Analysis of Business Process and Capability Hypergraphs

J. Q. Trelewicz, Jorge L. C. Sanz, Ankur Chandra

Abstract: *Business process modelling is receiving renewed attention as businesses restructure their business partnerships, sourcing agreements, and production arrangements. However, although processes may be connected to the strategic elements of a business, the process graphs are not themselves business models. When a business is restructured, either in terms of organization or in terms of strategic direction, process graphs may be completely changed. We discuss in this paper the linking of process and capability models with hypergraphs, facilitating richer modelling and analysis of the business.*

Key words: *Business Process Modelling, Business Model.*

INTRODUCTION

Business process modeling has received renewed interest since the business process reengineering efforts that many companies underwent in the 1990s. Ask many people how to represent the design of a business, and their first answer will be "process". Process graphs – graphs of process elements that can be tasks or subprocesses, which will from this point forward be called "tasks" – describe, on a time scale, sequences that are executed in the operation of the business. Although processes may be connected to the strategic elements of a business, the process graphs are not themselves business models. When a business is restructured, either in terms of organization or in terms of strategic direction, process graphs may be completely reinvented. It is for this reason that it is important to use the process graph as only one aspect of designing and documenting the business, so that the resulting model will be useful in making organizational or strategic changes. A model comprised solely of the process graph is extremely rigid for visualizing the impact of changes to the business.

The model of the business can be made much richer by also considering capabilities of the business. Capabilities describe those potentials for what a business must be able to perform, both internally and externally, and are strongly connected to the value proposition. One view of capabilities was described by Porter [1]. Capabilities, unlike processes, are not described in terms of time sequence. Linking capabilities to processes ties the operational aspect of a business, in terms of its processes, to its value to its customers.

In this paper, we explore the mathematical formulation of the interconnected capability and process graphs, and the methods by which they may be partitioned.

PROCESS GRAPHS

The process graph consists of tasks, which form the vertices of our process graph. Process graphs may be hierarchical; that is, one task in a process graph may itself be a process graph. Directed edges between the tasks represent the sequence of process flow. If more than one edge passes out of a process vertex, this indicates either a decision point or parallel paths to be undertaken in subsequent steps. If more than one edge passes into a process vertex, this may or may not indicate a resequencing of parallel paths. This may indicate a process which is performed as a step in several different process sequences. Furthermore, the edges do not indicate the type of flow; e.g., finish-before-start, start-before-start, etc. Such detailed semantics are useful at simulation levels of the business, but are not discussed in this paper.

All edges in our process graphs are unidirectional. Some processes have transition in both directions; e.g., it may be possible for process A to transition to process B, and for process B to transition to process A. These situations are modelled with separate edges for each possible transition.

The process edges may be weighted. The edge weights of the connections may represent several things, although a single meaning is chosen for a given process graph.

Some examples are the following:

- The monetary cost of this step in the process flow; e.g., the bandwidth charge required for sending the data associated with this flow step.
- The complexity of this step in the process flow, which may be indicated by the amount of training of personnel required to facilitate this flow step correctly.

If there is utility in analyzing the process graph with more than one meaning to the edge weights, then multiple graphs in the hypergraph are created with the same process vertices, using the different meanings of the edge weights in different graphs within the hypergraph. This preserves homogeneity within a graph, while allowing richer analysis.

CAPABILITY GRAPHS

The capability graph describes the ability to perform the key activities of a business. Capabilities are distinguished from processes, in that processes rely on a time sequence for definition, whereas capabilities have no explicit time sequence. Specifically, whereas processes transition from one to another, capabilities enable one another, which can correspond to activities occurring in parallel, or even out of sequence with the enablement. Edges between capability vertices are unidirectional, corresponding to the direction of enablement. The weight assigned to the edge reflects the degree to which the enabled capability depends on the enabling capability.

For example, a capability may be important to a business's value proposition to its customers, and an important enabler of its cost model. Without being able to document clearly the dependencies of this capability on the others, the business risks making organizational change decisions without realizing the full impact of its decisions on its full capabilities.

CONNECTING PROCESS AND CAPABILITY GRAPHS

Process is connected to capability, but the mapping is neither injective nor surjective. For example, a capability to provide order status information to a client may involve a process to perform the capability. However, a capability may connect to several or no processes. For example, the capability to provide low-cost fees to clients may be tied to no process, but may be enabled by other capabilities implemented by processes.

HYPERGRAPH PARTITIONING

Consider the partitioning of a capability graph, which may be performed for geographical diversity, service partnering, or some other purpose. It is often more intuitive to consider the partitioning of the capability graph than the process graph, since the capability graph is closer to the strategic formulation of the business. Partitioning requires several considerations:

- The capability graph should not be partitioned in such a way that the edge cut of the process graph is high.
- The edge cut of the capability graph should not include high-strength edges between highly-dependent capabilities, since the movement of a capability to a service provider or to another geographical location may result in subtle or unsubtle changes to the capability.

Notation

For $d_1, d_2 \in \mathbb{Z}$ (integers), we denote $[d_1, d_2] \cap \mathbb{Z}$ by $d_1..d_2$.

A weighted, directed graph will be denoted $\mathcal{G}(V, E, w)$, where $V = \{v_i\}$ is a collection of vertices, $E = \{e_i = (v_j, v_m): v_j, v_m \in V\}$ is a collection of directed edges, and $w : E \rightarrow [0,1]$ is a weighting function.

For a graph $\mathcal{G}(V, E, w)$ and $v_i, v_j \in V$ given, define $\Sigma_{vi,vj}(\mathcal{G}(V, E, w))$, denoting

$\Sigma_{vi,vj}$ ($\mathcal{G}$ (V, E, w)) = $\mathcal{G}'$(V', E', w'), as follows. Define a new vertex $v' \notin V$,

$$V' = \{v'\} \cup V \setminus \{v_i, v_j\}, \text{ and } \forall\ a \in \{i,j\},\ E' = E \setminus (\{(v_a, v_b): b \neq i,j\} \cup \{(v_b, v_a): b \neq i,j\}).$$
$$\text{Also, } w'(v, v_b) = w(v_i, v_b) + w(v_j, v_b),$$
$$w'(v_b, v) = w(v_b, v_i) + w(v_b, v_j), \quad (1)$$
$$w'(v_c, v_d) = w(v_c, v_d)\ \forall\ c,d \notin \{i,j\}.$$

Roughly, because we will use $\Sigma_{vi,vj}$ only when $\exists\ r \in V$ such that (v_i, r), $(v_i, r) \in E$, $\Sigma_{vi,vj}$ *contracts* a graph at vertices v_i, v_j [2]. Now these two vertices are replaced with a new vertex v', the edges at v_i and v_j are also combined where there would otherwise be more than one edge in the same direction in parallel, and the associated weights are added.

Let $\delta_I(v)$ be the degree into a vertex v in a graph and $\delta_O(v)$ be the degree out of the vertex; i.e.,

$$\delta_I(v) = \text{card } \{(a,v) \in E\}, \quad (2)$$
$$\delta_O(v) = \text{card } \{(v,b) \in E\}.$$

Let $\delta(v) = \delta_I(v) + \delta_O(v)$ be the total degree of vertex v in a graph.

Formulation

Consider a company with a large office. At this office, the employees and IT systems of the company perform tasks associated with the operation of the company's business. These tasks depend on each other; e.g., suppose that p_i involves the audit of a field on a paper form, after which data from the form is entered into a computer program at task p_j. This process flow suggests a model where tasks p_i are vertices in a graph, and the flow of the work between tasks are directed edges γ_I in the graph. It should be noted that several edges may flow into a vertex, indicating several possible tasks immediately preceding. Several edges may flow out of a vertex, when there are multiple tasks that must immediately follow, which can furthermore be associated with a decision point, which chooses between two or more options for subsequent flow.

Define $\mathcal{G}_P = \{C \cup P, \Gamma, \omega\}$ representing the process graph. The $P = \{p_i\}$ are tasks and the $C = \{c_i\}$ are capabilities. The p_i are connected by directed edges $\Gamma = \{\gamma_i\}$ representing task transition, and weights $\omega(\gamma_i)$, indicating some cost or complexity associated with the task transition. For example, if $\gamma = (p_i, p_j)$ has high weight, the transition from p_i to p_j is complex and requires much interaction.

Define $\mathcal{G}_C = \{C \cup P, E, w\}$ representing the capability graph. The c_i are connected by directed edges $E = \{e_i\}$ representing enablement, and weights $w(e_i)$, indicating some cost or importance associated with the enablement connection. For example, if $e = (c_i, c_j)$ has high weight, then c_i is a critical enabler for c_j. An edge $e = (p_i, c_j)$ indicates that process p_i (potentially only partially) implements capability c_j.[1] It should be noted that a capability c_i may enable no other capabilities, while another c_j may enable many other capabilities. The capability graph is not required to be tree-structured or fully-connected.

We may then consider one hypergraph $\mathcal{G} = \{C \cup P, E \cup \Gamma, w'\}$, where $w'(e) = w(e)$ if $e \in E$ and $w'(e) = \omega(e)$ if $e \in \Gamma$.

A capability c with $\delta_I(c) = 0$ is called a "core enabler"; i.e., the capability may enable other capabilities, but itself does not depend for enablement on other capabilities in the model. A capability c with $\delta_O(c) = 0$ is called an "offering"; i.e., it may be an end capability

[1] The capability graph may also be represented as $\mathcal{G}_C = \{C, E, w\}$, without the connection to the process graph, but this simplified representation will not be used in this paper.

offered to an external or internal consumer or client.

Note that, since a process may implement more than one capability, we allow that $\exists\ c_j, c_m \in C$ with $c_j \neq c_m$ such that (p_i, c_j), $(p_i, c_m) \in \Gamma$. The implication for partitioning is that if c_j and c_m are placed in separate partitions, the partitioning algorithm will need to choose which partition will contain the process p_i.

Now suppose that the company needs to create k partitions of its capability graph, A_i, $i \in 1..k$, where one or more partitions may be moved to other geographical locations or a service partner. It may also result that the process graph is partitioned when the capabilities are partitioned. The graph model with associated edges suggests a graph partitioning optimization approach to the problem. Since truly optimal graph partitioning is an NP-complete problem [3-4], the complexity of the calculation can be unfeasibly high for a large, complex business and its associated processes.[2] Thus, we are adapting and developing algorithms for addressing this problem in a sub-optimal manner.

Optimality

Measuring the effectiveness of suboptimal partitioning requires comparison to optimal partitioning. In this section, we define the meaning of optimal partitioning for this type of example scenario. No partition may be empty of capabilities by definition, since we are partitioning the capability graph. A partition empty of processes indicates one of several things:

- An opportunity for contracting with a service partner, especially if the capability is a core enabler, but not if it is an offering, since then this capability becomes an offering of the service partner instead;
- An opportunity for geographical separation, especially if the capability is a core enabler, and more so if it is also an offering;
- An incomplete process model which fails to capture the characteristics of the business architecture relevant to the partitioning analysis. In this case, the analyst may refine the hypergraph with the additional process details.

Furthermore, the weight functions w and ω are independent of the vertex partitions; e.g., the complexity of a task transition does not change with partition. It is in the scope of our future work to explore partition-dependent weight functions, but this topic is not covered in this paper.

Given these parameters, optimality is achieved when the total weight of the edge cut on $\mathcal{G}$ is minimized. The number of partitions, k, and the process and resource graph are predetermined parts of the model. The analyst may learn from the result of the partitioning that a different value of k will produce a better partitioning. In this case, the analyst will change k and perform the partitioning again. It is beyond the scope of our work to make k a variable output of the algorithms.

Partitioning algorithms

In this section, we describe the details of the algorithm that we are using for partitioning the graphs. In all that follows, it is assumed that $k << \text{card}(C)$, indicating a non-trivial partitioning, where it is possible that each partition can contain several capabilities.

[2] In [3], NP-completeness requires $k \geq 2$ partitions, where the partition sizes K_i can vary, or be of constant size for $K_i \geq 3$. For $K_i = 2$, there is a polynomial time algorithm.

Initialization for the greedy algorithm

It is well known that the results of many sub-optimal algorithms are strongly dependent on the method used for initialization. In our class of scenarios, we initialize with consideration for the kind of partitioning result that we expect to achieve.

It is required that each partition A_i satisfy $A_i \cap C \neq \varnothing$ after partitioning. Offerings and processes must be placed in partitions with other capabilities. Thus, we create the set $\mathcal{C}_0 = C \setminus \{c \in C: \delta_O(c) = 0\}$, so that offerings are not included in the initialization. Then we initialize the partitions A_i each with one vertex of the smallest remaining δ_l as follows: for each $i \in 1..k$, let

$$c = \arg\min \{\delta_l(c'): c' \in \mathcal{C}_i\}, A_i = \{c\}, \mathcal{C}_{i+1} = \mathcal{C}_i \setminus \{c\}. \quad (3)$$

Initialization is performed from the core enablers and other enablers with δ_l small, since these are the capabilities most likely to be candidates for contracting with service partners, or to benefit from a dedicated staff, which may be geographically separated from the offerings (the so-called "back office").

For computational reduction, the hypergraph is also simplified, by contracting $\mathcal{G}$ using all $\Sigma_{vi,vj}$ with

- $v_i \in P$, $v_j \in C$, $(v_i,v_j) \in E$, and not $\exists c_j \in C$ such that $v_j \neq c_j$ and $(v_i,c_j) \in E$. These are the processes with exactly one capability link. This will not change the result, since the process will be placed in the partition with the capability giving the smallest edge cut.
- $v_i, v_j \in C$, $\delta_O(v_j) = 0$, and not $\exists c_j \in C$ such that $v_i \neq c_j$ and $(c_j,v_j) \in E$. These are the offerings with exactly one enabler. This will not change the result, since minimized edge cut results in these offerings residing in the same partition as the enabler.

In both cases, the contracted vertices are assumed to be in C. From this point, we use $\mathcal{G}$ to denote this contracted graph.

Greedy algorithm

The nominal greedy algorithm operates as follows, where $A_{j,i}$ is the partition A_j at iteration i, and $A_{j,0}$ is the initialized partition A_j from above. Initialize "waiting" set $\mathcal{V}_1 = \mathcal{C}_k$ from above. For each i such that $\mathcal{V}_l \neq \varnothing$ (that is, until there are no more vertices to partition), let $c = \arg\min\{\delta_l(c'): c' \in \mathcal{V}_i\}$, so that we partition from the vertices with the smallest degree in. Let $\mathcal{V}_{i+1} = \mathcal{V}_i \setminus \{c\}$ (removing c from the "waiting" set for the next iteration) and $\mathcal{P}_i = \{v \in C \cup P : (v,c) \in E\}$ (the set of all vertices connected by edge to this vertex). Let G be the (potentially disconnected) "subgraph" of $\mathcal{G}$ generated by $\{c\} \cup \mathcal{P}_l \cup (\cup_{j \in 1..k} A_{j,l})$, by which we may calculate our current edge cut. Then each member of $\{c\} \cup \mathcal{P}_i$ is placed in a partition to minimize the total edge cut of G, $W(G)$. Note that a member of $\mathcal{P}_i$ may have been assigned to a partition in a previous iteration, and can be moved based on the weight of its connections to c. Then each $A_{j,i+1}$ is formed from $A_{j,i}$ with these partition choices.

This complexity is manageable, since each process and each offering will either stay in its current partition, or be collocated with the vertex of the current iteration, c_i. This is addressed in Lemma 1.

Lemma 1: At each iteration of the greedy algorithm, for vertex c_i to be placed in a partition, with a edges to vertices v_j, at most 2^a comparisons must be performed to place a vertex c_i. Each comparison tests c_i against the partitions for lowest-weight edge cut, with

each connected vertex v_j either in its current partition, or grouped with c_i.

Proof: Let L_{i-1} be the total weight of the edge cut at iteration i-1. By induction, c_m for m = 1..i-1, and the elements of $C \cup P$, are partitioned such that L_{i-1} is minimized. Let $R_i = \{r \in C \cup P: (c_i,r) \in E \cup \Gamma\}$. Consider $v_j \in R_i$. If c_i is placed in a partition A_m different from that of v_j, $L_i - L_{i-1} \geq w'(c_i,v_j)$ added to σ, the total weight of all edges from c_i to vertices in other partitions. Thus, by assumption on L_i, moving v_j to another partition other than A_m only increases L_i more, by the edge cut to other vertices connected to v_j. It follows that moving v_j can only potentially reduce L_i if collocated in partition A_m with c_i. The result for all a processes and offerings follows.

It should be noted that in our application scenarios, most processes will be connected to one capability, and the number of offerings will be much smaller than the number of capabilities.

Enhancement through iteration

The results of this procedure can be enhanced by using the resulting partition as the initialization for the Kernighan and Lin approach [5], called "KL" here. KL is not described in this paper in the interest of space. KL iteratively moves vertices to other partitions to reduce the overall edge cut. The complexity of KL can be decreased if only those capabilities and processes with an edge in the edge cut are considered for iteration. In [5], it is discussed that 2-4 iterations normally bring convergence of the algorithm.

CONCLUSIONS AND FUTURE WORK

We are engaged in a project in cooperation with Prof. S. Smrikarova of the University of Rousse to explore representations of these business models in computers for effective computation. This work will be the subject of future papers.

We intend to extend the mathematical formulation in the future by exploring the implications of partition-dependent weight functions, which can represent dependence on the partitions for the realization of the business architecture. For example, if splitting the process graph in a given manner changes the complexity of the task transitions, this needs to be considered in the partitioning. We may also consider in the future making the number of partitions, k, a variable output of the algorithms.

REFERENCES

[1] M. Porter, "What is strategy?", *Harvard Business Rev.*, Nov.-Dec. 1996.

[2] R. Diestel, *Graph Theory, 2nd ed., Graduate Texts in Mathematics #173*, Springer (New York), 2000, pp. 16-18.

[3] M. Garey, D. Johnson, *Computers and Intractability -- A Guide to the theory of NP-Completeness*, W. H. Freeman and Co. (New York), 1979, p. 209.

[4] G. Karypis, V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM J. Sci. Comput.*, vol. 20, no. 1, 1998, pp. 359-392.

[5] B. W. Kernighan, S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell System Tech. J.*, vol. 49, 1970, pp. 291-297.

ABOUT THE AUTHORS

J. Q. Trelewicz, PhD, IBM Almaden Research Center, Phone +1 408 927 2600, e-mail: trelewicz@us.ibm.com

Jorge L. C. Sanz, PhD, IBM Almaden Research Center, Phone +1 408 927 2176, e-mail: jorges@us.ibm.com

Ankur Chandra, MS, IBM Almaden Research Center, Phone +1 408 927 2455, e-mail: achandra@us.ibm.com