HOTFRAME

Heuristische Lösung diskreter Planungsprobleme mittels wiederverwendbarer Software-Komponenten[1]

Andreas Fink und Stefan Voß[2]

In diesem Artikel soll ein Beispiel dafür gegeben werden, inwiefern sich durch die interdisziplinäre Verknüpfung und Anwendung von primär theoretischen Erkenntnissen aus verschiedenen Fachgebieten – hier aus dem Operations Research und der Software-Technik – praktische Problemstellungen effizient lösen lassen. Hierzu werden zunächst beispielhaft zwei Planungsprobleme aus der Praxis beschrieben. Daraufhin wird diskutiert, auf welche Weise eine effiziente Lösung solcher Problemstellungen in einer homogenen Weise ermöglicht werden kann. Dazu wird ein Ansatz beschrieben, entsprechende Lösungsverfahren in einer Methoden- bzw. Problem-Bibliothek zusammenzufassen, deren Komponenten auf einfache Art und Weise adaptiert und wiederverwendet werden können.

1. Einleitung

In der Praxis ergeben sich eine Vielzahl von planerischen Aufgaben in Anwendungsbereichen wie Kommunikation, Produktion, Verkehr, u.v.a., die zu diskreten Problemstellungen der Art führen, daß eine Systemkonfiguration – z.B. im Sinne einer Allokation von Ressourcen oder der Planung eines Ablaufs – im Hinblick auf eine gewisse Zielsetzung unter Berücksichtigung gegebener Einschränkungen vorzunehmen ist. Im weiteren seien zur Veranschaulichung zunächst zwei Problemstellungen in vereinfachter Form beschrieben; diese werden unten im Hinblick auf eine „Lösung“ wieder aufgegriffen. Die Lösungen selbst lassen sich vermöge eines Frameworks für moderne heuristische Prinzipien bestimmen, dessen Konzeption im Mittelpunkt dieses Beitrags steht.

2. Probleme und Lösungsmethoden

Planung von Filmproduktionen

Zu planen sei das Drehen der Szenen eines Filmes. Dabei wird davon ausgegangen, daß durch das Drehbuch eine Menge von Szenen definiert ist; diese können in beliebiger Reihenfolge gedreht werden. Für jede Szene ist vorgegeben, welche Rollen bzw. diesen eindeutig zugeordnete Schauspieler jeweils beteiligt sind, und wie lange das Drehen dieser Szene erwartungsgemäß dauern wird. Die Rollen sind bereits eindeutig mit Schauspielern besetzt; jeder Schauspieler erhält ein spezifisches Entgelt, das sich über einen gewissen Periodensatz (z.B. einen Tagessatz) multipliziert mit der Dauer seiner Anwesenheit am Drehort bestimmt. Dabei wird davon ausgegangen, daß ein Schauspieler vom Beginn der ersten Szene bis zum Ende der letzten Szene, an denen er beteiligt ist, zu entlohnen ist. Die Zielsetzung besteht nunmehr darin, das Drehen der Szenen so zu planen, daß das an die Schauspieler zu entrichtende Gesamtentgelt minimiert wird.

In Abbildung 1 sind die entsprechenden Daten für ein fiktives Filmprojekt mit vier Szenen, angeordnet in ihrer „natürlichen“ Reihenfolge, und vier Schauspielern veranschaulicht.

Personen	A	B	C	D
				
Szenen	1	2	4	3 €
Einleitung		✖	✖	✖
Krise	✖	✖	✖	
Mord	✖			✖
Happyend		✖	✖	

Abbildung 1: Daten eines Filmprojekts

---

[1] Erschienen in: OR News, Nr. 4 / November 1998, S. 18–24.

[2] Technische Universität Braunschweig, Abteilung Allg. BWL, Wirtschaftsinformatik und Informationsmanagement, Institut für Wirtschaftswissenschaften, Abt-Jerusalem-Str. 7, D-38106 Braunschweig. E-mail: {a.fink, stefan.voss}@tu-bs.de.

Dabei wird vereinfachenderweise angenommen, daß das Drehen für jede der Szenen jeweils einen Tag beansprucht. Durch die Kreuze ist gekennzeichnet, welche der Schauspieler Antonio, Bridget, Charly oder Duffy (A, B, C oder D) an der jeweiligen Szene beteiligt sind. In der Euro-Zeile sind die Tagessätze der jeweiligen Schauspieler in Tausend-Euro angegeben. Wird das Drehen entsprechend der natürlichen Reihenfolge durchgeführt, so ergibt sich ein an die Schauspieler zu entrichtendes Gesamtentgelt von 2.000+8.000+16.000+9.000 = 35.000 Euro.

Kontinuierliche Fließfertigung

Im Rahmen der Produktionsplanung spielen Probleme der Maschinenbelegungsplanung eine wesentliche Rolle [1]. Hier behandeln wir ein Problem der Fließfertigung mit spezieller Auftragscharakteristik. Die Problemstellung besteht darin, die Fertigung von *n* Aufträgen durch Ausführung von jeweils *m* Arbeitsgängen auf entsprechenden Maschinen zu planen, wobei zur Ausführung jedes spezifischen Arbeitsgangs deterministische Bearbeitungszeiten als vorgegeben angenommen werden. Natürlich kann zu jedem Zeitpunkt jede Maschine höchstens einen Auftrag bearbeiten sowie jeder Auftrag von höchstens einer Maschine gleichzeitig bearbeitet werden. Ferner ist die Maschinenfolge, d.h. die Sequenz der zu den Arbeitsgängen korrespondierenden Maschinen, für alle Aufträge identisch. Bei der hier betrachteten Problemstellung kommt noch hinzu, daß die Bearbeitung aufeinanderfolgender Arbeitsgänge eines Auftrags kontinuierlich sein muß; solche Bedingungen können sich u.a. dadurch ergeben, daß aufgrund technologischer Restriktionen eine kontinuierliche Bearbeitung notwendig ist (z.B. in chemischen Verarbeitungsprozessen).

In Abbildung 2, einem sogenannten maschinenorientierten Gantt-Diagramm, ist eine mögliche Maschinenbelegung für eine Problemstellung mit vier Aufträgen und vier Maschinen abgebildet.

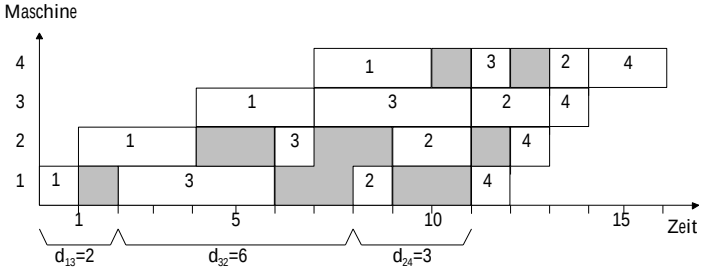

Abbildung 2: Maschinenorientiertes Gantt-Diagramm

Auf der Ordinate ist dabei die Zeit aufgetragen, auf der Abszisse sind die Maschinen angegeben. Dabei wird angenommen, daß alle Aufträge die selbe Maschinenfolge (1,2,3,4) besitzen; die Bearbeitungszeiten der einzelnen Arbeitsgänge der verschiedenen Aufträge sind durch die Breiten der Kästen ersichtlich. Die grau schraffierten Flächen stellen Leerzeiten dar, die sich bei der gewählten Reihung der Aufträge zwingend ergeben, damit die Bedingung der kontinuierlichen Bearbeitung aller Aufträge erfüllt werden kann. Beispielsweise ergibt sich auf Maschine 1 eine Mindestverzögerung zwischen dem Beginn von Auftrag 1 und dem Beginn von Auftrag 3 von d_{13}=2 Zeiteinheiten, obwohl der erste Arbeitsgang von Auftrag 1 lediglich eine Zeiteinheit benötigt.

Als Zielsetzung sei nun die Minimierung der Summe der Durchlaufzeiten aller Aufträge angenommen, wobei unter der Durchlaufzeit eines Auftrags die Zeitspanne von seiner Bereitstellung bis zur komplettierten Bearbeitung auf der letzten Maschine verstanden wird; hier sei angenommen, daß alle Aufträge zum Zeitpunkt 0 bereit stehen. Für die in Abbildung 2 dargestellte Maschinenbelegung ergibt sich damit eine Summe der Durchlaufzeiten von 10+12+14+16 = 52 Zeiteinheiten. Es stellt sich nun die Frage, ob über eine andere Reihung der Aufträge eine Verringerung der Summe der Durchlaufzeiten erreicht werden kann, bzw. bei welcher Auftragsfolge die Summe der Durchlaufzeiten minimal ist.

Problemmodellierung und Lösungsmethoden

Diese zwei beschriebenen Problemstellungen repräsentieren einen sehr geringen Anteil aus der Menge von diskreten (kombinatorischen) Optimierungsproblemen, die in der Praxis auftreten. Zur Lösung solcher Problemstellungen – im Sinne einer Entscheidungsunterstützung über die Generierung von Lösungsverschlägen – werden in der Regel adäquate Software-Systeme benötigt, die entsprechende Lösungsverfahren verkörpern. Die Forschung in den Bereichen Operations Research und Mathematische Optimierung beschäftigt sich zu großen Teilen mit der theoretisch orientierten Entwicklung entsprechender quantitativer Lösungsverfahren. Es ist jedoch zu konstatieren, daß die individuelle Adaption und Neu-Implementierung von Lösungsverfahren für die vielfältigen praktischen Problemstellungen in der Regel zu aufwendig ist. Dementsprechend benötigt man zunächst ein gemeinsames Paradigma, innerhalb dessen die Problemstellungen abgebildet werden können, wodurch die Nutzung implementierter allgemeiner Lösungsverfahren ermöglicht wird.

Ein weit verbreiteter Ansatz in diesem Zusammenhang ist die Nutzung von Standardsoftware für lineare bzw. ganzzahlige Optimierungsprobleme (z.B. CPLEX). Dies erfordert zunächst jedoch in der Regel eine vereinfachende Problemmodellierung als eine entsprechende mathematische Formulierung durch die Definition von Parametern, Variablen mit bestimmten Wertebereichen, und in der Regel einer linearen Zielfunktion und linearen Nebenbedingungen. Weiterhin sind eine sehr große Klasse von Problemstellungen (die sogenannten *NP*-schweren Probleme) ab einer gewissen Größenordnung bisher hiermit nicht mit vertretbarem Zeitaufwand optimal lösbar. Dies trifft u.a. auf die beiden oben beschriebenen Problemstellungen zu.

Auch aus diesen Gründen werden viele der in der wissenschaftlichen Forschung entwickelten Modelle und Lösungsmethoden in der Praxis nicht eingesetzt. Eine aus dieser Problematik resultierende Tendenz ist das größere Gewicht, das seit den 80er Jahren heuristischen Verfahren zugemessen wird. Heuristiken geben den (teilweise fraglichen) Anspruch einer *optimalen* Problemlösung explizit auf. Eine Heuristik (von griech. *heurìskein* = finden, entdecken) ist eine Methode, die über die Anwendung „sinnvoller" Regeln und Vorgehensweisen („patient rules of thumb") gute Lösungen generieren soll – beispielsweise über die sukzessive Auswahl „vielversprechender" Alternativen. Bei der Verwendung heuristischer Methoden wird implizit berücksichtigt, daß die Problemabbildung in der Regel vereinfacht ist und die Daten un-

scharf sind. Deshalb wird der unbedingte Anspruch nach dem Auffinden einer optimalen Lösung aufgegeben; Ziel ist nunmehr die effiziente Bestimmung einer „hinreichend guten" Lösung.

Wir beschränken uns hier auf heuristische Methoden, die auf der Idee der lokalen Suche basieren. Zentrales Paradigma lokaler Suchverfahren ist die iterative Modifikation von Lösungen, um sukzessive zu besseren Lösungen zu gelangen. Hierzu sei eine *Lösung* nicht als die optimale Lösung sondern als eine mögliche konkrete Systemkonfiguration, Ressourcenallokation, Ablaufplanung o.ä. einer spezifischen Probleminstanz definiert. Beispielsweise wäre damit eine Lösung für das oben beschriebene Filmproduktionsproblem als eine konkrete Szenensequenz definierbar. Nunmehr kann man eine „Lösungsnachbarschaft" definieren, die für eine gegebene Lösung eine Menge leicht modifizierter Lösungen bestimmt. Für das Filmproduktionsproblem kann dies beispielsweise heißen, zwei Szenen zu vertauschen. In jeder Iteration des sogenannten Greedy-Suchverfahrens wird nun aus einer solchermaßen definierten Nachbarschaft jeweils eine bestbewertete Nachbarlösung als neue Lösung ausgewählt. Dies wird fortgeführt, bis eine Lösung erreicht wird, die keinen besser bewerteten Nachbarn besitzt; diese Lösung wird dann als lokales Optimum bezeichnet.

Hierauf basierend gibt es verschiedenste Ansätze, solche Verfahren durch Strategien zu erweitern, über die versucht wird, lokale Optima zu überwinden und möglicherweise dem globalen Optimum näher zu kommen; solche allgemein und weitgehend unabhängig vom Problemtyp definierbare Methoden werden auch als Meta-Heuristiken bezeichnet [9]. Beispielsweise werden beim Simulated Annealing (mit im Laufe der Zeit abnehmender Wahrscheinlichkeit) auch Verschlechterungen des Zielfunktionswertes zugelassen, während die Idee des Tabu Search darauf basiert, Informationen über die bisher „besuchten" Lösungen bzw. die ausgeführten „Züge" zu den Nachbarlösungen zur Steuerung des Suchverlaufs über die Verwendung sogenannter Tabu-Kriterien, die die jeweilige Nachbarschaft dynamisch modifizieren, zu verwenden.

Zurückkommend auf die oben getroffenen Feststellung ist nunmehr zu analysieren, inwieweit solche heuristische Methoden eine homogene Modellierung von Problemstellungen ermöglichen, die die Wiederverwendung von entsprechenden Software-Komponenten gestatten. Bei einer solchen Modellierung stehen naturgemäß die Konzepte *Lösung* und *Nachbarschaft* im Mittelpunkt; darauf aufbauend sind lokale Suchverfahren bereits mittels eines abstrakten Pseudo-Codes problemunabhängig formulierbar. Das Greedy-Suchverfahren könnte für ein Minimierungsproblem z.B. folgendermaßen formuliert werden:

Greedy-Suche < Lösungsraum L mit Bewertungsfunktion f ,
Nachbarschaftsstruktur N >

Eingabe: Startlösung $s \in L$

Solange $\exists\, s' \in N(s)$ mit $f(s') < f(s)$
Setze $s := \text{argmin}\{ f(s') \mid s' \in N(s) \}$

Ausgabe: Lokales Optimum s

Die konkrete Parametrisierung dieses Verfahrens erfordert damit entsprechende Implementierungen des Lösungsraums und der Nachbarschaftsstruktur als Software-Komponenten. Im weiteren stellt sich damit die Zielsetzung, zu untersuchen, inwieweit es möglich ist, generische heuristische Lösungsmethoden problemunabhängig zu implementieren, was die Konzeption einer entsprechenden allgemeinen Software-Architektur bzw. die Spezifikation der von den Software-Komponenten zu erfüllenden Schnittstellen bedingt. Dies ist auch vor dem Hintergrund zu sehen, daß in dem hier diskutierten Problembereich die Entwicklung adäquater, wiederverwendbarer Software („systems building", [8]) ein wesentlicher Schritt im Rahmen eines entsprechenden Technologietransfers auf dem Weg zu einer effektiven praktischen Entscheidungsunterstützung ist.

3. Konzeption eines Frameworks für heuristische Suchverfahren

Wiederverwendung von Software

Die Zielsetzung, in Analogie zur Produktion materieller Güter Software zu entwickeln, indem lediglich aus einer Menge von wiederverwendbaren Komponenten die benötigten ausgewählt, gegebenenfalls adaptiert und kombiniert werden, existiert seit langem. Auch wenn man von diesem Ziel immer noch weit entfernt ist, versprechen einige neuere Konzepte und Techniken einen erheblichen Schritt nach vorne. Der zentrale Ansatz des *Framework-Konzepts* ist die Abbildung der „Gemeinsamkeiten" abgegrenzter Anwendungsbereiche als generische Software-Komponenten, die in eine Architektur eingebettet sind, welche die Komponenten-Kooperation definiert [2]. Die Software-Komponenten besitzen in der Regel Mechanismen zur Anpassung an die jeweiligen Anforderungen. Durch die Vorgabe einer konzeptionellen Architektur im Sinne eines Anwendungsgerüsts, über das wesentliche logische Abläufe definiert werden, geht ein Framework über eine Klassenbibliothek (im Sinne einer „Tool-Box") hinaus. Frameworks stehen in einem engen Zusammenhang mit dem objektorientierten Software-Paradigma, da in der Regel erst durch die Anwendung objektorientierter Software-Techniken [7] angestrebte Framework-Eigenschaften wie insbesondere Flexibilität, Erweiterbarkeit und Wiederverwendbarkeit erreicht werden können. Nichtsdestotrotz beeinflußt die hierzu in der Regel konkurrierende Zielsetzung Effizienz die Gestaltung einer entsprechenden Architektur, die damit immer einen Kompromiß darstellen wird.

Die zentrale Idee objektorientierter Software-Entwicklung ist die abstrahierende Abbildung allgemeiner Konzepte des Anwendungsbereichs in Klassen. (In diesem Beitrag wird der Begriff „Komponente" im wesentlichen austauschbar zu dem Begriff „Klasse" verwendet.) Eine Wiederverwendung dieser Klassen in einem neuen Kontext bedingt in der Regel eine entsprechende Anpaßbarkeit der Klassen, die im wesentlichen durch die beiden alternativen Mechanismen Vererbung (dynamischer Polymorphismus) und Generizität (statischer Polymorphismus) erreicht

werden kann. Über Vererbung können Eigenschaften und Verhalten allgemeiner Klassen auf spezialisierte Unterklassen übertragen und dort angepaßt werden. Die Anpassung generischer Klassen findet über Typ-Parameter statt. Das in diesem Zusammenhang anwendbare Konzept des *Generischen Programmierens* wurde erst in letzter Zeit geprägt; dementsprechend sind die Erfahrungen mit diesem Programmierparadigma noch eingeschränkt. Als offenes Problem ist beispielsweise die ambivalente Beziehung zum objektorientierten Paradigma zu sehen; so wird die Kapselung beim generischen Programmierstil in einem gewissen Sinne explizit verletzt [6]. Gerade auch die hier relevante Wiederverwendung von Algorithmen erscheint unter Anwendung traditioneller objektorientierter Techniken problematisch [10]. Generizität kann in Abhängigkeit von der verwendeten Implementierungssprache auf verschiedene Arten erreicht werden. In C++ wird Generizität durch die Nutzung des Template- bzw. Schablonen-Mechanismus ermöglicht, über den gewisse Typ-Parameter variabel gehalten werden.

Grundlegende Framework-Konzeption

Die im folgenden diskutierte Konzeption basiert primär auf generischen Komponenten; sekundär kommt Vererbung zum Einsatz, um die Gemeinsamkeiten entsprechender Komponenten über Spezialisierungshierarchien abzubilden. In dem hier diskutierten Kontext sind erwartungsgemäß grundlegenden Komponenten zu entwickeln, vermöge derer verschiedene heuristische Verfahren abgebildet werden. Im weiteren sei dies beispielhaft für lokale Suchverfahren veranschaulicht. Eine auf dem Prinzip der lokalen Suche basierende Methode sollte bezüglich verschiedener Nachbarschaftsdefinitionen generisch verwendbar sein; eine entsprechende Implementierung muß damit die konkret zu verwendende Nachbarschaft „offen“ lassen. Eine entsprechende Konzeptionierung einer Framework-Architektur erfordert zunächst eine „Faktorisierung“ in fein-granulare, im wesentlichen hinsichtlich ihrer Funktionalität orthogonaler Komponenten; bei der Implementierung konkreter Software-Systeme erfolgt dann in der Regel eine hierarchische Instanziierung der Typ-Parameter generischer Komponenten. Da eine vollständige Beschreibung eines solchen Frameworks den hiesigen Rahmen sprengen würde, seien im weiteren einige zentrale Konzepte beispielhaft veranschaulicht.

Meta-Heuristiken für diskrete Optimierungsprobleme besitzen in der Regel einen gewissen Anteil nicht-problemspezifischer Anteile. Selbst eine so einfache Methode wie Simulated Annealing umfaßt bei entsprechend weitgehenden Parametrisierungsmöglichkeiten durch verschiedene Akzeptanz- und Abkühlungsfunktionen (beispielsweise die Durchführung eines „strategischen Wiederaufheizens“), bei gleichzeitiger Implementierung eines robusten „Default“-Verhaltens über eine entsprechende Autoadaption, einen nicht unerheblichen Implementierungsaufwand, so daß sich eine grobkörnige Trennung der Implementierung von Meta-Heuristiken auf der einen Seite und problemspezifischen Konzepten auf der anderen Seite zwingend anbietet. Deutlich wird dies insbesondere bei entsprechend komplexen Verfahren des Tabu Search [5], bei denen eine problemspezifische Implementierung in der Praxis in der Regel zu aufwendig wäre. Die verschiedenen Konzepte werden im wesentlichen mittels der Definition generischer Klassen abgebildet, die in Abhängigkeit vom jeweiligen Zusammenhang parametrisierbar sein müssen bzw. sich entsprechend polymorph verhalten.

Zur Veranschaulichung wird im weiteren die zentrale Schnittstelle zwischen einem Verfahren der lokalen Suche und einer spezifischen Problemstellung skizziert. Die Verbindung zwischen Heuristik und Lösung wird dabei im wesentlichen über die Nachbarschaft stattfinden; als Verbindungsglied bieten sich dabei sogenannte Nachbarschafts-Iteratoren an, über die eine Traversierung der Nachbarschaft einer Lösung möglich ist. Zunächst muß ein solcher Nachbarschafts-Iterator lediglich einige wesentliche Funktionalitäten erfüllen:

- Erzeugung eines Nachbarn einer Lösung *s* („Konstruktor“).
- Inkrementieren zum nächsten Nachbarn („Inkrement-Operator“).
- Bewertung eines Nachbarn („Dereferenzierung“); ggf. wird die eigentliche Bewertung an die entsprechende Lösungsklasse delegiert.

Ein einfaches Greedy-Suchverfahren operiert im wesentlichen auf der hierdurch abgebildeten Nachbarschaft (Auswahl des am besten bewerteten Nachbarn, Übergabe des Nachbarn an eine Lösungs-Komponente zur Ausführung).

Für eine der oben beschriebenen Problemstellungen heißt dies konkret, daß entsprechend des zentralen Paradigmas objektorientierten Programmierens zwei grundlegende gekapselte Klassen mit einer spezifizierten Schnittstelle zu definieren sind. Eine Lösungsklasse für das Filmproduktionsproblem wird dabei in etwa folgende Funktionalitäten zu erfüllen haben:

- Erzeugung einer Startlösung aus einer gegebenen Problemstellung, beispielsweise die „natürliche“ Szenensequenz.
- Bewertung der aktuellen Lösung über die Berechnung des an die Schauspieler bei der jeweiligen Szenensequenz zu entrichtenden Gesamtentgelts.
- Modifikation der aktuellen Lösung zu einem angegebenen Nachbarn durch die Vertauschung der durch diesen Nachbarn spezifizierten Szenen.

Eine entsprechende Nachbarschafts-Iterator-Klasse mag dann folgende Funktionalitäten bereitstellen:

- Erzeugung des „ersten“ Nachbarn über die Speicherung der Indizes der ersten und zweiten Szene.
- Übergang zum nächsten Nachbarn durch Inkrementieren der Indizes derart, daß letztendlich alle möglichen Vertauschungen von zwei Szenen betrachtet werden.
- Bewertung eines Nachbarn durch adaptive Berechnung der Veränderung des Gesamtentgelts.

Durch einige Ergänzungen sind weitere auf der lokalen Suche aufbauende Heuristiken abbildbar. So erfordern Methoden wie

Simulated Annealing darüber hinaus lediglich die Möglichkeit, einen Nachbarn (pseudo-) zufällig zu generieren (über einen zusätzlichen Konstruktor). Komplexere Methoden wie Tabu Search benötigen eine weitergehende Funktionalität (z.B. die Erzeugung gegebenenfalls mehrerer Zug-Attribute, durch die ein Übergang zu einem Nachbarn abgebildet wird, zur Abspeicherung in einer Tabu-Liste). Über eine durchgängige Anwendung dieser Konzepte erhält man eine weitgehende Generizität implementierter Meta-Heuristiken, die damit unabhängig von spezifischen Problemstellungen implementierbar sind. So kann man damit entsprechend parametrisierte Komponenten *GreedySuche<Lösungsraum, Nachbarschaftsstruktur>*, *SimulatedAnnealing<Lösungsraum, Nachbarschaftsstruktur, Akzeptanz-Funktion, Abkühlungs-Schema>* oder *TabuSearch<Lösungsraum, Nachbarschaftsstruktur, Tabu-Kriterium, Diversifikation>* implementieren und nachfolgend für beliebige Problemstellungen einsetzen, für die entsprechende problemspezifische Komponenten entwickelt wurden.

Von den Autoren wurde ein auf den oben skizzierten Ideen basierendes Framework konzipiert und prototypenhaft implementiert: HOTFRAME („Heuristic OpTimization FRAMEwork"). Dieses wurde für eine Reihe von bekannten diskreten Optimierungsproblemen erfolgreich angewandt – neben den in diesem Beitrag verwendeten Beispielen u.a. für Netzwerkkonstruktions- und -konfigurationsprobleme sowie weitere Produktionsplanungsprobleme (vgl. hierzu z.B. [3, 4] sowie http://www.winforms.phil.tu-bs.de/winforms/research/hotframe.html).

Es sei nun angenommen, daß ein Pool von allgemeinen Lösungsmethoden als generisch verwendbare Software-Komponenten vorliegt. Für die beiden oben beschriebenen Problemstellungen heißt dies nun, daß die Anwendung dieser Lösungsverfahren nur noch eine entsprechende Implementierung der problemspezifischen Klassen gemäß der Spezifikation bedingt. Die Spezifikation führt in der Regel dazu, daß eine Implementierung durch die exakte Vorgabe der abzubildenden Funktionalitäten relativ einfach wird. Der Programmierer muß nicht mehr alle Zusammenhänge und die globale Anwendungslogik konzipieren bzw. implementieren, sondern hat nur noch kleine „Häppchen" zu vervollständigen.

Analysiert man das Fließfertigungs- und das Filmproduktionsplanungsproblem genauer, so stellt man fest, daß sich eine Lösung einer der beiden Problemstellungen jeweils als eine Permutation der Aufträge oder der Szenen bzw. im allgemeinen der Zahlen 1,...,*n* darstellen läßt. Potentielle Nachbarschaftsstrukturen (z.B. Verschiebung, Austausch, Invertierung einer Teilsequenz) für solche als Permutation abbildbare Lösungen sind damit problemunabhängig implementierbar. Dementsprechend kann man prinzipiell bei der Implementierung der problemspezifischen Klassen für das Fließfertigungsproblem alle bisher implementierten Lösungsmethoden wiederverwenden; bei einer anschließenden Implementierung problemspezifischer Klassen für das Filmproduktionsplanungsproblem können darüber hinaus auch die Klassen wiederverwendet werden, die allgemeine Lösungskonzepte und Nachbarschaftsstrukturen für Permutationsprobleme abbilden. Unter Fortführung dieser Vorgehensweise gelangt man damit schließlich zu einem Pool von implementierten Lösungsmethoden und verschiedensten problemspezifischen Komponenten, was dementsprechend im Idealfall dazu führt, daß der Grenzaufwand für die Implementierung neuer Methoden bzw. Probleme kontinuierlich sinkt.

Lösung der aufgeworfenen Probleminstanzen

Nunmehr soll noch auf die Lösung der oben dargestellten Probleminstanzen zurückgekommen werden. Hierzu wird für das Fließfertigungsproblem, beginnend mit der in Abbildung 2 dargestellten Maschinenbelegung als Startlösung, ein einfaches Greedy-Suchverfahren angewandt. Die einer Lösung benachbarten Lösungen ergeben sich hier unter Verwendung der Zug-Definition „Vertauschung von zwei Aufträgen". Dies führt dann zu folgendem Ablauf, wenn in jedem Iterationsschritt ein bester Nachbar ausgewählt wird:

Startlösung: (1,3,2,4), Summe der Durchlaufzeiten 52

Iteration 1: Vertauschung der Aufträge 1 und 4

- Neue aktuelle Lösung: (4,3,2,1)
  Summe der Durchlaufzeiten: 48

Iteration 2: Vertauschung der Aufträge 3 und 2

- Neue aktuelle Lösung: (4,2,3,1)
  Summe der Durchlaufzeiten: 41

Iteration 3: Vertauschung der Aufträge 3 und 1

- Neue aktuelle Lösung: (4,2,1,3)
  Summe der Durchlaufzeiten: 40

Lösung: (4,2,1,3) stellt ein lokales Optimum dar.

Die hiermit erhaltene Lösung mit einer Gesamtbearbeitungsdauer von 40 Zeiteinheiten, die in diesem Fall auch eine optimale Lösung verkörpert, ist in Abbildung 3 dargestellt.

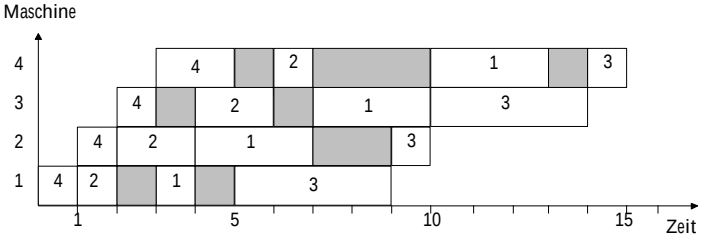

Abbildung 3: Verbesserte Maschinenbelegung

Nunmehr bleibt noch die Optimierung der Filmproduktion. Hierzu kann prinzipiell die gleiche Implementierung wie bei dem Maschinenbelegungsproblem verwendet werden. Lediglich die Bewertung einer Lösung ist zu modifizieren. Startet man mit der in Abbildung 1 dargestellten Reihung, so gelangt man über die sukzessive Ausführung der beiden Vertauschungen (Einleitung ↔

Mord) und (Mord ↔ Happyend) zur in Abbildung 4 veranschaulichten Szenensequenz. Das Happyend kommt hier ausnahmsweise mal bereits am Anfang; dafür kosten die Schauspieler jetzt lediglich noch 27.000 Euro.

Personen	A 	B 	C 	D 	
Szenen	1	2	4	3	€
Happyend		✖	✖		
Krise	✖	✖	✖		
Einleitung		✖	✖	✖	
Mord	✖			✖	

Abbildung 4: Optimierte Filmproduktion

Es ist hier jedoch darauf hinzuweisen, daß die durch diese Beispiele veranschaulichten Vorgehensweisen sich in der Praxis in der Regel natürlich nicht ganz so einfach darstellen. So wird es bei einer Filmproduktion gewisse Abhängigkeiten zwischen den Szenen geben (beispielsweise sind bestimmte Szenen gegebenenfalls aufgrund eines außergewöhnlichen Drehortes geblockt zu drehen), oder es wird eine Erweiterung der Zielsetzung von „Minimierung des Gesamtentgelts der Schauspieler“ auf eine „Minimierung der Gesamtkosten der Filmproduktion“ sinnvoll sein. Aufgrund einer modular aufgebauten Implementierung sind solche Erweiterungen durch entsprechende Anpassungen und Ergänzungen einzelner Software-Komponenten aber effizient durchführbar.

4. Ausblick

Im Anschluß an die Konzeptionierung und Implementierung eines entsprechenden Frameworks für heuristische Suchverfahren ergibt sich unmittelbar der Zwang für eine Erprobung und evolutionäre Anpassung der Framework-Architektur unter gleichzeitigem Aufbau eines entsprechenden Pools wiederverwendbarer Komponenten für verschiedene heuristische Methoden und vielfältige Problemstellungen aus mannigfaltigen Bereichen.

Letztendlich soll hier aber nochmals die oben beschriebene eigentliche Zielsetzung betont werden: die Entwicklung von Software-Systemen zur Entscheidungsvorbereitung und -unterstützung. Hierunter fällt neben der Neuentwicklung entsprechender Entscheidungsunterstützungssysteme auch die Anbindung an bestehende Software-Systeme („legacy systems“, z.B. Anbindung an PP-Modul von SAP/R3 o.ä.). Abschließend ist festzuhalten, daß über ein interdisziplinäres Zusammenspiel von Disziplinen wie Operations Research, Wirtschaftsinformatik und Software-Technik die Ausschöpfung von bisher oft ungenutzten Potentialen des Technologietransfers gerade im Bereich einer praxisorientierten Umsetzung von Planungsmethoden vorangebracht werden kann.

Literatur

[1] Domschke, W., Scholl, A., Voß, S. (1997). *Produktionsplanung – Ablauforganisatorische Aspekte*, 2. Auflage. Springer, Berlin.

[2] Fayad, M.E., Schmidt, D.C. (Hrsg.) (1997). *Object-oriented application frameworks*. Communications of the ACM 40 (10), 32–87.

[3] Fink, A., Voß, S. (1998a). *Applications of modern heuristic search methods to pattern sequencing problems*. Computers & Operations Research, to appear.

[4] Fink, A., Voß, S. (1998b). *Generic metaheuristics application to industrial engineering problems*. Proceedings of the 24th International Conference on Computers & Industrial Engineering, Elsevier, Amsterdam.

[5] Glover, F., Laguna, M. (1997). *Tabu Search*. Kluwer, Boston.

[6] Langer, A., Kreft, K. (1997). *Combining object-oriented design and generic programming*. OBJEKTspektrum, 9/97, 45–69.

[7] Meyer, B. (1997). *Object-Oriented Software Construction*, 2. Auflage. Prentice Hall, Englewood-Cliffs.

[8] Nievergelt, J. (1994). *Complexity, algorithms, programs, systems: The shifting focus*. Journal on Symbolic Computing 17, 297–310.

[9] Voß, S., Martello, S., Osman, I.H., Roucairol, C. (Hrsg.) (1998). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston.

[10] Weihe, K. (1997). *Reuse of algorithms: Still a challenge to object-oriented programming*. ACM SIGPLAN Notices 32 (10), 34–48.