Parallel Double Greedy Submodular Maximization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Many machine learning problems can be reduced to the maximization of a submodular function. While this insight can gain theoretical guarantees, exploiting these results in a large scale parallel setting remains an open research topic, in particular for *non-monotone* functions for which recently proposed scalable greedy algorithms do not apply. Recently, Buchbinder et al. [1] achieved a tight 1/2-approximation for unconstrained non-monotone submodular maximization using a double greedy algorithm. This algorithm was developed and analyzed in the serial setting, limiting our ability to leverage parallel hardware. In this work, we bridge this gap and make the theoretical benefits available in a scalable setting. We propose and analyze two parallel double greedy algorithms. The first, *coordination-free* approach emphasizes speed at the cost of a weaker approximation guarantee. The second, *concurrency control* approach guarantees the tight 1/2-approximation, at the potential, quantifiable cost of additional coordination and reduced parallelism. 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 [2], image co-segmentation [3], MAP inference in the determinantal point processes [4], influence maximization [5], and document summarization [6] may be expressed as the maximization of a submodular functions. The submodular formulation enables the use of target algorithms [1, 7] 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 [7] 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. [1].

While these algorithms are both theoretically optimal and perform well in practice, their design is inherently serial limiting their scalability to large problem instances and 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 study the 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.

By exploiting the exchangeability of the greedy ordering and the sparsity of the submodular functions we develop two parallel algorithms for the maximization of non-monotone submodular functions. We propose CF-2g as a coordination free double greedy 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 allows one to choose between speed and tight approximation guarantees.
2. We prove that CC-2g produces an outcome equivalent to the sequential double greedy algorithm, thus preserving the tight approximation guarantee of 1/2.
3. We analytically bound the amount of coordination of CC-2g for example problems.
4. We provide approximation guarantees for CF-2g and analytically bound the expected loss in objective value for example problems.
5. We demonstrate empirically using two synthetic and three 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 subdmoular maximization and introduces the double greedy algorithm. Sec. 3 provides background on concurrency control in databases. We describe and provide intuition for our CF-2g and CC-2g algorithms in Sec. 4 and Sec. 5. Sec. 6 analyzes the parallel algorithms. Both algorithms are evaluated in Sec. 7, and we conclude with related work and discussions in Sec. 8 and Sec. 9.

2 Submodular Maximization

A set function $F : 2^V \rightarrow \mathbb{R}$ defined over subsets of a ground set V is *submodular* if it satisfies *diminishing marginal returns*: for all $A \subseteq B \subseteq V$ and $e \notin V$, 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 [8], graph theory [9], combinatorial optimization [10], and, more recently, in machine learning [11, 12]. Phrasing machine learning problems as submodular optimization problems enables e.g. the use of algorithms for submodular maximization [1, 7] that offer theoretical worst-case guarantees on the quality of the solution.

While those algorithms are theoretically beneficial and tend to work very well in practice, their design is inherently sequential. Recent work has addressed faster [13, 14] and parallel [15, 16] versions of the greedy algorithm by Nemhauser et al. [7] for maximizing *monotone* submodular functions that satisfy $F(A) \leq F(B)$ for any $A \subseteq B \subseteq V$. However, in several important cases such as inference [**?** **?**], and when trading off gains with (linear) cost functions (functions of the form $F(S) = G(S) + \lambda M(S)$, where $G(S)$ is monotone submodular and $M(S)$ is a linear (modular) cost function), we aim to maximize a *non-monotone* submodular function. For non-monotone functions, the simple greedy algorithm in [7] can fail terribly (see the Appendix for an example). Intuitively, while for non-monotone functions, including more elements eventually neither helps nor hurts, for non-monotone functions it can hurt to the extent of reducing the function value back to zero. For non-monotone functions, Buchbinder et al. [1] recently proposed an optimal double greedy algorithm that works well in a serial setting. In this paper, we study parallelizations of this algorithm.

The sequential double greedy algorithm. Seq-2g, the sequential double greedy algorithm (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 contains the items selected before item/iteration i, and B^i contains A^i and the items that are so far undecided. The algorithm sequentially 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). The decision depends on the gain of i with respect to A^i, and the gain of removing i from B^i. If the submodular function is monotone, then the double greedy algorithm essentially becomes a randomized version of the well-known greedy algorithm for monotone functions [7].

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:

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

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. We fixed the source of randomness in advance to simplify the presentation.

By composing the transactions $T_n(T_{n-1}(\dots T_1(\emptyset, V)))$ 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:

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

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 corresponds to a serial composition of transactions. As a consequence, we cannot directly apply the analysis of Buchbinder et al. [1] to derive strong approximation guarantees for our parallel execution.

Fortunately, several decades of research [17, 18] in database systems have explored the design of efficient parallel transaction processing systems. The systems span a space ranging from extremely fast coordination free approaches which provide minimal guarantees on the composition of transactions to sophisticated concurrency control techniques which ensure a direct correspondence between parallel and serial executions at the cost of slightly reduced performance.

In Alg. 1 we describe a common pattern for concurrency control. A transaction begins by requesting a set of guarantees $\mathfrak{g}_e$ and an associated logical time i, such that $\mathfrak{g}_e$ is guaranteed to hold at i. Under these conditions, the transaction attempts to propose and commit an update. However, if the guaranteed conditions are insufficient to do so, the transaction fails locally and the computation needs to be deferred to the server when it has the known true state $\mathfrak{S}$.

The server is in charge of providing and guaranteeing the conditions, and processing commits in the logical order of i, producing the output $\partial_n(\partial_{n-1}(\dots \partial_1(\mathfrak{S})))$. A parallel algorithm is said to be *serializable* if for any input, the output of the parallel algorithm corresponds to that of some sequential execution.

Typically, the cost of proposing the update ∂_i dominates the overall computation cost. Thus, distributing the work of proposals over multiple threads allows database systems to achieve parallelism, even with the strongly serial commit process on the server. Crucially, having a small number of deferred proposals, leaving the server to mainly perform the lightweight advancement of program state.

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

```
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 for 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.

Algorithm 3: Seq-2g: Sequential 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

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

Algorithm 5: CC-2g: concurrency control

$\hat{A} = \tilde{A} = \emptyset$, $\hat{B} = \tilde{B} = V$
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
    $(\hat{A}_e, \tilde{A}_e, \hat{B}_e, \tilde{B}_e, i)$ = getGuarantee(e)
    (result, u_e) = propose(e, $\hat{A}_e$, $\tilde{A}_e$, $\hat{B}_e$, $\tilde{B}_e$)
    commit(e, i, u_e, result)

Algorithm 6: CC-2g getGuarantee(e)

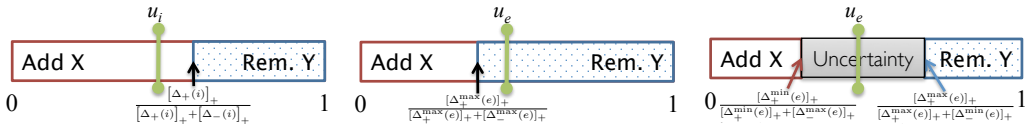
$\tilde{A}(e) \leftarrow 1$; $\tilde{B}(e) \leftarrow 0$
$i = \iota$; $\iota \leftarrow \iota + 1$
$\hat{A}_e = \hat{A}$; $\hat{B}_e = \hat{B}$
$\tilde{A}_e = \tilde{A}$; $\tilde{B}_e = \tilde{B}$
return $(\hat{A}_e, \tilde{A}_e, \hat{B}_e, \tilde{B}_e, i)$

Algorithm 7: CC-2g propose

$\Delta_+^{\min}(e) = F(\tilde{A}_e) - F(\tilde{A}_e \backslash e)$
$\Delta_+^{\max}(e) = F(\hat{A}_e \cup e) - F(\hat{A}_e)$
$\Delta_-^{\min}(e) = F(\tilde{B}_e) - F(\tilde{B}_e \cup e)$
$\Delta_-^{\max}(e) = F(\hat{B}_e \backslash e) - F(\hat{B}_e)$
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 8: CC-2g: commit(e, i, u_e, result)

wait until $\forall j < i$, processed$(j) = true$
if *result* $= fail$ **then**
  $\Delta_+^{\text{exact}}(e) = F(\hat{A} \cup e) - F(\hat{A})$
  $\Delta_-^{\text{exact}}(e) = F(\hat{B} \backslash e) - F(\hat{B})$
  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** $\hat{A}(e) \leftarrow 1$; $\tilde{B}(e) \leftarrow 1$
else $\tilde{A}(e) \leftarrow 0$; $\hat{B}(e) \leftarrow 0$
processed$(i) = true$



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

Figure 2: Illustration of algorithms. Seq-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. CF-2g approximates the threshold based on stale $\hat{A}$, $\hat{B}$, possibly choosing the wrong action. 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.

4 Coordination Free Double Greedy Algorithm

The coordination-free approach attempts to reduce the need to coordinate guarantees and logical ordering. This is achieved by operating on potentially stale states – the guarantee reduces to requiring

$\mathfrak{g}_e$ be a stale version of $\mathfrak{S}$, and 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 Seq-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 "bounds" $\hat{A}, \hat{B}$, where $\hat{A}$ is a subset of the "true" A and $\hat{B}$ 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.

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 ordering allows us to define monotonically non-decreasing sets $A^i = \{e' : e' \in A, \iota(e') < i\}$, and monotonically non-increasing sets $B^i = A^i \cup \{e' : \iota(e') \geq i\}$. These "true" sets A^i, B^i provide a serialization against which we can compare CF-2g; in this serialization, Alg. 3 computes: $\Delta_+(e) = F(A^{\iota(e)-1} \cup i) - F(A^{\iota(e)-1})$ 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] $\hat{A}_e, \hat{B}_e$: Alg. 4 computes $\Delta_+^{\max}(e) = F(\hat{A}_e \cup e) - F(\hat{A}_e)$ and $\Delta_-^{\max}(e) = F(\hat{B}_e \backslash e) - F(\hat{B}_e)$.

The next lemma shows that $\hat{A}_e, \hat{B}_e$ are bounding sets for the serialization's sets $A^{\iota(e)-1}, B^{\iota(e)-1}$. Intuitively, the bounds hold because $\hat{A}_e$, $\hat{B}_e$ 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 $e \in V$, $\hat{A}_e \subseteq A^{\iota(e)-1}$, and $\hat{B}_e \supseteq B^{\iota(e)-1}$.*

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 Double Greedy Algorithm

CC-2g, the 'concurrency-control double greedy' algorithm, 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 ensures that when concurrent transactions are serialized when they not indpendent.

Serializability is achieved by maintaining sets $\hat{A}$, $\tilde{A}$, $\hat{B}$, $\tilde{B}$, which serve as upper and lower bounds on A and B at commit time. Each thread can determine locally if a decision to include / 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, 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 $\hat{A}_e$, $\hat{B}_e$, $\tilde{A}_e$, and $\tilde{B}_e$ 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, $\forall e \in V$, $\hat{A}_e \subseteq A^{\iota(e)-1} \subseteq \tilde{A}_e \backslash e$, and $\hat{B}_e \supseteq B^{\iota(e)-1} \supseteq \tilde{B}_e \cup e$.*

Intuitively, these bounds are maintained by recording potential effects of concurrent transactions in $\tilde{A}$, $\tilde{B}$, and only recording the actual effects in $\hat{A}$, $\hat{B}$; we leave the full proof to Appendix A. Furthermore, since later transactions are blocked, we have $\hat{A} = A^{\iota(e)-1}$ and $\hat{B} = B^{\iota(e)-1}$ at commit.

Lemma 5.2. *In CC-2g, during the commit process for deferred element e, we have $\hat{A} = A^{\iota(e)-1}$ and $\hat{B} = B^{\iota(e)-1}$.*

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

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

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

6 Analysis of Algorithms

Our two algorithms appear to be diametrically opposed in their parallelization strategies. However, we show that CF-2g emphasizes speed but nevertheless suffers a bounded loss in its approximation. On the other hand, CC-2g is a scalable algorithm that guarantees the tight 1/2 approximation.

6.1 Approximation of CF-2g double greedy

Theorem 6.1. *Let F be a non-negative (monotone or non-monotone) submodular function. CF-2g solves the unconstrained problem $\max_{A \subset V} F(A)$ with approximation $E[F(A_{CF})] \geq \frac{1}{2}F^* - \frac{1}{4}\sum_{i=1}^{n} E[\rho_i]$, where A_{CF} is the output of the algorithm, F^* is the optimal value, and $\rho_i = \max\{\Delta_+^{\max}(e) - \Delta_+(e), \Delta_-^{\max}(e) - \Delta_-(e)\}$.*

The proof of Thm. 6.1 follows the structure in [1]. We present the details in Appendix C.

Example: max graph cut. Assuming bounded delay of τ and edges with unit weight, we can bound $\sum_i E[\rho_i] \leq 2\tau \frac{\#\text{edges}}{2N}$ The approximation of CF-2g is then $E[F(A^n)] \geq= \frac{1}{2}F(OPT) - \tau\frac{\#\text{edges}}{2N}$. In sparse graphs, CF-2g 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.

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.*

It suffices to show that CC-2g takes the same decision as Seq-2g for each element – locally if it is safe to do so, and otherwise deferring to the server. Full details of the proof are given in Appendix B. As an immediate consequence, theoretical properties of Seq-2g are preserved by CC-2g, including the approximation guarantees:

Theorem 6.3. *Let F be a non-negative (monotone or non-monotone) submodular function. CC-2g solves the unconstrained 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.*

6.3 Scalability of CC-2g

Whenever an element deferred for computation the server, it needs to wait for all earlier elements to be processed, and the server is blocked from committing all later elements. Each deferment effectively constitutes a barrier to the parallel processing. Hence, the scalability of CC-2g is dependent on the number of deferments. Nevertheless, we show for a couple of example problems that the number of deferments can be bounded. Full details of the bounds are given in Appendix D.

Example: max graph cut. Assume that there is a bounded delay τ. The expected *fraction* of deferred elements is upper bounded by $\tau\frac{2\#edges}{N^2}$.

Example: set cover. Assuming a bounded delay τ and non-overlapping sets S_l's, the expected *number* of deferred elements is upper bounded by 2τ.

7 Evaluation

We implemented the parallel and sequential 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 $F(A_{CF}) - F(A_{Seq})$, the difference between the objective value on the sets

returned by CF-2g and Seq-2g. We verified the correctness of CC-2g by comparing the output of CC-2g with Seq-2g. We also the fraction of elements deferred by CC-2g.

Graph	# vertices	# edges	Description
Erdos-Renyi	20,000,000	$\approx 2 \times 10^6$	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. [19]
Arabic-2005	22,744,080	631,153,669	2005 crawl of Arabic web sites [20, 21, 22].
UK-2005	39,459,925	921,345,078	2005 crawl of the .uk domain [20, 21, 22].
IT-2004	41,291,594	1,135,718,909	2004 crawl of the .it domain [20, 21, 22].

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

We tested our parallel algorithms 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 vertices close to each another in the graph were also close in their indices. To reduce this dependency, we randomly permuted the ordering of vertices.

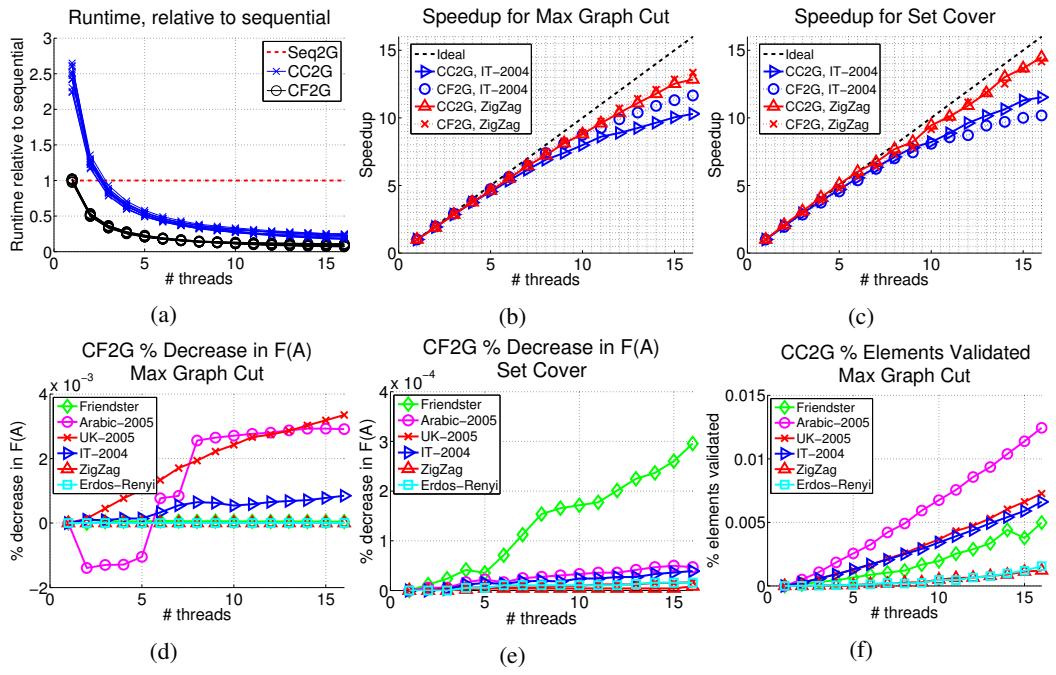


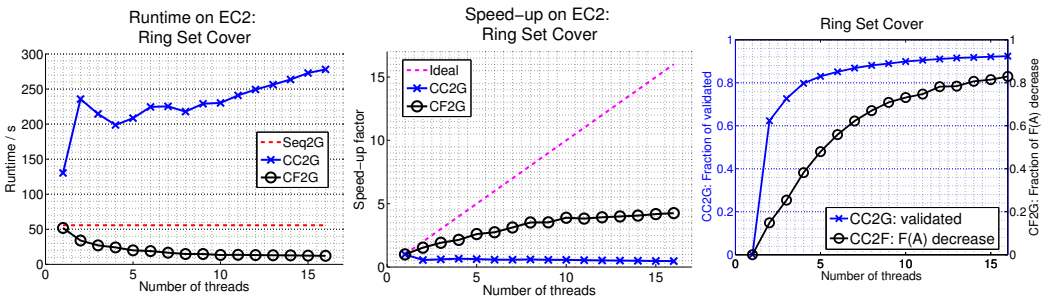
Figure 3: Experimental results. Fig. 3a – runtime of the parallel algorithms as a ratio to that of the sequential algorithm. Each curve shows the runtime of a parallel algorithm on a particular graph for a particular function F. Fig. 3b, 3c – speedup (ratio of runtime on one thread to that on p threads).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

Due to space constraints, we only present part of our results in Fig. 3, deferring full results to Appendix F. **Runtime, Speedup:** Both parallel algorithms are faster than the sequential 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 sequential objective value by less than 0.01%. **Deferments:** CC-2g defers more elements as threads are added, but less than 0.015% are deferred with 16 threads, which has negligible effect on the runtime / speedup.

7.1 Adversarial ordering

To highlight the differences in approaches between the two parallel algorithms, we conducted experiments on a ring Cayley 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 defer more proposals. [3]

As Fig. 4 shows, CC-2g sacrifices speed to ensure serial equivalence, eventually deferring $> 90\%$ of elements. On the other hand, CF-2g focuses on speed, resulting in faster runtime, but delivering an objective value $F(A_{CF})$ that is only 20% of $F(A_{Seq})$. For the set cover problem, we maintain statistics that are updated atomically by both algorithms. The adversarial ordering forces more concurrent atomic updates, and hence, CF-2g does not achieve good speedup. [4]



(a) (b) (c)

Figure 4: Experimental results for ring graph on set cover problem.

8 Related Work

Similar approach, different problem: OCC DP-means [23]; Hogwild SGD [24]; Coordination-free LDA [25] / parameter servers [26, 27]

Similar problem, different approach: distributed greedy submodular maximization for monotone functions [15]

9 Discussions

Conclusion: [XP: link back to intro, motivation]; we present two approaches to parallelizing unconstrained submodular maximization, which allows one to choose between speed and tight approximation guarantees.

Future work: interpolate between CC-2g and CF-2g; constrained maximization, minimization; monotone maximization using lazy greedy; distributed setting, where communication costs and delays are higher, and function evaluations are challenging.

References

[1] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, pages 649–658, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4874-6. doi: 10.1109/FOCS.2012.73. URL http://dx.doi.org/10.1109/FOCS.2012.73.

[3] We point out by using a partitioning scheme, it is possible to avoid the problems caused by the adversarial ordering, and to improve scalability. Nevertheless, we present results that do *not* use the partitioning scheme, so as to better highlight the differences between the two parallel algorithms.

[4] We could have reduced coordination by computing F directly, but doing so would result in longer runtimes.

[2] 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.

[3] G. Kim, E. P. Xing, F. Li, and T. Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *ICCV*, 2011.

[4] A. Kulesza J. Gillenwater and B. Taskar. Near-optimal MAP inference for determinantal point processes. In *NIPS*, 2012.

[5] 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.

[6] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, 2011.

[7] 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.

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

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

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

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

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

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

[14] K. Wei, R. Iyer, and J. Bilmes. Fast multi-stage submodular maximization. In *ICML*, 2014.

[15] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems 26*. 2013.

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

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

[18] Hsiang-Tsung Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.

[19] Jure Leskovec. Stanford network analysis project, 2011. URL `http://snap.stanford.edu/index.html`.

[20] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[21] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*. ACM Press, 2011.

[22] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[23] Xinghao Pan, Joseph E Gonzalez, Stefanie Jegelka, Tamara Broderick, and Michael Jordan. Optimistic concurrency control for distributed unsupervised learning. In *Advances in Neural Information Processing Systems 26*. 2013.

[24] Benjamin Recht, Christopher Re, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS) 24*, pages 693–701, Granada, 2011.

[25] Amr Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM)*, 2012.

[26] 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.

[27] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26*. 2013.

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

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

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 $e' \in \hat{A}_e$, it must be the case that the algorithm set $\hat{A}(e')$ to 1 (line 10) before T_e, which implies $\iota(e') < \iota(e)$, and hence $e' \in A^{\iota(e)-1}$. So $\hat{A}_e \subseteq A^{\iota(e)-1}$.

Similarly, if $e' \notin \hat{B}_e$, then the algorithm set $\hat{B}(e')$ 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 $\hat{B}_e \subseteq B^{\iota(e)-1}$. □

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

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

Consider any $e' \neq e$. Suppose $e' \in \hat{A}_e$. 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 $\tilde{A}(e') \leftarrow 1$ (by the ordering imposed by ι on Line 2), but never execute $\tilde{A}(e') \leftarrow 0$ (since $e' \in A$), so $e' \in \tilde{A}_e$.

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

Lemma 5.2. *In CC-2g, during the commit process for deferred element e, we have $\hat{A} = A^{\iota(e)-1}$ and $\hat{B} = B^{\iota(e)-1}$.*

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 $e' \in \hat{A} \iff e' \in A^{\iota(e)-1}$ and $e' \in \hat{B} \iff e' \in B^{\iota(e)-1}$. □

B Proof of serial equivalence of CC-2g

Theorem 6.2. *CC-2g is serializable.*

Proof. We will denote by A^i_{seq}, B^i_{seq} the sets generated by Seq-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 Seq-2g. Hence, the CC-2g algorithm includes $e \in A$ iff Seq-2g includes $e \in A$. (An analogous argument works for the case where e is excluded from B.) □

C Proof of bound for hogwild

We follow the proof outline of [1].

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 [1]. □

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 [1]. First, note that it suffices to prove the inequality conditioned on knowing A^{i-1}, $\hat{A}_i$ and $\hat{B}_i$, 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 $\Delta_-^{\max}(i)/(\Delta_+^{\max}(i) + \Delta_-^{\max}(i))$.

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

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

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:

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

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 $O^i = O^{i-1} \cup i$:

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

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

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

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

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

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

□

We will now prove the main theorem.

Theorem 6.1. *Let F be a non-negative (monotone or non-monotone) submodular function. CF-2g solves the unconstrained problem $\max_{A \subset V} F(A)$ with approximation $E[F(A_{CF})] \ge \frac{1}{2}F^* - \frac{1}{4}\sum_{i=1}^n E[\rho_i]$, where A_{CF} is the output of the algorithm, F^* is the optimal value, and $\rho_i = \max\{\Delta_+^{\max}(e) - \Delta_+(e), \Delta_-^{\max}(e) - \Delta_-(e)\}$.*

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

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

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

C.1 Example: max graph cut

Let $C_i = (A^{i-1} \backslash \hat{A}_i) \cup (\hat{B}_i \backslash B^{i-1})$ 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 $\bar{A}_j = V \backslash \hat{A}_i \backslash C_i \backslash D_i = \{1, \dots, j\} \backslash \hat{A}_i$ be the elements up to j that are not included in A. 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

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

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

Suppose we have bounded delay τ, so $|C_i| \le \tau$. Then $w_i(C_i)$ has a hypergeometric distribution with mean $\frac{\deg(i)}{N}\tau$, and $E[\rho_i] \le 2\tau\frac{\deg(i)}{N}$. The approximation of the hogwild algorithm is then $E[F(A^n)] \ge \frac{1}{2}F^* - \tau\frac{\#\text{edges}}{2N}$. 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 $E[F(A^n)] \ge F^*\left(\frac{1}{2} - \frac{\tau}{N}\right)$, 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 $\lambda < 1$:

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

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. Let n_l denote $|S_l|$ 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. $|\{e' : \pi(e') \le \pi(e_l^t) \wedge e' \in S_l\}| = t$.

For any $e \in S_l$, we get

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

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 $\hat{B}_{e_l^{n_l}} \backslash e_l^{n_l}$, and there is some element e' in $\hat{B}_{e_l^{n_l}} \backslash e_l^{n_l}$ that is rejected and not in $B^{\iota(e_l^{n_l})-1}$. Denote by $m_l \le \min(\tau, n_l - 1)$ the number of elements before $e_l^{n_l}$ that are inconsistent between $\hat{B}_{e_l^{n_l}}$ 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

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

If $\lambda = 1$, $\Delta_+^{\max}(e) \le 0$, 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 $(A^{\iota(e)-1} \backslash \hat{A}_e) \cap S_l \neq \emptyset$, that is, if an element has been accepted in A but not yet observed in $\hat{A}_e$. Since we assume a bounded delay, only the first τ elements after the first acceptance of an $e \in S_l$ may be affected.

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

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 $\#\{e : e \in S_l \wedge e_l^\eta \in A^{\iota(e)-1} \wedge e_l^\eta \notin \hat{A}_e\}$ 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

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

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 elements sent for validation

Let N be the number of elements, i.e. the cardinality of the ground set. Let $C_i = (A^{i-1} \backslash \hat{A}_i) \cup (\hat{B}_i \backslash B^{i-1})$. 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 be validated. Conversely, if i is validated, there must be some element $i' \in C_i$ such that i is dependent on i'.

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

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 validated elements 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 validated 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 $\hat{B}_e \backslash e \cap S_l \neq \emptyset$ and $\tilde{B}_e \backslash e \cap S_l = \emptyset$. This is only possible if $e_l^{n_l} \notin \tilde{B}_e$ and $\tilde{B}_e \supset \hat{A}_e \cap S_l = \emptyset$, 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

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

Now we consider any element $e \in S_l$ with $\pi(e) < \pi(e_l^{n_l}) - \tau$ that is validated. (Note that $e_l^{n_l} \in \hat{B}_e$ and $\tilde{B}_e$, so $\Delta_-^{\min}(e) = \Delta_-^{\max}(e) = \lambda$.) It must be the case that $\hat{A}_e \cap S_l = \emptyset$, for otherwise $\Delta_+^{\min}(e) = \Delta_+^{\max}(e) = -\lambda$ and we do not need to validate. This implies that $\Delta_+^{\max}(e) = 1 - \lambda \geq u_i$. At validation, 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 is validated must be within τ of the first accepted element $e_l^{\eta}\ in S_l$. 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 we need to validate is upper bounded as

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

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 Full experiment results

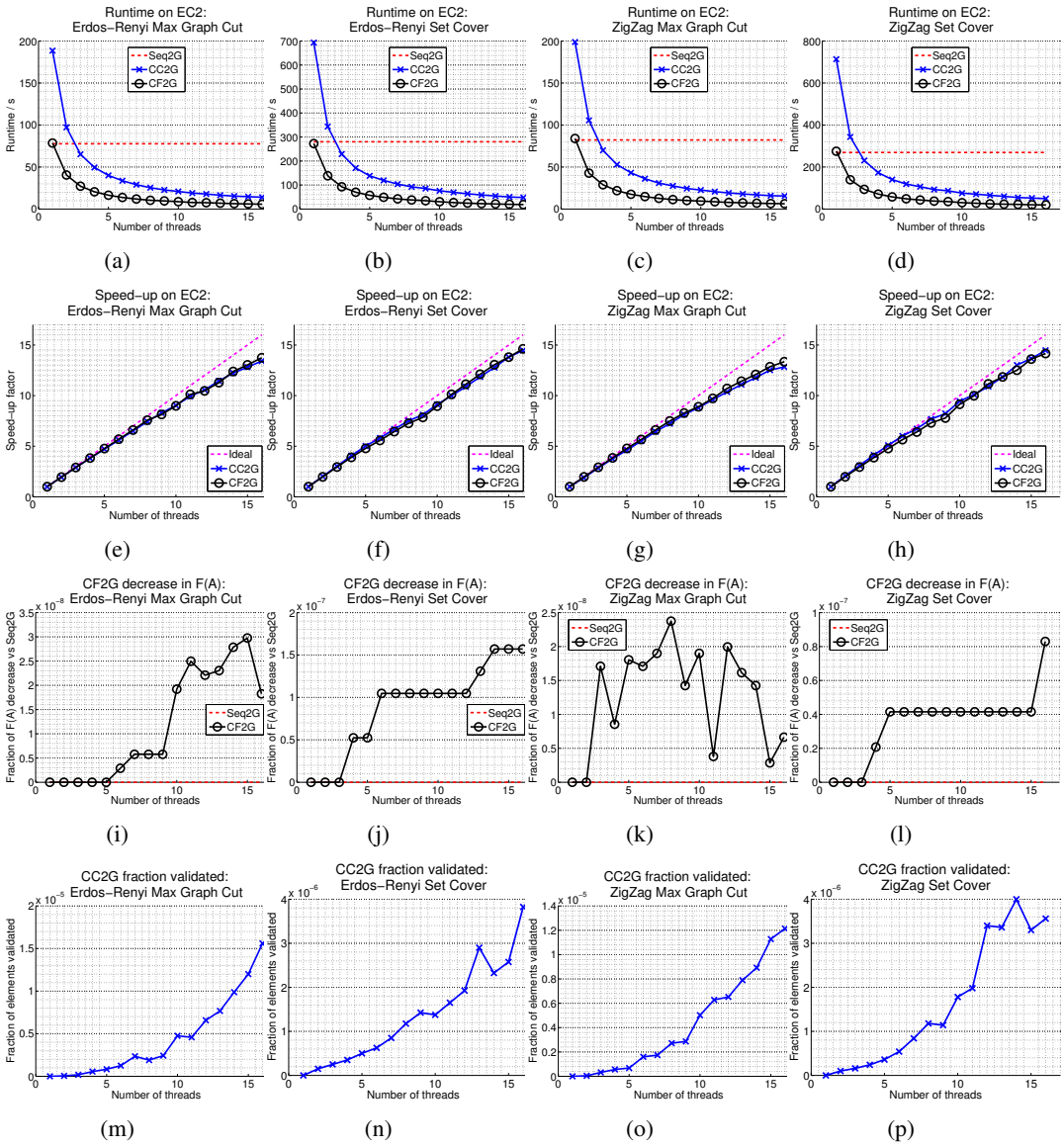


Figure 5: Experimental results on Erdos-Renyi and ZigZag synthetic graphs.

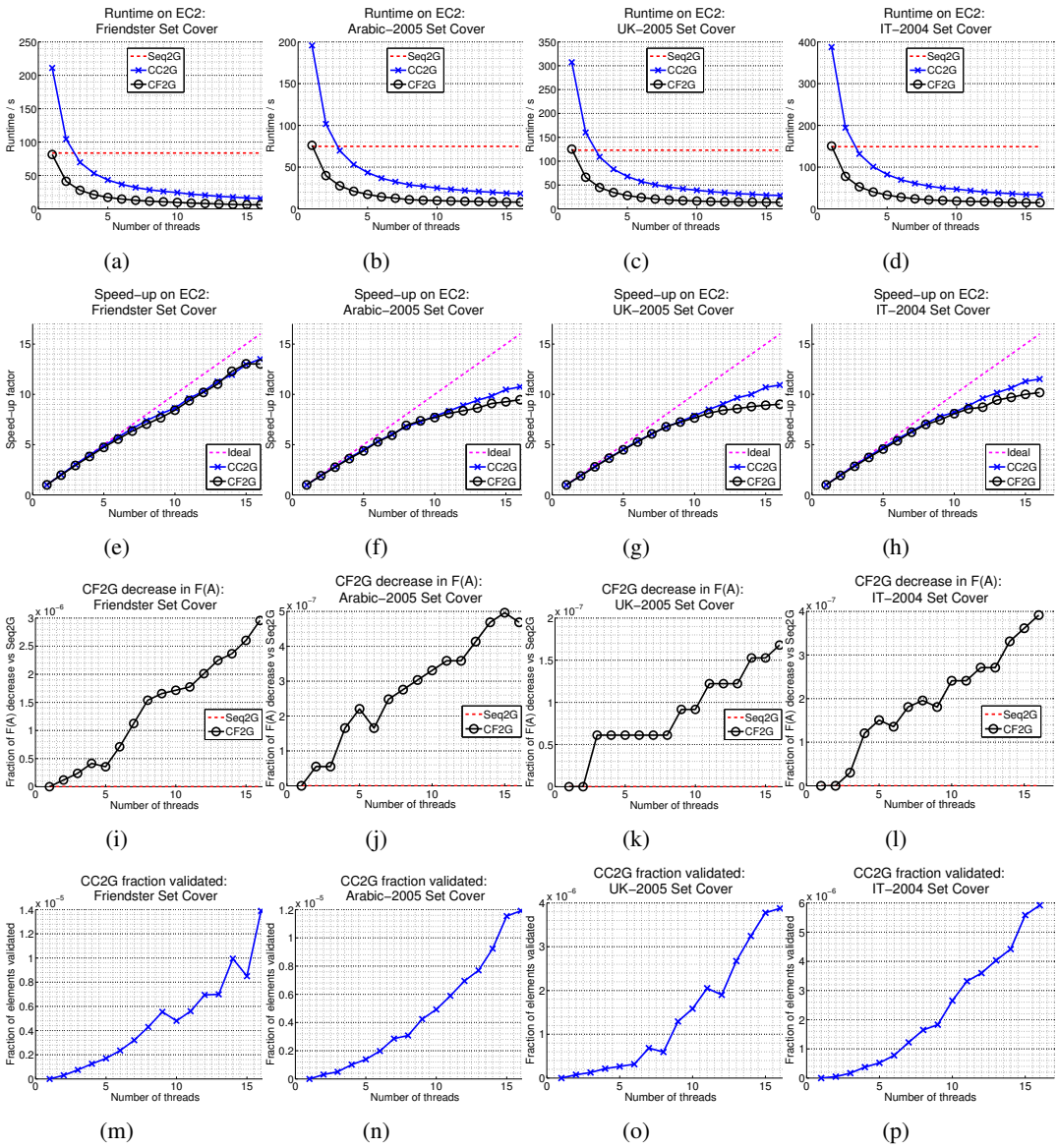


Figure 6: Set cover on 4 real graphs.

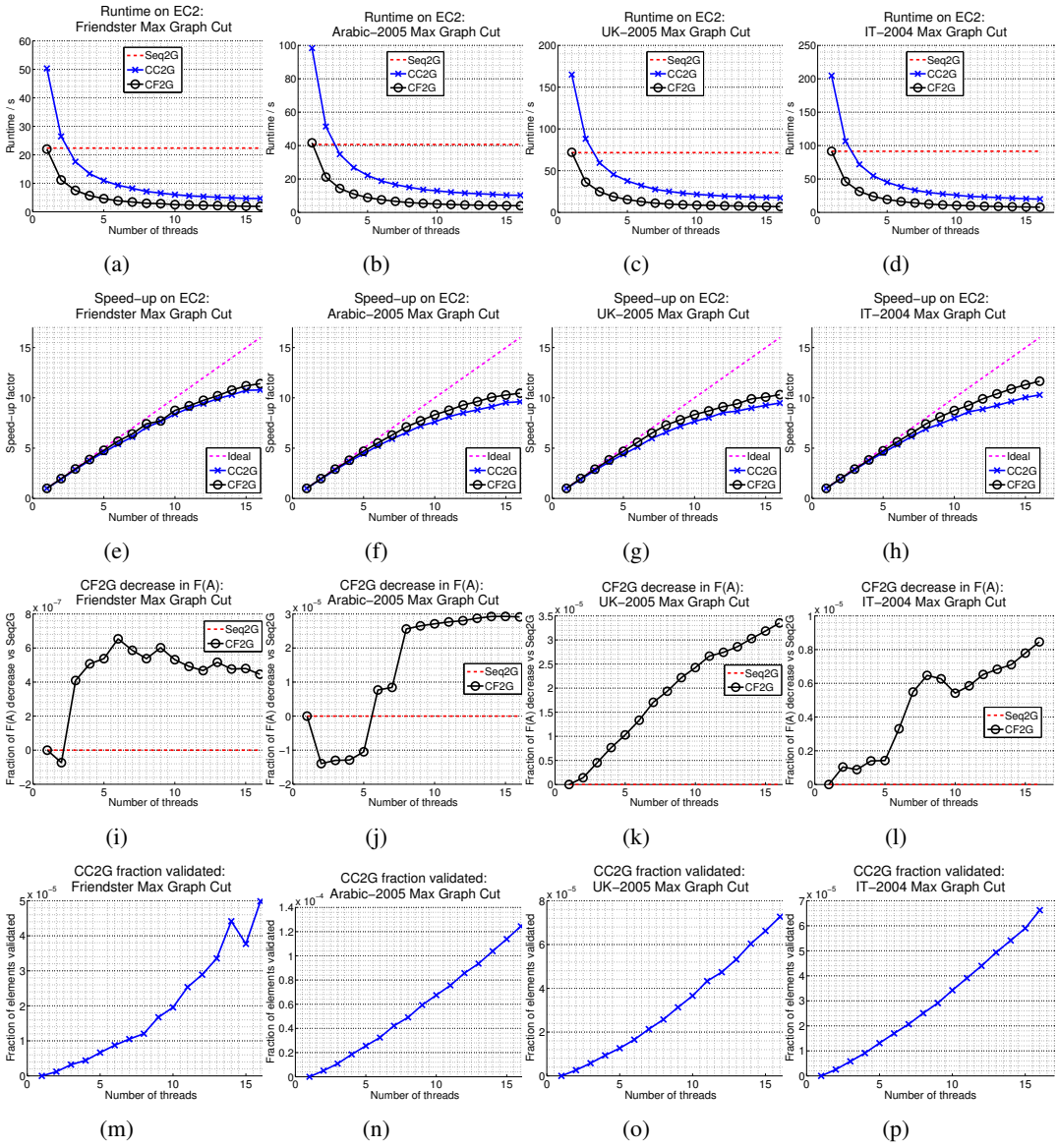


Figure 7: Max graph cut on 4 real graphs.

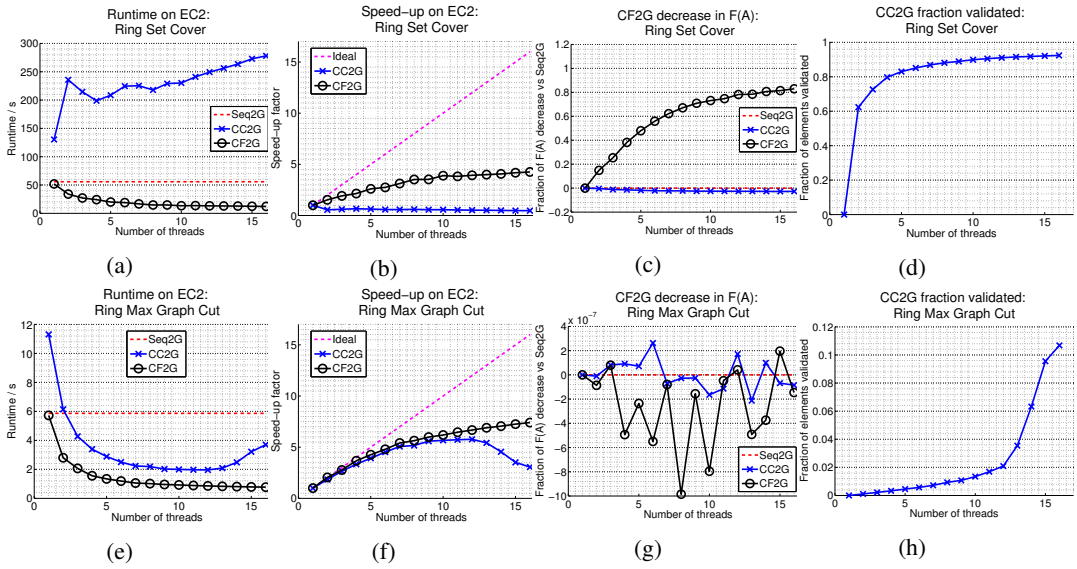


Figure 8: Experimental results for ring graph on set cover problem.