Constraint Programming

Aufgaben Beschreiben statt selbst Lösungen programmieren

Heiko Spindler (IT-Berater)

1

Agenda

- Einführung in Constraint Programming
- Variablen und Constraints
- Ein erstes Beispiel
- Frameworks
- Abstrakte Sprachen für CP
- Beispiel: Prozess-Ablauf-Planung
- Fazit

2

Einsatz von Constraint Programming

3

Constraint Programming (CP)

- Ein Constraint Satisfaction Problem (CSP) ist eine Trippel ⟨X,D,C⟩, für das gilt:
  - $X=\{X_1, \ldots, X_n\}$ ist eine Menge von Variablen.
  - D ist eine Funktion, die jeder Variablen einen Wertebereich (Domain) zuweist.
  - C ist eine Menge von Bedingungen (Constraints).
- Ein CSP kann leicht in ein Constraint Optimization Problem (COP) umgewandelt werden:
  - Bewertung der gefundenen Lösungen vornehmen

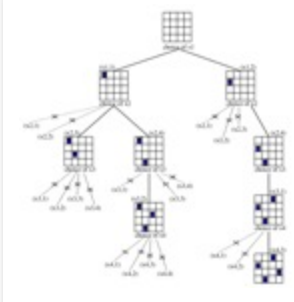
4

Bedingungen (Constraints)

- Gesucht werden für alle Variablen konkrete Werte aus dem jeweiligen Wertebereich (Domain) für die alle Bedingungen (Constraints) erfüllt sind.
- Eine Bedingung (Constraint) definiert eine Beziehung zwischen Variablen.

5

Wie arbeiten CP Solver?



6

Coding: Arithmetische Constraints

```
Model model = new Model("Pythagoras");

 // Define integer variables for a, b, and c
IntVar a = model.intVar("a", 1, 10); // Set bounds for demonstration
IntVar b = model.intVar("b", 1, 10);
IntVar c = model.intVar("c", 1, 100);

// Add the Pythagorean constraint (a^2 + b^2 = c^2)
c.mul(c).eq(a.mul(a).add(b.mul(b))).post();
//model.arithm(c.mul(c).intVar(), "=", a.mul(a).intVar(), "+", b.mul(b).intVar()).post();

// Print the solution if found
if (model.getSolver().solve()) {
    System.out.println("Solution found:");
    System.out.println("a: " + a.getValue());
    ...
}
```

7

Arten von Constraints

- **Arithmetische Constraints:**
  - definieren Beziehungen zwischen Variablen mit arithmetischen Operationen und Relationen (z. B. Gleichheit, Ungleichheit, kleiner als, größer als).
- **Logische Constraints:**
  - formulieren logische Beziehungen zwischen Variablen (z. B. UND, ODER, NICHT).
- **Globale Constraints:**
  - drücken komplexe Beziehungen zwischen mehreren Variablen aus, wie z. B. „allDifferent", „cumulative".
- **Tabellen Constraints:**
  - definieren erlaubte Wertekombinationen für einen Satz von Variablen.
- **Reguläre Ausdrücke**
- **Graphen**

8

Choco-solver

- Kostenlose Open-Source-Java-Bibliothek für Constraint-Programmierung.
- Der Benutzer modelliert sein Problem deklarativ, indem er die Constraints definiert, die in jeder Lösung erfüllt werden müssen
- Anschließend wird das Problem durch abwechselnde Constraint-Filteralgorithmen und einen Suchmechanismus gelöst.
- https://choco-solver.org/

9

Choco-Solver mit Python API

- https://pypi.org/project/pychoco/
- The pychoco library uses a *native-build* of the original Java Choco-solver library, in the form of a shared library, which means that it can be used without any JVM. This native-build is created with GraalVM native-image tool.

10

Alternative Frameworks

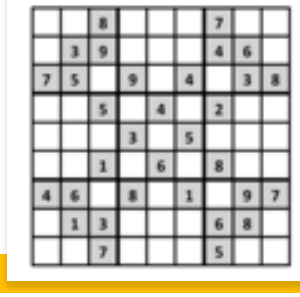
- JaCoP
  - „Java Constraint Programming solver"
  - https://github.com/radsz/jacop
  - Letzte Änderungen im Jahr 2023
- ILOG Solver (www.ilog.fr)
- GECODE (www.gecode.org)
- https://developers.google.com/optimization
- https://github.com/chuffed/chuffed

- Rust: https://github.com/ptal/pcp
- C++: https://github.com/pothitos/naxos

Und viele mehr …

11

Choco-Solver: Sokudo-Beispiel



13

Coding: Implementierung Sudoku

```
Model model = new Model("sudoku");
…

for (int row = 0; row != SIZE; row++) {
   for (int col = 0; col != SIZE; col++) {
      int value = predefinedRows[row][col];
      // is this an unknown? if so then create it as a bounded variable
      if (value < MIN_VALUE) {
         grid[row][col] = model.intVar(format("[%s.%s]", row, col), MIN_VALUE,
MAX_VALUE);
      } else {
         // otherwise we have an actual value, so create it as a constant
         grid[row][col] = model.intVar(value);
      }
   }
}
```

14

Coding: Implementierung Sudoku

```
for (int i = 0; i != SIZE; i++) {
   model.allDifferent(getCellsInRow(grid, i)).post();
   model.allDifferent(getCellsInColumn(grid, i)).post();
   model.allDifferent(getCellsInSquare(grid, i)).post();
}
…
Solver solver = model.getSolver();
solver.solve();
…
```

15

Begrenzen der Suche

- limitTime
  - Begrenzen nach vorgegebener Zeit
- limitSolution
  - Begrenzen nach Anzahl gefundener Lösungen

- Spezielle Begrenzungen
  - limitNode
  - limitFail
  - limitBacktrack

16

Abstrakte Sprachen für das Formulieren von Constraints

- MiniZinc is a free and open-source constraint modeling language
  - https://github.com/minizinc
  - https://www.minizinc.org/index.html
  - MiniZinc IDE: https://play.minizinc.dev/
- FlatZink
- XCSP (https://xcsp.org/)
- …

17

MiniZinc IDE Demo

18

Anwendungsbeispiel: Optimierung einer hypothetischen Prozess-Ablauf-Planung

- Prozesse bestehen aus sequenziellen Schritten.
- Jeder Schritt hat eine Dauer und einen Bedarf an Ressourcen.
- Je Zeiteinheit darf die max. Kapazität nicht überschritten werden (z.B. Energiebedarf, Bandbreite oder Speicherplatz).

Globales Ziel:

Alle anstehenden Prozesse so früh wie möglich vollständig abarbeiten.

19

Darstellung des Ablaufs von Schritten im Prozess

[A1][A1][A2][A2][A2][A3][A3][A3][A4][A4][A4][A5][A5][A5][__][__]
[A1][A1][A2][A2][A2][A3][A3][A3][A4][A4][A4][A5][A5][A5][__][__]
[A1][A1][A2][A2][A2][__][__][__][__][__][__][__][__][__][__][__]
[B1][B1][B2][B2][__][__][__][__][__][__][__][__][__][__][__][__]
[B1][B1][B2][B2][__][__][__][__][__][__][__][__][__][__][__][__]
[__][__][B2][B2][__][__][__][__][__][__][__][__][__][__][__][__]

- Spalten sind Zeiteinheiten
- Zeilen stehen für Bedarf an Ressourcen

20

Zwei Ansätze bei der Planung

Einfache Optimierung

- Prozesse und ihre Schritte werden sequenziell in den Arbeitsablauf eingefügt.
- Sucht eine passende Lücke für jeden neuen Task: So früh wie möglich.

➔ Lokale Optimierung mit wenig Aufwand

Globale Optimierung mit CP

- Alle Prozesse mit ihren Schritten sind bekannt für den Planungshorizont
- Volle Freiheit bei der Einplanung

➔ Globale Optimierung

23

Vergleich der beiden Verfahren

24

Bewertung

- Globale Optimierung mit CP nutzt die vorhandene Kapazität besser aus.
- Gewinn bei der Auslastung der Kapazität ca. 8 – 12%

25

Integration der Optimierung in eine größere Applikation

In den Anforderungen ist meist formuliert:

• Planung und Steuerung brauchen Zugriff auf alle Daten: Stammdaten (inkl. des aktuellen Zustands), Aufträge, …

Erster Impuls:
• Steuerungskomponente mit direktem Zugriff auf die Daten ausstatten.

Bei genauer Analyse:
• Es reicht meist eine reduzierte Sicht auf die Daten.

➔ Erstellen einer speziellen Sicht auf die „notwendigen" Informationen.
➔ Damit kann die Steuerungskomponente leichter separiert werden.
➔ Erleichtert die Simulation der Planung und Steuerung.

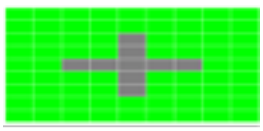
26

The Challenge

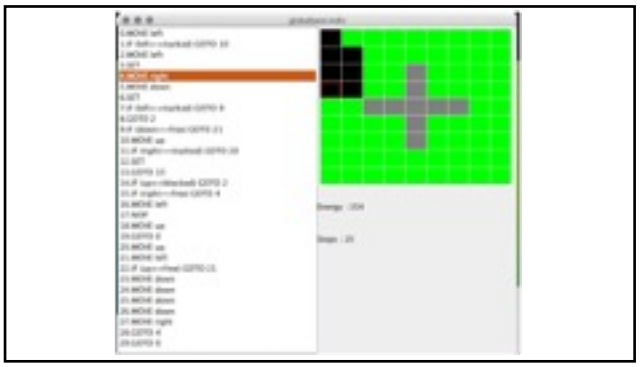
A robot should visit and mark all cells of a given board (9 x 9 cells).

Some cells are blocked (gray).

The memory holds up to 30 commands.



27



28

Simulierte Evolution

- Sinnvoll bei einem sehr großer Suchraum
- Es muss keine optimale Lösung sein (gut genug)
- Stark vom Zufall abhängig
- Konfiguration des Verfahrens schwierig: Viele Parameter
- Konvergenz ist nicht immer gesichert
- Aufgabe kann gut parallelisiert werden

Constraint Programming

- Sinnvoll bei einer überschaubaren Größe des Suchraums
- Beste Lösung, Auflisten aller Lösungen möglich
- Mehr Einfluss auf die Navigation im Suchraum
- Systematischere Suche
- Formale Beschreibung der Aufgabe

29

Fazit CP

- Bietet mächtige Möglichkeiten Aufgabenstellung formal und (möglichst) technik-neutral zu Beschreiben.
- Gute Möglichkeiten der Integration in Applikationen.
- Es gibt ausgereifte und eingeführte Frameworks.
- Bietet Möglichkeiten den Prozess der Lösungsfindung zu konfigurieren.

Feedback:

- Folien und Beispiele finden sich unter:

https://github.com/brainbrix/cpdemo

30