Optimizing Enterprise Architecture Considering Different Budgets

Niklas Dohmen, Kevin Koopmann, and Simon Hacks[1]

Abstract: Enterprise Architectures (EA) are used to define the structure and operation of an organization and commonly find usage in the realization and modification of IT business strategies. We propose a technique to optimize the costs incurred between two layers of the EA, especially, considering differing departmental budgets. This is achieved through consideration of a flow problem aiming to optimize a graph consisting of different nodes, allowing budgets to be used by different departments. Additionally, we implement techniques previously published to allow operational and transitioning costs to be taken into consideration, in an effort to better reflect the organizational problems found in reality.

Keywords: Enterprise Architecture, Linear Integer Programming, Optimization, Department Budgets, Minimum-Cost Flow Problem

1 Introduction

IT-Projects in large companies realize technical requirements, which are requested by different business departments. To adjust IT-Projects to the overall strategy of the company, one way is to develop and manage enterprise architectures [AW09]. The most common definition of the term Architecture is found in the ISO norm ISO/IEC/IEEE 42010:2011. The norm defines architecture as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”. Therefore, an EA is a holistic view over all underlying structures, elements and their relations. It gives large enterprises a centralized framework, to consolidate their strategic plans and business mission. According to IT, it includes all IT-relevant components in the business, like information resource management, life cycle planning, systems or software reengineering. Furthermore, it is considered to be the backbone for coordination of actual and further development of the business IT systems and data [Ro94].

As in the revolution of information technology (especially in businesses), IT landscapes become increasingly larger [VSM10]. Because of that, the complexity of IT-systems is continuously growing, so that manual maintenance is unfeasible. Nevertheless, to keep systems preferably facile, it is a common practice trying to reduce the complexity of such systems, e.g., eliminating elements to reduce the complexity. There are several options on which parameters a system can be optimized, e.g., minimal quantity of elements or

[1] RWTH Aachen University, Research Group Software Construction, Ahornstr. 55, Aachen, 52074, {niklas.dohmen, kevin.koopmann}@rwth-aachen.de, hacks@swc.rwth-aachen.de

minimum operating costs. Aspects, which are important for an optimal solution, are depending on the usage, but often a hybrid solution is necessary. In this paper, we facilitate the approach from Hacks and Lichter [HL18b], in which they find an optimal solution considering transition costs for adding and removing applications to be economically reasonable. In larger businesses, departments commonly have department budgets to get an easier financial overview for managers. Additionally, different departments often share budgets or one department is using more than one budget. For example, this will be the case if two departments benefit from the same application. To achieve a more realistic optimization model, we integrate such department budgets into the proposed optimization model of Hacks and Lichter.

As the goal of this paper is to improve modeling of EAs in an optimization sense, we will not focus on suggestions for management decisions that could be derived from this data. We will represent EAs in the form of graphs, where elements are detailed as nodes and relations as edges. Interpretation of what constitutes elements and edges will be specific to the particular use case.

In section 2 of our paper, we introduce related work according to mathematically described EAs and their optimization models. Additionally, we present two previous approaches, on which this paper is based. The formal definition of EAs budgets and our approach to optimizing considering different budgets is introduced in section 3. To evaluate and prove our findings, we give an insight on how our approach is scaling in reality according to our implementation, which we will introduce in section 4. In section 5, we summarize our results and give a short perspective on further regarding topics.

2 Related Work

Different work has been done to describe and optimize EAs in a mathematical programming formulation fitted to their field of use.

As mentioned above, the most important related work, on which this paper is mainly based on, is worked-out from Hacks and Lichter [HL17, HL18b]. The authors optimize relations between two adjacent layers of EAs considering operational costs and transition costs to change systems from the as-is state to its optimal state. In this graph, optimization constraints are interpreted between two layers as triangles. These triangles consist of the connection between a needed capability of an upper layer element. These capabilities are realized by certain lower elements. The optimization model is solved using a linear integer program, which is even appropriable for realistic scenarios. In our approach, we extend this optimization problem from Hacks and Lichter regarding department budgets, which also can be not mutually exclusive.

A similar approach from Giakoumakis et al. [Gi12] focused on replacing existing services with new services without disrupting the enterprise. They formalized this problem into a graph and solved it using the multi-objective optimization problem, which considers departments, existing services, new services and IT-components. To apply changes in the EA, they link services and modules, department and services and attach transition costs for changes.

Another optimization approach is made by Franke et al. [Fr10]. They use a binary integer-programming model to find an optimal mapping between IT systems and processes based on needed functionalities. These functionalities are connected with processes they use and with certain fulfilling IT systems. In a next step, connections for the as-is state will be made through connecting processes and IT systems directly, which has to be optimized by applying change costs and operation costs.

MacCormack et al [MLB15] present the most complete optimization approach. They use Design Structure Matrices to capture the coupling between the components of the EA. The work extends existing research by also considering future states of the EA, considering layers, and generating measures that can be used to predict performance.

One fundamental different approach concerning transition costs is made from Lagerström et al. [LJE10] by using a probabilistic relation model with a meta-model based on literature research and evaluated through interviews and workshops. This is used to supply transitions costs estimation for large software projects. By providing a prediction and not optimizing the as-is state this approach differs from our formulation.

Further, Antunes et al. [ACB14] propose to use ontologies to analyze the EA. More concrete, they facilitate means of description logic to enable an automatic analysis of the underlying EA model.

Such optimizations need to be guided by processes. Hacks and Lichter [HL18a] propose a process to keep the central EA model up-to-date. The optimization approaches can be placed in the quality assurance step to suggest improvements to the architects. Hartmann [Ha17] illustrates a process for the alignment between software development and EA. He embeds the EA governance process into the software development process and, though, enforces the governance.

3 Modeling a Budgeted Enterprise Architecture Optimization

Before presenting our approach, we first give a quick overview of how EAs are modeled in literature. Later on, we especially take care of the foundations, which are already acquired by Hacks and Lichter [HL17, HL18b]. Based on these foundations we provide our optimizing approach.

3.1 Towards an Optimization Model

According to Winter and Fisher [WF06], enterprise architectures are most commonly composed of hierarchical layers. Each of these layers consists of various architectural artifacts, which can be connected through different relations with elements of the same layer. Additionally, artifacts on a layer can be explicitly influenced by elements on the superseding layer, reflecting a priority of importance of decisions undertaken on higher levels of the architecture [MMT00].

For instance, an enterprise architecture could consist of business entities such as *Human Resources*, *Enterprise Resource Planning* and *Liquidity Management*, each requiring a certain set of capabilities. Such capabilities could be functions such as *Hiring* or *Stock Trading*. This is modeled by directed edges from the business entities to the capability nodes. Furthermore, applications selectively implement certain capabilities, represented through edges from the capabilities to implementing applications. Fig. 1 shows such an exemplary architecture naming the layers as *Cost Centers* and *Applications*.

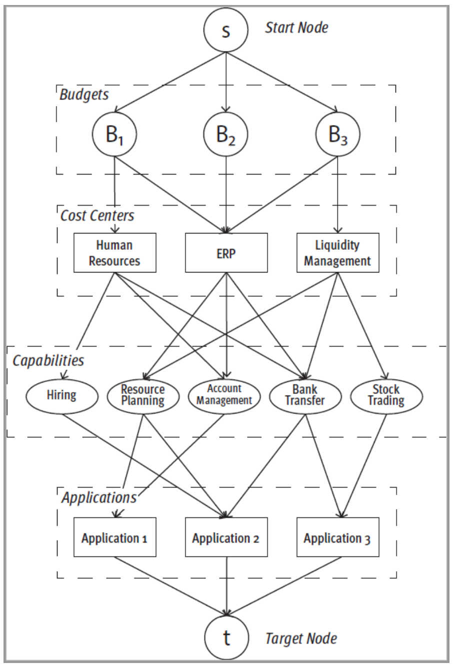

Fig. 1: Concrete example for two layers in an enterprise architecture (reproduced from Hacks and Lichter [HL18b]

An example of an application of this concept could be the simplification of an IT business infrastructure in order to reduce costs. A business resource may require certain capabilities fulfilled by applications on the application layer. In this case, an optimization model is suitable to reduce the operational cost of the architecture by selecting a set of applications that implements all required capabilities at the smallest possible cost to the enterprise.

In order to be able to apply common graph algorithms to the solution of problems related to the enterprise architecture, we will model these architectures as directed graphs, accordingly to Hacks and Lichter [HL18b]. Vertices will represent an element of the architecture describing an aggregation hierarchy, and the edges between these vertices mirror dependencies between elements on different layers of the architecture.

In our subsequent analysis of the optimization problem for enterprise architectures, we will consider only the interactions between two directly adjacent layers of an overall architecture. This is permissible because decisions on higher layers of the architecture reduce the degree of freedom on subsequent layers [MMT00], and greatly simplify understanding of the optimization process. To perform an analysis of the entire architecture, simply repeat the analysis in top-down for each interface between two layers.

Hacks and Lichter [HL17] present different objectives that could be used to optimize the model, such as minimal coupling, amount of lower layer elements or operational costs. In real-world applications, we often find certain business entities, such as departments, to be restricted to a specific budget. These budgets are important for business managers to estimate the project and department costs. It is common that various departments are sharing the same IT infrastructure so that each department has its own scot, which needs a specific budget. We will extend the model by considering these budget restrictions, while also incorporating transitional costs for the introduction or decommissioning of currently active applications in the architecture. This seems to be warranted as moving away from old applications or implementing new ones often arises substantial costs necessary for example to educate users, transfer data or resolving other dependencies.

3.2 Foundations

Building on this initial understanding of enterprise architecture modeling and objectives in its optimization, we will now introduce a more formal representation of these architectures.

As described by Hacks and Lichter [HL17, HL18b] we want to represent the enterprise architecture as a quadruple $EA = (\mathfrak{L}, \mathfrak{C}, E, R)$ where $\mathfrak{L}$ is an ordered set of architectural layers, $\mathfrak{C}$ describes a set of sets of capabilities, E is a set of architectural elements, and R represents a set of relations between these elements and capabilities.

We assume each layer $\mathrm{L} \in \mathfrak{L}$ to consist of architectural elements so that $\mathrm{L} \subset \mathrm{E}$, and to be disjoint as prescribed:

$$L_i \cap L_j = \emptyset \mid \forall L_i, L_j \in \mathfrak{L} \quad i \neq j \tag{1}$$

For each layer element, $ul_i \in L_j$, of the upper layer there may be an associated budget, $\mathfrak{b}_i \in \mathbb{N}_0$. For each element on the lower layer, $ll_i \in L_{j+1}$, there is an associated operational cost, p_i, and a transitional cost, tp_i.

Furthermore, we place capabilities, $c_i \in C$, between two adjacent layers. There exists between every neighbored layers, L_i and L_{i+1}, a set of capabilities, $C \in \mathfrak{C}$. As each capability is associated with a specific business requirement, we assume all capabilities to be unique to a set of capabilities between two adjacent architectural layers:

$$C_i \cap C_j = \emptyset \mid \forall\, C_i, C_j \in \mathfrak{C}\ i \neq j \tag{2}$$

The set of relations consists of tuples of architectural elements and capabilities, which couple upper layer elements with capabilities and capabilities with lower layer elements:

$$R \subset \{(e,c): e \in L_i, c \in C_i\} \cup \{(c,e): c \in C_i, e \in L_{i+1}\} \tag{3}$$

This object EA is the subject of optimization.

3.3 Modeling Cash Flows in Enterprise Architectures

Hacks and Lichter [HL17] present a solution to the optimization problem for enterprise architectures by introducing intermediate relations between all elements of an adjacent layer constituting a bipartite graph. In order to be able to incorporate budget constraints into our model in a meaningful manner and benefit from graph algorithms, we chose a different approach by modeling cost flows originating from budget nodes.

In order to obtain the desired solution, we apply a modified maximum flow problem based on the input parameters of the problem. The maximum flow problem is concerned with obtaining a maximum flow through a single-source single-sink flow network by assigning a feasible flow, $f_e \to \mathbb{R}_+$, to all edges, $e \in \mathfrak{E}$, such that the total flow $\sum f_{i,t}$, $(i,t) \in \mathfrak{E}$ into the sink is maximal. Additionally, the maximum flow problem imposes an constraint by limiting the flow across an edge to a certain capacity limit, $\mathfrak{c}$. Naturally, in a flow network, the flow out of a node must be equivalent to the flow entering the node, excepting source and sink nodes. Instead of considering a maximum flow across the network, we want to minimize cost flow, and we limit the flow across certain budget-related arcs to a specific budget value. In linear programming terms, we formulate this objective in Eq. (4), and formulate these initial constraints in Eq. (5) and (10).

We begin to construct a graph with vertices, $\mathfrak{V}$, consisting of elements from two adjacent layers, L_i and L_{i+1}, as well as the related capabilities, C_i. We also start with a set of edges, $\mathfrak{E}$, equivalent to the set of relations, R, for these adjacent layers.

We also add a start node, s, as an entry point for a flow algorithm. Furthermore, for each budget, $\mathfrak{b}_i$, it is necessary to construct a node, $b \in B$, and add relations to all upper layer elements associated with this budget. For additional relations, we introduce for each budget between, s, and the budget node, b. Next, we apply constraints to these relations to ensure a maximum cost flow equivalent to the budget.

From each of the lower layer elements, we construct additional relations to the sink node, t. The activation of a lower layer element bears operational costs and may additionally arise transitional costs. The deactivation of a lower layer application may also arise transitional costs, but no operational costs. To incorporate this requirement into our cash flow model, we must also enforce an additional constraint to force the flow on these edges to be equivalent to exactly the cost raised by (de-)activation of the element.

This model represents the flow of monetary resources originating from operational budgets through the architecture. To optimize towards minimal costs, we can now simply attempt to minimize the flow of cash through the network.

3.4 Final Modeling

To summarize our findings, we will now combine the previous results to formulate the following selection problem for two adjacent layers of the architecture:

- All upper layer elements, ul_i, are to be implemented. We also refer to these elements as *cost centers*.
- The implementation of a distinct upper layer element ul_i requires fulfilling a known subset of capabilities, $c_i \subset C_j$. We refer to those subsets indicating the necessary capabilities for the operation of a cost center as the cost center's *capability set*.
- Fulfilling a certain capability, c_j, requires the activation of at least one element of a known subset, S_j, of lower layer elements, ll_k. We refer to these lower layer elements as *applications*. Furthermore, we refer to the subset, S_i, which indicates the applications suitable for implementing a capability as the capability's *implementing set*. These implementing sets are not mutually exclusive. If an application belongs to multiple implementing sets, then selecting the application for the solution would simultaneously satisfy all implementing sets it is contained in, and, thereby, all corresponding capabilities.
- An application, which has been activated, arises an operational cost, p_k.
- Implementing an application, which has previously been inactive, arises an additional transitioning cost, tp_k, and vice versa.
- All cost centers are subject to budget constraints. These budgets are distributed among multiple cost centers and are not mutually exclusive. That is, a cost center can receive funds from multiple designated budgets, and a budget can be designated for multiple upper layer elements.

The problem we consider is to determine which applications should be implemented in order to minimize operational and transitional costs, as well as how the budgets are consumed by the cost centers in order to realize all capabilities.

In order to accomplish this, we derive a network flow minimization problem from the selection problem and formulate the following integer program. We first construct a graph from our problem description. Let $\mathfrak{V}$ denote the set of vertices, and $\mathfrak{E}$ the set of edges.

Let s be a source, and t be a sink node, and let $\mathfrak{V}_-$ be the set of vertices $\mathfrak{V}$ excluding these nodes s and t. Construct a node, b_i, for each budget, $\mathfrak{b}_i$, and edges, (b_i, ul_j), for each cost center, ul_j, assigned to this budget. Construct one edge, (s, b_i), for each budget node, b_i.

Construct a capability node, c_i, for each capability, $c_i \in C_k$, and let (ul_j, c_i) be the edges from each cost center, ul_j, to the capabilities it requires. Then let (c_i, ll_k) be the edges from the capability, c_i, to all applications, ll_m. Finally, construct one edge, (ll_m, t), for each application, ll_m. Fig. 2 shows an example architecture modeled accordingly.

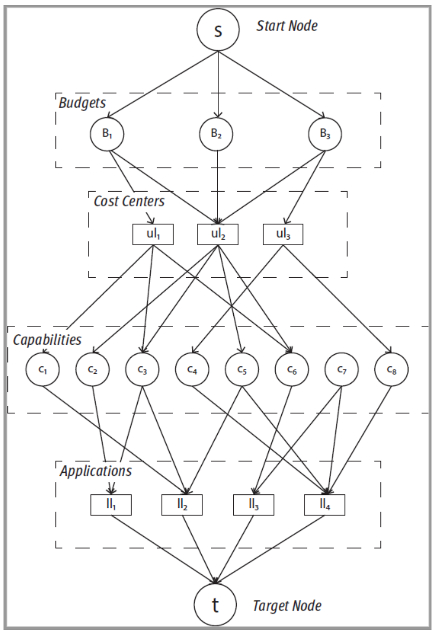

Fig. 2: Abstract example for two layers in an enterprise architecture

Let $\mathfrak{Z}$ be the set of applications already implemented (as to be considered for transitional analysis), and $z_i \in \mathfrak{Z}$ be the implementation state of the application, ll_i. $f_{(i,t)}$ designates the flow between two nodes, (i, t), and i_i is a boolean determining whether an application, ll_i, is to be implemented. Let t_i be a boolean variable indicating whether a transition is necessary for an application, ll_i. These are the variables of the integer program.

The integer program describing the problem we have specified can now be defined as follows:

$$\min \qquad \sum_{e \in \mathfrak{E}} f_e \qquad (4)$$

s.t. $$f_{(i,j)} \leq b_j \;\mid\; \forall\, (i,j) \in \mathfrak{E}, j \in B \tag{5}$$

$$\sum_{(i,j)\in\mathfrak{E}} i_i \geq 1 \;\mid\; \forall\, i \in \mathfrak{C} \tag{6}$$

$$\mathfrak{t}_i \geq i_i - z_i \;\mid\; \forall (i,t) \in \mathfrak{E} \tag{7}$$

$$\mathfrak{t}_i \geq z_i - i_i \;\mid\; \forall (i,t) \in \mathfrak{E} \tag{8}$$

$$f_{(i,t)} = (i_i * p_i) + [tp_i(z_i - i_i)^2] \;\mid\; \forall\, (i,t) \in \mathfrak{E} \tag{9}$$

$$\sum_{(i,j)\in\mathfrak{E}} f_{(i,j)} = \sum_{(i,j)\in\mathfrak{E}} f_{(j,i)} \tag{10}$$

Eq. (4) describes the objective function. As we are attempting to minimize accruing costs, our objective is the minimization of flow across all incoming arcs at the sink node, t.

Eq. (5) enforces budget limits on the edges connecting the source, s, to the budget nodes.

Eq. (6) ensures that at least one of the applications connected to a capability is activated.

Eq. (7) and (8) set the variable, $\mathfrak{t}_i$, that indicates whether a transition of an application, i_i, is necessary to the correct value.

Eq. (9) enforces a valid flow on the edges connecting the applications to the sink, t; that is, the flow is either 0 if the application is disabled, or exactly the sum of operational and transitional costs. This is accomplished by adding up the product of the boolean variable, i_i, defining whether the application is to be activated and the corresponding cost, p_i, and the transitional cost function, tp_i.

Eq. (10) is a constraint common to all flow problems and forces the amount of flow entering a node to be equal to the amount of flow leaving it.

Thereby, we have implemented all requirements originally specified in the definition of the selection problem for budgeted enterprise architecture optimization.

The variables, i_i, will contain the implementation state for an application, ll_i, in the optimal solution, and $f_{(i,j)}$ will describe the amount of cash flow between the two entities belonging to i and j.

3.5 Applying the Optimization Model

To apply the optimization model to an example graph, the budget constraints have to be implemented as nodes between the cost centers and the start node as described in subsection 3.4. The edges between the start and the budget nodes function as the budget constraints according to Eq. (5). Cost centers are provided by different budgets, which are represented as the connecting edges between those nodes. Fig. 3 shows an example of such a graph where the cost centers *Human Resources* and *Liquidity Management* are each sharing their budgets with *ERP*.

Here *Application 1* and *Application 2* are not yet used in the model and can be activated by applying the denoted transition costs. To calculate an optimal solution, we apply our above-described approach to such a graph with the given costs. Fig. 3 sketches the optimal solution calculated by our model regarding a minimum cost-flow. This optimal solution suggests to include *Application 1* and *Application 2* with a total cost of 70 for each consisting of transition and operating costs. The total costs for this solution amounts to 165 and is reflected to the minimum cost-flow in the network.

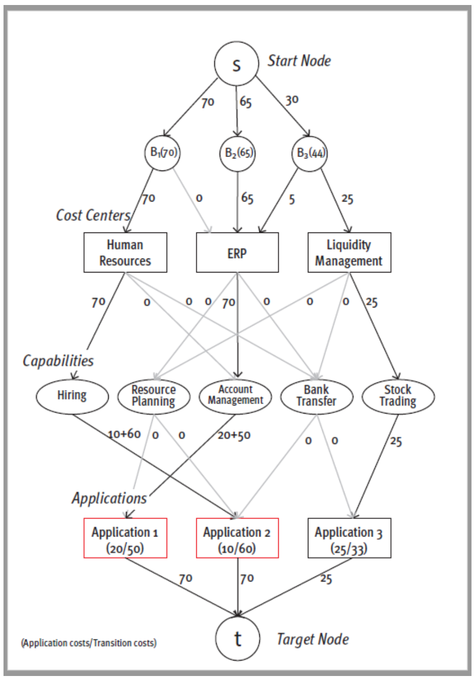

Fig. 3: Solution for concrete example with cash flow

We have provided a copy of the source code of our implementation in a git repository[2].

Building on our implementation, we will provide both a reasoning for our expectations pertaining to the algorithm's runtime, as well as an experimental evaluation of this runtime. This will allow us to predict the practical value of the solution we have presented.

[2] https://git.rwth-aachen.de/kevin.koopmann/swcseminar

4 Evaluation

In order to evaluate the performance of the modeling approach we have presented and to apply it to exemplary inputs, we felt it be warranted to implement rudimentary software capable of generating and processing appropriate enterprise architecture representations.

To accomplish the desired goals we have reformulated the objectives and constraints from the previous section using the Pyomo[3] optimization modeling DSL and utilized the Gurobi[4] solver to derive an optimal solution.

4.1 Scalability

Runtime Expectations As is evident from Eq. (4), the objective function of our model is a linear expression and dependent upon the flow on edges between applications and the sink node, t. The number of linear objective terms in the model is therefore equivalent to the number of applications.

We expect an increase in the number of edges, for instance, due to a positive variation of the amount of budget, cost center and capability nodes, to lead to linear increases in runtime due to the additional linear constraint terms.

Experimental Runtime Evaluation For experimental evaluation, we have generated random architecture representations to be processed by our software with different values for the number of budgets, cost centers, capabilities and applications. For the first part of the evaluation, all density values were set to 0.1, and for each of these value sets the average runtime and variable count over 20 runs was determined on a system with 102 GFlops of floating-point performance and 8 GB of RAM. At this density we expect the variable count to be closely related to the number of edges.

Fig. 4 shows the resulting runtimes for varying numbers of budgets, cost centers, capabilities and applications with fixed ratios at a connectivity density of 0.1. We have decided on these values, as they represent a common balance between the various entities, even though the connection density is higher than in most real-world architectures [LJE10]. However, this reduces the impact of time spent in pre-solving and optimization phases on overall measurement results, and is still representative of actual use cases, as we will show the relationship between density and runtime to be inversely proportional.

Time measured includes time spent on problem generation by Pyomo and time in the Gurobi solver, including preprocessing and pre-solving time. Fig. 4 shows the runtime of the solver compared to the number of variables corresponding to edge and vertex counts.

[3] http://www.pyomo.org/
[4] https://www.gurobi.com/

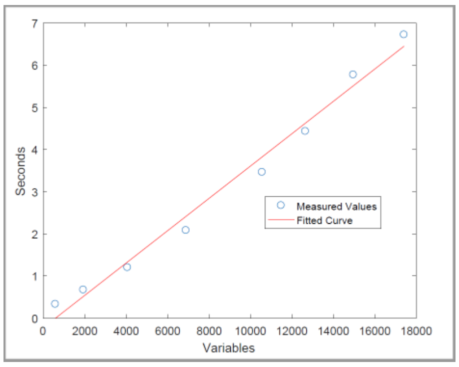

Fig. 4: Time spent in solver and linearly fitted corresponding model

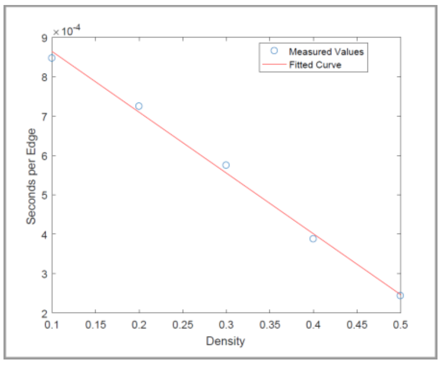

Fig. 5: Seconds per edge depending on graph connectivity

This shows that with increasing architectural complexity measured in terms of connectivity between different entities in the organization graph, in accordance with our expectations, the runtime of our algorithm increases linearly. We have found that this relationship holds in most cases for architectures with equivalent connection densities.

By varying connection densities in a second test run, we confirm that execution time is not solely dependent on the edge count, but on connectivity as well. Comparison of varying density values against the average time spent in the solver for each edge in the architecture yields Fig. 5. This is to show that connection density is inversely linearly related to execution time.

We assume that the reduction in runtime in very highly connected architectures compared to architectures with lower edge counts is related to particular details regarding the implementation of the relaxation or SOCP algorithms of the solver we have used, or in the simplification of certain problem parameters in highly connected graphs. However, a detailed analysis of this effect is beyond the scope of this article.

In the following section, we will analyze the consequences this entails for the application of this algorithm on real-world optimization problems.

4.2 Value for Real-World Applications

As is evident from our runtime analysis, the execution time of the Integer Program we have described is dependent upon a multitude of factors, including the number of entities on different architectural layers and the connectivity between such entities. There was a tendency towards a linear correlation between variable or edge count and runtime for all evaluated architectures.

Even at linear runtimes, this may still present challenges for scalability towards optimizing extremely large-scale enterprise architectures. Even though, such optimizations will not be performed frequently in most cases, and can often be executed on large-scale systems, it is conceivable that on certain architectures execution time might prove to be prohibitive.

However, most architectures encountered in real-world applications seem to be of a rather limited size in comparison to the scenarios of our evaluation model that required high execution times. Lagerström et al. [LJE10], for instance, considered an architecture consisting of 407 nodes and 1157 edges, translating to a density of just 0.00698 in our notation. Evaluating a similar architecture on our test system consistently yielded a runtime of less than one second.

5 Conclusion

Information Technology projects within enterprises demand novel solutions of the organization problem, devised with awareness of business requirements such as departmental budgets and transition costs. With the ever-increasing relevance of IT systems in business cases, we predict a correlating rise in demand for consolidation of such systems.

We have shown how this consolidation can currently be performed using optimization models and presented a different, novel model taking the business need of budgeting for varying cost centers into account, and incorporated the transition cost model introduced by Hacks and Lichter [HL17, HL18b].

Furthermore, we have demonstrated through our runtime evaluation that our approach performs adequately for most real world use cases, which we showed for both synthetic test cases and instances from literature.

Currently, the approach considers budgets and transitional as well as operational costs only at a single point in time. In reality, certain costs and budgets may constantly change; an extension of the approach to determine the ideal point of investment in time could address this challenge.

Bibliography

[ACB14] Antunes, C.; Caetano, A.; Borbinha, J.: Enterprise Architecture Model Analysis Using Description Logics: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, 2014; pp. 237–244.

[AW09] Aier, S.; Winter, R.: Virtual Decoupling for IT/Business Alignment – Conceptual Foundations, Architecture Design and Implementation Example. In Business & Information Systems Engineering, 2009, 1; pp. 150–163.

[Fr10] Franke, U. et al.: IT Consolidation: An Optimization Approach: 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, 2010.

[Gi12] Giakoumakis, V. et al.: Technological architecture evolutions of information systems: Trade-off and optimization. In Concurrent Engineering, 2012, 20; pp. 127–147.

[Ha17] Hartmann, A.: Enterprise Architecture als Katalysator zwischen Qualität, Effizienz und Governance. In (Eibl, M.; Gaedke, M. Eds.): INFORMATIK 2017. Gesellschaft für Informatik, Bonn, 2017; pp. 2121–2126.

[HL17] Hacks, S.; Lichter, H.: Optimizing Enterprise Architectures Using Linear Integer Programming Techniques. In (Eibl, M.; Gaedke, M. Eds.): INFORMATIK 2017. Gesellschaft für Informatik e.V., Bonn, 2017; pp. 623–636.

[HL18a] Hacks, S.; Lichter, H.: Towards an Enterprise Architecture Model Evolution. In (Czarnecki, C.; Sultanow, E.; Brockmann, C. Eds.): Workshops der Informatik 2018. Gesellschaft für Informatik e.V, Bonn, 2018.

[HL18b] Hacks, S.; Lichter, H.: Optimierung von Unternehmensarchitekturen unter Berücksichtigung von Transitionskosten. In HMD Praxis der Wirtschaftsinformatik, 2018, 55; pp. 928–941.

[LJE10] Lagerström, R.; Johnson, P.; Ekstedt, M.: Architecture analysis of enterprise systems modifiability. A metamodel for software change cost estimation. In Software Quality Journal, 2010, 18; pp. 437–468.

[MLB15] MacCormack, A.; Lagerstrom, R.; Baldwin, C. Y.: A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility. In Harvard Business School Working Paper, 2015.

[MMT00] Mesarovic, M. D.; Macko, D.; Takahara, Y.: Theory of hierarchical, multilevel, systems. Elsevier, 2000.

[Ro94] Rood, M. A.: Enterprise Architecture: Definition, Content, and Utility: Proceedings of the Third Workshop on Enabling Technologies. Infrastructure for Collaborative Enterprises. IEEE, New York, NY, USA, 1994; pp. 106–111.

[VSM10] Vodanovich, S.; Sundaram, D.; Myers, M.: Research Commentary—Digital Natives and Ubiquitous Information Systems. In Information Systems Research, 2010, 21; pp. 711–723.

[WF06] Winter, R.; Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture: 10th IEEE International Enterprise Distributed Object Computing Conference Workshops. IEEE, New York, NY, USA, 2006; pp. 30–38.