1 Social Network Mining from Event Logs

Van der Aalst, Reijers, and Song [9] present a foundational approach for discovering social networks from event logs. This work presents a collection of algorithms to extract system relationships by analyzing performer information in event logs, augmenting the control-flow graphs produced from traditional process-mining. The authors define four categories of metrics for constructing sociograms.

Metrics based on causality track how work flows between performers. The *handover of work* metric measures how often one performer's activity is directly followed by another's for the same case. The *subcontracting* metric identifies when work is delegated and returned. **Metrics based on joint cases** measure how frequently two individuals perform activities for the same process instance, indicating collaborative relationships. **Metrics based on joint activities** compare performer "profiles" based on which activities they execute, using distance measures like Hamming distance and Pearson's correlation coefficient to identify similar roles. **Metrics based on special event types** analyze events like reassignment or delegation to reveal hierarchical re-

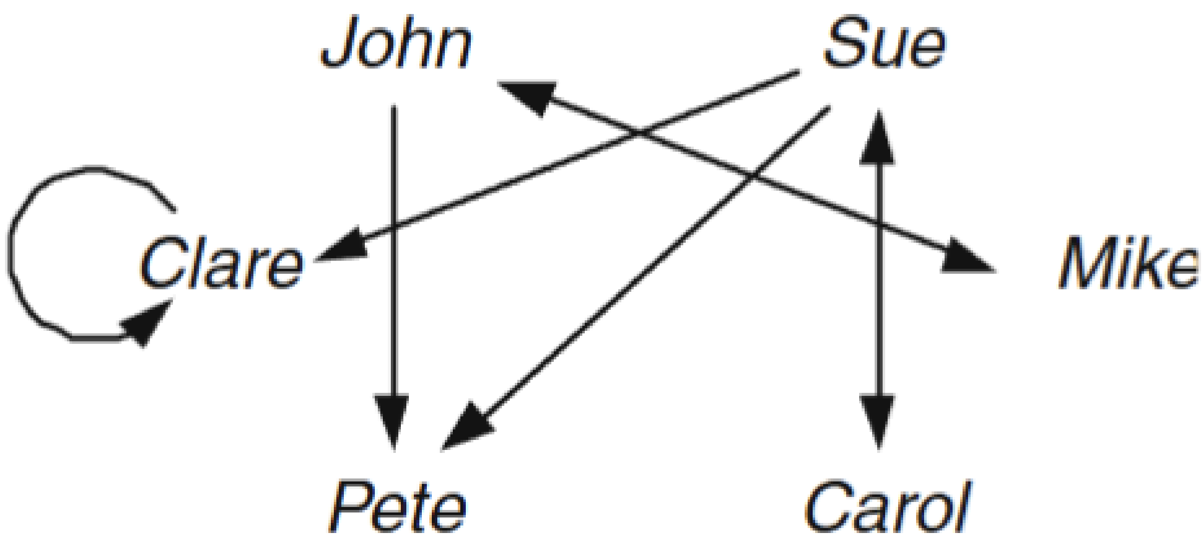

Figure 1: An Example Social Network [9]

lationships.

2 Event Detection from Time Series Data

Guralnik and Srivastava [7] presents an approach to detect events from raw sensor telemetry. This work focuses on determining when significant behavioral changes occur in a system using two key methodologies. **Unknown number of change-points**: Rather than requiring the number of change-points to be specified a priori, the algorithm iteratively discovers them using a likelihood-based stopping criterion. **Flexible model selection**: Instead of assuming a fixed functional form between change-points, the approach uses universal approximation theory with basis functions (e.g., polynomials) and selects the best model via cross-validation.

The core algorithm uses maximum likelihood estimation (MLE) to evaluate candidate change-points. For a time series $y(t)$, the method fits piecewise models $f_i(t, \mathbf{w}_i)$ to segments and minimizes the likelihood criterion $\mathcal{L} = \sum_{i=1}^{k} m_i \log \sigma_i^2$ for heteroscedastic errors. A hierarchical splitting procedure examines each segment to determine if partitioning improves the overall likelihood significantly.

Two algorithm variants are presented:

- **Batch algorithm**: Processes complete datasets, achieving global optimization. Uses a stability threshold s to stop when likelihood improvements become negligible.

- **Incremental algorithm**: Detects change-points in real-time as data arrives, useful for online monitoring applications like highway traffic control. Uses a likelihood increase threshold δ to confirm detected changes.

3 Fault Diagnosis Using Digraph-Based Fault Trees

James, Gandhi, and Deshmukh [5] demonstrate the usage of a digraph-based fault tree for analysis of automobile systems. The researchers break the automobile system down into a hierarchical structure of subsystems, assemblies, and components. The researchers map the critical parameters of Pressure (P), Mass flow rate (M), and Temperature (T) to each system component as input and output streams. An *interrelationship table* is constructed to quantify how input paramters affect output paramters using a gain value. The individual component digraphs are then combined into a complete system digraph.

The methodology identifies events at multiple levels. **Top events** represent observable failure symptoms (loss of power, inefficient operation, hard steering, steering wheel vibration, and noise for the hydraulic system). **Intermediate events** capture parameter deviations propagating through the system, such as elevated oil temperature. **Basic events** terminate each fault tree branch and include component failures (vane pump failure, steering cylinder leakage, filter clogged), sub-failures (cavitation erosion, vane tip damage, seal failure), and maintenance-related causes (lack of maintenance, corrosion).

4 PM4Py: A Process Mining Library for Python

Berti, van Zelst, and van der Aalst [1] introduced PM4Py, an open-source Python library designed to bridge the gap between process mining and data science. The library addresses three central disciplines: **Process discovery**: Automatically constructing process models from event logs using algorithms such as the alpha miner, inductive mining, heuristics miner, ILP miner, and correlation miner. **Conformance checking**: Comparing observed behavior against modeled behavior using techniques including footprints, token-based replay, alignments, and log skeleton methods. **Process enhancement**: Improving existing models based on observed data, including performance analysis and decision mining.

PM4Py supports multiple standard data formats for interoperability: the XES format for traditional event logs, OCEL for object-centric event logs, PNML for Petri nets, PTML for process trees, and BPMN 2.0 for business process models. The library provides diverse visualizations including directly-follows graphs (DFGs), process trees, accepting Petri nets, and BPMN diagrams, as well as analytical views such as dotted charts and performance spectra for temporal pattern detection.

Beyond process-focused analysis, PM4Py supports organizational mining through social network analysis (handover of work, working together, subcontracting metrics), role discovery, and resource profiling—connecting directly to the sociogram extraction work of van der Aalst et al. [9].

5 Workflow Mining: The α-Algorithm

Van der Aalst, Weijters, and Maruster [8] present a foundational algorithm for discovering process models from event logs. The work addresses the *rediscovery problem*: given a workflow log containing sequences of tasks executed for each case, can we reconstruct the underlying process model? The authors represent discovered processes as Petri nets, specifically *Workflow nets* (WF-nets) with designated source and sink places.

The approach relies on four **log-based ordering relations** derived from observed task sequences. The *direct follows* relation ($a >_W b$) indicates task

b immediately followed task a in some trace. The *causal* relation ($a \rightarrow_W b$) holds when $a >_W b$ but not $b >_W a$, suggesting a dependency. The *parallel* relation ($a \|_W b$) holds when tasks appear in both orders, indicating concurrent execution. The *never follow* relation ($a \#_W b$) captures tasks that never directly succeed each other.

The α-algorithm constructs a WF-net by: (1) identifying all tasks from the log, (2) finding initial and final tasks, (3) discovering causal relationships to create places connecting causally related tasks, and (4) merging places appropriately to distinguish AND-splits/joins from OR-splits/joins based on the $\#_W$ relation.

The authors prove that the α-algorithm can *rediscover* any process belonging to the class of **Structured Workflow nets (SWF-nets)**—sound WF-nets that are free-choice (no mixing of choice and synchronization) and contain no implicit places—provided the log is complete and the process contains no **short loops** (loops of length one or two). Short loops confound the algorithm because they make the causal relation symmetric, obscuring the underlying structure.

The work acknowledges limitations: the algorithm cannot detect hidden (invisible) tasks, cannot distinguish duplicate tasks with identical labels, and requires a notion of log completeness where all directly-follows relationships have been observed. The authors validated their approach on real-world data from healthcare (patient flow analysis) and judicial processes (processing of 130,136 fine cases).

6 In-Vehicle Telematics for Driving Behavior Monitoring

Boylan, Meyer, and Chen [4] present a systematic review of 45 studies examining how in-vehicle telematics data has been used to monitor and model driving behaviors. The most commonly analyzed telematics variables were speed, acceleration, braking, distance/driving time, and turning/cornering.

Machine learning dominated the analysis approaches (22 studies), particularly for insurance applications. Neural networks, random forests, support vector machines, and XGBoost were used to create predictive models for crash risk and driver classification. For claim frequency prediction, Poisson regression models enhanced with neural network-derived risk scores significantly outperformed classical insurance models. Logistic regression, while slightly less accurate than complex machine learning models, was recommended for crash risk prediction due to its interpretability.

7 Fault Tree Analysis for Automotive Reliability and Safety

Lambert [6] presents the application of fault tree analysis (FTA) to automotive systems, advocating for its adoption alongside the failure modes and effects analysis (FMEA) traditionally used in the automotive industry. While FMEA is an *inductive* technique asking "what if?" to determine the effects of component failures, FTA is a *deductive* technique asking "how can something occur?" by working backward from an undesired top event to identify all possible causes.

FTA originated in the aerospace industry in the 1960s for analyzing ballistic missile systems and has since become essential for probabilistic risk assessment in nuclear power and chemical processing. A fault tree is a graphical and logical representation of event combinations using standard AND and OR gates. Basic events at the bottom of the tree represent the limit of resolution: component failures, human errors, software failures, and environmental conditions.

The paper distinguishes two types of fault events:

- **Type I**: A system element fails to perform an intended function (e.g., "air bag fails to deploy when collision occurs")
- **Type II**: A system element performs an inadvertent function (e.g., "inadvertent deployment of air bag")

The FTA methodology follows a structured sequence:

1. Define the undesired event (top event)
2. Acquire system understanding through diagrams and flow sheets
3. Establish scope, bounds, and environmental conditions
4. List assumptions
5. Construct the fault tree using logic gates

6. Perform qualitative analysis: identify single-point failures, common-cause failures, and minimal cut sets
7. Perform quantitative analysis: calculate top event probability, importance measures, and uncertainty
8. Conduct tradeoff studies and document results

Lambert emphasizes that FTA's value lies in showing logical progressions and system interactions not displayed in traditional FMEA, while FMEA can identify candidate undesired events for FTA. With increasing automotive electronics in engine control, cruise control, and braking systems, FTA is particularly well-suited to address complex failure mode interactions—complementing the digraph-based approach described by James et al. [5].

8 The SHIP Safety Case Approach

Bishop and Bloomfield [3] present a formalized safety case approach developed under the EU SHIP project (Safety of Hazardous Industrial Plants). A key contribution is the **system failure behavior model**, which defines five system states. **Perfect**: The system contains no faults. **OK**: The system contains faults but operates correctly because no triggering input has been encountered. **Erroneous**: A fault has been activated, causing computed values to deviate from design intent. **Safe**: The system has failed but in a non-hazardous manner. **Dangerous**: The system has failed in a hazardous manner.

The model identifies six state transitions, each governed by distinct factors. **Perfect → OK** (fault introduced): Probability depends on development process quality, maintenance procedures, and system documentation. **OK → Perfect** (fault removal): Enabled by testing, bug reporting and correction, and accumulated field usage that reveals latent faults. **OK → Erroneous** (error activation): Probability depends on the number of faults, their size and distribution within the code, and the operational profile (input distribution). **Erroneous → OK** (error correction): Achieved through fault-tolerant design features, application characteristics such as "grace time," or natural self-healing in subsequent computations. **Erroneous → Safe** (safe failure): Enabled by fail-safe design that detects deviations and overrides with safe alternatives. **Erroneous → Dangerous** (dangerous failure): Occurs when failure containment mechanisms are absent or insufficient for the hazards present.

This model enables three main safety argument strategies: (1) *fault elimination* arguments that maximize the probability of remaining in the "perfect" state, (2) *failure containment* arguments that strengthen recovery transitions back to OK or to the safe state, and (3) *failure rate estimation* arguments based on black-box testing or field experience that directly estimate the OK → dangerous transition probability.

References

[1] A. Berti, S.J. van Zelst, and W.M.P. van der Aalst, "Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science," in *Proceedings of the 1st International Conference on Process Mining (ICPM 2019), Demo Track*, pp. 13–16, 2019.

[2] A. Berti, S. van Zelst, and D. Schuster, "PM4Py: A Process Mining Library for Python," *Software Impacts*, vol. 17, p. 100556, 2023.

[3] P.G. Bishop and R.E. Bloomfield, "The SHIP Safety Case Approach," in *Proceedings of SafeComp95*, Belgirate, Italy, pp. 437–451, 1995.

[4] J. Boylan, D. Meyer, and W.S. Chen, "A Systematic Review of the Use of In-Vehicle Telematics in Monitoring Driving Behaviours," *Accident Analysis and Prevention*, vol. 199, p. 107519, 2024.

[5] A.T. James, O.P. Gandhi, and S.G. Deshmukh, "Fault Diagnosis of Automobile Systems Using Fault Tree Based on Digraph Modeling," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 494–504, 2017.

[6] H.E. Lambert, "Use of Fault Tree Analysis for Automotive Reliability and Safety Analysis," in *Proceedings of the 2004 SAE World Congress*, Detroit, Michigan, 2004.

[7] V. Guralnik and J. Srivastava, "Event Detection from Time Series Data,"

in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*, pp. 33–42, 1999.

[8] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[9] W.M.P. van der Aalst, H.A. Reijers, and M. Song, "Discovering Social Networks from Event Logs," *Computer Supported Cooperative Work*, vol. 14, pp. 549–593, 2005.