Des Autom Embed Syst (2009) 13: 115–137
DOI 10.1007/s10617-008-9034-y

Optimistic chordal coloring: a coalescing heuristic for SSA form programs

Philip Brisk · Ajay K. Verma · Paolo Ienne

Received: 21 July 2008 / Accepted: 27 October 2008 / Published online: 11 November 2008

Abstract The interference graph for a procedure in *Static Single Assignment (SSA) Form* is chordal. Since the k-colorability problem can be solved in polynomial-time for chordal graphs, this result has generated interest in SSA-based heuristics for spilling and coalescing. Since copies can be folded during SSA construction, instances of the coalescing problem under SSA have fewer affinities than traditional methods. This paper presents *Optimistic Chordal Coloring (OCC)*, a coalescing heuristic for chordal graphs. OCC was evaluated on interference graphs from embedded/multimedia benchmarks: in all cases, OCC found the optimal solution, and ran, on average, 2.30× faster than *Iterated Register Coalescing*.

Keywords Algorithms · Performance · Coalescing · Chordal graphs

1 Introduction

Register allocation is one of the most widely studied NP-Complete problems in computer science. Register allocation is typically broken down into two sub-problems: spilling and coalescing, both of which are NP-Complete [3–5, 19, 31, 33]. *Spilling* is the problem of partitioning all of the variables that are live at each point in the program between registers and memory. *Coalescing* is the problem of assigning variables to registers such that: (1) no variables whose lifetimes overlap are assigned to the same register; and (2) the minimum (weighted) number of register-to-register copy instructions remain in the program (or, equivalently, the maximum number of copies are removed).

For many years, register allocation and other related storage assignment problems have been modeled using graph coloring, another well-known NP-Complete problem [12, 13, 17, 18, 29], or its inverse, clique partitioning [39]. Although graph coloring is NP-Complete in the general case, there are many classes of graphs for which polynomial-time coloring

This paper is an extension of a paper that appeared at CASES 2007 [10].

P. Brisk (✉) · A.K. Verma · P. Ienne
Swiss Federal Institute of Technology, Lausanne (EPFL), Switzerland
e-mail: philip.brisk@epfl.ch

solutions are known. Two such classes of particular importance are *chordal graphs* [20] and *interval graphs* [26].

Recently, there has been interest in performing register allocation using intermediate representations using *Static Single Assignment (SSA) Form*; *pruned* SSA Form [14] is assumed throughout the paper. In particular, several research groups have recently proven that the interference graph for a program in SSA Form belongs to the class of chordal graphs [2, 8, 24]; Brisk and Sarrafzadeh [9] also proved that the interference graph for a program in *Static Single Information (SSI) Form*, an extension of SSA Form, belongs to the class of interval graphs. Unfortunately, both spilling and coalescing remain NP-Complete for chordal and interval graphs [4, 5]; spilling is NP-Complete for straight-line code with no control flow [19].

That being said, SSA Form does have some advantages that could be useful for register allocation. First and foremost, the problem of *spill-free register allocation* can be solved optimally in polynomial-time: i.e., if the target architecture has k registers, then a minimum coloring of the interference graph determines whether spilling is necessary; if spilling is unnecessary, then a solution to the coalescing problem must be found that eliminates as many copy instructions as possible. A second benefit is that all of the copy operations in the program can be eliminated during the conversion to SSA Form [6]. Copies are only inserted during the translation out of SSA Form to eliminate φ-functions: conditional parallel copy operations that are an integral part of SSA Form [16]. The number of copies to be eliminated via coalescing in SSA Form is significantly less than traditional register allocation.

1.1 Contribution

This paper contributes a novel heuristic for the coalescing problem for SSA Form programs whose interference graph is chordal. The heuristic is called *Optimistic Chordal Coloring (OCC)*, and it ensures that a legal coloring is found while attempting to minimize the (weighted) number of copy operations that remain in the program after coloring the interference graph. The worst-case time complexity of OCC is $O(|V|^2)$.

OCC was tested on a set of interference graphs generated from a set of Mediabench [30] and MiBench [23] applications and compared against the well-established *Iterated Register Coalescing (IRC)* heuristic [21] and an optimal formulation (*OPT*) of the coalescing problem as an *integer linear program (ILP)* [22]. In all of our test cases, OCC found the optimal solution to the problem, while running more than twice as fast as IRC, on average.

1.2 The coalescing problem

Consider variables, u and v, connected by a copy operation $v \leftarrow u$. If both u and v are assigned to the same register, r, then the copy will become $r \leftarrow r$—an identity operation that does not change the state of the processor—a NOP which can be eliminated.

Let V be the set of variables in the program. Two variables *interfere* if their lifetimes overlap. Let $E \subseteq V \times V$ be the set of *interference edges*, i.e. $e = (v_1, v_2) \in E$ if v_1 and v_2 interfere. Let $A \subseteq V \times V - E$ be the set of *affinity edges*, i.e., $a = (v_1, v_2) \in A$ if and only if v_1 and v_2 do *not* interfere and there is a copy operation $v_1 \leftarrow v_2$ or $v_2 \leftarrow v_1$ in the program. If runtime profiling information is available, and the copy is known to execute w times, then the affinity edge is given a weight, denoted $w(a) = w(u, v)$, i.e. $a = (v_1, v_2, w)$.

The graph $G = (V, E, A)$ is called an *interference graph*. It can be constructed as described in the textbook by Cooper and Torczon [15]. A *k-coloring* is a function $color : V \rightarrow \{1, 2, \ldots, k\}$. A k-coloring is *legal* if for every interference edge $(v_1, v_2) \in E, color(v_1) \neq$

color(v_2); otherwise, it is *illegal*. An affinity edge $a = (v_1, v_2)$ is *satisfied* by a k-coloring *color* if *color*(v_1) = *color*(v_2). An *unsatisfied* affinity edge requires the insertion of a copy operation $v_1 \leftarrow v_2$, or $v_2 \leftarrow v_1$; a satisfied affinity edge eliminates the copy by assigning v_1 and v_2 to the same register.

In general, the problem of determining whether or not a graph is k-colorable is NP-Complete; however, in the case of the coalescing problem, we are given an interference graph that is known to be k-colorable, where k is the number of registers in the target architecture. The goal of the coalescing problem is to find a k-coloring of the interference graph that maximizes the sum of the weights of the satisfied affinity edges.

In SSA Form, all copies can be folded; however, affinities are introduced by φ-functions, an integral part of the SSA Form. To understand this paper, the reader does not need to understand the details of SSA Form or how φ-functions are instantiated; it suffices to understand the concept of an affinity edge.

The details of the translation out of SSA Form can be found in the paper by Hack and Goos [24]. Similar to their model, we assume that swap instructions are available for SSA elimination, when needed. The details are beyond the scope of this work.

1.3 Paper organization

Section 2 introduces preliminary notation and concepts; Sect. 3 summarizes related work on coalescing; Sect. 4 presents the OCC heuristic; Sect. 5 presents the experimental evaluation and results; Sect. 6 concludes the paper.

1.4 Statement of extension of prior work

The version of OCC presented here is an extension of prior work [10]. Material that is unique to this paper includes: the example interference graphs taken from Mediabench and MiBench applications; the discussion of illegal pseudo-coalescing (Sect. 4.4.2); the BMCS-2 method to compute a PEO (Sect. 4.3); the full-blown color assignment and propagation method (Sect. 4.5.2); complexity analyses of OCC (Sect. 4.6); new experimental results based on a new implementation of OCC (Sect. 5).

2 Preliminaries

2.1 Graph notation

Let $G = (V, E, A, w)$ be an interference graph as introduced in Sect. 1.1. $\chi(G)$ is defined to be the *chromatic number* of G, i.e., the smallest value k for which there is a legal k-coloring for G. For vertex v, let: $N(v)$ be the set of interference neighbors of v; $N_A(v)$ be the set of affinity neighbors of v; $R_A(v)$ be the set vertices reachable from v by affinity edges; and $C_A(v)$ be a set of vertices with which v has been pseudo-coalesced (see Sect. 4.2). Let S be a set of vertices. Then:

$$N(S) = \bigcup_{v \in S} N(v) - S. \tag{1}$$

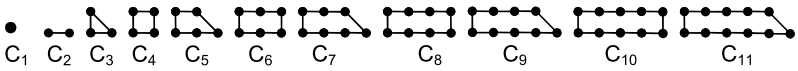


Fig. 1 k-cycles $C_1, \ldots, C_{11}$

Algorithm: MCS
Input: Chordal Graph G = (V, E)
Output: PEO σ : V → {1… |V|}

1. ∀ v ∈ V: Let T(v) ← 0; σ(v) ← ϕ
2. For Integer : k ← 1 to |V|
3. Select v ∈ V ∋ T(v) is maximum
4. σ(v) ← k
5. ∀ u ∈ N(v) ∋ σ(v) = ϕ :
6. T(u) ← T(u) + 1
7. EndFor

(a)

Algorithm: Chordal Color
Input: Chordal Graph G = (V, E), PEO σ
Output: Color Assignment f : V → {1… χ_G}

1. For Integer : i ← 1 to |V| in PEO order
2. Let c be the smallest color not assigned to a vertex in $N^i(v_i)$
3. color(v_i) ← c
4. EndFor

(b)

Fig. 2 Pseudocode for MCS [38] (**a**) and optimal chordal color assignment [20] (**b**)

2.2 Chordal graphs

A k-cycle is the graph $C_k = (U_k, E_k)$, where $U_k = \{v_0, v_1, \ldots, v_{k-1}\}$ and $E_k = \{(v_i, v_{(i+1) mod\, k}) | 0 \leq i \leq k-1\}$. Figure 1 lists the k-cycles $C_1, \ldots, C_{11}$. Graph $G = (V, E)$ is *chordal* if it has no subgraph isomorphic to a C_k for $k \geq 4$; other equivalent definitions of chordal also exist. The interference graph of an SSA Form procedure is provably chordal [2, 8, 24].

An *Elimination Order (EO)* is a one-to-one and onto function $\sigma : V \rightarrow \{1, 2, \ldots, |V|\}$; given an EO, vertices are renamed so that $\sigma(v_i) = i$. Let $V_i = \{v_1, v_2, \ldots, v_i\}$ and $G_i = (V_i, E_i)$ be the subgraph of G induced by V_i. For vertex $v_i : N^i(v_i) = N(v_i) \cap V_i$, $N^i_A(v_i) = N_A(v_i) \cap V_i$, and $C^i_A(v_i) = C_A(v_i) \cap V_i$.

Vertex v is *simplicial* if $N(v)$ forms a clique. A *Perfect Elimination Order (PEO)* is an EO where v_i is simplicial in G_i for $1 \leq i \leq |V|$. Graph G is chordal if and only if G has a PEO. A PEO can be computed in $O(|V| + |E|)$ time by an algorithm called *Maximum Cardinality Search (MCS)* [38] shown in Fig. 2(a). Given a PEO, G is colored optimally in $O(|V| + |E|)$ time using an algorithm shown in Fig. 2(b) [20]. Colors are assigned to vertices in PEO order. Then $color(v_i)$ is the smallest color not assigned to a vertex in $N^i(v_i)$; optimality ensues because $N^i(v_i)$ is a clique.

3 Related work

In graph coloring register allocation, coalescing refers to the process of merging vertices in the interference graph, often to eliminate copy operations via register assignment. Given an affinity edge $(u, v) \in A$, coalescing u and v ensures that they receive the same color. Let uv be the resulting vertex. Then:

$$N(uv) = N(u) \cup N(v), \tag{2}$$

$$N_A(uv) = [N_A(u) \cup N_A(v)] - N(uv). \tag{3}$$

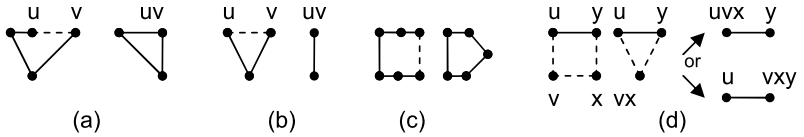


Fig. 3 Coalescing can increase the degree of the merged vertex and the chromatic number of the graph (**a**); it can also decrease the degree of a vertex adjacent to the two vertices that have been merged (**b**); and it can cause a chordal graph to become non-chordal, while increasing the chromatic number of the graph (**c**); in some cases, it is impossible to coalesce every affinity edge (**d**)

If affinity edges (u, x) and (v, x), exist then $w(uv, x) = w(u, x) + w(v, x)$.

Coalescing has both positive and negative side effects [32, 40]. In Fig. 3(a), $|N(u)| = |N(v)| = 1$ and $\chi_G = 2$ before coalescing, and afterwards $|N(uv)| = 2$ and $\chi_G = 3$. In Fig. 3(b), $|N(u)| = |N(v)| = 2$ before coalescing, and $|N(uv)| = 1$ afterwards. In Fig. 3(c) coalescing causes a chordal graph to become non-chordal and increases χ_G from 2 to 3. Figure 3(d) shows that it is generally impossible to coalesce all affinity edges.

Let k be the number of registers in the target processor and assume that $\chi_G \leq k$. Three variants of the coalescing problem are NP-Complete [5]:

Aggressive coalescing [12, 13] tries to satisfy as many affinity edges as possible without constraining χ_G; in register allocation, aggressive coalescing can increase the number of spills. Aggressive coalescing can also minimize the number of copy instructions required to eliminate φ-functions during translation out of SSA Form. Heuristics for aggressive coalescing have been proposed by Sreedhar et al. [37], Budimlic et al. [11], Rastello et al. [35] and Boissinot el al. [1].

Conservative coalescing [7, 21, 25, 27] attempts to find a legal k-coloring of G such that $\chi_G \leq k$ and as many affinity edges as possible are satisfied.

Optimistic coalescing [32] begins with an aggressively coalesced interference graph G, and performs a de-coalescing phase that tries to minimize the number of affinity edges de-coalesced while ensuring that $\chi_G \leq k$. In their study, Park and Moon [32] found that optimistic coalescing eliminated more copies than conservative coalescing.

Previous heuristics for conservative and optimistic coalescing are based on vertex merging. In SSA Form, the fact that the interference graph is chordal yields a stronger guarantee of k-colorability than traditional coalescing methods, because merging vertices may not preserve the chordal graph property; the color assignment procedure guarantees a legal coloring because it uses the same correctness invariant as Gavril's [20] chordal coloring algorithm. Pseudo-coalescing, introduced in Sect. 4.2, mimics the behavior of coalescing but without merging vertices. Thus, the OCC heuristic has stronger conservative guarantees than conservative coalescing, while retaining the benefits of optimistic coalescing in terms of solution quality.

4 Optimistic chordal coloring

The *Optimistic Chordal Coloring (OCC)* heuristic consists of six sequential steps, described in Sects. 4.1–4.5:

(1) *Simplify*: Remove vertices of low degree incident on no affinity edges.
(2) *Pseudo-coalesce*: Find (independent) sets of affinity-connected vertices.
(3) *PEO*: Compute a PEO; favor vertices incident on high-weight affinity edges.

(4) *Color Assignment*: Assign colors to vertices in PEO order maintaining Gavril's invariant; color assignment is biased by the sets of pseudo-coalesced vertices.
(5) *Refinement*: Attempt to improve the solution via local improvement.
(6) *Unsimplify*: Assign colors to the vertices removed in Step (1).

The key points of the heuristic are as follows:

- Steps (3) and (4) exploit the fact that chordal graphs are k-colorable.
- Step (2) is cognizant of the fact that traditional coalescing heuristics often find good solutions. Step (4) tries to assign the same color to all vertices that have been pseudo-coalesced with one another; but it cannot guarantee that such a coloring is found.
- Steps (1), (3), and (6) exploit the observation that only vertices incident on affinity edges contribute to the objective value of the solution; all other vertices must be colored to ensure legality; if possible, deferring the assignment of colors to these vertices tends to yield better conservative solutions.

Sections 4.1–4.5 describe the 6 steps in detail; the complexity of OCC is analyzed in Sect. 4.6. The interference graph $G = (V, E, A)$ is assumed to be chordal throughout.

4.1 Steps (1)/(6): simplify/unsimplify

Let $v \in V$ be a vertex with $|N(v)| < k$. Then the subgraph of G induced by $V - \{v\}$ is k-colorable if and only if G is k-colorable [28]; it is also chordal. v is called a *simplifiable* vertex. *Simplify* repeatedly removes each simplifiable vertex v with $|N_A(v)| = 0$ from G and pushes v onto a stack. Steps (2)–(5) then compute a k-coloring of the resulting chordal interference graph. *Unsimplify* repeatedly pops a vertex v from the stack and reattaches it to G. When processing v, $|N(v)| \leq k - 1$, so a color is available for v.

Some variation of simplification has been used during register allocation dating back to the first graph coloring allocator by Chaitin et al. [13] and Chaitin [12]. Our approach to simplification is the same as that of Hack and Grund [22].

Condition $|N_A(v)| = 0$ ensures that removing v cannot degrade the solution quality. Let $u \in N(v)$. If $|N_A(u)| > 0$, then $color(u)$ influences the solution quality; assigning a color to v before u can only constraint the spectrum of colors available for u when a color is chosen; thus, removing v is beneficial at best and benign at worst. If $|N(u)| = k$ and $|N_A(u)| = 0$, then the removal of v renders u simplifiable.

In Steps (2)–(5), all vertices in V are *unsimplifiable* or are incident on at least one affinity edge.

4.2 Step (2): pseudo-coalescing

The purpose of pseudo-coalescing is to find a set of affinity-connected vertices; these sets are used to guide the color assignment phase in Step (4). Unlike traditional coalescing methods, these vertices are not merged. $C_A(v)$ is defined to be the set of vertices with which v is pseudo-coalesced. Section 4.2.1 presents *Optimistic Pseudo-coalescing* [9], an adaptation of Park and Moon's [32] heuristic for aggressive coalescing. Section 4.2.2 introduces *Illegal Pseudo-coalescing*, in which $C_A(v)$ is defined to be the set of vertices reachable from v via affinity edges.

4.2.1 Optimistic pseudo-coalescing

Initially, let $C_A(v) = \{v\}$ for each vertex v with $|N_A(v)| > 0$; the invariant that $C_A(v)$ is an independent set is maintained throughout. First, the set A of affinity edges is sorted in de-

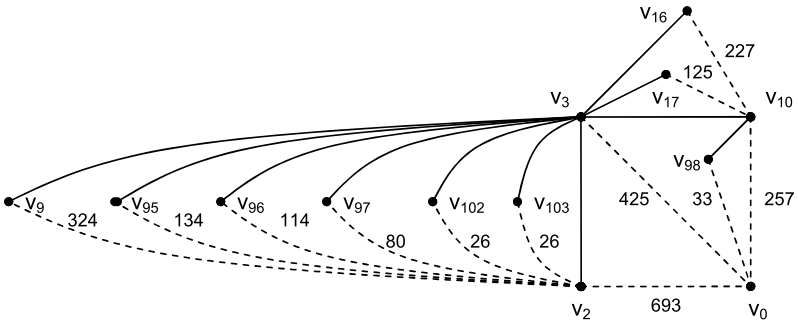


Fig. 4 An interference graph fragment taken from the *frame_ME* function in the *mpeg2enc* benchmark. After optimistic pseudo-coalescing, $C_A(v_0) = \{v_0, v_2, v_9, v_{10}, v_{16}, v_{17}, v_{95}, v_{96}, v_{97}, v_{102}, v_{103}\}$, $C_A(v_3) = \{v_3\}$, and $C_A(v_{98}) = \{v_{98}\}$, which leaves affinity edges (v_0, v_3) and (v_0, v_{98}) unsatisfied

scending order of weight; affinity edges are processed in sorted order. Consider affinity edge (v_1, v_2, w). If $C_A(v_1) = C_A(v_2)$, then v_1 and v_2 are already pseudo-coalesced; otherwise, v_1 and v_2 are pseudo-coalesced if and only if $C_A(v_1) \cup C_A(v_2)$ is an independent set; it suffices to check that $C_A(v_1) \cap N(C_A(v_2))$(or, equivalently, $N(C_A(v_1)) \cap C_A(v_2)$) is empty. Sorting the affinity edges in advance favors pseudo-coalescing of high-weight affinity edges over lower-weight affinity edges.

Figure 4 shows a fragment of the interference graph for the procedure *frame_ME* taken from the *MPEG-2* encoder benchmark [30]; many vertices are not shown, for clarity. Pseudo-coalescing affinity edges in sorted order, yields independent sets: $C_A(v_0) = \{v_0, v_2, v_9, v_{10}, v_{16}, v_{17}, v_{95}, v_{96}, v_{97}, v_{102}, v_{103}\}$, $C_A(v_3) = \{v_3\}$, and $C_A(v_{98}) = \{v_{98}\}$; affinity edges (v_0, v_3) and (v_0, v_{98}) are unsatisfied. This solution is optimal.

4.2.2 Illegal pseudo-coalescing

The use of one NP-Complete problem to solve another is a poor strategy; thus, the use of aggressive coalescing, which is NP-Complete in its own right [5], within a heuristic to solve conservative coalescing, is a poor strategy: even if aggressive coalescing is solved optimally, it does not guarantee an optimal solution to conservative coalescing. Thus, the time spent on finding an aggressive solution would be better spent searching for a conservative solution directly. To this end, the *illegal pseudo-coalescing* strategy replaces the optimistic strategy with a much more efficient computation, sidestepping this issue completely.

Let $R_A(v)$ be the set of vertices reachable from v by affinity edges. Under *illegal pseudo-coalescing*: $C_A(v) = R_A(v)$ for each vertex v. When illegal coalescing is performed, $C_A(v)$ may not be an independent set; thus, illegal pseudo-coalescing does *not* attempt to solve the aggressive coalescing problem; for example $C_A(v_0)$ in Fig. 4 would contain every vertex in G, including those that interfere. This is not, however, problematic: there is no actual requirement that $C_A(v_0)$ be an independent set. Color assignment and propagation, in Step (4), is given leeway to de- and re-coalesce vertices in an adaptive fashion when one color cannot be assigned to all of the vertices in $C_A(v)$.

The details of illegal pseudo-coalescing cannot be understood without first understanding the color assignment heuristic, and are therefore delayed until Sect. 4.4.6.

4.3 Step (3): compute PEO

A graph is chordal if and only if it has a PEO; in fact, it may have many distinct PEOs. Gavril's [20] algorithm can use any PEO to compute a k-coloring. The quality of coalescing solutions produced by OCC, however, which account for affinity edges, is highly dependent on the PEO used. The ideal PEO, intuitively, assigns colors to vertices incident on high-weight affinity edges as early as possible during its execution.

A *Biased Maximum Cardinality Search (BMCS)* [9] makes the following modification to the MCS algorithm. Referring to Fig. 2(a), let M be the set of vertices v such that $T(v)$ is maximum (line 3). For vertex $m \in M$, let $W^*(m)$ be the sum of the weights of the affinity edges incident on m. When there is a choice between multiple vertices with maximum T-values, W^* is used as a tiebreaker, and the vertex with the maximum w^* value is chosen. No other modifications are required.

A second alternative, *BMCS-2*, is a minor modification to BMCS. For vertex $m \in M$, let $W_2^*(m)$ be the sum of the weights of the affinity edges incident on m such that the other vertex incident on the affinity edge precedes m in the PEO. When there is a choice between multiple vertices with maximal T-values, W_2^* is used as a tiebreaker; W^* is used as a second tiebreaker if multiple vertices remain.

In most cases, BMCS and BMCS-2 find the same solution; however, we identified one case where BMCS-2 does better. This example, once again, can only be understood in the context of Step (4), and is delayed until Sect. 4.4.7.

4.4 Step (4): color assignment and propagation

Gavril's [20] algorithm for optimal chordal coloring does not account for affinity edges and their weights when assigning colors. OCC's color assignment and propagation is an adaptation of Gavril's algorithm to perform better coalescing; it borrows the following correctness invariant: a legal k-coloring is maintained for the induced subgraph $G_i = (V_i, E_i)$ when vertex v_i is assigned a color. OCC's method, however, is completely different because it must account for affinity edge weights.

Section 4.4.1 describes a simplified version of the color assignment heuristic. Section 4.4.2 introduces the process of de- and re-pseudo-coalescing, which are unique contributions of this paper; examples are shown in Sects. 4.4.3 and 4.4.4. Section 4.4.5 describes the full heuristic in detail. Lastly, Sects. 4.4.6 and 4.4.7 present the examples alluded to in Sects. 4.2 and 4.3; these examples illustrate situations where illegal pseudo-coalescing outperforms optimistic pseudo-coalescing and where BMCS-2 outperforms BMCS.

4.4.1 Simplified color assignment heuristic

Let $C_A(v_i)$ be the set of vertices with which v_i is pseudo-coalesced. OCC makes every effort to assign the same color to all of the vertices in $C_A(v_i)$; unfortunately, this is not always possible. Let v_i be the first vertex in $C_A(v_i)$ to receive a color $c = color(v_i)$. There must be an affinity edge $a = (v_i, v_j, w)$, $j > i$, such that $v_j \in C_A(v_i)$; otherwise, there would be no incentive, from the perspective of satisfying affinity edges, to assign the same color to v_i and the other vertices in $C_A(v_i)$. v_i receives a color before v_j since $j > i$.

When processing v_j, the ideal color to select would be c, since this choice would satisfy a. This is similar to *biased coloring* [7]. Unfortunately, there may be a vertex $v_l, i < l < j$, that interferes with v_j and has $color(v_l) = c$. Thus, after assigning c to v_i, we also wish to bias the choice of color assigned to v_l *away* from c. To do this, OCC optimistically

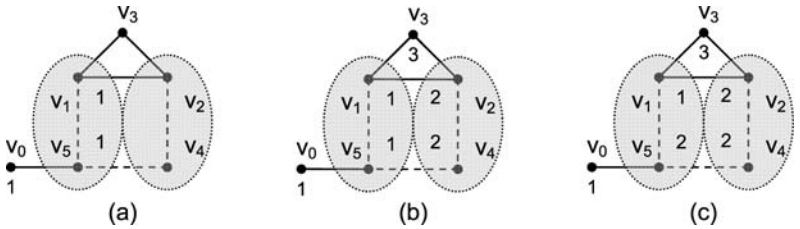


Fig. 5 Illustration of color assignment and propagation: a pre-assigned color cannot be confirmed for v_5. v_1 and v_5 are pseudo-coalesced, as are v_2 and v_4. Color 1 is assigned to v_0 and v_1, and then propagated to v_5 (**a**); color 2 is assigned to v_2 and propagated to v_4; color 3 is assigned to v_3 (**b**). v_4 confirms color 2; v_5 cannot confirm color 1 due to the interference with v_0; given a choice between colors 2 and 3, v_5 selects color 2 due to the affinity with v_4 (**d**)

propagates color c to all vertices in $C_A(v_i)$, including v_j, when c is assigned to v_i. OCC may change the color when v_j is processed; this does not violate the invariant since $v_j \notin V_i$. The choice of color assigned to v_l is biased away from c. If no other colors are available for v_l when v_l is processed, then c will be chosen for v_l; the fact that interfering vertices v_l and v_j have the same color do not violate the invariant since $v_j \notin V_l$; v_j, in this case, will not receive color c.

Let *free_colors* be the set of colors available for v_i, as used in traditional chordal color assignment; i.e.: *free_colors* contains the colors *not* assigned to vertices in $N^i(v_i)$. Let *optimistic_free_colors* be the set of colors not assigned to vertices in $N(v_i)$. *optimistic_free_colors* accounts for colors that have been propagated to vertices occurring after v_i in the PEO. If possible, a color from *optimistic_free_colors* is selected for v_i; if *optimistic_free_colors* is empty, then a color from *free_colors* is selected instead. Since the interference graph is chordal, *free_colors* is non-empty.

When c is propagated to v_i, it is assumed that c is preferred when choosing $color(v_i)$. If $c \in$ *free_colors*, then c is assigned to v_i; this process is called *confirmation*; otherwise v_i cannot confirm its pre-assigned color c. If there is an affinity edge (v_i, v_k, w), and $color(v_k) = c'$, then c' can be chosen for v_i as long as $c' \in$ *free_colors*.

Figure 5 shows an example, where optimistic pseudo-coalescing is used. Initially, v_1 and v_5 are optimistically pseudo-coalesced, as are v_2 and v_4. v_0 and v_1 are assigned color 1, which is then propagated from v_1 to v_5. v_2, v_3, and v_4 are assigned colors 2, 3, and 2. v_5, however, cannot confirm 1; 2 is chosen instead to satisfy affinity edge (v_4, v_5).

4.4.2 De- and re-pseudo-coalescing

If v_i cannot confirm c, then another color should be chosen for v_i. Although not explicitly stated above, this effectively de-pseudo-coalesces v_i from the other vertices in $C_A(v_i)$. Assigning color 2 to v_5 in Fig. 5(c) de-pseudo-coalesces v_5 from v_1; furthermore, the choice to assign color 2 to v_5 effectively re-pseudo-coalesces v_5 with v_4 and v_2.

Let v_i be the first vertex in $C_A(v_i)$ to be assigned a color, c; then c is propagated to the other vertices in $C_A(v_i)$. If later, c cannot be confirmed for some vertex $v_j \in C_A(v_i)$, then v_j is de-pseudo-coalesced from $C_A(v_i)$, yielding a new set, $L_A(v_j) = C_A(v_i) - \{v_j\}$. We then attempt to re-pseudo-coalesce v_j with a new set of pseudo-coalesced vertices: $C_A(v_k)$ for some other vertex $v_k \notin L_A(v_j)$, where adding v_j to $C_A(v_k)$ satisfies at least one affinity. It may also be possible to attract some affinity neighbors of v_i that belong to $L_A(v_j)$ into $C_A(v_k)$; details of this process are presented in the following examples.

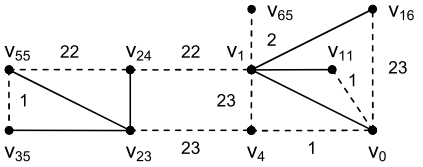


Fig. 6 Part of the interference graph for the sym_decrypt function, used to illustrate re-coalescing. The optimal solution satisfies all affinity edges except $(v_1, v_4, 23)$; leaving $(v_1, v_{24}, 22)$ and $(v_0, v_4, 1)$ unsatisfied is an equivalent optimal solution

4.4.3 Example 1: sym_decrypt

Figure 6 shows part of the interference graph of the *sym_decrypt* function of the *pegwit* benchmark [30]. Assume that illegal pseudo-coalescing is used: $C_A(v_0)$ contains all of the vertices. First, color 1 is assigned to v_0 and propagated to the vertices in $C_A(v_0)$. Since v_1 interferes with v_0, v_1 is de-pseudo-coalesced from $C_A(v_0)$ and color 2 is assigned to v_1.

Now, consider vertex v_{65}; the only affinity edge incident on v_{65} is $(v_1, v_{65}, 2)$; there is no benefit to keeping color 1 assigned to v_{65}, since this color can no longer satisfy this affinity. Instead, v_1 is able to de-pseudo coalesce v_{65} from $C_A(v_0)$ and re-pseudo-coalesce v_{65} with $C_A(v_1)$.

Continuing with the example, v_4 and v_{24} remain in $C_A(v_0)$. v_4 is incident on two affinity edges $(v_0, v_4, 1)$ and $(v_4, v_{23}, 23)$, in addition to $(v_1, v_4, 23)$. Keeping 1 as v_4's color satisfies the first two affinity edges, whose weight totals 24; changing v_4's color to 2 would satisfy the third affinity edge whose weight is 23; therefore, v_4 keeps its color and remains in $C_A(v_0)$.

In PEO order, v_4, v_{11}, v_{16} and v_{23} all confirm pre-assigned color 1. v_{24} cannot confirm color 1 since it interferes with v_{23}. Therefore, v_{24} de-pseudo-coalesces itself from $C_A(v_0)$; the only other set of pseudo-coalesced vertices incident on v_{24} is $C_A(v_1)$; therefore, v_{24} is re-pseudo-coalesced with $C_A(v_1)$. v_{24} then invites v_{55} to de-pseudo-coalesce from $C_A(v_0)$ and re-pseudo-coalesce with $C_A(v_1)$. Retaining v_{55}'s membership in $C_A(v_0)$ yields a net benefit of 1, due to affinity edge $(v_{35}, v_{55}, 1)$, while de-pseudo-coalescing and re-pseudo-coalescing with $C_A(v_1)$ yields a net benefit of 22 due to affinity edge $(v_{24}, v_{55}, 22)$; the latter option is preferable, so v_{55} is removed from $C_A(v_0)$ and inserted into $C_A(v_1)$; by similar reasoning, v_{35} is removed from $C_A(v_0)$ and inserted into $C_A(v_1)$ as well.

As the heuristic proceeds, color 2 is confirmed for v_{35}, v_{55}, and v_{65}, in order. This yields an optimal solution, wherein the only unsatisfied affinity edge is $(v_1, v_4, 23)$; another optimal solution leaves affinity edges $(v_0, v_4, 1)$ and $(v_1, v_{24}, 22)$ unsatisfied.

4.4.4 Example 2: RPE_grid_positioning

A fragment of the interference graph fragment for function *RPE_grid_positioning* from the *gsm* benchmark [30] is shown in Fig. 7. Assume that illegal pseudo-coalescing is used: $C_A(v_2)$ contains all of the vertices in the interference graph.

First, color 1 is assigned to vertex v_2, and is propagated to all other vertices in the graph. v_{13} and v_{22} confirm color 1; v_{31} takes color 2 instead, due to an interference with v_{22}. Vertices v_{53} and v_{59} are successfully attracted into $C_A(v_{22})$, so $C_A(v_2) = \{v_2, v_{13}, v_{22}, v_{46}, v_{47}, v_{52}, v_{57}\}$ and $C_A(v_{31}) = \{v_{31}, v_{53}, v_{59}\}$. v_{46} then confirms color 1; v_{47} cannot confirm color 1 due to interference edge (v_{46}, v_{47}); v_{47} is de-pseudo-coalesced from $C_A(v_2)$, and successfully attracts v_{57} as well. At this point, $C_A(v_2) = \{v_2, v_{13}, v_{22}, v_{46}, v_{52}\}$, $C_A(v_{31}) = \{v_{31}, v_{53}, v_{59}\}$ and $C_A(v_{47}) = \{v_{47}, v_{57}\}$. v_{52}, v_{53}, v_{57}, and v_{59} then confirm their

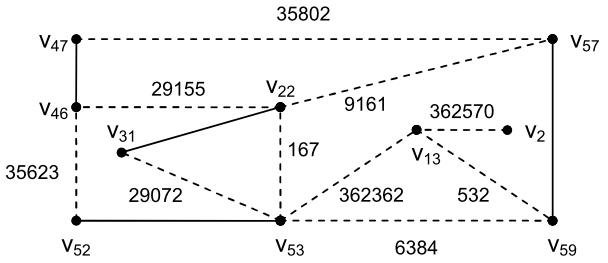


Fig. 7 Part of the interference graph for the RPE_grid_positioning function used to illustrate de- and re-coalescing. The optimal solution leaves affinity edges $(v_{22}, v_{57}, 9161)$ and $(v_{22}, v_{53}, 167)$ unsatisfied

colors in order. This yields two unsatisfied affinity edges: $(v_{22}, v_{53}, 167)$ and $(v_{22}, v_{57}, 9161)$: the optimal solution.

4.4.5 Detailed description of color assignment and propagation

Assume that vertex v_i cannot confirm its pre-assigned color, so it must be de-pseudo-coalesced from $C_A(v_i)$; recall that $L_A(v_i) = C_A(v_i) - \{v_i\}$. Let $v_j \notin L_A(v_i)$ be a vertex that is affinity-adjacent to v_i. Here, we entertain the possibility of adding v_i to $C_A(v_j)$. v_i cannot be added to $C_A(v_j)$ if v_i interferes with at least on vertex in $C_A(v_j)$, or if v_i has an interference neighbor v_k such that $color(v_k) = color(v_j)$; otherwise, it is possible to add v_i to $C_A(v_j)$. There may be multiple sets of pseudo-coalesced vertices to which v_i is affinity-adjacent; in this case, we must select the best one in terms of satisfying affinity edges.

The gain associated with pseudo-coalescing v_i with $C_A(v_j)$, under the assumption that doing so is legal, is as follows:

$$gain\left(v_i, C_A\left(v_j\right)\right) = \sum_{\substack{v_k \in C_A(v_j) \\ (v_i, v_k) \in A}} w\left(v_i, v_k\right) \tag{4}$$

v_i is added to the set of pseudo-coalesced vertices $C_A(v_j)$ such that $gain(v_i, C_A(v_j))$ is maximal. If there is no set $C_A(v_j)$ with positive aggregate gain, then v_i is inserted into a singleton set, i.e.: $C_A(v_i) = \{v_i\}$. If the vertices in the chosen set have been colored, then v_i is assigned that color; if they have not been colored, then an appropriate color is assigned to v_i and that color is propagated to all of the vertices in $C_A(v_i)$.

Now, we should entertain the possibility of de-pseudo-coalesecing affinity neighbors of v_i from their current sets and re-pseudo-coalescing them with $C_A(v_i)$, as was done in the case of vertices v_1 and v_{24} in Fig. 6, and v_{31} and v_{47} in Fig. 7; this is done only if doing so increases the number of satisfied affinity edges.

The process itself is a breadth-first search. Initially, each affinity neighbor u of v_i is inserted into an empty queue, denoted Q. At every step, a vertex is dequeued from Q. A test, described below, determines whether u is de-pseudo-coalesced from its current set and re-pseudo-coalesced with $C_A(v_i)$, or not. If u passes the test and is brought into $C_A(v_i)$, then each affinity neighbor of u that is neither enqueued nor belongs to $C_A(v_i)$ already is enqueued. The process repeats until Q is empty. This process may bring many of the vertices in $L_A(v_i)$ into the new $C_A(v_i)$.

Without loss of generality, let us dequeue vertex u. If u interferes with any vertex in $C_A(v_i)$, or if u is assigned the same color as any vertex in $N(C_A(v_i))$, then u cannot be pseudo-coalesced with $C_A(v_i)$. Otherwise, it is perfectly legal to de-pseudo-coalesce u from $C_A(u)$ and re-pseudo-coalesce u with $C_A(v_i)$; however, this should only be done if doing so improves the objective function.

Let *component_gain*(u) be the gain associated with leaving u in $C_A(u)$ and *component_gain*(v_i) be the gain associated with bringing u into $C_A(v_i)$:

$$component\_gain(u) = \sum_{x \in N_A(u) \cap C_A(u)} w(u, x), \tag{5}$$

$$component\_gain\,(v_i) = \sum_{y \in N_A(u) \cap C_A(v_i)} w\,(u, y)\,. \tag{6}$$

If *component_gain*$(u) >$ *component_gain*(v_i), then it is beneficial to remove v_i from $C_A(v_i)$ and insert v_i into $C_A(u)$; otherwise, v_i remains in $C_A(v_i)$.

For example, refer back to processing vertex v_{53} in Fig. 7, which is attracted to $C_A(v_{31})$. *component_gain*$(v_{53}) = w(v_{22}, v_{53}) + w(v_{53}, v_{59}) = 167 + 6384 = 6551$; meanwhile, *component_gain*$(v_{31}) = w(v_{31}, v_{53}) = 29072 >$ *component_gain*(v_{53}). Therefore, v_{53} de-pseudo-coalesces with $C_A(v_{53})$ and re-pseudo-coalesces with $C_A(v_{31})$.

Likewise, when v_{47} attracts v_{57} in Fig. 7, *component_gain*$(v_{57}) = w(v_{22}, v_{57}) = 9161$, while *component_gain*$(v_{47}) = w(v_{47}, v_{57}) = 35802$. Therefore, v_{57} de-pseudo-coalesces with $C_A(v_{57})$ and re-pseudo-coalesces with $C_A(v_{47})$.

4.4.6 Example 3: susan_smoothing

A fragment of the interference graph of the procedure *susan_smoothing*, taken from the *susan* benchmark [23], will be used to illustrate the advantage of illegal over optimistic pseudo-coalescing; it is shown in Fig. 8. The optimal solution leaves affinity edges (v_0, v_{31}) and (v_{56}, v_{60}) unsatisfied. Since many affinity edges have the same weight, optimistic pseudo-coalescing may produce different results, depending on the order in which the edges are processed. The worst possible solution is $C_A(v_0) = \{v_0, v_{31}, v_{33}, v_{35}, v_{53}, v_{56}, v_{60}\}$, which leaves 4 unsatisfied affinity edges: $(v_{29}, v_{31}, 1)$, $(v_{30}, v_{31}, 1)$, $(v_{58}, v_{60}, 1)$, and $(v_{59}, v_{60}, 1)$. In this case, color 1 is assigned to v_0 and then propagated to the remaining vertices in $C_A(v_0)$; color 1 is then confirmed when each vertex is processed during color assignment.

Under illegal pseudo-coalescing, $C_A(v_0)$ initially contains all of the vertices in Fig. 8. v_0 is assigned color 1; v_{29} and v_{30}, which both interfere with v_0 are assigned color 2. When v_{29} is assigned color 2, it de-pseudo-coalesces from $C_A(v_0)$ and tries to attract v_{31} into $C_A(v_{29})$; the attraction fails, because v_0 and v_{30}, the two other affinity neighbors of v_{31}, belong to $C_A(v_0)$. Assigning color 2 to v_{30} also de-pseudo-coalesces v_{30} from $C_A(v_0)$. Thus, at this point, $C_A(v_{29}) = \{v_{29}\}$ and $C_A(v_{30}) = \{v_{30}\}$. v_{30} successfully attracts v_{31} into $C_A(v_{30})$, since v_{31} has two affinity neighbors with color 2 (v_{29} and v_{30}) and one with color 1 (v_0); v_{31} then successfully attracts v_{29} into $C_A(v_{31}) = C_A(v_{30})$. The same thing happens on the left-hand-side of Fig. 8 with vertices v_{58}, v_{59}, and v_{60}, yielding an optimal solution.

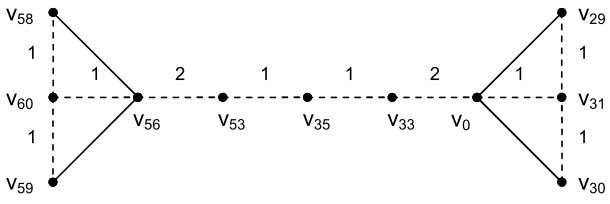


Fig. 8 Part of the interference graph for the susan_smoothing function, used to illustrated re-coalescing

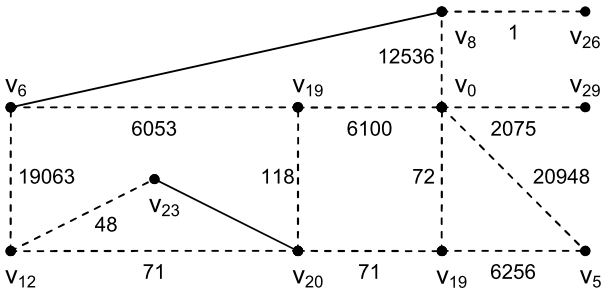


Fig. 9 Part of the interference graph for the jpeg_fill_bit_buffer function

4.4.7 Example 4: jpeg_fill_bit_buffer

Figure 9 shows a fragment of the interference graph of the *jpeg_fill_bit_buffer* function from the *jpeg_6a* benchmark [30]. The optimal solution leaves affinity edges $(v_6, v_{14}, 6053)$ and $(v_{12}, v_{20}, 71)$ unsatisfied. Vertices are listed in their BMCS ordering; this example will show that BMCS-2 yields a better solution.

The difference between the PEOs generated by the BMCS and BMCS-2 heuristic involves vertex v_6 and v_8. v_0 is the first vertex selected; $v_1, \ldots, v_5$ are not shown in Fig. 9. In the traditional BMCS, $W^*(v_8) = 12357 > W^*(v_6) = 25116$, so v_6 precedes v_8; in BMCS-2, $W_2^*(v_6) = 0 < W_2^*(v_8) = 12356$, v_8 would precede v_6 in the PEO. We assume that illegal pseudo-coalescing is used in both cases.

Consider the vertices in BMCS order. First, vertex v_0 receives color 1, which is propagated to the other vertices; v_5 and v_6 confirm color 1. Due to the interference edge (v_6, v_8), v_8 does not confirm color 1 and choses color 2 instead, which leaves affinity edge $(v_0, v_8, 12356)$ unsatisfied: this is sub-optimal. v_8 is unable to attract v_0 to $C_A(v_8)$, due to the weights of the other three affinity edges incident on v_0, which sum to 29195: all of the vertices incident on three edges still belong to $C_A(v_0)$. In the PEO produced by BMCS-2, $\sigma(v_8) = 6, \sigma(v_6) = 8$; the other vertices keep the same order as the BMCS. v_8 confirms color 1, while v_6 is forced to accept color 2. v_6 then successfully attracts v_{12} and v_{23} into $C_A(v_6)$; the remaining vertices then confirm their colors and the optimal solution is found.

4.5 Refinement

OCC guarantees a legal coloring but does not guarantee optimality in terms of the number of affinities satisfied. Two refinement techniques introduced in the next two subsections, can be used to try to locally improve the coalescing solution of OCC.

4.5.1 Local refinement

Local refinement processes each affinity edge $a = (u, v, w)$ where u and v are assigned different colors. If swapping the colors assigned to u and v causes an illegal coloring, then a is discarded. If the swap is legal, then the swap is accepted if it improves the solution quality; otherwise, the swap is suppressed. After deciding to accept or reject the swap, the next affinity edge is processed.

4.5.2 Aggressive refinement

Like its local counterpart, aggressive refinement processes affinity edges one-by-one, only refining those that are not satisfied; in this case, the refinement process is much more complicated and has a greater time complexity per miscolored affinity edge.

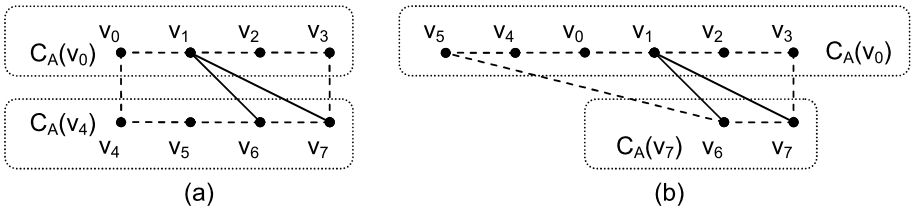


Fig. 10 Illustration of aggressive refinement. Two sets of pseudo-coalesced vertices are shown (**a**); after aggressive refinement, vertices v_4 and v_5 can be de-pseudo-coalesced from $C_A(v_7)$ and re-pseudo-coalesced with $C_A(v_0)$ because they do not interfere with any vertices in $C_A(v_0)$; v_6 and v_7 remain in $C_A(v_7)$ after pseudo-coalescing because they interfere with $v_1 \in C_A(v_0)$

Let $a = (u, v)$ be an unsatisfied affinity edge. The intuitive explanation for the aggressive refinement step, illustrated in Fig. 10, is as follows: we consider the possibility of de-coalescing all of the vertices in $C_A(v)$ that *do not* interfere with any vertices in $C_A(u)$, and re-psuedo-coalescing all of these vertices into $C_A(u)$. This is done only if it improves the objective function; it may also be necessary to find a new color to assign to the vertices in $C_A(u)$ after this transformation, in order to ensure legality.

The first step is to compute a set, *free_colors*, which contains a spectrum of colors which we *may* choose to assign to u and v during refinement. Ideally, we will find a new color that can be assigned to both u and v, thereby satisfying a, that can also be assigned to some of the vertices in $C_A(u)$ and $C_A(v)$, while ensuring a legal coloring and improving the objective function. Initially, $free\_colors = \{1, 2, \ldots, k\}$. For each vertex $n \in N(C_A(u)) \cup N(C_A(v))$, $color(n)$ is removed from *free_colors*. If *free_colors* becomes empty, we discard a and move on to the next unsatisfied affinity edge; otherwise, we proceed with a. *free_colors* is no longer needed after this point. It simply evaluates the feasibility of finding a color assignment that satisfies a given the colors assigned to the remainder of the interference graph.

Let $T_{u,v} = N(C_A(u)) \cap C_A(v)$; these are the sets of vertices belonging to $C_A(v)$ that cannot possibly be re-pseudo-coalesced with $C_A(u)$ due to interferences, e.g.: if $v_0 = u$ and $v_4 = v$ in Fig. 10(a), then $T_{u,v} = \{v_6, v_7\}$ due to the interference edges. Aggressive refinement tries to de-pseudo-coalesce all of the vertices belonging to $C_A(v) - T_{u,v}$ and re-pseudo-coalesce them with $C_A(u)$ instead.

Let t be a vertex in $T_{u,v}$. The interfering vertex affinity cost of t, denoted $I(t)$, effectively measures the force with which the vertices in $T_{u,v}$ hold onto the remaining vertices in $C_A(v)$. $I(t)$ is the sum of the weights of the affinity edges between t and its affinity neighbors in $C_A(v)$, i.e.:

$$I(t) = \sum_{x \in N_A(t) \cap C_A(v)} w(t, x). \tag{7}$$

The aggregate interfering vertex affinity cost, denoted I^*, is the sum of the interfering vertex affinity costs over all vertices in $T_{u,v}$:

$$I^* = \sum_{t \in T_{u,v}} I(t). \tag{8}$$

I^* effectively measures the estimated benefit of doing nothing, i.e.: leaving the vertices in $C_A(v) - T_{u,v}$ pseudo-coalesced with $T_{u,v}$.

The counterpart to I^*, which measures the strength with which vertices in $C_A(u)$ attract vertices belonging to $C_A(v) - T_{u,v}$ is called the merged component affinity cost, and is denoted M^*. Let $z \in C_A(v) - T_{u,v}$. The merged component affinity cost associated with z, denoted $M(z)$, is computed as follows:

$$M(z) = \sum_{x \in N_A(z) \cap C_A(u)} w(x, z). \tag{9}$$

M^* is then computed as follows:

$$M^* = \sum_{z \in C_A(v) - T_{u,v}} M(z). \tag{10}$$

If $M^* > I^*$, then it is beneficial to de-pseudo-coalesce the vertices in $C_A(v) - T_{u,v}$ from $C_A(v)$ and re-pseudo-coalesce them with $C_A(u)$. Let $C_A^*(u) = C_A(u) \cup (C_A(v) - T_{u,v})$ be a new name given to $C_A(u)$ after the re-pseudo-coalescing. In general, we may not be able to simply assign the same color used for $C_A(u)$ to the vertices in $C_A(v) - T_{u,v}$ due to interferences. Thus, we entertain the possibility of recoloring all of the vertices in $C_A^*(u)$ and $T_{u,v}$ in order to ensure legality. If $v \in C_A^*(u)$, then a new color must be found for the vertices in $C_A(v) \cap T_{u,v}$ as well. This is done using *free_colors* arrays, as described previously, for both $C_A^*(u)$ and $T_{u,v}$: if a free color is found for the vertices in $C_A^*(u)$ and for $T_{u,v}$, then the aggressive refinement has been successful for affinity edge a; otherwise, it is rejected since it cannot find a legal coloring.

Aggressive refinement is not commutative with respect to both u and v. If aggressive refinement fails, the process can be repeated for a with the roles of u and v reversed.

If aggressive refinement fails, the process can be repeated for affinity edge $a' = (v, u)$, which is the same affinity edge, but with the roles of u and v reversed.

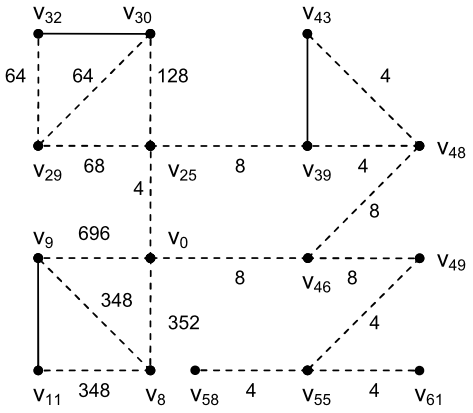
4.5.3 Example of aggressive refinement

Figure 11 shows a fragment of the interference graph for the procedure *get_dht* from the *jpeg_djpeg* benchmark. The table on the right-hand-side of Fig. 11 shows the color assignment before and after aggressive refinement. Initially, four affinity edges are not satisfied: $(v_8, v_{11}, 348)$, $(v_{29}, v_{32}, 64)$, $(v_{46}, v_{48}, 8)$ and $(v_{39}, v_{48}, 4)$; after aggressive refinement, the optimal solution leaves three affinity edges unsatisfied: $(v_8, v_{11}, 348)$, $(v_{29}, v_{32}, 64)$, and $(v_{43}, v_{48}, 4)$.

During aggressive refinement, unsatisfied affinity edges are processed in descending order of weight. Aggressive refinement fails for the first three affinity edges, but triggers recoloring when processing the unsatisfied affinity edge $(v_{39}, v_{48}, 4)$.

Let $u = v_{39}$ and $v = v_{48}$. $C_A(u) = \{v_0, v_8, v_9, v_{25}, v_{29}, v_{30}, v_{39}, v_{46}, v_{49}, v_{55}, v_{58}, v_{61}\}$ and $C_A(v) = \{v_{43}, v_{48}\}$; $T_{u,v} = \{v_{43}\}$ due to the interference edge (v_{39}, v_{43}). $I^* = 4$, which is the weight of affinity edge $(v_{39}, v_{48}, 4)$; there is no other affinity edge incident on v_{39}. $M^* = 12$, due to affinity edges $(v_{39}, v_{48}, 4)$ and $(v_{46}, v_{48}, 8)$.

All of the vertices in $C_A(u)$ are initially assigned color 0; this color is unavailable for v_{48}: although not shown in Fig. 11, v_{48} interferes with another vertex whose color is 0. The search for a free color for the set $C_A^*(u) = C_A(u) \cup \{v_{48}\}$ finds 10 as a free color. v_{43} retains its originally assigned color, 9.



Vertex	Before	After
v_0	0	10
v_8	0	10
v_9	0	10
v_{11}	7	7
v_{25}	0	10
v_{29}	0	10
v_{30}	0	10
v_{32}	8	8
v_{39}	0	10
v_{43}	9	9
v_{46}	0	10
v_{48}	9	10
v_{49}	0	10
v_{55}	0	10
v_{58}	0	10
v_{61}	0	10

Fig. 11 Illustration of aggressive refinement on a portion of the interference graph for the procedure *get_dht* in the *jpeg_djpeg* benchmark. The color assigned to each vertex is shown before and after aggressive refinement; miscolored affinity edge (39, 48, 4) triggers the re-coloring

4.6 Time complexity

4.6.1 Time complexity of simplification

The time complexity of simplification is $O(|V| + |E|)$: each vertex must be examined once: when a vertex is removed, the degree of its neighbors is reduced, following which the neighbor can be tested for simplifiability.

4.6.2 Time complexity of optimistic pseudo-coalescing

The time complexity of optimistic pseudo-coalescing is $O[|A|(\log|A| + |V| + |E|)]$. The time complexity of sorting the affinity edges is $O(|A| \log|A|)$, the time complexity of a comparison-based sort; alternatively, if the maximum affinity edge weight is W, then Counting Sort, which has a time complexity of $O(|A| + W)$, can be used [36]. Each affinity edge must be processed, and the time complexity of enumerating the neighborhood $N(S)$ of each set of pseudo-coalesced vertices S is $O(|V| + |E|)$: the time complexity of a depth- or breadth-first search; this justifies the $O[|A|(|V| + |E|)]$ term.

4.6.3 Time complexity of illegal pseudo-coalescing

The time complexity of illegal pseudo-coalescing is $O(|V| + |A|)$: the R_A sets are found by a breadth/depth-first search through the affinity edges of the interference graph.

4.6.4 Time complexity of color assignment and propagation

The time complexity of Gavril's chordal coloring algorithm is $O(|V| + |E|)$. Let P represent the complexity of the color propagation step for each vertex. Thus, the overall time complexity is $O(|V|P + |E|)$; the complexity of the P is discussed next.

During color assignment and propagation, u is the vertex that has been removed from the queue; we must process each affinity neighbor v of u to determine whether or not v can be added to $C_A(u)$, which requires computing the *component_gain* terms. The time complexity

of both of these steps is $T = O[|N(C_A(u))| + |C_A(u)| + |N(v)| + |N_A(v)|]$, ignoring vertices that have already been enqueued; for the sake of simplicity, we assume that these complexity terms are the same for each vertex.

If q is the total number of vertices that are enqueued during color propagation, so $P = O(qT)$ and the overall complexity is $O(|V|qT + |E|)$. In actuality, there is a different T term for each vertex, since the four components of the T term vary from vertex-to-vertex (and may change dynamically as vertices are moved from one C_A set to another). It follows that $qT = O(|V|)$, a loose bound, since at most $|V|$ vertices are processed each time color propagation is performed. At most $O(|A|)$ vertices will be uncovered during the breadth first search; however, $|N(C_A(u))| + |N_A(v)| = O(|V|)$ in the worst case, since $|N(C_A(u))|$ and $|N_A(v)|$ include their interference neighbors. Therefore, we report the overall complexity as $O(|V|^2 + |E|) = O(|V|^2)$.

4.6.5 Time complexity of local refinement

The time complexity of processing an affinity edge $a = (u, v)$ is $O[|N(u)| + |N_A(u)| + |N(v)| + |N_A(v)|]$. Local refinement examines the interference neighbors of u and v to determine the legality of the swap, and the affinity neighbors of u and v to incrementally compute objective function resulting from the swap. In the worst case, $O(A)$ affinity edges are processed. The complexity term for a single affinity edge is aggregated over all of the unsatisfied affinity edges. In the worst case, it becomes $O(|A||V|)$.

4.6.6 Time complexity of aggressive refinement

The time complexity of aggressive refinement is $O(|A||V|)$. In the worst case, $O(|A|)$ affinity edges must be processed, under the assumption that only a constant number of affinity edges are satisfied to begin with. For each unsatisfied affinity edge, $O(|V|)$ vertices must be processed in order to compute the I and M terms.

4.6.7 Time complexity of unsimplify

During unsimplification, each vertex v is popped from the stack and reattached to the graph; it is given the smallest color not assigned to its neighbors. At most $|N(v)|$ neighbors are processed, and $|N(v)| + 1$ colors are examined as possibilities for v; in aggregation, at most $|E|$ interferences edges in G will be processed. Since at most $|V|$ vertices are on the stack to begin with, the time complexity is $O(|V| + |E|)$.

4.6.8 Time complexity of OCC

From Sects. 4.6.1–4.6.7, the time complexity of OCC is $O(|V|^2 + |A||V|)$.

All of the copies in a procedure can be eliminated during SSA construction; thus, the only affinities that remain are those due to φ-functions. In practically all cases we have observed, $|A| \ll |V|$; under this assumption, the complexity simplifies to $O(|V|^2)$.

5 Experimental results

A set of embedded and multimedia benchmarks taken from the Mediabench [30] and MiBench [23] benchmark suites were selected for evaluation. The *Machine SUIF* compiler

Table 1 Number of registers used for each benchmark

Benchmark	Registers	Benchmark	Registers
adpcm_coder	14	jpeg_cjpeg	18
adpcm_decoder	14	jpeg_djpeg	39
blowfish	14	mpeg2dec	21
crc32	8	mpeg2enc	45
dijkstra	6	patricia	9
FFT	13	pegwit	13
g721_decoder	16	sha	10
g721_encoder	16	susan	20
gsm	16		

was used to compile an instrumented version of each application to collect profiling data (basic block execution frequencies) and to generate the interference graphs with affinity edges weighted based on the profiling information gathered during compilation. Each procedure was converted to SSA Form before generating its interference graph, which is provably chordal.

As our goal is to study the coalescing problem, we did not perform spilling. For each application let $G = \{G_1, \ldots, G_n\}$ be the set of interference graphs for each procedure; we targeted each application toward a RISC architecture with $R = \max\{\chi(G_1), \ldots, \chi(G_n)\}$ general purpose registers. R is the minimum number of registers that ensure no variables must be spilled, except possibly across procedure calls. Since each procedure is in SSA Form, each G_i is chordal, so $\chi(G_i)$ can be computed in polynomial-time [20].

Table 1 lists each benchmark and the number of registers in the corresponding target architecture. Each interference graph was colored 3 times: first using the OCC heuristic; second, using the well-established iterated coalescing (IRC) heuristic [21]; and third, using an optimal ILP formulation (OPT) [22].

Table 2 shows the number of copies dynamically executed for each benchmark, i.e.: the weighed sum of the unsatisfied affinity edges after coloring; the number of dynamically executed copies (based on the profiling information) is shown for OPT; the number of additional copies executed is shown for IRC and OCC. OCC was able to find the optimal coloring solution for each interference graph in the study; IRC was able to find optimal solutions for many, but not all of the benchmarks.

In Table 2, OCC was implemented with BMCS-2, illegal pseudo-coalescing, and both refinement stages (aggressive followed by local). To better understand the behavior of OCC, Table 3 presents results using four different configurations with refinement suppressed: OCC-BI (BMCS-2 + illegal pseudo-coalescing), OCC-BO (BMCS-2 + optimistic pseudo-coalescing), OCC-MI (MCS + illegal pseudo-coalescing), and OCC-MO (MCS + optimistic pseudo-coalescing).

OCC-MI performed poorly: its results are substandard compared to the others, suggesting that illegal pseudo-coalescing is only useful in conjunction with a BMCS that tries to ensure that vertices incident on high-weight affinity edges are assigned colors as early as possible. OCC-BO and OCC-MO found the same results for all benchmarks: all but three benchmarks were solved optimally, and, across all benchmarks, only executed 7 copies in excess of the optimal solution.

Beginning with a set of optimistic pseudo-coalesced vertices, the lack of interferences between vertices that were pseudo-coalesced together minimized the amount of de- and re-pseudo-coalescing; the quality of the results, however, was dependent of the quality of

Table 2 Number of dynamically executed copy instructions for each benchmark. − means that IRC/OCC found optimal solutions; $+X$ indicates that IRC found solutions that execute X copies in excess of the optimal solution

Benchmark	OPT	IRC	OCC
adpcm_rawcaudio	6995016	–	–
adpcm_rawdaudio	6995016	–	–
blowfish	0	–	–
crc32	53322406	–	–
dijkstra	0	–	–
fft	8209	–	–
g721_decode	0	–	–
g721_encode	0	–	–
gsm	1909187	+2990	–
jpeg_cjpeg	541326	–	–
jpeg_djpeg	272636	+6283	–
mpeg2dec	2115	–	–
mpeg2enc	95197	–	–
patricia	1820	–	–
pegwit	48	+24	–
sha	442	–	–
susan	2	–	–

Table 3 Number of dynamically executed copies for different configurations of the OCC heuristic

Benchmark	OCC-BI	OCC-BO	OCC-MI	OCC-MO
adpcm_rawcaudio	–	–	–	–
adpcm_rawdaudio	–	–	–	–
blowfish	–	–	–	–
crc32	–	–	–	–
dijkstra	–	–	–	–
fft	–	–	–	–
g721_decode	–	–	–	–
g721_encode	–	–	–	–
gsm	–	–	+792005	–
jpeg_cjpeg	–	–	–	–
jpeg_djpeg	+21	+4	+742	+4
mpeg2dec	–	–	–	–
mpeg2enc	–	–	+224	–
patricia	–	–	+66971	–
pegwit	–	+1	+1	+1
sha	–	–	–	–
susan	–	+2	+1	+2

the optimistic pseudo-coalescing; as there are known dependencies on the order in which affinity edges are pseudo-coalesced, this situation seems less than ideal, even if it the results were generally favorable.

OCC-BI found the optimal solution for all benchmarks except *jpeg_djpeg*, for which it executed 21 copies in excess of the optimal solution. With local, but not aggressive, re-

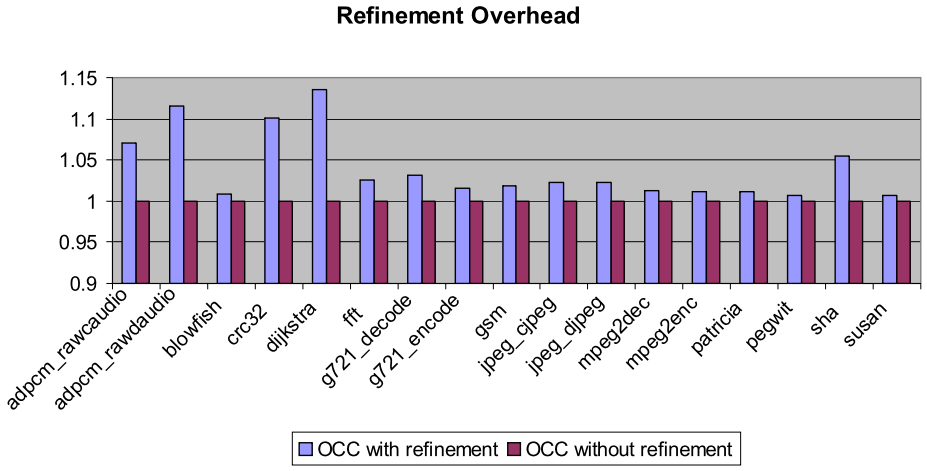


Fig. 12 The overhead of performing refinement

finement the number of excess copies was reduced to +13; with aggressive, but not local, refinement, the number of excess copies was reduced to +8; with both types of refinement, the optimal solution was achieved.

Next, we measured the overhead of performing refinement. We colored each interference graph with the OCC-BI heuristic, with and without (both types of) refinement; for each measurement, each interference graph was colored 5000 times and the average runtime was taken. The results are shown in Fig. 12. The IRC and OCC heuristics were run on a *Dell Latitude D810* laptop with an *Intel Pentium M* processor running at 2.0 GHz with 1.0 GB of RAM; the operating system used was *Fedora Core 5*.

The overhead of refinement ranged from 1.00 (*susan*) to 1.14 (*dijkstra*). In general, the largest overheads were observed for the benchmarks with the smallest overall runtimes (*adpcm_rawcaudio*, *adpcm_rawdaudio*, *crc32*, *dijkstra*, *sha*); these benchmarks all had very few interference graphs, all of which were colored quite quickly, thereby amplifying the impact of refinement. For the remaining benchmarks, the overhead of refinement was 1.03 or less.

The overhead of refinement is clearly tolerable for traditional "offline" compilation; however, for *just-in-time (JIT)* compilers, the overhead of refinement can only be justified by its savings in runtime execution. For OCC-MI, refinement saves a total of 21 dynamically executed instructions; this cannot be justified; as the compiler executes, the cost of saving/restoring caller/callee-save registers for the call to one (of the two) refinement function(s) alone will easily eclipse 21 instructions, let alone the runtime cost of performing the refinement.

It should also be noted that a JIT compiler is likely to use a faster register allocator, such as linear scan [34], or one of its variants, that sacrifices solution quality in order to reduce compile time; in particular, linear scan does not use an interference graph, and does not perform coalescing at all.

Table 4 compares the runtime of the OCC heuristic with the runtime of OPT and IRC. IRC and OCC were run using the same setup as described above. The experiments using OPT were performed by Sebastian Hack on a *Dell Latitude D420* laptop with an *Intel Core Duo U2500* processor running at 1.2 GHz and with 2.0 GB of RAM; the operating system was *Ubuntu Feisty*, and *CPLEX 7.0* was used to solve the ILP.

Table 4 Runtime (normalized to OCC) of OPT, IRC, and OCC

Benchmark	OPT	IRC	OCC
adpcm_rawcaudio	904	3.47	1
adpcm_rawdaudio	2841	3.39	1
blowfish	3111	2.05	1
crc32	139	1.28	1
dijkstra	188	1.12	1
fft	573	2.69	1
g721_decode	2394	3.36	1
g721_encode	3611	3.25	1
gsm	1403	2.21	1
jpeg_cjpeg	1659	2.13	1
jpeg_djpeg	10144	2.37	1
mpeg2dec	2661	2.03	1
mpeg2enc	2088	4.17	1
patricia	442	1.73	1
pegwit	140881	0.871	1
sha	670	2.88	1
susan	279256	0.105	1
Average	26645	2.30	1

On average, the runtime of IRC was 2.3× greater than that of OCC. For two benchmarks, *pegwit* and *susan*, IRC was faster: significantly, in the case of *susan*. OPT, however, ran considerably slower than OCC: 139× to 279, 256×. This is to be expected, given that, absent a proof that P = NP, the optimal solution to NP-Complete problems can only be computed in exponential worst-case time. It should be noted that the implementation of the OCC heuristic used for these experiments is based on a different code base than an earlier version [9] that was published previously.

6 Conclusion

The optimistic chordal coloring (OCC) coalescing heuristic was introduced for chordal graphs, which arise from SSA Form programs. OCC is an extension of Gavril's [20] algorithm that computes a minimum coloring of a chordal graph. OCC maintains the same correctness invariant as Gavril's algorithm, which offers stronger conservative guarantees than traditional mechanisms of conservative coalescing. Through pseudo-coalescing, rather than traditional coalescing via merging, OCC is able to identify sets of affinity-related vertices to which it would ideally like the assign the same color, while preserving the chordal graph property and ensuring that a k-coloring is found. On a set of interference graphs from a set of multimedia and embedded benchmarks, OCC found the optimal coalescing solution in all cases, and ran 2.30× faster than iterated register coalescing.

References

1. Boissinot B, Hack S, Grund D, De Dinechin BD, Rastello F (2008) Fast liveness checking for SSA-form programs. In: Proceedings of the 2008 IEEE/ACM symposium on code generation and optimization, Boston, MA, USA, 6–9 April 2008

2. Bouchez F (2005) Register allocation and spill complexity under SSA. MS thesis, Technical Report RR2005-33, ENS-Lyon, Lyon France, August 2005
3. Bouchez F, Darte A, Guillon C, Rastello F (2006) Register allocation: what does the NP-Completeness proof of Chaitin et al. really prove? Or revisiting register allocation: why and how? In: Proceedings of the 19th international workshop on languages and compilers for parallel computing, New Orleans, LA, USA, November 2006
4. Bouchez F, Darte A, Rastello F (2007) On the complexity of register coalescing. In: Proceedings of the international symposium on code generation and optimization, San Jose, CA, USA, March 2008, pp 102–114
5. Bouchez F, Darte A, Rastello F (2007) On the complexity of spill everywhere under SSA form. In: Proceedings of the ACM SIGPLAN/SIGBED conference on languages, compilers, and tools for embedded systems, San Diego, CA, USA, June 2007, pp 103–112
6. Briggs P, Cooper KD, Harvey TJ, Simpson LT (1998) Practical improvements to the construction and destruction of static single assignment form. Softw Pract Exp 28:859–881
7. Briggs P, Cooper KD, Torczon L (1994) Improvements to graph coloring register allocation. ACM Trans Program Lang Syst 16:428–455
8. Brisk P, Dabiri F, Jafari R, Sarrafzadeh M (2006) Optimal register sharing for high-level synthesis of SSA form programs. IEEE Trans Comput-Aided Des Integr Circuits Syst 25:772–779
9. Brisk P, Sarrafzadeh M (2007) Interference graphs for procedures in static single information form are interval graphs. In: Proceedings of the 10th international workshop on software and compilers for embedded systems, Nice, France, 20 April 2007, pp 101–110
10. Brisk P, Verma AK, Ienne P (2007) An optimistic and conservative register assignment heuristic for chordal graphs. In: Proceedings of the international conference on compilers, architecture, and synthesis for embedded systems, Salzburg, Austria, 30 September–3 October 2007, pp 209–217
11. Budimlic Z, Cooper KD, Harvey TJ, Kennedy K, Oberg TS, Reeves SW (2002) Fast copy coalescing and live range identification. In: Proceedings of the ACM SIGPLAN conference on programming language design and implementation, Berlin, Germany, 17–19 June 2002, pp 25–32
12. Chaitin GJ (1982) Register allocation and spilling via graph coloring. In: Proceedings of the 1982 SIGPLAN symposium on compiler construction, Boston, MA, USA, 23–25 June 1982, pp 98–105
13. Chaitin GJ, Auslander MA, Chandra AK, Cocke J, Hopkins ME, Markstein PW (1981) Register allocation via coloring. Comput Lang 6:47–57
14. Choi J-D, Cytron R, Ferrante J (1991) Automatic construction of sparse data flow evaluation graphs. In: Proceedings of the 18th ACM SIGPLAN/SIGACT symposium on principles of programming languages, Orlando, FL, USA, 21–23 January 1991, pp 55–66
15. Cooper KD, Torczon L (2003) Engineering a compiler. Morgan Kaufmann, Los Altos (now a subsidiary of Elsevier)
16. Cytron R, Ferrante J, Rosen BK, Wegman MN, Zadeck FK (1991) Efficiently computing static single assignment form and the control dependence graph. ACM Trans Program Lang Syst 13:451–490
17. Ershov AP (1962) Reduction of the problem of memory allocation in programming to the problem of coloring the vertices of graphs. Dokl Akad Nauk SSSR 142. English translation in Sov Math 3:163–165 (1962)
18. Fabri J (1979) Automatic storage optimization. In: Proceedings of the ACM SIGPLAN symposium on compiler construction, Denver, CO, USA, 6–10 August 1979, pp 83–91
19. Farach-Colton M, Liberatore V (2000) On local register allocation. J Algorithms 37:37–65
20. Gavril F (1972) Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SIAM J Comput 1:180–187
21. George L, Appel AW (1996) Iterated register coalescing. ACM Trans Program Lang Syst 18:300–324
22. Grund D, Hack S (2007) A fast cutting-plane algorithm for optimal coalescing. In: 16th international conference on compiler construction, Braga, Portugal, 26–30 March 2007, pp 111–125
23. Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB (2001) MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of the 4th IEEE workshop on workload characterization, Austin TX, USA, 2 December 2001, pp 3–14
24. Hack S, Goos G (2006) Register allocation for SSA-form programs in polynomial time. Inf Process Lett 98:150–155
25. Hailperin M (2005) Comparing conservative coalescing criteria. ACM Trans Program Lang Syst 27:571–582
26. Hashimoto A, Stevens J (1971) Wire routing by optimizing channel assignment within large apertures. In: Proceedings of the 8th workshop on design automation, Atlantic City, NJ, USA, 28–30 June 1971, pp 155–169
27. Kaluskar V (2003) An aggressive live range splitting and coalescing framework for efficient register allocation. MS thesis, Georgia Institute of Technology, Atlanta, GA, USA, December 2003

28. Kempe AB (1879) The geographical problem of the four colors. Am J Math 2:193–200
29. Lavrov SS (1961) Store economy in closed operator schemes. J Comput Math Math Phys 1:678–701. English translation in USSR Comput Math Math Phys 3:810–828 (1962)
30. Lee C, Potkonjak M, Mangione-Smith WH (1997) MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In: Proceedings of the 30th international symposium on microarchitecture, Research Triangle Park, NC, USA, 1–3 December 1997, pp 330–335
31. Lee JK, Palsberg J, Pereira FMQ (2007) Aliased register allocation for straight-line programs is NP-Complete. In: Proceedings of the 34th international colloquium on automata, languages, and programming, Wroclaw, Poland, 9–13 July 2007, pp 680–691
32. Park J, Moon S-M (2004) Optimistic register coalescing. ACM Trans Program Lang Syst 26:735–765
33. Pereira FMQ, Palsberg J (2006) Register allocation after classical SSA elimination is NP-Complete. In: Proceedings of the 9th international conference on foundations of software science and computation structures, Vienna, Austria, 25–31 March 2006, pp 79–83
34. Poletto M, Sarkar V (1999) Linear scan register allocation. ACM Trans Program Lang Syst 21:895–913
35. Rastello F, De Ferriere F, Guillon C (2004) Optimizing translation out of SSA using renaming constraints. In: Proceedings of the 2nd IEEE/ACM symposium on code generation and optimization, San Jose, CA, USA, 20–24 March 2004, pp 265–278
36. Seward HH (1954) Information sorting in the application of electronic digital computers to business operations. MS thesis. Massachusetts Institute of Technology, Cambridge, MA, USA
37. Sreedhar VC, Ju RD-C, Gillies DM, Santhanam V (1999) Translating out of static single assignment form. In: Proceedings of the 6th international symposium on static analysis, 22–24 September 1999, pp 194–210
38. Tarjan RE, Yannakakis M (1984) Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acylic hypergraphs. SIAM J Comput 13:566–579
39. Tseng C-J, Siewiorek DP (1986) Automated synthesis of data paths in digital systems. IEEE Trans Comput-Aided Des Integr Circuits Syst 5:379–395
40. Vegdahl SR (1999) Using node merging to enhance graph coloring. In: Proceedings of the ACM SIGPLAN conference on programming language design and implementation, Atlanta, GA, USA, 1–4 May 1999, pp 150–154