# Formal Foundation of Workflow Hyperpaths and a Graph Search Algorithm for Workflow Hyperpath Generation

Sinnakkrishnan Perumal[†] and Ambuj Mahanti
*Indian Institute of Management Calcutta, Joka, D.H. Road, Kolkata-700104, India*
*{krish, am}@iimcal.ac.in*

## Abstract

*While executing a process, many decisions are taken at its various decision points for selecting paths. There is a need for understanding and analyzing on various paths that emanate from various decision points to ensure that right decision is taken at these moments. For this purpose, this paper presents the notion of hyperpaths in the context of workflows, its properties, and an algorithm to generate various hyperpaths from a node of a workflow graph. The algorithm is presented with detailed workouts using examples. The paper also details various similarities between hyperpaths and related concepts like instance subgraphs, instance flows, etc., and briefs on various algorithms available in literature for such concepts. Finally, the paper presents various applications of hyperpaths such as process mining, business process re-engineering, etc. Hyperpaths can be used in service-oriented computing model for resource management, business activity monitoring (BAM), dynamic orchestration, and mission-critical service oriented systems.*

## 1. Introduction

[5] defines a business process as "the definition of tasks and the sequence of those tasks necessary to deliver a business function". [8] states that "To perform BPR, several sets of guidelines have been proposed, including the five-step approach by Davenport [4], the six-step approach by Furey [6], and AT&T's seven-step approach [9]. Regardless of differences in their subtle details, these guidelines suggest that analysis of existing critical business processes as well as redesign of these processes are two essential BPR tasks". To analyze existing processes, it is necessary to understand the process paths starting from various decision points of processes. This is necessary to determine the implication on various factors such as time, cost, etc., on selecting one process path over another.

In this paper, we term various multi-pronged process paths that emanate from a node of a workflow process and terminate in its end node as hyperpaths. Hyperpaths can be used to understand various alternative executions that are possible below a node in a workflow process. It can also be used to analyze processes on various process metrics as follows. [7] presents a framework for Business Process Intelligence where process taxonomies are used for analyzing processes. An example of process taxonomy could be where process instances are classified into two types, successful and unsuccessful, by using some success criteria [7]. Process taxonomies can be extended by using the concept of workflow hyperpaths. For example, hyperpaths that typically provide better results could be given a higher priority at various decision points. Hyperpaths can be used in resource optimization and business activity monitoring for a workflow management system in a service-oriented computing model as explained in Section 9. It can also be used to create alternative flows when an exception occurs during process execution.

Hyperpaths are defined in this paper in the context of acyclic workflow processes, and an algorithm is presented to generate all hyperpaths below a node of a workflow graph. Workflow graph is one of the many representations available for representing a workflow process.

This paper is organized as follows. Section 2 describes workflows and instance subgraphs, and introduces the notion of workflow hyperpaths. Section 3 presents the literature survey. Section 4 presents the properties of hyperpaths. Section 5 presents various hyperpaths for a toy problem. Section 6 defines concept of hyperpaths. Section 7 presents an algorithm to generate the hyperpaths.

[†] This research was partially supported by Infosys Technologies Limited, Bangalore under the Infosys Fellowship Award.

Section 8 details the workout of the algorithm. Section 9 and 10 present various applications of hyperpaths with examples.

## 2. Foundation: Workflows, Instance Subgraphs and Workflow Hyperpaths

A workflow process is executed for a specific case, where example of a case could be an insurance claim, an order, or a request for an information [1].

This paper uses various node types for representing various components of a workflow graph such as OR-split, OR-join, AND-split, AND-join and sequence as given in [13]. An OR-split node is used to create mutually exclusive alternative paths from that node, and an OR-join node is used to merge these mutually exclusive paths. An AND-split node is used to create concurrent paths from that node, and an AND-join node is used to synchronize these concurrent paths. OR-split and OR-join nodes are depicted using circles, while sequence, AND-split and AND-join nodes are depicted using rectangles.

An instance subgraph is a subgraph of a workflow graph that will be executed for a case [13]. Thus, for an instance subgraph, for any included OR-split node, exactly one child node of it and the edge to it will be included, and for any other included node, all its child nodes and the edges to them will be included.

A directed hypergraph is considered in Operations Research literature as "a pair H = (V,E), where V = ($v_1$, … , $v_n$) is a set of nodes, and E=($e_1$, … , $e_m$) is a set of hyperarcs. A hyperarc $e \in E$ is a pair e=(T(e), h(e)), where $T(e) \subset V$ denotes the tail nodes and $h(e) \in V \backslash T(e)$ denotes the head node" [10]. Similarly, a hyperpath is defined as "A hyperpath $\Pi_{st}$ of origin s and destination t, is an acyclic minimal hypergraph (with respect to deletion of nodes and hyperarcs) $H_\Pi$ =(V,E) satisfying the following conditions:

1. $E_\Pi \subseteq E$
2. $s,t \in V_\Pi = \cup_{e \in E\Pi} (T(e) \cup \{h(e)\})$
3. $u \in V_\Pi \backslash \{s\} \Rightarrow$ u is connected to s in $H_\Pi$" [10].

We define the notion of hyperpath rooted at or below a node 'n' in the context of workflow, i.e., a workflow hyperpath as follows,

1. A workflow hyperpath is defined starting from any node of a workflow graph *G*, and this node is included in the workflow hyperpath.
2. For any OR-split node included in the workflow hyperpath, exactly one of its child nodes and the edge to it are also included in the workflow hyperpath.
3. For any other type (i.e., AND-split, sequence, and AND-join) of node included in the workflow hyperpath, all its child nodes and the edges to them are also included in the workflow hyperpath.

Henceforth, for keeping it simple, we will call a workflow hyperpath as a "hyperpath".

## 3. Literature Survey

[3] provides a procedure INSGraph which generates all instance flows within an inline block of a workflow graph. Here, inline block refers to a subset of nodes and edges between them satisfying the property that any inward transition to the inline block can only occur at the starting node of the inline block and any outward transition can only occur at the end node of the inline block [3]. Concept of instance flow is equivalent to that of an instance subgraph that is constrained within an inline block. The procedure scans the block and chooses a set of unexplored OR-split nodes which do not have any other unexplored OR-split node as its ancestor in the block. Various combinations of outgoing edges from these OR-split nodes are used to generate different sets of instance flows in an inline block. For each combination of outgoing edges from these OR-split nodes, the procedure INSGraph is called recursively. Thus, the procedure INSGraph generates all the instance flows within an inline block. A minor disadvantage of this method is that if all instance flows below a node of a workflow graph have to be generated, then the subgraph below this node has to be carved out of the original workflow graph and given to INSGraph procedure as input.

[8] provides a framework and algorithm for generating a process from the past executions of that process. This method is useful when there are no pre-defined workflow processes in an organization, or if the pre-defined workflow processes are not followed strictly. The concept described in our paper works in reverse of this method, i.e., it generates all possible execution subgraphs below a node when there are pre-defined processes in an organization.

[14] provides a mechanism to restructure workflow processes in such a way that the decision points in a process are moved to the earliest possible points based on the input data requirements for each task. Thus, it provides a mechanism to improve the efficiency of workflow processes. [14] also provides a crude way of finding all the instance types (instance types are similar to instance subgraphs) by

generating the instance types by traversing various nodes of the workflow from the start node. After generating various instance types, the duplicate entries are removed. Disadvantage of this method is that there will be many duplicate entries when there are many decision points in a sequence in the process. Issues in generating various hyperpaths below a node of a workflow graph are, to generate all the hyperpaths below that node without leaving out any of them, and generating hyperpaths without creating duplicate entries. Further, it could be noted that only our paper formally defines hyperpaths, which is a superset of instance subgraphs.

## 4. Properties of Hyperpaths

Various properties of a hyperpath are as follows,

1. Various maximal graph paths in a hyperpath should terminate at the end node of the workflow graph *G*.
   **Proof:** As the workflow graph has a single end node, all the maximal graph paths from any node should terminate at the end node of the workflow graph. Hence, proved.
2. Any instance subgraph which includes a node *n* of the workflow graph should contain one of the hyperpaths from that node.
   **Proof:** For creating a hyperpath from *n*, for every included OR-split node, exactly one child node and the edge to it are included, and for every other included node, all its child nodes and the edges to them are included. If an instance subgraph includes the node *n*, then below *n*, for every included OR-split node, exactly one child node and the edge to it are included, and for every other included node, all its child nodes and the edges to them are included. Hence, proved.
3. If a node *n* is an ancestor of a node *m* in a workflow graph *G*, then all the hyperpaths from *m* should be contained within various hyperpaths from *n*.
   **Proof:** Since *n* is an ancestor of *m* in *G*, if a hyperpath from *n* includes *m*, then the subgraph below *m* in this hyperpath should form a hyperpath starting from *m*. Hence, proved.

## 5. Examples

Various hyperpaths from node "0" for the toy problem in (Figure 1) are given in (Table 1). Similarly, various hyperpaths from node "1" are given in (Table 2), and those from node "2" are given in (Table 3).

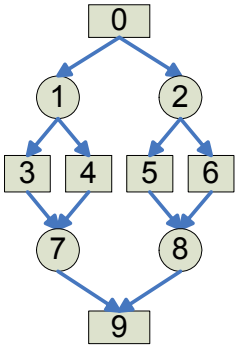



**Figure 1. Toy problem 1**

**Table 1. Hyperpaths below node "0" for toy problem in Figure 1**

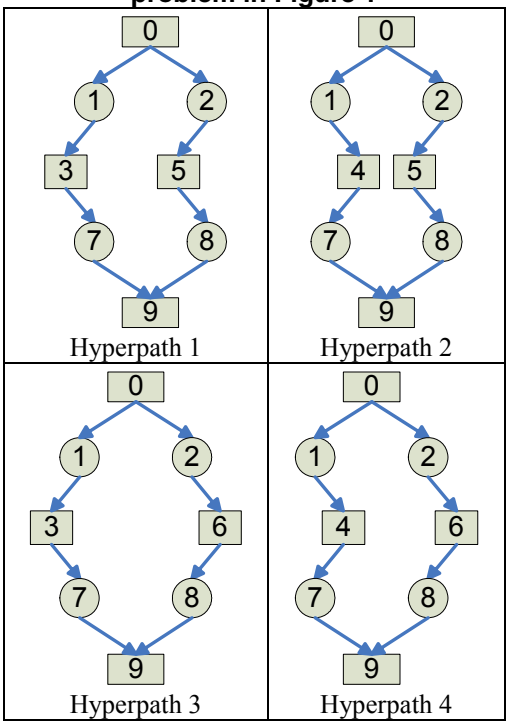



**Table 2. Hyperpaths below node "1" for toy problem in Figure 1**

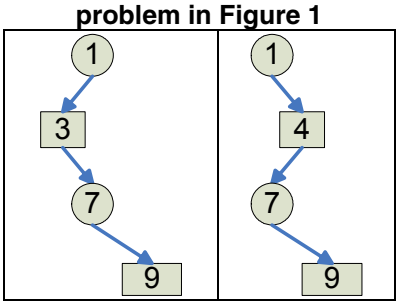
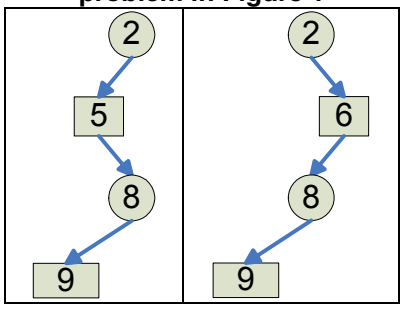


**Table 3. Hyperpaths below node "2" for toy problem in Figure 1**




## 6. Hyperpaths as Instance Subgraphs

If a hyperpath *H* is created from the start node *s* of a workflow graph *G* (i.e., the start node of hyperpath *H* is also *s*), then the hyperpath so obtained will also be an instance subgraph of the workflow graph. Thus, various instance subgraphs of a workflow graph can be obtained by generating various hyperpaths from the start node of that workflow graph.

## 7. Workflow Hyperpath Generation (WHG) Algorithm

### 7.1. Algorithm Description

Workflow Hyperpath Generation (WHG) algorithm proposed in this paper is given in the appendix. This algorithm generates all hyperpaths that start from a node of a workflow graph.

WHG algorithm consists of three procedures Main, Generate_Hyperpath and Create_Hyperpath, of which procedure Main is the main procedure of the algorithm. Procedure Main calls the procedure Generate_Hyperpath with appropriate inputs to start generating the hyperpaths for the given workflow graph. Procedure Generate_Hyperpath generates a set of hyperpaths while keeping the markings fixed for certain OR-split nodes given to it as input. Markings indicate the choice of outgoing edges from various OR-split nodes. Procedure Create_Hyperpath creates a new hyperpath as per the markings specified for various OR-split nodes (where marking is not specified for an OR-split node, it will take the first outgoing edge from that OR-split node for creating the hyperpath). An approach similar to the graph search method used in [11, 12] for creating and verifying various instance subgraphs of a workflow graph is used in the procedure Create_Hyperpath for creating the hyperpath.

Inputs for procedure Main are workflow graph G, the initial node I from which to generate hyperpaths, and a stack of OR-split nodes called OR_split_stack. Algorithm WHG can be applied on a workflow process, or for a case executing as per a workflow process. In the latter scenario, paths from some OR-split nodes could have been chosen for the case. Then, OR_split_stack should have OR-split nodes from which outgoing edges have been chosen for that case, and also markings indicating the corresponding outgoing edges. Procedure Main calls the procedure Generate_Hyperpath and passes G, I, OR_split_stack and a stack pointer pointing to the top of OR_split_stack as parameters.

Procedure Generate_Hyperpath takes the inputs as G, I, OR_split_stack and a stack pointer pointing to a location in OR_split_stack below which it will not change the markings of OR-split nodes. It calls the procedure Create_Hyperpath to create a new hyperpath as per the markings for various OR-split nodes in input OR_split_stack. Then, it takes a top-most OR-split node in OR_split_stack which has its marking in a non-last outgoing edge, shifts the marking for this OR-split node to the next outgoing edge, and calls Generate_Hyperpath recursively to generate all hyperpaths which have a similar marking. It repeats this step till it can change marking of any OR-split node as mentioned above.

Procedure Create_Hyperpath traverses the graph using depth-first search to create a new hyperpath as per the markings specified in input OR_split_stack. It uses a stack Z for this depth-first search. If it traverses through any OR-split node which is not in input OR_split_stack, it pushes that OR-split node into OR_split_stack, and sets the marking for that OR-split node as the first outgoing edge. From such an OR-split node, it will traverse through the first outgoing edge for creating the hyperpath.

Data structures used in the algorithm are given below.

### 7.2. Data Structures

**G** : Implicit Graph G, i.e., given workflow graph.
**I** : Initial node from which hyperpaths will be generated. I may be equal to the start node s of the given workflow graph.
**OR_split_stack** : Stack containing OR-split nodes along with their marking.
**G'** : Explicit Graph. Here, it is used to represent a hyperpath.
**Z** : Stack containing nodes to be expanded while creating a hyperpath.

### 7.3. Implementation

This algorithm was implemented in C language on Linux platform.

## 8. Workout of the Algorithm

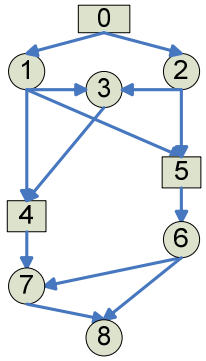



**Figure 2. Toy problem 2**

Workout of the algorithm for the toy problem in (Figure 2) is given below. Inner boxes in this workout indicate the recursive call of the procedure Generate_Hyperpath. Workout illustrates the steps that are executed for the first call of the procedure Generate_Hyperpath from the procedure Main.

**Workout:**

1. Create_Hyperpath called to create a new hyperpath:

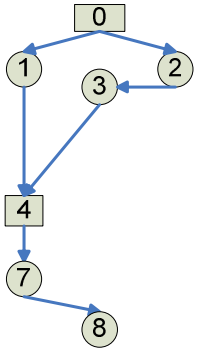



**Figure 3. Hyperpath 1**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 0 |
| 2 | 0 |

2. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 1 |
| 2 | 0 |

3. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

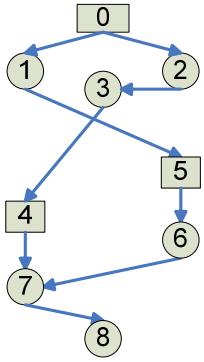



**Figure 4. Hyperpath 2**

OR_split_stack status:

| Node | Marking |
|---|---|
| 6 | 0 |
| 1 | 1 |
| 2 | 0 |

2. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 6 | 1 |
| 1 | 1 |
| 2 | 0 |

3. Generate_Hyperpath called recursively after changing the marking for node "6" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

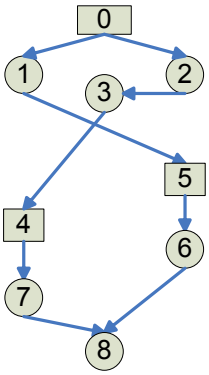



**Figure 5. Hyperpath 3**

OR_split_stack status:

| Node | Marking |
|---|---|
| 6 | 1 |
| 1 | 1 |
| 2 | 0 |

4. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 2 |
| 2 | 0 |

5. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

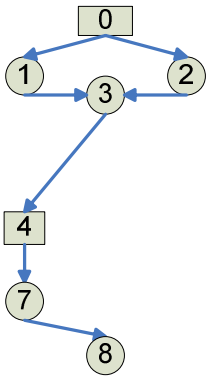



**Figure 6. Hyperpath 4**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 2 |
| 2 | 0 |

6. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 2 | 1 |

7. Generate_Hyperpath called recursively after changing the marking for node "2" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

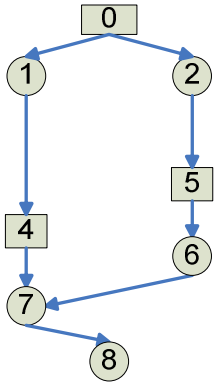



**Figure 7. Hyperpath 5**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 0 |
| 6 | 0 |
| 2 | 1 |

2. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 1 |
| 6 | 0 |
| 2 | 1 |

3. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

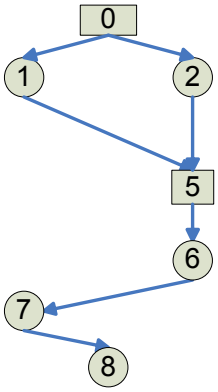



**Figure 8. Hyperpath 6**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 1 |
| 6 | 0 |
| 2 | 1 |

4. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 2 |
| 6 | 0 |
| 2 | 1 |

5. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

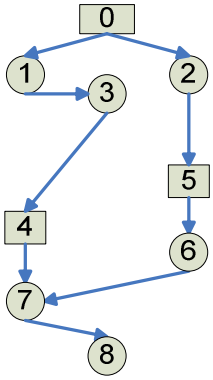



**Figure 9. Hyperpath 7**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 2 |
| 6 | 0 |
| 2 | 1 |

6. OR_split_stack updated as :

| Node | Marking |
|---|---|
| 6 | 1 |
| 2 | 1 |

7. Generate_Hyperpath called recursively after changing the marking for node "6" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

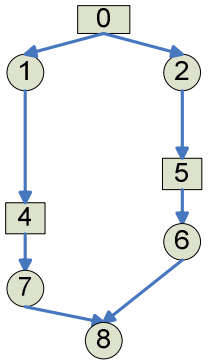



**Figure 10. Hyperpath 8**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 0 |
| 6 | 1 |
| 2 | 1 |

2. OR_split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 1 |
| 6 | 1 |
| 2 | 1 |

3. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

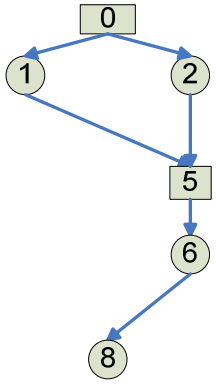



**Figure 11. Hyperpath 9**

OR-split_stack status:

| Node | Marking |
|---|---|
| 1 | 1 |
| 6 | 1 |
| 2 | 1 |

4. OR-split_stack updated as :

| Node | Marking |
|---|---|
| 1 | 2 |
| 6 | 1 |
| 2 | 1 |

5. Generate_Hyperpath called recursively after changing the marking for node "1" in OR_split_stack:

1. Create_Hyperpath called to create a new hyperpath:

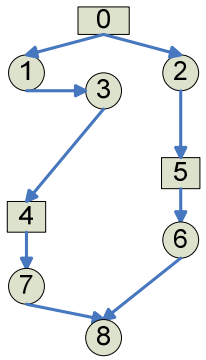



**Figure 12. Hyperpath 10**

OR_split_stack status:

| Node | Marking |
|---|---|
| 1 | 2 |
| 6 | 1 |
| 2 | 1 |

## 9. Applications of Hyperpaths

Various hyperpaths from a node *n* can be used to determine the optimal execution of the workflow from that node in terms of time, cost, or any other factor under consideration. Further, if execution details of various tasks are appropriately logged, it is possible to determine the hyperpaths that lead to frequent errors, or any other undesirable behavior during workflow execution. Thus, hyperpaths find applications in wide range of topics on processes such as workflow implementation, process mining, business process re-engineering, process design, and process performance management.

## 10. Application Example

(Figure 13) presents an application example. This example is adopted from [13] and adapted to meet the needs of this paper. Suppose that this workflow process is executed for a case, wherein the payment request is US dollars. Then, if this case has been executed till the OR-split node C1, the edge C1 to T2 would be marked for this case. All hyperpaths that are possible for this case below the node C1 can be obtained using WHG algorithm as follows: (a). insert C1 into OR_split_stack, along with its marking corresponding to the edge from C1 to T2, and (b). call the procedure Main with the initial node as C1. All instance subgraphs that correspond to the execution of the case up till now can be generated as follows: (a). insert C1 into OR_split_stack, along with its marking corresponding to the edge from C1 to T2, and (b). call the procedure Main with the initial node as the start node T1. Thus, it is possible to understand the various alternative executions for the case. Accordingly, resource utilization in the organization can be optimized.

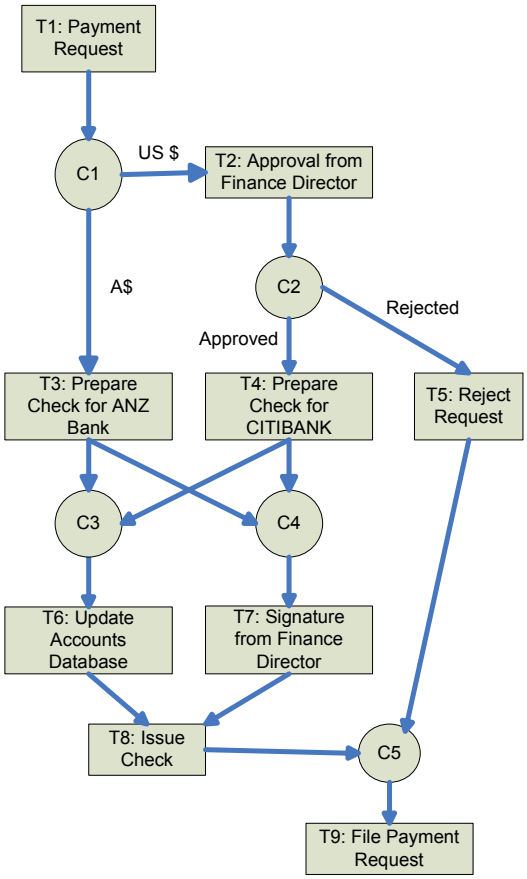



**Figure 13. Application Example 1 adopted from [13]**

(Figure 14) presents another application example. This example is adopted from [14] and adapted to meet the needs of this paper. Suppose that this workflow process is executed for a case, wherein it has been decided that manager need not be consulted. Then, all hyperpaths that are possible for this case below the node C1 can be obtained using WHG algorithm as follows: (a). insert C1 into OR_split_stack, along with its marking corresponding to the edge from C1 to C2, and (b). call the procedure Main with the initial node as C1. With this, it can be determined that the additional time taken for executing the case further will be maximum of {((time taken to take the decision whether to grant the loan) + (time taken to grant the loan)), ((time taken to take the decision whether to grant the loan) + (time taken to reject the loan))}.

Thus, the customer who had applied for loan can be informed about the time within which the case will be completed. Hyperpaths can also be used in mission-critical service oriented systems to determine various alternative multi-pronged process paths when a fault occurs.

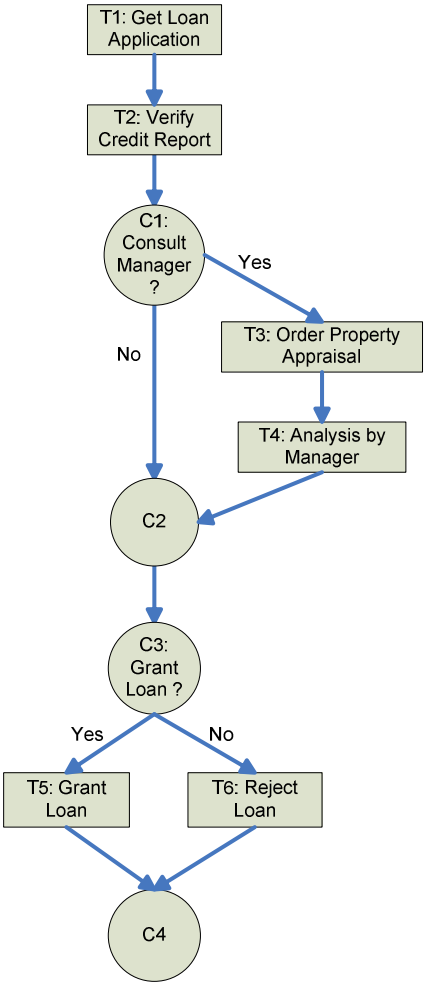



**Figure 14. Application Example 2 adopted from [14]**

## 11. Conclusion

In this paper, the notion of hyperpaths was introduced for acyclic workflow graphs. Further, an algorithm was presented to generate all the hyperpaths in acyclic workflow graphs. Future work in this area could involve, (a). extending the concept of hyperpaths to cyclic workflow graphs, (b). creating an algorithm for generating hyperpaths in cyclic workflow graphs, (c). theorizing the notion of hyperpaths using concepts from operations research literature, workflow management, and others, (d). analyzing various applications of hyperpaths, and (e). extending the concepts of hyperpaths to other workflow languages such as YAWL as given in [2].

Our hyperpath generation algorithm imbibes depth-first search technique available for directed acyclic graphs, thereby helping in better intuitive understanding of the search, intelligent generation of hyperpaths and efficient theoretical analysis.

Hyperpaths can be used in service-oriented architecture, (a) to invoke services which provide the alternative flow(s) prioritizing on time, cost and quality benefits, (b) to mine historical data to identify services that provide better results, and (c) to analyze alternative flows when an exception occurs while executing a business process.

## 12. References

[1] W. M. P. v. d. Aalst, "Re-engineering knock-out processes", *Decision Support Systems,* vol. 30, pp. 451-468, 2001.

[2] W. M. P. v. d. Aalst and A. H. M. t. Hofstedeb, "YAWL: yet another workflow language", *Information Systems,* vol. 30, pp. 245–275, 2005.

[3] Y. Choi and J. L. Zhao, "Matrix-based abstraction and verification of e-business processes", in *1st Workshop on e-Business*, 2002, pp. 154-165.

[4] T. H. Davenport, *Process Innovation—Reengineering Work through Information Technology*, Harvard Business School, Boston, MA, 1993.

[5] R. Davis, *Business Process Modelling with ARIS: A Practical Guide*, Springer, London et al., 2001.

[6] T. R. Furey, "A Six Step Guide to Process Reengineering", *Planning Review,* vol. 21, pp. 20-23, 1993.

[7] D. Grigoria, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan, "Business Process Intelligence", *Computers in Industry,* vol. 53, pp. 321-343, 2004.

[8] S.-Y. Hwang and W.-S. Yang, "On the discovery of process models from their instances", *Decision Support Systems,* vol. 34, pp. 41-57, 2003.

[9] R. L. Manganelli and M. M. Klein, *The Reengineering Handbook: A Step-by-Step Guide to Business Transformation*, American Management Association, New York, 1994.

[10] L. R. Nielsen, K. A. Andersen, and D. Pretolani, "Finding the K shortest hyperpaths", *Computers & Operations Research,* vol. 32, pp. 1477–1497, 2005.

[11] S. Perumal and A. Mahanti, "A Simple and Efficient Algorithm for Verifying Workflow Graphs", in *Workflow Handbook 2005*, L. Fischer, Ed., Future Strategies Inc., Lighthouse Point, 2005, pp. 233-256.

[12] S. Perumal and A. Mahanti, "Applying Graph Search Techniques for Workflow Verification", in *HICSS*, 2007, p. 48.

[13] W. Sadiq and M. E. Orlowska, "Analyzing Process Models Using Graph Reduction Techniques", *Inf. Syst.,* vol. 25, pp. 117-134, 2000.

[14] S. Subramaniam, V. Kalogeraki, D. Gunopulos, F. Casati, M. Castellanos, U. Dayal, and M. Sayal, "Improving process models by discovering decision points", *Information Systems,* doi:10.1016/j.is.2006.11.001 2007.

[15] W. M. P. van der Aalst, A. Hirnschall, and H. M. W. (Eric) Verbeek, "An Alternative Way to Analyze Workflow Graphs", in *CAiSE*, 2002, pp. 535-552.

## Appendix – A: Workflow Hyperpath Generation (WHG) Algorithm

**Procedure Main(Graph G, Node I, Stack OR_split_stack)**

Let OR_split_stack_pointer point to the top of OR_split_stack.

Generate all the required hyperpaths by calling the procedure Generate_Hyperpath as, Generate_Hyperpath(G, I, OR_split_stack, OR_split_stack_pointer).

End Procedure

**Procedure Generate_Hyperpath (Graph G, Node I, Stack OR_split_stack, StackPointer OR_split_stack_fixed_pointer)**

Create a new hyperpath by calling the procedure Create_Hyperpath as, Create_Hyperpath(G, I, OR_split_stack).

Let OR_split_stack_pointer point to the top of OR_split_stack.

While OR_split_stack_pointer is above OR_split_stack_fixed_pointer in OR_split_stack

Let $S_i$ be the top-most node in OR_split_stack.

If $S_i$ has its marking in the last outgoing edge then

Pop $S_i$ from OR_split_stack, and update OR_split_stack_pointer.

Else

Choose the next outgoing edge from $S_i$, shift the marking to this outgoing edge, and update marking for $S_i$ in OR_split_stack.

Generate a set of hyperpaths keeping the marking from $S_i$ and other OR-split nodes below $S_i$ in OR_split_stack as constant by calling the procedure Generate_Hyperpath as, Generate_Hyperpath(G, I, OR_split_stack, OR_split_stack_pointer).

End While

End Procedure

**Procedure Create_Hyperpath(Graph G, Node I, Stack OR_split_stack)**

Initialize a stack Z containing only the node I.

Initialize the explicit graph G' by installing the node I in it. Label I as not expanded in G'.

While Z is not empty do

Pop the top node from Z. Let this node be called "q".

If q is not already expanded in G' then

In G', label q as expanded.

If q is OR-split node then

If q is in OR_split_stack then

Let r be the child node of q through the edge corresponding to marking for q in OR_split_stack.

Install r if it is not already present in G'.

Install the edge from q to r in G'.

Else

Install the first child node of q in G' if it is not already present in G'.

Install the edge to this child node of q in G' and mark this edge.

Push q to OR_split_stack and set the marking for q in OR_split_stack.

Push the child node through marked edge of q to the top of Z.

Else

Install all the child nodes of q in G' if they are not already present in G'.

Install the edges to these child nodes of q in G' and mark these edges.

Push these child nodes to the top of Z in, say, left-to-right order such that the right-most child node is on the top of Z.

End while

Print the new hyperpath G'.

End Procedure