A Sample Decreasing Threshold Greedy-Based Algorithm for Big Data Summarisation

Teng Li
Cranfield University
Hyo-Sang Shin (h.shin@cranfield.ac.uk)
Cranfield University https://orcid.org/0000-0001-9938-0370
Antonios Tsourdos
Cranfield University

Research

Keywords: Big data summarisation, Submodular maximisation, k-extendible system constraints, Personalised recommendation

Posted Date: November 18th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-107397/v1

Version of Record: A version of this preprint was published on February 9th, 2021. See the published version at https://doi.org/10.1186/s40537-021-00416-y.

RESEARCH

A sample decreasing threshold greedy-based algorithm for big data summarisation

Teng Li, Hyo-Sang Shin* and Antonios Tsourdos

*Correspondence:
h.shin@cranfield.ac.uk
School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield, MK43 0AL UK
Full list of author information is available at the end of the article

Abstract

As the scales of datasets expand rapidly in the applications of big data, increasing efforts have been made to develop fast algorithms. This paper addresses big data summarisation problems using the submodular maximisation approach and proposes an efficient algorithm for maximising general non-negative submodular objective functions subject to k-extendible system constraints. Leveraging the sampling process and the decreasing threshold strategy, we develop an algorithm, named Sample Decreasing Threshold Greedy (SDTG). The proposed algorithm obtains an expected approximation guarantee of $\frac{1}{1+k} - \epsilon$ for maximising monotone submodular functions and of $\frac{k}{(1+k)^2} - \epsilon$ in non-monotone cases with expected computational complexity of $O(\frac{n}{(1+k)\epsilon} \ln \frac{r}{\epsilon})$. Here, r is the largest size of feasible solutions, and $\epsilon \in (0, \frac{1}{1+k})$ is an adjustable designing parameter for the trade-off between the approximation ratio and the computational complexity. The performance of the proposed algorithm is verified through experiments with a movie recommendation system and compared with that of benchmark algorithms.

Keywords: Big data summarisation; Submodular maximisation; k-extendible system constraints; Personalised recommendation

Introduction

The research of big data has received extensive attention due to its great significance [1]. Data summarisation, which involves extracting representative information with certain constraints from a large-scale dataset, is one of the compelling directions of big data processing [2]. Typical applications of big data summarisation include personalised recommendation systems [3–6], exemplar-based clustering [7–9], and summarisation of text [10,11], images [12–14], corpus [8,15], and videos [16,17], just to name a few.

The unprecedented growth of modern datasets requires efficient and effective techniques to process a mass of data. Computational complexity is one of the grand challenges of big data operations [1]. Fortunately, the quality of data summarisation outcome can be often measured by submodular objective functions [11,12,14], where the marginal value of an element decreases as more elements have already been selected, namely diminishing returns [18]. It is well known that the greedy-related algorithms are efficient and can provide an approximation guarantee for maximising submodular functions [19]. Hence, the big data summarisation problem can be handled as maximising a submodular objective function based on a large-scale dataset, meanwhile satisfying a certain constraint or a combination of several constraints [2].

This paper addresses big data summarisation problems using the submodular maximisation approach, especially subject to k-extendible system constraints. Note that the k-extendible system constraint is a general type of constraint that has been widely studied. The concept of k-extendible systems was first introduced by Mestre in 2006 [20]. The intersection of k matroids based on the same ground set is always k-extendible [20]. Many types of constraints handled in submodular maximisation fall into the k-extendible system constraint: cardinality constraint, partition matroid constraint, and k-matroid constraint are good examples of the k-extendible system constraint.

The issue is that finding the optimal solution of submodular maximisation is NP-hard, and the sizes of datasets tend to increase. NP-hard problems are known to significantly suffer from "curse of dimensionality", which implies that the complexity of the problem explodes as the problem size increases. Therefore, the trend of increasing sizes of datasets combined with the NP-hardness of the problem urges the development of more computationally efficient optimisation algorithms. To this end, there have been numerous works recently carried out to develop more efficient constrained submodular maximisation algorithms, and many of them endeavour to increase computational efficiency even by sacrificing some degree of approximation ratio. These works are classified by the types of constraints, and their developments are summarised in the following.

Cardinality constraint The Sieve-Streaming proposed by Badanidiyuru et al. [12] is the first single-pass streaming algorithm for maximising monotone submodular functions, achieving approximation guarantee of $1/2 - \epsilon$ with computational complexity of $O(\frac{n}{\epsilon} \log r)$. Here, n is the size of the ground set, r is the size of the largest feasible solution. Norouzi-Fard et al. [9] proposed another single-pass algorithm Salsa that improved the approximation guarantee to a value better than $1/2$. They also extended their work to a multi-pass algorithm P-Pass that provided the trade-off between the approximation ratio and the number of passes. The Decreasing Threshold Greedy proposed in [21] obtained an approximation ratio of $1 - 1/e - \epsilon$ with time complexity of $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$ for monotone submodular functions. This is the first streaming algorithm whose computational complexity is independent of r. Later, the sampling-based Stochastic Greedy proposed by Mirzasoleiman et al. [22] achieved an expectantly the same approximation ratio with lower time complexity of $O(n \log \frac{1}{\epsilon})$. The Stochastic Greedy gets orders of magnitudes faster by losing only a bit of approximation ratio compared with other benchmark algorithms. Then in [23], Buchbinder et al. extended the analysis to general non-monotone submodular functions and achieved an approximation guarantee of $1/e - \epsilon$ with the computational complexity of $O(\frac{n}{\epsilon^2} \log \frac{1}{\epsilon})$. Recently, Breuer et al. [24] proposed an efficient algorithm Fast for the monotone case, using the adaptive sequencing technique. The Fast algorithm achieves an approximation ratio of $1 - 1/e - \epsilon$, with only $O(n \log \log(r))$ queries.

Matroid constraint The original Greedy algorithm [19] provides an approximation ratio of $1/2$ with time complexity of $O(nr)$ for monotone submodular maximisation. The continuous greedy was utilised to increase the approximation ratio to $1 - 1/e$ [25]

and provide the approximation guarantee for non-monotone submodular functions [26], but such algorithms are quite time-consuming. To remedy this, the idea of decreasing threshold [21] was adopted to reduce the computational complexity [27]. Badanidiyuru and Vondrak [21] proposed a new variant of the continuous greedy algorithm and achieved an approximation ratio of $1 - 1/e - \epsilon$ with complexity of $O(\frac{nr}{\epsilon^4}\log^2\frac{r}{\epsilon})$ for monotone submodular functions. Then, a close variant of the Decreasing Threshold Greedy described in [23] provided an approximation ratio of $1/2 - \epsilon$ with computational complexity of $O(\frac{n}{\epsilon}\log\frac{r}{\epsilon})$ for the monotone case.

k-extendible system constraint The Decreasing Threshold Greedy [21] provides a slightly worse approximation guarantee of $\frac{1}{1+k+\epsilon}$ but lower computational complexity of $O(\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon})$ than the original Greedy algorithm [19] for monotone submodular functions. For the non-monotone case, Gupta et al. [28] achieved an approximation ratio of $\frac{k}{(k+1)(3k+3)}$ with time complexity of $O(nrk)$. Then, the approximation ratio was improved to $\frac{k}{(k+1)(2k+1)}$ by an algorithm called FANTOM proposed by Mirzasoleiman et al. [5] with the same complexity. After this, Feldman et al. [29] made a significant breakthrough in terms of both approximation ratio and time complexity. The Sample Greedy (Sample, for short) they proposed achieved an approximation ratio of $\frac{k}{(k+1)^2}$ with complexity of $O(n + nr/k)$.

Although gradual improvements have been made for general k-extendible system constraints recently, they are not as fruitful as those for cardinality constraints or matroid constraints. An immediate research question would be whether or not we can develop an algorithm that can further improve the efficiency of maximising general non-negative submodular functions subject to k-extendible system constraints.

Inspired by the sampling strategy from [29] and the decreasing threshold idea from [21], we propose an algorithm that is even faster than Sample [29]. The proposed algorithm, which is named as Sample Decreasing Threshold Greedy (SDTG), provides an expected approximation guarantee of $p - \epsilon$ for maximising monotone submodular functions and of $p(1-p)-\epsilon$ for non-monotone cases with expected time complexity of only $O(\frac{pn}{\epsilon}\ln\frac{r}{\epsilon})$, where $p \in (0, \frac{1}{1+k}]$ is the sampling probability and $\epsilon \in (0, p)$ is the threshold decreasing parameter. We fix the sampling probability p as $\frac{1}{1+k}$, then SDTG provides the best approximation ratios for both monotone and non-monotone submodular functions which are $\frac{1}{1+k} - \epsilon$ and $\frac{k}{(1+k)^2} - \epsilon$, respectively. Here, ϵ acts as a design parameter for the trade-off between the approximation ratio and the computational complexity. The proposed algorithm is verified through experiments with a movie recommendation system based on the MovieLens [30] which is a widely used movie information database. Experimental results demonstrate that the proposed algorithm outperforms benchmark algorithms in terms of both solution quality and computational efficiency.

The remaining part of this work is organised as follows. In Section "Preliminaries" we presents some basic knowledge related to the proposed algorithm. Section "Algorithm and Analysis" demonstrates the proposed algorithm and analyses its theoretical performance in detail. The performance and validity of the theoretical results are then testified through experiments with a movie recommendation system in Section "Experiments". Section "Conclusions" provides the conclusions of this paper and possible future research directions.

Preliminaries

This section presents some necessary definitions and basic concepts related to the proposed algorithm. The definitions and concepts can also be found in our previous works [31–33].

Definition 1 (Submodularity [29]) *A set function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is submodular if, $\forall\ X, Y \subseteq \mathcal{N}$,*

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y).$$

where $\mathcal{N}$ is a finite set containing all elements which is named as "ground set". Equivalently, $\forall\ A \subseteq B \subseteq \mathcal{N}$ and $u \in \mathcal{N} - B$,

$$f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B). \tag{1}$$

Definition 2 (Marginal value [34] (*mgv*)) *For a set function $f : 2^{\mathcal{N}} \to \mathbb{R}$, $S \subseteq \mathcal{N}$ and $u \in \mathcal{N}$, define the marginal value of f at S with respect to u as*

$$\Delta f(u|S) \doteq f(S \cup \{u\}) - f(S),$$

where $\doteq$ means equal by definition. We denote marginal value as "mgv" for tidiness.

The inequality (1) is known as the *diminishing return*, which is a crucial property of submodular functions: the *mgv* of a given element u will never increase as more elements have already been added into the set S.

Definition 3 (Monotonicity [34]) *A set function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is monotone if, $\forall A \subseteq B \subseteq \mathcal{N}$, $f(A) \leq f(B)$. f is non-monotone if it is not monotone.*

This paper only considers normalised (i.e. $f(\emptyset) = 0$) non-negative (i.e. $f(S) \geq 0$, $\forall S \subseteq \mathcal{N}$) submodular maximisation problems.

Definition 4 (Matroid [21]) *A matroid is a pair $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{N}$ is a finite set and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is a collection of independent sets, satisfying:*

- $\emptyset \in \mathcal{I}$
- *if $A \subseteq B, B \in \mathcal{I}$, then $A \in \mathcal{I}$*
- *if $A, B \in \mathcal{I}, |A| < |B|$, then $\exists\ u \in B - A$ such that $A \cup \{u\} \in \mathcal{I}$*

Specifically, matroid constraints include uniform matroid constraint and partition matroid constraint. The uniform matroid constraint is also called *cardinality constraint*, which is a special case of matroid constraint where any subset $S \subseteq \mathcal{N}$ satisfying $|S| \leq k$ is independent. The *partition matroid constraint* means that a subset S can contain at most a certain number of elements from each partition.

Definition 5 (Extension [29]) *If an independent set B strictly contains an independent set A, then B is called an extension of A.*

Definition 6 (k-extendible [20]) *A k-extendible system is an independence system $(\mathcal{N}, \mathcal{I})$ that for every independent set $A \in \mathcal{I}$, an extension B of A, and an element $u \notin A$, $A \cup \{u\} \in \mathcal{I}$ there exists a subset $X \subseteq B - A$ with $|X| \leq k$ such that $(B - X) \cup \{u\} \in \mathcal{I}$.*

Intuitively, if an element u is added into an independent set A of a k-extendible system, it requires at most k other elements to be removed from A in order to keep the set independent [29].

The following is an important claim [35] that provides the mathematical foundation for Sample [29] to work well in non-monotone submodular maximisation. Readers are referred to [35] for the proof of Claim 1.

Claim 1 *Let $g : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be a submodular function, and let S be a random subset of $\mathcal{N}$. If each element of S appears with a probability at most p (not necessarily independently). Then, $\mathbb{E}[g(S)] \geq (1 - p)g(\emptyset)$.*

Algorithm and Analysis

This section describes SDTG in Algorithm 1 and analyses its theoretical performance in detail. Note that the proposed algorithm is based on submodular optimisation like in our previous studies [31–33]. Hence the analysis shares some essences of logic in our previous works. An equivalent version of Algorithm 1 is introduced as Algorithm 2 to better analyse SDTG.

Algorithm

Some notations that are used in Algorithm 1 are described in the following: $\mathcal{N}$ is the ground set containing all elements. $\mathcal{I}$ is the collection of all valid sets (independent); r is the maximum size of the valid sets in $\mathcal{I}$; p is the sampling probability (uniform distribution); ϵ is the threshold decreasing parameter determining the decreasing speed of the threshold; S is the selection set containing the selected elements; R is the set containing the remaining elements in the sample set; θ is the decreasing threshold.

This work proposes to leverage the sampling strategy [29] and develop a variant of decreasing threshold idea to design a summarisation algorithm. The structure of Algorithm 1 consists of two phases. The first phase (lines $1 \sim 4$) is sampling where elements are randomly selected from the ground set $\mathcal{N}$ with probability p to form a sample set R. The probability distribution of sampling is uniform. The second phase (lines $5 \sim 22$) is selecting where an independent solution set S is selected from R using decreasing threshold greedy. The initial threshold is set as the largest *mgv* given the empty set and denoted as d (line 5). The terminal threshold is set as $\frac{\epsilon}{r}d$ (line 6). The reason for choosing this value as the termination condition will be given later in the proof part. We call one loop of the inner "for" loops as one *iteration*.

At the beginning of each iteration, SDTG checks independency of $S \cup \{u\}$. If it is not independent, then remove element u from R (lines $8 \sim 9$). Otherwise, calculate the *mgv* of u and compare it with the current threshold θ. If the *mgv* of u is greater than or equals to θ, then add u to S and remove it from R (lines

Algorithm 1 SDTG

Input: $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}, \mathcal{N}, \mathcal{I}, r, p, \epsilon$
Output: A set $S \in \mathcal{I}$

1: $S \leftarrow \emptyset$, $R \leftarrow \emptyset$
2: **for** $u \in \mathcal{N}$ **do**
3: $\quad R \leftarrow R \cup \{u\}$ with probability p
4: **end for**
5: $d \leftarrow \max_{u \in R} \Delta f(u|S)$
6: **for** $(\theta = d; \theta \geq \frac{\epsilon}{r} d; \theta \leftarrow \theta(1-\epsilon))$ **do**
7: $\quad$ **for** $u \in R$ **do**
8: $\quad\quad$ **if** $S \cup \{u\} \notin \mathcal{I}$ **then**
9: $\quad\quad\quad R \leftarrow R - \{u\}$
10: $\quad\quad$ **else**
11: $\quad\quad\quad$ **if** $\Delta f(u|S) \geq \theta$ **then**
12: $\quad\quad\quad\quad S \leftarrow S \cup \{u\}$
13: $\quad\quad\quad\quad R \leftarrow R - \{u\}$
14: $\quad\quad\quad$ **else**
15: $\quad\quad\quad\quad$ **if** $\Delta f(u|S) < \frac{\epsilon}{r} d$ **then**
16: $\quad\quad\quad\quad\quad R \leftarrow R - \{u\}$
17: $\quad\quad\quad\quad$ **end if**
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: **return** S

11 $\sim$ 13). We call such an element u a *qualified element* if the *mgv* of u given S is no less than the current threshold θ. If the *mgv* of an element is already less than $\frac{\epsilon}{r} d$, it will never become greater or equal to $\frac{\epsilon}{r} d$ in subsequent iterations due to submodularity. Therefore, this element can be removed from R immediately, as stated in lines 15$\sim$17. Note that each element in R will be evaluated only for one time under one threshold. If the *mgv* of an element is between $\frac{\epsilon}{r} d$ and θ, this element will remain in R for the next outer loop where the threshold will decrease. The remaining elements in R will be reevaluated and compared with a decreased new threshold. The threshold keeps decreasing after all remaining elements in R have been evaluated until reaching the termination condition.

Analysis

To better analyse the approximation ratio performance of Algorithm 1, some auxiliary variables have been introduced to transform SDTG to an equivalent version, i.e., Algorithm 2.

In Algorithm 2, variables C, S_c, Q, and K_c are introduced only for the convenience of analysis and have no effect on the final output S. Therefore, Algorithm 2 and Algorithm 1 are equivalent in terms of solution quality. The rules of these variables are as follows.

C is a set that contains all considered elements that have *mgv*s greater or equal to the threshold θ in a certain iteration of Algorithm 2 no matter whether they are added into S or not.

S_c is a set that contains the selected elements at the beginning of each iteration. At the end of this iteration, S_c equals to $S - \{c\}$ if c is added into S and Q, otherwise S_c equals to S.

Algorithm 2 Equivalent SDTG

Input: $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}, \mathcal{N}, \mathcal{I}, r, p, \epsilon$
Output: A set $S \in \mathcal{I}$

$S \leftarrow \emptyset$, $\mathcal{N}_s \leftarrow \emptyset$, $R \leftarrow \mathcal{N}$,
$C \leftarrow \emptyset$, $Q \leftarrow OPT$
for $u \in R$ **do**
    $\mathcal{N}_s \leftarrow \mathcal{N}_s \cup \{u\}$ with probability p
end for
$d \leftarrow \max_{u \in \mathcal{N}_s} \Delta f(u|S)$
for $(\theta = d; \theta \geq \frac{\epsilon}{r} d; \theta \leftarrow \theta(1-\epsilon))$ **do**
    for $u \in R$ **do**
        if $S \cup \{u\} \notin \mathcal{I}$ **then**
            $R \leftarrow R - \{u\}$
        else
            if $\Delta f(u|S) \geq \theta$ **then**
                $c \leftarrow u$
                $S_c \leftarrow S$
                $C \leftarrow C \cup \{c\}$
                $R \leftarrow R - \{c\}$
                if $u \in \mathcal{N}_s$ **then**
                    $S \leftarrow S \cup \{c\}$
                    $Q \leftarrow Q \cup \{c\}$
                    Let $K_c \subseteq Q - S$ be the smallest set s.t. $Q - K_c \in \mathcal{I}$
                else
                    if $c \in Q$ **then**
                        $K_c \leftarrow \{c\}$
                    else
                        $K_c \leftarrow \emptyset$
                    end if
                end if
                $Q \leftarrow Q - K_c$
            else
                if $\Delta f(u|S) < \frac{\epsilon}{r} d$ **then**
                    $R \leftarrow R - \{u\}$
                end if
            end if
        end if
    end for
end for
return S

Q is a set that bridges the relationship between the solution S and the optimal solution OPT. Q starts at OPT at the beginning of the algorithm and changes over time. Note that, Q is introduced only for analysis and there is no need to know the exact value of Q or OPT. In each iteration, the element added into S is also added into Q. At the same time, a set K_c is removed from Q to keep the independence of Q if an element c is added into Q. Notice that, if an element c is already in Q and is considered but not added into S at the current iteration, then this element c should be removed from Q.

K_c is a set that is introduced to keep Q independent and help Q to remove c that is not added to S. According to the property of k-extendible systems, Algorithm 2 is able to remove a set $K_c \subseteq Q - S$ which contains at most k elements from Q if an element is added into the currently independent set Q. In addition, if c is not added to S and $c \in Q$ at the beginning of some iteration, then $K_c = \{c\}$.

The theoretical performance of the proposed algorithm SDTG is summarised in Theorem 1.

Theorem 1 *SDTG achieves an approximation guarantee of at least* $\frac{1}{1+k} - \epsilon$ *for monotone submodular maximisation subject to* k*-extendible system constraints and of* $\frac{k}{(1+k)^2} - \epsilon$ *for non-monotone cases with computational complexity of* $O(\frac{n}{(1+k)\epsilon} \ln \frac{r}{\epsilon})$*, where* n *is the size of the ground set,* r *is the largest size of a feasible solution, and* $\epsilon \in (0, \frac{1}{1+k})$ *is the threshold decreasing parameter.*

The computational complexity can be easily proved. Assume that there are totally x number of loops in the outer "for" loop. Thus,

$$(1-\epsilon)^x = \frac{\epsilon}{r}.$$

Solving the above equation, we get

$$x = \frac{\ln \frac{r}{\epsilon}}{\ln \frac{1}{1-\epsilon}} \leq \frac{1}{\epsilon} \ln \frac{r}{\epsilon}.$$

There are expectantly at most $p \cdot n$ function evaluations in each outer loop. Therefore, the time complexity of Algorithm 1 is $O(\frac{pn}{\epsilon} \ln \frac{r}{\epsilon})$. □

The following part of this section analyses the approximation ratios of SDTG in both monotone and non-monotone cases through Algorithm 2.

Lemma 1 $\mathbb{E}[|K_u|] \leq Pr_{max}$ *where* $Pr_{max} = \max(pk, 1-p)$.

Proof We have three cases to analyse, depending on whether the current element u is considered at some point of iteration, i.e. $u \in C$, and whether u is already in Q at the beginning of the iteration in Algorithm 2. Note that the size of K_u is kept as small as possible.

i. If $u \notin C$ for whole iterations, $K_u = \emptyset$ and thus the expectation is obtained as:

$$\mathbb{E}[|K_u|] = 0$$

ii. If $u \in C$ and $u \in Q$ at the beginning of the iteration, then $K_u = \emptyset$ for $u \in \mathcal{N}_s$ and $K_u = \{u\}$ for $u \notin \mathcal{N}_s$. Since u is sampled in $\mathcal{N}_s$ with probability p, the expectation is obtained as:

$$\mathbb{E}[|K_u|] = p \cdot |\emptyset| + (1-p)|\{u\}| = 1 - p.$$

iii. If $u \in C$ and $u \notin Q$ at the beginning of the iteration, then K_u contains at most k elements for $u \in \mathcal{N}_s$, and $K_u = \emptyset$ for $u \notin \mathcal{N}_s$. According to the property of k-extendible systems, if Q becomes dependent after adding u, then Q can remove at most k elements to remain independence. If Q is still independent after adding u, then $K_u = \emptyset$. Therefore, we have

$$\mathbb{E}[|K_u|] \leq p \cdot k + (1-p)|\emptyset| = pk.$$

In summary, $\mathbb{E}[|K_u|] \leq \max(pk, 1-p)$. □

Lemma 2 $\mathbb{E}[f(S)] = \sum_{u \in \mathcal{N}} p\mathbb{E}[\Delta f(u|S_u)]$.

Proof Let us define a random variable G_u such that its value is equal to the increase of $f(S)$ when $u \in \mathcal{N}$ is considered, i.e.

$$f(S) = f(\emptyset) + \sum_{u \in \mathcal{N}} G_u.$$

Note that since f is assumed to be normalised, $f(\emptyset) = 0$. Given the event $\mathcal{E}_u$ specifying all the decisions made before considering u, the conditional expectation of G_u is obtained as

$$\mathbb{E}[G_u|\mathcal{E}_u] = \sum_{G_u} P(G_u|\mathcal{E}_u)G_u.$$

Here, if u is sampled, G_u is equal to $\Delta f(u|S'_u)$ with the probability of $P(G_u|\mathcal{E}_u) = p$, where S'_u is defined as S_u given the event $\mathcal{E}_u$. Note that if u is sampled but not in C, $\Delta f(u|S'_u)$ is defined as 0 by convention. Otherwise if u is not sampled, G_u is zero. Hence, the conditional expectation of G_u is:

$$\mathbb{E}[G_u|\mathcal{E}_u] = p\Delta f(u|S'_u) = p\mathbb{E}[\Delta f(u|S_u)|\mathcal{E}_u]$$

By the law of total expectation, expectation of G_u is obtained as:

$$\mathbb{E}[G_u] = \mathbb{E}[\mathbb{E}[G_u|\mathcal{E}_u)]] = \sum_{\mathcal{E}_u} P(\mathcal{E}_u)\mathbb{E}[G_u|\mathcal{E}_u] = p\mathbb{E}\left[\Delta f(u|S_u)\right]$$

Hence, the expectation of $f(S)$ is obtained as:

$$\mathbb{E}[f(S)] = \sum_{u \in \mathcal{N}} p\mathbb{E}\left[\Delta f(u|S_u)\right]$$

□

Lemma 3 $\mathbb{E}[f(S)] > \frac{(1-\epsilon)p}{(1-\epsilon^2)p+Pr_{max}}\mathbb{E}[f(S \cup OPT)]$.

Proof According to Algorithm 2, at the end of each iteration, the set Q is independent i.e. $Q \in \mathcal{I}$. S is a subset of Q, i.e. $S \subseteq Q$, since every element c that is added to S is also in Q. Therefore, $S \cup \{q\} \in \mathcal{I}\ \forall q \in Q - S$ by the property of independent systems and $|Q - S| \leq r$. At the termination of Algorithm 2, $\Delta f(q|S) < \frac{\epsilon}{r}d\ \forall q \in Q - S$ and $f(S) \geq d$. Thus,

$$\sum_{q \in Q-S} \Delta f(q|S) < \sum_{q \in Q-S} \frac{\epsilon}{r}d \leq \epsilon \cdot \frac{|Q-S|}{r} f(S) \leq \epsilon \cdot f(S).$$

Let $Q - S = \{q_1, q_2, \cdots, q_{|Q-S|}\}$, then

$$\begin{aligned} f(S) &= f(Q) - \sum_{i=1}^{|Q-S|} \Delta f(q_i | S \cup \{q_1, \cdots, q_{i-1}\}) \\ &\geq f(Q) - \sum_{i=1}^{|Q-S|} \Delta f(q_i | S) \qquad \text{(submodularity)} \\ &> f(Q) - \epsilon \cdot f(S). \end{aligned}$$

Rearranging the above inequality yields

$$f(S) > \frac{1}{1+\epsilon} f(Q). \tag{2}$$

Eqn. (2) indicates that if the mgv of an element is less than $\frac{\epsilon}{r}d$, then this element is considered negligible under a small ϵ. This is the reason why the final threshold is set as $\frac{\epsilon}{r}d$.

In a certain iteration and given the current threshold θ, if $u \in C$ it implies that

$$\Delta f(u | S_u) \geq \theta. \tag{3}$$

While if an element $q \in K_u - S$ was not selected before this iteration, then

$$\Delta f(q | S_u) < \theta / (1 - \epsilon). \tag{4}$$

Combining Eqns. (3) and (4) we have

$$\Delta f(u | S_u) > (1 - \epsilon) \Delta f(q | S_u) \ \forall q \in K_u - S. \tag{5}$$

Additionally, any element can be removed from Q at most once. In other words, the element that is contained in K_u at one iteration is always different from other iterations when K_u is not empty. Therefore, the sets $\{K_u - S\}_{u \in \mathcal{N}}$ are disjoint. According to the definition and evolution of Q, Q can be expressed as

$$Q = (OPT - \cup_{u \in \mathcal{N}} K_u) \cup S = (S \cup OPT) - \cup_{u \in \mathcal{N}} (K_u - S). \tag{6}$$

Denote $\mathcal{N}$ as $\mathcal{N} = \{u_1, u_2, \cdots, u_{|\mathcal{N}|}\}$. Then we define Q_u^i as

$$Q_u^i \doteq (S \cup OPT) - \cup_{u \in \mathcal{N}_i} (K_u - S)$$

where $\mathcal{N}_i = \{u_1, \cdots, u_i\}$. Denote K_u and S_u corresponding to u_i in the i-th iteration as K_u^i and S_u^i, respectively. It is clear that $S_u^i \subseteq S \subseteq Q_u^i$. Using Eqn. (6), we have

$$\begin{aligned} f(Q) &= f(S \cup OPT) - \sum_{i=1}^{|\mathcal{N}|} \Delta f(K_u^i - S | Q_u^i) \\ &\geq f(S \cup OPT) - \sum_{i=1}^{|\mathcal{N}|} \sum_{q \in K_u^i - S} \Delta f(q | S_u^i) && \text{(submodularity)} \\ &> f(S \cup OPT) - \sum_{u \in \mathcal{N}} |K_u - S| \frac{1}{1-\epsilon} \Delta f(u | S_u). && \text{(Eqn. (5))} \end{aligned}$$

Taking expectation over $f(S)$ yields

$$\begin{aligned} \mathbb{E}[f(S)] &> \frac{1}{1+\epsilon} \mathbb{E}[f(Q)] && \text{(Eqn. (2))} \\ &> \frac{1}{1+\epsilon} \mathbb{E}\left[f(S \cup OPT) - \sum_{u \in \mathcal{N}} |K_u - S| \frac{1}{1-\epsilon} \Delta f(u | S_u) \right] \\ &\geq \frac{1}{1+\epsilon} \mathbb{E}[f(S \cup OPT)] - \frac{1}{(1+\epsilon)(1-\epsilon)} \cdot \mathbb{E}[|K_u|] \cdot \sum_{u \in \mathcal{N}} \mathbb{E}[\Delta f(u | S_u)] \\ &\geq \frac{1}{1+\epsilon} \mathbb{E}[f(S \cup OPT)] - \frac{1}{(1+\epsilon)(1-\epsilon)} \cdot Pr_{max} \cdot \sum_{u \in \mathcal{N}} \mathbb{E}[\Delta f(u | S_u)] \\ & && \text{(Lemma 1)} \\ &= \frac{1}{1+\epsilon} \mathbb{E}[f(S \cup OPT)] - \frac{1}{(1+\epsilon)(1-\epsilon)} \cdot \frac{Pr_{max}}{p} \cdot \mathbb{E}[f(S)]. \\ & && \text{(Lemma 2)} \end{aligned}$$

The result is clear by rearranging the above inequality. □

Let us finish the proof of Theorem 1 in the following part of this section.

Proof (Theorem 1) Recall that, p is the sampling probability and $p \in (0, 1]$. We have

$$Pr_{max} = \max(pk, 1-p) = \begin{cases} 1-p & \text{for } p \in (0, \frac{1}{1+k}] \\ pk & \text{for } p \in (\frac{1}{1+k}, 1]. \end{cases}$$

It is necessary to analyse the relationship between $f(S \cup OPT)$ and $f(OPT)$ with monotone and non-monotone submodular objective functions, respectively, to get the approximation guarantees for both cases.

- If f is monotone, then $f(S \cup OPT) \geq f(OPT)$. According to Lemma 3,

$$\begin{aligned} \mathbb{E}[f(S)] &> \frac{(1-\epsilon)p}{(1-\epsilon^2)p + Pr_{max}} \cdot \mathbb{E}[f(S \cup OPT)] \\ &\geq \frac{(1-\epsilon)p}{(1-\epsilon^2)p + Pr_{max}} \cdot f(OPT). \end{aligned}$$

When $p \in (0, \frac{1}{1+k}]$, we get

$$\begin{aligned}\mathbb{E}[f(S)] &> \frac{(1-\epsilon)p}{(1-\epsilon^2)p+1-p} \cdot f(OPT) \\ &> (p-\epsilon) \cdot f(OPT).\end{aligned}$$

When $p \in (\frac{1}{1+k}, 1]$, we have

$$\begin{aligned}\mathbb{E}[f(S)] &> \frac{(1-\epsilon)p}{(1-\epsilon^2)p+pk} \cdot f(OPT) \\ &> (\frac{1}{1+k}-\epsilon) \cdot f(OPT).\end{aligned}$$

• If f is non-monotone, we define a new submodular and non-monotone function $h : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ as $h(X) = f(X \cup OPT)\ \forall X \subseteq \mathcal{N}$. Since S contains each element with probability at most p and according to Claim 1, we get

$$\mathbb{E}[f(S \cup OPT)] = \mathbb{E}[h(S)] \geq (1-p)h(\emptyset) = (1-p)f(OPT). \tag{7}$$

Combining Eqn. (7) with Lemma 3 yields

$$\begin{aligned}\mathbb{E}[f(S)] &> \frac{(1-\epsilon)p}{(1-\epsilon^2)p+Pr_{max}} \cdot \mathbb{E}[f(S \cup OPT)] \\ &\geq \frac{(1-\epsilon)p(1-p)}{(1-\epsilon^2)p+Pr_{max}} \cdot f(OPT).\end{aligned}$$

When $p \in (0, \frac{1}{1+k}]$, we get

$$\begin{aligned}\mathbb{E}[f(S)] &> \frac{(1-\epsilon)p(1-p)}{(1-\epsilon^2)p+1-p} \cdot f(OPT) \\ &> [p(1-p)-\epsilon] \cdot f(OPT).\end{aligned}$$

When $p \in (\frac{1}{1+k}, 1]$, we have

$$\begin{aligned}\mathbb{E}[f(S)] &> \frac{(1-\epsilon)p(1-p)}{(1-\epsilon^2)p+pk} \cdot f(OPT) \\ &> (\frac{1}{1+k}-\epsilon)(1-p) \cdot f(OPT).\end{aligned}$$

In summary, if f is monotone, the expected approximation ratios are

$$\mathbb{E}[f(S)] > \begin{cases} (p-\epsilon) \cdot f(OPT) & \text{for } p \in (0, \frac{1}{1+k}] \\ (\frac{1}{1+k}-\epsilon) \cdot f(OPT) & \text{for } p \in (\frac{1}{1+k}, 1]. \end{cases} \tag{8}$$

If f is non-monotone, the expected approximation ratios are

$$\mathbb{E}[f(S)] > \begin{cases} [p(1-p)-\epsilon] \cdot f(OPT) & \text{for } p \in (0, \frac{1}{1+k}] \\ (\frac{1}{1+k}-\epsilon)(1-p) \cdot f(OPT) & \text{for } p \in (\frac{1}{1+k}, 1]. \end{cases} \tag{9}$$

Eqns. (8) and (9) show that, for $p \in (\frac{1}{1+k}, 1]$, the expected approximation ratio becomes stagnated in the monotone case and decreasing in the non-monotone case. Moreover, the computational complexity increases as the sampling probability gets larger. On the other side, for $p \in (0, \frac{1}{1+k}]$, the sampling probability provides trade-off capability between the approximation ratio and computational complexity. As the probability increases for $p \in (0, \frac{1}{1+k}]$, the expected approximation ratios improve for both monotone and non-monotone cases, but the computational complexity also increases. □

Recall that the theoretical time complexity is $O(\frac{pn}{\epsilon} \ln \frac{r}{\epsilon})$. The impact of ϵ on the solution quality and time complexity is more desirable than that of p. Therefore, we set the sampling probability as $p = \frac{1}{1+k}$ and leave ϵ as an adjustable designing parameter for the trade-off of solution quality versus time complexity. According to Eqns. (8) and (9), the best expected approximation guarantees can be readily obtained, when $p = \frac{1}{1+k}$, as:

$$\mathbb{E}[f(S)] > \begin{cases} (\frac{1}{1+k} - \epsilon) \cdot f(OPT) & \text{if } f \text{ is monotone} \\ [\frac{k}{(1+k)^2} - \epsilon] \cdot f(OPT) & \text{if } f \text{ is non-monotone.} \end{cases}$$

Experiments

This section testifies the proposed algorithm SDTG through experiments using a real database and compares its performance with those of Greedy [19] and Sample [29]. Note that the performance of Sample and FANTOM [5] has already been compared in [29].

Experimental setup

The database used in the experiments is MovieLens 20M [30]. This database contains 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users. Movies in the database are classified into 19 genres, such as Action, Comedy, Drama, etc.. In addition, each movie is also scored according to the relevance with 1128 genome tags forming 12 million relevance scores in total.

The objective of the movie recommendation system in the experiments is to select a shortlist of movies that are representative yet diverse for users based on their favourite movie genres. The objective function is introduced from [5] and [29]. Let $\mathcal{N}$ be the set of all movies and G be the set of all movie genres. Denote $\mathcal{N}(g)$ as the set of all movies that belong to the movie genre $g \in G$. Denote $G(i)$ as the set of genres that the movie i belongs to. Note that one movie can belong to different genres, hence $|G(i)| \geq 1$. Let s_{ij} represent the similarity value between movie i and movie j. Denote G_u as the set of all movie genres that the user u likes, $G_u \subseteq G$. The movies that can be considered by the user u is contained in the set $\mathcal{N}_u = \cup_{g \in G_u} \mathcal{N}(g)$. The objective function of movie recommendation for user u is given by

$$f_u(S) = \sum_{i \in S} \sum_{j \in \mathcal{N}_u} s_{ij} - \lambda \sum_{i \in S} \sum_{j \in S} s_{ij} \tag{10}$$

where $\lambda \in [0, 1]$ is the penalty parameter of the similarity between movies within S. The objective function Eqn. (10) is non-negative, non-monotone, and submodular.

The first term of Eqn. (10) reflects the representativeness of the selected movies, and the second term helps to increase diversity. We desire to achieve high objective function value with low computational time.

The similarity value between movie i and movie j can be calculated based on the Euclidean distance of relevance scores

$$s_{ij} = \frac{1}{\sqrt{\sum_{t=1}^{N_t}(\gamma_t^i - \gamma_t^j)^2}}$$

where $N_t = 1128$ is the number of all genome tags, γ_t^i and γ_t^j are the relevance scores in terms of the tag t for movie i and movie j, respectively. The calculation of the similarity map took around 35 days on Cranfield HPC - Delta[1], using 128 CPUs with parallel computing.

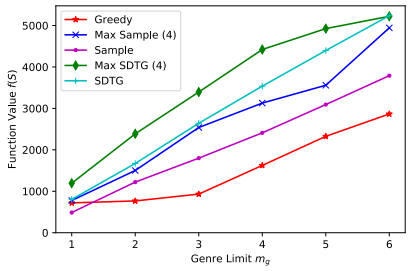
The constraints of the movie recommendation system come from the upper limits of the number of movies in total and in each movie genre. The first constraint is an upper limit m on the total number of movies in the movie recommendation list for the user. The second one is an upper limit m_g (named as a *genre limit*) on the number of movies that belong to the movie genre g. According to [29], the movie recommendation system is subject to a $|G_u|$-extendible system constraint.

In the experiments, suppose that the user's favourite movie genres are Action, Adventure, and Sci-Fi. Then, the constraint of the movie recommendation system is 3-extendible system constraint. Movies with ids less than 30,000 are within consideration since not all movies have genome scores in the database. Set the upper limit on the total number of movies as $m = 15$, and the genre limits as varying numbers from 1 to 6. Set the sampling probability of Sample and SDTG as $p = 0.25$, and the threshold decreasing parameter for SDTG as $\epsilon = 0.2$. Set the penalty parameter as $\lambda = 0.8$. We run Sample and SDTG for 4 rounds and record the best selections which are denoted as Max Sample (4) and Max SDTG (4), respectively. The results of Sample and SDTG are based on 100 rounds of these two algorithms. The running time for all algorithms is measured as the number of objective function evaluations which is independent on the computer conditions. Note that, the experimental results for Sample and SDTG vary somehow each time we run the algorithms because of random sampling.

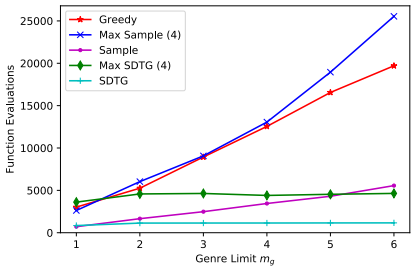
Results

The performance of SDTG is compared with that of benchmark algorithms in terms of both function values and running time in Fig. 1. It is clear from Fig. 1 (a) that, on average, Sample and SDTG related algorithms outperform Greedy in terms of solution quality. The quality of solutions provided by SDTG is better than Sample, although SDTG has a slightly worse theoretical approximation guarantee than Sample. Overall, Max SDTG (4) achieves the highest function value. Fig. 1 (b) shows the number of function evaluations consumed by different algorithms. Four rounds of Sample requires the largest number of function evaluations when $m_g \geq 2$.

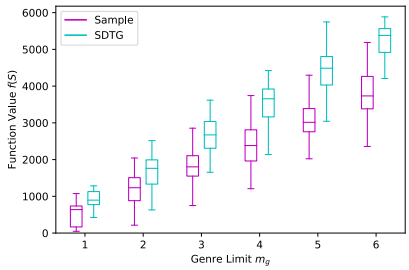
[1]Please refer to https://www.cranfield.ac.uk/study/it-services for details about Delta.



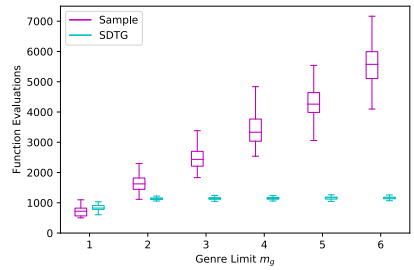
(a) Function Value



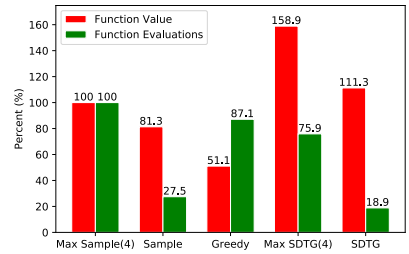
(b) Running Time



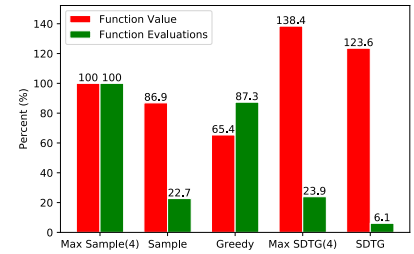
(c) Function Value Distribution



(d) Running Time Distribution



(e) Ratio Comparison $m_g = 2$



(f) Ratio Comparison $m_g = 5$

Figure 1: Performance comparison of different algorithms

Relatively, Greedy requires a bit fewer function evaluations than Max Sample (4). But four rounds of SDTG requires significantly fewer evaluations. Overall, Greedy and Sample-related algorithms consume increasing numbers of function evaluations as m_g goes up. However, the numbers of function evaluations of the SDTG-related algorithms almost stay constant when $m_g \geq 2$. When $m_g = 6$, four rounds of SDTG is even faster than one round of Sample.

Fig. 1 (c) and (d) illustrate the distribution of function value and running time for 100 rounds of Sample and SDTG algorithms. Overall, the function value distribution of SDTG has similar spreads with Sample, but SDTG achieves higher median values than Sample. In terms of running time, SDTG has significantly narrower spreads and lower median values than Sample. The comparison between Sample and SDTG indicates that SDTG not only achieves better function value but also is more efficient and reliable than Sample.

Fig. 1 (e) and (f) demonstrate the ratio comparison of the solution quality and running time of different algorithms. The performance of Max Sample (4) is set

as a baseline for other algorithms in comparison. When $m_g = 2$, Max SDTG (4) achieves significantly better function value than Max Sample (4) but consumes fewer function evaluations. While $m_g = 5$, Max SDTG (4) achieves a much better function value (38.4% more) and dramatically reduces the number of evaluations (76.1% less). On average, SDTG finds better solutions but only consumes 6.1% of function evaluations compared with Max Sample (4). In both cases, Greedy is the least competitive one among all algorithms because it achieves the worst function values and requires the second largest number of evaluations. SDTG provides high-quality solutions yet consumes the least number of function evaluations, which is of great advantage when handling large-scale data sets.

Discussion

Table 1: Movies recommended by different algorithms, $m_g = 2$

Algorithm	Movie id	Genres
Greedy	2367	*Adventure*, Fantasy, Romance, *Sci-Fi*, Thriller
	26513	*Action*, *Adventure*, Comedy, *Sci-Fi*
	4629	*Action*, Crime, Thriller
Max Sample (4)	4629	*Action*, Crime, Thriller
	736	*Action*, *Adventure*, Romance, Thriller
	6106	*Adventure*
	4545	Comedy, *Sci-Fi*
	4738	Romance, *Sci-Fi*
Max SDTG (4)	4369	*Action*, Crime, Thriller
	42	*Action*, Crime, Drama
	146	*Adventure*, Children
	231	*Adventure*, Comedy
	1965	Comedy, *Sci-Fi*
	2656	Horror, *Sci-Fi*

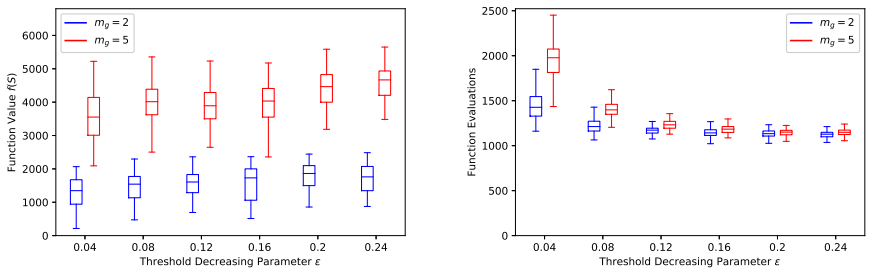
42 146 231 736 1965 2367 2656 4369 4545 4629 4738 6106 26513

The reason why Greedy performs poorly in terms of solution quality is that it greedily selects the best element during each iteration heading to bad local optima. On the other side, with the help of the sampling process, Sample and SDTG related algorithms are able to avoid those elements that can get the algorithms trapped in bad local optima. The threshold in SDTG can further help the algorithm to avoid those local optima. This is why SDTG practically outperforms Sample in terms of solution quality. Table 1 explains the reason in details. According to the definition of the genre limit constraint, at most two movies can be selected from each genre of Adventure, Action, and Sci-Fi when $m_g = 2$. The maximum number of movies without violating the aforementioned constraint is six. Greedy only recommends three movies and reaches the upper genre limit. However, Max Sample (4) and Max SDTG (4) are able to recommend five and six movies, respectively, which better fit the objective of the movie recommendation system.

The reason why Greedy performs poorly in terms of running time is that it has to calculate the *mgv*s of all remaining elements to find the best one given the current selection. Sample is faster than Greedy because it only considers a small portion of the ground set, although it also needs to evaluate all remaining elements in the sample set. Different from Sample, SDTG can stop evaluating once it finds one qualified element and add this element to the selection set immediately. This means that SDTG does not have to evaluate all the remaining elements in the sample set in order to select a new element. Therefore, SDTG consumes fewer average number of function evaluations than Sample does. In addition, the running time of Sample is highly dependent on the size of the sample set because it needs to evaluate all elements in the sample set. On the contrary, SDTG can usually find a qualified element from the front positions of the sample set and stop evaluating. Therefore, the running time of SDTG is less related to the size of the sample set compared with Sample's. This is the reason why the spread of running time distribution of SDTG is narrower than Sample's.

Trade-off of solution quality vs running time

This section also examines the impact of the threshold parameter ϵ on solution quality and running time. This will help us to choose a desirable value of ϵ and to have deeper comprehension of SDTG. The value of ϵ varies from 0.04 to 0.24 with a step of 0.04. Two cases are checked where m_g equals to 2 and 5, respectively. Other settings are as same as previous ones. We run 100 rounds of SDTG and record the function values and the number of function evaluations in each round.



(a) Function Value Distribution (b) Running Time Distribution

Figure 2: The effect of ϵ on function value and running time

Fig. 2 demonstrates the experimental results with varying values of the threshold decreasing parameter. The distributions of function value and running time are illustrated in Fig. 2 (a) and (b), respectively. Fig. 2 (a) shows that the impact of changing ϵ on function values is not significant. Function values fluctuate slightly when $\epsilon \geq 0.08$. However, the solution quality for both $m_g = 2$ and $m_g = 5$ is obviously worse when m_g equals to 0.04 than larger values. This is because the threshold decreases very slowly with an extremely small ϵ. In this case, the *mgv* of the element selected by SDTG in each iteration is very close to the largest one which is selected by Sample. As mentioned before, the decreasing threshold can also help SDTG to avoid local optima. An extremely small ϵ makes SDTG close

to Sample, which weakens the advantage of the decreasing threshold. Fig. 2 (b) shows that the median values of running time decrease obviously as ϵ increases. The spreads of running time also become narrower as ϵ goes up. The reason is that the threshold decreases faster with a larger ϵ. When evaluating the *mgv*s of the remaining elements one by one, SDTG can find a qualified element more quickly with a smaller threshold. The running time of SDTG also becomes less dependent on the size of the sample set.

Conclusions

This paper has presented an efficient algorithm, Sample Decreasing Threshold Greedy (SDTG), to deal with big data summarisation problems. The proposed algorithm achieves an expected approximation ratio of $\frac{k}{(1+k)^2} - \epsilon$ for maximising general non-monotone submodular objective functions subject to k-extendible system constraints with only $O(\frac{n}{(1+k)\epsilon} \ln \frac{r}{\epsilon})$ value oracle calls. The performance of SDTG is testified and compared with benchmark algorithms through experiments with a movie recommendation system based on a widely-used movie information database. The experimental results indicate that the proposed algorithm has great application potentials in large-scale discrete optimisation problems where the sizes of data sets are enormous such as the applications of machine learning and big data science. We believe that our results are also instrumental for the personalised recommendation systems on the internet platforms, like Netflix, YouTube, and Amazon, etc.. SDTG can be further accelerated by adapting the Lazy Greedy strategy [36]. Future research direction could also be accelerating the proposed algorithm by combining distributed computing.

Abbreviations

SDTG: Sample decreasing threshold greedy; *mgv*: Marginal value; OPT: Optimal solution; Max Sample (4): Run 4 rounds of Sample and get the maximum function value; Max SDTG (4): Run 4 rounds of SDTG and get the maximum function value.

Availability of data and materials

The datasets generated during the current study are available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Author's contributions

TL contributed to the algorithm design and analysis, experiments, and manuscript drafting. HS contributed to the theoretical and experimental analysis, manuscript drafting. AT helped to arrange the resources required by the experiments. All authors read and approved the manuscript.

Acknowledgements

The authors thank the Cranfield IT Department team for helping with the Cranfield HPC - Delta operations.

References

1. Jin X, Wah BW, Cheng X, Wang Y. Significance and challenges of big data research. Big Data Research. 2015;2(2):59–64.
2. Mirzasoleiman B. Big data summarization using submodular functions. ETH Zurich; 2017.
3. Tschiatschek S, Djolonga J, Krause A. Learning probabilistic submodular diversity models via noise contrastive estimation. In: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS); 2016. p. 770–779.
4. Yu Q, Xu EL, Cui S. Submodular maximization with multi-knapsack constraints and its applications in scientific literature recommendations. In: 2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP); 2016. p. 1295–1299.

5. Mirzasoleiman B, Badanidiyuru A, Karbasi A. Fast constrained submodular maximization: Personalized data summarization. In: Proceedings of the 33rd International Conference on Machine Learning (ICML). vol. 48; 2016. p. 1358–1367.
6. Mirzasoleiman B, Karbasi A, Krause A. Deletion-robust submodular maximization: Data summarization with the right to be forgotten. In: Proceedings of the 34th International Conference on Machine Learning (ICML). vol. 70; 2017. p. 2449–2458.
7. Mirzasoleiman B, Karbasi A, Sarkar R, Krause A. Distributed submodular maximization: Identifying representative elements in massive data. In: Advances in Neural Information Processing Systems (NIPS); 2013. p. 2049–2057.
8. Mirzasoleiman B, Karbasi A, Sarkar R, Krause A. Distributed submodular maximization. The Journal of Machine Learning Research. 2016;17(1):8330–8373.
9. Norouzi-Fard A, Tarnawski J, Mitrović S, Zandieh A, Mousavifar A, Svensson O. Beyond $1/2$-approximation for submodular maximization on massive data streams. In: Proceedings of the 35th International Conference on Machine Learning (ICML); 2018. .
10. Balkanski E, Mirzasoleiman B, Krause A, Singer Y. Learning sparse combinatorial representations via two-stage submodular maximization. In: Proceedings of the 33rd International Conference on Machine Learning (ICML); 2016. p. 2207–2216.
11. Lavania C, Bilmes J. Auto-summarization: A step towards unsupervised learning of a submodular mixture. In: Proceedings of the 2019 SIAM International Conference on Data Mining (SDM). SIAM; 2019. p. 396–404.
12. Badanidiyuru A, Mirzasoleiman B, Karbasi A, Krause A. Streaming submodular maximization: Massive data summarization on the fly. In: 20th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD). ACM; 2014. p. 671–680.
13. Balkanski E, Breuer A, Singer Y. Non-monotone submodular maximization in exponentially fewer iterations. In: 32nd Conference on Neural Information Processing Systems (NeurIPS 2018); 2018. p. 2353–2364.
14. Mitrovic M, Kazemi E, Zadimoghaddam M, Karbasi A. Data summarization at scale: A two-stage submodular approach. In: Proceedings of the 35th International Conference on Machine Learning (ICML); 2018. .
15. Mirzasoleiman B, Karbasi A, Badanidiyuru A, Krause A. Distributed submodular cover: Succinctly summarizing massive data. In: Advances in Neural Information Processing Systems (NIPS); 2015. p. 2881–2889.
16. Xu J, Mukherjee L, Li Y, Warner J, Rehg JM, Singh V. Gaze-enabled egocentric video summarization via constrained submodular maximization. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2015. p. 2235–2244.
17. Gygli M, Grabner H, Van Gool L. Video summarization by learning submodular mixtures of objectives. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2015. p. 3090–3098.
18. Krause A, Guestrin C. Near-optimal observation selection using submodular functions. In: Proceedings of the 22nd National Conference on Artificial Intelligence. vol. 2. AAAI Press; 2007. p. 1650–1654.
19. Nemhauser GL, Wolsey LA, Fisher ML. An analysis of approximations for maximizing submodular set functions — I. Mathematical Programming. 1978;14(1):265–294.
20. Mestre J. Greedy in approximation algorithms. In: European Symposium on Algorithms (ESA). Springer; 2006. p. 528–539.
21. Badanidiyuru A, Vondrák J. Fast algorithms for maximizing submodular functions. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM; 2014. p. 1497–1514.
22. Mirzasoleiman B, Badanidiyuru A, Karbasi A, Vondrák J, Krause A. Lazier than lazy greedy. In: 29th AAAI Conference on Artificial Intelligence. AAAI Press; 2015. p. 1812–1818.
23. Buchbinder N, Feldman M, Schwartz R. Comparing apples and oranges: Query trade-off in submodular maximization. Mathematics of Operations Research. 2016;42(2):308–329.
24. Breuer A, Balkanski E, Singer Y. The FAST algorithm for submodular maximization. arXiv preprint arXiv:190706173. 2019;.
25. Calinescu G, Chekuri C, Pál M, Vondrák J. Maximizing a monotone submodular function subject to a matroid constraint. SIAM Journal on Computing. 2011;40(6):1740–1766.
26. Feldman M, Naor J, Schwartz R. A unified continuous greedy algorithm for submodular maximization. In: 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS). IEEE; 2011. p. 570–579.
27. Segui-Gasco P, Shin HS. Fast non-monotone submodular maximisation subject to a matroid constraint. arXiv preprint arXiv:170306053. 2017;.
28. Gupta A, Roth A, Schoenebeck G, Talwar K. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In: International Workshop on Internet and Network Economics (WINE). Springer; 2010. p. 246–257.
29. Feldman M, Harshaw C, Karbasi A. Greed is good: Near-optimal submodular maximization via greedy optimization. In: Proceedings of the 2017 Conference on Learning Theory (COLT). vol. 65. PMLR; 2017. p. 1–27.
30. Harper FM, Konstan JA. The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TIIS). 2016;5(4):19:1–19.
31. Li T, Shin HS, Tsourdos A. Fast submodular maximization subject to k-extendible system constraints. arXiv preprint arXiv:181107673v1. 2018;.
32. Shin HS, Li T, Segui-Gasco P. Sample greedy based task allocation for multiple robot systems. arXiv preprint arXiv:190103258. 2019;.
33. Li T, Shin HS, Tsourdos A. Threshold greedy based task allocation for multiple robot operations. arXiv preprint arXiv:190901239. 2019;.
34. Krause A, Golovin D. Submodular function maximization; 2014.
35. Buchbinder N, Feldman M, Naor JS, Schwartz R. Submodular maximization with cardinality constraints. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM; 2014. p. 1433–1452.
36. Minoux M. Accelerated greedy algorithms for maximizing submodular set functions. In: Optimization

Techniques. Springer; 1978. p. 234–243.

Figures

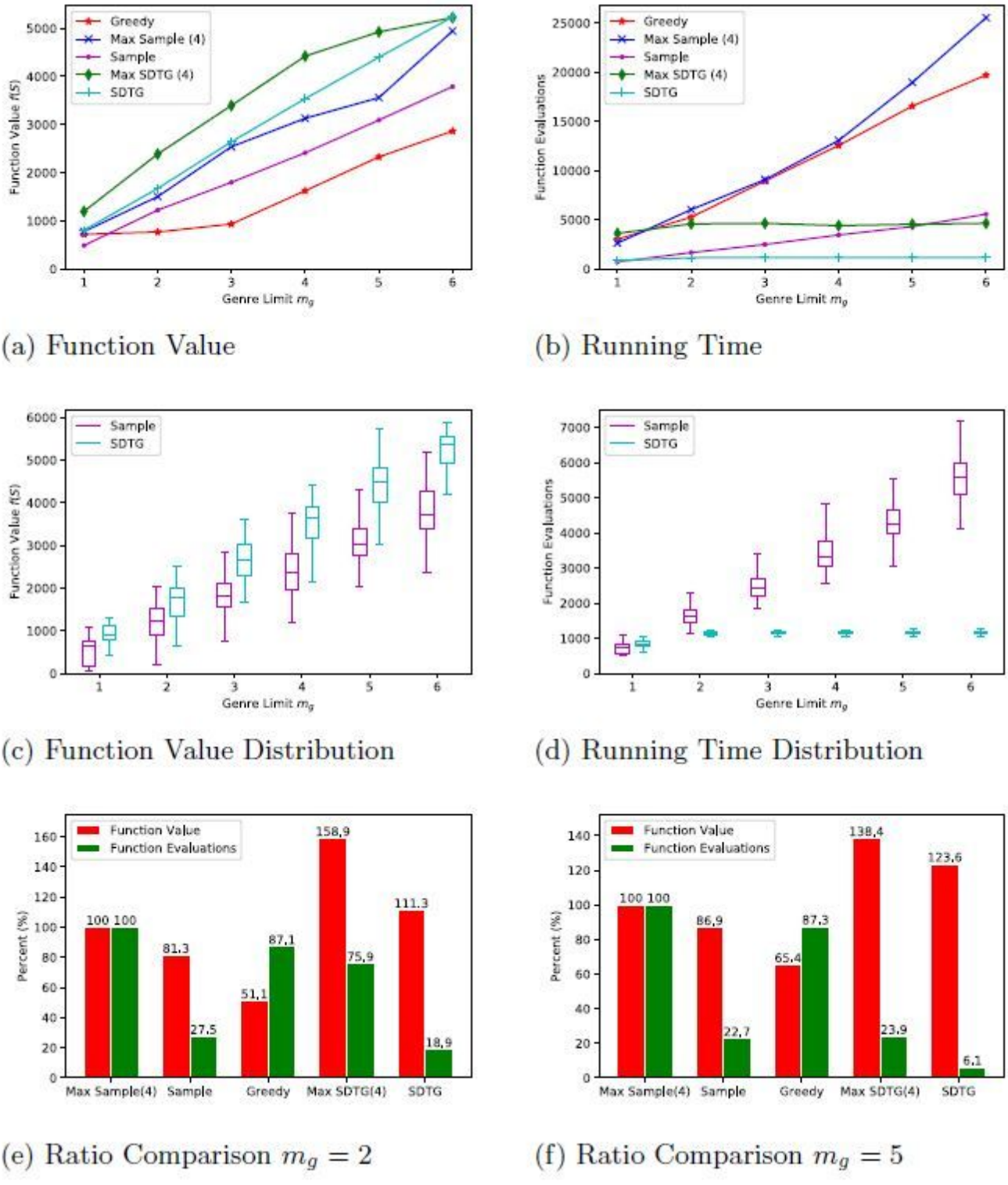
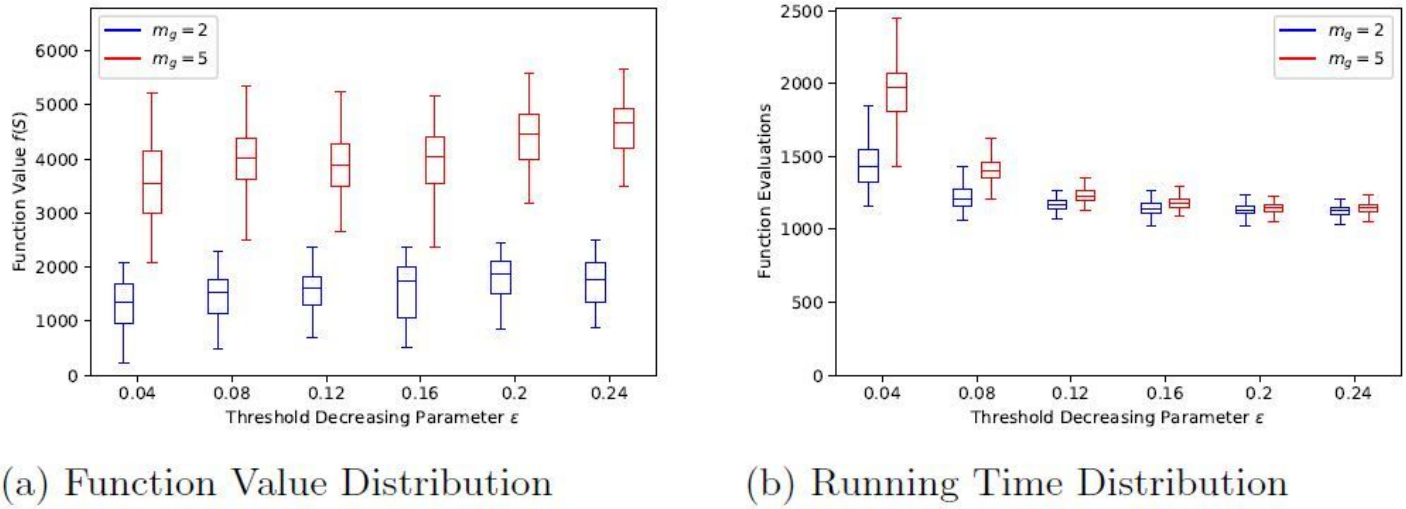


Figure 1

Performance comparison of different algorithms



(a) Function Value Distribution

(b) Running Time Distribution

Figure 2

The effect of ϵ on function value and running time