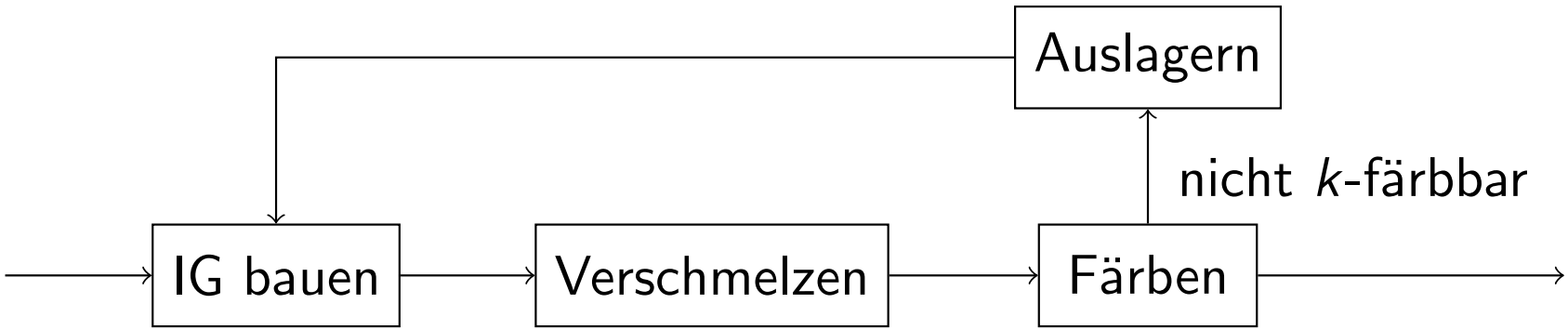
SSA Registerzuteilung

Inhalt

- Idee
- Grundlagen
- Auslagern
- Verschmelzung / SSA Abbau

Registerzuteilung mit Graphfärbung (nach Chaitin/Briggs)

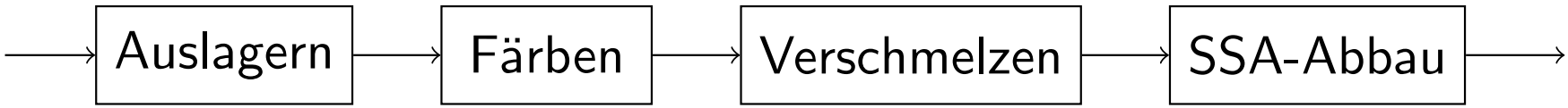
Architektur mit Iteration (non-SSA)



- Jeder ungerichtete Graph kann als Interferenzgraph (IG) auftreten
- Die Bestimmung der chromatischen Zahl ist NP-vollständig
- Färben ist eine Heuristik $\Rightarrow$ Iteration notwendig
  Selbst wenn Heuristik $n + k$ Farben schätzt, muss nach Auslagen von n Registern keine Lösung mit k Registern zu finden sein
- Auslagern ist auf IG fokussiert

Registerzuteilung mit Graphfärbung (nach Hack/Goos)

Architektur ohne Iteration (SSA)



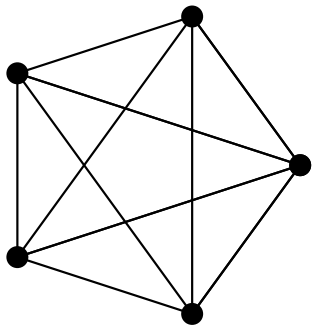
- Wegen Chordalität der SSA IGs:
  - Trennen von Auslagern und Verschmelzen möglich
  - Färben in polynomieller Zeit
- Registerdruck ist ein präzises Maß für Zahl benötigter Register
- Wir wissen vor dem Färben welche Werte ausgelagert werden müssen
  - ▶ Alle Marken bei denen der Registerdruck größer als k ist
- Auslagern kann vor dem Färben geschehen und
- Das Färben scheitert anschließend nie!
- Verschmelzen neu formuliert als Optimierungsproblem mit einer Kostenfunktion für Färbungen ▶ Komplexität ist jetzt hier

Inhalt

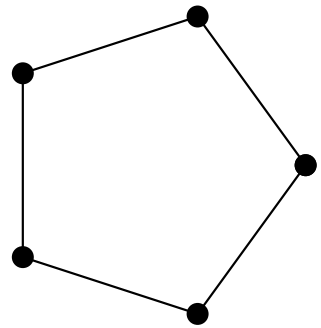
- Idee
- Grundlagen
- Auslagern
- Verschmelzung / SSA Abbau

Vollständige Graphen and Zyklen

Grundlagen



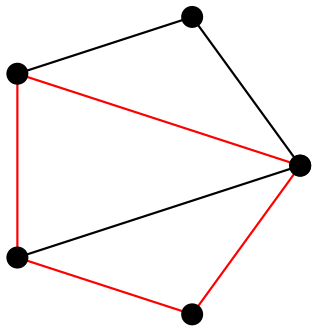
Vollständiger Graph K^5



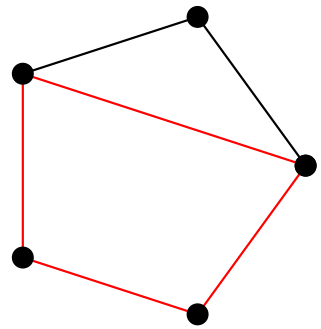
Zyklus C^5

Teilgraphen / Untergraphen

Grundlagen



Graph mit einem
C^4-Teilgraphen



Graph mit einem
C^4-Untergraphen

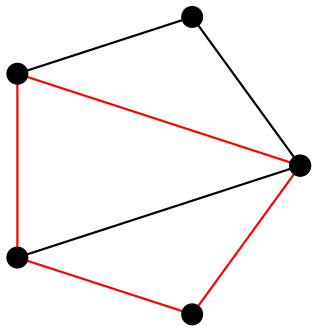
Untergraphen werden auch induzierte Teilgraphen genannt

Definition

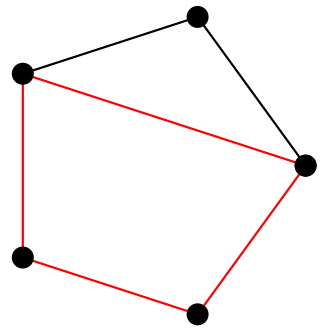
Komplette Untergraphen heißen Cliquen

Teilgraphen / Untergraphen

Grundlagen



Graph mit einem
C^4-Teilgraphen



Graph mit einem
C^4-Untergraphen

Untergraphen werden auch induzierte Teilgraphen genannt

Definition

Komplette Untergraphen heißen Cliquen

Cliquenzahl and chromatische Zahl

Grundlagen

$\omega(G)$ Eckenzahl der größten Clique in G

$\chi(G)$ Anzahl der Farben in einer minimalen Färbung von G

Corollary

$\omega(G) \leq \chi(G)$ gilt für jeden Graphen G

$\omega(G)$	3	2	2	3
$\chi(G)$	3	2	3	3

Cliquenzahl and chromatische Zahl

Grundlagen

$\omega(G)$ Eckenzahl der größten Clique in G

$\chi(G)$ Anzahl der Farben in einer minimalen Färbung von G

Corollary

$\omega(G) \leq \chi(G)$ gilt für jeden Graphen G

$\omega(G)$	3	2	2	3
$\chi(G)$	3	2	3	3

Cliquenzahl and chromatische Zahl

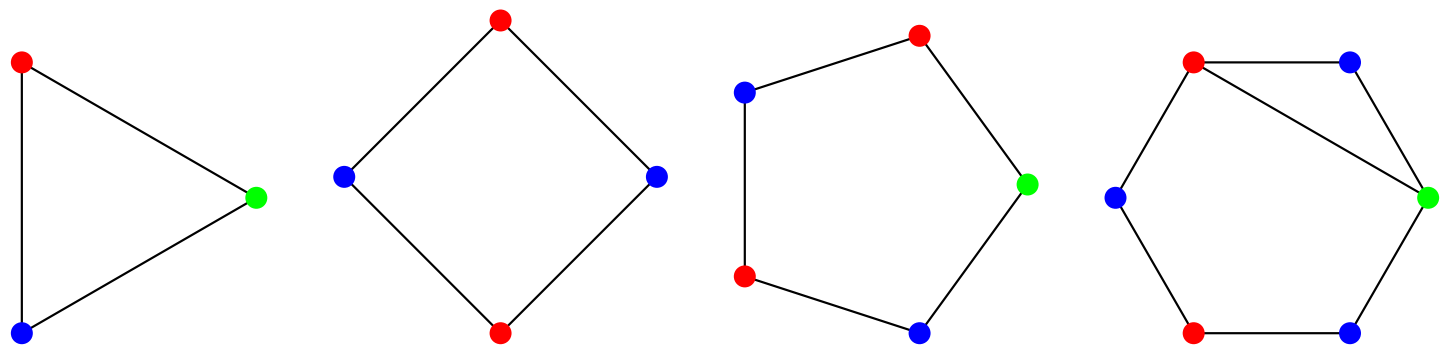
Grundlagen

$\omega(G)$ Eckenzahl der größten Clique in G

$\chi(G)$ Anzahl der Farben in einer minimalen Färbung von G

Corollary

$\omega(G) \leq \chi(G)$ *gilt für jeden Graphen* G



$\omega(G)$	3	2	2	3
$\chi(G)$	3	2	3	3

Perfekte Graphen

Grundlagen

Definition

G ist perfekt $\iff$ $\chi(H) = \omega(H)$ gilt für alle Untergraphen H von G

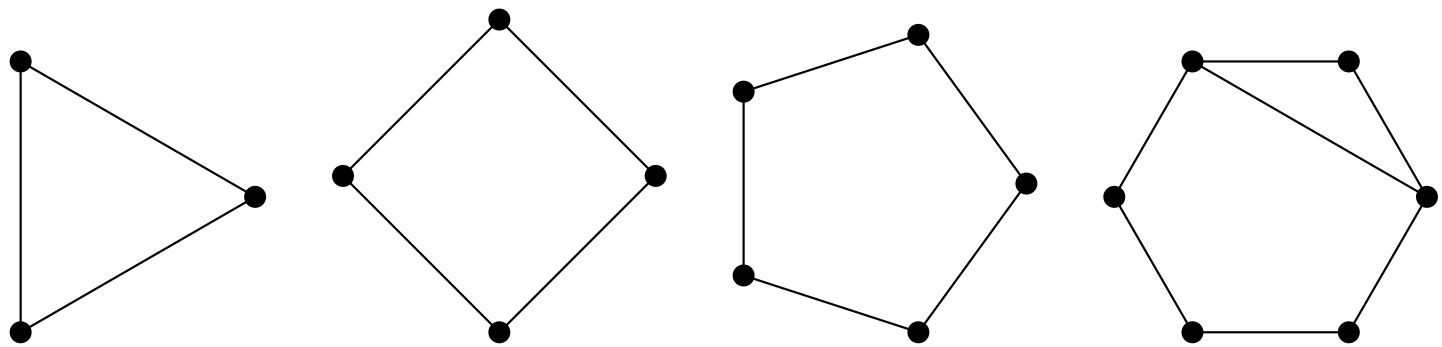
perfekt?	✓	✓		

Perfekte Graphen

Grundlagen

Definition

G ist perfekt $\iff$ $\chi(H) = \omega(H)$ gilt für alle Untergraphen H von G



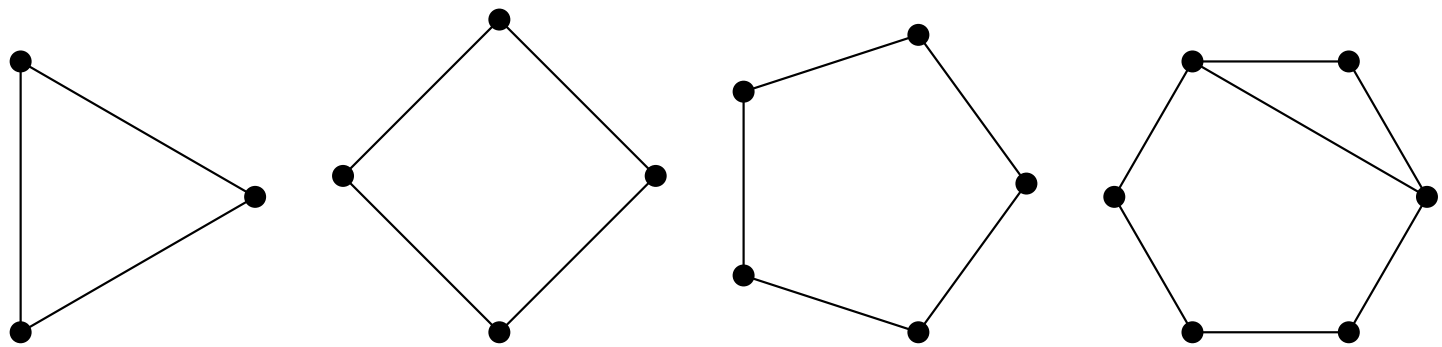
perfekt?	✓	✓		

Perfekte Graphen

Grundlagen

Definition

G ist perfekt $\iff$ $\chi(H) = \omega(H)$ gilt für alle Untergraphen H von G



perfekt?	✓	✓		

Chordale Graphen

Grundlagen

Definition

G ist chordal $\iff$ G enthält keine Untergraphen mit Zyklen länger 3

chordal?	✓	✓	

Theorem

Chordale Graphen sind perfekt

Theorem

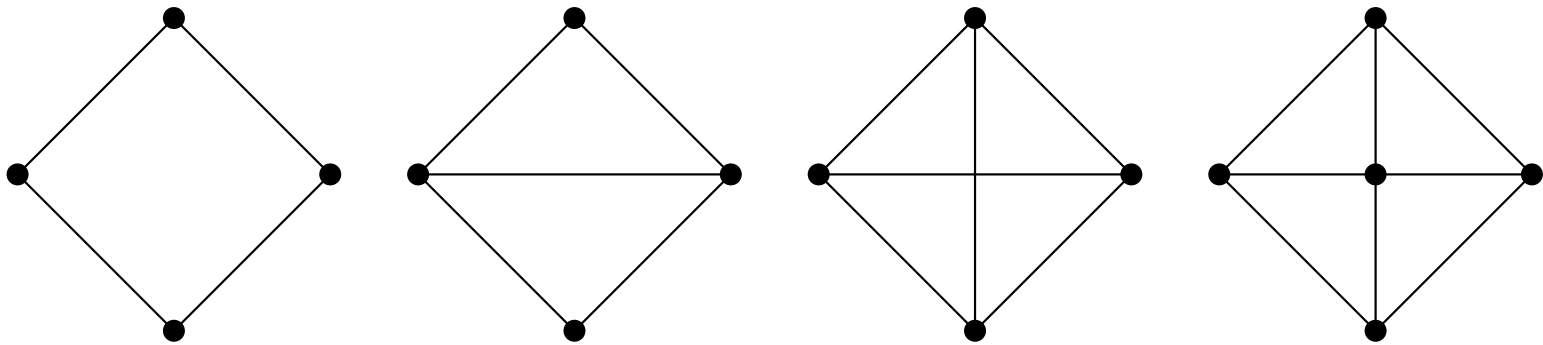
Chordale Graphen sind in $O(\lvert V\rvert \cdot \omega(G))$ optimal färbbar

Chordale Graphen

Grundlagen

Definition

G ist chordal $\iff$ G enthält keine Untergraphen mit Zyklen länger 3



chordal?	✓	✓	

Theorem

Chordale Graphen sind perfekt

Theorem

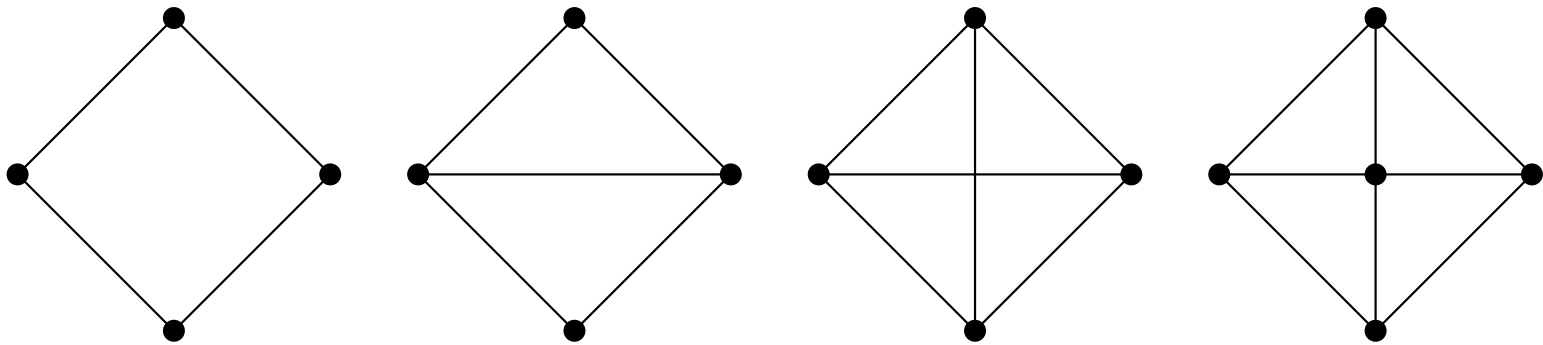
Chordale Graphen sind in $O(\lvert V\rvert \cdot \omega(G))$ optimal färbbar

Chordale Graphen

Grundlagen

Definition

G ist chordal $\iff$ G enthält keine Untergraphen mit Zyklen länger 3



chordal?		✓	✓

Theorem

Chordale Graphen sind perfekt

Theorem

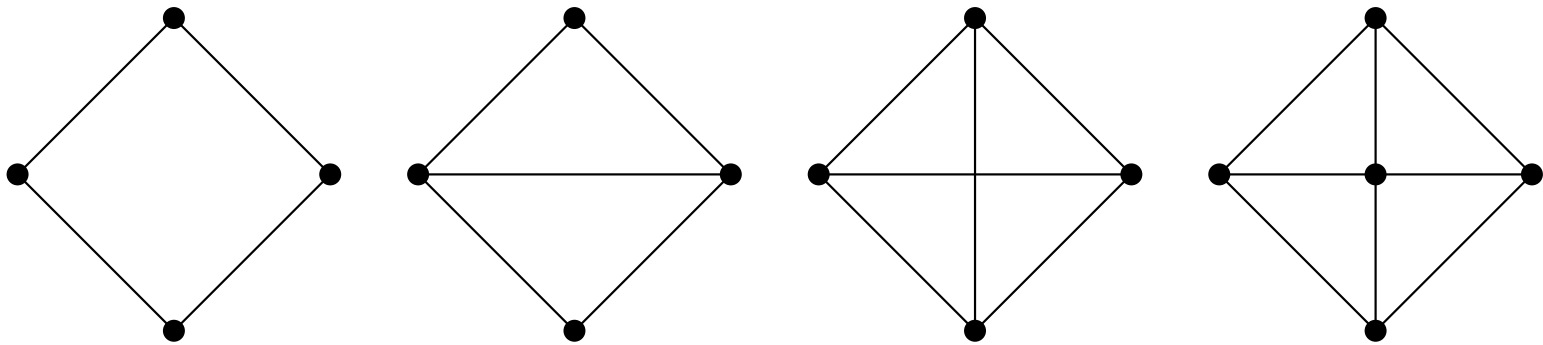
Chordale Graphen sind in $O(\lvert V\rvert \cdot \omega(G))$ optimal färbbar

Chordale Graphen

Grundlagen

Definition

G ist chordal $\iff$ G enthält keine Untergraphen mit Zyklen länger 3



chordal?		✓	✓

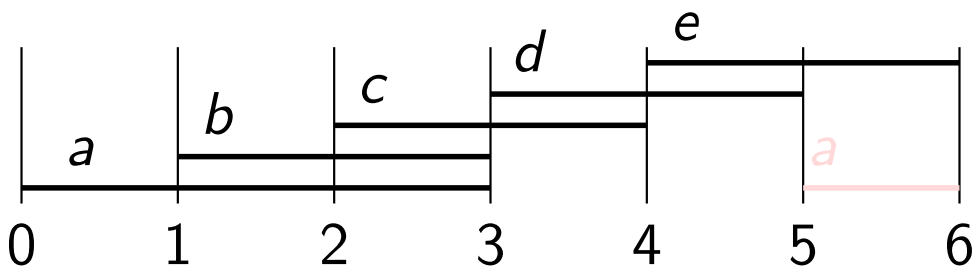
Theorem

Chordale Graphen sind perfekt

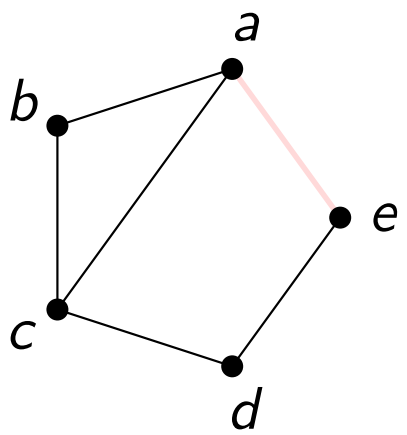
Theorem

Chordale Graphen sind in $O(|V| \cdot \omega(G))$ optimal färbbar

Warum sind SSA-IGs chordal? — intuitiv

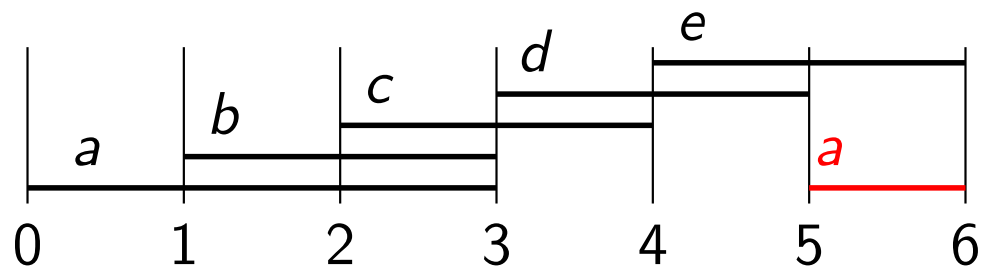


- Jedes Intervall entspricht der Lebenszeit einer Variablen
  - einer Ecke im IG

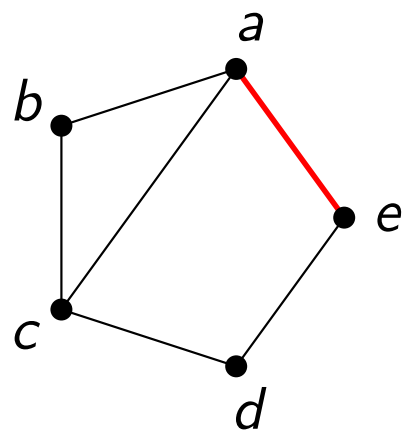


- Kann durch Einfügen einer Kante (a, e) ein Zyklus gebildet werden?
- Das geht nur wenn a erneut bei 5 beginnt
- Das verletzt jedoch die SSA-Eigenschaft, da a 2 Definitionen hätte!

Warum sind SSA-IGs chordal? — intuitiv

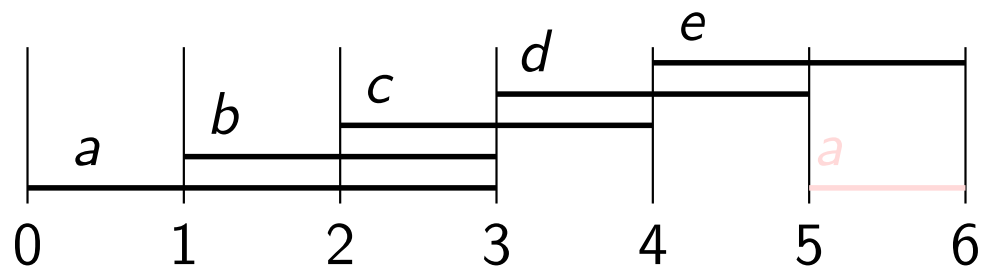


- Jedes Intervall entspricht der Lebenszeit einer Variablen
  - einer Ecke im IG

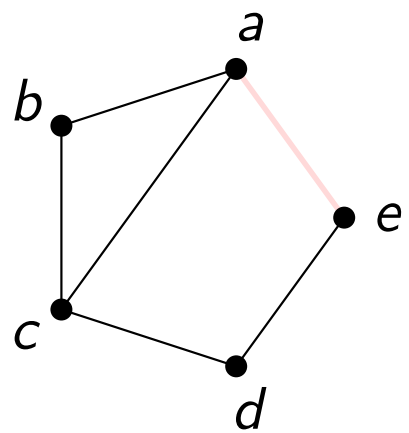


- Kann durch Einfügen einer Kante (a, e) ein Zyklus gebildet werden?
- Das geht nur wenn a erneut bei 5 beginnt
- Das verletzt jedoch die SSA-Eigenschaft, da a 2 Definitionen hätte!

Warum sind SSA-IGs chordal? — intuitiv



- Jedes Intervall entspricht der Lebenszeit einer Variablen
  - einer Ecke im IG



- Kann durch Einfügen einer Kante (a, e) ein Zyklus gebildet werden?
- Das geht nur wenn a erneut bei 5 beginnt
- Das verletzt jedoch die SSA-Eigenschaft, da a 2 Definitionen hätte!

Dominanz und Perfekte Eliminationsschemata

- Bevor ein Wert v zu einem PES hinzugefügt wird, füge alle Werte, deren Definitionen von $\mathcal{D}_v$ dominiert werden, ein
- Somit führt eine Post-Order Besuchsreihenfolge des Dominanzbaums zu einem PES
- IGs von SSA-Programmen können in $O(\chi(G) \cdot |V|)$ gefärbt werden
- Dazu muss der IG selbst nicht einmal berechnet werden

Inhalt

- Idee
- Grundlagen
- Auslagern
- Verschmelzung / SSA Abbau

Auslagern

Theorem

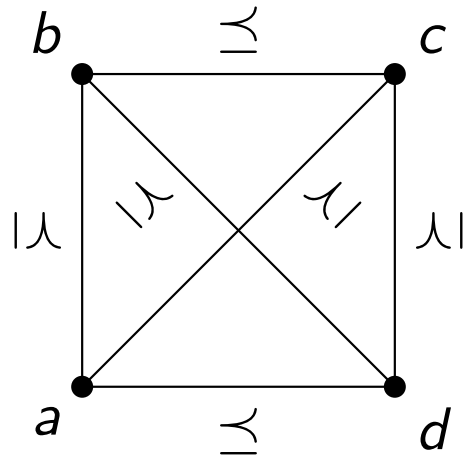
Für jede Clique in einem IG existiert eine Programmstelle, an der alle Ecken der Clique lebendig sind.

- Die Dominanz induziert also eine *totale Ordnung* innerhalb der Clique
  $\Rightarrow$ Es existiert ein "größter" Wert d
- Alle anderen sind lebendig an der Definitionsstelle von d
- Existieren nur drei Register, muss a, b oder c an $\mathcal{D}_d$ ausgelagert sein

Auslagern

Theorem

Für jede Clique in einem IG existiert eine Programmstelle, an der alle Ecken der Clique lebendig sind.



- Die Dominanz induziert also eine *totale Ordnung* innerhalb der Clique
  $\Rightarrow$ Es existiert ein "größter" Wert d
- Alle anderen sind lebendig an der Definitionsstelle von d
- Existieren nur drei Register, muss a, b oder c an $\mathcal{D}_d$ ausgelagert sein

Auslagern

Konsequenzen

- Die chromatische Zahl des IG ist exakt durch die Anzahl der lebendigen Werte an den Programmstellen festgelegt
- Reduktion der Anzahl der lebendigen Werte an jeder Programmstelle auf k macht den IG k-färbbar
- Wir kennen a-priori die Programmstellen, an denen Werte ausgelagert werden müssen
  - An allen Programmstellen mit einem Registerdruck größer k
- Auslagern kann also vor dem Färben durchgeführt werden und
- Färben mit k Farben gelingt danach stets

Schlussfolgerung

- Im Gegensatz zu Chaitin/Briggs-Zuteilern keine Iteration notwendig

Auslagern

Konsequenzen

- Die chromatische Zahl des IG ist exakt durch die Anzahl der lebendigen Werte an den Programmstellen festgelegt
- Reduktion der Anzahl der lebendigen Werte an jeder Programmstelle auf k macht den IG k-färbbar
- Wir kennen a-priori die Programmstellen, an denen Werte ausgelagert werden müssen
  - An allen Programmstellen mit einem Registerdruck größer k
- Auslagern kann also vor dem Färben durchgeführt werden und
- Färben mit k Farben gelingt danach stets

Schlussfolgerung

- Im Gegensatz zu Chaitin/Briggs-Zuteilern keine Iteration notwendig

Auslagern

Allgemein zu lösende Probleme beim Auslagern

- Wenn mehrere Werte ausgelagert werden könnten, welchen wählen?
- Wo genau platziert man die Auslagerungs-/Einlagerungsinstruktionen?
- Ist die Wiederberechnung (Rematerialisierung) manchmal günstiger als ein Auslagern?
  - Konstanten
  - Prozedurargumente auf dem Keller
  - Einfache Berechnungen ...
- Auslagerungsplätze zusammenfassen (um Platz zu sparen)?

Farach und Liberatore

- Optimales Auslagern innerhalb eines Grundblocks ist NP-vollständig.

Auslagern

Allgemein zu lösende Probleme beim Auslagern

- Wenn mehrere Werte ausgelagert werden könnten, welchen wählen?
- Wo genau platziert man die Auslagerungs-/Einlagerungsinstruktionen?
- Ist die Wiederberechnung (Rematerialisierung) manchmal günstiger als ein Auslagern?
  - Konstanten
  - Prozedurargumente auf dem Keller
  - Einfache Berechnungen ...
- Auslagerungsplätze zusammenfassen (um Platz zu sparen)?

Farach und Liberatore

- Optimales Auslagern innerhalb eines Grundblocks ist NP-vollständig.

Auslagern

Spezielle SSA-Probleme

- ϕ-Funktionen werden gleichzeitig und am Anfang eines Grundblocks ausgeführt, deshalb kann weder vor noch zwischen ihnen aus-/eingelagert werden
- Deshalb müssen Werte vor Betreten des Grundblocks ausgelagert werden
  - Argumente von ϕ-Funktionen sind ausgelagert
  - ϕ-Funktionen bei denen alle Argumente ausgelagert sind benötigen kein Register
  - Vorsicht: Beim SSA-Abbau entstehen hier Speicherkopien statt Registerkopien

Auslagern

Implementierung

- Heuristik nach Belady:
  - Bevorzuge diejenigen Werte bei der Auslagerung, deren nächste Benutzung am weitesten in der Zukunft liegt
    - Was ist die “Entfernung” bei Verzweigungen?
  - Beachte: Benutzungen außerhalb von Schleifen sind stets “weiter” in der Zukunft als innerhalb!
  - Einlagerungsinstruktionen möglich vor einer Schleife platzieren
- Auch als ILP formulierbar

Inhalt

- Idee
- Grundlagen
- Auslagern
- Verschmelzung / SSA Abbau

SSA-Abbau

- Gegeben eine k-Färbung eines SSA-IGs
- Können wir daraus eine gültige Registerzuteilung mit k Registern für das zugehörige non-SSA Programm erzeugen?

Zentrale Frage

Wie behandeln wir ϕ-Funktionen?

SSA-Abbau

- Gegeben eine k-Färbung eines SSA-IGs
- Können wir daraus eine gültige Registerzuteilung mit k Registern für das zugehörige non-SSA Programm erzeugen?

Zentrale Frage

Wie behandeln wir ϕ-Funktionen?

ϕ-Funktionen

- Alle ϕ-Funktionen in einem Grundblock

$$\begin{aligned} y_1 &\leftarrow \phi(x_{11}, \ldots, x_{n1}) \\ &\vdots \\ y_m &\leftarrow \phi(x_{1m}, \ldots, x_{nm}) \end{aligned}$$

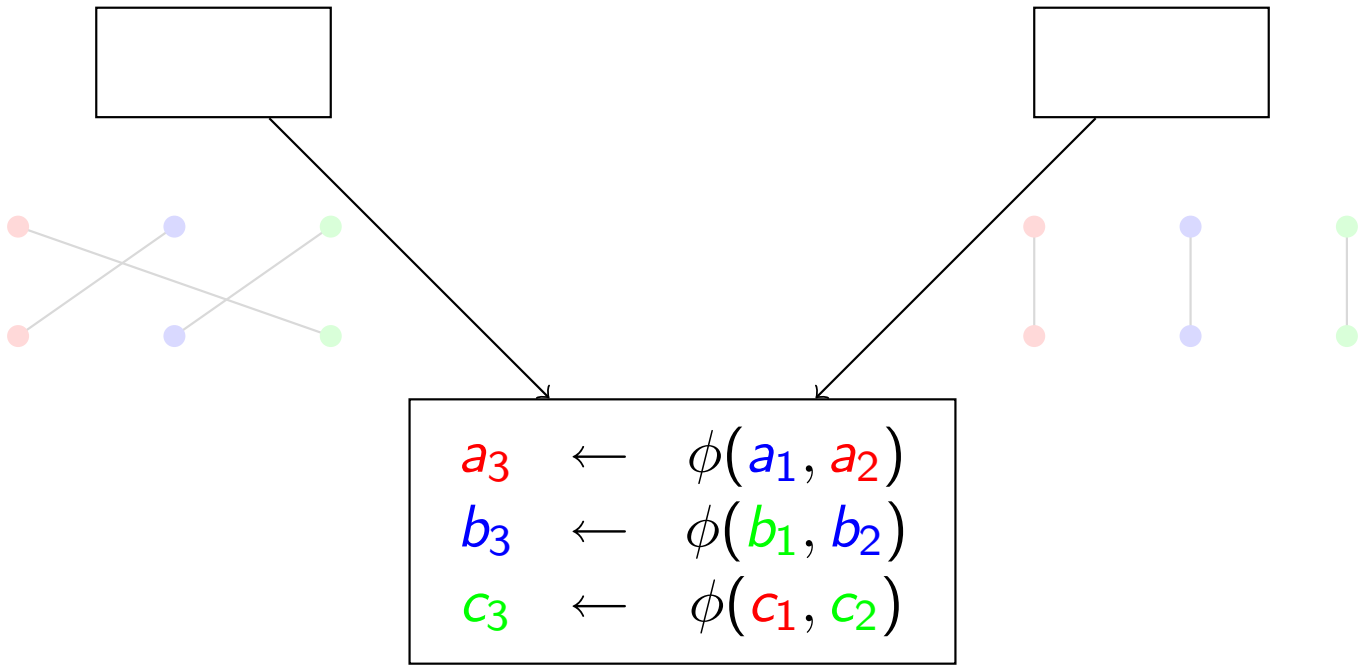
  werden gleichzeitig und vor allen anderen Instruktionen in diesem Grundblock ausgeführt.

- Betreten wir diesen Block über die i-te Kante, wirken die ϕ-Funktionen wie eine parallele Kopierinstruktion

$$(y_1, \ldots, y_m) \leftarrow (x_{i1}, \ldots, x_{im})$$

ϕ-Funktionen

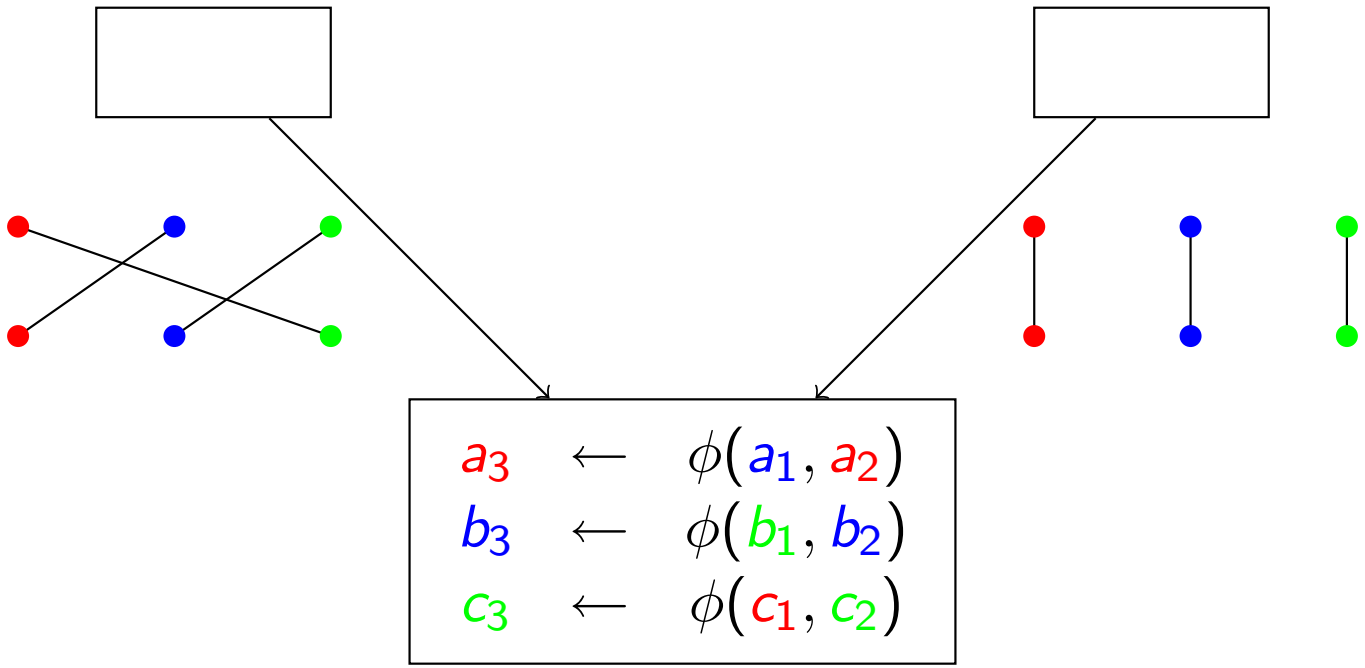
- Beispiel



- Die ϕs stellen Registerpermutationen auf den Steuerflusskanten dar

ϕ-Funktionen

- Beispiel



- Die ϕs stellen Registerpermutationen auf den Steuerflusskanten dar

ϕ-Funktionen

- Beispiel

$$
\begin{aligned}
a_3 &\leftarrow \phi(a_1, a_2) \\
b_3 &\leftarrow \phi(b_1, b_2) \\
c_3 &\leftarrow \phi(c_1, c_2)
\end{aligned}
$$

- Die ϕs stellen Registerpermutationen auf den Steuerflusskanten dar

Auf Kante 1

Auf Kante 2

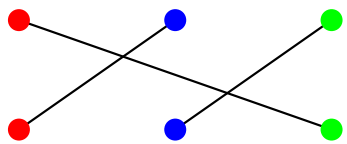
ϕ-Funktionen

- Beispiel

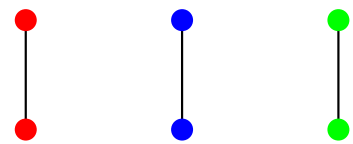
$$
\begin{aligned}
a_3 &\leftarrow \phi(a_1, a_2) \\
b_3 &\leftarrow \phi(b_1, b_2) \\
c_3 &\leftarrow \phi(c_1, c_2)
\end{aligned}
$$

- Die ϕs stellen Registerpermutationen auf den Steuerflusskanten dar

Auf Kante 1

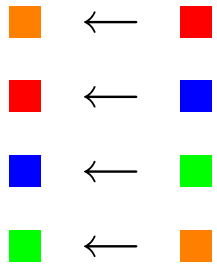


Auf Kante 2



Permutationen

- Eine Permutation kann mit Hilfe von Kopien implementiert werden, wenn ein Hilfsregister ■ verfügbar ist

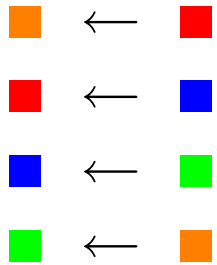


- Permutationen können als Folge von Transpositionen (also Vertauschungen) implementiert werden

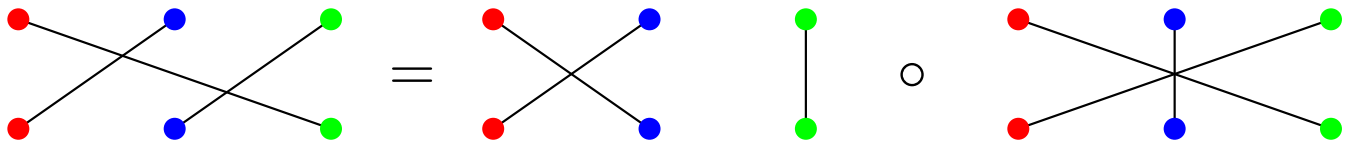
- Eine Transposition kann als Folge von drei `xor`-Instruktionen ohne Verwendung eines zusätzlichen Registers implementiert werden

Permutationen

- Eine Permutation kann mit Hilfe von Kopien implementiert werden, wenn ein Hilfsregister ■ verfügbar ist



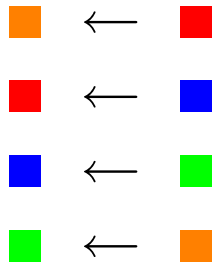
- Permutationen können als Folge von Transpositionen (also Vertauschungen) implementiert werden



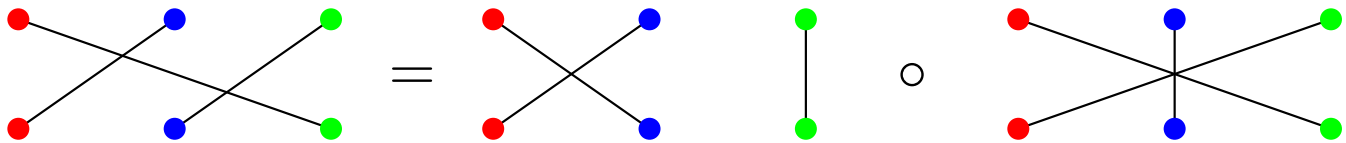
- Eine Transposition kann als Folge von drei `xor`-Instruktionen ohne Verwendung eines zusätzlichen Registers implementiert werden

Permutationen

- Eine Permutation kann mit Hilfe von Kopien implementiert werden, wenn ein Hilfsregister ■ verfügbar ist



- Permutationen können als Folge von Transpositionen (also Vertauschungen) implementiert werden

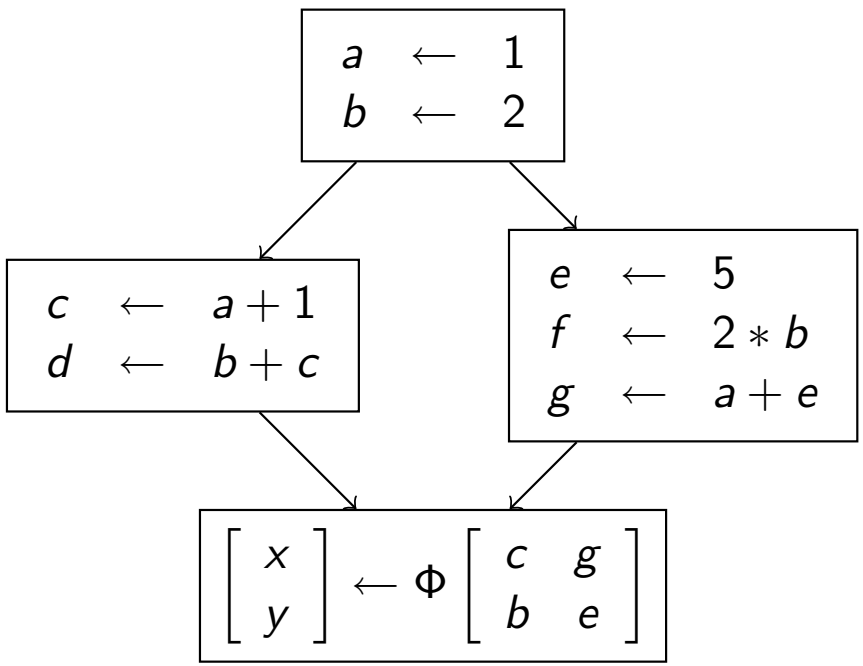


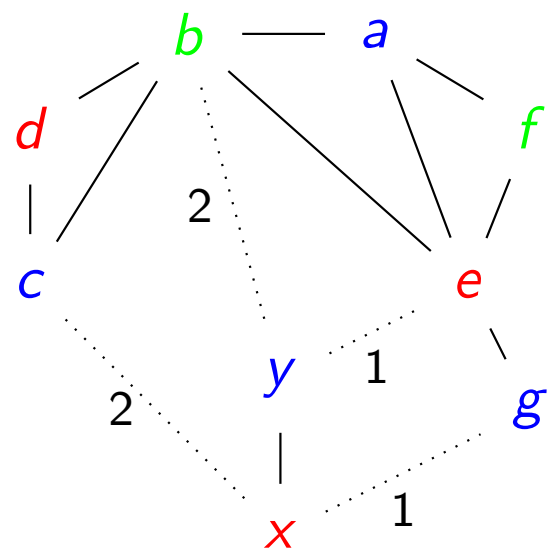
- Eine Transposition kann als Folge von drei `xor`-Instruktionen ohne Verwendung eines zusätzlichen Registers implementiert werden

Verschmelzung

Problemmodell

- Gegeben: Eine minimale Färbung des IG
- Gesucht: Eine erlaubte ($< k$) Färbung mit minimalen Kosten
  - Kosten sind die gewichtete Summe der Gleichfärbekanten
  - Unbenutzte Farben dürfen verwendet werden
  - Weder IG noch Programm dürfen geändert werden

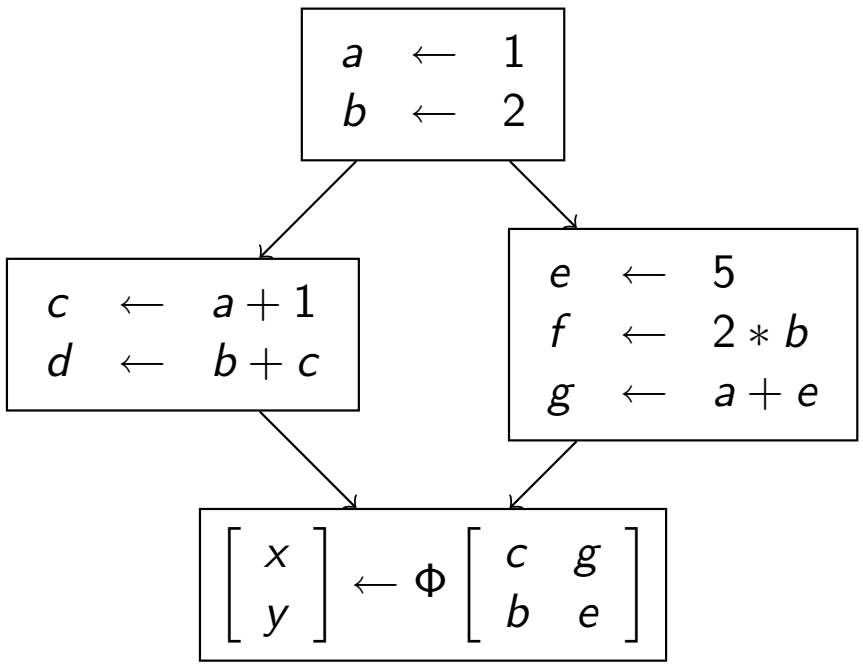


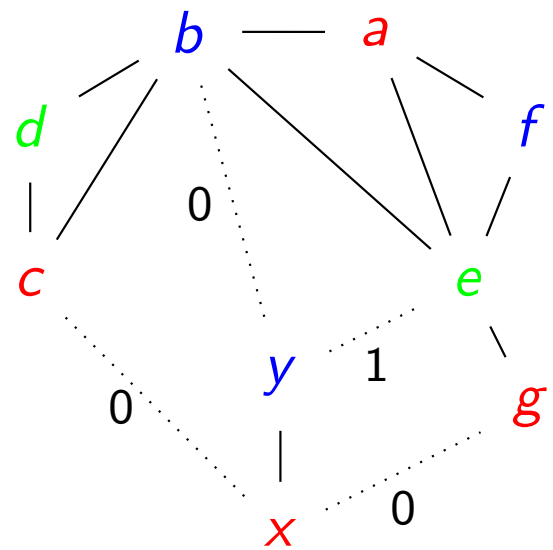


Verschmelzung

Problemmodell

- Gegeben: Eine minimale Färbung des IG
- Gesucht: Eine erlaubte ($< k$) Färbung mit minimalen Kosten
  - Kosten sind die gewichtete Summe der Gleichfärbekanten
  - Unbenutzte Farben dürfen verwendet werden
  - Weder IG noch Programm dürfen geändert werden





Verschmelzung

Lösungsstrategien

Komplexität

- Problem ist NP-vollständig in der Anzahl der ϕs

Algorithmen

- Eine Greedy-Heuristik
- Eine optimale Methode, die ILP[1] benutzt

[1] Integer Linear Programming

Registerzuteilung – Zusammenfassung

- Registerzuteilung ist NP-vollständig
- Hack/Goos liefert ein polynomielles Verfahren zur Graphfärbung, das die Registerzuteilung in 3 sequentielle Einzelschritte zerlegt
- Verschmelzung und Auslagerung bleibt aber NP-vollständig
- Selbst bei Verwendung heuristischer Verfahren kann die Registerzuteilung den Großteil der Übersetzungszeit brauchen
- Für große Graphen liefert *linear-scan* in vielen Anwendungsszenarien schneller eine brauchbare Lösung
- Ein weiterer Ansatz: Formulierung als ganzzahliges lineares Programm (ILP) (langsam)
- Das Zusammenspiel der zielmaschinenabhängigen Optimierungen ist offen