Solving Power Supply Restoration Problems with Planning via Symbolic Model Checking

Piergiorgio Bertoli[1], **Alessandro Cimatti**[1], **John Slaney**[2] and **Sylvie Thiébaux**[2]

Abstract. The past few years have seen a flurry of new approaches for planning under uncertainty, but their applicability to real-world problems is yet to be established since they have been tested only on toy benchmark problems. To fill this gap, the challenge of solving power supply restoration problems with existing planning tools has recently been issued. This requires the ability to deal with incompletely specified initial conditions, fault conditions, unpredictable action effects, and partial observability in real-time. This paper reports a first response to this nontrivial challenge, using the approach of planning via symbolic model-checking as implemented in the MBP planner. We show how the problem can be encoded in MBP's input language, and report very promising experimental results on a number of significant test cases.

1 Introduction

It has long been recognized that real-world planning requires coping with uncertainty arising from exogenous events, nondeterministic actions, and partial observability. Accordingly, a wide range of approaches for planning under uncertainty have been proposed; see [4, 5, 8, 9, 10] for recent examples. While theoretically well-founded, these approaches have only been tested on very artificial examples (tigers behind doors, bombs in toilets) and, at a practical level, there is no evidence that they will scale up to address interesting problems. Therefore, one of the most significant outstanding tasks for the field is to demonstrate the applicability of these approaches to realistic problems, identify their bottlenecks, and suggest improvements.

Recently, Thiébaux and Cordier made a concrete proposal in this direction [14]. They recast the problem of power supply restoration (PSR) as a benchmark for planning under uncertainty, and issued the challenge of solving PSR problems with existing planning tools. PSR consists in planning actions to reconfigure a faulty power distribution network, with a view to resupplying the customers affected by the faults. Due to sensor and actuator uncertainty, the location of the faulty areas and the current network configuration are only partially observable. This results in a tradeoff between acting to achieve a suitable configuration, and acting (intrusively) for the purpose of acquiring additional information. This tradeoff is typical not only of planning for partially observable domains, but also of diagnosis, repair, and reconfiguration problems [6, 13, 2]. Even relatively small instances of this benchmark stretch the limit of what can be achieved with state of the art general purpose planning approaches.

[1] IRST – Istituto per la Ricerca Scientifica e Tecnologica, Trento, I-38050, Italy. email: {bertoli, cimatti}@irst.itc.it

[2] Computer Sciences Laboratory, The Australian National University, Canberra, ACT 0200, Australia. email: {John.Slaney, Sylvie.Thiebaux}@anu.edu.au

In this paper, we take on the PSR challenge via one such approach: planning via symbolic model-checking, as implemented in the MBP planner [4]. MBP makes aggressive use of binary decision diagrams, and generates conditional plans which are guaranteed to achieve the goal. We show how PSR can be encoded in MBP's input language $\mathcal{AR}$ [7], and present experimental evidence that MBP is able to exploit network topology and solve problems whose complexity is representative of that of real situations in better than real-time. Given the difficulty of the benchmark, these results are beyond the expectation with which we began this investigation.

This paper is organized as follows. Section 2 outlines the PSR benchmark, Section 3 introduces the planning as model checking paradigm, Section 4 discusses the modeling of PSR in $\mathcal{AR}$, Section 5 reports the experimental results, and Section 6 concludes with some remarks about related and future work.

2 The PSR Benchmark

We start with a short statement of the power supply restoration problem given in [14], to which we refer for a more detailed description. As shown in Figure 1, a power distribution system is a network of electric lines connected by switching devices (the small squares in the figure) and fed by circuit-breakers (represented by large squares). Switching devices and circuit-breakers are connected to at most two lines, and have two possible positions: closed or open (open devices, e.g. SD8, are white in the figure). When a circuit-breaker is closed, it supplies power to the network, and this power propagates downstream until an open switching device stops the propagation. The positions of the devices are initially set so that each circuit-breaker feeds a given area of the network (e.g. the area fed by CB4 is boxed in the figure), with no line being fed by more than one circuit-breaker. In the figure, gray and dark are used to distinguish adjacent areas.

Permanent faults can affect one or more lines of the network. When a line is faulty, the circuit-breaker feeding this line opens in order to protect the rest of its area from overloads. As a result, not just the faulty line but the entire area is left without power. The supply restoration problem consists in reconfiguring the network by opening and closing devices so as to electrically isolate the faulty lines and resupply a maximum of non-faulty lines on the lost areas. This must be done within minutes. For instance, suppose that l20 becomes faulty. This leads CB4 to open and the boxed area to be without power. Assuming that the location of the fault and the current network configuration are known, an adequate restoration plan would be the following: open SD16 and SD17 to isolate the faulty line, then close SD15 to have CB7 resupply l19, and finally re-close CB4 to resupply the others. Unfortunately, as we now explain, fault locations and device positions are not always known with certainty, which makes plan construction very difficult.

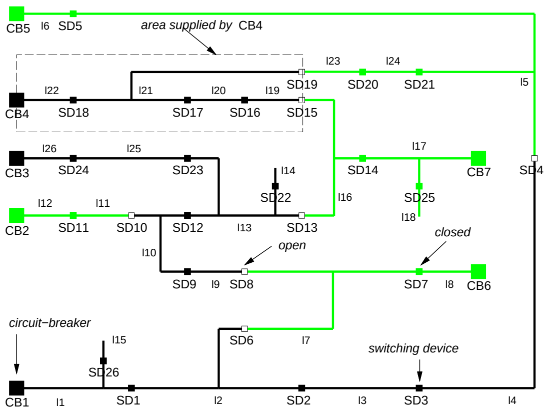


Figure 1. Rural Power Distribution System (from [14])

Opening and closing operations are the only available actions in the benchmark. A first source of uncertainty is that these actions can fail and that failures are not always observable. More specifically, in normal operation the actuator of the prescribed switching device executes the requested action and sends a positive notification. However, the actuator sometimes fails to alter the device's position and sends a negative notification – this is called the "out of order" mode – or fails to alter the position but still sends a positive notification – this is called the "liar" mode. Both abnormal modes are permanent. Clearly, only the out of order mode is directly observable via the notification.

In addition to action-triggered notifications, two types of action-independent sensing information are continuously provided. Firstly, each device is equipped with a position detector which, when in normal mode, indicates the device's current position. Unfortunately, the position detector can be "out of order" for an indeterminate time, during which it does not return any information. Because of the liar actuator mode, the position of an operated device cannot be known with certainty while its position detector remains out of order. It is then difficult to know whether faults have been correctly isolated.

Secondly, each switching device is equipped with a fault detector which senses the presence of faults. In normal operation, as long as the device is fed, its fault detector indicates whether there exists a fault downstream of it on the area. If the device is not fed, its fault detector keeps the status it had when last fed. For example, if l20 is faulty, only the fault detectors of SD17 and SD18 should indicate a fault downstream. Then CB4 should open and the fault information returned by the devices on its area should remain the same until they are fed again. If position detectors were reliable, this scheme would be sufficient to locate single faults, as well as multiple faults on different areas. Unfortunately, fault detectors sometimes do not return any information at all ("out of order mode"), or can lie and return the negation of the correct status ("liar mode"). Both modes are permanent and again only the out of order mode is directly observable.

It follows that several fault location hypotheses are consistent with the observations, and that each of them corresponds to an assumption about the modes (normal or liar) of the detectors. The same applies to positions. The only hope of gaining sufficient information to invalidate a hypothesis is to change the network configuration and compare the new sensing information with the predicted one. This results in a tradeoff between acting to resupply and acting to reduce uncertainty, which is typical of partially observable domains.

3 Planning via Symbolic Model Checking

In this section we briefly outline the Planning via Symbolic Model Checking (PSMC) approach, which we use to solve PSR problems. In PSMC, planning domains are represented as nondeterministic finite state automata (see [4] for formal definitions). Modeling a domain involves the two following steps. Firstly, the state and dynamics of the system (e.g. in PSR, the plant to be restored) are specified. A state of the domain is an assignment to a set of variables (e.g. whether a switch is open or closed, or whether a line is faulty or not). The effects of (possibly indeterministic) actions are modeled by relating a starting state and an action with one or more target states. Secondly, the only information available at execution comes from observation variables that are assigned values depending on the state of the system. For instance, in PSR it is impossible to directly observe whether a line is fed or faulty, but it is possible to observe the fault-status of switches via (possibly untrustworthy) detectors.

In planning under uncertainty and with partial observability, the problem is to reach a given goal condition (e.g. feed every non-faulty line that can be fed) starting from a given, possibly uncertain, initial condition. A solution to the problem is a *strong plan* that, when executed, guarantees that all possible executions starting from any initial state will reach a goal state. In general, the search space of planning under partial observability can be seen as an AND-OR graph in the space of belief states, i.e. sets of states that represent uncertain situations. OR nodes represent choices among possible actions and observations, and AND nodes represent the effect of observations: while an action maps a belief state into a belief state, an observation conveys information by splitting the belief state into smaller belief states, one per possible observation value, which must be all planned for. Plan formation amounts to finding an AND-OR subtree within the search space.

MBP [3] is a general purpose planner that provides for different styles of planning, e.g. conformant, strong and strong cyclic planning, and extended goals, allowing for partial observability and uncertainty. One of its strengths lies in the use of Binary Decision Diagrams (BDDs) to represent and manipulate belief states. BDDs are compact data structures for the representation and manipulation of propositional formulae. In particular, in MBP the effect of actions and observations is efficiently computed by means of symbolic relational operations on BDDs.

4 Modelling PSR in AR

We designed a tool to automatically generate models of PSR problems (including domain descriptions and supply restoration goals) in MBP's input language, starting from a description of the network topology. MBP's input language is an extension of the $\mathcal{AR}$ language [7]; it allows for describing domains where fluents may be inertial or not, where actions may feature preconditions, conditional effects and uncertain effects, and where observations may or may not be triggered by an action. The tool can easily be adapted to other languages with similar features, and is based on the following formal description of the core elements of PSR: network topology, network states, problem dynamics, and observations.

Network topology. The basic elements of the network are circuit-breakers, switching devices, and lines connecting them. By device, we mean either a breaker or a switch. A device d has two sides d^+, d^-. By convention, a breaker b has side b^- attached to the power supply, and b^+ attached to the network. If d^s is a device side, we indicate by $d^{\bar{s}}$ its complementary side. A connection either links two device sides via a line, or consists of a "hanging" line from a device side to earth (see e.g. l15 in Figure 1). Thus, a side-to-side connection is a triple $\langle\delta; l; \delta'\rangle$, and a hanging connection is a pair $\langle\delta; l\rangle$, where δ and δ' are device sides, $\delta \neq \delta'$, and l is a line.

A supply network is a 4-tuple $N = \langle B, S, L, C\rangle$ where B is a set of breakers, S is a set of switches, L is a set of lines and C is a set of connections over B, S and L, satisfying the conditions:

1. Each side of a switch in S occurs in some connection in C.
2. For each breaker $b \in B$, b^+ occurs in some connection in C and b^- in no connection in C.
3. Every line in L occurs in some connection in C.
4. No device side is incident on more than one line: if $\langle\delta; l[; \delta']\rangle$ and $\langle\delta; l'[; \delta'']\rangle$ occur in C then $l = l'$.
5. The connection relation is symmetric, and transitive up to identity: $\langle\delta; l; \delta'\rangle \in C \implies \langle\delta'; l; \delta\rangle \in C$ $\{\langle\delta; l; \delta'\rangle, \langle\delta'; l; \delta''\rangle\} \subseteq C, \delta \neq \delta'' \implies \langle\delta; l; \delta''\rangle \in C$

To describe power propagation we use the notion of a *path*: a sequence of connected devices and lines in the network, starting from a breaker and ending with a line. Formally, a path P of N is a sequence $b^-, b^+, l_1[, d_i^{s_i}, d_i^{\overline{s_i}}, l_i]^*$ where each subsequence $[\delta, l, \delta'] \in P$ corresponds to a side-to-side connection $\langle\delta; l; \delta'\rangle \in C$, and the tail $[\delta, l]$ of P corresponds either to a hanging connection $\langle\delta; l\rangle$ of C, or to a side-to-side connection $\langle\delta, l, \delta'\rangle$ in C. We indicate by $\mathsf{dev}(P)$ the set of devices in P, by $\mathsf{lin}(P)$ the set of lines in P, and by $\mathsf{last}(P)$ the last line of P. A path is acyclic iff it does not contain duplicate lines, cyclic otherwise. A cyclic path P is minimal iff no prefix of P is a cyclic path. We define $\mathsf{AP}(N)$ as the set of all acyclic paths of network N, and $\mathsf{CP}(N)$ as the set of all minimal cyclic paths of N. Reasoning about N amounts to determining the properties of $\mathsf{AP}(N)$ and $\mathsf{CP}(N)$, which are explicitly built by the model-builder tool by a visit of the topology graph.

Network state. The state of a supply network is described by:

- The position of each device, modeled by a dynamic predicate $\texttt{closed}(d)$, defined on $B \cup S$. We say that a path P is active iff it brings power to every line in $\mathsf{lin}(P)$, which it does when every device in P is closed: $\texttt{active}(P) = \forall d \in \mathsf{dev}(P) : \texttt{closed}(d)$. We say that P is active upon closing d iff closing d leads to P being active: $\texttt{active\_upon\_closing}(P, d) = \forall d' \in \mathsf{dev}(P) : ((d' = d) \vee \texttt{closed}(d'))$.
- The permanent modes of the lines (faulty or not), modeled by a predicate $\texttt{faulty}(l)$, statically defined on L.
- The fault-status of each device, i.e., whether there was a fault downstream of the device when it was last fed. It is modeled by a dynamic predicate $\texttt{affected\_when\_last\_fed}(d)$ which is set every time d is part of an active path whose last line is faulty, and is reset when d is fed with no fault downstream.
- The permanent modes of the devices' actuators, fault detectors and position detectors. These are modeled by predicates $\texttt{AC\_correct}(d)$, $\texttt{AC\_liar}(d)$, $\texttt{FD\_liar}(d)$, $\texttt{PD\_correct}(d)$, $\texttt{FD\_correct}(d)$, statically defined on $B \cup S$.

In $\mathcal{AR}$, statically defined predicates, whose value is known in the given problem, and derived predicates, can be compiled away as DEFINEs. For instance, $\texttt{active}(P)$ is defined as a propositional conjunction. Dynamic predicates, and static predicates whose value is not known, are fluents and build the actual state of the domain. Under the current assumptions, every such fluent is inertial.

Dynamics. Two kinds of phenomena may affect the state of the network: (a) user-induced actions on a device d may affect d's position, and (b) faults and power propagation may affect the fault-status of various devices and the positions of breakers. Their effects can be described as follows:

a1 When opening [closing] a device d, if the actuator of d is correct, d opens [resp. closes]. Otherwise, it keeps its current position.
b1 If there exists an active path P whose last line is faulty and $d \in P$, then the fault-status of d is set. If d is a breaker, it opens; otherwise, it keeps its current position.
b2 If d is in an active path, but in no active path ending in a faulty line, then the fault-status of d is reset. d keeps its current position.
b3 If no active path P exists such that $d \in P$, then position and fault-status of d are unchanged.

Several options are possible for modeling the above dynamics. We adopted a modeling style where the combined effects of (a) and (b) are computed as a "one-step" consequence of each user-triggered action. This proved experimentally superior to other "interleaved" modelings where the effects of (a) and (b) are considered in turn.

In our modeling, the description of an action upon a device d_0 considers the effects on every device d as follows:

1. As a consequence of closing d_0, a non-active path P, whose last line is faulty, becomes active, and d belongs to P. In this case the fault-status of d is set. If d is a breaker, it opens; otherwise it keeps its current position. We say that closing d_0 has affected d.
2. d_0 does not affect d and, as a consequence of closing d_0, a non-active path P, whose last device is d, becomes active. The device d_0 is said to have fed d. In this case, the fault-status of d is reset. d's position is unchanged.
3. As a consequence of closing d_0, neither case (1) nor case (2) applies. In this case, closing d_0 has no effect on d.
4. As a consequence of opening d_0, an active path becomes inactive. This does not change the position or fault-status of any device d other than d_0 itself.
5. As a consequence of opening [closing] d_0, d_0 opens [resp. closes] if its actuator is correct, unless d_0 is a breaker whose closing affects itself (in which case it reopens, see (1)).

The definitions above amount to logical statements that can be directly encoded into $\mathcal{AR}$. For instance, a propositional definition for $\texttt{closing\_SD17\_affects\_CB4}$ can be provided as a DEFINE consisting of the propositional expansion of the following instantiation of def.1:

$$\begin{aligned}&\texttt{closing\_SD17\_affects\_CB4} =\\ &\quad \exists P \in \mathsf{AP}(N) :\\ &\qquad ((\texttt{CB4} \in P) \wedge\\ &\qquad \texttt{active\_upon\_closing}(P, \texttt{SD17}) \wedge\\ &\qquad \texttt{faulty}(\mathsf{last}(P)))\end{aligned}$$

Given this, the $\mathcal{AR}$ encoding of actions is a direct translation of defs. 1-5 taking into account every possible cause-effect relationship. For instance, the effect of closing switch SD17 upon breaker CB4's position (see b1) is described as follows:

```
CAUSES  act = close_SD17
        next(closed_CB4) := 0
        IF
           AC_correct_SD17 &
           closing_SD17_affects_CB4;
```

In addition, we must take into account the possibility of situations where breakers feed cyclic paths, or in which devices are fed both ways. In these situations, the direction of the electricity flow cannot uniquely be established (unless additional physical data are modeled); thus, the status of fault sensors is not uniquely determined. Although a deployed system would have to be extended somewhat to model these eventualities, in the benchmark, they must be prevented from arising. This is easily achieved by determining "cycle-causing" and "multiple-feed-causing" conditions for any device, and preconditioning the action of closing of a device to the absence of such conditions. We omit details, for reasons of space.

Observations. Modeling sensing is straightforward, and independent of the modeling of actions. The observation returned by the sensors and actuators of a device depend on their mode and on the actual fault-status and position of the device. For instance, the position detector of CB1 signaling that CB1 is open is captured by the boolean observation:

```
OBSERVE says_open_CB1: PD_correct_CB1 & !closed_CB1;
```

5 Solving PSR Problems with MBP

We used MBP to generate strong plans resupplying every feedable line, for different topologies. The first topology we considered was a simple linear one with a single breaker, making it easy to test scalability by varying the network size and the reliability of lines, sensors and actuators. Then we considered a still "simple" but slightly more complex network allowing for a greater variety of configurations, and experimented by varying the reliability of the lines. Finally, we considered a realistic problem taken from [14], based on the topology of Figure 1. Every experiment was run on a 700 Mhz Pentium III Linux machine with 6 GBytes of RAM, but in no case more than 140 MBytes of RAM were used by MBP.

MBP makes it possible to encode search strategies either by incorporating control knowledge into the action descriptions, or by adding ad-hoc heuristics. To speed up the search, we used the following simple ideas which appear to be generic to the benchmark rather than specific to the topologies we considered: (a) do not open a device which is fed or which has previously been fed, (b) favor closing actions to opening actions. Idea (a) is obvious, given that a fed device cannot be incident on a faulty line. Idea (b) comes from the fact that `open` actions are only useful to isolate faults while `close` actions either lead towards the goal by feeding more lines or give the planner information by unexpectedly refeeding a fault.

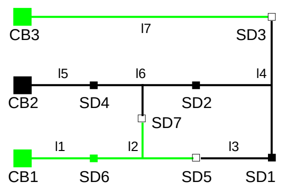
Linear topology. For the linear topology, we first considered the problem of restoring supply given that (a) *exactly* n, and (b) *at most* n of the lines are faulty, starting from a state where all devices are open and all sensors and actuators are reliable, but the locations of the faults are unknown. Figure 3 shows the results for problem (a), considering linear topologies of size varying from 5 to 20 lines, plotted against the exact percentage of faulty lines.

An easy-hard-easy pattern emerges. It is not too hard to see why this might happen: if there is no fault the problem is trivial; if there are faults, the lines downstream of the first fault are essentially irrelevant because they can never be fed, so the more faults there are, the less likely it is that a given line needs to be reasoned about. Problem (b), by contrast, showed no such cost peak: when only the maximum number of faults is known, the problem difficulty is roughly constant for any maximum number of faults greater than zero (search times are always below 7 seconds). This is because the difficulty of problem (b) is roughly the integral of that of problem (a), which for linear networks is dominated by the first few values. In summary, the results of the experiments with the linear topology are intuitively explicable, but it is important to note that MBP achieves these results, as it shows that the planner is able to exploit the structure of the problem.

To experiment with varying reliability of the actuators and sensors, we selected one of the problems above: a linear network of 9 lines, with at most 6 faults. We independently considered that:

1. at least (exactly) n fault detectors are reliable;
2. at least (exactly) n actuators are reliable;
3. at least (exactly) n position detectors are reliable;
4. at most (exactly) n fault detectors are liars;
5. at most (exactly) actuators are liars.

Again, in the initial situation all devices are open; we try to feed lines that are reachable through reliable devices. Fig.4 shows the results; $\mathrm{FD}_{c(=)}, \mathrm{AC}_{c(=)}, \mathrm{PD}_{c(=)}, \mathrm{FD}_{l(=)}, \mathrm{AC}_{l(=)}$ refer to problem 1,2,3,4,5 respectively, in both their versions. As in the case of faulty lines, problems become more constrained and therefore easier if it is known that most devices are unreliable. Of course, combining of faults upon lines with "issues" upon sensors/actuators adds to the complexity of the problems, leading to higher search times than the previous ones.



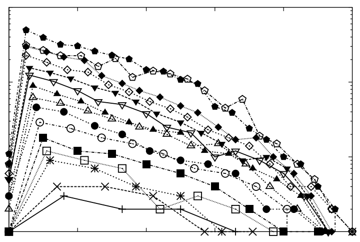
Simple. Here we considered the topology above and an experimental setting similar to those we used upon the linear topology, varying the (maximum/exact) fault percentage F of lines. The results appear very similar to those for lines, and show that MBP is able to effectively exploit a more complex structure. Considering exactly n faults, the easy-hard-easy pattern reaches its top when $F \approx 40\%$, with 1.2 seconds of search time. Considering up to n faults, problems are increasingly hard up to $F \approx 50\%$; the higher search times are around 13 seconds. Of course, higher times are expected due to the fact that the topology is more complex than a simple line, and a strong restoration plan is more complex to build. The results are presented in Fig.5.

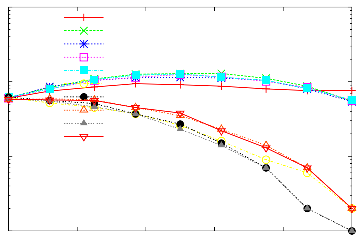
Rural. This is an example taken from [14], see fig. 1. The initial situation is that in the figure, except that CB1 is open; no fault sensor signals a fault, and every line/device/sensor is known to be correct apart from: lines l3 and l15, the fault sensors of SD1, SD2, SD3, SD26, the actuator of SD26, the position detector of SD26. For these devices, no hypothesis is made. Supply restoration must feed every feedable line. Here, this means that, if the actuator of SD26 is correct, then every non-faulty line must be fed; if the actuator of SD26 is not working and l15 is faulty, then neither l15 nor l1 can be fed.

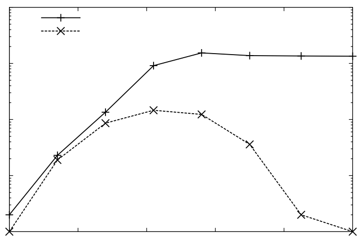
The $\mathcal{AR}$ description is 8.5 megabytes; however, it takes MBP only 30 seconds to parse and to construct the automata for the machine. Once this is done, a plan is found in 1.2 seconds, using approximately 140 megabytes of RAM. Vital to this is the fact that a "good" ordering for the variables is automatically established by MBP; this is achieved by a reusable off-line pre-computing which depends on the model (but not on the problem), and thus has to be performed only once for a given topology. In this case, precomputing takes approximately 30 minutes. Given that the complexity of that particular instance is representative of that of real-life situations (see [14] for details), the generated plan, presented in fig. 2, attests to the feasibility of the approach.

6 Conclusion, Related and Future Work

This paper demonstrates that the planning via symbolic model checking paradigm, as implemented in the MBP planner, is not limited to insignificant artificial examples, but is able to solve realistic supply







```
close_SD4;                  -- feed every line
if says_fault_SD4 then -- l3 and/or l15 faulty
   open_SD3;                -- MBP assumes l15 ok
   open_SD2;                -- and isolates l3
   close_CB5;               -- refeed...
   close_SD26;
   close_SD6;
   if says_fault_SD6 then -- l15 faulty
      open_SD1;             -- MBP assumes l3 ok
      close_SD2;            -- and isolates l15
      close_CB6;            -- refeed...
      if says_fault_SD6 then -- both l3, l15 faulty
         open_SD26;          -- try isolating l15
         open_SD2;           -- isolate l3
         close_CB1;          -- ...and refeed.
         close_SD8;
      else                   -- l3 ok
         open_SD26;          -- try isolating l15
         close_CB1;          -- ...and refeed.
```

Figure 2. Plan for Rural Network problem

restoration benchmark problems. We have developed a systematic representation of the dynamics of the plant as a finite state automaton in MBP's input language, modeled fault conditions, and reformulated power supply restoration as a problem of planning under partial observability. A key difficulty of the planning task is the need to intertwine diagnosis (identifying the causes of the problem) with the search for corrective actions. An experimental evaluation shows that MBP is able to manage some basic plant configurations, and to automatically produce strong plans for problems when different degrees of information are available. The underlying symbolic machinery of Binary Decision Diagrams is able to compactly store and efficiently traverse the automaton.

Although the model-based diagnosis community has investigated similar power supply restoration problems in the context of distribution and transport networks, to our knowledge, MBP is the first general-purpose system able to cope with the presence of uncertainty in such problems. For instance, SyDRe, the reactive supply restoration system in [15] is able to handle the full PSR benchmark including sensor and actuator uncertainty, but is entirely domain-specific. Supply restoration of power transmission systems using a general-purpose diagnosis and planning engine has been studied e.g. in [6], but a crucial difference with the PSR benchmark is that observations and actions are assumed to be reliable. Another work related to ours is the application of the model-based reactive planner Burton to spacecraft engine reconfiguration [16]. Burton's compilation of a transition system into prime implicants is clearly related to the compilation into BDDs performed by MBP, but again Burton does not handle partial observability, the main source of difficulty in PSR.

This work is the first step towards the integration of MBP into a complex real-world domain such as PSR. In the future, we plan to proceed with a more in-depth experimental analysis of the PSR problem. This will require some extensions and improvements to MBP. First, realistic restoration plans must often obey a number of constraints that cannot be expressed as reachability goals; e.g., that some lines must be "safe" throughout the restoration process. These can be expressed by extending temporally extended goals [11] to partially observable domains. Moreover, when dealing with very large networks, trying to construct strong plans considering every possible contingency is overkill (even when feasible). Thus, we intend to investigate the potential for an on-line integration as in [15], interleaving probability-based diagnosis, planning and execution modules. Interleaving planning with action might have a dramatic impact on the search times, by restricting the search to only that part of the belief space which is admissible given the current observations. Further, we intend to identify ways to cut the search space, e.g. by means of user-defined strategies [1], or by automatically-detected heuristics, where promising results have been shown in [12] for conformant planning.

References

[1] F. Bacchus and F. Kabanza, 'Using temporal logic to express search control knowledge for planning', *Art. Int.*, **116**(1-2), (2000).

[2] C. Baral, S. McIlraith, and T. Son, 'Formulating diagnostic problem solving using an action language with narratives and sensing', in *Proc. KR*, (2000).

[3] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, 'MBP: a Model Based Planner', in *Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, (2001).

[4] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso, 'Planning in nondeterministic domains under partial observability', in *Proc. IJCAI*, pp. 473–478, (2001).

[5] B. Bonet and H. Geffner, 'Planning with incomplete information as heuristic search in belief space', in *Proc. AIPS*, pp. 52–61, (2000).

[6] G. Friedrich and W. Nejdl, 'Choosing observations and actions in model-based diagnosis-repair systems', in *Proc. KR*, pp. 489–498, (1992).

[7] E. Giunchiglia, N. Kartha, and V. Lifshitz, 'Representing action: indeterminacy and ramifications', *Art. Int.*, **95**, 409–443, (1997).

[8] E. Hansen and Z. Feng, 'Dynamic programming for POMPDs using a factored state representation', in *Proc. AIPS*, (2000).

[9] F. Kabanza, M. Barbeau, and R. St-Denis, 'Planning control rules for reactive agents', *Artificial Intelligence*, **95**, 67–113, (1997).

[10] S.M. Majercik and M.L. Littman, 'Contingent planning under uncertainty via stochastic satisfiability', in *Proc. AAAI*, pp. 549–556, (1999).

[11] M.Pistore and P.Traverso, 'Planning as model checking for extended goals in non-deterministic domains', in *Proc. IJCAI'01*, (2001).

[12] P.Bertoli and A.Cimatti, 'Improving heuristics for planning as search in belief space', in *Proc. AIPS'02*, (2001).

[13] Y. Sun and D. Weld, 'Beyond simple observation: Planning to diagnose', in *Proc. AAAI*, pp. 182–187, (1993).

[14] S. Thiébaux and M.-O. Cordier, 'Supply restoration in power distribution systems — a benchmark for planning under uncertainty', in *Proc. ECP*, pp. 85–95, (2001).

[15] S. Thiébaux, M.-O. Cordier, O. Jehl, and J.-P. Krivine, 'Supply restoration in power distribution systems — a case study in integrating model-based diagnosis and repair planning', in *Proc. UAI*, pp. 525–532, (1996).

[16] B. Williams and P. Nayak, 'A reactive planner for a model-based executive', in *Proc. IJCAI*, pp. 1178–1185, (1997).