Towards Realistic Benchmarks for Planning: the Domains Used in the Classical Part of IPC-4 – extended abstract –

Jörg Hoffmann[*] **Stefan Edelkamp**[†]

Roman Englert[‡] **Frederico Liporace**[§] **Sylvie Thiébaux**[¶] **Sebastian Trüg**[||]

Introduction

Today, the research discipline of AI planning is largely concerned with improving the performance of general problem solving mechanisms. Performance is measured by testing systems on example instances of the problem to be solved. Clearly, since no mechanism will ever be able to perform well on *all* instances of a (hard) problem, one of the most crucial issues in such a research context is what kind of examples are used for the testing. Add on top of this that, more and more, researchers draw their testing examples from the collections used in the IPC, and it becomes evident that the IPC benchmarks are nowadays one of the most important instruments for the field.

In the organisation of the (classical part of the) 4th IPC, we therefore invested considerable effort into creating a set of "appropriate" benchmarks for planning. The criteria applied for appropriateness were that the benchmarks should be:

1. **Oriented at applications** – a benchmark should reflect an application that the field is heading for.
2. **Diverse in structure** – a set of benchmarks should cover different kinds of structure that can occur in the attacked problem.
3. **Suitable for basic research** – a set of benchmarks for a field of basic research should not omit the basic aspects of that research.

The first of these criteria is probably the one most widely agreed upon – indeed, AI planning has frequently been criticised for its "obsession with toy examples". In recent years, the performance of state-of-the-art systems has improved dramatically, and with that more realistic examples came within reach. We made another step in this direction by orienting most of the IPC-4 benchmarks at application domains. While traditionally planning benchmarks were more or less phantasy products created having some "real" scenario in mind, we took actual (possible) applications of planning technology, and turned them into something suitable for the competition. In the process of adapting an application for use in the (current form of the) IPC, inevitably some of the realism has to give way to more pragmatic considerations (expected planner performance, language capabilities, etc.). Nevertheless, we believe that the IPC-4 domains are a significant step into the right direction.

The second of the above listed appropriateness criteria has traditionally been given less attention than the first one, but we believe that it is not less important. The structure underlying a testing example determines the performance of the applied solving mechanism. This is particularly true for solving mechanisms whose performance rises and falls with the quality of a heuristic they use. Hoffmann (2002)'s results suggest that much of the spectacular performance of modern heuristic search planners is due to structural similarities between most of the traditional planning benchmarks. While this does by no means imply that modern heuristic search planners aren't useful, it certainly shows that in the creation of benchmarks there is a risk of introducing a bias towards one specific way of solving them. In selecting the benchmark domains for IPC-4, we took care to cover a range of intuitively very different kinds of problem structure.[1]

Finally, the third of our appropriateness criteria is probably agreed on by nobody – except all the people whose planners can only handle STRIPS. More seriously, we believe that, with all the new PDDL extensions, the planning community ought to not let completely go of its most basic language. Most if not all of the algorithmic approaches that have proved successful for solving temporal and numeric planning problems have originally been developed for the STRIPS language. If someone has a new idea for a planning algorithm or heuristic, he or she most certainly won't implement it for PDDL2.1 level 3 in the first go. There is also the issue of accessibility of the competition, particularly to newcomers. We made a serious effort to make even the STRIPS versions of the IPC-4 domains an interesting range of benchmarks. Instead of dropping the more interest-

[*] Institut für Informatik, Universität Freiburg, Germany
[†] Fachbereich Informatik, Universität Dortmund, Germany
[‡] T-Online, Germany
[§] Departamento de Informática, PUC Rio, Brazil. Supported by CNPq.
[¶] National ICT Australia & Computer Sciences Laboratory, The Australian National University, Canberra, Australia
[||] Institut für Informatik, Universität Freiburg, Germany

[1] We even thought of separating the domains into a set of "application" benchmarks and a set of "structurally characteristic" benchmarks. We gave up on the idea to not overly complicate the competition and its evaluation.

ing problem constraints, we *compiled* as much of the domain semantics as possible down into the STRIPS format. While in most cases this lead to rather unusual (fully grounded) encodings, we believe that the IPC-4 STRIPS benchmarks are structurally a lot more interesting than most of the previous STRIPS benchmarks.

In the rest of this extended abstract, we include a short description of each of the IPC-4 domains. We list the domains in alphabetical order, and close the article with a few concluding remarks.

Airport

The *Airport* domain was developed by Jörg Hoffmann and Sebastian Trüg. It is a PDDL adaption of an application domain developed by Wolfgang Hatzack (Hatzack & Nebel 2001), dealing with the problem of controlling the ground traffic on an airport (in such a way that the summed up travel time of all airplanes is minimised).

The problem instances in Airport specify the topology of the airport, as well as the inbound (planes that need to go to a parking position) and outbound (planes that need to go to a runway) traffic. The main problem constraint is that planes must not endanger each other. Which means that no two planes can share the same airport segment, and that a plane with running engines "blocks" a set of segments behind it (where the blocked set depends on the size category of the plane). The available actions are to "pushback" (move a plane away backwards from a parking position), to "startup" the engines, to "move" between segments, to "park" (turning off the engines), and to "takeoff" (which amounts to removing the plane from the airport).

The Airport domain versions are *non-temporal*, *temporal*, *temporal-timewindows*, and *temporal-timewindows-compiled*. The first of these versions is, as the name suggests, non-durational PDDL. In the second version, actions take time (e.g. moving across a segment takes the length of the segment divided by the speed of the plane). In the third version, there are additional time windows during which certain segments must not be used – namely, segments that belong to a runway and time windows during which a plane is known to land on that runway. The time windows are modelled using timed initial literals. In the fourth domain version, the timed initial literals are compiled into artificial (temporal) PDDL constructs, in order to make the domain version accessible to more planners.

In none of the domain versions were we able to model the true optimisation criterion – minimising makespan means minimising the travel time of the latest plane, rather than the summed up travel time of all planes. The difficulty in modelling the real optimisation criterion lies in accessing the time spans during which a plane does nothing, i.e., stays on an airport segment waiting until some other plane got out of the way. If one uses an explicit "wait" action, then one needs to introduce a discretisation of time (in order to say how long the plane is supposed to wait). We considered introducing a special "current-time" variable into PDDL2.2, returning the time of its evaluation in the plan execution. But, in a discussion with the IPC-4 organising committee, we decided against this language feature as it seemed problematic from an algorithmic point of view, and didn't seem to be very relevant anywhere except in Airport.

In all the domain versions, the problem constraints are modelled using ADL, i.e., complex preconditions and conditional effects. We compiled the ADL encodings to STRIPS by grounding out most of the operator parameters (for each individual problem instance, yielding an instance-specific domain file). The resulting STRIPS encodings formed alternative *formulations* of the domain versions, i.e. within each domain version we let the competitors choose to either attack the ADL formulation or the STRIPS formulation. The data were then evaluated together, i.e. treated as if they were all obtained on the same encoding. We applied this concept of domain *versions* and domain *version formulations* in all the IPC-4 domains.[2]

The Airport example instances were generated by Sebastian Trüg, using an airport simulation tool, called *Astras*, by Wolfgang Hatzack. Five scaling airport topologies were designed, the simulator was run, and code was implemented that, during a simulation, put out the traffic situations at selected individual time spots as the PDDL problem instances. 50 traffic situations were generated, and put out in the format needed for each of the domain versions. The second largest of the five airport topologies corresponds to one half of Munich airport, MUC. The largest of the topologies corresponds directly to the full MUC airport.

Pipesworld

The *Pipesworld* domain is a PDDL adaption of an application domain developed by Frederico Liporace and others (Milidiu, dos Santos Liporace, & de Lucena 2003), dealing with complex problems that arise when transporting oil derivative products through a pipeline system. Note that, while there are many planning benchmarks dealing with variants of transportation problems, transporting oil derivatives through a pipeline system has a very different and characteristic kind of structure. The pipelines must be filled with liquid at all times, and if you push something into the pipe at one end, something possibly completely different comes out of it at the other end. Additional difficulties that have to be dealt with are, e.g., *interface restrictions* (different types of products that must not interface each other in a pipe), *tankage restrictions* in areas (i.e., limited storage capacity defined for each product in the places that the pipe segments connect), and *deadlines* on the arrival time of products. In the form used in IPC-4, the Pipesworld domain was developed by Frederico Liporace and Jörg Hoffmann. In all versions of the domain, the product amounts dealt with are discrete in the sense that we assume a smallest product unit, called "batch". Of course, in reality the product amounts dealt with are rational numbers. Using such a numeric encoding in IPC-4 seemed completely infeasible due to complications in the modelling, and the expected capabilities of

[2] We are aware that encoding details can have a significant impact on system performance. On the other hand, we believe it is important to keep the number of distinction lines in the competition data – which is already high – as low as possible. Most current systems ground the operators out as a pre-process anyway.

the participating planners.

The problem instances in Pipesworld specify the topology of the pipeline network, the initial positions for all the batches and the goal positions for some of the batches, and the additional constraints imposed – interface restrictions, tankage restrictions, and/or deadlines. A possible action is to "push" a batch from an area into a pipe segment, making the last batch in the pipe come out at the other end. Pipe segments are modelled in a directional fashion, and we also need the inverse "pop" action where a new batch is inserted at the far end of the pipe, and the first batch in the pipe comes out. In the actual PDDL encodings used, these actions are split in several ways, to ease the modelling of their semantics. The main difficulty is that the actions must keep track of the internal state of the pipe segment involved. We introduced special case actions for pipe segments of length 1 (i.e., 1 batch). For pipe segments containing more than 1 batch, we split the push (pop) action into a push-start (pop-start) and a push-end (pop-end) action. While there is in principle no problem with doing the necessary updates within a single action, such an action contains rather many parameters. In particular, 3 parameters ranging over batches are needed – the batch to be pushed (poped), the first batch inside the pipe segment, and the last batch inside the pipe segment. Thus such an action has at least n^3 ground instances in the presence of n batches. We found that this made the domain completely infeasible for any planner that grounded out the actions. In the splited encoding, each action takes at most two batch parameters.

The Pipesworld domain versions are *notankage-nontemporal*, *tankage-nontemporal*, *notankage-temporal*, *tankage-temporal*, *notankage-temporal-deadlines*, and *notankage-temporal-deadlines-compiled*. All versions include interface restrictions. The versions with "tankage" in their name include tankage restrictions. In the versions with "temporal" in their name, actions take (different amounts of) time. The motivation for the durative actions, from an operational point of view, is that each pipeline segment has a maximum flow rate, and thus the content of some segments may be moved faster than others. The versions with "deadlines" in their name include deadlines on the arrival of the goal batches. One of these versions models the deadlines using timed initial literals, in the other version (naturally, with "compiled" in its name) these literals are compiled into artificial (temporal) PDDL constructs. None of the encodings uses any ADL constructs, so of each version there is just one (STRIPS) formulation.

The Pipesworld example instances were generated by Frederico Liporace, in a process going from random generators to XML files to PDDL files.[3] Five scaling network topologies were designed. For the domain versions without tankage restrictions and deadlines, for each of the network topologies 10 scaling random instances were generated. (Within a network, the instances scaled in terms of the total number of batches and the number of batches with a goal location.) For the instances featuring tankage restrictions or deadlines, the generation process was more complicated because we wanted to make sure to obtain only solvable instances. For the tankage restriction examples, we ran Mips on the respective "notankange" instances, with incrementally growing tankage. We chose each instance at a random point between the first instance solved by Mips, and the maximum needed tankage (enough tankage in each area to accommodate all instance batches). Some instances could not be solved by Mips even when given several days of runtime, and for these we inserted the maximum tankage. For the deadline examples, we ran Mips on the corresponding instances without deadlines, then arranged the deadline for each goal batch at a random point in the interval between the arrival time of the batch in Mips's plan, and the end time of Mips's plan. The instances not solved by Mips were left out.

[3]The same XML file is mapped into different PDDL files depending on the kind of encoding used; there was a lot of trial and error before we came up with the final IPC-4 encoding.

Promela

Promela is the input language of the ACM awarded model checker SPIN (Holzmann 1997). It is designed to ease specification of asynchronous communication protocols, which are to be validated by SPIN for having no specification error. Otherwise the tool returns an error trail as a counterexample. A Promela model consists of a set of processes, and communication between them is performed via message queues or shared access to global variables. Each process can nondeterministically choose one of its transitions that fulfills the condition an optional guard imposes. The IPC-4 Promela domain was created by Stefan Edelkamp.

To allow STRIPS encodings for IPC-4, we selected two simple communication protocols: a solution for the *Dining Philosopher* problem, and the *Optical Telegraph* protocol. Both domains restrict to pure message passing, so that no shared access to global variables is used. The models are distributed together with our experimental model checking tool HSF-SPIN (Edelkamp, Leue, & Lluch-Lafuente 2004), that extends SPIN with heuristic search strategies to improve error detection. In both cases we used one scaling parameter, namely the number of philosophers and the number of control stations, respectively.

In order to generate problem instances fully automatically, we apply a compiler that transforms Promela specifications into PDDL2.2. The compilation process and an exposition for one of the protocols are described in (Edelkamp 2003). The compiler features some but not all static language constructs of Promela. Although not covered by the IPC-4 benchmark set, the work also showed that including communication via global variables and assignments of (not necessarily linear) arithmetic expressions to variables can be expressed in PDDL2.2. Besides deadlocks, violations to assertions and global invariances can also be converted into PDDL2.2 planning goals. For more complex error descriptions, e.g. liveness errors, temporally extended goals are needed. One of the core differences between Promela and PDDL2.2 expressiveness are dynamic processes. An according PDDL model would require a language extension for *dynamic object creation*. Fortunately, the core of most Promela specifications in our own collection is static.

Both protocols are known to contain deadlocks. In the PDDL2.2 descriptions, we utilised the finite state automata representation for the processes and communication queues that is inferred by SPIN. All active Promela processes are typed, enumerated and assigned to a unique object id. Each process consists of local states and transitions, with the queue read and write operations specifically tagged. In the PDDL model, a local state transition is first *activated* before according changes to the state variables or updates to the queue are executed. Finally the state change is *performed*. To ease parsing, state transitions use a reduced ASCII set.

Queues model communication channels, in which messages (and optional data) is written and read by the processes. The main idea in modelling queues is to represent arrays of size k in a ring structure: bucket 0 is the successor of bucket $k-1$ with a head and a tail pointer that are moving. A queue is either empty or full if both pointers refer to the same queue state. As a special case the queues can consist of only one queue state, so the successor bucket of bucket 0 is the bucket itself. In this case the grounded propositional encoding includes operators with add and delete lists that share the same atom, so that we rely on the semantics of STRIPS, saying that deletion is done first.

If the message for reading does not match or the queue capacity is either too small or too large, the according local state transitions will block. If all active transitions in a process block, the process itself will block. If all processes are blocked, we have a deadlock in the system. Detection of a deadlock is crucial and is implemented either as a collection of PDDL2.1 actions or, more elegantly, as a set of PDDL2.2 derived predicates, automatically inferring that all processes for a state transition are blocked.

With each protocol we provide four different domain versions: *plain*, a purely propositional specification with specific actions that have to be applied to fix the deadlock; *fluents* an alternative to the above with numerical state variables that encodes the size of the queues and the messages used to access their contents; *derivedpredicates*, which contains derived predicates to infer deadlocks; and *fluents-derivedpredicates*, which is equivalent to *derivedpredicates* and uses fluents instead of propositions for encoding queue sizes and messages. We use one formulation that uses the ADL constructs *quantification*, *disjunctive* and *negated preconditions*; and one where the same semantics are compiled into pure (propositional) STRIPS. Unfortunately, the larger problem instances of these STRIPS formulations were too big to be stored on disk. We kept *fluent*-domains as separated *versions* instead of different *formulations* to compare pure propositional and numerical exploration efficiencies and to emphasise that numerical state variables are essential for more complex model checking domains.

PSR

The *Power Supply Restoration (PSR)* domain is a PDDL adaptation of an application domain investigated by Thiébaux and others (Thiébaux *et al.* 1996; Thiébaux & Cordier 2001), which deals with reconfiguring a faulty power distribution system to resupply customers affected by the faults. A power distribution system is viewed as a network of electric lines connected by switches and fed via a number of power sources. When a power source feeds a faulty line, the circuit-breaker fitted to this source opens to protect the rest of the network from overloads. This leaves *all* the lines fed by the source without power. The problem consists in planning a sequence of switching operations (opening or closing switches and circuit-breakers) bringing the network into a configuration where non-faulty lines are resupplied.

In the original PSR problem (Thiébaux & Cordier 2001), various numerical parameters such as breakdown costs and power margins need to be optimised, subject to power capacity constraints. Furthermore, the location of the faults and the current network configuration are only partially observable, which leads to a tradeoff between acting to resupply lines and acting to reduce uncertainty. In contrast, the version used for IPC-4 is set up as a pure goal-achievement problem (the goal specifies which lines must be (re)-supplied), numerical aspects are ignored, and total observability is assumed. The choice of leaving out the numerical aspects was motivated by the difficulty of encoding and solving even the basic problem. The IPC-4 PSR domain was developed by Sylvie Thiébaux and Jörg Hoffmann. We benefited from contributions by Piergiorgio Bertoli, Blai Bonet, Alessandro Cimatti, and John Slaney, some of which are reported in (Bertoli *et al.* 2002; Bonet & Thiébaux 2003).

PSR problem instances specify (1) the network topology, i.e., the objects in the network (the lines, the switches, the sources/circuit-breakers), and their connections, (2) the initial configuration, i.e., the initial positions (open/closed) of the switches and circuit-breakers, and (3) the modes (faulty or not) of the various lines. Among those, only the devices' positions can change. A number of other predicates are derived from these basic ones. They model the propagation of the current into the network with a view to determining which lines are currently fed and which sources are *affected* by a fault, i.e. feed a fault. The closed-world assumption semantics of PDDL2.2 derived predicates is exactly what is needed to elegantly encode such relations. These require a recursive traversal of the network paths which is naturally represented as the transitive closure of the connection relation of the network.

The goal in a problem instance asks that given lines be fed and all sources be unaffected.[4] The available actions are closing and opening a switch or a circuit-breaker. In addition, there is an action *wait*, which models the event of circuit-breakers opening when they become affected. Wait is applicable when an affected source exists, and is the only applicable action in that case. The goal and this together ensures that the wait action is applied as soon as a source is affected. The effect of the wait action is to open all the affected circuit-breakers. It would have been possible to encode the opening of affected breakers as a conditional effect of the close action. However, this would have required more

[4]Note that after the circuit-breaker of an affected source opens, this source is not affected any more, as it does not feed any line.

complex derived predicates with an additional device as parameter and a conditional flavor, specifying, e.g., whether or not a circuit-breaker *would be* affected *if* we were to close that device.

We use four domain versions of PSR in IPC-4. Primarily, these versions differ by the size of the problem instances encoded. The instance size determined in what languages we were able to formulate the domain version. We tried to generate instances of size appropriate to evaluate current planners, i.e, we scaled the instances from "push-over for everybody" to "impossibly hard for current automated planners", were we got our intuitions by running a version of FF enhanced to deal with derived predicates. The largest instances are of the kind of size one typically encounters in the real world. More on the instance generation process below. The domain versions are named 1. *large*, 2. *middle*, 3. *middle-compiled*, and 4. *small*. Version 1 has the single formulation *adl-derivedpredicates*. Version 2 has the formulations *adl-derivedpredicates*, *simpleadl-derivedpredicates*, and *strips-derivedpredicates*. Version 3 has the single formulation *adl*, and version 4 has the single formulation *strips*. The formulation names simply give the language used. Version 1 contains the largest instances, versions 2 and 3 contain (the same) medium instances, and version 4 contains the smallest instances. The *adl-derivedpredicates* formulation is inspired from (Bonet & Thiébaux 2003), makes use of derived predicates as explained above, and of ADL constructs in the derived predicate, action, and goal definitions. In the *simpleadl-derivedpredicates* and *strips-derivedpredicates* formulations, all ADL constructs (except conditional effects in the *simpleadl* case) are compiled away using automated software (basically, FF's pre-processor). The resulting encodings are fully grounded and significantly larger than the original, while on the other hand the length of plans remains completely unaffected. The pure *adl* formulation is obtained from the *adl-derivedpredicates* formulation by compiling derived predicates away using the method described in (Thiébaux, Hoffmann, & Nebel 2003). While there is no increase in the domain size, this compilation scheme can lead to an exponential increase in plan length in the worst case. For the PSR instances we generated, we observed only a polynomial blow up. Nevertheless we felt that this increase in plan length was too much to make for a useful direct comparison of data generated for *adl-derivedpredicates* as opposed to *adl*, and we separated the *adl* formulation out into domain version 3 as listed above.

The *strips* domain formulation proved quite a challenge. No matter how hard we tried, compiling both derived predicates and ADL constucts away led to either completely unmanageable domain descriptions or completely unmanageable plans. We therefore adopted a different fully-grounded encoding inspired from (Bertoli *et al.* 2002), which is generated from a description of the problem instance by a tool performing some of the reasoning devoted to the planner under the other domain versions. As a result, the STRIPS encoding is much simpler and only refers to the positions of the devices and not to the lines, faults, or connections. Also we were still only able to formulate comparatively small instances in STRIPS, without a prohibitive blow-up in the encoding size.

The PSR instances were randomly generated using John Slaney's randomnet program. Power distribution networks often have a meshable structure exploited radially: the path taken by the power of each source forms a tree whose nodes are switches and whose arcs are electric lines; terminal switches connect the various trees together. Randomnet takes as input the number of sources, a percentage of faulty lines, and a range of parameters for controling tree depth, branching, and tree adjacency, whose default values are representative of real networks. Randomnet randomly selects a network topology and a set of faulty lines. These are turned into the various PDDL encodings above by a tool called net2pddl,[5] implemented by Piergiorgio Bertoli and Sylvie Thiébaux. The instances we generated make use of randomnet default settings, except for the maximal depth of trees which takes a range of values up to twice the default, leading to harder problems. The percentage of faulty lines ranges from 0.1 to 0.7.

Satellite

The *Satellite* domain was introduced in IPC-3 by Derek Long and Maria Fox (2003). It is motivated by a NASA space application: a number of satellites has to take images of a number of spatial phenomena, obeying constraints such as data storage space and fuel usage. In IPC-3, there were 5 versions of the domain, corresponding to different levels of the language PDDL2.1: *Strips*, *Numeric*, *SimpleTime* (action durations are constants), *Time* (action durations are expressions in static variables), and *Complex* (durations *and* numerics, i.e. the "union" of Numeric and Time).

The adaption of the Satellite domain for IPC-4 was done by Jörg Hoffmann. All IPC-3 domain versions and example instances were re-used, except SimpleTime – like in the other IPC-4 domains, we didn't want to introduce an extra version distinction just for the difference between constant durations and static durations. On top of the IPC-3 versions, 4 new domain versions were added. The idea was to make the domain more realistic by additionally introducing time windows for the sending of the image data to earth, i.e. to antennas that are visible for satellites only during certain periods of time – according to Derek Long, the lack of such time windows was the main shortcoming of the IPC-3 domain.

We extended the IPC-3 Time domain version to two IPC-4 domain versions, *Time-timewindows* and *Time-timewindows-compiled*. We extended the IPC-3 Complex domain version to the two IPC-4 domain versions *Complex-timewindows* and *Complex-timewindows-compiled*. In all cases, we introduced a new action for the sending of data to an antenna. An antenna can receive data of only a single satellite at a time, an antenna is visible for only subsets of the satellites for certain time periods, and the sending of an image takes time proportional to the size of the image.

[5]Randomnet and net2pddl are available from the PSR benchmark resource web page `http://csl.anu.edu.au/thiebaux/benchmarks/pds`, along with various other tools and papers of interest.

The time windows were modelled using timed initial literals, and in the "-compiled" domain versions, these literals were compiled into artificial PDDL constructs. None of the domain versions uses ADL constructs, so of all versions there is only a single (STRIPS) formulation.

The instances were generated as follows. Our objectives were to clearly demonstrate the effect of additional time windows, and to produce solvable instances only. To accomplish the former, we re-used the IPC-3 instances, so that the only difference between, e.g., Time and Time-timewindows, lies in the additional time window constructs. To ensure solvability, we implemented a tool that read the plans produced by one of the IPC-3 participants, and then arranged the time windows so that the input plan was suitable to solve the enriched instance. It is important to note here that the time windows were *not* arranged to exactly meet the times extracted from the IPC-3 plan. Rather, we introduced one time window per each 5 "take-image" actions, made the antenna visible during that time window for only the respective 5 satellites, and let the image sizes be random values within a certain range where the time window was 5 times as long as the sending time resulting from the maximum possible size.

Of course, the above generation process is arranged rather arbitrarily, and the resulting instances might be a long way away from the typical characteristics of the Satellite problem as it occurs in the real world. While this isn't nice, it is the best we could do without inside knowledge of the application domain, and it has the advantage that the enriched instances are solvable, and directly comparable to the IPC-3 ones.

In the new domain versions derived from Complex, we also introduced utilities for the time window inside which an image is sent to earth. For each image, the utility is either the same for all windows, or it decreases monotonically with the start time of the window, or it is random within a certain interval. Each image was put randomly into one of these classes, and the optimisation requirement is to minimise a linear combination of makespan, fuel usage, and summed up negated image utility.

Settlers

The *Settlers* domain was introduced in IPC-3 by Derek Long and Maria Fox (2003). It makes extensive use of numeric variables. These variables carry most of the domain semantics, which is about building up an infrastructure in an unsettled area, involving the building of housing, railway tracks, sawmills, etc. The domain was included into IPC-4 in order to pose a challenge for the numeric planners – the other domains mostly do not make much use of numeric variables, other than computing the (static) durations of actions. We used the exact same domain file and example instances as in IPC-3, except that we removed some universally quantified preconditions to improve accessibility for planners. The quantifiers ranged over domain constants only so they could easily be replaced by conjunctions of atoms.

UMTS

The *UMTS* domain has been developed by Roman Englert (2003). It enables the execution of several (data) applications in mobile terminals. To start an application in a mobile terminal the UMTS call set-up is required. This procedure takes between a couple of seconds for an interactive game like chess and 30 seconds for WAP access. Often users start several applications and as a consequence the waiting period until the call set-ups are executed takes several minutes. Therefore, optimisation of the UMTS call set-up is needed, where each application call is partitioned into modules (Englert 2005). The call set-up via software agents consists of eight discrete modules:

- terminal resource management (*trm*): an application start follows the resource availability check in the mobile terminal and the resource allocation
- connection timing (*ct*): connection set-up duration is monitored in the bearer and in case of failure feedback to the terminal is given (within a certain time, e.g. 1 sec.)
- agent management (*am*) : requirements of mobile applications are transferred to bearer, e.g. Quality of service (QoS), required data volume, ...
- agent execution environment mobile (*aeem*): information about mobile application are sent to *am*, e.g. required servers, ...
- radio resource control (*rrc*): allocation of QoS by logical resources
- radio access bearer: (*rab*) bearer allocation of QoS and in case of failure initiation of resource negotiation with mobile terminal
- agent execution environment internet (*aeei*): data transfer for application set-up from mobile terminal to core network and PDN, and vice versa
- bearer service (*bs*): bearer establishment and feedback to mobile application,

To start the execution of a mobile application the modules are executed in sequential order. If several applications are initiated, some modules can be executed in parallel. The modules obey the following partial execution order: *trm* before *ct*, *ct* before *rrc* and *am*, *am* before *aeem*, *aeem* and *rrc* before *rab*, *rab* before *aeei*, *aeei* before *bs*, with *bs* being final. A detailed documentation on UMTS can be found in (Holma & Toskala 2000).

The PDDL2.2 translation of UMTS was established by Stefan Edelkamp and Roman Englert. Actions were attached to execution time, calling for Level 3 temporal planning. Instances are scaled to setup 1 up to 10 applications, a range that is practically motivated. Compared to other benchmarks, problem and domain description are comparable small to rise a challenge especially for optimal temporal planning approaches. However, real-time is required for practical purposes. Action durations are given in milliseconds and are selected due to practical constraints. The entire benchmark set was completed by running a problem generator that performs a realistic perturbation on the action execution times.

In the form used in IPC-4, the UMTS domain has six versions. The first three are: *temporal*, a domain version with no timing constraints, *temporal-timewindows*, a domain version with PDDL2.2 timed initial facts, and *temporal-timewindows-compiled*, a domain version with a PDDL2.1 wrapper encoding for the timed initial literals. The second domain version set *flaw-temporal*, *flaw-temporal-timewindows*, and *flaw-temporal-timewindows-compiled*, includes an additional but practical motivated *flaw* action that can affect plan finding, since it offers a shortcut to a relaxed plan not needed for a valid one, and, in order to determine that this action is not required, negative interactions have to be computed.

All domain versions have one formulation, namely *strips-fluents-temporal*, where numerical fluents, but - except typing - no ADL constructs are used. In all instances, the plan objective is to minimise *makespan*. The *temporal* and *temporal-timewindow* problem specifications were tested with the MIPS planner (Edelkamp 2004).

Besides action duration, the domain encodes scheduling types of resources, consuming some amount at action initialisation time and releasing the same amount at action ending time. Renewable global resources have not been used in planning benchmarks before, and the good news are that PDDL2.2 is capable of expressing them. In fact we used a similar encoding to the one that we found for *Job-* and *Flow-Shop* problems. As one feature, actions are defined to temporarily produce rather than to temporarily consume resources. As PDDL2.2 has no way of stating such resource constraints explicitly, planners that want to exploit that knowledge have to look for a certain patterns of *increase/decrease* effects to recognise them.

In UMTS, two actions can both check and update the value of some resources (e.g. *has-mobile-cpu*) at their starting (resp. ending) time points as far as the start (resp. ending) events are separated by ϵ time steps, where ϵ is minimum slack time required between two dependent events. We first thought about modelling renewable resources with an *over all* construct. But in this case, the invariant condition of the action has to check, what the *at start* event did change. We decided that this is not the best choice for a proper durative action. Consequently, the durative actions require that there is enough of the resource available *before* adding the amount used.

The domain assumes that the mobile applications run on one mobile terminal. However, they can also be distributed on to several mobile terminals. Additionally, the resource modeling of the UMTS network is constrained to the most important parameters (in total 15). In real networks several hundred parameters are applied.

Concluding Remarks

In a field of research about general reasoning mechanisms, such as AI planning, it is essential to have appropriate benchmarks – benchmarks that reflect possible applications of the developed technology, and that help drive research into new and fruitful directions. In the development of the benchmark domains and instances for IPC-4, the authors have invested significant effort into creating such a set of appropriate benchmarks for AI planning. The domains are mostly still far away from “real-world” problems, and we are aware that, e.g., fully grounded STRIPS encodings aren't nice and pose a serious problem for systems that don't use the standard pre-processes. Nevertheless we believe that the IPC-4 domains constitute a significant step into the right direction, and that they form an interesting range of benchmarks. We hope they will become standard benchmarks in the coming years.

Acknowledgements. We would like to thank the competitors for their detailed comments about found bugs in our domains, and we would like to thank Malte Helmert for various useful tools that helped remove some of these bugs.

References

Bertoli, P.; Cimatti, A.; Slaney, J.; and Thiébaux, S. 2002. Solving power supply restoration problems with planning via symbolic model-checking. In *Proc. 15th European Conference on Artificial Intelligence (ECAI-02)*, 576–580.

Bonet, B., and Thiébaux, S. 2003. Gpt meets psr. In *13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, 102–111.

Edelkamp, S.; Leue, S.; and Lluch-Lafuente, A. 2004. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology*. To appear.

Edelkamp, S. 2003. Promela planning. In *Workshop on Model Checking Software (SPIN)*, Lecture Notes in Computer Science, 197–212. Springer.

Edelkamp, S. 2004. Extended critical paths in temporal planning. In *Proceedings ICAPS-Workshop on Integrating Planning Into Scheduling*.

Englert, R. 2003. Re-scheduling with temporal and operational resources for the mobile execution of dynamic UMTS applications. In *KI-Workshop AI in Planning, Scheduling, Configuration and Design (PUK)*.

Englert, R. 2005. Planning to optimize the umts call set-up for the execution of mobile agents. *Journal of Applied Artificial Intelligence (AAI)*. To appear.

Hatzack, W., and Nebel, B. 2001. The operational traffic control problem: Computational complexity and solutions. In Cesta, A., and Borrajo, D., eds., *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, 49–60. Toledo, Spain: Springer-Verlag.

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, 92–100. Toulouse, France: Morgan Kaufmann.

Holma, H., and Toskala, A. 2000. *WCDMA for UMTS - Radio Access for 3rd Generation Mobile Communications*. Wiley & Sons.

Holzmann, G. J. 1997. The model checker Spin. *IEEE Trans. on Software Engineering* 23(5):279–295. Special issue on Formal Methods in Software Practice.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*. Special issue on the 3rd International Planning Competition, to appear.

Milidiu, R. L.; dos Santos Liporace, F.; and de Lucena, C. J. 2003. Pipesworld: Planning pipeline transportation of petroleum derivatives. In *Proceedings ICAPS-03 Workshop on the Competition*.

Thiébaux, S., and Cordier, M.-O. 2001. Supply restoration in power distribution systems — a benchmark for planning under uncertainty. In *Proc. 6th European Conference on Planning (ECP-01)*, 85–95.

Thiébaux, S.; Cordier, M.-O.; Jehl, O.; and Krivine, J.-P. 1996. Supply restoration in power distribution systems — a case study in integrating model-based diagnosis and repair planning. In *Proc. 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 525–532.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2003. In defense of pddl axioms. In *18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 961–966.