Diagnosis As Planning: Two Case Studies

P@trik Haslum and **Alban Grastien**
Australian National University & NICTA
`firstname.lastname@anu.edu.au` or `@nicta.com.au`

Abstract

Diagnosis of discrete event systems amounts to finding good explanations, in the form of system trajectories consistent with a given set of partially ordered observations. This problem is closely related to planning, and in fact can be recast as a classical planning problem. We formulate a PDDL encoding of this diagnosis problem, and use it to evaluate planners representing a variety of planning paradigms on two realistic case studies. Results demonstrate that certain planning techniques have the potential to be very useful in diagnosis, but on the whole, current planners are far from a practical means of solving diagnosis problems.

Introduction

In automating the operation of complex technical systems, automated monitoring and diagnosis are as important as automated planning and control.

Traditionally, the diagnosis task is to answer the question: what is wrong? In model-based diagnosis, this means inferring, from a system description and a set of observations, what *mode* the system may be operating in: nominal, one of a number of known fault modes, or an unknown fault mode (e.g., de Kleer and Williams 1989). A typical example is the diagnosis of an electronic circuit, where observations are test inputs paired with measured outputs, and a fault mode is a subset of faulty components. In this setting, the system is static: observations are assumed to all be generated by the same mode, and the system imposes no order on them.

In contrast, diagnosis of dynamical systems may be posed as the question: what has happened? That is, given a system model and a (partially) ordered set of observations, the diagnosis task is to identify possible evolutions of the system over time which would generate the given observations, in an order consistent with that given (Cordier and Thiébaux 1994; McIlraith 1994). Crucially for our case studies, this permits system models that are inherently non-deterministic, even when operating correctly. In most cases, there are many system histories consistent with observations. Thus, there is a notion of preference over histories, and the diagnoser is required to find one, some or all preferred explanations.

There is a close connection between diagnosis of dynamical systems and planning (noted by Cordier & Thiébaux, 1994, and McIlraith, 1994), as the task of generating a (most preferred) system event history to match given observations can be viewed as a plan generation problem. In spite of this, the two fields have developed quite different methods, with much work in diagnosis exploiting off-line analysis of the system model to enhance on-line diagnosis, e.g., by building a system-specific diagnoser (Sampath et al. 1996), identifying if sufficient conditions for using faster methods hold (e.g., Basile et al. 2003), or decentralising the work of diagnosis (e.g., Pencolé and Cordier 2005). The SAT-based diagnoser by Grastien et al. (2007), inspired by the use of SAT for planning and model-checking, is one of a few examples of transfer of techniques between the two fields. To determine if methods that have been successful at solving planning problems will be so also for diagnosis of discrete event systems, we formulate a reduction of the diagnosis problem to planning, i.e., an encoding of the problem in PDDL. It is of course unlikely that such a direct problem translation is the most effective way of applying planning techniques to diagnosis problems, but it is a very useful tool to evaluate a spectrum of planning methods, due to the availability of diverse domain-independent planning systems.

Comparing the effectiveness of planning techniques with existing diagnosis methods raises further challenges. First, there is not one single diagnosis problem. The problem we consider can be characterised as *passive*, *non-exhaustive* diagnosis (of discrete event systems). In contrast, much work concerns exhaustive diagnosis (i.e., finding all preferred explanations consistent with the facts), or active diagnosis (where the diagnoser can take actions to control which observations are made; cf., e.g., Kuhn et al. 2008). These are different problems, and require different solutions. Second, the diagnosis research community does not have the same focus on empirical evaluation over common sets of benchmarks as in planning. Although diagnosis researchers develop domain-independent methods, very few "off the shelf" implementations of domain-independent diagnosis systems are available, and also very few benchmark problem sets.

As a step towards remedying this situation, we formalise two realistic diagnosis problems[1], which we use to evaluate the effectiveness of planners representing several different paradigms. The first case study problem comes from a UAV research project, and is a fault detection problem. The sec-

[1]To the extent that we are permitted, we will make these formalisations available.

ond comes from the domain of electric power transmission, where we aim to do "intelligent alarm processing", by minimising the number of "unexplained" observations. In both cases, the purpose of diagnostic reasoning is to support the situational awareness of a human decision maker. Thus, the diagnoser must reliably provide information that is timely and correct. The meaning of "timely" is not precisely defined, but as a rule of thumb, solutions should be available within a few tens of seconds. Results are mixed: For the first case, a combination of planning techniques present a viable solution. In the second case, the SAT-based diagnosis engine performs better than the best planners, but no approach quite meets the demands of the application.

We are aware only of one similar study. Sohrabi, Baier & McIllraith (2010) proposed an encoding of discrete event diagnosis as a planning problem with temporally extended goals (via the situation calculus), and tested the ability of a heuristic search planner supporting such goals to solve a diagnosis problem. (They also used our encoding to apply the FF planner, which performed better.) The problem they used, however, is an artificial example (introduced by Grastien et al. 2007), and the results they observed do not generalise to our case study problems.

Diagnosis of Discrete Event Systems

The dynamical systems we consider are modelled as discrete state, discrete event systems. This fits naturally with classical planning models. In principle, there is no obstacle to applying the same reduction to diagnosis of timed or hybrid systems – indeed, we construct both a timed and a classical model for our second case study – but the range of planners capable of dealing effectively with such problems is much narrower. In this section, we present a brief review of the discrete event system diagnosis problem. Examples, and details of the reduction to planning, will be described along with our two case studies in the following sections.

A finite discrete event system consists of a finite collection of state variables, each with a finite domain of values, and a finite set of transitions. Each transition has a precondition and a deterministic effect on some subset of the state variables. (Other variables keep their values.) Although each transition is deterministic, the system model as a whole is generally non-deterministic, in the sense that in any state, several transitions may be applicable and there is no deterministic rule that dictates which is taken. For modelling convenience, it is common to divide the system up into components, each an instance of a component type which defines variables and transitions in a schematic way. To model non-trivial behaviour the components must be able to interact, using some communications mechanism such as shared variables, message queues or synchronised transitions.

Some transitions emit one or more *observable events*. There can be several transitions emitting the same event. Given a system model, a set of possible initial system states, a set of observations, O, each labelled with an event $\mathrm{event}(o_i)$, and a partial order $\prec$ on O, a *discrete-event diagnosis* is a sequence of transitions, applicable from some initial state, such that the events emitted by the sequence correpond one-to-one with the set of observations, and the order of these events induced by the sequence of transitions is consistent with the given partial order on the observations. In both our case studies, observations are time stamped, but time stamps are not precise or accurate enough to totally order the observations. Therefore, we assume only a partial order. In any case, this does not complicate the formulation of the diagnosis task as a planning problem.

In the models we consider, transitions are divided into "good" and "bad", and the objective is to minimise the number of bad transitions occurring in the explanation sequence. (The bad transitions are often referred to as *faults*, but this terminology is a bit misleading in the case of alarm processing.) In planning terms this corresponds to the objective of minimising total cost, where only fault transitions have a non-zero cost. In the models we construct, the bad transitions are all equally bad, and therefore all have unit cost, but obviously varied costs could be used to express degrees of badness. For example, if we know the probability of each fault transition occurring, we can assign a cost proportional to its negative logarithm to obtain a most likely explanation. Other preference criteria are also conceivable.

We can also consider more stringent notions of diagnosis: In planning, a *disjunctive action landmark* is defined as a set of actions at least one of which must appear in any valid plan (Helmert and Domshlak 2009). Analogously, we may consider a set of fault transitions at least one of which must occur in any system history consistent with the observations. This is more informative, since it identifies a – albeit disjunctive – set of faults that must have happened, rather than a (minimum cost) set of faults that may have happened. It also has some similarity with the notion of *conflicts* in static system diagnosis (Reiter 1987). A conflict is a set of statements about the system mode that is inconsistent with the system model and the set of observations. Thus, every diagnosis must include the negation of at least one statement from each conflict set. For static systems, there is a one-to-one correpondence between minimal diagnoses and minimal hitting sets over the set of all minimal conflict sets. For discrete event systems, however, such a correspondence does not hold in general.

When modelling a system for diagnosis we have the freedom to trade off the fidelity of the model (and thus its complexity) and its diagnostic power. Abstracting away some aspects of the real system means only that the model may allow more explanations of a given set of observations (some of which may not correspond to possible system behaviours), and thus fewer certain conclusions. We will see this repeatedly in modelling our case study problems.

Intelligent Alarm Processing

The availability of remote sensing and control facilities means that today very large industrial systems, such as power or telecommunications networks, can be overseen and managed by a few operators in a central location. Fault conditions in such large systems 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. This problem has been recognised for some time (e.g., Prince, Wollenberg, and Bertagnolli 1989), and there has been a lot of work on the use of AI techniques to aid operators by filtering, prioritising and synthesising alarms. This is known as "intelligent alarm processing".

Two approaches to the problem can be distinguished: One is to view it as pattern recognition, i.e., identifying "situations" or "episodes", that are meaningful to operators, in the alarm stream. This has been developed using methods such as neural networks and chronicles (e.g., Dousson 1996). The other is to view it as root cause analysis, i.e., finding a smaller set of "root cause events" that together would cause the observed alarms. From this perspective, there is no technical difference between alarm processing and diagnosis: the difference lies in the model, and in how it is interpreted. The root cause events, which we seek to minimise, are "unexplained" but not necessarily faults. Model validity means only that for those alarms which are explained, the explanations are correct. It is acceptable, if undesirable, to leave any alarm unexplained. Most work on this approach uses a static system model (e.g., Dijk 1992; Larsson 2009), relying on *ad hoc* methods to divide the flow of alarms into sets for analysis. (An exception is the method described by Guo et al., 2010, which uses a system model based on a temporal constraint network and computes windows for analysis on the basis of the time constraints.) We formulate it instead as discrete event diagnosis, which allows for potentially more powerful explanatory models.

First Case Study: Detecting Missing Events

Input data for our first case study is a set of event logs recorded on the ground during test flights of an autonomous unmanned helicopter (UAV) in 2004 as part of the WITAS project.[2] The logs contain events issued by the UAV control system. The high-level control system consists of dynamically created instances of concurrent software components, called "task procedures" (TPs), which perform mission tasks (e.g., navigating the UAV to a position) by a combination of issuing commands to the low-level (real-time) control system, calling on-board "services" (e.g., the path planner), and invoking other TPs to perform subtasks. Events are sent by TPs, e.g., to signal when a task has been completed, and are tagged with a unique identifier of the sending TP instance. Some events also include additional data. The low-level control system sends events to signal changes in state and in response to commands (e.g., completed or failed).

Due to occasional data overload and loss of telemetry, logs are not complete: some events are missing. The diagnosis task is to detect when such gaps exist. Ideally, we would also like to infer which events are missing, but it is in most cases not possible to do this uniquely.

Formulation as a Planning Problem

TPs are finite state machines augmented with data variables and bits of code executed at transitions. TPs are deterministic, but because we abstract away many details, our models of them are non-deterministic finite automata. For example, the sequence of events generated by the Fly3D TP, which flies the UAV along a given path, depends on the number of waypoints in the path. By leaving the path out of the model, the number of iterations of the TPs main loop becomes non-deterministic. This is an example of the trade-off between model complexity and diagnostic power: if more details were included we could, in some cases, detect missing events that are not detectable with the current model.

Encoding automata with predicates and actions in PDDL is straightforward. Some transitions are synchronised, modelling communication between TPs, but since synchronised transitions involve a fixed number of automata, this can be simply modelled by an action that conjoins the preconditions and effects of the participating transitions. Synchronising an unbounded number of simultaneous transitions requires a more elaborate encoding. (An example of a PDDL encoding of general inter-process communication using message queues is described by Hoffmann et al., 2006, for the Promela domain.) The set of TP identifiers is unbounded, but when creating a planning problem instance for a particular event log, we include only those identifiers mentioned in the log. (In one instance, it was necessary to include an identifier that does not appear in the log for the problem to be solvable. While there are a number of ways that such missing identifiers could be inferred, we have no general solution to this issue.) Finally, each log begins at the start of a mission, with the system in an idle state. Thus, we can assume a single, fully known initial state.

Encoding Observations Some transitions emit an observable event. Recall that in the discrete event diagnosis problem, we have a set of observations, $O = \{o_1, \ldots, o_n\}$, each labelled with an event $\mathrm{event}(o_i)$, and a partial order $\prec$ on O, and we seek a transition sequence that reproduces the observations. This can be formulated as a planning goal as follows: Each observation o can be in one of three states: (future o), meaning that some observation ordered before o is yet to be made; (pending o), meaning that o can be the next observation made; and (observed o), meaning that o has been generated. The initial state is (pending o) if o is minimal in the order on observations and (future o) otherwise. The goal is (observed o), for all $o \in O$. Each action corresponding to a transition that emits an observable event e is given an additional parameter ?o; its precondition is extended with (pending ?o) and $\mathrm{event}(?\mathrm{o}) = e$, and its effect with (not (pending ?o)) and (observed ?o). An action corresponding to several synchronised transitions that emit events will have one such observation parameter for each event. These parameters are required to be distinct. To ensure that observations are made consistently with the given order, an observation o can change state from (future o) to (pending o) only when all observations preceding o have been made. We encode this with an action, (advance-to o), whose precondition is (future o) and (observed o') for all $o' \prec o$, and whose effect is (not (future o)) and (pending o).

Proposition 1 *This encoding is correct, in the sense that any valid plan will generate each observation in O exactly once, in an order that is consistent with the given order $\prec$.*

[2] A detailed presentation of the WITAS project and the architecture of the control system is provided by Doherty *et al.* (2004). See also `http://www.ida.liu.se/ext/witas/`.

Proof: For each observation $o \in O$, a valid plan must contain at least one action that exchanges (pending o) for (observed o), and no more than one since it is not possible to reverse the exchange. By construction, this action corresponds to a transition that emits $\mathrm{event}(o)$. Suppose two observations $o' \prec o$ are generated in an inconsistent order, i.e., o before o': Since o is not minimal in $\prec$, (future o) holds in the initial state, and the only action that may exchange (future o) for (pending o), which is a precondition of any action generating o, is (advance-to o). But the precondition of this action includes (observed o'), so it cannot be applied before o' has been generated. □

A slightly simpler encoding would distinguish only between observations made and not made, with the ordering condition part of the precondition of each action emitting an observable event. However, distinguishing future and pending observations allows using observations as "time stamps", which will be useful in modelling our second case study.

Events in the flight logs are stamped with the time that the event was generated. The order on observations is still partial, because events separated by too small a margin (less than $^1/_{100}$th of a second) cannot be reliably ordered. It is, however, a special kind of partial order, namely, a sequence of sets of mutually unordered elements.

Encoding Faults The faults that we wish to detect are lost events. Thus, for every transition that emits an observable event, the model has an identical fault transition without any observation. In the PDDL formulation, actions corresponding to fault transitions have a cost of 1 while all non-fault actions have a cost of zero, and the objective is to minimise the cost of the plan. In particular, a zero-cost plan exists iff the observations can be explained without missing events.

Experiments and Results

The data set comprises 8 logs, ranging in length from 41 to 273 observations. Five are complete, while three have between 5 and 17 missing events. The number of model components (i.e., TP instances) ranges from 3 to 26, and the number of states per component between 20 and 129. To obtain a larger set of problems for experimentation, we take prefixes of these logs, of increasing length, and remove randomly chosen events up to a desired total. This way, we obtain 196 instances, 36 of which are complete.

Since the main task is to decide if a zero-cost plan exists, it is natural to use planner that guarantees minimal-cost plans. For this we use the Fast Downward implementation of A* search with the admissible landmark-cut heuristic (Helmert and Domshlak 2009).[3] This planner finds zero-cost plans for all 36 instances without missing events: 30 problems are solved in less than 10 seconds, but the longest runtime for a problem is over 150 seconds. It does not solve all 160 instances with missing events. However, it does prove a lower bound on cost greater than zero – thus *detecting* that some event must be missing – for all but one of these problems, never taking more than 3 seconds to do so. (The one problem were it fails to detect missing events actually admits a zero cost solution, so in this case the blame lies with the model.) In fact, only applying the landmark-cut heuristic to the initial state is sufficient to detect some event is missing in 150 instances (including the three full-length logs).

We also use a cost-ignorant planner: greedy best-first search using the FF heuristic (also implemented in Fast Downward). It solves all but two problems, but generates false positives, i.e., plans of non-zero cost, for 35 of the 36 logs without missing events. However, when run on a model without fault transitions, and thus forced to find only zero-fault plans, the planner solves all instances that admit such solutions, even within a 30 second time limit. In summary, using the combination of two one-sided tests – an admissible heuristic for fault detection and a fast planner for "no-fault detection" – seems to be a viable approach, though it fails to reach a decision for a few problems.

The SAT-based diagnoser (Grastien et al. 2007) suffers from the fact that explanation trajectories are very long, and fails to solve any problem with more than 100 observations. The SAT-based planner that we tried (Mp, by Rintanen 2010) exhibits the same behaviour, though it scales a little further, solving one problem with 170 observations.

Second Case Study: Alarm Processing

Input data for our second case study is an alarm log from the operations center of TransGrid, the company that owns and operates the electricity transmission network in NSW and the ACT, Australia. The log contains alarms generated by automatic equipment – switch gear, voltage and power sensors and regulators, etc. – located throughout the transmission network, as well as commands issued by the operators. It 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. Figure 2 gives an indication of how alarms are distributed over time.

Our aim is to do "intelligent" alarm processing, which begins with finding a consistent system history with the fewest "unexplained" events. What is an unexplained event can depend on context. For example, an alarm indicating that a circuit breaker has opened may be explained by the observation that it was commanded to open a short time earlier. Four breakers isolating a line (cf. figure 1) opening almost simultaneously may be explained by an electrical fault on the line triggering the line protection relays, if the line was energised. In this case, the occurrence of the line fault is itself an unexplained (and unobservable) event. If no reason for the breaker opening is discernable within the model, the alarm remains unexplained.

The log contains 2246 entries (alarms and commands) in total. However, our model considers only a subset of alarm types and restricted to these the total number of observations is only 731. The model is also overly simplified in some other respects (discussed below); to achieve a level of alarm processing that would truly benefit end users will require a much more detailed and comprehensive model. This, however, is not a limitation of the approach of formulating alarm

[3] We replaced the Fast Downward translator component with a different translator. Except where otherwise noted, experiments were run with 30 minutes CPU time and 2Gb memory limits.

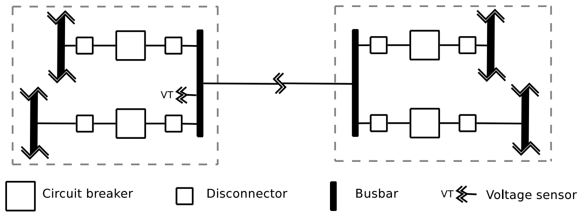


Figure 1: Schematic of a part of the transmission network, showing a (long-distance) line and its isolators.

processing as a diagnosis problem, or of reducing this to a planning problem: the limitation is in our current knowledge of the system, and in the ability of existing planners to reason with a more detailed model.

Formulation as a Planning Problem

We model components of the system as non-deterministic finite state machines, and use a combination of synchronised transitions and shared variables to model their interactions. The dynamics of the system depend a great deal on the propagation of electricity through the network. Most of this we abstract away from the model, since to capture it would require a complex hybrid continuous/discrete model. Moreover, the electrical state of any component can depend on network-wide topological properties, such as the existence of a conducting path to an active generator. Such conditions could be modelled using PDDLs derived predicates, as done by Hoffmann et al. (2006) for the PSR domain. However, the range of planners that support this feature is very limited, and the size of the network – especially coupled with the incomplete initial state – mean that such a formulation would most likely be intractable.

Therefore, the model includes only local topological information, and reasoning about electrical properties is limited to what can be inferred from this. For example, the circuit breakers in figure 1 are all marked as being isolators of the line. Any transition that changes the state of one of these breakers to "open" flags that the isolation state of the line may have changed. This enables the line to transition to being isolated (and having changed), if all isolators are now open. When the line has changed to the isolated state, we know it is not energized, and this can explain an alarm signalling voltage on the line dropping to zero shortly thereafter. Again, this is an example of trading reduced explanatory power for a a simpler, but still valid, model. For example, the line may also become de-energized as a result of all generators currently feeding it switching off, but this explanation is not discernible in our model, so in that case the voltage drop alarm would go unexplained.

Because the model does not use global relations, such as connectivity, the set of components that can influence any given component is bounded. Thus, when creating a planning problem instance for a particular event log, we include only components mentioned in the log, or directly related to those mentioned in the log. This reduces the size of problems significantly: the whole network has over 10,000 components in the primary electrical system alone, while the largest problems we consider contain a few hundred.

Encoding an Incomplete Initial State In this study, we have no knowledge of the initial state.[4] However, a diagnosis is a consistent system history starting from *some* possible initial state, so we can let the planner choose the values of initially unknown state variables (as noted by Sohrabi et al. 2010). This is done by initialising unknown variables to an "unknown" value and including actions that allow them to be set, once, to any value. Variables may remain unknown as long as no action that depends on them is taken.

Encoding Time-Dependent Conditions As illustrated by many of the examples above, time plays an important role in the dynamics of the electricity network, and it is therefore important to include some time constraints into the system model. For example, if a breaker opens, causing a line to become isolated, a voltage drop alarm may be emitted. But if so, we expect that alarm to follow within a few seconds of the breaker opening: a change in isolation state cannot explain a voltage drop that takes place hours later. In other words, the line component should remain in the "changed to isolated" state only for a limited time, and then transition to a different state (although still isolated). Similarly, a fault causing protection to trip should cause all isolation breakers to open within a second of each other – if they don't, protection tripping the breakers is not a plausible explanation.

We can model such time constraints using the durative actions of PDDL2.1. However, we can also make use of the observations as "time stamps", that way obtaining a classical planning model. This is advantageous because the classical model is accessible to many more planners. We found only one planner capable of solving the timed PDDL2.1 model, and even that planner performs better on the classical model!

As in our first case study, observations in the event log are time stamped (though in this case only to one seconds precision). Thus, for any pair of observations we know the delay, $\delta(o_i, o_j) = \tau(o_j) - \tau(o_i)$, between them, and order the observations when that delay exceeds a fixed threshold: $o_i \prec o_j$ iff $\delta(o_i, o_j) > T$. (We use $T = 0$, i.e., we order observations whenever there is a noticable difference in their time stamps. A greater threshold is appropriate when time stamps are not accurate.)

The encoding of time constraints in the classical model is most easily explained by an example: Suppose (open-isolator ?b ?l ?o) is the action of opening breaker ?b, which is an isolator of line ?l (?o is the observation to be matched by the event, signalling the breaker opening, emitted by the transition). As described above, besides changing the state of ?b to open, this action adds a proposition (iso-maybe-chgd ?l), representing that the isolation state of ?l may have changed. Now, we want to model that this proposition will only persist for a fixed time (say, 10 seconds). To achieve this, we add to the predicate iso-maybe-chgd an extra pa-

[4]In actual application, this would likely not be the case. The state of network devices is polled on a regular basis, and even if that information is not used, the log of past events would indicate the state of many components. The case of an unknown initial state may be thought of as a "cold start" of the alarm processor.

rameter ?o, to be filled by an observation that is "current" at the time when (iso-maybe-chgd ?l ?o) is made true. This way, the observation acts as a time stamp for the proposition. An observation is current if (pending ?o) is true. (Action (open-isolator ?b ?l ?o) already has an observation argument, which must be pending when the action takes place. To any action that adds a transient condition and which does not emit an observation, we must add such an observation argument.) Every occurrence of iso-maybe-chgd in the precondition of an action must now existentially quantify the extra observation parameter. (For simple conjunctive action preconditions, existential quantification is equivalent to adding an extra action parameter.) To ensure that (iso-maybe-chgd ?l ?o) does not remain true beyond its time limit, we add to each (advance-to o) action the effect (not (iso-maybe-chgd ?l o')) for each o' such that $\delta(o', o)$ exceeds the limit.

Proposition 2 *This encoding is correct, in the sense that any valid plan can be scheduled in a way that respects both observation time stamps and the maximum time lag constraint, to within the threshold T.*
Proof: Transitions are instantaneous. Schedule each transition t_i that generates an observation o exactly in the middle between the earliest and latest time of all observations that are pending when t_i takes place. They cannot be more than T apart, and hence t_i's scheduled time is no more than $T/2$ from $\tau(o)$. Suppose t_i adds a transient proposition, $p(o)$, with time window w (o is the observation that serves as time stamp for p). If t_i is not the transition that generates o, schedule it at the time of the next transition t' that generates an observation o'. Since o and o' are both pending at this point, t_i's scheduled time is no more than $T/2$ from $\tau(o)$. Suppose t_j requires $p(o)$: the scheduled time of any transition t' between t_i and t_j in the sequence which generates an observation o' is no more than $T/2 + w$ later than $\tau(o)$ (as otherwise (advance-to o') would negate $p(o)$). Thus it is possible to schedule t_j no later than $T + w$ after t_i. □

This encoding can be over-constraining, in the sense that in some system models there could be trajectories consistent with time constraints that do not correspond to valid plans. However, this does not happen in our model.

The encoding has some similarity to the use of "envelope" actions in temporal modelling, as suggested by Fox et al. (2004). Indeed, our PDDL2.1 model uses precisely such envelope actions to achieve the same effect.

Experiments and Results

No system that we tried was capable of finding a solution to the problem corresponding to the complete event log. However, it may be argued that an alarm processor should only have to consider windows of time spanning sets of causally related observations. To obtain a set of problem instances for evaluation, we divide the event log up in two ways: (1) by splitting it into "chunks" separated by intervals of at least 1 minute during which no alarm is observed, and (2) by taking a 1 minute time "window" starting from each distinct alarm time stamp. Figure 2 shows the distribution of observations over the "chunk" problems: while most have only

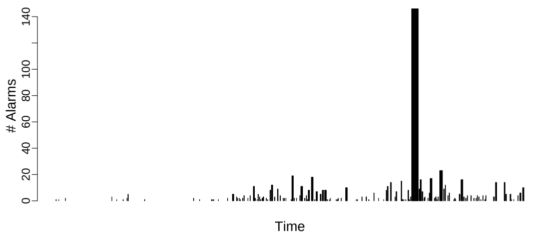


Figure 2: Number of observations in each "chunk" of the log (bar width and spacing is proportional to time). Counts include only the subset of alarm types included in the model.

a few observations, the alarm cascade that results from the fault incident creates a few large spikes. Those are precisely the times when intelligent alarm processing is most needed.

For each subsection of the log, we estimate the number of unexplained alarms that would result without processing by counting all except commands and command acknowledgements. Removing problems where the highest known lower bound matches this estimate leaves 129 problem instances in which there is scope for alarm processing to make a non-trivial improvement. Table 3(a) summarises the number of instances solved within the time limit, grouped by problem size as measured by the number of observations. The systems we compare are:

- The cost optimal A*/LM-Cut planner.
- Gamer (cost optimal; Edelkamp and Kissmann 2008).
- LAMA (Richter and Westphal 2010).[5]
- Greedy best-first search using the cost-ignorant FF heuristic (Fast Downward implementation). To find better plans, we also use two variants: one continues search past the first solution and one runs many repeated searches, randomising decisions that are otherwise made arbitrarily.
- Mp (a heuristically-enhanced SAT-planner, also cost-ignorant; cf. Rintanen, 2010).
- Crikey, a heuristic-search based temporal planner (Coles et al. 2008), run on both the timed (PDDL2.1) and the classical model.
- The SAT-based diagnosis engine (Grastien et al. 2007).

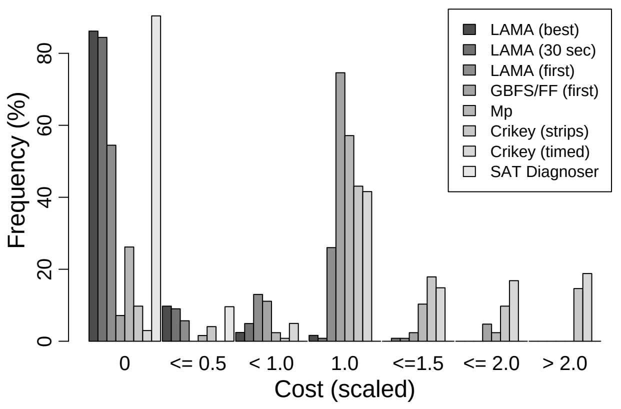
Figure 3(b) shows the distribution of the quality of non-optimal solutions. LAMA, and our two GBFS/FF variants, may find better solutions given more time. In figure 3(b), we separate the quality of the first solution, the best found within 30 seconds, and the best solution found at all.

Results are quite predictable: The cost-optimal planners cannot solve large problems, while the cost-ignorant plan-

[5]LAMA avoids issues related to zero-cost actions in search by uniformly adding 1 to all action costs. To better preserve the distinction between "good" (explained) and "bad" (unexplained) transitions through this transformation, we scale up the cost of bad transitions to 10 in the input to this planner. (The plans it finds are of course evaluated using same costs costs as for all other planners, i.e., zero for good and one for bad transitions.) If we do not apply scaling, LAMA is slightly more efficient, but finds worse (first and best) solutions.

	# Observations						
	1-5	6-10	11-20	21-30	31-50	51-100	>100
# Problems	36	43	35	6	4	4	1
# Solved by							
A*/LM-Cut	36	43	30	0	0	0	0
Gamer	36	43	31	0	0	0	0
LAMA	36	43	35	6	4	2	0
GBFS/FF	36	43	35	6	4	4	1
Mp	36	43	35	6	4	4	1
Crikey (timed)	36	40	19	6	2	1	0
Crikey (strips)	36	43	35	6	4	2	0
Diagnoser	36	43	35	6	4	4	0
# Optimal	36	43	35	1	1	0	0

(a)



(b)

Figure 3: (a) Number of alarm processing problems solved. Problems are grouped by the number of observations, as a measure of size. The last line shows the number of problems for which optimal solution cost is known. (The two optimal planners solve slightly different sets of problems. There are also some cases where a solution found by one of the other systems matches the highest proven lower bound.) (b) Distribution of the cost of solutions found by non-optimal systems, scaled to the interval between the lower bound and the cost of the trivial solution, with no alarm processing (i.e., 0 means equal to the lower bound, 1 means equal to the trivial solution).

ners, though fast, find solutions that are often as bad as using no alarm processing at all, and sometimes even worse than that. LAMA, and continued GBFS, strike a balance between these extremes, finding good solutions quickly most of the time. However, both fail to find improving solutions for the largest instances. Repeated GBFS with randomisation is generally worse, but is the only method to find a solution better than the trivial one for the largest instance. The diagnoser solves all problems but one, though somewhat slowly. It produces minimal-cost solutions, but because it is based on a SAT encoding, these are only optimal w.r.t. a bound (on the number of “steps” between observations), which was fixed in this experiment. However, no other solver finds a better solution for any instance.

We also measure the impact of incomplete knowledge of the initial state, by running the optimal planners on instances with complete initial states (taken from the solutions found by the diagnoser). In this setting, the A*/LM-Cut planner solves an additional 13 problems and Gamer an additional 6, but still neither planner solves any problem with more than 50 observations.

Discussion & Conclusions

We summarise our findings by discussing three questions:

Are the case study problems solved? For our first case study, the combination of two one-sided tests, viz. using an admissible heuristic such as the landmark-cut procedure to detect event loss and running a fast planner on a fault-free model to identify loss-free scenarios, seems to be a viable approach (though it does not decide every problem).

For intelligent alarm processing, existing approaches use either very simple system models (most not even including any notion of time) or lack capability to choose the best among competing explanations. In contrast, formulating it as a discrete event diagnosis problem allows for minimisation of unexplained events over very expressive system models. On the other hand, it is clear that this problem cannot yet be solved efficiently: Planners that deliver solutions quickly find solutions that are often as bad as using no alarm processing at all. The diagnoser produces high quality solutions, but is still too slow to be practical. LAMA makes a valiant attempt to balance fast plan generation and the quality of plans found, and does produce fairly good solutions, even within a 30 second time limit. However, it, and the other iterated search methods we tried, fail to find a solution better than the trivial for the problems with the highest number of alarms. Since those represent precisely the situations when effective alarm reduction is most sorely needed, we cannot call this an acceptable performance overall. It must also be remembered that our model is highly simplified, and covers only a subset of alarm types. To achieve the level of alarm filtering and synthesis that would truly benefit end users we would need a much more sophisticated model, placing a higher computational burden on the system used to solve it.

What can planning technology contribute to discrete event diagnosis? The question we seek to address is if and how the techniques that have been successful in solving planning problems can be effectively brought to bear on the problem of discrete event system diagnosis. To this end, we devised a general reduction of this diagnosis problem to planning, and used it to test a variety of planning approaches.

It is not so easy to identify the “state of the art” in discrete event diagnosis, for several reasons: what are reasonable assumptions and what is a useful diagnosis differs between applications, and quantitative data on the relative performance of different approaches is scarce. Among approaches proposed in the literature for the kind of problem we consider, many rely on expensive off-line preprocesing of the model, making them unusable for our first case study, where the set of components is dynamic, and severely impractical for the second, where the complete system has some 10,000 components. Decentralised diagnosis methods (e.g., Pencolé and Cordier 2005) may be usable for the second case study.

Nevertheless, we have shown that there are planning techniques, such as the use of an admissible heuristic to identify necessary fault events, which have not previously been applied to diagnosis problems and which show enough promise to merit further development. On the other hand, it is also clear that direct reduction to planning is not a universal solution to discrete event diagnosis.

What are the implications for planning research? We believe that our case study problems are interesting and challenging also for planning, since they highlight some issues not commonly encountered in other benchmark domains.

One such issue is the essential requirement of taking into account action costs. It has been noted (e.g., by Richter and Westphal 2010) that in many benchmark domains, planners that ignore action costs frequently find solutions as good as – sometimes even better than – those found by planners that pay attention to cost and try to minimise it, and that planners of the former kind have an advantage in efficiency. Likewise, Sohrabi et al. (2010) report that cost-ignorant planners always find near-optimal solutions for the artificial example problem they consider. The problems we study clearly demonstrate that this phenomenon is an artefact of those particular problem domains, not a universal rule.

Acknowledgements

We thank TransGrid and the LiU UAVTech group for their permission to use the data, and Sylvie Thiébaux for many helpful discussions. This work was supported by ARC project DP0985532. 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

Basile, F.; Chiacchio, P.; and De Tommasi, G. 2003. An efficient approach for online diagnosis of discrete event systems. *IEEE Trans. on Automatic Control* 54(4):748–759.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *AAAI'08*.

Cordier, M., and Thiébaux, S. 1994. Event-based diagnosis for evolutive systems. In *DX'94*, 64–69.

de Kleer, J., and Williams, B. 1989. Diagnosis with behavioral modes. In *IJCAI'89*, 1324–1330.

Dijk, H. 1992. AI-based techniques for alarm handling. *Int'l Journal of Electrical Power & Energy* 14(2–3):131–137.

Doherty, P.; Haslum, P.; Heintz, F.; Merz, T.; Persson, T.; and Wingman, B. 2004. A distributed architecture for intelligent unmanned aerial vehicle experimentation. In *Proc. 7th Int'l Symp. on Distributed Autonomous Robotic Systems*.

Dousson, C. 1996. Alarm driven supervision for telecommunication network: II – on-line chronicle recognition. *Annals of Telecommunications* 51(9-10):501–508.

Edelkamp, S., and Kissmann, P. 2008. GAMER: Bridging planning and general game playing with symbolic search. In *IPC 2008*.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *ECAI'04*, 338–342.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *AAAI'07*.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS'09*.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; Liporace, F.; and Trüg, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of AI Research* 26:453–541.

Kuhn, L.; Price, B.; de Kleer, J.; Do, M.; and Zhou, R. 2008. Pervasive diagnosis: The integration of diagnostic goals into production plans. In *AAAI'08*, 1306–1312.

Larsson, J. 2009. Real-time root cause analysis with multilevel flow models. In *DX'09*.

McIlraith, S. 1994. Toward a theory of diagnosis, testing and repair. In *DX'94*, 185–192.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunications networks. *Artificial Intelligence*.

Prince, W.; Wollenberg, B.; and Bertagnolli, D. 1989. Survey on excessive alarms. *IEEE Trans. on Power Systems* 4(3):950–956.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research* 39:127–177.

Rintanen, J. 2010. Heuristics for planning with SAT. In *CP'10*, 414–428.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Trans. on Control Systems Technology* 4(2):105–124.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *KR'10*.