# ENHANCEMENT OF THE PROCESS MINING VISUALIZATION TOOL:

## MERGE PROJECTS, ADD FILTERING, UI-ENHANCEMENTS AND GENETIC MINER IMPLEMENTATION

Student: David Kapfhammer

Supervisor: Dipl.-Ing. Dr.techn Marian Lux

# AGENDA

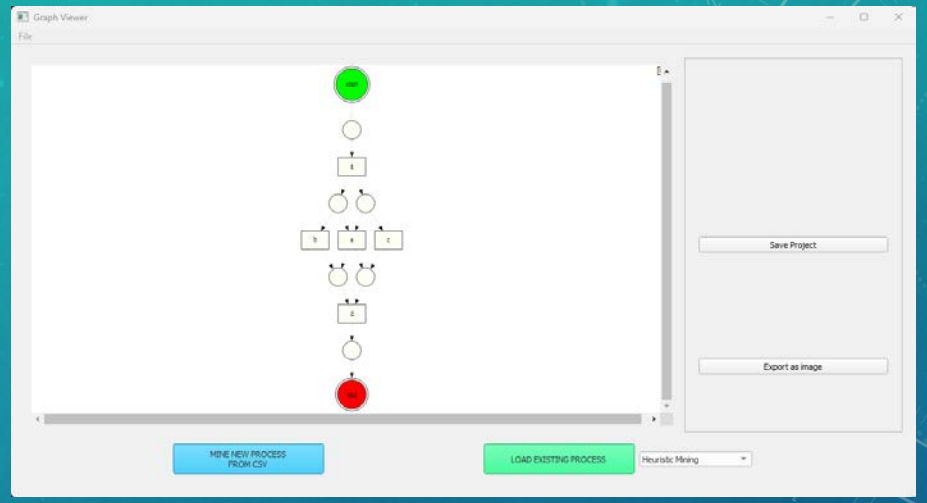
- Ausgangslage/Problemstellung
- Grundlagen
- Architektur
- Probleme Implementierung
- Evaluation & Testing
- Live Demo
- Herausforderungen
- Zukünftige Arbeit

# AUSGANGSLAGE / PROBLEMSTELLUNG

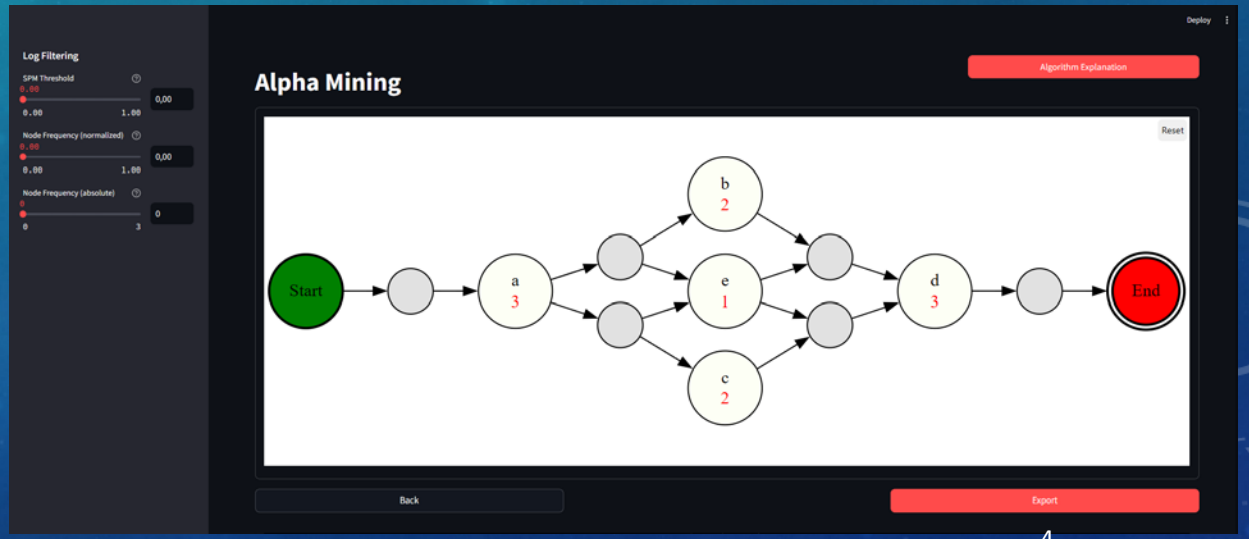
- **Motivation:** Process Mining für Transparenz & Optimierung von Geschäftsprozessen [1]
- **Bisherige Ansätze:** Alpha, Heuristic, Inductive, Fuzzy Miner [1, 2]
- **Limitationen:** empfindlich bei Noise, unvollständigen Logs, komplexen Strukturen [1]
- **Ziel:**
  - Zusammenführen der Vorgängerprojekte (Rustemi, Fraunberger) [3, 4]
  - Neue Filtermöglichkeiten & UI-Erweiterungen
  - Genetic Miner implementieren [5]
    - Robust bei Noise (globale Fitness-Berechnung, Crossover & Mutation)

# GRUNDLAGEN – MERGING

1. Alpha Miner & SPM Filter [6] in alter UI [4]
2. Inductive Miner & neue Streamlit-UI [3]
   - → Projekte gemerged
   - → Neue Streamlit UI übernommen
   - → Alpha Miner + SPM-Filter eingebaut



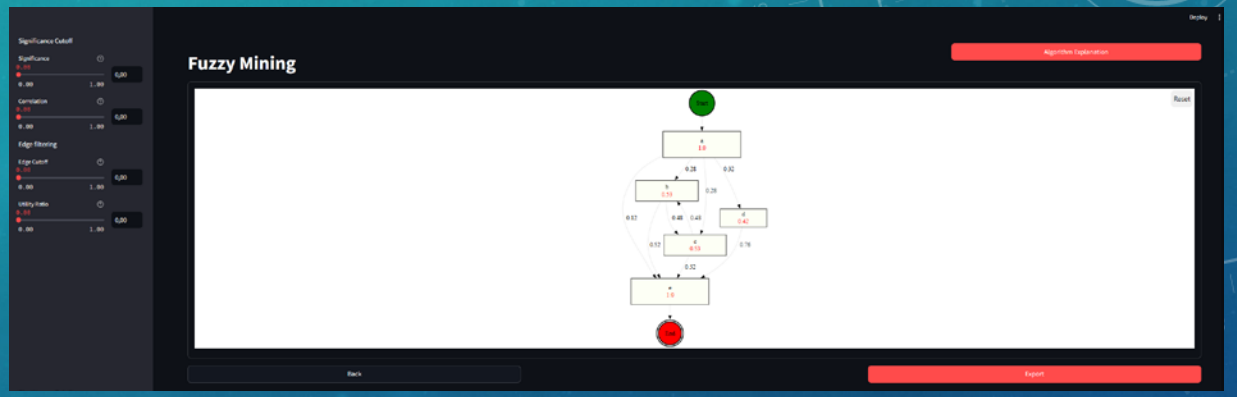

Alpha Miner in alter UI [4]



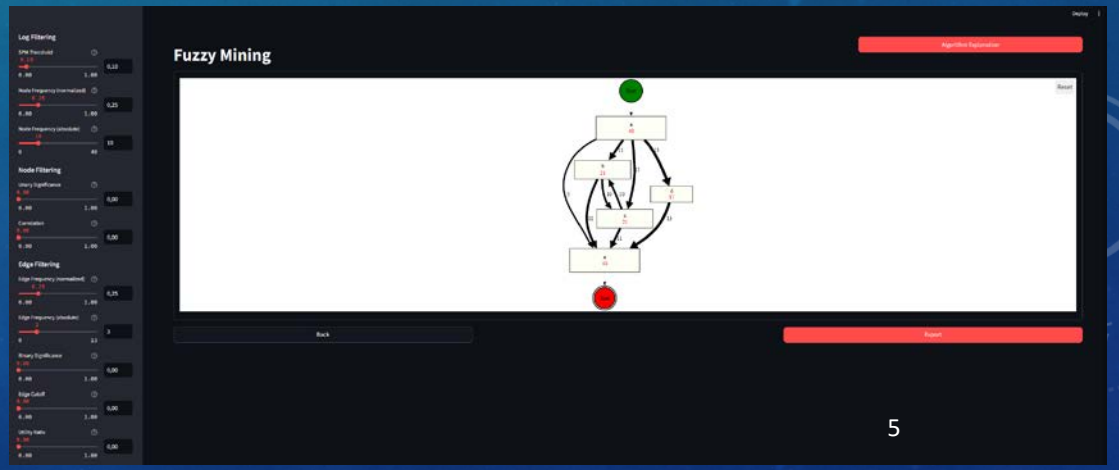

Alpha Miner in neuer UI

# GRUNDLAGEN – ERWEITERUNGEN (MAIN POINTS)

- **Filtermöglichkeiten:**
  - SPM-Filter (alle Algorithmen) [6]
  - Node-Filter (alle Algorithmen)
  - Edge-Filter (Heuristic & Fuzzy Miner)
- **UI-Verbesserungen:**
  - Absolute & Normalized Slider → synchronisiert
  - Edge-Click-Handler hinzugefügt
  - Edge Clustering: Bugfix (Heuristic), Integration (Fuzzy)
- **Fuzzy Miner:**
  - Significance getrennt: Unary (Nodes) & Binary (Edges)



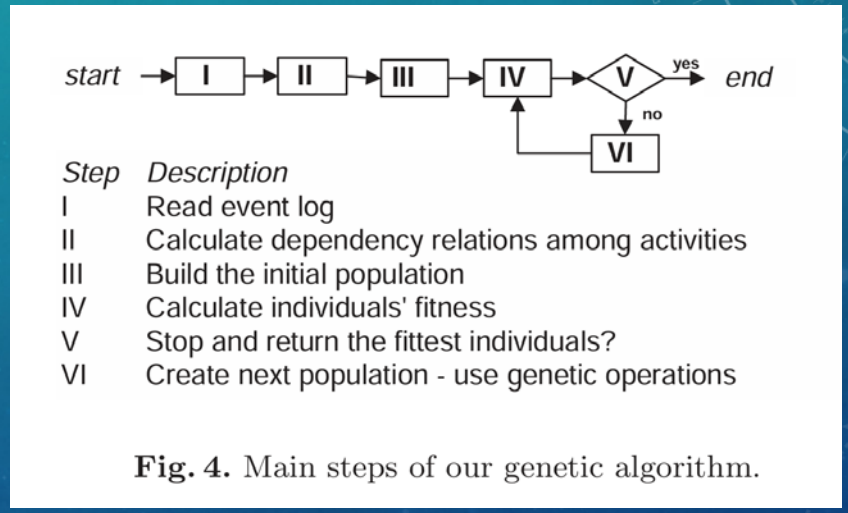

Fuzzy Miner alte Version [3]




Fuzzy Miner neue Version

# GRUNDLAGEN – GENETIC MINER [5] ABLAUF

1. Event Log einlesen
2. Dependency-Matrix berechnen
3. Initialpopulation erstellen
4. Fitness berechnen
5. Termination prüfen
6. Evolution (falls nicht beendet)




Main Steps [5]

**Ausgabe**: Bestes Individuum wird als **Petri-Netz Graph** visualisiert

# GRUNDLAGEN – GENETIC MINER [5] KONZEPT & INDIVIDUAL

- **Genetic Miner =** Anpassung des Genetic Algorithmus auf Process Mining
- **Vorteil:** Robuster bei Noise & unvollständigen Logs
- **Ziel:** Prozessmodell (Petri-Netz) aus Event Logs ableiten
- **Individual =**
  - Aktivitäten A
  - Input-Sets I(a) je Aktivität a
  - Output-Sets O(a) je Aktivität a
  - Kausalrelationen C = {(a, b)| b in O(a)}

# GRUNDLAGEN – GENETIC MINER [5] INITIALISIERUNG

- Dependency Matrix: **D(a, b)**
- Counts aus Event Log:
  - **follows:** f(a, b) = #{Traces mit a -> b}
  - **L1L:** L1L(a)= #{a -> a} (Self-Loops)
  - **L2L:** L2L(a, b)= #{a -> b -> a} (Length-2 Loops)
- **Start/End Measures**
  - S(t) = D(start, t)
  - E(t) = D(t, end)
    - Je größer, desto wahrscheinlicher beginnt/endet ein Trace mit Aktivität t

**Definition 3.1. (Dependency Measure)** Let $t_1$ and $t_2$ be two activities in event log $T$. Then:

$$D(t_1,t_2) = \begin{cases} \frac{L2L(t_1,t_2) + L2L(t_2,t_1)}{L2L(t_1,t_2) + L2L(t_2,t_1) + 1} & \text{if } t_1 \neq t_2 \text{ and } L2L(t_1,t_2) > 0 \\ \frac{follows(t_1,t_2) - follows(t_2,t_1)}{follows(t_1,t_2) + follows(t_2,t_1) + 1} & \text{if } t_1 \neq t_2 \text{ and } L2L(t_1,t_2) = 0 \\ \frac{L1L(t_1,t_2)}{L1L(t_1,t_2) + 1} & \text{if } t_1 = t_2 \end{cases}$$

Dependency Measure [5]

# GRUNDLAGEN – GENETIC MINER [5] HEURISTIC INDIVIDUAL

- Aus **D(a, b) kausale Matrix** erzeugt
- Steuerung durch power_value (p) → bestimmt Kantendichte/Seltenheit
- **Steps**:
  - Zufällige Kausal-Matrix basierend auf $D^p$
  - Wipes mit Start-/End-Measures
  - Partitionierung der Inputs/Outputs
  - Ableitung von I (INPUT), O (OUTPUT) & C (CAUSALITY)

# GRUNDLAGEN – GENETIC MINER [5] FITNESS FUNCTION (CONTINUOUS)

- **Ziel:** Modell muss Log so gut wie möglich durchlaufen
- **Bewertet durch Parse-Token-Game:**
  - Anteil korrekt geparster Aktivitäten (lokale Genauigkeit)
  - Anteil vollständig abgeschlossener Traces (globale Korrektheit)
- **Fitness$_C$ Formel:**
  - Wertebereich: $0.0 \leq F \leq 1.0$
  - Perfektes Modell: $F = 1.0$

**Definition 3.3.** **(Fitness$_C$)** Let $L$ be an event log and $PM$ be a process model. Then:

$$Fitness_C(PM, L) =$$

$$0.40 \times \frac{allParsedActivities_C(PM,L)}{numActivitiesLog(L)} + 0.60 \times \frac{allProperlyCompletedLogTraces_C(PM,L)}{numTracesLog(L)}$$

Fitness$_C$ [5]

# GRUNDLAGEN – GENETIC MINER [5] EVOLUTION

- **Standard-Operatoren des GA, angepasst für Process Mining:**
  - **Selektion (Tournament Selection)**
    - Zufällige Stichprobe von k Individuen
    - Bestes Individuum wird gewählt
  - **Crossover**
    - Wähle zufällige Aktivität t
    - Tausche Input-/Output-Sets I(t), O(t) zwischen Eltern
    - Repariere Konsistenz: b in O(a) ⇔ a in I(b)
  - **Mutation**
    - Für jede Aktivität mit Wahrscheinlichkeit **mutation_rate**:
      - Input-/Output-Sets verändern
      - Veränderung = **Merge** (Sets zusammenführen) oder **Split** (Set aufteilen)
  - **Elitismus**
    - Top Elitismus % der Individuals werden unverändert in nächste Population übernommen

# GRUNDLAGEN – GENETIC MINER [5] TERMINATION

- **Algorithmus stoppt wenn:**
  - Fitness-Threshold erreicht
  - Maximale Generationen durchlaufen
  - Fitness-Stagnation (keine Verbesserung über n/2 Generationen)

# GRUNDLAGEN:
# DEFAULT GA VS. GENETIC MINER (VEREINFACHT)

```
***** Default GA *****

INPUT:
    TARGET string
    POPULATION_SIZE
    GENES (alphabet)

INITIALIZATION:
    population = zufällige Strings (Länge = TARGET)

EVOLUTION LOOP:
    WHILE bestes Individuum hat Fitness > 0:
        1. BERECHNE Fitness jedes Individuums
            Fitness = Anzahl falscher Zeichen
        2. SORTIERE Population nach Fitness (beste zuerst)
        3. ELITISM: Kopiere die besten 10 % unverändert
        4. REPRODUKTION:
            - Wähle Eltern zufällig aus den besten 50 %
            - Crossover: Gene aus Parent1 / Parent2 übernehmen
            - Mutation: mit 10%iger Wahrscheinlichkeit zufälliges Zeichen ersetzen
        5. Population = neue Generation

OUTPUT:
    bestes Individuum (exakter Zielstring)
```

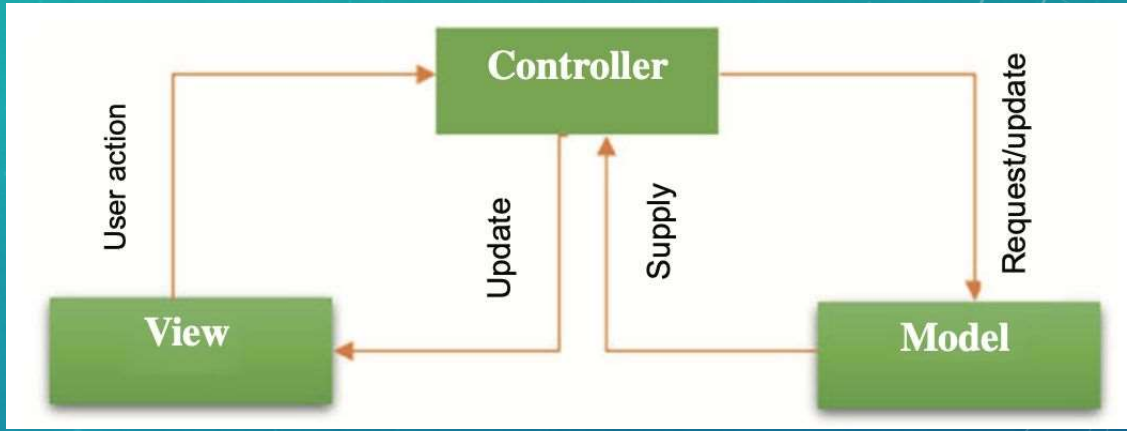
Genetic Algorithm [9]

```
***** Genetic Mining *****

INPUT:
    EVENT LOG (Traces + Frequenzen)
    GA-Parameter: population_size, max_generations, crossover_rate, mutation_rate,
                  elitism_rate, tournament_size, power_value, fitness_threshold

INITIALIZATION:
    1. BERECHNE Dependency-Matrix (follows, L1L, L2L)
    2. BERECHNE Start- und End-Measure
    3. population = heuristisch erzeugte Prozessmodelle
       - Repräsentation: Individual: Aktivitäten, INPUT-Sets, OUTPUT-Sets, Kausalrelationen
       - power_value steuert die Dichte/Seltenheit der initialen Kanten

EVOLUTION LOOP:
    WHILE (beste Fitness < fitness_threshold) UND (Generation < max_generations)
UND (keine Stagnation erreicht):
        1. BERECHNE Fitness jedes Individuums
            Fitness = 0.40 * (korrekt geparste Events / total Events)
                    + 0.60 * (vollständig beendete Traces / total Traces)
        2. SORTIERE Population nach Fitness (beste zuerst)
        3. ELITISM: Kopiere die besten (elitism_rate * population_size) unverändert
        4. REPRODUKTION:
            - Elternwahl per Tournament Selection (tournament_size)
            - Crossover: tausche I/O-Sets einer Aktivität zwischen Eltern
            - Mutation: split/merge I/O-Sets mit Wahrscheinlichkeit mutation_rate
            - Konsistenzprüfung: b in O(a) <=> a in I(b)
        5. Population = neue Generation

OUTPUT:
    bestes Prozessmodell (Individual)
    -> übersetzt in Graph
```

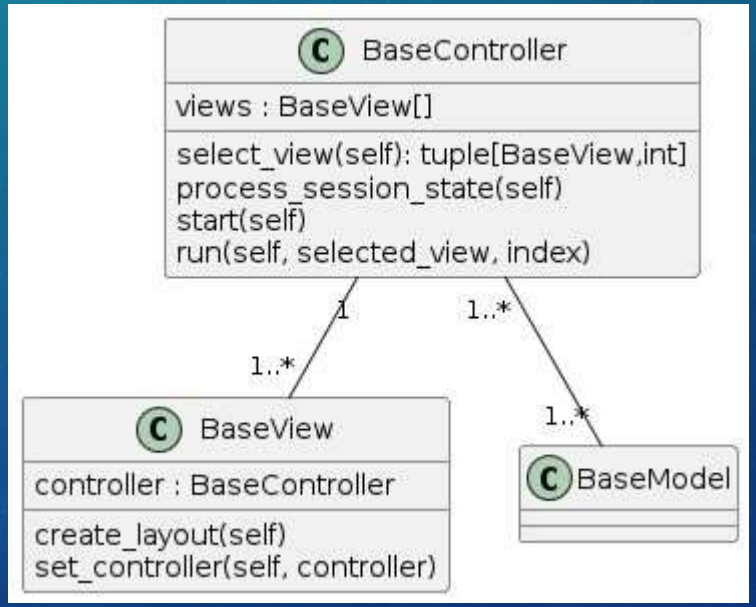
# ARCHITEKTUR

**MVC als Design Pattern:**

- **Model** – genetic_mining.py
- **View** – genetic_miner_view.py
- **Controller** – genetic_miner_controller.py



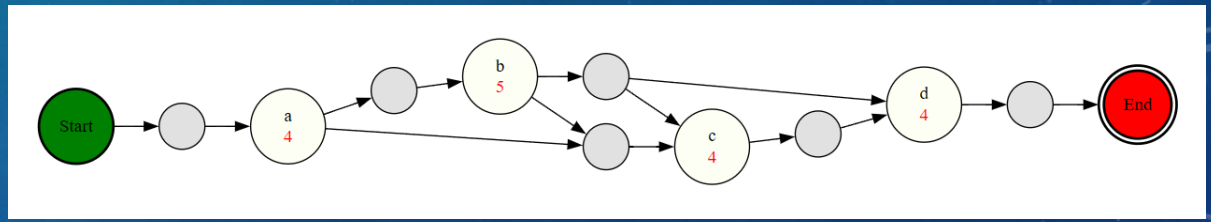

MVC-Pattern [10]




MVC Pattern UML Klassen Diagramm

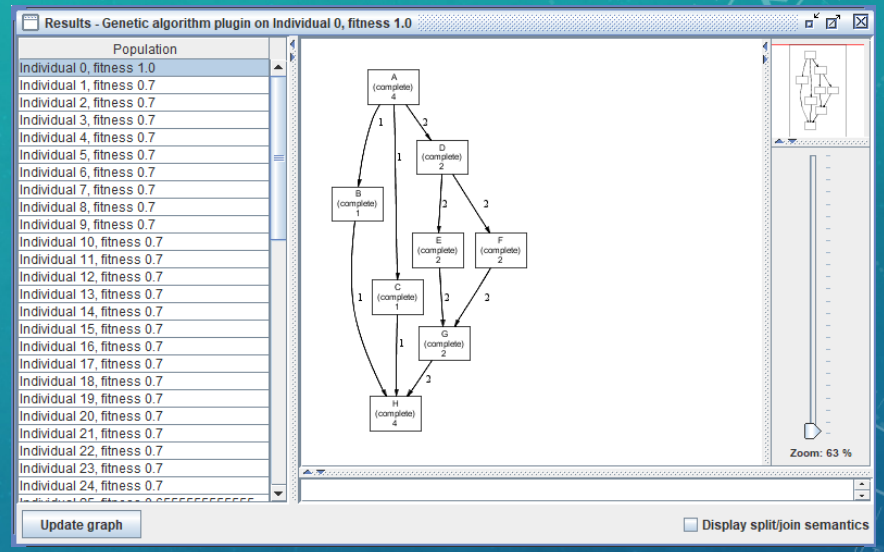
# PROBLEME IMPLEMENTIERUNG

- „Token-Game“ teilweise unvollständig (v. a. Completion-Logik)
- Silent-Transitions fehlen (Edge Cases bei Graph-Erzeugung)

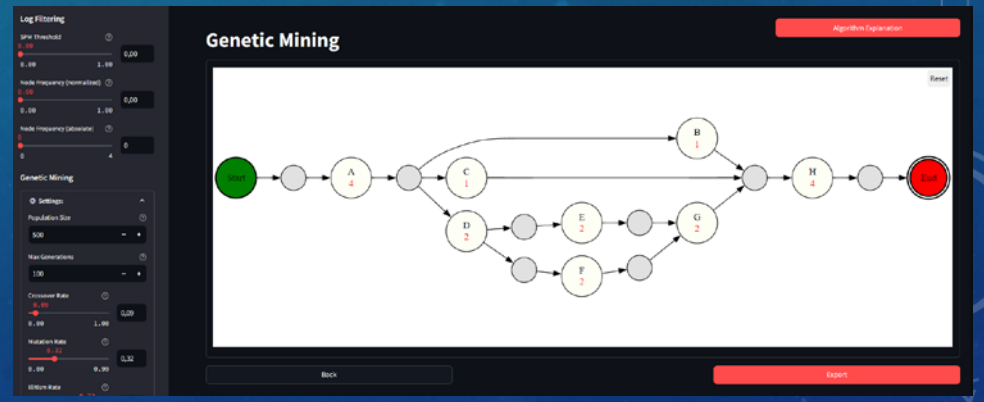


# EVALUATION & TESTING

- Neue Features getestet:
  - Node/Edge/SPM Filter
  - Synchronisierte Slider
  - Edge Clustering (Fuzzy + Heuristic Miner)
- Unit Tests für Genetic Miner (Parsing, Fitness, Operatoren, etc.)
- Outcome Genetic Miner = ProM-Ergebnis (bei Testlog)




Genetic Miner ProM 5.2 [7]




Genetic Miner Process Mining Visualization Tool

# LIVE DEMO

# HERAUSFORDERUNGEN

- Mein erstes größeres Python-Projekt
- Streamlit Limitierungen (v. a. bei Slider Synchronisation)
- Parsing-Semantic (Token-Game) [5]
- Code-Komplexität & Übersichtlichkeit

# ZUKÜNFTIGE ARBEIT

- Genetic Miner:
  - Token-Game completion fix [5]
  - Silent-Transitions integrieren [5]
  - Verbesserte Fitness (PFcomplete & PFprecise) [8]
- Bessere Visualisierung für Edge Clustering: Edge-Färbung (grün → rot je nach Häufigkeit)
- Weitere Mining-Algorithmen hinzufügen

# LITERATUR


- [1] van der Aalst, W. M. P. Process Mining Discovery, Conformance and Enhancement of Business Processes. Springer Verlag Berlin Heidelberg, 2011.
- [2] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. Discovering block-structured process models from event logs - a constructive approach. In Application and Theory of Petri Nets and Concurrency, J.-M. Colom and J. Desel, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 311–329.
- [3] Fraunberger, F. Enhancing the Process Mining Visualization Tool by adding the Inductive Miner and improving the UI. Bachelor thesis, University of Vienna, 2024.
- [4] Rustemi, A. Process Mining Visualization Tool in Python. Bachelor thesis, University of Vienna, 2024.
- [5] Medeiros, A & Weijters, A. & Aalst, Wil. (2005). Using genetic algorithms to mine process models: representation, operators and results. Systems Research and Behavioral Science.
- [6] Lux, M., Rinderle-Ma, S., & Preda, A. (2018). Assessing the quality of search process models. In Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16 (pp. 445-461). Springer International Publishing.
- [7] ProMTools. Prom tools. https://promtools.org/, 2025. Accessed on 2025-09-20.
- [8] Medeiros, A. & Weijters, A. & Aalst, Wil. (2007). Genetic process mining: An experimental evaluation. Data Mining and Knowledge Discovery. 14. 245-304. 10.1007/s10618-006-0061-7.
- [9] GeeksforGeeks. Genetic Algorithms. https://www.geeksforgeeks.org/dsa/genetic-algorithms/, 2025. Accessed on 2025-09-20.
- [10] Rana, M. E., and Saleh, O. S. Chapter 15 - high assurance software architecture and design. In System Assurances, P. Johri, A. Anand, J. Vain, J. Singh, and M. Quasim, Eds., Emerging Methodologies and Applications in Modelling. Academic Press, 2022, pp. 271–285.

# VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!