RC23438 (W0411-159) November 17, 2004
Computer Science

IBM Research Report

Stream Processing Planning

G. Ya. Grabarnik, Z. Liu, A. Riabov, J. Bigus
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Stream Processing Planning

G. Ya. Grabarnik, Z. Liu, A. Riabov, and J. Bigus
IBM T.J. Watson Research Center,
19 Skyline Dr., Hawthorne, NY 10532, USA
{genady,riabov,zhenl,bigus}@us.ibm.com

Abstract

In this paper we investigate planning approaches to the practical problem of composing stream processing applications from a set of processing components. The problems of assembling components into an application that achieves a specified goal has become an important research topic. Recently AI planning approaches have been successfully applied in similar applications such as web services, business processes, component-based software composition, automated installation. An important property of stream processing, which not handled by existing models, is the functional dependence between the attributes of incoming and outgoing streams. We propose new planning models taking this property into account, analyze decidability and complexity of the problem, and propose planning methods.

Classification: I.2.8, Plan execution, formation, and generation; G.2.2 Graph Algorithm

Key words and phrases: planning, graphplan, planning heuristic, planning graph, unstructured information, stream processing

Overview

As powerful computers and high-bandwidth communication become increasingly affordable, stimulating the growth in the use of high-performance distributed computing and as modern software development tools and open communication protocols enable large-scale component-based software architectures, the practical problem of making a choice between possible components or services and establishing the interconnections between the components such that the functionality required of the application is achieved and the resource usage is optimized has come to the attention of researchers in several related fields. These composition problems are naturally related to planning, since a sequence of decisions must be made in order to choose and interconnect components into a processing system that satisfies a predefined goal.

In recent work planning methods have been successfully applied to web service composition, automated software installation, and deployment of component-based software (for example see (Doshi & Verma 2004), (Kichkaylo & Karamcheti 2003)). In this work the matching between the entry points of the components is typically implemented by assigning a type identifier to the entry point, and requiring that each connection can only establish a link between entry points having exactly the same type identifiers.

The problem described in this paper arises in stream processing applications. These are the applications, such as video processing, streaming databases or sensor networks, in which the volume of the data being processed is too large to be stored, and therefore the information must be processed on the fly. Processing components in these applications expose one or more input port, and one or more output port. Once each input port of a component is connected to a stream, the component produces an output stream for each of the output ports, by filtering, annotating, or otherwise analyzing and transforming the information it receives. Once a stream is created any number of components can receive data from it by connecting an input port to the stream. However, no more than one stream can be connected to one input port.

The streams that come into the system from the outside world are referred to as primal streams. The components do not make any distinction between the primal and derived streams, and can work with either kind, given that the precondition for the input port is satisfied. The goal of this processing can be formulated as a requirement on the output stream.

In the stream processing scenario, the streams cannot be described by a single type identifier. Instead, a stream carries a set of properties (tags). The goal requirements for output streams and the input requirements of processing components can be described as the subset of tags that are required to be present in the stream description. The tags that are as-

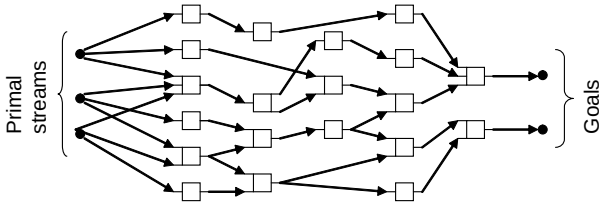

Figure 1: Example of a stream processing application graph.

signed to the input streams of a processing component are transformed into the tags of output streams according to a mapping function defined for the component.

During our attempts to map this problem into 'standard' (STRIPS or PDDL) formulation we encountered two difficulties.

- Tag set corresponding to each stream does not map 'nicely' to the 'standard' formulation - one needs significantly larger number of variables (or predicates) to cover the notion of grouping tags by stream.
- Output streams are independent from each other, and are dependent on (possibly) each input. This is different from say PDDL that expects output propositions to be connected with AND relation.

In this paper we introduce a general formulation for the stream processing planning problem. We also consider a restricted formulation, which can be seen as an extension of STRIPS model. This formulation scales better that the general one, and still captures the most important properties of the model. These models are applicable in any environment where components interact by producing and consuming data, and can be used to model the composition of web services or software components. Finally, we propose an algorithm for finding optimal plans, and discuss future research directions.

Related Work

One of the first strict formulation of automated planning was done by (Green 1969). STRIPS schema was introduced in the (Fikes & Nilsson 1971). Strict formulation of the STRIPS may be found in (Nilsson 1980). UCPOP planner (Penberthy & Weld 1992) worked with some modification of STRIPS.

Semantics of STRIPS planning was investigated by (Lifschitz 1986). Framework for studying of actions presentation is done in (Sandewall 1995).

ADL introduced in (Pednault 1986) shows nice balance between expressiveness of language and computational complexity of the reasoning. Usage of state variables directly was introduced in (Bäckström 1991) (planner SAS).

Equivalency of *expressivity* of different planning formalisms was considered in (Nebel 2000). Two presentations are called equivalent if there is a polynomial transform mapping one presentation into another such that plan size are preserve precisely.

Complexity of the propositional planning considered in (Bylander 1992) and (Erol & Subramanian 1995). Complexity of the state space planning considered in (Bäckström 1991) etc.

PDDL as a computer parsable language representing STRIPS, ADL etc., was introduced in (Ghallab & McDermott 1998).

Resource base restrictions considered in the (Koehler 1998).

General branch-and-bound algorithm was described in (Nau & Kanal 1984)(see also (Pearl 1985)). Distance-based heuristics considered in (Ghallab & Laruelle 1994). Systematic analysis of distance based heuristics provided in (Bonet & Geffner 1999). Fast-Forward (Hoffman 2001) planner refined heuristic estimates used in HSP.

Model checking technique was first considered in (Kabanza & St-Denis 1997). Further the idea used in the (Jensen & Bryant 2003), where model checking was reinforced by usage of BDD based local search. Further, BDD was used for the reduction of the state space size using elaborate technique for translating PDDL into BDD format, see (Edelkamp & Helmert 1999).

More detailed information about automated planning may be found in the monographs (Ghallab & Traverso 2004) and (Russel & Norvig 2002).

For Workflow planning of the webservices see workshop part of ICAPS04(03): (Pla 2003) or (Pla 2004)

General Model

We start with a strict description of generic Stream Processing Planning. We show that it is more expressible than an associated STRIPS model (see (Nebel 2000)). Then we show that complexity of the planning for such model changes between PSPACE and EXPSPACE. For the state variable formalism see (Jonsson & Bäckström 1998).

Types and Constants Types represent a finite tree based on the inheritance relation. Only single (not multiple) inheritance is allowed. Type $object$ is a root type for all types.

```
(:types person city address - object,
                        s - stream)
```

Constants may be of certain types. There are only finitely many constants of a specific type.

```
(:constants ernie scott dan - person)
```

Streams and Variables A Stream is special type of object. It corresponds to the set of the constants of types. To mark that stream s contains constant o we use predicate

```
has_object ( s, o) or (s and (o))
```

In the state variables presentation of the world, streams correspond to an additional grouping of the state variables. We consider that all the variables are streams. For convenience we think of streams as partially grounded complete lists with enumeration corresponding to types (or we can add a new constant ϵ that would mean that a value is not specified or not important, see for example (Jonsson & Bäckström 1998). Let s_1 be a stream consisting from constants $o_1, o_2, ..., o_l$. Stream s_2 extends stream s_1 if the stream s_2 contains at least l constants $o_1, o_2, ..., o_l$ on proper type positions. Stream s_g is grounded extension of s_1 if all possible types-placeholders in s_g are set to certain values and s_g extends s_1.

Relations and Actions Relations are the fixed relations on the streams that do not depend on the change of the streams. Actions have preconditions and effects (for action A we denote them by precond(A) and effects(A)) . Both preconditions and effects are expressed in terms of streams. Each precondition is a set of expressions on a stream and relations. We consider that preconditions are independent from each other (express precondition on different streams). Each effect is a set of assignments of values to the stream depending on preconditions. Assignments may depend in principal on certain boolean expression. We separate the following cases: simple assignment, conditional assignment ($when\ E\ then\ S$), and boolean expression assignment containing more complicated boolean expression. Effects are independent from each other (connected with OR expression).

```
Sample assignment action
(:action AA
  :parameters (?in1 ?in2 ?in3 ?out1 ?out2)
  :precondition (in1 and (P1)(P2))
  :precondition (in2 and (P3)(P2))
  :precondition (in3 and (P5)(P6)(P7))
  :effect (out1 and (P4)(P6))
  :effect (out2 and (P3)(not (P2)))
  )
Sample conditional action
(:action CA
  :parameters (?in1 ?in2 ?in3 ?out1 ?out2)
  :precondition (in1 and (P1)(P2))
  :precondition (in2 and (P3)(P2))
  :precondition (in3 and (P5)(P6)(P7))
  :effect (out1 when in1.P4 then (P6))
  :effect (out2 when in3.P2
                then ( and (P3) (P5) )
  )
```

Note that the difference with state-variable presentation is that streams contain a complete set of types, and that actions have multiple independent (different streams) preconditions and multiple independent effects (ORed) (see (Jonsson & Bäckström 1998)). Since effects are independent from each other, actions may be decomposed on set of actions with one stream output. However dependency of each effect on set of inputs is a strong differentiator.

Planning language $\mathcal{L}$ in the stream representation is a set of the stream variables $\mathcal{S}$, a finite set of fixed relations $\mathcal{R}$ and sets of constants that define their streams. Define set X of all partially grounded streams such that there exists grounded extension x_g of $x \in X$ satisfying fixed relations R.

Planning domain in language $\mathcal{L}$ is a state transition system $\Sigma(\mathcal{S}, \mathcal{A}, \gamma)$ satisfying

- $\mathcal{S} \subset \Pi_{x \in X} D_x$, where D_x is a range of the partially grounded stream variable x. In this case state s defined as $s = \{(x = c \mid x \in X\}$ for $c \in D_x$.
- $\mathcal{A} = \{$ all partially grounded instances of actions that meet relations in $\mathcal{R}\}$. We say that action A is applicable to the stream set $\{s_1, ..., s_k\}$ if action A has k preconditions $\{in_1, ..., in_k\}$ and stream s_i extends precondition in_i for $i = 1, ..., k$.
- $\gamma(s, A) = \{(x = c) \mid x \in X\}$ where c is specified by assignment $c \mapsto x$ in effects (A).
- $\mathcal{S}$ is closed under γ, meaning for $s \in \mathcal{S}$ and every action A applicable to s one has $\gamma(s, A) \in \mathcal{S}$.

Planning goal is a triple $\mathcal{P} = (\Sigma, s_0, g)$ where s_0 is a set of grounded streams representing initial state in $\mathcal{S}$ and the goal is a set of expressions on the stream variables.

Planning problem is a 4-tuple $P = (\mathcal{A}, \mathcal{R}, s_0, g)$ where $\mathcal{A}$ is a set of actions, $\mathcal{R}$ is a set of fixed relations, s_0 is the initial state, and g is a goal.

Decidability of the planning problem (see (Erol & Subramanian 1995)). Let $P = (\mathcal{A}, \mathcal{R}, s_0, g)$ be a stream processing planning problem. Suppose that solution for the planning problem exists and has length l. First, we ground all possible actions in such a way that they still satisfy relations $\mathcal{R}$. Next, we can enumerate all admissible paths of length l: first enumerate all admissible paths of length 1, then, given enumeration of length $m-1$, we can enumerate all admissible paths of length m, by adding to each path every admissible grounded action. Suggestion of existing of the planning solution implies that solution maybe found. This means that stream planning problem is at most semidecidable. It also implies that finding length of the plan is decidable problem. Semidecidability follows from the fact that this model covers classical planning, and the fact that classical planing is semidecidable (see Corollary 3.1 and Theorem 3.3 in the reference above).

Complexity of the planning problem. Set of total admissible grounded streams is not more than exponential in terms of input sequence. From initial state we choose non deterministically next action and apply it. We repeat procedure until we reach the goal. This solves problem in NEXPSPACE. Since NEXPSPACE and EXPSPACE are equal, it implies that planning problem is at most EXPSPACE. Since model covers classical planning, this fact and Theorem 5.7 of (Erol & Subramanian 1995) implies that stream processing planning problem is EXPSPACE hard.

Simplified Model

Our current implementation of stream processing planner supports a simplified planning domain description for stream processing, which can be seen as a direct derivative of STRIPS, with the addition of stream processing logic. The simplified model is more tractable than the general one, since the conditional effects allowed by this model have well defined structure. Yet, even with this simplification, the model can capture the most important practical applications.

In this section we briefly describe a standard model for describing planning domains, STRIPS, show how it can be extended for stream processing planning and explain why the extension is necessary, describe our approach to modeling resource constraints, provide a mathematical formulation for the problem, and describe an algorithm that we have implemented for finding admissible plans with minimum resource utilization.

Propositional STRIPS Formulation

In the propositional STRIPS planning problem (Fikes & Nilsson 1971), the state of the world is described by a fixed number of predicates (boolean variables). The initial set of values for these variables is given. A number of possible actions is defined, and for each action the precondition and the effect are specified. Action precondition is described by a list of predicates, which must be present in the state of the world (or, equivalently, must evaluate to true). The precondition in STRIPS corresponds to the logical AND operation on the values of the chosen predicates. An action is said to be legal in particular state of the world if the precondition of the action is satisfied. Effect of an action is described by the two lists of predicates: add list and remove list.

When a legal action is executed, the state of the world is updated: the predicates specified in the add list are set to true, and those present in the remove list to false. The planner must construct a legal sequence of actions, which starting from the inital values of predicates, modifies the state such that after applying these actions in a sequence, all of the predicates specified in the goal list become true.

To illustrate this simple model, below we include a PDDL description of an action named A that has predicates P1 and P2 in its precondition, and sets predicate P3 and removes P2 as an effect. PDDL is a standard language for describing planning domains.

```
(:action A
  :precondition (and (P1)(P2))
  :effect (and (P3)(not (P2))) )
```

Predicate STRIPS Formulation

In the predicate STRIPS class of formulations, parameteric actions can be described: the predicates used in preconditions and effects can have parameters. These parameters are variables of an enumeration type, and can take one of a finite set of values, and the action is assumed to be defined for each of the possible instantiations of parameters. Goal and inital state lists in this formulation can only contain ground predicates.

Below is an example of action definition in a predicate STRIPS domain.

```
(:action A
  :parameters (?x ?y)
  :precondition (and (P1 ?x)(P2 ?y))
  :effect (and (P3 ?x ?y)(not (P2 ?x))))
```

Multiple Streams

We noticed that the traditional STRIPS assumes that an action is applied to the global state of the world, described by a multidimensional vector. In contrast with this assumption, in the scenario of distributed stream processing, each of the processing components works only with the data it receives on incoming streams. Furthermore, descriptions of all streams have the same structure: stream descriptor for each stream can be seen as an instance of a common stream properties class.

We needed a structure for describing properties of streams, and also a language for describing input requirements, as well as modifications to these properties performed by stream processing components. We have chosen STRIPS-like language of predicates, preconditions and add/remove effect lists. Although STRIPS has its limitations, in particular the lack of expressivity in describing preconditions and effects, it has become a de-facto standard domain formulation for deterministic planners. Our decision was motivated mainly by the popularity of the language, and by additional search difficulties associated with more expressive planning domain classes.

Since each processing component can receive more than one input stream, and produce more than one output stream, for each of these streams preconditions and effects must be specified. We will say that the action has multiple input and output ports, to which the streams can be connected. Cor-

respondingly, action description must be extended as in the following example:

```
(:action A
  :precondition (and (P1)(P2))
  :precondition (and (P3)(P2))
  :precondition (and (P5)(P6)(P7))
  :effect (and (P4)(P6))
  :effect (and (P3)(not (P2))) )
```

Application of each action creates new streams. The number of streams created is equal to the number of effects specified in the action description. Properties of output streams depend both on the effects of the action and on the properties of incoming streams. The stream created when an action is applied is never modified or removed from the world state. Each stream can be used as input in multiple actions, and each application of an action creates new streams with new properties, and does not change properties of any streams that existed before the action was applied.

Relationship to STRIPS

In the simplest case, when the properties of output streams are specified explicitly in action description, and are independent of properties of input streams, the following procedure can be used to convert the stream processing planning problem to STRIPS:

1. Declare type `stream`; each object of this type will correspond to a name of an unique stream. The number of streams, and therefore of objects of type `stream` that are needed, in the worst case is exponential in the number of predicates used in the original problem formulation. However, this number is also limited by the number of streams used in the solution, and therefore an estimated upper bound (based on available resources and minimum resource cost of action, for example) can be used.
2. Declare predicate `unused(?s - stream)`. We will use this predicate to indicate that a stream with the name `?s` has not yet been created.
3. Predicate `hasproperty(?s - stream ?p - property)` will indicate that property `?p` is present in stream `?s`. `property` is a type that contains names of all possible predicates of the original problem.
4. For each object of class *stream* in initial state, except to the objects corresponding to initial (primal) streams, define `unused(?s)` in the initial state. For the initial streams, define `hasproperty(?s ?p)` accordingly.
5. Goal stream specification translates to a set of `hasproperty(?s ?p)` goals, where `?s` has value corresponding to a chosen goal stream name, and `?p` is property predicate required in the goal.

The description of the action given in previous example translated to STRIPS will be the following:

```
(:action A
  :parameters (?in1 ?in2 ?in3 ?out1 ?out2)
  :precondition (and
     (unused ?out1)
     (unused ?out2)
     (hasproperty ?in1 P1)
     (hasproperty ?in1 P2)
     (hasproperty ?in2 P3)
     (hasproperty ?in2 P2)
     (hasproperty ?in3 P5)
     (hasproperty ?in3 P6)
     (hasproperty ?in3 P7) )
  :effect (and
    (not (unused ?out1))
    (not (unused ?out2))
    (hasproperty ?out1 P4)
    (hasproperty ?out1 P6)
    (hasproperty ?out2 P3) )
```

This compilation schema can be easily extended for actions with parameters by listing the parameters in `hasproperty` predicate after property name.

One may notice that this translation does not take into account `(not (P2))` effect of the stream action. This is because it assumes that none of input properties are propagated to the output streams, and therefore negating a predicate does not have any effect: the property will not be present in the output even without this applying negation, unless it is specified in the add list.

This restriction does not allow STRIPS model to satisfy our requirements: we need to extend effect operations and allow output values of predicates corresponding to new streams to be functions of input predicates on different streams. In general this is equivalent to allowing conditional effects, which is a harder problem than traditional STRIPS, as shown in (Nebel 2000).

Predicate Merging: AND, OR, FALSE.

Extending the STRIPS model to single input, multiple output case is straightforward: it is sufficient to specify the add and remove lists for each of the outgoing streams. However, the formulas that compute the properties of the outgoing links based on the properties of more than one input stream require a language that is more expressive than STRIPS.

To alleviate the search complexity associated with allowing general Boolean formulas for computing action effects, and based on evaluation of stream planning problems that arise in practice, we have made the following simplifying assumptions:

1. The value of a predicate `x` in the output stream depends only on the values of the same predicate in the input

streams, and on the effects (add/remove lists) associated with the output port of an action.

2. The operation used to combine the values of the predicate x associated with each of the input streams to compute the value to which effects are applied, depends only on the predicate and not on the operator, and can be one of the following choices: logical AND of x values in all inputs, logical OR of x values in all inputs, and the constant FALSE, used for computing output value independent of input values.

The predicates are divided into three groups, according to the default stream merging operation to be used on the predicates. The AND-logic group contains predicates that can be required by a constraint (such as precondition or goal expression) to be present on all streams that lead to satisfying the constraint. Predicates in the OR-logic group are used for describing stream properties that can satisfy the constraint as long as they are present on one of the input streams. Predicates in FALSE group can be used to implement simple input-to-output matching in cases when the output of the action can be specified independently of the input. It is important to note that this independence of output from input values during effect computation of course does not remove the requirement that the input values must satisfy all preconditions of corresponding input ports for the action to be legal.

Resource Constraint Model

Many researchers have considered planning domains where a resource metric is used as a minimization objective during construction of the plan. For example, the planning algorithm described in (Kichkaylo & Karamcheti 2003) considers the characteristics of the underlying hardware network, on which the software stream processing components are deployed and make deployment decisions based on resource availability in the network. Due to the anticipated large number of processing components and the need for more expressivity in specifying action effects in our model we decided to separate the planning and resource allocation problems, and solve them separately.

In the planning problem, which is the subject of this paper, we assign a constant cost $c_i \geq 0$ to each action i, and the minimization objective is the sum of cost of all actions used in the plan. This resource model roughly corresponds to scheduling all processing components on the same processor and minimizing utilization of the limited processor resource. Alternatively, it can be seen as minimization of the cost associated with implementing the plan, given the cost of performing each action.

Mathematical Formulation

In this subsection we introduce the notation used for describing the planning problem, and formally define actions, effects and preconditions for the propositional modification of the problem. The planning problem is defined by the following parameters:

- World state $\mathcal{S}$ is a set of streams. Each stream in $\mathcal{S}$ is represented by a Boolean vector of predicates (stream properties) of dimension n: $\mathbf{x} \in \{0,1\}^n$.
- Each property vector $\mathbf{x} \in \{0,1\}^n$ can be represented as a combination of sub-vectors corresponding to three predicate groups: AND, OR, and FALSE; $\mathbf{x} = (\mathbf{x}_\wedge, \mathbf{x}_\vee, \mathbf{x}_F) \in \{0,1\}^{n_\vee} \times \{0,1\}^{n_\wedge} \times \{0,1\}^{n_F}$, where $n_\vee + n_\wedge + n_F = n$.
- Initial world state $\mathcal{S}^0 = \{\mathbf{x}_i\}_{i=1}^I$ is a set of primal streams (streams available before any actions take place).
- $\mathcal{A} = \{A_i\}_{i=1}^m$ is the set of all actions. Each action A_i has J_i input ports and K_i output ports, $i = 1..m$. Let $K = \max\{K_i\}_{i=1}^m$ and $J = \max\{J_i\}_{i=1}^m$.
- $c_i \in I\!R_+$ is the cost of action A_i; $i{=}1..m$.
- $\mathbf{p}^{ij} \in \{0,1\}^n$ is the precondition corresponding to the input port j of action A_i; $i{=}1..m, j{=}1..J_i$.
- $\mathbf{s}^{ik} \in \{0,1\}^n$ is the add list vector corresponding to the output port k of action A_i; $i{=}1..m, k{=}1..K_i$.
- $\mathbf{r}^{ik} \in \{0,1\}^n$ is the remove list vector corresponding to the output port k of action A_i; $i = 1..m, k = 1..K_i$. We assume that $\mathbf{r}^{ik} \wedge \mathbf{s}^{ik} = \mathbf{0}$ for all i and k.
- $\mathcal{G} = \{\mathbf{g}^i\}_{i=1}^G$, $\mathbf{g}^i \in \{0,1\}^n$ – the set of goal vectors.

Action A_i, $i = 1..m$ is legal in state $\mathcal{S}$ if for each input precondition $\mathbf{p}^{ij}$, $j = 1..P_i$, there exists $\mathbf{x} \in \mathcal{S}$ such that $\mathbf{x} \geq \mathbf{p}^{ij}$. We will say that $\mathbf{x}$ is a matching stream for $\mathbf{p}^{ij}$.

Let A_i be a legal action in state $\mathcal{S}$, let $\{\mathbf{x}^{ij}\}_{j=1}^{J_i}$ be the streams from $\mathcal{S}$ that match the preconditions[1] $\{\mathbf{p}^{ij}\}_{j=1}^{J_i}$. Action A_i changes the state of the world from $\mathcal{S}$ to $\widehat{\mathcal{S}}$, where $\widehat{\mathcal{S}} = \mathcal{S} \cup \{\widehat{\mathbf{x}}^{ik}\}_{k=1}^{K_i}$. In this new state vectors $\{\widehat{\mathbf{x}}^{ik}\}_{k=1}^{K_i}$ are the property vectors of the newly created output streams, which are computed as following:

$$\widehat{\mathbf{x}}^{ik} = \begin{pmatrix} \bigwedge_{j=1}^{J_i} \mathbf{x}_\wedge^{ij} \\ \bigvee_{j=1}^{J_i} \mathbf{x}_\vee^{ij} \\ \mathbf{0} \end{pmatrix} \wedge \neg\mathbf{r} \vee \mathbf{s}$$

[1] Streams $\mathbf{x}^{ij}$ are not necessarily unique: it can happen that the same stream matches more than one input precondition of A_i.

The problem is to construct an optimal plan - - a sequence of legal actions that, when applied to the initial state $\mathcal{S}^0$ generates a state $\mathcal{S}^*$, in which for every $\mathbf{g}^i \in \mathcal{G}$ there exists $\mathbf{x}^* \in \mathcal{S}^*$ such that $\mathbf{x}^* \geq \mathbf{g}^i$, and such that the cost of this plan, defined as the total cost of all actions in the plan, is the minimum among all feasible (legal) plans.

Problem Complexity

We list the following simple complexity results for stream planning problem without proof. If $K = J = 1$, and $n_F = 0$, the problem of finding a legal plan becomes equivalent to finding a legal plan in STRIPS domain, which is known to be PSPACE (Bylander 1992), and therefore the problems of finding the optimal or feasible plan are at least as hard.

If, in addition to $K = J = 1$, it holds that $n = n_F$ or $\mathbf{r}^{ij} \equiv \mathbf{1}$, both problems of fining a feasible and an optimal plan are polynomially solvable.

If $K \geq 1, J \geq 1$ and $n = n_F$ or $\mathbf{r}^{ij} \equiv \mathbf{1}$, the problem of finding a feasible plan is polynomially solvable, however the optimal plan problem is NP-hard (for the proof see Appendix).

The expressivity of our model, as defined in (Nebel 2000), is at least that of STRIPS and less than that of planning models with general conditional effects, since for any STRIPS problem a corresponding formulation for stream planning can be constructed, and since the values of the output predicates are computed using restricted merging formulas.

The advantage of using stream planning formulation directly, instead of constructing a STRIPS formulation first, solving it, and later converting the solution back to streams and stream processing components, lies in the added efficiency that the search algorithm can gain from the additional structure present and explicitly specified in the stream planning formulation. There are two options for conversion: to use variables as stream names or to identify streams by their content.

The approach of using variables to identify streams will cause the solver that optimizes resources to generate at least $O(N!)$ solutions for each single solution of length N that the stream planning solver will consider, due to the fact that the names can be assigned differently, and the general STRIPS solver will not detect that the solutions generated are the same.

The approach of creating a variable for each stream requires the state space to define 2^n variables for streams, and $O(2^n m)$ actions, since effects of the action can vary depending on the set of input streams, and $O(2^n)$ streams can potentially satisfy a precondition. Therefore, in the worst case the STRIPS solver will consider $O(2^n m)$ alternatives for making a transition to next state, in cases where the stream planning solver will only need to investigate $O(m)$ alternatives.

Search Algorithm

We have implemented a general branch-and-bound procedure for solving the problem, that allows us to experiment with different search methods. Branch and bound is a standard approach to solving combinatorial problems, and it have been shown to be a successful solution method for planning problems (cite Hoffman Geffner 2003).

Currently, the backward search (from the goal) is implemented: at each branching node a goal is chosen from the set of available nodes, and is connected to an existing primal or derived stream or to a newly placed action. If an action is placed, the input ports of the action are registered as new goals to be satisfied at the next step. The preconditions of the action in combination with the constraints on the output of the action are used to specify the new goal constraint for each of the inputs. The search tree is pruned if the best achieved total cost is exceeded.

The algorithm allows predicates and actions with parameters in the input. The parameters are substituted before the search.

The algorithm gains additional efficiency from precomputing pairs of commuting actions and considering only one of the two possible orderings in the pair, therefore achieving the same effect as GraphPlan (Blum & Furst 1995) does in allowing the commuting actions to be executing in parallel, extending this approach to the more general stream planning scenario. Potential conflicts (mutexes) are precomputed at the same time, and for each input port of each action a list of output ports that can be connected to it are constructed. These lists are reduced during branching according to the revised goals.

Detecting Commutative Actions

For two actions $A_u, A_v \in \mathcal{A}$ we denote a composition of actions as $A_u \triangleright_{(kj)} A_v$. The composition is defined with respect to the output port k of action A_u, $1 \leq k \leq K_u$, and the input port j of action A_v, $1 \leq j \leq J_v$, and corresponds to the aggregate action obtained by connecting the stream produced by the output port of action A_u to the input port of action A_v. The composition is legal if $\mathbf{r}^{uk} \wedge \mathbf{p}^{vj} = \mathbf{0}$ and $\mathbf{s}_F^{uk} \geq \mathbf{p}_F^{vj}$, i.e. if the none of the predicates listed in the precondition of the input port are explicitly removed by the remove list associated with the connected ouput port, and all of the preconditions in the FALSE group are satisfied by the corresponding add list.

In general, the composition cannot be represented in the same format as action, i.e. by a set of input ports with preconditions and a set of output ports with add/remove lists. The planning algorithm can precompute for each action A_u with selected input port j_u and output port k_u

and each action A_v with selected input port j_v and output port k_v whether the two compositions $A_u \triangleright_{(k_u j_v)} A_v$ and $A_v \triangleright_{(k_v j_u)} A_u$ define equivalent aggregates, and only consider search subtrees in which A_u is connected to A_v, but not A_v to A_u, where the connections are made via the corresponding ports.

Conclusion and Future directions

In this paper we described a real-world planning problem requiring that we define a sequence of processors over multiple data streams. We built the model describing the problem and provided initial analysis of the decidability and complexity of the model. Next, we simplified and analyzed the model, showing that the simplified model is more practical to solve. We outlined our planning algorithm and some technical specifics we used to solve the problem.

As the next steps we are going to extend our simplified model to handle more advanced resource constraints, described by monotonic function on the action composition.

We also plan to add model checking-based planning to the possible solvers, since our problem is close to the models that are handled by model checking well.

References

Bäckström, C. 1991. Planning in polynomial time: the SAS-PUB class. *Computational Intelligence* 7:181–197.

Blum, A. L., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1636.1642. Montreal.: Morgan Kaufmann.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, 360–372. Durham, UK.: Springer-Verlag.

Bylander, T. 1992. Complexity results for serial decomposability. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 729–734. San Jose.: AAAI Press.

Cook, S. A., and Mitchell, D. G. 1997. Finding hard instances of the satisfiability problem: A survey. In D. Du, J. G., and M.Pardalos, P., eds., *Satisfiability Problem: Theory and Applications*, 1–17. Piscataway, NJ: Amer. Math. Soc.

Doshi, P. Goodwin, R. A. R., and Verma, K. 2004. Dynamic workflow composition using markov decision processes. In *Proceedings of IEEE Second International Conference on Web Services*.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In Buindo, S. Fox, M., ed., *Proceedings of the European Conference on Planning (ECP)*, volume 1809 of *LNAI*, 135–147. Springer-Verlag.

Erol, K. Nau, D., and Subramanian, V. 1995. Complexity, deciddability and undecidability results for domain independant planning. *Artificial Intelligence* 76(1-2):75–88.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 61–67. Chicago: AAAI Press.

Ghallab, M. Howe, A. K. C. A., and McDermott, D. 1998. PDDL. the planning domain definition language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control.

Ghallab, M. Nau, D., and Traverso, P. 2004. *Automated Planning*. Sun Francisco: Morgan Kaufman.

Green, C. 1969. Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*. Washington, DC.: IJCAII.

Hoffman, J. 2001. FF: The fast forward planning system. *AI Magazine* 22(3):57–62.

Jensen, R. Veloso, M., and Bryant, R. 2003. Guided symbolic universal planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*. Trento: AAAI Press.

Jonsson, P., and Bäckström, C. 1998. State variable planning under structured restrictions: Algorithm and complexity. *Artificial Intelligence* 100(1-2):125–176.

Kabanza, F. Barbeau, M., and St-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Kichkaylo, T. Ivan, A., and Karamcheti, V. 2003. Constrained component deployment in wide-area networks using ai planning techniques. In *IPDPS*.

Koehler, J. 1998. Planning under resource constraints. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 489–493.

Lifschitz, V. 1986. On the semantics of STRIPS. In Georgeff, M. P., and Lansky, A. L., eds., *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, 1–9. Timberline, Oregon.: Morgan Kaufmann.

Nau, D.S. Kumar, V., and Kanal, L. 1984. General branch and bound, and its relation to A^* and AO^*. *Artificial Intelligence* 23(1):29–58.

Nebel, B. 2000. On the compilability and expressive power

of propositional planning formalisms. *Journal of AI Research* 12:271–315.

Nilsson, J. 1980. *Principles of Artificial Intelligence*. Tioga Publishing.

Pearl, J. 1985. *Intelligent Search Strategiesfor Computer Problem Solving.* Addison-Wesley.

Pednault, E. P. D. 1986. Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff, M. P., and Lansky, A. L., eds., *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, 47.82. Timberline, Oregon: Morgan Kaufmann.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP:a sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, 103–114. Morgan Kaufmann.

2003. Workshop on planning of web services, http://www.isi.edu/info-agents/workshops/icaps2003-p4ws/.

2004. Workshop on planning of web services and grid, http://www.isi.edu/ikcap/icaps04-workshop/.

Russel, S., and Norvig, P. 2002. *Artificial Intelligence: Modern Approach.* Prentice Hall, 2nd edition.

Sandewall, E. 1995. *Features and fluents*. Oxford University Press.

Appendix. Resource Minimization Complexity

Consider the stream processing planning problem in the case when $n = n_F$ or $\mathbf{r}^{ij} \equiv \mathbf{1}$ for all output ports j of all actions i. If either of these condition holds, the set of output predicates of an action does not depend on the input predicates, once the precondition for applying the action is satisfied.

It is easy to solve the problem of finding a legal plan under these conditions. In every feasible plan each action will be used no more than once, since output streams produced by the action are independent of input streams, and therefore are always exactly the same every time the action is applied. Therefore a simple algorithm that proceeds in iterations and at each iteration applies a legal action that has not been applied yet, terminates after at most m iterations when it can no longer find an action to apply. This algorithm either constructs a legal plan or proves that one does not exist.

In what follows we formally prove that under these restrictions the problem of finding an optimal plan (a plan that has minimum resource usage among all legal plans) is NP-hard. To prove this, we formulate an equivalent problem in which exact matching between the types produced by output ports and the requirements on input ports is required.

Color Planning Problem

Suppose we are given a set of operators $\mathcal{O}$, and a set of colors $\mathcal{C}$. Operator $O = \langle O^{in}, O^{out} \rangle$ can be applied only if all colors in the set O^{in} are defined. If operator O is applied, it adds all colors in the set O^{out} to the set of available colors. We assume that for any operator O, $|O^{out}| \geq 1$ and $|O^{in}| \geq 1$. In the multi-in and multi-out variant of the problem, there exist two operators $O_1, O_2 \in \mathcal{O}$ such that $|O_1^{in}| \geq 2$ and $|O_2^{out}| \geq 2$ (it is possible that $O_1 = O_2$).

It is easy to see that this problem is equivalent to the stream processing planning problem, if we map each action to an operator, each $\mathbf{p}^{ij}$ to a distinct color that is required by the operator, and define the set of colors produced by the action as the set of preconditions that can be satisfied by the add list vectors corresponding to all output ports of this action.

Without loss of generality we can assume that for any operator O it holds that $O^{out} \setminus O^{in} \neq \emptyset$. In other words, each operator adds at least one color to the input set – otherwise that operator has no effect.

The color planning problem then is: given a set of input colors D_{in}, and set of output colors D_{out} find a composition of operators that, after being applied to colors in D_{in}, produces all colors in D_{out}. It is easy to show that this problem can be solved in at most $O(|\mathcal{O}|^2)$ steps, by simply searching for an applicable operator, applying it, and repeating the search in the reduced set of operators. This algorithm either finds a solution, or provides a proof that no solutions exist.

In the optimal color planning problem each operator O has weight $w(O) \geq 0$. The problem is to find a composition of operators that solves the color planning problem and has minimum weight among all solutions.

Satisfiability Problem

We will show that given an instance of NP-complete SATISFIABILITY problem (SAT, see (Cook & Mitchell 1997) for details) we can construct an instance of optimal color planning that will satisfy multi-in single-out condition, and the optimal solution to which can be used to construct a solution for SAT. All transformations involve polynomially many steps and size of the planning problem is polynomial in the size of SAT instance.

Given a set of Boolean variables $\{x_1, x_2, ..., x_n\}$, satisfiability problem is to find values for these variables that makes "true" a given Boolean formula

$$F(x) = \bigwedge_{i=1}^{q} \left[\left(\bigvee_{j=1}^{m_i} x_{I(i,j)} \right) \vee \left(\bigvee_{j=1}^{p_i} \bar{x}_{J(i,j)} \right) \right] \quad (1)$$

(written in conjunctive normal form).

Reduction

Suppose we are given an instance of SAT: variables $\{x_1, x_2, ..., x_n\}$ and formula (1). First, we will define $D_{in} = \{C_S\}$, $D_{out} = \{C_E\}$ for some unique colors C_S and C_E.

Next, for each variable x_k, $k = 1..n$ define three colors, C_k^T, C_k^F and U_k, and four operators:

$$O_k^T : \{C_S\} \to \{C_k^T\},\ w(O_k^T) = 1$$

$$O_k^F : \{C_S\} \to \{C_k^F\},\ w(O_k^F) = 1$$

$$O_k^{UT} : \{C_k^T\} \to \{U_k\},\ w(O_k^{UT}) = 0$$

$$O_k^{UF} : \{C_k^F\} \to \{U_k\},\ w(O_k^{UF}) = 0$$

For each multiplicative term i of F, $i = 1..q$, define color T_i.

For each additive variable appearance in F without negation $x_{I(i,j)}$, $i = 1..q$ and $j = 1..m_i$, define operator $O_{ij}^{VT} : \{C_{I(i,j)}^T\} \to \{T_i\}$. For each additive variable appearance in F with negation $\bar{x}_{J(i,j)}$, $i = 1..q$ and $j = 1..p_i$, define operator $O_{ij}^{VF} : \{C_{J(i,j)}^F\} \to \{T_i\}$. Set weights of these operators equal to 0.

Finally, define operator O^E with weight 0 :

$$O^E : \left(\bigcup_{i=1}^{q} \{T_i\} \cup \bigcup_{k=1}^{n} \{U_k\} \right) \to \{C_E\}.$$

In total, we have defined $4n + L^\vee + 1$ operators and $L^\wedge + 3n$ colors, where $L^\vee$ is the number of additive terms in F, and $L^\wedge$ is the number of multiplicative terms in F.

It is easy to show that if SAT has a solution, the optimal solution to the color planning problem will have value n: for each variable we will either apply O_k^T or O_k^F, but not both, which corresponds to setting variable x_k to either "true" or "false". The artificial colors U_k that must be defined in any solution to planning problem ensure that either O_k^T or O_k^F is applied for each variable, so planning problem solution value will never be less than n. On the other hand, if we are given a solution to SAT, we can construct a solution for the planning problem, applying O_k^T for all "true" variables and O_k^F for all "false" variables k.