Constraint Satisfaction Problems – CSP.01–CSP.05

Enes Pohland

16. November 2025

CSP.01: Logikrätsel als CSP

Es gibt fünf Häuser, durchnummeriert von links nach rechts mit den Positionen

$$H = \{1, 2, 3, 4, 5\}.$$

Jedes Haus besitzt genau eine Ausprägung in fünf Attributkategorien: Nationalität, Farbe, Getränk, Zigarettenmarke und Haustier.

Variablen und Wertebereiche

Für jedes Attribut und jede Hausposition definiere ich eine Variable. Beispielhaft:

- $\mathit{Color}_h \in \{\text{rot}, \text{grün}, \text{weiß}, \text{gelb}, \text{blau}\}$,
- $\mathit{Nat}_h \in \{\text{Engländer}, \text{Spanier}, \text{Ukrainer}, \text{Norweger}, \text{Japaner}\}$,
- $\mathit{Drink}_h \in \{\text{Kaffee}, \text{Tee}, \text{Milch}, \text{Orangensaft}, \text{Wasser}\}$,
- $\mathit{Smoke}_h \in \{\text{OldGold}, \text{Kools}, \text{Chesterfield}, \text{LuckyStrike}, \text{Parliaments}\}$,
- $\mathit{Pet}_h \in \{\text{Hund}, \text{Schnecken}, \text{Fuchs}, \text{Pferd}, \text{Zebra}\}$,

für alle $h \in H$. Insgesamt entstehen also $5 \cdot 5 = 25$ Variablen.

All-Different-Constraints

In jeder Attributkategorie muss jede Ausprägung genau einmal vorkommen. Für jede Kategorie verwende ich daher einen AllDifferent-Constraint, den man als Menge binärer Ungleichheitsconstraints formulieren kann. Für die Farben etwa:

$$\forall h \neq k \in H : \quad \mathit{Color}_h \neq \mathit{Color}_k.$$

Analoge Constraints gibt es für Nat_h, Drink_h, Smoke_h und Pet_h.

Binäre und unäre Constraints aus den Hinweisen

Im Folgenden schreibe ich beispielhaft je einen Constraint pro Hinweis, jeweils als Relation zwischen den betroffenen Variablen.

(1) "Der Engländer wohnt im roten Haus."
Für jedes Haus h:

$$C_1(\mathit{Nat}_h, \mathit{Color}_h) = \{(\text{Engländer}, \text{rot})\} \cup \{(n, c) \mid n \neq \text{Engländer} \wedge c \neq \text{rot}\}.$$

(2) "Der Spanier hat einen Hund."
Für jedes Haus h:

$$C_2(Nat_h, Pet_h) = \{(\text{Spanier}, \text{Hund})\} \cup \{(n, p) \mid n \neq \text{Spanier} \wedge p \neq \text{Hund}\}.$$

(3) "Kaffee wird im grünen Haus getrunken."
Für jedes Haus h:

$$C_3(Drink_h, Color_h) = \{(\text{Kaffee}, \text{grün})\} \cup \{(d, c) \mid d \neq \text{Kaffee} \wedge c \neq \text{grün}\}.$$

(4) "Der Ukrainer trinkt Tee."
Für jedes Haus h:

$$C_4(Nat_h, Drink_h) = \{(\text{Ukrainer}, \text{Tee})\} \cup \{(n, d) \mid n \neq \text{Ukrainer} \wedge d \neq \text{Tee}\}.$$

(5) "Das grüne Haus ist direkt rechts vom weißen Haus."
Für $h \in \{1, \ldots, 4\}$:

$$C_5(Color_h, Color_{h+1}) = \{(\text{weiß}, \text{grün})\} \cup \{(c_1, c_2) \mid c_1 \neq \text{weiß} \wedge c_2 \neq \text{grün}\}.$$

(6) "Old-Gold-Raucher halten Schnecken als Haustiere."
Für jedes Haus h:

$$C_6(Smoke_h, Pet_h) = \{(\text{OldGold}, \text{Schnecken})\} \cup \{(s, p) \mid s \neq \text{OldGold} \wedge p \neq \text{Schnecken}\}.$$

(7) "Kools werden im gelben Haus geraucht."
Für jedes Haus h:

$$C_7(Smoke_h, Color_h) = \{(\text{Kools}, \text{gelb})\} \cup \{(s, c) \mid s \neq \text{Kools} \wedge c \neq \text{gelb}\}.$$

(8) "Milch wird im mittleren Haus getrunken."
Unärer Constraint:

$$C_8(Drink_3): \quad Drink_3 = \text{Milch}.$$

(9) "Der Norweger wohnt im ersten Haus."
Unärer Constraint:

$$C_9(Nat_1): \quad Nat_1 = \text{Norweger}.$$

(10) "Der Chesterfield-Raucher wohnt neben dem Mann mit dem Fuchs."
Modelliert als Nachbarschaftsconstraints zwischen $Smoke_h$ und den Haustieren in den Nachbarhäusern. Beispielsweise für $h = 1$:

$$C_{10,1}(Smoke_1, Pet_2) = \{(\text{Chesterfield}, \text{Fuchs})\} \cup \{(s, p) \mid s \neq \text{Chesterfield} \wedge p \neq \text{Fuchs}\},$$

analog für alle anderen Positionen; für mittlere Häuser werden Kombinationen mit linkem und rechtem Nachbarn zugelassen.

(11) "Kools werden im Haus neben dem Haus mit dem Pferd geraucht."
Analog zu (10) mit Kools und Pferd.

(12) "Lucky-Strike-Raucher trinken Orangensaft."
Für jedes Haus h:

$$C_{12}(Smoke_h, Drink_h) = \{(\text{LuckyStrike}, \text{Orangensaft})\} \cup \{(s, d) \mid s \neq \text{LuckyStrike} \wedge d \neq \text{Orangensaft}\}.$$

(13) "Der Japaner raucht Parliaments."
Für jedes Haus h:

$$C_{13}(Nat_h, Smoke_h) = \{(\text{Japaner}, \text{Parliaments})\} \cup \{(n, s) \mid n \neq \text{Japaner} \wedge s \neq \text{Parliaments}\}.$$

(14) "Der Norweger wohnt neben dem blauen Haus."
Zusammen mit $Nat_1 = \text{Norweger}$ folgt direkt:

$$C_{14}(Color_2): \quad Color_2 = \text{blau}.$$

Antwort auf die Fragen

Eine vollständige Lösung des CSP (zum Beispiel mit Backtracking und Konsistenzprüfung) liefert eindeutig:

- Wasser trinkt der Norweger.
- Das Zebra gehört dem Japaner.

CSP.02: Einstein-Rätsel mit BT_Search, MAC und Min-Conflicts

Für die algorithmische Lösung verwende ich eine leicht veränderte Modellierung: Jede Ausprägung (z. B. *Engländer*, *rot*, *Kaffee*) wird als Variable modelliert, deren Domäne aus den Hauspositionen $1, \ldots, 5$ besteht. Beispiel:

$$Englaender, Spanier, \ldots, Zebra \in \{1, 2, 3, 4, 5\}.$$

All-Different-Constraints sorgen dafür, dass etwa alle Nationalitäten verschiedene Hausnummern besitzen. Struktur-Constraints wie "Engländer im roten Haus" werden als Gleichungen der Form

$$Englaender = rot$$

modelliert, Nachbarschaftsbedingungen wie "Grün rechts von Weiß" als

$$gruen = weiss + 1,$$

und ähnliche Gleichungen für die übrigen Hinweise.

Backtracking mit Basisalgorithmus BT_Search

Der Basisalgorithmus entspricht rekursivem Tiefensuchen:

- Es wird jeweils eine noch unbelegte Variable ausgewählt (einfache Reihenfolge, z. B. wie in einer Liste).
- Für jeden möglichen Wert in der Domäne wird geprüft, ob die bisherige Belegung alle Constraints erfüllt. Bei Erfolg wird rekursiv weiter gesucht, bei einem Widerspruch wird zurückgesetzt (Backtracking).

Der Algorithmus findet die eindeutige Lösung des Einstein-Rätsels. Die Anzahl der besuchten Suchknoten ist relativ hoch, weil noch keine Heuristiken oder Vorwärtskonsistenz genutzt werden.

Backtracking mit MRV- und Gradheuristik

Zur Verbesserung der Variablenwahl wird BT_Search um zwei Heuristiken ergänzt:

MRV (Minimum Remaining Values): Wähle die unzugewiesene Variable mit der kleinsten verbleibenden Domänengröße.

Gradheuristik: Falls mehrere Variablen dieselbe kleinste Domänengröße besitzen, wähle diejenige mit den meisten Constraints zu noch unzugewiesenen Nachbarn.

In der verwendeten Modellierung bleiben die Domänen während der Suche ohne zusätzliche Vorwärtspropagation allerdings relativ groß, sodass die Vorteile von MRV hier begrenzt sind. Qualitativ gilt dennoch:

- Der Algorithmus findet weiterhin dieselbe eindeutige Lösung.
- Die Suche konzentriert sich früher auf stark eingeschränkte Variablen, was in vielen CSPs die Knotenanzahl reduziert.
- In dieser konkreten Modellierung kann die Heuristik jedoch auch zu einer ungünstigen Reihenfolge führen und im Vergleich zur einfachen Reihenfolge sogar mehr Knoten erzeugen.

Einsatz von AC-3 vor BT_Search

Vor dem Start von BT_Search wende ich den AC-3-Algorithmus auf das vollständige CSP an.

- AC-3 entfernt bereits viele Werte aus Domänen, die mit keinem Wert der Nachbarvariablen kompatibel sind, und stellt Kantenskonsistenz her.
- AC-3 allein liefert noch keine vollständige Lösung des Rätsels: Mehrere Variablen besitzen danach noch Domänen mit mehr als einem Wert.

Anschließend starte ich BT_Search mit den reduzierten Domänen:

- BT_Search mit AC-3-Vorverarbeitung benötigt deutlich weniger Backtracking-Schritte als ohne AC-3, da viele Kombinationen bereits ausgeschlossen sind.
- Sowohl mit als auch ohne MRV/Gradheuristik wird so die gleiche eindeutige Lösung gefunden; die reine Laufzeit hängt stark von der Implementierung und der Wahl der Heuristik ab.

Min-Conflicts-Heuristik

Min-Conflicts verfolgt einen lokalen Suchansatz:

1. Initialisierung mit einer zufälligen vollständigen Belegung (alle Variablen haben einen Wert, Constraints können verletzt sein).
2. Solange noch Konflikte existieren:
   (a) Wähle eine Variable, die an mindestens einem Konflikt beteiligt ist.
   (b) Weise ihr den Wert zu, der (unter ihren Domänenwerten) die Zahl der Konflikte minimal macht.
3. Optional nach einer festen Schrittzahl ein Neustart.

Für das Einstein-Rätsel zeigt sich:

- Das Problem ist klein, stark eingeschränkt und besitzt genau eine Lösung.
- Min-Conflicts kann in lokalen Minima hängen bleiben und findet die Lösung nicht immer zuverlässig, insbesondere bei beschränkter Schrittzahl.
- Systematische Suche mit Backtracking plus AC-3 ist hier deutlich robuster.

CSP.03: Kantenskonsistenz mit AC-3

Gegeben ist das CSP mit

$$V = \{v_1, v_2, v_3, v_4\}, \qquad D = \{0, \dots, 5\},$$

und Constraints

$$\begin{aligned} c_1 &: (v_1, v_2),\ x + y = 3, \\ c_2 &: (v_2, v_3),\ x + y \leq 3, \\ c_3 &: (v_1, v_3),\ x \leq y, \\ c_4 &: (v_3, v_4),\ x \neq y. \end{aligned}$$

Constraint-Graph

Der Constraint-Graph besitzt vier Knoten $v_1, \dots, v_4$ und Kanten

$$(v_1, v_2),\ (v_2, v_3),\ (v_1, v_3),\ (v_3, v_4),$$

also ein Dreieck (v_1, v_2, v_3) und einen zusätzlichen Knoten v_4, der an v_3 hängt.

AC-3 Schritt für Schritt

Zu Beginn gilt

$$D(v_1) = D(v_2) = D(v_3) = D(v_4) = \{0, 1, 2, 3, 4, 5\}.$$

Die Queue Q enthält alle gerichteten Bögen, z. B.

$$Q_0 = [(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_2), (v_1, v_3), (v_3, v_1), (v_3, v_4), (v_4, v_3)].$$

Iteration 1: Bogen (v_1, v_2), Constraint $x + y = 3$ Revision von $D(v_1)$:

$$x = 4, 5$$

haben keinen Partner in $D(v_2)$, werden entfernt.

$$D(v_1) = \{0, 1, 2, 3\}.$$

Da sich $D(v_1)$ geändert hat, werden Bögen (X, v_1) mit Nachbarn X von v_1 (außer v_2) wieder in die Queue eingefügt.

Iteration 2: Bogen (v_2, v_1), Constraint $x + y = 3$ Revision von $D(v_2)$:

$$x = 4, 5$$

haben keinen Partner in $D(v_1) = \{0, 1, 2, 3\}$, werden entfernt.

$$D(v_2) = \{0, 1, 2, 3\}.$$

Iteration 3: Bogen (v_2, v_3), Constraint $x+y \leq 3$ Revision von $D(v_2)$ ergibt keine Änderung, da es für jeden Wert in $D(v_2)$ einen kompatiblen Wert in $D(v_3)$ gibt.

Iteration 4: Bogen (v_3, v_2), Constraint $x + y \leq 3$ Revision von $D(v_3)$:

$$y = 4, 5$$

haben keinen Partner in $D(v_2) = \{0, 1, 2, 3\}$ mit $x + y \leq 3$ und werden entfernt.

$$D(v_3) = \{0, 1, 2, 3\}.$$

Änderungen bei v_3 führen dazu, dass Bögen (v_1, v_3) und (v_4, v_3) erneut in die Queue kommen.

Weitere Bögen Die verbleibenden Bögen (v_1, v_3) und (v_3, v_1) mit Constraint $x \leq y$ sowie (v_3, v_4) und (v_4, v_3) mit Constraint $x \neq y$ führen zu keinen weiteren Domänenreduktionen, da alle Werte jeweils kompatible Partner besitzen.

Ergebnis

Nach Abschluss von AC-3 ist das CSP kantenkonsistent mit

$$D(v_1) = D(v_2) = D(v_3) = \{0, 1, 2, 3\}, \qquad D(v_4) = \{0, 1, 2, 3, 4, 5\}.$$

Es liegt noch keine vollständige Lösung vor, die Kantenkonsistenz reduziert aber die Suchräume von v_1, v_2, v_3 deutlich.

CSP.04: Forward Checking und Kantenskonsistenz

Wieder das CSP aus CSP.03, nun mit partieller Belegung

$$\alpha = \{v_1 \mapsto 2\}.$$

Kantenskonsistenz unter α

Zunächst gelten unter α die Domänen

$$D(v_1) = \{2\}, \quad D(v_2) = D(v_3) = D(v_4) = \{0, \ldots, 5\}.$$

Nun wird Kantenskonsistenz bzgl. aller Bögen hergestellt.

- Aus $c_1(v_1, v_2) : v_1 + v_2 = 3$ folgt mit $v_1 = 2$ direkt
$$D(v_2) = \{1\}.$$
- Aus $c_3(v_1, v_3) : v_1 \leq v_3$ folgt mit $v_1 = 2$
$$D(v_3) = \{2, 3, 4, 5\}.$$
- Aus $c_2(v_2, v_3) : v_2 + v_3 \leq 3$ mit $v_2 = 1$ folgt
$$D(v_3) = \{2\},$$
da nur $1 + 2 \leq 3$ gilt.
- Aus $c_4(v_3, v_4) : v_3 \neq v_4$ mit $v_3 = 2$ folgt
$$D(v_4) = \{0, 1, 3, 4, 5\},$$
weil $v_4 = 2$ keinen ungleichen Partner hätte.

Endergebnis der Kantenkonsistenz unter α:

$$D(v_1) = \{2\}, \quad D(v_2) = \{1\}, \quad D(v_3) = \{2\}, \quad D(v_4) = \{0, 1, 3, 4, 5\}.$$

Forward Checking unter α

Forward Checking betrachtet nur die Constraints der aktuell gesetzten Variable v_1 zu noch unzugewiesenen Variablen.

- Aus $c_1(v_1, v_2)$ folgt wie oben $D(v_2) = \{1\}$.
- Aus $c_3(v_1, v_3)$ folgt wie oben $D(v_3) = \{2, 3, 4, 5\}$.
- v_4 ist nicht direkt mit v_1 verbunden, daher bleibt $D(v_4) = \{0, 1, 2, 3, 4, 5\}$ unverändert.

Endergebnis des Forward Checking:

$$D(v_1) = \{2\}, \quad D(v_2) = \{1\}, \quad D(v_3) = \{2, 3, 4, 5\}, \quad D(v_4) = \{0, 1, 2, 3, 4, 5\}.$$

Vergleich

Kantenkonsistenz ist deutlich stärker als Forward Checking:

- Forward Checking propagiert nur einen Schritt von der aktuell gesetzten Variablen aus und reduziert die Domänen von v_3 und v_4 nicht so stark.
- Kantenkonsistenz berücksichtigt zusätzlich die Constraints zwischen den noch unzugewiesenen Variablen (hier insbesondere c_2 und c_4) und führt daher zu

  $$D(v_3) = \{2\}, \quad D(v_4) = \{0, 1, 3, 4, 5\},$$

  also zu einer deutlich stärkeren Einschränkung des Suchraums.

CSP.05: Planung von Indoor-Spielplätzen

Problemabstraktion

Die rechteckige Halle habe die Größe 40 m × 20 m und wird zu einem Gitter von 1 m × 1 m diskretisiert. Die Rasterkoordinaten sind

$$X = \{1, \ldots, 40\}, \qquad Y = \{1, \ldots, 20\}.$$

Es existieren:

- ein normaler Eingang im unteren Bereich der Halle,
- mehrere Notausgänge an der Hallenperipherie,
- verschiedene rechteckige Spielgeräte (Go-Kart-Bahn, Hüpfburg, Kletterberg),
- ein rechteckiger Barbereich,
- Entspannungszonen (ebenfalls rechteckig).

Alle Rechtecke sind rastergenau, Sicherheitsabstände betragen 1 m, und Notausgänge dürfen nicht blockiert werden. Vom Barbereich aus soll der Kletterberg beobachtet werden können.

Variablen und Domänen

Ich definiere eine Variable pro Gerät mit Position und Orientierung als Wert.
Beispielsweise:

- B (Bar),
- G (Go-Kart),
- H (Hüpfburg),
- K (Kletterberg),
- E (Entspannungszone).

Jede Variable $i \in \{B, G, H, K, E\}$ hat Werte der Form

$$i = (x_i, y_i, o_i),$$

wobei (x_i, y_i) die linke untere Koordinate des Gerätes beschreibt und $o_i \in \{0, 90\}$ die Orientierung (Rotation um 90°).

Die Domäne $D(i)$ besteht aus allen Tripeln (x_i, y_i, o_i), bei denen das zugehörige Rechteck vollständig innerhalb der Halle liegt. Die genauen Breiten und Höhen (je nach Gerät und Orientierung) sind bekannt, z. B.

$$\begin{aligned} \text{Bar: } & 6 \times 4, \\ \text{Go-Kart: } & 16 \times 6, \\ \text{Hüpfburg: } & 8 \times 6, \\ \text{Kletterberg: } & 6 \times 6, \\ \text{Entspannungszone: } & 10 \times 4. \end{aligned}$$

Die Domäne wird außerdem so gefiltert, dass keine Türbereiche überdeckt werden.

Unäre Constraints

Innenlage der Rechtecke Für jedes Gerät i mit Breite $w_i(o_i)$ und Höhe $h_i(o_i)$:

$$1 \leq x_i \leq 40 - w_i(o_i) + 1, \qquad 1 \leq y_i \leq 20 - h_i(o_i) + 1.$$

Bar nahe am Eingang Sei c_{door} der Mittelpunkt des Eingangsbereichs, c_B der Mittelpunkt des Barrechtecks. Dann:

$$\text{dist}(c_B, c_{\text{door}}) \leq 8,$$

z. B. in Manhattan- oder euklidischer Distanz.

Türen freihalten Für jedes Gerät i gilt

$$\text{rect}(i) \cap \text{DOORS} = \emptyset,$$

wobei rect(i) die Menge aller von Gerät i belegten Rasterzellen und DOORS die Menge der Türzellen bezeichnet.

Binäre Constraints

Sicherheitsabstand und Nichtüberlappung Zwischen zwei Geräten i und j ist ein Sicherheitsabstand $d = 1$ m einzuhalten. Sei R_i das Rechteck von i und R_j das von j. Die Condition lautet:

$$\text{expanded}(R_i) \cap \text{expanded}(R_j) = \emptyset,$$

wobei expanded(R) das Rechteck R ist, das um d nach außen vergrößert wird. In Koordinatenform:

$$x_i + w_i + d \leq x_j \ \vee \ x_j + w_j + d \leq x_i \ \vee \ y_i + h_i + d \leq y_j \ \vee \ y_j + h_j + d \leq y_i.$$

Sichtlinie Bar–Kletterberg Vereinfachte Modellierung: Die Projektionen des Bar- und Kletterbergrechtecks auf einer Achse sollen sich überlappen, und der Kletterberg soll "hinter" der Bar stehen, zum Beispiel:

$$[y_B, y_B + h_B] \cap [y_K, y_K + h_K] \neq \emptyset, \qquad x_B < x_K.$$

Optional kann man weitere Constraints hinzufügen, die verhindern, dass andere Geräte (z. B. Go-Kart oder Hüpfburg) diese Sichtlinie blockieren.

Entspannungszone in der Nähe des Kletterbergs Die Entspannungszone soll in der Nähe des Kletterbergs liegen, aber weiterhin den Sicherheitsabstand einhalten:

$$\text{dist}(c_E, c_K) \leq 6$$

und zusätzlich wieder ein Nichtüberlappungs- und Sicherheitsabstandsconstraint zwischen E und K.

Beispiel einer konsistenten Lösung

Eine mögliche Placement-Lösung (in Metern, Orientierung $o = 0$) ist:

$$B = (1, 1, 0),$$
$$K = (12, 3, 0),$$
$$G = (22, 12, 0),$$
$$H = (10, 12, 0),$$
$$E = (18, 4, 0).$$

Alle Rechtecke liegen innerhalb der Halle, halten Notausgänge frei, verletzen keine Sicherheitsabstände und erfüllen die Sichtlinienbedingung zwischen Bar und Kletterberg (z. B. horizontale Sicht auf einer gemeinsamen Höhe).

Lösung mit MAC

Zur Lösung als CSP wird eine Backtracking-Suche mit MAC (Maintaining Arc Consistency) eingesetzt:

- Nach jeder Variablenzuweisung wird mittels AC-3 Kantenskonsistenz auf dem restlichen Problem aufrechterhalten.
- Es bietet sich an, zuerst Variablen mit stark eingeschränkter Domäne zu belegen, etwa die Bar (wegen Nähe zum Eingang) und den Kletterberg (wegen Sichtlinie), danach Go-Kart, Hüpfburg und Entspannungszone.
- Durch MAC werden viele unpassende Placements früh aus den Domänen entfernt, sodass nur ein relativ kleiner Teil des kombinatorischen Raums tatsächlich durchsucht werden muss.

Lösung mit Min-Conflicts

Für Min-Conflicts wird das gleiche Modell als lokale Suche verwendet:

1. Initialisiere jede Variable mit einem zufälligen zulässigen Placement (innerhalb der Halle, Türen frei).
2. Definiere die Konfliktzahl eines Zustands als Anzahl verletzter Constraints (Überlappungen, fehlende Sichtlinie, zu geringer Abstand usw.).
3. Wiederhole:
   (a) Wähle eine Variable, die an mindestens einem Konflikt beteiligt ist.
   (b) Setze sie auf den Wert in ihrer Domäne, der die Konfliktzahl minimal macht.
4. Falls nach einer vorgegebenen Schrittzahl keine konfliktfreie Belegung erreicht wurde, starte mit einer neuen zufälligen Belegung (Restart).

Vergleichend:

- MAC ist vollständig und zeigt auch Unlösbarkeit an, eignet sich gut für kleinere Instanzen mit starken Constraints.
- Min-Conflicts ist heuristisch, skaliert gut für große Instanzen und viele mögliche Lösungen, bietet aber keine Garantie, eine Lösung zu finden.
- Für realistische Hallen mit vielen möglichen Plazierungen kann Min-Conflicts sehr effizient sein; MAC liefert dagegen verifizierbare Korrektheit, ist aber bei sehr vielen Geräten potenziell langsamer.