

Briefing: Method for electronics optimisation
using a suite of fast specialist optimisation
scenarios produced via an evolutionary algorithm

Adrian Thompson,
Dept. of Informatics,
University of Sussex
adrianth@sussex.ac.uk

Contents

1	Background	1
2	Prior Work	3
3	The Proposed Method	3
4	The Prototype System	8
4.1	Performance evaluation.	8
4.2	Technology Independent Optimisation	9
4.3	Details of the Operation Phase: Single-stepping, Iteration, and Sequential Execution	18
4.4	Example of co-adaptation with other phases of optimisation: tech- nology dependent optimisation	20
4.5	General Observations From The Prototype	20
4.5.1	<i>A Priori</i> Reasons why standard SIS scripts may be im- proved upon through evolutionary methods	21
4.5.2	Some experimental observations on the evolution of SIS scripts	21
4.6	Some Details of the Evolutionary Algorithm	22

1 Background

Modern electronics design relies on computer software that automates or aids some of the design process. This is true for all kinds of electronics, but digital electronics has the most highly developed (and most relied-upon) software tools. Figure 1 shows a rough breakdown of a traditional design flow, where all of the processes shown are achieved through computer software tools. These tools come under the banner “Electronics Design Automation” (EDA).

The main input from the user is the high-level specification. In the past this was in the form of schematics (circuit diagrams) but often now a hardware description language is used. Increasingly, reference is also made to pre-designed

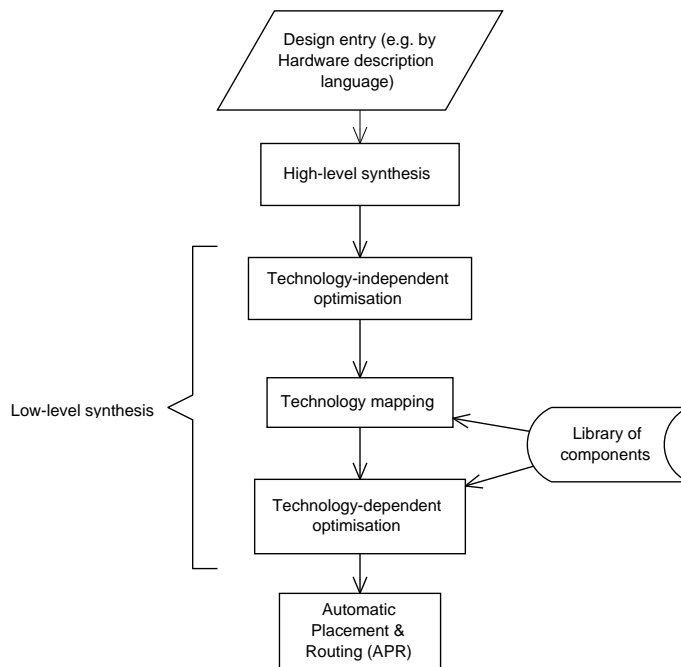


Figure 1: Simplified breakdown of a traditional design flow for digital electronics

proprietary subsystems. This input is transformed by the process of high level synthesis into a form almost ready to be mapped onto networks of electronic components. High level synthesis involves many activities not shown in the figure, such as assigning tasks to particular modules and scheduling how these will be used.

Low level synthesis transforms the results of high level synthesis into a form that can be built in the chosen technology, usually an integrated circuit. (‘Technology’ refers to a particular way of building circuits, such as a certain fabrication process for integrated circuits.) Technology-independent optimisation aims to simplify the design as much as possible, but still at the level of abstract logic, rather than a network of physical components of the technology. Typically, technology independent optimisation manipulates directed acyclic graphs (DAGs), at the nodes of which are Boolean equations. Simplifications to this network of logic usually give a payoff in terms of smaller circuits later.

The design is then mapped onto the components available in the technology, and attempts are made to perform further optimisations. Automatic placement and routing finds good physical locations for the components and routes for the connections between them.

This is a simplified view of a contemporary design flow. In fact, there are not such clear divisions between the activities, and it is possible to backtrack to reconsider decisions made earlier at a higher level. Important aspects such as verification and testing are not discussed here.

This proposed method concerns the processes of automated optimisation that take place at many of the stages in the design flow. One goal is always to minimise the size of the resulting circuit, as this leads to reduced cost and the

ability to fit more onto a given silicon area. Other important objectives include speed of circuit operation, power consumption, and testability; these can be applied either as constraints on the size optimisation, or can play an equal or even greater role than size considerations, depending on the application.

2 Prior Work

An optimisation may include many steps. Sequencing these steps and setting their parameters can be a difficult problem. A specification of the optimisation steps is often termed an optimisation *scenario*[1] or an optimisation *script*[2]. Typically, optimisation scenarios are derived through a great deal of analysis and manual experimentation; a scenario is designed that optimises nearly all circuits well (hereafter termed ‘general purpose’) and it is built in to the software tool.

[1] proposed the use of an evolutionary search algorithm (EA) (such as a Genetic Algorithm [3], Fig. 2) to derive a good optimisation scenario. It was found that it was possible to produce an optimisation scenario, tailored to optimise one particular circuit, that performed far better for that one circuit than general-purpose scenarios designed by human experts. (An analogous observation has been made in the domain of software optimisation [4]). [1] went on to suggest that if the EA could test each candidate scenario on *many* different circuits, it may be possible to evolve a scenario that outperforms those designed by human experts even in the general-purpose case.

The length of time taken to evolve a good optimisation scenario is problematic. Nearly all of the computational effort is spent on testing candidate optimisation scenarios. When the scenario is evolved for just one circuit, the amount of time and computing resource required may be justifiable if that particular circuit is especially important. When a general-purpose scenario is required, and an evolutionary fitness evaluation involves testing the candidate scenario’s ability to optimise many different circuits, the required evolution time and computational effort can be very great indeed. However, if the result is good, and is truly general-purpose, then it is widely applicable and again the resources required may be justified.

[5] followed a similar line of research targeted at a particular optimisation problem often encountered in EDA tools: a logic function is represented as a Binary Decision Diagram (BDD)[6] and the goal of optimisation is to find an ordering of the binary variables in the diagram such that it is minimised. In this case, an optimisation scenario specifies an algorithm for ordering the variables. Again, it was found that an evolutionary algorithm could produce an optimisation scenario that performed better on one particular circuit than did general-purpose scenarios designed by human experts. It was also found to be practical to evolve general-purpose scenarios by testing on only a few different circuits during evolution: the results generalised well to optimise circuits not encountered during evolution.

3 The Proposed Method

The method proposed here builds on some of the above prior findings, but is also motivated by the observation that the evolution of general-purpose scenarios can

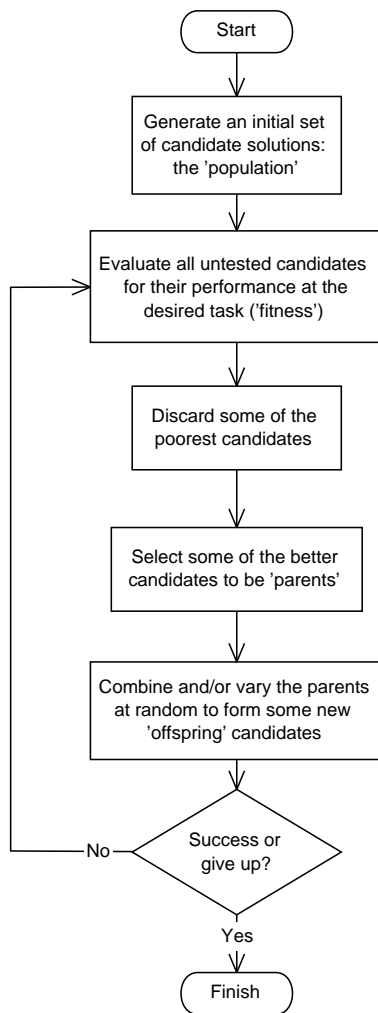


Figure 2: Outline of an evolutionary algorithm

be much more difficult for some optimisation problems than [5] report for the BDD variable-ordering problem. The prototype system finds scenarios for the most widely known tool for low-level synthesis: Berkeley SIS [7]. Hence, the findings from the prototype are readily comparable to a large body of literature and are of immediate practical use. The standard general-purpose scenarios that are supplied with SIS are known to be good [2, 8], so will be hard to beat. The SIS software itself does not need to be modified to exploit the proposed method.

In common with the prior work, it was found to be straightforward to evolve SIS scenarios (usually called *scripts* in this context) that optimised one particular circuit better than any of the standard general-purpose scripts. However, evolving a good general-purpose script proved to be challenging. By using an evolutionary algorithm that dynamically selected especially problematic fitness test-cases from the set of 74 circuits given in Table 1 during evolution, it was possible to produce a good general-purpose script for technology-independent optimisation. If we let gps_c be the best result from any of the three general-purpose standard scripts for circuit c , and evo_c be the result from the evolved script, then the relative performance $\frac{1}{74} \sum_{c=0}^{73} \frac{\text{evo}_c}{\text{gps}_c}$ showed the evolved result to be a few percent better than the combined forces of the three standard scripts when tested over these 74 circuits. (See Section 4.1 for details of performance evaluation.) However, when tested on a further thirty-five circuits not used during evolution, this performance advantage was partly eroded.

This example shows that for some optimisation problems, it can be difficult to use evolutionary algorithms to find a very good general-purpose optimisation scenario. The computational expense can be huge, and even good performance on as many as 74 circuits during evolution does not guarantee such good results when the resulting scenario is used for unseen circuits. It is worthwhile to battle against these problems further, and already some good results have been found, but here an alternative is proposed.

The proposed method is shown in Fig. 3. Its usefulness relies on some surprising observations made from the prototype SIS system, and it will not be suitable for all optimisation problems. However, the SIS prototype system is a useful application in its own right, and is likely to be representative of some other optimisation problems found in other EDA tools.

The method relies on the following assumptions:

1. A scenario to optimise one particular circuit can be evolved sufficiently quickly that the whole process can be repeated several times. Multiple evolutionary runs can be performed for different circuits to produce multiple specialised scenarios.
2. Each resulting scenario produces high quality results for its specialist circuit, yet does so quickly. (Speed of optimisation should be explicitly encouraged in the evolutionary algorithm.)
3. Often, a specialist scenario will not only perform well on the circuit it was evolved for, but also some other circuits, even given the same strict time constraints. Let us call these the scenario's *auxilliary* circuits. We can also expect the scenario to perform badly for many circuits.
4. It is possible to build up a *suite* of specialist scenarios such that the union

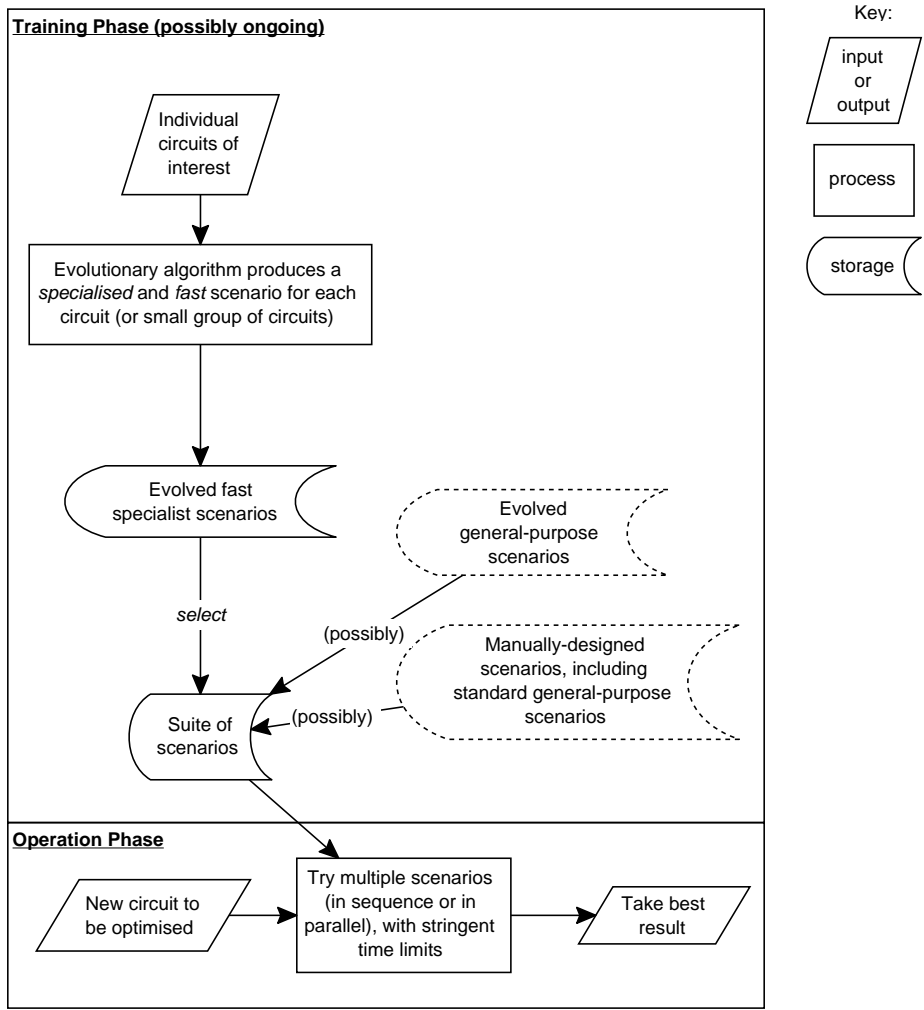


Figure 3: The Proposed Method

of their sets of auxiliary circuits covers many of the circuits that might be encountered.

If these assumptions hold, then on being presented with a new circuit to optimise, each of the specialist scenarios in the suite can be tried until a good result is achieved. All of the specialist scripts are fast, so it is feasible to try multiple scenarios. All being well, there is a high probability that at least one of the scenarios in the suite will perform excellently.

It is not obvious that the assumptions will hold. Most especially, that a specialist scenario will have a non-empty set of auxiliary circuits is partly down to luck (experimental findings are given later). Obviously, a newly evolved specialist scenario could undergo some initial screening against other circuits to determine whether it is likely to be a useful addition to the suite. Perhaps the most surprising finding from the prototype system is that it was possible to achieve excellent performance on many circuits using a suite of only a few specialist scenarios.

In some cases, it may be possible to classify circuits, to predict what scenarios are likely to be effective. This is likely to be beneficial when possible, and in that case the selection of scenarios from the suite could be more principled than simply taking each in turn. In the prototype system, no classification is done, yet the system still performs well.

Whilst the proposed method seeks to avoid the difficulties of evolving a general-purpose scenario (testing on multiple circuits during evolution), the evolutionary phase could be used to produce scenarios explicitly specialised for particular classes of circuits if these could be represented by *small* groups of circuits to be tested during evolution.

The evolved specialist scenarios could be augmented by scenarios produced by other methods to form the suite, as indicated in the diagram (Fig. 3). For example, if a standard manually-designed general-purpose scenario were added to the suite then the quality of optimisation in the operation phase would be guaranteed to be no worse than when using that scenario alone. A constraint is that the augmenting scenarios should operate sufficiently quickly that it is still feasible to try multiple optimisation scenarios from the suite during the operation phase to optimise a given circuit.

The requirements regarding the speed of evolution of the specialist scenarios (assumption 1 above) and their speed of operation (assumption 2) may be partially relaxed if multiple computers are available. Multiple evolutionary runs can be performed separately on separate microprocessors for the training phase, and multiple specialist scenarios can be tried in parallel during the operation phase.

The training phase need not cease once the operation phase has begun to be employed. As further beneficial fast specialist scenarios are found through ongoing evolutionary runs, they can be added to the suite. The choice of ‘circuits of interest’ on which to train (Fig. 3) can be influenced by weaknesses identified while the operation phase is employed to optimise many different circuits. The training phase could be performed using separate software and/or computers to the operation phase, and by different individuals or organisations. For example, an end-user of the EDA tool might only conduct the operation phase, with the suite being supplied by the tool vendor. Conversely, users could conduct their own training, using their own circuits of interest. It would be possible for

a community of users to maintain a communal database of evolved scenarios, perhaps distributed across the internet or the intranet of their organisation.

An evolutionary run may begin from a population of randomly generated scenarios, or can be seeded with scenarios that have already been evolved or designed manually. Note that a scenario may be evolved that spans optimisation activities that are normally considered separately, as long as this does not lead to impractically lengthy fitness evaluation tests. A simple example is given in Section 4.4.

The remainder of this document describes the prototype system, in which optimisation scripts for the Berkeley SIS system are produced. It should be remembered that the method is more general than this specific example, provided that the list of assumptions given above is satisfied.

4 The Prototype System

Berkeley SIS [7, 9] can perform technology-independent optimisation, technology mapping, and technology dependent optimisation, and incorporates the earlier system MIS [10]. It is widely used in the literature as the system to beat with any proposed new method. In the experiments here, scenarios are produced that allow SIS *itself* to perform better than usual, perhaps challenging alternative optimisation systems that were evaluated in comparison to SIS.

SIS may be viewed as old-fashioned in making a clear separation between technology-independent optimisation, technology mapping, and technology-dependent optimisation, although it is possible to evolve a scenario such that these phases are co-adapted. However, even those who propose alternatives often use SIS to perform technology-independent optimisation as a pre-processing step [11, 12, 13, 14]. Thus SIS is chosen for the prototype as the software is freely available and is widely used both in practice and the literature. The standard SIS-1.3 software was not altered except very minor additions to allow execution timings to be measured more accurately, and for optimisation results to be logged for easy access by the evolutionary software, which is completely separate.

SIS scenarios are given as a *script* supplied as a text file to the SIS software. Much effort has gone into deriving good general-purpose scripts through theory and manual experimentation [2, 8]. A selection of scripts is included in the SIS software distribution, and the three most widely used are given in Figs. 4–6. SIS provides many different optimisation commands that may appear in the scripts, and many of these take numerical parameters and option flags that fine-tune their behavior.

A common strategy is to try `script.rugged`, and if this fails to produce a result in the time available, to use `script.algebraic` [11]. The target to beat through the proposed method is the best that *any* of `script.rugged`, `script.algebraic` and `script.boolean` can produce.

4.1 Performance evaluation.

Seventy-four test circuits were taken from the widely-used MCNC'91 benchmark set [15, 16], Table 1. Each of `script.rugged`, `script.algebraic`, `script.boolean`, and `full_simplify` alone, were run on each of the circuits. The computer used was a 2.2GHz Intel Pentium PC with 512MBytes memory, running Linux. Each

```

# Initial pre-processing: try both with and without this command:
full_simplify

REPEAT until no further improvement possible, keeping best result:
{
  sweep
  eliminate -1
  simplify -m nocomp
  eliminate -1

  sweep
  eliminate 5
  simplify -m nocomp
  resub -a

  fx
  resub -a
  sweep

  eliminate -1
  sweep
  full_simplify -m nocomp
}

```

Figure 4: script.rugged

script was tried both with and without the initial `full_simplify` command separately; each combination was allowed up to 24 hours of processor time on each circuit, or 36 hours of real (wallclock) time, whichever was shortest. If no improvement was made over the initial circuit, then a script’s result was taken to be that initial circuit. The circuits were input to SIS in the form in which they are included in the SIS software distribution (better results can sometimes be achieved by trying different equivalent initial descriptions of a circuit [17], but that is often neglected in the literature, and was not done here). The “REPEAT until no further improvement possible” criterion was interpreted thus: a script would be terminated after the first iteration that did not give an improvement. (It is sometimes possible for the results to get worse before they get better, but to allow this makes it difficult to know when to stop. The criterion used here is also a common practical rule of thumb.) For each circuit c , the target for the proposed method to beat is the *best* result ever seen from these general-purpose standard scripts, and is denoted gps_c .

Some scripts are stochastic and can produce slightly different results when repeated under conditions identical but for the computer’s random number generator. The results herein were found to be sufficiently repeatable without taking this into account.

4.2 Technology Independent Optimisation

This is the level of logic (low-level) synthesis that takes place before technology mapping. The optimisation criterion chosen was to minimise the sum of the number of literals in factored form, as reported by the SIS command `print_stats -f`. It provides a measure of the overall complexity of the logic in the directed acyclic graph of Boolean functions that represents the circuit.

```
# Initial pre-processing: try both with and without this command:
full_simplify

REPEAT until no further improvement possible, keeping best result:
{
  sweep
  eliminate 5
  simplify -m nocomp -d
  resub -a

  gkx -abt 30
  resub -a
  sweep
  gcx -bt 30
  resub -a
  sweep

  gkx -abt 10
  resub -a
  sweep
  gcx -bt 10
  resub -a
  sweep

  gkx -ab
  resub -a
  sweep
  gcx -b
  resub -a
  sweep

  eliminate 0
  decomp -g *
}
```

Figure 5: script.algebraic

```

# Initial pre-processing: try both with and without this command:
full_simplify

REPEAT until no further improvement possible, keeping best result:
{
  sweep
  eliminate -1
  simplify
  eliminate -1

  sweep
  eliminate 5
  simplify

  resub -a

  gkx -abt 30
  resub -a
  sweep
  gcx -bt 30
  resub -a
  sweep

  gkx -abt 10
  resub -a
  sweep
  gcx -bt 10
  resub -a
  sweep

  gkx -ab
  resub -a
  sweep
  gcx -b
  resub -a
  sweep

  eliminate 0
  decomp -g *

  eliminate -1
  sweep
}

```

Figure 6: script.boolean

	Name	Type	Inputs	Outputs
0	majority	voter	5	1
1	b1	logic	3	4
2	C17	logic	5	2
3	cm82a	logic	5	3
4	parity	logic	16	1
5	tcon	logic	17	16
6	cm151a	logic	12	2
7	cmb	logic	16	4
8	cm150a	logic	21	1
9	mux	mux	21	1
10	cm85a	logic	11	3
11	cm163a	logic	16	5
12	cm138a	logic	6	8
13	x2	logic	10	7
14	il	logic	25	16
15	pm1	logic	16	13
16	cu	logic	14	11
17	pcle	logic	19	9
18	cm42a	logic	4	10
19	z4ml	2-bit add	7	4
20	cm162a	logic	14	5
21	unreg	logic	36	16
22	cc	logic	21	20
23	pcler8	logic	27	17
24	cordic	logic	23	2
25	decod	decoder	5	16
26	set	logic	19	15
27	c8	logic	28	18
28	count	counter	35	16
29	lal	logic	26	19
30	b9	logic	41	21
31	cht	logic	47	36
32	my_adder	adder	33	17
33	comp	logic	32	3
34	i5	logic	133	66
35	example2	logic	85	66
36	tvt2	logic	24	21
37	i3	logic	132	6
38	x1	logic	51	35
39	apex7	logic	49	37
40	term1	logic	34	10
41	frg1	logic	28	3
42	i2	logic	201	1
43	x4	logic	94	71
44	C880	ALU and control	60	26
45	x3	logic	135	99
46	apex6	logic	135	99
47	rot	logic	135	107
48	i6	logic	138	67
49	C499	error correcting	41	32
50	9symm1	count ones	9	1
51	frg2	logic	143	139
52	vda	logic	17	39
53	i7	logic	199	67
54	pair	logic	173	137
55	C2670	ALU and control	233	140
56	i9	logic	88	63
57	C5315	ALU and selector	178	123
58	C6288	16-bit multiplier	32	32
59	C3540	ALU and control	50	22
60	C7552	ALU and control	207	108
61	C1355	error correcting	41	32
62	C1908	error correcting	33	25
63	C432	priority decoder	36	7
64	k2	logic	45	45
65	alu2	ALU	10	6
66	alu4	ALU	14	8
67	des	data encryption	256	245
68	dalu	dedicated ALU	75	16
69	t481	logic	16	1
70	i10	logic	257	224
71	i4	logic	192	6
72	i8	logic	133	81
73	too_large	logic	38	3

Table 1: Test set of circuits from the MCNC'91 benchmark suite.

Even at this technology-independent level, it can sometimes be useful to take account of other optimisation criteria such as delay (longest path) from circuit inputs to outputs. That was not done here, allowing easy comparison with the majority of the literature, which makes the same choice.

The degree to which minimising the literals in factored form leads to high quality mapped circuits later depends on the target technology and the mapping and optimisation processes that work with it. Complex-gate CMOS technology, for example, can directly reflect a factored form representation, and the technology-independent minimisation would probably lead to a mapped circuit with nearly minimal area and acceptable delay characteristics. For other technologies, it is clear that minimising the count of literals in factored form does not guarantee that the final mapped area will be minimal [15], and constraints on maximum delay can actually conflict with area minimisation. Nevertheless, technology independent minimisation of the literals in factored form is widely accepted as a useful first step, even when there is much more technology-oriented optimisation to follow.

Hence, the quality metric used during the evolution of new specialist scripts was the count of literals in the factored form when the script terminates. We can compare this with the smallest corresponding figure produced by any of the general-purpose standard scripts run for up to 24hrs as described as in Section 4.1 above. The evolved scripts were allowed much less time. All of the evolutionary runs reported here were performed on a 1.6GHz laptop PC with 256MBytes of memory. The maximum amount of time allowed for a script to terminate during evolution depended on what circuit was being tackled, but was never more than 600s. Within this limit, there was also a selection pressure for scripts to be as fast as possible without sacrificing quality (see Section 4.6 for details). In the examples to follow, where an evolved specialist script outperforms the general-purpose standard scripts on some ‘auxiliary’ circuits, this never takes more than 550s on the 1.6GHz PC, usually much less.

Figure 7 shows a fast specialist script ‘A’ evolved to optimise the circuit C6288. This circuit was chosen as the first example, because it has already been remarked that it is troublesome both for SIS [2] and for alternative optimisation techniques based on BDDs [11]. ‘A’ also performs well on 12 of the other 73 circuits in the test set (Table 1), as shown in Table 2. Notice that while ‘A’ is tolerably fast for its specialist circuit, it is not necessarily faster than the general-purpose standard scripts for its auxiliary circuits. The length of time allowed during the operation phase for each scenario in the suite can be adjusted through experiments to give a desired overall tradeoff between time and performance for the suite. Speed considerations can also affect which evolved scripts are chosen for inclusion in the suite.

Fig. 8 shows a fast specialist script evolved to optimise circuit 73: `too_large`. Table 3 gives its set of auxiliary circuits from the 73 others on which it was tested.

Fig. 9 shows the performance that could be seen in the operation phase of the proposed method when the suite of fast specialist scripts contained ten evolved examples. These scripts included scripts A and B; the diagram indicates the other circuits for which specialist evolved scripts were included.

Even with just these ten examples of specialist scripts, performance in the operation phase would already beat the best of the general-purpose standard scripts for over half of the circuits, sometimes dramatically. Performance is infe-

```

eliminate -l 10000 -1
eliminate -l 2989 47
fx
eliminate -l 841 -1
fx
eliminate -l 3631 47
fx
eliminate -l 1077 75
sweep
fx
simplify -m dcsimp
decomp -g
decomp -g
resub *
eliminate -l 9995 0
simplify -m nocomp -f disj_sup
simplify -i 3 -m dcsimp
eliminate -l 9990 -1
simplify -i 3 -m dcsimp
eliminate -l 9366 0

```

Figure 7: Script A, evolved to optimise C6288

Circuit		General-Purpose Standard			Evolved	
		BEST		SLOWEST	evo_c^A	$t(\text{evo}_c^A)$
		gps_c	$t(\text{gps}_c)$	$t(\text{slowest})$		
6	cm151a	26	0.05	0.06	24	0.02
15	pm1	50	0.09	0.11	49	0.04
26	sct	75	0.19	0.20	71	0.08
27	c8	137	0.23	0.27	131	0.11
29	lal	100	0.20	0.24	94	0.09
30	b9	122	0.17	0.26	119	0.12
39	apex7	243	0.52	0.81	233	0.22
40	term1	142	0.77	0.98	141	0.42
42	i2	213	0.52	2.76	212	8.33
44	C880	408	3.13	3.14	399	5.93
55	C2670	712	14.16	14.16	709	146.8
58	C6288	3295	18.26	21.38	3222	18.35
71	i4	204	0.26	86400 timeout	192	0.33

Table 2: Circuits for which evolved script A performs better than any of the three standard scripts. gps_c is the smallest number of literals in factored form from any of the three general-purpose standard scripts, and $t(\text{gps}_c)$ is the processor time in seconds taken by that best script. evo_c^A and $t(\text{evo}_c^A)$ give the corresponding performance of script A. The time taken by the slowest of the standard scripts is also shown. For this table, script A was tested on the same 2.2GHz reference PC as were the standard scripts, to allow a direct timing comparison.

```

collapse
full_simplify -o 1 -m snocomp -l
fx -oz
full_simplify -d -o 0 -m dcsimp
xl_partition -n 18 -M 1 -t
full_simplify -o 0 -m dcsimp
eliminate -l 9395 0
full_simplify -d -o 0 -m dcsimp
full_simplify -o 0 -m dcsimp -l
fx -ol
fx -lz
full_simplify -o 0 -m dcsimp -l
decomp -g
eliminate -l 9531 3
eliminate -l 789 19
full_simplify -o 1 -m dcsimp
fx -l
full_simplify -d -o 0 -m dcsimp
full_simplify -o 1 -m dcsimp
eliminate -l 9365 8
full_simplify -o 0 -m snocomp
eliminate -l 4988 -1
eliminate -l 1394 10
full_simplify -o 0 -m dcsimp
fx -l
full_simplify -o 0 -m snocomp
eliminate -l 9371 7
fx -olz
fx
eliminate -l 9688 8
full_simplify -d -o 1 -m snocomp
fx -z
eliminate -l 9963 -1
full_simplify -o 0 -m dcsimp
decomp -q
eliminate -l 2032 6
simplify -i 20:1 -m nocomp
fx -olz
fx -ol
phase -g
eliminate -l 9838 -1

```

Figure 8: Script B, evolved to optimise the circuit 73: `too_large`. Here, any redundant commands have been pruned away: no single command can further be removed without degrading performance in optimising `too_large`.

Circuit	Improvement at script termination (%)	Best improvement if single-stepped (%)
2	11.1	11.1
6	3.8	7.7
7	26.0	38.0
8	-	7.8
9	-	7.8
10	30.4	30.4
13	4.3	4.3
15	-	4.0
16	12.1	12.1
23	-	1.1
26	-	1.3
27	4.4	4.4
30	1.6	1.6
35	5.3	5.3
36	4.1	4.1
37	-	19.5
40	9.2	9.2
41	5.0	10.0
45	-	2.5
50	15.3	18.6
51	-	1.1
71	2.0	5.9
73	30.1	30.1

Table 3: Performance of script B: Shown is the percentage improvement over the best of the three general-purpose standard scripts in each case, when that improvement is positive. See Section 4.3 for an explanation of the rightmost column.

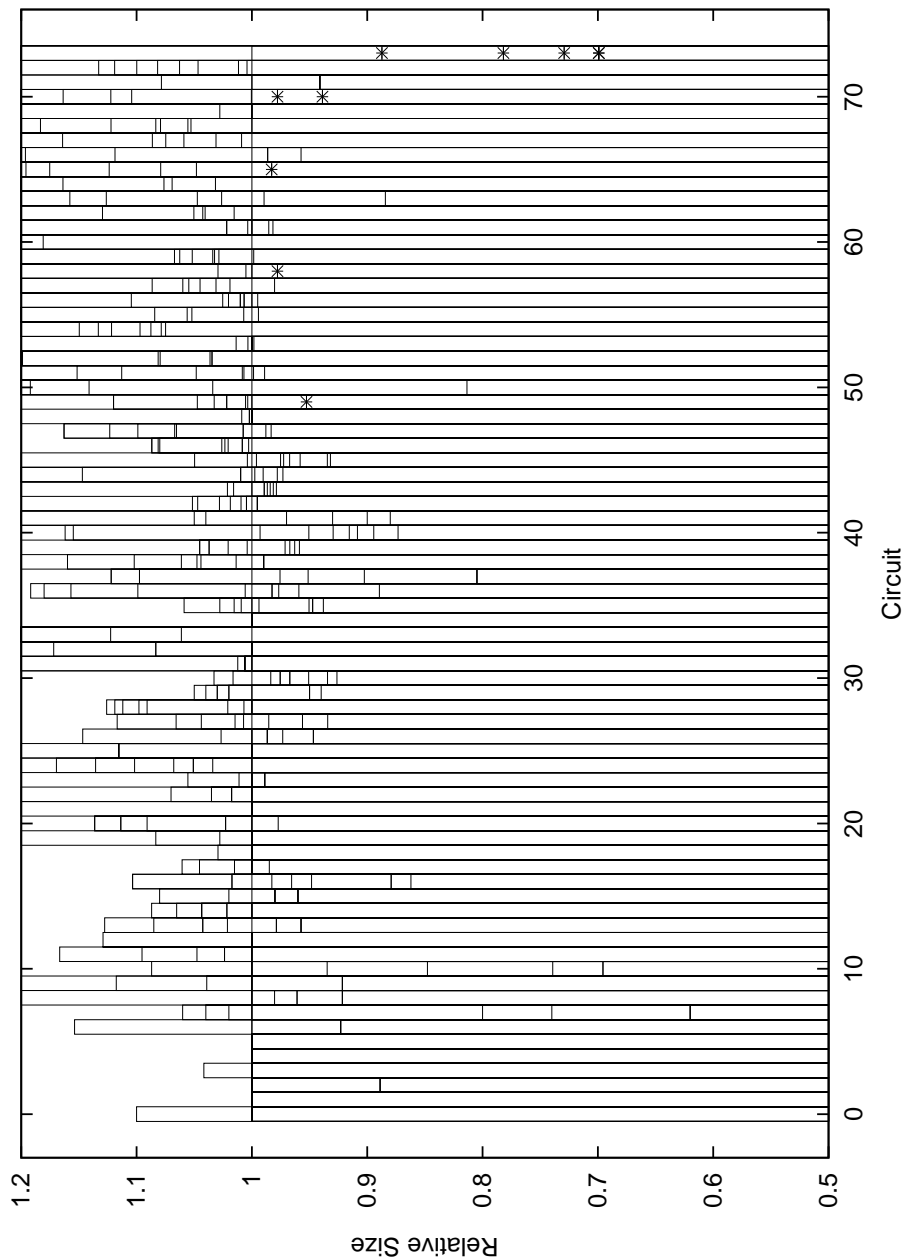


Figure 9: Performance of ten evolved fast specialist scripts across the 74 circuits, relative to the best of the general purpose standard scripts for each circuit (e.g. 0.9 means 10% better). A bar is drawn for each script tested on each circuit, some of which overlap identically. The * symbols show the conditions for which the scripts were actually evolved. The worst results go off the scale at the top.

rior for many circuits also: this could be avoided by adding the standard scripts to the suite (with appropriate timeouts to their operation) as shown by the solid horizontal line at Relative-Size=1 in the diagram. There is no evidence that this performance can not be improved by adding further or alternative evolved fast specialist scripts to the suite — these results are at an early stage. Suites have already been constructed giving an *average* performance of over 5% better than the general-purpose standard scripts alone. Perhaps more important than the average improvement is that there is a large improvement for significantly many circuits.

4.3 Details of the Operation Phase: Single-stepping, Iteration, and Sequential Execution

During the training phase, evolution crafts a scenario with the sole objective of optimising its specialist circuit. In the operation phase, such scenarios are used to optimise *different* circuits: a purpose for which they were not designed. Consequently, in the operation phase it can be beneficial to use the scenarios slightly differently to the way they were executed during the training phase:

Single-stepping A specialist scenario will evolve to deliver the most highly optimised version of its specialist circuit at the very end of the scenario’s execution. However, this same scenario, when processing a different circuit during the operation phase, may produce its best result at some intermediate point. Ideally, the quality of the optimisation should be measured after each step in the scenario and the best result taken, rather than the result at the end. This is possible when the computational cost of measuring the quality of the optimisation is not prohibitive. Table 3 illustrates the advantages of single-stepping SIS scripts in the operation phase of the prototype system, and this technique was used for Fig. 9.

Iteration A specialist scenario may be executed repeatedly, each time starting with the best result so far, as is the practice for the general-purpose standard scripts of SIS. There is no need to do this during the training phase because a single scenario can grow in length to accommodate repetitions within a single iteration. When the specialist script is applied to other circuits during the operation phase, iteration can be beneficial.

Sequential Execution If the scenarios of the suite are not physically executed in parallel on multiple processors during the operation phase, then Fig. 10 shows two alternatives for how to apply them. The first is logically equivalent to parallel execution, and each scenario begins work on the same initial description of the circuit to be optimised. The second possibility is to allow each scenario to begin work on the best result found by any of the previous scenarios. The ordering of the scenarios can be important. In the prototype system, it was beneficial to try both alternatives when time permitted.

Iteration and sequential execution were found to be useful when executing specialist scripts in the operation phase of the SIS prototype, but were not used in producing any of the results shown herein.

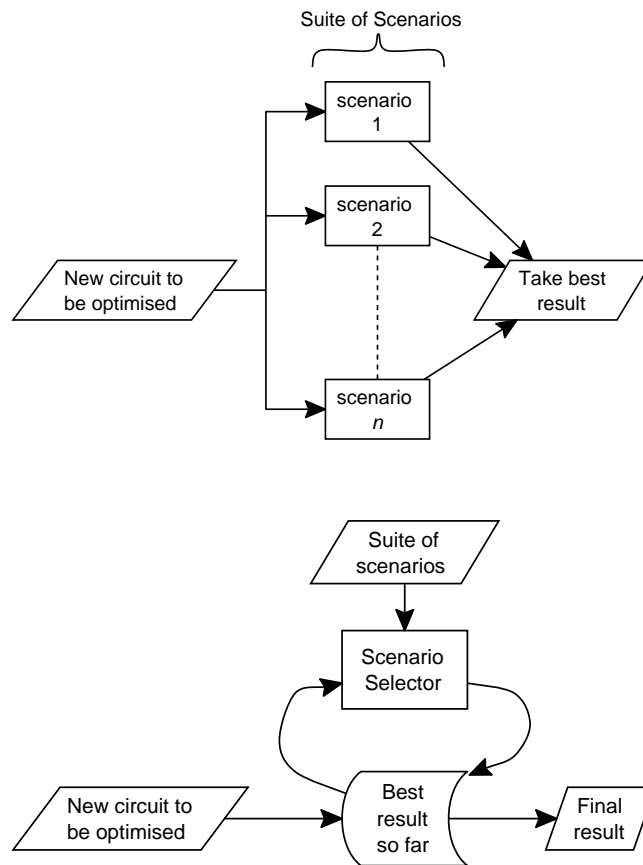


Figure 10: Alternatives for scenario execution in the operation phase.
Top: If each scenario works independently from the same initial description of the circuit to be optimised, they could be run in parallel on multiple processors.
Bottom: If the scenarios are executed sequentially, then each could start from the best result from the previous scenarios.

```

simplify -m dcsimp
map -f 3 -B 0 -i
decomp -q
tech_decomp -a 2 -o 2
xl_partition -t
simplify
simplify -m dcsimp
fx -z

# Predefined final mapping:
map -m 0 -AF; phase -g

```

Figure 11: A specialist script evolved to optimise `my_adder` with mapping to library `stdcell2_2`.

4.4 Example of co-adaptation with other phases of optimisation: technology dependent optimisation

The evolved specialist scenarios may span optimisation activities that are normally considered separately, as long as this does not lead to impractically lengthy fitness evaluation tests during the training phase. For example, instead of performing technology-independent optimisation in isolation, we may insist that the subsequent technology mapping is always performed, and that the quality of the optimisation is taken as the total mapped area, rather than the literals in the factored form of the network before mapping. The scenario must then take account of the characteristics of the mapping algorithm and the technology library.

For example, let us map to the CMOS standard-cell library `stdcell2_2.genlib` distributed with the SIS software. We predefine that the mapping is to be performed with the SIS commands `map -m 0 -AF; phase -g`, recommended for a minimum area circuit that respects the load limits given in the technology library. These commands are appended to every script during evolution, and the quality of the optimisation is given by the resulting mapped area.

Fig. 11 shows a fast specialist script evolved to optimise the circuit `my_adder`. In this example, the second command of the evolved script performs a preliminary mapping to the technology library. The subsequent commands destroy the perfect correspondence between cells in the technology library and nodes in the DAG representing the circuit. However, if the preliminary mapping command is removed, then the area after the predefined final mapping increases. The script appears to be taking greater account of the mapping process than if it were simply to optimise the literals in factored form before the final mapping.

Scripts such as this could readily be assembled into a suite, and the proposed method applied, giving a more 'technology aware' optimisation.

4.5 General Observations From The Prototype

This section first lists some *a priori* reasons why existing SIS scripts may not be optimal, and may be improved upon through evolutionary methods. Then observations are made from use of the proposed method in this context, indicating the power of the approach.

4.5.1 *A Priori* Reasons why standard SIS scripts may be improved upon through evolutionary methods

1. The existing hand designed scripts are not proven to be optimal but were produced by extensive experimentation. Therefore it is possible that better solutions exist.
2. The existing solutions consist of a short script that is repeated over and over until no further improvement occurs. This makes manual experimentation feasible, but there is no reason to think that the best scripts should be short, nor that repeated application of exactly the same script is optimal. Therefore our proposed evolutionary approach should be allowed to explore long scripts without any enforced repetition: another reason to think there may be better solutions to be found.
3. The evolutionary search for good scripts could be allowed to take account of the strengths and weaknesses of other activities in the design flow. See Section 4.4 for example.
4. There are well established techniques by which evolutionary algorithms can consider many different criteria of quality concurrently [18](multi-objective optimisation). This is ideal in the present application where there may be trade-offs or constraints among criteria of size, circuit delay, power use, etc. Crucial to success is the realisation that we can also include the time that the evolving optimisation scripts take to run on the computer into this multi-objective scheme. (Measuring computer processing time implicitly guards against excessive memory use also, since this would lead to very slow performance.) Only by constraining the scripts to be fast (at least in early stages of evolution) can we expect to evaluate enough of them for evolution to find a good solution. Once we move to considering scripts that take a relatively long time, we can only do a small amount of evolution for fine-tuning.
5. The fact that good hand-designed scripts already exist can be used to help set up the evolutionary system. For example, the maximum amount of processor time allowed to an evolved script may initially be based on how long it takes for the standard scripts to achieve their best result.

4.5.2 Some experimental observations on the evolution of SIS scripts

When evolving a script that is specialised to optimise just one circuit:

1. If the evolving scripts operate quickly enough to allow sufficient evolution, it is usually easy to produce specialist scripts that do better for their target circuits than do any of the standard general-purpose scripts.
2. The superior evolved scripts may take far less processor time to complete the optimisations than the general-purpose standard scripts. This means that the evolutionary approach may be successful even in cases where one would previously have thought it would take an unfeasible amount of computing time.

3. A script evolved to optimise one particular circuit often (not always) also produces superior results when applied to some (not all) other circuits: the set of ‘*auxiliary*’ circuits is often not empty, provided the target circuit is sufficiently challenging.
4. A script evolved to optimise one relatively small circuit may produce superior results even for some large circuits. A script evolved for one relatively large circuit may produce superior results even for some small circuits.
5. If we independently evolve several specialist scripts for the same target circuit, the sets of auxiliary circuits for which these scripts produce superior results can be different. It has even been seen that some good evolved scripts for a target circuit have extensive auxiliary sets, while other good scripts specialised for the same target circuit appear to be useless for any other.
6. If we evolve two different scripts, each to optimise a different target circuit, then their auxiliary sets can be very different. It is possible for the union of the auxiliary sets of only a few specialist scripts to cover many circuits. This (along with the previous observation) implies that we are not just “filling in” for a few circuits on which the standard hand-designed general-purpose scripts happen to be particularly weak.
7. If a specialist script contains some redundant commands, they can readily be pruned away through an automated systematic set of tests to see which commands are necessary. When these commands that are redundant with respect to the specialist target circuit are removed, the auxiliary set can change.
8. Scripts from part-way through an evolutionary run may or may not have more useful auxiliary sets than the final result that is fully honed to its specialist target.

Hence the set of conditions required for the proposed ‘*suite of fast specialist scenarios*’ method listed in Section 3 hold. In fact, it seems to be surprisingly easy to build up an effective suite in this case. It is unknown to what extent this will apply to other optimisation levels and methods within EDA.

4.6 Some Details of the Evolutionary Algorithm

The evolutionary algorithm was a Genetic Algorithm (GA) [3, 19] with no extraordinary features. The results are not thought to depend on the details of the evolutionary algorithm.

The GA used linear rank selection with truncation and elitism, acting on a population of 30. To rank the individuals (candidate scripts) into order, or to decide which one is the best, a way of comparing two individuals is required. The comparison is entirely according to the optimisation quality metric of each individual unless these are equal. In the latter case, the comparison is made according to the time taken for the individual scripts to terminate, favouring the faster. If there is still a tie, then the shorter script wins. (This method of dealing with multiple criteria of fitness having a fixed priority is often termed lexicographical or dictionary ordering.)

During an evolutionary run, even after the optimisation quality had stopped improving, there was usually a final phase during which the speed of optimisation was improved. The results shown here were typically obtained after a few hundred generations.

In all of the experiments here, each evolutionary run commenced with a different initial population of scripts, where each script was randomly generated and exactly three commands long. It is easy to seed an evolutionary run from a pre-existing result (whether hand-designed or evolved previously), and this remains worthy of further exploration.

The ‘genetic’ variation operators work on a direct numerical representation of the script using integers only. They are:

Mutation — New Command: A command of the script is replaced with a new random one, and any parameters are also chosen at random.

Mutation — New Command Variant: The symbolic flags defining a particular version of a command are randomised. If the number of numerical parameters associated with the command is not changed by this, and the valid ranges of those parameters are also unchanged, then the numerical parameters are left unchanged. Otherwise, any numerical parameters associated with the new command variant are generated at random.

Mutation — Parameters: One of the numerical parameters of a command is adjusted by the BGA mutation method [20].

Homologous crossover: Standard two-point crossover, always keeping the parameters unseparated from their associated commands.

Nonhomologous crossover: As above, but the segment between the crossover points is randomly translocated.

Insert: Inserts one new random command at a random position, randomly generating any parameters.

Delete: Removes one randomly chosen command.

Block insert: Chooses a consecutive sequence of commands at a random location and of random length in one parent script, and inserts it at a random position in the second parent (increasing the script length) to generate an offspring.

Block delete: Chooses a consecutive sequence of commands at a random location and of random length, and removes it.

The probabilities chosen for these operators were informally derived through experimentation and may not be optimal. The probabilities of the operators that change the length of a script were balanced such that there was no inbuilt tendency for scripts to grow or shrink in the absence of selection. The chosen numerical representation guarantees that each SIS command has a valid set of parameters: invalid scripts are not possible.

References

- [1] A. Kuehlmann and L.P.P.P. van Ginneken. Grammar-based optimization of synthesis scenarios. In *ICCD'94: Proc. IEEE Int. Conf. on Computer Design*, pages 20–25. IEEE Computer Society, 1994.
- [2] H. Savoj, H.-Y. Wang, and R.K. Brayton. Improved scripts in MIS-II for logic minimization of combinatorial circuits. In *Proc. International Workshop on Logic Synthesis (IWLS)*, May 1991.
- [3] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.
- [4] K.D. Cooper, T.J. Harvey, D. Subramanian, and L. Torczon. Compilation order matters. Technical report, Rice University, Houston, TX, January 2002.
- [5] F. Schmielde, N. Drechsler, D. Große, and R. Drechsler. Heuristic learning based on genetic programming. *Genetic Programming and Evolvable Machines*, 3:363–388, 2002.
- [6] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, c-27(6):509–516, June 1978.
- [7] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California, Berkeley, 1992.
- [8] H. Savoj. *Don't Cares in Multi-Level Network Optimization*. PhD thesis, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1992.
- [9] E.M. Sentovich. *Sequential Circuit Synthesis at the Gate Level*. PhD thesis, Dept. Electrical Engineering and Computer Sciences, University of California, Berkeley, 1993.
- [10] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang. MIS: Multiple-level logic optimization system. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, 1987.
- [11] G. Chen and J. Cong. Simultaneous logic decomposition with technology mapping in FPGA designs. In *FPGA'01: Proc. ACM/SIGDA 9th Int. Symp. on Field programmable gate arrays*, pages 48–55. ACM Press, 2001.
- [12] W. Günther and R. Drechsler. ACTION: Combining logic synthesis and technology mapping for MUX based FPGAs. In *Proc. 26th IEEE Euromicro Conference*, pages 130–137 vol.1, 2000.
- [13] B. Kumthekar and F. Somenzi. Power and delay reduction via simultaneous logic and placement optimization in FPGAs. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 202–207. IEEE, 2000.

- [14] S.-C. Chang and M. Marek-Sadowska. Perturb and simplify: Optimizing circuits with external don't cares. In *Proc. EDTC'96 Proc. European conf. on Design and Test*, pages 402–406. IEEE Computer Society, 1996.
- [15] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, Microelectronics Center of North Carolina, P.O. Box 12889, Research Triangle Park, NC 27709, 1991.
- [16] N. Whitaker. Status report on EDA benchmarks. Technical Report STEED/T1/01/4, MINT Group, Dept. Computer Science, Univ. Manchester, UK.
- [17] W. Günther and R. Drechsler. Creating hard problem instances in logic synthesis using exact minimization. In *Proc. IEEE Int. Symposium on Circuits & Systems (ISCAS)*, pages 436–439 vol.6, 1999.
- [18] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimisation. *Evolutionary Computation*, 3(1):1–16, 1995.
- [19] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, 1989.
- [20] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm BGA. *Evolutionary Computation*, 1(4):335–360, 1994.