ALGORITMO DIVISÃO UTILIZANDO A META-HEURÍSTICA *SIMULATED ANNEALING* APLICADO NA OTIMIZAÇÃO DE CIRCUITOS REVERSÍVEIS

Douglas Uka Rennó*, Alexandre A. A. de Almeida*, Gerhard W. Dueck†, Alexandre César Rodrigues da Silva*

*Universidade Estadual Paulista - UNESP
Faculdade de Engenharia de Ilha Solteira
Ilha Solteira, São Paulo, Brasil

†Universidade de New Brunswick - UNB
Faculdade de Ciencia da Computação
Fredericton, New Brunswick, Canadá

Emails: douglasukarenno@hotmail.com, aamaralalmeida@gmail.com, gdueck@unb.ca, acrsilva@dee.feis.unesp.br

Abstract— In this work two algorithms adapted for the optimization of reversible circuits were implemented, with the cost of the number of doors in the circuit. The simulated annealing algorithm used as a method of optimization, in which together the rewrite rules allows the change of the size of the circuit without changing its functionality and the Division algorithm, developed to solve the inefficacy of the simulated Annealing for circuits with more than 50 gates, aiming to avoid the undesirable growth of the circuit in the initial iterations. The functionality of the Division method is to apply the optimization method in small parts of the circuit at a time. For evaluation of the performance of the division algorithm, the heuristics Dynamic Template was used as a comparison. According to the results, there was a higher cost reduction in 15 circuits, the same cost in 25 circuits and worse costs in only 3 circuits.

Keywords— Simulated Annealing, Division, Optimization, Reversible circuit.

Resumo— Neste trabalho foram implementados dois algoritmos adaptados para otimização de circuitos reversíveis, tendo como custo a quantidade de portas no circuito. O algoritmo *Simulated Annealing* utilizado como método de otimização, em que junto as regras de reescrita permite a alteração do tamanho do circuito sem alterar a respectiva funcionalidade e o algoritmo Divisão, desenvolvido para resolver a inefiência do *Simulated Annealing* para circuitos com mais de 50 portas, tendo como objetivo evitar o crescimento indesejável do circuito nas iterações iniciais. A funcionalidade do método Divisão é aplicar o método de otimização em pequenas partes do circuito por vez. Para avaliação do desempenho do algoritmo Divisão, utilizou-se a heurística *Dynamic Template* como comparação. De acordo com os resultados, obteve-se maior redução de custo em 15 circuitos, mesmo custo em 25 circuitos e piores custos em apenas 3 circuitos.

Palavras-chave— *Simulated Annealing*, Divisão, Otimização, Circuito reversível.

1 Introdução

A lógica reversível apresenta um papel importante para o avanço da tecnologia e possivelmente se tornará predominante na computação em um futuro próximo. Por exemplo, o aumento na quantidade de transistores construídos em uma mesma área integrada.

De acordo com Lei de Moore(Gordon, 1998), a quantidade de transistores se dobra a cada 18 meses, consequentemente aumenta também a dissipação de energia. A dissipação de energia é uma das principais barreiras para o desenvolvimento de chips para computadores mais rápidos e eficientes. Conforme os princípios de Landauer (Landauer, 1961), a dissipação de energia é calculada a cada informação perdida causada pela utilização da lógica convencional (irreversível). A cada informação perdida tem-se uma dissipação de energia de $k.T.\ln(2)$ Joule, sendo k a constante de Boltzmann e T a temperatura ambiente. Com o intuito de resolver esse problema, Bennett (1973) afirmou ser possível diminuir ou até mesmo desaparecer com a dissipação de energia utilizando a lógica reversível (Yelekar et al., 2011).

A lógica reversível pode ser definida por uma função *booleana* que apresenta a mesma quantidade de entrada e saída e cada conjunto de entrada é mapeado para um único conjunto de saída, e vice-versa, para todas as combinações possíveis(Função bijetora).

O processo de aplicação da lógica reversível na construção de um circuito reversível para uma determinada função *booleana* é realizada em três etapas:

1. Transformar a função irreversível em reversível;
2. Construir o circuito reversível;
3. Otimizar o circuito reversível.

As etapas 1 e 2 representam a síntese da lógica reversível, tendo como objetivo a construção do circuito. Na literatura, apresentam-se vários métodos utilizados na síntese, por exemplo, para a etapa 1, os métodos $Greedy$, $Hungarian$ e XOR apresentado por Miller et al. (2009). Para a etapa

2, o método MDM e o método baseado no ciclo de permutação da função apresentados por Miller et al. (2003) e Datta et al. (2013), respectivamente.

Referente a etapa 3, vários métodos de otimização de circuitos também são disponibilizados na literatura, por exemplo, Rahman et al. (2015) que utilizou a heurística *Dynamic Template* para reduzir a quantidade de portas de circuitos reversíveis utilizando as regras de reescrita apresentadas por Soeken and Thomsen (2013).

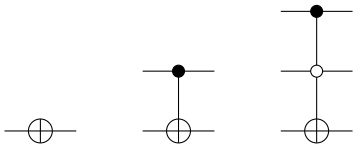
Neste trabalho são apresentados dois algoritmos, o *Simulated Annealing* e o Divisão. O *Simulated Annealing* é utilizado como um método adaptado junto as regras de reescrita para efetuar a diminuição da quantidade de portas de um circuito reversível, enquanto o algoritmo Divisão é utilizado para reduzir o espaço de busca que é aplicado o método de otimização. Sendo assim, o objetivo do método Divisão é efetuar a otimização em pequenas partes do circuito por vez.

O algoritmo Divisão foi desenvolvido para resolver o problema do algoritmo *Simulated Annealing*, na qual apresenta pior de desempenho conforme o aumento na quantidade inicial de portas no circuito reversível.

2 Definições

Uma função *booleana* é definida por $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, com $\mathbb{B} \in \{0,1\}$ e n, $m \in \mathbb{N}^*$. Sendo, os conjuntos de entrada definidos por $X = \{x_1, x_2, ..., x_n\}$ e os conjuntos de saída por $X' = \{x'_1, x'_2, ..., x'_m\}$. Uma função *booleana* é reversível se a quantidade de entradas e saídas são iguais e a função for bijetora.

As portas reversíveis utilizadas neste trabalho são NOT, CNOT e Toffoli, conforme são apresentadas na Figura 1.



(a) NOT (b) CNOT (c) Toffoli

Fig. 1: Portas reversíveis.

Uma porta reversível pode ser constituída por controles (positivo ou negativo) e, obrigatoriamente, o alvo. Os controles têm como funcionalidade "ativar" o alvo, enquanto o alvo, caso for ativo tem como objetivo inverter o valor da entrada. No caso da porta NOT, que não possui controle, o alvo é sempre ativo.

O controle positivo (ativo em 1) é representado pelo símbolo ●, o negativo (ativo em 0) é representado pelo símbolo ○ e o alvo é representado pelo símbolo ⊕.

Por exemplo, a porta Toffoli, apresentada na Figura 1 possui um controle positivo, um negativo e o alvo. Neste caso, o alvo é ativo apenas se o valor de entrada na linha que apresenta o controle positivo é 1 e na linha que contém o controle negativo for 0.

Um circuito reversível é uma sequência de portas reversíveis g_p com p $\in \mathbb{N}^*$. A partir desse circuito é possível encontrar um conjunto de saída para cada conjunto de entrada e vice-versa para todas as 2^n - 1 combinações possíveis da função.

Devido aos circuitos reversíveis serem constituídos por uma sequência de portas reversíveis, os circuitos reversíveis não podem apresentar *fan-out* e realimentação (Nielsen and Chuang, 2002). Na Figura 2 é apresentado o exemplo de um circuito reversível com 6 variáveis e 10 portas reversível.

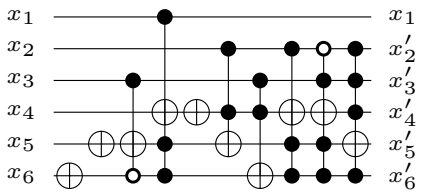


Fig. 2: Exemplo de um circuito reversível.

Como forma de alterar o tamanho do circuito, o método de otimização utiliza regras que tem por objetivo a troca de um circuito por outro. Essas regras foram apresentadas por Soeken and Thomsen (2013) e são conhecidas por regras de reescrita, pois não alteram a respectiva funcionalidade, ou seja, a tabela verdade. Entre essas regras, tem regras que aumentam, diminuem ou modificam o circuito.

Por exemplo, uma regra que diminui a quantidade de portas de um circuito é exemplificada na Figura 3, em que se apresentam dois circuitos. O circuito da esquerda é constituído por duas portas CNOT idênticas e o da direita não apresenta nenhuma porta reversível. Sendo assim, apesar de serem apresentarem quantidade de portas diferente, possuem a mesma funcionalidade.

Vale ressaltar que o custo utilizado é a quantidade de portas no circuito reversível.

As regras utilizadas nos algoritmos são apresentadas nas Figuras 3 a 9.

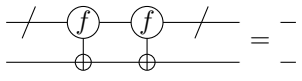


Fig. 3: Regra D1.

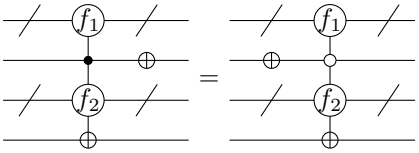


Fig. 4: Regra D2.

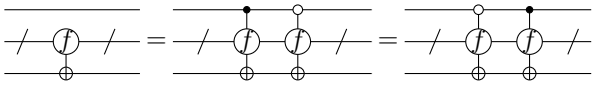


Fig. 5: Regra D3.

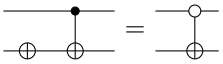


Fig. 6: Regra D4.

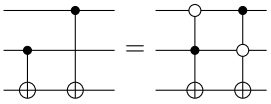


Fig. 7: Regra D5.

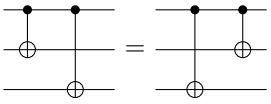


Fig. 8: Regra D6.

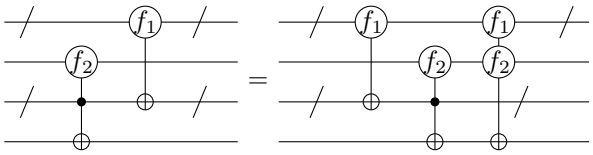


Fig. 9: Regra D7.

3 Algoritmo *Simulated Annealing*

O algoritmo desenvolvido foi baseado na meta-heurística *Simulated Annealing* e tem por objetivo reduzir a quantidade de portas no circuito utilizando as regras de reescrita apresentadas anteriormente.

A funcionalidade do algoritmo é apresentar duas soluções a cada iteração, a solução atual(S) que representa o circuito atual e a solução vizinha(S') que representa o novo circuito. A solução vizinha é criada para representar o circuito atual depois de aplicada uma das regras de reescrita. A aplicação de uma regra de reescrita ocorre da seguinte maneira: 1º listar as possíveis regras que podem ser aplicadas no circuito; 2º sortear apenas uma das regras listadas.

A cada camada de tempo são realizadas 'l' iterações, em que a cada iteração é calculada a função objetivo definida por Δ_{custo}, conforme é apresentada na Equação 1. O valor da função objetivo define se a solução vizinha é aceita.

$$\Delta_{custo} = S' - S \quad (1)$$

Se o algoritmo apresentar um $\Delta_{custo} \leq 0$, a solução vizinha é aceita. Por outro lado, para $\Delta_{custo} > 0$, contrário a heurístico *Greedy* que aceita apenas uma solução vizinha melhor, a meta-heurística *Simulated Annealing* possibilita que uma solução pior seja aceita, podendo assim evitar uma convergência prematura.

Na Figura 10 é apresentado o gráfico de como funciona a convergência da meta-heurística *Simulated Annealing*. Nota-se que a quantidade de portas no circuito aumenta para evitar ótimo local e assim podendo atingir o ótimo global.

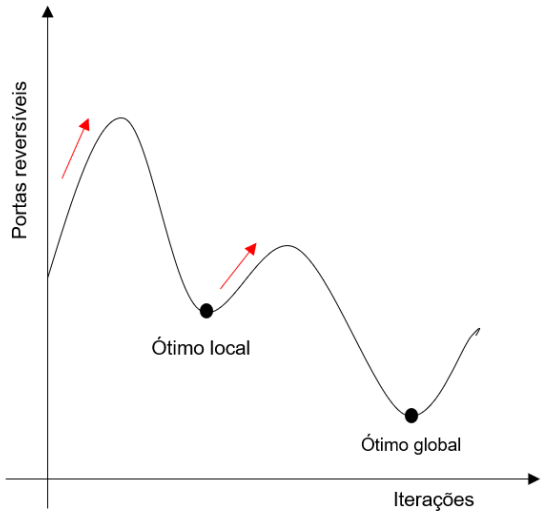


Fig. 10: Exemplo de convergência da *Simulated Annealin* na otimização de circuitos reversíveis.

A probabilidade de aceitar uma solução pior varia de acordo com a temperatura. Quanto maior a temperatura maior a probabilidade da pior solução ser aceita. Essa probabilidade é definida por $p(\Delta_{custo}) = e^{\frac{-\Delta custo}{T}}$. Dessa forma, circuitos com quantidade de portas acima de 50, possibilita que o circuito aumente de como descrontrolado nas iterações iniciais, sendo incapaz de otimizar e atingir um ótimo global. Esse é o problema encontrado com a aplicação do algoritmo *Simulated Annealing*.

A seguir são apresentadas os parâmetros utilizados no algoritmo *Simulated Annealing*:

- T: Temperatura;
- *Parada*: Critério de parada do algoritmo;

- l: Quantidade de iterações que são executadas a cada camada de tempo;
- α: Fator de redução da temperatura.

A seguir são apresentadas as funções utilizadas e as respectivas funcionalidades:

- **LimitaCircuito**(*circuito 1, circuito 2*): limita a quantidade de portas que o circuito 2 pode atingir. Se o circuito 2 apresentar um custo maior que o limite, o circuito 2 é substituído pelo circuito 1.
- **ListaRegras**(*circuito, x*): lista todas as x possibilidades de regras a serem aplicadas nas respectivas posições do *circuito*;
- **EscolheRegra**(*circuito 1, circuito 2, x*): escolhe 2 regras entre as x possibilidades listadas na função **ListaRegras** e as aplicam no circuito 1, assim gerando o circuito 2.

O algoritmo do *Simulated Annealing* é apresentado na Figura 11.

4 Algoritmo Divisão

O algoritmo Divisão foi desenvolvido para resolver o problema do método de otimização quando aplicados em circuitos reversíveis com mais de 50 portas. Tal problema é decorrente da probabilidade de regras a serem aplicadas no circuito que aumenta conforme o espaço de busca utilizado, podendo ocasionar perda na qualidade da otimização e maior tempo de execução.

A funcionalidade deste algoritmo é reduzir o espaço de busca do circuito e posteriormente aplicar o método de otimização. Vale ressaltar que utilizando um espaço de busca menor para aplicar o método de otimização, o circuito não aumentará de modo descontrolado nas iterações iniciais devido a probabilidade utilizada, o que dificultando a otimização.

Por exemplo, considerando um espaço de busca de 15 portas reversíveis, na primeira iteração são selecionadas as 15 primeiras portas do circuito atual e adicionadas em um circuito auxiliar, em que é aplicado o método de otimização. Depois de otimizado, no circuito auxiliar as 'x' últimas portas permanecem, enquanto as portas restantes são realocadas para o fim do circuito final. A partir da segunda iteração, são selecionadas as portas no circuito atual para serem adicionadas no fim do circuito auxiliar, porém a quantidade de portas é 15 - 'x'.

Vale ressaltar que a quantidade 'x' de portas são reutilizadas para serem otimizadas na próxima iteração. Esse processo foi utilizado para as portas que eram adjacentes no circuito atual, separadas ao serem adicionadas no circuito auxiliar, possam ser otimizadas.

Algoritmo 1: Algoritmo *Simulated Annealing*

```
1  procedimento Algoritmo Simulated_Annealing(circ);
2  //S = circuito atual;
3  //S' = novo circuito;
4  //M = circuito mínimo;
5  //*S = quantidade de portas do circuito atual;
6  //*S' = quantidade de portas do novo circuito;
7  //*M = quantidade de portas do circuito mínimo;
8  //x = quantidade de regras;
9  //lim = limite de portas reversíveis;
10 //α = fator de redução da temperatura;
11 k ← quantidade de portas reversíveis;
12 T ← 10 . *S;
13 Parada ← 0;
14 l ← 100 . *S;
15 M = S;
16 lim = min + 3;
17 início
18     enquanto Parada < 5 faça
19         otimizado ← Falso
20         para i = 1 até l faça
21             LimitaCircuito(circ, M);
22             ListaRegras(S, x);
23             EscolheRegras(S, S', x);
24             Δ_custo = *S' - *S;
25             se ( Δ_custo <= 0 ) então
26                 S = S';
27             fim
28             senão
29                 q ← Random(0, 1);
30                 se ( q < e^(-Δ_custo / T) ) então
31                     S = S';
32                 fim
33             fim
34             se ( *S < *M) então
35                 M = S;
36                 otimizado ← Verdadeiro;
37             fim
38         fim
39         T ← α T;
40         se !otimizado então
41             Parada ← Parada + 1;
42         fim
43         senão
44             Parada ← 0;
45         fim
46     fim
47 fim
```

Fig. 11: Algoritmo proposto para otimização de circuitos reversíveis.

A seguir são apresentadas as funções utilizadas e as respectivas funcionalidades:

- **AdicionaPortas**(*circuito 1, circuito 2*): adiciona todas as portas do *circuito 1* para o final do *circuito 2*;
- **AdicionaPortas**(*circuito*): deleta todas as portas do *circuito*;
- **CortaCircuito**(*circuito 1, pos1, pos2, circuito 2*): realoca as portas entre a *pos1* e *pos2* do *circuito 1* para o final do *circuito 2*.

O algoritmo Divisão é apresentado na Figura 12.

5 Resultados

Os algoritmos propostos neste trabalho foram programados utilizando a linguagem C++ na ferramenta RevKit desenvolvida por Soeken et al.

Algoritmo 2: Algoritmo Divisão

```
1  procedimento Algoritmo Divisão(circ, edb);
2  //edb = espaço de busca;
3  //circ = circuito atual;
4  //aux = circuito auxiliar;
5  //otimizado = circuito auxiliar 2;
6  //resultado = circuito final;
7  //*circ = quantidade de portas do circuito atual;
8  //*otimizado = quantidade de portas do circuito otimizado;
9  //x = quantidade de portas reutilizadas;
10 //pos1, pos2 = posição no circuito;
11 término ← Falso;
12 aux ← 0;
13 otimizado ← 0;
14 resultado ← 0;
15 início ← 1;
16 fim ← edb;
17 enquanto !término faça
18     se ( *circ >= fim ) então
19         término ← Verdadeiro;
20     fim
21     AdicionaPortas(aux, otimizado);
22     LimpaCircuito(aux);
23     CortaCircuito(circ, pos1, pos2, otimizado);
24     Algoritmo Simulated Annealing(otimizado);
25     CortaCircuito(otimizado, *otimizado - x,*otimizado, aux);
26     CortaCircuito(otimizado, 0, *otimizado - x - 1, resultado);
27     LimpaCircuito(otimizado);
28     início ← fim + 1;
29     fim ← início + edb - x - 1;
30 fim
31 AdicionaPortas(aux, resultado);
```

Fig. 12: Algoritmo proposto para divisão de circuitos reversíveis.

(2012). Para realizar os testes foram utilizados os circuitos reversíveis disponibilizados em *http://www.revlib.org/* (Wille et al., 2008).

Os parâmetros utilizados foram baseados no trabalho apresentado por Abdessaied et al. (2015) que tem como proposta a otimização do custo quântico de circuitos reversíveis utilizando a metaheurística *Simulated Annealing* e regras de reescrita. O fator de redução de temperatura utilizado neste artigo é de $\beta = 0{,}99$, diferente do que foi utilizado no trabalho de Abdessaied et al. (2015), em que se utilizou $\beta = 0{,}8$. Esse valor foi determinado de forma empírica, sendo escolhido o valor que apresentou maior redução na quantidade de portas no circuito. Os testes realizados utilizaram os espaços de busca 10, 15, 20,...,60. Nota-se que cada teste apresenta um acréscimo no espaço de busca de 5 portas. Outro parâmetro importante utilizado é o limite de portas que o circuito pode atingir, sendo testados de 3 à 10 portas a mais que o menor circuito encontrado. O valor do limite neste trabalho é de $min + 3$, sendo min a quantidade de portas do menor circuito encontrado (M). Esse parâmetro foi utilizado para que o circuito não aumente de modo inviável e posteriormente não dificulte a redução de custo.

Para o algoritmo Divisão, foram realizados testes para se determinar o melhor espaço de busca. A partir dos resultados obtidos, analisou-se que o espaço de busca com maior desempenho foi o de tamanho 15.

Para verificar o desempenho do algoritmo Divisão são apresentadas duas análises. Na primeira é demonstrado a eficiência da aplicação do algoritmo Divisão quanto ao tempo de execução e redução na quantidade de portas conforme o aumento na respectiva dimensão do circuito, enquanto na segunda, realiza-se a comparação dos resultados com o algoritmo Divisão junto ao *Simulated Annealing* e a heurística *Dynamic Template* desenvolvida por Rahman et al. (2015). Vale ressaltar que a dimensão de um circuito é a multiplicação entre a quantidade de variáveis e portas no circuito.

Na Tabela 1 é apresentada a primeira avaliação do algoritmo Divisão. De acordo com os dados apresentados é possível notar que conforme o aumento da quantidade de portas, o algoritmo *Simulated Annealing* apresenta um pior desempenho. Por outro lado, com a aplicação do algoritmo Divisão foi alcançado uma redução na quantidade de portas de 8,5% e um tempo de execução de 99,68% menor no exemplo nomeador de circuito *hwb9_123*.

Tab. 1: Avaliação do método *Simulated Annealing* com e sem a aplicação do algoritmo Divisão.

Circuito	$\#V$	$\#P_i$	$\#P_f$		$\%Red$		$Tempo\ (s)$	
			Alg. 1	*Alg. 2*	*Alg. 1*	*Alg. 2*	*Alg. 1*	*Alg. 2*
3_17_13	3	6	4	4	33,3	33,3	0,1	0,08
decod24-v0_38	4	6	5	5	16,7	16,7	0,22	0,12
hwb4_49	4	17	14	14	17,6	17,6	0,99	0,59
alu-v2_30	5	18	15	14	16,7	22,2	1,61	0,82
mod5adder_127	6	21	16	16	23,8	23,8	1,24	0,73
hwb7_62	7	331	316	297	4,5	10,3	739,24	34,58
hwb8_116	8	749	697	636	6,9	15,1	7411,7	106,52
hwb9_123	9	1959	1887	1725	3,7	11,9	68656,14	216,75

$\#V$: Quantidade de variáveis

$\#P_i$: Quantidade de portas reversíveis inicial

$\#P_f$: Quantidade de portas reversíveis final

$\#Alg.\ 1$: Algoritmo *Simulated Annealing*

$\#Alg.\ 2$: Algoritmo Divisão

Na Figura 13 é apresentado um gráfico para ilustrar a diferença de tempo de execução do algoritmo *Simulated Annealing* com e sem o método Divisão.

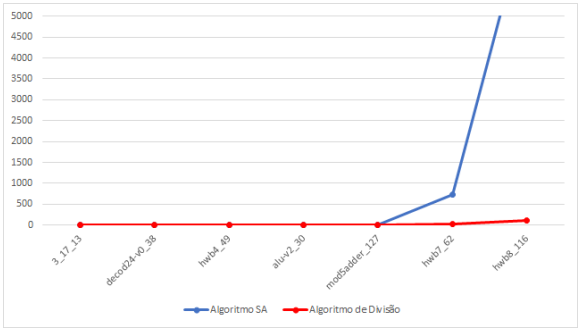


Fig. 13: Gráfico *Circuito x Tempo (s)*.

A segunda avaliação realizada foi para comparar o desempenho do algorimo Divisão junto ao *Simulated Annealing* como outro método de otimização. Na literatura foi encontrado método *Dynamic Template* apresentado por Rahman et al. (2015) e que também utiliza as regras de reescrita.

Foram realizados testes para 42 circuitos reversíveis, concluindo-se que a aplicação do algoritmo Divisão é eficiênte mesmo com o aumento da dimensão do circuito. Entre os 42 circuitos

utilizados, 15 testes apresentaram menor custo, alcançando uma redução de custo de até 44,44%. Na Tabela 2 são apresentados os resultados adquiridos de acordo com o respectivo circuito.

Tab. 2: Avaliação do algoritmo Divisão junto ao *Simulated Annealing* comparado a heurística *Dynamic Template*.

Circuito	$\#V$	$\#P_i$	$\#P_{fd}$	$\#P_f$	$\%Red$		
					P_{fd}/P_i	P_f/P_i	P_f/P_{fd}
alu-v2_31	5	13	9	5	30,77	61,54	44,44
sym6_145	7	36	31	19	13,89	47,22	38,71
mod10_171	4	10	9	6	10,00	40,00	33,33
mod8-10_177	5	14	11	8	21,43	42,86	27,27
mod5d1_63	5	7	5	4	28,57	42,86	20,00
rd53_130	7	30	22	18	26,67	40,00	18,18
mod8-10_178	5	9	8	7	11,11	22,22	12,50
4_49_17	4	12	10	9	16,67	25,00	10,00
aj-e11_165	4	13	11	10	15,38	23,08	9,09
alu-v2_30	5	18	15	14	16,67	22,22	6,67
urf1_149	9	11554	7347	6937	36,41	39,96	5,58
urf5_158	9	10276	5578	5301	45,72	48,41	4,97
urf3_155	10	26468	15736	15037	40,55	43,19	4,44
urf2_152	8	5030	3274	3153	34,91	37,32	3,70
hwb8_116	8	749	638	636	14,82	15,09	0,31
3_17_13	3	6	4	4	33,33	33,33	0
3_17_14	3	6	5	5	16,67	16,67	0
decod24-v0_38	4	6	5	5	16,67	16,67	0
mod10_176	4	7	5	5	28,57	28,57	0
aj-e11_168	4	10	9	9	10,00	10,00	0
4_49_16	4	16	13	13	18,75	18,75	0
hwb4_49	4	17	14	14	17,65	17,65	0
4gt11-v1_85	5	4	3	3	25,00	25,00	0
4gt13-v1_93	5	4	3	3	25,00	25,00	0
4gt12-v1_89	5	5	4	4	20,00	20,00	0
4mod5-v0_19	5	5	3	3	40,00	40,00	0
4mod5-v1_24	5	5	4	4	20,00	20,00	0
mod5mils_65	5	5	3	3	40,00	40,00	0
mod5mils_71	5	5	3	3	40,00	40,00	0
4gt4-v0_72	5	6	5	5	16,67	16,67	0
alu-v2_32	5	7	6	6	14,29	14,29	0
alu-v2_33	5	7	6	6	14,29	14,29	0
4mod5-v1_23	5	8	4	4	50,00	50,00	0
4gt12-v0_87	5	10	9	9	10,00	10,00	0
4gt13_91	5	10	9	9	10,00	10,00	0
4gt4-v0_73	5	17	13	13	23,53	23,53	0
mod5adder_127	6	21	16	16	23,81	23,81	0
rd53_137	7	16	13	13	18,75	18,75	0
ham7_106	7	25	22	22	12,00	12,00	0
hwb7_62	7	331	294	297	11,18	10,27	-1,02
hwb9_123	9	1959	1701	1725	13,17	11,94	-1,41
rd53_131	7	28	12	13	57,14	53,57	-8,33

#V: Quantidade de variáveis

$\#P_i$: Quantidade de portas reversíveis inicial

$\#P_{fd}$ Quantidade de portas reversíveis final utilizando a heurística *Dynamic Template*

$\#P_f$: Quantidade de portas reversíveis final utilizando o Algoritmo Divisão

$\%Red$: Porcentagem de redução

6 Conclusões

Neste trabalho foram propostos dois algoritmos, o algoritmo *Simulated Annealing* e o algoritmo Divisão.

O algoritmo Divisão foi desenvolvido para resolver o problema do *Simulated Annealing* conforme a quantidade de portas no circuito aumenta, enquanto o algoritmo *Simulated Annealing* é o método de otimização escolhido.

A estrutura do método de otimização é baseada em regras de reescrita. Essas regras têm como objetivo a troca de um circuito por outro, com a restrição de não alterar a respectiva funcionalidade.

Foram realizados testes para provar a eficiência do algoritmo Divisão, concluindo-se que o algoritmo Divisão apresentou melhor desempenho, tanto no tempo de execução quanto na redução de portas no circuito.

Por fim, o algoritmo Divisão foi comparado com a heurística *Dynamic Template*. De acordo com os resultados, obteve-se maior redução de portas reversíveis em 15 entre os 42 circuitos, alcançando uma melhora de custo de até 44,44%.

7 Agradecimento

Os autores agradecem à CAPES pelo fomento nesta pesquisa, ao CNPq, processo n° 309193/2015-0 e a Pós-graduação em Engenharia Elétrica da Faculdade de Engenharia de Ilha Solteira - UNESP (Universidade Estadual Paulista) ao suporte concedido a este trabalho.

Referências

Abdessaied, N., Soeken, M., Dueck, G. W. and Drechsler, R. (2015). Reversible circuit rewriting with simulated annealing, *International Conference on Very Large Scale Integration - VLSI-SoC*, IEEE, pp. 286–291.

Bennett, C. H. (1973). Logical reversibility of computation, *IBM journal of Research and Development* **17**: 525–532.

Datta, K., Ghuku, B., Sandeep, D., Sengupta, I. and Rahaman, H. (2013). A cycle based reversible logic synthesis approach, *International Conference on Advances in Computing and Communications - ICACC*, IEEE, pp. 316–319.

Gordon, E. M. (1998). Cramming more components onto integrated circuits, *Proceedings of the IEEE* **86**: 1.

Landauer, R. (1961). Irreversibility and heat generation in the computing process, *IBM journal of research and development* **5**: 183–191.

Miller, D. M., Maslov, D. and Dueck, G. W. (2003). A transformation based algorithm for reversible logic synthesis, *Design Automation Conference*, IEEE, pp. 318–323.

Miller, D. M., Wille, R. and Dueck, G. W. (2009). Synthesizing reversible circuits for irreversible functions, *Euromicro Conference on Digital System Design, Architectures, Methods and Tools - DSD*, IEEE, pp. 749–756.

Nielsen, M. A. and Chuang, I. (2002). Quantum computation and quantum information.

Rahman, M. M., Soeken, M. and Dueck, G. W. (2015). Dynamic template matching with mixed-polarity toffoli gates, *International Symposium on Multiple-Valued Logic - ISMVL*, IEEE, pp. 72–77.

Soeken, M., Frehse, S., Wille, R. and Drechsler, R. (2012). Revkit: A toolkit for reversible circuit design., *Multiple-Valued Logic and Soft Computing* **18**: 55–65.

Soeken, M. and Thomsen, M. K. (2013). White dots do matter: rewriting reversible logic circuits, *International Conference on Reversible Computation*, Springer, pp. 196–208.

Wille, R., Große, D., Teuber, L., Dueck, G. W. and Drechsler, R. (2008). Revlib: An online resource for reversible functions and reversible circuits, *International Symposium on Multiple Valued Logic - ISMVL*, IEEE, pp. 220–225.

Yelekar, P. R., Chiwande, S. S. et al. (2011). Introduction to reversible logic gates & its application, *National Conference on Information and Communication Technology*, pp. 5–9.