Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

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, United States

A R T I C L E   I N F O

Article history:
Received 10 August 2007
Received in revised form 28 May 2008
Accepted 30 May 2008
Communicated by J. Diaz

Keywords:
Register allocation
Weighted interval graphs
Graph coloring
Network flow

A B S T R A C T

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 has been 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. We also present a proof of a related result by Stockmeyer: the shipbuilding problem 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 [14]. A register allocator can also be a significant part of the code of a compiler implementation: 10% for lcc [9] 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. [4] 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.

* Corresponding address: UCLA, 4531 K Boelter Hall, 90095 Los Angeles, CA, United States. Tel.: +1 310 825 6320.
E-mail addresses: jkenl@cs.ucla.edu (J.K. Lee), palsberg@cs.ucla.edu (J. Palsberg), fernando@cs.ucla.edu (F.M.Q. Pereira).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.05.025

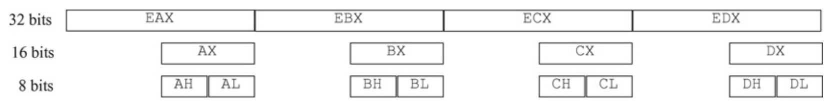

Fig. 1. General purpose registers from the x86 architecture.

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* [19]. For example, Fig. 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 [18] 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. The existing approaches to aliased register allocation are based on heuristics for graph coloring [2,3,13,15,16,19], on integer linear programming [12], or on partitioned Boolean quadratic optimization [11,18]. Integer linear programming and partitioned Boolean quadratic optimization are flexible enough to describe many architecture irregularities but lead 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. Intuitively, the first step transforms a Boolean formula into a graph, the second step transforms the graph into an interval graph, and the third step transforms the interval graph into a program. 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], although until now, to the best of our knowledge, no proof is publicly available. While we can reduce the aligned coloring problem to the unaligned coloring problem (and thereby give a 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 [1]; and the placement of load and store instructions to transfer values to and from memory [8]. Our proof was inspired in part by a paper of Biró, Hujter, and Tuza [1] 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 [7] 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. In Section 7 we prove our main result, and in Section 8 we give a proof of Stockmeyer's result. Our proof of Stockmeyer's result builds upon our main result; we are not aware of any other publicly available proof.

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:

$$\begin{array}{llll} (\text{Statement})\, S & ::= & \texttt{short}\ v = (\text{definition of } v) \\ & \mid & \texttt{long}\ v = (\text{definition of } v) \\ & \mid & = v\ (\text{use of } v). \end{array}$$

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, $i \le j$ and v is used at $S_k, j < k$ [20]. 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* [4].

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 $r_0, \ldots, r_{2K-1}$.
Question: 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 finite 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 *weighted* interval graphs for which each vertex is assigned a weight of either one or two.

For one of our programs P, the interference graph is the intersection graph of the family of live ranges of P. For each short variable, we assign the corresponding live range the weight one, and for each long variable, we assign the corresponding live range the weight two.

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 define a 1-2-*coloring* to be a mapping from vertices to colors which (1) assigns one color to each vertex of weight one and (2) assigns two consecutive colors $i, i+1$ to each vertex of weight two, such that adjacent vertices have colors that are all different. A 1-2-coloring is also known as an *interval coloring* [10, p. 203]; we prefer to use the term 1-2-coloring to avoid confusion with the notion of interval graphs. We define an *aligned* 1-2-*coloring* to be a 1-2-coloring that assigns two aligned colors to each vertex of weight two.

Aligned 1-2-coloring of weighted interval graphs (A12wig):
Instance: a weighed interval graph G and a number $2K$.
Question: does G have an aligned 1-2-coloring with $2K$ colors?

From aligned 1-2-coloring to aliased register allocation. We now present a reduction of aligned 1-2-coloring of weighted interval graphs to aliased register allocation with alignment restrictions. The key step is to show that any weighted 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 the intervals have startpoints and endpoints that are all different, and the startpoints and endpoints of the intervals form the set $1..2q$, where q is the number of intervals.

Chen [5, Lemma 3] and Saha et al. [17, p. 2488] have shown how to convert an interval graph G with q intervals to an isomorphic program-like interval graph $\mathsf{plig}(G)$ in $O(q \log q)$ time.

From a program-like weighted interval graph G with q intervals, we construct a program $\mathsf{prog}(G) = S_1; \ldots; S_{2q}$ as follows. Define

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

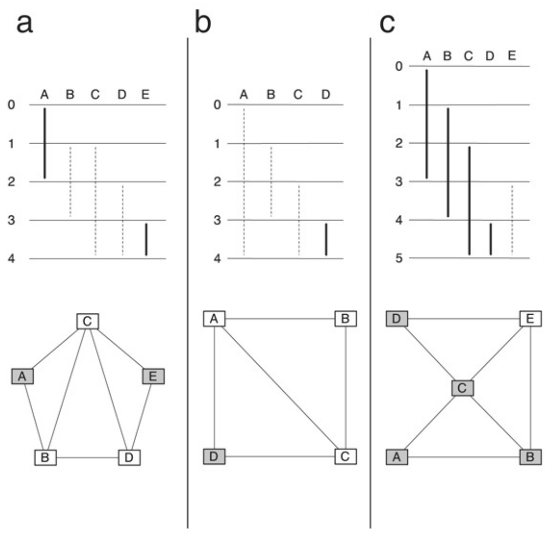

Fig. 2. Weighted interval graphs.

We can represent prog(G) as an array of instructions and then the construction of prog(G) takes $O(q)$ time.

Following Cormen et al. [6], we use the notation $L_1 \leq_P L_2$ to denote that L_1 is polynomial-time reducible to L_2.

Lemma 1. A12WIG $\leq_P$ CARAAR.

Proof. From an instance $(G, 2K)$ of A12WIG we can construct the instance (prog(plig(G)), $2K$) of CARAAR. If G has q intervals, then we can construct prog(plig(G)) in $O(q \log q)$ time. Notice that plig(G) is the interference graph of prog(plig(G)) because for every interval I in plig(G), the live range of v_I in prog(plig(G)) is I. We have that G has an aligned 1-2-coloring with $2K$ colors *if and only if* plig(G) has an aligned 1-2-coloring with $2K$ colors *if and only if* the interference graph of prog(plig(G)) has an aligned 1-2-coloring with $2K$ colors *if and only if* the answer to CARAAR question for (prog(plig(G)), $2K$) is *true*. □

Example. Let us explain why aligned 1-2-coloring is a nontrivial problem. Fig. 2 shows three weighted interval graphs; each graph is displayed both as a collection of intervals and in a conventional way. In the upper part of Fig. 2, we use fat lines to denote intervals of weight two and we use dashed lines to denote intervals of weight one. In the lower part of Fig. 2, we use shaded boxes to denote vertices of weight two, and we use white boxes to denote vertices of weight one.

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 a weighted interval graph does not necessarily have that property. For example, Fig. 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 vertices of weight two 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].) A weighted interval graph does not necessarily have that property. For example, Fig. 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 intervals of weight one and then proceeds to color the intervals of weight two. That does not necessarily lead to an optimal 1-2-coloring. For example, Fig. 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 intervals of weight one come before the intervals of weight two. So, we will get the same suboptimal colorings as above.

Alternatively, we might try to first apply greedy coloring to the intervals of weight two, and then proceed to color the intervals of weight one. That method is not optimal either. For example, Fig. 2(c) shows a graph for which the "weight-two-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

Following Cormen et al. [6], we define a *network* $(V, E, Source, Sink, c)$ to be a directed graph with vertex set V, edge set E, distinguished vertices *Source* and *Sink*, and a capacity function $c : E \to Nat$ where *Nat* denotes the nonnegative natural

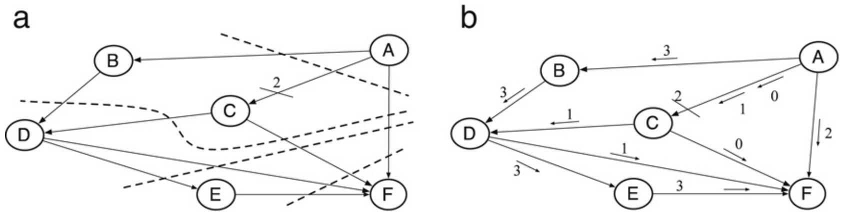

Fig. 3. (a) Simple graph; (b) colored flow.

numbers, such that every vertex is on some path from *Source* to *Sink*. A *cut* (S, T) of a network $(V, E, Source, Sink, c)$ is a partition of V into S and $T = V \setminus S$ such that $Source \in S$ and $Sink \in T$. The capacity of a cut (S, T), written $c(S, T)$, is given by the formula:

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

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.

For an acyclic network $(V, E, Source, Sink, c)$, a *straight cut* (S, T) is a cut for which $S = \{v_1, v_2, \ldots, v_i\}$, $T = \{v_{i+1}, v_{i+2}, \ldots v_n\}$, and $v_1, \ldots, v_n$ is a topological ordering of V. A *simple graph* is an acyclic network for which all straight cuts have the same capacity.

Fig. 3 (a) shows a simple graph. Each dashed line marks a straight cut. Each unlabeled edge has unit capacity while each remaining edge is marked with a small bar and its capacity.

Lemma 2. *Let $G = (V, E, Source, Sink, c)$ be an acyclic network. All straight cuts of G 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. $\Rightarrow$) Suppose all straight cuts of G have the same capacity. Let $v_1, \ldots, v_n$ be a topological ordering of V. Notice that $v_1 = Source$ and $v_n = Sink$. Let $i \in 2..n - 1$. We need to show $\sum_{(u,v_i)\in E} c(u, v_i) = \sum_{(v_i,w)\in E} c(v_i, w)$. Define

$$S_1 = \{v_1, \ldots, v_{i-1}\} \qquad T_1 = \{v_i, \ldots, v_n\}$$
$$S_2 = \{v_1, \ldots, v_i\} \qquad T_2 = \{v_{i+1}, \ldots, v_n\}.$$

The edges that cross the cut from S_1 to T_1 but not from S_2 to T_2 are the set of edges of the form (u, v_i). The edges that cross the cut from S_2 to T_2 but not from S_1 to T_1 are the set of edges of the form (v_i, w). We have that the two straight cuts (S_1, T_1) and (S_2, T_2) have the same capacity so $\sum_{(u,v_i)\in E} c(u, v_i) = \sum_{(v_i,w)\in E} c(v_i, w)$, as desired.

$\Leftarrow$) Suppose $\forall v \in V \setminus \{Source, Sink\} : \sum_{(u,v)\in E} c(u, v) = \sum_{(v,w)\in E} c(v, w)$. We will show that for every straight cut (S, T), we have $c(S, T) = c(\{Source\}, V \setminus \{Source\})$. Let (S, T) be a straight cut and let $v_1, \ldots, v_n$ be a topological ordering of V such that $S = \{v_1, v_2, \ldots, v_{i'}\}$ and $T = \{v_{i'+1}, v_{i'+2}, \ldots v_n\}$. For every $i \in 1..n - 1$, define $S_i = \{v_1, v_2, \ldots, v_i\}$, $T_i = \{v_{i+1}, v_{i+2}, \ldots v_n\}$. Notice that, for every $i \in 1..n - 1$, (S_i, T_i) is a straight cut. Notice also that $(S, T) = (S_{i'}, T_{i'})$.

Let us show by induction that for every $i \in 1..n - 1$, $c(S_i, T_i) = c(S_1, T_1)$. In the case of $i = 1$, the property is immediate. In the induction step, we assume $c(S_i, T_i) = c(S_1, T_1)$, and by reasoning similar to what we used on two straight cuts in the other half of the proof of the lemma, we have that $c(S_i, T_i) = c(S_{i+1}, T_{i+1})$. From transitivity, we conclude that $c(S_{i+1}, T_{i+1}) = c(S_i, T_i) = c(S_1, T_1)$.

Finally we have $(S_1, T_1) = (\{Source\}, V \setminus \{Source\})$ so every straight cut has a capacity equal to the capacity of $(\{Source\}, V \setminus \{Source\})$, so all straight cuts have the same capacity. □

For a network $(V, E, Source, Sink, c)$, a *flow* is a function $f : E \to Nat$, such that

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

For a simple graph, the capacity function is a flow. To see that, notice that the capacity function trivially satisfies the Capacity constraint and that Lemma 2 guarantees that the capacity function satisfies the Conservation constraint.

We say that an element of $0..K - 1$ is a *color*. For a function $h : E \to 2^{0..K-1}$, the set of colors used to color edges across a cut (S, T), written $h(S, T)$, is given by the formula:

$$h(S,T) = \bigcup_{(u,v)\in E, u\in S, v\in T} h(u,v).$$

We define a *colored flow* for a simple graph $(V, E, Source, Sink, c)$ with every straight cut of capacity K as a function $h : E \to 2^{0..K-1}$ such that for every straight cut (S, T), we have $h(S, T) = 0..K - 1$. Thus, for any straight cut, every color is used exactly once in the coloring of the edges that cross the cut. Every colored flow h satisfies that for all $e \in E$, $|h(e)| = c(e)$. To see that, notice that every edge e crosses some straight cut (S, T) and that $|h(S, T)| = c(S, T)$. Fig. 3 (b) shows an example of a colored flow.

Lemma 3. *Let $G = (V, E, Source, Sink, c)$ be a simple graph with every straight cut of capacity K. A function $h : E \to 2^{0..K-1}$ is a colored flow if and only if $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$, and there exists a straight cut (S, T) such that $h(S, T) = 0..K - 1$.*

Proof. $\Rightarrow$) Suppose h is a colored flow. First, let us show that $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$. Let $v_1, \ldots, v_n$ be a topological ordering of V. Notice that $v_1 = Source$ and $v_n = Sink$. Let $i \in 2..n - 1$. We need to show $\cup_{(u,v_i)\in E} h(u, v_i) = \cup_{(v_i,w)\in E} h(v_i, w)$. Define

$$S_1 = \{v_1, \ldots, v_{i-1}\} \qquad T_1 = \{v_i, \ldots, v_n\}$$
$$S_2 = \{v_1, \ldots, v_i\} \qquad T_2 = \{v_{i+1}, \ldots, v_n\}.$$

The edges that cross the cut from S_1 to T_1 but not from S_2 to T_2 are the set of edges of the form (u, v_i). The edges that cross the cut from S_2 to T_2 but not from S_1 to T_1 are the set of edges of the form (v_i, w). From h being a colored flow we have that the same set of colors is used to color both the edges that cross (S_1, T_1) and the edge that cross (S_2, T_2). So, the set of colors used for edges of the form (u, v_i) must be the same set of colors used for edges of the form (v_i, w).

Second, for the straight cut $(\{Source\}, V \setminus \{Source\})$ we have from h being a colored flow that every color is used exactly once in the coloring of edges that cross the cut.

$\Leftarrow$) Suppose h is a function such that $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$, and there exists a straight cut (S, T) such that $h(S, T) = 0..K - 1$. We must show that for any straight cut (S, T), we have $h(S, T) = 0..K - 1$. It is sufficient to show that for every straight cut (S, T), we have $h(S, T) = h(\{Source\}, V \setminus \{Source\})$. Let (S, T) be a straight cut and let $v_1, \ldots, v_n$ be a topological ordering of V such that $S = \{v_1, v_2, \ldots, v_{i'}\}$ and $T = \{v_{i'+1}, v_{i'+2}, \ldots v_n\}$. For every $i \in 1..n - 1$, define $S_i = \{v_1, v_2, \ldots, v_i\}$, $T_i = \{v_{i+1}, v_{i+2}, \ldots v_n\}$. Notice that, for every $i \in 1..n - 1$, (S_i, T_i) is a straight cut. Notice also that $(S, T) = (S_{i'}, T_{i'})$.

Let us show by induction that for every $i \in 1..n - 1$, $h(S_i, T_i) = h(S_1, T_1)$. In the case of $i = 1$, the property is immediate. In the induction step, we assume $h(S_i, T_i) = h(S_1, T_1)$, and by reasoning similar to what we used on two straight cuts in the other half of the proof of the lemma, we have that $h(S_i, T_i) = h(S_{i+1}, T_{i+1})$. From transitivity, we conclude that $h(S_{i+1}, T_{i+1}) = h(S_i, T_i) = h(S_1, T_1)$.

Finally, we have $(S_1, T_1) = (\{Source\}, V \setminus \{Source\})$ so the set of colors used for the edges across any given straight cut is the set same used for the edges across $(\{Source\}, V \setminus \{Source\})$. □

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

For a natural number B, a network $(V, E, Source, Sink, c)$ is *B-bounded* if for all $e \in E : c(e) \le B$.

ALIGNED COLORED FLOW (ACFLOW):
Instance: a 6-bounded simple graph G.
Question: does G have an aligned colored flow?

It is straightforward to see that ACFLOW is in NP. Our reduction from 3-SAT to ACFLOW (Section 6) produces a 6-bounded simple graph.

5. From aligned colored flow to aligned 1-2-coloring

In this section we present a reduction of the aligned colored flow problem to aligned 1-2-coloring of weighted interval graphs.

Lemma 4. ACFLOW $\le_P$ A12WIG.

Proof. Let $G = (V, E, Source, Sink, c)$ be an instance of ACFLOW, and let $2K$ be the capacity of all straight cuts of G. From G we construct a weighted interval graph $\mathsf{wig}(G)$ in the following way. Let $v_1, \ldots, v_n$ be a topological ordering of V. The intervals of $\mathsf{wig}(G)$ are defined as follows. For each $(v_i, v_j) \in E$ such that $2 \le c(v_i, v_j)$, we create one interval $[i, j[$ of weight two, and we create $c(v_i, v_j) - 2$ intervals $[i, j[$ of weight one. For each $(v_i, v_j) \in E$ such that $c(v_i, v_j) = 1$, we create one interval $[i, j[$ of weight one.

From G we can construct the instance $(\mathsf{wig}(G), 2K)$ of A12WIG. We can construct the topological ordering of V in $O(|V|+|E|)$ time and we can construct the intervals of $\mathsf{wig}(G)$ in $O(|E|)$ time, so we can construct $(\mathsf{wig}(G), 2K)$ in $O(|V|+|E|)$ time.

We will show that G has an aligned colored flow if and only if $\mathsf{wig}(G)$ has an aligned 1-2-coloring with $2K$ colors.

$\Rightarrow$) Suppose G has an aligned colored flow h. We can then define a mapping κ that assigns colors to each vertex of $\mathsf{wig}(G)$ as follows. For each $(v_i, v_j) \in E$ such that $2 \le c(v_i, v_j)$, we have that $h(v_i, v_j)$ contains two aligned colors; assign those two colors to the interval of weight two created from (v_i, v_j), and assign each of the rest of the colors in $h(v_i, v_j)$ to each of the intervals of weight one created from (v_i, v_j). For each $(v_i, v_j) \in E$ such that $c(v_i, v_j) = 1$, assign the one color in $h(v_i, v_j)$ to the interval of weight one created from (v_i, v_j). We need to show that adjacent vertices in $\mathsf{wig}(G)$ have colors that are all different. Suppose we have two adjacent intervals I_1, I_2 in $\mathsf{wig}(G)$, 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 $\mathsf{wig}(G)$. Define $S = \{ v_i \in V \mid i < p \}$ and define $T = \{ v_i \in V \mid i > p \}$. We have that (S, T) is a straight cut. 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 $\mathsf{wig}(G)$ has an aligned 1-2-coloring κ. We can then define a mapping $h : E \to 2^{0..2K-1}$ as follows. For each $(v_i, v_j) \in E$, let $h(v_i, v_j)$ be the union of the colors assigned by κ to the intervals in $\mathsf{wig}(G)$ created from (v_i, v_j). We need to show that h is an aligned colored flow.

Let us first show that h is a colored flow. From Lemma 3 we have that we must show that (1) $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$, and (2) there exists a straight cut (S, T) such that $\cup_{(u,v)\in E, u\in S, v\in T} h(u, v) = 0..2K - 1$.

To prove (1), let v_j be a vertex in $V \setminus \{Source, Sink\}$. We have that

$$\begin{aligned} C_1 &= (\{ v_i \mid i < j \}, \{ v_k \mid j \le k \}) \\ C_2 &= (\{ v_i \mid i \le j \}, \{ v_k \mid j < k \}) \end{aligned}$$

are straight cuts of G and hence both of capacity $2K$. Next define E_1 to be the set of edges in E of the form (v_i, v_j), and define E_2 to be the set of edges in E of the form (v_j, v_k). 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 (2), let us examine the edges that traverse the straight cut $(\{Source\}, V \setminus \{Source\})$, which by assumption has capacity $2K$. From those edges, we create intervals in $\mathsf{wig}(G)$ that all overlap and hence get all different colors by κ.

Finally, we have that h is aligned because the construction of $\mathsf{wig}(G)$ ensures that for any e for which we have $2 \le c(e)$, one vertex of weight two is created; the construction of h then ensures that the aligned colors assigned by κ to that interval will be two of the colors assigned by h to e. $\square$

6. From 3-SAT to aligned colored flow

The 3-SAT problem is to decide whether a given Boolean formula in conjunctive normal form (CNF) is satisfiable. In this section we present a reduction of 3-SAT to the aligned colored flow problem. Let

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

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 $\bar{x}_i$. For convenience, we define $p_0 = 0$ and $q_0 = 0$.

From a Boolean formula F in CNF we construct a simple graph $\mathsf{sg}(F) = (V, E, Source, Sink, c)$. The graph is akin to the graph used by Even, Itai and Shamir [7, Section 4] in their proof of NP-completeness for the multicommodity flow problem. The vertices of $\mathsf{sg}(F)$ are:

$$\begin{aligned} V = {} & \{Source, Sink, Q, R\} \cup \\ & \{ c_j \mid 1 \le j \le m \} \cup \\ & \{ a_i, b_i \mid 1 \le i \le n \} \cup \\ & \{ s_{i0}, s_{iy}, s_{i(p_i+1)}, x_{iy}, x_{i(p_i+1)} \mid 1 \le i \le n \wedge 1 \le y \le p_i \} \cup \\ & \{ \quad \bar{s}_{iz}, \bar{s}_{i(q_i+1)}, \bar{x}_{iz}, \bar{x}_{i(q_i+1)} \mid 1 \le i \le n \wedge 1 \le z \le q_i \}. \end{aligned}$$

For convenience, we will some times use the alias a_{n+1} for R. Fig. 4 shows a listing of the edges in E and their capacities. Fig. 4 also shows a set of colors for each edge, using the abbreviations in Fig. 5(a); we will need that later.

Fig. 6 illustrates the graph constructed from the formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. We use c_1 to denote $(x_1 \vee x_2 \vee x_3)$ and we use c_2 to denote $(\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Let us now give an intuitive explanation of the construction of the graph. For each variable x_i, we construct a lobe. First we add the edges $(x_{iy} \xrightarrow{1} x_{i(y+1)})$ to form an upper path and we add the edges $(\bar{x}_{iz} \xrightarrow{1} \bar{x}_{i(z+1)})$ to form a lower path. Then 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_{iy}, we create an edge $(Source \xrightarrow{2} s_{iy})$ with a capacity two. Likewise for all the s_{i0} and the $\bar{s}_{ih}$ vertices. For each $s_{i(p_i+1)}$ and $\bar{s}_{i(q_i+1)}$ we add the edges $(Source \xrightarrow{2} s_{i(p_i+1)})$ and $(Source \xrightarrow{2} \bar{s}_{i(q_i+1)})$ each with capacity two. Next we will add an edge $(Source \xrightarrow{2} Q)$ also with capacity of two. In total we have made a capacity of $2(3m + 3n + 1)$ leaving the source. We want to make $\mathsf{sg}(F)$ 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 $(c_j \xrightarrow{6} Sink)$ with a capacity

	Edge	Color		Edge	Color
1	$x_{iy} \xrightarrow{1} x_{i(y+1)}$	$\psi(x_i)\ ?\ \alpha(i,y) : \theta$	18	$s_{i0} \xrightarrow{1} Sink$	$\overline{\gamma(i)}$
2	$\bar{x}_{iz} \xrightarrow{1} \bar{x}_{i(z+1)}$	$\psi(x_i)\ ?\ \theta : \beta(i,z)$	19	$s_{i0} \xrightarrow{1} a_i$	$\gamma(i)$
3	$a_i \xrightarrow{1} x_{i1}$	$\psi(x_i)\ ?\ \gamma(i) : \theta$	20	$b_i \xrightarrow{1} Sink$	$\psi(x_i)\ ?\ \delta(i) : \eta(i)$
4	$a_i \xrightarrow{1} \bar{x}_{i1}$	$\psi(x_i)\ ?\ \theta : \gamma(i)$	21	$s_{iy} \xrightarrow{1} x_{iy}$	$\alpha(i,y)$
5	$x_{i(p_i+1)} \xrightarrow{1} b_i$	$\psi(x_i)\ ?\ \delta(i) : \theta$	22	$s_{i(p_i+1)} \xrightarrow{1} x_{i(p_i+1)}$	$\delta(i)$
6	$\bar{x}_{i(q_i+1)} \xrightarrow{1} b_i$	$\psi(x_i)\ ?\ \theta : \eta(i)$	23	$\bar{s}_{iz} \xrightarrow{1} \bar{x}_{iz}$	$\beta(i,z)$
7	$Source \xrightarrow{2} s_{iy}$	$\alpha(i,y), \overline{\alpha(i,y)}$	24	$\bar{s}_{i(q_i+1)} \xrightarrow{1} \bar{x}_{i(q_i+1)}$	$\eta(i)$
8	$Source \xrightarrow{2} s_{i(p_i+1)}$	$\delta(i), \overline{\delta(i)}$	25	$s_{i(p_i+1)} \xrightarrow{1} Sink$	$\overline{\delta(i)}$
9	$Source \xrightarrow{2} \bar{s}_{iz}$	$\beta(i,z), \overline{\beta(i,z)}$	26	$\bar{s}_{i(q_i+1)} \xrightarrow{1} Sink$	$\overline{\eta(i)}$
10	$Source \xrightarrow{2} \bar{s}_{i(q_i+1)}$	$\eta(i), \overline{\eta(i)}$	27	$s_{iy} \xrightarrow{1} c_j$ if $\mathrm{occ}(y, x_i, c_j)$	$\overline{\alpha(i,y)}$
11	$Source \xrightarrow{2} s_{i0}$	$\gamma(i), \overline{\gamma(i)}$	28	$\bar{s}_{iz} \xrightarrow{1} c_j$ if $\mathrm{occ}(z, \bar{x}_i, c_j)$	$\overline{\beta(i,z)}$
12	$Source \xrightarrow{2} Q$	$\theta, \bar{\theta}$	29	$x_{i(y+1)} \xrightarrow{1} c_j$ if $\mathrm{occ}(y, x_i, c_j)$, $(y \neq p_i)$	$\psi(x_i)\ ?\ \alpha(i,y) : \alpha(i,y+1)$
13	$c_j \xrightarrow{6} Sink$	See Figure 5(b)	30	$x_{i(p_i+1)} \xrightarrow{1} c_j$ if $\mathrm{occ}(p_i, x_i, c_j)$	$\psi(x_i)\ ?\ \alpha(i,p_i) : \delta(i)$
14	$R \xrightarrow{2} Sink$	$\theta, \bar{\theta}$	31	$\bar{x}_{i(z+1)} \xrightarrow{1} c_j$ if $\mathrm{occ}(z, \bar{x}_i, c_j)$, $(z \neq q_i)$	$\psi(x_i)\ ?\ \beta(i,z+1) : \beta(i,z)$
15	$Q \xrightarrow{1} R$	$\bar{\theta}$	32	$\bar{x}_{i(q_i+1)} \xrightarrow{1} c_j$ if $\mathrm{occ}(q_i, \bar{x}_i, c_j)$	$\psi(x_i)\ ?\ \eta(i) : \beta(i,q_i)$
16	$b_i \xrightarrow{1} a_{i+1}$	θ	33	$x_{i1} \xrightarrow{1} Sink$	$\psi(x_i)\ ?\ \gamma(i) : \alpha(i,1)$
17	$Q \xrightarrow{1} a_1$	θ	34	$\bar{x}_{i1} \xrightarrow{1} Sink$	$\psi(x_i)\ ?\ \beta(i,1) : \gamma(i)$

Fig. 4. The edges of $\mathtt{sg}(F)$. We have $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq y \leq p_i$, $1 \leq z \leq q_i$. The notation $\psi(x_i)?d_T : d_F$ 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. An expression $\mathrm{occ}(y, x_i, c_j)$ means that the yth occurrence of x_i appears in c_j.

a

$$\alpha(i,y) = 2(\Sigma_{j=1}^{i} p_{j-1} + y - 1)$$
$$\beta(i,z) = 2(\Sigma_{j=1}^{n} p_j + \Sigma_{j=1}^{i} q_{j-1} + z - 1)$$
$$\gamma(i) = 2(3m + i - 1)$$
$$\delta(i) = 2(3m + n + i - 1)$$
$$\eta(i) = 2(3m + 2n + i - 1)$$
$$\theta = 2(3m + 3n)$$

b

	Condition	Colors
i	$\mathrm{occ}(y, x_i, c_j) \wedge \psi(x_i) = true$	$\alpha(i,y), \overline{\alpha(i,y)}$
ii	$\mathrm{occ}(y, x_i, c_j) \wedge \psi(x_i) = false \wedge y \neq p_i$	$\alpha(i,y+1), \overline{\alpha(i,y)}$
iii	$\mathrm{occ}(p_i, x_i, c_j) \wedge \psi(x_i) = false$	$\delta(i), \overline{\alpha(i,p_i)}$
iv	$\mathrm{occ}(z, \bar{x}_i, c_j) \wedge \psi(x_i) = false$	$\beta(i,z), \overline{\beta(i,z)}$
v	$\mathrm{occ}(z, \bar{x}_i, c_j) \wedge \psi(x_i) = true \wedge z \neq q_i$	$\beta(i,z+1), \overline{\beta(i,z)}$
vi	$\mathrm{occ}(q_i, \bar{x}_i, c_j) \wedge \psi(x_i) = true$	$\eta(i), \overline{\beta(i,q_i)}$

Fig. 5. (a) Abbreviations. (b) For each literal in c_j, the two colors in $h(c_j \xrightarrow{6} Sink)$.

of six and finally we add $(R \xrightarrow{2} Sink)$ with a capacity of two. Now all that remains to make the graph simple is to connect the Q, s_{iy}, $s_{i(p_i+1)}$, $\bar{s}_{iz}$ and $\bar{s}_{i(q_i+1)}$ vertices to R, c_j, and *Sink*.

We will first add edges to send the current excess capacity at Q to R. We will add a direct edge $(Q \xrightarrow{1} R)$ to get one unit of capacity to R. To get the other unit to R, we will connect the lobes serially, by adding the edges $(b_i \xrightarrow{1} a_{i+1})$. Finally, we add $(Q \xrightarrow{1} a_1)$, resulting in a path to send the other unit of capacity to R 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 $(s_{i0} \xrightarrow{1} Sink)$, $(s_{i0} \xrightarrow{1} a_i)$, and $(b_i \xrightarrow{1} Sink)$. This results in a current total of $2n + 2$ units of capacity reaching *Sink*, and that the vertices on the lobe are balanced.

We will now connect the remaining $s_{iy}, s_{i(p_i+1)}, \bar{s}_{iz}$ and $\bar{s}_{i(q_i+1)}$ vertices to the c_j vertices and *Sink*. We add the edges $(s_{iy} \xrightarrow{1} x_{iy})$, $(s_{i(p_i+1)} \xrightarrow{1} x_{i(p_i+1)})$, $(\bar{s}_{iz} \xrightarrow{1} x_{iz})$ and $(\bar{s}_{i(q_i+1)} \xrightarrow{1} \bar{x}_{ih})$ 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_{iy} and $\bar{s}_{iz}$ vertices will be sent to some c_j vertex while those of $s_{i(p_i+1)}$ and $\bar{s}_{i(p_i+1)}$ will be sent directly to *Sink*. We add the edges $(s_{i(p_i+1)} \xrightarrow{1} Sink)$ and $(\bar{s}_{i(q_i+1)} \xrightarrow{1} Sink)$, which now results in an additional $2n$ units of capacity reaching *Sink* for a running total of $4n + 2$. For the remaining vertices, we add an edge $(s_{iy} \xrightarrow{1} c_j)$ if the yth occurrence of x_i appears in c_j. For the $\bar{s}_{iz}$ 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 $x_{iy}, x_{i(p_i+1)}, \bar{x}_{iz}$ and $\bar{x}_{i(q_i+1)}$ vertices and we will have a simple graph. We add the edges $(x_{i(y+1)} \xrightarrow{1} c_j)$ if the yth appearance of x_i occurs in c_j as well as

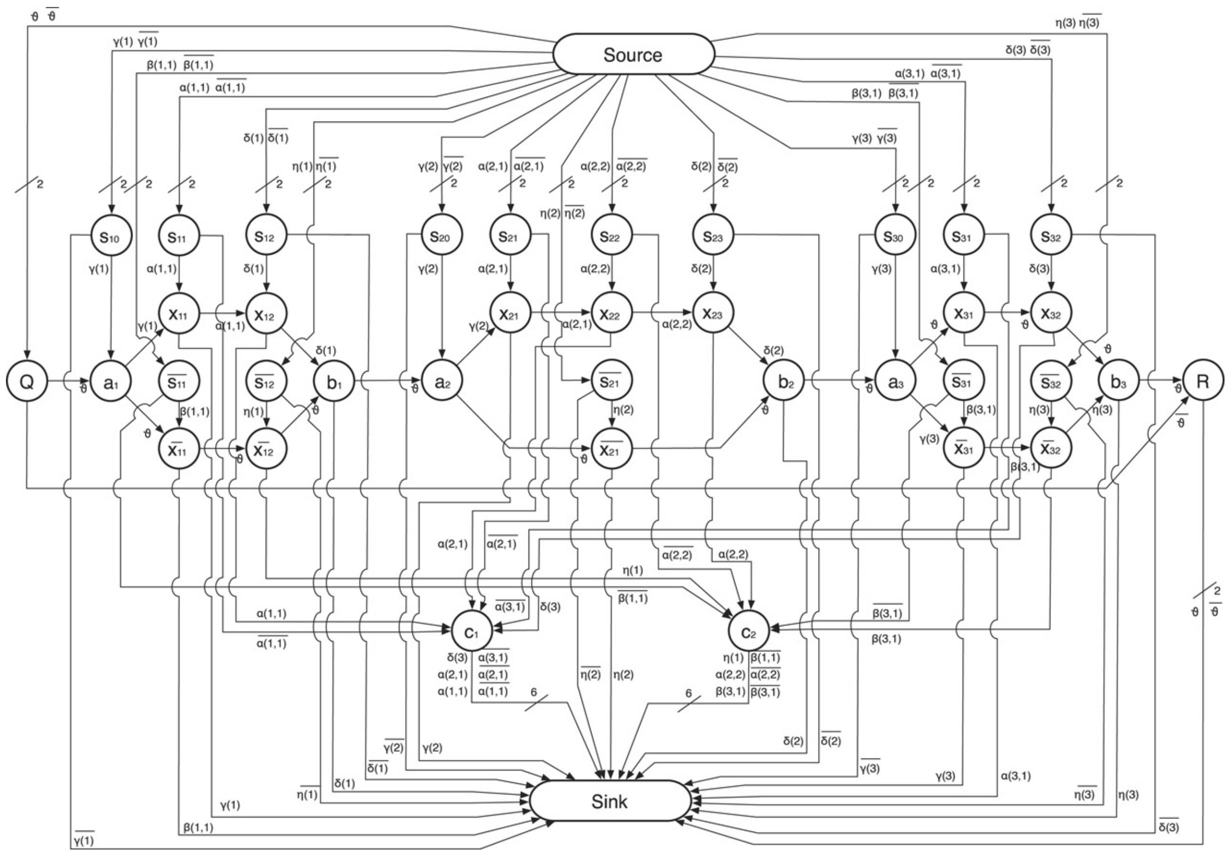

Fig. 6. Construction of a simple graph from $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Edges with a labeled capacity require aligned colors. The edge labels are explained in Fig. 5(a). The coloring corresponds to the Boolean assignment ψ where $\psi(x_1) = \psi(x_2) = \mathit{true}$ and $\psi(x_3) = \mathit{false}$.

$(x_{i(z+1)} \xrightarrow{1} c_j)$ if the zth 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 5. 3-SAT $\leq_P$ ACFlow.

Proof. From an instance F of 3-SAT with n variables and m clauses, we can construct the instance $\mathtt{sg}(F)$ of ACFlow. The graph $\mathtt{sg}(F)$ has $7m+7n+4$ vertices and $16m+17n+4$ edges and we can construct $\mathtt{sg}(F)$ in polynomial time. It is straightforward to check that $\mathtt{sg}(F)$ is a 6-bounded acyclic network. We must also show that all straight cuts of $\mathtt{sg}(F)$ have the same capacity. From Lemma 2 we have that we must show $\forall v \in V \setminus \{\mathit{Source}, \mathit{Sink}\} : \sum_{(u,v)\in E} c(u, v) = \sum_{(v,w)\in E} c(v, w)$. We will use the terminology that for a vertex $v \in V \setminus \{\mathit{Source}, \mathit{Sink}\}$, the *in-capacity* of v is $\sum_{(u,v)\in E} c(u, v)$, and the *out-capacity* of v is $\sum_{(v,w)\in E} c(v, w)$. We will examine each vertex in $V \setminus \{\mathit{Source}, \mathit{Sink}\}$ in turn. From Fig. 4 we see that, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq y \leq p_i$, $1 \leq z \leq q_i$:

Node	in-capacity	using rules	out-capacity	using rules
Q	2	12	2	15, 17
R	2	15, 16	2	14
c_j	6	27–32	6	13
a_i	2	16, 17, 19	2	3, 4
b_i	2	5, 6	2	16, 20
s_{i0}	2	11	2	18, 19
s_{iy}	2	7	2	21, 27
$s_{i(p_i+1)}$	2	8	2	22, 25
x_{iy}	2	1, 3, 21	2	1, 29, 33
$x_{i(p_i+1)}$	2	1, 22	2	5, 30
$\bar{s}_{iz}$	2	9	2	23, 28
$\bar{s}_{i(q_i+1)}$	2	10	2	24, 26
$\bar{x}_{iz}$	2	1, 2, 23	2	2, 31, 34
$\bar{x}_{i(q_i+1)}$	2	2, 24	2	6, 32

So, $\mathsf{sg}(F)$ is a simple graph.

We will show that F is satisfiable if and only if $\mathsf{sg}(F)$ has an aligned colored flow.

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

$\Rightarrow$) Suppose we have a Boolean assignment ψ for the variables in F which satisfies F. Let $h : E \to 2^{0..2K-1}$ be defined by mapping each edge to the set of colors given in Fig. 4, using the abbreviations in Fig. 5(a). We need to show that h is an aligned colored flow.

From Lemma 3 we have that we can show that h is a colored flow by showing that: (1) $\forall v \in V \setminus \{Source, Sink\} : \cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$, and (2) there exists a straight cut (S, T) such that $\cup_{(u,v)\in E, u\in S, v\in T} h(u, v) = 0..2K - 1$.

To prove (1), we first note that it is straightforward to check that the sets of edges and colors defined in Fig. 4 are the same as the sets of edges and colors shown in Figs. 8–10 in the Appendix. Then, for each vertex $v \in V \setminus \{Source, Sink\}$, we examine the corresponding illustration in one of the figures Figs. 8–10, and in each case it is straightforward to check that $\cup_{(u,v)\in E} h(u, v) = \cup_{(v,w)\in E} h(v, w)$.

To prove (2), we will focus on the straight cut $(Source, V \setminus Source)$ and show that $\cup_{(u,v)\in E, u\in Source, v\in V\setminus Source} h(u, v) = 0..2K-1$. From Fig. 4, rules 7–12, we can see that six forms of edges cross the straight cut. Notice that $\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} q_i = 3m$. From that observation and the abbreviations in Fig. 5(a), it is straightforward to see that the colors of those edges are:

$$
\begin{aligned}
&\bigcup\{\, \alpha(i, y), \overline{\alpha(i, y)} \mid 1 \le i \le n \wedge 1 \le y \le p_i \,\} = (0 \,..\, 2(\Sigma_{i=1}^{n} p_i) - 1)\\
&\bigcup\{\, \beta(i, z), \overline{\beta(i, z)} \mid 1 \le i \le n \wedge 1 \le z \le q_i \,\} = (2(\Sigma_{i=1}^{n} p_i) \,..\, 6m - 1)\\
&\bigcup\{\, \gamma(i), \overline{\gamma(i)} \mid 1 \le i \le n \,\} = (6m \,..\, 6m + 2n - 1)\\
&\bigcup\{\, \delta(i), \overline{\delta(i)} \mid 1 \le i \le n \,\} = (6m + 2n \,..\, 6m + 4n - 1)\\
&\bigcup\{\, \eta(i), \overline{\eta(i)} \mid 1 \le i \le n \,\} = (6m + 4n \,..\, 6m + 6n - 1)\\
&\{\, \delta, \overline{\delta} \,\} = (6m + 6n \,..\, 6m + 6n + 1).
\end{aligned}
$$

We conclude that the colors of the edges that cross the straight cut $(Source, V \setminus Source)$ form the interval $0 \,..\, 6m + 6n + 1 = 0 \,..\, 2K - 1$.

Finally, we must show that h is an aligned colored flow, that is, we must show that for every edge e, if $2 \le c(e)$, then $\exists i : 0 \le i \le K - 1 \wedge \{2i, 2i + 1\} \subseteq h(e)$. We can easily verify that by inspection of Fig. 4.

$\Leftarrow$) Suppose $\mathsf{sg}(F)$ has an aligned colored flow $h : E \to 2^{0..2K-1}$. We can then define a Boolean assignment ψ for the variables of F:

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

Let $j \in 1..m$. We will show that ψ satisfies c_j. Let us first prove a property of h:

– Claim 1: Either there exists i, y such that $h(Source \xrightarrow{2} s_{iy}) \subseteq h(c_j \xrightarrow{6} Sink)$, or there exists i, z such that $h(Source \xrightarrow{2} \bar{s}_{iz}) \subseteq h(c_j \xrightarrow{6} Sink)$.

Proof of Claim 1. The set $h(c_j \xrightarrow{6} Sink)$ contains two aligned colors $d, \bar{d}$. The edges crossing the straight cut $(\{Source\}, V \setminus \{Source\})$ are of the six forms: (i) $Source \xrightarrow{2} Q$, (ii) $Source \xrightarrow{2} s_{i0}$, (iii) $Source \xrightarrow{2} s_{i(p_i+1)}$, (iv) $Source \xrightarrow{2} \bar{s}_{i(q_i+1)}$, (v) $Source \xrightarrow{2} s_{iy}$, (vi) $Source \xrightarrow{2} \bar{s}_{iz}$. 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). From Lemma 3 and the edges $Source \xrightarrow{2} Q \xrightarrow{1} R \xrightarrow{2} Sink$, we have $h(Source \xrightarrow{2} Q) = h(R \xrightarrow{2} Sink)$. The straight cut $(V \setminus \{Sink\}, \{Sink\})$ is crossed by the edges $R \xrightarrow{2} Sink$ and $c_j \xrightarrow{6} Sink$, hence those edges must have colors that are all different. We conclude $h(Source \xrightarrow{2} Q) \not\subseteq h(c_j \xrightarrow{6} Sink)$. Consider then case (ii). The straight cut $(V \setminus \{Sink\}, \{Sink\})$ is crossed by the edges $s_{i0} \xrightarrow{1} Sink$ and $c_j \xrightarrow{6} Sink$, hence those edges must have colors that are all different. From the edges $Source \xrightarrow{2} s_{i0} \xrightarrow{1} Sink$ and Lemma 3 we have that $h(s_{i0} \xrightarrow{1} Sink) \subseteq h(Source \xrightarrow{2} s_{i0})$. We conclude $h(Source \xrightarrow{2} s_{i0}) \not\subseteq h(c_j \xrightarrow{6} Sink)$. Consider then cases (iii)–(iv): the proofs are similar to the proof of (ii), we omit the details.

From Claim 1 we have two cases. We will consider each case in turn.

First, suppose there exists i, y such that $h(Source \xrightarrow{2} s_{iy}) \subseteq h(c_j \xrightarrow{6} Sink)$. We begin by proving three properties of h:

– Claim 2: x_i appears in c_j.

– Claim 3: $h(Source \xrightarrow{2} Q) = h(Q \xrightarrow{1} R) \cup h(b_i \xrightarrow{1} a_{i+1})$.

– Claim 4: $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \subseteq h(Source \xrightarrow{2} s_{iy})$.

Proof of Claim 2. The kth occurrence of x_i is in c_r for some r, and $\mathtt{sg}(F)$ contains the edges $s_{iy} \xrightarrow{1} c_r$ and $x_{i(y+1)} \xrightarrow{1} c_r$. We must show $j = r$. Suppose $j \neq r$. The straight cut $(V \setminus \{c_r, Sink\}, \{c_r, Sink\})$ is crossed by the edges $c_j \xrightarrow{6} Sink$ and $s_{iy} \xrightarrow{1} c_r$, hence those edges must have colors that are all different. However, from Lemma 3 and $h(Source \xrightarrow{2} s_{iy}) \subseteq h(c_j \xrightarrow{6} Sink)$, we have that $h(s_{iy} \xrightarrow{1} c_r) \subseteq h(Source \xrightarrow{2} s_{iy}) \subseteq h(c_j \xrightarrow{6} Sink)$, a contradiction. We conclude $j = r$.

Proof of Claim 3. From Lemma 3, $h(Source \xrightarrow{2} Q) = h(Q \xrightarrow{1} R) \cup h(Q \xrightarrow{1} a_1)$. The color in $h(Q \xrightarrow{1} a_1)$ takes a path from Q to R that contains the edge $b_i \xrightarrow{1} a_{i+1}$. From Lemma 3 we have that $h(Q \xrightarrow{1} a_1) = h(b_i \xrightarrow{1} a_{i+1})$. From $h(Source \xrightarrow{2} Q) = h(Q \xrightarrow{1} R) \cup h(Q \xrightarrow{1} a_1)$ and $h(Q \xrightarrow{1} a_1) = h(b_i \xrightarrow{1} a_{i+1})$ we have $h(Source \xrightarrow{2} Q) = h(Q \xrightarrow{1} R) \cup h(b_i \xrightarrow{1} a_{i+1})$.

Proof of Claim 4. From Claim 2 we have that x_i appears in c_j. From Lemma 3 we have that we can find a color d such that $h(Source \xrightarrow{2} s_{iy}) = \{d, \bar{d}\}$ and $h(s_{iy} \xrightarrow{1} c_j) = \{d\}$. We have that the literals in clause c_j are distinct so when we consider the edges $x_{iy} \xrightarrow{1} c_r$ and $x_{i(y+1)} \xrightarrow{1} c_j$, we have that $r \neq j$. From $h(Source \xrightarrow{2} s_{iy}) \subseteq h(c_j \xrightarrow{6} Sink)$ 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_{iy} has two outgoing edges $x_{iy} \xrightarrow{1} x_{i(y+1)}$ and $x_{iy} \xrightarrow{1} c_r$, so $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) = \{\bar{d}\}$. So we have $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \subseteq h(Source \xrightarrow{2} s_{iy})$.

Finally, we will show that ψ satisfies c_j. From Claim 4 we have $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \subseteq h(Source \xrightarrow{2} s_{iy})$. The edges $Source \xrightarrow{2} Q$ and $Source \xrightarrow{2} s_{iy}$ both cross the straight cut $(\{Source\}, V \setminus \{Source\})$ and hence have colors that are all different. We thus have that $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \not\subseteq h(Source \xrightarrow{2} Q)$. From $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \not\subseteq h(Source \xrightarrow{2} Q)$ and Claim 3 we have that $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \neq h(b_i \xrightarrow{1} a_{i+1})$. Notice that b_i has just two incoming edges, and that from a_i to b_i we have two paths: 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 3 we have that the color of $h(b_i \xrightarrow{1} a_{i+1})$ must be assigned to all edges on exactly one of these paths. We have already established $h(x_{iy} \xrightarrow{1} x_{i(y+1)}) \neq h(b_i \xrightarrow{1} a_{i+1})$, so $h(a_i \xrightarrow{1} \bar{x}_{i1}) = h(b_i \xrightarrow{1} a_{i+1})$. From $h(a_i \xrightarrow{1} \bar{x}_{i1}) = h(b_i \xrightarrow{1} a_{i+1})$ and Claim 3 we conclude $\psi(x_i) = true$. From Claim 2 we have that x_i appears in c_j. From $\psi(x_i) = true$ and that x_i appears in c_j, we conclude that ψ satisfies c_j.

Second, suppose there exists i, h such that $h(Source \xrightarrow{2} \bar{s}_{ih}) \subseteq h(c_j \xrightarrow{6} Sink)$. The proof is similar to the proof of the first case, we omit the details. □

7. Main result

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

Proof. First, the problem is in NP because a register assignment can be verified in polynomial time. Second, the problem is NP-hard because Lemmas 5, 4 and 1 give a chain of reductions 3-SAT $\leq_P$ ACFlow $\leq_P$ A12wig $\leq_P$ Caraar. □

Corollary 1. ACFlow *and* A12wig *are NP-complete.*

8. A proof of Stockmeyer's theorem

Stockmeyer proved a result closely related to ours, namely that the shipbuilding problem is NP-complete [10, Application 9.1, p. 204]. The shipbuilding problem for intervals with weights one or two is:

1-2-coloring of weighted interval graphs (12wig):
Instance: a weighted interval graph G and a number $2K$.
Question: does G have a 1-2-coloring with $2K$ colors?

Stockmeyer proved that 12wig is NP-complete [10, Remark 1, p. 204]. As far as we are aware, our proof below (Theorem 2) is the first publicly available proof of Stockmeyer's theorem.

Fig. 2(a) shows a graph that has a 1-2-coloring with 4 colors but no aligned 1-2-coloring with 4 colors. Hence, we have a case of a graph and a number of colors for which A12wig is unsolvable while 12wig is solvable.

We will reduce A12wig to 12wig. From an instance $(G, 2K)$ of A12wig, we define a weighted interval graph $\mathtt{uag}(G, 2K)$ in the following way. Let s be the minimum of the start points of all intervals in G and let e be the maximum of the end points of all intervals in G. The graph $\mathtt{uag}(G, 2K)$ consists of all the intervals in G and for each i, $0 \leq i < K$, the following intervals:

$$\begin{array}{lrcr}
\text{weight two:} & \delta_{2i} & = & [e, e+2K+i[\\
\text{weight two:} & \delta_{2i+1} & = & [s, e+3K+i[\\
\text{weight one:} & \gamma_{4i} & = & [e+2K+i, e+5K[\\
\text{weight one:} & \gamma_{4i+1} & = & [e+2K+i, e+4K+i[\\
\text{weight one:} & \gamma_{4i+2} & = & [e+3K+i, e+4K+i[\\
\text{weight one:} & \gamma_{4i+3} & = & [e+3K+i, e+5K[\\
\text{weight two:} & \beta_i & = & [e+4K+i, e+5K[.
\end{array}$$

Notice that for all p, $e \leq p < e + 5K$, the total weight of the intervals that contain p is $4K$. Fig. 7 illustrates $\mathtt{uag}(G, 2K)$.

Lemma 6. A12wig $\leq_P$ 12wig.

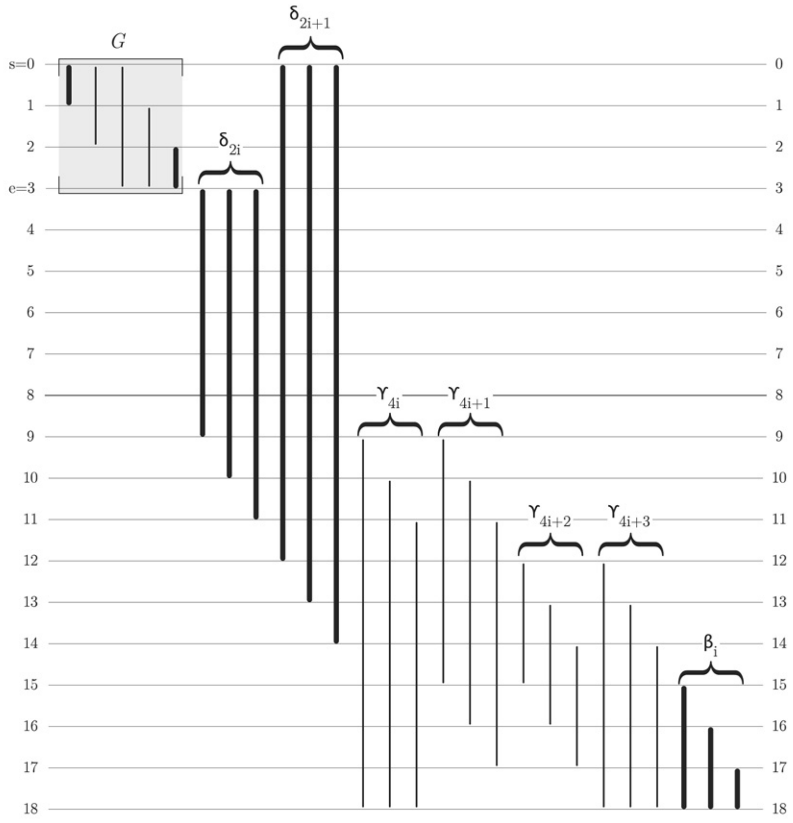

Fig. 7. From G to $\mathsf{uag}(G, 2K)$. $K = 3$. Intervals of weight two are bold.

Proof. Let H denote the weighted interval graph with no intervals. From an instance $(G, 2K)$ of A12WIG where G has q intervals, if $2K > q$, then we construct the instance $(H, 1)$ of 12WIG, otherwise, if $2K \le q$, we construct the instance $(\mathsf{uag}(G, 2K), 4K)$ of 12WIG. The graph $(\mathsf{uag}(G, 2K)$ has $q + 7K \le q + 3.5q = 4.5q$ intervals and we can construct both $(H, 1)$ and $(\mathsf{uag}(G, 2K), 4K)$ in polynomial time.

We now have two cases. In the case of $2K > q$, we have that both $(G, 2K)$ and $(H, 1)$ are solvable.

In the case of $2K \le q$, we will prove that G has an aligned 1-2-coloring with $2K$ colors if and only if $\mathsf{uag}(G, 2K)$ has a 1-2-coloring with $4K$ colors.

$\Rightarrow$) Let κ be an aligned 1-2-coloring of G with $2K$ colors. We define a function m as follows. For $0 \le i < K$, let $m(\{2i\}) = \{4i\}$, let $m(\{2i + 1\}) = \{4i + 1\}$, and let $m(\{2i, 2i + 1\}) = \{4i, 4i + 1\}$. We define a mapping θ from vertices of $\mathsf{uag}(G, 2K)$ to sets of colors:

$$\begin{aligned}
\theta(\alpha) &= m(\kappa(\alpha)), \text{ for each } \alpha \text{ in } G\\
\theta(\delta_{2i}) &= \{4i, 4i + 1\}\\
\theta(\delta_{2i+1}) &= \{4i + 2, 4i + 3\}\\
\theta(\gamma_{4i}) &= \{4i\}\\
\theta(\gamma_{4i+1}) &= \{4i + 1\}\\
\theta(\gamma_{4i+2}) &= \{4i + 2\}\\
\theta(\gamma_{4i+3}) &= \{4i + 3\}\\
\theta(\beta_i) &= \{4i + 1, 4i + 2\}.
\end{aligned}$$

It is straightforward to check that θ is a 1-2-coloring of $\mathsf{uag}(G, 2K)$ with $4K$ colors.

$\Leftarrow$) Let θ be a 1-2-coloring of $\mathsf{uag}(G, 2K)$ with $4K$ colors. We first prove five properties of θ:

- Claim 1: For each j, $0 \le j < 2K$, $\theta(\delta_j)$ is a set of two aligned colors.
- Claim 2: For each i, $0 \le i < K$:
  - (a) $\theta(\delta_{2i}) = \theta(\gamma_{4i}) \cup \theta(\gamma_{4i+1})$
  - (b) $\theta(\beta_i) = \theta(\gamma_{4i+1}) \cup \theta(\gamma_{4i+2})$.
  - (c) $\theta(\delta_{2i+1}) = \theta(\gamma_{4i+2}) \cup \theta(\gamma_{4i+3})$

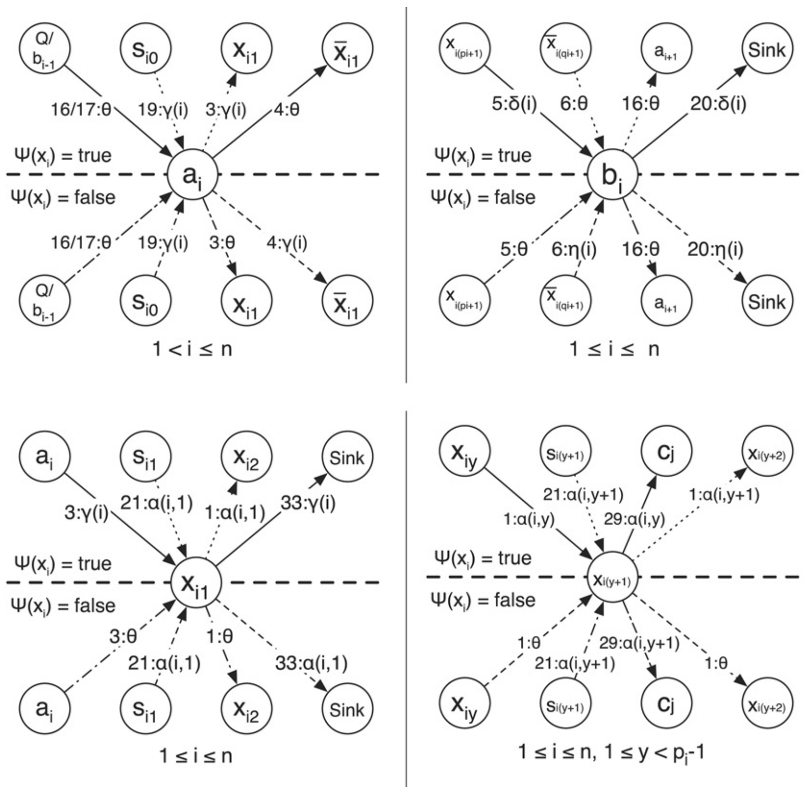

Fig. 8. Four illustrations of edges in $\text{sg}(F)$.

– Claim 3: For each i, $0 \le i < K$, there exists k, $0 \le k < K$, such that:

$$\begin{aligned}\theta(\gamma_{4i}) &= \{4k\}\\ \theta(\gamma_{4i+1}) &= \{4k+1\}\\ \theta(\gamma_{4i+2}) &= \{4k+2\}\\ \theta(\gamma_{4i+3}) &= \{4k+3\}\end{aligned}$$

– Claim 4: $\cup_{i=0}^{k-1}\theta(\delta_{2i+1}) = \{4k+2, 4k+3 \mid 0 \le k < K\}$.
– Claim 5: $\cup_{\alpha\in G}\theta(\alpha) \subseteq \{4k, 4k+1 \mid 0 \le k < K\}$.

Proof of Claim 1. The $2K$ intervals δ_j, $0 \le j < 2K$, all overlap. Given that θ has just $4K$ colors available, the only way it can assign colors to those intervals is to assign each of them a set of aligned colors.

Proof of Claim 2. (a) For each i, $0 \le i < K$, the interval δ_{2i} meets the intervals γ_{4i} and γ_{4i+1} in the point $e+2K+i$, and no other intervals start or end at $e+2K+i$. Given that the total weight of the intervals that contain $e+2K+i$ is $4K$, we have the claimed equation. We can prove (b) and (c) in a manner similar to the proof of (a); we omit the details.

Proof of Claim 3. From Claims 1 + 2 we have that for every i, $0 \le i < K$, θ assigns a pair of aligned colors to γ_{4i} and γ_{4i+1}, θ assigns a pair of consecutive colors to γ_{4i+1} and γ_{4i+2}, and θ assigns a pair of aligned colors to γ_{4i+2} and γ_{4i+3}. From those observations, we have the claimed equations.

Proof of Claim 4. We have that all the intervals δ_{2i+1} overlap, for $0 \le i < K$, so θ assigns them disjoint sets of colors. From that observation, Claim 2(c), and Claim 3 we have the claimed equation.

Proof of Claim 5. Each interval $\alpha \in G$ overlaps with all the intervals δ_{2i+1} so from Claim 4 we have that only colors of the form $4k$ and $4k+1$, for $0 \le k < K$, are available to θ to assign to α.

Let m^{-1} be the inverse of the function m defined above. Let $\kappa = m^{-1} \circ \theta$, restricted to G. From the definition of κ and Claim 5 we have that the range of κ is $\{j \mid 0 \le j < 2K\}$ and that κ must assign every interval of weight two in G an aligned pair of colors. We conclude that κ is an aligned 1-2-coloring of G with $2K$ colors. □

Theorem 2. 1-2-*coloring of weighted interval graphs is NP-complete.*

Proof. First, the problem is in NP because a 1-2-coloring can be verified in polynomial time. Second, the problem is NP-hard because Lemmas 4–6 give a chain of reductions 3-SAT $\le_P$ ACFLOW $\le_P$ A12WIG $\le_P$ 12WIG. □

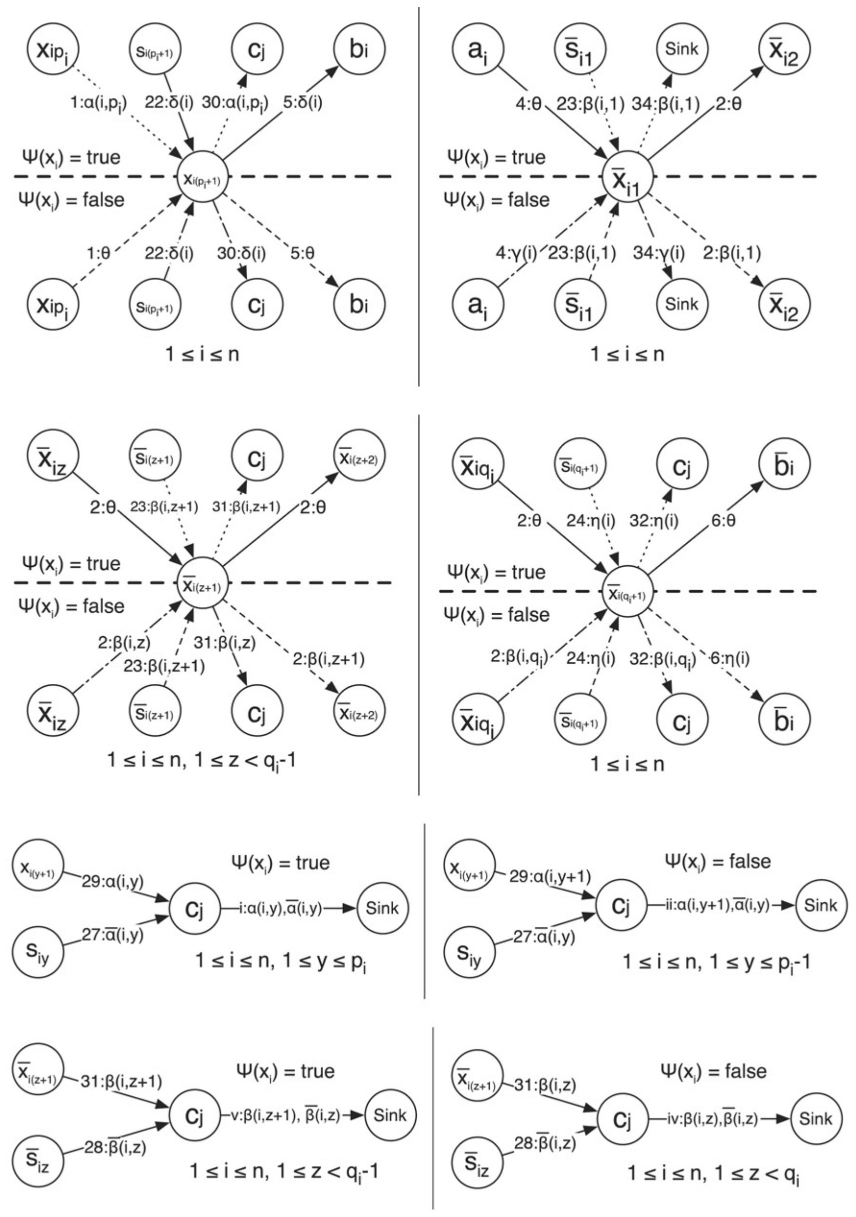
Ψ(x_i) = true
Ψ(x_i) = false
1:α(i,p_i) 22:δ(i) 30:α(i,p_i) 5:δ(i)
1:θ 22:δ(i) 30:δ(i) 5:θ
1 ≤ i ≤ n
4:θ 23:β(i,1) 34:β(i,1) 2:θ
4:γ(i) 23:β(i,1) 34:γ(i) 2:β(i,1)
Sink
2:θ 23:β(i,z+1) 31:β(i,z+1) 2:θ
2:β(i,z) 23:β(i,z+1) 31:β(i,z) 2:β(i,z+1)
1 ≤ i ≤ n, 1 ≤ z < q_i-1
2:θ 24:η(i) 32:η(i) 6:θ
2:β(i,q_i) 24:η(i) 32:β(i,q_i) 6:η(i)
29:α(i,y) 27:ᾱ(i,y) i:α(i,y),ᾱ(i,y)
1 ≤ i ≤ n, 1 ≤ y ≤ p_i
29:α(i,y+1) 27:ᾱ(i,y) ii:α(i,y+1),ᾱ(i,y)
1 ≤ i ≤ n, 1 ≤ y ≤ p_i-1
31:β(i,z+1) 28:β̄(i,z) v:β(i,z+1), β̄(i,z)
31:β(i,z) 28:β̄(i,z) iv:β(i,z),β̄(i,z)
1 ≤ i ≤ n, 1 ≤ z < q_i

Fig. 9. Eight illustrations of edges in $\mathrm{sg}(F)$.

9. Conclusion

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 for aliased register allocation that are used today.

In this paper we have considered register allocation as a decision problem. We can 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?

Acknowledgments

A preliminary version of this paper was presented at ICALP 2007. We thank the TCS reviewers for helpful suggestions. We were supported in part by a National Science Foundation Award number 0401691. Fernando Pereira was sponsored by the Brazilian Ministry of Education under grant number 218603-9.

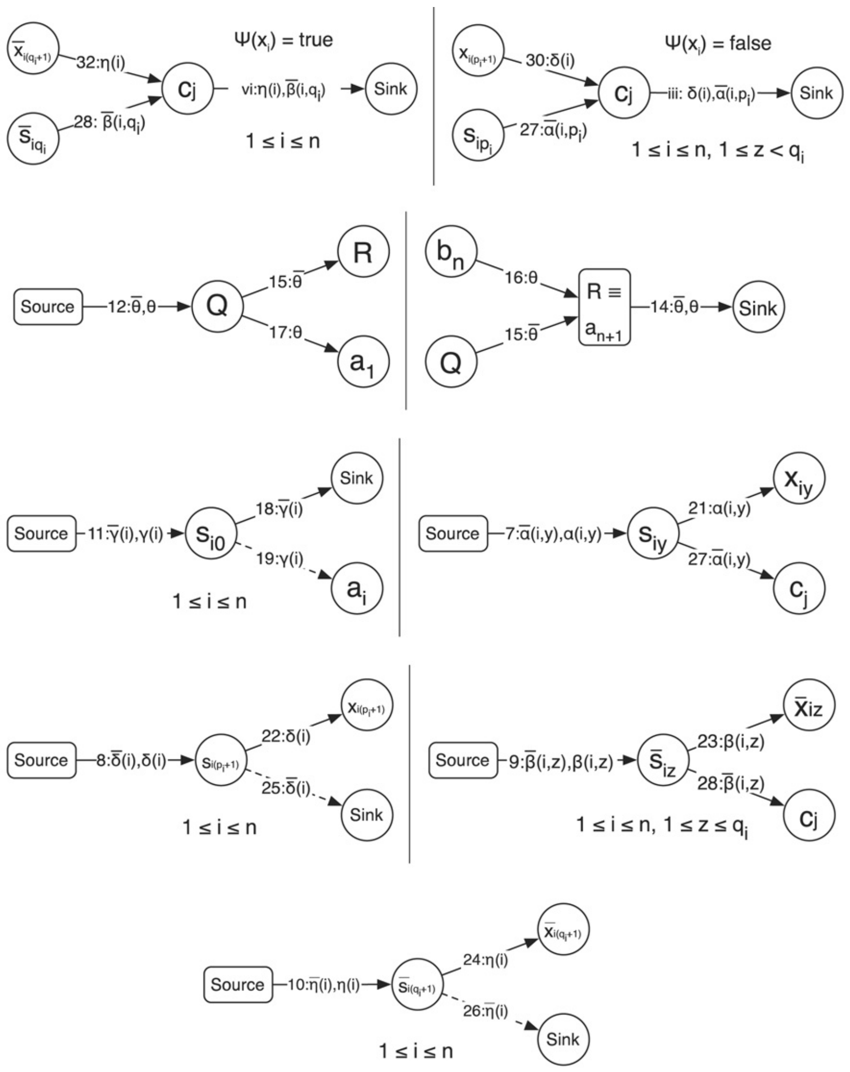

Fig. 10. Nine illustrations of edges in $\mathtt{sg}(F)$.

Appendix. Illustrations

In this section we illustrate the edges in $\mathtt{sg}(F)$. The illustrations show the edges and colors to and from each vertex except *Source* and *Sink*. For vertices where the colors depend on the value of $\psi(x_i)$, we use the same figure to show both the case when $\psi(x_i)$ is true and the case when $\psi(x_i)$ is false.

References

[1] M. Biró, M. Hujter, Zs. Tuza, Precoloring extension. I: Interval graphs, in: Discrete Mathematics, ACM Press, 1992, pp. 267–279. Special volume (part 1) to mark the centennial of Julius Petersen's "Die theorie der regularen graphs".

[2] Preston Briggs, Register allocation via graph coloring, Ph.D. Thesis, Rice University, 1992.

[3] Preston Briggs, Keith Cooper, Linda Torczon, Coloring register pairs, ACM Letters on Programming Languages 1 (1) (1992) 3–13.

[4] Gregory J. Chaitin, Mark A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, Peter W. Markstein, Register allocation via coloring, Computer Languages 6 (1981) 47–57.

[5] Lin Chen, Optimal parallel time bounds for the maximum clique problem on intervals, Information Processing Letters 42 (4) (1992) 197–201.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithmics, second ed., McGraw-Hill, 2001.

[7] S. Even, A. Itai, A. Shamir, On the complexity of timetable and multicommodity flow problems, SIAM Journal on Computing 5 (4) (1976) 691–703.

[8] Martin Farach, Vincenzo Liberatore, On local register allocation, in: 9th ACM-SIAM Symposium on Discrete Algorithms, ACM Press, 1998, pp. 564–573.

[9] Christopher Fraser, David Hanson, A Retargetable C Compiler: Design and Implementation, Addison-Wesley, 1995.

[10] Martin Charles Golumbic, Algorithmic Graph Theory and Perfect Graphs, second ed., Elsevier, 2004.

[11] Ulrich Hirnschrott, Andreas Krall, Bernhard Scholz, Graph coloring vs. optimal register allocation for optimizing compilers, in: JMLC, Modular Programming Languages, Joint Modular Languages Conference, Springer, 2003, pp. 202–213.

[12] Timothy Kong, Kent D. Wilken, Precise register allocation for irregular architectures, in: Internation Symposium on Microarchitecture, ACM, 1998, pp. 297–307.

[13] Akira Koseki, Hideaki Komatsu, Toshio Nakatani, Preference-directed graph coloring, in: PLDI, ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2002, pp. 297–307.
[14] V. Krishna Nandivada, Fernando Magno Quintão Pereira, Jens Palsberg, A framework for end-to-end verification and evaluation of register allocators, in: Proceedings of SAS'07, International Static Analysis Symposium, Kongens Lyngby, Denmark, August 2007, pp. 153–169.
[15] Brian R. Nickerson, Graph coloring register allocation for processors with multi-register operands, in: PLDI, ACM SIGPLAN Conference on Programming Language Design and Implementation, 1990, pp. 40–52.
[16] Johan Runeson, Sven-Olof Nystrom, Retargetable graph-coloring register allocation for irregular architectures, in: SCOPES, Software and Compilers for Embedded Systems, Springer, 2003, pp. 240–254.
[17] Anita Saha, Madhumangal Pal, Tapan K. Pal, Selection of programme slots of television channels for giving advertisement: A graph theoretic approach, Information Sciences 177 (12) (2007) 2480–2492.
[18] Bernhard Scholz, Erik Eckstein, Register allocation for irregular architectures, in: LCTES/SCOPES, Joint Conference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems, ACM, 2002, pp. 139–148.
[19] Michael D. Smith, Norman Ramsey, Glenn Holloway, A generalized algorithm for graph-coloring register allocation, in: PLDI, ACM SIGPLAN Conference on Programming Language Design and Implementation, 2004, pp. 277–288.
[20] Andrew W. Appel, Jens Palsberg, Modern Compiler Implementation in Java, Cambridge University Press, 2002.