Identifying Duplicates in Large Collections of Petri Nets and Nested-Unit Petri Nets

Pierre Bouvier Hubert Garavel

INRIA – Univ. Grenoble Alpes

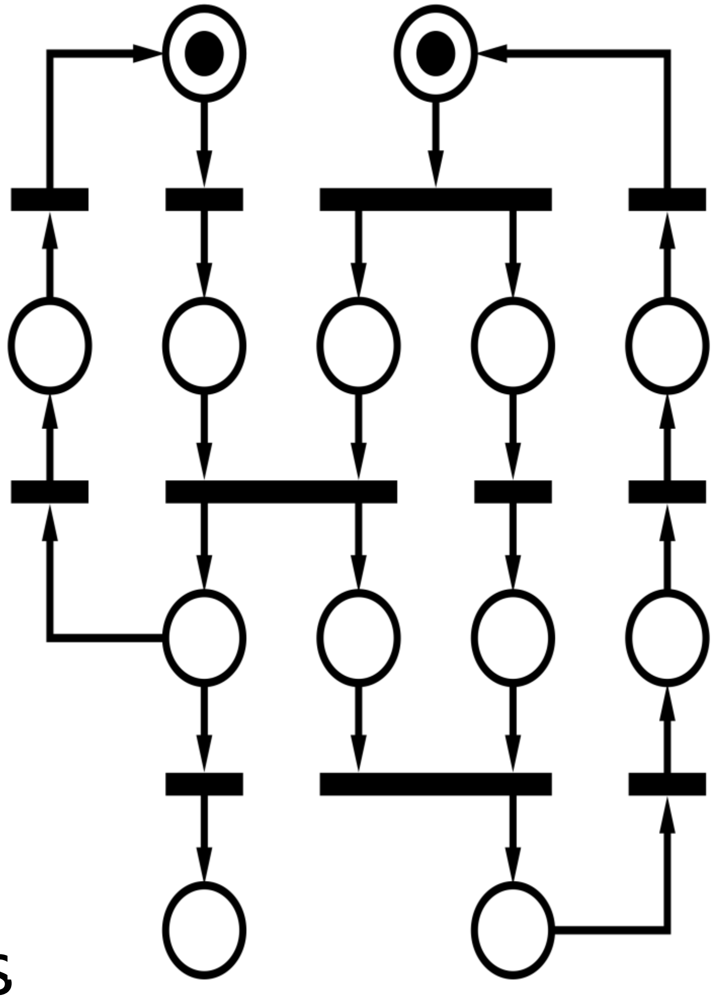
France

1. Motivation

P/T Nets

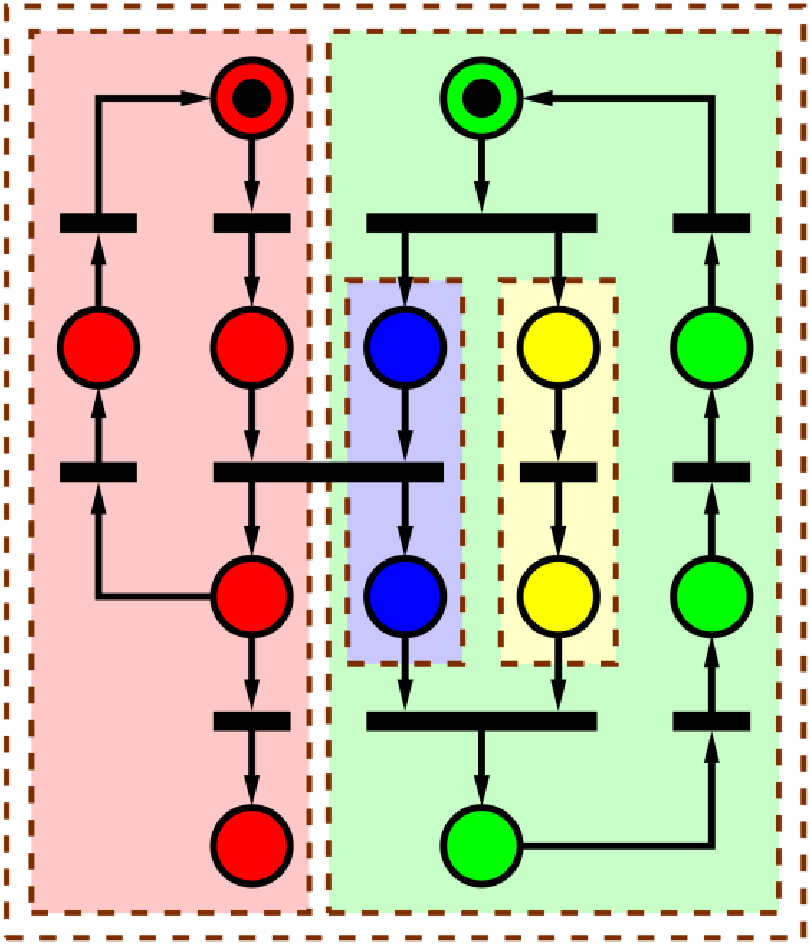
- Standard notion of Petri nets:
  - places, transitions, arcs
  - markings, tokens, firing rules
- We assume that nets are:
  - ordinary (no multiple arcs)
  - safe (at most one token per place in any reachable marking)

  If not: over-approximations
- We do not handle colored nets



Nested Unit Petri Nets (NUPNs)

- Extension of Petri nets
  - units encapsulate places
  - units are pairwise disjoint
  - units are recursively nested (they form a tree of units)
- Transition firing rules are exactly those of Petri nets
- Logarithmic gains when storing reachable markings



Collections of Petri nets

- Collections of benchmarks are crucial for:
  - testing software under development
  - software competitions (Model Checking Contest)

- Building "good" collections is difficult:
  - models originate from many authors
  - collections grow as time passes
  - properly maintaining them is tedious
  - few people do it

Duplicates in collections

- Duplicates = "similar" models in a collection
- Multiple causes:
  - models coming from many sources
  - several maintainers adding models in a collection
  - transformations applied to models
- Bad consequences:
  - wasted disk space
  - redundant calculations
  - biases in competitions
  - tedious discussions between users, maintainers, etc.

Our benchmarks: 4 collections

- Collection 1 (Univ. Zielona Gora, Poland)
  - 244 P/T nets obtained from the HIPPO Web service
- Collection 2 (Model Checking Contest, 2022 edition)
  - 1387 P/T nets accumulated since 2011
    (56% ordinary and safe, 50% non-trivial NUPNs)
- Collection 3 (INRIA Grenoble, France)
  - 16,200 NUPNs from multiple sources
- Collection 4 (INRIA Grenoble)
  - 241,657 NUPNs (extension of Collection 3, with many permutations, and file deduplication)

Benchmarks: statistics

	collection 1		collection 2		collection 3		collection 4	
	avg.	max.	avg.	max.	avg.	max.	avg.	max.
#places	15.4	200	2,801.5	537,708	345.8	131,216	740.8	131,216
#trans.	11.8	51	10,798	1,070,836	7,998.1	16,967,720	15,645	16,967,720
#arcs	34.2	400	83,384	25,615,632	71,217.9	146,528,584	113,102.9	146,528,584
#units	—	—	1,970	537,709	123.4	78,644	270.4	78,644
height	—	—	15.4	2,891	4.3	2,891	6.3	2,891
width	—	—	1,959.1	537,708	117.6	78,643	259.9	78,643

- The four collections are diverse
- Some models are huge (25 M places, 146 M trans.)
- NUPN structures are involved (large trees of units)

How can we find duplicates in these collections?

2. Basic methods

File deduplication

- Basic idea:
  - each net is stored as a file (in PNML format)
  - use tools that search for identical files on a disk
  - e.g., Fdupes (on Linux), Jdupes (on Linux), etc.

- Caveat:
  - PNML offers too much lexical/syntactic freedom
  - two identical nets may differ by one extra space
  - thus, file deduplication will miss many duplicates

Pre-canonization

- Convert nets from PNML format to NUPN format
  - using the PNML2NUPN tool (LIP6 Paris)
  - the NUPN format is stricter and more concise
- Put NUPN files under "pre-canonical" form:
  - using CAESAR.BDD -precanonical-nupn (Grenoble)
  - remove blank lines, extra spaces, tabulations, etc.
  - renumber from zero all places, transitions, and units
  - sorts all lists of places, transitions, and units...
- Finally, invoke a file deduplication tool

3. Graph-isomorphism methods

Graph isomorphism (1/2)

- Chosen graph model:
  - vertices are colored
  - edges are oriented
- Isomorphism between two graphs:
  - existence of a bijection between vertices
  - that preserves edges and colors

Definition 6. *Two colored graphs* $G = (V, E, c)$ *and* $G' = (V', E', c')$ *are* isomorphic *iff there exists a bijection* $\pi_v : V \to V'$ *such that:*
- $(\forall v_1, v_2 \in V)\ (v_1, v_2) \in E \Leftrightarrow (\pi_v(v_1), \pi_v(v_2)) \in E'$.
- $(\forall v \in V)\ c(v) = c'(\pi_v(v))$.

Graph isomorphism (2/2)

- Problem complexity:

  P ⊆ GI (Graph Isomorphism) ⊆ QP (Quasi Polynomial) ⊆ NP

  - recently, GI = QP according to L. Babai (2019)

- Various algorithms:
  - Weisfeiler-Leman (1968)
  - Luks (1982)

- Many tools: Bliss, Conauto, Nishe, Saucy, etc.
  - among them, we select Nauty and Traces

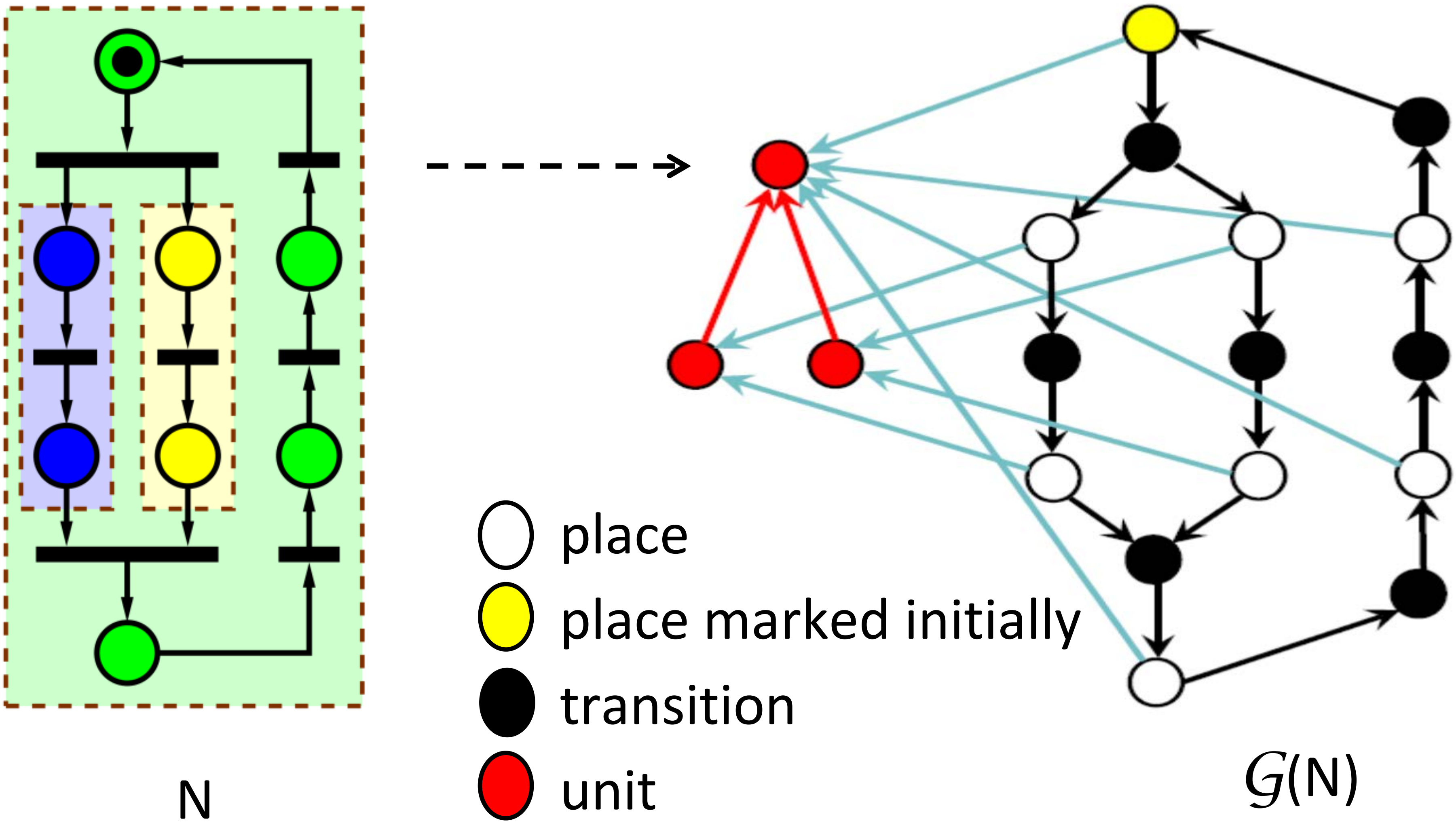
Net Isomorphism

- Isomorphism between two NUPNs (or P/T nets):
  - there exist three bijections between places, transitions, and units
  - that preserve arcs, initial markings, root units, inclusion between units, containment of places in units

Definition 7. *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *and* $N' = (P', T', F', M_0', U', u_0', \sqsubseteq', \mathsf{unit}')$ *be two NUPNs.* N *and* N' *are said to be* isomorphic *iff there exist three bijections* $\pi_p : P \rightarrow P'$, $\pi_t : T \rightarrow T'$, *and* $\pi_u : U \rightarrow U'$ *such that:*

- $(\forall (p, t) \in P \times T)\ (p, t) \in F \Leftrightarrow (\pi_p(p), \pi_t(t)) \in F'$.
- $(\forall (t, p) \in T \times P)\ (t, p) \in F \Leftrightarrow (\pi_t(t), \pi_p(p)) \in F'$.
- $(\forall p \in P)\ p \in M_0 \Leftrightarrow \pi_p(p) \in M_0'$.
- $u_0' = \pi_u(u_0)$.
- $(\forall u_1, u_2 \in U)\ u_1 \sqsubseteq u_2 \Leftrightarrow \pi_u(u_1) \sqsubseteq' \pi_u(u_2)$
- $(\forall p \in P)\ \mathsf{unit}'(\pi_p(p)) = \pi_u(\mathsf{unit}(p))$.

Translation: NUPNs →colored graphs



Net isomorphism in terms of graphs

Proposition 1. *Two NUPNs N and N' are isomorphic iff their corresponding graphs $\mathcal{G}_N$ and $\mathcal{G}_{N'}$ are isomorphic.*

- Application to Collection 2 "MCC" (1387 nets):
  - NUPN→graph translator (in Python) + Nauty (in C)
  - parallel runs: (one server, 60 minutes, 96 GB) per net
  - low success rate: 22.4% – no duplicate found
- Experimented with 5 alternative translations:
  - fewer vertices, more colors, non-oriented graphs, etc.
  - use of Traces instead of Nauty
  - best success rate: 35.9% – no duplicate found

4. Specific methods for nets: Signatures

Net signatures

- A net signature function sig(N) computes a *digest* (or *checksum*) for a net N, and satisfies:
  N and N' are isomorphic nets $\Rightarrow$ sig(N) = sig(N')
- In practice, one uses the converse implication

Proposition 2. *If* sig *is a signature, then for any net* N *and any permutation* π *of places, transitions, and/or units,* $\text{sig}(\pi(N)) = \text{sig}(N)$.

- Many possible signatures, e.g.:
  - number of transitions
  - number of sink places
  - number of reachable markings $\rightarrow$ too expensive!

One proposed signature function

- sig(N) = fixed-size tuple of 100+ natural numbers
  - places→16 fields, transitions→3 fields, units→13 fields
  - each field is either a natural or a 5-tuple of naturals (multiset hashing)
- Sample signature for a given net:

121-0-1-110-3457260137-0-2-118-336755784-0-0-0-748333948-1-10-1111-4036028534-0-0-0-748333948-11-20-2222-3840480353-0-0-0-
748333948-0-0-0-748333948-11-20-2222-3840480353-0-0-0-748333948-0-99-4150790648-2-2-4444-21470205-2-2-4444-21470205-2-2-
4444-21470205-2-2-4444-21470205-1111-2-2-3858300795-2-2-3858300795-12-11-622163923-11-622163923-0-11-3856429020-11-121-
242-688397522-11-15643205-22-894725254-1-11-15643205-13-370702091-11-22-894725254-0-220-204525584-0-220-204525584-
19139339-2032892459-822461942-4275843631

- Implemented in the CAESAR.BDD tool (Grenoble)
  - 0.12 second per net on average

5. Specific methods for nets: Canonization

Net canonization

- A net canonization function can(N) permutes the places/transitions/units of a net N, and satisfies:

  can(N) = can(N') $\Rightarrow$ N and N' are isomorphic nets

- This is the reverse implication of signatures

- There may be several canonization functions

One proposed canonization function

- can(N) = successive composition of 3 functions:
- 1. unit-sorting function
  - for each unit, we compute a 35-tuple of fields
  - we sort this tuple lexicographically (using Unix sort)
  - this gives a (possibly non unique) permutation of units
- 2. place-sorting function
  - for each place, we compute a 27-tuple of fields, etc.
- 3. transition-sorting function
  - for each transition, we compute a 2-tuple of fields, etc.

Proposed canonization function

- Implementation:
  - the CAESAR.BDD tool computes the permutations
  - the NUPN_INFO tool applies the permutations
  - 8 seconds per net on average
  - finally, a file deduplication tool is invoked

- Relation between canonization and signatures:
  - if each of the three permutation is unique, can(N) is also a signature function, i.e.:
    can(N) = can(N') ⇔ N and N' are isomorphic nets

6. Experimental results

Combination of methods

- No single method solves the problem efficiently
- 5 methods are applied in combination
- By order of increasing complexity:
  - file deduplication
  - pre-canonization (+ file deduplication)
  - signatures
  - canonization (+ file deduplication)
  - graph-isomorphism tool

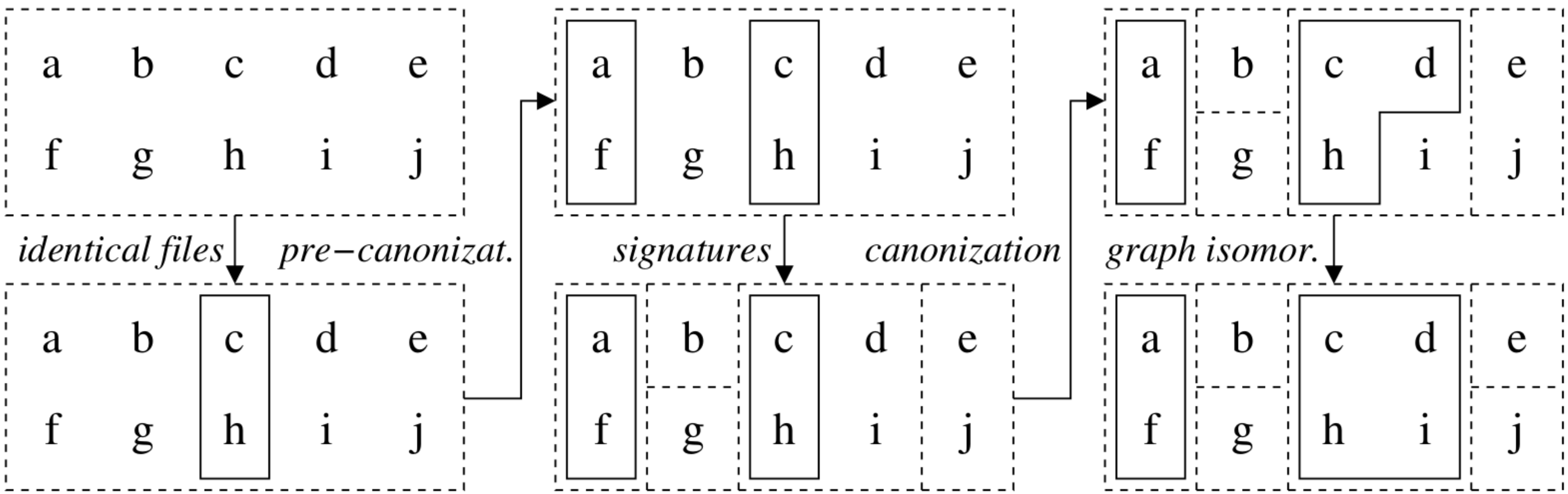
Approximated equivalence relation

- Positive methods detect isomorphic nets:
  - file deduplication, pre-canonization, canonization, graph isomorphism
  - "certain" equivalence classes increase by merging

- Negative methods detect non-isomorphic nets:
  - signatures, canonization (if permutations are unique), graph isormorphism
  - "potential" equivalence classes decrease by splitting (i.e., partition refinement)

Sample collection of 10 nets

- straight boxes: "certain" equivalence classes
- dotted boxes: "potential" equivalence classes



Results on the 4 collections

	collection 1			collection 2			collection 3			collection 4		
	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)
identical files	4.10	0.00	95.90	0.00	0.00	100.0	0.00	0.00	100.0	0.00	0.00	100.0
pre-canonizat.	4.10	0.00	95.90	—	—	—	0.17	0.00	99.83	22.35	0.00	77.65
signatures	4.10	86.88	9.02	0.00	98.56	1.44	0.17	92.87	6.96	22.35	0.12	77.53
canonization	5.74	91.39	2.87	0.58	98.84	0.58	2.26	94.87	2.87	79.44	4.74	15.82
graph isomor.	6.97	93.03	0.00	0.58	99.42	0.00	2.79	97.20	0.01	90.05	9.01	0.94

- Collections 1, 2, 3 have few duplicates (< 7%)
- Collection 4 has many duplicates (> 90%)
- High success rate (99-100%) but unknowns remain
- Experiments done on Grid 5000 clusters

Duplicates found in MCC collection

- In MCC model CloudReconfiguration (2017):
  - reconf_3_05 and reconf_3_15 are duplicates
- In MCC model DNAwalker (2016):
  - dnawalk-04 and dnawalk-07
  - dnawalk-05 and dnawalk-06
  - dnawalk-08 and dnawalk-10
  - dnawalk-09 and dnawalk-11
  - dnawalk-12 and dnawalk-13
  - dnawalk-14 and dnawalk-15
  - dnawalk-16 and dnawalk-17

  } potential duplicates *(these nets are neither ordinary nor safe)*

7. Conclusion

Conclusion

- A concrete, useful problem:
  - detecting duplicates in large sets of P/T-nets or NUPNs
- A pragmatic combination of approaches:
  - file deduplication and pre-canonization
  - signatures
  - canonization
  - reduction to graph isomorphism
- Application to 4 large collections:
  - from 244 to 241,000 nets
  - sucesss rate: 99-100%

Future work

- Enhance signature and canonization functions
  - reduce the number of components in tuples
- Additional approach based on SMT solving
  - express net isomorphism as QF_IDL formulas
- Extend the approach to:
  - non-ordinary and non-safe nets
    (currently handled using over-approximations)
  - colored nets