

# Interactive Reconfiguration in Power Supply Restoration

Tarik Hadzic and Henrik Reif Andersen

Department of Innovation, IT University of Copenhagen,  
{tarik,hra}@itu.dk

**Abstract.** Given a configuration of parameters that satisfies a set of constraints, and given external changes that change and fix the value of some parameters making the configuration invalid, the problem of *interactive reconfiguration* is to assist a user to interactively reassign a subset of the parameters to reach a consistent configuration again. In this paper, we present two BDD-based algorithms for solving the problem, one based on a monolithic BDD-representation of the solution space and another using a set of BDDs. We carry out experiments on a set of *power supply restoration* benchmarks and show that the set-of-BDDs algorithm scales much better.

## 1 Introduction

In this paper we look at the problem of *interactive reconfiguration* where an already existing (and valid) configuration of parameters becomes inconsistent due to change of one or more of the parameters forced upon the configuration for external reasons. For example, in power supply distribution, a fault could cause a power distribution line to be shut down and a new configuration of the distribution network must be found. In this situation, our approach is to change a small subset of the parameters in order to restore consistency. Besides the number of changed parameters, other user-specific criteria are also relevant to consider. Therefore, the user should be given control to interactively reassign this subset of variables, thus effectively exploring the trade-offs between different criteria, for example, finding a configuration of the power distribution network that tries to maximize the number of customers regaining electricity without significantly changing the standard network topology.

## 2 Theoretical Background

The knowledge about parameters and rules in a configuration problem is captured as a special kind of CSP model:

**Definition 1.** A configuration model (CP)  $C$  is a triple  $(X, D, F)$  where  $X$  is a set of variables  $\{x_1, \dots, x_n\}$ ,  $D = D_1 \times \dots \times D_n$  is the cartesian product of their finite domains  $D_1, \dots, D_n$  and  $F = \{f_1, \dots, f_m\}$  is a set of propositional formulae over atomic propositions  $x_i = v$ , where  $v \in D_i$ , specifying conditions on the values of the variables.

A *total configuration* is an assignment  $\rho$  of values  $v_1, \dots, v_n$  to each of the variables represented as a set of pairs  $(x_i, v_i)$  such that  $v_i \in D_i$ . A *partial configuration*  $\rho$  is an assignment to a subset of the variables. A *total configuration*  $\rho$  is *valid* if it satisfies all the formulae, i.e.  $\rho \models f_j$  for  $j = 1, \dots, m$ , which we also abbreviate as  $\rho \models F$ . A *partial configuration*  $\rho$  is *valid*, abbreviated as  $\rho \models_p F$ , if it can be extended to a total valid configuration  $\rho' \supseteq \rho$ .

Given a configuration model  $C = (X, D, F)$  and a partial configuration  $\rho$ , *interactive configuration* is the process of assisting a user in interactively reaching a total valid configuration starting from  $\rho$ . The interaction satisfies the user-friendly requirement of *completeness of inference* which demands that at every interaction step, for every unassigned variable  $x$ , and every selectable value  $v_x$ , there is a total configuration satisfying this selection, i.e.  $\exists \rho'. (\rho' \supseteq \rho \cup \{(x, v_x)\} \wedge \rho' \models F)$ . In previous work [1, 2] this functionality was obtained by representing the set of valid configurations  $Sol = \{\rho \mid \rho \models F\}$ , as a Binary Decision Diagram (BDD) [3] through a proper encoding of the finite domains with Boolean variables. It is called the *monolithic* approach, since  $Sol$  is represented as a single BDD. The algorithm facilitating interactive configuration given the already made partial assignment  $\rho$  and solution space  $Sol$  is denoted as  $InCo(Sol, \rho)$  and described in more details in [4, 2].

### 3 Interactive Reconfiguration

For reconfiguration, we model externally forced changes to the current total assignment  $\rho$  as a partial assignment  $\rho_f$  ( $f$  for fixed assignments). The resulting, externally modified configuration is denoted by  $\rho[\rho_f] = \{(x_i, v_i) \mid (x_i, v_i) \in \rho_f \text{ or } (x_i \notin \text{dom}(\rho_f) \text{ and } (x_i, v_i) \in \rho)\}$ . Given the set of variables  $R$  to be unassigned we define  $\rho_1 \uparrow R = \{(x_i, v_i) \in \rho_1 \mid x_i \notin R\}$  read as “ $\rho_1$  release  $R$ ”.

**Definition 2 (Interactive Reconfiguration).** *Given a configuration problem  $C(X, D, F)$ , a starting valid total configuration  $\rho \models F$  and a forced partial assignment  $\rho_f$  such that the updated total configuration  $\rho_1 = \rho[\rho_f]$  is invalid. The reconfiguration problem is to find a (small) release set  $R \subseteq X \setminus \text{dom}(\rho_f)$  such that the partial assignment  $\rho_2 = \rho_1 \uparrow R$  is valid if such a set exists or report that it does not exist.*

The algorithm in figure 1 presents interactive reconfiguration in the monolithic approach.

Sometimes the monolithic approach is not feasible because the intermediate or resulting BDD for representing the solutions  $Sol$  becomes too big. We therefore develop an algorithm based on a *set of BDDs*. There will be a BDD for each of the formulae  $f_j \in F$ . We denote the  $j$ 'th BDD by  $Sol_j$  and the full set of BDDs by  $SSol$ . The algorithm in figure 2 illustrates this approach. A key element in the algorithm, is the incremental computation of the release set (line 2) as presented in figure 3.

```

InRecoMono(Sol,  $\rho$ ,  $\rho_f$ )                                /*  $\rho$  is valid and total */
1:  $\rho_1 \leftarrow \rho[\rho_f]$                                /*  $\rho_1$  is invalid and total */
2: if Sol $\rho_f$  is empty then halt                            /* no solution:  $\rho_f \not\models F$  */
3:  $R \leftarrow \text{ShortestPath}(\text{Sol}^{\rho_f}, \rho, \text{cost})$ 
4:  $\rho_2 \leftarrow \rho_1 \uparrow R$                              /*  $\rho_2$  is valid and partial */
5:  $\rho' \leftarrow \text{InCo}(\text{Sol}, \rho_2)$                    /*  $\rho'$  is valid and total */
6: return  $\rho'$ 

```

**Fig. 1.** The key part of the monolithic algorithm is the *ShortestPath*(*Sol* <sup>$\rho_f$</sup> ,  $\rho$ , *cost*) function (line 3) which computes a release set *R* given the BDD for the full solution space *Sol*. We first restrict *Sol* to *Sol* <sup>$\rho_f$</sup>  as BDD operations. We then find the set of variables corresponding to the path of lowest cost (according to the function *cost*) using a depth-first traversal of the BDD. We assign a positive cost to edges representing choices we want to avoid (such as electricity consumers switched off) and zero cost to all other edges (a similar algorithm is described in [5]).

```

InRecoSoB(SSol,  $\rho$ ,  $\rho_f$ )                                /*  $\rho$  is valid and total */
1:  $\rho_1 \leftarrow \rho[\rho_f]$                                /*  $\rho_1$  is invalid and total */
2:  $R \leftarrow \text{PickReleaseSetSoB}(\text{SSol}, \rho, \rho_f)$ 
3:  $\rho_2 \leftarrow \rho_1 \uparrow R$                              /*  $\rho_2$  is valid and partial */
4:  $\text{RelSol} \leftarrow \bigwedge_{j=1}^m \text{Sol}_j^{\rho_2}$ 
5:  $\rho' \leftarrow \text{InCo}(\text{RelSol}, \rho_2)$                    /*  $\rho'$  is valid and total */
6: return  $\rho'$ 

```

**Fig. 2.** In a precompilation step, *SSol* will be computed. We then find (line 2) a release set *R* in an incremental fashion and compute a single BDD *RelSol* of the relevant part of the solution space to be used for reconfiguration. The BDD *RelSol* is found as a conjunction of the BDDs *Sol*<sub>*j*</sub> <sup>$\rho_2$</sup>  corresponding to the BDDs *Sol*<sub>*j*</sub> restricted with the assignment  $\rho_2$ .

```

PickReleaseSetSoB(SSol,  $\rho$ ,  $\rho_f$ )
1:  $\Delta \leftarrow \text{dom}(\rho_f)$ 
2:  $\rho_1 \leftarrow \rho[\rho_f]$ 
3:  $R \leftarrow \emptyset$ 
4: while not SoBSAT(SSol,  $\rho_1 \uparrow R$ ) do
5:   if  $R \cup \Delta = X$  then
6:     halt /* all variables tried, no solution:  $\rho_f \not\models F$  */
7:    $R \leftarrow \text{next}(R \cup \Delta) \setminus \Delta$ 
8: end
9: return R

```

**Fig. 3.** In each incremental step, the *next*(*Y*) function (line 7) finds from a set of variables *Y* a next larger set of variables to be tried for a release set. The set is checked for being a release set through the satisfiability check performed with the algorithm *SoBSAT*(*SSol*,  $\rho$ ) (line 4) which determines whether there exists a total  $\rho' \supseteq \rho$  fulfilling all the BDDs in *SSol*. The algorithm *SoBSAT* is implemented as a Propositional Constraint Solver that is based on a BDD representation of individual (propositional) constraints, using the learning and conflict-resolution mechanisms of modern SAT solvers [6, 7]. It is implemented on top of the BDD-package Buddy [8].

**Table 1.** We measured the time needed to calculate a release set  $R$ , and to compile a resulting BDD for interactive configuration (fig. 1, fig. 2 - both up to line 4), denoted as  $t, t_1, t_2$  for the monolithic,  $H_1$  and  $H_2$  algorithm respectively. We measured the maximum percentage of defined sinks that can be left powered ( $S, S_1, S_2$ ) as indication of quality of restoration w.r.t. resupplying the maximum number of customers. We also measured the maximum percentage of unaffected lines (unchanged line directions) denoted as  $RDir, RDir_1, RDir_2$ , indicating the restoration quality w.r.t. stability of network topology. All the numbers reported are averaged over 100 seed-based pseudo-random simulations (for Complex-P2 and Complex-P3 only 10 simulations) carried at a Pentium-Xeon machine with 4GB RAM and 1MB L2 Cache, running Linux.

Benchmark	Restoration quality						Avg. RT (sec)		
	$S(\%)$	$S_1(\%)$	$S_2(\%)$	$RDir(\%)$	$RDir_1(\%)$	$RDir_2(\%)$	$t$	$t_1$	$t_2$
Std-diagram	98	96.00	96.00	75.54	75.54	75.54	0.17	0.87	1.31
1-6+22-32	100	99.47	99.47	77.33	77.33	77.33	0.50	0.16	0.25
Complex-P2	100	85.19	97.22	84.17	77.31	84.17	3.88	0.14	0.36
Complex-P3	100	90.00	98.42	91.07	91.07	91.07	132.02	0.12	4.44
1-32	-	91.53	99.00	-	91.82	91.82	-	0.10	0.28
Large	-	93.98	98.73	-	94.89	94.89	-	0.27	1.43
Complex-P1	-	79.26	96.94	-	78.96	96.86	-	0.77	15.58
Complex*	-	86.5	91.92	-	85.52	91.67	-	3.11	12.05

## 4 Experimental Evaluation

For experimental evaluation we use a number of instances from the Power Supply Restoration domain [9, 10]. They encode the part of the power distribution network that contains local *power sources* each of which supply a number of electric *lines*, some of which are connected to *sinks*: transformer stations that consume electricity from the network and deliver it to final consumers. The instances were created by Stuart Henney, Tine Bak, Rene Jensen and Lars Sonne [11, 12] in collaboration with NESAs - the Danish power distributor in the Copenhagen area [13]. All the instances are made available for download at [14]. Structural properties of these instances are reported in [4].

Electric lines can become faulty, for example during bad weather conditions, in which case the power source supplying the line is turned off. This affects the entire area supplied from the source, and the problem is in reconfiguring the network by opening and closing lines, to resupply the maximum number of consumers in the affected areas while addressing a number of other domain specific goals (such as minimizing the change of the standard network topology).

For evaluation purposes, three reconfiguration algorithms were developed, a monolithic-BDD algorithm, and two versions of the set-of-BDD algorithms, based on different unassignment heuristics  $H_1$  and  $H_2$  for implementing the *next* function (fig. 3, line 7). In general, heuristic  $H_1$  unassigns lines powered only from the affected power source, while  $H_2$  additionally unassigns lines powered from unaffected neighbouring power sources (more details in [4]). In each simulation,

we loaded a precalculated valid configuration  $\rho$  (representing operational power configuration), and randomly picked a powered line forcing it off. We then ran the reconfiguration algorithms measuring the number of parameters as shown in Table 1.

The numbers in Table 1 indicate that the set-of-BDDs approach scales dramatically better. The biggest instance where the monolithic approach was applicable was the instance Complex-P3 (28 lines and 19 sinks) with response time of 132.02 seconds, compared to the five times bigger instance Complex (146 lines and 119 sinks) that was handled in 42 times shorter time (3.11 seconds). The high percentage of ressuppliable sinks and unaffected lines (most quality estimates are above 90%) supports the intuition about the locality of external effects in the real world instances (recovery within the 10% of change in network topology).

## Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions.

## References

1. Hadzic, T., Subbarayan, S., Jensen, R.M., Andersen, H.R., Møller, J., Hulgaard, H.: Fast backtrack-free product configuration using a precompiled solution space representation. In: PETO Conference, DTU-tryk (2004) 131–138
2. Subbarayan, S., Jensen, R.M., Hadzic, T., Andersen, H.R., Hulgaard, H., Møller, J.: Comparing two implementations of a complete and backtrack-free interactive configurator. In: CP'04 CSPIA Workshop. (2004) 97–111
3. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* **8** (1986) 677–691
4. Hadzic, T., Andersen, H.R.: Interactive Reconfiguration in Power Supply Restoration. Technical Report ITU-TR-2004-68, IT University of Copenhagen (2005)
5. Subbarayan, S.: Integrating CSP decomposition techniques and BDDs for compiling configuration problems. In: CP-AI-OR Conference. (May 2005)
6. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust SAT-solver. In: Design, Automation, and Test in Europe (DATE '02). (2002) 142–149
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC'01). (2001)
8. Lind-Nielsen, J.: BuDDy - A Binary Decision Diagram Package. <http://sourceforge.net/projects/buddy> (online)
9. Thiébaux, S., Cordier, M.O.: (Supply restoration in power distribution systems – a benchmark for planning under uncertainty)
10. Bertoli, P., Cimatti, A., Slanley, J., Thiébaux, S.: Solving power supply restoration problems with planning via symbolic model checking. In: ECAI'02. (2002)
11. Bak, T., Henney, S.: Power Supply Restoration - a Constraint-based Model for Reconfiguration of 10kv Electrical Distribution Networks. ITU (2004)
12. Sonne, L., Jensen, R.: Power Supply Restoration. Master's thesis, Department of Innovation, IT University of Copenhagen (2005)
13. Nesa. <http://www.nesa.dk> (online)
14. Power Supply Restoration Benchmarks. <http://www.itu.dk/people/tarik/psr> (online)