

Optimization of Activity-Based Design Structure Matrices Using Genetic Algorithms

von **Christoph Meier**

Lehrstuhl für Raumfahrttechnik
Technische Universität München
85748 Garching, München
meierc@in.tum.de

1. Einführung und Motivation

Produktdesign und Produktentwicklung wird von vielen Firmen als ein Garant für ökonomischen Erfolg angesehen. Insbesondere ist die Fähigkeit, qualitativ hochwertige Produkte möglichst schnell und preiswert auf einem hochgradig konkurrenzbetonten und globalisierten Markt zu bringen von primärer Wichtigkeit [2]. Dies ist Motivation dafür Produktentwicklungsprozesse für komplexe Produkte (z.B. für die Fertigung eines Passagierflugzeuges) zu verbessern um letztendlich Produktionszeit und Produktionskosten zu senken [20], die über den wirtschaftlichen Erfolg eines Unternehmens entscheiden können. Die einschlägigen Literatur über Produktdesign und Produktentwicklung [13, 15, 16] vermag zwar keine universelle Theorie über die beste Strategie zu vertreten, dennoch können übereinstimmend vor allem zwei wichtige Schlussfolgerungen gefunden werden:

- Ein typisches Zeichen komplexer Produktentwicklungsprozesse sind Iterationen im Prozess.
- Iterationen in Produktentwicklungsprozessen sind maßgeblich für erhöhte Produktionskosten und für eine verlängerte Produktentwicklungszeit verantwortlich.

Iterationen in Produktentwicklungsprozessen treten genau dann auf, wenn es symmetrisch relationale Abhängigkeiten zwischen verschiedenen Prozessaktivitäten gibt. In diesem Fall tauschen Aktivitäten Informationen untereinander in beiden Richtungen aus und sind dementsprechend auch gegenseitig voneinander abhängig. Solche Abhängigkeiten in einem Produktentwicklungsprozess sorgen dafür, dass eine oder mehrere Aktivitäten im Prozess ohne vollständige Inputinformationen, die für die Ausführung notwendig sind aber erst zu späteren Zeitpunkten in der Prozesssequenz vorhanden sind, ausgeführt werden. Diese fehlenden Inputinformationen müssen dann bestmöglich geschätzt werden, d.h. man trifft eine Annahme über die Outputinformationen von Prozessaktivitäten zu späteren Zeitpunkten. Falls sich diese Annahmen als fehlerhaft erweisen, sollten (aber müssen nicht notwendigerweise) alle Aktivitäten, die diese falschen Annahmen als Inputinformation zur eigenen Ausführung herangezogen haben, wiederholt werden. Diese Änderungen ergeben möglicherweise wiederum Änderungen an den jeweiligen Outputergebnissen und weitere Aktivitäten im Prozess können wiederholt werden. Letztendlich kann eine Iteration im Produktentwicklungsprozess als Verbesserung bzw. Verfeinerung des Prozesses angesehen werden, der die resultierende Qualität erheblich verbessern kann. Firmen planen oftmals absichtlich solche Iterationen um ihre Produktqualität zu erhöhen oder um Innovation während der Produktentwicklung zu ermöglichen [5]. In einer Studie von Osborne [11] wurde z.B. aufgezeigt, dass 13% bis maximal 70% der

kompletten Produktentwicklungszeit bei Intel Iterationen zugeschrieben werden kann. Das heißt, Iterationen können durchaus geplant werden und sinnvoll sein. Nachteilige Auswirkungen ergeben sich allerdings genau dann, wenn diese im Prozess nicht geplant sind und damit lediglich Kosten und Zeit in die Höhe treiben.

In der Literatur können hauptsächlich drei Lösungsansätze gefunden werden um Iterationen zu handhaben bzw. zu reduzieren:

- Durch eine Verbesserung der Prozesssequenz indem die einzelnen Prozessaktivitäten umgeordnet werden. Da der Informationsfluss innerhalb eines Prozesses durch zeitliche Abhängigkeiten der jeweiligen Aktivitäten und somit durch die jeweilige Sequenz zur Ausführung der Aktivitäten gegeben ist, können somit Iterationen umgangen werden [19].
- Durch die Entwicklung neuer Automationstools um Iterationen schneller durchführen zu können [6, 22].
- Durch das Hinzufügen von Ressourcen (z.B. Personal) an kritischen und zeitintensiven Aktivitäten (Stichwort: Activity crashing) [21].

Die vorliegende Diplomarbeit untersucht das Auffinden einer bestmöglichen Sequenz eines Produktentwicklungsprozesses. Vor einer eigentlichen Verbesserung des Prozesses empfehlen einige Autoren als zunächst die Konstruktion eines geeigneten Modells für Produktentwicklungsprozesse [5, 17]. Für die Modellierung solcher Prozesse, die man auch als ein System mit Elementen und Relationen auffassen kann, wurden so genannte Design Structure Matrices (DSMs) [12] benutzt. Hier repräsentieren die Prozessaktivitäten die einzelnen Systemelemente und als Relation zwischen einzelnen Aktivitäten kann deren Informationsfluss angesehen werden. Eine DSM ist im Grunde die Adjazenzmatrixrepräsentation eines Digraphen (ein gerichteter Graph), die sowohl binäre als auch numerische Datenelemente enthalten kann. Abbildung 1 vergleicht die Darstellung eines kleinen Systems durch einen Digraph und durch eine DSM. Per Definition bedeutet ein Eintrag in Zelle $a_{i,j}$, dass die Aktivität i Inputinformationen von Aktivität j bekommt. Aus dieser gewählten Definition lässt sich schließen, dass Einträge oberhalb der DSM – Diagonalen Rücksprünge im modellierten Prozess bedeuten und damit Iterationen verursachen falls die einzelnen Prozessaktivitäten zeitlich in der Reihenfolge ausgeführt werden, in der sie auch in der DSM erscheinen. Die Sensitivität des Informationsflusses zwischen den Aktivitäten i und j kann durch den numerischen Wert der Zelle $a_{i,j}$ kontrolliert werden. In der gegenwärtigen DSM Literatur wurden reflexive Relationen nicht verwendet. Stattdessen können Diagonalelemente mit Zusatzinformationen versehen werden (z.B. Index oder Name der betreffenden Aktivität). Spalten und Zeilen werden üblicherweise mit dem Namen der Aktivität bezeichnet.

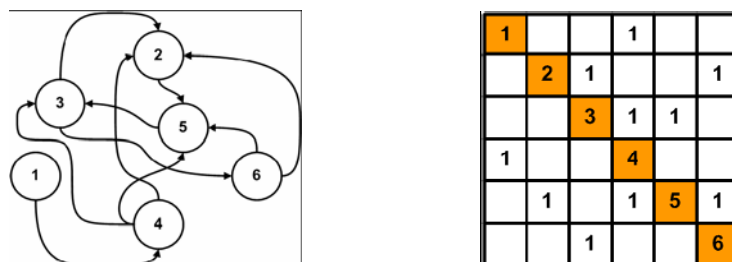


Abb. 1: Modellierung eines Systems mittels Digraph (links) und mit einer DSM (rechts) bestehend aus sechs Elementen und 10 Relationen.

Mit Hilfe von DSMs kann insbesondere die Prozessstruktur von großen und komplexen Prozessen sehr schön visualisiert werden und Abhängigkeiten des Informationsflusses sind wesentlich ersichtlicher und deutlicher erkennbar als im Falle eines Digraphen. Die generische DSM Methode ist des Weiteren aufgrund der Matrixrepräsentation für die elektronische Weiterverarbeitung mittels Computer von Vorteil da Digraphen zur Speicherung und Bearbeitung als Datenstruktur eine Liste (Adjazenzliste) oder ohnehin eine Matrix (Adjazenzmatrix oder Inzidenzmatrix) verlangen. Gegenüber weiteren traditionellen Modellen für Prozessabläufe, z.B. CPM, PERT oder Gantt Diagramme, bieten DSMs den Vorteil alle gängigen Arten des Informationsflusses (sequentielle, parallele oder gekoppelte Abhängigkeiten) zwischen zwei Aktivitäten zu modellieren. Vor allem Rücksprünge innerhalb des Prozesses können adäquat modelliert werden. Dies trifft nicht für die zuvor genannten Modelle zu.

Schlussendlich bleibt festzuhalten, dass durch eine Umordnung von DSM Elementen, unter Berücksichtigung eines vordefinierten Ziels (z.B. die Reduzierung von Relationen oberhalb der Diagonalen), die Sequenz, und damit auch der Informationsfluss, eines Produktentwicklungsprozesses verbessert werden kann. Dies wiederum kann zu kürzeren Entwicklungszeiten, niedrigeren Kosten und höherer Produktqualität führen. Ein Ziel, das jede Firma anstrebt um heutzutage auf dem Markt bestehen zu können.

2. Aufgabenstellung

Die Aufgabenstellung der Diplomarbeit umfasste im Wesentlichen drei Punkte:

- Untersuchung einer monokriteriellen Optimierung von DSMs mittels Genetischer Algorithmen (GAs). Dabei sollten die neuesten Erkenntnisse über GAs für die monokriterielle Optimierung berücksichtigt werden und ein entsprechendes Software Tool entwickelt werden, welches beliebige DSMs optimiert.
- Basierend auf einer Simulation für DSMs, die Kosten und Dauer eines Prozesses schätzt, sollte durch eine multikriterielle Optimierung mit Hilfe von GAs die so genannte Pareto-Front, eine Menge von besten Kompromisslösungen, für Kosten und Dauer einer DSM approximiert werden.
- Leistungsvergleich von weiteren meta-heuristischen Optimierungsverfahren mit GAs für eine mono- und multikriterielle Optimierung von DSMs.

Aus zeitlichen Gründen konnte der letzte Punkt in der Liste nur teilweise abgearbeitet werden, indem statt eines umfangreichen Leistungsvergleichs lediglich die Vor- und Nachteile der jeweiligen Strategie im Vergleich zu Genetischen Algorithmen genannt wurden.

3. Lösungsstrategie

Im Folgenden wird zunächst das Optimierungsproblem an sich aus mathematischer Sicht beschrieben und basierend auf diesen Erkenntnissen wird die Optimierungsstrategie besprochen, die einen Produktentwicklungsprozess gemäß eines vordefinierten Ziels umordnet um Produktionskosten und Produktionszeit zu reduzieren.

3.1 Problemkomplexität

Erstaunlicherweise wurde bis dato in der DSM Literatur die Komplexität einer Sequenzoptimierung von DSM Aktivitäten nicht erörtert. Diese zu bestimmen ist allerdings sinnvoll um das Problem einer bestimmten Komplexitätsklasse zuzuordnen, in der sich auch andere bekannte Probleme befinden und über die es eventuell umfangreiche Literatur gibt. In der Tat konnte relativ schnell die Verwandtschaft des Sequenzierungsproblems in DSMs mit dem Quadratischen Zuordnungsproblem (QAP) [10] aus dem Bereich des Operations Research aufgezeigt werden. Das QAP beschreibt ursprünglich das Problem eine bestmögliche Anordnung für die Platzierung von n Produktionseinrichtungen auf n Standorte zu finden, um einen Warentransport zwischen ihnen möglichst günstig zu halten. Formal ist das QAP wie folgt formuliert:

$$\min_{\psi \in S(n)} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \cdot b_{\psi(i)\psi(j)}$$

mit einer Matrix für den Materialfluss $A = \{a_{ij}\}$ zwischen den Produktionsstandorten i und j , einer Distanzmatrix $B = \{b_{ru}\}$ mit den Entfernungen zwischen i und j , mit $S(n)$ als Menge aller möglichen Permutation (Anordnungen der Fabriken auf die Standorte) und $\psi(i)$ bezeichnet die Position des Produktionsstandortes i in der aktuellen Permutation $\psi \in S(n)$. Die Anzahl aller möglichen Permutationen beträgt $n!$ und damit gehört das QAP zu den schwierigsten zu lösenden Kombinatorikproblemen.

Angewandt auf das Problem eine optimale DSM Permutation für einen Produktentwicklungsprozess zu finden, können in diesem Kontext Produktionseinrichtungen und Produktionsstandorte mit Prozessaktivitäten und deren Platzierung in der DSM Sequenz gleichgesetzt werden. Nun stellt sich lediglich die Frage wie die Flussmatrix A und die Distanzmatrix B auszusehen haben da. Als Flussmatrix kann die ursprüngliche DSM benutzt werden da diese letztendlich den Informationsfluss innerhalb des Prozesses repräsentiert und durch numerische Einträge in den Matrixzellen die „Stärke“ des Flusses zwischen Aktivitäten angibt. Die Darstellung der Distanzmatrix ist dagegen abhängig vom jeweiligen Optimierungsziel zu wählen. Falls z.B. die Anzahl der Einträge oberhalb der DSM Diagonalen reduziert werden soll müsste man, unter Berücksichtigung der Minimierung der Doppelsumme aus Fluss und Distanz in obiger QAP Definition, alle Matrixzellen unterhalb der Diagonalen mit einer 0 belegen und Einträge oberhalb der Diagonalen mit einheitlichen Werten größer als 0. Solch eine DSM Zerlegung in Fluss- und Distanzmatrix mit der Minimierung der Distanzen oberhalb der Diagonalen als Optimierungsziel ist in Abbildung 2 dargestellt.

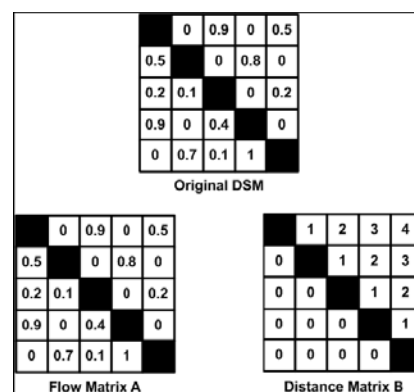


Abb. 2: Zerlegung einer DSM in Fluss- und Distanzmatrix.

3.2 Voroptimierung

Für gewisse Optimierungsziele ist es möglich eine DSM in mehrere Sub-DSMs zu zerlegen um so die Gesamtanzahl möglicher DSM Permutationen drastisch zu reduzieren. *Partitioning* bezeichnet den Vorgang auf deterministische Weise einen oder mehrere Blöcke innerhalb einer DSM zu finden, in denen ausschließlich jede Aktivität mit jeder anderen in Relation steht. Insbesondere bedeutet dies, dass in diesen Blöcken Kreisschlüsse, d.h. Rücksprünge im Prozess, auftreten. Außerhalb dieser Blöcke erscheinen in der DSM lediglich Relationen zwischen Aktivitäten unterhalb der Diagonalen, d.h. diese Aktivitäten haben eine sequentielle oder parallele Abhängigkeit untereinander. Es sei angemerkt, dass durch Partitioning keine Sequenzierung innerhalb der Blöcke hinsichtlich eines Kriteriums vorgenommen wird! Partitioning teilt die Menge aller Aktivitäten nur in zwei disjunkte Mengen: zum Einen in Aktivitäten, die entweder sequentielle oder parallele Relationen aufweisen und zum Anderen in eine Menge von Aktivitäten mit gekoppelten Relationen.

Zahlreiche Partitioning Algorithmen existieren in der Literatur [7, 12, 18]. Diese Ansätze basieren darauf explizit alle Kreisschlüsse in einer DSM zu identifizieren. Da die bekannten Algorithmen zur Kreisidentifizierung aufgrund der hohen Zeitkomplexität von mindestens $O(n^3)$ unpraktisch für große Produktentwicklungsprozesse mit mehreren hundert oder tausend Aktivitäten sind, wurden stattdessen so genannte streng zusammenhängende Komponenten (Strongly Connected Components = SCCs) in der DSM mittels einer modifizierten Tiefensuche [14] gesucht. Per Definition besitzen alle Komponenten innerhalb einer SCC eine Abhängigkeit zu jeder anderen Komponente in derselben SCC. Im Vergleich zur Identifizierung von Kreisschlüssen werden bei der Suche nach SCCs keine in sich verschachtelten Kreisschlüsse identifiziert. Dies ist aber auch nicht nötig, da weiterhin die Menge aller DSM Aktivitäten in die oben genannten zwei disjunkten Mengen aufgeteilt wird. Falls eine DSM mit n Aktivitäten in m verschiedene SCCs zerlegt werden kann, wird die Anzahl möglicher Permutationen von $n!$ auf folgende Anzahl reduziert:

$$\sum_{i=1}^m |SCC_i|!$$

Unabhängig von anderen SCCs kann für jede einzelne SCC eine optimale Sequenz hinsichtlich eines Optimierungskriteriums gefunden werden, d.h. es können in paralleler Ausführung mehrere SCCs optimiert werden.

3.3 Monokriterielle Optimierung von Design Structure Matrices durch Genetische Algorithmen

Mit der Absicht Zeit und Kosten für Produktentwicklungsprozesse zu reduzieren, gibt es in der DSM-Literatur zahlreiche Optimierungsziele für DSMs, die dies bewerkstelligen sollen. Zum Beispiel nimmt man an, dass eine Reduzierung von superdiagonalen Einträgen zu weniger Iterationen führen und damit auch zu niedrigeren Kosten, Dauer und Risiko. Mit Hilfe eines Partitioning Algorithmus ist man oftmals dazu in der Lage die DSM in kleiner Teilprobleme – in SCCs – zu zerlegen bzw. manchmal sogar komplett so umzuordnen, dass das Optimierungskriterium erfüllt ist. Eine Studie von Yassine et al. [22] zeigt allerdings, dass es in komplexen Produktentwicklungsprozessen sehr unwahrscheinlich ist durch alleinige Anwendung eines Partitioning Algorithmus eine, den individuellen Bedürfnissen angepasste, DSM Permutation zu finden. Im schlimmsten Fall wird eine DSM in lediglich eine einzige SCC zerlegt, d.h.

Partitioning war sinnlos. In solch einem Fall muss über eine Optimierung innerhalb der SCCs nachgedacht werden.

Die einfachste Methode hierfür wäre eine erschöpfende Suche, die alle möglichen Permutationen erzeugt und jeder Permutation einen Zielwert oder Fitnesswert gemäß der vordefinierten Zielfunktion zuordnet. Doch aufgrund des faktoriellen Zuwachs an möglichen Permutationen in Abhängigkeit der SCC Größe ist diese Strategie nur sinnvoll für kleine DSMs bzw. kleine SCCs mit ungefähr 10-12 Aktivitäten. Für große SCCs mit 100 oder mehr Aktivitäten müssen andere Suchstrategien angewandt werden.

Ein erfolgsversprechender Ansatz hierfür sind Genetische Algorithmen (GAs), eingeführt von John Holland [8]. Dieser meta-heuristische Ansatz ist äußerst robust, d.h. gute Lösungen können mit den richtigen GA-Parametern selbst für große und komplexe Suchräume erwartet werden. Genetische Algorithmen sind dem natürlichen Evolutionsprozess nachempfunden da sich dieser im Laufe der Geschichte als ein äußerst effektives Optimierungs-verfahren erwiesen hat. In der vorliegenden Diplomarbeit wurden zwei verschiedene GA Designs für eine monokriterielle Optimierung – d.h. die DSM wird nur hinsichtlich eines Kriteriums optimiert – von DSMs ausführlich beschrieben und getestet: simple GAs und so genannte kompetente GAs.

Im Allgemeinen kann jeder GA in seine Komponenten und Operatoren unterteilt werden. Zu den Komponenten gehören die Datenstruktur und die Fitnessfunktion des GA. Dabei gibt die Datenstruktur hauptsächlich an inwiefern das zugrunde liegende Optimierungsproblem kodiert wird. Eine kodierte Lösung des Problems, d.h. in unserem Fall eine mögliche DSM Permutation, wird dabei in der GA – Terminologie als *Chromosom* bezeichnet. Da GAs ihre Suche von mehreren Punkten im Suchraum starten, werden bei der Initialisierung des GAs mehrere, zufällig gewählte, Lösungen kodiert. Diese Menge an Lösungen wird als *Population* bezeichnet. Um im Verlauf des GAs immer bessere Chromosome, und damit immer bessere DSM-Permutationen, zu erhalten, muss man in der Lage sein den Chromosomen eine Güte der Qualität zuzuordnen. Hierfür ist die Fitnessfunktion zuständig, die jedem Chromosom einen so genannten Fitnesswert gibt. Zum Beispiel könnte eine Fitnessfunktion die Anzahl der Einträge oberhalb der Diagonalen zählen und den Kehrwert dieses Wertes dem jeweiligen Chromosom zuteilen.

Basierend auf dem Evolutionsprozess in der Natur benutzen Genetische Algorithmen als Operatoren die Selektion, Rekombination und Mutation. Während der Selektion werden je nach Selektionsstrategie die Chromosome innerhalb der Population ausgewählt, die das Potenzial haben zu besseren neuen Chromosomen verändert werden zu können, nachdem die beiden anderen Operatoren auf sie angewandt wurden. Hier ist es insbesondere wichtig zu erwähnen, dass keinesfalls die n absolut besten Chromosome aus einer Population ausgewählt werden. Vielmehr muss ein vernünftiger Kompromiss zwischen der Aufrechterhaltung der Diversität in der GA Population und der Qualität der Fitness gefunden werden. Im Anschluss an die Selektion untergehen die Chromosome die Rekombinationsphase. In diesem Schritt werden die zuvor selektierten Chromosome miteinander „gekreuzt“, in der Hoffnung, dass diese neuen Chromosome die guten Eigenschaften ihrer Eltern erben, diese dann miteinander vereinigen und so letztendlich noch bessere Chromosome erzeugt werden. Der Rekombinationsoperator ist derjenige, der für die tendenziell global ausgerichtete Suche von GAs verantwortlich ist. Um nicht nur global den Suchraum zu explorieren, sondern auch interessante Regionen intensiv auszuwerten, besitzen GAs oftmals einen Mutationsoperator (es gibt auch GAs, die darauf verzichten). Hier werden, ähnlich der biologischen Mutation, lediglich kleine Bruchstücke des

Chromosoms manipuliert. Nach Abarbeitung aller Operatoren wird für jedes Chromosom der neuen Population der Fitnesswert bestimmt und eine neue Generation im GA-Zyklus beginnt falls das gewählte Konvergenzkriterium (z.B. Anzahl der Generationen) noch nicht erfüllt wurde. Eine bildliche Darstellung dieses simplen GA Ablaufs liefert Abbildung 3.

Dieses traditionelle GA Design kann durch verschiedene Maßnahmen deutlich verbessert werden. Insbesondere erhöht die explizite Einbindung einer lokalen Suchmethode in den GA-Ablauf dessen Leistung dramatisch. Der Grund hierfür ist die eher global ausgerichtete Suche in GAs und die relativ niedrige Wahrscheinlichkeit für das Auftreten einer Mutation. Eine *Hybridisierung* des simplen GAs mit einer lokalen Suche kombiniert stattdessen die Vorteile beider Suchen und führt zu besseren Ergebnissen als eine separate lokale bzw. globale Suche! Daher wurde eine lokale Suchstrategie speziell für DSMs entwickelt, die sich in Tests als äußerst hilfreich erwiesen hat.

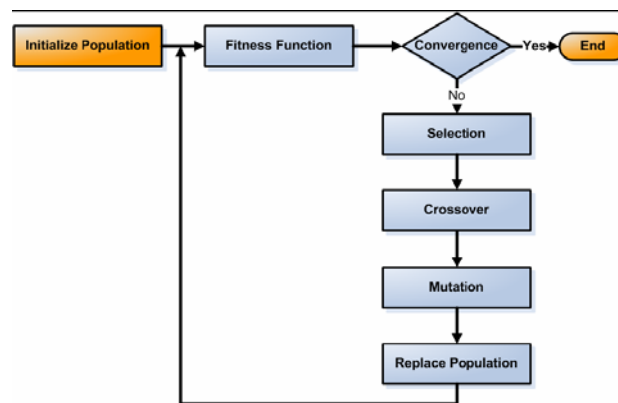


Abb. 3: Flussdiagramm eines simplen Genetischen Algorithmus.

In Tests für den simplen GA mit künstlich konstruierten Kombinatorikproblemen und DSMs fielen insbesondere zwei Aspekte auf. Zum Einen benötigt ein simples GA Design (das Einbinden einer lokalen Suche ändert nichts am Design des GA) eine stark exponentiell ansteigende Anzahl von Aufrufen der Fitnessfunktion mit steigender Schwere des Problems. Zum Anderen hängt die Leistung des GA wesentlich von den Charakteristika der DSM und des Optimierungsziels ab. Um ein gleichmäßiges Leistungsverhalten des GA, unabhängig von der Komplexität des Problems bzw. den Eigenschaften einer DSM, zu erhalten bei gleichzeitiger Vermeidung eines übermäßig starken exponentiellen Wachstums der Fitnessfunktionsaufrufe, wurde die neueste Klasse von GAs, die so genannten kompetenten GAs, für eine Nutzung des DSM Sequenzierungsproblems untersucht. Diese GAs identifizieren explizit die *Building Blocks* (BBs) des Problems. Informell sind Building Blocks Teilsequenzen eines Chromosoms, die eine sehr gute Lösung (im besten Fall das globale Optimum) unbedingt beinhalten sollte. Falls nun BBs entdeckt werden, können diese untereinander kombiniert werden um so, im Stile eines Legobaukastens, das globale Optimum zu formen. Für Kombinatorikprobleme gibt es momentan lediglich einen kompetenten GA, nämlich den Ordering Messy GA (OmeGA) [9]. Selbst wenn die Building Block Struktur (Größe und Anzahl der BBs) eines DSM Sequenzierungsproblems nicht bekannt ist, und davon ist auszugehen, können mit diesem Ansatz sehr gute Ergebnisse erzielt werden. Vor allem durch eine Erweiterung des ursprünglichen OmeGA Designs mit der eigens für DSMs entwickelten lokalen Suche ist die Leistung dieses GA im Vergleich zum simplen GA Design für schwierige Probleme erheblich besser. Allerdings sind für leichte

Probleme simple GAs im Vorteil weil sich in diesem Fall die Identifizierung von BBs, die Aufrufe der Fitnessfunktion „kostet“, einfach nicht rentiert. Da es im Allgemeinen aufgrund der Größe des Suchraumes eher unwahrscheinlich ist eine Aussage über dessen Komplexität zu treffen, hat der erweiterte OmeGA den signifikanten Vorteil, dass man sich über die Schwere des Problems weniger Gedanken machen braucht und im Vergleich zu simplen GAs „auf der sicheren Seite“ ist.

Aufgrund der Verwandtschaft des DSM Sequenzierungsproblems mit dem QAP wurden auch Tests mit einem erweiterten simplen GA Design und dem erweiterten OmeGA für bekannte QAP Instanzen durchgeführt. Die Ergebnisse bestätigten noch einmal die allgemeine Überlegenheit des erweiterten OmeGA gegenüber seinem simplen Pendant. Des Weiteren waren die Resultate identisch oder sehr nahe an den bisher besten Ergebnissen für die jeweiligen QAP Instanzen, die teilweise mit unterschiedlichen Optimierungsstrategien (z.B. Simulated Annealing, Tabu Suche oder Ant Colony Optimization) gefunden wurden – ein Zeichen für die Robustheit dieses GA!

Als ein Beispiel aus der Praxis wurde ein Produktentwicklungsprozess bei Ford für das Design einer Motorhaube mit Hilfe eines GA umgeordnet. Ein Optimierungsziel war unter anderem die Reduzierung der Abstände von superdiagonalen Einträgen zur DSM Diagonalen. Dadurch wird zum Einen die Gesamtanzahl der Rücksprünge reduziert als auch die „Länge“ der Rücksprünge, d.h. Iterationen im Prozess werden schneller und zusätzlich schrumpft das Risiko vernetzeter Iterationen. Die linke DSM in Abbildung 4 zeigt die originale DSM von Ford mit gelb hervorgehobenen superdiagonalen Einträgen. Die rechte DSM ist optimiert und die zwei schwarz umrahmten Blöcke stellen die SCCs dar, die während des Partitioning entdeckt wurden. Durch Anwendung von Partitioning und eines GA konnte die Summe aller superdiagonalen Distanzen zur Diagonalen von ursprünglich 301 auf 93 verbessert werden!

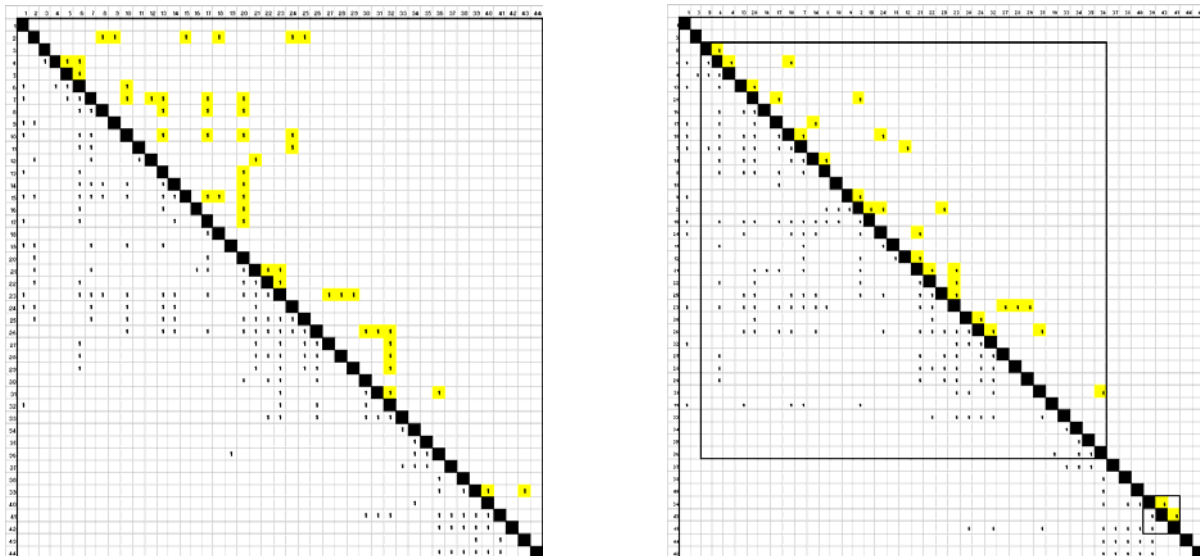


Abb. 4: Originale DSM von Ford (links) und optimierte DSM (rechts).

3.4 Multikriterielle Optimierung von Design Structure Matrices durch Genetische Algorithmen

Die oben beschriebene monokriterielle Optimierung von DSMs benutzte Fitnessfunktionen, die die Güte eines Prozesses auf deterministische Art und Weise bestimmten. Es wurde die Annahme getroffen, dass beispielsweise Prozesse mit vielen Rücksprüngen automatisch eine schlechte Qualität haben und im Gegensatz Prozesse mit wenigen Iterationen gute Prozesse sind. Tests, die als Fitnessfunktion eine Simulation über DSMs [1] benutzte, konnten allerdings aufzeigen, dass solche eine Annahme nicht haltbar ist.

Vielmehr können Prozessstrukturen, die wenig Kosten verursachen, und solche, die zu schnellen Durchführungszeiten führen, weit auseinander liegen. In Tests hatten die entsprechenden DSMs für kostengünstige Prozesse als Charakteristikum wenige Einträge oberhalb der Diagonalen und dementsprechend auch wenige Rücksprünge. Dagegen wiesen schnelle Prozesse oftmals eine hohe Anzahl von Rücksprüngen und superdiagonalen Einträgen auf. Der Grund hierfür ist die Tatsache, dass es oftmals schneller ist bestimmte Aktivitäten ohne komplette Inputinformationen einfach zu starten da diese dann parallel zu anderen Aktivitäten ausgeführt werden als sich an eine streng sequentielle Abfolge zu halten. Im Gegenzug können die Kosten für solche Prozesse exorbitant hoch sein da im Zuge falscher Annahmen über Inputinformationen ungeplante Iterationen entstehen.

Anscheinend muss entweder zwischen einem schnellen, aber teuren, Prozess und einem billigen, aber langsamen, gewählt werden. Da solche Extremfälle aufgrund von Randbedingungen (z.B. ein beschränktes Budget oder wichtige Projektfristen) nicht unbedingt erwünscht sind, wäre es für Entscheidungsträger von Vorteil eine Menge von besten Kompromisslösungen zu kennen, d.h. eine Liste mit Prozessen, die für ein vorgegebenes Budget nicht schneller sind bzw. für ein vorgegebenes Zeitlimit am billigsten sind.

Diese Menge bester Kompromisslösungen wird Pareto-Front genannt und diese für Prozesse, die mit DSMs modelliert sind, zu approximieren war ebenfalls ein Ziel der Diplomarbeit. Für diese Aufgabe bieten sich Genetische Algorithmen besonders an, da diese die Suche mit einer Population von Lösungen starten um diese dann komplett gegen die Pareto-Front zu bewegen anstatt nacheinander in mehreren Berechnungen einzelne Punkte auf der Pareto-Front zu identifizieren. Als Fitnessfunktion für alle Tests wurde die in [1] beschriebene Simulation verwendet. Das Hauptproblem bei der Verwendung von GAs für eine Mehrzieloptimierung sind die folgenden zwei Punkte:

- Erhalt der Diversität in der GA-Population um die Pareto-Front möglichst großflächig zu approximieren.
- Die geeignete Definition eines Konvergenzkriteriums um unnötige Laufzeit für den GA zu vermeiden.

Da bisher in der Literatur von keinem GA berichtet wurde, der beide Kriterien zuverlässig erfüllt, wurde zuerst der so genannte Nondominated Sorting Genetic Algorithm II (NSGA-II) [3] implementiert. Dieser GA zur Mehrzielfunktionsoptimierung schafft es im Verlaufe der Zeit seine Population stets divers zu halten und so die gesamte Pareto-Front gut abzudecken. Zur Feststellung der Konvergenz wurde NSGA-II um ein eigens definiertes Konvergenzkriterium erweitert, das auf Histogrammen über die durchschnittlichen Werte von Kosten und Dauer der einzelnen Lösungen in der Pareto-Region basiert.

Für zwei Produktentwicklungsprozesse aus der Industrie konnte mit Hilfe dieses GAs die Pareto-Front für Kosten und Dauer gut angenähert werden. Dadurch sind z.B. Projektmanager in der Lage einen bestmöglichen Prozess für Ihre individuellen Anforderungen auszuwählen. Der einzige Nachteil der beschriebenen GA Implementierung ist momentan noch die nicht-deterministische Simulation für Kosten und Dauer eines Prozesses als Fitnessfunktion. Diese Tatsache erschwert zum Einen das Auffinden von qualitativ guten Pareto-Lösungen und zum Anderen verlangsamt es aufgrund ihres Zeitaufwands doch erheblich die Geschwindigkeit des GA. Daher sollten sich zukünftige Studien in diesem Bereich vor allem auf eine geschlossene und deterministische Gleichung für die Schätzung von Dauer und Kosten von Prozessen, die mit DSMs modelliert wurden, konzentrieren.

Für eine multikriterielle Optimierung wurde unter anderem auch der in Abbildung 4 dargestellte Produktentwicklungsprozess bei Ford für das Design einer Motorhaube getestet. Abbildung 5 zeigt die ursprüngliche GA Population am Anfang mit blauen Punkten und die zur Pareto-Front konvergierte GA Population mit roten Punkten. Man kann sehr schön sehen wie der GA die komplette Population gegen eine Front drückt, die nicht überschritten werden kann. In Abbildung 6 ist der sinkende Verlauf der durchschnittlichen Kosten und Dauer aller DSM – Permutationen in der Pareto-Front dargestellt.

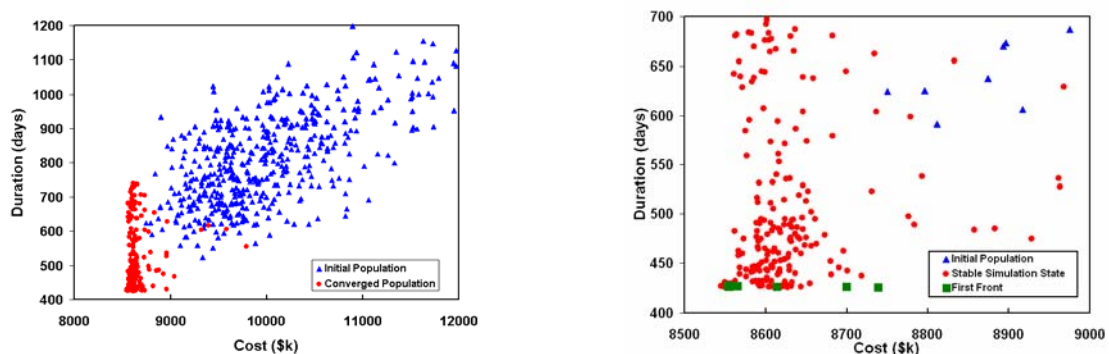


Abb. 5: Vergleich von ursprünglicher GA Population und konvergierter GA Population im kompletten Suchraum,(links) und in der Pareto-Region (rechts).

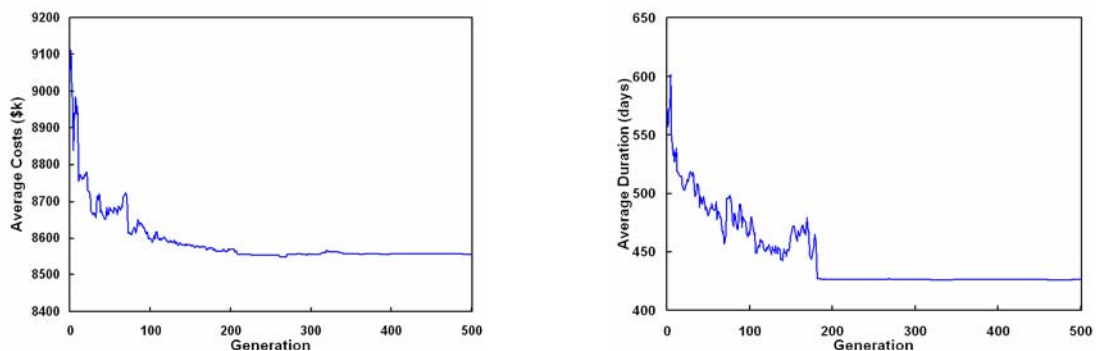


Abb. 6: Verlauf der durchschnittlichen Kosten und Dauer in der Pareto-Front

5. Zusammenfassung und Ausblick

In meiner Diplomarbeit habe ich Genetische Algorithmen benutzt um komplexe Produktentwicklungsprozessen hinsichtlich beliebiger Ziele zu verbessern. Solch eine Prozessverbesserung kann zu weniger Risiko im gesamten Prozessablauf führen, ebenso wie zu geringeren Produktionskosten und Produktionszeit für die jeweilige Firma. Dabei wurde als Modell für Produktentwicklungsprozesse die DSM Methode verwendet um das gesamte Spektrum an möglichen Abhängigkeiten zwischen einzelnen Prozessaktivitäten zu modellieren. Angewandt auf dieses Modell wurden die neuesten Erkenntnisse über Genetische Algorithmen. Diese wurden im Falle einer monokriteriellen Suche außerdem erweitert um unabhängig von der Schwere des Suchproblems eine robuste Suchstrategie zu besitzen. Eine multikriterielle Prozessoptimierung eröffnet zudem Entscheidungsträgern die Möglichkeit einen besten Prozess für gewisse Randbedingungen auszusuchen.

Zukünftige Arbeit im Bereich der monokriteriellen Optimierung von DSMs sollte sich besonders um eine Analyse des Schwierigkeitsgrades für eine Optimierung von DSMs konzentrieren. Dann wäre es z.B. möglich die individuell beste Optimierungsstrategie anzuwenden, z.B. für einfach zu lösende Probleme einen simplen GA einzusetzen und für schwere einen kompetenten GA. Des Weiteren sollten weitere kompetente GAs für Kombinatorikprobleme entwickelt werden, da Kombinatorikprobleme meist zu den wirtschaftlich interessanten gehören. Zudem könnte ein detaillierter Vergleich von meta-heuristischen Optimierungsverfahren mit GAs hilfreich sein, um eine Funktion in der Art $f(\text{Art der DSM}) \rightarrow \text{Optimierungsstrategie}$ zu finden. In Bezug auf eine multikriterielle Optimierung von simulierten DSMs muss die höchste Priorität darin bestehen, eine geschlossene, deterministische Gleichung für die Abschätzung von Kosten und Dauer eines Prozesses zu finden. Außerdem wäre auch hier ein Vergleich von GAs insbesondere mit Ant Colony Optimization (ACO) sinnvoll, da in letzter Zeit einige erfolgsversprechende multikriterielle ACO Ansätze entstanden (z.B. [4]).

6. Literatur

- [1] T.R. Browning and S.D. Eppinger. Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Transactions on Engineering Management*, 2002.
- [2] K. Clark and T. Fujimoto. Product Development Performance. Boston, MA: Harvard Business School Press, 1991.
- [3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions On Evolutionary Computation*, 2002.
- [4] K. Doerner, W.J. Gutjahr, R.F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 2004.
- [5] S.D. Eppinger. Innovation at the speed of information. *Harvard Business Review*, 2001.
- [6] J.L. Funk. Concurrent engineering and the underlying structure of the design problem. *IEEE*

Transactions on Engineering Management, 1997.

[7] D.A. Gebala and S.D. Eppinger. Methods for analyzing design procedures. ASME 3rd Int. Conf. On Design Theory and Methodology, 1991.

[8] J.H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor, MI: The University of Michigan Press, 1975.

[9] D. Knjazew. *OmeGA: A Competent Genetic Algorithm for Solving Permutation and Scheduling Problems*. Norwell, MA: Kluwer Academic Publishers Group, 2002.

[10] T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 1957.

[11] S.M. Osborne. *Product Development Cycle Time Characterization Through Modeling of Process Iteration*. Master's Thesis, MIT, Cambridge, MA., 1993.

[12] D.V. Steward. *Systems Analysis and Management: Structure, Strategy and Design*. New York, NY: Petrocelli Books, 1981.

[13] N. Suh, *The Principles of Design*. Oxford University Press, New York, 1990.

[14] R. Tarjan, "Depth-First Search and Linear Graph Algorithms", *SIAM Journal of Computing*, vol. 1, no. 2, pp. 146-160, 1972.

[15] D. Ulman, *The Mechanical Design Process*. McGraw-Hill, 2003, 3rd edition.

[16] K. Ulrich and S.D. Eppinger, *Product Design and Development*. McGraw-Hill, 2004, 3rd edition.

[17] Z. Wang and H. Yan. Optimizing the concurrency for a group of design activities. *IEEE Transactions on Engineering Management*, 2005.

[18] J. N. Warfield, "Binary Matrices in System Modeling" *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, pp. 441-449, 1973.

[19] A. Yassine, and D. Braha, D., "Four Complex Problems in Concurrent Engineering and the Design Structure Matrix Method," *Concurrent Engineering Research & Applications*, 11(3), 2003.

[20] A. Yassine, N. Joglekar, S.D. Eppinger and D. Whitney, "Performance of Coupled Product Development Activities with a Deadline," *Management Science*, Vol. 47 (12), Dec. 2001.

[21] A. Yassine, N. Joglekar, S.D. Eppinger and D. Whitney, "Information Hiding in Product Development: The Design Churn Effect," *Research in Engineering Design*, 14(3), 2003.

[22] A. Yassine, D. Whitney, J. Lavine, and T. Zambito. Do-it-right-first-time (DRFT) approach to design structure matrix restructuring. Proceedings of the 12th International Conference on Design Theory and Methodology, 2000.