ColorAnt-RT: Algoritmo de Coloração de Grafo que utiliza Colônia de Formigas aplicado a Alocação de Registradores

Carla Negri Lintzmayer[1], Mauro Henrique Mulati[2] and Anderson Faustino da Silva[3]

[1]Instituto de Computação – Universidade Estadual de Campinas

[2]Departamento de Informática – Universidade Estadual de Maringá

[3]Departamento de Computação – Universidade do Centro-Oeste

{carla0negri,mhmulati}@gmail.com, anderson@din.uem.br

Abstract. *This paper presents the ColorAnt-RT algorithms for graph coloring problems, that was developed to be used in a register allocator. The experiments demonstrate that ColorAnt$_3$-RT is a promising option among the developed ones in finding good approximations for several graphs. Besides, the experiments also demonstrate that our register allocator outperforms the George-Appel register allocator.*

Resumo. *Este artigo apresenta os algoritmos ColorAnt-RT para o problema da coloração de grafos, os quais foram desenvolvidos para serem utilizados em um alocador de registradores. Os experimentos demonstram que ColorAnt$_3$-RT é uma boa opção dentre os desenvolvidos para encontrar boas aproximações para diversas classes de grafos. Além disto, os experimentos também demonstram que o alocador de registradores implementado possui um desempenho superior aquele obtido pelo alocador de registradores proposto por George e Appel.*

1. Introdução

Obter uma solução para o problema de coloração de grafos (PCG) consiste basicamente em encontrar uma quantidade k de cores que possam ser atribuídas aos vértices de forma que não existam vértices adjacentes com a mesma cor. Trivialmente, se um grafo G possui n vértices, então basta escolher $k = n$ cores; porém, o objetivo é encontrar o valor mínimo de k que respeite a restrição do problema, denominado número cromático do grafo e denotado por $\chi(G)$.

Uma aplicação real do k-PCG é vista na alocação de registradores. Neste problema, a solução não se restringe apenas a verificar se um grafo é k-colorível, mas também deve utilizar alguma heurística que possibilite "eliminar" as arestas conflitantes da melhor forma possível, já que é obrigatório colorir o grafo apenas com k cores. Contudo, um problema relacionado a algumas abordagens baseadas em coloração de grafos é o fato de utilizarem uma heurística simples, ocasionando um código de má qualidade, o que acarretará em um considerável tráfego entre processador e memória, ocasionando uma perda de desempenho.

Este trabalho reporta o projeto e desenvolvimento dos algoritmos heurísticos *ColorAnt*-RT para o k-PCG e sua aplicação como uma fase de um alocador de registradores

tradicional[1]. *ColorAnt*-RT, além de ser um algoritmo heurístico baseado em colônia de formigas [Dorigo and Stützle 2004], utiliza a busca tabu reativa como busca local, com o objetivo de melhorar a qualidade dos resultados.

Os resultados obtidos por *ColorAnt*-RT mostram que ele é uma boa opção para encontrar boas aproximações para diversas classes de grafos. Como também, uma boa opção para ser utilizado no processo de alocação de registradores. Em termos de valores que são efetivamente representados em memória e tamanho de código, utilizar uma heurística mais agressiva ocasiona um código com melhor qualidade.

O restante deste texto encontra-se organizado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados; a Seção 3 apresenta os algoritmos *ColorAnt*-RT e uma análise de desempenho destes; a Seção 4 apresenta uma aplicação do melhor *ColorAnt*-RT dentre os desenvolvidos, uma modificação do alocador de registradores proposto por George e Appel, bem como uma análise de desempenho deste novo alocador; e por fim, a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Coloração de grafos com colônia de formigas foi originalmente proposto com o algoritmo *ANTCOL* [Costa and Hertz 1997], no qual cada formiga tenta colorir o grafo com o menor valor de k possível, utilizando métodos construtivos. A diferença entre *ANTCOL* e *ColorAnt*-RT está no uso da probabilidade, que envolve o feromônio e a informação heurística: no *ANTCOL* é utilizada para escolher um novo vértice a ser colorido, e em *ColorAnt*-RT é utilizada para escolher a cor que irá colorir um vértice.

Shawe-Taylor e Zerovnik modelam cada formiga como um procedimento iterativo que tenta minimizar o número de conflitos [Shawe-Taylor and Zerovnik 2001]. No trabalho de Hertz e Zufferey, cada formiga colore um único vértice, de forma que a colônia inteira encontra apenas uma solução [Hertz and Zufferey 2006] e com base na informação heurística e na trilha de feromônio, as formigas andam pelo grafo. Em um trabalho mais recente de Plumettaz *et al*, o *ALS-COL* (*Ant Local Search*) [Plumettaz et al. 2010], cada formiga é uma busca local derivada da busca tabu. A diferença destes trabalhos para *ColorAnt*-RT está no fato de em *ColorAnt*-RT as formigas não serem procedimentos iterativos, nem buscas locais.

Alocador de registradores baseada em coloração de grafo foi originalmente proposta por Chaitin [Chaitin 1982]. Contudo, o projeto mais bem sucedido foi o desenvolvido por Briggs *et al.* [Briggs et al. 1994], que reprojetou o alocador de Chaitin para adiar as decisões de *spill*. George e Appel [George and Appel 1996] projetaram um alocador que utiliza os passos de simplificação de Chaitin e o passo de *coalescing* conservativo de Briggs. Estes trabalhos formam a base do alocador de registradores no qual *ColorAnt*-RT foi utilizado.

O trabalho mais próximo do alocador de registradores que utiliza *ColorAnt*-RT (CARTRA) é o trabalho desenvolvido por Wu e Li [Wu and Li 2007] que propõe um

[1]Projeto de Iniciação Científica, Processo 6500/2010, Título: "Algoritmo baseado em colônia de formigas para solução do problema de coloração de grafos", Período: 01/08/2010 a 31/07/2011, Discente: Carla Negri Lintzmayer, Orientadores: Mauro Henrique Mulati e Anderson Faustino da Silva, Universidade Estadual de Maringá.

algoritmo heurístico híbrido que combina diversas idéias de alocadores de registradores clássicos, com algoritmos evolucionários e busca tabu.

3. O Algoritmo *ColorAnt*-RT

Para solucionar o k-PCG, foi desenvolvido inicialmente o algoritmo *ColorAnt*-RT [Lintzmayer et al. 2011c], um algoritmo heurístico que utiliza como método construtivo para cada formiga um método sugerido com *ANTCOL* [Costa and Hertz 1997], aqui chamado de *Ant_Fixed_k*. Posteriormente, *ColorAnt*-RT passou por melhorias as quais originaram duas novas versões [Lintzmayer et al. 2011a, Lintzmayer et al. 2011d].

Para construir uma solução s, o *Ant_Fixed_k* escolhe, a cada ciclo, um vértice v ainda não colorido que possui o maior grau de saturação[2] e uma cor c entre as k disponíveis para colorí-lo. A cor c é escolhida de acordo com a probabilidade p, apresentada na Equação 1, que é calculada baseada na trilha de feromônio τ, apresentada na Equação 2, e na informação heurística η, apresentada na Equação 3.

$$p(s,v,c) = \frac{\tau(s,v,c)^\alpha \cdot \eta(s,v,c)^\beta}{\sum\limits_{i \in \{1,\ldots,k\}} \tau(s,v,i)^\alpha \cdot \eta(s,v,i)^\beta} \tag{1}$$

onde α e β são parâmetros do algoritmo e controlam a influência dos valores associados a eles na equação, e

$$\tau(s,v,c) = \begin{cases} 1 & \text{if } C_c(s) = \{\} \\ \dfrac{\sum\limits_{u \in C_c(s)} P_{uv}}{|C_c(s)|} & \text{otherwise} \end{cases} \tag{2}$$

$$\eta(s,v,c) = \frac{1}{|N_{C_c(s)}(v)|} \tag{3}$$

onde P_{uv} é a trilha de feromônio entre os vértices u e v, $C_c(s)$ é a classe da cor c da solução s, ou seja, o conjunto de vértices já coloridos com c nesta solução, e $N_{C_c(s)}(v)$ são os vértices $x \in C_c(s)$ adjacentes a v em s.

A trilha de feromônio, armazenada na matriz $P_{|V|\times|V|}$, é inicializada com 1 para cada aresta entre vértices não adjacentes e com 0 para cada aresta entre vértices adjacentes. Sua atualização (reforço) envolve a persistência da trilha atual (por um fator ρ, onde $1-\rho$ é a taxa de evaporação) e depende da experiência obtida por cada formiga (arestas entre nós não adjacentes são atualizados quando eles recebem a mesma cor). A evaporação é apresentada na Equação 4 e a forma geral de depósito de feromônio é apresentada na Equação 5.

$$P_{uv} = \rho P_{uv} \quad \forall u, v \in V \tag{4}$$

$$P_{uv} = P_{uv} + \frac{1}{f(s)} \quad \forall u, v \in C_c(s) \mid (u,v) \notin E, c = 1..k \tag{5}$$

onde $C_c(s)$ é o conjunto de vértices coloridos com c na solução s e f é a função objetivo, que retorna o número de arestas conflitantes da solução.

[2]Grau de saturação de um vértice é o número de cores diferentes que já coloriram seus vértices adjacentes.

Algoritmo 1 *ColorAnt-RT*.

COLORANT-RT($G = (V, E), k$) // V: vértices; E: arestas

1 $P_{uv} = 1\ \forall (u,v) \notin E$; $P_{uv} = 0\ \forall (u,v) \in E$;
2 $f^* = \infty$; // melhor valor da função objetivo até o momento
3 **while** condição não encontrada **do**
// Linha 4 existe apenas em *ColorAnt*$_1$*-RT*:
4 $\Delta P_{uv} = 0 \quad \forall u, v \in V$;
5 $f' = \infty$; // melhor valor da função no ciclo
6 **for** $a = 1$ **to** $nants$ **do**
7 $s =$ ANT_FIXED_K(G, k);
// Linha 8 existe apenas em *ColorAnt*$_1$*-RT*:
8 $\Delta P_{uv} = \Delta P_{uv} + \frac{1}{f(s)} \qquad \forall u, v \in C_c(s) \mid (u,v) \notin E, c = 1..k$;
// Linha 9 existe apenas em *ColorAnt*$_3$*-RT*:
9 $s =$ REACT_TABUCOL(G, k, s);
10 **if** $f(s) == 0$ **or** $f(s) < f'$ **then** $\{ s' = s; f' = f(s'); \}$
// Linha 11 existe apenas em *ColorAnt*$_1$*-RT* e *ColorAnt*$_2$*-RT*:
11 $s' =$ REACT_TABUCOL(G, k, s');
12 **if** $f' < f^*$ **then** $\{ s^* = s'; f^* = f(s^*); \}$
13 $P_{uv} = \rho P_{uv} \quad \forall u, v \in V$; // de acordo com a Equação 4
// Linha 14 existe apenas em *ColorAnt*$_1$*-RT*:
14 $P_{uv} = P_{uv} + \Delta P_{uv} \quad \forall u, v \in V$;
// Linhas 15–16 existem apenas em *ColorAnt*$_1$*-RT* e *ColorAnt*$_2$*-RT*:
15 $P_{uv} = P_{uv} + \frac{1}{f(s')}$ i $\quad \forall u, v \in C_c(s') \mid (u,v) \notin E, c = 1..k$;
16 $P_{uv} = P_{uv} + \frac{1}{f(s^*)} \quad \forall u, v \in C_c(s^*) \mid (u,v) \notin E, c = 1..k$;
17 $cycle = cycle + 1$;
// As próximas linhas existem apenas em *ColorAnt*$_3$*-RT*:
18 **if** $cycle \mod \sqrt{max\_cycles} == 0$ **then** $phero\_counter = cycle \div \sqrt{max\_cycles}$;
19 **if** $phero\_counter > 0$ **then**
20 $P_{uv} = P_{uv} + \frac{1}{f(s^*)} \qquad \forall u, v \in C_c(s^*) \mid (u,v) \notin E, c = 1..k$; // de acordo com a Equação 5
21 **else**
22 $P_{uv} = P_{uv} + \frac{1}{f(s')} \qquad \forall u, v \in C_c(s') \mid (u,v) \notin E, c = 1..k$; // de acordo com a Equação 5
23 $phero\_counter = phero\_counter - 1$;

Os três algoritmos *ColorAnt-RT* são sintetizados no Algoritmo 1. A principal diferença entre as três versões está na maneira de depositar feromônio e é como segue:

- *ColorAnt*$_1$*-RT*: além de cada formiga da colônia ser utilizada para atualizar a trilha de feromônio, a melhor formiga da colônia no ciclo (s') e a melhor formiga até o momento (s^*) também são utilizadas para atualizar a trilha de feromônio;
- *ColorAnt*$_2$*-RT*: apenas s' e s^* são utilizadas para atualizar a trilha de feromônio;
- *ColorAnt*$_3$*-RT*: s' e s^* não atualizam a trilha de feromônio simultaneamente; inicialmente s' atualiza mais frequentemente do que s^*. Uma gradual mudança na frequência é feita baseada no número máximo de ciclos do algoritmo: em cada intervalo de ciclos, a quantidade de ciclos na qual s^* irá atualizar a trilha de feromônio (ao invés de s') é incrementada em uma unidade.

Os três algoritmos *ColorAnt*-RT utilizam um método de busca local para melhorar a qualidade dos seus resultados: a busca tabu reativa *React-Tabucol* (RT) [Blöchliger and Zufferey 2008]. Em *ColorAnt*$_1$*-RT* e *ColorAnt*$_2$*-RT*, a busca local é aplicada apenas na melhor formiga da colônia, e ao final de um ciclo. Em *ColorAnt*$_3$*-RT*, a busca local é aplicada em todas as formigas da colônia em cada ciclo.

3.1. Resultados e Discussão

As três versões de *ColorAnt*-RT foram implementadas na linguagem `C` e compiladas com GCC 4.4.3 utilizando o nível de otimização `O3`. Ambas foram executadas em um computador Intel Xeon E5504 de 2.00 GHz, 24GB de memória RAM e sistema operacional Ubuntu 10.04.3 LTS com Kernel 2.6.32-37-server.

Os experimentos foram realizados em 10 grafos[3], a saber: *dsjc500.1* e *dsjc500.5* (grafos aleatórios); *dsjr500.1c* e *dsjr500.5* (grafos aleatórios geométricos); *le450_25c* e *le450_25d* (que possuem sempre 450 vértices e número cromático χ conhecido); e *fpsol2.i.1*, *fpsol2.i.2*, *mulsol.i.1* e *mulsol.i.2* (grafos de interferência). Nestes experimentos cada instância foi calibrada com o objetivo de encontrar os melhores valores para os parâmetros: *quantidade de formigas*, α, β, ρ e *quantidade de ciclos da busca local*.

A Tabela 1 apresenta os resultados obtidos por *ColorAnt*$_1$-RT, *ColorAnt*$_2$-RT e *ColorAnt*$_3$-RT. Nesta tabela a primeira coluna apresenta o nome do grafo e o par χ/k^* (com '?' caso χ não seja conhecido, onde k^* é o valor da melhor solução encontrada até o momento) e a segunda coluna os melhores valores de k encontrados. Nesta tabela ainda são apresentados a quantidade de execuções com sucesso (S) sobre o total (T) de execuções (S/T), a média do tempo total de execução em segundos (Time) e a quantidade média de conflitos (Cfs).

		ColorAnt$_1$-RT			*ColorAnt*$_2$-RT			*ColorAnt*$_3$-RT		
Graph (χ/k^*)	k	S/T	Time(s)	Cfs.	S/T	Time(s)	Cfs.	S/T	Time(s)	Cfs.
dsjc500.1 (?/12)	13	10/10	526,87	0	10/10	11,93	0	10/10	44,11	0
dsjc500.5 (?/48)	51	-	-	-	-	-	-	3/10	1690,42	2,8
	52	-	-	-	5/10	2465,66	1,9	9/10	682,20	0,4
	53	4/10	3075,58	2,1	10/10	515,04	0	10/10	297,62	0
dsjr500.1c (?/85)	**85**	2/10	3369,69	1,7	6/10	1719,05	0,6	9/10	29,49	0,1
dsjr500.5 (?/122)	**122**	8/10	1709,11	0,2	-	-	-	1/10	625,38	1,5
	123	10/10	17,69	0	10/10	26,19	0	9/10	164,49	0,1
le450_25c (25/25)	26	7/10	1235,02	1,3	-	-	-	2/10	456,19	3,4
	27	10/10	4,88	0	10/10	148,08	0	10/10	15.59	0
le450_25d (25/25)	26	3/10	2690.31	2,8	1/10	3270,29	4,6	2/10	724,87	3,4
fpsol2.i.1 (65/65)	**65**	6/10	1515,91	0,4	8/10	979,77	0,2	6/10	9,43	0,4
fpsol2.i.2 (30/30)	**30**	8/10	730.83	0,2	7/10	1085,84	0,4	2/10	10,61	0,8
mulsol.i.1 (49/49)	**49**	10/10	0,99	0	9/10	365,76	0,1	9/10	0,54	0,1
mulsol.i.2 (31/31)	**31**	9/10	432,42	0,1	5/10	1804,69	0,6	7/10	1,00	0,3

Tabela 1. Resultados obtidos pelos algoritmos *ColorAnt*$_1$*-RT*, *ColorAnt*$_2$*-RT* e *ColorAnt*$_3$*-RT*.

Os resultados apresentados na Tabela 1 demonstram que as três versões de *ColorAnt*-RT possuem resultados similares quanto à qualidade das soluções encontradas; apenas para as instâncias *dsjc500.5* e *dsjr500.5* os algoritmos diferem. No caso da primeira instância, a qualidade do resultado foi melhorado a medida que a maneira de depositar feromônio nas trilhas era modificado. Alternar entre s' e s^* foi a melhor abordagem para esta instância. Para o caso da segunda instância, utilizar s' e s^* simultaneamente para atualizar a trilha de feromônio, ocasiona uma perda na qualidade da solução encontrada.

Os piores resultados foram encontrados para os grafos *aleatórios*, *flat* e *geométricos*. Para estes, as distâncias médias entre as soluções encontradas e as melhores soluções conhecidas são de 8,3%, 4% e 0,4%, respectivamente. Contudo um ponto importante a ser observado é o fato de todas as versões de *ColorAnt*-RT encontrar a melhor aproximação para todas as instâncias que representam grafos de interferência.

Estes resultados demonstram que não existe uma relação direta entre densidade do grafo ou a quantidade de vértices e qualidade dos resultados encontrados. Todas as

[3]Disponível em `http://mat.gsia.cmu.edu/COLOR/instances.html`, acessado em maio de 2012.

versões de *ColorAnt*-RT encontraram a melhor solução para a instância que possui a maior densidade, *dsjr500.1c* cuja densidade é de 0,97. Além disto, para as instâncias *mulsol*, cuja densidade (0,22) é maior do que as instâncias *le450* (0,17), ambas versões encontraram as melhores soluções conhecidas. Quanto à quantidade de vértices, é interessante observar que apenas as instâncias *mulsol* possuem uma quantidade reduzida de vértices (190 na média), enquanto esta quantidade varia entre 450 e 500 vértices para as outras instâncias. Contudo, a distância entre as soluções obtidas e as melhores conhecidas variam consideravelmente. Portanto, não existe uma relação direta entre quantidade de vértices e qualidade dos resultados. Embora fosse naturalmente esperado que um grafo com uma densidade alta e/ou uma quantidade excessiva de vértices ocasionasse uma perda de desempenho quanto a qualidade dos resultados.

Um ponto importante a ser destacado é o impacto que a mudança na maneira de depositar feromônio ocasiona ao tempo de execução do algoritmo; em outras palavras, em como o algoritmo converge para uma determinada solução. No geral, a estratégia utilizada por *ColorAnt*$_3$-RT é a melhor para ocasionar uma convergência mais rápida. Para as melhores soluções encontradas, *ColorAnt*$_3$-RT obteve um desempenho melhor do que *ColorAnt*$_1$-RT, para todos os casos, o que não ocorre quando *ColorAnt*$_2$-RT é comparado com *ColorAnt*$_1$-RT. Isto demonstra que as modificações efetuadas em *ColorAnt*-RT alcançaram diversos objetivos, melhorando a qualidade das soluções como também melhorando a convergência do algoritmo. Outro ponto a ser destacado é a capacidade de *ColorAnt*-RT reduzir a quantidade de conflitos. Em geral, *ColorAnt*$_3$-RT é melhor que *ColorAnt*$_2$-RT, que por sua vez é melhor que *ColorAnt*$_1$-RT.

Em síntese, os resultados demostraram que a maior influência na qualidade dos resultados obtidos está na maneira de depositar feromônio, bem também demonstraram que *ColorAnt*-RT é uma boa opção para ser aplicado à alocação de registradores.

4. O Alocador de Registradores CARTRA

CARTRA (*ColorAnt*$_3$-RT Register Allocator) [Lintzmayer et al. 2011b] modifica o algoritmo proposto por George e Appel (George-Appel) [George and Appel 1996] com o objetivo de adicionar uma fase composta pelo algoritmo *ColorAnt*$_3$-RT.

Sendo um algoritmo iterativo, o George-Appel executa diversas vezes até que não existam mais *spills*[4]. Os resultados obtidos por este algoritmo demonstraram como mesclar coloração com heurísticas de *coalescing*[5], produzindo um algoritmo que é seguro e agressivo. Este possui as seguintes fases: *build* (construir o grafo de interferência); *simplify* (simplificar o grafo de interferência); *coalesce* (realizar *conservative coalescing*); *freeze* (congelar vértices relacionados à movimentação); *spill* (selecionar um vértice para *spill*); e *select* (atribuir cores aos vértices do grafo).

Duas modificações foram feitas no algoritmo George-Appel, a saber:

1. A fase *select* foi substituída pelo algoritmo *ColorAnt*$_3$-RT: desta forma a coloração é mais agressiva do que aquela implementada em George-Appel; e
2. A estratégia utilizada para selecionar *spill* não é mais baseada no grau de um nó, mas sim baseada na quantidade de conflitos.

[4]Valores que serão efetivamente representados em memória.

[5]União de vértices que representam uma cópia – origem e destino de uma instrução de movimentação.

Inicialmente, as fases clássicas de George-Appel constroem o grafo de interferência e o reduzem. Após, o algoritmo *ColorAnt*$_3$-RT colore o grafo de interferência. E finalmente, a nova fase *Spill* seleciona um nó apropriado para representar em memória.

No algoritmo George-Appel, se não existir oportunidade para *simplify* ou *freeze*, um nó será escolhido para ser representado em memória (*spilled*). Neste caso, a fase *spill* irá calcular a prioridade para cada nó utilizando a Equação 6.

$$P_n = ((uses_{out} + defs_{out}) + 10 \times (uses_{in} + defs_{in}))/degree \quad (6)$$

onde $uses_{out}$ é a quantidade de usos fora de um laço; $defs_{out}$ é a quantidade de definições fora do laço; $uses_{int}$ é a quantidade de usos dentro de um laço; $defs_{in}$ é a quantidade de definições dentro de um laço; e $degree$ é quantidade de arestas incidentes no vértice (grau).

O vértice que tiver menor prioridade será selecionado para ser representado em memória. Esta abordagem é uma aproximação otimista: o vértice removido não irá interferir com nenhum outro vértice.

CARTRA utiliza uma abordagem diferente para selecionar um nó para *spill*. Como o grafo resultante da fase *ColorAnt*$_3$*-RT* pode conter vértices conflitantes, a fase *spill* funciona selecionando o vértice com maior frequência no conjunto de vértices conflitantes por classe de cor; em outras palavras, considerando cada cor c, o nó colorido com c que tiver a maior quantidade de arestas conflitantes incidentes é removido do grafo e considerado um *spill*. Isso se repete até que não existam mais conflitos no grafo. Se houver pelo menos um *spill*, o programa será reescrito como em George-Appel, e uma nova iteração será executada. Portanto, o algoritmo termina quando não existir mais nenhum *spill* ao final desta fase.

4.1. Resultados e Discussão

O algoritmo George-Appel e CARTRA foram implementados em um compilador que gera código IA32, e então comparados. Para este objetivo foram utilizados doze programas do *benchmark* SNU-RT[6], e os programas *Merge Sort* e *Queens*. Cada programa foi executado 10 vezes em cada alocador. Além disto, os parâmetros utilizados por *ColorAnt*$_3$*-RT* foram fixados para todos os programas, portanto os *programas* não foram calibrados. Isto pelo fato de cada programa consistir em N funções, ocasionando N grafos distintos. Assim, os parâmetros utilizados foram: $nformigas = 80$, $\alpha = 3$, $\beta = 16$, $\rho = 0,7$ e $max\_ciclos = 625$. A busca tabu foi limitada ao máximo de 300 ciclos.

4.1.1. Spill e Fetch

Como pode ser observado pelos resultados apresentados na Tabela 2, CARTTA tem um desempenho melhor do que o algoritmo George-Appel. CARTRA representa uma quantidade menor de valores em memória, pelo fato de gerar uma melhor coloração para o grafo, de forma que a quantidade de conflitos seja minimizada. Neste caso, CARTRA é capaz de utilizar uma quantidade menor de registradores por função. Isto melhora o desempenho da função a medida que reduz a quantidade de instruções de acesso à memória

[6]Disponível em `http://www.cprover.org/goto-cc/examples/snu.html`.

– instruções que tipicamente possuem um alto custo quando comparadas com outras classes de instruções. Além disto, como CARTTA tende a representar uma quantidade menor de valores em memória e reduzir a pressão por registradores, ele é capaz de encontrar mais oportunidades para *coalescing*.

Tabela 2. Spill

Programa	CARTRA		George-Appel	
	Spill	*Fetch*	*Spill*	*Fetch*
Binary Search	**19**	**18**	126	142
FFT	**80**	**138**	91	159
FFT Complex	**53**	**86**	68	103
Fibonacci	**4**	**3**	5	5
FIR	**49**	**112**	68	128
Insert Sort	**12**	**32**	21	39
Jfdctint	89	178	**87**	**165**
LMS	**83**	**127**	136	186
Merge Sort	**40**	**58**	50	70
Quick sort	**40**	**100**	171	277
Queens	**16**	**42**	18	44
Qurt	**27**	**37**	95	126
Select	**37**	**82**	191	265
Sqrt	**8**	**11**	12	19

Em treze programas, CARTRA alcança uma redução que varia entre 2,778% e 85,317% na quantidade de *spills*. Apenas para um programa George-Appel obteve um resultado superior a CARTRA, a saber: `Jfdctint`. Além disto, CARTRA obteve uma redução que varia entre 11,165% e 86,620% na quantidade de dados que necessitam ser buscados da memória (*fetch*). Porém, para buscas, George-Appel obteve melhores resultados para `FIR` e `Jfdctint`. Em síntese, apenas para um programa CARTRA não obteve um melhor resultado que George-Appel. Isto demonstra que o uso de *ColorAnt*-RT e da estratégia de selecionar *spill* usada por CARTRA são boas alternativas para reduzir a quantidade de *spills*.

4.1.2. Convergência

Como ambos algoritmos são iterativos é importante analisar a convergência de ambos. Por restrições de espaço, para esta seção não serão apresentados os dados coletados. Em geral CARTRA converge mais rapidamente do que George-Appel, que necessita de no mínimo 5 vezes a quantidade de iterações necessárias por CARTRA. A abordagem baseada em *defs-uses* utilizada por George-Appel ocasiona uma gradual redução da quantidade de *spills*, até que esta alcance o valor zero. Por outro lado, uma abordagem baseada na quantidade de conflitos converge rapidamente.

CARTRA não necessita de mais do que três iterações para encontrar uma solução, enquanto George-Appel necessita na maioria dos casos de pelo menos cinco iterações. Além disto, em George-Appel algumas iterações não reduzem a quantidade de *spills*, ocasionando mais iterações.

4.1.3. Tamanho do Código

CARTRA gera um código com melhor qualidade, no tocante a quantidade de instruções Assembly e tamanho do código, o que pode ser observado pelos resultados apresentados

na Tabela 3. Portanto, o código gerado por CARTRA é mais compacto do que o código gerado por George-Appel.

Tabela 3. Tamanho do Código

Programa	CA-RT-RA		George-Appel	
	Assembly Instruções	Tamanho (Bytes)	Assembly Instruções	Tamanho (Bytes)
Binary Search	**523**	**4884**	809	6148
FFT	**991**	7596	1021	**7720**
FFT Complex	**797**	**6276**	840	6412
Fibonacci	**43**	**860**	47	872
FIR	**1732**	**15104**	1759	15192
Insert Sort	**337**	**3496**	358	3564
Jfdctint	1525	10276	**1501**	**10204**
LMS	**867**	**6352**	1000	6996
Merge Sort	**501**	**4424**	519	4488
Quick sort	**1314**	**12132**	1676	13840
Queens	**398**	**3740**	**398**	**3740**
Qurt	**802**	**7496**	981	8180
Select	**1216**	**11392**	1618	13324
Sqrt	**108**	**1436**	119	1472

A redução da quantidade de instruções Assembly varia entre 8,511% e 35,352%, o que ocasiona uma redução no tamanho do código entre 1,76% e 20,559%. Apenas para `Jfdctint` e `Queens` CARTRA não ultrapassa o desempenho obtido por George-Appel. Portanto, estes resultados são importantes para sistemas que utilizam microprocessadores embarcados, devido ao fato de seus componentes consistirem usualmente de limitado poder computacional, além de uma memória limitada.

5. Conclusões e Trabalhos Futuros

Encontrar uma solução para o k-PCG é um problema $\mathcal{NP}$-completo. Desta forma, não existe um algoritmo em tempo polinomial que o resolva de forma exata (a menos que $\mathcal{P} = \mathcal{NP}$), o que encoraja o uso de algoritmos heurísticos para encontrar boas soluções, como por exemplo otimização por colônia de formigas.

Este artigo apresentou as versões do algoritmo *ColorAnt*-RT, um algoritmo heurístico baseado em colônia de formigas aplicado ao problema de coloração de grafos. Embora os resultados obtidos por *ColorAnt*-RT não sejam os mesmos das melhores soluções conhecidas, para algumas instâncias este se aproxima em alguns casos dos melhores algoritmos propostos na literatura. Contudo, um ponto positivo da terceira versão de *ColorAnt*-RT é o fato desta minimizar a quantidade de conflitos, e em muitos casos reduzir o tempo de execução quando comparado com suas versões anteriores.

Quando empregado como uma fase do alocador proposto por George e Appel, *ColorAnt*$_3$-RT ocasionou uma melhora considerável no desempenho do alocador. Em geral a quantidade de valores do programa que foram efetivamente representados em memória teve uma redução significativa, o que proporcionou um código gerado de melhor qualidade.

Trabalhos futuros compreendem: implementar outras heurísticas para avaliá-las (e compará-las) sobre as mesmas condições; estudar e implementar versões paralelas de *ColorAnt*-RT; e um trabalho mais ambicioso compreende tornar *ColorAnt*-RT auto-adaptável às características da instância de entrada, de forma que os parâmetros sejam *calibrados* automaticamente.

Referências

Blöchliger, I. and Zufferey, N. (2008). A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975.

Briggs, P., Cooper, K. D., and Torczon, L. (1994). Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16:428–455.

Chaitin, G. J. (1982). Register Allocation & Spilling via Graph Coloring. *SIGPLAN Notices*, 17:98–101.

Costa, D. and Hertz, A. (1997). Ants Can Colour Graphs. *The Journal of the Operational Research Society*, 48(3):295–305.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. Bradford Books. MIT Press, Cambridge, Massachusetts.

George, L. and Appel, A. W. (1996). Iterated register coalescing. *ACM Transactions on Programming Languages and Systems*, 18:300–324.

Hertz, A. and Zufferey, N. (2006). A New Ant Algorithm for Graph Coloring. In *Workshop on Nature Inspired Cooperative Strategies for Optimization NICSO*, pages 51–60, Granada, Espanha.

Lintzmayer, C. N., Mulati, M. H., and da Silva, A. F. (2011a). Algoritmo Heurśtico Baseado em Colônia de Formigas Artificiais ColorAnt2 com Busca Local Aplicado ao Problema de Coloração de Grafo. In *X Congresso Brasileiro de Inteligência Computacional*, Fortaleza, BRA.

Lintzmayer, C. N., Mulati, M. H., and da Silva, A. F. (2011b). Register Allocation with Graph Coloring by Ant Colony Optimization. In *XXX International Conference of the Chilean Computer Science Society*, Curico, Chile.

Lintzmayer, C. N., Mulati, M. H., and da Silva, A. F. (2011c). RT-ColorAnt: Um Algoritmo Heurístico Baseado em Colônia de Formigas Artificiais com Busca Local para Colorir Grafos. In *XLIII Simposio Brasileiro de Pesquisa Operacional 2011*, Ubatuba, SP, BRA.

Lintzmayer, C. N., Mulati, M. H., and da Silva, A. F. (2011d). Toward Better Performance of ColorAnt ACO Algorithm. In *XXX International Conference of the Chilean Computer Science Society*, Curico, Chile.

Plumettaz, M., Schindl, D., and Zufferey, N. (2010). Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, 61(5):819–826.

Shawe-Taylor, J. and Zerovnik, J. (2001). Ants and graph coloring. In *International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 276–279.

Wu, S. and Li, S. (2007). Extending Traditional Graph-Coloring Register Allocation Exploiting Meta-heuristics for Embedded Systems. In *Proceedings of the Third International Conference on Natural Computation*, pages 324–329.