Search-Based Software Engineering

Sommersemester 2024, Übungsblatt 2
Multi-State Meta-Heuristiken & CSP

Diskussion am: **15.05.2024**

SOFTWARE SYSTEME
Prof. Dr.-Ing. Norbert Siegmund
Johannes Dorn, M. Sc.
Max Weber, M. Sc.
Stefan Jahns, B. Sc.

Aufgabe M1: Grundlagen

a) Was bedeuten (1 + 1), (1 + λ), (1, λ), (μ + λ) und (μ, λ)? Erklären Sie ihre Unterschiede. Gibt es eine Korrespondenz mit bekannten Algorithmen aus der Vorlesung?

b) Wie unterscheiden sich Multi-State-Meta-Heuristiken von mehreren parallel durchgeführten Hill-Climbing-Verfahren? Unter welchen Umständen kann man einen Vorteil von Multi-State-Meta-Heuristiken erwarten?

Aufgabe M2: Line Recombination

a) Wie funktioniert Line Recombination im Detail?

b) Wie unterscheiden sich Line Recombination und Intermediate Line Recombination?

c) Implementieren Sie Intermediate Line Recombination als Python-Funktion.

Aufgabe M3: Selection-Methoden

a) Was ist Fitness-Proportionate Selection (FPS) und wie funktioniert sie?

b) Was ist Stochastic Universal Sampling (SUS) und wie funktioniert es?

c) Implementieren Sie SUS als Python-Funktion.

Aufgabe M4: Implementierung

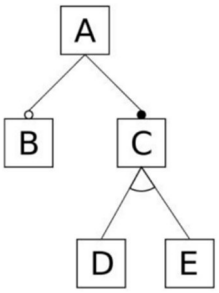
Implementieren Sie einen genetischen Algorithmus, um eine nah-optimale Konfiguration für BDBC und x264 zu finden. Nutzen Sie feature.txt und interactions.txt als Einflüsse für Features und deren Interaktionen. Für diese Aufgabe ist es nicht notwendig, Kandidaten auf Validität zu prüfen.

Berücksichtigen Sie folgende Komponenten:

a) Initialization procedure
b) Copy procedure
c) Tweak / mutation procedure
d) Selection procedure (nutzen Sie SUS aus Aufgabe 3)
e) Crossover / breeding procedure

Hinweise: Kleinere Performance-Werte sind besser. Eine Population-Size von 50 und max. 1000 Generationen sind realistische Werte zum Evaluieren Ihres Ansatzes.

Aufgabe C1: Feature-Modelle



a) Gegeben ist das folgende Feature-Diagramm eines konfigurierbaren Software-Systems A. Bestimmen und nennen Sie alle validen Konfigurationen.

b) Wandeln Sie das Feature-Diagramm in einen äquivalenten aussagenlogischen Ausdruck um.

Aufgabe C2: Automatisierte Analyse von Feature-Modellen

Feature-Modelle können in einigen Fällen inkonsistent modelliert sein. Zwei konkrete Probleme, welche häufiger auftreten, sind Dead Features und False Optional Features. Ein *Dead Feature* ist ein Feature, welches in *keiner* gültigen Konfiguration ausgewählt werden kann. Ein *False Optional Feature* ist ein Feature, welches in einem Feature-Diagramm zwar als optionales Feature modelliert wurde, jedoch in *jeder* gültigen Konfiguration ausgewählt ist.

Mittels eines CSP- bzw. SAT-Solvers lassen sich Feature-Modelle in vielfältiger Weise analysieren. In der Praxis wird dazu das Feature-Modell um Constraints erweitert und auf Erfüllbarkeit (Existenz mindestens einer gültigen Konfiguration) überprüft. Hierbei werden häufig Feature-Modelle in Form eines aussagenlogischen Ausdrucks in konjunktiver Normalform (KNF) verwendet.

a) Gegeben sei ein Feature-Modell mit den Features F_1, F_2, …, F_n. Das Feature-Modell liegt als aussagenlogische Formel Ψ (in KNF) vor. Wie lässt sich mittels eines Solvers überprüfen, ob das Feature-Modell Dead Features enthält?

b) Gegeben sei ein Feature-Modell mit den Features F_1, F_2, …, F_n. Das Feature-Modell enthält optionale Features *sowie* mandatorische Features, aber keine Cross-Tree Constraints. Wie können durch das Hinzufügen von Cross-Tree Constraints False Optional Features entstehen? Beschreiben Sie kurz ein mögliches Szenario.

c) Gegeben sei ein Feature-Modell mit den Features F_1, F_2, …, F_n. Das Feature-Modell enthält optionale Features *sowie* mandatorische Features. Das Feature-Modell liegt als aussagenlogische Formel Ψ (in KNF) vor. Wie lässt sich mittels eines Solvers überprüfen, ob das Feature-Modell False Optional Features enthält?

d) Im Moodle-Kurs finden Sie ein Feature-Modell zum Datenbanksystem “SQLite”. Dieses Feature-Modell liegt als konjunktive Normalform im DIMACS-Format vor. Ebenfalls im Moodle finden Sie ein Jupyter-Notebook, mit dessen Hilfe sich DIMACS-Dateien parsen lassen. Mit einem Solver, etwa Pycosat, können mit diesem Skript Lösungen für einen logischen Ausdruck berechnet werden.

e) Lesen Sie die DIMACS-Datei zu “SQLite” ein und verwenden Sie einen Solver wie Pycosat um die folgenden Fragen zu beantworten: Sind in diesem Feature-Modell Dead Features enthalten? Wenn ja, welche Features sind nicht auswählbar?

Aufgabe M5: Validität

Führen Sie Ihre Implementierung aus Aufgabe M4 für BDBC und x264 aus.

a) Überprüfen Sie, ob die Ergebnisse valide Konfigurationen sind.

b) Diskussion: Wie können Sie sicherstellen, dass Ihr genetischer Algorithmus nur valide Konfigurationen ergibt?

Aufgabe C3: Constraint Propagation und Forward Checking

Gegeben ist ein lateinisches Quadrat mit 3 x 3 = 9 Feldern. Das Constraint-Problem ist folgendermaßen definiert: Jedes Feld entspricht einer Variable, sodass es in diesem Beispiel 9 Variablen gibt, die entsprechend den Koordinaten indexiert sind. Die Domäne jeder Variable besteht aus der Menge {A, B, C}. Jeder Wert muss sowohl in jeder Zeile als auch in jeder Spalte *genau einmal* vorkommen. In der Grafik wird eine Teillösung vorgegeben, bei der die Variablen $V_{1,1}$ = A; $V_{1,2}$ = B; $V_{2,1}$= B; $V_{3,1}$ = C bereits Werte zugewiesen bekommen haben.

a) Erläutern Sie den Unterschied zwischen Forward Checking und Constraint Propagation.

b) Führen Sie Forward Checking für den aktuellen Stand der vorgegebenen Teillösung durch und tragen Sie das Ergebnis in ein Lösungsschritt-Quadrat ein. Markieren Sie dieses Quadrat entsprechend.

c) Wenden Sie den Algorithmus aus der Vorlesung für Constraint Propagation auf den aktuellen Stand der vorgegebenen Teillösung an. Notieren Sie einzelne Schritte in separaten Lösungsschritt-Quadraten. Möglicherweise benötigen Sie nicht alle Quadrate. Kennzeichnen Sie jeden Arc, durch welchen inkonsistente Werte erkannt werden. Geben Sie für jeden Schritt die Wertebereiche der Variablen an, ohne dabei eine Zuweisung vorzunehmen.