Roteamento em Projeto de Circuitos: uma solução com Colônia de Formigas.

Leandro I. Pinto, Marcela Leite, Cristiano D. Vasconcellos, Marco A. Wehrmeister

[1] Universidade do Estado de Santa Catarina (UDESC)
Departamento de Ciência da Computação
Mestrado em Computação Aplicada
Joinville, SC – Brasil

{leandroip, dcc6ml, damiani, marcow}@joinville.udesc.br

Resumo. *O roteamento em circuitos eletrônicos é uma importante fase no projeto de sistemas embarcados. Sabe-se que o problema do roteamento em circuitos eletrônicos é NP-Completo. Este artigo descreve como é executada essa etapa comum em projetos de sistemas embarcados, discutindo algumas soluções para o tratamento desse problema em tempo polinomial. Alguns estudos relacionados são apresentados e avaliados buscando demonstrar a eficiência de seus algoritmos para tratar o problema em questão. Adicionalmente, este trabalho apresenta um algoritmo que utiliza a abordagem de colônia de formigas para o roteamento eficiente de circuitos eletrônicos. Através dos experimentos realizados, observou-se o comportamento do algoritmo para diferentes tamanhos de circuitos e a influência dos parâmetros sobre a solução encontrada e o seu tempo de execução. Os resultados mostraram a necessidade de se adaptar o algoritmo da formiga para o problema do roteamento em FPGAs, entre essas adaptações, destaca-se a inclusão de um novo parâmetro para aumentar a sua capacidade de exploração.*

1. Introdução

O projeto físico de circuitos eletrônicos é composto basicamente de duas fases: (i) a definição da localização (*placement*) dos componentes na placa de circuito; (ii) o roteamento dos caminhos entre estes componentes. Na fase de localização é determinada a posição de cada elemento lógico na placa e as entradas e saídas do circuito. A fase de roteamento é a criação da rede de comunicação entre os componentes, na qual os impulsos elétricos trafegarão para transmitir informações entre eles. O roteamento é feito em duas etapas: roteamento global e detalhado. No roteamento global são definidas rotas para todo o circuito. Por sua vez, no roteamento detalhado, são selecionados conjuntos específicos de rotas do roteamento global que serão utilizadas de fato para conectar os componentes do circuito elétrico [WOLF 2004].

O roteamento é uma fase importante do projeto de circuitos eletrônicos, pois irá influenciar diretamente no desempenho do sistema projetado. Alguns exemplos das principais restrições observadas para sistemas embarcados são o atraso na comunicação e o consumo de energia [TRIMBERGER 1995]. A forma como os elementos são conectados, pode introduzir atrasos na comunicação entre eles, o que irá impactar no projeto devido a sua natureza de tempo-real e a previsibilidade necessária para este tipo de sistema. Um roteamento ineficiente pode levar a um consumo de energia desnecessário.

Sendo este um fator chave, o roteamento dos caminhos deve ser otimizado para minimizar o consumo de energia e recursos necessários para o projeto [WOLF 2004].

Tanto a localização quanto o roteamento são tarefas complexas devido ao número de possibilidades e fatores que irão influenciar no projeto. Segundo Nam, Sakallah e Rutenbar (1999), uma das questões fundamentais nesta etapa é: *Se dada uma localização existe um roteamento para o circuito?* Atualmente não há um algoritmo eficiente para responder essa pergunta pois trata-se de um problema *NP-Completo* [WU 1993] e [WU 1994]. Este artigo discute algumas técnicas utilizadas para obter uma solução aproximada para esse problema em tempo polinomial.

Na área de Automação de Projetos Eletrônicos (EDA-*Eletronic Design Automation*) existem vários problemas, como o de roteamento de circuitos, que não possuem um algoritmo eficiente com uma solução de tempo polinomial. Este artigo descreve como é realizado o roteamento de circuitos em placas FPGA e apresenta uma proposta de um algoritmo para tratar desse problema de forma mais eficiente aplicando a abordagem de Colônia de Formigas. Foram realizados experimentos que mostram o tempo de execução do algoritmo proposto ao aumentar o tamanho do circuito projetado e a influência da variação de diferentes parâmetros. Os resultados obtidos indicam que os valores 3 e 4 para os parâmetros α e β, respectivamente, tornaram o algoritmo mais eficiente.

O restante deste trabalho é organizado da seguinte forma: é apresentada a complexidade computacional do problema de roteamento em placas FPGA na Seção 2. Trabalhos relacionados sobre este problema são apresentados na Seção 3. Na Seção 4 é apresentado o algoritmo proposto de Colônia de Formigas e adaptações neste para tratar o problema abordado. Na Seção 6 são apresentados os experimentos realizados e resultados alcançados com o algoritmo proposto. Por fim, são apresentadas as conclusões obtidas com o trabalho realizado.

2. Sobre o Problema de Roteamento de Circuitos

Com o avanço da tecnologia aumenta a complexidade da tarefa de projetar sistemas embarcados, pois requerem cada vez mais funcionalidades com o menor uso de recursos possível. Neste cenário, o roteamento em circuitos eletrônicos tem um papel importante: ajuda na tarefa de diminuir os recursos necessários para o projeto, maximizando o desempenho do sistema [BENINI 2002] e [GARRISON 2006].

Este artigo aborda o roteamento de circuitos para placas FPGAs (*Field Programmable Gate Array*). Segundo Wu et al. (1993,1994) para este caso o problema é provado ser *NP-Completo*, ou seja, não há uma solução conhecida que execute em tempo polinomial.

Uma FPGA é uma estrutura de duas dimensões, sendo que o roteamento global já vem pré-definido de fábrica. Neste circuito são permitidos apenas caminhos horizontais e verticais, que são selecionados do roteamento global. Os caminhos selecionados formam um canal de comunicação entre as unidades lógicas (CL), que são compostas por elementos lógicos (LE) e blocos lógicos configuráveis (CLBs) [WOLF 2004]. Os CL são conectados por meio de blocos de conexão (*C-blocks*) e blocos de roteamento (*S-blocks*) [NAM 1999]. Uma arquitetura genérica de uma FPGA é apresentada na Figura 1, na

qual os blocos L representam os CLs, os blocos S representam os *S-blocks* e os blocos C representam os *C-blocks*.

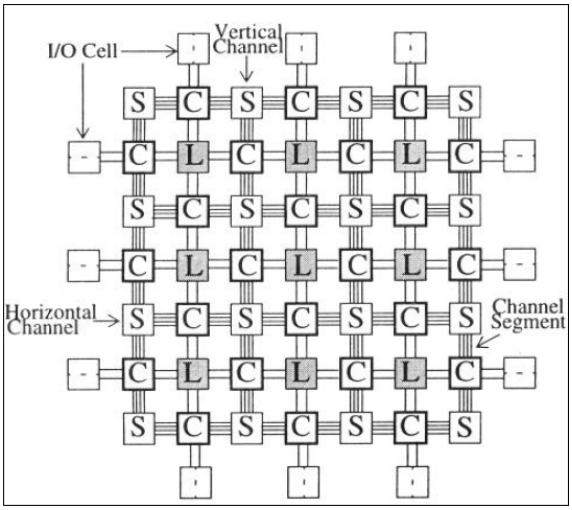

Figura 1. Estrutura genérica de uma FPGA. Fonte [NAM 1999].

Segundo Nam et al. (1999), a capacidade de roteamento pode ser determinada por três parâmetros: W indica o número de rotas verticais ou horizontais existentes; F_c indica o número de rotas de um CL que se conectam a um *C-block*; e F_s indica o número de possíveis rotas que um *S-block* possui para outros *C-blocks* além da entrada. A **rede** formada na placa será um fio conectando dois CLs através de uma rota ininterrupta usando uma sequência de *C-blocks* e *S-blocks* [NAM 1999].

Para a estrutura apresentada na Figura 1, temos $F_c = 2$, pois cada *C-block* (bloco C) possui duas conexões para um CL (bloco L) e $F_s = 3$, no qual cada *S-block* (bloco S) pode se conectar a outros três *C-blocks*, além de uma entrada. Na Figura 2 é apresentado um exemplo de uma rota selecionada no roteamento global entre os CLs L_A e L_B.

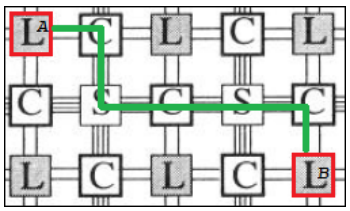

Figura 2. Exemplo de uma rota em uma placa FPGA. Adaptado de [NAM 1999].

O problema de roteamento em FPGAs é provado ser *NP-Completo*, conforme apresentado em [GREENE 1990], [WU 1993] e [WU 1996]. Em seu artigo, Wu et al. (1993) demonstra que o problema de roteamento em placas FPGA é *NP-Completo* reduzindo o problema de **coloração de grafos** em tempo polinomial a ele. Já Greene et al. (1990) apresenta uma redução do problema **Subset-Sum** em tempo polinomial ao problema de roteamento. Ele também apresenta um algoritmo de aproximação através de uma implementação de SAT. Segundo Greene et al. (1990) a complexidade para determinar um rota ótima é $O(M \cdot T^2 \cdot T!)$, sendo M o número de conexões e T o número de rotas possíveis.

3. Trabalhos Relacionados

Conforme Alexander e Robins (1996) pode-se modelar a estrutura de uma FPGA como um grafo não direcionado $G = (V, A)$, onde os vértices do grafo correspondem a caminhos disponíveis na FPGA. Cada aresta possui um peso, que pode corresponder a uma restrição imposta como o tempo de atraso para o caminho selecionado. O algoritmo proposto por Alexander e Robin (1996) utiliza a heurística de árvores geradoras mínimas para obter o melhor roteamento. Com este algoritmo, o roteamento utilizou menos 22% de rotas do roteamento global em relação a métodos tradicionais. Além disso, aquele algoritmo obteve segmentos de fio menores para criar as redes, melhorando assim, a utilização dos recursos. Entretanto, Alexander e Robins (1996) observaram que pode haver um grande aumento no tempo de processamento do algoritmo, conforme o tamanho do circuito aumenta, podendo demorar dias para obtenção de uma solução que atenda aos requisitos do sistema [ALEXANDER 1996].

Arora (2009) propõem em seu trabalho a utilização de um algoritmo de colônia de formigas para o roteamento de circuitos elétricos. O algoritmo proposto implementa a heurística de árvores geradoras mínimas, obtida através da colônia de formigas. Em seus experimentos, Arora (2009), verificou que para o roteamento em circuitos que permitem apenas canais horizontais e verticais, que é o caso das FPGAs, o algoritmo proposto apresentou uma otimização de recursos de 9% para o tamanho dos fios e 7% para a utilização de rotas [ARORA 2009].

Zhang et al. (2009) propõem a utilização do algoritmo de colônia de formigas com uma adaptação para tratar os componentes que trabalham com valores discretos e contínuos, como por exemplo transistores e indutores. No algoritmo proposto, cada componente da placa de circuito possui um peso diferente, permitindo assim, priorizar combinações e outros parâmetros do sistema como atraso e consumo de energia. Em seu artigo, apresenta uma comparação dos resultados obtidos com o algoritmo proposto em relação a abordagens tradicionais de algoritmos genéticos. O algoritmo proposto apresentou um desempenho superior quanto a combinação dos componentes para obter o melhor desempenho, com uma taxa final de aptidão de 198 contra 132 do algoritmo genético [ZHANG 2009].

O algoritmo de colônia de formigas também é proposto em Chopra e Singh (2011) para a solução do problema de roteamento em placas FPGA. A eficiência do algoritmo é demonstrada em relação a outras soluções para os problemas de otimização combinatória como GRASP (*Greedy Randomized Adaptive Search Procedure*) (Marques-Silva e Sakallah appud [CHOPRA 2011]) e ZChaff (Moskewicz et al appud [CHOPRA 2011]). O algoritmo de colônia de formigas levou em média 50% menos tempo de CPU para solução do problema.

A implementação apresentada por Chopra e Singh (2011) é uma adaptação do algoritmo da colônia de formigas com um resolvedor SAT. Sua proposta consiste em gerar um vetor de expressões booleanas que representam cada possibilidade de conexão da placa FPGA, no qual cada formiga irá resolver uma das expressões e se movimentar através destas. As soluções geradas por cada formiga, representam uma conexão válida ou não. Essa solução é avaliada e serve como entrada para as demais conexões a serem resolvidas. O critério de parada é quando a formiga obtém uma conexão válida.

A vantagem desta implementação é poder buscar a solução para várias *netlist* paralelamente, pois as expressões booleanas garantem que não ocorrerão sobreposições. Contudo, não é avaliada ou apresentada no trabalho, a influência dos parâmetros α e β, utilizados para alterar a influência do ferômonio e da distância, respectivamente.

A abordagem apresentada neste artigo utiliza uma colônia de formigas baseado no algoritmo apresentado por Solnon (2010) para solucionar o problema do caixeiro viajante. No problema do caixeiro viajante busca-se encontrar o menor caminho em um grafo completo que é semelhante ao roteamento em placas FPGA. Contudo, existem algumas diferenças que devem ser tratadas. Estas diferenças são apresentadas na Seção 5.

O objetivo deste algoritmo é encontrar uma solução aproximada da solução ótima para o roteamento de circuitos na estrutura de uma FPGA, considerando as características inerentes a sistemas embarcados. Através dos experimentos realizados, buscou-se investigar a influência na variação dos parâmetros sobre o comportamento do algoritmo.

4. Colônia de Formigas

O algoritmo da colônia de formigas (*Ant Colony Optimization - ACO*), proposto por Dorigo e Stützle (2004) é baseado no comportamento das formigas, que buscam o menor caminho entre o ninho e a comida. Ele é composto por vários agentes (as *formigas*) que trabalham de forma colaborativa para solucionar problemas combinacionais de otimização. Os agentes se auto-organizam através de modificações no ambiente através da representação do feromônio. Trata-se de um método probabilístico baseado em um processo estocástico [DORIGO 2004].

O problema do caixeiro viajante, um dos problemas *NP-Hard* mais estudados, foi o primeiro a ser atacado pelo método ACO. Neste problema, busca-se encontrar o menor caminho, passando por todos os nós de um grafo e retornar ao nó inicial. No ACO, dado um grafo completo, cada formiga da colônia inicia em um nó diferente e cria um ciclo independente passando por todos os nós uma única vez. O ponto chave está em definir apropriadamente uma regra de transição para escolher o próximo vértice a ser visitado [SOLNON 2010].

Uma abordagem tradicional para tratar esse problema é um algoritmo guloso, construindo um ciclo através do vizinho mais próximo [DASGUPTA 2009]. Entretanto, essa abordagem não levará a uma solução ótima. O ACO utiliza uma regra de transição baseada em um processo estocástico, no qual o próximo vértice a ser visitado depende de dois fatores: (i) Fator heurístico: Escolher o vértice mais próximo; (i) Fator Feromônio: Escolher baseado em experiências passadas.

Em cada iteração, cada formiga constrói um ciclo Hamiltoniano. Primeiro, escolhe-se um ponto inicial. Em seguida, escolhem-se os novos vértices a serem visitados até que todos eles sejam visitados, baseando-se na probabilidade da Equação 1, para se visitar até que todos tenham sido visitados [SOLNON 2010].

$$p_{ij} = \frac{[T_{ij}]^{\alpha}.[1/d_{ij}]^{\beta}}{\sum_{l \in Cand}[T_{il}]^{\alpha}.[1/d_{il}]^{\beta}} \quad (1)$$

Na Equação 1:

- $Cand$: Corresponde a um conjunto de vértices que podem ser visitados;
- T_{ij}: Quantidade de feromônio na aresta (i, j);
- d_{ij}: Comprimento da aresta (i, j);
- α e β: Parâmetros de balanceamento para definir qual fator terá mais influência;

Depois que todas as formigas construíram um ciclo Hamiltoniano, as trilhas de feromônio são atualizadas com as respectivas experiências. Primeiro o feromônio é evaporado em cada trilha, conforme Equação 2 [SOLNON 2010].

$$T_{ij} = T_{ij}(1 - \rho) \tag{2}$$

Onde ρ é um parâmetro de evaporação.

Em seguida cada formiga deposita feromônio nas trilhas que utilizou, sendo que a quantidade de feromônio depende do tamanho do caminho, esses valores são atualizados conforme Equação 3 [SOLNON 2010], na qual caminhos mais curtos recebem mais feromônio:

$$T_{ij} = T_{ij} + \frac{Q}{l_k} \tag{3}$$

Onde Q é uma constante próxima do tamanho do caminho ótimo e l_k é o tamanho do caminho gerado pela formiga k [SOLNON 2010].

Deve-se destacar que cada formiga inicia seu ciclo a partir de uma "cidade" diferente, fazendo com que cada formiga tenha um percurso diferente. Os experimentos apresentados na Seção 6 demonstram que essa abordagem é diferente para o roteamento de FPGAs, pois tem-se várias formigas partindo de um mesmo nó.

5. Implementação do ACO para Roteamento Detalhado em FPGAs

No problema do caixeiro viajante, o ACO considera a distância do nó atual até os nós adjacentes. Em geral, há uma maior probabilidade da formiga se movimentar para o nó que esteja mais próximo. Nas FPGAs, as distâncias entre os nós adjacentes são constantes, conforme mostrado na Figura 1. Por isso, o algoritmo proposto considera a distância dos nós adjacentes até o nó destino d_{iu}.

É interessante que a formiga escolha o nó adjacente que seja mais próximo fisicamente do seu nó de destino, pois o objetivo é conectar dois nós através do menor caminho. Portanto, a probabilidade da escolha de um determinado nó será calculada conforme a Equação 4.

$$p_{ij} = \frac{[T_{ij}]^{\alpha}.[1/d_{iu}]^{\beta}}{\sum_{l \in Cand}[T_{il}]^{\alpha}.[1/d_{iu}]^{\beta}} \tag{4}$$

Onde d_{iu} é a distância física do nó i ao nó de destino da formiga.

Outra característica do problema do caixeiro viajante é que cada formiga inicia em uma cidade diferente, sendo que cada formiga cria um ciclo diferente passando por todas as cidades. No roteamento dos FPGAs, deseja-se conectar apenas dois nós,

portanto várias formigas iniciarão a partir de um mesmo nó. Se todas as formigas que partiram de um mesmo nó utilizam a Equação 4 para criar seus percursos, então, durante uma determinada iteração, todas essas formigas farão o mesmo percurso. Isso dificulta o surgimento de diferentes soluções para o roteamento, tornando o algoritmo pouco exploratório.

Para tratar esse problema, foi introduzido no algoritmo um fator probabilístico σ, o qual permite que formigas com o mesmo nó de origem percorram caminhos diferentes. Ao invés de escolher o nó com maior probabilidade P como próximo destino, conforme mostrado na Equação 4, um dos nós possíveis é escolhido aleatoriamente. A probabilidade do nó deve satisfer $|P - p_{ij}| < \sigma$, no qual P é a melhor opção e p_{ij} é a probabilidade de escolha um determinado nó j. Esta é a principal diferença entre o algoritmo proposto neste trabalho e o apresentado por Solnon (2010).

Baseando-se na arquitetura proposta na Figura 1, cada *CL*, *C-block* e *S-block* representam um nó do grafo. Uma *netlist* contém todos os nós do tipo *CL* que devem ser conectados entre si, ou seja, representa as conexões entre os nós. Uma solução S para uma *netlist* é um conjunto que pode conter apenas nós do tipo *C-blocks* e *S-blocks*.

Esta solução consiste em conectar dois ou mais nós através de um único caminho. O objetivo de cada formiga é construir um caminho a partir de um vértice V_1 até um vértice V_2 passando pelas arestas existentes e nós permitidos conforme as restrições mencionadas anteriormente.

A Figura 3 mostra o algoritmo proposto para resolver o problema do caixeiro viajante, que foi baseado em Solnon (2010). Foram feitos ajustes no algoritmo original para atender as restrições inerentes ao roteamento de FPGAs: (i) um nível inicial T_0 de feromônio é atribuído a todas as arestas (ver linha 7); (ii) uma iteração busca uma solução para cada *netlist* individualmente (ver for_1 mostrado na linha 8).

Como pode ser observado, utiliza-se a Equação 4 (linha 17) e o parâmetro σ (linha 18) para permitir uma maior variação nas soluções geradas pelas formigas. A linha 9 apresenta um critério de parada $NetInvalid$ apenas para verificar quando a conexão entre os dois nós já foi concluída.

Outra restrição a ser observada é que, uma vez que um caminho foi utilizado para conectar dois *CLs*, ele não poderá mais ser reutilizado para conectar outros CLs. Dessa forma, cada *netlist* "consome" um conjunto de arestas que não podem ser reutilizadas em outras netlists.

É importante destacar que existe um número limitado de arestas entre cada um dos nós, conforme a flexibilidade da FPGA. Nos experimentos executados, considerou-se que existe apenas uma aresta entre nós adjacentes.

O próximo passo é realizar a evaporação do feromônio e sua posterior atualização pelas formigas que utilizaram cada aresta (ver linhas 22 à 28). A solução é então armazenada e as arestas utilizadas são removidas para não serem reutilizadas pelas próximas soluções (ver linhas 30 e 31).

```
1  Input:
2  G = (V, E) : graph with vertices of types (L, C ou S)
3  N = {(v_a, v_b), (v_c, v_d), ...} : Netlist, connections between V_L
4  S = {} : solutions set, one for each netlist
5  α, β, ρ, T_0, Q, nbAnts (ant quantity), σ: numeric parameters
6  begin
7     for each edge (i, j) ∈ E do T_ij ← T_0
8     for_1 each net in N
9          while_1 NetInvalid do
10           for each ant k ∈ 1, ..., nbAnts
11             put ant k on (choose a v ∈ net)
12             while ant k not in target
13               i ← actual ant position
14               Cand ← Neighboring vertices allowed
15               if Cand = {} then
16                  reset ant and Break
17               P ← max p_ij ∈ Cand with: p_ij = [T_ij]^α.[1/d_iu]^β / Σ_{l∈Cand}[T_il]^α.[1/d_iu]^β
18               select a random j ∈ Cand where |P − p_ij| < σ
19               move ant k to j
20             end while
21           end for
22           for each edge (i, j) ∈ E do T_ij ← T_ij.(1 − ρ)
23           for each ant k ∈ 1, ..., nbAnts do
24             l_k ← lenght of the path by ant k
25             for each edge(i, j) of the path build by ant k do
26               T_ij ← T_ij + Q/l_k
27             end for
28           end for
29        end while_1
30      S = S ∪ solution_k
31      remove edges_i,j ∈ solution from G
32    end for_1
33  return S
34 end
```

Figura 3. Algoritmo proposto.

6. Experimentos

A validação deste trabalho aconteceu na forma da implementação do algoritmo proposto e a sua execução usando alguns estudos de caso. Os experimentos realizados são descritos e discutidos nesta seção. O algoritmo proposto foi implementado em Python (v2.7) e simula a arquitetura de uma FPGA através um grafo que deve ser percorrido pelas formigas. Esse grafo é representado como uma matriz quadrada de nós conforme mostra a Figura 4.

Foram executados experimentos com este software, simulando diferentes valores para os parâmetros e diferentes tamanhos de circuitos. Para executar os testes foi utilizado

um computador com o sistema operacional Windows 7, equipado com processador AMD Phenom II x4 2.8 GHz e com 4 GB de memória RAM.

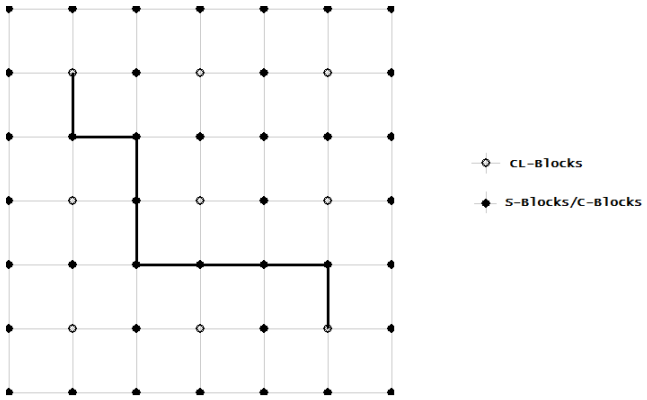

Figura 4. Exemplo de execução do programa.

Para cada conjunto de parâmetros e tamanho de circuito, os testes foram repetidos 10 vezes para possibilitar uma análise da execução. Dessa forma, foi possível avaliar o tempo necessário para obter a solução mais aproximada da ótima e os melhores valores para os parâmetros do algoritmo proposto. Os experimentos foram realizados em um circuito de 225 nós (matriz 15x15) com somente uma *netlist* conectando o CL superior esquerdo ao CL inferior direito. O parâmetro σ foi mantido em 0,15 em 70 experimentos repetidos 10 vezes cada e, em seguida, o parâmetro σ foi alterado para 0,25 e os mesmos experimentos foram repetidos. Os resultados desses experimentos são apresentados nas Figuras 5 e 6.

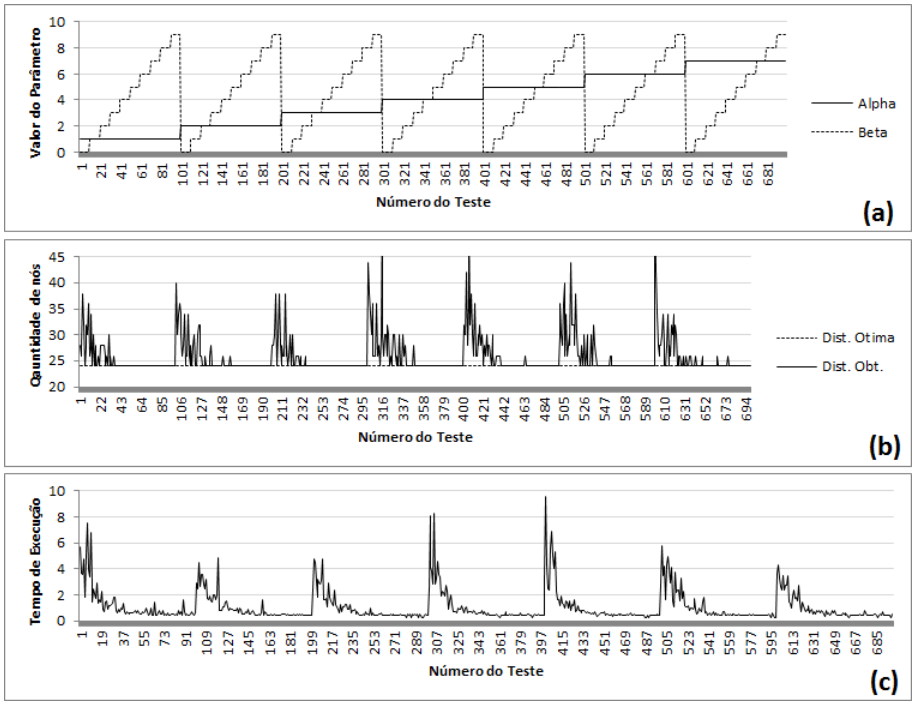

Figura 5. a) Valores dos parâmetros α e β. b) Comparação da solução encontrada em relação a ótima. c) Tempo de execução em segundos. Parâmetro $\sigma = 0,15$

Inicialmente, diferentes valores para os parâmetros α e β foram utilizados, sendo que α variou de 1 a 7, e β de 0 a 9, sendo que cada combinação de α e β foram repetidas 10 vezes. Na Figura 5 são apresentados os resultados obtidos nos experimentos executados. Nos gráficos, o eixo x indica número do experimento executado.

A Figura 5 (b) apresenta a qualidade da solução e a distância ótima que é 24, nota-se que quando $\beta < \alpha$, na Figura 5 (a), ou β é um valor próximo de α, o algoritmo não encontra soluções ótimas e tem o maior tempo de execução como mostra a Figura 5 (c). A medida que β aumenta em relação a α, como mostra a Figura 5 (a), o tempo de execução diminui e a qualidade da solução é ótima. Nota-se também, que os picos acima de 4 se tornam cada vez menores a medida que o parâmetro α aumenta.

A Figura 6 apresenta a mesma análise, porém nesses experimentos, o valor do parâmetro σ é 0,25. Observa-se que o algoritmo apresentou um desempenho inferior ao anterior, pois o parâmetro σ aumentou a aleatoriedade do algoritmo. Também foi observado que quando σ é menor que 0,12 aproximadamente, o algoritmo não encontra nenhuma solução quando o grafo apresenta obstáculos.

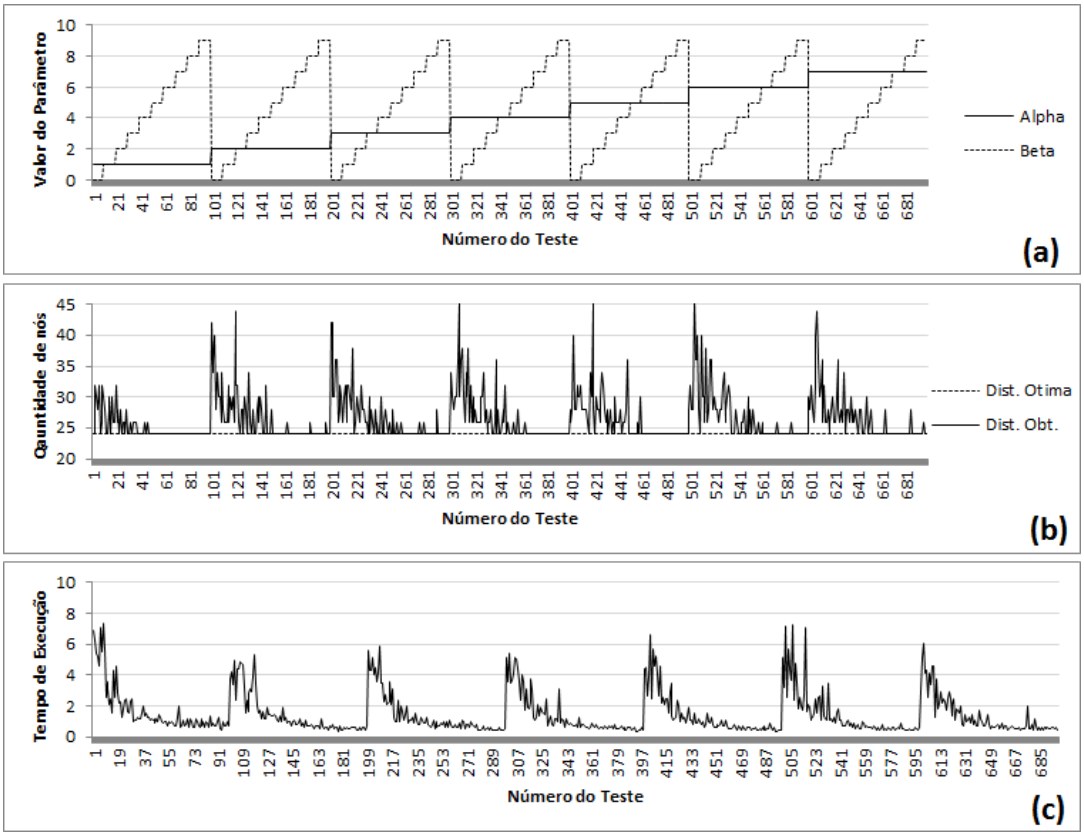

Figura 6. a) Valores dos parâmetros α e β. b) Comparação da solução encontrada em relação a ótima. c) Tempo de execução em segundos. Parâmetro $\sigma = 0,25$

Entretanto, ao aumentar o valor de σ o algoritmo tem seu desempenho reduzido, pois faz mais buscas por diferentes caminhos obtendo soluções melhores. A Figura 7 mostra o aumento do tempo de execução em relação ao aumento de σ.

A Figura 8 mostra a média e o desvio padrão dos tempos obtidos ao aumentar a quantidade de nós. Esse gráfico mostra que o tempo de processamento cresce de forma

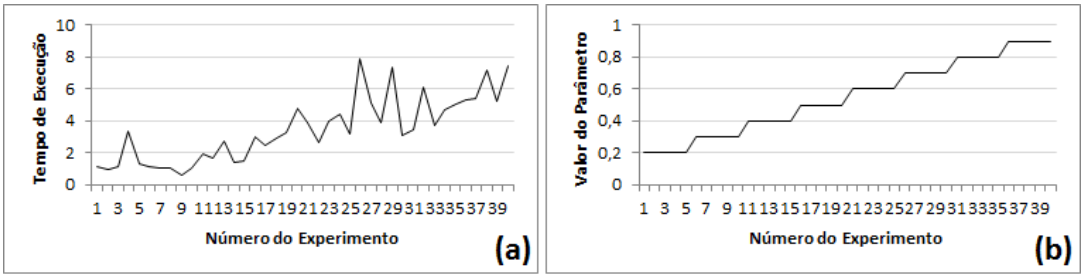

Figura 7. Desempenho do algoritmo em relação ao aumento do parâmetro σ em um grafo com obstáculos.

polinomial em relação ao aumento da entrada. Para esses experimentos foram utilizados os parâmetros $\alpha = 3$, $\beta = 4$ e $\sigma = 0,15$.

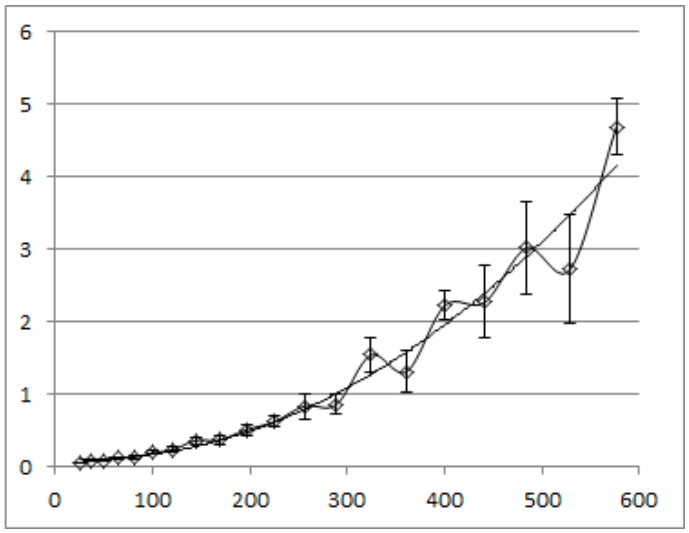

Figura 8. Desempenho do algoritmo em relação ao tamanho do circuito.

7. Conclusões

O projeto de sistemas embarcados possui muitos desafios, dos quais se destaca o roteamento em placas de circuito, que é considerado um problema *NP-Completo*. Este artigo buscou apresentar a problemática envolvida nesta fase de projeto, discutindo abordagens utilizadas pera minimizar o tempo computacional necessário para resolver este problema.

Este artigo apresentou uma solução para tratar a complexidade do roteamento em placas FPGAs, descrevendo as principais restrições de projeto. Um Algoritmo de Colônia de Formigas foi proposto como solução para tratar este problema. O algoritmo proposto baseou-se no algoritmo para a solução do problema do caixeiro viajante, adaptando-a para atender as regras do roteamento de placas FPGAs.

O comportamento do algoritmo proposto foi avaliado através de experimentos. Verificou-se que os parâmetros α, β e σ do algoritmo alteram significativamente o desempenho do mesmo.

Os parâmetros α e β controlam a influência do feromônio e a distância do nó ao destino, respectivamente. Por isso, quando β é muito maior que α o algoritmo pode apresentar um comportamento guloso, buscando sempre o nó que esteja mais próximo do destino.

Como trabalhos futuros sugere-se avaliar o tratamento dos requisitos não funcionais inerentes aos projetos de sistemas embarcados, como atraso e a quantidade de blocos utilizados para o roteamento, com os parâmetros identificados neste trabalho. Outros resultados também podem ser obtidos com uma variação fracionada dos parâmetros em vez de inteira.

Referências

ALEXANDER, M. J.; ROBINS, G. (1996). New Performance-Driven FPGA Routing Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1505–1517.

ARORA, T. (2009). Using Ant Colony Optimization for Routing in Microprocessors. Master's thesis.

BENINI, L.; DE MICHELI, G. (2002). Networks on Chips: A new SoC Paradigm. 2(4):70–78.

CHOPRA, V.; SINGH, A. (2011). Solving FPGA Routing using Ant Colony Optimization with Minimum CPU Time. *International Journal of Computer Science & Technology*, 2(4):223–226.

DASGUPTA, S.; PAPADIMITRIOU, C. V. U. (2009). *Algoritimos*. McGraw-Hill, 1th edition.

DORIGO, M.; STÜTZLE, T. (2004). *Ant Colony Optimization*. MIT, 1th edition.

GARRISON, J. (2006). How to achieve fast timing closure on FPGA designs.

GREENE, J. W.; ROYCHOWDHURY, V. P. K. S. G. A. (1990). Segmented Channel Routing. pages 567–572.

NAM, G.; SAKALLAH K. A.; RUTENBAR, R. A. (1999). Satisfiability-Based Layout Revisited:Detailed Routing of Complex FPGAs Via Search-Based Boolean SAT Automation. pages 167–175.

SOLNON, C. (2010). *Ant Colony Optimization and Constraint Programming*. Wiley, 1th edition.

TRIMBERGER, S. (1995). Effects of FPGA architecture on FPGA routing. pages 574–578.

WOLF, W. (2004). *FPGA-Based System Design*. Pearson Education, 4th edition.

WU, YL.; TSUKIYAMA, S. M.-S. M. (1993). Graph Based Analysis of FPGA Routing. 15(12):104–109.

WU, YL.; TSUKIYAMA, S. M.-S. M. (1994). On Computational Complexity of a Detailed routing Problem in Two-Dimensional FPGAs. pages 70–75.

WU, YL.; TSUKIYAMA, S. M.-S. M. (1996). Graph Based Analysis of 2-D FPGA Routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(1):33–44.

ZHANG, J.; CHUNG, H. S. L. A. W. (2009). Extended Ant Colony Optimization Algorithm for Power Electronic Circuit Design. *IEEE Transactions on Power Eletronics*, 24(1):147–162.