

Markus Richter 614027

Maximilian Spolert 607513

Mats Pfeiffer 607718

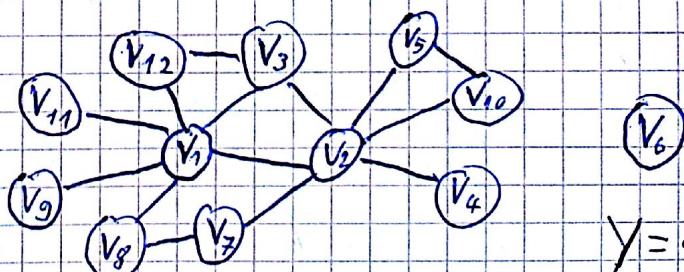
Felix Esch 607571

Einführung

Das Ziel des Breakers ist es, einen Knoten des Graphen  $G$  so zu umzingeln, dass dieser nicht mehr färbbar ist (im Folgenden als gesättigt bezeichnet). ✓

Wir geben immer zuerst die Absicht an gefolgt von den Aktionen des Algorithmus, markiert mit einem Pfeil ( $\rightarrow$ ).

Außerdem sei  $Y$  die Anzahl der Farben,  $g(v)$  der Grad von  $v$  (Ausgangsgrad) und  $s(v)$  die der Grad der Färbung; also die Anzahl der verschiedenen Farben in den Nachbarn.

Beispielgraph: $Y = 4$  Farben

- Am diesem Graphen erkennen wir bereits einige Strukturen, die von Interesse sind.

- Nur  $V_1$  und  $V_2$  können gesättigt werden.

- Speichere Liste von Kandidaten in Candidates ✓

- \* mit Eigenschaft  $g(v) \geq Y$

- $V_3$  erhöht die ist mit 2 Kandidaten verbunden und kann somit besser umzingeln.

- höherer Grad ist vorzuziehen. ✓

Das Ziel ist es nun einen Kandidaten zu sättigen um zu gewinnen, denn so wird der Graph unfarbbar.

### Initialisierungsphase (1. Zug)

Der Algorithmus speichert den Graphen und analysiert ihn.

→ Suche alle Knoten  $v$  mit  $g(v) \geq Y$  und füge sie in Candidates so ein, dass die Liste ~~ist~~ absteigend nach dem Grad sortiert ist. ✓

→ Erstelle eine verlinkte Liste von Farben, sodass sie aufsteigend sortiert werden kann nach Häufigkeit der Farbe. ✓

→ Erstelle eine Map, die jedem Knoten eine Liste von blockierten Farben speichert. Sie ist anfangs leer, da keine Sättigung vorliegt. ✓

guter  
Aufbau

### Auswahl eines Kandidaten:

Jeder Zug beginnt mit der Auswahl eines Kandidaten, um ihn zu umzingeln.

Ein hoher Grad  $g(v)$  erleichtert das Umzingeln von  $v$ .

→ Wähle erstes Element von Candidates, da die Liste sortiert ist. (Gehe zur Auswahl des Nachbarn)

## Auswahl des Nachbarn eines Kandidaten ✓

Wir maximieren die Lättigung aller Kandidaten, indem dessen Nachbarn eingefärbt werden.

Dabei beachten wir:

- Der Nachbar sollte mit möglichst vielen Kandidaten verbunden sein, aber kein Kandidat selbst. So arbeiten wir nicht für den Macher. ✓

→ Von allen Nachbarn  $w_i$  mit  $g(w_i) < Y$  den mit  $\max \{g(w_i)\}$ .

Falls kein solches  $w_i$  die Lättigung  $s(v)$  erhöht, dann wählen wir  $\min_i \{g(w_i)\}$ , aber aus allen Nachbarn  $w_i$ .

Ist so ein Knoten ausgewählt, dann färbe ihn (gehe zur Farbung).

Ansonsten prüfe die verbleibende Zeit. Entferne  $v$  aus Candidates und gehe wieder zur Auswahl eines Kandidaten.

Bei knapper Zeit wähle  $v$  als Notlösung.

- Die Farbe sollte so gewählt werden, dass möglichst viele Knoten an Lättigung zunehmen ✓

→ Wir versuchen alle Farben gleich häufig im Graphen zu verteilen. ✓ gut!

## Bei Färbung eines Knotens $v$

Unabhängig von dem Spieler verhält sich der Algorithmus bei einer Färbung wie folgt.

→  $v$  muss ggf. aus Candidates gelöscht werden.

→ Die sortierte Liste colors muss angepasst werden.

→ Für alle Nachbarn  $w_i$  muss die Liste der Lättigung angepasst werden.

→ Da Knoten nur eine Lättigung besitzen  $s$  ungleich null besitzen, kann diese für die Sortierung beachtet werden.

Dabei ist ein Kandidat  $v$  mit hohen Werten für  $s(v)$  und  $g(v)$  leichter zu färbigen.

Sortiere nach Heuristik:  $h(v) = \alpha g(v) + (1-\alpha)s(v)$   
mit  $\alpha \in [0, 1]$ , wobei  $\alpha$  durch Erfahrungswerte gesetzt werden sollte.

## Fazit:

- Wir halten diesen Algorithmus für sinnvoll, weil er eine natürliche Bewertung von Knoten vornimmt und dabei ein gutes Laufzeitverhalten verspricht. ✓