(N-1) contingency planning in radial distribution networks using genetic algorithms

Alexandre Mendes, Natashia Boland, Patrick Guiney, and Carlos Riveros

***Abstract*—(N-1) contingency planning has been object of study in the area of distribution networks for several decades. Energy distribution companies have to reconnect areas affected by an outage within a very short time, and observe operational constraints, to avoid the possibility of severe financial penalties by regulatory bodies. Distribution networks are often operated with a radial topology, but, ideally, should have more than one route to deliver energy to any node of the network. Switches in the network are opened to create the radial topology used in normal operation, and, in the case of an outage, alternate routes are activated by opening or closing switches located at specific points of the network. Given an outage situation (in our case represented by the disconnection of a single branch), the choice of which switches should change their state is a combinatorial optimisation problem, with a search space of 2^k, where k is the number of switches. Because of the exponential complexity, exact methods are prohibitively time-consuming. This work presents a genetic algorithm that provides a rapid answer to network managers in terms of a switching strategy to reconnect the affected area. The method takes into account the radial topology of the power flow and the operational limits of voltage and cable load. Computational tests were conducted on a real network with 96 buses and 16 switches, located within the operational area of Energy Australia. This paper describes the genetic algorithm in detail, presents thorough computational tests, and a complete contingency plan for the test network.**

I. Introduction

THIS paper describes the application of a genetic algorithm to the problem of network reconfiguration via switching operations. Network reconfiguration arises in a number of situations faced by electricity distribution companies. Two of the most common applications are power loss minimization coupled with load balancing [1][2][3][4], and power outage restoration [5][6][7], but there are several others, as pointed to in reference [8]. This study deals with the problem of network reconfiguration for power restoration in an (n-1) contingency scenario – in our case, represented by a single cable fault.

The distribution network considered in this study can be represented by sets of generators, feeders, buses, loads, switches and branches. Distribution networks in Energy Australia's concession area operate a radial topology. Such radial networks, as the name says, have no cycles and each load is served by only one feeder. When designing a distribution network, though, companies create robustness by adding excess connectivity, allowing power to follow different paths to reach the same customer, if needed. Together with the excess connectivity, switches are positioned in strategic points of the network and can be set either as *closed* or *opened*. Specific switching configurations will create the radial topology of the power flow, assigning each load to a generator or feeder. Redundancy in the distribution network is particularly critical in situations of (n-1) contingency, in which a cable becomes faulty, affecting the supply for all customers located after it. In this case, an alternate path to supply power to the affected area can be activated by a series of switching states changes.

The problem of finding alternate routes for the power supply, given an outage scenario, is very complex and has been studied for several decades. Alternate routes are determined by searching the solution space of switches states. That is, if there are k switches present in the network, the search space has a size 2^k, corresponding to each switch being either *closed* or *open*. Because of the exponential nature of the search space, distribution networks with a few dozen switches or more will render exhaustive search approaches prohibitively time-consuming. In that case, the use of heuristics becomes justified. Furthermore, the need to solve power flow equations for each possible switched state to determine its feasibility and quality particularly motivates the use of metaheuristics such as genetic algorithms [9][10], which can optimise over a "black box" power flow solver. This convenience is not available in traditional mathematical programming based optimisation methods.

In this study we implemented a genetic algorithm that, given the loss of a specific branch, searches though the space of possible switching configurations for a solution that restores power to the affected area without violating specified operational limits, i.e. provides a contingency plan based on a switching strategy. Tests were carried out in a real network with 96 buses, 97 branches, 2 generators and 16 switches. Faults on each cable (i.e. branch) were simulated and a contingency strategy for the entire network was produced with the union of the contingency plans for each individual fault.

This paper is organized as follows. In Section II we describe the reconfiguration problem, the operational constraints and the test network. The solution approach via genetic algorithms is then thoroughly described in Section III. Section IV shows the computational tests, which determine how the genetic

A. Mendes is with the School of Electrical Engineering and Computer Science, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, 2308, Australia. e-mail: (see http://www.cs.newcastle.edu.au/~mendes).

N. Boland is with the School of Mathematical and Physical Sciences, Faculty of Science and Information Technology, The University of Newcastle, Callaghan, NSW, 2308, Australia.

P. Guiney is with Energy Australia, Wallsend, NSW, 2287, Australia.

C. Riveros is with the School of Electrical Engineering and Computer Science, Faculty of Engineering and Built Environment, The University of Newcastle, Callaghan, NSW, 2308, Australia.

Manuscript received April 19, 2005; revised January 11, 2007.

978-1-4577-0487-1/10/$26.00 ©2010 IEEE

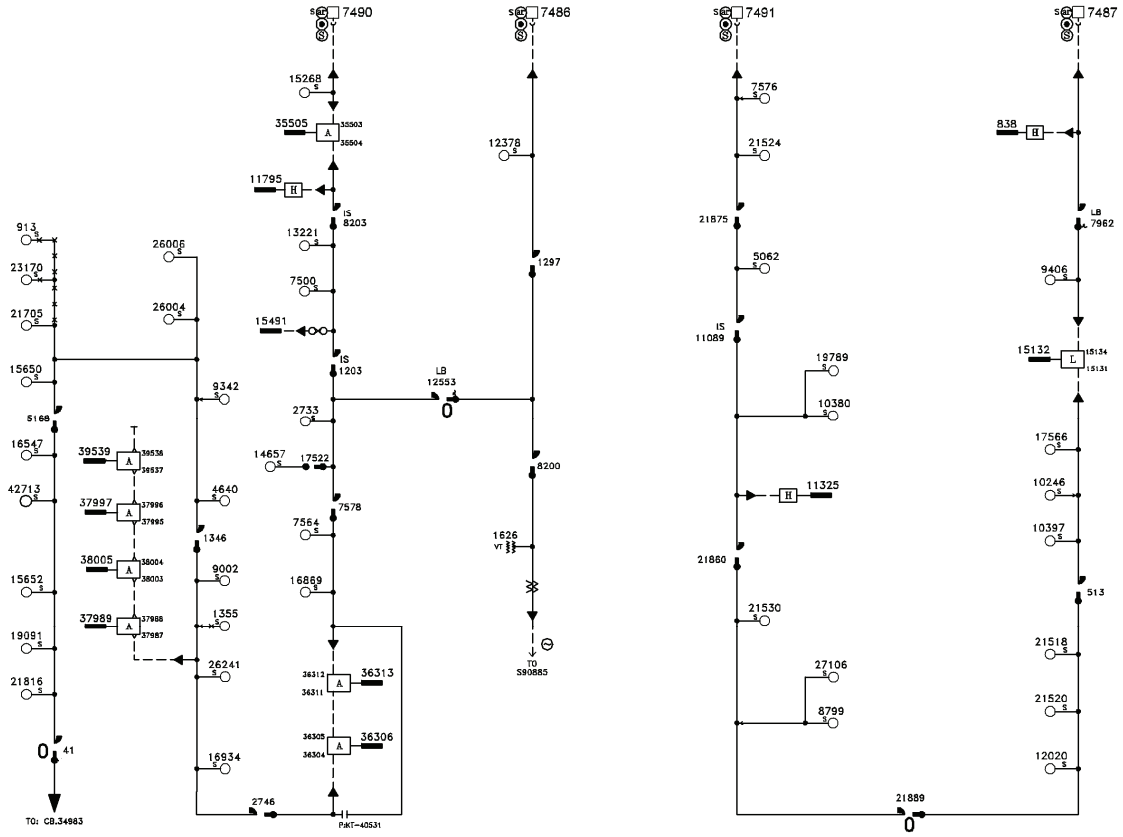

Fig. 1. Diagram of the test network used in this study. There are four feeders located on the top of the diagram and represented by the numbers 7490, 7486, 7491 and 7487. Three switches are opened in the original configuration (indicated by an 'O'): 21889, 12553 and 43398. The other 13 switches are closed. This configuration creates the radial topology of the distribution network. Notice that, for example, if switch 21889 was closed, there would be a cycle connecting feeders 7491 and 7487. Switching reconfiguration becomes clear as a way of redirecting power to regions affected by outages. For instance, suppose that the branch between 17566 and 10246 (middle-right of the diagram) becomes faulty. In this case, the entire region between load 10246 and the switch 21889 will suffer an outage. A simple reconfiguration strategy would be to close switch 21889 – however, it is still necessary to run a power flow simulation model to check whether or not feeder 7491 becomes overloaded because of the expansion of its supply area.

algorithm parameters affect its performance. In Section V the results are assessed in terms of reliability and limitations, mostly from the practical standpoint, i.e. their use by distribution manager professionals in real outage recovery scenarios. Finally, in Section VI we present the conclusions and some future research paths related to this study.

II. The Problem of Network Reconfiguration in Practice

Distribution networks generally operate a radial topology, with power flowing from the substation/generator, located at the root node of the tree, towards the leaf nodes and supplying the intermediate loads. However, when we analyse the diagram of a real network, the radial topology only exists because of switches that are opened in specific locations, otherwise there would be cycles in the power flow. As mentioned before, the existence of cycles in the physical network topology allows alternate routes for the power supply and increases the robustness of the energy distribution to outages.

In Figure 1 we show the diagram of the test network used in this work. It has four feeders on the top of the figure, and 16 switches represented by an open switch symbol. Note that all switches are closed, except the three marked with 'O', and that most regions of the network have alternate supply routes available. For instance, any load in the rightmost branch (feeder 7487) is reachable from feeder 7491 if switch 21889 (bottom-right) is closed. In such a small network, visual inspection suffices to determine the switching changes required, and thus the network manager simply needs to run a power flow model to verify if the new configuration will be within operational limits. However, this situation changes dramatically when larger networks are considered. In that case, visual inspection is painstakingly slow and inaccurate, requiring some degree of automation in the decision making process.

In our implementation, the genetic algorithm receives as input the physical network, as shown in Figure 1, including the current state of the switches. Then, a given branch is removed from the network (i.e. all references to it in the network model are removed). This simulates an outage in which the branch containing the fault has been identified and isolated from the network. Finally, the genetic algorithm does the search and communicates the best solution (switching state) found at the end. In this problem, the majority of the switching states will be infeasible, because either they leave sections of the network still disconnected or they introduce cycles; or because the load on the cables or the voltage limits are not within the operational bounds. The criteria we use to define a switching

state to be *feasible* are:

- *Connection/topology*: All sections of the network are connected and the network has radial topology.
- *Load*: The load on any cable does not exceed its operational limit.
- *Voltage*: The voltage in any section of the network lies between 0.95 and 1.05 (measured as per-unit).

It is worth mentioning that because the test network is 11kV, distances are relatively small and loads are predominantly active, there is no need to account for variations in the angle, but that restriction could be easily added to the model in the future.

If the solution (switching state) is infeasible, the objective function is penalized proportionally to the degree of infeasibility, as given in the penalty function $P(s)$ defined below. Otherwise, the algorithm looks at two characteristics that a good quality switching state should have:

- *Low number of switch changes*: solutions which require fewer switch changes are better.
- *Voltage deviation*: voltage should be as close as possible to 1.0 in all sections of the network.

The first criterion is important from an operational standpoint because most switches in the network are manually operated and require a crew to drive to their exact location to perform a change. If there are too many changes to be made, it might take too long to restore power to the affected region. The second criterion is also important, as legislation protects consumers from being supplied electricity that is outside certain voltage boundaries, and which could either damage equipment or make them not work properly. These two criteria are put together with the infeasibility penalty in the objective function as:

$$obj(s) = P(s) + swt_{changes}(s) + \sum_{i=1}^{n_{buses}} \left| v_{dev}(i,s) \right| \quad (1)$$

$$P(s) = \begin{cases} \infty, \text{ if the network given by } s \text{ contains a cycle;} \\ M_1 * n_{buses_{out}}(s) + \\ M_2 * \left[n_{volt_{out}}(s) + n_{load_{out}}(s) \right], \text{ otherwise.} \end{cases} \quad (2)$$

Where:

- $s \rightarrow$ a solution (i.e. a switching configuration);
- $obj(s) \rightarrow$ the objective function to be minimized;
- $P(s) \rightarrow$ the infeasibility penalty of solution s;
- $swt_{changes}(s) \rightarrow$ the number of switches that change state in solution s;
- $n_{buses} \rightarrow$ the number of buses in the network;
- $v_{dev}(i,s) \rightarrow$ the voltage deviation from 1.0 at bus i in solution s;
- $n_{buses_{out}}(s) \rightarrow$ the number of buses still without power supply in s (positive if the network is disconnected);
- $n_{volt_{out}}(s) \rightarrow$ the number of buses with voltage outside the operational boundaries in s;
- $n_{load_{out}}(s) \rightarrow$ the number of branches with load above the operational limit in s;
- $M_1, M_2 \rightarrow$ large numbers ($M_2 < M_1 \ll \infty$).

For a given failure in the network, the genetic algorithm will test different switch configurations (switching states) in its search for the optimum solution. Some of these intermediate configurations will have buses or branches not fulfilling the feasibility criteria above. The only feasibility condition being penalised with an infinite penalty value is the radial topology condition; the purpose of the two large constants M_1, M_2 is to allow the differentiation between distinct "degrees of unfeasibility", penalising disconnected buses (M_1) more than sections exceeding the operational limits (M_2).

The power flow model calculation is critical to any power distribution-related problem, and this work is no exception. This calculation needs to be reliable and fast. For this application in particular, we decided to use the Matlab package MATPOWER [11], which has a 10-year development history and has appeared in hundreds of scientific publications in the last few years. Tests with the network used in this study indicate no difference between the results from MATPOWER and the well-known ASPEN Power Flow[1] (the licensed software currently used by Energy Australia in its daily operations), apart from rounding errors.

III. Genetic Algorithm approach

This study proposes a genetic algorithm-based search method to find a switching strategy for power restoration after a branch of the network is lost. Genetic algorithms are population-based search methods that use analogies of the Evolution Theory to find high quality solutions for complex computational problems. The method starts with a population of low quality solutions, usually randomly generated, and then 'evolves' this population via genetic operators, i.e. crossover, mutation and selection, towards better quality individuals, corresponding to solutions with better objective function values. The genetic algorithm used in this study is described next.

A. Pseudocode

The genetic algorithm has a standard structure. The first part of the method creates an initial random population of solutions. Then, in the main loop section, solutions are created via crossover and mutation, and then inserted (or not, according to an acceptance criteria) back into the population, replacing one of their "parent" solutions if accepted. This process continues until a convergence checking procedure detects that the population lost its diversity and no new individuals are being accepted for insertion. When that happens, the population is reset by replacing all individuals, except the currently best one, by random individuals. The main loop continues until a user-specified time limit is reached. Next we will describe the main elements of the genetic algorithm.

B. Fitness function

The fitness function will verify whether or not a solution is feasible, and measure its quality according to Equation 1. Because the genetic algorithm aims at maximizing the fitness, but the objective function is to minimize Equation 1, the fitness

[1] http://www.aspeninc.com/aspen/index.php

```
Method GeneticAlgorithm
begin
do
   initializePopulation(pop);
   do % main loop
      updatePopStructure(pop);
      while(populationNotConverged(pop))
         for j = 1 to num_ind * xover_rate % generation loop
            newSolution = generateOffspring(pop);
            newSolution = mutate(mut_rate, newSolution);
            acceptNewSolution(pop, newSolution);
         endFor
         checkPopulationConvergence(pop);
      endWhile
      restartPopulation(pop);
   while(cpuTime < limit)
end
```

Fig. 2. Pseudo-code of the genetic algorithm implemented. Initially, a population of random switching strategies is created; and then the algorithm enters the main loop. The main loop creates $num_{ind} * xover_{rate}$ new individuals (switching strategies) in each generation, which can either be accepted or not into the population. For every generation loop completed, the algorithm verifies whether the population has converged. Population convergence triggers a restart procedure, where all individuals are replaced by randomly generated ones, except for the currently best solution. This process is repeated until a time limit is reached.

of an individual becomes simply the inverse of the objective function. There is no need to consider the case $obj(s) = 0$ because either the solution corresponds to a new switched state, so $swt_{changes}(s)$ will be positive, or the solution is the original state, which is known to have positive voltage deviation.

C. Representation and initialization

Solutions for the problem of switching reconfigurations have a binary representation. That is, an array of bits of size n will represent the states of the n switches, with 0 and 1 indicating opened and closed, respectively. The population initialization occurs within the procedure **initializePopulation**, which creates a population of random solutions by assigning values 0 or 1 to each switch, uniformly at random. The probability is 20% for any given switch to be open, and 80% for it to be closed, which very closely matches the proportions of open and closed switches in the original switching state. These proportions aim at reducing the likelihood of creating solutions with cycles or disconnected sections.

D. Population structure

In our implementation, populations have a structure that organizes individuals according to their quality and constrains how parent solutions are chosen to produce new solutions. The structure follows a ternary tree as seen in Figure 3.

The population structure is enforced by the **updatePopStructure** method, which performs a sorting on the tree. The sorting procedure simply compares each node with its children, and whenever a child has a better fitness than its parent node, they swap solutions. At the end of the sorting, the best solution of the population will be positioned at the root node, and the worst solutions will tend to be placed in the lower layers. This structure is interesting from the evolutionary point of view, as it induces parallel sub-populations within each branch. That is, since crossover can only occur between solutions in nodes connected by an edge, the three branches that emanate from the root node will evolve almost independently, as three separate populations, receiving the influence only from the current best individual at the root node. This reduces premature convergence, and improves the exploration of the search space. This structure has been compared to non-structured populations and to other types of structured populations, and has been used in other combinatorial optimisation problems [12][13]. It has consistently outperformed other approaches, providing a good trade-off between population size and convergence rate.

E. Recombination – selection, crossover and mutation

Selection of parents follows the approach described above, dictated by population structure. Whenever a new solution is to be created, an internal node is chosen uniformly at random, and one of its three child nodes is selected as the second parent.

The implementation in this study uses the Uniform Crossover (UX) [9], where the value of each switch state in the child solution is chosen uniformly at random from one of its parents. Therefore, if both parents have the same state for a specific switch (either 0 or 1), the child will inherit that state. If each parent has a different value, then the value inherited can be 0 or 1, with equal probability.

As the representation is an array of bits, the logical choice for mutation is the bit-swap [9]. If a solution is selected to go through mutation (according to a probability mut_{rate}), a switch is chosen uniformly at random, and its state is swapped, either $0 \rightarrow 1$ or $1 \rightarrow 0$.

F. Acceptance policy, convergence checking, and population restart

When a new solution is created, it might be inserted into the population or not, depending on its quality. In our implementation, the acceptance policy determines that new solutions are accepted if their objective function value is better than at least one of its parents. In this case, the new solution will replace the worst parent. This policy guarantees that the average fitness of the population increases over time, up to a

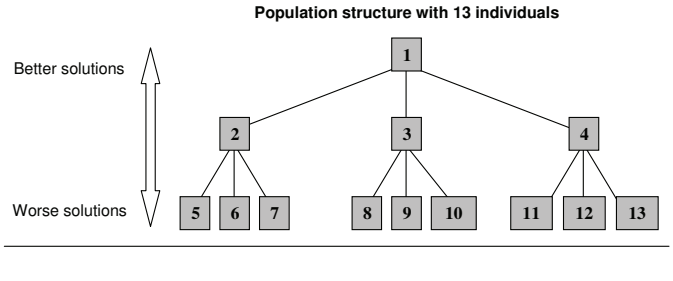

Fig. 3. Diagram of the population structure based on a ternary tree. The figure depicts a population with 13 individuals, corresponding to a tree with three levels. Better solutions are positioned in the upper layers of the tree, whereas worse solution go to the bottom layers. Recombination is restricted to pairs of parents chosen from nodes connected by an edge.

point where all individuals have very good fitness values, and are probably very similar.

To avoid wasting CPU time when the population converges to individuals that are too similar, and thus new incumbent solutions are less likely to be found, a convergence checking criteria is used. The criteria checks whether in the last generation, any of the $num_{ind} * xover_{rate}$ new individuals created was accepted for insertion in the population (num_{ind} represents the number of individuals in the population; $xover_{rate}$ is the crossover rate of the population). If no individuals created were accepted, then it is fair to assume that the population has evolved for a reasonable number of generations, and is composed of highly similar solutions, making any improvement very unlikely.

When the convergence checking criteria is satisfied, it triggers a population restart. That is, all individuals except the best one are replaced by randomly generated solutions and the evolutionary process resumes.

IV. Computational tests

In this section we will detail the tests and results for the test network. The genetic algorithm was run with several alternative parameter settings, which aimed at examining the influence of the parameters on the quality of the solutions obtained. The parameters tested were:

- *Size of population* (num_{ind}): 13, 40 and 121, corresponding to ternary trees with 3, 4 and 5 levels.
- *Crossover rate* ($xover_{rate}$): 1.0, 2.0 and 3.0.
- *Mutation rate* (mut_{rate}): 0.1, 0.3 and 0.5, corresponding to 10%, 30% and 50% of the new individuals going through one bit-swap mutation.
- *CPU time limit*: 5, 10 and 30 seconds per removed branch.

For each run, all possible failures in the network are tested by removing the corresponding branch and searching for a feasible solution using the given parameters and within the CPU time limit. Branches are labelled either *non-critical* or *critical* if the algorithm does (or does not) find a feasible solution when that particular branch has been excluded from the network (i.e. simulating the failure of the branch). The performance for each parameter set is evaluated by the counting the number of non-critical branches. The higher the number of non-critical branches, the better the method is in finding alternate feasible network reconfigurations.

All tests were conducted in a PC computer, with an Intel Centrino Duo 3.0GHz processor and 2Gb RAM; and to produce statistically stronger results, each configuration was tested 10 times. The parameter with the strongest impact on the performance of the genetic algorithm was the CPU time. Although still important, num_{ind}, $xover_{rate}$ and mut_{rate} had a much smaller impact. Results are summarized in Table I, which shows the worst, average and best number of non-critical branches for each parameter set configuration. As expected, results improve quickly as more CPU time is given to the algorithm, going from averages around 46-47 non-critical branches when the GA runs for 5 seconds per removed branch, up to 58-59 non-critical branches for 30 seconds. The best results were obtained with 40 individuals, $xover_{rate}$ = 1.0 and mut_{rate} = 0.3; and with 121 individuals, $xover_{rate}$ and mut_{rate} = 1.0/0.3, 3.0/0.1 and 3.0/0.3, in a CPU time of 30 seconds. With those configurations the genetic algorithm obtained 59 non-critical branches in all ten runs.

Note that of the 97 branches in the network, it is easily calculated that 34 of them will disconnect the network, irrespective of switching state. Although not provable, of the remaining 63, it is likely that the 4 other critical branches cannot supply load within operational limits for any switching state, and that the best switching states found for the 59 non-critical branches yields the optimal contingency plan for the network. The value of 59 non-critical branches was reached at least once in all tests with 30 seconds in Table I, no matter the values of the other parameters.

In Figure 4, we show the diagram of the contingency plan for the test network with 59 non-critical branches. The two rectangles on the top and bottom of the figure represent generators; hexagons are switches (solid/dashed borders indicate closed/open switches in the original network configuration, respectively); and ellipses indicate buses. Critical branches are painted in red; non-critical branches are in black. Branches connected to the three opened switches are dashed, to indicate that no power is flowing through them in the original configuration. All non-critical branches have their switching strategies indicated next to them, along with the final state of each switch, either open (0) or close (1). All critical branches have either no alternate physical path to reconnect them, or their reconnection will violate the operational limits imposed by Energy Australia.

The four feeders present in Figure 1 are indicated by the arrows in Figure 4. Feeder 1 is the longest one, extending from *Generator A* to bus 59 and towards the right and bottom of the figure, until it reaches *Generator B*. Notice, though, that Generator B is disconnected from the rest of the network, as switch P41 is open. All non-critical branches in this feeder require at least the closing of switch P41 to redirect power. Interestingly enough, though, is that the only branches that clearly require the closing of P41 are those located between bus 27 and generator B. For those, closing switch S12553F alone will not suffice; the only way to redirect power to the affected area is through P41. Additional switch changes in those branches are due to branch overloading and voltage bounding constraints.

Feeder number 2 is very short, extending from *Generator A* to bus 93 and towards bus 65. It is connected to feeder 1 by switch S12553F and has a critical section with one switch and three buses, extending from bus 61. All branches located before that bus are non-critical, requiring the closing of switch S12553F to redirect power, and a few other operations to improve operational parameters.

Finally, feeders 3 and 4 constitute a single distribution loop with an open switch (S21889C) to produce the radial topology. All switching operations in these two feeders require the closing of S21889C (except for the two branches connected to the switch itself). In addition, the closing of P41 is present in all switching strategies for these two feeders again due to branch overloading constraints.

TABLE I

PARAMETER TESTING FOR CPU TIME = 5, 10 AND 30 SECONDS PER REMOVED BRANCH. FOR EACH *CPU time*, WE TESTED DIFFERENT VALUES OF *number of individuals*, $xover_{rate}$ AND mut_{rate}. THE TABLE DEPICTS THE WORST, AVERAGE AND BEST VALUES OBTAINED IN 10 RUNS FOR *number of non-critical branches*. THE AVERAGES VALUE OBTAINED FOR EACH CONFIGURATION OF CPU TIME AND NUMBER OF INDIVIDUALS IS SHOWN ON THE RIGHTMOST COLUMN OF THE TABLE. NOTICE THE IMPROVEMENT OF THE SOLUTIONS AS CPU TIME INCREASES, STARTING AT AN AVERAGE OF 46-47 NON-CRITICAL BRANCHES FOR 5 SECONDS; 54-55 IN 10 SECONDS; AND 58-59 IN 30 SECONDS. THE BEST CONFIGURATIONS WERE FOR 40 INDIVIDUALS, $xover_{rate}$ = 1.0 AND mut_{rate} = 0.3; AND 121 INDIVIDUALS, $xover_{rate}$ = 1.0/3.0 AND mut_{rate} = 0.3. ALSO, IT IS WORTH MENTIONING THAT FOR 30 SECONDS CPU TIME, THE METHOD ALWAYS REACHES A SOLUTION WITH 59 NON-CRITICAL BRANCHES, NO MATTER THE VALUES OF $xover_{rate}$ AND mut_{rate}.

CPU time: 5 seconds

13 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 13 indiv. / 5 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	42	46.4	50	41	47.0	51	39	44.9	50	40.8	46.2	50.9
2.0	41	46.1	50	42	46.8	51	41	45.7	53			
3.0	36	44.2	50	40	45.5	49	45	49.5	54			

40 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 40 indiv. / 5 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	42	48.2	53	42	46.0	51	39	46.8	53	42.4	47.3	52.1
2.0	42	46.0	51	46	48.7	52	44	47.9	52			
3.0	42	47.7	52	43	46.1	52	42	48.6	53			

121 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 121 indiv. / 5 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	44	47.7	51	42	47.9	52	47	50.5	54	43.0	47.9	52.2
2.0	42	47.1	51	47	50.0	55	37	42.0	47			
3.0	41	48.0	53	45	49.1	54	42	48.9	53			

CPU time: 10 seconds

13 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 13 indiv. / 10 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	54	55.5	58	53	55.1	58	53	55.1	58	51.7	54.6	57.4
2.0	50	53.9	57	50	53.9	56	52	55.2	58			
3.0	51	54.0	57	50	54.7	57	52	54.3	58			

40 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 40 indiv. / 10 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	53	54.8	56	50	54.7	58	50	55.4	58	51.9	55.2	57.4
2.0	51	54.6	57	53	56.1	58	54	55.8	58			
3.0	52	55.0	57	52	55.1	58	52	55.2	57			

121 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 121 indiv. / 10 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	52	54.9	58	54	55.8	58	53	56.0	58	51.8	55.2	57.9
2.0	52	55.0	57	54	56.4	58	50	53.5	56			
3.0	49	55.6	59	53	56.1	58	49	53.5	59			

CPU time: 30 seconds

13 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 13 indiv. / 30 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	57	58.6	59	56	58.3	59	58	58.8	59	57.3	58.6	59.0
2.0	56	58.4	59	58	58.7	59	58	58.7	59			
3.0	57	58.3	59	58	58.9	59	58	58.6	59			

40 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 40 indiv. / 30 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	58	58.8	59	**59**	**59.0**	**59**	58	58.6	59	57.8	58.7	59.0
2.0	57	58.5	59	57	58.5	59	57	58.4	59			
3.0	58	58.7	59	58	58.8	59	58	58.9	59			

121 individuals

$mut_{rate} \rightarrow$	0.1			0.3			0.5			Average 121 indiv. / 30 sec.		
$xover_{rate} \downarrow$	worst	average	best	worst	average	best	worst	average	best	worst	average	best
1.0	58	58.7	59	**59**	**59.0**	**59**	58	58.9	59	58.2	58.8	59.0
2.0	58	58.8	59	58	58.8	59	58	58.7	59			
3.0	**59**	**59.0**	**59**	**59**	**59.0**	**59**	57	58.5	59			

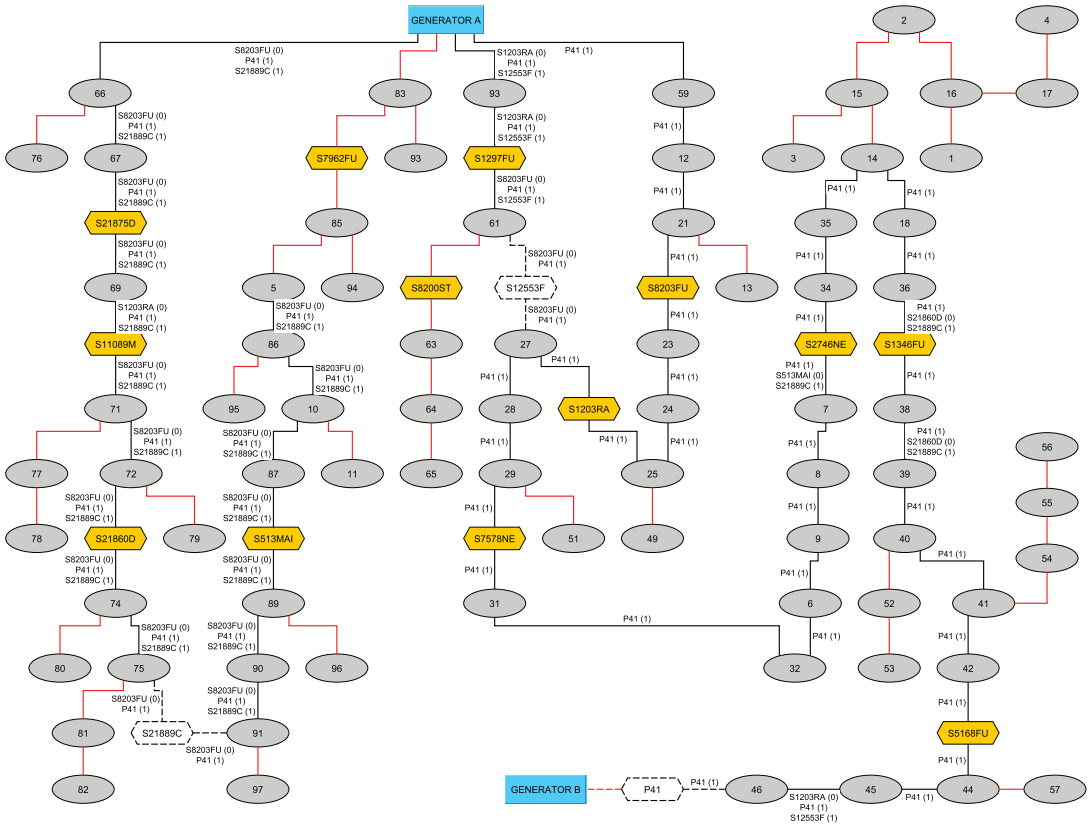

Fig. 4. Diagram of the network showing critical (in red) and non-critical (in black) branches. Next to each non-critical branch, we present the switching strategy to restore power supply, along with switches IDs and final states (0 = open; 1 = closed). The two rectangles represent generators and hexagons are switches. Dashed hexagons represent open switches in the original network configuration. The feeders numbered 1-4 correspond to the feeders in Figure 1.

V. Discussion

Operationally, the development of effective switching strategies is essential both for optimisation of the normal system arrangement and the optimal rearrangement of the electricity network under (n-1) contingency conditions. A major indicator of utility performance is supply reliability, which is in turn determined by the fault rates of the network and the restoration times for supply. The development of optimal switching strategies which provide supply conditions within acceptable limits while restoring supply in the minimum number of switchings (reducing outage times) is paramount.

For network planners, this study has the potential to provide quasi-optimal switching arrangements for a multiply-interconnected distribution network. It eliminates the need for existing *ad-hoc* analysis methods that use a trial and error approach to obtaining alternative switching strategies. For a large distribution network (e.g. Energy Australia has of the order of 2,000 distribution feeders) this will substantially reduce the time that network planners will be required to spend manually carrying out load flow studies to determine restoration strategies for (n-1) contingency conditions.

Other evident future outcomes derived from this analysis are the identification of network capacity constraints; and remediation alternatives based on network augmentation or demand reduction projects. Further development of this algorithm could attempt to incorporate optimised augmentation strategies into its design. In a simplified sense this could involve the identification of point-to-point augmentation opportunities for network planners. In a more advanced scheme this could incorporate geographic data from a utilities GIS system to identify optimised geographic paths for augmentation.

Typically, network planning aimed at evaluating risk and augmenting the distribution network utilises a switched (N-1) criterion, in which load has to be restored within a specific time frame. This criterion is often evaluated at peak conditions on all feeders; a situation that does not necessarily reflect the status of a real network, where fluctuation of loads on different feeders will not necessarily match their peak loads in time. This creates the possibility of a stochastic analysis of peak loading conditions on the network, correlating the transfer capability with this load fluctuation. This would require contingency analysis to be performed under a variety of load scenarios.

For control room operators, this algorithm will provide them with alternate optimised arrangements to quickly restore supply in the event of a contingency, reducing outage times and improving network reliability. To be utilised in this manner the algorithm needs to become integrated into the existing Distribution Management System (DMS) of the utility.

A DMS usually utilises an associated network model which provides the network data for monitoring and analysis. The software will have to be modified to recognise the edges of a feeder and the edges of its interconnection area to minimise

the analysis size and relevance of the solution. This would then allow a feeder by feeder analysis to be performed, evaluating the present loads on the network (in contrast to the commonly used peak load analysis).

A further consequence of this sort of implementation is the reduction of the time to carry out a solution. The time frame to obtain a solution needs to be short enough to allow an operator to restore supply to the network without restricting their existing activities, such as field operator coordination and DMS monitoring. The present analysis has been carried out at a Zone distribution network level. Reducing this to a feeder by feeder analysis, as would be the case in practice, is also desirable.

Furthermore, as the electricity network moves towards 'Smart Grid' technologies, this algorithm has the potential to facilitate the uptake of this technology by providing the alternate switching strategies automatically for use in control schemes. The time frames for an optimum solution in this scenario would be required to be able to cater for use on a 'self-healing' network.

This is a situation where the network will comprise a large number of automatically controllable switches in place of the existing manual air break switches. These devices will allow for the identification of a fault location, the isolation of this fault and the initiation of supply restoration automatically. In this case, the solution routine would need to provide an optimised solution in real time. If the automation were designed to suit the present zone and subtransmission substation restoration times, then the time from fault occurrence to restoration would be required to be less than one minute. Given the present times of operation of automated equipment the algorithm would be required to complete its analysis over the order of a few seconds.

As a final comment, a key element not yet considered in this approach, but which will be in the near future, is the switching scheduling associated to each strategy. The switching scheduling corresponds to the sequence of switching operations needed to change the distribution network from *original* to *final state*. In real operational scenarios, the sequence of switching operations is as important as the network final state itself. Indeed, that final state might not be reachable without violating some operational constraints during the process, causing safety devices to trip, and rendering the solution infeasible. Research currently underway will address this issue.

VI. Conclusion

This paper presents a new genetic algorithm for the $(n-1)$ contingency network reconfiguration problem. The results indicate that the algorithm implemented can quickly produce a complete $(n-1)$ contingency plan for a small network (96 buses, 16 switches and 2 generators), under real operational constraints. The results in this study could be reproduced, apart from rounding errors, by Energy Australia using the industry standard package ASPEN Power Flow. The method is very fast, taking 30 seconds of CPU time to determine a switching strategy for any given $(n-1)$ network contingency situation. It also respects operational constraints of voltage and load in all sections, as well as the radial topology of the network. In terms of scalability, we expect the CPU time to increase by $O(n^3)$ for larger networks, in the worst case, where n is the number of buses. The main reason is that the AC power flow calculation uses Newton's method to solve a non-linear system. The package used in our implementation, namely MATPOWER, takes advantage of Matlab's sparse matrix library and this calculation is actually very efficient. Therefore, its impact on the overall computational time might in practice be much smaller.

Another delicate topic is the increase in the number of switches, as larger networks will have more switches. Even though the search space will grow exponentially, the number of solutions evaluated until the population converges typically increases polynomially in genetic algorithms. Therefore, again the impact on CPU time should be limited. Extensions of this study currently underway include the parallelization of the method; testing on a large scale network, with over 1,600 buses and 100 switches; and the introduction of *switching scheduling* considerations.

Acknowledgment

This work was funded by the Energy Australia Pilot Grant G0900080. The authors would like to thank Energy Australia for providing the network data and validating the results.

References

[1] J. Zhu, Optimal reconfiguration of electrical distribution network using the refined genetic algorithm. *Electric Power Systems Research* 62:37-42, 2002.

[2] D. Zhang, Z. Fu, L. Zhang, An improved TS algorithm for loss-minimum reconfiguration in large-scale distribution systems. *Electric Power Systems Research* 77:685694, 2007.

[3] A.Y. Abdelaziz, F.M. Mohameda, S.F. Mekhamera and M.A.L. Badra, Distribution system reconfiguration using a modified Tabu Search algorithm. *Electric Power Systems Research* (in press), 2010.

[4] P. Prasad, S. Sivanagaraju and N. Sreenivasulu, Network Reconfiguration for Load Balancing in Radial Distribution Systems Using Genetic Algorithm. *Electric Power Components and Systems* 36(1):63-72, 2008.

[5] W. Lin and H. Chin, A New Approach for Distribution Feeder Reconfiguration for Loss Reduction and Service Restoration. *IEEE Transactions on Power Delivery* 13(3):870-875, 1998.

[6] V.J. Garcia and P. Franca, Multiobjective service restoration in electric distribution networks using a local search based heuristic. *European Journal of Operational Research* 189:649-705, 2008.

[7] S.P. Singh, G.S. Raju and G.K. Rao, A heuristic method for feeder reconfiguration and service restoration in distribution networks. *Electrical Power and Energy Systems* 31:309-314, 2009.

[8] A.P.A. Silva, D.M. Falcao and K.Y. Lee. Overview of Applications in Power Systems. In: *Modern heuristic optimization techniques: theory and applications to power systems*, K.Y. Lee and M.A. El-Sharkawi (eds.), Wiley-IEEE, 2008.

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, USA, 1989.

[10] D. Goldberg and K. Sastry. *Genetic Algorithms: The Design of Innovation*. 2^{nd} ed., Springer, USA, 2010.

[11] R.D. Zimmerman, C.E. Murillo-Sanchez and D. Gan. *MATPOWER: A MATLAB Power System Simulation Package*. Available at `http://www.pserc.cornell.edu/matpower`, 2010.

[12] L. Buriol, P. Franca and P. Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics* 10:483-506, 2004.

[13] P. Moscato, A. Mendes and R. Berretta. Benchmarking a memetic algorithm for ordering microarray data. *Biosystems* 88:56-75, 2007.

[14] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Springer, USA, 2000.