Alarm Processing with Model-Based Diagnosis of Event Discrete Systems

Andreas Bauer, Adi Botea, Alban Grastien, Patrik Haslum, Jussi Rintanen
NICTA and the Australian National University
Canberra, Australia

Abstract

Reliable and informative alarm processing is important for improving the situational awareness of operators of electricity networks and other complex systems. Earlier approaches to alarm processing have been predominantly syntactic, based on text-level filtering of alarm sequences or shallow models of the monitored system. We argue that a deep understanding of the current state of the system being monitored is a prerequisite for more advanced forms of alarm processing.

We use a model-based approach to infer the (unobservable) events behind alarms and to determine causal connections between events and alarms. Based on this information, we propose implementations of several forms of alarm processing functionalities. We demonstrate and evaluate the resulting framework with data from an Australian transmission network operator.

1 Introduction

Many views grouped under the broad Smart Grid umbrella agree that the power networks of the future will produce very large volumes of data. Compared to current-day networks, one source of additional data will be the deployment of more sensors, including new types of sensors such as smart meters. Furthermore, future networks will change their configuration more dynamically, to work with wind turbines and other variable-output sources, and to handle overload, voltage, and phase imbalance issues caused by charging electric cars [Ipakchi and Albuyeh, 2009]. This dynamic nature of networks will further increase the volume of data generated.

An alarm is a message that signals a discrete event in the network, such as automatic protection equipment triggering or an analog sensor measurement crossing a pre-defined threshold value. As an example, Figure 1 shows a small extract from an alarm log generated by a transmission network. The purpose of alarms is to alert operators to changes, signifying potential error conditions. However, actual faults frequently give rise to "alarm cascades", where the original fault causes a range of secondary abnormalities, all of which generate multiple alarms, thus quickly overwhelming operators' attention.

```
00:00:00   CB 1B A-B    –OPEN–
00:00:00   CB 2B A-B    –OPEN–
00:00:00   CB 2A A-B    –OPEN–
00:00:00   CB 1A A-B    –OPEN–
00:00:01   Line A-B     KV LIMIT LOW
00:00:04   Line C-D     KV LIMIT NORMAL
00:00:15   CB 1A A-B    –CLOSED–
00:00:17   Line A-B     KV LIMIT NORMAL
00:00:17   CB 1B A-B    –CLOSED–
00:00:17   CB 2B A-B    –CLOSED–
00:00:20   CB 2A A-B    –CLOSED–
00:00:20   Line C-D     KV LIMIT HIGH
```

Figure 1: Excerpt from an alarm log, spanning 20 seconds. (Names have been simplified.)

This problem is widely recognised, and techniques that aim to deal with it are collectively known as "intelligent alarm processing". Kirschen and Wollenberg [1992] identify three functions that intelligent alarm processors should provide: reduce the number of alarms, provide a clearer idea of the cause of the alarms, and recommend corrective actions to the operator. Adopting a similar view, McDonald *et al.* [1991] argue that the role of alarm processing is to transform raw alarm data into a format that is more digestible to a human operator. The increasing availability of networked sensors, and the increasing dynamicity of the future power grid, will only further emphasize the need for intelligent alarm processing techniques to assist human operators in making sense of the data that systems provide them with.

In this paper we present a model-based approach to analysing the evolution of systems like electricity distribution and transmission networks. We use a discrete-event model that describes the possible behavior of the network, at an abstract level. An online diagnosis process infers comprehensive information about the (recent) evolution and the current state of the network. This information is the basis for multiple alarm processing functions, each of which provides a different kind of meaningful summary that can be quickly grasped by operators. For example, in the log excerpt in figure 1, the group of alarms relating to the "A–B" line can all be explained by one single cause (a transient ground fault on the line), and clearly separated from the independent events (voltage fluctuations) taking place at the same time on the "C–D" line.

The diagnosis engine takes as input a log of alarms, the model of the network, and the state of the network (e.g., closed/open status of switches) at the beginning of the interval to analyse. Taking the system state into account is essential, because the causal links between events often depend on the current status of network components. For instance, where an electrical fault in one part of the network will propagate, and thus which alarms it will lead to, depends on the status of switches. Our model also includes time constraints between events. Few earlier alarm processing techniques do this, notable examples being chronicles [Laborie and Krivine, 1997a; 1997b; Taisne, 2006; Cordier *et al.*, 1998] and temporal constraint networks [Guo *et al.*, 2010]. The result of the diagnosis process is a *scenario*, which is an evolution of the system (i.e., series of states and transitions) that is consistent with the observed alarms. There may be several scenarios that are compatible with the observations, and preference is given to those that contain the fewest unexplained and unlikely events.

The purpose of alarm processing is to extract those pieces of information that are most relevant to the control room operators in a given situation. This can take the form of compact summaries of long event sequences, or highlighting the most important events that the operator should be aware of. The scenarios themselves contain far too much detail to be presented to operators in raw form. Different filtering methods are applied to the generated scenarios, each extracting a different type of summary to be presented to the operator. In this paper, we present four different techniques, namely alarm clustering, root cause analysis, fault-independent alarm highlighting and highlighting of live alarms (cf. Section 4). However, the result of the diagnosis process can be put to more uses than these specific alarm processing functions. It provides a powerful basis for implementing further functions useful in a control room, such as computing plans for corrective action.

We have developed a prototype system, which we tested on a real alarm log, provided by TransGrid, the transmission network operator in New South Wales and the Australian Capital Territory, Australia. In spite of being only a prototype, the diagnosis engine and alarm processing functions are fast enough to process all but the largest alarm cascades in less than a minute. The network model that we developed is also a highly simplified prototype; in spite of this, the system generates meaningful summaries in many instances.

2 Related Work

The literature contains a wide variety of techniques applied to alarm processing and fault diagnosis, including neural networks [Lin *et al.*, 2004], expert systems [Protopapas *et al.*, 1991; McDonald *et al.*, 1991], fuzzy logic [Meza *et al.*, 2001], tabu search [Wen and Chang, 1997], genetic algorithms [Wen *et al.*, 1995], chronicle-based approaches [Taisne, 2006; Cordier *et al.*, 1998], and temporal constraint networks [Guo *et al.*, 2010].

Model-based alarm processing is a recurring theme in the literature. Lin *et al.* [1998] use a power flow relation model. Dahlgren *et al.* [1998] view causal relations between types of alarms as model-based knowledge. A similar approach is followed by Larsson and DeBor [2007; 2002]. More recently, Guo *et al.* [2010] have used temporal constraint networks (TCNs) as a model. This permits specifying temporal constraints on the occurrence of individual events, as well as between cause-effect pairs of events. However, this model is still limited to defining such causal relations statically, i.e., whether or not one event causes another does not depend on the state of the network. This precludes specifying many essential aspects of the behaviour of the network. For example, a circuit breaker opening may cause automatic shutdown of a generator, *if* the opening of that breaker isolated the generator from a significant part of its load; but whether or not this is the case clearly depends on the state of other switches.

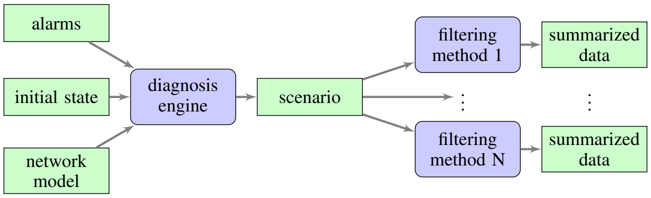

Figure 2: Architecture of the alarm processing system. The diagnosis engine creates a scenario (cf. Sections 3.2 and 3.3), which is the basis for multiple filtering methods. Each filter creates a different kind of summary view of the alarm log, which is presented to the operator. In Section 4, we describe four different filtering techniques.

A notable work in this context is that of Laborie and Krivine [1997a; 1997b]. They use a more detailed model, very similar to ours, containing the topology of the network together with an automata-based description of the behavior of each component type. The main difference between their approach and ours lies in how this model is used. They use the model for an off-line simulation of the system, which is used to pre-compute a collection of event patterns called chronicles. These chronicles are then compared on-line against the evolving alarm log, and the output of alarm processing is matching instances of chronicles appearing in the log.

Lin *et al.* [1998] present a combination of rule-based and model-based alarm processing. Rule-based reasoning performs message synthesis. A power flow relation model is used to find relations between alarms. The model contains the topology of the system (components and connections) together with information on how changes in the parameters of one component will affect the parameters of other, adjacent components. The program is tested on alarms corresponding to incidents at the Taipei Mass Rapid Transit System.

3 Overview of the Approach

The overall architecture of our alarm processing system is summarized in Figure 2. In this section we discuss briefly each of the main components and the way they work together.

3.1 Network Model

The model we use contains all the application-specific knowledge on which the inferences made by the diagnosis and

```
componenttype Generator {
  var status : { on, off };
  var status_changing : bool;
  var runback : bool;

  // output events (observable)
  observable event runback_alarm;
  observable event runback_reset;
  observable event unit_status_ON;
  observable event unit_status_OFF;

  // output events (for synchronization)
  event isolator_open;
  event isolator_close;

  // shutdown

  transition begin_shutdown
    status = on and status_changing = false
    -> status_changing := true;

  transition shutdown_breaker_open
    status = on and status_changing = true
    -> isolator_open;

  transition shutdown_turn_off
    status = on and status_changing = true
    -> unit_status_OFF;

  transition shutdown_complete
    status = on and status_changing = true
    -> status := off, status_changing := false;

  // startup

  transition begin_startup
    status = off and status_changing = false
    -> status_changing := true;

  transition startup_breaker_close
    status = off and status_changing = true
    -> isolator_close;

  transition startup_turn_on
    status = off and status_changing = true
    -> unit_status_ON;

  transition startup_complete
    status = off and status_changing = true
    -> status := on, status_changing := false;

  // runback

  transition runback_alarm_unexplained
    true -> runback := true, runback_alarm;

  transition runback_reset_unexplained
    true -> runback := false, runback_reset;

  // the runback alarm can reset when the generator is off

  transition runback_reset_when_off
    status = off -> runback := false, runback_reset;
}
```

Figure 3: MMLD model of the Generator component type. The model has been slightly simplified for readability. For example, time constraints have been left out.

alarm processing system are made. The advantage of the model-based approach is reusability: to adapt the software to changes in the network, or to deploy it in a new network, only the model needs to be updated. We designed a specification language, called MMLD, for describing the model. Here, we will not discuss the syntax of MMLD, but focus on the underlying concepts.

The model is a component-based, discrete-event dynamical system [Cassandras and Lafortune, 1999]. (It can be viewed as a factored description of a very large finite state machine, augmented with time constraints.) Power grids are very large, but consist of many instances of the same, relatively few, component types. We separate the model into descriptions of the behavior of component types, and the topology, which enumerates the actual components, of each type, and describes how they are interconnected in the grid. This separation makes the model specification more compact, and further improves reusability since component types are also generic, to some extent. An example of a component type model, slightly simplified, in MMLD syntax, is shown in Figure 3. The component type is a power generator.

The model of a component type describes its dynamical behaviors, both nominal and abnormal, at a relatively high level of abstraction. At this level of abstraction, a component (instance of the type) can be in one of a finite set of discrete states, and changes between states by discrete events. Each component type has a set of (local) transition rules, which define under what conditions it may, or must, change its (local) state. Causal relations between events are enforced by the transition rules preconditions and effects on the component state, and by time constraints: a minimum time constraint specifies that the transition must be enabled for at least a certain amount of time before it fires; a maximum time constraint specifies that the transition must take place if it is continuously enabled long enough. For example, fault current through a protection relay not only enables it to trigger circuit breakers, but forces it to do so within a set time limit.

Components interact by two mechanisms. The first is synchronisation of events. When a certain event takes place in one component, the synchronised event occurs simultaneously in the other component. This is used to model fast interactions, such as, for example, the trip signal from a protection relay to a circuit breaker. The second is that transition rule preconditions may refer to variables in neighbouring components, and to certain global properties of the state of the network, such as, for example, the existence of a conducting path to an active generator.

For the purpose of alarm processing, the most important distinction in the model is between events that can be explained, i.e., that follow logically from earlier events (observed or unobserved), and events that have no visible explanation, i.e., that are simply hypothesised to have occured. These are analogous to the "root cause" events that explain observed alarms in simpler model-based approaches [Wen and Chang, 1997; Wen *et al.*, 1995; Dahlgren *et al.*, 1998; Guo *et al.*, 2010]. Given a log of observations, the diagnosis engine will find a scenario that contains the fewest unexplained events. We can also assign different likelihoods to unexplained events, to find the most likely explanation.

The level of detail captured in the model represents a tradeoff between its explanatory power and the complexity of reasoning. For example, circuit breakers opening to isolate a line (perhaps triggered by a protection relay) may cause a change in the flow of power through other parts of the network, triggering, for instance, a low voltage alarm. Because our model does not include many detailed aspects of power flow (those calculations would be too expensive to carry out as part of the diagnostic reasoning) we may not be able to explain the voltage alarm. But, we can explain, e.g., an alarm indicating that voltage has dropped to zero on the isolated line, because this follows from a simpler, qualitative causal relationship.

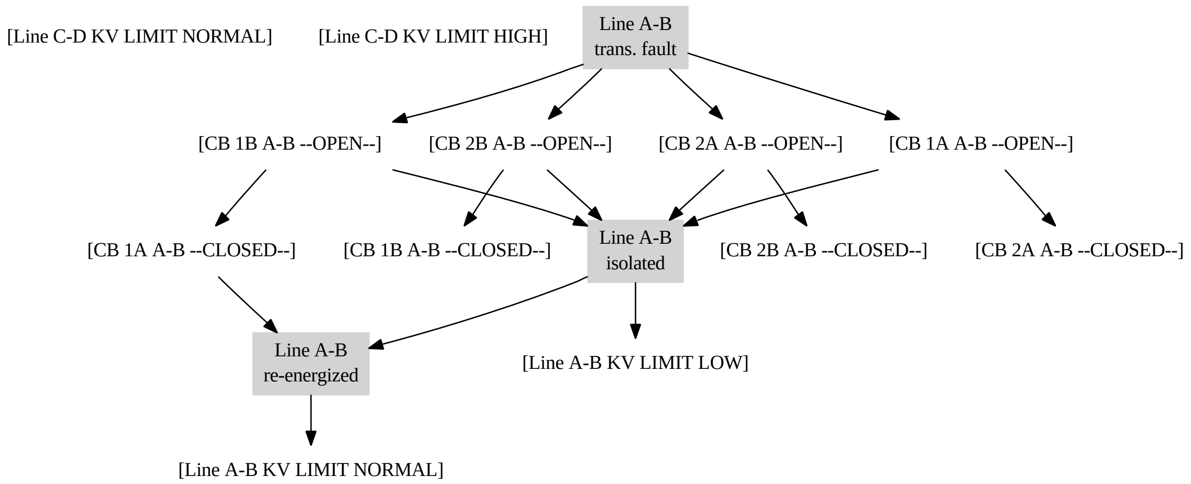

Figure 4: Scenario found by the diagnosis engine (simplified for readability). Shaded events are unobservable. Edges represent causal relationships.

3.2 Diagnosis Engine

We use a generic discrete-event systems diagnosis solver (diagnoser). It takes as input 1) a network model in MMLD format, 2) the (possibly partially known) initial state of the network describing e.g. which switches are open/closed, and 3) the observations (alarms) generated by network devices.

The reasoning performed by the diagnoser is similar to discrete state estimation under incomplete information [Zanella and Lamperti, 2003]. It involves finding a discrete state sequence that is compatible with the observations and that minimizes the occurrence of unlikely events, such as faults and other spontaneous, inexplicable events. This is a hard combinatorial problem, due to the very high number of states the system may be in. For example, in an electricity network with n switches, the open/closed status of the switches alone represents 2^n different possible discrete states. This makes the use of trivial enumerative methods ineffective.

Our diagnoser uses satisfiability (SAT) algorithms as the search mechanism. The SAT problem involves finding an assignment of values *true* and *false* (equivalently, 1 and 0) to a number of Boolean variables so that the set of clauses (Boolean sum of variables and negated variables) representing a Boolean function is satisfied (evaluates to *true*). The Boolean variables correspond to the state variables of the system at different time points, representing a sequence of states, and the events that take place at different time points. Clauses express dependencies between state variables within one time point or between consecutive time points and the observation sequence [Grastien *et al.*, 2007].

A SAT algorithm, enhanced with conflict-directed clause learning [Moskewicz *et al.*, 2001], is used for finding a value assignment to the variables. Minimization of unlikely events is achieved by repeatedly finding solutions with stronger and stronger constraints on the number of such events.

The SAT approach to diagnosis requires a bound N on the number of consecutive states. This bound is obtained from the number of observations made. Then a propositional formula Φ is defined that constrains the value of each state variable after the ith timestep (for all $i \in \{0, \dots, N\}$) and the event occurrence at the ith timestep (for all $i \in \{1, \dots, N\}$). For instance, if the event $\texttt{cb\_open}$ can take place only when the state variable $\texttt{cb\_state}$ equals $\texttt{closed}$, then the formula would include: $(\neg\texttt{cb\_open}@1 \vee (\texttt{cb\_state} = \texttt{closed})@0)$ which specifies the above property for the first transition. The formula is augmented to enforce the initial state and the observations. Any assignment of the variables satisfying Φ represents a scenario, i.e., a system evolution consistent with the given observations.

The time constraints require specific treatment. The timesteps are defined with respect to the alarms and the alarms are timestamped. Additional constraints are included in the SAT formula to ensure these time contraints are satisfied. Consider for instance that a specific circuit breaker requires at least $1s$ delay to notice an overload; consider further that the timestep 1 to 6 are stamped as follows: $\tau(1) = 0s; \tau(2) = 0.2s; \tau(3) = 0.5s; \tau(4) = 1s; \tau(5) = 1.3s; \tau(5) = 1.6s$. Then the event $\texttt{overload\_detected}$ can take place at timestep 5 only if the $\texttt{overload}$ state variable equals $\texttt{true}$ before $0.3s$, i.e. at timestep 2. This is encoded by the following formula for all $t \in \{2, \dots, 5\}$: $\texttt{overload\_detected}@5 \rightarrow (\texttt{overload} = \texttt{true})@t$.

To minimise the number of undesirable (i.e., unexplained or unlikely) events, we augment the formula Φ with a cardinality constraint on the number of occurrences of such events [Anbulagan and Grastien, 2009]. This constraint is initially set to 0 in order to find an explanation with no unlikely event and it is incremented until the SAT formula becomes satisfiable, thus exhibiting a preferred scenario.

3.3 Scenarios

A scenario $\mathcal{S} = (\Sigma, \Theta)$ is a double sequence of global states Σ and global transitions Θ, where a global transition $t \in \Theta$ is a collection of synchronized local transitions, and a global state to a collection of local states. The first state q_0 of Σ is an instantiation of the (possibly partially unspecified) initial state of the instance. Each transition $t_i \in \Theta$ leads the system from state q_{i-1} to state q_i. There are several ways a transition can affect a component state: directly if the triggering event of the transition takes place in the component, through event synchronisation if the triggering event forces an event on the component (for instance, a sensor detecting a fault may force a circuit breaker to open), or through dependant variables (for instance, an isolator electrifying all lines in a feeder).

Figure 4 shows a graphical representation of the scenario found by the diagnosis engine for the small example log in Figure 1. (The picture shows only the events that occur in the scenario, not states.)

4 Scenario Filtering

Scenarios computed with model-based diagnosis are a rich source of information, but are too detailed to present directly to operators. Further processing is required to obtain summarized data that an operator can quickly understand. In this section we discuss four alarm processing functions, each providing different types of information to the operator.

4.1 Alarm Clustering

Alarm clustering partitions an alarm log into causally independent subsets which can be understood more easily.

Clustering is performed on all events in a scenario, including unobservable ones that have been inferred from entries in the alarm log. The clustering of the alarm log is then trivially obtained by filtering away all hidden events. In the rest of this section, we use the term cluster to refer to collections of events including unobservable ones.

For example, in the scenario found by the diagnosis engine for the alarm log in Figure 1, the set of all alarms referring to the A-B line, together with several inferred unobservable events, form a cluster. The two alarms referring to the C-D line, however, are not clustered, because there is no (apparent) causal relationship between them.

Our clustering is based on partitioning by an equivalence relation (transitive, reflexive, symmetric), which expresses causal dependencies between events. Let $\mathcal{S} = (\Sigma, \Theta)$ be a scenario with an ordered set of states and an ordered set of global transitions. A global transition $t \in \Theta$ is a collection of synchronized local transitions. As synchronized local transitions are interrelated, they will all belong to the same cluster. Additionally, transitions connected with cause-effect relations belong to the same cluster.

The clustering algorithm (Algorithm 1) captures both these types of relations between transitions. It processes global transitions from Θ in order. When the ith transition $t \in \Theta$ is processed, RelatedClusters computes a collection $\mathcal{R}$ of zero or more clusters which we call a *support cluster set* for t. By definition, a support cluster set is a subset of the clusters built so far with the property that applying their transitions in the same order as in Θ, starting from the initial state, results in a state where the preconditions of t hold. Furthermore, a support cluster set is set-inclusion minimal.

Notice that it is possible that several support cluster sets exist for a transition t. For example, a given precondition of t could be supported independently by two different transitions t' and t'' belonging to two different clusters c' and c''. We can add either c' or c'' to a support cluster set. Our algorithm builds only one support cluster set per transition t. The implementation detailed in Algorithms 2 and 3 favors considering as related to t the most recent of transitions t' and t''.

Depending on the size of $\mathcal{R}$, one of the following outcomes can happen in MergeClusters: (i) If $|\mathcal{R}| = 0$, a new cluster is created and t is added to the new cluster. Since t doesn't need any of the previous transitions in the scenario to support its preconditions, t is considered independent from the previous transitions. In this case, t becomes the seed of a new cluster. (ii) When only one cluster is sufficient to support t's precondition, t is added to that cluster ($|\mathcal{R}| = 1$). (iii) When two or more existing clusters are required to support t's preconditions ($|\mathcal{R}| > 1$), all those clusters are merged and t is added to the resulting cluster.

Algorithm 1 Clustering(Σ, Θ)

```
$\mathcal{C} \leftarrow \emptyset$ {initialize set of clusters}
for all $t \in \Theta$ do
  $\mathcal{R} \leftarrow$ RelatedClusters$(\mathcal{C}, t)$ {return zero or more clusters related to transition $t$}
  $c \leftarrow$ MergeClusters$(\mathcal{R})$ {return one (possibly empty) cluster}
  extend $c$ to include $t$
  $c \rightarrow \mathcal{C}$ {add $c$ to set of clusters}
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{R}$ {remove previously merged clusters}
return $\mathcal{C}$
```

Algorithm 2 RelatedClusters$(\mathcal{C}, t)$

```
$\mathcal{R} \leftarrow \emptyset$ {initialize set of clusters related to $t$}
$\mathcal{T} \leftarrow \mathcal{C}$ {copy the contents of $\mathcal{C}$ into temporary data structure}
for all $c \in \mathcal{C}$ do
  if Related$(\mathcal{T}, c, t)$ then
    $c \rightarrow \mathcal{R}$
  else
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \{c\}$
return $\mathcal{R}$
```

Algorithm 3 Related$(\mathcal{C}, c, t)$

```
$\mathcal{T} \leftarrow \mathcal{C} \setminus \{c\}$ {initialize temporary data structure}
$\theta \leftarrow$ all transitions from $\mathcal{T}$ in order
$\sigma \leftarrow \gamma(\sigma_0, \theta)$ {apply transitions contained in $\theta$ starting in initial state $\sigma_0$ and store resulting state $\sigma$}
if $t$ is applicable in $\sigma$ then
  return false {$t$ is independent from $c$}
else
  return true {$t$ depends on $c$}
```

4.2 Root Cause Analysis

Root cause events are the unexplained spontaneous events, whose preconditions do not necessarily depend on other events. For example, in the scenario depicted in Figure 4, the transient fault on line A-B is the root cause of the entire large cluster of events. Root cause idenfication is immediate from the scenario: we simply select those spontaneous events labelled as unexplained or faults.

The combination of alarm clustering and root cause analysis offers an additional option, allowing to focus root cause analysis on a subset of the alarms.

4.3 Alarms Independent of Catastrophic Events

A catastrophic fault can create a cascade of many follow-up alarms. Such follow-up alarms get mixed in the alarm log with alarms that are independent from the catastrophic event. As a result, incident-independent alarms could get lost unless they can be automatically filtered from the alarm log.

Alarm clustering is well suited to provide such functionality. Once a scenario is partitioned into clusters, the set of incident-independent alarms can be computed as the set of alarms from the clusters that do not contain the incident event.

4.4 Live Alarms

Our fourth alarm filtering technique is highlighting what we call "live" alarms. Intuitively, live events are part of a process that has started but not yet converged back to the normal (nominal) state of the involved network components. For example, the alarm "Line C-D KV LIMIT HIGH" in Figure 1 is a live alarm, since it indicates that the component is not in its nominal state. However, subsequence "Line A-B KV LIMIT LOW" followed by "Line A-B KV LIMIT NORMAL" is not live, since the component is at the end back in its nominal state.

Model based reasoning followed by scenario clustering offers a good basis for highlighting live alarms. Selecting only the alarms from clusters whose current last state is not a nominal state will provide the desired functionality.

5 Experiments

TransGrid, the operator of the transmission network in New South Wales, provided us with the topology of their network (i.e., the list of components and connections between components), and a log containing 2246 entries (alarms and commands). The log covers roughly fifteen hours: the first two thirds are routine operation, then a major fault situation arises and the rest of the log chronicles the operators' efforts to reconfigure the network to restore service.

Based on this data we built a model of the TransGrid network, including a discrete state/transition model of each component type, and the network topology which determines event synchronisations between components, as discussed in Section 3.1. We focused on a subset of alarm types, mainly those related to the primary electrical system (e.g., switch gear state changes, high/low voltages, etc). Restricted to these types, the total number of observations (alarms and commands) is only 731.

For the purpose of experimentation, we split the alarm log into (a) fixed size (one minute) time windows, and (b) variable sized windows separated by periods of at least one minutes silence (i.e., where no alarm is generated). This gave us 129 problem instances. For each of those we extract a corresponding model, which contains only the parts of the network that are potentially relevant to reasoning about this set of alarms. This subnetwork is computed iteratively, starting with the components that emitted the alarms in the instance alarm log. Components that may influence (according to the model) components already in the subnetwork are added, until a fixpoint is reached.

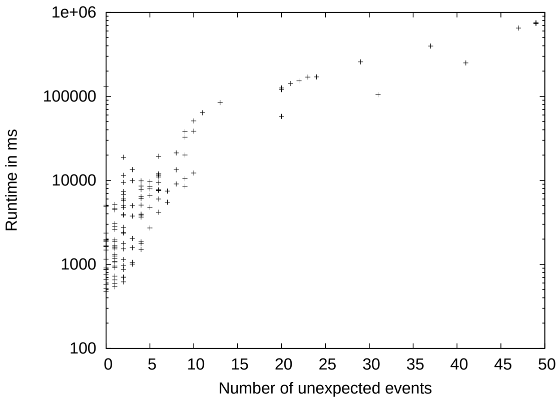

Figure 5: Runtime wrt the number of unexplained events.

For each problem instance, we ran the diagnoser to generate a preferred scenario, and then used the methods presented in the previous section to generate clusters of alarms. We set a limit of one minute for the entire process (scenario generation and filtering). This is somewhat generous, as operators would probably prefer an answer within a few seconds, but our implementation being only a prototype, we believe it can be easily improved by a factor of 10 or more.

Out of 129, 16 instances were not solved within the time limit. The most difficult of these instances involves 105 components and 146 alarms, and the SAT problem was so large (in terms of boolean variables and clauses in the SAT formula) that it could not even be read by the SAT solver; the simplest one involves 36 components and 16 alarms. However, what really makes the problem hard is the number of unexplained events. Remember that the preferred scenario is defined as the path in the model that matches the observations and minimizes the number k of unexplained events. Therefore, the problem is much simpler when k is small. This is illustrated Figure 5. The model we used in the experiments is quite simple, and in many cases is not able to link a single unexplained event to as many alarms as it should. A more precise model would decrease k, and hence improve both the runtimes and the quality of scenarios.

There are no metrics to determine how good a filtering is: on the one hand, having few clusters shows that many alarms are related, while on the other, having many clusters shows that unrelated events have been successfully taken apart. Nevertheless, the usefulness of the filtering method is obvious in several cases. For the example log in Figure 1, for example, a large cluster of related alarms (the four circuit breakers opening to isolate a line and later reclosing), and a single root cause for this cluster, is identified, and the second of the remaining two of unrelated alarms (showing voltage fluctuating in a different part of the network) is highlighted as live. An unrelated event like this can be quite important, while being very easy to miss in the flood of alarms. Our alarm processor can draw the operator's attention to it.

6 Conclusions

Intelligent alarm processing tools are important for control room operators who supervise and manage large-scale systems, such as power networks. Scalable and accurate alarm processing capabilities are becoming even more critical as increasing instrumentation of networks generates a dramatically larger volume of alarm data.

Earlier approaches to alarm processing have been predominantly syntactic, based on text-level filtering or shallow system models. We have presented an approach that uses a comprehensive model of a network, including functional models of all components as well as the global behavior that emerges from intercomponent interactions. Online model-based diagnosis ensures that important information, such as the current state of the network, is taken into account when applying alarm processing algorithms. We demonstrated our ideas on a model representing the network of a leading Australian transmission company.

Future work includes developing a more refined and detailed network model, and analysing several scenarios at a time, for a more comprehensive diagnosis. We are also interested in performing alarm processing incrementally, updating the output as new alarms arrive.

Acknowledgments

We thank TransGrid for permission to use their data.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[Anbulagan and Grastien, 2009] Anbulagan and A. Grastien. Importance of variables semantic in CNF encoding of cardinality constraints. In *Symposium on Abstraction, Reformulation and Approximation*, 2009.

[Cassandras and Lafortune, 1999] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[Cordier *et al.*, 1998] M.-O. Cordier, J.-P. Krivine, P. Laborie, and S. Thiebaux. Alarm processing and reconfiguration in power distribution systems. In *Tasks and Methods in Applied Artificial Intelligence*, volume 1416, pages 230–241. Springer, 1998.

[Dahlgren *et al.*, 1998] R. Dahlgren, G. Rosenwald, C. C. Liu, S. Muchlinski, A. Eide, and D. Sobajic. Model-based synthesis and suppression of transformer alarms in a control center environment. *Power Delivery, IEEE Transactions on*, 13(3):843–848, 1998.

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, 2007.

[Guo *et al.*, 2010] W. Guo, F. Wen, Z. Liao, L. Wei, and J. Xin. An analytic model-based approach for power system alarm processing employing temporal constraint network. *Power Delivery, IEEE Transactions on*, 25(4):2435–2447, 2010.

[Ipakchi and Albuyeh, 2009] A. Ipakchi and F. Albuyeh. Grid of the future. *IEEE Power and Energy Magazine*, 7:52–62, 2009.

[Kirschen and Wollenberg, 1992] D. S. Kirschen and B. F. Wollenberg. Intelligent alarm processing in power systems. *Proceedings of the IEEE*, 80(5):663–672, 1992.

[Laborie and Krivine, 1997a] P. Laborie and J-P. Krivine. Automatic generation of chronicles and its application to alarm processing in power distribution systems. In *International Workshop on Principles of Diagnosis (DX'97)*, pages 61–68, 1997.

[Laborie and Krivine, 1997b] P. Laborie and J.-P. Krivine. Gemo: A model-based approach for an alarm processing function in power distribution networks. In *International Conference on Intelligent System Application to Power Systems (ISAP'97)*, pages 135–141, 1997.

[Larsson and DeBor, 2007] J. E. Larsson and J. DeBor. Realtime root cause analysis for complex technical systems. In *IEEE 8th Human Factors and Power Plants and HPRCT 13th Annual Meeting*, pages 156–163, 2007.

[Larsson, 2002] J. E. Larsson. Diagnostic reasoning based on means-end models: experiences and future prospects. *Knowledge-Based Systems*, 15(1-2):103–110, 2002.

[Lin *et al.*, 1998] Y.-K. Lin, M.-S. Tsai, Y.-Y. Hong, and Y.-C. Tsuei. Application of model based intelligent alarm processor for the power supply system of the mass rapid transit system of taipei. pages 387–392, 1998.

[Lin *et al.*, 2004] W.-M. Lin, C.-H. Lin, and Z.-C. Sun. Adaptive multiple fault detection and alarm processing for loop system with probabilistic network. *Power Delivery, IEEE Transactions on*, 19(1):64–69, 2004.

[McDonald *et al.*, 1991] J. R. McDonald, G.M. Burt, and D. J. Young. Alarm processing and fault diagnosis using knowledge based systems for transmission and distribution network control. *Transmission and Distribution Conference, Proceedings of the IEEE Power Engineering Society*, pages 280–286, 1991.

[Meza *et al.*, 2001] E. M. Meza, J. C. S. de Souza, M. T. Schilling, and M. B. Do Coutto Filho. Exploring fuzzy relations for alarm processing and fault location in electrical power systems. In *IEEE Power Tech Proceedings*, 2001.

[Moskewicz *et al.*, 2001] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th ACM/IEEE*

Design Automation Conference (DAC'01), pages 530–535. ACM Press, 2001.

[Protopapas *et al.*, 1991] C. A. Protopapas, K. P. Psaltiras, and A. V. Machias. An expert system for substation fault diagnosis and alarm processing. *Power Delivery, IEEE Transactions on*, 6(2):648–655, 1991.

[Taisne, 2006] J. Taisne. Intelligent alarm processor based on chronicle recognition for transmission and distribution system. *Power Systems Conference and Exposition PSCE '06. IEEE PES*, pages 1606–1611, 2006.

[Wen and Chang, 1997] E. S. Wen and C. S. Chang. Tabu search approach to alarm processing in power systems. *Generation, Transmission and Distribution, IEE Proceedings*, 144(1):31–38, 1997.

[Wen *et al.*, 1995] F. Wen, C. S. Chang, and D. Srinivasan. Alarm processing in power systems using a genetic algorithm. *Evolutionary Computation, IEEE International Conference on*, 1995.

[Zanella and Lamperti, 2003] M. Zanella and G. Lamperti. *Diagnosis of active systems*. Kluwer Academic Publishers, 2003.