

Rekonfigurierbare Fertigungszellen (Re-configurable Manufacturing Cells)

Anwendung des Evolutionsverfahrens auf ein ökonomisch-technisches Optimierungsproblem – Projektdokumentation

Vorwort

Dieses Projekt ist ein Ergebnis des Forschungsvorhabens „Beispielhafte Anwendungen der Evolutionsverfahren auf technische Optimierungsprobleme“, für das mir die Hochschule im Sommersemester 2009 ein Forschungssemester gewährt hat. Aus dem F&E-Antrag:

„Ziel des ... Projekts ist weitestgehende Übertragung der KoopEgo-Strategie (Grams, 2007) auf ökonomisch-technische Optimierungsprobleme. Die Resultate sind für die Lehrveranstaltung *Problemlösen* (ET540) im Master-Studiengang *Systems Design & Production Management* (SDPM) relevant.

Im Kern geht es bei der KoopEgo-Strategie darum, wie sich die Kreativität mit Computerunterstützung steigern lässt. Zwei Ebenen der Kreativität sind zu unterscheiden:

1. Das Programm realisiert einen Evolutionsprozess. Dieser Sachverhalt wird durch das Oxymoron *automatisierte Kreativität* treffend beschrieben.
2. Die Software-Konstruktion legt die Evolutionsstrategie des Prozesses fest. Hier ist menschliche Kreativität gefragt.

Besonders Wert gelegt wird auf ein gutes Oberflächendesign des Programms, um eine gute Beobachtbarkeit und effiziente Steuerung des Evolutionsprozesses zu gewährleisten.“

Im hier dokumentierten Projekt wird die KoopEgo-Strategie auf ein praxisnahes Problem der Fertigungsplanung angewendet und demonstriert, inwieweit sich Evolutionsverfahren für die Lösung ökonomisch-technischer Optimierungsprobleme eignen. Die Problemstellung orientiert sich an Projekten der Firma FFT EDAG, Fulda.

Danksagung. Der Fachbereich Elektrotechnik und Informationstechnik der Hochschule Fulda hat dieses Vorhaben durch die Gewährung eines Forschungssemesters ermöglicht. Von der Firma FFT EDAG in Fulda, besonders von Herrn Patrik Möller, habe ich freundliche Unterstützung erfahren. Dank geht auch an die Studentinnen und Studenten meiner Praktika. Durch ihre Kritik, ihre guten Ideen, ihre Neugier, ihre Entdeckungen und auch durch ihre Fehler habe ich viel über das Programmieren gelernt.

Inhalt

Einführung	4
Rekonfigurierbare Fertigungszellen	5
<i>Fertigungsplanung – Begriffsbestimmungen</i>	5
Werkstücke	5
Arbeitsvorgänge	5
Restriktionen	6
Werkzeuge	6
Fertigungsplätze (Stationen)	6
Zielfunktionen	7
<i>Ein idealisiertes Modell der Fertigungszelle</i>	8
Mathematisches Modell	8
Beschreibungssprache für die Anforderungen	10
Beispiel 1: Anforderungen an eine fiktive Fertigungszelle	10
Optimierung mit exakten Methoden	11
Beispiel 2: Erweitertes Beispiel	12
Pflichtenblatt	14
<i>Allgemeine Beschreibung</i>	14
<i>Bedienoberfläche: Eingabe, Ausgabe</i>	14
<i>Verarbeitung</i>	15
Modellierung der Individuen (Representation of Individuals)	15
Variationsoperatoren (Variation Operators)	15
Bewertungsfunktionen (Evaluation Functions)	16
Auswahloperatoren (Selection Operators)	16
Behandlung von Randbedingungen (Constraint-Handling Techniques)	17
Visualisierung und interaktive Steuerung (Visualization and Control)	17
<i>Validierung, Treffsicherheit, Testszenarien, Effizienz</i>	17
Entwurf	18
<i>Entwurf der grafischen Oberfläche</i>	18
Das Steuerungsfenster	18
<i>Klassendiagramm</i>	18
<i>Eingabe</i>	18
Eingabe der Arbeitsschritte und Netzdarstellung	18
Topologische Sortierung der Arbeitsschritte und Schleifenerkennung	19
<i>Die Anfangsbelegung</i>	19
<i>Repräsentation der Welt und der Individuen</i>	19
Farbkodierung	19
Realisierung	20
<i>Parser: Dateneingabe und Anlegen der Strukturdaten für die Fertigungszelle</i>	20
Die find-Methode der Element-Klasse	20
Die topologicalSort-Methode der Element-Klasse	20
Verifikation der topologicalSort-Methode (Programmbeweis)	21
<i>Nachbarschaft und Feldauswahl</i>	21
<i>Algorithmen für Variationsoperatoren</i>	21
Selektion	21
Rekombination	22
Mutationsverfahren	22
<i>Bewertungsoperatoren</i>	22
<i>Die zentrale Schleife: Plan.run()</i>	23
Experimente	23

1. Versuchsserie: Grundlegenden Optimierungsstrategien am Beispiel 1	23
Optimierungsziele und Prognose	23
Mutations-Selektionsverfahren (n=1, k=0, T=0)	23
Simulated Annealing (variables T).....	24
Parallele Mutation-Selektion (n=40, k=0)	24
Crossing-Over-Verfahren mit globaler Paarung (n=40, k=20)	24
Crossing-Over-Verfahren mit lokaler Paarung (n=40, k=3).....	24
2. Versuchsserie: Effizienz der Optimierungsstrategien am Beispiel 2	26
Voreinstellungen	26
Optimierungsziele und Prognose	26
Mutations-Selektionsverfahren (n=1, k=0, T=0)	26
Mutations-Selektionsverfahren mit Simulated Annealing (variables T).....	27
Parallele Mutation-Selektion (n=40, k=0, variables T).....	27
Crossing-Over-Verfahren mit globaler Paarung (n=40, k=20, variables T).....	27
Crossing-Over-Verfahren mit lokaler Paarung (n=40, k=3, variables T).....	27
3. Versuchsserie: Die Beherrschung des Chaos (gemessen am 2. Beispiel).....	28
Voreinstellungen	28
Parallele Mutation-Selektion (n=40, k=0)	28
Crossing-Over-Verfahren mit globaler Paarung (n=40, k=20)	29
Crossing-Over-Verfahren mit lokaler Paarung (n=40, k=3).....	29
Ergebnisse	30
<i>Einfluss der Variationsoperatoren</i>	30
<i>Regeln für die Auswahl der Optimierungsstrategie</i>	30
<i>Regeln für die Steuerung der Optimierung (Wahl der Temperatur T)</i>	30
Zeitbedarf	30
<i>Messen am System</i>	30
<i>Komplexitätsanalyse</i>	31
Weiteres Vorgehen	31
<i>Anstehende Entscheidungen und Untersuchungen</i>	31
<i>Schritte hin zu einem produktiven Programm</i>	31
<i>Durchführung</i>	32
Literatur	32

Einführung

Die Optimierung technischer Systeme nach Prinzipien der biologischen Evolution gehört spätestens seit dem Erscheinen des Buches „Evolutionstrategie“ von Ingo Rechenberg (1973) auch für Ingenieure zu den Mitteln der Wahl. Erfahrungen mit solchen „direkten Optimierungsmethoden“ konnte ich bereits während meiner Industrietätigkeit im Zusammenhang mit der Optimierungssoftware für elektrische Schaltungen sammeln. Als Betreuer von Diplomarbeiten brachte ich diese Methoden immer wieder zur Sprache. Zu einer besonders interessanten Anwendung kam es mit der Arbeit „Computersimulation von Fahrzeugbewegungen in landwirtschaftlichen Betrieben“. Sie wurde auf der CeBit im Jahr 1991 vorgestellt – übrigens der erste CeBit-Auftritt des Fachbereichs AI.

Heute laufen solche Arbeiten unter den Titeln „Genetische Algorithmen“ (Holland, 1992), „Evolutionäre Algorithmen“ und „Moderne Heuristiken“ (Michalewicz, Fogel, 2000). Mit den inzwischen verfügbaren gewaltigen Rechenleistungen bietet sich die Chance, auf der Grundlage sehr einfacher Regeln neuartige Lösungen für technische Probleme zu finden, Lösungen, auf die man allein durch angestregtes Nachdenken nicht kommen würde.

Evolutionsverfahren werden beim Entwurf von Systemen, deren Zielgrößen von sehr vielen Parametern abhängen, und bei denen diese Abhängigkeiten nur schwer zu durchschauen sind, weiter an Bedeutung gewinnen. Ein Beispiel für eine solche Problemlage ist die Optimierung von Videosystemen.

Hier geht es um ein Thema der Fabrikplanung, speziell um die *Fließfertigung* im Automobilbau mit festen Taktzeiten für die einzelnen Arbeitsstationen.

In vielen Fällen läuft die Fertigungs- oder Maschinenbelegungsplanung darauf hinaus, eine Menge von Objekten in eine optimale zeitliche oder räumliche Reihenfolge zu bringen. Solche Objekte können Aufträge (Jobs) oder Arbeitsschritte (Tasks) sein. Die Objekte unterliegen Randbedingungen hinsichtlich der verfügbaren Ressourcen (Werkzeuge und Maschinen) und der Reihenfolge. Beispielsweise können zwei Teile erst dann miteinander verschweißt werden, wenn sie vorher zusammengeführt worden sind. Die Optimalität misst sich an Bewertungsfunktionen wie Zeitbedarf und materiellem und personellem Aufwand.

Das Musterproblem aus dem Problemkreis der Planung von Reihenfolgen ist das Handlungsreisenden-Problem (Traveling Salesman Problem). Generationen von Unternehmensforschern haben sich die Zähne daran ausgebissen, denn es handelt sich um ein *schweres Problem*. Es ist schwer in dem Sinn, dass der Aufwand für das Finden einer optimalen Lösung mit der Anzahl n der anzuordnenden Objekte gewaltig steigt. Präzise: Der Aufwand – die Zahl der Iterationsschritte – lässt sich nicht durch $A \cdot n^k$ begrenzen, egal wie groß man die Werte A und k auch wählt.

Ein nahe liegender Algorithmus zum Auffinden der optimalen Anordnung ist das Ausprobieren aller $n!$ möglichen Anordnungen. Das ist die Methode der *vollständigen Enumeration*. Bereits bei einer mäßig großen Anzahl von Objekten ist sie praktisch unmöglich durchführbar. Für $n=170$ lässt sich die Anzahl der möglichen Anordnungen, die Zahl der Permutationen, gerade noch mit dem Double-Format darstellen. Es handelt sich um eine 307-stellige Dezimalzahl, eine unvorstellbare Größe. Könnte ein Computer eine Milliarde Anordnungen je Sekunde prüfen, hätte er nach zwanzig Milliarden Jahren erst einen verschwindend kleinen Teil der Arbeit erledigt, darstellbar als 27-stellige Dezimalzahl.

Die Suche nach Verfahren, die den Aufwand für die Optimumsuche auf ein erträgliches Maß reduzieren, hat bisher nichts Ermutigendes zu Tage gefördert.

Evolutionsverfahren versprechen zwar nicht das Optimum, aber man kann damit gute Lösungen in überschaubarer Zeit finden. Deshalb gehören die schweren Probleme zu den typischen Anwendungen für Evolutionsverfahren.

Die Teilziele des Projekts *Rekonfigurierbare Fertigungszellen* (Re-configurable Manufacturing Cells) sind

1. Formulierung eines Modells, das in idealisierter Form die für die Planung von rekonfigurierbaren Fertigungszellen relevanten Strukturelemente enthält;
2. Konstruktion eines Java-Programms, mit dem gute Lösungen für das abstrakte Modell gefunden werden;
3. Abschätzung der Programmlaufzeiten in Abhängigkeit vom Problemumfang;
4. Aufzeigen von Entwicklungsmöglichkeiten für das Programm, so dass damit reale Probleme behandelt werden können; Aufwandsabschätzung für die Programmentwicklung.

Literaturhinweise: Was ein schweres Problem (NP-hard problem) ist, erläutert Dewdney (1993) in seinem „Turing Omnibus“. Einzelheiten der Komplexitätsanalyse und eine stärkere Hinwendung zur Praxis bietet „The Design and Analysis of Computer Algorithms“ von Aho, Hopcroft und Ullman. Hinsichtlich der Terminologie der Fertigungssysteme lehne ich mich an das Buch von Dangelmaier (2001, Abschnitt „6.2 Anordnungsplanung“, S. 312-317) an. Anhaltspunkte zur Modellierung und zur Klassifizierung der Modelle bieten Neumann und Morlock (1993, Abschnitt „3.6 Maschinenbelegungsplanung“, S. 474-506). Das Traveling Salesman Problem ist im Buch von Michalewicz und Fogel (2000) ein Musterbeispiel für die Anwendung von Evolutionsverfahren.

Rekonfigurierbare Fertigungszellen

Fertigungsplanung – Begriffsbestimmungen

Werkstücke

Beispiele für Werkstücke (Teile, Parts) im Automobilbau sind

- Bodenplatte (Floor Panel)
- Verstärkungsteile (Seat Spring Reinforcement, ...)
- Muttern (Rear Axle Nuts, ...)
- ...

Arbeitsvorgänge

Die Planung und Optimierung rekonfigurierbarer Fertigungszellen ist ein *Spezialfall der Arbeitsplanerstellung für Fertigungssysteme*. Gemeinsam ist diesen Planungsprozessen, dass es eine Menge von Grundelementen gibt. Diese Grundelemente werden – je nach betrieblichem Kontext – Einzelbearbeitungselemente, Fertigungsvorgänge, Arbeitsvorgänge, Tasks oder Arbeitsschritte genannt. Es gibt die folgenden Arten von *Arbeitsvorgängen* für rekonfigurierbare Fertigungszellen:

- Fördern und Handhaben (Handling)
- Zusammenführen, Ablegen und Spannen von Teilen. Unterschieden werden einfache Ablage ohne Spanneinrichtung (Deposit), passgenaues Verspannen (Geostation) und leichtes Verspannen (Respot)
- Fügen: Falzen, Schweißen (Welding), Kleben

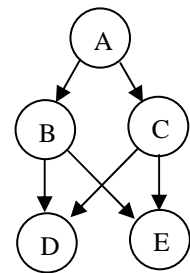
Ziel der Fertigungsplanung ist ein *Arbeitsplan*, der die Reihenfolge der Arbeitsvorgänge und damit den Fertigungsablauf festlegt (Dangelmaier, 2001, Abschnitt „7.3 Arbeitsplanerstellung“, S. 469).

Jedem Arbeitsvorgang ist ein Zeitbedarf zugeordnet. Diese Zeit umfasst das jeweilige Anfahren des Zielpunktes und die Ausführung. Je Schweißpunkt sind das etwa drei Sekunden.

Restriktionen

Nicht alle möglichen Folgen von Arbeitsvorgängen sind zulässige Repräsentationen von Arbeitsplänen, denn Arbeitspläne unterliegen *aufgabenspezifischen Restriktionen* hinsichtlich der Reihenfolge, beispielsweise derart, dass ein bestimmter Arbeitsvorgang *B* erst nach Erledigung des Arbeitsvorgangs *A* begonnen werden kann. Diese Reihenfolgebeziehung prägt der Menge der Arbeitsvorgänge eine partielle Ordnung auf, die durch einen zyklensfreien Digraphen darstellbar ist.

Dangelmaier (2001, S. 477 ff.) gibt ein Beispiel aus der Fertigung eines rotations-symmetrischen Werkstücks. Der stark vereinfacht wiedergegebene *Bearbeitungsgraph* (Digraph) drückt aus, dass es egal ist, ob zuerst die linke Seite des Werkstücks bearbeitet wird (Arbeitsschritte *B* und *D*) oder die rechte (Arbeitsschritte *C* und *E*). Andererseits muss das Schruppen des Werkstücks (*B* und *C*) dem Schlichten (*D* und *E*) vorangehen. Schön wäre der Arbeitsplan *ABDCE*. Bei ihm müsste das Werkstück nur einmal umgespannt werden. Er ist leider unzulässig, da er die Restriktion bezüglich der Reihenfolge von *C* und *D* verletzt. Beim zulässigen Arbeitsplan *ABCDE* fällt der Umspannvorgang gleich dreimal an. Günstiger ist der (zulässige) Arbeitsplan *ABCED* mit nur zwei Umspannvorgängen.



In einer Fertigungszelle der hier zu behandelnden Art geht es um die Montage von Fahrzeugteilen. Restriktionen ergeben sich dabei hauptsächlich aus der Notwendigkeit einer Synchronisation von Teileflüssen (Dammelaier, 2001, S. 509): Teile können erst zusammengefügt werden, wenn sie zusammengeführt und gemeinsam gespannt sind.

Werkzeuge

Durch den Arbeitsvorgang ist die Art des benötigten Werkzeugs festgelegt. *Werkzeugarten* sind:

Handhabung und Transport: Transportband, Operator, Roboter, Gabelstapler, ...

Fügeoperationen: Schweißzangen, Klebepistolen, Bolzenschweißpistolen, Schutzgasschweißpistolen, Rollfalkköpfe, ...

Fertigungsplätze (Stationen)

Die zu planende Fertigungszelle hat einen Eingangs- und einen Ausgangsspeicher und eine Reihe von Fertigungsplätzen oder *Stationen* (Dangelmaier, 2001, S. 506 ff.). Jede der Stationen besteht im Wesentlichen aus einem Roboter für Bearbeitung und Handhabung. Zwischen den Stationen gibt es Ablagen oder Transporteinrichtungen.

Das folgende Bild (Abdruck mit der freundlichen Genehmigung durch FFT EDAG) gibt einen Eindruck vom Aufbau einer solchen Fertigungszelle wieder. Eingezeichnet ist außerdem der Weg eines Werkstücks. An der Peripherie der Zelle werden an mehreren Stellen Teile hinzugeführt.

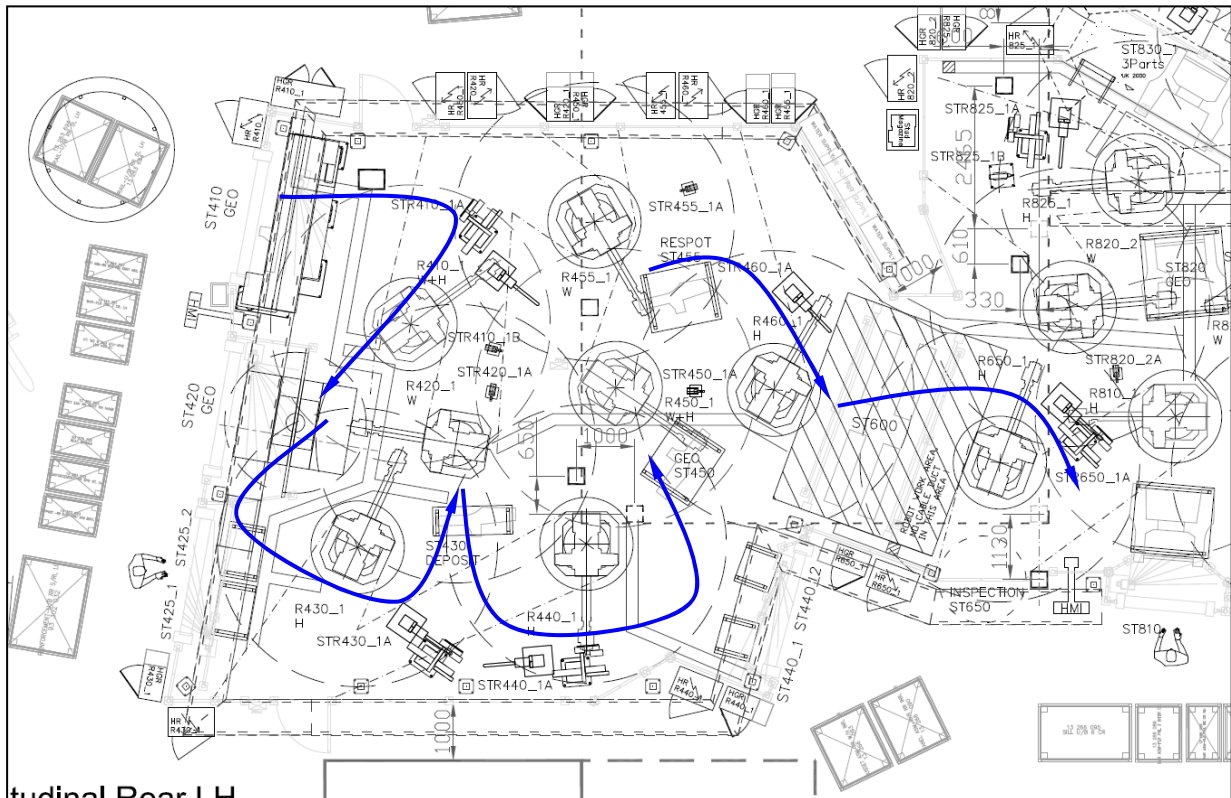
Das Hauptwerkstück, dem weitere Teile hinzugefügt werden, durchläuft die Stationen in einer Reihenfolge, die durch den Arbeitsplan bestimmt ist. Jede Station arbeitet einen zusammenhängenden Abschnitt des Arbeitsplans ab. Das heißt: Es werden der Reihe nach Arbeitsvorgänge durchgeführt, die im Arbeitsplan unmittelbar aufeinander folgen. Dafür muss sie mit den entsprechenden Werkzeugen für die Arbeitsvorgänge dieses Abschnitts ausgerüstet sein.

Danach wird das Werkstück an die folgende Station weitergegeben, die die Arbeitsvorgänge des unmittelbar folgenden Arbeitsplanabschnitts erledigt.

Es sind *stationspezifische Restriktionen* wirksam:

- Art und maximale Anzahl der Werkzeuge sind für jede Roboterart festgelegt.
- Die Menge der durchführbaren Arbeitsvorgänge ist durch die Taktzeit begrenzt. Die Taktzeit wird vom Auftraggeber vorgegeben und stellt eine feste Randbedingung dar.
- Weitere Restriktionen resultieren aus der maximal zulässigen Last, der Erreichbarkeit der Arbeitspunkte usw.

Die wesentliche Zielgröße für die Planung und Optimierung der Fertigungszelle sind die durch Roboter und Transporteinrichtungen verursachten Kosten.



Allokation der Betriebsmittel: Der Arbeitsplan lässt sich fortschreitend von vorn nach hinten in Abschnitte zerlegen derart, dass alle Arbeitsvorgänge eines Abschnitts durch eine Station erledigt werden können. Die Länge eines Abschnitts ist begrenzt durch die Werkzeuge, über die eine Arbeitsstation verfügen kann und durch die Taktzeit. Sämtliche stationsbezogenen Restriktionen müssen im Zuge der Abschnittsbildung berücksichtigt werden. Die Allokation von Betriebsmitteln und menschlicher Arbeitskraft (Potentialfaktoren) ist selbst wieder ein *Optimierungsproblem*, das im Zuge der Bewertung eines Arbeitsplans gelöst werden muss.

Zielfunktionen

Jeder Arbeitsplan zieht eine spezifische Allokation von Betriebsmitteln und menschlicher Arbeitskraft nach sich. Diese Allokation ist Ergebnis eines *untergeordneten Optimierungsprozesses*: Die Ermittlung der minimalen Kosten eines Arbeitsplans verlangt die optimale Allokation von Betriebsmitteln und menschlicher Arbeitskraft unter Berücksichtigung der stationsbezogenen Restriktionen. Daraus ergeben sich die produktionsspezifischen Kosten des Arbeitsplans.

Da im Rahmen des Optimierungsprozesses auch unzulässige Arbeitspläne auftreten können, kommen zu den Arbeitsplankosten *fiktive Kosten* aufgrund der Verletzung von aufgabenspezifischen Restriktionen hinzu.

Die Optimierungsaufgabe besteht darin, Arbeitspläne zu finden, die sämtliche aufgabenspezifischen Restriktionen einhalten und einen möglichst kleinen Zielfunktionswert besitzen.

Die relative Gewichtung von Restriktionsverletzungen und Betriebsmittelkosten ist im Zuge der Optimierung interaktiv und erfolgsorientiert abzustimmen.

Erläuterung: Unzulässige Arbeitspläne werden im Rahmen der Optimierung nicht definitiv ausgeschlossen. Dadurch sollen möglichst viele Wege hin zu guten Lösungen offen gehalten werden.

Ein idealisiertes Modell der Fertigungszelle

Mathematisches Modell

Das Modell gibt in äußerst stark vereinfachter Form die Planungsaufgabe für rekonfigurierbare Fertigungszellen wieder.

Es ist definiert durch eine Menge A von Arbeitsvorgängen, auf der eine Vorrangrelation $R \subset A \times A$ definiert ist, eine Menge T der verfügbaren Werkzeugtypen, eine Menge S der verfügbaren Stationstypen und die Taktzeit d_{Takt} .

Jedem Arbeitsvorgang $a \in A$ ist ein Zeitbedarf $a.d \in \mathbb{R}$ zugeordnet¹, ferner eine Menge von Werkzeugtypen $a.t \subset T$. Das sind die Typen der für die Bearbeitung erforderlichen Werkzeuge.

Jeder Stationstyp $s \in S$ wird vor allem durch seine Werkzeugausstattung $s.t$ charakterisiert: $s.t \subset T$.

Die Anzahl der Arbeitsvorgänge ist $n = |A|$. Die Vorrangrelation R ist als zyklensfreier Digraph darstellbar und definiert eine partielle Ordnung der Arbeitsvorgänge².

Ein Arbeitsplan p ist eine Folge aus sämtlichen Arbeitsvorgängen: $p = a_1 a_2 a_3 \dots a_n$. Die $a_i \in A$ sind also paarweise voneinander verschieden und sie schöpfen die Menge der Arbeitsvorgänge A aus. Es gibt $n!$ verschiedene Arbeitspläne.

Ein Arbeitsplan heißt zulässig, wenn für jede Vorrangbeziehung $(a, b) \in R$ gilt, dass im Arbeitsplan der Arbeitsvorgang a dem Arbeitsvorgang b vorausgeht³.

Ein Arbeitsvorgang a ist auf einer Station vom Typ s durchführbar, wenn $a.t \subset s.t$.

Für die Bewertung eines Arbeitsplans $p = a_1 a_2 a_3 \dots a_n$ werden mehrere Bewertungsfunktionen f_0, f_1, f_2, \dots definiert. Diese Funktionen werden durch lineare Kombination zu einer Gesamtbewertungsfunktion zusammengefasst: $f(p) = f_0(p) + \alpha_1 f_1(p) + \alpha_2 f_2(p) + \dots$. Die nichtnegati-

¹ In diesem Dokument auch für die Beschreibung mathematischer Funktionen einfacher Objekte eine objektorientierte Notation verwendet. Anstelle der in der Mathematik üblichen Funktionsschreibweise $f(x)$ steht dann $x.f$. Die mathematische Funktionsschreibweise wird für die Bewertungsfunktionen reserviert.

² Für $a, b \in A$ gilt $a < b$ genau dann, wenn es eine Folge von Vorrangbeziehungen $(a, x_1), (x_1, x_2), (x_2, x_3), \dots, (x_k, b) \in R$ gibt. Anders ausgedrückt: Es ist $a < b$ genau dann, wenn in dem Digraphen der Vorrangrelation ein Pfad vom Knoten a zum Knoten b existiert.

³ Ein Arbeitsplan ist genau dann zulässig, wenn er topologisch sortiert ist, d.h. wenn die partielle Ordnung in die lineare Ordnung des Arbeitsplans eingebettet ist: $a_i < a_j$ impliziert $i < j$.

ven Gewichtungsfaktoren $\alpha_1, \alpha_2, \dots$ sind Parameter, die im Rahmen der Optimierung ergebnisorientiert festgelegt werden können.

Die Funktion f_0 erfasst die Anzahl der Verletzungen der Vorrangrelation:

$$f_0(p) = f_0(a_1 a_2 a_3 \dots a_n) = \sum_{i < j} ((a_j, a_i) \in R)$$

Zur Schreibweise: Ein logischer Ausdruck wie $(a, b) \in R$ erhält in arithmetischen Ausdrücken den Zahlenwert 1, wenn er wahr ist, und andernfalls den Wert 0.

Für jeden zulässigen Arbeitsplan p gilt $f_0(p) = 0$.

Die Funktion f_1 ist gleich der Anzahl der für die Abarbeitung eines Arbeitsplans p minimal nötigen Stationen. (Eine zukünftige mögliche Verfeinerung dieser Funktion besteht darin, dass man alle wesentlichen Kosten der Fertigungszelle erfasst.)

Eine näherungsweise Berechnung dieser Funktion kann so vor sich gehen:

1. Setze $f_1 := 0$.
2. Teile den Arbeitsplan p in einen möglichst langen Anfangsabschnitt u und den Restabschnitt r auf: $p = ur$. Dabei ist die Randbedingung einzuhalten, dass der Anfangsabschnitt $u = u_1 u_2 u_3 \dots u_k$ die Taktzeit einhalten muss: $\left(\sum_{i=1}^k u_i \cdot d \right) \leq d_{\text{Takt}}$.
3. Wähle einen Stationstyp s mit einer Werkzeugausstattung für sämtliche Arbeitsvorgänge des Abschnitts u : $u_i \cdot t \subset s \cdot t$ für $i = 1, 2, \dots, k$.
4. Falls es keinen solchen Stationstyp gibt, reduziere k um eins ($k := k-1$) und passe die Abschnitte u und r entsprechend an. Dann gehe zurück zu Punkt 3.
5. Erhöhe den Zähler für Stationen um eins: $f_1 := f_1 + 1$.
6. Falls r leer ist, beende die Rechnung. Ansonsten setze $p := r$ und fahre bei Punkt 2 fort.

Weitere Funktionen lassen sich im Hinblick auf das konkrete Optimierungsproblem definieren. Mit der Funktion f_2 kann man beispielsweise dafür sorgen, dass Stationen mit Ladeoperationen möglichst spät im Arbeitsplan erscheinen. Auf diese Weise würden die Materialflüsse innerhalb der Zelle minimiert und die Positionierung der Stellen für das Zuführen von Teilen besser über die Peripherie der Fertigungszelle verteilt. Wichtiger noch ist dieser Aspekt: Je später Teile hinzugefügt sind, desto besser ist die Zugänglichkeit der Arbeitspunkte am Werkstück. Auf diese Weise lassen sich allerdings harte Anforderungen an die Zugänglichkeit nicht formulieren. Dafür sind Vorrangbeziehungen besser geeignet.

Diesen Überlegungen folgend, wird die Bewertungsfunktion f_2 eingeführt. Sie erfasst je Ladevorgang Gutschriften für alle die Arbeitsvorgänge, die vor diesem Ladevorgang liegen. Zu diesem Zweck braucht man nur die Position eines Ladevorgangs negativ zu verbuchen. Diese Werte werden über alle Ladevorgänge akkumuliert. Beispiel: Im Arbeitsplan $a_1 a_2 a_3 \dots a_{100}$ seien die Arbeitsvorgänge 5, 22, 57, und 88 mit Fügeoperationen verbunden, dann wäre $f_2 = -(5+22+57+88) = -172$.

Beschreibungssprache für die Anforderungen

Die Beschreibung der Anforderungen an die Fertigungszelle geschieht als (linearer) Text. Die Grammatik der Beschreibungssprache wird folgendermaßen mittels erweiterter Backus-Naur-Form (EBNF) festgelegt:

```
Requirements = TaktTime { StructureElement }
StructureElement = StationType | ProcessElement
TaktTime = Integer-Constant ;
StationType = S { ToolType } [ weight ];
weight = Integer-Constant
ProcessElement = Ident ( Duration ) [ Predecessors ] ;
Duration = Integer-Constant
Predecessors = Predecessor { , Predecessor }
Predecessor = Ident
Ident = Parts _ Tools [ _ No ]
Parts = Part { Part }
Tools = Tool { Tool }
Part = P IdentNo
Tool = ToolType [ Steps ]
IdentNo = Integer
Steps = Integer
No = Integer
ToolType = l | w | s | ...
```

Startsymbol ist *Requirements*. Die Terminalsymbole l, w und s stehen für Laden/Spannen (loading), Punktschweißen (spot welding) und Bolzenschweißen (stud welding). Der Anwender ist frei in der Wahl dieser Kleinbuchstaben und in der Bedeutungszuordnung. Die Dauer eines Arbeitsvorgangs (*Duration*) wird in Sekunden angegeben. Das Gewicht (*weight*) einer Station wird zusätzlich eingeführt, um die relativen Kosten zu bewerten. Es ist eine Angabe in Prozent. Fehlt diese Gewichtsangabe, wird das Gewicht auf 100% gesetzt: „Plw“ ist gleichbedeutend mit „Plw 100“. Die Vorgänger (*Predecessors*) eines Arbeitsvorganges sind alle die Arbeitsvorgänge, der vorher erledigt sein müssen. Die Integer-Zahl hinter dem Unterstrich einer Vorgangsidentifizierung (*Ident*) dient der Unterscheidung von Vorgängen mit ansonsten gleicher Vorgangsidentifizierung. Beispielsweise könnte man damit die Vorgänge allesamt einfach durchnummerieren.

Die Anzahl der Schritte (*Steps*) ist gleich der Anzahl der Einzelvorgänge, beispielsweise die Anzahl der Schweißpunkte in diesem Arbeitsschritt. Sie haben programmintern keine Bedeutung und stellen quasi Kommentare dar. Weitere Kommentare können am Anfang des Dokuments und hinter jedem Semikolon stehen. Diese weiteren Kommentare beginnen mit der Zeichenfolge /* und enden mit der Zeichenfolge */. Wo ein Kommentar stehen kann, dürfen auch mehrere stehen – allerdings immer schön nacheinander, Schachtelung ist nicht erlaubt.

Beispiel 1: Anforderungen an eine fiktive Fertigungszelle

Es werden Anforderungen an eine fiktive Fertigungszelle formuliert. Bei den Zeiten wird ein Lade- und Spannungsvorgang mit 25 Sekunden und das Ausführung einer Punkt- oder Bolzenschweißung einschließlich der Bewegung des Schweißwerkzeugs mit 3 Sekunden veranschlagt. Die Vorbereitung (beispielsweise Lageänderung) einer Serie von Schweißpunkten wird mit 5 Sekunden veranschlagt.

Das Beispiel ist in Anlehnung an die Arbeitsschritte einer realen Planung einer Fertigungszelle entwickelt worden. Etwa die Hälfte der tatsächlichen Fertigungsschritte ist in das Beispiel

eingegangen. Die Gruppierung von Schweißpunkten ist aus dieser Planung einfach übernommen worden und nicht zwingend.

```
/******  
Anforderungen an eine fiktive Fertigungszelle  
******/  
  
/*Takzeit*/  
70;  
  
/*Verfuegbare Stationstypen*/  
Sw;  
Ss;  
Sws;  
Slw 120;  
Sls 120;  
Slws 130;  
  
/*Lade- und Fuegevorgaenge*/  
P1P2_lw2_1 (31);  
P1P3_lw4_1 (37);  
P1P4_lw6_1 (43) P1P2_lw2_1;  
P1P5_lw4_1 (37);  
P1P6_lw10_1 (55);  
P1P8_lw9_1 (52);  
  
/*Schweissvorgaenge*/  
P1P2_w3_2 (14) P1P2_lw2_1;  
P1P2_w3_3 (14) P1P2_lw2_1;  
P1P2_w9_4 (32) P1P2_lw2_1;  
P1P2_s1_5 (8) P1P2_lw2_1;  
P1P2_w1_6 (8) P1P2_lw2_1;  
P1P3_w1_2 (8) P1P3_lw4_1;  
P1P3_w6_3 (23) P1P3_lw4_1;  
P1P3_w3_4 (14) P1P3_lw4_1;  
P1P3_w1_5 (8) P1P3_lw4_1;  
P1P5_w5_2 (20) P1P5_lw4_1;  
P1P6_w3_2 (14) P1P6_lw10_1;  
P1P6_w11_3 (38) P1P6_lw10_1;  
P1P6_w7_4 (26) P1P6_lw10_1;  
P1P6_w10_5 (35) P1P6_lw10_1;  
P1P8_w5_2 (20) P1P8_lw9_1;  
P1P5P8_w5_1 (20) P1P5_lw4_1, P1P8_lw9_1;
```

Optimierung mit exakten Methoden

Das Beispiel 1 ist klein genug, so dass es sich noch mit exakten Methoden bearbeiten lässt. Dadurch ist es möglich, zu überprüfen, ob das Evolutionsverfahren überhaupt grundsätzlich tauglich ist. Dem Algorithmus wird die Invariante $f_0 = 0$ zugrunde gelegt. Das heißt: Die Verletzung von aufgabenspezifischen Restriktionen ist von vornherein nicht zugelassen. Nur die Bewertungsfunktion f_1 wird bei der Optimierung berücksichtigt.

Informale Beschreibung des Algorithmus:

1. Es werden nur Stationen vom Typ Slws eingeplant. Damit lassen sich grundsätzlich alle Typen von Arbeitsvorgängen bearbeiten.
2. Für die Lade- und Fügevorgänge wird eine (neue) Reihenfolge festgelegt.
3. Für jeden der Lade- und Fügevorgänge wird eine eigene Station eingeplant.

4. Die eingeplanten Stationen werden der Reihe nach mit weiteren Aufträgen optimal aufgefüllt. Dabei werden je Station nur die jeweils zulässigen Aufträge berücksichtigt.
5. Falls die Menge der noch nicht eingeplanten Arbeitsvorgänge nicht leer ist, wird eine weitere Station angefügt und optimal mit Aufträgen aufgefüllt. Dieser Vorgang wird so oft wie nötig wiederholt.
6. Falls noch nicht alle Möglichkeiten der Anordnung von Lade- und Fügevorgängen ausgeschöpft sind, geht es mit Punkt 2 weiter.

Der Algorithmus basiert auf der *vollständigen Enumeration*. Da sich die Enumeration auf die sechs Lade- und Fügevorgänge beschränkt, sind für das 1. Beispiel nur 720 verschiedene Anordnungen zu erzeugen.

Das Auffüllen in den Schritten 4 und 5 geschieht mit dem rekursiven *Knapsack-Algorithmus* (Aho, Hopcroft, Ullman, 1983, S. 67). Im Beispiel ist die Anzahl der zu „verstauenden Teile“ maximal gleich sechzehn. Entsprechend begrenzt ist die Aufruftiefe der Rekursion im Knapsack-Algorithmus. Wegen der begrenzten Kapazität einer jeden Station ist die Aufruftiefe noch deutlich geringer.

Beispiel 2: Erweitertes Beispiel

Beispiel 2 geht es aus dem Beispiel 1 durch Vervielfachung hervor. Die Anzahl der Arbeitsvorgänge wurde vervierfacht, von 22 auf 88. Die Bezeichnungen sind im Hinblick auf die Auswertung der Experimente vereinfacht und systematisiert. Es wird eine kostenlose Scheinstation mit einem „unmöglichen Werkzeug“ z eingeführt. Damit lassen sich Voraussetzungen bündeln.

```
/*Takzeit*/
70;

/*Verfuegbare Stationstypen*/
Sz 0; //"Scheinstation" zur Synchronisation
Sw;
Ss;
Sws;
Slw;
Sls;
Slws;

/*Lade- und Fuegevorgaenge Serie 100*/
P_lw_101 (31);
P_lw_102 (37);
P_lw_103 (43) P_lw_101;
P_lw_104 (37);
P_lw_105 (55);
P_lw_106 (52);

/*Schweissvorgaenge Serie 100*/
P_w_111 (14) P_lw_101;
P_w_112 (14) P_lw_101;
P_w_113 (32) P_lw_101;
P_s_114 (08) P_lw_101;
P_w_115 (08) P_lw_101;
P_w_116 (08) P_lw_102;
P_w_117 (23) P_lw_102;
P_w_118 (14) P_lw_102;
P_w_119 (08) P_lw_102;
P_w_120 (20) P_lw_104;
P_w_121 (14) P_lw_105;
P_w_122 (38) P_lw_105;
```

P_w_123 (26) P_lw_105;
P_w_124 (35) P_lw_105;
P_w_125 (20) P_lw_106;
P_w_126 (20) P_lw_104, P_lw_106;
P_z_199 (0) P_w_11, P_w_113, P_w_117, P_w_121, P_w_124, P_w_125,
P_w_126;

/*Ab hier: phantasievolle Ergaenzungen*/
/*Lade- und Fuegevorgaenge Serie 200*/

P_lw_201 (31);
P_lw_202 (37);
P_lw_203 (43) P_lw_201;
P_lw_204 (37) P_z_199;
P_lw_205 (55);
P_lw_206 (52) P_z_199;

/*Schweissvorgaenge Serie 200*/

P_w_211 (14) P_lw_201;
P_w_212 (14) P_lw_201;
P_w_213 (32) P_lw_201;
P_s_214 (08) P_lw_201;
P_w_215 (08) P_lw_201;
P_w_216 (08) P_lw_202;
P_w_217 (23) P_lw_202;
P_w_218 (14) P_lw_202;
P_w_219 (08) P_lw_202;
P_w_220 (20) P_lw_204;
P_w_221 (14) P_lw_205;
P_w_222 (38) P_lw_205;
P_w_223 (26) P_lw_205;
P_w_224 (35) P_lw_205;
P_w_225 (20) P_lw_206;
P_w_226 (20) P_lw_204, P_lw_206;
P_z_299 (0) P_w_217, P_w_216, P_w_221, P_w_222, P_w_223, P_w_224,
P_w_225, P_w_226;

/*Lade- und Fuegevorgaenge Serie 300*/

P_lw_301 (31);
P_lw_302 (37) P_z_199, P_z_299;
P_lw_303 (43) P_lw_301;
P_lw_304 (37);
P_lw_305 (55) P_z_299;
P_lw_306 (52) P_z_299;

/*Schweissvorgaenge Serie 300*/

P_w_311 (14) P_lw_301;
P_w_312 (14) P_lw_301;
P_w_313 (32) P_lw_301 P_z_199;
P_s_314 (08) P_lw_301;
P_w_315 (08) P_lw_301;
P_w_316 (08) P_lw_302 P_z_199;
P_w_317 (23) P_lw_302;
P_w_318 (14) P_lw_302;
P_w_319 (08) P_lw_302;
P_w_320 (20) P_lw_304;
P_w_321 (14) P_lw_305;
P_w_322 (38) P_lw_305;
P_w_323 (26) P_lw_305;
P_w_324 (35) P_lw_305 P_z_199;
P_w_325 (20) P_lw_306;
P_w_326 (20) P_lw_304, P_lw_306;
P_z_399 (0) P_w_311, P_w_319, P_w_321, P_w_322, P_w_323, P_w_324,
P_w_326;

/*Lade- und Fuegevorgaenge Serie 400*/

```
P_lw_401 (31) P_z_199, P_z_299, P_z_399;  
P_lw_402 (37) P_lw_401;  
P_lw_403 (43) P_lw_401;  
P_lw_404 (37) P_lw_401;  
P_lw_405 (55) P_lw_401;  
P_lw_406 (52) P_lw_401;
```

```
/*Schweissvorgaenge Serie 400*/  
P_w_411 (14) P_lw_401;  
P_w_412 (14) P_lw_401;  
P_w_413 (32) P_lw_401;  
P_s_414 (08) P_lw_401;  
P_w_415 (08) P_lw_401;  
P_w_416 (08) P_lw_402;  
P_w_417 (23) P_lw_402;  
P_w_418 (14) P_lw_402;  
P_w_419 (08) P_lw_402;  
P_w_420 (20) P_lw_404;  
P_w_421 (14) P_lw_405;  
P_w_422 (38) P_lw_405;  
P_w_423 (26) P_lw_405;  
P_w_424 (35) P_lw_405;  
P_w_425 (20) P_lw_406;  
P_w_426 (20) P_lw_404, P_lw_406;
```

Pflichtenblatt

Allgemeine Beschreibung

Ziel des Projekts ist die Realisierung eines Hilfsmittels für die Planung von *rekonfigurierbaren Fertigungszellen*. Das Java-Programm soll gute Zell-Designs mittels Evolutionsmethoden finden. Ausgangspunkt und Richtschnur der Programmentwicklung ist das Programm KoopEgo.java. Außerdem soll das Programm für Spezialfälle und für reduzierte Anforderungen (nahezu) optimale Zell-Designs mit exakten Methoden ermitteln.

Bedienoberfläche: Eingabe, Ausgabe

Das Steuerfenster besteht im oberen Teil aus einer Leiste von Steuerungsknöpfen (Buttons):

- Eingabe
- Start
- Weiter
- Halt
- Ende
- Hilfe

Damit wird das Evolutionsverfahren eingeleitet, gestartet und beendet. Nach dem Start erweitert sich das Steuerungsfenster nach unten um weitere Steuerungselemente für die interaktive Beeinflussung des Evolutionsprozess. Das Eingeben und Ausgeben von Textdateien wird mithilfe der Javax.swing-Klasse `JFileChooser` flexibel gestaltet.

Nach Drücken des Eingabe-Buttons werden die Verfahrensparameter und die Anforderungen an die Fertigungszelle von einer Hintergrunddatei geladen. Die Liste der Verfahrensparameter:

- n Größe der *Welt* (integer): ein Spielfeld aus $n \times n$ Feldern.
- k Umgebungsparameter. Legt die k-Umgebung fest
- h Anzahl der Spielzüge bis zum Halt
- p Mutationswahrscheinlichkeit

- q Wahrscheinlichkeit für Crossing-Over
- a1 Gewichtungsfaktor für die Bewertung f_1 (programminterne Darstellung des Wertes der Bewertungsfunktion f_1)
- a2 Gewichtungsfaktor für die Bewertung f_2 (Bewertungsfunktion f_2)
- T Temperatur (Simulated Annealing)

Die den Programmvariablen entsprechenden mathematischen Variablen werden kursiv geschrieben. Die den Programmvariablen a1 und a2 entsprechenden mathematischen Variablen sind α_1 und α_2 .

Das Format der Eingabedatei wird mittels EBNF definiert durch:

```
Input = { Parameter } Req Requirements end  
Parameter = intPar | floatPar  
intPar = intName integer-constant  
floatPar = floatName float-constant  
intName = n | k | h  
floatName = p | q | a1 | a2 | T
```

Fehlt in der Eingabedatei die Festlegung eines Parameterwertes, wird mit einem Default-Wert gerechnet.

Verarbeitung

Die simulierte Evolution findet auf einer Matrix `world` mit $n \times n$ Plätzen statt. Jeder dieser Plätze referenziert ein Individuum – ein Objekt vom Typ `Plan`. Die weitere Beschreibung der Verarbeitung folgt dem Schema aus Michalewicz/Fogel (2000).

Modellierung der Individuen (Representation of Individuals)

Jedes Objekt vom Typ `Plan` entspricht einem *Arbeitsplan*. Der Arbeitsplan ist ein Array von Arbeitsvorgängen vom Typ `Element`. Durch die Indizes ist die Bearbeitungsreihenfolge definiert. Weitere Attribute der Klasse `Plan` sind die Bewertungen, die sich aus den Bewertungsfunktionen ergeben, der Farbcode für die Darstellung im in der Welt und eine boolesche Variable (`Tag`), das von Variationsoperationen zur Kennzeichnung bereits berücksichtigter Arbeitsvorgänge genutzt werden kann.

Variationsoperatoren (Variation Operators)

Durch Zufallsauswahl wird ein Platz der Welt ausgewählt, das momentane Zentrum. Aus der k -Umgebung dieses Zentrums wird ein Individuum, das erste Elter, und gegebenenfalls ein zweites Elter ausgewählt. Mittels zweier Variationsoperatoren wird ein Nachkomme (Kind) dieser Eltern erzeugt⁴. Die Erzeugung des Arbeitsplans des Nachkommen geschieht in zwei aufeinander folgenden Stufen:

1. *Crossing-Over*: Dieser Operator wird mit der Wahrscheinlichkeit q ausgeführt. Falls das Crossing-Over nicht ausgeführt wird, erhält das Kind den Arbeitsplan des ersten Elters. Andernfalls entsteht der Arbeitsplan des Kindes durch Kombination der Arbeitspläne der Eltern derart, dass möglichst viele der zielführenden Eigenschaften beider Elter-Arbeitspläne gewahrt bleiben. Im Hinblick auf die zu erreichende topologische Sortierung können also Kombinationsverfahren von vornherein ausgeschlossen werden, die zu einer gewürfelten Anordnung der Arbeitsvorgänge eines der Elternteile

⁴ Ein Arbeitsplan wird als lineare Folge (Array) von Arbeitsvorgängen repräsentiert. Auf dieser Darstellungsform (Path-Representation) basieren die Variationsoperatoren (Michalewicz/Fogel, 2000, s. 197 ff.).

führen. Mehrere Crossing-Over-Methoden stehen zur Wahl. 1. *CO-Verfahren (COX)*: Zwei zufällig ausgewählte Stellen des Arbeitsplans werden als Bruchstellen gewählt. Der Arbeitsplan des Kindes besteht auf den Positionen von der unteren bis zur oberen Bruchstelle aus einer exakten Kopie des Arbeitsplans des ersten Elters. Der Rest wird aufgefüllt mit den noch nicht berücksichtigten Arbeitsvorgängen entsprechend der Reihenfolge im zweiten Elter. Eine Variante ist das 2. *CO-Verfahren (COS)*: Anstelle eines zusammenhängenden Abschnitts des ersten Elters wird eine Zufallsauswahl (Selektion) von Elementen positionsgetreu auf das Kind übertragen. Die anderen Positionen werden wieder mit den noch nicht berücksichtigten Elementen entsprechend der Reihenfolge im zweiten Elter aufgefüllt.

2. *Mutationen*: Der Arbeitsplan des Kindes wird durch Mutation verändert. Mutationsschritte werden mit der Wahrscheinlichkeit p durchgeführt bzw. fortgesetzt. Der Erwartungswert für die Anzahl von Mutationsschritten ist also gleich $p/(1-p)$. Je Mutationsschritt wird ein Element rein zufällig nach dem Indifferenzprinzip ausgewählt. Im darauf folgenden Schritt werden zwei Vorgehensweisen unterschieden: 1. *M-Verfahren (INSERT)*: Das ausgewählte Element wird herausgenommen und an einer ebenfalls zufällig ausgewählten anderen Stelle im Arbeitsplan wieder eingefügt. 2. *M-Verfahren (SWAP)*: Das ausgewählte Element wechselt mit einem anderen, das ebenfalls rein zufällig ausgewählt wird, die Plätze. 3. *M-Verfahren (SWAP0)*: Das ausgewählte Element tauscht mit dem direkt folgenden die Plätze.

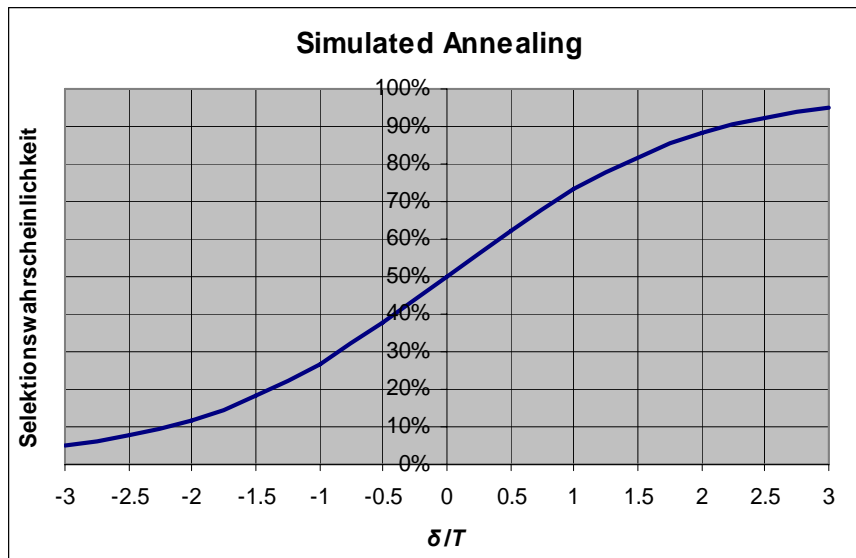
Die Eignung der Verfahren muss sich in Experimenten zeigen.

Bewertungsfunktionen (Evaluation Functions)

Die Gesamtbewertung des Arbeitsplans ist gegeben durch die *Zielgröße* $f = f_0 + a_1 \cdot f_1 + a_2 \cdot f_2$.

Auswahloperatoren (Selection Operators)

Die Selektion besteht hier darin, dass ein Kind mit ausreichender Fitness das Individuum im Zentrum ersetzen kann. Die Entscheidung, ob die Fitness für eine Ersetzung ausreicht, wird nach der Methode der *simulierten Abkühlung* (Simulated Annealing) getroffen: Mit δ wird die Verbesserung der Zielgröße bezeichnet: $\delta = f_{\text{Zentrum}} - f_{\text{Kind}}$. Die Ersetzung findet mit einer gewissen Wahrscheinlichkeit statt. Diese *Selektionswahrscheinlichkeit* ist durch die Funktion $1/(1+e^{-\delta/T})$ gegeben. Das heißt: Ist die Mutation neutral ($\delta=0$), dann wird eine Ersetzung mit der Wahrscheinlichkeit $1/2$ durchgeführt. Ergibt sich tatsächlich eine Verbesserung ($0 < \delta$), ist die Wahrscheinlichkeit für Ersetzung größer als $1/2$. Bei einer Verschlechterung ($\delta < 0$) wird die Ersetzung mit einer geringeren Wahrscheinlichkeit als $1/2$ durchgeführt. Das Verfahren wird über die Wahl des Parameters T („Temperatur“, programmintern: T) gesteuert.



Die Auswahl eines Individuums, die Variation und die Selektion bilden einen *Elementarprozess* oder *Spielzug*.

Behandlung von Randbedingungen (Constraint-Handling Techniques)

Sämtliche Randbedingungen werden in den Bewertungsfunktionen f_0 und f_1 berücksichtigt.

Visualisierung und interaktive Steuerung (Visualization and Control)

Parametern wie Temperatur (Simulated Annealing) oder Mutationsrate lassen sich über Texteingabefenster während der Simulation verändern.

Außerdem geht – in Analogie zu KoopEgo – ein Beobachtungsfenster auf. Es zeigt die *Welt*, in der jedes Individuum als farbiges Quadrat erscheint. Die Bewohner dieser Welt sind Lösungen bzw. Lösungsversuche für das Zelldesign. Der Klick auf einen Bewohner der Welt öffnet ein Fenster, das die wesentlichen Informationen über den Bewohner zeigt, insbesondere die Werte der Bewertungsfunktionen und die nach Stationen gruppierte Vorgangsfolge (Arbeitsplan). Ein Save-Button ermöglicht die Abspeicherung dieser Information in einer Textdatei.

Die Quadratfarbe wird nach folgender Regel festgelegt: Zu Beginn und nach Aktivierung durch Anklicken eines entsprechenden Buttons werden das Minimum und das Maximum der Gesamtbewertungsfunktion in der Welt ermittelt. Die Zahlen im Bereich von Minimum bis zum Maximum werden abschnittsweise in Regenbogenfarben codiert: Je näher am Minimum eine Bewertung ist, desto „blauer“ (kälter) ist die Farbe des Quadrats, und je näher am Maximum, desto „röter“ (heißer) ist sie. Die Farbcodierung wird am rechten Rand der *Welt* angezeigt.

Validierung, Treffsicherheit, Testszenarien, Effizienz

Beispiel 1 wird mit sowohl mit der Evolutionsmethode als auch mit den exakten Methoden behandelt. Die Resultate sind hinsichtlich der Güte und hinsichtlich des Rechenaufwands zu bewerten und miteinander zu vergleichen.

Mit den exakten Methoden soll gezeigt werden, inwieweit die hier implementierte Evolutionsmethode überhaupt in der Lage ist, ein Optimum aufzufinden. Folgende Fragen sind zu beantworten:

1. Was sind Bedingungen dafür, dass ein Optimum gefunden wird?
2. Gibt es Gegenbeispiele? Wenn ja: Welche?

Es werden Experimente mit den verschiedenen Mutationsverfahren durchgeführt. Die Wirksamkeit der Verfahren ist hinsichtlich Zeitbedarf und Treffsicherheit zu beurteilen.

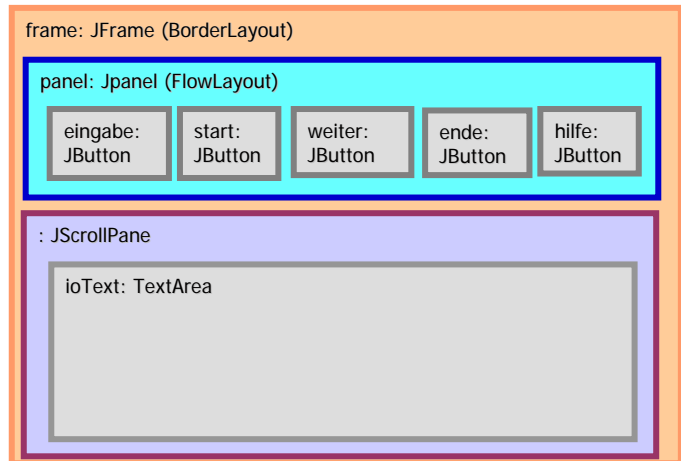
Entwurf

Entwurf der grafischen Oberfläche

Das Steuerungsfenster

Die nebenstehende Grafik zeigt einen ersten Entwurf des Steuerungsfensters und die dafür verwendeten Java-Klassen des API.

Nach Drücken des Eingabe-Buttons startet ein Dialog zur Auswahl der Textdatei mit den Eingabedaten. Sie wird in das Textfenster geladen und ist dort editierbar. Durch Änderung des Dateinamens entsteht eine neue Datei für die so veränderten Daten. Bleibt der Dateiname unverändert, wird die alte Datei mit den eventuell geänderten



Daten überschrieben. Mit einem Start-Button wird die aktualisierte Textdatei abgespeichert und der Evolutionsprozess *initialisiert*. Das Textfenster verschwindet und ein Bild der Welt im *Urzustand* erscheint. Außerdem erscheint ein Panel mit Textfeldern zur interaktiven Beeinflussung von Simulationsparametern: Gewichtsfaktoren, Temperatur, Mutations- und Crossing-over-Wahrscheinlichkeit.

Mit dem Weiter-Button wird der Evolutionsprozess gestartet. Nach einer vorgebbaren Anzahl von Spielzügen (Elementarprozessen) wird das Bild der Welt jeweils aktualisiert. Durch verschiedene Buttons des Steuerungsfensters (nicht dargestellt) lässt sich der Prozess vorübergehend unterbrechen. Die Bewohner der Welt können dann inspiziert und Ergebnisse abgespeichert werden.

Drücken des Ende-Buttons beendet die Simulation.

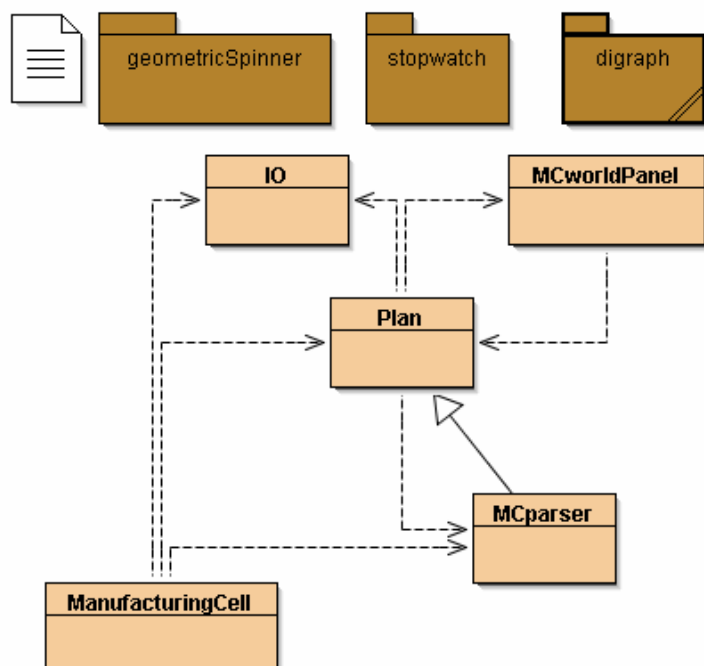
Klassendiagramm

Ein erster Entwurf des Klassendiagramms ist in der vereinfachten UML-Darstellung von BlueJ rechts wiedergegeben.

Eingabe

Eingabe der Arbeitsschritte und Netzdarstellung

Die eingegebene Liste der Arbeitsvorgänge wird vom Parser in einem einfachen Durchlauf bearbeitet (One-pass-Compilierung) und unmittelbar



als Netz (Digraph) abgespeichert. Das singleLink-Package enthält die Link-Klasse für die Graphendarstellung. Gegenüber der Version des SingleLink-Packages ist sie effizienter (weniger rekursiv, weniger elegant, aber schneller).

Topologische Sortierung der Arbeitsschritte und Schleifenerkennung

Ergebnis des Eingabeprozesses ist eine topologisch sortierte lineare Liste der Arbeitsschritte. Die Sortierung wird hier nicht mit dem less-fitIn-Mechanismus der Link-Klasse besorgt. Grund: Man weiß noch gar nicht, ob die Eingabedaten tatsächlich eine topologische Sortierung zulassen: Möglicherweise sind in die Daten zyklische Abhängigkeiten eingebaut – aus Versehen oder aufgrund einer schlechten Analyse des Fertigungsprozesses. Deshalb wird hier eine Methode aufgegriffen, die bereits im Programm BlockSim.java – einem Programm zur blockorientierten Simulation – verwendet worden ist. Neben der topologischen Sortierung zyklensfreier Digraphen führt diese Methode auch eine Prüfung auf Zyklen durch. Dazu werden die Knoten (Elemente) zunächst in einer Hilfsliste untergebracht und die Originalliste wird geleert. Danach wird der folgenden Arbeitsschritt wiederholt ausgeführt: Die Hilfsliste wird nach einem Knoten durchsucht, dessen Vorgänger alle bereits in der Originalliste stehen. Dieser Knoten wird aus der Hilfsliste genommen und an die Originalliste angehängt. Das wird gemacht, bis die Hilfsliste leer ist. Misslingt das, dann muss das Netz Schleifen enthalten und seine Knoten sind nicht topologisch sortierbar.












Die Anfangsbelegung

Alle Plätze der Welt werden einheitlich mit der topologisch sortierten Liste, die Ergebnis des Eingabevorgangs ist, belegt. Durch Mutationen und eine Vorgabe einer hohen Temperatur des Simulated Annealing, kann man für das nötige schöpferische Chaos sorgen.

Repräsentation der Welt und der Individuen

Farbkodierung

Die Wesen der Welt werden durch die Farben der folgenden Tabelle charakterisiert. Jede Farbe entspricht einer Bewertungsstufe: Rot entspricht hohen Werten und Blau den niedrigen. Die Mitte bildet das Grün. Die Farbintensität ist auf der 256-stufigen Skala angegeben mit Werten von 0 bis 255.

<i>Stufe</i>	<i>rot</i>	<i>grün</i>	<i>blau</i>	<i>Darstellung</i>	<i>Bezeichnung</i>
10	255	127	127		rosa
9	255	0	0		rot
8	255	127	0		orange
7	255	255	0		gelb
6	127	255	0		gelbgrün
5	0	255	0		grün
4	127	255	127		grünlich
3	0	255	255		türkis (zyan)
2	0	127	255		hellblau
1	0	0	255		blau
0	0	0	127		dunkelblau

Bei einer Renormierung der Darstellung werden zunächst das Minimum a und das Maximum b der Bewertungen der Bewohner der Welt erfasst. Im weiteren Verlauf der Simulation können diese Werte auch unter- bzw. überschritten werden.

Die Auswahl der Stufe s für ein Individuum mit der Bewertung f geschieht nach folgender Formel: $s = \text{round}(8 \cdot (f-a)/(b-a)) + 1$. Also: Alle um den niedrigsten Wert herum haben die Stufe 1 (blau). Wird der Wert deutlich unterschritten, erhalten die Individuen die Stufe 0 (dunkelblau). Entsprechendes gilt für die höheren Werte und die Farben Rot und Rosa.

Falls alle Bewohner der Welt dieselbe Bewertung f haben – das gilt zumindest für den Anfangszustand der Welt –, dann wird $a=f-1$ und $b=f+1$ gesetzt.

Realisierung

Der Programmtext ist im Java-Archiv ManufacturingCell.jar enthalten⁵. Hier sollen nur einige der besonders erläuterungsbedürftigen Abschnitte besprochen werden.

Parser: Dateneingabe und Anlegen der Strukturdaten für die Fertigungszelle

Die find-Methode der Element-Klasse

Der Parser fügt die Netzknoten (Elemente) in die Liste list ein. Er bearbeitet die Eingabedaten in einem einzigen Durchgang. Er ist ein rudimentärer One-pass-Compiler. Da Knoten bereits als Vorgänger eines anderen Knotens benannt sein können, ohne dass der Knoten selbst bereits erfasst ist, wird sowohl zum Erfassen als auch zum Auffinden eines Vorgängerknotens die folgende find-Methode der Element-Klasse verwendet.

```
public Element find(String name) {
    Element n= list;
    while ((n=n.next())!=null) if (name.equals(n.name)) return n;
    list.add(n= new Element(name));
    return n;
}
```

Jeder Knoten wird also in dem Moment angelegt, wenn sein Name erstmals erscheint, egal ob das im Zuge der vollen Beschreibung des Knotens geschieht, oder wenn er nur als Vorgänger benannt ist. Im letzteren Fall erhält er nur Default-Parameter und die vollständige Versorgung mit Parametern wird nachgeholt, wenn die Knotenbeschreibung in der Liste der Eingabedaten erscheint.

Die topologicalSort-Methode der Element-Klasse

Die list-Variable steht im Folgenden für die lineare Liste aller Netzknoten (Elemente).

```
//Sortiert die Liste der Elemente unter Beruecksichtigung des Vorrangs. Ist die
//Sortierung aufgrund von Schleifen unmoeglich, wird "false" zurueckgegeben.
static boolean topologicalSort() {
    Element l=new Element();
    l.add(list.bisect());
    while (!l.empty()){
        Element o=l;
        while (o.next()!=null&&!o.next().predContainedInList()) o=o.next();
        if (o.next()!=null) list.add(o.out());
        else {list.add(l.bisect()); return false;}
    }
    return true;
}
```

Die predContainedIn-Methode überprüft, ob alle Vorgänger des o-Elements in der als Parameter angegebenen Liste enthalten sind.

⁵ Es ist eines der Programme zum Thema [Problemlösen](#) und im Internet frei erhältlich.

Verifikation der topologicalSort-Methode (Programmbeweis)

Zu beweisen ist, dass die topologicalSort-Methode eine Fehlermeldung ausgibt, falls das Netz Zyklen enthält, und dass die Methode anderenfalls die Liste der Netzknoten topologisch sortiert. Der Korrektheitsbeweis geschieht mittels *Schleifensatz*.

Schleifeninvariante: Die Liste von Netzknoten (list) ist topologisch sortiert.

Die Schleifeninvariante gilt unmittelbar nach der Initialisierung „l.add(list.bisect());“, da dieser Befehl die Liste leert. Da im Schleifenkörper immer nur Elemente an die Liste hinten (!) angefügt werden, deren Vorgänger alle bereits darin sind, bleibt die topologische Sortierung bei Durchlaufen des Schleifenkörpers erhalten. Falls schließlich die Menge der noch nicht einsortierten Knoten leer ist, muss die Liste schließlich alle Netzknoten in topologischer Sortierung enthalten.

Falls sich unter den noch nicht einsortierten Knoten keiner finden lässt, dessen Vorgänger sämtliche bereits in der Liste enthalten sind, muss das Netz Zyklen enthalten. Um das zu zeigen, beginnt man mit irgendeinem Knoten, der nicht in die Liste einsortiert ist. Einer seiner Vorgänger ist ebenfalls nicht in der sortierten Liste. Er muss selbst wieder einen Vorgänger haben, der noch nicht in der Liste ist, und so weiter. Wir erhalten eine Folge von Knoten, die allesamt in einer direkten Vorgänger-Nachfolger-Beziehung stehen. Diese Folge ist unendlich. Da aber das Netz nur aus insgesamt endlich vielen Knoten besteht, müssen in der Folge Knoten wiederholt auftreten. Diese müssen in Zyklen liegen.

Nachbarschaft und Feldauswahl

Anders als in KoopEgo wird hier das Zentrum einer k-Nachbarschaft als Elter zugelassen. Dadurch ist eine 0-Nachbarschaft nicht mehr ausgeschlossen und die Variation kann auf Einzelwesen beschränkt werden. Durch k=0 sind Paarungen ausgeschlossen und damit auch das Crossing-Over. Da auch eine Welt der Dimension $n = 1$ zulässig ist, ist mit dem Programm das ursprüngliche Mutations-Selektionsverfahren von Rechenberg darstellbar. Mit k=0 kann auch das Simulated-Annealing Verfahren ohne Vermischung mit Vererbungsmechanismen isoliert untersucht werden.

Algorithmen für Variationsoperatoren

Selektion

Beim COS-Verfahren ist eine Zufallsauswahl von Plätzen (Indizes) eines Arbeitsplans zu bilden. Es ist also eine Auswahl aus der Zahlenfolge 1, 2, ..., n zu treffen. Jeder Index wird mit der Wahrscheinlichkeit $\frac{1}{2}$ ausgewählt. Man könnte mit der Integervariablen i die Zahlen der Reihe nach durchgehen und für jede entscheiden, ob sie zur Auswahl gehören soll oder nicht. Etwa so:

```
for int i=1; i<=n; i++) if(Math.random()<.5) select(i);
```

Das macht n Aufrufe des Zufallszahlengenerators notwendig. Die Zahl der Aufrufe lässt sich halbieren, indem man nicht die Wahrscheinlichkeit der Selektion direkt ermittelt, sondern den Abstand der selektierten Werte als Zufallsvariablen nimmt. Die diskrete Verteilung dieser Abstände D ist gegeben durch $p_k = P(D=k) = 1/2^k$. Seien s_k die Partialsummen dieser Zahlen (Verteilungsfunktion von D): $s_0 = 0, s_1 = s_0 + p_1, s_2 = s_1 + p_2, \dots$ Die Intervalle $[s_{k-1}, s_k)$ haben die Länge p_k und sie bilden eine Zerlegung des Intervalls $[0, 1)$. Die Realisierung einer gleichverteilten Zufallsvariable (wie sie beispielsweise von Math.random() erzeugt wird) fällt mit der Wahrscheinlichkeit p_k in das Intervall $[s_{k-1}, s_k)$. Es ist also nur nötig, eine solche Realisierung r zu erzeugen und dann zu sehen, in welches der Intervalle die Zahl fällt. Das entsprechende k ist dann der gesuchte Abstand. Da der Mittelwert der Abstände gleich 2 ist, muss

man im Mittel nur $n/2$ -mal den Zufallszahlengenerator aufrufen. Eine programmtechnische Realisierung könnte so aussehen:

```
for (int i=1;;) {
  for (double r= Math.random(), x=.5, y=x; y<=r; x/=2, y+=x, i++);
  if (i<=n) select(i); else break;
  i++;
}
```

Die immer wiederkehrende Berechnung der Intervallgrenzen kann man sich sparen, indem man sie vorab in einem Vektor (sum) abspeichert. Der Vektor muss nicht allzu lang sein. Weil bereits Intervalle der Länge $1/2^{100}$ extrem selten sind, genügt ein Array der Länge 100. Vorsichtshalber setzt man die letzte Intervallgrenze auf 1. Mit dieser Version des Algorithmus lässt sich der Rechenaufwand weiter verringern:

```
for (int i=-1;;) {
  k=1;
  for (double r= Math.random(); sum[k]<=r; k++);
  i+=k;
  if (i<=n) select(i); else break;
}
```

Die Zeitersparnis gegenüber der ersten Version beträgt bei der zweiten etwa 38% und bei der dritten 46%, also knapp die Hälfte.

Rekombination

Bei den Crossing-Over-Verfahren wird eine Auswahl von Elementen eines Elters auf das Kind übertragen. Diese Elemente werden mit dem Tag-Attribut als bereits ausgewählt markiert. Anschließend werden die nicht markierten Elemente gemäß der Anordnung im zweiten Elter übernommen.

Mutationsverfahren

Für die Mutationen wird das SWAP0-Verfahren vorgesehen.

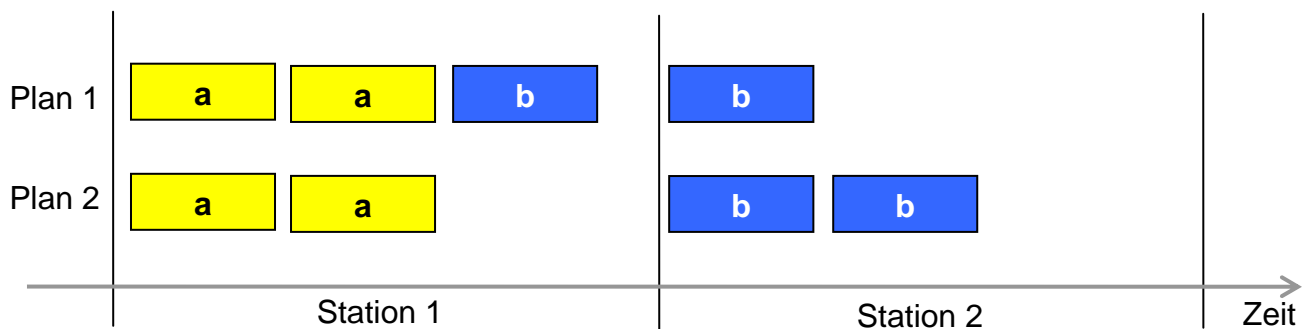
Bewertungsoperatoren

Die Bewertungsfunktion f_1 wurde auf der Grundlage von Schleifeninvarianten realisiert. Die Invarianten sind in den Programmtext als Kommentar eingetragen. Die Konstruktion auf Basis des Schleifensatzes liefert eine effizientere und textlich kompaktere Lösung als eine direkte Umsetzung des Algorithmus aus dem Abschnitt „Ein idealisiertes Modell der Fertigungszelle“.

```
//Einplanung der Stationen
int i, j=0;
//Schleifeninvariante: Die letzte eingeplante Station bearbeitet plan[i..j-1]
//UND j ist maximal UND Dieser Stationstyp ist der erste in der Folge der
//Stationstypen, der das leistet.
while (j<plan.length) {
  int acc=0, k=i=j;
  StationType stationType= StationType.list.next(), optimum=null;
  //Schleifeninvariante: Stationstyp optimum kann plan[i..j-1] bearbeiten
  //UND Kein anderer Stationstyp vor stationType erreicht ein groesseres j
  while (
    k<plan.length
    &&acc+plan[k].duration<=Element.taktTime
    &&stationType!=null
  )
    if(stationType.capable(plan[k].tool)) {
      acc+=plan[k].duration;
      if (j<k) { j=k; optimum=stationType; }
    } else {
      stationType=stationType.next();
      acc=0; k=i;
    }
  if (i<j) sList.add(new Station(optimum, j));
  else break;
}
```

Von großer Bedeutung ist, dass der Algorithmus nur linear mit der Anzahl der Arbeitsvorgänge wächst. Zwar werden vor einer Erhöhung von j möglicherweise alle Maschinentypen für einen kurzen Abschnitt des Arbeitsplans ausprobiert (innere While-Schleife). Aber mit j geht es dann nur aufwärts.

Dass diese Methode nicht immer eine optimale Zuordnung der Stationen zu den Vorgängen liefert, zeigt ein kleines Gegenbeispiel: Wir haben vier Aufträge der Dauer 20, die ersten beiden mögen das Werkzeug a benötigen und die letzten beiden das Werkzeug b. Es gibt Stationen mit dem Werkzeug a und welche mit dem Werkzeug b, aber auch etwas teurere Stationen mit beiden Werkzeugtypen. Die Taktzeit betrage 70. Der obige Algorithmus würde die ersten drei Arbeitsvorgänge einer Station vom Typ ab zuordnen und den letzten einer vom Typ b (Plan 1 im Bild). Besser wäre es, nur die ersten beiden Arbeitsvorgänge von einer Station des (billigeren) Typs a und die letzten beiden von der Station des Typs b abarbeiten zu lassen.



Die zentrale Schleife: Plan.run()

Die run-Methode von Plan wird für jeden Lauf aufgerufen. Sie besteht nur aus einer Schleife, in der jeweils h neuen Kandidaten erzeugt, bewertet und im Erfolgsfall in die Welt übernommen werden.

```
for (int i=0; i<h; i++) { //Feldauswahl
    int xy = rand.nextInt(nn), x0 = xy/n, y0 = xy%n;
    Plan newPlan;
    if(rand.nextFloat()<q) newPlan= neighbour(x0, y0).combine(neighbour(x0, y0));
    else newPlan= new Plan(neighbour(x0, y0).plan.clone());
    if (rand.nextFloat()<1/(1+Math.exp((newPlan.f()-world[x0][y0].f())/t)))
        world[x0][y0]= newPlan;
```

Experimente

1. Versuchsserie: Grundlegenden Optimierungsstrategien am Beispiel 1

Optimierungsziele und Prognose

Ziel ist eine möglichst geringe Anzahl von Stationen. Die Stationen werden also alle gleich bewertet und die Gewichtsfaktoren werden folgendermaßen gesetzt: $a_1=1$, $a_2=0$.

Beim ersten Beispiel beträgt der Zeitbedarf aller Arbeitsvorgänge zusammen 557 Sekunden. Bei einer Taktzeit von 70 Sekunden sind demnach wenigstens 8 Stationen nötig und die wären im Schnitt zu 99% ausgelastet. Die Eingabedaten sind bereits topologisch sortiert und erfordern in dieser Folge 10 Stationen. Von der Optimierung ist also die Einsparung von nur einer Station zu erwarten.

Mutations-Selektionsverfahren ($n=1$, $k=0$, $T=0$)

Vorversuche legen nahe, ein relativ großes p zu wählen. Mit $p = 0.9$ löst bereits das einfache Mutations-Selektionsverfahren – das ist im Wesentlichen der Algorithmus von Rechenberg –

diese Optimierungsaufgabe mit Bravour. Mit deutlich weniger als 100 Schritten wird eine Lösung mit 9 Stationen ausfindig gemacht.

Simulated Annealing (variables T)

Eine Temperaturerhöhung ist bei diesem Beispiel nicht erforderlich, da es offenbar genügend direkte Pfade hin zu optimalen Lösungen gibt. Deshalb wird beim Beispiel 1 grundsätzlich die Temperatur $T=0.1$ gesetzt.

Parallele Mutation-Selektion ($n=40$, $k=0$)

Diese Parameter bewirken, dass praktisch 1600 Mutations-Selektionsverfahren parallel ablaufen. Der Aufwand erhöht sich dementsprechend. Andererseits entsteht ein Wettrennen. Ich wähle einen der ersten Kandidaten, der mit 9 Stationen auskommt. Er hat die Versuchsplannummer = 2216 und ist also der 2216. Kandidat. Er ist in einer der 1600 Linien entstanden und ist aus entsprechend wenigen Variationsschritten entstanden.

Bei nur wenigen Mutationsschritten ist zu erwarten, dass die Arbeitsvorgänge gegenüber der Anfangsfolge nicht so stark durcheinander geraten. Die optimierte Lösung unterscheidet sich von der Anfangsanordnung nur wenig.

Inwieweit sich zwei Anordnungen unterscheiden, wird üblicherweise mit der Zahl der Abfolgen ausgedrückt, das ist die Anzahl der Paare, die die Plätze getauscht haben. Beim einfachen Mutations-Selektionsverfahren sind 31 Abfolgen entstanden. Hier sind es 29.

Crossing-Over-Verfahren mit globaler Paarung ($n=40$, $k=20$)

Es wird hundertprozentiges Crossing-Over gesetzt ($q=1$). Auch hier entsteht ein optimaler Kandidat nach weniger als 2000 Versuchen (). Er hat in diesem Lauf die Versuchsplannummer 1778. Die Zahl der Abfolgen ist 21.

Crossing-Over-Verfahren mit lokaler Paarung ($n=40$, $k=3$)

Es wird hundertprozentiges Crossing-Over gesetzt ($q=1$). Auch hier entsteht ein optimaler Kandidat nach weniger als 2000 Versuchen (Versuchsplannummer = 1704). Er erzeugt nur noch 8 Abfolgen. In der folgenden Tabelle ist der ursprüngliche Plan dem schließlich erreichten gegenübergestellt. Die Verschiebungen sind mit Pfeilen markiert. Jedem Kreuzungspunkt entspricht genau eine Abfolge

Es geht mit noch weniger Abfolgen, wie man an den gestrichelt und farbig eingezeichneten Verschiebungspfeilen sehen kann.

Der Versuch wurde mehrmals wiederholt. Die Zahlen für die Abfolgen der ersten 9-er Kandidaten: 17, 19, 15, 16.

ARBEITSPLAN

Datei der Anforderungen: vers1.1.txt

Bewertung:

Identität = 0

Vorrangverletzungen f0= 0

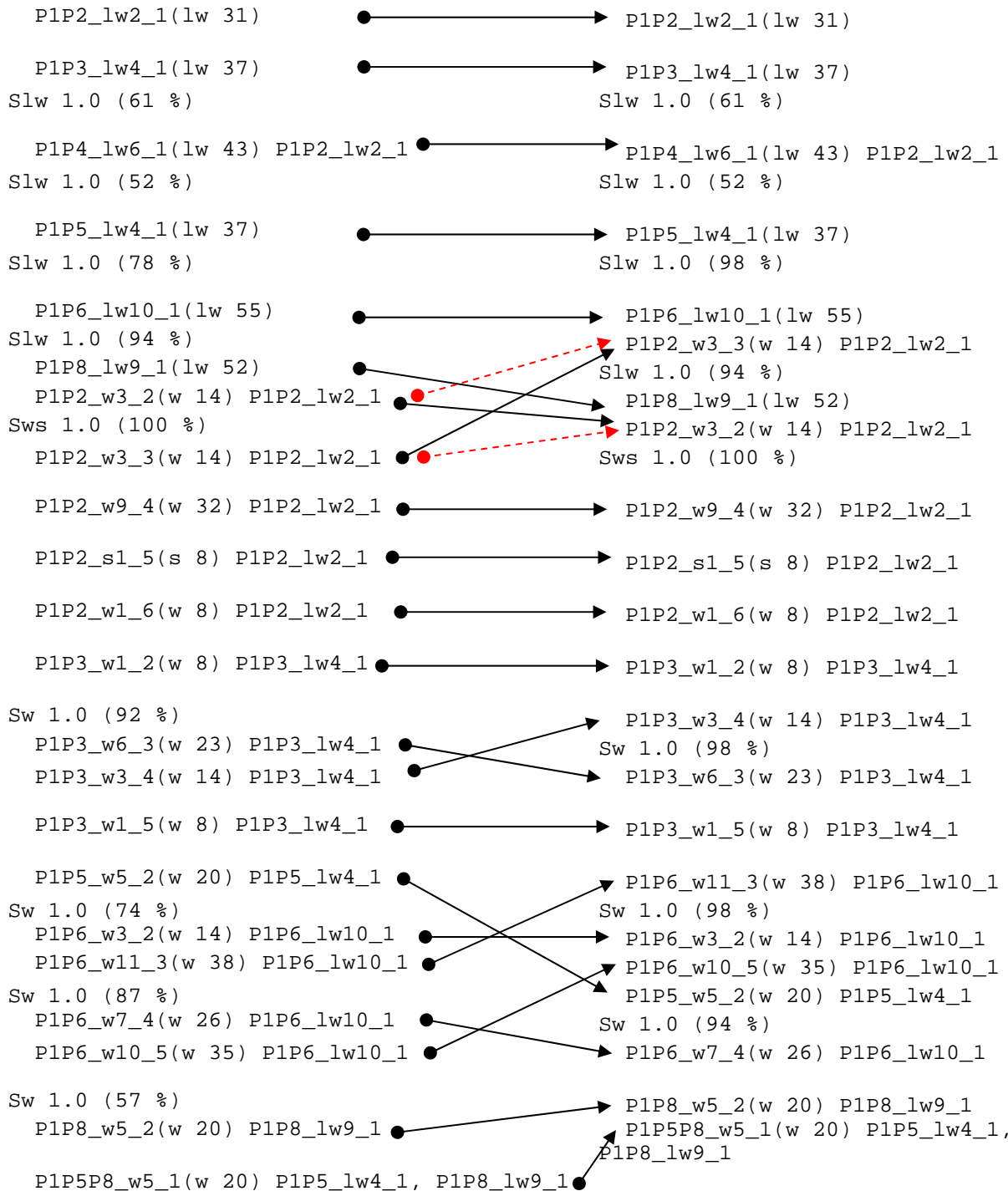
Aufwand Stationen f1= 10.0

Gutschriften f2= -15

Gesamtbewertung f()= 10.0

Die Belegung der Stationen:

Slw 1.0 (97 %)



ARBEITSPLAN

Datei der Anforderungen: vers1.5.txt

Bewertung:

Identität = 1704

Vorrangverletzungen f0= 0

Aufwand Stationen f1= 9.0

Gutschriften f2= -16

Gesamtbewertung f()= 9.0

Die Belegung der Stationen:

Slw 1.0 (97 %)

2. Versuchsserie: Effizienz der Optimierungsstrategien am Beispiel 2

Voreinstellungen

Ziel ist eine möglichst geringe Anzahl von Stationen. Die Stationen werden also alle gleich bewertet und der Gewichtungsfaktor für die Stationen wird auf $a_1=1$ gesetzt.

In allen Versuchen wird die Mutationswahrscheinlichkeit auf $p=0.9$ und die Crossing-Over-Wahrscheinlichkeit (relevant nur für die Crossing-Over-Verfahren) auf $q=1$ gesetzt.

Optimierungsziele und Prognose

Das Beispiel 2 ist im Grunde eine Vervierfachung des Beispiels 1. Die ursprüngliche Anordnung der Arbeitsvorgänge (Ergebnis der – notfalls topologisch sortierten – Eingabedaten) ist mit 40 Stationen zu bewältigen. Der aufsummierte Zeitbedarf aller Arbeitsvorgänge ist gleich 2228 Sekunden. Bei einer Taktzeit von 70 Sekunden sind also wenigstens 32 Stationen erforderlich. Die mittleren Auslastungen in Abhängigkeit von der Stationszahl zeigt die Tabelle.

Bei einem Auslastungsbeitrag der kürzesten Arbeitsvorgänge von $8/70 = 11\%$ wird eine mittlere Auslastung der Stationen von 94% kaum zu erreichen sein. Das erreichbare Minimum an Stationen wird mit **35** prognostiziert.

Stationen	Auslastung
32	99%
33	96%
34	94%
35	91%
36	88%

Ein erster Versuch mit dem einfachsten Verfahren (Mutations-Selektionsverfahren) hat gezeigt, dass mit der durch $a_1=1$ und $a_2=0$ definierten Zielfunktion der prognostizierte Optimalwert problemlos erreicht wird. Der siegreiche Kandidat hat die Nummer 61999. Keine Vorrangverletzungen. 35 Stationen und 1097 (nicht bewertete) Gutschriften.

Um eine differenzierte Leistungsbewertung zu erhalten, werden die Gutschriften in die Bewertung einbezogen. Daran ist nur wichtig, dass die Zielgröße feinere Abstufungen ermöglicht. Im konkreten Anwendungsfall werden andere Differenzierungen der Bewertungsfunktion nötig sein.

Um den Gewichtsparameter a_2 richtig definieren zu können, ist erst einmal die Größenordnung der zu erwartenden Gutschriften zu ermitteln. Wären alle 24 Ladevorgänge gleichmäßig über die 88 Arbeitsvorgänge verteilt, ergäbe sich ein mittlerer Gutschriftenwert von $(0+1+2+\dots+23) \cdot 88/24 = 1012$. Verbesserungen um wenige Gutschriftspunkte dürfen den Wert einer Station nicht aufwiegen. Versuchsweise wird für alle folgenden Versuche **$a_2 = 0.1$** gesetzt.

Mutations-Selektionsverfahren ($n=1, k=0, T=0$)

1. Versuch: Versuchsplannummer = 1586929, Vorrangverletzungen $f_0= 0$, Aufwand Stationen $f_1= 35.0$, Gutschriften $f_2= -1336$, Gesamtbewertung $f()= -98.6$

2. Versuch: Versuchsplannummer = 614992, Vorrangverletzungen $f_0= 0$, Aufwand Stationen $f_1= 35.0$, Gutschriften $f_2= -1248$, Gesamtbewertung $f()= -89.8$

3. Versuch: Versuchsplannummer = 2531994, Vorrangverletzungen $f_0= 1$, Aufwand Stationen $f_1= 36.0$, Gutschriften $f_2= -1345$, Gesamtbewertung $f()= -97.5$

Der erste Versuch hat das beste Ergebnis geliefert. Der dritte Versuch ein nicht akzeptables. Wenn der Algorithmus sich verrennt, wie hier beim dritten Versuch, dann gibt es kein Zurück. Da Alternativentwicklungen nicht sichtbar sind, ist der Optimierungsvorgang so etwas wie ein Blindflug. Diese Beobachtungen sprechen für die mehrfache Wiederholung des Optimierungsprozesses oder – was gleichbedeutend ist – für eine Parallelisierung.

Mutations-Selektionsverfahren mit Simulated Annealing (variables T)

Durch wiederholte Temperaturerhöhung und erneutes Einfrieren sollen bessere Alternativen sichtbar gemacht werden.

1. Versuch: Temperaturerhöhung mit darauf folgender Abkühlung hat zu inakzeptablen Plänen geführt.
2. Versuch: Dasselbe Ergebnis

Das kann man sich so erklären: Durch die Chaotisierung gerät der durchaus passable Ausgangspunkt außer Sicht. Und die Mutations-Selektionsverfahren finden offenbar dann keine Lösung mehr, wenn die Zielpunkte weitab liegen und eventuell nur über Umwegen erreichbar sind. Das wird durch die 3. Versuchsserie bestätigt.

Parallele Mutation-Selektion (n=40, k=0, variables T)

1. Versuch: Anfangs wurde die Temperatur erhöht. Dann wurde eingefroren. Zwischenzeitlich wurden a1 und a2 gleich null gesetzt um wieder auf Lösungen ohne Vorrangverletzungen zu kommen. Nach über drei Mio. Plänen stellte sich der folgende als einer der besten heraus:

```
Identität = 3224198
Vorrangverletzungen f0= 0
Aufwand Stationen f1= 37.0
Gutschriften f2= -1241
Gesamtbewertung f()= -87.1
```

2. Versuch: Reine Abkühlung

```
Identität = 7503443
Vorrangverletzungen f0= 0
Aufwand Stationen f1= 36.0
Gutschriften f2= -1262
Gesamtbewertung f()= -90.2
```

Auch bei paralleler Mutation-Selektion ist der Effekt zu beobachten, dass eine zwischenzeitliche Temperaturerhöhung und Chaotisierung zu schlechteren Ergebnissen führt, als der geordnete Anfangszustand.

Crossing-Over-Verfahren mit globaler Paarung (n=40, k=20, variables T)

1. Versuch: Das folgende Resultat kam nach einer zwischenzeitlichen Temperaturerhöhung zustande. Das Individuum war Bewohner einer monotonen Welt.

```
Identität = 2904451
Vorrangverletzungen f0= 0
Aufwand Stationen f1= 36.0
Gutschriften f2= -1343
Gesamtbewertung f()= -98.3
```

2. Versuch:

```
Identität = 2710346
Vorrangverletzungen f0= 0
Aufwand Stationen f1= 36.0
Gutschriften f2= -1338
Gesamtbewertung f()= -97.8
```

Crossing-Over-Verfahren mit lokaler Paarung (n=40, k=3, variables T)

1. Versuch:

```
Identität = 4011170
Vorrangverletzungen f0= 0
Aufwand Stationen f1= 35.0
Gutschriften f2= -1391
```

Gesamtbewertung $f() = -104.1$

2. Versuch:

Identität = 2715797
Vorrangverletzungen $f_0 = 0$
Aufwand Stationen $f_1 = 36.0$
Gutschriften $f_2 = -1359$
Gesamtbewertung $f() = -99.9$

3. und 4. Versuch: keine Verbesserung

5. Versuch:

Identität = 4665037
Vorrangverletzungen $f_0 = 0$
Aufwand Stationen $f_1 = 36.0$
Gutschriften $f_2 = -1371$
Gesamtbewertung $f() = -101.1$

3. Versuchsserie: Die Beherrschung des Chaos (gemessen am 2. Beispiel)

Voreinstellungen

Zweck der dritten Versuchsserie ist, am Beispiel 2 zu zeigen, wie die Verfahren mit dem reinen Chaos zurechtkommen. Aus Gründen der Vergleichbarkeit werden nur die drei letzten Verfahren untersucht (Parallele Mutation-Selektion, Crossing-Over mit globaler Paarung und Crossing-Over mit lokaler Paarung).

Es werden jeweils 1 Mio. Schritte mit den Parametern $T=100$, $p=0.9$ und $q=0$ durchgeführt. Hierbei wird auf jedem Platz ein möglichst zufälliger Plan erzeugt. Die Bedingungen sind – wegen $q=0$ – für alle Verfahren gleich. Dann kommen 1 Mio. Optimierungsschritte mit dem Parametern $T=0.001$, $p=0.9$ und $q=1$ (wobei der letzte Parameter im reinen Mutations-Selektions-Verfahren wirkungslos bleibt). Danach wird jeweils einer der besten Kandidaten protokolliert.

Die Parameter der Bewertungsfunktion werden auf $a_1=1$ und $a_2=0.1$ gesetzt. In einem zweiten Durchlauf bleiben die Gutschriften unberücksichtigt: $a_1=1$, $a_2=0$. Für die parallele Mutation-Selektion und das Crossing-Over mit lokaler Paarung wird noch ein Versuch mit $a_1=0.2$ und $a_2=0$ durchgeführt, um den Vorrangverletzungen besser beizukommen. Die Ergebnisse waren aber im Wesentlichen dieselben wie vorher.

Es sieht so aus, als seien bei anfänglichem Chaos die Crossing-Over-Verfahren den reinen Mutation-Selektions-Strategien überlegen.

Parallele Mutation-Selektion ($n=40$, $k=0$)

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0.1$:

Datei der Anforderungen: versuch2.3.txt
Bewertung:
Identität = 1980444
Vorrangverletzungen $f_0 = 6$
Aufwand Stationen $f_1 = 37.0$
Gutschriften $f_2 = -1241$
Gesamtbewertung $f() = -81.1$

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0$:

Datei der Anforderungen: versuch2.3.txt
Bewertung:
Identität = 1999713
Vorrangverletzungen $f_0 = 5$
Aufwand Stationen $f_1 = 37.0$

Gesamtbewertung $f() = 42.0$

Crossing-Over-Verfahren mit globaler Paarung (n=40, k=20)

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0.1$:

Datei der Anforderungen: versuch2.4.txt

Bewertung:

Identität = 1989556

Vorrangverletzungen $f_0 = 6$

Aufwand Stationen $f_1 = 37.0$

Gutschriften $f_2 = -1547$

Gesamtbewertung $f() = -111.7$

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0$:

Datei der Anforderungen: versuch2.4.txt

Bewertung:

Identität = 1772414

Vorrangverletzungen $f_0 = 1$

Aufwand Stationen $f_1 = 36.0$

Gesamtbewertung $f() = 37.0$

Crossing-Over-Verfahren mit lokaler Paarung (n=40, k=3)

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0.1$:

Datei der Anforderungen: versuch2.5.txt

Bewertung:

Identität = 2000920

Vorrangverletzungen $f_0 = 4$

Aufwand Stationen $f_1 = 35.0$

Gutschriften $f_2 = -1491$

Gesamtbewertung $f() = -110.100006$

Die Nummer des Plans liegt hier über 2 Mio. Grund: Bei den Identitäten wird die Vorbelegung (1600 Pläne) mitgezählt, bei den Spielrunden nicht.

Kurzprotokoll des Besten bei der Bewertung mit $a_1=1$ und $a_2=0$:

Datei der Anforderungen: versuch2.5.txt

Bewertung:

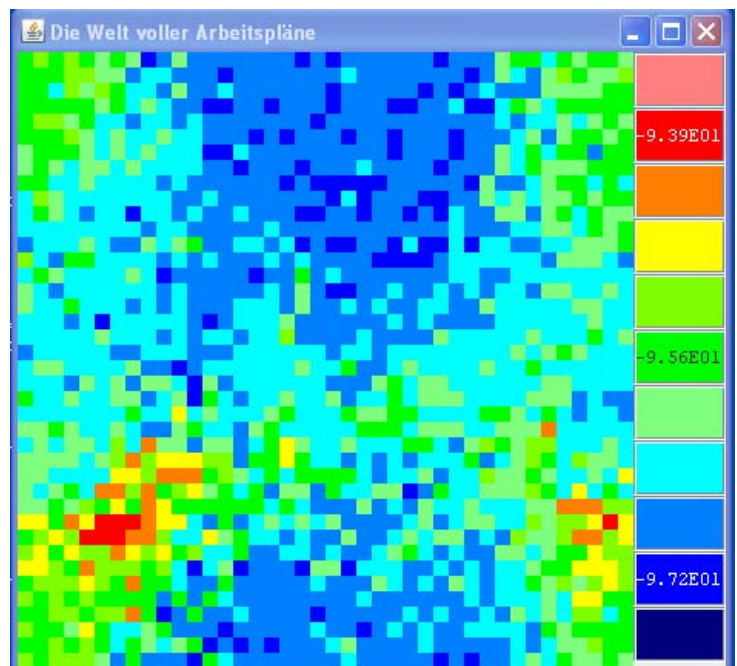
Identität = 1978858

Vorrangverletzungen $f_0 = 1$

Aufwand Stationen $f_1 = 34.0$

Gesamtbewertung $f() = 35.0$

Nebenan ist ein Schnappschuss der Welt inmitten eines Abkühlungsprozesses zu sehen.



Ergebnisse

Einfluss der Variationsoperatoren

Die Versuche wurden mit den Variationsoperatoren COS und SWAP0 durchgeführt. (Weitere Variationsoperatoren sind im derzeitigen Programm nicht realisiert.)

Der Mutationsoperator SWAP0 nimmt größte Rücksicht auf die Problemstruktur. Er geht einerseits mit bereits topologisch sortierten Arbeitsplänen pfleglich um und er leistet dennoch eine ordentliche Verwirbelung. Mit einem Mutationsparameter von $p=0.9$ wird ein ausreichendes schöpferisches Chaos erreicht.

Regeln für die Auswahl der Optimierungsstrategie

Die Experimente haben gezeigt, dass die Crossing-Over-Verfahren den reinen Mutations-Selektionsstrategien überlegen sind. Das zeigt sich besonders bei dem komplexeren zweiten Beispiel. Mutations-Selektionsstrategien tendieren dazu, in Sackgassen zu enden. Temperaturerhöhung führt oft dazu, dass neue Ausgangspunkte für die Optimierung entstehen, die zu weit von guten Lösungen entfernt sind, so Mutation-Selektion auf einem isolierten Optimierungspfad diese Entfernungen nicht mehr überbrücken kann.

Die Crossing-Over-Methoden vertragen Temperaturerhöhungen offenbar ziemlich gut. Es müssen nur einige brauchbare Individuen den „Hitzeschock“ überstanden haben. Durch den Vererbungsmechanismus können sich die guten Gene in der Abkühlungsphase dann wieder ausbreiten.

Auch die Crossing-Over-Verfahren kommen im Rahmen der Abkühlung zu einem gut sichtbaren Ergebnis: Schließlich habe die guten Lösungen die Welt überflutet und schlechtere kommen nicht mehr zum Zuge.

Im Rahmen der durchgeführten Experimente hat die lokale Paarung ($k=3$) nicht zu schlechteren Ergebnissen geführt als das Verfahren mit globaler Paarung. Die lokale Paarung hat den Vorteil, dass sich der Optimierungsprozess im „Weltfenster“ besser beobachten lässt. Die Lösungen bilden sozusagen Kolonien von Quasi-Spezies. Dadurch lässt sich dieser Prozess gut steuern.

Regeln für die Steuerung der Optimierung (Wahl der Temperatur T)

Erste Beobachtung: Eine Temperaturerhöhung ist dann sinnvoll, wenn der Optimierungsprozess im Zuge der Abkühlung zum Stillstand gekommen ist. Kritisch ist die Dauer der Phase erhöhter Temperatur: Wird das Chaos zu groß und bleiben keine halbwegs brauchbaren Individuen übrig, kann der Optimierungsprozess ins Leere laufen. Man verspielt den Vorteil der ziemlich guten, weil topologisch sortierten, Anfangslösung.

Zeitbedarf

Messen am System

Der Zeitbedarf eines Optimierungslaufs für das zweite Beispiel lag, je nach Zahl der steuernden Handeingriffe, im Bereich von einer bis zehn Minuten. Das Programm lief dabei auf einem handelsüblichen Laptop mit 1.6 GHz-Pentium-Prozessor und 504 MB RAM unter dem Betriebssystem Windows XP.

Komplexitätsanalyse

Die Abhängigkeit des Zeitaufwands aller Algorithmen von der Anzahl der einzuplanenden Arbeitsvorgänge (Auftragsumfang) ist höchstens linear. Im Einzelnen:

- Der Mutationsoperator hängt gar nicht vom Auftragsumfang ab und der Crossing-Over-Operator linear.
- Bei der Bewertungsschleife ist die Sachlage etwas komplizierter. Der Aufwand hängt hier auch von der Anzahl der Maschinentypen ab und von der Anzahl der Vorgänger je Arbeitsvorgang. Sind diese Anzahlen jeweils begrenzt, wächst die Anzahl der Schleifendurchläufe ebenfalls maximal linear mit dem Auftragsumfang.

Weiteres Vorgehen

Anstehende Entscheidungen und Untersuchungen

- Das Programm wird der interessierten Öffentlichkeit und den Industrievertretern vorgeführt mit dem Ziel, eine Aussage über die Brauchbarkeit zu bekommen und das weitere Vorgehen festzulegen. Möglicherweise müssen zu diesem Zweck weitere Tests durchgeführt werden.
- Es sind weitere Mutationsoperatoren, insbesondere SWAP und INSERT, zu implementieren und in ihren Leistungen mit SWAP0 zu vergleichen.
- Der COX-Operators ist zu implementieren und in seiner Leistung mit COS zu vergleichen.
- Die Regeln für die Steuerung des Optimierungsprozesses sind auf der Basis systematischer Experimente zu präzisieren.

Schritte hin zu einem produktiven Programm

1. *Schritt.* Erfassung der in der Praxis relevanten Restriktionen. Hier ist eine erste Liste:

- Vorgabe der verfügbaren Fläche (Berücksichtigung von Hallensäulen, Pausenplätzen, Lifterschnittstellen zu übergeordneten Transportsystemen, Nachbaranlagen)
- Vorgabe der Fügetechnologie (Fügetechnologien werden teilweise vorgegeben und beeinflussen die Dauer der Operation)
- Sicherheitstechnische Bestimmungen (Sicherheit von Werkerarbeitsplätzen, Abstand von Robotern zum Schutzzaun)
- Vorgaben zur Instandhaltung (Zugang zu Robotern von außen zwecks Austausch im Falle eines Defekts, unter Umständen Einbau von Roboterentnahmeschienen, Abstände von Komponenten zueinander, Stichwort Wartungszugänglichkeit)
- Vorgabe des Materialtransports (innerhalb der Anlage, Robotertransport)
- Werkerauslastung und Materialanstellung (Kann man Werker einsparen durch Teilepuffer, die nur einmal pro Schicht beladen werden oder ist eine zyklische Beladung erforderlich, kann ein Werker mehrere Arbeitsbereiche bedienen, zurückgelegte Wegstrecke pro Schicht)
- Ergonomie (Arbeitshöhe der Einlegevorrichtung, Ladehilfen auf Grund von erhöhten Bauteilgewichten,.....)

2. *Schritt.* Anpassung der Bewertungsfunktionen an die Optimierungswünsche der Fertigungszellen-Designer.

3. *Schritt*. Neuformulierung des Modells und der Eingabesprache.

4. *Schritt*. Erprobung an einer konkreten Designaufgabe.

Durchführung

Denkbar ist ein Kooperationsprojekt Industrie-Hochschule. Die Leistungen der Hochschule könnten als Praxissemester und Bachelorarbeiten sowie im Rahmen von Forschungssemestern oder außerplanmäßigen Freistellungen erbracht werden. Die außerplanmäßige Freistellung von Professoren müssten durch die Industrie oder durch einen Drittmittelgeber finanziert werden.

Literatur

- Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass. 1974
- Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D.: Data Structures and Algorithms. Addison-Wesley, Reading, Mass. 1983
- Dangelmaier, Wilhelm: Fertigungsplanung. Springer-Verlag, Berlin, Heidelberg 2001
- Dewdney, Alexander K.: The (New) Turing Omnibus. 66 Excursions in Computer Science. Freeman, 1993
- Grams, Timm: KoopEgo. Links zur Programmdokumentation und zum Programmtext (2007)
<http://www.hs-fulda.de/~grams/OekoSimSpiele/KoopEgoProgramm/KoopEgo.pdf>
<http://www.hs-fulda.de/~grams/OekoSimSpiele/KoopEgoProgramm/KoopEgo.jar>
- Holland, John H.: Genetische Algorithmen. Spektrum der Wissenschaft (1992) 9, 44-51
- Michalewicz, Zbigniew; Fogel, David B.: How to Solve It: Modern Heuristics. Springer-Verlag, Berlin, Heidelberg 2000
- Neumann, Klaus; Morlock, Martin: Operations Research. Carl Hanser Verlage München, Wien 1993
- Rao, Y.; Li, P.; Shao, X.; Shi, K.: Agile manufacturing system control based on cell re-configuration. International Journal of Production Research 44 (2006), 10, 1881-1905
- Rechenberg, Ingo: Evolutionsstrategie. Optimierung technischer System nach den Prinzipien der biologischen Evolution. Friedrich Frommann Verlag, Stuttgart-Bad Cannstatt 1973
- Sanchez, Luis, M.; Rakesh, Nagi: A review of agile manufacturing systems. INT. J. PROD. RES., 39 (2001) 16, 3561-3600