

Greedy Algorithms for Cardinality-Constrained Submodular Maximization

Feier Lyu

July 14, 2026

Abstract

We formalize in Isabelle/HOL the classical approximation guarantee for monotone non-negative submodular maximization under a cardinality constraint on a finite ground set. The main result is the Nemhauser–Wolsey bound for deterministic greedy: after k steps, the greedy solution achieves the finite-step guarantee $1 - (1 - 1/k)^k$, and hence also the standard $(1 - 1/e)$ approximation ratio. The development also includes a verified stateful lazy greedy variant, based on cached upper bounds on marginal gains, and proves that it satisfies the same approximation guarantee.

Contents

0.1	Optimal feasible sets	4
0.2	Basic non-emptiness facts	4
1	Greedy construction	5
1.1	Preliminaries on finite maximizers	5
1.2	Hilbert-choice arg-max oracle for marginal gain	6
2	Lazy selection via cached upper bounds	10
3	Stateful lazy greedy with cached upper bounds	11
3.1	State: selected set and cached upper bounds	12
3.2	Inner lazy selection returning updated upper bounds	12
3.3	Preservation of upper-bound validity across outer iterations .	12
3.4	One outer step and the full stateful algorithm	13
3.5	Main invariants: subset property and validity on the remain- ing set	13
3.6	Correctness of the lazy greedy step	14

4 Greedy approximation for monotone submodular maximization	15
4.1 Greedy gap analysis	16
4.1.1 Gap sequence	16
4.2 Non-negativity of OPT and approximation ratio	17
4.3 Corollaries	18
5 Step-spec corollary	18
6 Acknowledgements	21

Background

The classical greedy algorithm for monotone submodular maximization under a cardinality constraint is due to Nemhauser, Wolsey, and Fisher [2]. This entry also formalizes a stateful lazy-greedy variant, following the accelerated greedy idea of Minoux [1]: instead of recomputing all marginal gains at every outer iteration, the algorithm keeps cached upper bounds on marginal gains and recomputes them only when needed. The formalization proves that this lazy implementation still selects a valid greedy element at each step and therefore inherits the same approximation guarantee.

```
theory Submodular_Base
  imports Complex_Main
begin
```

```
lemma finite_has_maximal_on:
  fixes g :: "'a ⇒ 'b::linorder"
  assumes fin: "finite A"
    and nonempty: "A ≠ {}"
  shows "∃ x∈A. ∀ y∈A. g y ≤ g x" <proof>
```

The main development is carried out in a single locale for normalized monotone submodular functions, which is the setting needed for the cardinality-constrained greedy approximation guarantees below. Some auxiliary facts hold under weaker assumptions; for this entry we keep them in the same locale to maintain a compact and uniform development.

```
locale Submodular_Func =
  fixes V :: "'a set" and f :: "'a set ⇒ real"
  assumes finite_V: "finite V"
    and monotone_f: "⋀ S T. S ⊆ T ⇒ T ⊆ V ⇒ f S ≤ f T"
    and submodular_f:
      "⋀ S T. S ⊆ V ⇒ T ⊆ V ⇒ f (S ∪ T) + f (S ∩ T) ≤ f S + f T"
    and f_empty: "f {} = 0"
begin
```

Marginal gain of adding a single element to a set.

```

definition gain :: "'a set  $\Rightarrow$  'a  $\Rightarrow$  real" where
  "gain S e = f (S  $\cup$  {e}) - f S"

```

```

lemma f_nonneg:
  assumes "S  $\subseteq$  V"
  shows "0  $\leq$  f S"
<proof>

```

```

lemma monotone_on_PowV:
  shows "monotone_on (Pow V) ( $\subseteq$ ) ( $\leq$ ) f"
<proof>

```

```

lemma gain_nonneg:
  assumes "S  $\subseteq$  V" and "x  $\in$  V - S"
  shows "0  $\leq$  gain S x"
<proof>

```

Diminishing returns for single-element marginal gains.

```

lemma gain_decreasing:
  assumes "S  $\subseteq$  T" "T  $\subseteq$  V" "x  $\in$  V" "x  $\notin$  T"
  shows "gain S x  $\geq$  gain T x"
<proof>

```

Set-valued diminishing returns.

```

lemma gain_decreasing_set:
  assumes "S  $\subseteq$  T" "T  $\subseteq$  V" "A  $\subseteq$  V"
  shows "f (S  $\cup$  A) - f S  $\geq$  f (T  $\cup$  A) - f T"
<proof>

```

end

This entry treats cardinality-constrained monotone submodular maximization. Other constraint systems, such as matroid or knapsack constraints, are outside the scope of the present development.

```

locale Cardinality_Constraint = Submodular_Func +
  fixes k :: nat
  assumes k_le_cardV: "k  $\leq$  card V"
begin

```

```

definition feasible :: "'a set  $\Rightarrow$  bool" where
  "feasible S  $\longleftrightarrow$  S  $\subseteq$  V  $\wedge$  card S  $\leq$  k"

```

```

lemma feasibleI:
  assumes "S  $\subseteq$  V" "card S  $\leq$  k"
  shows "feasible S"
<proof>

```

```

lemma feasibleD:
  assumes "feasible S"

```

```

shows "S ⊆ V" "card S ≤ k"
⟨proof⟩

lemma feasible_empty[simp]: "feasible {}"
⟨proof⟩

lemma feasible_family_nonempty: "Collect feasible ≠ {}"
⟨proof⟩

lemma finite_feasible_family: "finite {S. feasible S}"
⟨proof⟩

```

0.1 Optimal feasible sets

We select a canonical optimal feasible set OPT_set using Hilbert choice and define OPT_k as its value. These are problem-level objects for the cardinality-constrained maximization problem, independent of any particular greedy implementation.

```

definition OPT_set :: "'a set" where
  "OPT_set =
    (SOME X. feasible X ∧ (∀ Y. feasible Y ⟶ f Y ≤ f X))"

```

```

lemma exists_max_feasible:
  "∃ X. feasible X ∧ (∀ Y. feasible Y ⟶ f Y ≤ f X)"
⟨proof⟩

```

```

lemma OPT_set_props:
  shows OPT_set_in: "feasible OPT_set"
  and OPT_set_max: "∀ Y. feasible Y ⟶ f Y ≤ f OPT_set"
⟨proof⟩

```

```

definition OPT_k :: real where
  "OPT_k = f OPT_set"

```

```

lemma exists_opt_set:
  "∃ X. feasible X ∧ f X = OPT_k"
⟨proof⟩

```

```

lemma OPT_k_upper_bound:
  assumes "feasible S"
  shows "f S ≤ OPT_k"
⟨proof⟩

```

0.2 Basic non-emptiness facts

```

lemma nonempty_candidates:
  assumes "S ⊆ V" "card S < k"
  shows "V - S ≠ {}"
⟨proof⟩

```

```

lemma nonempty_gap:
  assumes "S ⊆ V" "Opt ⊆ V" "f S < f Opt"
  shows "Opt - S ≠ {}"
<proof>

```

```

lemma OPT_k_nonneg: "0 ≤ OPT_k"
<proof>

```

Submodular telescoping: sum of marginals upper-bounds the joint gain.

```

lemma submod_sum_upper:
  assumes "finite A" "A ⊆ V" "S ⊆ V" "A ∩ S = {}"
  shows "f (S ∪ A) - f S ≤ (∑ x∈A. gain S x)"
<proof>

```

Average marginal bound against any candidate set $Opt \subseteq V$ with $card\ Opt \leq k$: if $S \subseteq V$ and $card\ S < k$, then there exists an element $e \in V - S$ such that $gain\ S\ e \geq (f\ Opt - f\ S) / real\ k$.

```

lemma marginal_gain_lower_bound:
  fixes Opt S :: "'a set"
  assumes S_sub: "S ⊆ V"
    and O_sub: "Opt ⊆ V"
    and cardS_lt_k: "card S < k"
    and cardO_le_k: "card Opt ≤ k"
  shows "∃ e∈V - S. gain S e ≥ (f Opt - f S) / real k"
<proof>

```

end

end

```

theory Greedy_Submodular_Construct
  imports "../Core/Submodular_Base"
begin

```

1 Greedy construction

This theory sets up the greedy construction for monotone submodular maximization under a cardinality constraint. The locale *Greedy_Setup* fixes a finite ground set V , a budget k , and a normalized monotone submodular set function f . The greedy sequence starts from the empty set and repeatedly adds an element of maximum marginal gain, with ties broken by the abstract oracle.

1.1 Preliminaries on finite maximizers

Finite arg-max via the standard maximality predicate.

```

lemma finite_is_arg_max_in:

```

```

fixes g :: "'a ⇒ 'b::linorder"
assumes fin: "finite A" and ne: "A ≠ {}"
shows "∃ x∈A. is_arg_max g (λx. x ∈ A) x"
⟨proof⟩

```

```

lemma is_arg_maxD_le:
  fixes f :: "'b ⇒ 'a::linorder"
  assumes "is_arg_max f P x" "P y"
  shows "f y ≤ f x"
⟨proof⟩

```

Abstract setup for the greedy algorithm: a finite ground set V , a budget k , and a non-negative monotone submodular function f with $f = 0$.

This theory focuses on the greedy construction and basic structural properties, without yet proving approximation guarantees.

1.2 Hilbert-choice arg-max oracle for marginal gain

```

context Submodular_Func
begin

```

```

definition argmax_gain_some :: "'a set ⇒ 'a set ⇒ 'a" where
  "argmax_gain_some S A =
    (SOME x. x ∈ A ∧ is_arg_max (gain S) (λy. y ∈ A) x)"

```

```

lemma argmax_gain_some_mem:
  assumes fin: "finite A" and ne: "A ≠ {}"
  shows "argmax_gain_some S A ∈ A"
⟨proof⟩

```

```

lemma argmax_gain_some_max:
  assumes fin: "finite A" and ne: "A ≠ {}" and yA: "y ∈ A"
  shows "gain S y ≤ gain S (argmax_gain_some S A)"
⟨proof⟩

```

```

end

```

```

locale Greedy_Setup = Cardinality_Constraint V f k
  for V :: "'a set" and f :: "'a set ⇒ real" and k :: nat +
  fixes argmax_gain :: "'a set ⇒ 'a set ⇒ 'a"
  assumes argmax_gain_mem:
    "finite A ⇒ A ≠ {} ⇒ argmax_gain S A ∈ A"
  assumes argmax_gain_max:
    "finite A ⇒ A ≠ {} ⇒ ∀ y∈A. gain S y ≤ gain S (argmax_gain S A)"
begin

```

Greedy construction: start from $\{\}$ and, as long as there are remaining elements, add one with maximum marginal gain.

```

fun greedy_set :: "nat  $\Rightarrow$  'a set" where
  "greedy_set 0 = {}"
| "greedy_set (Suc i) =
  (let S = greedy_set i in
   if V - S = {} then S else insert (argmax_gain S (V - S)) S)"

```

Greedy sets are always subsets of the ground set V .

```

lemma greedy_subset_V: "greedy_set i  $\subseteq$  V"
<proof>

```

If a genuinely new element is inserted, the cardinality increases by one.

```

lemma greedy_card_step:
  assumes ne: "V - greedy_set i  $\neq$  {}"
  shows "card (greedy_set (Suc i)) = card (greedy_set i) + 1"
<proof>

```

If the remainder is empty, the greedy set stays unchanged.

```

lemma greedy_card_idle:
  assumes "V - greedy_set i = {}"
  shows "card (greedy_set (Suc i)) = card (greedy_set i)"
<proof>

```

State transition: when the remainder is non-empty, S evolves by adding the arg-max element.

```

lemma state_transition_nonempty:
  assumes "0 < i" and "V - greedy_set (i - 1)  $\neq$  {}"
  shows "greedy_set i =
    greedy_set (i - 1)
     $\cup$  {argmax_gain (greedy_set (i - 1)) (V - greedy_set (i - 1))}"
<proof>

```

At most one new element is added in each greedy step.

```

lemma card_greedy_le_i: "card (greedy_set i)  $\leq$  i"
<proof>

```

If $i \leq k$ then $\text{card}(\text{greedy_set } i) \leq k$ (used later in the gap bound).

```

lemma card_greedy_le_k:
  assumes "i  $\leq$  k"
  shows "card (greedy_set i)  $\leq$  k"
<proof>

```

If $\text{card}(\text{greedy_set } t) < \text{card } V$, then the remainder $V - \text{greedy_set } t$ is non-empty.

```

lemma remainder_nonempty_if_card_ltV:
  assumes "card (greedy_set t) < card V"
  shows "V - greedy_set t  $\neq$  {}"
<proof>

```

Under $k \leq \text{card } V$, the greedy transition up to step k always adds a new element.

```

lemma state_transition_upto_k:
  assumes "0 < i" "i ≤ k"
  shows "greedy_set i =
        greedy_set (i - 1)
        ∪ {argmax_gain (greedy_set (i - 1)) (V - greedy_set (i - 1))}"
⟨proof⟩

```

Intermediate greedy states as a list $[S, \dots, S]$.

```

definition greedy_sequence :: "nat ⇒ 'a set list" where
  "greedy_sequence n = map greedy_set [0.. $\text{Suc } n$ ]"

```

Indexing lemma for the sequence representation.

```

lemma greedy_sequence_nth[simp]:
  assumes "i ≤ n"
  shows "greedy_sequence n ! i = greedy_set i"
⟨proof⟩

```

Every greedy state is finite.

```

lemma greedy_set_finite [simp]: "finite (greedy_set i)"
⟨proof⟩

```

One-step monotonicity: $S \subseteq S$.

```

lemma greedy_mono_Suc: "greedy_set i ⊆ greedy_set (Suc i)"
⟨proof⟩

```

Chain monotonicity: if $i \leq j$ then $S \subseteq S$.

```

lemma greedy_chain_mono:
  assumes "i ≤ j"
  shows "greedy_set i ⊆ greedy_set j"
⟨proof⟩

```

Cardinality is non-decreasing along the greedy sequence.

```

lemma greedy_card_mono:
  "i ≤ j ⇒ card (greedy_set i) ≤ card (greedy_set j)"
⟨proof⟩

```

A compact cardinality bound: $\text{card } S \leq \min i (\text{card } V)$ for all i .

```

lemma greedy_card_min:
  "card (greedy_set i) ≤ min i (card V)"
⟨proof⟩

```

Length and endpoints of the intermediate sequence.

```

lemma greedy_sequence_length [simp]:
  "length (greedy_sequence n) = Suc n"
⟨proof⟩

```

```

lemma greedy_sequence_0 [simp]:
  "greedy_sequence n ! 0 = {}"
⟨proof⟩

```

```

lemma greedy_sequence_last [simp]:
  "greedy_sequence n ! n = greedy_set n"
  <proof>

```

At a non-empty remainder, the chosen element is new and lies in $V - S$.

```

lemma chosen_in_remainder_nonempty:
  assumes rem_ne: "V - greedy_set i ≠ {}"
  defines x_def: "x ≡ argmax_gain (greedy_set i) (V - greedy_set i)"
  shows "x ∈ V - greedy_set i" "x ∉ greedy_set i"
  <proof>

```

At a non-empty step, $\text{greedy_set } (\text{Suc } i)$ is obtained by inserting the arg-max element into $\text{greedy_set } i$. This is often useful in counting arguments.

```

lemma greedy_increment_nonempty[simp]:
  assumes rem_ne: "V - greedy_set i ≠ {}"
  shows "greedy_set (Suc i) =
        insert (argmax_gain (greedy_set i) (V - greedy_set i)) (greedy_set i)"
  <proof>

```

One-step update of the objective value along the greedy sequence.

```

lemma greedy_step_f_eq:
  assumes "V - greedy_set i ≠ {}"
  shows
    "f (greedy_set (Suc i)) =
     f (greedy_set i) +
     gain (greedy_set i)
     (argmax_gain (greedy_set i) (V - greedy_set i))"
  <proof>

```

end

```

context Cardinality_Constraint
begin

```

```

interpretation Greedy_Concrete: Greedy_Setup V f k argmax_gain_some
  <proof>

```

end

```

end
theory Lazy_Greedy_Oracle
  imports "Greedy_Submodular_Construct"
begin

```

This theory provides auxiliary lazy-selection primitives based on cached upper bounds for marginal gains. It is used by the stateful lazy greedy

construction below, while the final approximation theorem is stated in the proof layer.

2 Lazy selection via cached upper bounds

```

context Submodular_Func
begin

definition ub_valid :: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  bool" where
  "ub_valid S A ub  $\longleftrightarrow$  ( $\forall x \in A. \text{gain } S \ x \leq \text{ub } x$ )"

definition untight :: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  'a set" where
  "untight S A ub = {x  $\in$  A. ub x > gain S x}"

definition tighten :: "'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\Rightarrow$  real)" where
  "tighten S ub x = ub(x := gain S x)"

definition pick_ub_some :: "'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  'a" where
  "pick_ub_some A ub = (SOME x. x  $\in$  A  $\wedge$  is_arg_max ub ( $\lambda y. y \in A$ ) x)"

lemma pick_ub_some_mem:
  assumes finA: "finite A" and neA: "A  $\neq$  {}"
  shows "pick_ub_some A ub  $\in$  A"
  <proof>

lemma pick_ub_some_max:
  assumes finA: "finite A" and neA: "A  $\neq$  {}" and yA: "y  $\in$  A"
  shows "ub y  $\leq$  ub (pick_ub_some A ub)"
  <proof>

fun lazy_argmax_gain_fuel ::
  "nat  $\Rightarrow$  'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  'a"
where
  "lazy_argmax_gain_fuel 0 S A ub = pick_ub_some A ub"
| "lazy_argmax_gain_fuel (Suc n) S A ub =
  (let x = pick_ub_some A ub in
   if ub x = gain S x then x
   else lazy_argmax_gain_fuel n S A (tighten S ub x))"

definition lazy_argmax_gain :: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  'a"
where
  "lazy_argmax_gain S A ub = lazy_argmax_gain_fuel (card A) S A ub"

lemma ub_valid_tighten:
  assumes ubv: "ub_valid S A ub"
  shows "ub_valid S A (tighten S ub x)"
  <proof>

```

```

lemma untight_tighten:
  assumes xA: "x ∈ A" and gt: "ub x > gain S x"
  shows "untight S A (tighten S ub x) = untight S A ub - {x}"
  ⟨proof⟩

lemma finite_untight:
  assumes finA: "finite A"
  shows "finite (untight S A ub)"
  ⟨proof⟩

lemma lazy_argmax_gain_fuel_max:
  assumes finA: "finite A" and neA: "A ≠ {}"
  shows "ub_valid S A ub ⇒ card (untight S A ub) ≤ n ⇒
    ∀y∈A. gain S y ≤ gain S (lazy_argmax_gain_fuel n S A ub)"
  ⟨proof⟩

lemma lazy_argmax_gain_fuel_mem:
  assumes finA: "finite A" and neA: "A ≠ {}"
  shows "lazy_argmax_gain_fuel n S A ub ∈ A"
  ⟨proof⟩

lemma lazy_argmax_gain_mem:
  assumes finA: "finite A" and neA: "A ≠ {}"
  shows "lazy_argmax_gain S A ub ∈ A"
  ⟨proof⟩

lemma lazy_argmax_gain_max:
  assumes finA: "finite A" and neA: "A ≠ {}"
  and ubv: "ub_valid S A ub"
  shows "∀y∈A. gain S y ≤ gain S (lazy_argmax_gain S A ub)"
  ⟨proof⟩

end

end
theory Lazy_Greedy_Stateful
  imports Lazy_Greedy_Oracle
begin

```

3 Stateful lazy greedy with cached upper bounds

```

record 'a lg_state =
  Sg  :: "'a set"
  ubg :: "'a ⇒ real"

context Cardinality_Constraint
begin

```

3.1 State: selected set and cached upper bounds

```
definition init_ub :: "'a  $\Rightarrow$  real" where
  "init_ub x = gain {} x"
```

```
definition init_state :: "'a lg_state" where
  "init_state = ( $\emptyset$  Sg = {}, ubg = init_ub  $\emptyset$ )"
```

```
definition remaining :: "'a lg_state  $\Rightarrow$  'a set" where
  "remaining st = V - Sg st"
```

3.2 Inner lazy selection returning updated upper bounds

```
fun lazy_argmax_gain_fuel_state ::
  "nat  $\Rightarrow$  'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  ('a  $\times$  ('a  $\Rightarrow$  real))"
where
  "lazy_argmax_gain_fuel_state 0 S A ub =
    (let x = pick_ub_some A ub in (x, ub))"
| "lazy_argmax_gain_fuel_state (Suc n) S A ub =
    (let x = pick_ub_some A ub in
     if ub x = gain S x then (x, ub)
     else lazy_argmax_gain_fuel_state n S A (tighten S ub x))"
```

```
definition lazy_select :: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('a  $\Rightarrow$  real)  $\Rightarrow$  ('a  $\times$  ('a
 $\Rightarrow$  real))" where
  "lazy_select S A ub = lazy_argmax_gain_fuel_state (card A) S A ub"
```

```
lemma lazy_argmax_gain_fuel_state_fst:
  "fst (lazy_argmax_gain_fuel_state n S A ub) = lazy_argmax_gain_fuel
n S A ub"
<proof>
```

```
lemma lazy_select_fst:
  "fst (lazy_select S A ub) = lazy_argmax_gain S A ub"
<proof>
```

```
lemma lazy_argmax_gain_fuel_state_ub_valid:
  assumes ubv: "ub_valid S A ub"
  shows "ub_valid S A (snd (lazy_argmax_gain_fuel_state n S A ub))"
<proof>
```

```
lemma lazy_select_ub_valid:
  assumes ubv: "ub_valid S A ub"
  shows "ub_valid S A (snd (lazy_select S A ub))"
<proof>
```

3.3 Preservation of upper-bound validity across outer iterations

```
lemma ub_valid_init:
```

```
"ub_valid {} V init_ub"
⟨proof⟩
```

```
lemma ub_valid_after_insert:
  assumes ubv: "ub_valid S (V - S) ub"
    and Ssub: "S ⊆ V"
    and x_in: "x ∈ V - S"
  shows "ub_valid (insert x S) (V - insert x S) ub"
⟨proof⟩
```

3.4 One outer step and the full stateful algorithm

```
definition lazy_step :: "'a lg_state ⇒ 'a lg_state" where
  "lazy_step st =
    (let S = Sg st;
      A = remaining st;
      ub = ubg st
    in if A = {} then st
      else
        (let p = lazy_select S A ub;
          x = fst p;
          ub' = snd p
        in ( Sg = insert x S, ubg = ub' )))"
```

```
lemma lazy_step_nonempty_Sg:
  assumes rem_ne: "remaining st ≠ {}"
  shows
    "Sg (lazy_step st) =
      insert (fst (lazy_select (Sg st) (remaining st) (ubg st))) (Sg st)"
⟨proof⟩
```

```
lemma lazy_step_nonempty_ubg:
  assumes rem_ne: "remaining st ≠ {}"
  defines "p ≡ lazy_select (Sg st) (remaining st) (ubg st)"
  shows "ubg (lazy_step st) = snd p"
⟨proof⟩
```

```
fun lazy_state :: "nat ⇒ 'a lg_state" where
  "lazy_state 0 = init_state"
| "lazy_state (Suc i) = lazy_step (lazy_state i)"
```

```
definition lazy_set :: "nat ⇒ 'a set" where
  "lazy_set i = Sg (lazy_state i)"
```

3.5 Main invariants: subset property and validity on the remaining set

```
lemma lazy_step_idle:
  assumes "remaining st = {}"
```

```

shows "lazy_step st = st"
⟨proof⟩

lemma lazy_state_subset_V:
  "Sg (lazy_state i) ⊆ V"
⟨proof⟩

lemma lazy_state_ub_valid:
  "ub_valid (Sg (lazy_state i)) (remaining (lazy_state i)) (ubg (lazy_state i))"
⟨proof⟩

```

3.6 Correctness of the lazy greedy step

```

lemma lazy_step_is_argmax:
  assumes rem_ne: "remaining st ≠ {}"
    and ubv: "ub_valid (Sg st) (remaining st) (ubg st)"
    and finA: "finite (remaining st)"
  defines "p ≡ lazy_select (Sg st) (remaining st) (ubg st)"
  defines "x ≡ fst p"
  shows "∀ y ∈ remaining st. gain (Sg st) y ≤ gain (Sg st) x"
⟨proof⟩

end

end

theory Greedy_Step_Spec
  imports
    "../Algorithms/Greedy_Submodular_Construct"
begin

```

Step-specification interface for greedy-style algorithms.

The main construction locale is *Greedy_Setup*. The following locale is an intentionally thin named view of this setup, using the name *select* for an oracle that chooses a maximum-marginal-gain element from every finite non-empty candidate set. This keeps later corollaries independent of the concrete choice-based oracle used for the basic greedy construction.

```

locale Greedy_Step_Oracle =
  Greedy_Setup V f k select
  for V :: "'a set"
    and f :: "'a set ⇒ real"
    and k :: nat
    and select :: "'a set ⇒ 'a set ⇒ 'a"
begin

end

end

```

```

theory Greedy_Submodular_Approx
  imports
    Greedy_Step_Spec
begin

```

We first derive analytic bounds for the coefficient $1 - (1 - 1/k)^k$ appearing in the Nemhauser–Wolsey approximation ratio. These bounds are later combined with the greedy gap recurrence to obtain the standard $1 - 1/\exp 1$ guarantee.

First, we relate the finite quantity $(1 - 1/k)^k$ to the exponential function via a standard exponential inequality.

```

lemma pow_one_minus_inv_le_exp_neg1:
  fixes k :: nat
  assumes "k ≥ 1"
  shows "(1 - 1 / real k) ^ k ≤ exp (-1 :: real)"
  <proof>

```

As a corollary, we obtain a uniform lower bound $1 - (1 - 1/k)^k \geq 1 - 1/\exp 1$ for all $k \geq 1$.

```

lemma coeff_ge_1_minus_inv_exp:
  fixes k :: nat
  assumes "k ≥ 1"
  shows "1 - (1 - 1 / real k) ^ k ≥ 1 - 1 / exp 1"
  <proof>

```

```

context Greedy_Setup
begin

```

4 Greedy approximation for monotone submodular maximization

In this section we formalize the classical Nemhauser–Wolsey guarantee: for a non-negative, monotone, submodular function on a finite ground set, the greedy algorithm that selects k elements achieves at least $1 - (1 - 1/k)^k$ times the optimal value. Combining this with the analytic bound above yields the familiar $1 - 1/e$ approximation factor.

Elementary algebraic identity used in the gap recurrence.

```

lemma one_minus_inv_times:
  fixes x :: real
  shows "(1 - 1 / real k) * x = x - x / real k"
  <proof>

```

4.1 Greedy gap analysis

We use the problem-level optimal value and the reusable marginal-gain averaging lemma from the base cardinality context to derive the greedy gap recurrence.

4.1.1 Gap sequence

We introduce the gap sequence $\text{gap } i = \text{OPT}_k - f(\text{greedy_set } i)$ and show that it satisfies a simple linear recurrence under the greedy update.

definition $\text{gap} :: \text{"nat"} \Rightarrow \text{"real"}$ where
"gap $i = \text{OPT}_k - f(\text{greedy_set } i)$ "

One-step inequality: the marginal gain of the greedy choice lower-bounds the average improvement towards OPT_k .

lemma greedy_step_ineq :
assumes "i < k"
and S_{sub} : " $\text{greedy_set } i \subseteq V$ "
and R_{nonempty} : " $V - \text{greedy_set } i \neq \{\}$ "
shows " $\text{gain } (\text{greedy_set } i)$
 $(\text{argmax_gain } (\text{greedy_set } i) (V - \text{greedy_set } i))$
 $\geq (\text{OPT}_k - f(\text{greedy_set } i)) / \text{real } k$ "
(proof)

Greedy sets are feasible whenever their size is at most k .

lemma $\text{greedy_set_feasible}$:
assumes S_{sub} : " $\text{greedy_set } i \subseteq V$ "
and card_le_i : " $\text{card } (\text{greedy_set } i) \leq i$ "
and $i_{\text{le_k}}$: " $i \leq k$ "
shows " $\text{feasible } (\text{greedy_set } i)$ "
(proof)

corollary greedy_feasible :
assumes " $i \leq k$ "
shows " $\text{feasible } (\text{greedy_set } i)$ "
(proof)

The gap is non-negative along the greedy sequence.

lemma gap_nonneg :
assumes S_{sub} : " $\text{greedy_set } i \subseteq V$ "
and card_le_i : " $\text{card } (\text{greedy_set } i) \leq i$ "
and $i_{\text{le_k}}$: " $i \leq k$ "
shows " $0 \leq \text{gap } i$ "
(proof)

corollary greedy_gap_nonneg :
assumes " $i \leq k$ "
shows " $0 \leq \text{gap } i$ "

<proof>

corollary *greedy_remainder_nonempty*:

assumes " $i < k$ "

shows " $V - \text{greedy_set } i \neq \{\}$ "

<proof>

Gap recurrence: each step reduces the gap by at least a $1/k$ fraction.

lemma *gap_step_diff*:

assumes " $i < k$ "

and S_{sub} : " $\text{greedy_set } i \subseteq V$ "

and R_{nonempty} : " $V - \text{greedy_set } i \neq \{\}$ "

shows " $\text{gap } (\text{Suc } i) \leq \text{gap } i - \text{gap } i / \text{real } k$ "

<proof>

In multiplicative form, the gap shrinks by a factor of at most $1 - 1/\text{real } k$ per step.

lemma *gap_step*:

assumes " $i < k$ "

and " $\text{greedy_set } i \subseteq V$ "

and " $V - \text{greedy_set } i \neq \{\}$ "

shows " $\text{gap } (\text{Suc } i) \leq (1 - 1 / \text{real } k) * \text{gap } i$ "

<proof>

lemma *gap_0[simp]*: " $\text{gap } 0 = \text{OPT}_k$ "

<proof>

Geometric decay of the gap: after i greedy steps the remaining gap is bounded by $(1 - 1/\text{real } k)^i * \text{OPT}_k$.

lemma *gap_geometric*:

assumes k_{pos} : " $k > 0$ "

and i_{le_k} : " $i \leq k$ "

shows " $\text{gap } i \leq (1 - 1 / \text{real } k) ^ i * \text{OPT}_k$ "

<proof>

As a consequence, the value of the greedy solution after k steps is at least $(1 - (1 - 1/\text{real } k)^k) * \text{OPT}_k$.

lemma *greedy_sequence_bound*:

assumes k_{pos} : " $k > 0$ "

shows " $f (\text{greedy_set } k) \geq (1 - (1 - 1 / \text{real } k) ^ k) * \text{OPT}_k$ "

<proof>

4.2 Non-negativity of OPT and approximation ratio

Combining the discrete bound with the analytic inequality for $1 - (1 - 1/k)^k$ yields the standard $1 - 1/e$ approximation factor.

theorem *greedy_approximation*:

assumes k_{pos} : " $k > 0$ "

```

    shows "f (greedy_set k) ≥ (1 - 1 / exp 1) * OPT_k"
  <proof>

```

4.3 Corollaries

Define the approximation ratio of the greedy algorithm for a given k (with the convention that the ratio is 1 when $OPT_k = 0$), and show that it is always at least $1 - 1/\exp 1$.

```

definition greedy_ratio :: real where
  "greedy_ratio = (if OPT_k = 0 then 1 else f (greedy_set k) / OPT_k)"

```

```

corollary greedy_ratio_ge_1_minus_inv_exp:

```

```

  assumes "k > 0"

```

```

  shows "greedy_ratio ≥ 1 - 1 / exp 1"

```

```

  <proof>

```

```

end

```

5 Step-spec corollary

Since *Greedy_Step_Oracle* is a named instance of *Greedy_Setup*, the Nemhauser–Wolsey approximation guarantee transfers directly to any oracle satisfying the step-specification assumptions.

```

context Greedy_Step_Oracle
begin

```

```

theorem greedy_step_oracle_approximation:

```

```

  assumes "k > 0"

```

```

  shows

```

```

    "f (Greedy_Setup.greedy_set V select k)
     ≥ (1 - 1 / exp 1) * OPT_k"

```

```

  <proof>

```

```

end

```

```

end

```

```

theory Lazy_Greedy_Stateful_StepSpec

```

```

  imports "../Algorithms/Lazy_Greedy_Stateful"

```

```

begin

```

Sequence-level lemmas for the verified stateful lazy run.

This theory packages the per-iteration facts needed by the approximation proof, such as the membership and maximal-gain properties of the chosen lazy element at step i , together with the update equation for the next lazy set.

It is not formalized by instantiating the stateless greedy step-oracle locale. Instead, it exposes the corresponding properties of the concrete verified

```

run.

context Cardinality_Constraint
begin

abbreviation st_i :: "nat  $\Rightarrow$  'a lg_state" where
  "st_i i  $\equiv$  lazy_state i"

abbreviation S_i :: "nat  $\Rightarrow$  'a set" where
  "S_i i  $\equiv$  lazy_set i"

abbreviation A_i :: "nat  $\Rightarrow$  'a set" where
  "A_i i  $\equiv$  remaining (st_i i)"

lemma A_i_eq: "A_i i = V - S_i i"
  <proof>

definition lazy_choice :: "nat  $\Rightarrow$  'a" where
  "lazy_choice i = fst (lazy_select (S_i i) (A_i i) (ubg (st_i i)))"

lemma A_i_finite: "finite (A_i i)"
  <proof>

lemma lazy_choice_mem:
  assumes ne: "A_i i  $\neq$  {}"
  shows "lazy_choice i  $\in$  A_i i"
  <proof>

lemma lazy_choice_max:
  assumes ne: "A_i i  $\neq$  {}"
  shows " $\forall y \in A_i i. \text{gain } (S_i i) y \leq \text{gain } (S_i i) (\text{lazy\_choice } i)$ "
  <proof>

lemma lazy_set_Suc_insert:
  assumes ne: "A_i i  $\neq$  {}"
  shows "S_i (Suc i) = insert (lazy_choice i) (S_i i)"
  <proof>

lemma lazy_choice_in_V_minus_S:
  assumes "V - lazy_set i  $\neq$  {}"
  shows "lazy_choice i  $\in$  V - lazy_set i"
  <proof>

lemma lazy_choice_argmax_V_minus_S:
  assumes "V - lazy_set i  $\neq$  {}"
  shows " $\forall y \in V - \text{lazy\_set } i. \text{gain } (\text{lazy\_set } i) y \leq \text{gain } (\text{lazy\_set } i) (\text{lazy\_choice } i)$ "
  <proof>

lemma lazy_set_Suc_insert_V_minus_S:

```

```

    assumes "V - lazy_set i  $\neq$  {}"
    shows "lazy_set (Suc i) = insert (lazy_choice i) (lazy_set i)"
    <proof>

end
end
theory Lazy_Greedy_Stateful_Approx
  imports
    Greedy_Submodular_Approx
    Lazy_Greedy_Stateful_StepSpec
begin

  Approximation guarantee for the verified stateful lazy greedy construction.

  This theory treats the stateful lazy algorithm as an implementation-oriented variant of the greedy construction. It reuses the optimal-value infrastructure from the greedy approximation development, together with the per-iteration lemmas from the lazy step-spec theory, and proves a corresponding gap recurrence for the lazy construction.

  In particular, this theory does not instantiate the stateless step-spec locale. Instead, it works directly with the verified lazy run and its sequence-level properties.

  context Cardinality_Constraint
  begin

  definition gapL :: "nat  $\Rightarrow$  real" where
    "gapL i = OPT_k - f (lazy_set i)"

  lemma lazy_set_0[simp]: "lazy_set 0 = {}"
    <proof>

  lemma lazy_set_subset_V[simp]: "lazy_set i  $\subseteq$  V"
    <proof>

  lemma lazy_set_finite[simp]: "finite (lazy_set i)"
    <proof>

  lemma remaining_lazy_state[simp]: "remaining (lazy_state i) = V - lazy_set i"
    <proof>

  lemma card_lazy_le_i: "card (lazy_set i)  $\leq$  i"
    <proof>

  lemma card_lazy_lt_k:
    "i < k  $\implies$  card (lazy_set i) < k"
    <proof>

```

```

lemma lazy_remainder_nonempty:
  "i < k  $\implies$  V - lazy_set i  $\neq$  {}"
  <proof>

lemma lazy_set_feasible:
  assumes "i  $\leq$  k"
  shows "feasible (lazy_set i)"
  <proof>

lemma gapL_nonneg:
  assumes "i  $\leq$  k"
  shows "0  $\leq$  gapL i"
  <proof>

lemma lazy_step_ineq:
  "i < k  $\implies$  gain (lazy_set i) (lazy_choice i)  $\geq$  gapL i / real k"
  <proof>

lemma gapL_step:
  "i < k  $\implies$  gapL (Suc i)  $\leq$  (1 - 1 / real k) * gapL i"
  <proof>

lemma gapL_geometric:
  "k > 0  $\implies$  i  $\leq$  k  $\implies$  gapL i  $\leq$  (1 - 1 / real k) ^ i * OPT_k"
  <proof>

theorem lazy_stateful_approximation:
  assumes k_pos: "k > 0"
  shows "f (lazy_set k)  $\geq$  (1 - 1 / exp 1) * OPT_k"
  <proof>

end

```

6 Acknowledgements

The author is grateful to Wenda Li for careful reviews, comments, and guidance from the early stages of this project through the preparation of this AFP entry.

end

References

- [1] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978.

- [2] 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.