

Gerichtete Modellprüfung mit Explorationsverfahren der Künstlichen Intelligenz

Stefan Edelkamp und Alberto Lluch Lafuente

Institut für Informatik
Georges-Köhler-Allee, Geb 51
Albert-Ludwigs-Universität, 79110 Freiburg
eMail: {edelkamp,lafuente}@informatik.uni-freiburg.de

Zusammenfassung Gegenstand der *Gerichteten Modellprüfung* ist die Entwicklung und Anwendung von gerichteten Suchverfahren zur expliziten und symbolischen Exploration von Zustandsräumen in Modellprüfungsproblemen, die sowohl bei der Verifikation sicherheitskritischer nebenläufiger Systeme als auch bei Handlungsplanungsfragestellungen der Künstlichen Intelligenz entstehen.

In diesem Papier reflektieren wir den Stand der Forschung und eigene Arbeiten in diesem zukunftssträchtigen Gebiet.

1 Einleitung

Modellprüfung (Model Checking) [11] ist eine formale Methode zur automatischen Verifikation und Validation von verteilten Soft- und Hardwaresystemen. Das zu prüfende System wie auch die relevanten Aspekte des Verhaltens müssen formal modellierbar sein. Damit stellt das Verfahren eine Alternative oder mögliche Ergänzung zum Einsatz von Theorembeweisern sowie von Simulations- und Testverfahren dar. Findet die Modellprüfung keine Fehler, so ist die Korrektheit des Systems in Bezug auf seine Spezifikation nachgewiesen. Wenn ein Teil einer Spezifikation nicht erfüllt ist, dann liefert das Verfahren einen Zeugen (ein Gegenbeispiel), bei dem das Versagen des Systems zutage tritt. Test- und Simulationsverfahren garantieren dagegen lediglich die Korrektheit in den getesteten bzw. simulierten, stichprobenartig ausgewählten Einzelfällen und bei der Anwendung eines Theorembeweisers muß die Verifikation manuell gesteuert werden.

Als Eingabe der Modellprüfung dient auf der einen Seite ein üblicherweise endliches Modell des zu prüfenden Systems und auf der anderen Seite die Spezifikation der geforderten Eigenschaft. Sicherheits- und Korrektheitseigenschaften werden dabei als Sätze einer geeigneten Zeitlogik formuliert. Der Model Checker prüft nun automatisch, ob diese Anforderungen im Modell gelten.

Die Modellprüfung exploriert den Raum aller erreichbaren Zustände. In einem verteilten System [12] ergibt sich der Gesamtzustandsraum dabei aus dem (asynchronen) Produkt der lokalen Systemzustände und aus der globalen Information (z.B. globalen Variablen, Kommunikationsschlangen). Für die Mo-

dellprüfung wird zusätzlich noch ein (synchrones) Produkt aus dem Gesamtzustandsraum und der Repräsentation der geforderten Eigenschaft gebildet.

In den Techniken zur Bewältigung der enormen Zustandsraumgröße, die insbesondere durch die Festlegung konkurrierender Systeme hervorgerufen wird, ist die gerichtete Exploration ein neuer, vielversprechender Ansatz zur Fehlersuche, der sowohl die explizite als auch die symbolische Repräsentation des Zustandsraumes umfaßt. Dabei werden die Zustandsräume als implizite Graphen interpretiert, die durch eine Nachfolgererzeugungsfunktion aufgespannt werden und in denen ein möglichst kurzer Pfad als Zeuge eines Fehlers gesucht wird.

Der in diesem Papier beschriebene Ansatz der *Gerichteten Modellprüfung* umfaßt demnach die Anwendung algorithmischer Methoden der Künstlichen Intelligenz, ein Gebiet, in dem seit Jahrzehnten die gerichtete Suche in großen Zustandsräumen sehr erfolgreich eingesetzt wird. Da Problembeschreibungen in der Modellprüfung und Handlungsplanung trotz ihrer Vielfalt eine sehr starke Nähe zueinander aufweisen, wird das Spektrum der verschiedenen Lösungsmethoden in diesen Bereichen gemeinsam studiert.

2 Modellprüfungsverfahren

In diesem Abschnitt werden unterschiedliche Methoden zur Bewältigung der enormen Zustandsgröße in der Modellprüfung vorgestellt.

Symbolische Verfahren: Die symbolische Modellprüfung [47] setzt Binäre Entscheidungsdiagramme, sogenannte BDDs [7], zur Zustandsraumkomprimierung ein. Hierbei werden die Zustände binär codiert und Zustandsmengen über die Disjunktion ihrer charakteristischen Funktionen platzsparend verwaltet. Über die binäre Darstellung aller Übergänge kann die Erreichbarkeit der Zustände über eine symbolische Breitensuche getestet werden. Die vollständige Speicherung des Zustandsraumes selbst für große Zustandsmengen gelingt jedoch auch mit geschichteten Automaten [34]. Letztere können als Entscheidungsdiagramme angesehen werden, in denen nur eine der zwei Reduktionsregeln für BDDs angewandt wird: Isomorphe Knoten werden verschmolzen, aber triviale Knoten, deren 0- und 1- Kanten auf denselben Nachfolger zeigen, werden nicht eliminiert. Dies ermöglicht die Einfügung eines Zustandes in Linearzeit zu seiner binären Codierungslänge. Dennoch ist eine Speicherung der Zustände in einer Hashtabelle, soweit möglich, effizienter zu implementieren.

Zustandskomprimierung: Zur Komprimierung von Zuständen bei der Speicherung wird häufig das Bit-State Hashing [33] eingesetzt. Im Bit-State Hashing wird analog zu den Erweiterungen des Double-Bit-State Hashings und der Bit-Compact-Methode der Zustand auf wenige Bits vereinfacht. Bei einer fest vorgegebenen Speichergrenze kann somit eine wesentlich größere Hashtabelle adressiert werden. Dies ermöglicht die Duplikatserkennung auch in dem Fall, daß die Hauptspeicherressourcen für eine vollständige Speicherung aufgebraucht sind. Allerdings können Kollisionen komprimierter Synonyme nicht mehr eindeutig aufgelöst werden, so daß implizit eine Beschneidung des Suchraumes stattfindet.

Partialordnungsmethoden: Partialordnungsmethoden [56] dienen der Vereinfachung des Suchraumes. Sie erzwingen eine feste Reihenfolge von Übergangsfolgen und senken somit den Verzweigungsgrad in der Zustandsraumexploration. Die Relevanz von Partialordnungsmethoden ist eng mit dem gewählten Suchverfahren und der Ausdrucksmächtigkeit der zu überprüfenden Formel verknüpft.

Unendliche Zustandsräume: Die Überprüfung unendlicher Zustandsräume ist so schwer, daß man die Endlichkeit des unterliegenden Modells im allgemeinen voraussetzt. Wir machen allerdings auf die effiziente Darstellung von unbeschränkt langen Wörtern mit regulären Automaten aufmerksam, wie sie in dem Modellprüfer Mona des Forschungsinstituts BRICS/Aarhus gewählt wurde. Hierbei wird die Spezifikation des Systems und der Anforderung in Monadischer Logik zweiter Ordnung gewählt und ein minimaler endlicher Automat durch induktive Konstruktion der geforderten Äquivalenzbeziehung erstellt. Zur Minimierung muß stets ein nicht-deterministischer Automat zuerst per Potenzmengekonstruktion in einen deterministischen umgewandelt werden. Ist die Äquivalenz gegeben, so ist der konstruierte Automat trivial, ansonsten liefert jedes von dem Automaten repräsentierte Wort ein Gegenbeispiel. Dabei werden die Automaten durch eine spezielle BDD-Struktur dargestellt: Die Senken der Einzel-BDDs weisen auf den nächsten Zustand. Queue-BDDs [27], und QBDDs [5] sind Alternativen zu dieser Darstellung, die in der Protokollvalidation eingesetzt werden: Sie fügen endliche Automaten in die Schichten der BDD-Struktur ein.

Kompositionale Methoden: Als weiteres aktuelles Forschungsfeld gelten kompositionale Ansätze [39], die die Modularisierung des Modells ausnutzen und die insbesondere in der symbolischen Suche eingesetzt werden. Ein Ansatz ist es, die Menge der Nachfolger mit Hilfe der Transitionsfunktion durch eine entsprechende Partitionierung des Suchraumes gruppenweise nacheinander zu berechnen; ein anderer die BDD-Struktur bei diesem Vorgang zu optimieren. Als sehr befriedigend erwiesen sich die bisher von uns eingesetzten Methoden jedoch nicht.

Abstraktion: Abstraktion ist eine sehr aktuelle Lösungsmethode in der Modellprüfung. Dabei wird eine Reduktion des Zustandsraumes durch Vereinfachung der Problemstellung erreicht. Falls in dem abstrakten Modell eine Formel gilt, so auch im konkreten. Man unterscheidet das Prinzip abstrakter Interpretationen [45] und die Erzeugung abstrakter Modelle [28]. Abstraktion entspricht dem Grundgedanken der gerichteten Suche. Durch die Lösung vereinfachter, relaxierter Problemstellungen gewinnt man Information zur Lösung des übergeordneten Problems. Dennoch unterscheiden sich die bestehenden Ansätze der Abstraktion von der gerichteten Suche in der Modellprüfung: Während in dem ersten Fall die Abstraktion definitive Aussagen für die Korrektheit liefern soll, wirkt die vorgeschlagene Fehlerheuristik genau entgegengesetzt; sie ist richtungsweisend bei der Entdeckung eines Fehlers.

Gerichtete Modellprüfung: Die Hauptschwäche von allen vorgestellten, mitunter sehr ausgefeilten Suchmethoden, ist ihre Uninformiertheit; die Traversierung des Suchraumes verläuft sozusagen *blind*. Demgegenüber steht die starke Reduktion in der Exploration von Suchräumen mit Hilfe der informierten bzw. gerichteten Suche. Die Praxis zeigt auf, daß selbst ein vergleichsweise großer

(doch polynomieller) Aufwand zur Berechnung unterer Schranken sich positiv sowohl auf die Speicherressourcen als auch auf die Explorationszeit auswirkt. In der gerichteten Modellprüfung betrachten wir die Suche in Richtung der Fehlerzustände, die Zielknoten im Zustandsgraphen entsprechen. Durch die Fokussierung des Suchvorganges können so im Entwicklungsprozeß Fehler in nebenläufigen Systemen (Deadlocks, verletzte Zusicherungen, Invarianzen und allgemeine temporale Eigenschaften) mit oder ohne Unterstützung des System-Designers schnell gefunden werden. Erfreulicherweise lassen sich relativ einfach automatische und halb-automatische Heuristiken finden, die eine drastische Reduktion des Suchraumes für die Fehlersuche in Modellprüfungsproblemen bewirken.

3 Handlungsplanung

Die *Handlungsplanung* in der Künstlichen Intelligenz kann als Explorationsfragestellung in einem endlichen und diskreten Zustandsraum aufgefaßt werden. Dabei ist ein Zustand über die Konjunktion von Fakten (instanziierte Prädikate) festgelegt, die in diesem Zustand gelten. Aufgrund der kombinatorischen Vielfalt möglicher Zustände selbst bei einer moderaten Anzahl von Fakten werden zur effizienten Speicherung üblicherweise sich gegenseitig ausschließende Fakten in Gruppen zusammengefaßt. Somit betrachtet man ein deterministisches Kontrollproblem, das durch die Suche nach einer Lösungssequenz, die den Startzustand in einen der durch ein Prädikat festgelegten Zielzustände überführt, gelöst wird.

Erfüllbarkeitsplanen: Es gibt sehr unterschiedliche Ansätze zur Lösung von Planungsproblemen. Theorembeweiser, wie SAT-Plan [37], nutzen eine binäre Codierung der Planungszustände und ihrer Übergänge, um durch Überprüfung der Erfüllbarkeit einer das Problem beschreibenden Boole'schen Formel für ansteigendes l festzustellen, ob es in l Schritten lösbar ist. Eine aus diesem Ansatz entstandene insbesondere in der Hardwareverifikation erfolgreiche Modellprüfungsmethode ist das *Bounded Model Checking*, das prüft, ob ein Spezifikations- bzw. Implementationsfehler in dem bis zur Tiefe l explorierten Zustandsgraphen existiert. Die Handlungsplanung mit binären Entscheidungsdiagrammen [10] ist eine konsequente Erweiterung der SAT-Planungsidee, in der die effiziente und eindeutige Darstellungsmöglichkeit der erreichbaren Zustände durch ein BDD genutzt wird. Bei der Erreichbarkeitsanalyse mit Hilfe der symbolisch definierten Transitionsfunktion bleibt zudem auch die Zahl der verwendeten Variablen in der Exploration konstant. SAT-Plan wurde durch Methoden des Constraint-Satisfaction-Problemlösens erweitert, wobei Faktgruppen als Zahlenmenge codiert sind [51]. Die Verfahren können dann leicht um Ressourcen, d.h. um weitere Variablen ergänzt werden. Ein vergleichbarer Schritt zur Zahlrepräsentation wird in der Planung mit Hilfe der ganzzahligen Programmierung anvisiert [38].

Gerichtetes Planen: Der Vorteil, Zahlen und große Zustandsmengen symbolisch darstellen zu können, wird in den bis dato verfolgten Ansätzen im allgemeinen durch den Nachteil einer ungerichteten Exploration erkauft. Daß dies ein gravierender Mangel ist, zeigen die Wettbewerbsergebnisse, bei der heuristische Suchverfahren und trickreiche oder anwendergegebene Beschneidungs-

techniken besonders erfolgreich waren. Man kann die erfolgreichen Ansätze grob in A*-Varianten [30] und lokale Suchverfahren [32] unterteilen. Bei genauerer Betrachtung läßt sich die effiziente Problemgraph-Analyse in *Graphplan* [4] als heuristisches Suchverfahren deuten, in dem in einem Vorwärtsexplorationsschritt Abstandsinformation für die Rückwärtssuche aufgesammelt wird.

Vorverarbeitung: Entscheidend für die Effizienz von Handlungsplanungssystemen ist die Vorverarbeitung der Problembeschreibung [24]. Da die gefundenen Problemvarianzen bei den unterschiedlichen Ansätzen eine vielfältige Ausprägung besitzen, schlagen wir vor [17], die Effizienz der Vorverarbeitung in der Länge der erzielten Zustandskodierung zu messen. Ein weiterer Ansatz [46] versucht Merkmale, sogenannte *Generische Typen*, für verschiedene Domänen zu extrahieren, so daß Transportprobleme von Schedulingfragestellungen u.ä. voneinander unterschieden und effizientere, weil speziellere, Algorithmen angewandt werden können. Dieser Ansatz, obwohl effektiv, widerspricht der allgemeinen Planungsidee, da er die begrenzte Vielfalt der Beispieldomänen explizit nutzt. Die Reduzierung der Operatorenanzahl [31] innerhalb eines Planungsproblems kann ebenfalls in der Vorverarbeitungsphase erfolgen und somit zur Suchraumbeschnidung führen. Hierbei wird eine vereinfachte Exploration zur Invariantenerkennung und Pfadanalyse angestoßen.

Modellprüfungsplanen: Aktuell erobern symbolische Techniken der Modellprüfung das Gebiet der Handlungsplanung [26], wie die unterschiedlichen Aspekte von *Traditioneller Planung* [17], *Nicht-Deterministischer Planung* [8], *Universeller Planung* [10] und *Konformanter Planung* [9] zeigen. Dabei bezieht der Nicht-Determinismus und das konformante Planen mehrere Nachfolgezustände mit ein, während universelles Planen zusätzlich unsicheres Wissen über Initialzustände erlaubt. Es wurde festgestellt, daß die Bestimmung *starker zyklischer* Pläne, die unabhängig vom Initialzustand und gegebenen Nicht-Determinismus das Ziel erreichen, der Temporalformel AGEF Goal in der Temporal-Logik CTL entsprechen, während bei *schwachen* Plänen die Erfüllung der Formel EF Goal gegeben ist. In dem Planungs-Wettbewerb AIPS-2002 soll die Beschreibungssprache auch vom Planer unkontrollierbare Prozesse und Ereignisse beinhalten, die u.U. mit Modellprüfungsansätzen in einer Vorverarbeitung als Zeit-Hybride Automaten [2] erschlossen werden können [25]. Dennoch ist die Eingliederung in Modellprüfungsmethoden hier noch nicht abschließend gelungen.

Allgemeines Planen: Ein weiterer Zweig in der Handlungsplanung befaßt sich mit Markov'schen Entscheidungsprozessen, kurz MDPs, die zufällige Übergänge zwischen den Zuständen gestatten. Lösungen von MDPs sind dann sogenannte *Politiken*, die durch eine Zustands-Aktionstabelle dargestellt werden können. Weitere Verallgemeinerungen, sogenannte partiell beobachtbare MDPs (POMDPs [52]) erlauben es, den Zustand nur an bestimmten Eigenschaften einzusehen. Die Lösung von POMDPs erweist sich theoretisch und praktisch als schwer, jedoch insbesondere in der Robotik als wichtig. Während die propositionale Handlungsplanung *nur* PSPACE-vollständig ist, bewegen wir uns bei der POMDP-Planung an den Grenzen der Berechenbarkeit. Erfolgreiche, gerichtete und approximative Suchansätze verlegen das Problem, nach optimalen Politiken zu su-

chen, üblicherweise in den Hypothesenraum [6], in dem ein Zustand einer Menge von möglichen Zuständen im Ursprungszustandsraum entspricht. Zur Lösung dieses nun wieder deterministischen Problems werden unter anderem die Realzeitalgorithmen RTA* und LRTA* [40]. Realzeitverfahren erzwingen Übergänge in einem engen Zeitfenster, haben allerdings die Möglichkeit, über Iteration die optimale Bewertung für die Politik zu lernen. Ein neuer symbolischer Ansatz [3] konzentriert sich anstatt auf das POMDP-Modell auf das Problem des nicht-deterministischen Planens mit partieller Beobachtbarkeit (partial observability).

4 Suchverfahren

Der Bereich der *Heuristischen Suche* stellt einen, wenn nicht den algorithmischen Kernbereich der künstlichen Intelligenz dar. Hierbei wird das Problemlösen als Suche in einem Zustandsraum modelliert, d.h. man geht von einem Startzustand aus, der das gegebene Problem beschreibt und hat Übergangsregeln, um von einem Zustand zu einem oder mehreren Nachfolgezuständen zu gelangen. Diese Übergangsregeln müssen ausgehend vom Startzustand dann solange angewandt werden, bis schließlich ein gewünschter Endzustand erreicht ist. In der Regel ist man an einer kürzestmöglichen Folge derartiger Übergänge interessiert und beschreibt den Problemgraphen implizit durch eine Nachfolgererzeugungsfunktion. Allen gerichteten Suchverfahren ist eine allgemeine Schätzfunktion (Heuristik) der verbleibenden Weglänge zum Ziel gemein. Ist sie eine untere Schranke, so heißt die Schätzfunktion optimistisch. Eine stärkere, aber häufig anzutreffende Eigenschaft ist die der Konsistenz, die zu monotonen Kostenfunktionen führt.

Speicherplatzsensible Suche: Das A*-Verfahren [29] mit monotonen Kostenfunktionen entspricht dem Kürzesten-Wege Algorithmus von Dijkstra in einem durch den Schätzer umgewichteten Graphen. Ist $w(e)$ das alte Gewicht einer Kante $e = (v, w)$, so ist $w(e) + h(v) - h(w)$ das neue Gewicht. Für optimistische Heuristiken können Kantengewichte dadurch negativ werden, so daß manche Knoten mitunter mehrfach exploriert werden müssen. Der Hauptnachteil des A*-Verfahrens ist der hohe Speicheraufwand, da alle einmal generierten Knoten im Hauptspeicher verwaltet werden müssen. Im Gegensatz dazu läßt sich mit binären Entscheidungsdiagrammen (BDDs) eine große Menge von Zuständen sehr kompakt beschreiben [7]: Die einzelnen Problemsituationen werden binär codiert und die charakteristischen Funktionen von mehreren Stellungen zusammengefaßt in einem BDD dargestellt.

Speicherplatzbeschränkte Suche: Einfache Tiefensuchverfahren wie Branch and Bound und IDA* [41] kompensieren das Speicherplatzproblem durch einen Mehraufwand an Zeit. Erweiterungen, die den gesamten Hauptspeicher in die Berechnung integrieren, loten den Bereich zwischen A* und IDA* i.a. durch verschiedene Zwischenspeicherungsstrategien aus. Insbesondere die praktische Relevanz von Zwischenspeicherungsstrategien ist zu hinterfragen. Zwar ist der Einsatz von einer Transpositionstabelle in IDA* unbestritten, aber darüber hinaus ist das Spektrum der bis dato untersuchten Probleme durch reguläre Einpersonenspiele mit uniform gewichteten Problemgraphen zu eingeschränkt, um

durch verfeinertes Zwischenspeichern zum Erfolg zu kommen. Eine aktuelle Arbeit nutzt statistische Methoden, um die Zwischenspeicherung von Knoten zu verbessern [49]. Sie wurden in der praktisch immer wichtiger werdenden Domäne der DNA-Sequenzalignierung eingesetzt. Dies ist eines der wenigen betrachteten Beispiele im Bereich der KI-Suche mit einem gewichteten Problemgraphen.

Explorationsbeschleunigung: Derzeit konzentriert sich die Forschung der Heuristischen Suche auf verschiedene Explorationsbeschleunigungs- und Pruning-techniken. Divide-And-Conquer Verfahren [44] umgehen die Speicherung der mitunter sehr viel größeren Closed-Liste. Erste Erfolge wurden auch hier bei der DNA-Sequenzalignierung erzielt. Die untersuchten Algorithmen besitzen eine starke Verwandtschaft zur effizienten Berechnung der minimalen Editierdistanz mit beschränktem Platz und können sich in der Performanz noch nicht ganz mit dem State-of-the-Art in der Sequenzalignierung messen.

Suchraumbeschneidung: Weitere Optionen sind automatische Suchbaumbeschneidungstechniken, wie das Finite-State-Pruning [53], das einen endlichen Automaten synchron zur Zustandsraumtraversierung durchläuft und bei Akzeptanz die Nachfolgenergenerierung unterdrückt. Es wird angenommen, daß der Zustandsübergang einem Buchstaben aus einem festen Alphabet entspricht und somit der Automat eine Teilworterkennung unverheißungsvoller Wege durchführt. Die Automaten können in einem Vorverarbeitungsschritt gelernt werden. Dabei findet der Algorithmus von Aho und Corasick zur Teilstringerkennung mit Hilfe eines endlichen Automaten Anwendung. Relevance-Cuts [36] schließen eine stete Umorientierung innerhalb des Suchraumes aus. Hierbei wird der Bearbeitung eines Teilproblems eine gewisse Bedeutung und Bearbeitungszeit zugemessen. Dies entspricht einer der Annahme einer zum Teil vorhandenen Unabhängigkeit der zu erreichenden Ziele. Die Technik ist recht effektiv, führt jedoch i.a. zum Verlust der optimalen Lösung.

Musterdatenbanken: Musterdatenbanken [13] verwalten tabellarisch die Ergebnisse der Exploration eines relaxierten Problems als untere Schranke im Hauptspeicher. In der Vereinfachung des Problems und vor der eigentlichen Suche startet man eine vollständige Breitensuche vom Zielzustand ausgehend und nutzt perfektes Hashing zur Speicherung der gefundenen Distanzen. Mit diesem Abstraktions-Ansatz zur Erstellung speicherplatzbasierter Heuristiken gelang es erstmalig, den Zauberwürfel (Rubik's Cube) zu lösen [42].

Kürzeste-Wege Datenstrukturen: Auf der Seite der kürzesten Wegesuche in gewichteten Graphen wurde insbesondere an effektiven Datenstrukturen gefeilt. Für ganzzahlige Kantengewichte wurden ausgehend von Dials Implementation einer Vorrangwarteschlange durch eine einfache Bucket-Liste verschiedene Strukturen untersucht [1]: *Zweistufige Bucketlisten* reduzieren die worst-case Komplexität von $O(C)$ zu $O(\sqrt{C})$, *Radix Heaps* zu $O(\log C)$ amortisierter Laufzeit und ein Hybridverfahren mit Fibonacci Heaps erreicht $O(\sqrt{\log C})$. Dabei ist C das maximale Kantengewicht in dem zu explorierenden Graphen. Für den Fall, daß die Knotenanzahl größer als $\log C$ ist, wird ein Verfahren mit Laufzeit $O(\log \log C)$ vorgestellt. Für eine RAM mit beliebiger Wortbreite ist derzeit ein $O(\log \log n)$ Zugriff State-of-the-Art [55]. Sehr aktuell sind theoretische Resulta-

te, die die Linearzeit der kürzesten Wege-Suche, z.B. in ungerichteten Graphen garantieren [54]. Auf der Seite reeller Kantengewichte konkurrieren trickreiche Bucketierungsverfahren, wie das δ -stepping und ADAP-SP [48] (Linearzeit im Mittel für zufällige Kantengewichte in $[0, 1]$) mit den etablierten Fibonacci-Heaps ($O(1)$ bzw. $O(\log n)$ amortisierter worst-case Zeit). Da das A*-Verfahren eine entscheidende Erweiterung und Verbesserung des Algorithmus von Dijkstra darstellt, steht hier eine intensive praktische und theoretische Effektivitätsanalyse der verschiedenen Datenstrukturen aus.

5 Eigene Arbeiten

In diesem Abschnitt stellen wir einige unserer Arbeiten in dem Schnittfeld von Heuristischer Suche, Handlungsplanung und Modellprüfung vor.

Softwaredesign: Zur Validierung und Verifikation von Kommunikationsprotokollen wurden die Implementationsbemühungen dahingehend fokussiert, einen auf gerichteter Suche beruhenden Modellprüfer für Kommunikationsprotokolle zu verwirklichen. Unser Tool HSF-SPIN baut auf den von G. Holzmann in den *Bell Laboratories* entwickelten Automaten-basierten Modellprüfer SPIN und der Beschreibungssprache Promela auf. In HSF-SPIN wird die Suche nach Fehlern in der Spezifikation (wie z.B. Deadlocks, verletzte Zusicherungen und System-Invarianten) durch automatisch generierte Heuristiken unterstützt [20]. Der Vorteil zu den bestehenden Ansätzen ist eine in der Anzahl expandierter Knoten exponentielle Ersparnis. Auch für Lebendigkeits- und LTL-Eigenschaften wurden die ersten gerichteten Algorithmen entwickelt und evaluiert [19].

Hardwareverifikation: Die *Gerichtete Modellprüfung* wurde weiterhin effektiv in der Hardwareverifikation zur Fehlersuche für zwei skalierbare Beispiele angewandt [50] (Tree-Arbitrator und Distributed Mutual Exclusion). Dabei wurden eine Strategie zur automatischen Erschließung symbolisch-repräsentierter, aussagekräftiger monotoner Heuristiken für Schaltkreise eingesetzt und der Modellprüfer μ cke um die Option einer gerichteten Exploration erweitert.

Suchverfahren: Die im Bereich der *Heuristischen Suche* verfaßten Arbeiten beinhalten erste Ergebnisse zur Symbiose der symbolischen Exploration und der gerichteten Suche [22]. In dem Algorithmus BDDA* wird erstmalig versucht, die Vorteile des A*-Verfahrens mit denen der auf traditionellen BDDs basierenden Breitensuche zu verbinden. Damit wird ein neuer Mittelweg zwischen der Zeit- und Speicheranforderung beschritten. Um die charakteristische Funktion aller von dem Startzustand s aus erreichbaren Positionen zu beschreiben, wenden wir die Nachfolgerrelation T auf die charakteristische Funktion von s an. Damit fragen wir nach allen Nachfolgern s' , die $T(s, s')$ erfüllen. In der Erweiterung auf die heuristische Suche kann der Schätzer h als eine Relation von Tupeln der Form $(estimate, state)$ angesehen werden, die genau dann den Wert 1 ergibt, wenn $h(state)$ gleich $estimate$ ist. Wir nehmen an, daß h sich als BDD für den gesamten Problemraum darstellen läßt - eine realistische Annahme, falls h nicht allzu kompliziert ist. Die Vorrangwarteschlange kann ebenso als BDD dargestellt werden. Das BDD beschreibt eine Relation von Tupeln der Form $(merit, state)$.

Dabei ist *merit* die zu einem Zustand *state* assoziierte Priorität. Wir konnten zeigen, daß für konsistente Heuristik die Anzahl der Iterationen des BDDA*-Verfahrens höchstens quadratisch in der Lösungslänge ist.

Suchraumbeschneidung: Die Arbeit zum automatenbasierten Pruning [14] konzentriert sich auf die inkrementelle Suchraumbeschneidung durch generalisierte Suffix-Bäume. Es erweitert den bestehenden Ansatz von Taylor und Korf dahingehend, daß neu gefundene zu vermeidende Teilzeichenketten in Linearzeit in den Automaten eingefügt werden können, ohne den eigentlichen Suchvorgang zu behindern. Weiterhin bleibt der Speicherplatzverbrauch bei mehreren Einfüge- und Löschoptionen im Zeichenkettenwörterbuch optimal. Der Ansatz entlastet die für die Suche erforderliche Hashtabelle in speicherplatzbeschränkten Verfahren und reduziert den Verzweigungsgrad des Suchbaumes. Der Effekt läßt sich genau quantifizieren, da die Suchbaumgröße des IDA*-Verfahrens [43] wie auch der asymptotische Verzweigungsgrad [18] effizient prognostiziert werden können. Die Sackgassenerkennung in gerichteten Graphen [15] kann als maschinelles Lernverfahren gedeutet werden, die die Beschneidung des Suchraumes dann ermöglicht, wenn sich genug Indikatoren als Muster im aktuellen Zustand finden lassen. Im Sokoban-Problem war dieser Ansatz, der einfache Sackgassenmuster des unterliegenden Problemgraphen zu immer größeren im Divide-and-Conquer Verfahren zusammenfaßt, das entscheidende Gradmaß zur Lösung vieler Benchmarkprobleme. Dieses Prinzip der Sackgassenerkennung eignet sich sehr gut zur Behandlung von Problemen bei lokalen Suchverfahren im Planen mit gerichteten Problemgraphen.

Zustandsraumspeicherung: Die bisher erzielten Ergebnisse konzentrieren sich insbesondere auf die Weiterentwicklung von speicherplatzbeschränkten Algorithmen [21]. Wir konnten z.B. zeigen, daß es mit sogenannten *Suffix-Listen* möglich ist, eine Anzahl von Zuständen verlustfrei zu speichern, die sehr nahe an der informations-theoretischen Grenze liegt. In der Literatur wird die Grenze zwischen der verlustbehafteten und verlustfreien Speicherung hingegen nicht ausreichend behandelt. In der effektive Verbindung der partiellen mit der heuristischen Suche wird die vollständige Speicherung von Zuständen in der Transpositionstabelle des IDA*-Verfahrens durch Bit-State Hashing ersetzt. Der Ansatz ist sehr erfolgreich im Bereich der Protokollvalidation [20], da die Zielzustandsdichte größer ist als in regulären Einpersonenspielen, bei denen die Partielle Suche wenig fruchtet. Neuere ergebnisse zeigen jedoch, daß sich die partielle Suche in komplexen Einpersonenspielen wie dem Atomix-Problem [35] effektiv einsetzen läßt. Die effektive Nutzung des Sekundärspeichers [23] ist ein recht neues Ergebnis. Hierbei wurde ein Routenplaner um die Option bereichert, Lokalität bezüglich des Sekundärspeichers in der Suche auszunutzen. Die Minimierung des genutzten Sekundärspeichers entspricht der Minimierung des Kommunikationsaufwandes in einer parallelen Implementierung des A*-Verfahrens. Eigene Studien zeigen außerdem, daß auch die gerichtete Handlungsplanung von der Technik der Musterdatenbanken profitiert [16], mit der in einem Vorverarbeitungsschritt in mehreren Datenbank gute, aufzuaddierende Schätzwerte für die Gesamtexploration ermittelt werden.

6 Ausblick

Die Forschungsbemühungen im Bereich der gerichteten Modellprüfung zielen auf eine in der Praxis einsetzbare halb- bzw. vollautomatischen Fehlerbereinigung im Designprozess verteilter Systeme durch das Paradigma der gerichteten Suche. Dabei wird die algorithmische Grundlagenforschung und durch die verschiedenen Anwendungsgebiete in der Planung und Modellprüfung und gerichteter Suche getragen.

Dank Wir danken Stefan Leue für die Unterstützung in der angestrebten Forschungsrichtung.

Literatur

1. R. K. Ahuja, K. Mehlhorn, J. B. Orbin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, pages 213–223, 1990.
2. R. Alur and D. Dill. Automata for modelling real-time systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pages 322–335. Springer, 1990.
3. P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. To appear.
4. A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1636–1642, 1995.
5. B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 1–12. Springer, 1996.
6. B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 52–61, 2000.
7. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):142–170, 1992.
8. A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 130–142. Springer, 1997.
9. A. Cimatti and M. Roveri. Conformant planning via model checking. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 21–33. Springer, 1999.
10. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, pages 875–881, 1998.
11. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
12. G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 3 edition, 2001.
13. J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(4):318–334, 1998.

14. S. Edelkamp. Suffix tree automata in state space search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science, pages 381–385. Springer, 1997.
15. S. Edelkamp. Neue Wege in der Exploration. In *Informatik*, Lecture Notes in Computer Science, pages 65–77. Springer, 2000.
16. S. Edelkamp. Planning with pattern databases, 2001. Submitted (Technical Report 142, Computer Science Institute, University of Freiburg).
17. S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 135–147. Springer, 1999.
18. S. Edelkamp and R. E. Korf. The branching factor of regular search spaces. In *National Conference on Artificial Intelligence (AAAI)*, 1998. 299–304.
19. S. Edelkamp, A. L. Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In *SPIN Workshop*, Lecture Notes in Computer Science, pages 57–79. Springer, 2001.
20. S. Edelkamp, A. L. Lafuente, and S. Leue. Protocol verification with heuristic search. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, pages 75–83, 2001.
21. S. Edelkamp and U. Meyer. Theory and practice of time-space trade-offs in memory limited search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science. Springer, 2001. To appear.
22. S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science, pages 81–92. Springer, 1998.
23. S. Edelkamp and S. Schrödl. Localizing A*. In *National Conference on Artificial Intelligence (AAAI)*, pages 885–890, 2000.
24. M. Fox and D. Long. The automatic inference of state invariants in TIM. *Artificial Intelligence Research*, 9:367–421, 1998.
25. M. Fox, D. Long, S. Bradley, and J. McKinna. Using model checking for pre-planning analysis. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, pages 23–31, 2001.
26. F. Giunchiglia and P. Traverso. Planning as model checking. In *European Conference on Planning (ECP)*, pages 1–19, 1999.
27. P. Godefroid and D. E. Long. Symbolic protocol verification with Queue BDDs. *Formal Methods in System Design*, 1997.
28. S. Graf and H. Saidi. Construction of abstract state graphs of infinite systems with PVS. In *Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science. Springer, 1997.
29. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for heuristic determination of minimum path cost. *IEEE Transaction on SSC*, 4:100, 1968.
30. P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 140–149, 2000.
31. P. Haslum and P. Jonsson. Planning with reduced operator sets. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 150–158, 2000.
32. J. Hoffmann. Fast plan generation through heuristic search. *Artificial Intelligence Research*, 2001.
33. G. J. Holzmann. An analysis of bitstate hashing. *Formal Methods in Systems Design*, 13(3):287–305, 1998.
34. G. J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools For Technology Transfer*, 2(3):270–278, 1999.

35. F. Hüffner, S. Edelkamp, H. Fernau, and R. Niedermeier. Finding optimal solutions to atomix. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science. Springer, 2001. To appear.
36. A. Junghanns. *Pushing the Limits: New Developments in Single-Agent Search*. PhD thesis, University of Alberta, 1999.
37. H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, pages 1194–1201, 1996.
38. H. Kautz and J. Walser. State-space planning by integer optimization. In *National Conference on Artificial Intelligence (AAAI)*, 1999.
39. Y. Kesten and A. Pnueli. Modularization and abstraction: The keys to formal verification. In *Mathematical Foundations of Computer Science*, pages 54–71. Computer Society Press, 1998.
40. R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.
41. R. E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.
42. R. E. Korf. Finding optimal solutions to Rubik’s Cube using pattern databases. In *National Conference on Artificial Intelligence (AAAI)*, pages 700–705, 1997.
43. R. E. Korf, M. Reid, and S. Edelkamp. Time Complexity of Iterative-Deepening-A*. *Artificial Intelligence*, 2001.
44. R. E. Korf and W. Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. In *National Conference on Artificial Intelligence (AAAI)*, pages 910–916, 2000.
45. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Perserving abstractions for the verification of concurrent systems. *Methods in Systems Design*, 6:1–35, 1995.
46. D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 196–205, 2000.
47. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Press, 1993.
48. U. Meyer. Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In *Symposium on Discrete Algorithms (SODA)*. ACM, 2001. To appear.
49. T. Minura and T. Ishida. Stochastic node caching for efficient memory-bounded search. In *AAAI*, pages 450–459, 1998.
50. F. Reffel and S. Edelkamp. Error detection with directed symbolic model checking. In *World Congress on Formal Methods (FM)*, Lecture Notes in Computer Science, pages 195–211. Springer, 1999.
51. J. Rintanen and H. Jungholt. Numeric state variables in constraint-based planning. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pages 109–121. Springer, 1999.
52. E. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.
53. L. A. Taylor and R. E. Korf. Pruning duplicate nodes in depth-first search. In *National Conference on Artificial Intelligence (AAAI)*, pages 756–761, 1993.
54. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
55. M. Thorup. On RAM priority queues. *SIAM Journal of Computing*, 30:86–109, 2000.
56. P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Conference on Concurrency Theory (CONCUR)*, Lecture Notes in Computer Science, pages 233–246. Springer, 1993.