Current Status and Planned Next Steps

Repository. https://github.com/lyuf09/isabelle-submodular-greedy

1. Current Status of the Formalisation

The development is organised into a small collection of theory files with a modular locale structure. At a high level, the core proof of the greedy approximation guarantee is completed; the remaining work is focused on (i) reusable instantiations and (ii) lightweight executable experiments and documentation.

Submodular_Base

- Basic definitions for set functions on a finite ground set, including monotonicity and submodularity.
- General lemmas used as "submodular calculus" in later files.

Greedy_Submodular_Construct

- Established the locale `Greedy_Setup`, fixing a finite ground set V, a budget k, and a non-negative, monotone, submodular set function f with $f(\emptyset) = 0$.
- Defined the marginal gain $\text{gain}(S, e)$, the maximiser argmax_gain, and the greedy sequence

$$\text{greedy\_set} : \mathbb{N} \Rightarrow \text{"sets of items"}.$$

- Proved fundamental structural properties:
  - $\text{greedy\_set}(i) \subseteq V$ and every greedy set is finite.
  - Monotonicity: $\text{greedy\_set}(i) \subseteq \text{greedy\_set}(j)$ for $i \leq j$.
  - Cardinality bounds: $\text{card}(\text{greedy\_set}(i)) \leq i$ and $\text{card}(\text{greedy\_set}(i)) \leq \min(i, \text{card}(V))$.
  - A precise state-transition lemma describing the evolution of $\text{greedy\_set}(i+1)$ when the remainder $V - \text{greedy\_set}(i)$ is non-empty.
- Defined a list view of the greedy construction (`greedy_sequence`) and established indexing lemmas.

Greedy_Submodular_Approx

- Proved the analytic inequality for all $k \geq 1$:

$$(1 - 1/k)^k \leq e^{-1} \quad \text{and} \quad 1 - (1 - 1/k)^k \geq 1 - 1/e.$$

- Within the locale:
  - Established non-negativity of marginal gains and two basic non-emptiness lemmas.
  - Proved diminishing returns and the submodular telescoping inequality.
  - Proved the averaging lemma ensuring the existence of an element $e \in V \setminus S$ with

$$\text{gain}(S, e) \geq \frac{f(\text{Opt}) - f(S)}{k}.$$

  - Collected feasible sets $\mathcal{F}_k = \{S \subseteq V : |S| \leq k\}$, defined a canonical maximiser OPT_set, and set $\text{OPT}_k = f(\text{OPT\_set})$.
  - Introduced the gap sequence:

$$\text{gap}(i) = \text{OPT}_k - f(\text{greedy\_set}(i)),$$

    and showed the linear recurrence and geometric decay:

$$\text{gap}(i) \leq (1 - 1/k)^i \cdot \text{OPT}_k.$$

– Derived the Nemhauser–Wolsey guarantee:

$$f(\text{greedy\_set}(k)) \geq \left(1 - (1 - 1/k)^k\right) \cdot \text{OPT}_k \geq (1 - 1/e)\,\text{OPT}_k.$$

– Defined an approximation-ratio function and proved it is always at least $1 - 1/e$ (under the locale assumptions).

This completes the formal development of the classical greedy approximation bound for monotone submodular maximisation under a cardinality constraint.

2. Informal and Empirical Complexity Comparison

While a fully formal complexity analysis is possible (and may be desirable in the long term), an informal comparison already helps motivate the next experimental phase and clarifies why executable, verified code may appear slow on non-trivial instances.

Oracle-call perspective

A common abstraction is to measure complexity in terms of *value-oracle calls* to f (equivalently, evaluations of $\text{gain}(S, e) = f(S \cup \{e\}) - f(S)$).

- **Greedy (cardinality k).** At iteration i, the algorithm scans the remaining elements $e \in V \setminus \text{greedy\_set}(i)$ and selects one maximising $\text{gain}(\text{greedy\_set}(i), e)$. Hence the number of gain evaluations is bounded by

$$\sum_{i=0}^{k-1} |V \setminus \text{greedy\_set}(i)| \;\leq\; k\,|V|.$$

  Therefore, in oracle terms, greedy uses $O(k|V|)$ evaluations.

- **Exhaustive maximisation.** Enumerating all feasible sets $S \subseteq V$ with $|S| \leq k$ requires evaluating

$$\sum_{j=0}^{k} \binom{|V|}{j}$$

  candidates, which is already exponential in $|V|$ in the worst case (and approaches $2^{|V|}$ when k is not very small).

Why verified executable code can be slow

Even when the asymptotic oracle count is favourable, extracted Isabelle code is typically unoptimised: finite sets are represented via lists (or list-like structures), `argmax` requires full scans, and repeated set operations may incur noticeable overhead. Hence the goal of the next phase is not to compete with hand-tuned implementations, but to (i) validate correctness on small instances and (ii) observe qualitative trends (e.g. approximation ratios and growth of oracle-call counts) against baselines such as exhaustive maximisation.

Planned empirical reporting

For small instances (e.g. $|V| \leq 15$), I plan to report:

- the selected greedy set and its value $f(\text{greedy\_set}(k))$;
- the exact optimum value OPT_k from exhaustive search;
- the observed approximation ratio $f(\text{greedy\_set}(k))/\text{OPT}_k$;
- a simple operational count, such as the number of gain evaluations performed by greedy.

3. Planned Next Steps

At this stage, it seems natural to shift the focus toward applications, reusable instantiations, lightweight experiments, and polishing. The directions below are intended to be incremental: each should be independently useful even if later steps change.

A. Coverage Function Instantiation

- Introduce a locale `Coverage_Setup` with:
  - a finite universe U,
  - a finite ground set V,
  - a mapping $g : V \Rightarrow \mathcal{P}(U)$,
  - a cardinality budget k.
- Define the coverage function
$$f_{\text{cov}}(S) = \left| \bigcup_{x \in S} g(x) \right|.$$
- Prove f_{cov} satisfies the `Greedy_Setup` assumptions:
  - non-negativity and $f_{\text{cov}}(\emptyset) = 0$;
  - monotonicity under inclusion;
  - submodularity (via the standard union–intersection argument).
- Use `interpret` to instantiate the greedy theory with f_{cov}, and derive the specialised guarantee
$$f_{\text{cov}}(\text{greedy\_set}(k)) \geq (1 - 1/e)\, \text{OPT}_k.$$
- Add a short explanatory section summarising the set-cover model and the meaning of this instantiation.

B. Baseline: Exhaustive Search (Exact Optimum on Small Instances)

- Define an exhaustive maximiser for small finite instances, e.g.
$$\text{enum\_opt}(V) = \arg \max_{S \subseteq V,\ |S| \leq k} f(S).$$
- Prove this construction yields a feasible maximiser with value OPT_k, using finiteness of the feasible family.
- Use this as a baseline for executable experiments (Section 2), and to empirically illustrate the approximation guarantee on small instances.

C. Tightness and Counterexamples (Clarifying Assumptions)

The classical $(1 - 1/e)$ bound is *guaranteed* under the locale assumptions (non-negative, monotone, submodular, $f(\emptyset) = 0$, and cardinality constraint). Therefore, a counterexample *below* $1 - 1/e$ cannot exist within the same assumption set. The goal here is to clarify (i) tightness and (ii) necessity of assumptions:

- **Tightness family (within assumptions).** Formalise a small instance (or a small member of a standard tightness family) where the greedy ratio is close to $1 - 1/e$, and document the observed ratio for increasing k on these instances.
- **Assumption-violation counterexamples.** Construct small functions violating monotonicity and/or submodularity and show that the greedy ratio can fall below $1 - 1/e$ on those instances. This would support a lemma-style statement: *dropping submodularity (or monotonicity) invalidates the guarantee.*

D. Executable Code and Small Experiments

- Experiment with Isabelle's `export_code` to generate executable code for `greedy_set` and the coverage instantiation.
- Evaluate greedy on a small suite of instances and record the outputs and operational counts, following the reporting items in Section 2.
- Where feasible, compare against alternative implementations (e.g. an exhaustive maximiser) to highlight trends rather than performance.

E. Documentation and Polishing

- Gradually refine the structure and explanatory text within the theories to make the development easier to follow for future users.
- Add a high-level roadmap of the logical flow:

$$\begin{aligned} &\text{construction} \\ \to\ &\text{submodular calculus} \\ \to\ &\text{averaging lemma} \\ \to\ &\text{gap recurrence} \\ \to\ &\text{approximation theorem} \\ \to\ &\text{instantiations and experiments.} \end{aligned}$$

- Normalise naming conventions and clean up unused auxiliary lemmas, with an eye toward a workshop-style report or a more polished journal version.

Longer-term outlook. In the longer term, these components could grow into a small reusable Isabelle library for submodular optimisation, and the executable examples could serve as a compact demonstration of verified algorithms in optimisation.