Parallel Double Greedy Submodular Maximization

Xinghao Pan[1] **Stefanie Jegelka**[1] **Joseph Gonzalez**[1] **Joseph Bradley**[1] **Michael I. Jordan**[1,2]
[1]Department of Electrical Engineering and Computer Science, and [2]Department of Statistics
University of California, Berkeley, Berkeley, CA USA 94720
{xinghao,stefje,jegonzal,josephkb,jordan}@eecs.berkeley.edu

Abstract

Many machine learning problems can be reduced to the maximization of submodular functions. Although well understood in the serial setting, the parallel maximization of submodular functions remains an open area of research with recent results [1] only addressing monotone functions. The optimal algorithm for maximizing the more general class of non-monotone submodular functions was introduced by Buchbinder et al. [2] and follows a strongly serial double-greedy logic and program analysis. In this work, we propose two methods to parallelize the double-greedy algorithm. The first, *coordination-free* approach emphasizes speed at the cost of a weaker approximation guarantee. The second, *concurrency control* approach guarantees a tight 1/2-approximation, at the quantifiable cost of additional coordination and reduced parallelism. As a consequence we explore the tradeoff space between guaranteed performance and objective optimality. We implement and evaluate both algorithms on multi-core hardware and billion edge graphs, demonstrating both the scalability and tradeoffs of each approach.

1 Introduction

Many important problems including sensor placement [3], image co-segmentation [4], MAP inference for determinantal point processes [5], influence maximization in social networks [6], and document summarization [7] may be expressed as the maximization of a submodular function. The submodular formulation enables the use of targeted algorithms [2, 8] that offer theoretical worst-case guarantees on the quality of the solution. For several maximization problems of *monotone* submodular functions (satisfying $F(A) \leq F(B)$ for all $A \subseteq B$), a simple greedy algorithm [8] achieves the optimal approximation factor of $1 - \frac{1}{e}$. The optimal result for the wider, important class of *non-monotone* functions — an approximation guarantee of $1/2$ — is much more recent, and achieved by a *double greedy* algorithm by Buchbinder et al. [2].

While theoretically optimal, in practice these algorithms do not scale to large real world problems, since the inherently serial nature of the algorithms poses a challenge to leveraging advances in parallel hardware. This limitation raises the question of parallel algorithms for submodular maximization that ideally preserve the theoretical bounds, or weaken them gracefully, in a quantifiable manner.

In this paper, we address the challenge of parallelization of greedy algorithms, in particular the double greedy algorithm, from the perspective of *parallel transaction processing systems*. This alternative perspective allows us to apply advances in database research ranging from fast coordination-free approaches with limited guarantees to sophisticated concurrency control techniques which ensure a direct correspondence between parallel and serial executions at the expense of increased coordination.

We develop two parallel algorithms for the maximization of non-monotone submodular functions that operate at different points along the coordination tradeoff curve. We propose CF-2g as a coordination-free algorithm and characterize the effect of reduced coordination on the approximation ratio. By bounding the possible outcomes of concurrent transactions we introduce the CC-2g algorithm which

guarantees serializable parallel execution and retains the optimality of the double greedy algorithm at the expense of increased coordination. The primary contributions of this paper are:

1. We propose two parallel algorithms for unconstrained non-monotone submodular maximization, which trade off parallelism and tight approximation guarantees.
2. We provide approximation guarantees for CF-2g and analytically bound the expected loss in objective value for set-cover with costs and max-cut as running examples.
3. We prove that CC-2g preserves the optimality of the serial double greedy algorithm and analytically bound the additional coordination overhead for covering with costs and max-cut.
4. We demonstrate empirically using two synthetic and four real datasets that our parallel algorithms perform well in terms of both speed and objective values.

The rest of the paper is organized as follows. Sec. 2 discusses the problem of submodular maximization and introduces the double greedy algorithm. Sec. 3 provides background on concurrency control mechanisms. We describe and provide intuition for our CF-2g and CC-2g algorithms in Sec. 4 and Sec. 5, and then analyze the algorithms both theoretically (Sec. 6) and empirically (Sec. 7).

2 Submodular Maximization

A set function $F : 2^V \to \mathbb{R}$ defined over subsets of a ground set V is *submodular* if it satisfies *diminishing marginal returns*: for all $A \subseteq B \subseteq V$ and $e \notin B$, it holds that $F(A \cup \{e\}) - F(A) \geq F(B \cup \{e\}) - F(B)$. Throughout this paper, we will assume that F is nonnegative and $F(\emptyset) = 0$. Submodular functions have emerged in areas such as game theory [9], graph theory [10], combinatorial optimization [11], and machine learning [12, 13]. Casting machine learning problems as submodular optimization enables the use of algorithms for submodular maximization [2, 8] that offer theoretical worst-case guarantees on the quality of the solution.

While those algorithms confer strong guarantees, their design is inherently serial, limiting their usability in large-scale problems. Recent work has addressed faster [14] and parallel [1, 15, 16] versions of the greedy algorithm by Nemhauser et al. [8] for maximizing *monotone* submodular functions that satisfy $F(A) \leq F(B)$ for any $A \subseteq B \subseteq V$. However, many important applications in machine learning lead to *non-monotone* submodular functions. For example, graphical model inference [5, 17], or trading off any submodular gain maximization with costs (functions of the form $F(S) = G(S) - \lambda M(S)$, where $G(S)$ is monotone submodular and $M(S)$ a linear (modular) cost function), such as for utility-privacy tradeoffs [18], require maximizing non-monotone submodular functions. For non-monotone functions, the simple greedy algorithm in [8] can perform arbitrarily poorly (see Appendix H.1 for an example). Intuitively, the introduction of additional elements with monotone submodular functions never decreases the objective while introducing elements with non-monotone submodular functions can *decrease* the objective to its minimum. For non-monotone functions, Buchbinder et al. [2] recently proposed an optimal double greedy algorithm that works well in a serial setting. In this paper, we study parallelizations of this algorithm.

The serial double greedy algorithm. The serial double greedy algorithm of Buchbinder et al. [2] (Ser-2g, in Alg. 3) maintains two sets $A^i \subseteq B^i$. Initially, $A^0 = \emptyset$ and $B^0 = V$. In iteration i, the set A^{i-1} contains the items selected before item/iteration i, and B^{i-1} contains A^i and the items that are so far undecided. The algorithm serially passes through the items in V and determines online whether to keep item i (add to A^i) or discard it (remove from B^i), based on a threshold that trades off the gain $\Delta_+(i) = F(A^{i-1} \cup i) - F(A^{i-1})$ of adding i to the currently selected set A^{i-1}, and the gain $\Delta_-(i) = F(B^{i-1} \setminus i) - F(B^{i-1})$ of removing i from the candidate set, estimating its complementarity to other remaining elements. For any element ordering, this algorithm achieves a tight 1/2-approximation in expectation.

3 Concurrency Patterns for Parallel Machine Learning

In this paper we adopt a transactional view of the program state and explore parallelization strategies through the lens of parallel transaction processing systems. We recast the program state (the sets A and B) as data, and the operations (adding elements to A and removing elements from B) as

transactions. More precisely we reformulate the double greedy algorithm (Alg. 3) as a series of *exchangeable*, *Read-Write* transactions of the form:

$$T_e(A, B) \triangleq \begin{cases} (A \cup e, B) & \text{if } u_e \leq \frac{[\Delta_+(A,e)]_+}{[\Delta_+(A,e)]_+ + [\Delta_-(B,e)]_+} \\ (A, B \backslash e) & \text{otherwise.} \end{cases} \tag{1}$$

The transaction T_e is a function from the sets A and B to new sets A and B based on the element $e \in V$ and the predetermined random bits u_e for that element.

By composing the transactions $T_n(T_{n-1}(\ldots T_1(\emptyset, V)))$ we recover the serial double-greedy algorithm defined in Alg. 3. In fact, any ordering of the *serial* composition of the transactions recovers a permuted execution of Alg. 3 and therefore the optimal approximation algorithm. However, this raises the question: *is it possible to apply transactions in parallel?* If we execute transactions T_i and T_j, with $i \neq j$, in parallel we need a method to merge the resulting program states. In the context of the double greedy algorithm, we could define the parallel execution of two transactions as:

$$T_i(A, B) + T_j(A, B) \triangleq (T_i(A, B)_A \cup T_j(A, B)_A,\ T_i(A, B)_B \cap T_j(A, B)_B)\,, \tag{2}$$

the union of the resulting A and the intersection of the resulting B. While we can easily generalize Eq. (2) to many parallel transactions, we cannot always guarantee that the result will correspond to a serial composition of transactions. As a consequence, we cannot directly apply the analysis of Buchbinder et al. [2] to derive strong approximation guarantees for the parallel execution.

Fortunately, several decades of research [19, 20] in database systems have explored efficient parallel transaction processing. In this paper we adopt a coordinated bounds approach to parallel transaction processing in which parallel transactions are constructed under bounds on the possible program state. If the transaction could violate the bound then it is processed serially on the server. By adjusting the definition of the bound we can span a space of coordination-free to serializable executions.

Algorithm 1: Generalized transactions

```
for p ∈ {1, ..., P} do in parallel
    while ∃ element to process do
        e = next element to process
        (g_e, i) = requestGuarantee(e)
        ∂_i = propose(e, g_e)
        commit(e, i, ∂_i) // Non-blocking
```

Algorithm 2: Commit transaction i

```
wait until ∀j < i, processed(j) = true
Atomically
    if ∂_i = FAIL then
        // Deferred proposal
        ∂_i = propose(e, 𝔖)
    // Advance the program state
    𝔖 ← ∂_i(𝔖)
```

Figure 1: Algorithm for generalized transactions. Each transaction requests its position i in the commit ordering, as well as the bounds $\mathfrak{g}_e$ that are guaranteed to hold when it commits. Transactions are also guaranteed to be committed according to the given ordering.

In Fig. 1 we describe the coordinated bounds transaction pattern. The clients (Alg. 1), in parallel, construct and commit transactions under bounded assumptions about the program state $\mathfrak{S}$ (*i.e.,* the sets A and B). Transactions are constructed by requesting the latest bound $\mathfrak{g}_e$ on $\mathfrak{S}$ at logical time i and computing a change ∂_i to $\mathfrak{S}$ (*e.g.,* Add e to A). If the bound is insufficient to construct the transaction then $\partial_i =$ FAIL is returned. The client then sends the proposed change ∂_i to the server to be committed atomically and proceeds to the next element without waiting for a response.

The server (Alg. 2) *serially* applies the transactions advancing the program state (*i.e.,* adding elements to A or removing elements from B). If the bounds were insufficient and the transaction failed at the client (*i.e.,* $\partial_i =$ FAIL) then the server *serially* reconstructs and applies the transaction under the true program state. Moreover, the server is responsible for deriving bounds, processing transactions in the logical order i, and producing the serializable output $\partial_n(\partial_{n-1}(\ldots \partial_1(\mathfrak{S})))$.

This model achieves a high degree of parallelism when the cost of constructing the transaction dominates the cost of applying the transaction. For example, in the case of submodular maximization, the cost of constructing the transaction depends on evaluating the marginal gains with respect to changes in A and B while the cost of applying the transaction reduces to setting a bit. It is also essential that only a few transactions fail at the client. Indeed, the analysis of these systems focuses on ensuring that the majority of the transactions succeed.

Algorithm 3: Ser-2g: serial double greedy

$A^0 = \emptyset$, $B^0 = V$
for $i = 1$ *to* n **do**
  $\Delta_+(i) = F(A^{i-1} \cup i) - F(A^{i-1})$
  $\Delta_-(i) = F(B^{i-1} \backslash i) - F(B^{i-1})$
  Draw $u_i \sim Unif(0,1)$
  if $u_i < \frac{[\Delta_+(i)]_+}{[\Delta_+(i)]_+ + [\Delta_-(i)]_+}$ **then**
    $A^i := A^{i-1} \cup i$; $B^i := B^{i-1}$
  else $A^i := A^{i-1}$; $B^i := B^{i-1} \backslash i$

Algorithm 4: CF-2g: coord-free double greedy

$\widehat{A} = \emptyset$, $\widehat{B} = V$
for $p \in \{1, \ldots, P\}$ **do in parallel**
  while $\exists$ *element to process* **do**
    $e =$ next element to process
    $\widehat{A}_e = \widehat{A}$; $\widehat{B}_e = \widehat{B}$
    $\Delta_+^{\max}(e) = F(\widehat{A}_e \cup e) - F(\widehat{A}_e)$
    $\Delta_-^{\max}(e) = F(\widehat{B}_e \backslash e) - F(\widehat{B}_e)$
    Draw $u_e \sim Unif(0,1)$
    if $u_e < \frac{[\Delta_+^{\max}(e)]_+}{[\Delta_+^{\max}(e)]_+ + [\Delta_-^{\max}(e)]_+}$ **then**
      $\widehat{A}(e) \leftarrow 1$
    else $\widehat{B}(e) \leftarrow 0$

Algorithm 5: CC-2g: concurrency control

1 $\widehat{A} = \widetilde{A} = \emptyset$, $\widehat{B} = \widetilde{B} = V$
2 **for** $i = 1, \ldots, |V|$ **do** processed$(i) = false$
3 $\iota = 0$
4 **for** $p \in \{1, \ldots, P\}$ **do in parallel**
5 **while** $\exists$ *element to process* **do**
6 $e =$ next element to process
7 $(\widehat{A}_e, \widetilde{A}_e, \widehat{B}_e, \widetilde{B}_e, i)$ = getGuarantee(e)
8 (result, u_e) = propose(e, $\widehat{A}_e$, $\widetilde{A}_e$, $\widehat{B}_e$, $\widetilde{B}_e$)
9 commit(e, i, u_e, result)

Algorithm 6: CC-2g getGuarantee(e)

1 $\widetilde{A}(e) \leftarrow 1$; $\widetilde{B}(e) \leftarrow 0$
2 $i = \iota$; $\iota \leftarrow \iota + 1$
3 $\widehat{A}_e = \widehat{A}$; $\widehat{B}_e = \widehat{B}$
4 $\widetilde{A}_e = \widetilde{A}$; $\widetilde{B}_e = \widetilde{B}$
5 **return** $(\widehat{A}_e, \widetilde{A}_e, \widehat{B}_e, \widetilde{B}_e, i)$

Algorithm 7: CC-2g propose

1 $\Delta_+^{\min}(e) = F(\widetilde{A}_e) - F(\widetilde{A}_e \backslash e)$
2 $\Delta_+^{\max}(e) = F(\widehat{A}_e \cup e) - F(\widehat{A}_e)$
3 $\Delta_-^{\min}(e) = F(\widetilde{B}_e) - F(\widetilde{B}_e \cup e)$
4 $\Delta_-^{\max}(e) = F(\widehat{B}_e \backslash e) - F(\widehat{B}_e)$
5 Draw $u_e \sim Unif(0,1)$
6 **if** $u_e < \frac{[\Delta_+^{\min}(e)]_+}{[\Delta_+^{\min}(e)]_+ + [\Delta_-^{\max}(e)]_+}$ **then**
7 result $\leftarrow 1$
8 **else if** $u_e > \frac{[\Delta_+^{\max}(e)]_+}{[\Delta_+^{\max}(e)]_+ + [\Delta_-^{\min}(e)]_+}$ **then**
9 result $\leftarrow -1$
10 **else** result $\leftarrow$ FAIL
11 **return** (result, u_e)

Algorithm 8: CC-2g: commit(e, i, u_e, result)

1 **wait until** $\forall j < i$, processed$(j) = true$
2 **if** *result* $=$ *FAIL* **then**
3 $\Delta_+^{\text{exact}}(e) = F(\widehat{A} \cup e) - F(\widehat{A})$
4 $\Delta_-^{\text{exact}}(e) = F(\widehat{B} \backslash e) - F(\widehat{B})$
5 **if** $u_e < \frac{[\Delta_+^{exact}(e)]_+}{[\Delta_+^{exact}(e)]_+ + [\Delta_-^{exact}(e)]_+}$ **then** result $\leftarrow 1$
6 **else** result $\leftarrow -1$
7 **if** *result* $= 1$ **then** $\widehat{A}(e) \leftarrow 1$; $\widetilde{B}(e) \leftarrow 1$
8 **else** $\widetilde{A}(e) \leftarrow 0$; $\widehat{B}(e) \leftarrow 0$
9 processed$(i) = true$

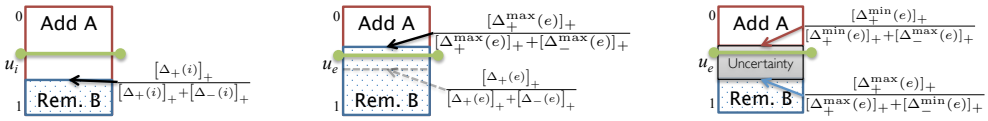
4 Coordination-Free Double Greedy Algorithm

The coordination-free approach attempts to reduce the need to coordinate guarantees and the logical ordering. This is achieved by operating on potentially stale states: the transaction guarantee reduces to requiring $\mathfrak{g}_e$ be a stale version of $\mathfrak{S}$, and the logical ordering is implicitly defined by the time of commit. In using these weak guarantees, CF-2g is overly optimistically assuming that concurrent transactions are independent, which could potentially lead to erroneous decisions.

Alg. 4 is the coordination-free parallel double greedy algorithm.[1] CF-2g closely resembles the serial Ser-2g, but the elements $e \in V$ are no longer processed in a fixed order. Thus, the sets A, B are replaced by potentially stale local estimates (bounds) $\widehat{A}, \widehat{B}$, where $\widehat{A}$ is a subset of the true A and $\widehat{B}$ is a superset of the actual B on each iteration. These bounding sets allow us to compute bounds $\Delta_+^{\max}, \Delta_-^{\max}$ which approximate Δ_+, Δ_- from the serial algorithm. We now formalize this idea.

To analyze the CF-2g algorithm we order the elements $e \in V$ according to the commit time (*i.e.,* when Alg. 4 line 8 is executed). Let $\iota(e)$ be the position of e in this total ordering on elements. This

[1] We present only the parallelized probabilistic versions of [2]. Both parallel algorithms can be easily extended to the deterministic version of [2]; CF-2g can also be extended to the multilinear version of [2].



(a) Ser-2g (b) CF-2g (c) CC-2g

Figure 2: Illustration of algorithms. **(a)** Ser-2g computes a threshold based on the true values Δ_+, Δ_-, and chooses an action based by comparing a uniform random u_i against the threshold. **(b)** CF-2g approximates the threshold based on stale $\widehat{A}$, $\widehat{B}$, possibly choosing the wrong action. **(c)** CC-2g computes two thresholds based on the bounds on A, B, which defines an uncertainty region where it is not possible to choose the correct action locally. If the random value u_e falls inside the uncertainty interval than the transaction FAILS and must be recomputed serially by the server; otherwise the transaction holds under all possible global states.

ordering allows us to define monotonically non-decreasing sets $A^i = \{e' : e' \in A, \iota(e') < i\}$ where A is the final returned set, and monotonically non-increasing sets $B^i = A^i \cup \{e' : \iota(e') \geq i\}$. The sets A^i, B^i provide a serialization against which we can compare CF-2g; in this serialization, Alg. 3 computes $\Delta_+(e) = F(A^{\iota(e)-1} \cup e) - F(A^{\iota(e)-1})$ and $\Delta_-(e) = F(B^{\iota(e)-1} \backslash e) - F(B^{\iota(e)-1})$. On the other hand, CF-2g uses stale versions[2] $\widehat{A}_e$, $\widehat{B}_e$: Alg. 4 computes $\Delta_+^{\max}(e) = F(\widehat{A}_e \cup e) - F(\widehat{A}_e)$ and $\Delta_-^{\max}(e) = F(\widehat{B}_e \backslash e) - F(\widehat{B}_e)$.

The next lemma shows that $\widehat{A}_e, \widehat{B}_e$ are bounding sets for the serialization's sets $A^{\iota(e)-1}, B^{\iota(e)-1}$. Intuitively, the bounds hold because $\widehat{A}_e$, $\widehat{B}_e$ are stale versions of $A^{\iota(e)-1}$, $B^{\iota(e)-1}$, which are monotonically non-decreasing and non-increasing sets. Full details of proof are given in Appendix A.

Lemma 4.1. *In CF-2g, for any $e \in V$, $\widehat{A}_e \subseteq A^{\iota(e)-1}$, and $\widehat{B}_e \supseteq B^{\iota(e)-1}$.*

Corollary 4.2. *Submodularity of F implies for CF-2g $\Delta_+(e) \leq \Delta_+^{\max}(e)$, and $\Delta_-(e) \leq \Delta_-^{\max}(e)$.*

The error in CF-2g depends on the tightness of the bounds in Cor. 4.2. We analyze this in Sec. 6.1.

5 Concurrency Control for the Double Greedy Algorithm

The concurrency control-based double greedy algorithm[1], CC-2g, is presented in Alg. 5, and closely follows the meta-algorithm of Alg. 1 and Alg. 2. Unlike in CF-2g, the concurrency control mechanisms of CC-2g ensure that concurrent transactions are serialized when they are not independent.

Serializability is achieved by maintaining sets $\widehat{A}$, $\widetilde{A}$, $\widehat{B}$, $\widetilde{B}$, which serve as upper *and* lower bounds on the true state of A and B at commit time. Each thread can determine locally if a decision to include or exclude an element can be taken safely. Otherwise, the proposal is deferred to the commit process (Alg. 8) which waits until it is certain about A and B before proceeding.

The commit order is given by $\iota(e)$, which is the value of ι at line 2 of Alg. 5. We define $A^{\iota(e)-1}$, $B^{\iota(e)-1}$ as before with CF-2g. Additionally, let $\widehat{A}_e$, $\widehat{B}_e$, $\widetilde{A}_e$, and $\widetilde{B}_e$ be the sets that are returned by Alg. 6.[2] Indeed, these sets are guaranteed to be bounds on $A^{\iota(e)-1}$, $B^{\iota(e)-1}$:

Lemma 5.1. *In CC-2g, $\forall e \in V$, $\widehat{A}_e \subseteq A^{\iota(e)-1} \subseteq \widetilde{A}_e \backslash e$, and $\widehat{B}_e \supseteq B^{\iota(e)-1} \supseteq \widetilde{B}_e \cup e$.*

Intuitively, these bounds are maintained by recording potential effects of concurrent transactions in $\widetilde{A}$, $\widetilde{B}$, and only recording the actual effects in $\widehat{A}$, $\widehat{B}$; we leave the full proof to Appendix A. Furthermore, by committing transactions in order ι, we have $\widehat{A} = A^{\iota(e)-1}$ and $\widehat{B} = B^{\iota(e)-1}$ during commit.

Lemma 5.2. *In CC-2g, when committing element e, we have $\widehat{A} = A^{\iota(e)-1}$ and $\widehat{B} = B^{\iota(e)-1}$.*

[2] For clarity, we present the algorithm as creating a copy of $\widehat{A}$, $\widehat{B}$, $\widetilde{A}$, and $\widetilde{B}$ for each element. In practice, it is more efficient to update and access them in shared memory. Nevertheless, our theorems hold for both settings.

Corollary 5.3. *Submodularity of F implies that the Δ's computed by CC-2g satisfy $\Delta_+^{\min}(e) \leq \Delta_+^{exact}(e) = \Delta_+(e) \leq \Delta_+^{\max}(e)$ and $\Delta_-^{\min}(e) \leq \Delta_-^{exact}(e) = \Delta_-(e) \leq \Delta_-^{\max}(e)$.*

By using these bounds, CC-2g can determine when it is safe to construct the transaction locally. For failed transactions, the server is able to construct the correct transaction using the true program state. As a consequence we can guarantee that the parallel execution of CC-2g is serializable.

6 Analysis of Algorithms

Our two algorithms trade off performance and strong approximation guarantees. The CF-2g algorithm emphasizes speed at the expense of the approximation objective. On the other hand, CC-2g emphasizes the tight $1/2$-approximation at the expense of increased coordination. In this section we characterize the reduction in the approximation objective as well as the increased coordination. Our analysis connects the he degradation in CC-2g scalability with the degradation in the CF-2g approximation factor via the maximum inter-processor message delay τ.

6.1 Approximation of CF-2g double greedy

Theorem 6.1. *Let F be a non-negative submodular function. CF-2g solves the unconstrained problem $\max_{A \subset V} F(A)$ with worst-case approximation factor $E[F(A_{CF})] \geq \frac{1}{2}F^* - \frac{1}{4}\sum_{i=1}^{N} E[\rho_i]$, where A_{CF} is the output of the algorithm, F^* is the optimal value, and $\rho_i = \max\{\Delta_+^{\max}(e) - \Delta_+(e), \Delta_-^{\max}(e) - \Delta_-(e)\}$ is the maximum discrepancy in the marginal gain due to the bounds.*

The proof (Appendix C) of Thm. 6.1 follows the structure in [2]. Thm. 6.1 captures the deviation from optimality as a function of width of the bounds which we characterize for two common applications.

Example: max graph cut. For the max cut objective we bound the expected discrepancy in the marginal gain ρ_i in terms of the sparsity of the graph and the maximum inter-processor message delay τ. By applying Thm. 6.1 we obtain the approximation factor $E[F(A^N)] \geq \frac{1}{2}F^* - \tau\frac{\#\text{edges}}{2N}$ which decreases linearly in both the message delays and graph density. In a complete graph, $F^* = \frac{1}{2}\#\text{edges}$, so $E[F(A^N)] \geq F^*\left(\frac{1}{2} - \frac{\tau}{N}\right)$, which makes it possible to scale τ linearly with N while retaining the same approximation factor.

Example: set cover. Consider the simple set cover function, $F(A) = \sum_{l=1}^{L} \min(1, |A \cap S_l|) - \lambda|A| = |\{l : A \cap S_l \neq \emptyset\}| - \lambda|A|$, with $0 < \lambda \leq 1$. We assume that there is some bounded delay τ. Suppose also the S_l's form a partition, so each element e belongs to exactly one set. Then, $\sum_e E[\rho_e] \geq \tau + L(1 - \lambda^\tau)$, which is linear in τ but independent of N.

6.2 Correctness of CC-2g

Theorem 6.2. *CC-2g is serializable and therefore solves the unconstrained submodular maximization problem $\max_{A \subset V} F(A)$ with approximation $E[F(A_{CC})] \geq \frac{1}{2}F^*$, where A_{CC} is the output of the algorithm, and F^* is the optimal value.*

The key challenge in the proof (Appendix B) of Thm. 6.2 is to demonstrate that CC-2g guarantees a serializable execution. It suffices to show that CC-2g takes the same decision as Ser-2g for each element – locally if it is safe to do so, and otherwise deferring the computation to the server. As an immediate consequence of serializability, we recover the optimal approximation guarantees of the serial Ser-2g algorithm.

6.3 Scalability of CC-2g

Whenever a transaction is reconstructed on the server, the server needs to wait for all earlier elements to be committed, and is also blocked from committing all later elements. Each failed transaction effectively constitutes a barrier to the parallel processing. Hence, the scalability of CC-2g is dependent on the number of failed transactions.

We can directly bound the number of failed transactions (details in Appendix D) for both the max-cut and set cover example problems. For the max-cut problem with a maximum inter-processor message

delay τ we obtain the upper bound $2\tau\frac{\#edges}{N}$. Similarly for set cover the expected *number* of failed transactions is upper-bounded by 2τ. As a consequence, the coordination costs of CC-2g grows at the same rate as the reduction in accuracy of CF-2g. Moreover, the CC-2g algorithm will slow down in settings where the CF-2g algorithm produces sub-optimal solutions.

7 Evaluation

We implemented the parallel and serial double greedy algorithms in Java / Scala. Experiments were conducted on Amazon EC2 using one cc2.8xlarge machine, up to 16 threads, for 10 repetitions. We measured the runtime and speedup (ratio of runtime on 1 thread to runtime on p threads). For CF-2g, we measured $F(A_{CF}) - F(A_{Ser})$, the difference between the objective value on the sets returned by CF-2g and Ser-2g. We verified the correctness of CC-2g by comparing the output of CC-2g with Ser-2g. We also measured the fraction of transactions that fail in CC-2g. Our parallel algorithms were tested on the max graph cut and set cover problems with two synthetic graphs and three real datasets (Table 1). We found that vertices were typically indexed such that nearby vertices in the graph were also close in their indices. To reduce this dependency, we randomly permuted the ordering of vertices.

Graph	# vertices	# edges	Description
Erdos-Renyi	20,000,000	$\approx 2 \times 10^9$	Each edge is included with probability 5×10^{-6}.
ZigZag	25,000,000	2,025,000,000	Expander graph. The 81-regular zig-zag product between the Cayley graph on $\mathbb{Z}_{2500000}$ with generating set $\{\pm 1, \ldots, \pm 5\}$, and the complete graph K_{10}.
Friendster	10,000,000	625,279,786	Subgraph of social network. [21]
Arabic-2005	22,744,080	631,153,669	2005 crawl of Arabic web sites [22, 23, 24].
UK-2005	39,459,925	921,345,078	2005 crawl of the .uk domain [22, 23, 24].
IT-2004	41,291,594	1,135,718,909	2004 crawl of the .it domain [22, 23, 24].

Table 1: Synthetic and real graphs used in the evaluation of our parallel algorithms.

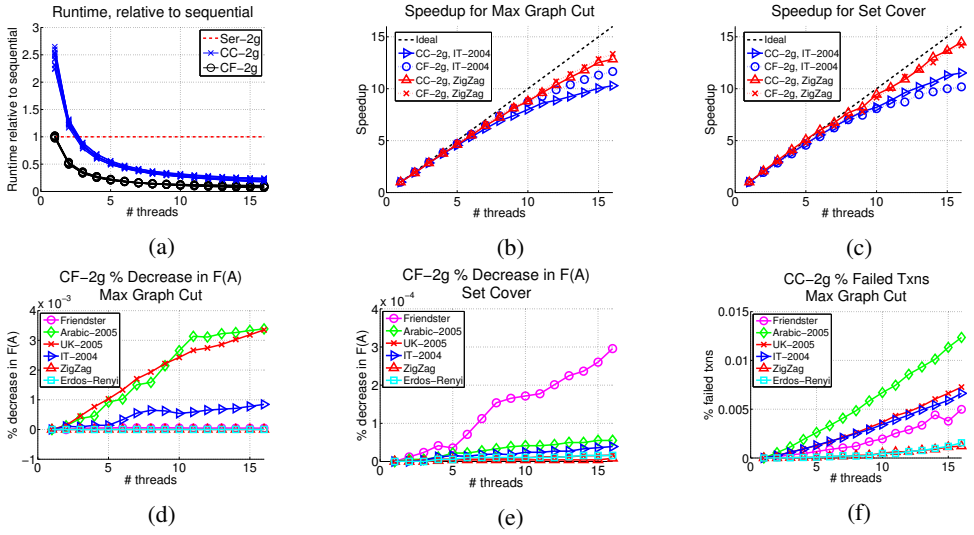
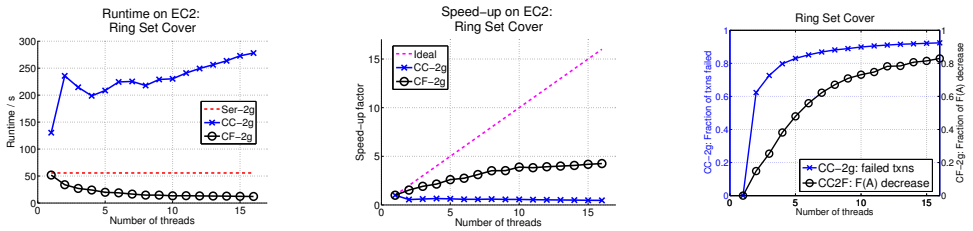


Figure 3: Experimental results. Fig. 3a – runtime of the parallel algorithms as a ratio to that of the serial algorithm. Each curve shows the runtime of a parallel algorithm on a particular graph for a particular function F. Fig. 3b, 3c – speedup (ratio of runtime on one thread to that on p threads). Fig. 3d, 3e – % difference between objective values of Ser-2g and CF-2g, i.e. $[F(A_{CF})/F(A_{Ser}) - 1] \times 100\%$. Fig. 3f – percentage of transactions that fail in CC-2g on the max graph cut problem.

We summarize of the key results here with more detailed experiments and discussion in Appendix G. **Runtime, Speedup:** Both parallel algorithms are faster than the serial algorithm with three or more threads, and show good speedup properties as more threads are added ($\sim$ 10x or more for all graphs and both functions). **Objective value:** The objective value of CF-2g decreases with the number of threads, but differs from the serial objective value by less than 0.01%. **Failed transactions:** CC-2g fails more transactions as threads are added, but even with 16 threads, less than 0.015% transactions fail, which has negligible effect on the runtime / speedup.



(a) (b) (c)

Figure 4: Experimental results for set cover problem on a ring expander graph demonstrating that for adversarially constructed inputs we can reduce the optimality of CF-2g and increase coordination costs for CC-2g.

7.1 Adversarial ordering

To highlight the differences in approaches between the two parallel algorithms, we conducted experiments on a ring Cayley expander graph on $\mathbb{Z}_{10^6}$ with generating set $\{\pm 1, \ldots, \pm 1000\}$. The algorithms are presented with an adversarial ordering, without permutation, so vertices close in the ordering are adjacent to one another, and tend to be processed concurrently. This causes CF-2g to make more mistakes, and CC-2g to fail more transactions. While more sophisticated partitioning schemes could improve scalability and eliminate the effect of adversarial ordering, we use the default data partitioning in our experiments to highlight the differences between the two algorithms. As Fig. 4 shows, CC-2g sacrifices speed to ensure a serializable execution, eventually failing on $> 90\%$ of transactions. On the other hand, CF-2g focuses on speed, resulting in faster runtime, but achieves an objective value that is 20% of $F(A_{Ser})$. We emphasize that we contrived this example to highlight differences between CC-2g and CF-2g, and we do not expect to see such orderings in practice.

8 Related Work

Similar approach: Coordination-free solutions have been proposed for stochastic gradient descent [25] and collapsed Gibbs sampling [26]. More generally, parameter servers [27, 28] apply the CF approach to larger classes of problems. Pan et al. [29] applied concurrency control to parallelize some unsupervised learning algorithms. **Similar problem:** Distributed and parallel greedy submodular maximization is addressed in [1, 15, 16], but only for monotone functions.

9 Conclusion and Future Work

By adopting the transaction processing model from parallel database systems, we presented two approaches to parallelizing the double greedy algorithm for unconstrained submodular maximization. We quantified the weaker approximation guarantee of CF-2g and the additional coordination of CC-2g, allowing one to trade off between performance and objective optimality. Our evaluation on large scale data demonstrates the scalability and tradeoffs of the two approaches. Moreover, as the approximation quality of the CF-2g algorithm decreases so does the scalability of the CC-2g algorithm. The choice between the algorithm then reduces to a choice of guaranteed performance and guaranteed optimality.

We believe there are a number of areas for future work. One can imagine a system that allows a smooth interpolation between CF-2g and CC-2g. While both CF-2g and CC-2g can be immediately implemented as distributed algorithms, higher communication costs and delays may pose additional challenges. Finally, other problems such as constrained maximization of monotone / non-monotone functions could potentially be parallelized with the CF and CC frameworks.

Acknowledgments

This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adobe, Apple, Inc., Bosch, C3Energy, Cisco, Cloudera, EMC, Ericsson, Facebook, GameOnTalis, Guavus, HP, Huawei, Intel, Microsoft, NetApp, Pivotal, Splunk, Virdata, VMware, and Yahoo!. This research was in part funded by the Office of Naval Research under contract/grant number N00014-11-1-0688. X. Pan's work is also supported by a DSO National Laboratories Postgraduate Scholarship.

References

[1] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems 26*. 2013.

[2] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, 2012.

[3] A. Krause and C. Guestrin. Submodularity and its applications in optimized information gathering: An introduction. *ACM Transactions on Intelligent Systems and Technology*, 2(4), 2011.

[4] G. Kim, E. P. Xing, F. Li, and T. Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *Int. Conference on Computer Vision (ICCV)*, 2011.

[5] J. Gillenwater, A. Kulesza, and B. Taskar. Near-optimal MAP inference for determinantal point processes. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[6] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

[7] H. Lin and J. Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[8] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

[9] L. S. Shapley. Cores of convex games. *International Journal of Game Theory*, 1(1):11–26, 1971.

[10] A. Frank. Submodular functions in graph theory. *Discrete Mathematics*, 111:231–243, 1993.

[11] A. Schrijver. *Combinatorial Optimization – Polyhedra and efficiency*. Springer, 2002.

[12] A. Krause and S. Jegelka. Submodularity in machine learning – new directions. ICML Tutorial, 2013.

[13] J. Bilmes. Deep mathematical properties of submodularity with applications to machine learning. NIPS Tutorial, 2013.

[14] A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, 2014.

[15] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. In *SPAA*, 2013.

[16] K. Wei, R. Iyer, and J. Bilmes. Fast multi-stage submodular maximization. In *Int. Conference on Machine Learning (ICML)*, 2014.

[17] C. Reed and Z. Ghahramani. Scaling the Indian Buffet Process via submodular maximization. In *Int. Conference on Machine Learning (ICML)*, 2013.

[18] A. Krause and E. Horvitz. A utility-theoretic approach to privacy in online services. *JAIR*, 39, 2010.

[19] M. Tamer Ozsu. *Principles of Distributed Database Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007. ISBN 9780130412126.

[20] H. Kung and J.T. Robinson. On optimistic methods for concurrency control. *TODS*, 6(2), 1981.

[21] J. Leskovec. Stanford network analysis project, 2011. URL `http://snap.stanford.edu/`.

[22] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, 2004.

[23] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*. ACM Press, 2011.

[24] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[25] B. Recht, C. Re, S.J. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS) 24*, Granada, 2011.

[26] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A.J. Smola. Scalable inference in latent variable models. In *Proc. of the 5th ACM International Conference on Web Search and Data Mining*, 2012.

[27] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *Big Learn workshop, at NIPS*, Lake Tahoe, 2013.

[28] Q. Ho, J. Cipar, H. Cui, S. Lee, J.K. Kim, P.B. Gibbons, G.A. Gibson, G. Ganger, and E. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*. 2013.

[29] X. Pan, J.E. Gonzalez, S. Jegelka, T. Broderick, and M.I. Jordan. Optimistic concurrency control for distributed unsupervised learning. In *Advances in Neural Information Processing Systems 26*. 2013.

A Proofs of $\widetilde{A}_e$, $\widehat{A}_e$, $\widetilde{B}_e$, $\widehat{B}_e$ as bounds on $A^{\iota(e)-1}$ and $B^{\iota(e)-1}$

Lemma 4.1. *In CF-2g, for any $e \in V$, $\widehat{A}_e \subseteq A^{\iota(e)-1}$, and $\widehat{B}_e \supseteq B^{\iota(e)-1}$.*

Proof. For any element e, we write T_e to denote the time at which Alg. 4 line 8 is executed. Consider any element $e' \in V$. If $e' \in \widehat{A}_e$, it must be the case that the algorithm set $\widehat{A}(e')$ to 1 (line 10) before T_e, which implies $\iota(e') < \iota(e)$, and hence $e' \in A^{\iota(e)-1}$. So $\widehat{A}_e \subseteq A^{\iota(e)-1}$.

Similarly, if $e' \notin \widehat{B}_e$, then the algorithm set $\widehat{B}(e')$ to 0 (line 11) before T_e, so $\iota(e') < \iota(e)$. Also, $e' \notin A$ because the execution of line 11 excludes the execution of line 10. Therefore, $e' \notin A^{\iota(e)-1}$, and $e' \notin B^{\iota(e)-1}$. So $\widehat{B}_e \supseteq B^{\iota(e)-1}$. □

Lemma 5.1. *In CC-2g, $\forall e \in V$, $\widehat{A}_e \subseteq A^{\iota(e)-1} \subseteq \widetilde{A}_e \backslash e$, and $\widehat{B}_e \supseteq B^{\iota(e)-1} \supseteq \widetilde{B}_e \cup e$.*

Proof. Clearly, $e \in \widetilde{B}_e \cup e$ but $e \notin \widetilde{A}_e \backslash e$. By definition, $e \in B^{\iota(e)-1}$ but $e \notin A^{\iota(e)-1}$. CC-2g only modifies $\widehat{A}(e)$ and $\widehat{B}(e)$ when committing the transaction on e, which occurs after obtaining the bounds in getGuarantee(e), so $e \in \widehat{B}_e$ but $e \notin \widehat{A}_e$.

Consider any $e' \neq e$. Suppose $e' \in \widehat{A}_e$. This is only possible if we have committed the transaction on e' before the call getGuarantee(e), so it must be the case that $\iota(e') < \iota(e)$. Thus, $e' \in A^{\iota(e)-1}$.

Now suppose $e' \in A^{\iota(e)-1}$. By definition, this implies $\iota(e') < \iota(e)$ and $e' \in A$. Hence, it must be the case that we have already set $\widetilde{A}(e') \leftarrow 1$ (by the ordering imposed by ι on Line 2), but never execute $\widetilde{A}(e') \leftarrow 0$ (since $e' \in A$), so $e' \in \widetilde{A}_e$.

An analogous argument shows $e' \notin \widehat{B}_e \implies e' \notin B^{\iota(e)-1} \implies e' \notin \widetilde{B}_e \cup e$. □

Lemma 5.2. *In CC-2g, when committing element e, we have $\widehat{A} = A^{\iota(e)-1}$ and $\widehat{B} = B^{\iota(e)-1}$.*

Proof. Alg. 8 Line 1 ensures that all elements ordered before e are committed, and that no element ordered after e are committed. This suffices to guarantee that $e' \in \widehat{A} \iff e' \in A^{\iota(e)-1}$ and $e' \in \widehat{B} \iff e' \in B^{\iota(e)-1}$. □

B Proof of serial equivalence of CC-2g

Theorem 6.2. *CC-2g is serializable and therefore solves the unconstrained submodular maximization problem $\max_{A \subset V} F(A)$ with approximation $E[F(A_{CC})] \geq \frac{1}{2}F^*$, where A_{CC} is the output of the algorithm, and F^* is the optimal value.*

Proof. We will denote by A^i_{seq}, B^i_{seq} the sets generated by Ser-2g, reserving A^i, B^i for sets generated by the CC-2g algorithm. It suffices to show by induction that $A^i_{seq} = A^i$ and $B^i_{seq} = B^i$. For the base case, $A^0 = \emptyset = A^0_{seq}$, and $B^0 = V = B^0_{seq}$. Consider any element e. The CC-2g algorithm includes $e \in A$ iff $u_e < [\Delta^{\min}_+(e)]_+([\Delta^{\min}_+(e)]_+ + [\Delta^{\max}_-(e)]_+)^{-1}$ on Alg. 5 Line 6 or $u_e < [\Delta^{\text{exact}}_+(e)]_+([\Delta^{\text{exact}}_+(e)]_+ + [\Delta^{\text{exact}}_-(e)]_+)^{-1}$ on Alg. 8 Line 5. In both cases, Corollary 5.3 implies $u_e < [\Delta_+(e)]_+([\Delta_+(e)]_+ + [\Delta_-(e)]_+)^{-1}$. By induction, $A^{\iota(e)-1} = A^{\iota(e)-1}_{seq}$ and $B^{\iota(e)-1} = B^{\iota(e)-1}_{seq}$, so the threshold is exactly that computed by Ser-2g. Hence, the CC-2g algorithm includes $e \in A$ iff Ser-2g includes $e \in A$. (An analogous argument works for the case where e is excluded from B.) □

C Proof of bound for CF-2g

We follow the proof outline of [2].

Consider an ordering ι inducted by running CF-2g. For convenience, we will use i to flexibly denote the element e and its ordering $\iota(e)$.

Let OPT be an optimal solution to the problem. Define $O^i := (OPT \cup A^i) \cap B^i$. Note that O^i coincides with A^i and B^i on elements $1, \ldots, i$, and O^i coincides with OPT on elements $i+1, \ldots, n$. Hence,

$$O^i \backslash (i+1) \supseteq A^i$$
$$O^i \cup (i+1) \subseteq B^i.$$

Lemma C.1. *For every* $1 \leq i \leq n$, $\Delta_+(i) + \Delta_-(i) \geq 0$.

Proof. This is just Lemma II.1 of [2]. □

Lemma C.2. *Let* $\rho_i = \max\{\Delta_+^{\max}(e) - \Delta_+(e), \Delta_-^{\max}(e) - \Delta_-(e)\}$. *For every* $1 \leq i \leq n$,

$$E[F(O^{i-1}) - F(O^i)] \leq \frac{1}{2} E[F(A^i) - F(A^{i-1}) + F(B^i) - F(B^{i-1}) + \rho_i].$$

Proof. We follow the proof outline of [2]. First, note that it suffices to prove the inequality conditioned on knowing A^{i-1}, $\widehat{A}_i$ and $\widehat{B}_i$, then applying the law of total expectation. Under this conditioning, we also know B^{i-1}, O^{i-1}, $\Delta_+(i)$, $\Delta_+^{\max}(i)$, $\Delta_-(i)$, and $\Delta_-^{\max}(i)$.

We consider the following 6 cases.

Case 1: $0 < \Delta_+(i) \leq \Delta_+^{\max}(i)$, $0 \leq \Delta_-^{\max}(i)$. Since both $\Delta_+^{\max}(i) > 0$ and $\Delta_-^{\max}(i) > 0$, the probability of including i is just $\Delta_+^{\max}(i)/(\Delta_+^{\max}(i) + \Delta_-^{\max}(i))$, and the probability of excluding i is $\Delta_-^{\max}(i)/(\Delta_+^{\max}(i) + \Delta_-^{\max}(i))$.

$$\begin{aligned}
E[F(A^i) - F(A^{i-1}) | A^{i-1}, \widehat{A}_i, \widehat{B}_i] &= \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} (F(A^{i-1} \cup i) - F(A^{i-1})) \\
&= \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} \Delta_+(i) \\
&\geq \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} (\Delta_+^{\max}(i) - \rho_i) \\
E[F(B^i) - F(B^{i-1}) | A^{i-1}, \widehat{A}_i, \widehat{B}_i] &= \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} (F(B^{i-1} \backslash i) - F(B^{i-1})) \\
&= \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} \Delta_-(i) \\
&\geq \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} (\Delta_-^{\max}(i) - \rho_i)
\end{aligned}$$

$$
\begin{aligned}
&E[F(O^{i-1}) - F(O^i)|A^{i-1}, \widehat{A}_i, \widehat{B}_i] \\
=&\frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)}(F(O^{i-1}) - F(O^{i-1} \cup i)) \\
&+ \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)}(F(O^{i-1}) - F(O^{i-1} \backslash i)) \\
=&\begin{cases} \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}(F(O^{i-1}) - F(O^{i-1} \cup i)) & \text{if } i \notin OPT \\ \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}(F(O^{i-1}) - F(O^{i-1} \backslash i)) & \text{if } i \in OPT \end{cases} \\
\leq&\begin{cases} \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}(F(B^{i-1} \backslash i) - F(B^{i-1})) & \text{if } i \notin OPT \\ \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}(F(A^{i-1} \cup i) - F(A^{i-1})) & \text{if } i \in OPT \end{cases} \\
=&\begin{cases} \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}\Delta_-(i) & \text{if } i \notin OPT \\ \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}\Delta_+(i) & \text{if } i \in OPT \end{cases} \\
\leq&\begin{cases} \frac{\Delta_+^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}\Delta_-^{\max}(i) & \text{if } i \notin OPT \\ \frac{\Delta_-^{\max}(i)}{\Delta_+^{\max}(i)+\Delta_-^{\max}(i)}\Delta_+^{\max}(i) & \text{if } i \in OPT \end{cases} \\
=&\frac{\Delta_+^{\max}(i)\Delta_-^{\max}(i)}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)}
\end{aligned}
$$

where the first inequality is due to submodularity: $O^{i-1} \backslash i \supseteq A^{i-1}$ and $O^{i-1} \cup i \subseteq B^{i-1}$.

Putting the above inequalities together:

$$
\begin{aligned}
&E\left[F(O^{i-1}) - F(O^i) - \frac{1}{2}\left(F(A^i) - F(A^{i-1}) + F(B^i) - F(B^{i-1}) + \rho_i\right)\middle| A^{i-1}, \widehat{A}_i, \widehat{B}_i\right] \\
&\leq \frac{1/2}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)}\Big[2\Delta_+^{\max}(i)\Delta_-^{\max}(i) - \Delta_-^{\max}(i)(\Delta_-^{\max}(i) - \rho_i) \\
&\qquad\qquad - \Delta_+^{\max}(i)(\Delta_+^{\max}(i) - \rho_i)\Big] - \frac{1}{2}\rho_i \\
&= \frac{1/2}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)}\Big[-(\Delta_+^{\max}(i) - \Delta_-^{\max}(i))^2 + \rho_i(\Delta_+^{\max}(i) + \Delta_-^{\max}(i))\Big] - \frac{1}{2}\rho_i \\
&\leq \frac{\frac{1}{2}\rho_i(\Delta_+^{\max}(i) + \Delta_-^{\max}(i))}{\Delta_+^{\max}(i) + \Delta_-^{\max}(i)} - \frac{1}{2}\rho_i \\
&= 0.
\end{aligned}
$$

Case 2: $0 < \Delta_+(i) \leq \Delta_+^{\max}(i)$, $\Delta_-^{\max}(i) < 0$. In this case, the algorithm always choses to include i, so $A^i = A^{i-1} \cup i$, $B^i = B^{i-1}$ and $O^i = O^{i-1} \cup i$:

$$
\begin{aligned}
E[F(A^i) - F(A^{i-1})|A^{i-1}, \widehat{A}_i, \widehat{B}_i] &= F(A^{i-1} \cup i) - F(A^{i-1}) = \Delta_+(i) > 0 \\
E[F(B^i) - F(B^{i-1})|A^{i-1}, \widehat{A}_i, \widehat{B}_i] &= F(B^{i-1}) - F(B^{i-1}) = 0 \\
E[F(O^{i-1}) - F(O^i)|A^{i-1}, \widehat{A}_i, \widehat{B}_i] &= F(O^{i-1}) - F(O^{i-1} \cup i) \\
&\leq \begin{cases} 0 & \text{if } i \in OPT \\ F(B^{i-1} \backslash i) - F(B^{i-1}) & \text{if } i \notin OPT \end{cases} \\
&= \begin{cases} 0 & \text{if } i \in OPT \\ \Delta_-(i) & \text{if } i \notin OPT \end{cases} \\
&\leq 0 \\
&< \frac{1}{2}E[F(A^i) - F(A^{i-1}) + F(B^i) - F(B^{i-1}) + \rho_i|A^{i-1}, \widehat{A}_i, \widehat{B}_i]
\end{aligned}
$$

where the first inequality is due to submodularity: $O^{i-1} \cup i \subseteq B^{i-1}$.

Case 3: $\Delta_+(i) \leq 0 < \Delta_+^{\max}(i)$, $0 < \Delta_-(i) < \Delta_-^{\max}(i)$. Analogous to Case 1.

Case 4: $\Delta_+(i) \leq 0 < \Delta_+^{\max}(i)$, $\Delta_-(i) \leq 0$. This is not possible, by Lemma C.1.

Case 5: $\Delta_+(i) \leq \Delta_+^{\max}(i) \leq 0$, $0 < \Delta_-(i) \leq \Delta_-^{\max}(i)$. Analogous to Case 2.

Case 6: $\Delta_+(i) \leq \Delta_+^{\max}(i) \leq 0$, $\Delta_-(i) \leq 0$. This is not possible, by Lemma C.1.

□

We will now prove the main theorem.

Theorem 6.1. *Let F be a non-negative submodular function. CF-2g solves the unconstrained problem $\max_{A \subset V} F(A)$ with worst-case approximation factor $E[F(A_{CF})] \geq \frac{1}{2}F^* - \frac{1}{4}\sum_{i=1}^{N} E[\rho_i]$, where A_{CF} is the output of the algorithm, F^* is the optimal value, and $\rho_i = \max\{\Delta_+^{\max}(e) - \Delta_+(e), \Delta_-^{\max}(e) - \Delta_-(e)\}$ is the maximum discrepancy in the marginal gain due to the bounds.*

Proof. Summing up the statement of Lemma C.2 for all i gives us a telescoping sum, which reduces to:

$$\begin{aligned} E[F(O^0) - F(O^n)] &\leq \frac{1}{2}E[F(A^n) - F(A^0) + F(B^n) - F(B^0)] + \frac{1}{2}\sum_{i=1}^{n} E[\rho_i] \\ &\leq \frac{1}{2}E[F(A^n) + F(B^n)] + \frac{1}{2}\sum_{i=1}^{n} E[\rho_i]. \end{aligned}$$

Note that $O^0 = OPT$ and $O^n = A^n = B^n$, so $E[F(A^n)] \geq \frac{1}{2}F^* - \frac{1}{4}\sum_i E[\rho_i]$. □

C.1 Example: max graph cut

Let $C_i = (A^{i-1} \backslash \widehat{A}_i) \cup (\widehat{B}_i \backslash B^{i-1})$ be the set of elements concurrently processed with i but ordered after i, and $D_i = B^i \backslash A^i$ be the set of elements ordered after i. Denote $\bar{A}_i = V \backslash (\widehat{A}_i \cup C_i \cup D_i) = \{1, \ldots, i\} \backslash \widehat{A}_i$ be the elements up to i that are not included in $\widehat{A}_i$. Let $w_i(S) = \sum_{j \in S, (i,j) \in E} w(i,j)$. For the max graph cut function, it is easy to see that

$$\begin{aligned} \Delta_+ &\geq -w_i(\widehat{A}_i) - w_i(C_i) + w_i(D_i) + w_i(\bar{A}_i) \\ \Delta_+^{\max} &= -w_i(\widehat{A}_i) + w_i(C_i) + w_i(D_i) + w_i(\bar{A}_i) \\ \Delta_- &\geq +w_i(\widehat{A}_i) - w_i(C_i) + w_i(D_i) - w_i(\bar{A}_i) \\ \Delta_-^{\max} &= +w_i(\widehat{A}_i) + w_i(C_i) + w_i(D_i) - w_i(\bar{A}_i) \end{aligned}$$

Thus, we can see that $\rho_i \leq 2w_i(C_i)$.

Suppose we have bounded delay τ, so $|C_i| \leq \tau$. Then $w_i(C_i)$ has a hypergeometric distribution with mean $\frac{\deg(i)}{N}\tau$, and $E[\rho_i] \leq 2\tau\frac{\deg(i)}{N}$. The approximation of the hogwild algorithm is then $E[F(A^n)] \geq \frac{1}{2}F^* - \tau\frac{\#\text{edges}}{2N}$. In sparse graphs, the hogwild algorithm is off by a small additional term, which albeit grows linearly in τ. In a complete graph, $F^* = \frac{1}{2}\#\text{edges}$, so $E[F(A^n)] \geq F^*\left(\frac{1}{2} - \frac{\tau}{N}\right)$, which makes it possible to scale τ linearly with N while retaining the same approximation factor.

C.2 Example: set cover

Consider the simple set cover function, for $\lambda < L/N$:

$$F(A) = \sum_{l=1}^{L} \min(1, |A \cap S_l|) - \lambda|A| = |\{l : A \cap S_l \neq \emptyset\}| - \lambda|A|.$$

We assume that there is some bounded delay τ.

Suppose also that the sets S_l form a partition, so each element e belongs to exactly one set. Let $n_l = |S_l|$ denote the size of S_l. Given any ordering π, let e_l^t be the tth element of S_l in the ordering, i.e. $|\{e' : \pi(e') \leq \pi(e_l^t) \wedge e' \in S_l\}| = t$.

For any $e \in S_l$, we get

$$\begin{aligned}
\Delta_+(e) &= -\lambda + 1\{A^{\iota(e)-1} \cap S_l = \emptyset\} \\
\Delta_+^{\max}(e) &= -\lambda + 1\{\widehat{A}_e \cap S_l = \emptyset\} \\
\Delta_-(e) &= +\lambda - 1\{B^{\iota(e)-1} \backslash e \cap S_l = \emptyset\} \\
\Delta_-^{\max}(e) &= +\lambda - 1\{\widehat{B}_e \backslash e \cap S_l = \emptyset\}
\end{aligned}$$

Let η be the position of the first element of S_l to be accepted, i.e. $\eta = \min\{t : e_l^t \in A \cap S_l\}$. (For convenience, we set $\eta = n_l$ if $A \cap S_l = \emptyset$.) We first show that η is independent of π: for $\eta < n_l$,

$$\begin{aligned}
P(\eta|\pi) &= \frac{\Delta_+^{\max}(e_l^\eta)}{\Delta_+^{\max}(e_l^\eta) + \Delta_-^{\max}(e_l^\eta)} \prod_{t=1}^{\eta-1} \frac{\Delta_-^{\max}(e_l^t)}{\Delta_+^{\max}(e_l^t) + \Delta_-^{\max}(e_l^t)} \\
&= \frac{1-\lambda}{1-\lambda+\lambda} \prod_{t=1}^{\eta-1} \frac{\lambda}{1-\lambda+\lambda} \\
&= (1-\lambda)\lambda^{\eta-1},
\end{aligned}$$

and $P(\eta = n_l|\pi) = \lambda^{\eta-1}$.

Note that, $\Delta_-^{\max}(e) - \Delta_-(e) = 1$ iff $e = e_l^{n_l}$ is the last element of S_l in the ordering, there are no elements accepted up to $\widehat{B}_{e_l^{n_l}} \backslash e_l^{n_l}$, and there is some element e' in $\widehat{B}_{e_l^{n_l}} \backslash e_l^{n_l}$ that is rejected and not in $B^{\iota(e_l^{n_l})-1}$. Denote by $m_l \leq \min(\tau, n_l - 1)$ the number of elements before $e_l^{n_l}$ that are inconsistent between $\widehat{B}_{e_l^{n_l}}$ and $B^{\iota(e_l^{n_l})-1}$. Then $\mathbb{E}[\Delta_-^{\max}(e_l^{n_l}) - \Delta_-(e_l^{n_l})] = P(\Delta_-^{\max}(e_l^{n_l}) \neq \Delta_-(e_l^{n_l}))$ is

$$\lambda^{n_l-1-m_l}(1-\lambda^{m_l}) \quad = \quad \lambda^{n_l-1}(\lambda^{-m_l}-1) \quad \leq \quad \lambda^{n_l-1}(\lambda^{-\min(\tau,n_l-1)}-1) \quad \leq \quad 1-\lambda^\tau.$$

If $\lambda = 1$, $\Delta_+^{\max}(e) \leq 0$, so no elements before $e_l^{n_l}$ will be accepted, and $\Delta_-^{\max}(e_l^{n_l}) = \Delta_-(e_l^{n_l})$.

On the other hand, $\Delta_+^{\max}(e) - \Delta_+(e) = 1$ iff $(A^{\iota(e)-1} \backslash \widehat{A}_e) \cap S_l \neq \emptyset$, that is, if an element has been accepted in A but not yet observed in $\widehat{A}_e$. Since we assume a bounded delay, only the first τ elements after the first acceptance of an $e \in S_l$ may be affected.

$$\begin{aligned}
&\mathbb{E}\left[\sum_{e \in S_l} \Delta_+^{\max}(e) - \Delta_+(e)\right] \\
&= \mathbb{E}[\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \widehat{A}_e\}] \\
&= \mathbb{E}[\mathbb{E}[\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \widehat{A}_e\} \mid \eta = t, \pi(e_l^t) = k]] \\
&= \sum_{t=1}^{n_l} \sum_{k=t}^{N-n+t} P(\eta = t, \pi(e_l^t) = k)\mathbb{E}[\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \widehat{A}_e\} \mid \eta = t, \pi(e_l^t) = k] \\
&= \sum_{t=1}^{n_l} P(\eta = t) \sum_{k=t}^{N-n+t} P(\pi(e_l^t) = k)\mathbb{E}[\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \widehat{A}_e\} \mid \eta = t, \pi(e_l^t) = k].
\end{aligned}$$

Under the assumption that every ordering π is equally likely, and a bounded delay τ, conditioned on $\eta = t, \pi(e_l^t) = k$, the random variable $\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \widehat{A}_e\}$ has hypergeometric distribution with mean $\frac{n_l - t}{N-k}\tau$. Also, $P(\pi(e_l^t) = k) = \frac{n_l}{N}\binom{n-1}{t-1}\binom{N-n}{k-t}/\binom{N-1}{k-1}$, so

the above expression becomes

$$
\begin{aligned}
&\mathbb{E}\left[\sum_{e\in S_l} \Delta_+^{\max}(e) - \Delta_+(e)\right] \\
&= \sum_{t=1}^{n_l} P(\eta = t) \sum_{k=t}^{N-n+t} \frac{n_l}{N} \frac{\binom{n-1}{t-1}\binom{N-n}{k-t}}{\binom{N-1}{k-1}} \frac{n-t}{N-k}\tau \\
&= \frac{n_l}{N}\tau \sum_{t=1}^{n_l} P(\eta = t) \sum_{k=t}^{N-n+t} \frac{\binom{k-1}{t-1}\binom{N-k}{n-t}}{\binom{N-1}{n-1}} \frac{n-t}{N-k} && \text{(symmetry of hypergeometric)} \\
&= \frac{n_l}{N}\tau \sum_{t=1}^{n_l} \frac{P(\eta = t)}{\binom{N-1}{n-1}} \sum_{k=t}^{N-n+t} \binom{k-1}{t-1}\binom{N-k-1}{n-t-1} \\
&= \frac{n_l}{N}\tau \sum_{t=1}^{n_l} \frac{P(\eta = t)}{\binom{N-1}{n-1}} \binom{N-1}{n-1} && \text{(Lemma E.1, } a = N-2,\ b = n_l - 2,\ j = 1\text{)} \\
&= \frac{n_l}{N}\tau \sum_{t=1}^{n_l} P(\eta = t) \\
&= \frac{n_l}{N}\tau.
\end{aligned}
$$

Since $\Delta_+^{\max}(e) \geq \Delta_+(e)$ and $\Delta_-^{\max}(e) \geq \Delta_-^{\max}(e)$, we have that $\rho_e \leq \Delta_+^{\max}(e) - \Delta_+(e) + \Delta_-^{\max}(e) - \Delta_-(e)$, so

$$
\begin{aligned}
\mathbb{E}\left[\sum_e \rho_e\right] &= \mathbb{E}\left[\sum_e \Delta_+^{\max}(e) - \Delta_+(e) + \Delta_-^{\max}(e) - \Delta_-(e)\right] \\
&= \sum_l \mathbb{E}\left[\sum_{e\in S_l} \Delta_+^{\max}(e) - \Delta_+(e)\right] + \mathbb{E}\left[\sum_{e\in S_l} \Delta_-^{\max}(e) - \Delta_-(e)\right] \\
&\leq \tau \frac{\sum_l n_l}{N} + L(1-\lambda^\tau) \\
&= \tau + L(1-\lambda^\tau).
\end{aligned}
$$

Note that $\mathbb{E}\left[\sum_e \rho_e\right]$ does not depend on N and is linear in τ. Also, if $\tau = 0$ in the sequential case, we get $\mathbb{E}\left[\sum_e \rho_e\right] \leq 0$.

D Upper bound on expected number of failed transactions

Let N be the number of elements, i.e. the cardinality of the ground set. Let $C_i = (A^{i-1} \backslash \widehat{A}_i) \cup (\widehat{B}_i \backslash B^{i-1})$. We assume a bounded delay τ, so that $|C_i| \leq \tau$ for all i.

We call element i *dependent* on i' if $\exists A, F(A \cup i) - F(A) \neq F(A \cup i' \cup i) - F(A \cup i')$ or $\exists B, F(B \backslash i) - F(B) \neq F(B \cup i' \backslash i) - F(B \cup i')$, i.e. the result of the processing i' will affect the computation of Δ's for i. For example, for the graph cut problem, every vertex is dependent on its neighbors; for the separable sums problem, i is dependent on $\{i' : \exists S_l, i \in S_l, i' \in S_l\}$.

Let n_i be the number of elements that i is dependent on. Now, we note that if C_i does not contain any elements on which i is dependent, then $\Delta_+^{\max}(i) = \Delta_+(i) = \Delta_+^{\min}(i)$ and $\Delta_-^{\max}(i) = \Delta_-(i) = \Delta_-^{\min}(i)$, so i will not fail. Conversely, if i fails, there must be some element $i' \in C_i$ such that i is dependent on i'.

$$
\begin{aligned}
E(\text{number of failed transactions}) &= \sum_i P(i \text{ fails}) \\
&\leq \sum_i P(\exists i' \in C_i, i \text{ depends on } i') \\
&\leq \sum_i E\left[\sum_{i' \in C_i} 1\{i \text{ depends on } i'\}\right] \\
&\leq \sum_i \frac{\tau n_i}{N}
\end{aligned}
$$

The last inequality follows from the fact that $\sum_{i' \in C_i} 1\{i \text{ depends on } i'\}$ is a hypergeometric random variable and $|C_i| \leq \tau$.

Note that the bound established above is generic to functions F, and additional knowledge of F can lead to better analyses on the algorithm's concurrency.

D.1 Upper bound for max graph cut

By applying the above generic bound, we see that the number of failed transactions for max graph cut is upper bounded by $\frac{\tau}{N} \sum_i n_i = \tau \frac{2\#\text{edges}}{N}$.

D.2 Upper bound for set cover

For the set cover problem, we can provide a tighter bound on the number of failed items. We make the same assumptions as before in the CF-2g analysis, i.e. the sets S_l form a partition of V, there is a bounded delay τ.

Observe that for any $e \in S_l$, $\Delta_-^{\min}(e) \neq \Delta_-^{\max}(e)$ if $\widehat{B}_e \backslash e \cap S_l \neq \emptyset$ and $\widetilde{B}_e \backslash e \cap S_l = \emptyset$. This is only possible if $e_l^{n_l} \notin \widetilde{B}_e$ and $\widetilde{B}_e \supset \widehat{A}_e \cap S_l = \emptyset$, that is $\pi(e) \geq \pi(e_l^{n_l}) - \tau$ and $\forall e' \in S_l, (\pi(e') < \pi(e_l^{n_l}) - \tau) \implies (e' \notin A)$. The latter condition is achieved with probability $\lambda^{n_l - m_l}$, where

$m_l = \#\{e' : \pi(e') \geq \pi(e_l^{n_l}) - \tau\}$. Thus,

$$\begin{aligned}
\mathbb{E}\left[\#\{e : \Delta_-^{\min}(e) \neq \Delta_-^{\max}(e)\}\right] &= \mathbb{E}[m_l\, 1(\forall e' \in S_l, (\pi(e') < \pi(e_l^{n_l}) - \tau) \implies (e' \notin A))] \\
&= \mathbb{E}[\mathbb{E}[m_l\, 1(\forall e' \in S_l, (\pi(e') < \pi(e_l^{n_l}) - \tau) \implies (e' \notin A))|u_{1:N}]] \\
&= \mathbb{E}[m_l\, \mathbb{E}[1(\forall e' \in S_l, (\pi(e') < \pi(e_l^{n_l}) - \tau) \implies (e' \notin A))|u_{1:N}]] \\
&= \mathbb{E}[m_l \lambda^{n_l - m_l}] \\
&\leq \lambda^{(n_l-\tau)_+} \mathbb{E}[m_l] \\
&= \lambda^{(n_l-\tau)_+} \mathbb{E}[\mathbb{E}[m_l | \pi(e_l^{n_l}) = k]] \\
&= \lambda^{(n_l-\tau)_+} \sum_{k=n_l}^{N} P(\pi(e_l^{n_l}) = k) \mathbb{E}[m_l | \pi(e_l^{n_l}) = k]].
\end{aligned}$$

Conditioned on $\pi(e_l^{n_l}) = k$, m_l is a hypergeometric random variable with mean $\frac{n_l - 1}{k-1}\tau$. Also $P(\pi(e_l^{n_l}) = k) = \frac{n_l}{N}\binom{n_l-1}{0}\binom{N-n_l}{N-k}/\binom{N-1}{N-k}$. The above expression is therefore

$$\begin{aligned}
&\mathbb{E}\left[\#\{e : \Delta_-^{\min}(e) \neq \Delta_-^{\max}(e)\}\right] \\
&= \lambda^{(n_l-\tau)_+} \sum_{k=n_l}^{N} \frac{n_l}{N} \frac{\binom{n_l-1}{0}\binom{N-n_l}{N-k}}{\binom{N-1}{N-k}} \frac{n_l - 1}{k-1}\tau \\
&= \lambda^{(n_l-\tau)_+} \frac{n_l}{N}\tau \sum_{k=n_l}^{N} \frac{\binom{N-k}{0}\binom{k-1}{n_l-1}}{\binom{N-1}{n_l-1}} \frac{n_l - 1}{k-1} && \text{(symmetry of hypergeometric)} \\
&= \lambda^{(n_l-\tau)_+} \frac{n_l}{N} \frac{\tau}{\binom{N-1}{n_l-1}} \sum_{k=n_l}^{N} \binom{N-k}{0}\binom{k-2}{n_l-2} \\
&= \lambda^{(n_l-\tau)_+} \frac{n_l}{N} \frac{\tau}{\binom{N-1}{n_l-1}} \binom{N-1}{n_l-1} && \text{(Lemma E.1, } a = N-2, b = n_l - 2, j = 2, t = n_l) \\
&= \lambda^{(n_l-\tau)_+} \frac{n_l}{N}\tau.
\end{aligned}$$

Now we consider any element $e \in S_l$ with $\pi(e) < \pi(e_l^{n_l}) - \tau$ that fails. (Note that $e_l^{n_l} \in \widehat{B}_e$ and $\widetilde{B}_e$, so $\Delta_-^{\min}(e) = \Delta_-^{\max}(e) = \lambda$.) It must be the case that $\widehat{A}_e \cap S_l = \emptyset$, for otherwise $\Delta_+^{\min}(e) = \Delta_+^{\max}(e) = -\lambda$ and it does not fail. This implies that $\Delta_+^{\max}(e) = 1 - \lambda \geq u_i$. At commit, if $A^{\iota(e)-1} \cap S_l = \emptyset$, we accept e into A. Otherwise, $A^{\iota(e)-1} \cap S_l \neq \emptyset$, which implies that some other element $e' \in S_l$ has been accepted. Thus, we conclude that every element $e \in S_l$ that fails must be within τ of the first accepted element $e_l^{\eta}\ inS_l$. The expected number of such elements is exactly as we computed in the CF-2ganalysis: $\frac{n_l}{N}\tau$.

Hence, the expected number of elements that fails is upper bounded as

$$\begin{aligned}
\mathbb{E}[\#\text{failed transactions}] &\leq \sum_l (1 + \lambda^{(n_l-\tau)_+}) \frac{n_l}{N}\tau \\
&\leq \sum_l 2\frac{n_l}{N}\tau \\
&= 2\tau.
\end{aligned}$$

E Lemma

Lemma E.1. $\sum_{k=t}^{a-b+t} \binom{k-j}{t-j}\binom{a-k+j}{b-t+j} = \binom{a+1}{b+1}$.

Proof.

$$
\begin{aligned}
&\sum_{k=t}^{a-b+t} \binom{k-j}{t-j}\binom{a-k+j}{b-t+j} \\
&= \sum_{k'=0}^{a-b} \binom{k'+t-j}{t-j}\binom{a-k'-t+j}{b-t+j} \\
&= \sum_{k'=0}^{a-b} \binom{k'+t-j}{k'}\binom{a-k'-t+j}{a-b-k'} && \text{(symmetry of binomial coeff.)} \\
&= (-1)^{a-b} \sum_{k'=0}^{a-b} \binom{-t+j-1}{k'}\binom{-b+t-j-1}{a-b-k'} && \text{(upper negation)} \\
&= (-1)^{a-b} \binom{-b-2}{a-b} && \text{(Chu-Vandermonde's identity)} \\
&= \binom{a+1}{a-b} && \text{(upper negation)} \\
&= \binom{a+1}{b+1} && \text{(symmetry of binomial coeff.)}
\end{aligned}
$$

□

F Parallel algorithms for separable sums

For some functions F, we can maintain sketches / statistics to aid the computation of $\Delta_+^{\max}$, $\Delta_-^{\max}$, $\Delta_+^{\min}$, $\Delta_-^{\min}$. In particular, we consider functions of the form $F(X) = \sum_{l=1}^{L} g\left(\sum_{i \in X \cup S_l} w_l(i)\right) - \lambda \sum_{i \in X} v(i)$, where $S_l \subseteq V$ are (possibly overlapping) groups of elements in the ground set, g is a non-decreasing concave scalar function, and $w_l(i)$ and $v(i)$ are non-negative scalar weights. An example of such functions is set cover $F(A) = \sum_{l=1}^{L} \min(1, |A \cup S_l|) - \lambda |A|$. It is easy to see that $F(X \cup e) - F(X) = \sum_{l: e \in S_l} \left[g\left(w_l(e) + \sum_{i \in X \cup S_l} w_l(i)\right) - g\left(\sum_{i \in X \cup S_l} w_l(i)\right)\right] - \lambda v(e)$. Define

$$\widehat{\alpha}_l = \sum_{j \in \widehat{A} \cup S_l} w_l(j), \qquad \widehat{\alpha}_{l,e} = \sum_{j \in \widehat{A}_e \cup S_l} w_l(j), \qquad \alpha_l^{\iota(e)-1} = \sum_{j \in A^{\iota(e)-1} \cup S_l} w_l(j).$$

$$\widehat{\beta}_l = \sum_{j \in \widehat{B} \cup S_l} w_l(j), \qquad \widehat{\beta}_{l,e} = \sum_{j \in \widehat{B}_e \cup S_l} w_l(j), \qquad \beta_l^{\iota(e)-1} = \sum_{j \in B^{\iota(e)-1} \cup S_l} w_l(j).$$

F.1 CF-2g for separable sums F

Algorithm 9 updates $\widehat{\alpha}_l$ and $\widehat{\beta}_l$, and computes $\Delta_+^{\max}(e)$ and $\Delta_-^{\max}(e)$ using $\widehat{\alpha}_{l,e}$ and $\widehat{\beta}_{l,e}$. Following arguments analogous to that of Lemma 4.1, we can show:

Lemma F.1. *For each l and $e \in V$, $\widehat{\alpha}_{l,e} \leq \alpha_l^{\iota(e)-1}$ and $\widehat{\beta}_{l,e} \geq \beta_l^{\iota(e)-1}$.*

Corollary F.2. *Concavity of g implies that Δ's computed by Algorithm 9 satisfy*

$$\Delta_+^{\max}(e) \quad \geq \quad \sum_{S_l \ni e} \left[g(\alpha_l^{\iota(e)-1} + w_l(e)) - g(\alpha_l^{\iota(e)-1})\right] - \lambda v(e) \quad = \quad \Delta_+(e),$$

$$\Delta_-^{\max}(e) \quad \geq \quad \sum_{S_l \ni e} \left[g(\beta_l^{\iota(e)-1} - w_l(e)) - g(\beta_l^{\iota(e)-1})\right] + \lambda v(e) \quad = \quad \Delta_-(e),$$

The analysis of Section 6.1 follows immediately from the above.

Algorithm 9: CF-2g for separable sums

```
for e ∈ V do Â(e) = 0

for l = 1, ..., L do α̂_l = 0, β̂_l = Σ_{e∈S_l} w_l(e)

for p ∈ {1, ..., P} do in parallel
    while ∃ element to process do
        e = next element to process
        Δ+^max(e) = −λv(e) + Σ_{S_l ∋ e} g(α̂_l + w_l(e)) − g(α̂_l)
        Δ−^max(e) = +λv(e) + Σ_{S_l ∋ e} g(β̂_l − w_l(e)) − g(β̂_l)
        Draw u_e ~ Unif(0, 1)
        if u_e < [Δ+^max(e)]_+ / ([Δ+^min(e)]_+ + [Δ−^max(e)]_+) then
            Â(e) ← 1
            for l : e ∈ S_l do
                α̂_l ← α̂_l + w_l(e)
        else
            for l : e ∈ S_l do
                β̂_l ← β̂_l − w_l(e)
```

F.2 CC-2g for separable sums F

Analogous to the CF-2g algorithm, we maintain $\widehat{\alpha}_l$, $\widehat{\beta}_l$ and additionally $\widetilde{\alpha}_l = \sum_{j \in \widetilde{A} \cup S_l} w_l(j)$ and $\widetilde{\beta}_l = \sum_{j \in \widetilde{B} \cup S_l} w_l(j)$. Following the arguments of Lemma 5.1 and Corollary 5.3, we can show the following.

Lemma F.3. $\widehat{\alpha}_{l,e} \leq \alpha^{\iota(e)-1} \leq \widetilde{\alpha}_{l,e} - w_l(e)$ *and* $\widehat{\beta}_{l,e} \geq \beta^{\iota(e)-1} \geq \widetilde{\beta}_{l,e} + w_l(e)$

Corollary F.4. *Concavity of g implies that the Δ's computed by Algorithm 10 satisfy:*

$$\begin{aligned}
\Delta_+^{\max}(e) &= -\lambda v(e) + \sum_{S_l \ni e} [g(\widehat{\alpha}_{l,e} + w_l(e)) - g(\widehat{\alpha}_{l,e})] \\
&\geq -\lambda v(e) + \sum_{S_l \ni e} \left[g(\widehat{\alpha}_l^{\iota(e)-1} + w_l(e)) - g(\widehat{\alpha}_l^{\iota(e)-1})\right] && = \Delta_+(e) \\
&\geq -\lambda v(e) + \sum_{S_l \ni e} [g(\widetilde{\alpha}_{l,e}) - g(\widetilde{\alpha}_{l,e} - w_l(e))] && = \Delta_+^{\min}(e), \\
\Delta_-^{\max}(e) &= \lambda v(e) + \sum_{S_l \ni e} \left[g(\widehat{\beta}_{l,e} - w_l(e)) - g(\widehat{\beta}_{l,e})\right] \\
&\geq \lambda v(e) + \sum_{S_l \ni e} \left[g(\widehat{\beta}_l^{\iota(e)-1} - w_l(e)) - g(\widehat{\beta}_l^{\iota(e)-1})\right] && = \Delta_-(e) \\
&\geq \lambda v(e) + \sum_{S_l \ni e} \left[g(\widetilde{\beta}_l^{\iota(e)-1}) - g(\widetilde{\beta}_l^{\iota(e)-1} + w_l(e))\right] && = \Delta_-^{\min}(e).
\end{aligned}$$

The analysis of Section 6.3 and 6.2 follows immediately from the above.

Algorithm 10: CC-2g for separable sums

for $e \in V$ **do** $\widehat{A}(e) = \widetilde{A}(e) = 0$, $\widehat{B}(e) = \widetilde{B}(e) = 1$

for $l = 1, \ldots, L$ **do**
  $\widehat{\alpha}_l = \widetilde{\alpha}_l = 0$
  $\widehat{\beta}_l = \widetilde{\beta}_l = \sum_{e \in S_l} w_l(e)$
for $i = 1, \ldots, |V|$ **do** processed$(i) = false$

$\iota = 0$
for $p \in \{1, \ldots, P\}$ **do in parallel**
  while $\exists$ *element to process* **do**
    $e =$ next element to process
    $(\widehat{\alpha}_{\cdot,e}, \widetilde{\alpha}_{\cdot,e}, \widehat{\beta}_{\cdot,e}, \widetilde{\beta}_{\cdot,e}) =$ getGuarantee(e)
    (result, u_e) = propose$(e, \widehat{\alpha}_{\cdot,e}, \widetilde{\alpha}_{\cdot,e}, \widehat{\beta}_{\cdot,e}, \widetilde{\beta}_{\cdot,e})$
    commit$(e, i, u_e,$ result$)$

Algorithm 11: CC-2g getGuarantee(e) for separable sums

$\widetilde{A}(e) \leftarrow 1$; $\widetilde{B}(e) \leftarrow 0$
for $l : e \in S_l$ **do**
  $\widetilde{\alpha}_l \leftarrow \widetilde{\alpha}_l + w_l(e)$
  $\widetilde{\beta}_l \leftarrow \widetilde{\beta}_l - w_l(e)$
$i = \iota$; $\iota \leftarrow \iota + 1$
$\widehat{\alpha}_{\cdot,e} = \widehat{\alpha}_{\cdot}$; $\widehat{\beta}_{\cdot,e} = \widehat{\beta}_{\cdot}$
$\widetilde{\alpha}_{\cdot,e} = \widetilde{\alpha}_{\cdot}$; $\widetilde{\beta}_{\cdot,e} = \widetilde{\beta}_{\cdot}$
return $(\widehat{\alpha}_{\cdot,e}, \widetilde{\alpha}_{\cdot,e}, \widehat{\beta}_{\cdot,e}, \widetilde{\beta}_{\cdot,e})$

Algorithm 12: CC-2g propose$(e, \widehat{\alpha}_{\cdot,e}, \widetilde{\alpha}_{\cdot,e}, \widehat{\beta}_{\cdot,e}, \widetilde{\beta}_{\cdot,e})$ for separable sums

$\Delta_+^{\min}(e) = -\lambda v(e) + \sum_{S_l \ni e} g(\widetilde{\alpha}_l) - g(\widetilde{\alpha}_l - w_l(e))$
$\Delta_+^{\max}(e) = -\lambda v(e) + \sum_{S_l \ni e} g(\widehat{\alpha}_l + w_l(e)) - g(\widehat{\alpha}_l)$
$\Delta_-^{\min}(e) = +\lambda v(e) + \sum_{S_l \ni e} g(\widetilde{\beta}_l) - g(\widetilde{\beta}_l + w_l(e))$
$\Delta_-^{\max}(e) = +\lambda v(e) + \sum_{S_l \ni e} g(\widehat{\beta}_l - w_l(e)) - g(\widehat{\beta}_l)$
Draw $u_e \sim Unif(0,1)$
if $u_e < \frac{[\Delta_+^{\min}(e)]_+}{[\Delta_+^{\min}(e)]_+ + [\Delta_-^{\max}(e)]_+}$ **then** result $\leftarrow 1$

else if $u_e > \frac{[\Delta_+^{\max}(e)]_+}{[\Delta_+^{\max}(e)]_+ + [\Delta_-^{\min}(e)]_+}$ **then** result $\leftarrow -1$

else result $\leftarrow$ FAIL

return (result, u_e)

Algorithm 13: CC-2g commit$(e, i, u_e$,result$)$ for separable sums

wait until $\forall j < i$, processed$(j) = true$
if *result* $=$ *FAIL* **then**
  $\Delta_+^{\text{exact}}(e) = -\lambda v(e) + \sum_{S_l \ni e} g(\widehat{\alpha}_l + w_l(e)) - g(\widehat{\alpha}_l)$
  $\Delta_-^{\text{exact}}(e) = +\lambda v(e) + \sum_{S_l \ni e} g(\widehat{\beta}_l - w_l(e)) - g(\widehat{\beta}_l)$
  if $u_e < \frac{[\Delta_+^{exact}(e)]_+}{[\Delta_+^{exact}(e)]_+ + [\Delta_-^{exact}(e)]_+}$ **then** result$\leftarrow 1$

  else result$\leftarrow -1$

if *result*$= 1$ **then**
  $\widehat{A}(e) \leftarrow 1$
  $\widetilde{B}(e) \leftarrow 1$
  for $l : e \in S_l$ **do**
    $\widehat{\alpha}_l \leftarrow \widehat{\alpha}_l + w_l(e)$
    $\widetilde{\beta}_l \leftarrow \widetilde{\beta}_l + w_l(e)$
else
  $\widetilde{A}(e) \leftarrow 0$; $\widehat{B}(e) \leftarrow 0$
  for $l : e \in S_l$ **do**
    $\widetilde{\alpha}_l \leftarrow \widetilde{\alpha}_l - w_l(e)$
    $\widehat{\beta}_l \leftarrow \widehat{\beta}_l - w_l(e)$
processed$(i) = true$

G Full experiment results

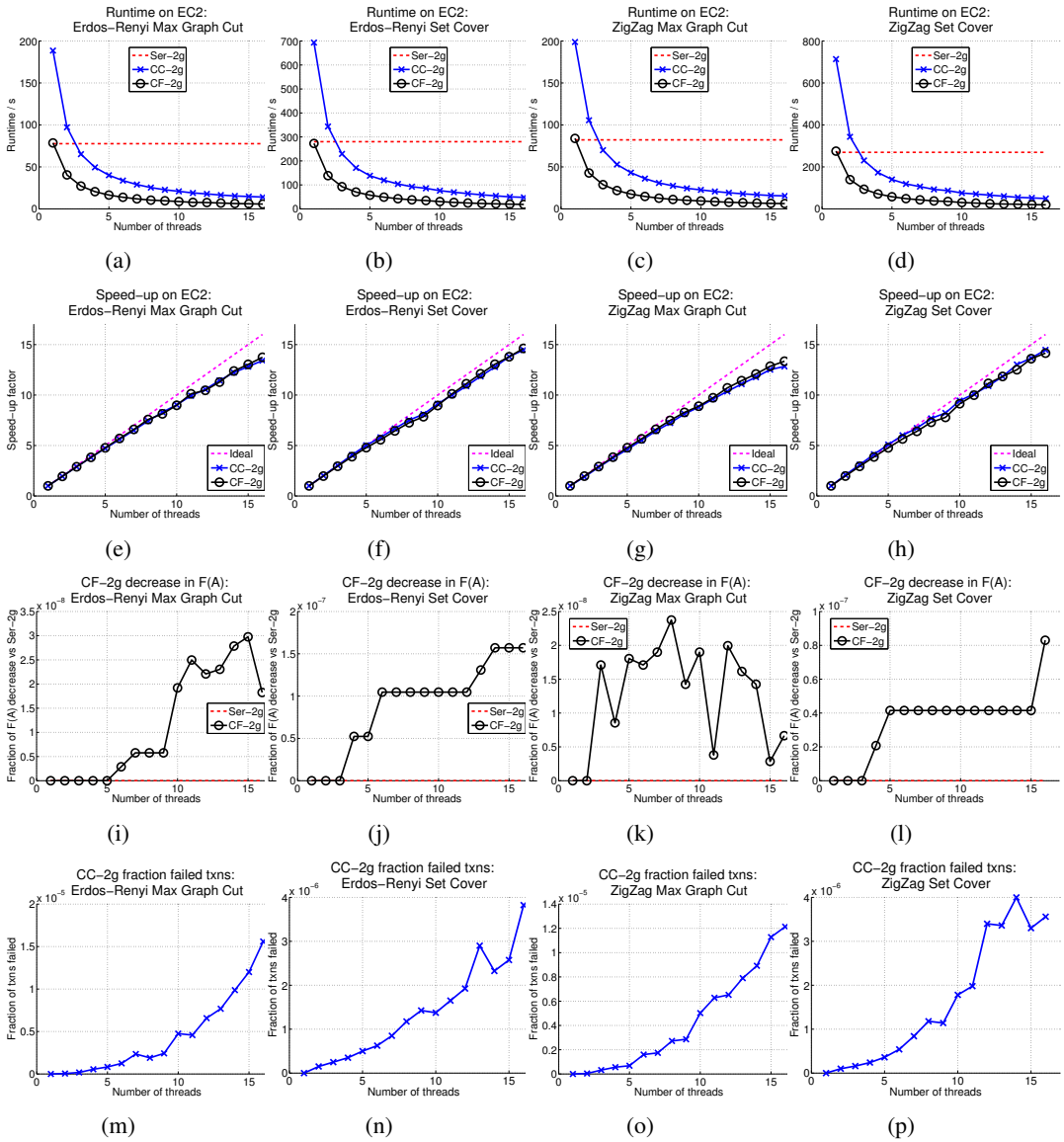


Figure 5: Experimental results on Erdos-Renyi and ZigZag synthetic graphs.

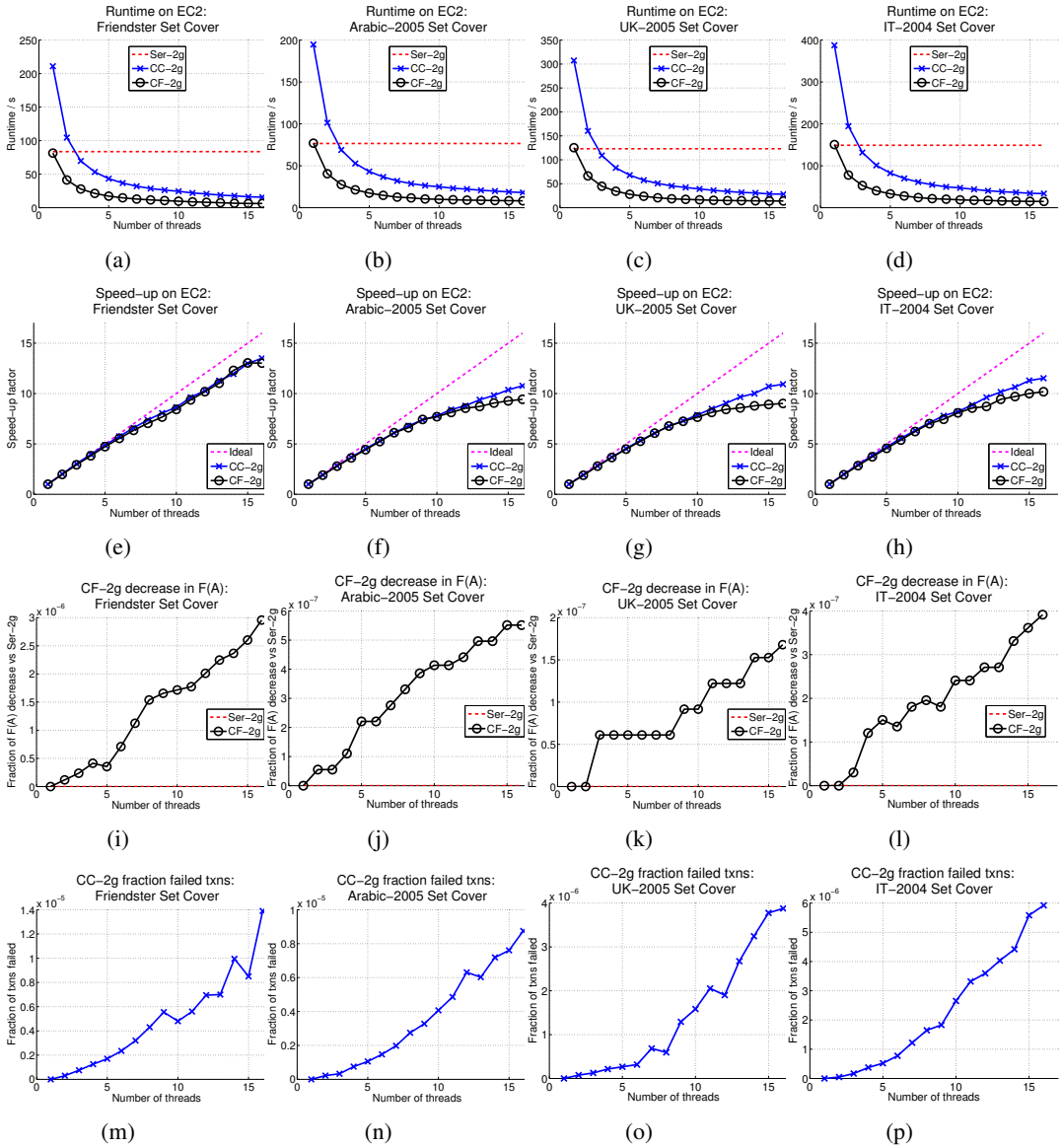


Figure 6: Set cover on 4 real graphs.

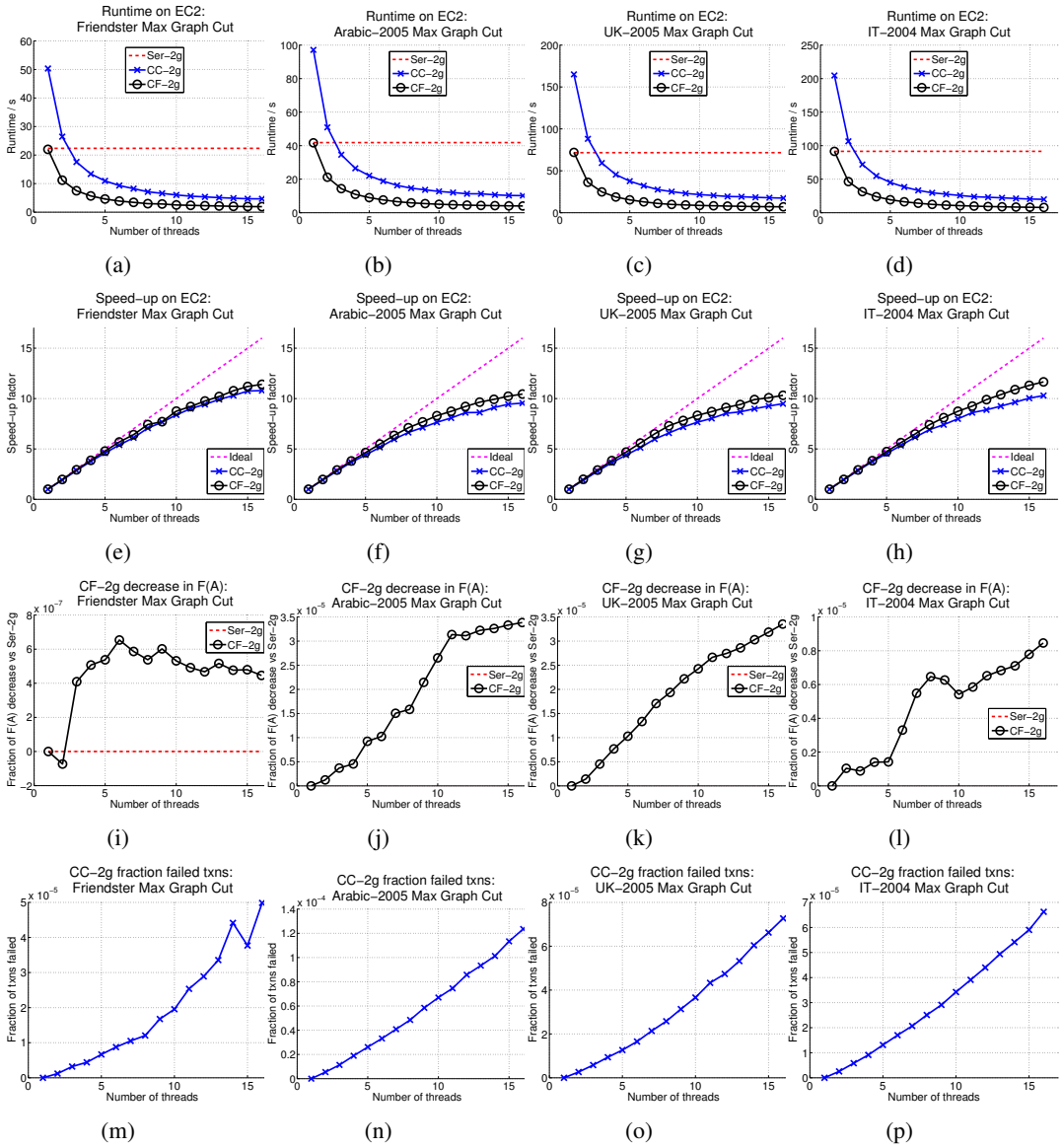


Figure 7: Max graph cut on 4 real graphs.

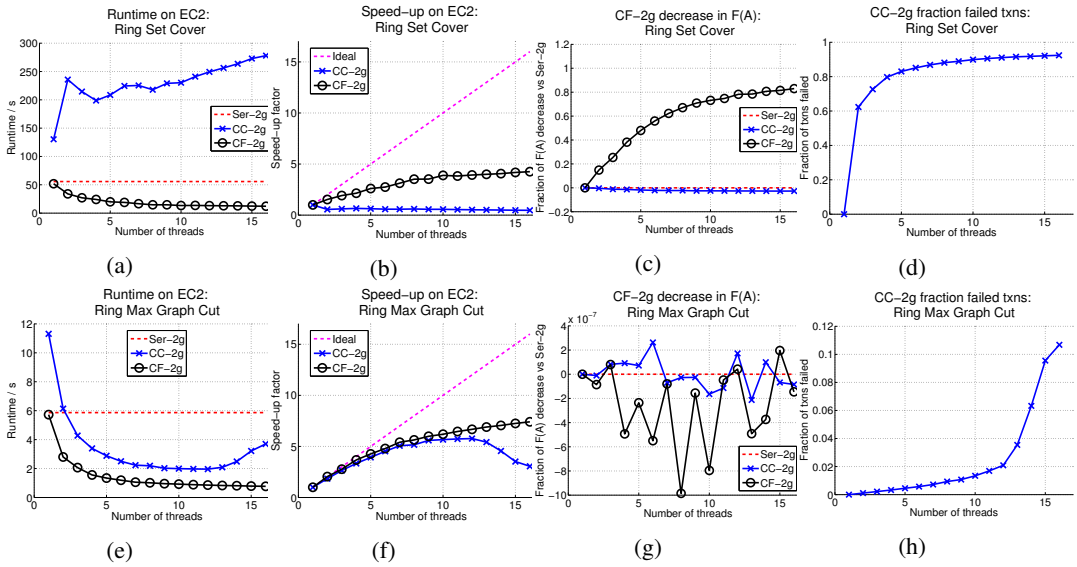


Figure 8: Experimental results for ring graph on set cover problem.

H Illustrative examples

The following examples illustrate how (i) the simple (uni-directional) greedy algorithm may fail for non-monotone submodular functions, and (ii) where the coordination-free double greedy algorithm can run into trouble.

H.1 Greedy and non-monotone functions

For illustration, consider the following toy example of a non-monotone submodular function. We are given a ground set $V = \{v_0, v_1, v_2, \ldots, v_k\}$ of $k+1$ elements, and a universe $U = \{u_1, \ldots, u_k\}$. Each element v_i in V covers elements $\mathrm{Cov}(v_i) \subseteq U$ of the universe. In addition, each element in V has a cost $c(v_i)$. We are aiming to maximize the submodular function

$$F(S) = \Big| \bigcup_{v \in S} \mathrm{Cov}(v) \Big| - \sum_{v \in S} c(v). \tag{3}$$

Let the costs and coverings be as follows:

$$\mathrm{Cov}(v_0) = U \qquad c(v_0) = k - 1 \tag{4}$$

$$\mathrm{Cov}(v_i) = u_i \qquad c(v_i) = \epsilon < 1/k^2 \quad \text{for all } i > 0. \tag{5}$$

Then the optimal solution is $S^* = V \setminus v_0$ with $F(S^*) = k - k\epsilon$.

The greedy algorithm of Nemhauser et al. [8] always adds the element with the largest marginal gain. Since $F(v_0) = 1$ and $F(v_i) = 1 - \epsilon$ for all $i > 0$, the algorithm would pick v_0 first. After that, any additional element only has a negative marginal gain, $F(\{v_0, v_i\}) - F(v_0) = -\epsilon$. Hence, the algorithm would end up with a solution $F(v_0) = 1$ or worse, which means an approximation factor of only approximately $1/k$.

For the double greedy algorithm, the scenario would be the following. If v_0 happens to be the first element, then it is picked with probability

$$P(v_0) = \frac{[F(v_0) - F(\emptyset)]_+}{[F(v_0) - F(\emptyset)]_+ + [F(V \setminus v_0) - F(V)]_-} = \frac{1}{1 + (k-1)} = \frac{1}{k}. \tag{6}$$

If v_0 is selected, nothing else will be added afterwards, since $[F(v_0, v_i) - F(v_0)]_+ = 0$. If it does not pick v_0, then any other element is added with a probability of

$$P(v_i \mid \neg v_0) = \frac{[F(v_i) - F(\emptyset)]_+}{[F(v_i) - F(\emptyset)]_+ + F(V \setminus \{v_0, v_i\}) - F(V \setminus v_0)]_-} = \frac{1-\epsilon}{1-\epsilon} = 1. \tag{7}$$

If v_0 is not the first element, then any element before v_0 is added with probability $p(v_i) = 1 - \epsilon$, and as soon as an element v_i has been picked, v_0 will not be added any more. Hence, with high probability, this algorithm returns the optimal solution. The deterministic version surely does.

H.2 Coordination vs no coordination

The following example illustrates the differences between coordination and no coordination. In this example, let V be split into m disjoint groups G_j of equal size $k = |V|/m$, and let

$$F(S) = \sum_{j=1}^{m} \min\{1, |S \cap G_j|\} - \frac{|S \cap G_j|}{k}. \tag{8}$$

A maximizing set S^* contains one element from each group, and $F(S^*) = m - m/k$.

If the sequential double greedy algorithm has not picked an element from a group, it will retain the next element from that group with probability

$$\frac{1 - 1/k}{1 - 1/k + 1/k} = 1 - 1/k. \tag{9}$$

Once it has sampled an element from a group G_j, it does not pick any more elements from G_j, and therefore $|S \cap G_j| \leq 1$ for all j and the set S returned by the algorithm. The probability that S

does not contain any element from G_j is k^{-k} —fairly low. Hence, with probability $1 - m/k^k$ the algorithm returns the optimal solution.

Without coordination, the outcome heavily depends on the order of the elements. For simplicity, assume that k is a multiple of the number q of processors (or q is a multiple of k). In the worst case, the elements are sorted by their groups and the members of each group are processed in parallel. With q processors working in parallel, the first q elements from a group G (up to shifts) will be processed with a bound $\widehat{A}$ that does not contain any element from G, and will each be selected with probability $1 - 1/k$. Hence, in expectation, $|S \cap G_j| = \min\{q, k\}(1 - 1/k)$ for all j.

If $q > k$, then in expectation $k - 1$ elements from each group are selected, which corresponds to an approximation factor of

$$\frac{m(1 - \frac{k-1}{k})}{m(1 - 1/k)} = \frac{1}{k-1}. \tag{10}$$

If $k > q$, then in expectation we obtain an approximation factor of

$$\frac{m(1 - \frac{q(1-1/k)}{k})}{m(1 - 1/k)} = 1 - \frac{q}{k} + \frac{1}{k-1} \tag{11}$$

which decreases linearly in q. If $q = k$, then the factor is $1/(q-1)$ instead of $1/2$.