PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 07/07

Plot Mining as an Aid to Characterization and Planning

Antonio L. Furtado **Marco A. Casanova**
Simone D.J. Barbosa **Karin K. Breitman**

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 07/07 ISSN 0103-9741
Editor: Prof. Carlos José Pereira de Lucena June 2007

Plot Mining as an Aid to Characterization and Planning

Antonio L. Furtado, Marco A. Casanova, Simone D.J. Barbosa, Karin K. Breitman

Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de S. Vicente, 225 – Rio de Janeiro, Brasil – CEP 22451-900
{furtado,casanova,simone,karin}@inf.puc-rio.br

Abstract: In parallel with data mining, plot mining is presented as a valuable way to gain knowledge on management information systems. A compact plot organization and methods for plot collection and handling are described. Plots are then shown to be a significant element in the characterization of the behaviour of agents, as well as a helpful resource for planning. Similarity and analogy are employed to extend the results, both to other cases within the domain involved and to other domains.

Keywords: Plot, narrative, data mining, characterization, planning, similarity, analogy, metaphor.

Resumo: Em paralelo com mineração de dados, a mineração de enredos é apresentada como um meio valioso de ganhar conhecimento sobre sistemas de informação empresariais. Uma organização compacta de enredos é descrita, juntamente com métodos para coletá-los e manipulá-los. Mostra-se então que enredos constituem um elemento significativo para a caracterização do comportamento de agentes, sendo ainda um recurso útil para planejamento. Similaridade e analogia são empregadas para ajudar a estender os resultados a outros casos, tanto no âmbito do domínio em questão como no de outros domínios.

Palavras chave: Enredo, narrativa, mineração de dados, caracterização, planejamento, similaridade, analogia, metáfora.

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

1. Introduction

Data mining [HK] is amply recognized today as a useful way to gain a better understanding of an information system. Another vital source of knowledge are the *stories* [Sc, Tu] that happen in the underlying mini-world, and, as a result, produce state-changes in the system's database. The thrust of this paper is to propose a *plot mining* technique to exploit such real-life stories.

Literary research visualizes narratives at successive levels, the most basic one consisting of the *fabula*, defined as "a series of logically and chronologically related events that are caused or experienced by actors" [Ba]. We adopt a plot structure that provides a compact representation of partially ordered sets of events, each event being the result of the execution of some *domain-oriented operation* by an authorized agent. What logically relates the events and determines precedence among them is the interplay of the pre-conditions and post-conditions in terms of which the operations are defined. Such definitions, in turn, reflect the *integrity constraints* and *business rules* prevailing in the application domain of the information system.

Plots can be extracted from a database **Log**, or, alternatively, from logs registering the actions of individual agents. After extraction, they are generalized into an indexed pattern format and memorized for future reference, as recommended in [Sc]. The indexing information consists of the *situation/ goal circumstance* that can be associated with the plot.

Plots involving a given agent provide one significant indicator of the agent's behaviour, and, as such, can be used for characterization and comparison purposes. In addition, the memorized indexed plot patterns can be regarded as *plans*, to be selected and re-used with the adaptations required by the current concrete circumstances.

In connection with the use of plots for both characterization and planning, we resort to the powerful notions of *similarity* and *analogy*, which have been applied to properties of data at previous stages of our research project [BBFC, BBCF].

The paper is organized as follows. Section 2 covers plot organization and extraction. Sections 3 and 4 describe their use for characterization and for planning, respectively. A small example serves as illustration throughout the discussion. Section 5 is a brief excursion on plot-based agent profiles. Section 6 contains the conclusion.

2. Plots and plot indices

2.1 The notion of plot

Consider a database schema organized according to the conventions of the Entity-Relationship model [BCN]. In addition, assume that a repertoire of domain-oriented operations has been defined through their pre- and post-conditions, as proposed in the STRIPS system [FN]. In order to preserve the integrity constraints regulating the mini-world

represented in the database, an operation so defined is only allowed to be executed if its pre-conditions currently hold, and the effect of its execution corresponds precisely to its post-conditions. A complementary requirement is that no state-change can occur in the database, except as a consequence of executing an operation from the predefined repertoire. Once this double discipline is imposed, it becomes intuitively meaningful to denote any mini-world *event* that may happen by a term expressing the execution of an operation.

As a first approximation, a plot can be viewed as any sequence of events. However, it is clear that, if the above requirements are adopted, not all sequences of events are valid. If an event E_1 contributes, as a result of its post-conditions, to the pre-conditions of another event E_2, then E_1 should precede E_2 in the sequence. On the contrary, the occurrence of E_1 before E_2 should be regarded as fortuitous if none of these events depends on the other.

Consider the sequences:

S_1 = [op_1, op_2, op_3, op_4]
S_2 = [op_2, op_1, op_3, op_4]

where each op_i denotes the execution of an operation with appropriate parameters. Suppose that the execution of op_3 depends on the execution of both op_1 and op_2, and that the execution of op_4 depends on the execution of op_1 and op_3, but no precedence dependency exists between op_1 and op_2. To express these dependencies, we prefix each event in the sequences with a distinct *tag*, and provide a list of ordered pairs of tags. Note that there is no need to explicitly indicate the declared dependency of op_4 on op_1, since this follows by transitivity, which also implies the dependency of op_4 on op_2:

P_1 = [[f1:op_1, f2:op_2, f3:op_3, f4:op_4], [f1–f3, f2–f3, f3–f4]]
P_2 = [[f7:op_2, f8:op_1, f9:op_3, f10:op_4], [f7–f9, f8–f9, f9–f10]]

If all pre-conditions and post-conditions are consistently defined, P_1 and P_2 should be equivalent, in the sense that, if we choose to execute either S_1 or S_2 starting at the same current database state s_0, the overall effect observed at a state s_f reached after executing one or the other entire sequence would be the same.

We are thus led to define a *plot* P = [S,D] as a set of events S conforming to a *partial order* imposed by the precedence dependencies indicated in D. In other words, P denotes the class of all totally ordered sequences composed of the tagged events in S that comply with D. Since the S and D components are sets, we can now recognize that P_1 and P_2 above in fact refer to the same plot, given that the tags in P_2 can be renamed to become identical to those of P_1.

2.2 Plots over a simple database schema

For concreteness, let us now introduce an example schema. Since our purpose is merely to illustrate the discussion, we shall keep it as simple and short as possible. In real applications, what we choose to call here a schema would be just a small fragment of the

entire schema resulting from the conceptual design of an information system over a given application domain, such as industrial production for instance.

Suppose the entity classes `product` and `component` have been introduced with suitable properties (attributes and binary relationships) including:

attribute of product:
  `pno`
attributes of component:
  `cno`
  `ctype`
  `defective`
relationship associating component with product:
  `iscompof`

where `pno` serves as an *identifier* for instances of product, whereas instances of component, which is a *weak entity* [BCN], are identified via the `iscompof` relationship coupled with the discriminating attribute `cno`. The value of attribute `ctype` corresponds to a component's description, and the Boolean attribute `defective` exclusively qualifies those components that have been found to be unsatisfactory.

Suppose further that the repertoire of operations includes, among others:

`repair(<pno>,<cno>)`
`order(<ctype>,<cno>)`
`replace(<pno>,<cno`$_1$`>,<cno`$_2$`>)`

Operations are defined in terms of their pre- and post-conditions (effects), and can only be executed by authorized agents. In the example, the initiative to determine that operations `repair`, `order` and `replace` be executed befalls to the foreman responsible for the product involved. In practice, other agents might perhaps be charged of their actual execution but, for simplicity, such agents will be ignored. In the notation of [BBCF], the Product schema can be displayed as follows:

```
Schema: Product
Clauses --
  entity(product, pno)
  attribute(product, pno)
  entity(component, [pno/cno-iscompof-pno, cno])
  attribute(component, cno)
  attribute(component, ctype)
  attribute(component, defective)
  relationship(iscompof, component/n/total, product/1/partial)
  operation(order, [ctype, cno])
  pre(order(A, B), [])
  post(order(A, B), [ctype(B, A), -defective(B)])
  operation(replace, [pno, cno, cno])
  pre(replace(A, B, C), [iscompof(B, A), ctype(B, D),
    ctype(C, D)])/diff(B, C)
  post(replace(A, B, C), [-iscompof(B, A), iscompof(C, A)])/diff(B, C)
  operation(repair, [pno, cno])
```

```
pre(repair(A, B), [defective(B)])
post(repair(A, B), [-defective(B)])
```

Now, let P_i and P_j be plots over this schema:

```
Pi = [[f1: order(ct,c4), f2: replace(pr,c3,c4)],[f1-f2]]
Pj = [[f7: order(ct,c2), f8: replace(pr,c1,c2)],[f7-f8]]
```

We declare these two plots to be *similar* in that, even with different parameter values and tags, they involve the same:

- number and type of events
- order dependencies
- co-designation/ non-co-designation schemes

Co-designation (or, respectively, non-co-designation) allows (forbids) the occurrence of the same value in different parameter positions. For instance, notice that, in P_i, `c4` occurs in the second position of `order` and in the third position of `replace`, and that plot P_j, with `c2` in respectively corresponding places, meets the same co-designation requirement. An even stronger coincidence occurs with the values `pr` and `ct`, which are equally placed in the two plots. Non-co-designation is well exemplified in positions two and three of `replace`. To verify whether the precedence dependencies agree, one looks for a renaming of the tags of one of the plots that can render the D sets of the two plots equal, as mentioned before.

The algorithm we use for comparing plots follows a *most specific generalization* criterion [Kn,WMSW], yielding plot P_m below, which can be unified with both P_i and P_j:

```
Pm = [[f1: order(ct,A), f2: replace(pr,B,A)],[f1-f2]]
```

where constants `ct` and `pr` were kept, but, by contrast, two different variables where introduced: `A` which generalizes `c4` and `c2`, and `B` which does the same for `c3` and `c1`. The number of variables introduced provides one indicator to measure how similar two plots are; in particular, zero variables signifies identity. The search for a most specific generalization makes it necessary to compare one of the lists of events against all the permutations of the other, so as to determine the minimum number of variables that is needed. Some heuristics can be devised to reduce the number of comparisons in certain cases. There is a worst case, however, wherein execution time is exponential, which occurs when there are no dependencies and all events consist of executions of the same operation – we shall return to this point in section 3.2.

2.3 Extracting indexed plots from the Log

Plots can, as illustrated here, be composed manually, taking the pre- and post-conditions requirements into due consideration. But an especially useful way to obtain plots, extensively discussed in [FC], is to extract them from a **Log** registering the execution of operations since the database has been installed. In the scenario described in [FC], the conceptual design of the database includes the definition of *goal-inference rules* of the form

S → G, where S is a *situation* and G a *goal*. S and G, in the simplest case, are sets of positive or negative literals expressing facts at, respectively, the state before and the state after the execution of the plot. The purpose of these rules is to capture the *motivation* of the various agents: an agent A, observing that S currently holds, would be expected to act by executing (directly or through the mediation of other authorized agents) the appropriate operations to reach a state where G would hold.

After running the database for some time, the Data Administrator in charge would apply an algorithm to extract plots from the **Log** on the basis of the predefined goal-inference rules. The first step of the algorithm uses a simulation process that essentially recapitulates the evolution of the database while traversing the **Log**. A sub-sequence ŝ is extracted from the **Log** if, prior to the execution of the first event in ŝ, the situation S of a rule S → G holds and, after the execution of the last event in ŝ, a state is reached where G finally holds.

Plot P is obtained from ŝ by a filtering process, which only keeps the events whose post-conditions contribute to G, plus, proceeding backwards, recursively, those events that achieve pre-conditions of events already included in P.

Consider the goal-inference rule S → G, where:

```
S = [iscompof(X, Y), ctype(X, Z), defective(X)]
G = [iscompof(W, Y), ctype(W, Z), -defective(W)]
```

and suppose that S and G are found to hold, respectively, before the first event, and immediately after the last event of the sub-sequence of the **Log** shown below (possibly interspersed with other events):

```
... order(ct,c4) ... replace(pr,c3,c4) ...
```

After the filtering step, the algorithm consistently substitutes variables for all constants, and determines the precedence dependencies from the pre- and post-conditions present in the schema. Once this has been done, a plot (or, more precisely, a *plot pattern*) P_1 associated with S and G is obtained, which in this case is:

```
P1 = [[f1: order(Z,W), f2: replace(Y,X,W)],[f1-f2]]
```

In practical applications, the injunction of replacing *all* constants by variables must admit exceptions. One may find that certain values should always occupy certain positions, or should be retained at least as *defaults*. In [FC] we described the gradual revision of patterns by performing the most specific generalization of each newly extracted plot against the currently stored pattern, thanks to which some constants were never dropped. However, for simplicity, this extra flexibility will not be considered here.

Another real-life consideration is that, as suggested by the possible existence of unrelated events in the unfiltered sequences, agents do not always execute an entire plot without interruptions; more than that, they may even be unaware that what they are doing will accomplish a state-change complying with a given [S,G]. Nonetheless, as will be

argued in the next sections, it is in general a useful practice to record P, in connection with the associated goal-inference rule, in an adequately structured *indexed* entry, to be simply referenced heretofore as [S,G,P]. As one may readily anticipate, the association between the [S,G] and P components is commonly not unique in either direction: alternative plots can often be found in connection with the same goal inference rule, and, conversely, there may exist other rules motivating the same plot. As a case of the former, consider the single-event plot:

```
P2 = [[f1: repair(Y,X)], []]
```

which can also be found in connection with the given S → G rule.

What was said regarding the extraction of plots from the database **Log** can be easily adapted, especially in a Web environment, to logs registering the actions of individual agents (more about this in section 5).

3. Using plots for characterization

3.1 Behavioural similarity of entity instances

Entity instances are primarily characterized in terms of attributes and relationships. In [BBFC], we showed how all such properties of each instance can be collected in a frame structure, which can be conveniently used for classification and for extended forms of queries, based on similarity and on analogy. In addition to that, a behavioural characterization can be provided in connection with plots, as will be explained in the sequel.

Three main kinds of behavioural similarity can be detected, if we are dealing with indexed plots [S,G,P]. They distinguish the plot P itself from the [S,G] pair, the *circumstance* associated with P:

- sgp-similarity, if a similar plot was used in response to the a similar [S,G] circumstance;
- p-similarity, if the plot used was similar, but in a different [S,G];
- sg-similarity, if a different plot was used in a similar [S,G] .

Continuing with the example introduced in the previous section, consider two foremen, Joe and Moe, charged, during different periods of time, of product `pr`. Suppose each of them had, in some occasion, faced the problem of a defective component of type `ct`, and what they did to remedy the problem is represented, respectively, in plots P_i and P_j:

```
Pi = [[f1: order(ct,c4), f2: replace(pr,c3,c4)],[f1-f2]]
Pj = [[f7: order(ct,c5), f8: replace(pr,c4,c5)],[f7-f8]]
```

By applying the plot-comparison algorithm described in section 2.2, these two plots will be found to be similar (noting, in passing, that their most specific generalization can be

accomplished with the introduction of two variables). Since the plots are similar and assuming, as we did, that they were motivated by the same [S,G] circumstance, we conclude that the case of Joe and Moe corresponds to full sgp-similarity.

The existence of a distinct goal-inference rule S' → G', with:

```
S' = [iscompof(X, Y), ctype(X, Z)]
G' = [iscompof(W, Y), ctype(W, Z)]/ diff(W,X)
```

expressing, conceivably, an intention to keep renewing a short-lived component, even if it is not currently defective, might lead to a plot P' similar to P_i and P_j above, which would constitute a case of mere p-similarity. Finally, as an example of sg-similarity, consider the alternative plot below for the same [S,G] circumstance faced by Joe and Moe:

P_k = `[[f1: repair(pr,c3)], []]`

In [BBFC], we argued that the frame-based comparison described there could be used to perform queries. One could search through the database for one or more entity instances with properties similar to those indicated in a given arbitrary frame. Clearly, plots can be used for the same purpose, possibly with variables in some parameter positions. A caveat is however mandatory here: while frame comparison is performed in time proportional to n×log(n), plot comparison is exponential in the worst case, which means that it is only feasible with relatively short plots. One strategy to alleviate this difficulty is to organize plots in hierarchies [Ka, FC], with is-a and part-of links, as illustrated in figure 1 below:

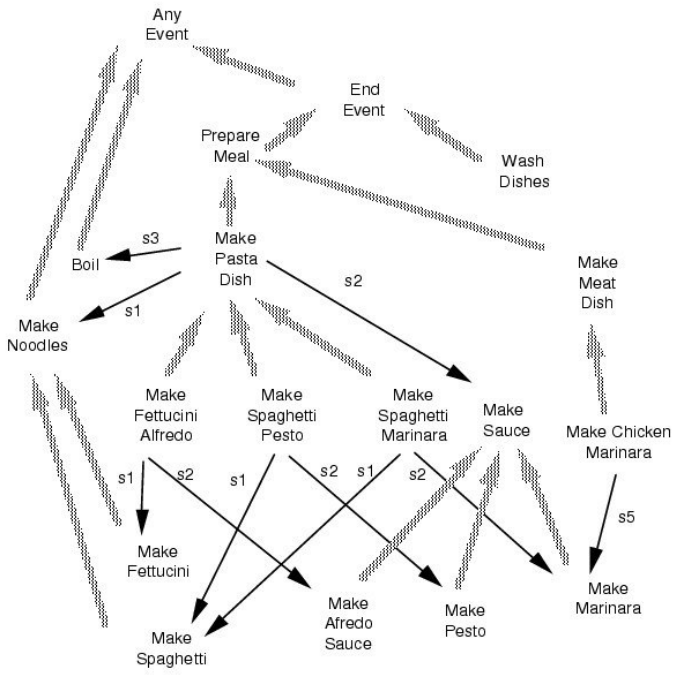

Fig. 1 – Cooking events hierarchy (from Kautz's thesis)
gray arrows: is-a links
black arrows: part-of links

By recognizing that a large plot may be divided into relatively short plots, the comparison algorithm could be applied separately to each small piece in a first step. At section 5 we shall sketch a different strategy.

Behavioural comparison is especially relevant if, as in the case of the two foremen, the entity instances associated with the plots are *agents* somehow involved in their execution. We mention the possibility – but will not exploit it here – that several agents participate, playing different roles, in the execution of a plot. Moreover, observe that entity instances other than agents can also be usefully characterized by their involvement in plots (e.g. product `pr`, components of type `ct`).

3.2 Behavioural analogy across different domains

The database schema that we have been considering provides, as mentioned in section 2.2, an example of weak entity. It has several typical features which recur in many other application domains. In [BBCF], we argued that database conceptual design, especially if complex notions such as weak entities are involved, can be significantly facilitated by deriving new schemas from previously specified *analogous* schemas. Repeating what was said at the beginning of section 2.2, such relatively small schemas would be no more than fragments of the overall schema of a real-life information system. And yet, according to the principles of modular design, it is surely expedient to obtain various such fragments with the help of the same method, and gradually compose them together until the whole design is complete.

We have extended the mapping algorithm described in [BBCF] to cover the domain-oriented operations with their pre- and post-conditions. Given the Product schema of section 2.2, the algorithm can create and record a Weak Entity *schema-pattern*:

```
Pattern: Weak Entity
Example scheme: Product
Clauses --
  entity(A, B)
  attribute(A, B)
  entity(C, [B/D-E-B, D])
  attribute(C, D)
  attribute(C, F)
  attribute(C, G)
  relationship(E, C/n/total, A/1/partial)
  operation(H, [F, D])
  pre(H, [I, J], [])
  post(H, [I, J], [[F, J, I], [-G, J]])
  operation(K, [B, D, D])
  pre(K, [L, M, N], [[E, M, L], [F, M, I], [F, N, I]])/diff(M, N)
  post(K, [L, M, N], [[-E, M, L], [E, N, L]])/diff(M, N)
  operation(O, [B, D])
  pre(O, [L, J], [[G, J]])
  post(O, [L, J], [[-G, J]])
Mappings --
  A:product
```

```
  B:pno
  C:component
  D:cno
  F:ctype
  G:defective
  E:iscompof
  H:order
  K:replace
  O:repair
```

The pattern can in turn, at any future time, be used to create new concrete schemas. Suppose that a designer, who does not have to be so experienced as the one who created from scratch the Product schema, wants to build a Team schema, involving teams and their members, and recognizes (or is told) that the intuitive mental image of the prospective schema "looks very much like" what occurs in the Product schema: members of teams are like components of products, reflecting the compelling "an organization is a machine" metaphor [Mo]. This motivates the introduction of another semantic link, in addition to is-a and part-of, namely *is-like*. In this example, the declaration Team is-like Product establishes that Product is to be regarded as a *source schema* on which the definition of the *target schema* Team can be partly accomplished [HT].

To experiment with these notions we developed a prototype supporting tool, to be used in an interactive mode. It prompts the designer to answer questions of the form: "What corresponds to <name>?", where each such <name> figures in the supposedly known source schema. The names typed by the designer will be used by the tool to instantiate the pattern variables appearing in the mapping component of the Weak Entity pattern. In the present example, the mapping correspondences, which should be saved for future use, would be:

```
product → team
pno → tno
component → member
cno → mno
ctype → spec
defective → unprepared
iscompof → ismembof
order → hire
replace → reassign
repair → train
```

based on which the tool will create the Team schema:

```
Schema: Team
Clauses --
  entity(team, tno)
  attribute(team, tno)
  entity(member, [tno/mno-ismembof-tno, mno])
  attribute(member, mno)
  attribute(member, spec)
  attribute(member, unprepared)
  relationship(ismembof, member/n/total, team/1/partial)
  operation(hire, [spec, mno])
```

```
pre(hire(A, B), [])
post(hire(A, B), [spec(B, A), -unprepared(B)])
operation(reassign, [tno, mno, mno])
pre(reassign(A, B, C), [ismembof(B, A), spec(B, D),
  spec(C, D)])/diff(B, C)
post(reassign(A, B, C), [-ismembof(B, A), ismembof(C, A)])/diff(B, C)
operation(train, [tno, mno])
pre(train(A, B), [unprepared(B)])
post(train(A, B), [-unprepared(B)]
```

Looking again at the Weak Entity pattern, one will observe that there are no constants in the clauses. In this overly simple example, there was no need to retain any constant when creating the pattern from the Product schema, but in other cases this might be convenient, and there should be a way to distinguish the constants to be preserved. In section 2.3, we remarked that most specific generalization may in such cases offer a working criterion: constants appearing in all (or in many) schemas conforming to the same abstract concept may deserve to be kept in the schema-pattern and hence replicated in the new target schemas.

In general, the mappings are not total in either direction. For some elements of the source schema, the designer will reply that there is no corresponding element in the target schema. On the other hand, after closing the dialogue, the designer must be allowed to declare additional elements that are specific to the target schema. Indeed, in more semantically rich cases, it is often convenient to proceed along successive dialogues, employing a series of source schemas, which can be interpreted as an attempt to cover different aspects by resorting to different metaphors [LJ].

Once it has been so derived, the Team schema can be used, perhaps incorporated to the design of a larger schema, directly employing the terminology appropriate to its distinct application domain. One can imagine that project leaders will be among the agents, instead of the foremen of the source domain.

Clearly, all sorts of applications of plots enumerated thus far, including those based on similarity presented in section 3.1, can be repeated in the new domain. But analogy brings in another possibility: to perform comparisons, queries, etc., that go across the two domains. For brevity, we shall consider just one example.

Imagine that a certain John, with a specialty designated as `spec_s3`, and who is currently a member of team `Ta`, is found to be unprepared. What can be done in this situation? Well, this recalls the case of component `c3` of product `pr`, when `c3` was marked as defective. Can we transpose to John what was done to `c3`?

We saw two possibilities for `c3`:

```
P1 = [[f1:repair(pr, c3)], []]
P2 = [[f1: order(ct,c4), f2: replace(pr,c3,c4)],[f1-f2]]
```

from which the analogous plots below can be readily derived:

```
P1'= [f1: train(Ta, John)], []]
P2'= [f1: hire(spec_s3, Peter), f2: reassign(Ta, John, Peter)], [f1-f2]]
```

In words, one can either submit the faulty person to training, or look for someone else with the same specialization and perform a substitution. This has undoubtedly a flavour of *case-based reasoning* [Le]. Indeed, when discussing behavioural similarity within the same domain in the previous section, this idea of adapting the same strategies for different defective components of different products would already suggest itself. The next section is dedicated to this topic.

4. Using plots for planning

4.1 Indexed plot patterns as plans

There are basically two planning paradigms. To effect the transition from a current state s_0 wherein a situation S holds into a state s_f where a desired goal G is achieved, one can either:

a. apply a *plan generator*, which generates a plan from scratch through some form of backward chaining process, taking advantage of the STRIPS pre- and post-condititions method for defining operations, or
b. retrieve a suitable ready-made plan from a *library of typical plans*, **LTP**, and re-use it, after appropriate adaptations if necessary.

What we have discussed thus far suggests a convenient way to implement paradigm b. We turn again to a scenario wherein, after the database has been running for a period of time, a Data Administrator (DA) applies the extraction and filtering algorithm to mine the **Log** in search of plots responding to the previously specified goal-inference rules. From these, the [S,G,P] indexed entries are obtained by consistently substituting variables for constants, so that the P component is now a *plot pattern*, indexed by the [S,G] *circumstance*.

We must now remark that the distinction between the notions of plot and plan is purely pragmatic: plots extracted from the **Log** register the past execution of operations, plans denote operations still to be executed – and can be represented in the same syntax used for plots, i.e. as partially-ordered sets subjected to certain precedence dependencies.

One can recognize, therefore, that the entire set of [S,G,P] indexed entries collected by the DA constitutes an **LTP**, as the use of paradigm b requires. To retrieve a plan, an interested agent supplies two lists L_s and L_g, containing positive or negative literals, to be matched against S and G, respectively, any of the two lists being allowed to be empty. It is required that the match should succeed for all literals in the two lists, i.e. every literal in L_s must unify with a literal of S, and the same must happen with the literals of L_g with respect to G. As a consequence of unification, variables (not all, in some cases) in S and G – and consequently in P – will be replaced by constants present in L_s and/or L_g. In other words,

$[L_s,L_g]$ is a *concrete circumstance* which must fit in the more general [S,G] circumstance to justify the use of the associated plan P. For instance, take:

```
Ls = [iscompof(c4,pr),ctype(c4,ct)]
Lg = [iscompof(X,pr),-defective(X)]
```

Searching the **LTP** with these lists will yield two plans:

```
P1 = [[f1:repair(pr, c4)], []]
P2 = [[f1:order(ct, X), f2:replace(pr, c4, X)], [f1-f2]]
```

noting that the second one still contains a variable. A slightly different L_g might specify that `X` must not turn out to be `c4` itself, with the consequence that only P_2 would be retrieved:

```
Lg = [iscompof(X,pr),-defective(X)]/ diff(X,c4)
```

Note that P_1 and P_2, even though they were designed to operate the required transition to a state where L_g holds, are not equivalent as to their full effects. A wise precaution is to ascertain beforehand all that may be caused by running each plan, in view of possible undesired side-effects. This can be done by *simulating* the execution of each plan, after supplying enough *context* information about the current state s_0. Simulation can employ a well-known recursive backward-chaining algorithm [FC], which, incidentally, is the basis for simple plan-generators following STRIPS formalisms. A fact F holds after an operation O is executed at a state s_R reached by executing a previous sequence of operations R, if either:

1) F is among the facts declared as added by O, and the pre-conditions of O hold at s_R;
2) F already held at s_R and is not among the facts declared as deleted by O;
3) All operations having been considered, R is empty, $s_R = s_0$, and F figures in the context informed for s_0.

With the context Ctx below:

```
Ctx = [iscompof(c4, pr),ctype(c4, ct),defective(c4)]
```

the overall result of choosing to apply either P_1 or P_2 would be as indicated:

```
R1 = [-defective(c4), ctype(c4, ct), iscompof(c3, pr)]

R2 = [-defective(X), -iscompof(c4, pr), ctype(X, ct), ctype(c4, ct),
     iscompof(X, pr)]
```

Simulation will help the agent involved not only in the choice of the more promising alternative, but also to handle cases that may call for *conditional* and/or for *iterative* execution of plans:

a. in different contexts, it may be the case that only P_1 (or only P_2) can be executed;
b. the context may indicate that more than one component is defective.

The fact that P_2 contains a variable `X` even after simulated execution, and that the variable figures in the result obtained, certainly deserves attention. We can understand that when the agent orders a component, all that has to be informed is its type. The value of the `cno` discriminating attribute will be known after the order is fulfilled, perhaps by finding a component already in store, or by purchasing it from an external supplier. This means that *plan composition*, a topic outside the scope of this paper (cf. [Tu] for the highly suggestive notion of *blend*), must take place, bringing in a complementary plan P_{2c} one of whose effects will be to instantiate `X`. An attractive possibility is that P_{2c} be also taken from **LTP**.

4.2 Finding plans by analogy

We shall now see how analogy plays a very helpful role in a multi-domain environment.

If one is dealing simultaneously with more than one application domain, it is necessary to insert the name of the domain in cause in the **LTP** entries. Suppose the **LTP** already contains entries related to the Product schema, from which the new Team schema was derived. Recall that the mapping information indicating the correspondence between names in the two schemas, gathered in the course of the derivation process, is stored for future use (as noted in section 3.2).

Suppose that a team leader, an agent in the mini-world of the Team schema, tries to access **LTP** by submitting the lists:

```
L_s = [ismembof(John, Ta), spec(John, spec_s3), unprepared(John)]
L_g = [ismembof(John, Ta), -unprepared(John), spec(John, spec_s3)]
```

A first direct attempt to perform a match inevitably fails, since there are still no entries for Team in **LTP**. But since it has been declared that Team is-like Product, and because the mappings between names have been kept, a search for analogous Product entries is automatically articulated, whereby the two following plans are produced:

```
P_1 = [[f1:train(Ta, John)], []] ;
P_2 = [[f1:hire(spec_s3, X), f2:reassign(ta, John, X)], [f1-f2]]
```

In agreement with the orientation suggested along the paper, one may also want to install indexed entries composed from such plans obtained by analogy. After that is done, they will become directly available in connection with the Team schema, and consequently P_1 and P_2 would not have to be generated again. We must ponder however that, in the scenario envisaged in the next section, it may be convenient to reserve to the Data Administrator (DA) the authority to expand the **LTP**.

5 Towards profile similarity

When comparing the behaviour of different agents, it would be far more meaningful to consider their actions along an extended period of time, during which they might have

utilized a large number of plans, in diverse circumstances. It is fair to conjecture that the totality of information about the extent to which an agent has resorted to the recorded plans gives some indication of the agent's *profile*. And it would make good sense to compare two or more agents on this ampler basis – but would it not look an overwhelming task, in view of the high cost of plot comparison? In this section we suggest that carefully organized data structures may significantly reduce the computational cost involved in dealing with profile similarity.

Let us examine in some detail a possible organization for the **LTP**, and for other auxiliary structures as well. To store the [S,G,P] entries, consider a file $\mathbf{F_{SGP}}$ with rows of the form [K,S,G,P], where K is a natural number that functions like an array subscript. Let n be the number of entries in $\mathbf{F_{SGP}}$. The DA would also create a second file $\mathbf{F_A}$, as a rectangular m×n array, with each row corresponding to one of the m agents involved – the various foremen, in our example. Under some sort of trigger control, a cell [i,j] of $\mathbf{F_A}$, of type integer, initially containing a zero value, would be incremented by one whenever agent i selected for use the P component at entry K = j of file $\mathbf{F_{SGP}}$.

As usual, inverted files can facilitate agent access to some extent. Consider two inverted files providing lists of subscripts pointing to the $\mathbf{F_{SGP}}$ entries: $\mathbf{F^{-1}_{SG}}$ and $\mathbf{F^{-1}_{P}}$, the former giving the list of integers pointing to entries of $\mathbf{F_{SGP}}$ containing plans corresponding to the given circumstance, and the latter for the opposite access, from a plan to the associated circumstances. Still faster performance could be attained if each agent had separate inverted files of both kinds restricted to the entries of individual interest. On the other hand, the maintenance of the files shared by all agents should be the sole responsibility of the DA. Recall that maintenance includes the addition of new entries, as also the decision to cover a plurality of schemas over different application domains. In the latter case, either the [K,S,G,P] rows would gain an additional D component, or separate partitions of the $\mathbf{F_{SGP}}$ file would be installed.

The most important gain in using these structures refers to the comparison of agent behaviour, which, at this point, can be done without consuming exponential time. A first simple-minded way to compare two agents Joe and Moe, assuming that rows i1 and i2 of $\mathbf{F_A}$ were set up to correspond to them, is to evaluate a summation of absolute differences:

$$V^{i1\text{-}i2} = \Sigma|\mathbf{F_A}[i1,J] - \mathbf{F_A}[i2,J]|, \text{ for } J = 1,2,...,n$$

where small values of $V^{i1\text{-}i2}$ would suggest closer similarity.

The policy of keeping a continued record of an agent's behaviour surely provides far more information than can be deduced from a single executed plot. With enough time for observation, each agent row in $\mathbf{F_A}$ can be interpreted as representing a behavioural *profile* helping to describe that agent. One may search for all agents with a profile similar, within a given threshold, to some arbitrary profile. Typical profiles might be detected in a *knowledge discovery* investigation. A person's habit to prefer correcting a failure (repair, train), instead of throwing away any faulty pieces may be a persistent personality trait, a characteristic behaviour reappearing in more than one environment.

As an interesting collateral aspect, notice that column-wise summation may also be helpful:

$R^j = \Sigma\ \mathbf{F_A}[I,j]$, for I = 1,2,...,m

since a high value of R^j, for some column j, may work as a *recommendation* [SKKR] in favour of applying a given plan to a given circumstance, suggesting that further computations over the array could serve in yet another knowledge discovery effort. Recommendation, as in the above reference, is a topic of special relevance in the context of *e-commerce*, which brings to mind our remark at the end of section 2.3 about extracting plots from individual agents' logs – a potential way to gain familiarity with the needs and tastes of specific customers and other participants in the business transactions.

The issues raised in this section open a number of possibilities. More flexible hierarchic structures for **LTP**, such as those devised in [Ka], have been implemented and then extensively explored in [FC], equally in connection with a *plan-recognition* algorithm. The algorithm permits the early detection of the ultimate goal of an agent, by matching the agent's observed execution of a few operations, in order to determine whether they are part of one (or more) typical plan(s).

6. Concluding remarks

We have argued that plots, indexed by the situation-goal circumstance that motivates their enactment, provide a compact and easy-to-handle representation for the real-life stories happening in the mini-world of management information systems. Their use for the characterization of agents' behaviour and for providing ready-made plans was discussed and illustrated with the help of a recurring example, observable in several application domains, and it was shown how similarity and analogy can extend their applicability.

Thus far we concentrated on single agent environments. More research is needed to cope with multi-agents environments, with a special attention to the inclusion of operations enabling communicative acts [LF], and to the treatment of goals and plans involving collaboration, competition and negotiation [Wi].

References

[Ba] Bal, M. *Narratology: Introduction to the Theory of Narrative*. University of Toronto Press, 2002.

[BBCF] Breitman, K. K., Barbosa, S. D. J., Casanova, M. A. and Furtado, A. L. “Conceptual modeling by analogy and metaphor”. Submitted for publication.

[BBFC] Barbosa, S. D. J., Breitman, K. K., Furtado, A. L. , Casanova, M. A.. "Similarity and Analogy over Application Domains". Submitted for publication.

[BS] Barbosa, S. D. J. and de Souza, C. S. “Extending software through metaphors and metonymies”. In *Knowledge-Based Systems*, 14, 2001.

[BCN] Batini, C., Ceri, S. and Navathe, S. *Conceptual Design – an Entity-Relationship Approach*. Benjamin Cummings, 1992.

[FC] Furtado, A. L., Ciarlini, A. E. M. "Constructing Libraries of Typical Plans". In *Proc. CaiSE'01, The Thirteenth International Conference on Computer Advanced Information System Engineering*, 2001.

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

[HK] Han, J. and Kamber, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2005.

[HT] Holyoak, K. J. and Thagard, P. *Mental Leaps: Analogy in Creative Thought*. MIT Press, 1996.

[Ka] Kautz, H. A. "A Formal Theory of Plan Recognition and its Implementation". In: Allen, J. F. et al (eds.) *Reasoning about Plans*. Morgan Kaufmann, 1991.

[Kn] Knight, K. Unification: "A Multidisciplinary Survey". *ACM Computing Surveys*, Vol. 21, No. 1, March, 1989.

[Le] Leake, D., B. *Case-Based Reasoning: Experience, Lessons & Future Directions*. The MIT Press, 1996.

[LF] Labrou, Y and Finin, T. "History, State of the Art and Challenges for Agent Communication Languages". In *Informatik – Informatique* 1, 2000.

[LJ] Lakoff, G. and Johnson, M. *Metaphors We Live By*. University of Chicago Press, 1980.

[Mo] Morgan, G. *Images of organization - Executive edition*. Sage Publications, 1998.

[Sc] Schank, R. *Tell me a Story: Narrative and Intelligence*. Northwestern University Press, 1990.

[SKKR] Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J. "Analysis of recommendation algorithms for e-commerce". In *Proc. ACM Conference on Electronic Commerce*, p. 158-167, 2000.

[Tu] Turner, M. *The Literary Mind*. Oxford University Press, 1996.

[Wi] Willensky, R. *Planning and Understanding - a Computational Approach to Human Reasoning*. Addison-Wesley, 1983.

[WMSW] Walker, A., McCord, M., Sowa, J. F. and Wilson, W. G. *Knowledge Systems and Prolog*. Addison-Wesley, 1987.