On the complexity of register coalescing

Florent Bouchez, Alain Darte, Fabrice Rastello

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

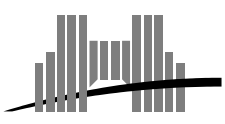
On the Complexity of Register Coalescing

Florent Bouchez
Alain Darte
Fabrice Rastello

April 2006

Research Report N° 2006-15

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr



On the Complexity of Register Coalescing

Florent Bouchez, Alain Darte, and Fabrice Rastello

Abstract

Due to the increasing latencies of memory accesses and recent developments on the SSA form, it has become important to revisit the spilling (load/store insertion) and register coalescing (removal of move instructions) problems in order to develop more aggressive register allocation strategies. This report is devoted to the complexity of register coalescing. We distinguish several optimizations that occur in most coalescing heuristics: a) aggressive coalescing removes as many moves as possible, regardless of the colorability of the resulting interference graph; b) conservative coalescing removes as many moves as possible while keeping the colorability of the graph; c) incremental conservative coalescing removes one particular move while keeping the colorability of the graph; d) optimistic coalescing coalesces all moves (when possible), aggressively, and gives up about as few moves as possible (de-coalescing) so that the graph becomes colorable. We (almost) completely classify the NP-completeness of these problems, discussing also on the structure of the interference graph (arbitrary, chordal, or k-colorable in a greedy fashion). We believe that such a study is a necessary step for designing new coalescing strategies.

Keywords: Register allocation, register coalescing, SSA form, chordal graph, NP-completeness, (greedy)-k-colorable graph.

Résumé

L'augmentation croissante de la durée des accès à la mémoire et des développements récents liés à la forme SSA poussent à revisiter les problèmes de « spilling » (placement des *loads* et *stores*) et de « coalescing » (suppression de *moves*) pour développer de nouvelles stratégies d'allocation de registres plus agressives. Ce rapport est consacré à la complexité des problèmes de coalescing. Nous distinguons plusieurs optimisations qui apparaissent dans les heuristiques de coalescing : a) le coalescing agressif supprime autant de *moves* que possible, quelle que soit la colorabilité du graphe d'interférences résultant ; b) le coalescing conservatif supprime autant de *moves* que possible tout en préservant la colorabilité du graphe ; c) le coalescing incrémental supprime un *move* particulier en conservant la colorabilité du graphe ; d) le coalescing optimiste supprime tous les *moves* (quand c'est possible), de façon agressive, et en réintroduit le plus petit nombre pour que le graphe redevienne colorable. Nous classifions (presque) complètement ces problèmes en termes de complexité, discutant également de la structure du graphe d'interférence (quelconque, triangulé, k-colorable de façon gloutonne). Nous pensons qu'une telle étude est un pas nécessaire pour concevoir de nouvelles stratégies de coalescing.

Mots-clés: Allocation de registres, "coalescing", forme SSA, graphe triangulé, NP-complétude, graphe (glouton)-k-colorable.

On the Complexity of Register Coalescing

Florent Bouchez, Alain Darte, and Fabrice Rastello

8th April 2006

Abstract

Due to the increasing latencies of memory accesses and recent developments on the SSA form, it has become important to revisit the spilling (load/store insertion) and register coalescing (removal of move instructions) problems in order to develop more aggressive register allocation strategies. This report is devoted to the complexity of register coalescing. We distinguish several optimizations that occur in most coalescing heuristics: a) aggressive coalescing removes as many moves as possible, regardless of the colorability of the resulting interference graph; b) conservative coalescing removes as many moves as possible while keeping the colorability of the graph; c) incremental conservative coalescing removes one particular move while keeping the colorability of the graph; d) optimistic coalescing coalesces all moves (when possible), aggressively, and gives up about as few moves as possible (de-coalescing) so that the graph becomes colorable. We (almost) completely classify the NP-completeness of these problems, discussing also on the structure of the interference graph (arbitrary, chordal, or k-colorable in a greedy fashion). We believe that such a study is a necessary step for designing new coalescing strategies.

1 Introduction

Register allocation is one of the most studied problem in compilation. Its goal is to find a way to map the temporary variables used in a program into physical memory locations (either main memory or machine registers). The complexity of register allocation (for a fixed schedule) comes from two main optimizations, *spilling* and *coalescing*. Spilling decides which variables should be stored in memory so as make register assignment possible while minimizing the cost of stores and loads. Register coalescing reduces the cost of moves between registers as much as possible. This report is devoted to the study of coalescing problems.

Classical approaches for register allocation integrate spilling, coalescing, and coloring (the final assignment of variables to registers) in the same framework. This is for example the case in the *iterated register coalescing* approach proposed by Appel and George [19], a modified version of the original allocation scheme of Chaitin [11] and of improvements due to Briggs et al. [6]. The problem is modeled with the *interference graph* (two variables interfere if they cannot share the same register, classically when they are simultaneously live at some program point) and a greedy approach is used to try to color the graph with k colors (when k registers are

available), with a combination of the following mechanisms: a) a vertex/variable with at most $(k-1)$ neighbors can be removed from the graph since it will be easy to color afterwards; b) removing a move instruction (coalescing) can be done by merging the two vertices involved in the move; such a coalescing is usually performed in a *conservative* way, i.e., with simple rules to guarantee that the graph remains k-colorable; c) when all vertices have at least k neighbors, some vertex is removed as a potential spill. The vertices are colored in the order opposite to their removal. Each vertex is given a color not used by its already-colored neighbors; if no color is available, an actual spill is performed, i.e., loads and stores are inserted. In this case, the interference graph is rebuilt and the coloring procedure is restarted.

Such an approach gives fairly good results. But the main reason for its success is certainly its simplicity both from a conceptual and an implementation point of view. Weights can be easily added to moves and to vertices to take into account different dynamic execution frequencies of basic blocks. Physical registers can be added as specific vertices. Some smarter coloring schemes favoring more coalescing, such as biased coloring, can be used. However, this approach has also several weaknesses both for spilling and coalescing. For spilling, once a vertex is removed as an actual spill, no clearly-specified approach (except a "spill-everywhere" approach) is available to decide where to place loads and stores. Even worse, it can happen that some spilling is done even if this actually does not help to make the graph k-colorable. For coalescing, although simple and appealing, conservative coalescing is sometimes not aggressive enough and too many moves may remain in the code. Finally, even if "splitting" (adding register-to-register moves) is sometimes considered in such a framework, it is very hard to control the interplay between spilling and splitting/coalescing. But, with the increasing cost of memory accesses on most architectures, it is important today to find mechanisms to hide memory latencies and, in particular, to find heuristics that spill less, possibly at the price of additional register-to-register moves.

Several variants have been proposed to enable more coalescing in the previous framework. *Aggressive coalescing* [5] merges move-related vertices, regardless of the k-colorability of the graph after the merge. To make the graph k-colorable afterwards, one can then "de-coalesce" some of the previously-merged vertices until the graph becomes easy to color with k colors. Such an approach is called *optimistic coalescing* [27, 28]. One can also merge vertices even if they are not related to a move because this can sometimes make a non k-colorable graph k-colorable [35, 34].

New coalescing problems have also appeared due to recent developments on the (strict) SSA form (static single assignment form) [15]. Today, most compilers go through this intermediate code representation, which makes many code optimizations simpler. In strict SSA, each variable is defined textually only once and is alive only along the dominance tree associated to the control-flow graph. Some so-called ϕ functions are used to transfer values along the control flow not covered by the dominance tree. These ϕ functions are not machine code and an out-of-SSA phase is necessary, which typically introduces many register-to-register moves. Several techniques are available to go out of SSA [15, 7, 24, 33, 9, 32], some with the objective of reducing the number of moves. This problem is a form of aggressive coalescing as no register constraint is taken into account in this phase, which is done *before* register allocation. With a particular interpretation of ϕ functions, this is an aggressive coalescing problem but on special graphs (interference graphs of SSA programs are chordal).

Our experiments with classical out-of-SSA approaches revealed many bad situations where a too aggressive coalescing can increase the number of spills in the subsequent register allocation phase (unless some splitting is done to undo the coalescing, but this is difficult to control). In other words, going out of SSA should be done at the same time as register allocation, or at least with a constraint on the number of registers. On the other hand, a standard conservative coalescing approach is not enough to coalesce most copies that arise in the out-of-SSA phase. Therefore, such coalescing problems need to be revisited too.

Finally, the fact that the interference graph of a strict SSA code is chordal, therefore easy to color, has also led to the developments of new heuristics for register allocation, based on two separate phases, one for spilling and one for coalescing. The first phase of spilling decides which values are spilled and where, so as to get to a code with Maxlive $\leq k$ where Maxlive is the maximal number of variables simultaneously live [1]. The second phase of coloring (called register assignment in [25]) maps variables to registers, possibly removing (i.e., coalescing) move instructions (also called shuffle code in [26]), but with no additional spill. This is the approach advocated by Appel and George [2] and, more recently, in [8, 3, 22]. The coalescing phase of such an approach seems *a priori* simpler than for Chaitin-like register allocators because the initial graph is already k-colorable (it can even be chordal); one just wants to coalesce as many moves as possible so that the graph remains k-colorable (or remains easy to color with k colors). However, the fact that the first phase of spilling can be much more aggressive (it spills just the necessary variables, the code may have a very high register pressure, possibly equal to Maxlive at many program points) makes the coalescing much more difficult and standard conservative coalescing approaches are not enough. This has led Appel and George to define a "coalescing challenge" (`http://www.cs.princeton.edu/~appel/coalesce`).

We believe that these new developments and variants of the coalescing problem motivate the need for a better study of its complexity, which has not been addressed in details so far. In this report, we distinguish the different coalescing optimizations previously mentioned: a) *aggressive coalescing* removes as many moves as possible, regardless of the colorability of the resulting interference graph; b) *conservative coalescing* removes as many moves as possible while keeping the colorability of the graph; c) *incremental conservative coalescing* removes one particular move while keeping the colorability of the graph; d) *optimistic coalescing* coalesces all moves (when possible), aggressively, and gives up about as few moves as possible (de-coalescing) so that the graph becomes colorable. We (almost) completely classify the complexity of these problems, considering also the structure of the interference graph (arbitrary, chordal, or k-colorable greedily). We view such a study as a necessary step for designing new coalescing strategies.

2 Definitions and general properties

Before analyzing the complexity of the different coalescing problems that arise in register allocation heuristics, we need to introduce a few definitions and properties.

[1] How to color with k colors a code with Maxlive $\leq k$ is more subtle that this quick explanation. See the discussions in [22, 30, 4] for more details on the interplay of critical edges, register swaps, color permutations, etc.

2.1 Interference graph, affinities, and coalescing

The *live-range* of a variable is the union of all the intervals of the control-flow graph that link the different definitions of this particular variable to their uses. A variable is said *live* at a program point if this point belongs to its live-range. Two variables *interfere* if they cannot be stored in the same register. In general, to define the notion of interference, one assumes that different variables can possibly have different values (i.e., there is no analysis of values) and that, for each use of a variable, there is a definition of this variable on any control path from the start of the program to this use (strict program). Then, two variables interfere iff (if and only if) their live-ranges intersect. Chaitin et al. [10] relaxed the previous interference condition by defining that two variables interfere iff the live-range of one contains a definition of the other one. For a strict program, the two definitions are equivalent. Maxlive is defined as the maximum number of variables simultaneously live at a program point, among all program points of the control flow graph. For a strict program, Maxlive is a lower bound on the number of registers required to store all variables of the program.

The *interference graph* $G = (V, E)$ is an undirected graph where each vertex $v \in V$ corresponds to a variable of the program. There is an *interference* $(u, v) \in E$ iff u and v interfere. Coloring the interference graph means assigning a color to each vertex so that vertices connected by an edge have different colors. This color is then interpreted as a register name. Notice that, in the interference graph model, each variable/name is considered as an atomic object, i.e., it will be placed in the same register all along its live-range. In other words, borrowing the subtle title of Cytron and Ferrante's paper [14], "what's in a name" has already been decided and no more live-range splitting [12] will be done.

In addition to interferences (usually represented as solid lines), each copy instruction between two variables u and v is represented by an *affinity* (u, v) (usually represented as a dotted line). Assigning u and v to the same register will save one register-to-register move. Affinities can be weighted to represent the dynamic execution count of the copy instruction. In this model, the goal is to remove copy instructions but not to decide where they should be placed (again, a possible live-range splitting has already decided the placement of moves). A *coalescing* of $G = (V, E)$ with affinities A is a function f such that $f(u) \neq f(v)$ whenever $(u, v) \in E$; an affinity $(u, v) \in A$ is *coalesced* if $f(u) = f(v)$. A coalescing can be viewed as a coloring, with no constraint on the number of colors. The *coalesced graph* $G_f = (V_f, E_f)$ is the graph obtained from G by merging all vertices with same value by f. More formally, if f takes n values, f defines a partition of V into n subsets $(S_i)_{1 \leq i \leq n}$ where u and v are in the same subset iff $f(u) = f(v)$. The vertices in V_f are the subsets $(S_i)_{1 \leq i \leq n}$ and there is an edge $(S_i, S_j) \in E_f$ iff $(u, v) \in E$ for some $u \in S_i$ and $v \in S_j$. The fact that f is a coloring guarantees that G_f has no loop (S_i, S_i).

2.2 Graph structures

Depending on the structure of the control-flow graph from which the interference graph is extracted and depending on the coloring heuristics used to color the interference graph, we need to distinguish different graph structures.

Perfect graphs [20] are graphs with some interesting properties for register allocation. In

particular, they can be colored in polynomial time, which suggests that we can design heuristics for spilling or coalescing in order to change the interference graph into a perfect graph. For a graph G, the maximal size of a clique (complete subgraph) is the *clique number* $\omega(G)$. The minimum number of colors needed to color G is the *chromatic number* $\chi(G)$. Of course, $\omega(G) \leq \chi(G)$ because vertices of a clique must have different colors. A graph G is perfect if each induced subgraph G' of G (including G itself) is such that $\chi(G') = \omega(G')$. Interval graphs, path graphs, or chordal graphs are perfect [20].

A graph G is *chordal* iff any cycle of length at least 4 has a chord. Chordal graphs are of particular interest for code optimization and, especially, register allocation [8, 3, 29, 22] due to the following result we recall for completeness.

Theorem 1 *Ignoring ϕ functions, the interference graph G of a strict SSA program is chordal and $\omega(G) = Maxlive$.*

Proof. If one makes the connection between SSA form and graph theory, this property can be explained briefly as follows. The interference graph of an SSA program is the intersection graph of a family of subtrees (the live-ranges) of a tree (the dominance tree), which is another characterization of chordal graphs [20, Thm. 4.8].

One can also give a direct proof using strict SSA properties [22, 3], by noticing that, on the dominance tree, if two program points p and q dominate a program point r, then either p dominates q or the converse [9]. If two variables and interfere, their definitions dominate some program point where they are both live, thus the definition of one dominates the definition of the other. Now, consider a cycle of length at least 4 in G. One can orient each edge (u, v) of this cycle from u to v if the definition of u dominates the definition of v. Since the dominance relation is a partial order, this cannot form a directed cycle, thus there are two edges (u, v) and (v, w), oriented from u to v and from w to v, i.e., the definitions of u and of w dominate the definition of v. This proves that u and w are both live at the definition of v, thus they interfere, which makes a chord.

To show that $\omega(G) = \text{Maxlive}$, consider a clique in G and orient its edges as above. There is a vertex u in the clique such that, for any other vertex v in the clique, (u, v) is oriented from v to u, in other words, a variable whose definition is dominated by the definition of any other vertex. Thus all variables in the clique are live at the definition of u, which proves $\omega(G) \leq \text{Maxlive}$. Finally $\text{Maxlive} \leq \omega(G)$ since variables simultaneously live form a clique. ■

The fact that $\chi(G) = \omega(G) = \text{Maxlive}$ for an SSA program shows that, in terms of coloring, there is no need to do additional live-range splitting in SSA. This does not help to avoid spilling (but, it has an effect on coalescing).

Another interesting class of graphs for register allocation is what we call *greedy-k-colorable* graphs. The greedy-k-colorability is defined as follows. While this is possible, remove a vertex of degree $< k$ (in the current graph). A graph is greedy-k-colorable iff this elimination scheme removes all vertices. This definition seems non-deterministic but it is easy to see that the order in which vertices are removed (when possible) is not important. A greedy-k-colorable graph is k-colorable because we can color its vertices in the opposite order of their removal, assigning to each vertex a color not used by its already-colored neighbors (this is possible because there are at most $(k - 1)$ such neighbors). This is the coloring heuristic used in Chaitin-like approaches.

It is worth pointing out that the smallest k such that G is greedy-k-colorable is the *coloring number* col(G) [23]. Define a *smallest last order* of the vertices of G as follows: let x_i be a vertex of minimum degree in G_i, the subgraph of G obtained after removing $x_1, \ldots, x_{i-1}$. If $\delta(H)$ denotes the minimum degree in a graph H, a classical result [23, Thm. 12] shows that $\text{col}(G) = 1 + \max_i \delta(G_i) = 1 + \max_{G' \subseteq G} \delta(G')$. In other words, G is not greedy-k-colorable iff G has a subgraph G' in which all vertices have degree (in G') at least k.

Property 1 *If G is a k-colorable chordal graph, it is greedy-k-colorable.*

Proof. Any chordal graph G has a simplicial vertex [20], i.e., a vertex v whose neighbors form a clique. If G is k-colorable, it has no clique of size k, thus v has at most $(k-1)$ neighbors. We can thus remove v from the graph. The remaining graph is still chordal and the same argument applies. We can repeat this process to show that the graph is greedy-k-colorable. ■

This basic property, to our knowledge not mentioned in the compiler literature, is particularly interesting for register allocation. It implies that if we do some spilling and live-range splitting to reduce Maxlive to k and get a chordal k-colorable interference graph, we can still reuse the same framework as Chaitin for the last coloring/coalescing phase.

In the next sections, we show several NP-completeness results, for a fixed k, for example $k = 3$, which is stronger than assuming that k is an input of the problem. However, one could wonder if the problem remains NP-complete for another fixed $k' \geq k$. The following property (with $p = k' - k$) will extend our NP-completeness results from k to k'.

Property 2 *Let G be a graph. Define G' by adding to G a clique of p new vertices and an edge between any vertex of the clique and any vertex of G. Then G is k-colorable iff G' is $(k+p)$-colorable, G is chordal iff G' is chordal, and G is greedy-k-colorable iff G' is greedy-$(k+p)$-colorable.*

Proof. The first property is obvious for, by construction, the additional clique must use p other colors. For the second property, if G' is chordal, G is also chordal as a subgraph of G'. Conversely, if G is chordal, consider a cycle of G' of length at least 4. If it is a cycle of G, it has a chord. Otherwise, it has a vertex v in the clique and two edges (v,u) and (u,w) with $w \neq v$. Since v is connected to any other vertex in G', (v,w) is a chord. For the third property, suppose that G is greedy-k-colorable, i.e., vertices can be removed in some order, with degree $< k$ in the remaining graph. In G', first remove the vertices of G in the same order, they have degree at most $(k-1+p)$. Then one can remove the vertices of the clique, with degree $< p$, and thus G' is greedy-$(p+k)$-colorable. Finally, if G is not greedy-k-colorable, it has a subgraph H such that all vertices have degree (in H) at least k. Adding the clique of size p to H shows that G' is not greedy-$(p+k)$-colorable. ■

3 Complexity of aggressive coalescing

The *aggressive coalescing* problem is to remove as many move instructions as possible, with no constraint on the number of registers. Only interferences can prevent coalescing. It can be formulated as follows:

Problem: AGGRESSIVE COALESCING
Instance Graph $G = (V, E)$, affinities $A \subseteq V^2$, integer K.
Question Is there a coalescing of G, i.e., a function f with $f(u) \neq f(v)$ whenever $(u, v) \in E$, such that at most K affinities $(u, v) \in A$ are not coalesced, i.e., satisfy $f(u) \neq f(v)$?

We will use a reduction from *multiway-cut* [16]. Other proofs related to aggressive coalescing and out-of-SSA translation are available in [31, 21].

Problem: MULTIWAY-CUT
Instance Graph $G = (V, E)$, set $S = \{s_1, \ldots, s_k\} \subseteq V$ of k specified vertices or *terminals*, integer K.
Question Can we remove at most K edges from E so that each terminal is in a different connected component?

In the general multiway-cut problem, edges are weighted but it is NP-complete even for the version above where all edges have equal weight, and even for $k = 3$.

Theorem 2 *The aggressive coalescing problem is NP-complete even if there are only* 3 *interferences.*

Proof. The reduction is as follows. Let $G = (V, E)$, S, K, be an instance of multiway-cut. First define $G' = (V', E')$ from G by replacing each edge $e = (u, v) \in E$ by two edges (u, x_e) and (x_e, v) where x_e is a new vertex. Clearly, G' is a positive instance of multiway-cut iff so does G since at most one of the two (u, x_e) and (x_e, v) may need to be removed.

We define the interference graph $G'' = (V', F)$ such that (S, F) is a clique (a triangle if $k = 3$). We interpret each edge in E' as an affinity to be coalesced, i.e., $A = E'$. Then (G', S, K) is a positive instance of multiway-cut iff (G'', A, K) is a positive instance of aggressive coalescing. Indeed, each connected component can be colored with a single color: edges removed from G' correspond to affinities not coalesced in G''. See Figure 1 for an example.

It remains to show that we can indeed build a code with an interference graph such as $G'' = (V', F)$ and affinities E'. We do as follows. For each vertex $v \in V \setminus S$, a basic block B_v defines a variable v. A basic block B defines all variables $s_i \in S$ together. For each edge $e = (u, v) \in E$, a basic block C_e uses a variable x_e, which is defined by a move instruction $x_e = u$ (resp. $x_e = v$) in a basic block predecessor of C_e and successor of the block where u (resp. v) is defined (this corresponds to the introduction of vertex x_e in G'). Figure 1 illustrates this reduction. ■

As mentioned in Section 1, going out of SSA while minimizing the number of moves is a form of aggressive coalescing. From a complexity point of view, Theorem 2 shows that aggressive coalescing is difficult even if the interference graph is very simple (but not the affinities), in particular even if it is chordal or greedy-k-colorable. These properties do not make the problem simpler. From a practical point of view, aggressive coalescing can degrade register allocation. Indeed, coalescing means fusing live-ranges and merging, in the interference graph G, the corresponding vertices. After these merges, the coalesced graph G_f may not be k-colorable. In this case, three alternatives are available:

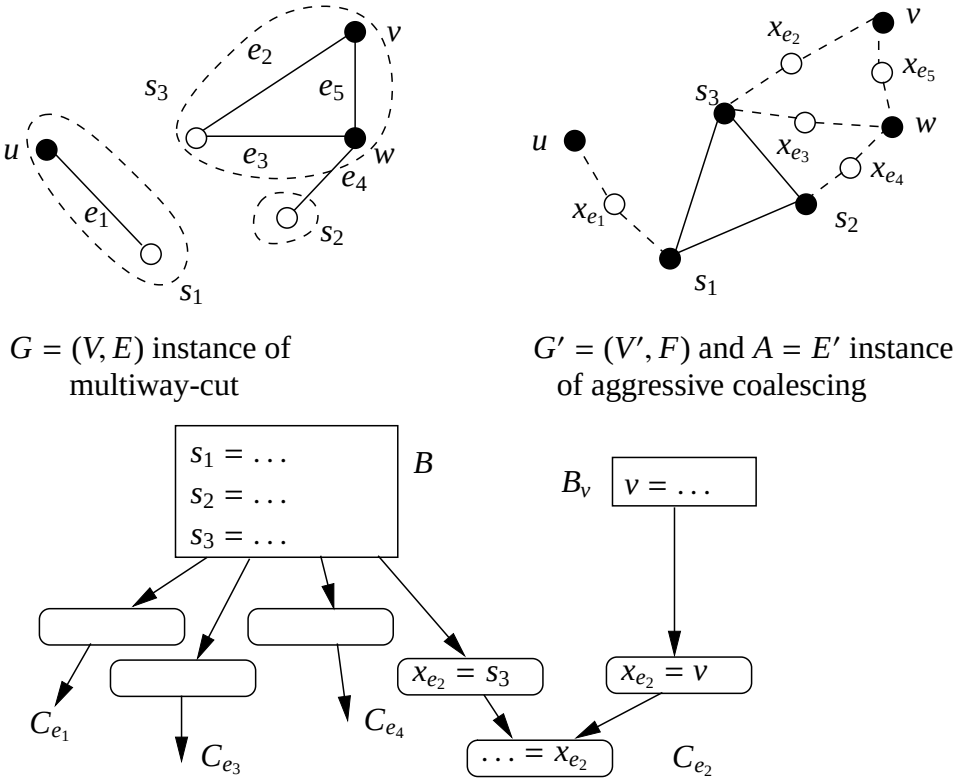


Figure 1: Aggressive coalescing: reduction.

- One can remove some vertices from the graph and spill the corresponding variables; this is the strategy proposed in Chaitin's register allocator [11].
- One can give up about some coalesced moves so that the graph gets greedy-k-colorable again; this is optimistic coalescing [27] that we analyze in Section 5;
- One can prefer to not use aggressive coalescing but to coalesce moves only if the graph is proved to remain greedy-k-colorable; this is conservative coalescing, introduced in [5], a technique we analyze in Section 4.

4 Complexity of conservative coalescing

The *conservative coalescing* problem (for a k-colorable graph) is to coalesce as many moves as possible so that, after this coalescing, the interference graph is still k-colorable.

Problem: CONSERVATIVE COALESCING
Instance Graph $G = (V, E)$, affinities A, integers K and k.
Question Is there a coalescing f of G such that the coalesced graph G_f is k-colorable and at most K affinities are not coalesced?

Another possible formulation [2] is to ask directly for a coalescing f that is a k-coloring of G. We prefer the first formulation: it is closer in spirit to what heuristics do and it allows us to discuss more precisely the complexity of the problem in terms of the structure of G and G_f. Indeed, the problem may seems simpler in practice, if we start from some graph G with a particular structure, if we merge only vertices connected by an affinity, or if we target a graph G_f not only k-colorable, but also greedy-k-colorable.

Theorem 3 *Conservative coalescing is NP-complete, even for $k = 3$, even if G is greedy-2-colorable, even if one asks G_f to be also chordal or greedy-3-colorable, and even if only vertices connected by affinities can be merged.*

Proof. As noticed in [2], a simple reduction from the well-known graph k-colorability [17, Problem GT4] shows that, even for $K = 0$, conservative coalescing is NP-complete. Given a graph $G = (V, E)$, define an instance of conservative coalescing as follows. Define an interference graph with the vertices V and a set of disjoint edges (x_e, y_e) (new vertices), one for each edge $e = (u, v) \in E$, and the affinities (u, x_e) and (y_e, v). See Figure 2 for an illustration. All moves can be aggressively coalesced and the coalesced graph is G. In other words, we just defined an instance of conservative coalescing that is positive for $K = 0$ iff G is k-colorable.

What if we add conditions on the structures of G and G_f? And if we can only merge vertices connected by affinities? The problem remains NP-complete even if G is chordal or greedy-k-colorable because the interference graph used in the previous reduction is greedy-2-colorable (disjoint edges). If we ask G_f to be not only k-colorable, but also greedy-k-colorable (resp. chordal), the problem is still NP-complete (but not for a fixed K because checking that a graph is chordal or greedy-k-colorable is polynomial). To see this, add two affinities $(u, x_{u,v})$ and $(v, x_{u,v})$ ($x_{u,v}$ is a new vertex) for any two vertices u and v of the original graph G. Then, an optimal conservative coalescing will lead to a clique and it can be obtained by merging only vertices connected by affinities. We conclude by noticing that a clique of size k is both chordal and greedy-k-colorable. ■

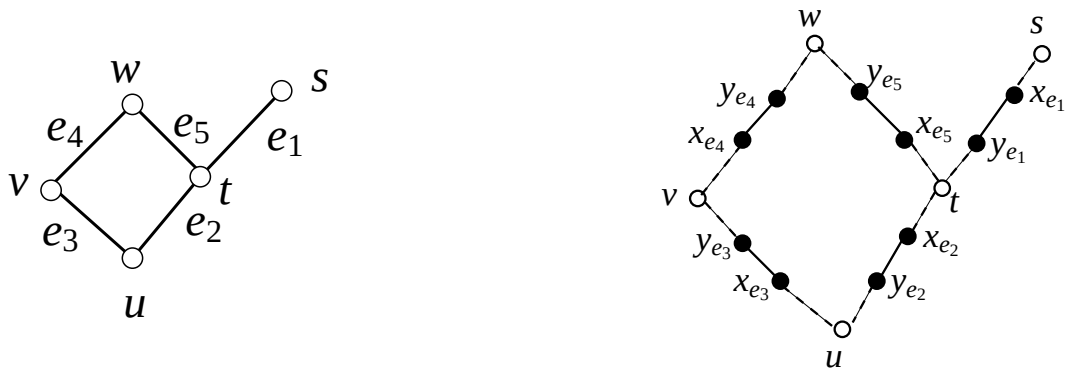


Figure 2: Reduction for Thm. 3 (first part).

We point out that we could have used the proof of Theorem 2 to show most of Theorem 3. Indeed, the graph used in Theorem 2 is a triangle (and some isolated vertices), thus it keeps such

a structure after any coalescing, thus it is chordal and greedy-3-colorable. We gave the reduction from graph k-colorability, maybe more natural, to show how to extend the remark of [2] into a more accurate complexity result.

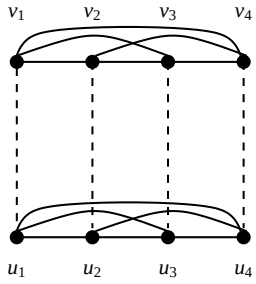
In practice, conservative coalescing heuristics do not consider the affinities all together, but one by one, according to some priority (e.g., moves in inner loops are coalesced first, if possible). This is what we call *incremental conservative coalescing*. Two incremental conservative tests are used [6, 19], referred as as Briggs' and George's tests in [1].

Briggs Two vertices u and v can be merged if the resulting vertex has at most $(k-1)$ neighbors of degree at least k.

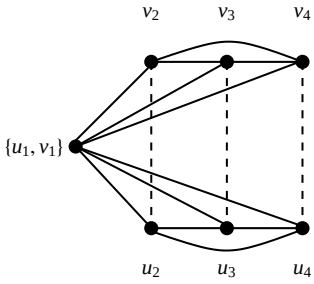
George Two vertices u and v can be merged if all neighbors of u with degree at least k are also neighbors of v.

These rules guarantee that the greedy-k-colorability of the graph does not change. Indeed, consider the elimination process that defines greedy-k-colorability. A vertex merged by Briggs' rule can always be removed from the graph once its neighbors of degree $< k$ are removed, thus such a coalescing is always safe. The situation is slightly different for George's rule: once the neighbors of degree $< k$ are removed, we end up with a subgraph of the original graph, thus not harder to color. But if v cannot be removed from the original graph (potential spill), the same is true for the merged vertex (possibly with a larger cost to spill the two merged live-ranges). Thus, if George's rule is used in a Chaitin-like allocator (where spilling and coalescing are done in the same framework), the interaction with spilling is unclear. This is the reason why George's rule is used in [19] only to merge a vertex u with a *precolored* vertex v (machine register) because such a vertex never leads to a spill. We point out however that, if we do spilling first as in [2, 3, 22] to get a greedy-k-colorable interference graph, then George's rule can be used (in addition to Briggs' rule) for any two vertices (possibly with two tests as the rule is asymmetric), resulting in more coalesced moves. The same applies for the last phase of Chaitin-like approaches, i.e., when no spill is introduced. Our current experiments show that this is indeed useful in practice.

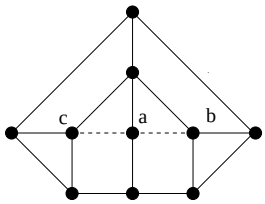
When the register pressure is high, such local rules are not enough, in particular to coalesce parallel copies, when Maxlive is close to the number of registers, as the experiments in [2] show. The problem is that the decision is local and, even worse, done before all vertices of small degree are removed from the graph (if they are related to some affinities, they are kept in the graph). Figure 3 (on the left) shows a permutation of 4 values. Assume $k = 6$; coalescing the 4 moves may lead to a greedy-6-colorable graph, but if we coalesce them one at a time and use a local rule to decide, the merged vertex has degree 6 (Figure 3 in the middle); if its neighbors have degree 6 (due to other vertices not shown), a local rule will decide to not coalesce. Another reason is due to the incremental nature of these rules. If G is a greedy-k-colorable graph and S a set of affinites such that G remains greedy-k-colorable when all affinities of S are coalesced simultaneously, it may happen that coalescing any affinity in S leads to a graph that is not greedy-k-colorable. This is illustrated in Figure 3 (in the right). The graph remains greedy-k-colorable the two affinities (a, b) and (a, c) are coalesced, but not if only one is coalesced. To get a sequence of coalescing that is conservative at each step, one would need to consider affinities "obtained by transitivity" (such as the pair (b, c) in Figure 3).



Affinities and interferences for a permutation of size 4.



If (u_1, v_1) is coalesced, the degree increases to 6.



Remains greedy-3-colorable if (a, b) and (a, c) are coalesced, but not if only one of them is coalesced.

Figure 3: Local rules are not enough.

One can try to improve these local conservative tests. As mentioned in [19], George's rule can be extended by considering that only the neighbors of u, with at most $(k-1)$ neighbors of degree $\geq k$, need to be neighbors of v. But this is more costly to implement. More generally, one can simply use brute force and coalesce the moves aggressively (i.e., merge the corresponding vertices if interferences allow it) and check, in linear time, whether the resulting graph is greedy-k-colorable or not. This is useful to coalesce a bit more aggressively. The same is true for a given set of moves. One can merge all corresponding vertices and check if the graph is greedy-k-colorable.

If G is k-colorable with a k-coloring f such that $f(x) = f(y)$, then there is of course a set of pairs of vertices, including the pair (x, y), that, once merged, lead to a greedy-k-colorable coalesced graph (simply merge all vertices with same color to get a k-clique). But which vertices should be merged? The previous heuristics do not answer this problem. This raises the question of the complexity of incremental conservative coalescing, which is the conservative coalescing problem for a *single* affinity.

Problem: INCREMENTAL CONSERVATIVE COALESCING
Instance Graph $G = (V, E)$, one affinity (x, y), integer k.
Question Can we coalesce (x, y) to get a k-colorable graph, i.e., is there a k-coloring f of G such that $f(x) = f(y)$?

We now show that this problem is NP-complete if G can be any k-colorable graph (Theorem 4), i.e., knowing that G is k-colorable does not help to decide if it remains k-colorable after a single coalescing! However, the problem is polynomial if G is k-colorable and chordal (Theorem 5). The complexity of the very interesting intermediate case, i.e., when G is greedy-k-colorable, is left open.

Theorem 4 *Incremental conservative coalescing is NP-complete if G can be any k-colorable graph, even for $k = 3$.*

Proof. We use a reduction similar to the proof of graph 3-colorability, i.e., with a reduction from the well-known 3SAT [17, Problem LO1]. However, here, we will make a small detour through 4SAT. We first show how to build, from an instance of 4SAT, a graph G that is 3-colorable iff there is an truth assignment for the 4SAT formula.

Consider an instance of 4SAT, i.e., a set U of n variables $x_1, \ldots, x_n$, and a set C of m clauses $c_1, \ldots, c_m$, each with 4 literals $y_{i,1}, \ldots, y_{i,4}$ (each $y_{i,j}$ is a x_k or its negation). We build a graph $G = (V, E)$ as follows. It has three vertices T (for true), F (for false), and R that form a triangle. For each variable $x_i \in U$, there are two vertices, denoted x_i and $\overline{x_i}$, which form a triangle with R (with 3 colors, this will force x_i and $\overline{x_i}$ to have the colors of T and F, or the converse). For each clause $c_i \in C$, there are four vertices $a_{i,j}$, two vertices $b_{i,j}$, and two vertices $c_{i,j}$, connected as depicted in Figure 4. As for the original proof of graph k-coloring (see [13, Page 962]), it is easy to see that G is 3-colorable iff there is a truth assignment for the clauses. Indeed, if G is 3-colorable, then the four literals $y_{i,j}$ cannot all be colored as F, otherwise the two $b_{i,j}$ must be colored as F, and the two $c_{i,j}$ cannot be colored. Thus interpreting the colors of each x_i gives a truth assignment. Conversely, if there is a truth assignment, color each x_i as T iff it is true in the 4SAT formula. Then, color $b_{i,1}$ as T (resp. F) if $y_{i,1}$ or $y_{i,2}$ is true (resp. both are false), the same for $b_{i,2}$. The rest of the 3-coloring follows.

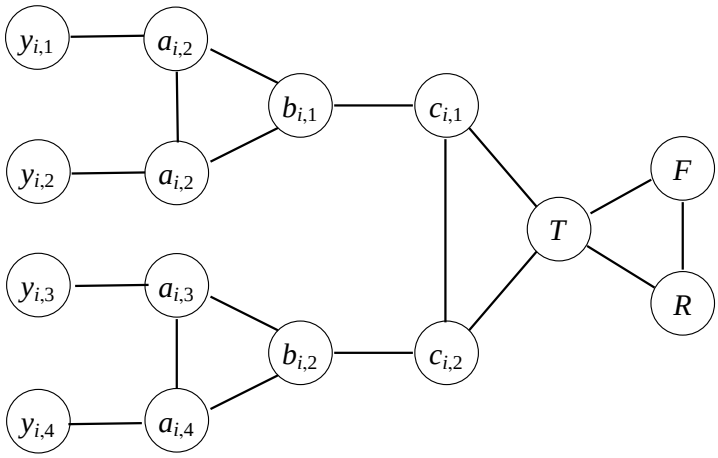


Figure 4: Reduction for Thm. 4.

Now consider an instance (U, C) of 3SAT. Add a new variable x_0 and define an instance (U', C') of 4SAT, where $U' = U \cup \{x_0\}$ and each clause $c'_i \in C'$ is defined from $c_i \in C$, $c_i = y_{i,1} \wedge y_{i,2} \wedge y_{i,3}$, by $c'_i = y_{i,1} \wedge y_{i,2} \wedge y_{i,3} \wedge x_0$. Notice that there is a truth assignment for C' (set x_0 to true). Moreover, there is a truth assignment for C iff there is a truth assignment for C' for which x_0 is false. Finally, we define a graph G from C' as before and we consider the affinity (x_0, F). From the previous study, G is 3-colorable. Furthermore there is a 3-coloring of G such that the vertices x_0 and F have the same color (coalescing) iff there is truth assignment for C' for which x_0 is false. ■

Theorem 5 *Incremental conservative coalescing can be solved in polynomial time if G is chordal.*

Proof. Let $G = (V, E)$ be a chordal graph and (x, y) be the affinity to coalesce. A fundamental property [20, Thm. 4.8] is that G can be represented as the intersection graph of a family of subtrees $(T_v)_{v \in V}$ of a tree T. We use the word nodes for the vertices of T to distinguish them with the vertices of G. The nodes of T are the maximal (for inclusion) cliques of G, each vertex $v \in V$

corresponds to a subtree T_v, and $(u, v) \in E$ iff T_u and T_v intersect. A chordal graph with the tree representation can be easily colored with $k \geq \omega(G)$ colors, starting from any node n of T. Orient the tree T so that n is the root and color the subtrees that contain n. Then, go down the branches of the tree and, at each new node, color the subtrees that start at this node with the available colors. This coloring is always possible because, at each node, at most $w(G)$ subtrees intersect. Furthermore, there is no cycle in T so no coloring decision can lead to a conflict.

Now the question is: Can we color G with k colors so that x and y have the same color? We can answer this question in polynomial time as follows. We assume that T_x and T_y do not intersect and $k \geq \omega(G)$, otherwise the answer is no. Consider a path $P = (n_1, \ldots, n_k)$ of T from $n_1 \in T_x$ to $n_k \in T_y$ (this path is unique if n_1 and n_k are the only nodes of P in T_x or T_y). The intersection of the subtrees $(T_v)_{v \in V}$ with P are *intervals* $(I_v)_{v \in V}$. Add new short intervals containing a single node so that all nodes of P are contained in exactly $\omega(G)$ intervals. We claim that T_x and T_y can have the same color iff there is a set of disjoint intervals, including I_x and I_y, that cover all nodes in P. Indeed, if G has a k-coloring such that x and y have the same color, then one can cover P with the intervals I_v such that v is colored as x and y (and possibly some short intervals if not all nodes are covered). Conversely, if such intervals exist, one can merge all corresponding subtrees to get the representation of a new chordal graph G' with $w(G') = w(G) \leq k$; it can thus be colored with k colors and this coloring corresponds in G to a k-coloring such that x and y have the same color.

It remains to show how to find such a set of intervals in polynomial time. This can be done as follows: represent the intervals horizontally on $w(G)$ lines (the lines are full because of the short intervals we added). There is a cover of P with disjoint intervals, including I_x and I_y, iff there is a path from the line of I_x to the line of I_y, following intervals and possibly changing line only from the end of an interval to the beginning of another (i.e., contiguous intervals). This can be checked in $O(V\omega(G)) = O(V^2)$ by a simple marking process from left to right. See Figure 5 for an illustration (dotted lines represent the possible changes of line). ■

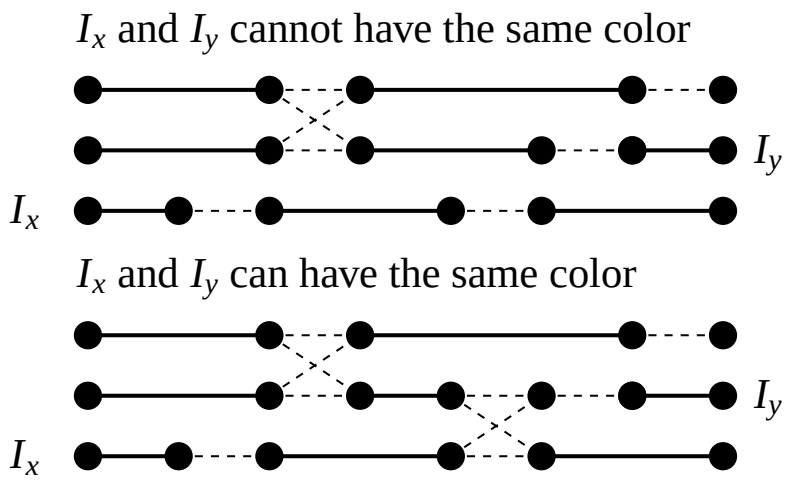


Figure 5: Covering by intervals for Thm. 5.

Theorem 5 shows that we could design an incremental conservative coalescing strategy for chordal graphs. If G is chordal and (x, y) is an affinity that we absolutely want to coalesce because

the corresponding move is expensive, we can decide if this is possible. The problem is that, then, if we coalesce the affinity, the graph may not be chordal anymore. However, we can still make it chordal by an appropriate merge of vertices (as we do in the proof of the theorem). However, these artificial merges may prevent to coalesce more important affinities afterwards. A better strategy would be to stay in the class of greedy-k-colorable graphs (which is larger than the class of chordal graphs). Unfortunately, we do not know the complexity of this problem.

5 Complexity of optimistic coalescing

If G is greedy-k-colorable, coalescing as many moves as possible to that the coalesced graph is k-colorable (or even greedy-k-colorable) is NP-complete (Theorem 3). To approximate this problem, incremental conservative coalescing coalesces moves, one by one, so that the graph remains greedy-k-colorable but, of course, with no guarantee that the chosen moves are the right ones. Park and Moon [27, 28] proposed a "dual" approach, *optimistic coalescing*. A first phase of aggressive coalescing coalesces moves regardless of the k-colorability of the graph. Then, a second phase gives up about some moves, i.e., "de-coalesce" some moves, so that the graph becomes greedy-k-colorable.

If most moves can be coalesced, this approach can be more efficient than using too-conservative local rules such as Briggs' or George's rules. However, in practice, it is not clear which moves should be coalesced aggressively in the first phase (remember that aggressive coalescing is NP-complete by Theorem 2). Moreover, even if all moves can be aggressively coalesced, it is not clear which moves should be de-coalesced in the second phase. The goal of this section is to address the complexity of this problem. If one requires the de-coalesced graph to be just k-colorable, it is of course NP-complete as the first part of the proof of Theorem 3 shows: after all affinities are coalesced, it is hard to decide if the resulting graph is k-colorable or not. In practice, the graph should be easy to color, for example greedy-k-colorable. So, the interesting instance of optimistic coalescing can be formulated as follows.

Problem: OPTIMISTIC COALESCING
Instance Graph $G = (V, E)$ greedy-k-colorable, affinities A that can all be coalesced (i.e., there is a coalescing f of G such that for all $(u, v) \in A$, $f(u) = f(v)$), integers k and K.
Question Is there a de-coalescing of G_f (i.e., a coalescing g of G such that $g(u) = g(v)$ implies $f(u) = f(v)$), such that at most K affinities (u, v) are not coalesced (i.e., satisfy $g(u) \neq g(v)$) and such that G_g is greedy-k-colorable?

Theorem 6 *The optimistic coalescing problem is NP-complete, even for $k = 4$, and even if G is chordal.*

Proof. The proof is by reduction from the well-known vertex cover [17, Problem GT1], which is NP-complete even if all vertices have degree at most three [18].

Let $G = (V, E)$ be a graph such that all vertices have degree at most 3. We build an instance of optimistic coalescing as follows. For each node $v \in V$, we create a structure as shown on the

left of Figure 6. Each hexagon in this structure is the widget shown on the lower right part of the figure. The central vertex A is in fact two vertices A and A' linked by an affinity (upper right part of the figure). On this structure, each vertex v_i, $1 \leq i \leq 3$, can be used to connect v to one of its neighbors. Since v has at most three neighbors, we can transform the whole graph G into this format. We get a new graph H. H is not chordal, but we will show later how to make it chordal. Our first goal is to de-coalesce some of the pairs (A, A') so that H is greedy-4-colorable.

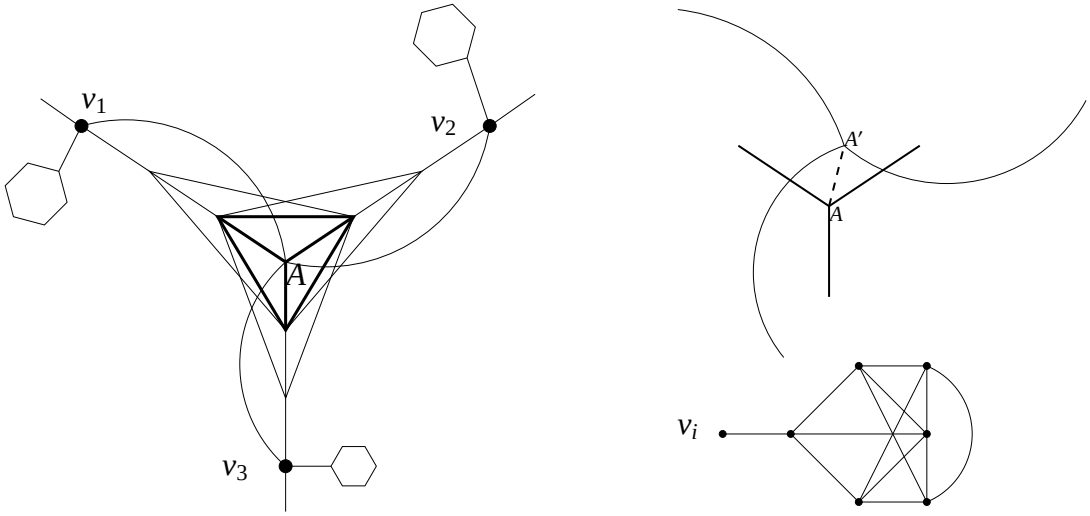


Figure 6: Vertex structure and ad-hoc widget.

The important point is to understand how the greedy 4-coloring algorithm can "eat" a structure. It can only works if there is at least one node of degree < 4. All the vertices of the hexagonal widgets have degree ≥ 4 so the structure cannot be attacked by these. If the structure for $v \in V$ has no neighbor (either because v has no neighbor in G or because the neighbor structures have already been eaten), then each v_i has degree 3: they can be eaten, then the hexagonal widgets and the inner structure can be eliminated too. Finally, notice that the structure cannot be attacked by any two of its branches: if one v_i remains, the inner 4-clique (in bold) cannot be removed. Hence the only remaining possibility is to attack the structure by A and A', if they are de-coalesced. This shows that there are only two ways for the greedy algorithm to attack the structure in H corresponding to a vertex v of G: either after having eaten all the structures corresponding to the neighbors of v, or by de-coalescing A and A' and attacking the structure from the heart.

The previous study shows that H after de-coalescing is greedy-4-colorable iff, for each $(u, v) \in E$, a de-coalescing occurred in one of the structures corresponding to u and v, i.e., iff the set of vertices u such that a de-coalescing occurred in the corresponding structure is a vertex cover for G.

For what we want to prove, H is not enough, we need to build a greedy-4-colorable graph H' (even chordal if possible) and affinities such that all affinities can be aggressively coalesced into H and such that these new affinities will not be chosen to de-coalesce optimally H into a greedy-4-colorable graph. In H, there are three kinds of chordless cycles: in the hexagonal widgets, inside each structure (there is a chordless cycle including (A', v_i, v_j)), and between structures if G

itself is not chordal. These cycles are broken by introducing some affinities as shown in Figure 7. The reduction is still correct because it is always better to choose to de-coalesce an affinity (A, A') instead of any other affinity in the structure because this allows to eat the whole structure with a single de-coalescing.

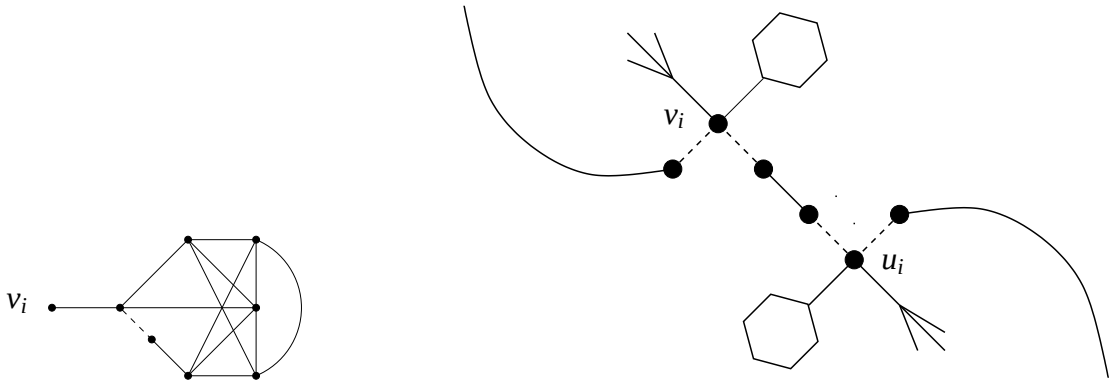


Figure 7: To a get a chordal graph.

To conclude, H' is chordal, greedy-4-colorable, and all affinities can be aggressively coalesced to get H. Furthermore, one can de-coalesce at most K affinities to get a greedy-4-colorable graph iff G has a vertex cover of size at most K. Finally, with Property 2, optimistic coalescing is NP-complete for any fixed $k \geq 4$. ■

6 Conclusion

We addressed the complexity of different variants of register coalescing: aggressive coalescing, conservative coalescing, incremental conservative coalescing, and optimistic coalescing, for three main classes of graphs: k-colorable, greedy-k-colorable, and chordal. Our study shows that most problems are NP-complete, which is certainly not a surprise though never really proved before in such details, and confirms the practical importance of chordal and greedy-k-colorable graphs. We believe that their properties are maybe not yet completely exploited for the design of good conservative coalescing heuristics and good de-coalescing heuristics (the second phase of optimistic coalescing, after an aggressive coalescing phase). We are currently exploring new heuristics based on these properties. Our first experiments on the base of graphs proposed by Appel and George in their "coalescing challenge" shows that there is indeed space for improvements. Our future work will be devoted to the design of such heuristics.

References

[1] A. Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, 1998.

[2] A. W. Appel and L. George. Optimal spilling for CISC machines with few registers. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'01)*, pages 243–253, Snowbird, Utah, United States, June 2001. ACM Press.

[3] F. Bouchez, A. Darte, C. Guillon, and F. Rastello. Register allocation and spill complexity under SSA. Technical Report RR2005-33, LIP, ENS-Lyon, France, Aug. 2005.

[4] F. Bouchez, A. Darte, and F. Rastello. Register allocation: What does Chaitin's NP-completeness proof really prove? Technical Report RR2006-13, LIP, ENS-Lyon, France, Mar. 2006.

[5] P. Briggs. Register allocation via graph coloring. PhD Thesis Rice COMP TR92-183, Department of Computer Science, Rice University, 1992.

[6] P. Briggs, K. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, May 1994.

[7] P. Briggs, K. D. Cooper, T. J. Harvey, and L. T. Simpson. Practical improvements to the construction and destruction of static single assignment form. *Software: Practice and Experience*, 28(8):859–881, 1998.

[8] P. Brisk, F. Dabiri, J. Macbeth, and M. Sarrafzadeh. Polynomial time graph coloring register allocation. In *14th International Workshop on Logic and Synthesis*, June 2005.

[9] Z. Budimlić, K. Cooper, T. Harvey, K. Kennedy, T. Oberg, and S. Reeves. Fast copy coalescing and live range identification. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*, pages 25–32, Berlin, Germany, 2002. ACM Press.

[10] G. Chaitin, M. Auslander, A. Chandra, J. Cocke, M. Hopkins, and P. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, January 1981.

[11] G. J. Chaitin. Register allocation & spilling via graph coloring. In *Proceedings of the 1982 ACM SIGPLAN Symposium on Compiler Construction*, volume 17(6) of *SIGPLAN Notices*, pages 98–105, 1982.

[12] K. D. Cooper and L. T. Simpson. Live range splitting in a graph coloring register allocator. In *Compiler Construction*, volume 1383 of *Lecture Notes in Computer Science*, pages 174–187. Springer Verlag, 1998.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[14] R. Cytron and J. Ferrante. What's in a name? Or the value of renaming for parallelism detection and storage allocation. In *Proceedings of the 1987 International Conference on Parallel Processing*, pages 19–27. IEEE Computer Society Press, Aug. 1987.

[15] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991.

[16] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *24th Annual ACM STOC*, pages 241–251, Victoria, Canada, 1992. ACM Press.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[19] L. George and A. W. Appel. Iterated register coalescing. *ACM Transactions on Programming Languages and Systems*, 18(3):300–324, May 1996.

[20] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[21] S. Hack, D. Grund, and G. Goos. Towards register allocation for programs in SSA-form. Technical Report RR2005-27, September 2005.

[22] S. Hack, D. Grund, and G. Goos. Register allocation for programs in SSA-form. In *Compiler Construction 2006*, volume 3923 of *LNCS*. Springer Verlag, 2006.

[23] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1995.

[24] A. Leung and L. George. Static single assignment form for machine code. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'99)*, pages 204–214. ACM Press, 1999.

[25] V. Liberatore, M. Farach-Colton, and U. Kremer. Evaluation of algorithms for local register allocation. In *8th International Conference on Compiler Construction*, volume 1575 of *Lecture Notes in Computer Science*, pages 137–152, Amsterdam, The Netherlands, Mar. 1999. Springer Verlag.

[26] G.-Y. Lueh, T. Gross, and A.-R. Adl-Tabatabai. Fusion-based register allocation. *ACM Transactions on Programming Languages and Systems*, 22(3):431–470, 2000.

[27] J. Park and S.-M. Moon. Optimistic register coalescing. In *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT'98)*, pages 196–204. IEEE Press, 1998.

[28] J. Park and S.-M. Moon. Optimistic register coalescing. *ACM Transactions on Programming Languages and Systems (ACM TOPLAS)*, 26(4), 2004.

[29] F. M. Q. Pereira and J. Palsberg. Register allocation via coloring of chordal graphs. In *Proceedings of APLAS'05, Asian Symposium on Programming Languages and Systems*, pages 315–329, Tsukuba, Japan, Nov. 2005.

[30] F. M. Q. Pereira and J. Palsberg. Register allocation after classical SSA elimination is NP-complete. In *Proceedings of FOSSACS'06, Foundations of Software Science and Computation Structures*, Vienna, Austria, Mar. 2006.

[31] F. Rastello, F. de Ferrière, and C. Guillon. Optimizing the translation out-of-SSA with renaming constraints. Technical Report RR2005-34, LIP, ENS Lyon, France, august 2005.

[32] F. Rastello, F. de Ferrière, and C. Guillon. Optimizing translation out of SSA using renaming constraints. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO'04)*, pages 265–278. IEEE Computer Society, 2004.

[33] V. C. Sreedhar, R. D. Ju, D. M. Gillies, and V. Santhanam. Translating out of static single assignment form. In A. Cortesi and G. Filé, editors, *Proceedings of the 6th international Symposium on Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 194–210. Springer Verlag, 1999.

[34] S. R. Vegdahl. Using node merging to enhance graph coloring. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation (PLDI'99)*, pages 150–154, New York, NY, USA, 1999. ACM Press.

[35] H. Yang, W. Hu, R. Qiao, and Z. Zhang. Approaches to enhance graph coloring register allocation. In *Proceedings of 1998 International Symposium for Future Software Technology (ISFST'98)*, 1998.