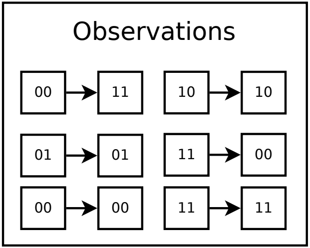
All or nothing change

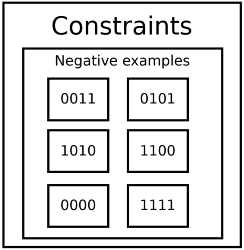


Degradation

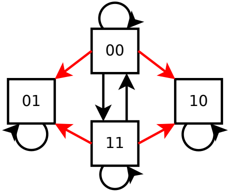


Inverse all values



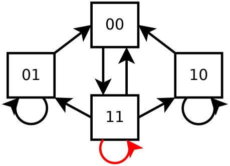


Examples of learned programs



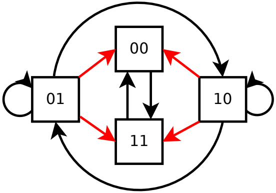
All or nothing change

a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Conservation rules
a(0,T) :- a(0,T-1).
a(1,T) :- a(1,T-1).
b(0,T) :- b(0,T-1).
b(1,T) :- b(1,T-1).
Constraints
:- a(0,T), b(1,T), b(0,T-1).
:- a(1,T), b(0,T), a(0,T-1).
:- a(1,T), b(0,T), b(1,T-1).
:- a(0,T), b(1,T), a(1,T-1).



Degradation

a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Conservation rules
a(1,T) :- a(1,T-1).
b(1,T) :- b(1,T-1).
Degradation
a(0,T) :- a(1,T-1).
b(0,T) :- b(1,T-1).
Constraints
:- a(1,T), b(1,T), a(1,T-1).



Inverse all values

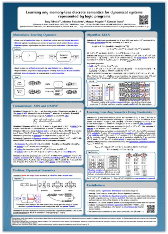
a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Inverse value
a(0,T) :- a(1,T-1).
a(1,T) :- a(0,T-1).
b(0,T) :- b(1,T-1).
b(1,T) :- b(0,T-1).
Constraints
:- a(1,T), b(1,T), a(1,T-1).
:- a(0,T), b(0,T), a(0,T-1).
:- a(1,T), b(1,T), b(1,T-1).
:- a(0,T), b(0,T), b(0,T-1).

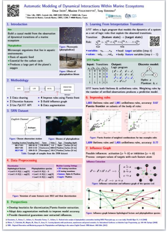
Outline

Conclusions

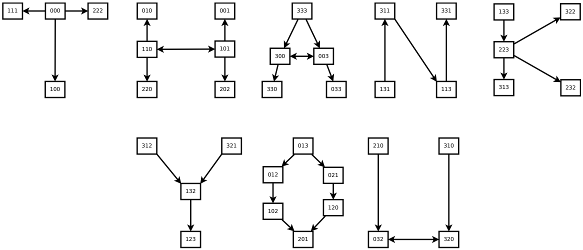
- **Previous works:** Synchronous deterministic transitions only.
- **Novelty:** Learn from any memory-less discrete dynamical semantics.
- **Application:** Selection of a semantics, can be done a posteriori.
- **Weakness:** Too costly/sensitive to deal with real systems.
- **Outlook:** Development of heuristic approaches to tackle real data.
- Source code (Python) available as open source on Github.
- Join us at posters session for details about theory and applications.

Manuscript





Source Code


111
000
222
100
010
001
110
101
220
202
333
300
003
330
033
311
331
131
113
133
322
223
313
232
312
321
132
123
013
012
021
102
120
201
210
310
032
320