EFFICIENT LARGE-SCALE SERVICE CLUSTERING VIA SPARSE FUNCTIONAL REPRESENTATION AND ACCELERATED OPTIMIZATION

QI YU
College of Computing and Information Science,
Rochester Institute of Technology
qi.yu@rit.edu

Clustering techniques offer a systematic approach to organize the diverse and fast increasing Web services by assigning relevant services into homogeneous service communities. However, the ever-increasing number of Web services poses key challenges for building large-scale service communities. In this paper, we tackle the scalability issue in service clustering, aiming to accurately and efficiently discover service communities over very large-scale services. A key observation is that service descriptions are usually represented by long but very sparse term vectors as each service is only described by a limited number of terms. This inspires us to seek a new service representation that is economical to store, efficient to process, and intuitive to interpret. This new representation enables service clustering to scale to massive number of services. More specifically, a set of anchor services are identified that allow to represent each service as a linear combination of a small number of anchor services. In this way, the large number of services are encoded with a much more compact anchor service space. Despite service clustering can be performed much more efficiently in the compact anchor service space, discovery of anchor services from large-scale service descriptions may incur high computational cost. We develop principled optimization strategies for efficient anchor service discovery. Extensive experiments are conducted on real-world service data to assess both the effectiveness and efficiency of the proposed approach. Results on a dataset with over 3,700 Web services clearly demonstrate the good scalability of sparse functional representation and the efficiency of the optimization algorithms for anchor service discovery.

1. Introduction

Service oriented computing holds tremendous promise by exploiting Web services as an efficient vehicle to deliver and access various functionalities over the Web. The past few years have witnessed a fast boost of Web services due to the wide adoption of service-oriented computing in both industry and government. The proliferating services have formed a functionality-centric repository, through which key computing resources can be conveniently accessed via the standard Web service interface. However, the ever increasing number of Web service poses key challenges to discover services with user required functionalities. A rigorous and systematic methodology is in demand for efficiently and accurately searching user desired services from a large and diverse service repository.
Universal Description Discovery and Integration (UDDI) provides a standard registry service to publish and discover Web services. To make a service searchable, the service provider needs to first publish its service in the UDDI registry. Nonetheless, as service providers are autonomous in nature, it is infeasible to enforce them to publish their services in the registry. In fact, most service vendors choose to directly advertise their services via their own websites. Furthermore, when change occurs to a published service, the service entry in the UDDI may need to be updated to ensure consistency. This gives rise to additional maintenance cost for service providers. Recent statistics show that more than 50% services in the public UDDI registries are invalid.

Service search engines have gained increasing popularity by automatically collecting service descriptions using crawlers. Service descriptions are then indexed and matched against user’s searching keywords. One key impediment towards the wide adoption of service search engines has been the poor search quality resulted from simple keyword matching. While keyword matching may perform reasonably well on regular Web pages, it suffers from service descriptions, which are usually generated from application programs using Web service deployment tools. Many service descriptions are comprised of very limited number of terms, most of which are not proper words. Therefore, there is a low chance for a service description to match a searching keyword even though the service may provide the exact user-desired functionality.

Clustering techniques have been adopted to improve the quality of service discovery\cite{9,21,7}. Service clustering computes the similarity among services to group together relevant services into homogeneous service communities. Clustering enables services to be discovered by exploiting the proximity to other services. Consider two similar services, $S_1$ and $S_2$, where $S_1$ contains the searching keyword while $S_2$ does not. Through service clustering, both $S_1$ and $S_2$ will be returned as they are deemed to provide similar functionality desired by the user. In this way, the search quality can be dramatically improved. Furthermore, service discovery can be directed to only relevant service communities so that more efficient performance is achieved.

As the number of services keeps increasing, building service communities over large-scale Web services arises as a central challenge. Following traditional document clustering, each service description $s_i$ is denoted by a term vector, in which $s_i(j)$ is set to the normalized frequency (or other metrics such as TF/IDF) of $t_j$ if $t_j \in s_i$ and 0 otherwise. The length of $s_i$ is equal to the size of the term dictionary, which consists of the distinct terms over all service descriptions. Most service descriptions are generated from program source codes, where various naming conventions may be used by different developers. This results in a large number of distinct terms especially when scaling to a massive number of services. For example, in one of the real service dataset used in our experiments\cite{24}, we extract around 17,000 distinct terms from over 3,700 service descriptions. However, each service description only consists of 20 distinct terms on average. Therefore, the term vector $s_i$ will be very
large and extremely sparse (density is around 0.1% in our dataset) when dealing with large-scale services.

Simple clustering algorithms, such as K-means, scales well with the number of services. The similarity between two term vectors is usually computed based on the number of terms that co-occur in these two vectors. However, directly applying these algorithms to large-scale service clustering usually leads to poor clustering quality because the term vectors for service descriptions are extremely sparse and hence less likely to share common terms. Advanced algorithms, such as matrix factorization based ones (e.g., SVD co-clustering and NMTF), have been demonstrated to be more effective in dealing with limitations of service descriptions and generate high-quality service communities. However, it remains unclear how these algorithms can handle extremely large and sparse term vectors. In addition, the high computational cost also prohibits them from scaling to a massive number of services. In this paper, we address the scalability issue in service clustering, aiming to discover service communities over very large-scale services. It is worth to note that our focus is on publicly available services from diverse organizations, where various description styles and naming conventions may be adopted. For services developed and used only within a single or several organizations, they usually follow a set of unified description rules with specially designed ontologies. Therefore, the ontologies used to design services would offer an idea way to organize these services.

The central idea of large-scale service clustering is that instead of using a highly diverse dictionary of terms, we seek a much more succinct representation of service descriptions. Inspired by recent works on sparse coding, we devise a novel strategy to learn a set of “anchor” services, which form a new dictionary to encode the service descriptions. This allows each service to be represented as a linear combination of a small number of anchor services. In general, the number of anchor services is smaller than the number of distinct terms with several orders of magnitude. Hence, the large number of services are encoded with a much more compact dictionary of anchor services. The new representation is essentially a projection onto the anchor service space. Similarity between services is determined based on how they are related to a small number of anchor services. Simple clustering algorithms, like K-means, can then be applied to this compact representation to efficiently and accurately cluster large-scale services. Some promising results have been reported in our preliminary work. Despite service clustering can be performed much more efficiently in the compact anchor service space, discovery of anchor services from large-scale service descriptions may incur high computational cost. We develop principled optimization strategies for efficient anchor service discovery. We demonstrate the effectiveness of the proposed clustering algorithm and efficiency of the optimization strategies via extensive experiments on two real-world service datasets.

The remainder of the paper is organized as follows. We discuss some related works in Section 2, which provide a background overview of the proposed approach. We present the details of sparse functional representation in Section 3. We use a
concrete example to explain how sparse functional representation works and provide intuitive justifications of its effectiveness. We also propose a novel clustering scheme that integrates information from both the anchor service space and the term vector space. We develop efficient optimization algorithms for anchor service discovery in Section 4. We apply sparse functional representation to two real-world service datasets and assess its effectiveness in Section 5. We conclude in Section 6.

2. Related Work

Service clustering and related technologies have been increasingly adopted to facilitate service discovery or other key tasks in service computing, such as service composition and service ontology construction.

Clustering has been a central technique to improve the accuracy of service search engines. Woogle, a Web service search engine, performs term clustering to generate a set of high-level concepts, which are then used to facilitate the matching between users’ queries and the service operations. Similarly, term clustering is also used in to facilitate service discovery and composition. Basic K-means algorithms are usually used and the similarity between terms are evaluated based on their co-occurrence in the service descriptions. Quality Threshold (QT) clustering is employed to cluster Web services in to bootstrap the discovery of Web services. WSDL descriptions are carefully parsed and important components are extracted, which include content, types, messages, ports, and name of the Web service. Weights are assigned to each component when similarity between two services is evaluated. More complicated algorithms, such as Probabilistic Latent Semantic Analysis (PLSA), have also been applied to service clustering and discovery. A SVD based algorithm is adopted to achieve the co-clustering of services and operations in. Co-clustering exploits the duality relationship between services and operations to achieve better clustering quality than one-side clustering.

Given the limitations of the WSDL service descriptions, some recent proposals seek to explore external information sources, such as Wordnet and Google, to improve service clustering and discovery, In matrix factorization and the semantic extensions of service descriptions have been integrated for service community discovery. The integration has the effect of placing the extended semantics into the context of the service, which more effectively leverages the extended semantics to benefit community discovery. As Web services usually consist of both WSDL and free text descriptors, novel approaches have been developed in to integrate both types of descriptors for effective bootstrapping of service ontologies. Another important piece of information that is complementary to service descriptions is the service tags that users use to annotate services. A novel approach, referred to as WTCIuster, is developed in that exploits both WSDL documents and service tags for Web service clustering.

\(^a\)http://wordnet.princeton.edu/
Table 1. Symbols and Descriptions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>set of services</td>
</tr>
<tr>
<td>$s_j, t_i$</td>
<td>the $j^{\text{th}}$ service and $i^{\text{th}}$ term</td>
</tr>
<tr>
<td>$X, A, W, Z$</td>
<td>matrices</td>
</tr>
<tr>
<td>$X'$</td>
<td>the transpose of matrix $X$</td>
</tr>
<tr>
<td>$X_{ij}$</td>
<td>the element at the $i^{\text{th}}$ row and $j^{\text{th}}$ column of matrix $X$</td>
</tr>
<tr>
<td>$x_j$</td>
<td>the $j$-th column vector of matrix $X$</td>
</tr>
<tr>
<td>$x_j(i)$</td>
<td>the $i$-th element of $x_j$</td>
</tr>
<tr>
<td>$x'_i$</td>
<td>the $i$-th row vector of matrix $X$</td>
</tr>
</tbody>
</table>

3. Sparse Functional Representation for Service Clustering

Sparse functional representation aims to seek a compact dictionary of anchor services to succinctly represent large-scale services. Consider a set of services $S = \{s_1, \ldots, s_m\}$, where each $s_i \in \mathbb{R}^n$ is denoted as a term vector and $n$ is the size of the term dictionary that is comprised of all distinct terms extracted from the services in $S$. By mapping terms into rows and services into columns, $S$ is conveniently represented by a two dimensional service matrix $X \in \mathbb{R}^{n \times m}$. Each entry $X_{ij} \in X$ is set to the normalized frequency of term $t_i$ in service $s_j$ and zero if $t_i \notin s_j$. Table 1 provides a quick reference to a set of symbols that are commonly used in the paper.

As discussed in Section 1, due to the diverse naming conventions used in service descriptions, the size of the term dictionary increases dramatically with the number of services in $S$. This will result in a huge and extremely sparse matrix $X$. For example, in our experiments, a $16,884 \times 3,738$ matrix is constructed from a real-world service dataset with 3,738 services. $X$ consists of approximately $6.3 \times 10^8$ entries, among which only 0.1% are nonzero, implying a 99.9% sparsity ratio. This poses a set of key challenges for clustering large-scale services. First, scalability arises as a significant challenge for storing and processing a large service matrix whose size grows quickly with the number of services. Second, the highly sparse term vectors are a key impediment for applying many clustering algorithms to generate high-quality clusters as sparse vectors are less likely to share common terms.

3.1. Sparse Functional Representation

The above observation implies that a large and diverse term dictionary does not provide a suitable representation for large-scale service clustering. Instead, concepts with coarser granularity may be more instrumental to produce a compact and cohesive service representation. Hence, we aim to seek a new service representation that is economical to store, efficient to process, and intuitive to interpret. This new representation will enable service clustering to scale to massive number of services. Inspired by recent advances in sparse coding $^{12,25}$, we devise a novel sparse
functional representation (SFR) strategy to discover a set of so called “anchor services”. The anchor services are expected to capture the high-level functionalities of services while significantly compressing the original term vector space. More specifically, SFR seeks a matrix $A = \{a_1, ..., a_k\}$, where each $a_i \in \mathbb{R}^n$ denotes an anchor service and is a linear combination of a set of term vectors (or columns of $X$):

$$A = \{a_1, ..., a_k\} = XW \quad (1)$$

$$a_i = Xw_i = \sum_{j=1}^{m} W_{ij} x_j, \forall i = 1, ..., k \quad (2)$$

where $W = \{w_1, ..., w_k\} \in \mathbb{R}^{m \times k}$ is a weight matrix. Each entry $W_{ij}$ denotes how much service $s_j$ contributes to anchor service $a_i$. It is worth to note that $W_{ij}$ may take a negative value, meaning that $s_j$ related information is removed from $a_i$. Hence, a negative entry in $W$ serves as the “de-noise” purpose to generate an anchor service with purer functionality or concept.

To illustrate, consider four travel services $s_1, s_2, s_3$, and $s_4$, among which $s_1$ is a flight search service (e.g., www.google.com/flights), $s_2$ allows to search flights and hotels both individually and as a package (e.g., www.hipmunk.com), $s_3$ further adds rental car into the mix (e.g., www.priceline.com), and finally $s_4$ includes both hotel and rental car search (e.g., www.travelhero.com). SFR aims to discover the underlying functionalities of these services, including flight, hotel, and rental car search, and represent them as anchor services $a_1, a_2, a_3$, respectively. Each anchor service is derived through a linear combination of term vectors that represent the actual Web services. Ideally, assume services with similar functionality are represented in the same way by all service providers. In this case, one possible way to derive the anchor services is: $a_1 = s_1$, $a_2 = s_2 - s_1$, and $a_3 = s_3 - s_2$. Hence, the information services can be represented by just using three anchor services (e.g., $s_4 = a_2 + a_3$) instead of involving all the terms that appear in their query interfaces.

In practice, different Web services may use distinct term vocabularies. For example, some flight service may use from and to to denote the departing and destination cities, respectively, while another service may use departure and destination for the same purpose. In addition, the functionalities of actual services may not be clearly defined. For instance, one flight service allows to select the number of passengers with the option to specify passenger categories (e.g., adult, child, senior, and infant) while another service does not offer such functionality. Hence, it is impractical to find a uniform flight anchor service to cover all diverse flight services. In this regard, rather than being regarded as actual services, the anchor services are more appropriately interpreted as latent functional factors that capture the underlying concepts or functionalities of the service space. In the flight services example, one anchor service may correspond to the flight search functional factor and a second anchor service may capture the passenger category specification factor. As the desired anchor services are expected to capture the underlying concepts or functionalities of the service space, we should be able to recover the original services by
Large-scale Service Clustering

using the anchor services. Meanwhile, as most services are designed with specific purposes, it is common for a single service to provide focused and limited functionalities. In other words, a typical service is only related to a small subset of anchor services that cover its functionalities. Therefore, a desired anchor service set should optimize the following objective function:

$$\min_{A, z_i} J_0 = \sum_{i=1}^{m} \|x_i - Az_i\|^2 + \lambda \|z_i\|_0$$ \hspace{1cm} (3)

subject to \(\|a_j\|^2 \leq c, \forall j = 1, ..., k\)

where \(z_i \in \mathbb{R}_+^k\) is the coefficient vector with \(z_i(j)\) signifying the correlation between \(x_i\) and anchor service \(a_j\). \(\|z_i\|_0\) is the \(L_0\) norm of \(z_i\) that counts the number of nonzero elements in \(s_i\). Since each service is expected to correlate with only a small subset of anchor services, \(z_i\) with many nonzero elements will be penalized and \(\lambda\) is the penalty parameter. Therefore, the second term of Eq. (3) corresponds to a sparsity constraint on \(z_i\). The norm constraint on the size of the anchor service, i.e., \(\|a_j\|^2 \leq c\), avoids arbitrarily large anchor services that keep \(Az_i\) unchanged while making \(z_i\) arbitrarily close to zero.

3.2. Relaxation of the Objective Function

It has been proved that finding \(A\) and \(s_i\) that optimize objective function in Eq. (3) is NP-hard \(^{25}\). Therefore, instead of directly solving Eq. (3), we tackle the following optimization problem with a relaxed constraint:

$$\min_{W, Z} J_1 = \|X - XWZ\|^2_F + \lambda \sum_{i=1}^{m} \|z_i\|_1$$ \hspace{1cm} (4)

subject to \(\|Xw_j\|^2 \leq c, \forall j = 1, ..., k\)

where \(\|Y\|_F = \sum_{ij} \sqrt{Y_{ij}}\) stands for Frobenius norm; \(z_i\) is the \(i\)-th column of \(Z \in \mathbb{R}^{k \times m}\); \(\|z_i\|_1 = \sum_{j=1}^{k} |Z_{ji}|\) is the \(L_1\) norm of \(z_i\). We replace \(A\) and \(a_j\) by \(XW\) and \(Xw_j\), respectively, due to Equations (1) and (2).

The first term of \(J_1\) is equivalent to the first term of \(J_0\) reformulated in the matrix form. The key difference between \(J_0\) and \(J_1\) is the change from the \(L_0\) norm of \(z_i\) to the \(L_1\) norm. The relaxed optimization is in essence to minimize a quadratic function with a \(L_1\) norm constraint on \(z_i\) and a \(L_2\) norm constraint on \(a_j\). The optimization problem in the form of Eq.(4) is commonly known as basis pursuit, which has been demonstrated to be effective in finding sparse coefficient vectors (i.e., \(z_i\)'s). Therefore, the solution of \(J_1\) is expected to provide a good approximation to the optimal solution of \(J_0\). Furthermore, the relaxed optimization problem is computational attractive, which can be efficiently tackled by iteratively solving a \(L_1\) regularized least squares problem and \(L_2\) regularized least square problem to obtain \(Z\) and \(W\), respectively \(^{12}\).
3.3. An Illustrating Example

In what follows, we use a simple example to further illustrate the key ideas of SFR as presented above. We randomly choose six services from a real-world service dataset, in which three services are from the travel domain and the other three are from the medical domain. Processing the service descriptions results in 12 distinct terms. Hence, a 12 × 6 service matrix X is constructed. The transpose of X is given in Eq. (5) for the convenience of presentation, in which each row denotes a service and each column denotes a distinct term. Each entry corresponds to a term frequency and each row vector is further normalized to have $L_2$ norm equal to 1.

$$X' = \begin{pmatrix}
0.29 & 0.29 & 0.29 & 0.096 & 0 & 0 \\
0.26 & 0.26 & 0.26 & 0.088 & 0.61 & 0.18 \\
0.33 & 0.33 & 0.33 & 0.33 & 0.11 & 0 \\
0 & 0 & 0 & 0.092 & 0 & 0 \\
0 & 0 & 0 & 0.092 & 0 & 0 \\
0 & 0 & 0 & 0.092 & 0 & 0
\end{pmatrix}$$

(5)

It is clear from Eq. (5) that even for a small service set with just six services, the entries will increase dramatically when the set scales to a large number of services.

We set the number of anchor services as 4 (i.e., $k = 4$) and solve the relaxed optimization problem in Eq. (4), which leads to:

$$W = \begin{pmatrix}
-0.12 & 0.4 & 0.37 & 0.14 \\
-0.13 & 0.39 & 0.35 & 0.15 \\
-0.14 & 0.31 & 0.34 & 0.17 \\
0.55 & -0.17 & 0.26 & 0.098 \\
0.32 & -0.12 & -0.19 & 0.44 \\
0.3 & -0.12 & -0.2 & 0.44
\end{pmatrix}, \quad Z = \begin{pmatrix}
0 & 0 & 0 & 0.24 & 0.22 & 0.21 \\
0.23 & 0.23 & 0.2 & 0 & 0 & 0 \\
0.27 & 0.26 & 0.24 & 0.13 & 0.0087 & 0.0054 \\
0.26 & 0.26 & 0.26 & 0.42 & 0.42 \\
\end{pmatrix}$$

(6)

The $i$-th column of $Z$ (i.e., $z_i$) corresponds to the new representation of the $i$-th service (i.e., $i$-th row of $X'$ in Eq. (5)) in the anchor service space. As expected, $Z$ has a sparse structure, which justifies the effectiveness of $L_1$ norm approximation of the original optimization problem. The first three columns of $Z$ imply that the first three services are only relevant to the last three anchor services. In contrast, the other three services, which correspond to the last three columns of $Z$, are tightly coupled with the first and last anchor services (the third entries of these three columns are close to zero). All entries in $Z$ are non-negative, which allows functionality of each service to be represented as an additive combination of functionalities encoded by a small number of relevant anchor services.

Some interesting observations are also revealed from the weight matrix $W$. These observations demonstrate that the interplay between the weight matrix $W$ and the coefficient matrix $Z$ helps achieve the effectiveness of SFR. For example, the first three entries of $w_1$ (i.e., the first column of $W$) take negative values. These negative entries imply that information related to the first three services are removed from the first anchor service $a_1$. Therefore, a negative entry $w_{ij}$ has the effect of “decoupling” the $i$-th service from $j$-th anchor service. In contrast, a positive entry
\( W_{ij} \) signifies the “addition” of the \( i \)-th service’s functionality to the \( j \)-th anchor service. The decoupling and addition mechanism helps discover cohesive concepts or service functionalities that are captured by the anchor services. It also leads to unambiguous service-to-anchor service relationships. For example, \( z_1^T \), the first row of the coefficient matrix \( Z \), consists of three zero and three nonzero entries. This implies that the first three services are completely irrelevant to anchor service \( a_1 \) whereas the last three are tightly coupled with \( a_1 \). In fact, the three zero entries are resulted from the first three entries in \( w_1 \), which decouple the first three services from \( a_1 \). The three nonzero entries are due to the last three entries of \( w_1 \) that add the functionalities of the last three service into \( a_1 \). Similarly, the second row of \( Z \) shows that the first three services are relevant to anchor service \( a_2 \) while the last three service are irrelevant to it.

### 3.4. Clustering in Anchor Service Space

By optimizing objection function \( J_0 \) or its relaxed version \( J_1 \), we aim to find an anchor service set \( A \) and a new sparse representation \( z_i \) to best approximate \( x_i \):

\[
x_i \approx Az_i = \sum_{j=1}^{k} a_j Z_{ji}
\]

Therefore, the new representation \( z_i \) can be regarded as the projection of \( x_i \) onto the anchor service space \( A = \{a_1, ..., a_k\} \). The coefficient vector \( z_i \) captures the relevance between the \( i \)-th service and all the \( k \) anchor services. The sparsity constraint on \( z_i \) leads to clear-cut relationships between a service and anchor services. Hence, services can be easily separated based on their distinct relationships with the anchor services using sparse functional representation.

![Anchor Service Space](image1)

Figure 1 provides a schematic view of service clustering in the anchor service space. Since each service is only related to a small subset of anchor services, the similarity between two services can be easily computed based on how they are related to the anchor services. More specifically, two services are similar if they are related to a similar set of anchor services. Therefore, the anchor services serve
as a bridge to relate different services. Since the anchor services capture the high-level concepts of the services, projection onto the anchor service space provides a better way to assess the similarity between services than using terms. The sparsity constraint on the coefficient vectors provides a clear separation between services, which significantly facilitates service clustering. Any simple clustering algorithms, such as K-means, may be directly applied to the coefficient matrix $Z$ to generate service clusters. Meanwhile, the anchor service space has a much low dimensionality than the term vector space (i.e., $k \ll n$), clustering in the anchor service space can easily scale to a massive number of services.

3.5. Term Vector and Anchor Service Integration

Sparse functional representation is formed by projecting term vectors $x_i$’s onto an anchor service space. Service clustering is then performed on the projected representation, which is independent on the original term vector space. In this section, we present a new clustering scheme that integrates information from both the anchor service space and the term vector space.

One piece of information in the term vector space that can be leveraged is the term vectors that share a decent number of distinct terms. If two term vectors have a reasonable number of common terms, it means that they share similar high-level concept and hence should be clustered together. This useful information can be incorporated into anchor space clustering in a semi-supervisory manner to improve the overall clustering quality. More specifically, we construct a neighborhood graph $G$, in which each vertex corresponds to a term vector $x_i \in X$. Two vertices $x_i$ and $x_j$ are connected in $G$ if the similarity between $x_i$ and $x_j$ is no less than a threshold value. The similarity can be simply computed by using cosine similarity. Assume that $B$ is the incidence matrix of $G$. Therefore, $B_{ij} = 1$ if $x_i$ and $x_j$ are connected in $G$ (i.e., similar to each other) and 0 otherwise. We expect $B$ to be sparse as it is less likely for most term vectors to share many common terms.

Consider any pair of term vectors $x_i$ and $x_j$ and their corresponding sparse functional representations $z_i$ and $z_j$. If $x_i$ and $x_j$ are similar, we expect $z_i$ and $z_j$ to be similar as well. In contrast, if $z_i$ and $z_j$ significantly deviate from each other, $||z_i - z_j||^2 B_{ij}$ will be large as $B_{ij} = 1$ when $x_i$ and $x_j$ are close in the term vector space. Therefore, we can use $||z_i - z_j||^2 B_{ij}$ as a penalty term and incorporate it into objection function $J_1$, which leads to

$$\min_{W,Z} J_2 = ||X - XWZ||^2_F + \lambda \sum_{i=1}^{m} ||z_i||_1 + \gamma \sum_{i=1}^{m} \sum_{j=1}^{m} ||z_i - z_j||^2 B_{ij}$$

subject to $||Xw_j||^2 \leq c, \forall j = 1, \ldots, k$

where $\gamma$ is the penalty parameter. It is worth to note that if $x_i$ and $x_j$ are not evaluated to be similar, we have $B_{ij} = 0$. In this case, the term $||z_i - z_j||^2 B_{ij}$ is set to 0 and will not affect the objective function $J_2$. 
4. Optimization Method

Computing the anchor service set $A$ and the sparse functional representation $Z$ requires to optimize (8). Since the weight matrix $W$ can be easily computed via (1) after $A$ is obtained, we replace $XW$ and $Xw_j$ in (8) by $A$ and $a_j$, respectively, to keep the notation uncluttered:

$$
\min_{W,Z} \|X - AZ\|^2_F + \lambda \sum_{i=1}^m \|z_i\|_1 + \gamma \sum_{i=1}^m \sum_{j=1}^m \|z_i - z_j\|^2 B_{ij} \tag{9}
$$

subject to $\|a_j\|^2 \leq c, \forall j = 1, ..., k$.

The coupling between $A$ and $Z$ through product $AZ$ makes (9) not convex in both variables simultaneously. However, the function is convex in $A$ and $Z$ individually while keeping the other variable fixed. This motivates us to employ an iterative process to solve (9), which alternatively optimizes with respect to $A$ and $Z$ while holding the other fixed.

4.1. Optimizing Sparse Functional Representation

In this section, we focus on optimizing (9) by assuming the anchor service set $A$ constant. Hence, (9) can be reformulated as

$$
\min_{Z} \|X - AZ\|^2_F + \lambda \sum_{i=1}^m \|z_i\|_1 + \gamma \sum_{i=1}^m \sum_{j=1}^m \|z_i - z_j\|^2 B_{ij} \tag{10}
$$

The coupling among $z_i$’s (i.e., $z_i'z_j$) caused by the last term in (10) makes the above function non-convex on all $z_i$’s simultaneously. Hence, we again employ an iterative process to optimize each $z_i$ while holding others fixed. Specifically, by fixing $\{z_j\}_{j \neq i}$, we optimize $z_i$ by solving the follow problem:

$$
\min_{z_i} \|x_i - Az_i\|^2_F + \lambda \|z_i\|_1 + \gamma u_i z_i'z_i + \gamma z_i'v_i \tag{11}
$$

where $u_i = 2 \sum_{j \neq i} B_{ij}$ and $v_i = 4 \sum_{j \neq i} B_{ij}z_j$. Since all the $z_i$’s will be optimized in turn, we omit the subscript to further simply the notation. Also, let $f(z) = \|x - Az\|^2_F + \gamma uz'z + \gamma z'v$ and $g(z) = \lambda \|z\|_1$, which reduces (11) to

$$
\min_{z} F(z) = f(z) + g(z) \tag{12}
$$

Since $g(z)$ in (12) is not differentiable, a commonly used approach for solving (12) is the subgradient method, which requires $N$ iterations to achieve an error rate at $O(\frac{1}{\sqrt{N}})$.

As a simple example, consider function $f(x, y) = xy$, which is non-convex on both $x$ and $y$ simultaneously.

In most optimization problems, the higher order derivatives are either not available or incur high cost to compute and store.
the convergence rate of the subgradient method is already optimal for non-smooth optimization 17.

To achieve more efficient optimization, we explore the special structure of the $L_1$ regularization term $g(z)$ and present two different algorithms. We focus on describing the key intuition of these algorithms and leave their low-level details to the Appendix of the paper. More specifically, the first algorithm leverages the sparsity structure of $z$ to efficiently compute the non-zero entries in $z$, referred to as Sparse Structure Algorithm or SSA. Since the large number of zero entries are not involved in the computation, the update of $z$ can be done very efficiently. Despite having a cheap update cost in each iteration, which guarantees to reduce the objective, the convergence rate of SSA can not be formally proved. The worst case analysis shows that the algorithm may take iterations exponential to the size of $z$ to converge even though it works much more efficiently than the worse case in practice. To overcome this limitation, the second algorithm leverages the separability of the $L_1$ term and exploits an Accelerated Gradient Algorithm, referred to as AGA, to optimize $z$. In AGA, the update of $z$ only requires to evaluate the gradient of $f(z)$, hence can be done efficiently. More importantly, AGA converges as $O(\frac{1}{N^2})$, which is optimal for smooth optimization problems 18. In this regard, leveraging the special structure of the $L_1$ regularization term has the effect of canceling out its non-differentiability, which guarantees the optimal convergence rate of AGA.

4.2. Optimizing the Anchor Service Set

Fixing the coefficient matrix $Z$ transforms (9) into a least square problem with quadratic constraints. By ignoring the constant terms, we format (9) into

$$
\min_A \|X - AZ\|^2_F \\
\text{subject to } \|a_j\|^2 \leq c, \forall j = 1, ..., k
$$

(13)

The Lagrangian of (13) is given by

$$
\mathcal{L}(A, \phi) = \|X - AZ\|^2_F + \phi'\left(\|a_1\|^2 - c, ..., \|a_k\|^2 - c\right) \\
= \|X - AZ\|^2_F + \langle A' A - cI, \text{diag}(\phi) \rangle
$$

(14)

where $\phi \in \mathbb{R}^k$ is the Lagrangian variable and $\langle A, B \rangle = \text{trace}(A'B)$ is the inner product of two matrices. Let $\Phi = \text{diag}(\phi)$, we have

$$
\mathcal{L}(A, \Phi) = \|X - AZ\|^2_F + \langle A' A - cI, \Phi \rangle
$$

(15)

From (15), we derive the dual function as

$$
f(\Phi) = \inf_A \mathcal{L}(A, \Phi)
$$

(16)
Assume that $\Phi^*$ is the optimal dual variable obtained by maximizing the dual function $f(\Phi)$. Since strong duality holds for $L(A, \Phi)^4$, we have

$$L(A^*, \Phi^*) = \inf_A L(A, \Phi^*)$$

where $A^*$ is the optimal primal variable, which is the anchor service set we aim to compute. Therefore, we can first optimize the dual function $f(\Phi)$ in (16) to get the optimal dual variable $\Phi^*$ and then compute $A^*$ based on (17).

The dual function $f(\Phi)$ is a continuously differentiable concave function $^4$, We apply the standard gradient step to approach its optimum:

$$\Phi_k = \Phi_{k-1} + t_k \nabla f(\Phi_{k-1})$$

where $t_k$ is the step size. $\nabla f(\Phi_{k-1})$ is the gradient of $f(\Phi)$ evaluated at $\Phi_{k-1}$, which is given by

$$\nabla f(\Phi_{k-1}) = \frac{\partial L(A_k, \Phi)}{\partial \Phi} = A_k' A_k - c I$$

where $A_k$ is the minimizer of the Lagrangian for $\Phi_{k-1}$:

$$A_k = XZ' Q^{-1}$$

where $Q = ZZ' + \Phi_{k-1} \in \mathbb{R}^{k \times k}$ is a small matrix, whose inverse is efficient to compute. Substituting (20) into (19), we have

$$\nabla f(\Phi_{k-1}) = (Q^{-1})' ZX' XZ' Q^{-1}$$

$ZX' XZ'$ can be computed once during initialization and then used as constant in each iteration. Hence, the gradient step in (38) can be performed efficiently.

5. Experimental Study

We apply the proposed SFR and the optimization method for anchor service discovery to two real-world service datasets. We evaluate the effectiveness SFR for service clustering and the efficiency of the optimization method for anchor service discovery. To evaluate the effectiveness of SFR in service clustering, we compare our approach with a set of competitive service clustering algorithms. As both clustering quality and efficiency are important evaluation metrics, we report both clustering accuracy and CPU times for clustering in our experimental results. We report the CPU times of the proposed optimization algorithms, SSA and AGA, for anchor service discovery and compare the results.

5.1. Service Dataset Description

We include two real-world service datasets: one middle scale dataset with 452 services $^{11}$ and one large-scale dataset with 3,738 services $^{24}$. We describe the properties of each dataset in what follows:
• **Dataset 1**: The first service dataset consists of 452 WSDL descriptions of services from 7 different application domains. More specifically, the services are distributed as follows: communication (42), education (139), economy (83), food (23), medical (45), travel (90), and weapon (30). The domain information provides labels of the service clusters, which will be used to evaluate the accuracy of the clustering algorithms in our experiments.

• **Dataset 2**: The second service dataset consists of WSDL descriptions of 3,738 services located in more than 20 countries. The services are more diverse and complicated, coming from a large number of domains varying from government to academia and industry. Unlike **Dataset 1**, no cluster labels are available in this dataset.

5.2. **Metrics for Clustering Quality**

For **Dataset 1**, the service domains will serve as the ground truth to evaluate the clustering quality. More specifically, we adopt two metrics to measure the service clustering quality: **Accuracy** (i.e., \(AC\)) and **Mutual Information** (i.e., \(MI\)). Both \(AC\) and \(MI\) are widely used metrics to assess the performance of clustering algorithms. For **Dataset 2**, since no true service cluster labels are available, we cannot use the above two metrics to evaluate clustering quality. Instead, we choose to use the **Silhouette Value** (i.e., \(SV\)), which is a commonly used metric for clustering quality evaluation when no ground truth is available.

• **Accuracy**: For a given service \(s_i\), assume that its cluster label is \(c_i\) and its domain label is \(d_i\) based on the domain information. The \(AC\) metric is defined as follows:

\[
AC = \frac{\sum_{i=1}^{m} \delta(d_i, map(c_i))}{m}
\]

where \(m\) is the total number of Web services in the service dataset. \(\delta(x, y)\) is the delta function that equals to one if \(x = y\) and equals to zero if otherwise. \(map(c_i)\) is the permutation mapping function that maps each assigned cluster label to the equivalent domain label. The best mapping between the two sets of labels is achieved by the Kuhn-Munkres algorithm.

• **Mutual Information**: Let \(D\) be the set of application domains obtained from the service dataset and \(C\) be the service clusters obtained a service clustering algorithm. The mutual information metric \(MI(D, C)\) is defined as follows:

\[
MI(D, C) = \sum_{d_i \in D, c_j \in C} p(d_i, c_j) \log_2 \frac{p(d_i, c_j)}{p(d_i)p(c_j)}
\]

where \(p(d_i)\) and \(p(c_j)\) are the probabilities that a randomly selected service from the service set belongs to domain \(d_i\) and cluster \(c_j\), respectively. \(p(d_i, c_j)\) is the joint probability that the randomly selected service belongs to both domain \(d_i\) and cluster \(c_j\).
Table 2. Clustering Results on Dataset_1

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Cluster Quality</th>
<th>Cluster Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AC (%)</td>
<td>MI(%)</td>
</tr>
<tr>
<td>NMTF</td>
<td>51.1</td>
<td>48.0</td>
</tr>
<tr>
<td>NMTFS</td>
<td>56.6</td>
<td>46.5</td>
</tr>
<tr>
<td>K-meanS</td>
<td>42.7</td>
<td>21.2</td>
</tr>
<tr>
<td>SVDC</td>
<td>45.3</td>
<td>36.3</td>
</tr>
<tr>
<td>SFR</td>
<td>60.4</td>
<td>56.6</td>
</tr>
</tbody>
</table>

- **Silhouette Value:** The silhouette value for service \( s_i \) measures how similar that \( s_i \) is to the services in its own cluster compared to services in other clusters, and ranges from -1 to +1. More specifically, \( SV \) is defined as the average over the silhouette values of all services:

\[
SV = \frac{\sum_{i=1}^{m} SV_i}{m}
\]

\[
SV_i = \frac{(b_i - a_i)}{\max(a_i, b_i)}
\]

where \( SV_i \) is the silhouette value for the \( i \)-th service \( s_i \); \( a_i \) is the average distance from \( s_i \) to the other services in the same cluster as \( s_i \), and \( b_i \) is the minimum average distance from \( s_i \) to services in a different cluster, minimized over clusters. Therefore, \( SV \) essentially measures the “cohesiveness” of the clusters.

5.3. **Clustering on Dataset_1**

Before running any service clustering algorithms, we need to preprocess the service descriptions in Dataset_1. We apply a standard text processing procedure that includes tokenization, stopword removal, and stemming to extract distinct terms from the service descriptions. As a result, 803 distinct terms are extracted. Thus, a 803 \( \times \) 452 service matrix \( X \) is constructed. We compare the proposed SFR based clustering with the following service clustering algorithms:

- **NMTF:** Non-negative matrix tri-factorization based approach to simultaneously cluster services and operations offered by the services.
- **NMTFS:** Extending service descriptions by including semantically similar terms to address the sparsity issue and then applying NMTF to the extended service descriptions.
- **K-meanS:** Applying K-means clustering to the semantically extended service descriptions.
- **SVDC:** Applying Singular Value Decomposition (SVD) to co-cluster services and operations they offers.
Since the services are from 7 domains, we set the number of service clusters as 7. We set the number of anchor services as 30, i.e., $k = 30$. The two penalty parameters $\lambda$ and $\gamma$ in objection function $J_2$ are set to 1 and 0.1, respectively. These will be used as default parameter values in our experiments unless specified otherwise. It is worth to note that a wide range of values work reasonably well for these parameters. We will investigate the impact of different parameters in Section 5.6.

Table 2 reports both clustering quality and efficiency in CPU times from all the algorithms under comparison. The clustering quality is evaluated using all the three evaluation metrics described in Section 5.2. SFR clearly outperforms all other competitors in terms of clustering quality. It achieves 60.4% in clustering accuracy, which is 7% better than the second highest accuracy achieved by NMTFS. In terms of mutual information, it is 17.9% better than second best, NMTF. The results on silhouette value are pretty much consistent with those on accuracy and mutual information. SFR achieves a silhouette value at 0.7, which is much higher than all other algorithms. This demonstrates that sparse functional representation provides good separation between similar services and dissimilar ones, which makes service clustering much easier. A higher silhouette value signifies that the generated clusters are more cohesive. The default similarity/distance measure that is used by SFR to cluster the sparse functional representations (i.e., the coefficient matrix $Z$) is the Euclidean distance. We also test the cosine similarity in SFR and the clustering accuracy, mutual information, and silhouette value are 64.4%, 59.3%, and 0.63, respectively. Again, the result is the best among all algorithms under comparison.

In terms of clustering efficiency, the CPU time used by SFR is far less than other clustering algorithms. This confirms that clustering can be efficiently performed in the compact anchor service space. SVDC also achieves a fast response time. This is because it computes a service-operation correlation matrix in order to perform co-clustering. Since the number of operations is much less than the number of terms, SVDC actually works on a much smaller matrix, which justifies its fast performance. Nonetheless, the poor clustering quality of SVDC implies that the service-operation correlation matrix does not provides a good representation for service clustering.

5.4. Clustering on Dataset_2

We adopt the same standard text processing procedure to process the 3,738 service descriptions in Dataset_2, which results in 16,884 distinct terms. Therefore, a $16,884 \times 3,738$ service matrix $X$ is constructed. Each term vector has a dimensionality of 16,884. Before applying any clustering algorithms on such high dimensional data, a common practice is to first reduce the dimensionality. Thus, we employ Principle Component Analysis (PCA) to reduce the dimensionality to 64. It is also worth to note that algorithms, such as NMTFS and K-means, require to perform semantic extensions on each distinct term. This will lead to a huge term dictionary for a large service set, like Dataset_2. The resultant service matrix will be several orders larger than $X$. To avoid prohibitive computational cost, we are not including
NMTF and K-means for comparison. Instead, we add another two algorithms into the mix:

- **K-means**: Directly applying K-means clustering to the terms vectors in X.
- **PK-means**: Applying K-means after PCA dimensionality reduction.

Since there are no cluster labels for Dataset 2, we only use silhouette value to evaluate clustering quality. We set the number of clusters to 30. The number of anchor services is set to 128 and all other parameters take their default values for SFR.

Table 3 reports the clustering result on Dataset 2. In terms of clustering quality, SFR achieves the highest silhouette value among all the algorithms. This is consistent with the results from Dataset 1. It is also worth to note that SVDC fails to converge after spending over 1,000 seconds, so no silhouette value is computed. For the clustering efficiency, SFR achieves the best clustering time, which is again consistent with the results from Dataset 1. The fast clustering performance of SFR further justifies that sparse functional representation indeed makes clustering easier. Once the anchor space is discovered, it can be stored and reused. Therefore, for large-scale service clustering, anchor services can be first discovered offline and then service clustering can be performed in realtime to meet different user requirements on number of clusters, distance metrics, clustering algorithms, and so on.

### 5.5. Efficiency of the Optimization Method

As discussed in Section 4, we employ an optimization process that alternatively optimizes with respect to A and Z while holding the other fixed. We propose two algorithms, SSA and AGA, to compute the coefficient matrix Z and another algorithm to compute A, which we refer to as anchor service optimization, ASO. This leads to two optimization algorithms for anchor service discovery: SSA+ASO and AGA+ASO. We report the efficiency of these two optimization algorithms in this section.
Besides running the two algorithms on both service datasets, we also select 1000, 2000, and 3000 services from Dataset 2 and apply the algorithms to these subsets of services. This helps investigate in detail how the performance of the algorithms varies with the scale of the service set. As can be seen from Table 4, AGA+ASO outperforms SSA+ASO in all cases, which clearly justifies the effectiveness of the accelerated gradient algorithm. When the number of services is relatively small, the two algorithms have comparable performance but SSA+ASO scales better with the size of the service set. This is because SSA uses less iterations to converge for small datasets while taking much longer iterations for large ones as it does not have a provable convergence behavior. AGA, on the other hand, has an optimal convergence rate at $O\left(\frac{1}{k^2}\right)$, which ensures its good performance.

### 5.6. Impact of Parameters

We investigate the impact of different parameters in this section, including the number of anchor services (i.e., $k$), and the two penalty parameters (i.e., $\lambda$ and $\gamma$). We vary one of these three parameters while keeping the other two fixed at their default values. Figure 2 shows how different parameters affect the clustering quality in Dataset 1.

Accuracy and mutual information always vary in a similar way with the changes of parameters. Both accuracy and mutual information reach their respective highest values when $k = 40$, $\lambda = 0.1$, and $\gamma = 1$, respectively. The silhouette value, on the other hand, varies differently with accuracy and mutual information. First, the SV value decreases as $k$ increases. Recall that in SFR, services are clustered based on their relationships with the anchor services. The sparsity constraint forces services to be only related to a small subset of the anchor services. Therefore, when $k$ is small, the sparse representation of a service will “concentrate” on a small number of anchor services. This will lead to very compact and cohesive clusters. Therefore, $SV$ will decrease as $k$ increases. Similar explanation is applied to the impact of $\lambda$, which enforces the sparsity constraint. Increasing $\lambda$ will make $z_i$’s more sparse, which has the effect of moving services closer to the relevant anchor services and further away from less relevant ones. This will also produce more cohesive clusters. Therefore, $SV$ increases as $\lambda$ increases. Instead of monotonically decreasing or increasing as with the increase of $k$ and $\lambda$, the $SV$ value reaches it peak value when $\gamma$ is 100 and then decreases when $\gamma$ increases further. In contrast, accuracy and mutual

---

**Table 4. Efficiency for Anchor Service Discovery**

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Number of Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>452</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>3738</td>
</tr>
<tr>
<td>SSA+ASO</td>
<td>22.43</td>
</tr>
<tr>
<td>AGA+ASO</td>
<td>28.4</td>
</tr>
<tr>
<td>AGA+ASO</td>
<td>42.7</td>
</tr>
<tr>
<td>AGA+ASO</td>
<td>57.2</td>
</tr>
<tr>
<td>AGA+ASO</td>
<td>68.3</td>
</tr>
</tbody>
</table>
information reach their peak values when $\gamma = 1$. The discrepancy may be due to that the domain definition of the service set is not in line with the cohesiveness of the service clusters. For example, some services may be cross-domain in nature but assigned to a domain that is inconsistent with the clustering result. The results on Dataset_2 show very similar patterns with regard to the parameters discussed above as those of Dataset_1 (in term of SV values because only SV values are reported for Dataset_2). Therefore, we skip the presentation of the results to avoid repetition.

Since there is no ground truth about the number of clusters for Dataset_2, we evaluate the algorithms over different clusters. As SVDC fails to converge over Dataset_2 and NMTF reports very poor results, we only compare three algorithms: PK-means, K-means, and SFR. Table 5 summarizes the results over different number of clustering, varying from 10 to 50. We have to observations. First, SFR reports the best clustering quality (in terms of silhouette values) over different clustering numbers. This further confirms the effectiveness of SFR. Second, as the number of clusters increases, all clustering algorithms report better silhouette value. This is in line with the definition of the silhouette value, which essentially measures the cohesiveness of the clusters. As the number of clusters increases, the resultant clusters become smaller and hence more cohesive.
6. Conclusion

We present Sparse Functional Representation (SFR), a novel service representation scheme, which is economical to store, efficient to process, and intuitive to interpret. SFR projects long and sparse term vectors onto an anchor service space, which consists of a small number of anchor services. The similarity between services is encoded by their proximity to the anchor services. The sparsity constraints enforce that each service is only related to a small subset of anchor services. This has the effect of moving services closer to the relevant anchor services and further away from irrelevant ones. These key features significantly facilitate large-scale service clustering. We develop a set of optimization algorithms to efficiently construct the anchor service space and compute the SFR in the space. The optimization process alternatively optimizes the anchor service set and the SFR while holding the other fixed. The optimization performance is further improved through an accelerated gradient scheme, which achieves an optimal convergence rate. Comprehensive experiments on two real-world service datasets clearly demonstrate the effectiveness of SFR in large-scale service clustering and the efficiency of the optimization algorithms for anchor space construction. As new services are continuously developed and deployed on the Web, one interesting future direction is to develop an online algorithm for anchor service discovery. The online algorithm will efficiently update the anchor service sets and the coefficient matrix whenever new services are added instead of performing the optimization process from scratch.

References

7. Appendix

We present the details of the two algorithms for optimizing sparse functional representation, which are introduced in Section 4.1. We also describe a further enhancement of the optimization algorithm for anchor service set computation presented in Section 4.2.

7.1. Sparse Structure Algorithm

Since $F(z)$ is a convex function of $z$, $z^*$ is the optimal solution of (12) if and only if $0 \in \partial F(z^*)$, where $0 \in \mathbb{R}^{k \times 1}$ is a vector of all zeros. $\partial F(z^*)$ is the subdifferential of $F(\cdot)$ at $z^*$, which is the set of subgradients of $F(\cdot)$ at $z^*$. $f(\cdot)$ is a smooth function, so the subdifferential of $f(\cdot)$ has only one subgradient, which coincides with $\nabla f(\cdot)$.
where \( \partial \) implies that there exists \( h \in \partial g(z^*) \), such that \( 0 = \nabla f(z^*) + h \). Recall that \( g(z) = \lambda \|z\|_1 = \lambda \sum_{i=1}^{k} |z(i)| \). The subdifferential of \( g(z) \) with respect to \( z(i) \) takes different values according to \( z(i) \):

\[
\partial_i g(z) = \begin{cases} 
-\lambda & \text{if } z(i) < 0 \\
\lambda & \text{if } z(i) > 0 \\
[-\lambda, \lambda] & \text{if } z(i) = 0 
\end{cases}
\]

(27)

where \( \partial_i g(z) \) is the \( i \)-th component of \( \partial g(z) \). Hence, the value of the \( i \)-th component of \( h^* \) lies in \([-\lambda, \lambda]\), i.e., \( h^*(i) \in [-\lambda, \lambda] \). This implies that \( z^* \) is the optimal solution of (12) if and only if

\[
|\nabla_i f(z^*)| \leq \lambda \quad i = 1, \ldots, k
\]

(28)

where \( \nabla_i f(z^*) \) is the \( i \)-th component of \( \nabla f(z^*) \). To approach \( z^* \), the SSA algorithm adopts a two-step process: identification and adjustment, where the former finds the components of \( z \) that violate the optimal condition (28) and the latter optimizes these components to satisfy the condition. To ensure a sparse structure, \( z \) is initialized as a zero vector. We follow a similar heuristic as in the feature sign algorithm for sparse coding computation \(^{12,25}\). In particular, SSA proceeds by iteratively performing the identification and adjustment steps.

**Identification:** The identification step first computes the gradient of \( f(\cdot) \) based on the current \( z \) value, which leads to

\[
\nabla f(z) = 2(A' A + \gamma u)z - 2A' x + \gamma v
\]

\[
= Pz - q = \sum_{i=1}^{k} p_i z(i) - q
\]

(29)

where \( P = 2(A' A + \gamma u) \) and \( q = 2A' x - \gamma v \). These two values need to be computed once and then stored as constants for later computation of \( \nabla f(z) \) based on the updated \( z \) values. From all the zero components of the current \( z \), SSA then identifies the \( i \)-th component, where \( i = \arg \max_i |\nabla_i f(z)| \). If \( |\nabla_i f(z)| > \lambda \), the \( i \)-th component will be included into an active set \( A \). Based on the sign of \( \nabla_i f(z) \), we further consider two cases:

1. If \( \nabla_i f(z) > \lambda \), for any \( h(i) \in \partial_i g(z) \), we have \( \nabla_i f(z) + h(i) > 0 \) due to (27), which violates the optimal condition. Also, the positive subgradient value implies that the current value (i.e., 0) of \( z(i) \) is larger than its optimal value. To improve the objective, \( z(i) \) should be decreased, which leads to a negative value. Let the vector \( \theta \) denote the signs all the components of \( z \), so \( \theta(i) = -1 \).

2. If \( \nabla_i f(z) < \lambda \), we have \( \nabla_i f(z) + h(i) < 0 \), which also violates the optimal condition. To improve the objective, \( z(i) \) should take a positive sign due to the negative subgradient. Hence, \( \theta(i) = 1 \).
The identification step in essence chooses a zero component in \( z \) that violates the optimal condition to the greatest extent. The selected component is added to an active set \( A \), whose value will be updated accordingly in the adjustment step.

**Adjustment:** The adjustment step takes as input the active set \( A \), which corresponds to components in \( z \) expected to take non-zero values. These components include the one selected by the identification step in the same iteration along with the ones kept from the previous iterations. Since the signs of these components are available (through identification) and assuming that components outside \( A \) stay as zeros, (12) is transformed into

\[
\min_{\hat{z}} F(\hat{z}) = f(\hat{z}) + g(\hat{z}) = \|\hat{x} - \hat{A}\hat{z}\|_F^2 + \gamma u\hat{z}'\hat{z} + \gamma \hat{z}'\hat{v} + \lambda \hat{z}'\hat{\theta} \tag{30}
\]

where \( \hat{z}, \hat{v}, \) and \( \hat{\theta} \) are the subvectors of \( z, v, \) and \( \theta \) that correspond to the nonzero components in \( A \). \( \hat{A} \in \mathbb{R}^{n \times |A|} \) consists of only columns that correspond to the components in \( \hat{z} \). (30) is a constrained quadratic programming (QP) problem, where the constraint requires that components of \( \hat{z} \) to take signs consistent with \( \hat{\theta} \). The sign constraint is in essence a bound constraint that forces each \( \hat{z}(i) \) to reside in a certain quadrant (i.e., based on the sign \( \hat{\theta}(i) \)).

A QP problem with a simple constraint as in (30) can be conveniently solved by the projected gradient method (PGM) \(^6\), which iteratively updates the solution through gradient descent and applies Euclidean projection to map invalid solution to the valid set. The sparse structure of \( z \) implies that \( |A| \ll |z| \), which makes \( \hat{A}'\hat{A} \) a small matrix. This allows us to apply a procedure that is similar to PGM but uses only one step to complete. Specifically, we compute the analytical solution \( \hat{z}^{update} \) of (30) by ignoring the sign constraint of \( \hat{z} \). This can be done efficiently as the major cost is just to inverse a small matrix \( \hat{A}'\hat{A} + \gamma uI \). We then perform a projection to map each invalid \( \hat{z}(i) \) to the quadrant that is consistent with the sign \( \hat{\theta}(i) \). This can be achieved by setting the invalid \( \hat{z}(i) \) to zero. The zero components will also be removed from the active set \( A \), accordingly.

The interplay of identification and adjustment ensures the reduction of the objective in each iteration. Specifically, identification selects the zero component in \( z \) that violates the optimal condition the most. It also determines the direction to change the component so that the objective will be reduced. Adjustment then updates the selected components accordingly by performing a constrained optimization, which ensures that the components are updated in the directions determined by the identification step. Since \( F(z) \) in (12) is bounded below, SSA guarantees to converge. The downside of SSA is that its convergence rate cannot be formally proved. It relies on a greedy heuristic that chooses the component violating the optimal condition the most for update and leverages the sparse structure of \( z \) to achieve efficient update. Since the objective will be reduced in each iteration, it is guaranteed that the same active set will only be evaluated at most once. However,
SSA may check all possible active sets in the worse case, which may take number of iterations exponential to the size of \( z \) to converge to the global optimum. Despite this, SSA works reasonably well in practice, which justifies the effectiveness of the greedy heuristic and the efficient update strategy.

### 7.2. Accelerated Gradient Algorithm

Let’s first consider minimizing \( f(z) \) in (12). Since \( f(z) \) is a continuously differentiable function, a simple and widely used method for solving the minimization problem is the gradient algorithm, which generates a sequence of approximation solutions using a gradient step:

\[
z_k = z_{k-1} - t_k \nabla f(z_{k-1})
\]

where \( z_k \) is the \( z \) value and \( t_k \) is the step size at iteration \( k \). It has been well-known fact that the above gradient step can be regarded as a proximal regularization of the linearized function \( f(z) \) at \( z_{k-1} \):

\[
z_k = \arg\min_z \left\{ f(z_{k-1}) + \langle z - z_{k-1}, \nabla f(z_{k-1}) \rangle + \frac{1}{2t_k} ||z - z_{k-1}||^2 \right\}
\]

where \( \langle a, b \rangle = a'b \) is the inner product of two vectors \( a \) and \( b \). The first two terms on the right side of (32) form the linear approximation of \( f(z) \) at \( z_{k-1} \) whereas the last term corresponds to a quadratic proximal regularization.

By adding \( g(z) \) to the right hand side of (32), we can form an optimization sequence \( \{z_k\} \) to minimize the nonsmooth function \( F(z) \) in (12) in a similar spirit of applying the gradient method to a smooth minimization problem \(^1\). In particular, the sequence \( \{z_k\} \) is generated via

\[
z_k = p_{t_k}(z_{k-1})
\]

\[
= \arg\min_z \left\{ f(z_{k-1}) + \langle z - z_{k-1}, \nabla f(z_{k-1}) \rangle + \frac{1}{2t_k} ||z - z_{k-1}||^2 + g(z) \right\}
\]

It is known that the gradient algorithm converges as \( O(\frac{1}{k}) \) for properly chosen step size \( t_k \) \(^1\). Hence, the optimization sequence \( \{z_k\} \) is expected to achieve a similar convergence rate for solving the nonsmooth problem in (12).

Another key benefit of the gradient algorithm is that each gradient step (as shown by (31)) only needs to evaluate the gradient of a smooth function and hence can be efficiently performed. Therefore, to achieve good performance of optimizing (12), it is necessary to show that each \( z_k \) in the optimization sequence can be efficiently evaluated. We reformulate (33) into the following equivalent form by only including terms relevant to \( z \):

\[
z_k = \arg\min_z \left\{ \frac{1}{2t_k} ||z - (z_{k-1} - t_k \nabla f(z_{k-1}))||^2 + g(z) \right\}
\]

\[
= \arg\min_z \left\{ \sum_{i=1}^{k} \left( \frac{1}{2t_k} (z(i) - r(i))^2 + \lambda|z(i)| \right) \right\}
\]

\[(34)\]
where \( \mathbf{r} = \mathbf{z}_{k-1} - t_k \nabla f(\mathbf{z}_{k-1}) \). Due to the separability of the \( L_1 \) norm, the components of \( \mathbf{z} \) are completely decoupled as shown in (34). Therefore, we can optimize each component of \( \mathbf{z} \) separately, which is achieved through a simple analytical form given by:

\[
\mathbf{z}_k(i) = \text{sign}(\mathbf{r}(i)) \max(|\mathbf{r}(i)|, 0)
\]  

(35) shows that \( \mathbf{z}_k \) is easily computed from vector \( \mathbf{r} \), which can be obtained by evaluating the gradient of \( f(\mathbf{z}) \). The cost of computing \( \mathbf{z}_k \) is essentially the same as the gradient step in the gradient algorithm. Therefore, applying the optimization sequence \( \{\mathbf{z}_k\} \) computed from (33) to minimize \( F(\mathbf{z}) \) guarantees to have the same provable convergence behavior as applying the gradient algorithm to a smooth problem.

Despite having an efficient gradient step and a provable convergence rate, it turns out that the gradient algorithm does not offer the optimal performance. The gap between the convergence of the gradient method and the lower complexity bounds for solving smooth problems motivated the development of an accelerated gradient algorithm, known as the Nesterov method. The Nesterov method converges as \( O\left(\frac{1}{k^2}\right) \), which is optimal for smooth optimization given that only the function value and the first-order derivative are used. The acceleration is achieved through the inclusion of a judiciously chosen search sequence \( \{\mathbf{y}_k\} \), where each search point \( \mathbf{y}_k \) is obtained as a linear combination of the latest two approximate solutions \( \mathbf{z}_{k-1} \) and \( \mathbf{z}_{k-2} \):

\[
\mathbf{y}_k = \mathbf{z}_{k-1} + \left( \frac{\alpha_k - 1}{\alpha_k - 1} \right) (\mathbf{z}_{k-1} - \mathbf{z}_{k-2}), \quad \text{where} \quad 
\alpha_k = \frac{1 + \sqrt{1 + 4\alpha_{k-1}^2}}{2}
\]  

(36)

The approximate solution at the \( k \)-th iteration is then computed from the search point \( \mathbf{y}_k \), which transforms (31) into

\[
\mathbf{z}_k = \mathbf{y}_k - t_k \nabla f(\mathbf{y}_k)
\]  

(37)

It is clear from (36) and (37) that the primary cost of each iteration in the Nesterov method is to evaluate the gradient of \( f \), which is essentially the same as the cost of the gradient step. Hence, the Nesterov method runs much faster than the simple gradient algorithm.

Following the same strategy as adopted by the Nesterov method, we include a search sequence \( \{\mathbf{y}_k\} \), which is used to form the optimization sequence \( \{\mathbf{z}_k\} \) based on (33). This leads to an accelerated gradient algorithm (see Algorithm 7.1) that achieves the optimal convergence rate at \( O\left(\frac{1}{k^2}\right) \).
Algorithm 7.1 Accelerated Gradient Algorithm

1: Initialize: $y_1 = z_0 \in \mathbb{R}^k$, $k = 1$, $\alpha_1 = 1$, max_iter = $N$
2: repeat
3: $z_k = p_{t_k}(y_k)$
4: $\alpha_{k+1} = \frac{1+\sqrt{1+4\alpha_k^2}}{2}$
5: $y_{k+1} = z_k + \left(\frac{\alpha_k-1}{\alpha_{k+1}}\right) (z_k - z_{k-1})$
6: $k = k + 1$
7: until the convergence condition is met or $k = \text{max\_iter}$

7.3. Enhancement of Anchor Service Set Optimization

We further improve the performance of the above optimization process by replacing the standard gradient algorithm with the accelerated gradient algorithm as discussed in Section 7.2. In particular, we include a search sequence $\{Y_k\}$ and each approximate solution $\Phi_k$ will be computed from the search point $Y_k$:

$$\Phi_k = Y_k + t_k \nabla f(Y_k)$$

(38)

where each search point $Y_k$ is computed in the same way as shown in (36). Once $\Phi^*$ is computed, $A^*$ can be computed as:

$$A^* = XZ'(Q^*)^{-1}$$

(39)

where $Q^* = ZZ' + \Phi^*$. Due to the adoption of the accelerated gradient algorithm, the optimization process converges as $O(\frac{1}{k^2})$. 