# Aliased Register Allocation for Straight-line Programs is NP-complete

Jonathan K. Lee Jens Palsberg Fernando Magno Quintão Pereira

UCLA, University of California, Los Angeles

**Abstract.** Register allocation is NP-complete in general but can be solved in linear time for straight-line programs where each variable has at most one definition point if the bank of registers is homogeneous. In this paper we study registers which may alias: an aliased register can be used both independently or in combination with an adjacent register. Such registers are found in commonly-used architectures such as x86, the HP PA-RISC, the Sun SPARC processor, and MIPS floating point. In 2004, Smith, Ramsey, and Holloway presented the best algorithm for aliased register allocation so far; their algorithm is based on a heuristic for coloring of general graphs. Most architectures with register aliasing allow only *aligned* registers to be combined: for example, the low-address register must have an even number. Open until now is the question of whether working with restricted classes of programs can improve the complexity of aliased register allocation with alignment restrictions. In this paper we show that aliased register allocation with alignment restrictions for straight-line programs is NP-complete.

## 1 Introduction

**Register Allocation.** Programmers write most software in high-level programming languages such as C, C++, and Java, and use compilers to translate their programs to a growing variety of hardware, including multicore platforms, graphics processing units, and network processors. To achieve high execution speed, programmers rely on compilers to optimize the program structure and to use registers and memory in clever ways. The latter task, known as *register allocation*, has grown steadily in significance because of the widening gap between the short time to access a register and the longer time to access memory. Today, the register allocator may be among the most important and most complicated parts of a compiler. For example, our experiments with the gcc compiler on the StrongARM architecture shows that a good register allocator typically improves execution speed by a factor of 2.5. A register allocator can also be a significant part of the code of a compiler implementation: 10% for lcc [8] and 12% for gcc 2.95.2.

Most programs use more variables than the number of registers on the target computer. The core of the register allocation problem is to determine whether *all* the program variables can be placed in machine registers. The reason why a register allocator may be able to place a high number of variables in a small

number of registers is that some variables are not live at the same time and so they can share a register. When the need for registers exceeds the number of available registers, the register allocator faces the difficult task of choosing which variables will be placed in registers and which variables will be *spilled*, that is, placed in memory. In this paper we focus on the core register allocation problem and do not discuss spilling of variables.

Chaitin et al. [5] showed that the core register allocation problem is NP-complete by a reduction from the graph coloring problem. The essence of Chaitin et al.'s proof is that every graph is the interference graph of some program. Chaitin et al.'s proof assumes a homogeneous bank of registers, where each register can be used to hold the value of any program variable.

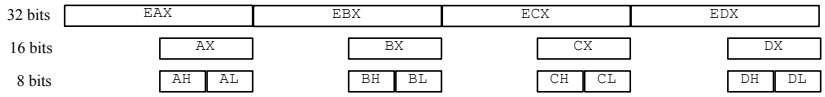


**Fig. 1.** General purpose registers from the x86 architecture

**Aliased Registers.** In this paper we study register allocation for hardware in which the bank of registers is *not* homogeneous. We focus on *aliased registers*: when an assignment to one register name can affect the value of another, such register names *alias* [18]. For example, Figure 1 shows the set of general purpose registers used in the x86 architecture. The x86 architecture has four general purpose 16-bit registers that can also be used as eight 8-bit registers. Each 8-bit register is called a *low* address or a *high* address. The initial bits of a 16-bit register must be *aligned* with the initial bits of a *low*-address 8-bit register. The x86 architecture allows the combination of two 8-bit registers into one 16 bit register. Another example of aliased registers is the combination of two aligned single precision floating-point registers to form one double-precision register. Examples of architectures with such aliased registers include early versions of HP PA-RISC, the Sun SPARC, and the ARM processors. For a different kind of architecture, Scholz and Eckstein [17] describe experiments with the Carmel 20xxDSP Core, which has six 40 bit accumulators that can also be used as six 32-bit registers or as twelve 16-bit aligned registers.

Architectures that allow unaligned pairing exist but are rare. Some models even allow registers wrapping around, that is, the last and the first registers in the bank combine to form one double register. An example of this type of architecture is the ARM VFP coprocessor.

**Aliased Register Allocation.** We will refer to register allocation for hardware with aliased registers as *aliased register allocation*.

Several research groups have proposed solutions to the aliased register allocation problem. Some of the solutions are based on heuristics for graph coloring [18, 3, 4, 15, 16, 14], while others are based on integer linear programming [11, 13,

1, 9, 17, 12] which is flexible enough to describe many architecture irregularities but leads to compile times that are worst-case exponential in the size of the input program.

**Our Results.** We prove that the core aliased register allocation problem with alignment restrictions is NP-complete for straight-line programs where each variable has at most one definition point. A straight-line program is a sequence of instructions without jumps. Our proof consists of three steps, from 3-SAT via a flow problem and then a coloring problem to our register allocation problem. Our coloring problem *without* alignment restrictions is equivalent to the *shipbuilding* problem; Stockmeyer proved that the shipbuilding problem is NP-complete [10, Application 9.1, p.204]. While we can easily reduce the aligned coloring problem to the unaligned coloring problem (and thereby give an alternative proof of Stockmeyer's theorem), we have been unsuccessful in doing a reduction in the opposite direction. The aligned case is more restricted than the unaligned case; yet our result shows that the complexity of aliased register allocation in the aligned case is NP-complete.

Our result and Stockmeyer's result may be surprising because straight-line programs where each variable has at most one definition point are extremely simple. For a homogeneous bank of registers, the core register allocation problem for straight-line programs can be solved in linear time. Our results show that register aliasing is sufficient to bump the complexity to NP-complete.

**Related Work.** At least two other important register allocation problems are NP-complete for straight-line programs: register allocation with precolored registers [2]; and the placement of load and store instructions to transfer values to and from memory [7]. Our proof was inspired in part by a paper of Biro, Hujter, and Tuza [2] who showed how to relate a coloring problem to a flow problem. They used a flow algorithm to solve the precoloring extension problem for interval graphs. Our proof was also inspired by a paper by Even, Itai and Shamir [6] who proved NP-completeness for the multicommodity flow problem.

**Rest of the Paper.** In Section 2 we define our register allocation problem and in Section 3 we define a coloring problem and reduce it to the register allocation problem. In Section 4 we introduce the notion of colored flow for simple graphs, and in Section 5 we reduce the flow problem to the coloring problem. In Section 6 we show how to reduce 3-SAT to the flow problem. Two key proofs are given in Appendices A+B.

## 2 Aliased register allocation for straight-line programs

**Programs.** We will define a family of programs that compute with short values and long values. A short value can be represented with half as many bits as a long value. We use $v$ to range over program variables; a variable is either of type *short* or of type *long*. A variable of type short can only hold short values, and a variable of type long can only hold long values. We define a *statement* by this

grammar:

$$\begin{array}{rll} \text{(Statement) } S ::= & \texttt{short } v = & \text{(definition of } v\text{)} \\ | & \texttt{long } v = & \text{(definition of } v\text{)} \\ | & = v & \text{(use of } v\text{)} \end{array}$$

A statement either defines a variable or uses a variable. We define a *straight-line program* to be a sequence of statements with the property that each variable is defined only once and used at least once, and every use of a variable comes after its definition.

In program $S_1; \ldots; S_q$, a variable $v$ is *live* at statement $S_j$, if $v$ is defined at $S_i, i \leq j$ and $v$ is used at $S_k$, $j < k$ [19]. Let $i$ be the index of the statement that defines $v$, and let $k$ be the index of the last statement that uses $v$. The *live range* of $v$ is the half open interval $[i, k[$, which includes $i$ and excludes $k$.

If $v_1, v_2$ are variables and their live ranges have a nonempty intersection, then we say that $v_1, v_2$ *interfere* [5].

**Aliased Register Allocation.** Suppose we have a machine with $2K$ registers that each can hold a short value. The registers are called $r_0, \ldots, r_{2K-1}$; we call them *short registers*. Suppose further that any two registers $r_{2i}$, $r_{2i+1}$, where $i \in 0..K-1$, can be used to hold a long value. Notice the restriction that two registers can hold a long value only if the indices are consecutive and the first index is even; we call this restriction the *alignment restriction*. The alignment restriction models, for example, the rule for how a programmer can use the 8-bit registers on the x86. For example, $r_4, r_5$ can hold a long value, while $r_7, r_8$ cannot. We say that the two numbers $2i, 2i+1$ are *aligned*, and that the two registers $r_{2i}$, $r_{2i+1}$ are aligned. We use the notation that for a natural number $i$, $\overline{2i} = 2i+1$ and $\overline{2i+1} = 2i$.

We will study the problem of mapping program variables to machine registers:

> Core aliased register allocation with alignment restrictions (Caraar):
> **Instance**: a straight line program with $s$ short variables and $l$ long variables, and a number $2K$ of available short registers $r_0, \ldots, r_{2K-1}$.
> **Problem**: Can each short variable be mapped to one of the short registers and can each long variable be mapped to a pair $r_{2i}$, $r_{2i+1}, i \in 0..K-1$, of short registers, such that variables with interfering live ranges are assigned registers that are all different?

## 3 Interval graphs and aligned 1-2-coloring

**Interval Graphs.** We recall the definitions of an *intersection* graph and an *interval* graph [10, p.9].

Let $\mathcal{S}$ be a family of nonempty sets. The intersection graph of $\mathcal{S}$ is obtained by representing each set in $\mathcal{S}$ by a vertex and connecting two vertices by an edge if and only if their corresponding sets intersect. An *interval* graph is an intersection graph of a family of subintervals of an interval of the real numbers.

We will examine interval graphs with two kinds of intervals, called short and long intervals; we call such graphs *aliased interval graphs*.

**Aligned 1-2 Coloring.** We will study a variant of graph coloring which we call *aligned 1-2-coloring*. We will use natural numbers as colors; for example, if we have $2K$ colors, then we will use $0, 1, \ldots, 2K-1$ as the colors. We will color each vertex, that is, each interval. We will use the terms "short interval" and "short vertex" interchangeably; and similarly for "long interval" and "long vertex". We define a *1-2-coloring* to be a mapping that assigns one color to each short vertex and two colors $i, i+1$, $i \in 0..2K-2$, to each long vertex such that adjacent vertices have colors that are all different. We define an *aligned 1-2-coloring* to be a 1-2-coloring that assigns two aligned colors to each long vertex.

ALIGNED 1-2-COLORING OF ALIASED INTERVAL GRAPHS (A12CAIG):
**Instance**: an aliased interval graph and a number $2K$ of colors.
**Problem**: Find an aligned 1-2-coloring that uses $2K$ colors.

We will show that A12CAIG is NP-complete.

**From aligned 1-2 coloring to aliased register allocation** We now present a reduction of aligned 1-2-coloring of aligned interval graphs to aliased register allocation with alignment restrictions. The key step is to show that any aliased interval graph is the interference graph of one of our straight-line programs. We first define a convenient representation of interval graphs. We say that an interval graph is *program like* if (1) all the intervals are of the form $[u, v[$, (2) the start and end points of the intervals form the set $1..2q$, where $q$ is the number of intervals, (3) no two intervals start at the same point, (4) no two intervals end at the same point, and (5) no interval starts in the point where another interval ends.

**Proposition 1.** *A graph is an interval graph if and only if it is a program-like interval graph.*

*Proof.* The right-to-left direction is immediate. To prove the left-to-right direction, let $H$ be an interval graph with $q$ intervals whose start and end points are drawn from the set $\mathcal{U}$. We can in polynomial time construct an enumeration $\ell : 1..2q \to \mathcal{U}$ such that if $u < v$, then $\ell(u) \leq \ell(v)$. The graph

$$\{\, [u, v[ \mid \exists \text{ an interval in } H \text{ for which } \ell(u) \text{ is start point and } \ell(v) \text{ is end point} \,\}$$

is a program-like interval graph. □

From a program-like interval graph $H$, we construct a program $P = S_1; \ldots; S_{2q}$ as follows. Define

$$\forall i \in 1..2q : S_i = \begin{cases} \texttt{short}\ v_I = & \text{if the short interval } I \text{ begins at } i \\ \texttt{long}\ v_I = & \text{if the long interval } I \text{ begins at } i \\ = v_I & \text{if the interval } I \text{ ends at } i \end{cases}$$

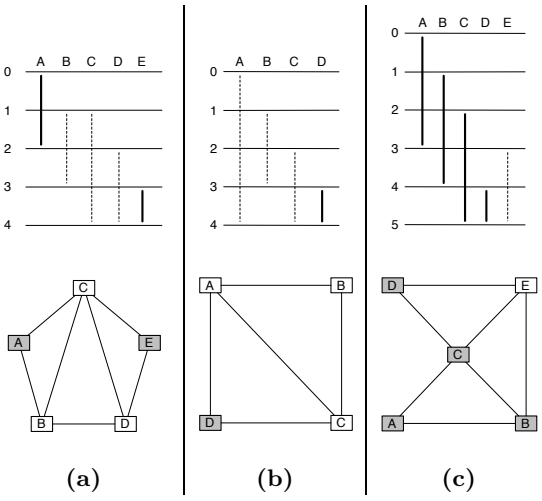


**Fig. 2.** Aliased interval graphs.

**Lemma 1.** *$H$ is the interference graph of $P$.*

*Proof.* For every interval $I$ in $H$, the live range of $v_I$ in $P$ is $I$. □

**Example** Let us explain why aligned 1-2-coloring is a nontrivial problem. Figure 2 shows three aliased interval graphs; each graph is displayed both as a collection of intervals and in a conventional way. In the upper part of Figure 2, we use fat lines to denote long intervals and we use dashed lines to denote short intervals. In the lower part of Figure 2, we use dark boxes to denote "long" vertices (representing long intervals) and we use light boxes to denote "short" vertices (representing short intervals).

A standard interval graph has the property that the size of the largest clique is equal to the minimal number of colors [10, p.17]. Aligned coloring of an aliased interval graph does not necessarily have that property. For example, Figure 2 (a) shows a graph for which the minimal 1-2-coloring uses four colors: $A = \{0, 1\}, B = 2, C = 3, D = 0, E = \{1, 2\}$, while the minimal aligned 1-2-coloring uses five colors: $A = \{0, 1\}, B = 2, C = 3, D = 4, E = \{0, 1\}$. Notice that the largest clique is of size 3; even if we treat long variables as counting as two nodes, the largest clique is of size 4.

A standard interval graph has the property that we can optimally color the graph by applying greedy coloring to any perfect elimination ordering of the vertices. (In a perfect elimination ordering, the neighbors of a node $v$ that come before $v$ in the ordering form a clique [10, p.82].) An aliased interval graph does not necessarily have that property. For example, Figure 2 (b) shows a graph for which we have the perfect elimination ordering $\langle A, B, C, D\rangle$ that leads greedy coloring to produce an aligned 1-2-coloring with five colors: $A = 0, B = 1, C = 2, D = \{4, 5\}$. If we drop the alignment restriction, greedy coloring again produces a 1-2-coloring with five colors: $A = 0, B = 1, C = 2, D = \{3, 4\}$. However,

in both the aligned and unaligned cases, there exists an optimal assignment using just four colors: $A = 0, B = 2, C = 1, D = \{2, 3\}$.

We might try an algorithm that first applies greedy coloring to the short intervals and then proceeds to color the longs. That does not necessarily lead to an optimal 1-2-coloring. For example, Figure 2 (b) shows a graph for which we have already studied the perfect elimination ordering $\langle A, B, C, D\rangle$ in which all the short intervals come before the long intervals. So, we will get the same suboptimal colorings as above.

Alternatively, we might try to first apply greedy coloring to the long intervals, and then proceed to color the shorts. That method is not optimal either. For example, Figure 2 (c) shows a graph for which the "longs-first" method produces the 1-2-coloring $A = \{0, 1\}, B = \{2, 3\}, C = \{4, 5\}, D = \{0, 1\}, E = 6$. Notice that the 1-2-coloring is also an aligned 1-2-coloring. However, in both the aligned and unaligned cases, an optimal assignment uses just six colors: $A = \{0, 1\}, B = \{2, 3\}, C = \{4, 5\}, D = \{2, 3\}, E = 0$.

None of the simple methods work because the aligned and unaligned 1-2-coloring problems are NP-complete.

## 4 Simple Graphs, Straight Cuts, and Colored Flows

Let $(V, E, \mathit{Source}, \mathit{Sink}, c)$ be a directed graph with vertex set $V$, edge set $E$, distinguished vertices $\mathit{Source}, \mathit{Sink} \in V$, and a capacity function $c : E \to \mathit{Nat}$, where *Nat* denotes the natural numbers.

A *flow* is function $f : E \to \mathit{Nat}$, such that

$$\forall (u, v) \in E : f(u, v) \leq c(u, v) \qquad \text{(Capacity)}$$

$$\forall v \in V \setminus \{\mathit{Source}, \mathit{Sink}\} : \Sigma_{(u,v)\in E} f(u, v) = \Sigma_{(v,w)\in E} f(v, w) \qquad \text{(Conservation)}$$

The value of a flow is the sum of the flows of the edges that reach *Sink*. A *maximal flow* is flow $f$ such that for any flow $g$, the value of $g$ is less than or equal to the value of $f$.

We define a set of vertices $S$ to be *backwards closed* if $\forall v \in S$ : if $(u, v) \in E$, then $u \in S$. We also define a set of vertices $T$ to be *forwards closed* if $\forall u \in T$ : if $(u, v) \in E$, then $v \in T$. A cut $(S, T)$ is a partition of $V$ such that $\mathit{Source} \in S, \mathit{Sink} \in T$, $S \cap T = \emptyset$, and $S \cup T = V$. The capacity of a cut $(S, T)$, written $c(S, T)$ is given by the formula:

$$c(S, T) = \Sigma_{(u,v)\in E, u\in S, v\in T}\, c(u, v)$$

which says that the capacity of the cut is the sum of the capacities of the edges that cross the cut from $S$ to $T$. A *straight cut* is a cut $(S, T)$ such that $S$ is backwards closed and $T$ is forwards closed. We define a *simple graph* to be an acyclic graph $(V, E, \mathit{Source}, \mathit{Sink}, c)$ in which *Source* has no incoming edges, *Sink* has no outgoing edges, and where all the straight cuts have the same capacity.

Figure 3 (a) shows a simple graph. Each dashed line marks a straight cut. Each edge with nonunit capacity is marked with a small bar and its capacity; unlabeled edges have unit capacity.

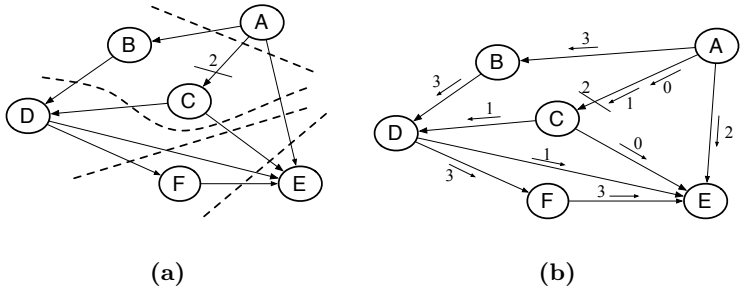


**Fig. 3.** (a) Simple graph; (b) colored flow.

**Lemma 2.** *All straight cuts have the same capacity if and only if* $\forall v \in V \setminus \{Source, Sink\} : \sum_{(u,v)\in E} c(u,v) = \sum_{(v,w)\in E} c(v,w)$.

*Proof.* $\Rightarrow$) We can choose a straight cut $(S,T)$ such that $v \in S$, and another straight cut $(S - \{v\}, \{v\} \cup T)$. The first cut contains the outgoing edges of $v$ and the second cut has all the incoming edges of $v$. The remaining edges of $v$ are common to both cuts and thus by the simple graph property of all straight cuts being equal, the capacity of the incoming edges must equal the capacity of the outgoing edges.

$\Leftarrow$) If all vertices have the same incoming capacity as outgoing capacity, then we must show that all straight cuts have the same capacity. The straight cut $(\{Source\}, V - \{Source\})$ has some capacity $K$. If we add one vertex $v$ connected to *Source* to the first set and remove $v$ from the second set, then by the incoming-equals-outgoing-capacity property of $v$, this new straight cut also has a capacity of $K$. We may enumerate the straight cuts by adding and subtracting vertices. By the same reasoning as above, they all have the same capacity. □

**Lemma 3.** *In a simple graph, c is the maximal flow.*

*Proof.* First we show that $c$ satisfies the capacity constraints. This is immediate because the flow is the capacity. Next we show that the flow is conserved for every vertex. From Lemma 2 we have that the incoming capacity equals the outgoing capacity for every vertex, thus the flow is conserved. The flow is maximal because any increase in the flow would exceed the capacity constraints. □

We say that an element of $0..K-1$ is a *color*. We define a *colored flow* for a simple graph with every straight cut of capacity $K$ as a function $h : E \to 2^{0..K-1}$ such that $\lambda e.|h(e)|$ is a flow and for any straight cut $(S,T)$, we have $\cup_{(u,v)\in E, u\in S, v\in T} h(u,v) = 0..K-1$. Thus, for any straight cut, every color is used exactly once in the coloring of the edges that cross the cut. Notice that every color is used *at most* once because the straight cut has capacity $K$. We use Lemma 3 to justify the terminology that a *maximal* colored flow is a colored flow with the property that $\lambda e.|h(e)| = c$.

Figure 3 (b) shows an example of colored flow.

**Lemma 4.** *For a simple graph, $h$ is a colored flow if and only if $\lambda e.|h(e)|$ is a flow, $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u,v) = \cup_{(v,w)\in E} h(v,w)$, and $\exists$ straight cut $(S,T)$ such that $\cup_{(u,v)\in E, u\in S, v\in T} h(u,v) = 0..K-1$,.*

*Proof.* The proof is similar to the proof of Lemma 2 where we here reason about colors instead of capacities. Notice that for the right-to-left direction, we can use the given straight cut to show that for any straight cut $(S,T)$, we have $\cup_{(u,v)\in E, u\in S, v\in T} h(u,v) = 0..K-1$. We omit the rest of the details. □

**Aligned colored flow.** Suppose we have a simple graph $(V, E, Source, Sink, c)$ with all straight cuts of capacity $2K$, a function $A : E \to Boolean$, and a number $2K$ of colors $0, \ldots, 2K-1$. We define an *aligned* colored flow to be a colored flow $h$ such that if $A(e) = true$ and $2 \leq c(e)$, then $\exists i : 0 \leq i \leq K-1 \wedge \{2i, 2i+1\} \subseteq h(e)$. Intuitively, the function $A$ indicates that an edge $e$ with a capacity of at least two requires $h$ to assign $e$ the colors $2i$ and $2i+1$, among others.

MAXIMAL, ALIGNED COLORED FLOW:
**Instance**: $(G, 2K, A)$, where $G$ is a simple graph $(V, E, Source, Sink, c)$ with all straight cuts of capacity $2K$, and $A : E \to Boolean$.
**Problem**: Find a maximal, aligned colored flow that uses $2K$ colors.

## 5 From maximal, aligned colored flow to aligned 1-2 coloring

In this section we present a reduction of the maximal, aligned colored flow problem to aligned 1-2-coloring of aliased interval graphs. Let $(G, 2K, A)$ be an instance of the maximal, aligned colored flow problem, where $G = (V, E, Source, Sink, c)$. From $(G, 2K, A)$ we construct an aliased interval graph $H$ in the following way.

First we sort $V$ into a topological order with *Source* first and *Sink* last. We can do that because $G$ is a simple graph so *Source* has no incoming edges and *Sink* has no outgoing edges.

Next we define an injective function $\ell : V \to Nat$ such that if $v_1$ is less than $v_2$ in the topological ordering, then $\ell(v_1) < \ell(v_2)$. The numbers assigned by $\ell$ to the vertices of $G$ will be the start and end points of the intervals in $H$. The intervals of $H$ are defined as follows. For each $(u,v) \in E$ such that $A(u,v) = true$, we create one long interval $[\ell(u), \ell(v)[$ and $c(u,v) - 2$ short intervals $[\ell(u), \ell(v)[$. For each $(u,v) \in E$ such that $A(u,v) = false$, we create $c(u,v)$ short intervals $[\ell(u), \ell(v)[$.

**Lemma 5.** *$(G, 2K, A)$ has a maximal, aligned colored flow if and only if $H$ has an aligned 1-2-coloring.*

*Proof.* See Appendix A. □

| Vertex | In Cap | Vertex | In Cap |
|---|---|---|---|
| *Source* | 0 | $x_{ik}$ | 2 |
| *Sink* | 2K | $s_{ik}$ | 2 |
| $S$ | 2 | $\bar{x}_{ih}$ | 2 |
| $T$ | 2 | $\bar{s}_{ih}$ | 2 |
| $c_j$ | 6 | $a_i$ | 2 |
| $s_{i0}$ | 2 | $b_i$ | 2 |

**Fig. 4.** Vertices; $K = 3m + 3n + 1$.

## 6 From 3-SAT to maximal, aligned colored flow

In this section we present a reduction of 3-SAT to the maximal, aligned colored flow problem. Let

$$F = \wedge_{j=1}^{m} c_j$$
$$c_j = l_{j1} \vee l_{j2} \vee l_{j3}$$

be a formula with $n$ Boolean variables $x_1, \ldots, x_n$ and $m$ clauses $c_1, \ldots, c_m$; each literal, $l_{j1}$ or $l_{j2}$ or $l_{j3}$, is either a variable or the negation of a variable, and in each clause the three literals are distinct. Let $p_i$ be the number of occurrences of $x_i$, and let $q_i$ be the number of occurrences of $\bar{x}_i$. We index a certain set of vertices using $i$ and $k$, where $1 \leq i \leq n$ and $1 \leq k \leq p_i + 1$. We index another set of vertices using $i$ and $h$, where $1 \leq i \leq n$ and $1 \leq h \leq q_i + 1$. For convenience, we define $p_0 = 0$ and $q_0 = 0$.

From $F$ we construct a simple graph $G = (V, E, \mathit{Source}, \mathit{Sink}, c)$. The graph is akin to the graph used by Even, Itai and Shamir [6, Section 4] in their proof of NP-completeness for the multicommodity flow problem. Figure 4 shows a listing of the vertices in $V$, along with, for each vertex, the sum of the capacities of the in-going edges (which, because the graph is simple, is equal to the sum of the capacities of the outgoing edges, for all vertices except *Source*, *Sink*). Figure 6 shows a listing of the edges in $E$. Figure 6 also shows a set of colors for each edge; we will need that later. Figure 5 illustrates the graph constructed from the formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Let us now briefly walk through the construction of the graph.

We have the vertices *Source*, *Sink*, two vertices $S$ and $T$, one vertex $c_j$ for each clause, and vertices $s_{i0}, x_{ik}, s_{ik}, \bar{x}_{ih}, \bar{s}_{ih}, a_i, b_i$, for each variable $x_i$. For each variable $x_i$, we construct a lobe. We add the edges $(x_{iu} \xrightarrow{1} x_{i(u+1)}), 1 \leq u \leq p_i$ to form an upper path. We construct a lower path with the edges $(\bar{x}_{iv} \xrightarrow{1} \bar{x}_{i(v+1)}), 1 \leq v \leq q_i$. We connect theses paths to $a_i$ and $b_i$ to form the lobe by adding the edges $(a_i \xrightarrow{1} x_{i1}), (a_i \xrightarrow{1} \bar{x}_{i1}), (x_{i(p_i+1)} \xrightarrow{1} b_i)$, and $(\bar{x}_{i(q_i+1)} \xrightarrow{1} b_i)$.

Next we are going to make several edges that have alignment requirements. For each $s_{ik}$, we create an edge $(\mathit{Source} \xrightarrow{2} s_{ik})$ with a capacity two. Likewise for all the $s_{i0}$ and the $\bar{s}_{ih}$ vertices. Next we will add an edge $(\mathit{Source} \xrightarrow{2} S)$ also with

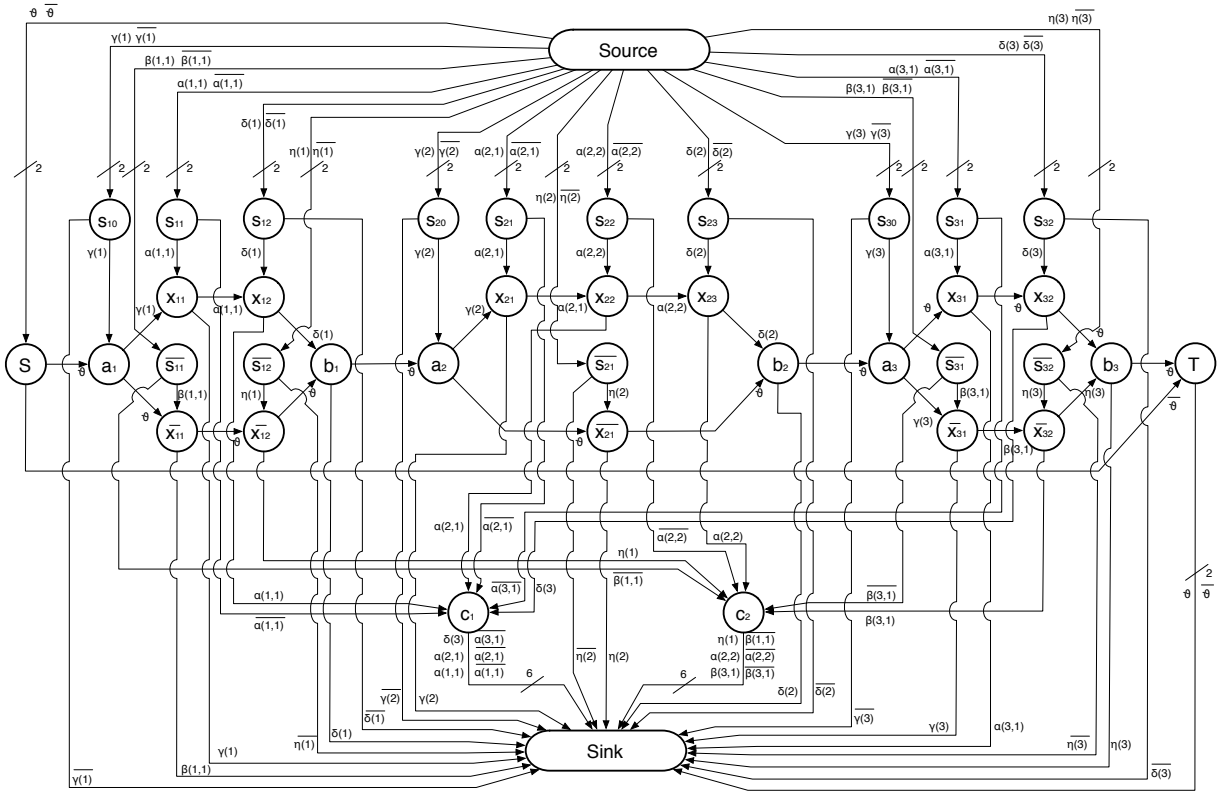


**Fig. 5.** Construction of a simple graph from $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Edges with a labeled capacity require aligned colors. The edge labels are explained in Figure 7(a).

capacity of two. In total we have made a capacity of $2(3m + 3n + 1)$ leaving the source. We want to make $G$ simple, so there must be capacities of two leaving each of these vertices and eventually reaching the *Sink*. We will create some more aligned edges which will now connect certain vertices to *Sink*. For each of the $c_j$ vertices, we create the edges $(c_j \xrightarrow{6} \mathit{Sink})$ with a capacity of six and finally we add $(T \xrightarrow{2} \mathit{Sink})$ with a capacity of two. Now all that remains to make the graph simple is to connect the $S, s_{ik}$, and $\bar{s}_{ih}$ vertices to $T, c_j$, and *Sink*.

We will first add edges to send the current excess capacity at $S$ to $T$. We will add a direct edge $(S \xrightarrow{1} T)$ to get one unit to $T$. To get the other unit to $T$, we will connect the lobes serially, by adding the edges $(b_l, a_{l+1}), 1 \leq l \leq n-1$. Finally, we add $(S \xrightarrow{1} a_1)$ and $(b_n \xrightarrow{1} T)$, resulting in a path to send the other unit of capacity to $T$ and two units of capacity reaching *Sink*.

The $a_i$ and $b_i$ vertices still have an imbalance of capacity and must have edges to supply capacity or drain it. To correct for these imbalances, we add the edges $(s_{i0} \xrightarrow{1} \mathit{Sink}), (s_{i0} \xrightarrow{1} a_i)$, and $(b_i \xrightarrow{1} \mathit{Sink})$. This results in a current total of $2n + 2$ units of capacity reaching *Sink*, and the vertices on the lobe currently balanced.

We will now connect the remaining $s_{ik}$ and $\bar{s}_{ih}$ vertices to the $c_j$ vertices and *Sink*. We add the edges $(s_{ik} \xrightarrow{1} x_{ik})$ and $(\bar{s}_{ih} \xrightarrow{1} \bar{x}_{ih})$ which will send one unit of capacity from each of these vertices to the corresponding vertices on the lobe. The other units from the $s_{ik}$ and $\bar{s}_{ih}$ vertices will be sent to either some $c_j$ vertex or directly to *Sink*. We add the edges $(s_{i(p_i+1)} \xrightarrow{1} \mathit{Sink})$ and $(\bar{s}_{i(q_i+1)} \xrightarrow{1} \mathit{Sink})$, which now results in an additional $2n$ units of capacity reaching *Sink* for a running total of $4n + 2$. The remaining vertices add an edge $(s_{iu} \xrightarrow{1} c_j)$ if the $u^{th}$ appearance of $x_i$ occurs in $c_j$. For the $\bar{s}_{iv}$ vertices, we add similar edges. From these edges we get $3m$ units of capacity reaching *Sink*, because each of these edges corresponds to a clause, and each clause has exactly three literals in it. All that remains is to drain the single unit of capacity currently residing at the $x_{ik}$ and $\bar{x}_{ih}$ vertices and we will have our simple graph. We add the edges $(x_{i(u+1)} \xrightarrow{1} c_j)$ if the $u^{th}$ appearance of $x_i$ occurs in $c_j$ as well as $(x_{i(v+1)} \xrightarrow{1} c_j)$ if the $v^{th}$ appearance of $\bar{x}_i$ occurs in $c_j$. This results in another $3m$ units of capacity reaching *Sink*. Finally, the last $2n$ units will be supplied by the edges $(x_{i1} \xrightarrow{1} \mathit{Sink})$ and $(\bar{x}_{i1} \xrightarrow{1} \mathit{Sink})$.

**Lemma 6.** *$G$ is simple.*

*Proof.* It is straightforward to check that $G$ is acyclic, that *Source* has no incoming edges, and that *Sink* has no outgoing edges. From Lemma 2 we have that we must show $\forall v \in V \setminus \{\mathit{Source}, \mathit{Sink}\} : \sum_{(u,v)\in E} c(u, v) = \sum_{(v,w)\in E} c(v, w)$. Let us examine each vertex $v \in V \setminus \{\mathit{Source}, \mathit{Sink}\}$ in turn. We see that each of the $c_i$ vertices has an in-capacity of six and an out-capacity of six. All the other vertices besides *Source* and *Sink* each has an in-capacity of two and an out-capacity of two. So, from Lemma 2 we have that $G$ is simple. □

Let $A : E \to \mathit{Boolean}$ be given by mapping each edge $e$ to the Boolean value obtained by computing $2 \leq c(e)$, where $c$ is the capacity function specified implicitly in Figure 6.

**Lemma 7.** *$F$ is satisfiable if and only if $(G, 2(3m + 3n + 1), A)$ has a maximal, aligned colored flow.*

*Proof.* See Appendix B. □

## 7 Main result and Conclusion

**Theorem 1.** *For straight-line programs, the core aliased register allocation problem with alignment restrictions is NP-complete.*

*Proof.* Firstly, we see the problem is in NP because a register assignment can be verified in polynomial time. We have a chain of reductions from 3-SAT to maximal, aligned colored flow (Lemma 7), from maximal, aligned colored flow to aligned 1-2 coloring (Lemma 5), and from aligned 1-2 coloring to aliased register allocation (Lemma 1). □

We have shown that aliased register allocation with alignment restrictions is difficult, even for straight-line programs where each variable has at most one definition point. Our result confirms the need for the heuristics and worst-case exponential time methods that are used today.

In this paper we have considered register allocation as a decision problem. We can view also view register allocation as an optimization problem: minimize the number of registers. Open problem: give nontrivial upper and lower bounds on the approximability of our register allocation problem. For example, is our register allocation problem APX-hard?

## Appendix A: Proof of Lemma 5

**Lemma** $(G, 2K, A)$ has a maximal, aligned colored flow if and only if $H$ has an aligned 1-2-coloring.

*Proof.* $\Rightarrow$) Suppose $(G, 2K, A)$ has a maximal, aligned colored flow $h$. We can then define a mapping $\kappa$ that assigns a color to each vertex of $H$ as follows. For each $(u, v) \in E$ such that $A(u, v) = true$, we have that $h(u, v)$ contains two aligned colors; assign those two colors to the long interval created from $(u, v)$, and assign each of the rest of the colors in $h(u, v)$ to each of the short intervals created from $(u, v)$. For each $(u, v) \in E$ such that $A(u, v) = false$, assign each of the colors in $h(u, v)$ to each of the short intervals created from $(u, v)$. We need to show that adjacent vertices in $H$ have colors that are all different. Suppose we have two adjacent intervals $I_1, I_2$ in $H$, that is, they have a nonempty intersection. Since all the intervals are half-open, the intersection consists of more than one point. Choose a point $p$ in the intersection which is not the start or end point of any interval in $H$. Define $S = \{\ v \in V \mid \ell(v) < p\ \}$ and define $T = \{\ v \in V \mid \ell(v) > p\ \}$. We have that $(S, T)$ is a straight cut because we use a topological sort of $V$. We also have that $I_1, I_2$ both cross the cut. For every straight cut, every color is used exactly once, so $I_1, I_2$ have colors that are all different.

$\Leftarrow$) Suppose $H$ has an aligned 1-2 coloring $\kappa$. We can then define a mapping $h : E \rightarrow 2^{0..2K-1}$ as follows. For each $(u, v) \in E$, let $h(u, v)$ be the union of the colors assigned by $\kappa$ to the intervals in $H$ created from $(u, v)$. We need to show that $h$ is a maximal, aligned colored flow.

Let us first show that $h$ is a colored flow. From Lemma 4 we have that we must show that (1) $\lambda e.|h(e)|$ is a flow, (2) $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$, and (3) $\exists$ straight cut $(S, T)$ such that $\cup_{(u,v)\in E, u\in S, v\in T} h(u, v) = 0..K - 1$. To prove (1), note that for each edge $e \in E$, the intervals in $H$, created from $e$, overlap and hence get all different colors by $\kappa$ so $|h(e)| = c(e)$. From Lemma 3 we have that $c$ is a flow. From that $c$ a flow and $|h(e)| = c(e)$, we have that $\lambda e.|h(e)|$ is a flow, as required. To prove (2), let $v$ be a vertex in $V \setminus \{Source, Sink\}$. We have that

$$C_1 = (\{\ u \mid \ell(u) < \ell(v)\ \}, \{\ w \mid \ell(v) \leq \ell(w)\ \})$$
$$C_2 = (\{\ u \mid \ell(u) \leq \ell(v)\ \}, \{\ w \mid \ell(v) < \ell(w)\ \})$$

are straight cuts in $G$ and hence both of capacity $2K$. Next define $E_1$ to be the set of edges in $E$ of the form $(u, v)$, and define $E_2$ to be the set of edges in $E$ of the form $(v, w)$. From Lemma 2 we have that $\Sigma_{e\in E_1} c(e) = \Sigma_{e\in E_2} c(e)$. We can find a subset $E' \subseteq E$ such that $E' \cap E_1 = \emptyset$, $E' \cap E_2 = \emptyset$, the edges that cross $C_1$ can be written $E_1 \cup E'$, and the edges that cross $C_2$ can be written $E_2 \cup E'$. We conclude that $\cup_{e\in E_1} h(e) = \cup_{e\in E_2} h(e)$, as required. To prove (3), let us examine the edges that traverse the straight cut $(\{Source\}, V \setminus \{Source\})$. By reasoning similar to what we used to proved (1), we see that all colors on the edges spanning the straight cut must be distinct.

Let us next show that $h$ is a maximal colored flow. We need to show that $\lambda e.|h(e)| = c$ and that is immediate from $|h(e)| = c(e)$.

Finally, we have that $h$ is aligned because the construction of $H$ ensures that for any $e$ for which we have $A(e) = true$, one long edge is created; the construction of $h$ then ensures that the aligned colors assigned by $\kappa$ to that edge will be two of the colors assigned to $e$. $\square$

## Appendix B: Proof of Lemma 7

**Lemma** $F$ is satisfiable if and only if $(G, 2(3m + 3n + 1), A)$ has a maximal, aligned colored flow.

*Proof.* Define $K = 3m + 3n + 1$.

$\Rightarrow$) Suppose we have a Boolean assignment $\psi$ for the variables in $F$ which satisfies $F$. Let $h : E \to 0..2K-1$ be defined by mapping each edge to the set of colors given in Figure 6, using the abbreviations in Figure 7. We need to show that $h$ is a maximal, aligned colored flow.

We can verify that $\lambda e.|h(e)|$ is a maximal flow by inspection of Figure 6: each edge has a capacity which matches the number of colors in $h(e)$.

From Lemma 4 we have that we can show that $h$ is a colored flow by showing that $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$. We can do this by examining each vertex, in a manner similar to what we did in the proof of Lemma 6; we omit the details (tedious!). We also must show that there exists a straight cut such that $\cup_{(u,v)\in E, u\in S, v\in T} h(u, v) = 0..K-1$. Examining Figure 6, we see that the cut $(\{Source\}, V \setminus \{Source\})$ uses all $K$ colors.

Finally we must show that $h$ is an aligned colored flow, that is, we must show that if $A(e) = true$, then $\exists i : 0 \le i \le K-1 \wedge \{2i, 2i+1\} \subseteq h(e)$. We can easily verify that by inspection of Figure 6.

$\Leftarrow$) Suppose we have a maximal, aligned colored flow $h$ of $G$ that uses the colors $0, \ldots, 2K-1$. We can then define a Boolean assignment $\psi$ for the variables of $F$:

$$\text{for all } i \in 1..n : \psi(x_i) = \begin{cases} \text{true} & \text{if } h(Source \xrightarrow{2} S) = h(S \xrightarrow{1} T) \cup h(a_i \xrightarrow{1} \bar{x}_{i1}) \\ \text{false} & \text{otherwise} \end{cases}$$

Let $j \in 1..m$. We will show that $\psi$ satisfies $c_j$. For convenience, we let $S$ have the alias $b_0$, and we let $T$ have the alias $a_{n+1}$; this simplifies the statement of Claim 2 below.

We first prove five properties of $h$:

- Claim 1: $h(Source \xrightarrow{2} S) = h(T \xrightarrow{2} Sink)$.
- Claim 2: $h(Source \xrightarrow{2} S) = h(S \xrightarrow{1} T) \cup h(b_i \xrightarrow{1} a_{i+1}), i \in 0..n$.
- Claim 3: (a) If there exists $k$ such that $h(Source \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} Sink)$, then $x_i$ appears in $c_j$. (b) if there exists $h$ such that $h(Source \xrightarrow{2} \bar{s}_{ih}) \subseteq h(c_j \xrightarrow{6} Sink)$, then $\bar{x}_i$ appears in $c_j$.
- Claim 4: For $i \in 1..n$: (a) $h(Source \xrightarrow{2} s_{i0}) \not\subseteq h(c_j \xrightarrow{6} Sink)$, (b) $h(Source \xrightarrow{2} s_{i(p_i+1)}) \not\subseteq h(c_j \xrightarrow{6} Sink)$, (c) $h(Source \xrightarrow{2} \bar{s}_{i(q_i+1)}) \not\subseteq h(c_j \xrightarrow{6} Sink)$.
- Claim 5: Either there exists $i, k$ such that $h(Source \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} Sink)$, or there exists $i, h$ such that $h(Source \xrightarrow{2} \bar{s}_{ih}) \subseteq h(c_j \xrightarrow{6} Sink)$.

Proof of Claim 1. From Lemma 4, $h(Source \xrightarrow{2} S) = h(S \xrightarrow{1} T) \cup h(S \xrightarrow{1} a_1)$ and $h(S \xrightarrow{1} T) \cup h(b_n \xrightarrow{1} T) = h(T \xrightarrow{2} Sink)$. This shows that one color $d$ from

$h(\mathit{Source} \xrightarrow{2} S)$ is in $h(T \xrightarrow{2} \mathit{Sink})$. From the alignment requirements, we have that both $h(\mathit{Source} \xrightarrow{2} S)$ and $h(T \xrightarrow{2} \mathit{Sink})$ must contain $\{d,\bar{d}\}$ and therefore are equal.

Proof of Claim 2. From Lemma 4, $h(\mathit{Source} \xrightarrow{2} S) = h(S \xrightarrow{1} T) \cup h(S \xrightarrow{1} a_1)$, so one color from $h(\mathit{Source} \xrightarrow{2} S)$ is in $h(S \xrightarrow{1} T)$. Additionally, the complementing color takes some other path from $S$ to $T$ and by inspection, we see that all paths from $S$ to $T$ contain the edges $(b_i, a_{i+1})$.

Proof of Claim 3. (a) Suppose $h(\mathit{Source} \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} \mathit{Sink})$. The $k^{th}$ occurrence of $x_i$ is in $c_r$ for some $r$, and $G$ contains the edges $s_{ik} \xrightarrow{1} c_r$ and $x_{i(k+1)} \xrightarrow{1} c_r$. We must show $j = r$. Suppose $j \neq r$. We can now choose a straight cut $(S,T)$ such that $c_j \in S$ and $c_r \in T$. Notice that the two edges $c_j \xrightarrow{6} \mathit{Sink}$ and $s_{ik} \xrightarrow{1} c_r$ both cross $(S,T)$ and hence must have colors that are all different. However, from Lemma 4 we have that $h(s_{ik} \xrightarrow{1} c_r) \subseteq h(\mathit{Source} \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} \mathit{Sink})$, a contradiction. We conclude $j = r$. (b) The proof is similar to the proof of (a), we omit the details.

Proof of Claim 4. (a) We can choose a straight cut $(S,T)$ such that $c_j \in S$ and $s_{i0} \in T$. Notice that the two edges $\mathit{Source} \xrightarrow{2} s_{i0}$ and $c_j \xrightarrow{6} \mathit{Sink}$ both cross $(S,T)$ and hence must have colors that are all different. (b),(c): The proofs are similar to the proof of (a), we omit the details.

Proof of Claim 5. The set $h(c_j \xrightarrow{6} \mathit{Sink})$ contains two aligned colors $d, \bar{d}$. The edges crossing the straight cut $(\mathit{Source}, V \setminus \mathit{Source})$ are of the six forms: (i) $\mathit{Source} \xrightarrow{2} S$, (ii) $\mathit{Source} \xrightarrow{2} s_{i0}$, (iii) $\mathit{Source} \xrightarrow{2} s_{i(p_i+1)}$, (iv) $\mathit{Source} \xrightarrow{2} \bar{s}_{i(q_i+1)}$, (v) $\mathit{Source} \xrightarrow{2} s_{ik}$, (vi) $\mathit{Source} \xrightarrow{2} \bar{s}_{ih}$. Because every color is used exactly once across a straight cut, one of those edges must have the colors $d, \bar{d}$. We must rule out cases (i)–(iv). Consider first case (i). The straight cut $(V \setminus \mathit{Sink}, \mathit{Sink})$ is crossed by the edges $T \xrightarrow{2} \mathit{Sink}$ and $c_j \xrightarrow{6} \mathit{Sink}$, hence those edges must have different colors. From Claim 1 we have $h(\mathit{Source} \xrightarrow{2} S) = h(T \xrightarrow{2} \mathit{Sink})$, so $h(\mathit{Source} \xrightarrow{2} S) \not\subseteq h(c_j \xrightarrow{6} \mathit{Sink})$. Consider then cases (ii)–(iv). Those cases are impossible because of Claim 4.

Finally, we will show that $\psi$ satisfies $c_j$. From Claim 5 we have that either (I) there exists $i,k$ such that $h(\mathit{Source} \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} \mathit{Sink})$, or (II) there exists $i,h$ such that $h(\mathit{Source} \xrightarrow{2} \bar{s}_{ih}) \subseteq h(c_j \xrightarrow{6} \mathit{Sink})$. We will show that $\psi$ satisfies $c_j$ in the case of (I); the case (II) can be proved in a similar fashion. From Claim 3 we have that $x_i$ appears in $c_j$. From Lemma 4 we have that we can find a color $d$ such that $h(\mathit{Source} \xrightarrow{2} s_{ik}) = \{d, \bar{d}\}$ and $h(s_{ik} \xrightarrow{1} c_j) = \{d\}$. We have that the literals in clause $c_j$ are distinct so when we consider the edges $x_{ik} \xrightarrow{1} c_r$ and $x_{i(k+1)} \xrightarrow{1} c_j$, we have that $r \neq j$. From $h(\mathit{Source} \xrightarrow{2} s_{ik}) \subseteq h(c_j \xrightarrow{6} \mathit{Sink})$ we have that $\bar{d}$ is a color of some edge to $c_j$ and hence not a color of any edge to $c_r$. The vertex $x_{ik}$ has two outgoing edges $x_{ik} \xrightarrow{1} x_{i(k+1)}$ and $x_{ik} \xrightarrow{1} c_r$, so $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) = \{\bar{d}\}$. So we have $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) \subseteq h(\mathit{Source} \xrightarrow{2} s_{ik})$. The edges $\mathit{Source} \xrightarrow{2} S$ and

$Source \xrightarrow{2} s_{ik}$ both cross the straight cut $(\{Source\}, V \setminus \{Source\})$ and hence have colors that are all different. We thus have that $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) \not\subseteq h(Source \xrightarrow{2} S)$. From $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) \not\subseteq h(Source \xrightarrow{2} S)$ and Claim 2 we have that $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) \neq h(b_i \xrightarrow{1} a_{i+1})$. By Claim 2, we have that $h(b_{i-1} \xrightarrow{1} a_i) = h(b_i \xrightarrow{1} a_{i+1})$. Note that there exists two paths from $a_i$ to $b_i$. One path consists of vertices $a_i$, $x_{i1},\ldots,x_{i(p_i+1)}$, $b_i$; the other of vertices $a_i$, $\bar{x}_{i1},\ldots,\bar{x}_{i(q_i+1)}$, $b_i$. From Lemma 4 we have that the color of $h(b_i \xrightarrow{1} a_{i+1})$ must be assigned to all edges on one of these paths. From $h(x_{ik} \xrightarrow{1} x_{i(k+1)}) \neq h(b_i \xrightarrow{1} a_{i+1})$ we conclude that the color of $h(b_i \xrightarrow{1} a_{i+1})$ must be assigned to all edges on the path $a_i$, $\bar{x}_{i1},\ldots,\bar{x}_{i(q_i+1)}$, $b_i$. We conclude $\psi(x_i) = true$. Since $x_i$ appears in $c_j$, we have that $\psi$ satisfies $c_j$. □

| Edge | Color | Edge | Color |
|---|---|---|---|
| $x_{iu} \xrightarrow{1} x_{i(u+1)}$ | $\psi(x_i)\ ?\ \alpha(i,u) : \theta$ | $s_{i0} \xrightarrow{1} Sink$ | $\overline{\gamma(i)}$ |
| $\bar{x}_{iv} \xrightarrow{1} \bar{x}_{i(v+1)}$ | $\psi(x_i)\ ?\ \theta : \beta(i,v)$ | $s_{i0} \xrightarrow{1} a_i$ | $\gamma(i)$ |
| $a_i \xrightarrow{1} x_{i1}$ | $\psi(x_i)\ ?\ \gamma(i) : \theta$ | $b_i \xrightarrow{1} Sink$ | $\psi(x_i)\ ?\ \delta(i) : \eta(i)$ |
| $a_i \xrightarrow{1} \bar{x}_{i1}$ | $\psi(x_i)\ ?\ \theta : \gamma(i)$ | $s_{iu} \xrightarrow{1} x_{iu}$ | $\alpha(i,u)$ |
| $x_{i(p_i+1)} \xrightarrow{1} b_i$ | $\psi(x_i)\ ?\ \delta(i) : \theta$ | $s_{i(p_i+1)} \xrightarrow{1} x_{i(p_i+1)}$ | $\delta(i)$ |
| $\bar{x}_{i(q_i+1)} \xrightarrow{1} b_i$ | $\psi(x_i)\ ?\ \theta : \eta(i)$ | $\bar{s}_{iv} \xrightarrow{1} \bar{x}_{iv}$ | $\beta(i,u)$ |
| $Source \xrightarrow{2} s_{iu}$ | $\alpha(i,u), \overline{\alpha(i,u)}$ | $\bar{s}_{i(q_i+1)} \xrightarrow{1} \bar{x}_{i(q_i+1)}$ | $\eta(i)$ |
| $Source \xrightarrow{2} s_{i(p_i+1)}$ | $\delta(i), \overline{\delta(i)}$ | $s_{i(p_i+1)} \xrightarrow{1} Sink$ | $\overline{\delta(i)}$ |
| $Source \xrightarrow{2} \bar{s}_{iv}$ | $\beta(i,u), \overline{\beta(i)}$ | $\bar{s}_{i(q_i+1)} \xrightarrow{1} Sink$ | $\overline{\eta(i)}$ |
| $Source \xrightarrow{2} \bar{s}_{i(q_i+1)}$ | $\eta(i), \overline{\eta(i)}$ | $s_{iu} \xrightarrow{1} c_j$ if $\text{occ}(u, x_i, c_j)$ | $\overline{\alpha(i,u)}$ |
| $Source \xrightarrow{2} s_{i0}$ | $\gamma(i), \overline{\gamma(i)}$ | $\bar{s}_{iv} \xrightarrow{1} c_j$ if $\text{occ}(v, \bar{x}_i, c_j)$ | $\overline{\beta(i,u)}$ |
| $Source \xrightarrow{2} S$ | $\theta, \bar{\theta}$ | $x_{i(u+1)} \xrightarrow{1} c_j$ if $\text{occ}(u, x_i, c_j)$, $(u \neq p_i)$ | $\psi(x_i)\ ?\ \alpha(i,u) : \alpha(i,u+1)$ |
| $c_j \xrightarrow{6} Sink$ | See Figure 7(b) | $x_{i(p_i+1)} \xrightarrow{1} c_j$ if $\text{occ}(p_i, x_i, c_j)$ | $\psi(x_i)\ ?\ \alpha(i,p_i) : \delta(i)$ |
| $T \xrightarrow{2} Sink$ | $\theta, \bar{\theta}$ | $\bar{x}_{i(v+1)} \xrightarrow{1} c_j$ if $\text{occ}(v, \bar{x}_i, c_j)$, $(v \neq q_i)$ | $\psi(x_i)\ ?\ \beta(i,v) : \beta(i,v+1)$ |
| $S \xrightarrow{1} T$ | $\bar{\theta}$ | $\bar{x}_{i(q_i+1)} \xrightarrow{1} c_j$ if $\text{occ}(q_i, \bar{x}_i, c_j)$ | $\psi(x_i)\ ?\ \eta(i) : \beta(i,q_i)$ |
| $b_l \xrightarrow{1} a_{l+1}$ | $\theta$ | $x_{i1} \xrightarrow{1} Sink$ | $\psi(x_i)\ ?\ \gamma(i) : \alpha(i,1)$ |
| $S \xrightarrow{1} a_1$ | $\theta$ | $\bar{x}_{i1} \xrightarrow{1} Sink$ | $\psi(x_i)\ ?\ \gamma(i) : \beta(i,1)$ |
| $b_n \xrightarrow{1} T$ | $\theta$ | | |

**Fig. 6.** Edge construction; $\psi(x_i)?d_T : d_F$ denotes that if $\psi(x_i) = true$ then the assigned color is $d_T$ and if $\psi(x_i) = false$ then the assigned color is $d_F$. $1 \leq i \leq n, 1 \leq j \leq m, 1 \leq u \leq p_i, 1 \leq v \leq q_i, 1 \leq l \leq n-1$. An expression $\text{occ}(u, x_i, c_j)$ means that the $u^{th}$ occurrence of $x_i$ appears in $c_j$.

$$\begin{aligned}
\alpha(i,k) &= 2(\Sigma_{z=1}^{i} p_{z-1} + k - 1) \\
\beta(i,h) &= 2(\Sigma_{z=1}^{n} p_z + \Sigma_{z=1}^{i} q_{z-1} \\
&\quad + h - 1) \\
\gamma(i) &= 2(3m + i - 1) \\
\delta(i) &= 2(3m + n + i - 1) \\
\eta(i) &= 2(3m + 2n + i - 1) \\
\theta &= 2(3m + 3n)
\end{aligned}$$

| Condition | Colors |
|---|---|
| $x_i$ is $u^{th}$ occ., $\psi(x_i) = true$ | $\alpha(i,u), \overline{\alpha(i,u)}$ |
| $x_i$ is $u^{th}$ occ., $\psi(x_i) = false, u \neq p_i$ | $\alpha(i,u+1), \overline{\alpha(i,u)}$ |
| $x_i$ is $p_i^{th}$ occ., $\psi(x_i) = false$ | $\delta(i), \overline{\alpha(i,p_i)}$ |
| $\bar{x}_i$ is $v^{th}$ occ., $\psi(x_i) = false$ | $\beta(i,v), \overline{\beta(i,v)}$ |
| $\bar{x}_i$ is $v^{th}$ occ., $\psi(x_i) = true, v \neq q_i$ | $\beta(i,v+1), \overline{\beta(i,v)}$ |
| $\bar{x}_i$ is $q_i^{th}$ occ., $\psi(x_i) = true$ | $\eta(i), \overline{\beta(i,q_i)}$ |

**Fig. 7.** (a) Abbreviations. (b) For each literal in $c_j$, the set of colors added to $h(c_j \xrightarrow{6} Sink)$.

## References


1. Andrew W. Appel and Lal George. Optimal spilling for CISC machines with few registers. In *PLDI*, pages 243–253. ACM Press, 2001.
2. M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. I:interval graphs. In *Discrete Mathematics*, pages 267 – 279. ACM Press, 1992. Special volume (part 1) to mark the centennial of Julius Petersen's "Die theorie der regularen graphs".
3. Preston Briggs. *Register Allocation via Graph Coloring*. PhD thesis, Rice University, 1992.
4. Preston Briggs, Keith Cooper, and Linda Torczon. Coloring register pairs. *ACM Letters on Programming Languages*, 1(1):3–13, 1992.
5. Gregory J. Chaitin, Mark A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, 1981.
6. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4), December 1976.
7. Martin Farach and Vincenzo Liberatore. On local register allocation. In *9th ACM-SIAM symposium on Discrete Algorithms*, pages 564–573. ACM Press, 1998.
8. Christopher Fraser and David Hanson. *A Retargetable C Compiler: Design and Implementation*. Addison-Wesley, 1995.
9. Changqing Fu and Ken D. Wilken. A faster optimal register allocator. In *Internation Symposium on Microarchitecture*, pages 245–256. ACM, 2002.
10. Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 2nd edition, 2004.
11. David W. Goodwin and Ken D. Wilken. Optimal and near-optimal global register allocations using 0-1 integer programming. *SPE*, 26(8):929–965, 1996.
12. Ulrich Hirnschrott, Andreas Krall, and Bernhard Scholz. Graph coloring vs. optimal register allocation for optimizing compilers. In *JMLC*, pages 202–213. Springer, 2003.
13. Timothy Kong and Kent D. Wilken. Precise register allocation for irregular architectures. In *Internation Symposium on Microarchitecture*, pages 297–307. ACM, 1998.
14. Akira Koseki, Hideaki Komatsu, and Toshio Nakatani. Preference-directed graph coloring. In *PLDI*, pages 297–307. ACM, 2002.
15. Brian R. Nickerson. Graph coloring register allocation for processors with multi-register operands. In *PLDI*, pages 40–52, 1990.
16. Johan Runeson and Sven-Olof Nystrom. Retargetable graph-coloring register allocation for irregular architectures. In *SCOPES*, pages 240–254. Springer, 2003.
17. Bernhard Scholz and Erik Eckstein. Register allocation for irregular architectures. In *LCTES/SCOPES*, pages 139–148. ACM, 2002.
18. Michael D. Smith, Norman Ramsey, and Glenn Holloway. A generalized algorithm for graph-coloring register allocation. In *PLDI*, pages 277–288, 2004.
19. Andrew W. Appel with Jens Palsberg. *Modern Compiler Implementation in Java*. Cambridge University Press, 2002.