Comparing Different Serial and Parallel Heuristics to Design Combinational Logic Circuits

Carlos A. Coello Coello
CINVESTAV-IPN
Evolutionary Computation Group
Dpto. de Ingeniería Eléctrica
Sección Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07300, MEXICO
ccoello@cs.cinvestav.mx

Enrique Alba, Gabriel Luque
Dpto. de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Campus Teatinos, 29071, Málaga, SPAIN
{eat,gabriel}@lcc.uma.es

Arturo Hernández Aguirre
Center for Research in Mathematics (CIMAT)
Department of Computer Science
Guanajuato, Gto. 36240, MEXICO
artha@cimat.mx

Abstract

In this paper, we perform a comparative study of different heuristics used to design combinational logic circuits. The use of local search hybridized with a genetic algorithm and the effect of parallelism are of particular interest in the study conducted. Our results indicate that a hybridization of a genetic algorithm with simulated annealing is beneficial and that the use of parallelism does not only introduce a speedup (as expected) in the algorithms, but also allows to improve the quality of the solutions found.

1 Introduction

In this paper, we perform a comparative study of several heuristics with respect to a traditional genetic algorithm in the design of combinational logic circuits. Despite the considerable amount of work currently available on the use of genetic algorithms and evolution strategies to design combinational logic circuits in the last few years (see for example [5, 13]), there have been few attempts to compare different heuristics in this problem. The main motivation for such a comparative study is to analyze if a certain type of heuristic (mainly hybrid approaches) could be more suitable for this type of problems.

Previous work has found, among other things, that designing combinational logic circuits is highly sensitive to the encoding [10], and to the degree of interconectivity allowed among gates [19]. There have also been studies on the fitness landscapes of these problems which are apparently rather simple but turn out to be quite difficult for any evolutionary algorithm [14, 18]. However, this sort of analysis has been conducted only on a single type of heuristics (e.g., a genetic algorithm or an evolution strategy).

Additionally, given the scalability problem associated with the design of combinational logic circuits using evolutionary algorithms, the use of parallelism seems a vital issue. Remarkably, however, few studies available in the literature have considered parallelism.

This paper presents a comparative study among a traditional genetic algorithm, and three heuristics that have local search capabilities. The rationale behind adopting these approaches is precisely to see if the design of combinational logic circuits (operating on a binary encoding) can benefit from local search strategies that are not included in a traditional genetic algorithm. For the study, we used both serial and parallel versions of each algorithm, so that we could analyze if the use of parallelism brings any benefits in terms of performance other than the obvious computational speedup.

The remainder of the paper is organized as follows. In Section 2, we provide a brief description of the approaches and the circuit encoding adopted in our study. Section 3 contains the examples and the results of the comparative study. Then, there is a further discussion on the results ob-

tained in Section 4. Finally, we provide some conclusions and possible paths of future research in Section 5.

2 Description of the approaches used

In this paper, we compare four heuristics in the design of combinational logic circuits:

1. A genetic algorithm (GA) with binary representation such as the one described in [3].

2. A CHC [7], which is a non-traditional GA which combines an elitist selection strategy (i.e., the approach always preserves the best individuals found so far) with a highly disruptive recombination (*HUX*, which is a variant of uniform crossover). Certain highly disruptive crossover operators provide more effective search in many problems, which represents the core idea behind the CHC search method. This algorithm also introduces a bias against mating individuals which are too similar (this is called *incest prevention* [8]). Mutation is not performed and, instead, a *restart* process reintroduces diversity whenever convergence is detected.

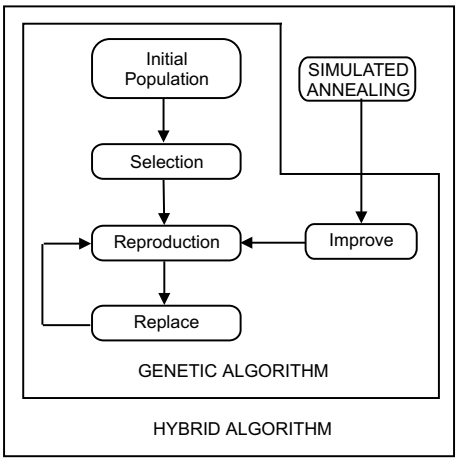

Figure 1. Hybrid scheme 1 (GASA1).

3. A hybrid called GASA1, where a GA uses simulated annealing (SA) as an evolutionary operator (see Fig. 1). The simulated annealing algorithm (SA) was first proposed in 1983 [9] based on a mathematical model originated in the mid-1950s [11]. SA is a stochastic relaxation technique that can be seen as a hill-climber with an internal mechanism to escape local optima. To allow escaping from a local optimum, moves that increase the energy function are accepted with a decreasing probability $\exp(-\delta/T)$, where T is a parameter called the "temperature" [6].

   The rationale for this selection of algorithms is that, while the GA locates "good" regions of the search space (exploration), SA allows for exploitation in the best regions found by its partner.

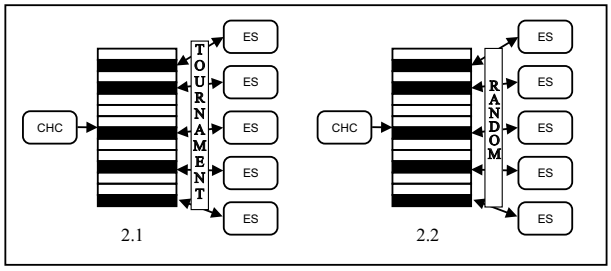

Figure 2. Hybrid scheme 2 (GASA2).

4. A second hybrid scheme called GASA2, which executes a GA until the algorithm completely finishes. Then the hybrid selects some individuals from the final population and executes a SA algorithm over them. Concretely, we analyze a version that uses a tournament selection (see Fig. 2)).

   Both in GASA1 and GASA2, we update the temperature of the simulated annealing component using the Fast SA (or FSA, for short) scheme [17]:

$$T_k = \frac{T_0}{1+k} \tag{1}$$

All of the previous approaches adopt the matrix representation proposed by Louis [10], and used in our previous work [3, 4] as shown in Figure 3. This matrix is encoded as a fixed-length string of integers from 0 to $N-1$, where N refers to the number of rows allowed in the matrix. Each of these integer values are encoded as binary numbers in the experiments reported within.

More formally, we can say that any circuit can be represented as a bidimensional array of gates $S_{i,j}$, where j indicates the *level* of a gate, so that those gates closer to the inputs have lower values of j. (Level values are incremented from left to right in Figure 3). For a fixed j, the index i varies with respect to the gates that are "next" to each other in the circuit, but without being necessarily connected. Each matrix element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE[1].) that receives its 2 inputs from any gate at the previous column as shown in Figure 3. Although our GA implementation allows gates with more inputs and these inputs might come from any previous level of the circuit, we limited ourselves to 2-input

[1]WIRE basically indicates a null operation, or in other words, the absence of gate, and it is used just to keep regularity in the representation used by the GA that otherwise would have to use variable-length strings.

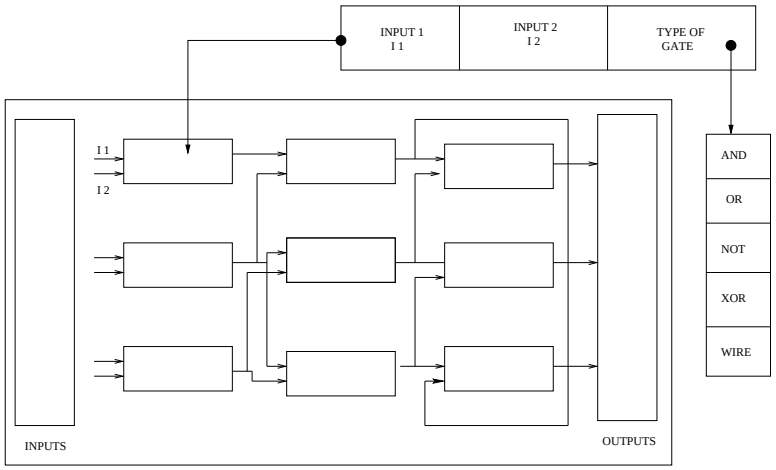

Figure 3. Matrix used to represent a circuit. Each gate gets its inputs from either of the gates in the previous column. Note the encoding adopted for each element of the matrix as well as the set of available gates used.

gates and restricted the inputs to come only from the previous level. This was mainly done to allow direct comparisons with our previous approaches.

A chromosomic string encodes the matrix shown in Figure 3 by using triplets in which the 2 first elements refer to each of the inputs used, and the third is the corresponding gate from the available set.

3 Comparison of Results

We compared our binary GA with respect to CHC, GASA1 and GASA2 both in a serial and a parallel version. For our parallel implementation, we used a kind of decentralized distributed evolutionary algorithm [1] in which separate subpopulations evolve independently and sparsely exchange one individual with a given frequency. The selection of the emigrant is through binary tournaments and the arriving immigrant replaces the worst one in the population only if the new one is better than this current worst individual.

The most relevant aspects that were measured in this comparison were the following: best fitness value obtained (we call this **opt**), the number of times that the approach found the best fitness value (we call this **hits**), the average fitness (called **avg**), and the average number of fitness function evaluations required to find the best fitness value (**#evals**).

Since our main goal was to analyze the behavior of different heuristics and the impact of parallelism, no particular effort was placed in fine-tuning the parameters for each of the circuits tried. The population sizes, mutation and crossover rates used correspond to the values previously reported for a traditional (binary) GA [3].

All of our experiments were conducted on PCs with a Pentium III processor running at 550 MHz and with 128 MB of RAM.

3.1 Example 1

Z	W	X	Y	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Table 1. Truth table for the circuit of the first example.

Our first example has 4 inputs and one output, as shown in Table 1. In this case, the matrix used was of size 5×5.

Algorithm	popsize	Xover prob.	Mut. prob.	Others
GA	320	0.6	0.00667	NONE
CHC	320	0.6	0.00667	35% population restart (uniform mutation 0.7)
GASA1	320	0.6	0.00667	SA operator (prob. 0.01) 100 iters., Markov C.L. = 10
GASA2	320	0.6	0.00667	GASA1 + SA execution with 3000 iterations (MCL = 300)

Table 2. Parameters used for the experiments of the first example.

We performed 20 independent runs per algorithm per circuit per version (either serial or parallel) using the parameters summarized in Table 2.

Our comparison of results is shown in Table 3. In this case both GASA1 and GASA2 were able to converge to the best known solution for this circuit (which has 7 gates and a fitness of 34) [2]. Note that both GASA1 and GASA2 required more fitness function evaluations to reach their best fitness value, but their final solution were significantly better than the solutions found by the GA and by CHC. Also note that the parallel versions of GASA1 and GASA2 slightly increased the average fitness value and the number of hits. However, the average number of fitness function evaluations to find the best fitness value did not decrease in the parallel versions of GASA1 and GASA2, as it occurred for the parallel versions of the traditional GA and CHC.

Just to give an idea of how good is the solution found by GASA2, we show in Table 4 a comparison of the best solution found by GASA2 with respect to other approaches previously used to design the circuit of this first example. This second comparison is only in terms of the boolean expression found. Note that the n-cardinality GA (NGA) used the same parameters as its binary counterpart. We can see that GASA2 found a solution significantly better than the other approaches against which it was compared (the n-cardinality GA, Sasao's simplification technique based on the use of ANDs & XORs [15], and a human designer using Karnaugh maps).

3.2 Example 2

Our second example has 5 inputs and one output, as shown in Table 5. In this case, the matrix used was of size 6×7 (for guidelines regarding how to setup the matrix size to be adopted, see [4]).

We performed 20 independent runs per algorithm per circuit per version (either serial or parallel) using the parameters summarized in Table 6.

Our comparison of results is shown in Table 7. Again, GASA2 found the best solution, but in this case, the parallel version produced a slightly better result than its serial

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

Table 5. Truth table for the circuit of the second example.

Algorithm	sequential				parallel			
	opt	hits	avg	#evals	opt	hits	avg	#evals
GA	31	10%	15.75	96806	33	5%	18.15	79107
CHC	27	5%	15.02	107680	32	5%	16.4	75804
GASA1	34	10%	23.2	145121	34	20%	25.46	151327
GASA2	34	10%	24.24	147381	34	30%	27.76	155293

Table 3. Comparison of results between a binary GA, CHC, GASA1 and GASA2 for the first example.

GASA2	NGA	HD 1	Sasao
$F = ((WX + (W \oplus Y)) \oplus ((Y + X)' + Z)$	$F = (WYX' \oplus ((W + Y) \oplus Z \oplus (X + Y + Z)))'$	$F = ((Z'X) \oplus (Y'W')) + ((X'Y)(Z \oplus W'))$	$F = X' \oplus Y'W' \oplus XY'Z' \oplus X'Y'W$
7 gates	10 gates	11 gates	12 gates
1 AND, 3 ORs, 2 XORs, 1 NOT	2 ANDs, 3 ORs, 3 XORs, 2 NOTs	4 ANDs, 1 OR, 2 XORs, 4 NOTs	3 XORs, 5 ANDs, 4 NOTs

Table 4. Comparison of the best solutions found by GASA2, the n-cardinality genetic algorithm (NGA) [3], a human designer using Karnaugh Maps (HD 1), and Sasao [15] for the circuit of the first example.

Algorithm	popsize	Xover prob.	Mut. prob.	Others
GA	600	0.6	0.00667	NONE
CHC	600	0.6	0.00667	35% population restart (uniform mutation 0.7)
GASA1	600	0.6	0.00667	SA operator (prob. 0.01) 500 iters., Markov C.L. = 50
GASA2	600	0.6	0.00667	GASA1 + SA execution with 10000 iterations (MCL = 1000)

Table 6. Parameters used for the experiments of the second example.

Algorithm	sequential				parallel			
	opt	hits	avg	#evals	opt	hits	avg	#evals
GA	60	5%	36.52	432170	62	10%	41	345578
CHC	58	15%	29.85	312482	61	5%	28.95	246090
GASA1	63	40%	45.12	694897	65	5%	50.62	593517
GASA2	64	10%	47.3	720106	65	10%	52.87	609485

Table 7. Comparison of results between a binary GA, CHC, GASA1 and GASA2 for the second example.

counterpart. Note also that the average fitness was increased both for GASA1 and GASA2 in their parallel versions, but the best solution was found only 10% of the time in the case of GASA2 (i.e., the hit rate did not improve in the parallel version of GASA2). Also note that in this case the use of parallelism decreased the average number of evaluations required to find the best possible fitness value produced by each of the algorithms under study.

GASA2	NGA	HD 1
$F = ((CA + B)(C + A) + D)$ $((CA + B)(C + A)E')'$	$F = ((A + C)(A + B)(D \oplus E)$ $(B + C) \oplus X')'$	$F = D((B + C)' + A'(BC)')$ $+E(A(B + C) + BC)$
9 gates	10 gates	12 gates
4 ANDs, 3 ORs 2 NOTs	3 ANDs, 3 ORs, 2 XORs, 2 NOTs	5 ANDs, 4 ORs, 3 NOTs

Table 8. Comparison of the best solutions found by GASA2, the n-cardinality genetic algorithm (NGA) [3], and a human designer using Karnaugh Maps (HD 1) for the circuit of the second example.

To have an idea of how good is the solution found by GASA2, we show in Table 8 a comparison of the best solution found by GASA2 with respect to other approaches previously used to design the circuit of the second example. This second comparison is only in terms of the boolean expression found. In this case, GASA2 improved the best solution found both by the NGA and by a human designer (using Karnaugh maps).

It is also important to mention that the best solution found by GASA2, which has 9 gates, is not the best possible solution for this circuit (there is another one with only 7 gates: $F = (A + BC)(D \oplus E)(B + C) \oplus D)$, which can be obtained with genetic programming [16]). However, as indicated before, no attempt was made to fine-tune the parameters of the algorithms used as to achieve a better solution.

3.3 Example 3

Our third example has 4 inputs and 3 outputs, as shown in Table 9. In this case, the matrix used was of size 6×7.

We performed 20 independent runs per algorithm per circuit per version (either serial or parallel) using the parameters summarized in Table 10.

Our comparison of results is shown in Table 11. In this case, both GASA1 and GASA2 found the best solution reported in the literature for this circuit [2], which has 9 gates and a fitness of 81. However, note that GASA2 had a better hit rate (in the parallel version). In this case, the use of parallelism produced a noticeable increase in the average fitness of GASA1 and GASA2, but the best solution was only rarely found. It is also interesting to see how GASA1 and GASA2 both have a computational cost of twice the traditional GA. Also note that, as in the previous example, in this case the use of parallelism decreased the average number of evaluations required to find the best possible fitness value produced by each of the algorithms under study.

We show in Table 12 a comparison of the best solution found by GASA2 with respect to other approaches previously used to design the circuit of the third example. This second comparison is only in terms of the boolean expression found. In this case, GASA2 again improved the best solution found by two human designers (one using Karnaugh maps and the other one using the Quine-McCluskey method), and by the NGA.

A	B	C	D	F1	F2	F3
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Table 9. Truth table for the circuit of the third example.

3.4 Example 4

Our fourth example is a 2-bit multiplier which has 4 inputs and 4 outputs, as shown in Table 13. A matrix of 5×5 was adopted in this case.

We performed 20 independent runs per algorithm per circuit per version (either serial or parallel) using the parameters summarized in Table 14.

Our comparison of results is shown in Table 15. In this case, GASA2 found the best solution reported in the literature for this circuit [2], which has 7 gates and a fitness value of 82. In this case, the use of parallelism produced

Algorithm	popsize	Xover prob.	Mut. prob.	Others
GA	600	0.6	0.00395	NONE
CHC	600	0.6	0.00395	35% population restart (uniform mutation 0.7)
GASA1	600	0.6	0.00395	SA operator (prob. 0.01) 100 iters., Markov C.L. = 10
GASA2	600	0.6	0.00395	GASA1 + SA execution with 10000 iterations (MCL = 1000)

Table 10. Parameters used for the experiments of the third example.

Algorithm	sequential				parallel			
	opt	hits	avg	#evals	opt	hits	avg	#evals
GA	71	10%	51.2	552486	76	15%	54.5	498512
CHC	64	20%	47.3	362745	70	5%	49.3	252969
GASA1	78	35%	70	1090472	81	5%	76.1	963482
GASA2	78	5%	69.32	1143853	81	10%	77.9	1009713

Table 11. Comparison of results between a binary GA, CHC, GASA1 and GASA2 for the third example.

GASA2	NGA	HD 1	HD 2
$F1 = ((B \oplus D) + (A \oplus C))'$	$F1 = ((B \oplus D) + (A \oplus C))'$	$F1 = (A \oplus C)'(B \oplus D)'$	$F1 = (A \oplus C)'(B \oplus D)'$
$F3 = ((B \oplus D) + (A \oplus C))(((A \oplus C) + (A \oplus B)) \oplus C))$	$F3 = ((B \oplus D) + (A \oplus C))((D + (A \oplus C))' + (A' + C)')$	$F3 = BD'(A + C') + AC'$	$F3 = (F1 + F2)'$
$F2 = F3 \oplus ((B \oplus D) + (A \oplus C))$	$F2 = ((B \oplus D) + (A \oplus C)) \oplus ((B \oplus D) + (A \oplus C))((D + (A \oplus C))' + (A' + C)')$	$F2 = B'D(A' + C) + A'C$	$F2 = A'C + (A \oplus C)'(B'D)$
9 gates	12 gates	19 gates	13 gates
3 XORs, 3 ORs, 2 ANDs, 2 NOTs	3 XORs, 4 ORs, 1 AND, 4 NOTs	2 XORs, 4 ORs, 7 ANDs, 6 NOTs	2 XORs, 2 ORs, 4 ANDs, 5 NOTs

Table 12. Comparison of the best solutions found by GASA2, the n-cardinality GA (NGA), and two human designers (HD 1 and HD 2) for the circuit of the third example.

only a slight increase in the average fitness of GASA1 and GASA2, but allowed GASA2 to converge to the best solution reported in the literature. It is also interesting to see how GASA1 and GASA2 both have a computational cost much higher than the traditional GA. Note however, that the parallel version of the parallel GA was able to converge to a better solution than the parallel version of GASA1, although the average fitness of the GA was still slightly below GASA1.

We show in Table 16 a comparison of the best solution found by GASA2 with respect to other approaches previously used to design the circuit of the fourth example. This second comparison is only in terms of the boolean expression found. In this case, GASA2 again improved the best

Algorithm	popsize	Xover prob.	Mut. prob.	Others
GA	600	0.6	0.00667	NONE
CHC	600	0.6	0.00667	35% population restart (uniform mutation 0.7)
GASA1	600	0.6	0.00667	SA operator (prob. 0.01) 500 iters., Markov C.L. = 50
GASA2	600	0.6	0.00667	GASA1 + SA execution with 10000 iterations (MCL = 1000)

Table 14. Parameters used for the experiments of the fourth example.

Algorithm	sequential				parallel			
	opt	hits	avg	#evals	opt	hits	avg	#evals
GA	78	15%	71.85	528390	81	5%	76.3	425100
CHC	76	5%	72.75	417930	80	10%	74.2	246090
GASA1	78	25%	74.15	711675	80	20%	76.9	852120
GASA2	80	10%	75.4	817245	82	20%	78.75	927845

Table 15. Comparison of results between a binary GA, CHC, GASA1 and GASA2 for the fourth example.

GASA2	NGA	HD 1	HD 2	MIL
$C_0 = A_0B_0$	$C_0 = A_0B_0$	$C_0 = A_0B_0$	$C_0 = A_0B_0$	$C_0 = A_0B_0$
$C_1 = A_0B_1 \oplus A_1B_0$	$C_1 = A_1A_0 B_0B_1 \oplus (A_0B_1 + A_1B_0)$	$C_1 = A_0B_1 \oplus A_1B_0$	$C_1 = (B_1 + B_0) (A_1 + A_0) ((A_1A_0) \oplus (B_1B_0))$	$C_1 = A_1B_0 \oplus A_0B_1$
$C_2 = A_1B_1 \oplus (A_0B_0 A_1B_1)$	$C_2 = (A_0B_0 + A_1B_1) \oplus A_0B_0$	$C_2 = A_1B_1 (A_0B_0)'$	$C_2 = A_1B_1 (A_0B_0)'$	$C_2 = (A_0B_0)' (A_1B_1)$
$C_3 = A_0B_0 A_1B_1$	$C_3 = A_1B_1 A_0B_0$	$C_3 = A_1A_0 B_1B_0$	$C_3 = A_1B_1 A_0B_0$	$C_3 = (A_1B_0 \oplus A_0B_1)' (A_1B_0)$
7 gates	9 gates	8 gates	12 gates	9 gates
5 ANDs, 2 XORs	5 ANDs, 2 ORs, 2 XORs	6 ANDs, 1 XOR, 1 NOT	8 ANDs, 1 XOR, 2 ORs, 1 NOT	6 ANDs, 1 XOR, 2 NOTs

Table 16. Comparison of the best solutions found by GASA2, the n-cardinality GA (NGA), two human designers (HD 1 & HD 2), and Miller et al. [12] (MIL) for the circuit of the fourth example.

solution found by two human designers (one using Karnaugh maps and the other one using the Quine-McCluskey method), by the NGA and by the cartesian genetic programming of [12]. It should be mentioned that Miller et al. [12] considered their solution to contain only 7 gates because of the way in which they encoded their Boolean functions (the reason is that they encoded NAND gates in their representation). However, since we considered each gate as a separate chromosomic element, we count each of them, including NOTs that are associated with AND & OR gates. It is also worth noticing that Miller et al. [12] found their solution with runs of 3,000,000 fitness function evaluations each.

4 Discussion of Results

Although this is just a preliminary study, a few things can be inferred from our results. First, the hybridization of a genetic algorithm with simulated annealing seems to be

A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Table 13. Truth table for the 2-bit multiplier of the fourth example.

benefitial for designing combinational logic circuits (at least when compared to a traditional GA). From the two hybrids tried, GASA2 had a better performance. This is apparently due to its use of simulated annealing over the final population of a GA, which allows to focus the search on more specific regions (something hard to do with the traditional genetic operators).

On the other hand, despite our belief that the highly disruptive recombination operator of CHC would be benefitial in circuit design, our results indicate that this approach has the worst overall performance of all the heuristics tried. Apparently, the mating restrictions of CHC (called *incest prevention*) and its *restart* process were not sufficient to compensate the diversity introduced by a conventional mutation operator, and the approach had difficulties to converge to feasible solutions.

5 Conclusions and Future Work

The comparative study conducted in this paper has shown that the hybridization of a genetic algorithm with simulated annealing may bring benefits when designing combinational logic circuits. Emphasis is placed on the fact that the GA hybridized is a traditional one using binary encoding. Additionally, the use of parallelism also brought benefits in terms of the quality of solutions produced, but did not necessarily improve the hit rate (i.e., the number of times that an algorithm converged to its best possible solution). Note however, that the use of parallelism tended to decrease the average number of evaluations required by each algorithm to achieve their best possible fitness value. The exception was the first example, but this might be due to the relative simplicity of the circuit. In examples 2 and 3, parallelism clearly reduced this number of evaluations, and in the fourth example, it produced reductions for the GA and CHC. Nevertheless, a more in-depth study of the impact of parallelism in combinational circuit design remains to be an open research area.

As part of our future work, we are interested in using a population-based multiobjective optimization approach (the so-called MGA that we proposed in [2]) hybridized with simulated annealing. Intuitively, this sort of approach should produce better results when hybridized, since by itself is a very powerful search engine for combinational circuit design. However, we ignore the possible bias that could arise from combining the local search capabilities of simulated annealing with the population-based selection mechanism of the MGA. Alternatively, the use of Pareto-based selection mechanisms would also be an interesting matter of study.

Acknowledgements

The first author acknowledges support from CONACyT through project 32999-A. The second author acknowledges that this work has been partially founded by the Ministry of Science and Technology and FEDER under contract TIC2002-04498-C05-02 (the TRACER project). The last author acknowledges support from CONACyT through project I-39324-A.

References

[1] Enrique Alba and Marco Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[2] Carlos A. Coello Coello, Arturo Hernández Aguirre, and Bill P. Buckles. Evolutionary Multiobjective Design of Combinational Logic Circuits. In Jason Lohn, Adrian Stoica, Didier Keymeulen, and Silvano Colombano, editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170. IEEE Computer Society, Los Alamitos, California, July 2000.

[3] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Automated Design of Combinational Logic Circuits using Genetic Algorithms. In D. G. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algo-*

rithms, pages 335–338. Springer-Verlag, University of East Anglia, England, April 1997.

[4] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design*, 2(4):299–314, June 2000.

[5] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering. An International Journal*, 27(1):1–28, January 2001.

[6] Kathryn A. Dowsland. Simulated Annealing. In Colin R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, chapter 2, pages 20–69. John Wiley & Sons, 1993.

[7] Larry J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search when Engaging in Nontraditional Genetic Recombination. In Gregory E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[8] Larry J. Eshelman and J. David Schaffer. Preventing Premature Convergence in Genetic Algorithms by Preventing Incest. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 115–122, San Mateo, California, July 1991. Morgan Kaufmann Publishers.

[9] S. Kirkpatrick, C.D. Gellatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[10] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.

[11] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[12] J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 105–131. Morgan Kaufmann, Chichester, England, 1998.

[13] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35, April 2000.

[14] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part II. *Genetic Programming and Evolvable Machines*, 1(3):259–288, July 2000.

[15] Tsutomu Sasao, editor. *Logic Synthesis and Optimization*. Kluwer Academic Press, 1993.

[16] Eduardo Serna Pérez. Diseño de Circuitos Lógicos Combinatorios utilizando Programación Genética. Master's thesis, Maestría en Inteligencia Artificial, Facultad de Física e Inteligencia Artificial, Universidad Veracruzana, Enero 2001. (In Spanish).

[17] Harold Szu and Ralph Hartley. Fast Simulated Annealing. *Physical Letters A*, 122(3):157–162, 1987.

[18] Vesselin K. Vassilev, Terence C. Fogarty, and Julian F. Miller. Information Characteristics and the Structure of Landscapes. *Evolutionary Computation*, 8(1):31–60, Spring 2000.

[19] Vesselin K. Vassilev, Julian F. Miller, and Terence C. Fogarty. Digital Circuit Evolution and Fitness Landscapes. In *1999 Congress on Evolutionary Computation*, volume 2, pages 1299–1306, Washington, D.C., July 1999. IEEE Service Center.