Safe, Strong and Tractable Relevance Analysis for Planning

Patrik Haslum
Research School of Computer Science
Australian National University
and NICTA Optimisation Research Group
patrik.haslum@anu.edu.au

Malte Helmert
Dept. Math. and Computer Science
University of Basel
Basel, Switzerland
malte.helmert@unibas.ch

Anders Jonsson
Dept. Info. and Comm. Technologies
Universitat Pompeu Fabra
Barcelona, Spain
anders.jonsson@upf.edu

Abstract

In large and complex planning problems, there will almost inevitably be aspects that are not relevant to a specific problem instance. Thus, identifying and removing irrelevant parts from an instance is one of the most important techniques for scaling up automated planning. We examine the path-based relevance analysis method, which is safe (preserves plan existence and cost) and powerful but has exponential time complexity, and show how to make it run in polynomial time with only a minimal loss of pruning power.

Introduction

The difficulty of planning increases, often exponentially, with the size of the problem description and – unfortunately – for most planners it makes little difference how much of the description is actually relevant for solving the problem. The larger and more complex planning problems become, the more likely is it that their formalisations contain irrelevant parts, and the greater is the computational cost of not realising it. It is fair to say that the identification and efficient treatment of irrelevance is one of the key issues in building scalable planners.

As important as relevance is, as difficult it is to define precisely. In a sense, the minimal set of actions relevant for solving a planning problem are those that appear in a (shortest) plan. What is relevant may also change as decisions are made during planning. We consider static relevance analysis, meaning the analysis is carried out as a preprocessing step before planning, in a way that is independent of the planner's approach to solving the problem. That is, the analysis acts as a simplifying problem transformation.

An ideal static relevance analysis method should have three properties: First, it should be *safe*, meaning that removing actions found not to be relevant preserves plan existence, as well as optimal plan cost. Second, it should be *strong*, meaning that it identifies much of the irrelevancies present in a planning problem. Third, it should be *efficient*, in a theoretical as well as a practical sense. These three aims are contradictory, so compromises have to be made. The standard method of back-chaining from subgoals to achievers and from actions to preconditions, and eliminating actions and atoms not reachable in this way from the problem goals (Gazen and Knoblock 1997) is efficient and preserves optimal plans, but is too weak to have any significant effect in most problems. Stronger conditions for action relevance, such as Scholz's (2004) path analysis method or Nebel et al.'s (1997) minimum sets of initial facts, for example, are intractable to decide. Most researchers have approached this obstacle by designing tractable approximations that sacrifice completeness (Nebel, Dimopoulos, and Koehler 1997; Hoffmann and Nebel 2001).

We aim for a different compromise: Starting from a previous adaptation of Scholz's method (Jonsson 2007), we design a polynomial-time approximation that preserves completeness and optimality, at the cost of only slightly weaker irrelevance pruning. We are still able to show that this analysis prunes actions that the standard back-chaining method does not. Moreover, our approach does not require the causal graph to be acyclic, although the analysis is stronger when actions are unary.

Notation

We present our algorithm in the SAS$^+$ finite domain planning framework, which we here briefly recap. The domain transition graphs of the SAS$^+$ variables are central to path-based relevance analysis methods, which operate on paths in these graphs.

Let V be a set of variables, and $D(v)$ the domain of each variable $v \in V$. We assume, w.l.o.g., that $D(v) = \{1, \ldots, N(v)\}$ for each $v \in V$, where $N(v) = |D(v)|$. A partial state s is a function on a subset of variables $V_s \subseteq V$ that assigns a value $s(v) \in D(v)$ to each variable $v \in V_s$. If $V_s = V$, s is a (full) state. The projection $s \mid W$ of a partial state s onto a subset of variables $W \subseteq V$ is a partial state t such that $V_t = V_s \cap W$ and $t(v) = s(v)$ for each $v \in V_t$. The composition $s \oplus t$ of two partial states s and t is a partial state u such that $V_u = V_s \cup V_t$, $u(v) = t(v)$ for each $v \in V_t$, and $u(v) = s(v)$ for each $v \in V_s - V_t$. Two partial states s and t *match* if $s \mid V_t = t \mid V_s$. If, in addition, $V_s \subseteq V_t$, we say that s *subsumes* t, which we denote $s \sqsupseteq t$.

A SAS$^+$ planning instance is a tuple $P = \langle V, A, I, G \rangle$ where V is a set of variables, A is a set of actions, I is an initial state, and G is a (partial) goal state. Each action $a \in A$ has precondition $\mathrm{pre}(a)$ and effect $\mathrm{eff}(a)$, both partial states on V. Action a is applicable in state s if $\mathrm{pre}(a) \sqsupseteq s$ and results in a new state $s' = s \oplus \mathrm{eff}(a)$. Action a is *unary* iff

$|V_{\mathrm{eff}(a)}| = 1$. We say the instance is unary iff all actions are.

The domain transition graph, or DTG, of a variable $v \in V$ is a directed graph $\mathrm{DTG}(v) = (D(v), E)$ with the values in the domain $D(v)$ of v as nodes. There is an edge $(x, y) \in E$ if and only if $x \neq y$ and there exists an action $a \in A$ such that $\mathrm{eff}(a)(v) = y$ and either $v \notin V_{\mathrm{pre}(a)}$ or $\mathrm{pre}(a)(v) = x$.

Path Relevance Analysis

The relevance analysis method proposed by Scholz (2004) works by identifying which paths through the DTGs can be replaced by other paths. Replaceable paths are irrelevant, and any action that appears only on such paths is irrelevant.

A path through the DTG of a variable $v \in V$ corresponds to a sequence of actions $\langle a_1, \ldots, a_k \rangle$. This induces a sequence of preconditions on variables other than v, i.e. partial states, $\langle \mathrm{pre}(a_1) \mid V - \{v\}, \ldots, \mathrm{pre}(a_k) \mid V - \{v\} \rangle$, and similarly a sequence of effects on $V - \{v\}$. Path replaceability means that the action sequence, when appearing in a valid plan, can be replaced by the actions of the replacement path without invalidating the plan. A path p from value x to value y in $\mathrm{DTG}(v)$ can be replaced by a different path q from x to y iff q and p have the same effects on variables other than v and the precondition sequence of q has no more requirements than that of p. To preserve optimal plan length (cost), the length (cost) of q must be no greater than that of p. Note that if all actions are unary, any path containing cycles is replaceable by the same path with the cycles removed.

For each variable $v \in V$, relevance analysis identifies a set of value pairs $(x, y) \in D(v)$ (called "starts" and "stops") such that a plan solving P may need to change the value of v from x to y. Initially, the only pair is $(I(v), G(v))$ for $v \in V_G$. It then identifies the irreplaceable paths from x to y in $\mathrm{DTG}(v)$. The preconditions and effects of these paths become new starts and stops for other variables. The process is iterated to a fixpoint where all starts and stops induced by the current set of paths are connected (if possible) by those paths. An action not appearing on any path in the fixpoint set is irrelevant.

The number of distinct paths through a DTG may be exponential in the number of actions affecting the variable (consider, as an example, a variable with n values in which every pair of values is connected by a different action: this yields $O(n^2)$ actions, but $O((n-1)!)$ distinct paths connecting any pair of values). Scholz' analysis explicitly examines all these paths, making its worst case complexity exponential. Note, however, that it is not known whether deciding path relevance of an action requires exponential time, since the question that needs to be answered is only if the action lies on some relevant path. In the next section, we introduce our approximation method, which groups paths into sets and performs the replaceability check on those sets. This is how we ensure tractability.

Approximate Analysis for Unary Domains

In this section we present our method for approximate path relevance analysis. It follows a previous adaptation of Scholz' method (Jonsson 2007), apart from the component that identifies irreplaceable paths. Our new component runs in polynomial time and approximates path irreplaceability in a safe way, by grouping together sets of "similar" paths. However, this also leads to some loss of precision, so that the approximate method may label fewer actions as irrelevant.

In the rest of this section we assume that all actions are unary. We will discuss the challenges and potential ways of extending it to non-unary actions in the next section. When all actions are unary, the side effect sequences of all paths are empty so the replaceability condition reduces to comparing the precondition sequences. Hence, each path is, for the purpose of our analysis, fully characterised by its start and end values and its precondition sequence, and we will view a path simply as a sequence of partial states.

Let $p = \langle s_1, \ldots, s_k \rangle$ and $q = \langle t_1, \ldots, t_m \rangle$ be two paths, i.e., sequences of partial states. We use $a(p)$ to denote the action sequence associated with path p. We say that p *subsumes* q, which we denote $p \sqsupseteq q$, if $k \leq m$ and there exist integers $j_1, \ldots, j_k$ such that $1 \leq j_1 \leq \cdots \leq j_k \leq m$ and $s_i \sqsupseteq t_{j_i}$ for each $1 \leq i \leq k$. Intuitively, one may read $p \sqsupseteq q$ as "p is more generally applicable than q". This is exactly what the next lemma shows. The additional condition $k \leq m$ ensures that optimal plans are preserved.

Lemma 1 *Let p and q be two paths between x and y in some DTG. Given an action sequence with subsequence $a(q)$, if $p \sqsupseteq q$ and no goal or action precondition other than y is provided by $a(q)$, $a(q)$ can be replaced with $a(p)$.*

Proof (sketch): Since $s_i \sqsupseteq t_{j_i}$, the precondition s_i is satisfied whenever t_{j_i} is. We can thus replace each action in $a(q)$ with the sequence of its associated actions in $a(p)$. □

Our algorithm works by maintaining lower and upper bounds on the set of *all* irreplaceable paths between two nodes in a DTG. These bounds are themselves paths, i.e. sequences of partial states. The idea is that a path is replaceable if its lower bound is subsumed by the upper bound of another path (or set of paths). To represent the fact that there may be no path between a pair of nodes, we use a special constant NoPath that satisfies $p \sqsupseteq$ NoPath and NoPath $\not\sqsupseteq p$ for each path p. Our algorithm also exploits the fact that no irreplaceable path through $\mathrm{DTG}(v)$ can have more than $N(v) - 1$ edges, else it would contain cycles.

To compute bounds we introduce two operations on paths. The intersection $p \sqcap q$ of two paths p and q is a path $r = \langle s_1, \ldots, s_k \rangle$ such that $r \sqsupseteq p$, $r \sqsupseteq q$, and $\sum_{i=1}^{k} |V_{s_i}|$ is maximised; in other words, $p \sqcap q$ is a "maximal" path that subsumes both p and q. The union $p \sqcup q$ of two paths p and q is a path $r = \langle s_1, \ldots, s_k \rangle$ such that $p \sqsupseteq r$, $q \sqsupseteq r$, and $\sum_{i=1}^{k} |V_{s_i}|$ is minimised; $p \sqcup q$ is thus a "minimal" path that is subsumed by both p and q. We define NoPath $\sqcap\, p = p$ and NoPath $\sqcup\, p =$ NoPath for each path p.

Algorithm 1 shows how to compute the intersection $p \sqcap q$ between two paths $p = \langle s_1, \ldots, s_k \rangle$ and $q = \langle t_1, \ldots, t_m \rangle$ using dynamic programming. For space reasons we omit the algorithm for computing the union, which is very similar. For each pair s_i, t_j of partial states, the algorithm decides whether to exclude s_i, exclude t_j or, if s_i and t_j match, include the intersection $s_i \cap t_j$. Among these three decisions, the algorithm chooses the one that maximises the size of the

Algorithm 1 Intersection $\langle s_1, \ldots, s_k \rangle \sqcap \langle t_1, \ldots, t_m \rangle$

```
for each $0 \leq j \leq m$
    $A(0, j) \leftarrow 0$
for each $1 \leq i \leq k$
    $A(i, 0) \leftarrow 0$
    for each $1 \leq j \leq m$
        $A(i, j) \leftarrow A(i, j-1)$
        if $A(i, j) < A(i-1, j)$
            $A(i, j) \leftarrow A(i-1, j)$
        if $s_i$ and $t_j$ match
            $u \leftarrow s_i \cap t_j$
            if $A(i, j) < A(i-1, j-1) + |V_u|$
                $A(i, j) = A(i-1, j-1) + |V_u|$
extract the path backwards from $A(k, m)$
```

variable sets of the path's partial states. The path can then be extracted by remembering the decisions made. The intersection of two paths is no longer than the shortest of the two, and the union is no longer than the sum of their lengths.

Algorithm 2 identifies all edges in $\mathrm{DTG}(v)$ that lie on relevant (i.e., irreplaceable) paths starting from a given value $x \in D(v)$. Each edge $e = \langle o, d, s \rangle$ consists of an origin $e.o \in D(v)$, a destination $e.d \in D(v)$, and a partial state $e.s$ that is the precondition of the associated action. For each $i \in D(v)$, $LN(i)$ and $UN(i)$ hold the lower and upper bounds on all paths from x to i. For each edge e, $LE(e)$ and $UE(e)$ hold the lower and upper bound on paths from x that end with e, and $RE(e)$ tracks whether e is relevant.

The algorithm is a modified breadth-first search that may revisit nodes, up to at most $N(v) - 1$ times, if their lower and upper bounds change. It maintains a set $Edges$ of edges and a set $Nodes$ of nodes, and updates the lower and upper bounds of edges in $Edges$ and nodes in $Nodes$. An edge is deemed irrelevant if there is another edge with the same destination whose upper bound subsumes the lower bound of the former. The length of an irreplaceable path, and hence its lower bound, can never exceed $N(v) - 1$. Thus, we can safely ignore upper bounds longer than $N(v)$, since they can never subsume any lower bound. This is done by setting q to NoPath when $|q| \geq N(v)$, at line 26.

Algorithm 2 computes the relevant edges to all nodes. We then extract edges on irreplaceable paths from x to all current stop values y backwards from each y, identifying each relevant incoming edge to a node, and recursively visiting the origin of each such edge.

Theorem 2 *If $RE(e) = false$, either a) there exists no path from x that includes e, or b) there exists an edge f with $f.d = e.d$ and $RE(f) = true$ such that $p \sqsupseteq q$ for each irreplaceable path p from x ending with f and each path q from x ending with e.*

Proof (sketch): If a) holds $RE(e)$ is never set to true on line 17 of the algorithm. If b) holds $RE(e)$ has to be set to false on line 20 or 22, implying $UE(f) \sqsupseteq LE(e)$ for some f such that $RE(f) = true$. By the definition of the intersection and union used to compute the bounds, this implies $p \sqsupseteq UE(f) \sqsupseteq LE(e) \sqsupseteq q$ for each such pair of paths p and q. □

Algorithm 2 Relevant(v, x)

```
1  for each $1 \leq i \leq N(v)$
2      $LN(i) \leftarrow UN(i) \leftarrow$ NoPath
3  for each edge $e$
4      $LE(e) \leftarrow UE(e) \leftarrow$ NoPath
5      $RE(e) \leftarrow false$
6  $LN(x) \leftarrow UN(x) \leftarrow \langle\rangle$
7  $Edges \leftarrow \{e = \langle o, d, s \rangle : e.o = x\}$
8  repeat $N(v) - 1$ times
9      $Nodes \leftarrow \emptyset$
10     for each edge $e \in Edges$
11         $LE(e) \leftarrow \langle LN(e.o), e.s \rangle$
12         $UE(e) \leftarrow \langle UN(e.o), e.s \rangle$
13         if $e.d \neq x$ add $e.d$ to $Nodes$
14     $Edges \leftarrow \emptyset$
15     for each $i \in Nodes$
16         for each edge $e$ such that $e.d = i$
17             $RE(e) \leftarrow LE(e) \neq$ NoPath
18         for each pair of edges $(e, f)$ s.t. $e.d = f.d = i$
19             if $RE(f)$ and $UE(f) \sqsupseteq LE(e)$
20                 $RE(e) \leftarrow false$
21             else if $RE(e)$ and $UE(e) \sqsupseteq LE(f)$
22                 $RE(f) \leftarrow false$
23         $p \leftarrow$ NoPath, $q \leftarrow \langle\rangle$
24         for each relevant edge $e$ such that $e.d = i$
25             $p = p \sqcap LE(e)$
26             $q = q \sqcup UE(e)$ (or NoPath if $|q| \geq N(v)$)
27         if $p \neq LN(i)$ or $q \neq UN(i)$
28             $LN(i) \leftarrow p$
29             $UN(i) \leftarrow q$
30             add each edge $e$ such that $e.o = i$ to $Edges$
```

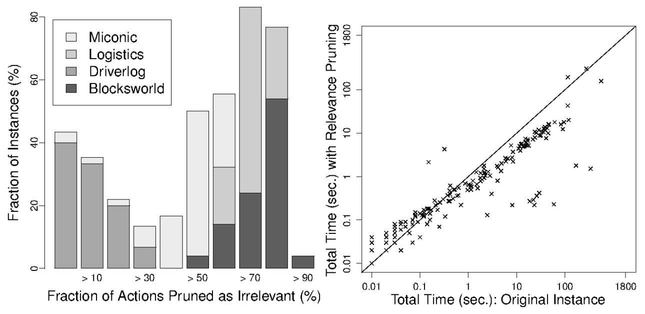
Complexity

The complexity of path subsumption, intersection and union is proportional to path lengths. Because algorithm Relevant never handles paths longer than $D = \max_{v \in V} N(v)$, these operations are bounded by $O(|V|D)$ and $O(|V|D^2)$, respectively, and Relevant runs in time $O(|E||V|D^3(|E| + D))$. Relevant is invoked at most $O(|V|D)$ times, once for each variable-value pair, so the total complexity of approximate path relevance analysis is $O(|E||V|^2 D^4(|E| + D))$, i.e., polynomial in the size of the planning instance P.

Generalisation to Non-Unary Domains

Generalising our method to non-unary domains is challenging. As already mentioned, path replacement in this case almost always requires that the two paths have exactly the same sequence of effects on other variables.[1] We can of course partition the sets of paths ending in each node into equivalence classes based on their side effects, and apply our approximate reasoning only within each such class; if the number of such classes is polynomially bounded, the algorithm remains polynomial. However, this approach is likely to suffer from greatly reduced pruning power, since it

[1]There are exceptions to this: a sequence of effects on a variable v is equivalent to the last effect in the sequence if the sequence forms a contiguous path in $\mathrm{DTG}(v)$ that does not pass through any value of v that is a start or stop.



(a) (b)

Figure 1: (a) Distribution of the fraction of actions pruned as irrelevant, using tractable analysis. (b) Total time to solve with and without relevance analysis. Total time with relevance pruning is the sum of the time for analysis and for search on the reduced problem.

is very unusual to find paths with the same effects but where one has more preconditions.

An alternative approach to extending the scope of our analysis is through problem reformulation. In many cases, a SAS$^+$ domain formulated with non-unary actions can be expressed using only unary actions, if those actions are permitted to have more complex (typically disjunctive) preconditions. Consider, for example, the Blocksworld domain: the Boolean variable $\text{CLEAR}(x)$ can be replaced by the formula $\neg\exists y\,\text{BELOW}(y) = x$, equivalent to $\forall y\,\text{BELOW}(y) \neq x$, which, because each block can be only in one place, is equivalent to $\forall y(\text{BELOW}(y) = \text{TABLE} \vee \exists z \neq x\,\text{BELOW}(y) = z)$. As long as each disjunction is over permitted values of the same variable (as in this example), the subsumption relation between states, and hence subsumption between paths, and the union and intersection operations, can be extended to this formalism in a fairly straightforward manner (Jonsson 2007). This is how we handle the non-unary BLOCKSWORLD and DRIVERLOG domains in the experiments. We remark that other reformulations that make actions unary (e.g. serialising the effects of actions) do not appear to combine well with our approach.

Experimental Results

We ran experiments using our approach in four benchmark domains: LOGISTICS, MICONIC, BLOCKS, and DRIVERLOG. We used Fast Downward (Helmert 2006) to translate planning instances to SAS$^+$. To make MICONIC unary we modified the domain so that passengers cannot be simultaneously boarded and served. For BLOCKSWORLD and DRIVERLOG we applied the problem reformulation described in the previous section. We used the reformulated instances only for path analysis, not for planning.

We measure the impact of relevance analysis in two ways: First, the fraction of actions removed as irrelevant (shown as a distribution in Figure 1(a)), and, second, the impact that this reduction has on the total running time when solving the planning problems optimally using A* search (implemented in Fast Downward). With the exception of the DRIVERLOG domain, even the approximate analysis prunes more than half of the actions in most instances. Moreover, this often has a dramatic impact on search: With relevance pruning, the number of instances solved increases by 9.9% (and is a strict superset of those solved without it). Across instances solved also without pruning, the total number of node evaluations is reduced by 49.9%, and the total time by 64%. In some simple instances that are solved very quickly, the time for analysis is greater than the saving in search time, but, as can be seen in Figure 1(b), the analysis clearly pays off as problems become harder.

To measure the impact of tractable analysis, we also performed exact analysis on all instances. Exact analysis only found fewer relevant actions in DRIVERLOG and larger instances of LOGISTICS, and did not allow solving more instances optimally than did approximate analysis. On the other hand, exact analysis timed out in 54% of the instances due to its exponential behavior.

Related Work

The standard backchaining relevance analysis can be strengthened by applying it to conjunctions of bounded size, analogously to how the standard forward reachability check is extended by the h^m heuristics. Brafman (2001) used this in the context of SAT planning. Because SAT planners solve only bounded planning problems, the set of actions relevant at a fixed number i of steps from the goal can be pruned more effectively, but to retain completeness in general only actions irrelevant at every i can be removed.

Nebel, Dimopoulos and Koehler (1997) formulated static relevance analysis as an optimisation problem that consists in selecting a set of relevant establishers that is sufficient, in the sense that every relevant atom is supported by at least one relevant action, and minimal, in the sense of inducing the smallest number of relevant atoms among those true in the initial state. Since solving this problem optimally is NP-hard, they propose several heuristic methods which are tractable, but which do not preserve completeness. Hoffmann and Nebel (2001) defined a variant of the method that is completeness preserving for *delete-free* problems. However, as it is still NP-hard, they too define tractable approximations: one that preserves completeness (for delete-free problems), but is weak in pruning power, and one that is stronger but incomplete.

Dynamic pruning methods, that work during the search, for planning have also been proposed, such as unjustified actions (Karpas and Domshlak 2011) and partial-order reductions (Alkhazraji et al. 2012). These methods, however, are complementary, as they prune certain paths, which may consist of relevant actions, from the search space, but do not remove irrelevant actions from the problem.

Detecting irrelevant actions is one of several, complementary, methods of problem simplification. Others include removing unreachable atoms/actions, or (safely) abstracting variables. These can all be combined to reduce problem complexity (Haslum 2007). Our tractable relevance analysis can be used, in addition to or as a drop-in replacement for other irrelevance detection methods, in any context where we seek to efficiently simplify a planning problem.

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. This work was partially supported by the German Research Foundation (DFG) as part of the project “Kontrollwissen für domänenunabhängige Planungssysteme” (KontWiss), as well as European FP7 project #270019 (SpaceBook).

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, 891–892.

Brafman, R. 2001. On reachability, relevance, and resolution in the planning as satisfiability approach. *Journal of Artificial Intelligence Research* 14:1–28.

Gazen, B., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proceedings of the 4th European Conference on Planning (ECP)*, 221–233.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1898–1903.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. RIFO revisited: Detecting relaxed irrelevance. In *Proceedings of the 6th European Conference on Planning (ECP)*, 325–336.

Jonsson, A. 2007. Efficient pruning of operators in planning domains. In *Lecture Notes in Artificial Intelligence: Current Topics in Artificial Intelligence – CAEPIA 2007*, volume 4788, 130–139.

Karpas, E., and Domshlak, C. 2011. Living on the edge: Safe search with unsafe heuristics. In *Proc. ICAPS'11 Workshop on Heuristics for Domain-Independent Planning*, 53–58.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proceedings of the 4th European Conference on Planning (ECP)*, 338–350.

Scholz, U. 2004. *Reducing Planning Problems by Path Reduction*. Ph.D. Dissertation, Technische Universität Darmstadt.