pmc

November 15, 2025

[1]:
```
import pm4py
import pandas as pd
import numpy as np
from pathlib import Path

print("pm4py version:", getattr(pm4py, "__version__", "unknown"))

from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristics_miner
from pm4py.objects.conversion.bpmn import converter as bpmn_converter
from pm4py.objects.conversion.heuristics_net import converter as hn_converter
from pm4py.objects.conversion.bpmn import converter as bpmn_to_petri_converter
from pm4py.algo.conformance.alignments.petri_net import algorithm as␣
 ↪alignments_alg


def export_bpmn(bpmn_graph, path):
    path = str(path)
    try:
        pm4py.write_bpmn(bpmn_graph, path)
    except Exception:
        try:
            from pm4py.objects.bpmn.exporter import exporter as bpmn_exporter
            bpmn_exporter.apply(bpmn_graph, path)
        except Exception as e:
            raise RuntimeError(f"Could not export BPMN: {e}")

def bpmn_to_petri(bpmn_graph):
    net, im, fm = bpmn_to_petri_converter.apply(bpmn_graph)
    return net, im, fm

try:
    # PM4Py   2.x (stable path)
    from pm4py.algo.conformance.tokenreplay import algorithm as token_replay
except Exception:
    from pm4py.algo.conformance import tokenreplay as token_replay
```

```
# Alignments
try:
    from pm4py.algo.conformance.alignments.petri_net import algorithm as␣
 ↪alignments_alg
    _HAS_ALIGNMENTS = True
except Exception:
    _HAS_ALIGNMENTS = False

# Precision
precision_apply = None
# Newer path
try:
    from pm4py.algo.conformance.precision.variants import etconformance_token␣
 ↪as _prec_et
    precision_apply = _prec_et.apply
except Exception:
    pass

if precision_apply is None:
    try:
        from pm4py.algo.conformance.precision import algorithm as _prec_algo
        precision_apply = _prec_algo.apply
    except Exception:
        precision_apply = None  # we'll handle below

# Generalization (optional; not in all builds)
def _try_generalization(event_log, net, im, fm):
    try:
        from pm4py.algo.evaluation.generalization import evaluator as␣
 ↪gen_eval_new
        val = gen_eval_new.evaluate(event_log, net, im, fm)
        if isinstance(val, dict) and "generalization" in val:
            return float(val["generalization"])
        return float(val)
    except Exception:
        try:
            from pm4py.evaluation.generalization import evaluator as␣
 ↪gen_eval_old
            val = gen_eval_old.evaluate(event_log, net, im, fm)
            if isinstance(val, dict) and "generalization" in val:
                return float(val["generalization"])
            return float(val)
        except Exception:
            return np.nan

# Built-in simplicity (optional; not in all builds)
def _try_builtin_simplicity(net):
    try:
        from pm4py.algo.evaluation.simplicity import evaluator as simp_eval_new
        val = simp_eval_new.apply(net)
        if isinstance(val, dict) and "simplicity" in val:
            return float(val["simplicity"])
        return float(val)
    except Exception:
        try:
            from pm4py.evaluation.simplicity import evaluator as simp_eval_old
            val = simp_eval_old.apply(net)
            if isinstance(val, dict) and "simplicity" in val:
                return float(val["simplicity"])
            return float(val)
        except Exception:
            return np.nan

def compute_all_metrics(event_log, net, im, fm, prefer_alignments=False):
    # fitness
    fitness = np.nan
    if prefer_alignments and _HAS_ALIGNMENTS:
        try:
            al = alignments_alg.apply(event_log, net, im, fm)
            per_trace = []
            for x in al:
                if isinstance(x, dict) and "fitness" in x:
                    per_trace.append(x["fitness"])
            if per_trace:
                fitness = float(np.mean(per_trace))
        except Exception:
            fitness = np.nan

    if np.isnan(fitness):
        try:
            diag = token_replay.apply(event_log, net, im, fm)
            # diag is a list per trace; each has 'trace_fitness' (0..1)
            per_trace = []
            for x in diag:
                if isinstance(x, dict) and "trace_fitness" in x:
                    per_trace.append(x["trace_fitness"])
            if per_trace:
                fitness = float(np.mean(per_trace))
        except Exception:
            fitness = np.nan

    if precision_apply is not None:
        try:
            precision = float(precision_apply(event_log, net, im, fm))
        except Exception:
            precision = np.nan
    else:
        precision = np.nan

    generalization = _try_generalization(event_log, net, im, fm)

    simplicity_builtin = _try_builtin_simplicity(net)

    num_places = len(net.places)
    num_transitions = len(net.transitions)
    num_arcs = len(net.arcs)
    size = num_places + num_transitions + num_arcs

    # Size-based simplicity
    simplicity_size = 1.0 / (1.0 + np.log1p(size))

    # Connectivity simplicity (inverse avg degree)
    nodes = (num_places + num_transitions)
    connectivity_simplicity = 1.0 / (1.0 + (2.0 * num_arcs / nodes)) if nodes >␣
↪0 else np.nan

    return {
        "fitness": fitness,
        "precision": precision,
        "generalization": generalization,
        "simplicity_builtin": simplicity_builtin,
        "simplicity_size": float(simplicity_size),
        "simplicity_connectivity": float(connectivity_simplicity),
        "places": int(num_places),
        "transitions": int(num_transitions),
        "arcs": int(num_arcs),
        "size": int(size),
    }
```

```
pm4py version: 2.7.18
```

[2]:
```
from pm4py.objects.log.util import sorting
from pm4py.objects.conversion.log import converter as log_converter

LOG_PATH = "bpi-chall.xes"

elog = pm4py.read_xes(LOG_PATH)

if isinstance(elog, pd.DataFrame):
    elog = log_converter.apply(elog, variant=log_converter.Variants.
↪TO_EVENT_LOG)
```

```python
elog = sorting.sort_timestamp(elog, timestamp_key="time:timestamp")

print(type(elog))
print(f"Number of cases: {len(elog)}")
```

```
/Users/manueljulianasbeck/anaconda3/lib/python3.10/site-
packages/pm4py/utils.py:991: UserWarning: Install the optional requirement
`rustxes` to import/export files faster.
  warnings.warn("Install the optional requirement `rustxes` to import/export
files faster.")
/Users/manueljulianasbeck/anaconda3/lib/python3.10/site-
packages/pm4py/util/dt_parsing/parser.py:82: UserWarning: ISO8601 strings are
not fully supported with strpfromiso for Python versions below 3.11
  warnings.warn(
/Users/manueljulianasbeck/anaconda3/lib/python3.10/site-
packages/tqdm/auto.py:22: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
parsing log, completed traces :: 100%|      | 31509/31509 [00:34<00:00,
913.51it/s]
```

```
<class 'pm4py.objects.log.obj.EventLog'>
Number of cases: 31509
```

[ ]:
```python
from pathlib import Path
import pm4py
import pandas as pd
import numpy as np
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristics_miner
from pm4py.objects.conversion.bpmn import converter as bpmn_converter
from pm4py.objects.conversion.heuristics_net import converter as hn_converter

# helpers
from pm4py.algo.evaluation.replay_fitness import algorithm as fitness_eval

def compute_fast_metrics(event_log, net, im, fm):
    fitness = np.nan
    try:
        fit_res = fitness_eval.apply(
            event_log, net, im, fm,
            variant=fitness_eval.Variants.TOKEN_BASED
        )
        # fit_res is typically a dict
        if isinstance(fit_res, dict) and "log_fitness" in fit_res:
            fitness = float(fit_res["log_fitness"])
        else:
            fitness = float(fit_res)
    except Exception as e:
        print("Token-based fitness error:", type(e), e)

    try:
        from pm4py.algo.evaluation.simplicity import algorithm as simp_alg
        simp_res = simp_alg.apply(net)
        if isinstance(simp_res, dict) and "simplicity" in simp_res:
            simplicity_builtin = float(simp_res["simplicity"])
        else:
            simplicity_builtin = float(simp_res)
    except Exception:
        simplicity_builtin = np.nan

    num_places = len(net.places)
    num_transitions = len(net.transitions)
    num_arcs = len(net.arcs)
    size = num_places + num_transitions + num_arcs

    # size-based simplicity
    simplicity_size = 1.0 / (1.0 + np.log1p(size)) if size > 0 else np.nan

    # connectivity-based simplicity
    nodes = num_places + num_transitions
    connectivity_simplicity = 1.0 / (1.0 + (2.0 * num_arcs / nodes)) if nodes >␣
↪0 else np.nan

    return {
        "fitness": fitness,
        "precision": np.nan,
        "generalization": np.nan,
        "simplicity_builtin": simplicity_builtin,
        "simplicity_size": float(simplicity_size),
        "simplicity_connectivity": float(connectivity_simplicity),
        "places": int(num_places),
        "transitions": int(num_transitions),
        "arcs": int(num_arcs),
        "size": int(size),
    }

def im_param_dict(noise_thr: float):
    # trying enum on variant.value
    try:
        key = inductive_miner.Variants.IMf.value.Parameters.NOISE_THRESHOLD
        return {key: noise_thr}
    except Exception:
        pass
    # trying module-level params
    try:
        key = inductive_miner.Parameters.NOISE_THRESHOLD
        return {key: noise_thr}
    except Exception:
        pass
    # fallback
    return {"noise_threshold": noise_thr}

def hm_param_dict(dep_thr: float):
    try:
        key = heuristics_miner.Variants.CLASSIC.value.Parameters.
 ↪DEPENDENCY_THRESH
        return {key: dep_thr}
    except Exception:
        pass
    try:
        key = heuristics_miner.Variants.CLASSIC.value.Parameters.
 ↪DEPENDENCY_THRESHOLD
        return {key: dep_thr}
    except Exception:
        pass
    for name in ["DEPENDENCY_THRESH", "DEPENDENCY_THRESHOLD"]:
        try:
            key = getattr(heuristics_miner.Parameters, name)
            return {key: dep_thr}
        except Exception:
            pass
    for s in ["dependency_threshold", "dependency_thresh"]:
        return {s: dep_thr}

def export_bpmn(bpmn_graph, path):
    path = str(path)
    try:
        pm4py.write_bpmn(bpmn_graph, path)
    except Exception:
        try:
            from pm4py.objects.bpmn.exporter import exporter as bpmn_exporter
            bpmn_exporter.apply(bpmn_graph, path)
        except Exception as e:
            raise RuntimeError(f"Could not export BPMN: {e}")

outdir = Path("models")
outdir.mkdir(exist_ok=True)

candidates = []
```

[4]:
```python
from pm4py.objects.conversion.wf_net import converter as wf_net_converter
from pm4py.objects.process_tree.obj import ProcessTree
from pm4py.objects.conversion.process_tree import converter as pt_converter
from pm4py.objects.conversion.wf_net import converter as wf_net_converter
```

[13]:
```python
def to_bpmn(res):
    if isinstance(res, ProcessTree):
        return pt_converter.apply(res, variant=pt_converter.Variants.TO_BPMN)
    else:
        net, im, fm = res
        # convert petri net -> BPMN explicitly
        return wf_net_converter.apply(
            net, im, fm,
            variant=wf_net_converter.Variants.TO_BPMN
        )

n_thr_1 = [0.0, 0.2, 0.4, 0.6, 0.7, 0.8, 0.9]
n_thr_2 = [0.7, 1.0]

d_thr_1 = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8]
d_thr_2 = [0.6]

# Inductive Miner
for noise_thr in n_thr_2:
    params = im_param_dict(noise_thr)
    res = inductive_miner.apply(
        elog,
        variant=inductive_miner.Variants.IMf,
        parameters=params
    )

    if isinstance(res, ProcessTree):
        net, im, fm = pt_converter.apply(res, variant=pt_converter.Variants.
 ↪TO_PETRI_NET)
    else:
        net, im, fm = res

    bpmn_graph = to_bpmn(res)
    bpmn_path = outdir / f"candidate_IM_noise{str(noise_thr).replace('.','_')}.
 ↪bpmn"
    export_bpmn(bpmn_graph, bpmn_path)

    metrics = compute_fast_metrics(elog, net, im, fm)
    metrics.update({"algo": "InductiveMiner",
                    "params": {"noise_threshold": noise_thr},
                    "bpmn_path": str(bpmn_path)})
    candidates.append(metrics)
```

```
# Heuristic Miner
for dep_thr in d_thr_2:
    params = hm_param_dict(dep_thr)
    try:
        hnet = heuristics_miner.apply_heu(elog, parameters=params)
    except Exception:
        hnet = heuristics_miner.apply(elog, variant=heuristics_miner.Variants.
 ↪CLASSIC, parameters=params)

    net, im, fm = hn_converter.apply(hnet)

    bpmn_graph = wf_net_converter.apply(
        net, im, fm,
        variant=wf_net_converter.Variants.TO_BPMN
    )
    bpmn_path = outdir / f"candidate_HM_dep{str(dep_thr).replace('.','_')}.bpmn"
    export_bpmn(bpmn_graph, bpmn_path)

    metrics = compute_fast_metrics(elog, net, im, fm)
    metrics.update({"algo": "HeuristicsMiner",
                    "params": {"dependency_threshold": dep_thr},
                    "bpmn_path": str(bpmn_path)})
    candidates.append(metrics)
```

```
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[01:06<00:00, 238.58it/s]
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[00:52<00:00, 304.59it/s]
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[01:00<00:00, 263.48it/s]
```

[14]:
```
results = pd.DataFrame(candidates)
```

[15]:
```
# score
simplicity_cols = ["simplicity_builtin", "simplicity_size",␣
 ↪"simplicity_connectivity"]
results["simplicity_avg"] = results[simplicity_cols].apply(lambda row: np.
 ↪nanmean(row), axis=1)

feasible = results[results["fitness"] >= 0.80].copy()

if feasible.empty:
    print("No candidate reached fitness  0.80. Selecting the best fitness␣
 ↪overall and consider parameter retuning.")
    feasible = results.copy()
```

```
results["score"] = (
    0.55 * results["fitness"].fillna(0)
    + 0.25 * results["precision"].fillna(0)
    + 0.20 * results["simplicity_avg"].fillna(0)
)

feasible["score"] = (
    0.55 * feasible["fitness"].fillna(0)
    + 0.25 * feasible["precision"].fillna(0)
    + 0.20 * feasible["simplicity_avg"].fillna(0)
)

display_cols = [
    "algo", "params", "fitness", "precision", "generalization",
    "simplicity_builtin", "simplicity_size", "simplicity_connectivity",
    "size", "places", "transitions", "arcs", "score", "bpmn_path"
]
print("===Candidates (sorted by score) ===")
display(results.sort_values("score", ascending=False)[display_cols])

final_row = feasible.sort_values("score", ascending=False).iloc[0]
final_bpmn_path = final_row["bpmn_path"]
final_row
```

```
===Candidates (sorted by score) ===

              algo                            params   fitness  precision  \
7  HeuristicsMiner  {'dependency_threshold': 0.6}  0.954485        NaN
3  HeuristicsMiner  {'dependency_threshold': 0.1}  0.947731        NaN
4  HeuristicsMiner  {'dependency_threshold': 0.2}  0.944331        NaN
1   InductiveMiner       {'noise_threshold': 1.0}  0.911450        NaN
6   InductiveMiner       {'noise_threshold': 1.0}  0.911450        NaN
2  HeuristicsMiner  {'dependency_threshold': 0.0}  0.927458        NaN
5   InductiveMiner       {'noise_threshold': 0.7}  0.895042        NaN
0   InductiveMiner       {'noise_threshold': 0.9}  0.885040        NaN

   generalization  simplicity_builtin  simplicity_size  \
7             NaN            0.516949         0.149022
3             NaN            0.521368         0.149096
4             NaN            0.515152         0.149544
1             NaN            0.649485         0.167509
6             NaN            0.649485         0.167509
2             NaN            0.524229         0.149697
5             NaN            0.669065         0.157552
0             NaN            0.657895         0.155643

   simplicity_connectivity  size  places  transitions  arcs     score  \
7                 0.254167   301      42           80   179  0.586309
```

```
3                  0.255230   300      45            77    178   0.582965
4                  0.253731   294      43            76    175   0.580611
1                  0.282511   143      23            40     80   0.574598
6                  0.282511   143      23            40     80   0.574598
2                  0.255914   292      44            75    173   0.572091
5                  0.286154   209      39            54    116   0.566458
0                  0.284091   226      41            59    126   0.559947

                          bpmn_path
7     models/candidate_HM_dep0_6.bpmn
3     models/candidate_HM_dep0_1.bpmn
4     models/candidate_HM_dep0_2.bpmn
1   models/candidate_IM_noise1_0.bpmn
6   models/candidate_IM_noise1_0.bpmn
2     models/candidate_HM_dep0_0.bpmn
5   models/candidate_IM_noise0_7.bpmn
0   models/candidate_IM_noise0_9.bpmn
```

[15]:
```
fitness                                                 0.954485
precision                                                    NaN
generalization                                               NaN
simplicity_builtin                                      0.516949
simplicity_size                                         0.149022
simplicity_connectivity                                 0.254167
places                                                        42
transitions                                                   80
arcs                                                         179
size                                                         301
algo                                             HeuristicsMiner
params                              {'dependency_threshold': 0.6}
bpmn_path                         models/candidate_HM_dep0_6.bpmn
simplicity_avg                                          0.306713
score                                                   0.586309
Name: 7, dtype: object
```

[9]:
```python
from pm4py.objects.conversion.bpmn import converter as bpmn_converter

candidates = ["candidate_IM_noise1_0.bpmn", "candidate_IM_noise0_7.bpmn",␣
 ↪"candidate_IM_noise0_6.bpmn", "candidate_IM_noise0_9.bpmn"]

bpmn_graph = pm4py.read_bpmn("models/candidate_IM_noise1_0.bpmn")

net, im, fm = bpmn_converter.apply(bpmn_graph)

try:
    # Newer API (pm4py >= 2.2)
    from pm4py.algo.evaluation import algorithm as eval_alg
```

```python
except ImportError:
    # Older API
    from pm4py.evaluation import algorithm as eval_alg

metrics = eval_alg.apply(elog, net, im, fm)
print(metrics)
```

```
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[01:09<00:00, 229.48it/s]
replaying log with TBR, completed traces :: 100%|      | 263907/263907
[01:18<00:00, 3375.17it/s]
```

```
{'fitness': {'perc_fit_traces': 0.0, 'average_trace_fitness':
0.8963245464929964, 'log_fitness': 0.9114499480830527,
'percentage_of_fitting_traces': 0.0}, 'precision': 0.43908894920407493,
'generalization': 0.5928135957316973, 'simplicity': 0.6494845360824743,
'metricsAverageWeight': 0.6482092572753247, 'fscore': 0.5926635667581387}
```

[11]:
```python
bpmn_graph_2 = pm4py.read_bpmn("models/candidate_IM_noise0_7.bpmn")
net, im, fm = bpmn_converter.apply(bpmn_graph_2)

metrics = eval_alg.apply(elog, net, im, fm)
print(metrics)
```

```
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[01:20<00:00, 197.78it/s]
replaying log with TBR, completed traces :: 100%|      | 263907/263907
[01:28<00:00, 2998.02it/s]
```

```
{'fitness': {'perc_fit_traces': 0.0, 'average_trace_fitness':
0.8923824778299337, 'log_fitness': 0.895747393382901,
'percentage_of_fitting_traces': 0.0}, 'precision': 0.3611684381172643,
'generalization': 0.776412573395656, 'simplicity': 0.6783216783216783,
'metricsAverageWeight': 0.6779125208043749, 'fscore': 0.5147770103740165}
```

[12]:
```python
bpmn_graph_3 = pm4py.read_bpmn("models/candidate_HM_dep0_6.bpmn")
net, im, fm = bpmn_converter.apply(bpmn_graph_3)

metrics = eval_alg.apply(elog, net, im, fm)
print(metrics)
```

```
replaying log with TBR, completed traces :: 100%|      | 15930/15930
[01:02<00:00, 253.14it/s]
replaying log with TBR, completed traces :: 100%|      | 263907/263907
[04:23<00:00, 1002.86it/s]
```

```
{'fitness': {'perc_fit_traces': 0.0, 'average_trace_fitness':
0.9522376960002054, 'log_fitness': 0.957832606143199,
'percentage_of_fitting_traces': 0.0}, 'precision': 0.6735288939449342,
'generalization': 0.9395885406134147, 'simplicity': 0.5210084033613445,
'metricsAverageWeight': 0.772989611015723, 'fscore': 0.7909073933216764}
```

[ ]:

[ ]:

[ ]: