Derandomization with Conditional Expectations

Emin Karayel

June 20, 2024

Abstract

The *Method of Conditional Expectations* [4] (sometimes also called "Method of Conditional Probabilities") is one of the prominent derandomization techniques. Given a randomized algorithm, it allows the construction of a deterministic algorithm with a result that matches the average-case quality of the randomized algorithm.

Using this technique, this entry starts with a simple example, an algorithm that obtains a cut that crosses at least half of the edges. This is a well-known approximate solution to the Max-Cut problem. It is followed by a more complex and interesting result: an algorithm that returns an independent set matching (or exceeding) the Caro-Wei bound [3]: $\frac{n}{d+1}$ where n is the vertex count and d is the average degree of the graph.

Both algorithms are efficient and deterministic, and follow from the derandomization of a probabilistic existence proof.

Contents

1 Some Preliminary Results

theory *Derandomization-Conditional-Expectations-Preliminary*
imports
HOL−Combinatorics.Multiset-Permutations
Universal-Hash-Families.Pseudorandom-Objects
Undirected-Graph-Theory.Undirected-Graphs-Root
begin

1.1 On Probability Theory

lemma *map-pmf-of-set-bij-betw-2*:
assumes *bij-betw* $(\lambda x.\ (f\ x,\ g\ x))\ A\ (B \times C)$ $A \neq \{\}$ *finite* A
shows *map-pmf* f $(\textit{pmf-of-set}\ A) = \textit{pmf-of-set}\ B$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *integral-bind-pmf*:
fixes f :: - ⇒ *real*
assumes $\bigwedge x.\ x \in \textit{set-pmf}\ (\textit{bind-pmf}\ p\ q) \Longrightarrow |f\ x| \leq M$
shows $(\int x.\ f\ x\ \partial \textit{bind-pmf}\ p\ q) = (\int x.\ \int y.\ f\ y\ \partial q\ x\ \partial p)$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *pmf-of-set-un*:
fixes $A\ B$ :: *'x set*
assumes $A \cup B \neq \{\}$ $A \cap B = \{\}$ *finite* $(A \cup B)$
defines $p \equiv \textit{real}\ (\textit{card}\ A)\ /\ \textit{real}\ (\textit{card}\ A + \textit{card}\ B)$
shows *pmf-of-set* $(A \cup B) = \textit{do}\ \{c \leftarrow \textit{bernoulli-pmf}\ p;\ \textit{pmf-of-set}\ (\textit{if}\ c\ \textit{then}\ A\ \textit{else}\ B)\}$
(**is** *?L* = *?R*)
⟨*proof*⟩

If the expectation of a discrete random variable is larger or equal to c, there will be at least one point at which the random variable is larger or equal to c.

lemma *exists-point-above-expectation*:
assumes *integrable* $(\textit{measure-pmf}\ M)\ f$
assumes $\textit{measure-pmf.expectation}\ M\ f \geq (c::\textit{real})$
shows $\exists x \in \textit{set-pmf}\ M.\ f\ x \geq c$
⟨*proof*⟩

1.2 On Convexity

A translation rule for convexity.

lemma *convex-on-shift*:
fixes f :: $('b :: \textit{real-vector}) \Rightarrow \textit{real}$
assumes *convex-on* $S\ f$ *convex* S
shows *convex-on* $\{x.\ x + a \in S\}$ $(\lambda x.\ f\ (x+a))$
⟨*proof*⟩

1.3 On *subseq* and *strict-subseq*

lemma *strict-subseq-imp-shorter*: $\textit{strict-subseq}\ x\ y \Longrightarrow \textit{length}\ x < \textit{length}\ y$
⟨*proof*⟩

lemma *subseq-distinct*: $\textit{subseq}\ x\ y \Longrightarrow \textit{distinct}\ y \Longrightarrow \textit{distinct}\ x$
⟨*proof*⟩

lemma *strict-subseq-imp-distinct*: $\textit{strict-subseq}\ x\ y \Longrightarrow \textit{distinct}\ y \Longrightarrow \textit{distinct}\ x$
⟨*proof*⟩

lemma *subseq-set*: *subseq xs ys* $\Longrightarrow$ *set xs* $\subseteq$ *set ys*
⟨*proof*⟩

lemma *strict-subseq-set*: *strict-subseq x y* $\Longrightarrow$ *set x* $\subseteq$ *set y*
⟨*proof*⟩

lemma *subseq-induct*:
assumes $\bigwedge ys.\ (\bigwedge zs.\ \textit{strict-subseq zs ys} \Longrightarrow P\ zs) \Longrightarrow P\ ys$
shows *P xs*
⟨*proof*⟩

lemma *subseq-induct'*:
assumes *P* []
assumes $\bigwedge y\ ys.\ (\bigwedge zs.\ \textit{strict-subseq zs } (y\#ys) \Longrightarrow P\ zs) \Longrightarrow P\ (y\#ys)$
shows *P xs*
⟨*proof*⟩

lemma *strict-subseq-remove1*:
assumes *w* $\in$ *set x*
shows *strict-subseq* (*remove1 w x*) *x*
⟨*proof*⟩

1.4 On Random Permutations

lemma *filter-permutations-of-set-pmf*:
assumes *finite S*
shows *map-pmf* (*filter P*) (*pmf-of-set* (*permutations-of-set S*)) =
pmf-of-set (*permutations-of-set* {*x* $\in$ *S*. *P x*}) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *permutations-of-set-prefix*:
assumes *finite S v* $\in$ *S*
shows *measure* (*pmf-of-set* (*permutations-of-set S*)) {*xs*. *prefix* [*v*] *xs*} = *1*/*real* (*card S*)
(**is** *?L* = *?R*)
⟨*proof*⟩

cond-perm returns all permutations of a set starting with specific prefix.

definition *cond-perm* **where** *cond-perm V p* = (@) *p* ‘ *permutations-of-set* (*V* − *set p*)

context *fin-sgraph*
begin

lemma *perm-non-empty-finite*:
permutations-of-set V ≠ {} *finite* (*permutations-of-set V*)
⟨*proof*⟩

lemma *cond-perm-non-empty-finite*:
cond-perm V p ≠ {} *finite* (*cond-perm V p*)
⟨*proof*⟩

lemma *cond-perm-alt*:
assumes *distinct p set p* $\subseteq$ *V*
shows *cond-perm V p* = {*xs* $\in$ *permutations-of-set V*. *prefix p xs*}
⟨*proof*⟩

lemma *cond-permD*:
assumes *distinct p set p* $\subseteq$ *V xs* $\in$ *cond-perm V p*
shows *distinct xs set xs* = *V*

⟨*proof*⟩

1.5 On Finite Simple Graphs

lemma *degree-sum*: $(\sum v \in V.\ degree\ v) = 2 * card\ E$ (**is** *?L* = *?R*)
⟨*proof*⟩

The environment of a set of nodes is the union of it with its neighborhood.

definition *environment* **where** $environment\ S = S \cup \{v.\ \exists s \in S.\ vert\text{-}adj\ v\ s\}$

lemma *finite-environment*:
assumes *finite S*
shows *finite* (*environment S*)
⟨*proof*⟩

lemma *environment-mono*: $S \subseteq T \Longrightarrow environment\ S \subseteq environment\ T$
⟨*proof*⟩

lemma *environment-sym*: $x \in environment\ \{y\} \longleftrightarrow y \in environment\ \{x\}$
⟨*proof*⟩

lemma *environment-self*: $S \subseteq environment\ S$ ⟨*proof*⟩

lemma *environment-sym-2*: $A \cap environment\ B = \{\} \longleftrightarrow B \cap environment\ A = \{\}$
⟨*proof*⟩

lemma *environment-range*: $S \subseteq V \Longrightarrow environment\ S \subseteq V$
⟨*proof*⟩

lemma *environment-union*: $environment\ (S \cup T) = environment\ S \cup environment\ T$
⟨*proof*⟩

lemma *card-environment*: $card\ (environment\ \{v\}) = 1 + degree\ v$ (**is** *?L* = *?R*)
⟨*proof*⟩

end

end

2 Method of Conditional Expectations: Large Cuts

The following is an example of the application of the method of conditional expectations [2, 1] to construct an approximation algorithm that finds a cut of an undirected graph cutting at least half of the edges. This is also the example that Vadhan [4, Section 3.4.2] uses to introduce the "Method of Conditional Expectations".

theory *Derandomization-Conditional-Expectations-Cut*
imports *Derandomization-Conditional-Expectations-Preliminary*
begin

context *fin-sgraph*
begin

definition *cut-size* **where** $cut\text{-}size\ C = card\ \{e \in E.\ e \cap C \neq \{\} \wedge e - C \neq \{\}\}$

lemma *eval-cond-edge*:
assumes $L \subseteq U$ *finite U* $e \in E$

shows ($\int C.$ *of-bool* ($e \cap C \neq \{\} \wedge e - C \neq \{\}$) *∂pmf-of-set* $\{C.\ L \subseteq C \wedge C \subseteq U\}$) =
((*if* $e \subseteq -U \cup L$ *then of-bool*($e \cap L \neq \{\} \wedge e \cap -U \neq \{\}$)::*real else 1/2*))
(**is** *?L* = *?R*)
⟨*proof*⟩

If every vertex is selected independently with probability $\frac{1}{2}$ into the cut, it is easy to deduce that an edge will be cut with probability $\frac{1}{2}$ as well. Thus the expected cut size will be *real graph-size / 2.*

lemma *exp-cut-size*:
($\int C.$ *real* (*cut-size C*) *∂pmf-of-set* (*Pow V*)) = *real* (*card E*) / *2* (**is** *?L* = *?R*)
⟨*proof*⟩

For the above it is easy to show that there exists a cut, cutting at least half of the edges.

lemma *exists-cut*: $\exists C \subseteq V.$ *real* (*cut-size C*) ≥ *card E*/*2*
⟨*proof*⟩

end

However the above is just an existence proof, but it doesn't provide a method to construct such a cut efficiently. Here, we can apply the method of conditional expectations.

This works because, we can not only compute the expectation of the number of cut edges, when all vertices are chosen at random, but also conditional expectations, when some of the edges are fixed. The idea of the algorithm, is to choose the assignment of vertices into the cut based on which option maximizes the conditional expectation. The latter can be done incrementally for each vertex.

This results in the following efficient algorithm:

fun *derandomized-max-cut* :: *'a list* ⇒ *'a set* ⇒ *'a set* ⇒ *'a set set* ⇒ *'a set* **where**
derandomized-max-cut [] *R - - = R* |
derandomized-max-cut (*v*#*vs*) *R B E* =
(*if card* $\{e \in E.\ v \in e \wedge e \cap R \neq \{\}\} \geq$ *card* $\{e \in E.\ v \in e \wedge e \cap B \neq \{\}\}$ *then*
derandomized-max-cut vs R ($B \cup \{v\}$) *E*
else
derandomized-max-cut vs ($R \cup \{v\}$) *B E*
)

context *fin-sgraph*
begin

The term *cond-exp* is the conditional expectation, when some of the edges are selected into the cut, and some are selected to be outside the cut, while the remaining vertices are chosen randomly.

definition *cond-exp* **where** *cond-exp R B* = ($\int C.$ *real* (*cut-size C*) *∂pmf-of-set* $\{C.\ R \subseteq C \wedge C \subseteq V - B\}$)

The following is the crucial property of conditional expectations, the average of choosing a vertex in/out is the same as not fixing that vertex. This means that at least one choice will not decrease the conditional expectation.

lemma *cond-exp-split*:
assumes $R \subseteq V$ $B \subseteq V$ $R \cap B = \{\}$ $v \in V - R - B$
shows *cond-exp R B* = (*cond-exp* ($R \cup \{v\}$) *B* + *cond-exp R* ($B \cup \{v\}$))/*2* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *cond-exp-cut-size*:
assumes $R \subseteq V$ $B \subseteq V$ $R \cap B = \{\}$

shows *cond-exp R B* = *real* (*card* {*e*∈*E*. *e*∩*R*≠{}∧*e*∩*B*≠{}}) + *real* (*card* {*e*∈*E*. *e*∩*V*−*R*−*B*≠{}})
/ 2
(**is** *?L* = *?R*)
⟨*proof*⟩

Indeed the algorithm returns a cut with the promised approximation guarantee.

theorem *derandomized-max-cut*:
assumes *vs* ∈ *permutations-of-set V*
defines *C* ≡ *derandomized-max-cut vs* {} {} *E*
shows *C* ⊆ *V* 2 ∗ *cut-size C* ≥ *card E*
⟨*proof*⟩

end

end

3 Method of Pessimistic Estimators: Independent Sets

A generalization of the the method of conditional expectations is the method of pessimistic estimators. Where the conditional expectations are conservatively approximated. The following example is such a case.

Starting with a probabilistic proof of Caro-Wei's theorem [1, Section: The Probabilistic Lens: Turán's theorem], this section constructs a deterministic algorithm that finds such a set.

theory *Derandomization-Conditional-Expectations-Independent-Set*
imports *Derandomization-Conditional-Expectations-Cut*
begin

hide-fact (**open**) *Henstock-Kurzweil-Integration.integral-sum*

The following represents a greedy algorithm that walks through the vertices in a given order and adds it to a result set, if and only if it preserves independence of the set.

fun *indep-set* :: *'a list* ⇒ *'a set set* ⇒ *'a list*
where
indep-set [] *E* = [] |
indep-set (*v*#*vt*) *E* = *v*#*indep-set* (*filter* (λ*w*. {*v*,*w*} ∉ *E*) *vt*) *E*

context *fin-sgraph*
begin

lemma *indep-set-range*: *subseq* (*indep-set p E*) *p*
⟨*proof*⟩

lemma *is-independent-set-insert*:
assumes *is-independent-set A x* ∈ *V* − *environment A*
shows *is-independent-set* (*insert x A*)
⟨*proof*⟩

Correctness properties of *indep-set*:

theorem *indep-set-correct*:
assumes *distinct p set p* ⊆ *V*
shows *distinct* (*indep-set p E*) *set* (*indep-set p E*) ⊆ *V is-independent-set* (*set* (*indep-set p E*))
⟨*proof*⟩

While for an individual call of *indep-set* it is not possible to derive a non-trivial bound on the size of the resulting independent set, it is possible to estimate its performance on

average, i.e., with respect to a random choice on the order it visits the vertices. This will be derived in the following:

definition *is-first* **where**
  is-first v p = *prefix* [*v*] (*filter* (λ*y*. *y* ∈ *environment* {*v*}) *p*)

lemma *is-first-subseq*:
  assumes *is-first v p distinct p subseq q p v* ∈ *set q*
  shows *is-first v q*
⟨*proof*⟩

lemma *is-first-imp-in-set*:
  assumes *is-first v p*
  shows *v* ∈ *set p*
⟨*proof*⟩

Let us observe that a node, which comes first in the ordering of the vertices with respect to its neighbors, will definitely be in the independent set. (This is only a sufficient condition, but not a necessary condition.)

lemma *set-indep-set*:
  assumes *distinct p set p* ⊆ *V is-first v p*
  shows *v* ∈ *set* (*indep-set p E*)
  ⟨*proof*⟩

Using the above we can establish the following lower-bound on the expected size of an independent set obtained by *indep-set*:

theorem *exp-indep-set*:
  defines Ω ≡ *pmf-of-set* (*permutations-of-set V*)
  shows (∫ *vs*. *real* (*length* (*indep-set vs E*)) ∂Ω) ≥ (∑ *v* ∈ *V*. *1* / (*degree v* + *1*::*real*))
    (**is** *?L* ≥ *?R*)
⟨*proof*⟩

The function λ*x*. *1* / (*x* + *1*) is convex.

lemma *inverse-x-plus-1-convex*: *convex-on* {−*1*<..} (λ*x*. *1* / (*x*+*1*::*real*))
⟨*proof*⟩

lemma *caro-wei-aux*: *card V* / (*2*∗*card E* / *card V* + *1*) ≤ (∑ *v* ∈ *V*. *1*/ (*degree v*+*1*))
⟨*proof*⟩

A corollary of the *exp-indep-set* is Caro-Wei's theorem:

corollary *caro-wei*:
  ∃ *S* ⊆ *V*. *is-independent-set S* ∧ *card S* ≥ *card V* / (*2*∗*card E* / *card V* + *1*)
⟨*proof*⟩

end

After establishing the above result, we may ask the question, whether there is a practical algorithm to find such a set. This is where the method of conditional expectations comes to stage.

We are tasked with finding an ordering of the vertices, for which the above algorithm would return an above-average independent set. This is possible, because we can compute the conditional expectation of

measure-pmf.*expectation* (*pmf-of-set* (*permutations-of-set V*)) (λ*vs*. ∑ *v*∈*V*. *of-bool* (*is-first v vs*))

when we restrict to permutations starting with a given prefix. The latter term is a pessimistic estimator for the size of the independent set for the given ordering (as discussed above.)

It then is possible to obtain a deterministic algorithm that obtains an ordering by incrementally choosing vertices, that maximize the conditional expectation.

The resulting algorithm looks as follows:

function *derandomized-indep-set* :: *'a list* ⇒ *'a list* ⇒ *'a set set* ⇒ *'a list*
where
derandomized-indep-set [] *p E* = *indep-set p E* |
derandomized-indep-set (*vh*#*vt*) *p E* = (
let node-deg = (λ*v*. *real* (*card* {*e* ∈ *E*. *v* ∈ *e*}));
is-indep = (λ*v*. *list-all* (λ*w*. {*v*,*w*} ∉ *E*) *p*);
env = (λ*v*. *filter is-indep* (*v*#*filter* (λ*w*. {*v*,*w*} ∈ *E*) (*vh*#*vt*)));
cost = (λ*v*. (∑ *w* ← *env v*. *1* /(*node-deg w*+*1*)) − *of-bool*(*is-indep v*));
w = *arg-min-list cost* (*vh*#*vt*)
in derandomized-indep-set (*remove1 w* (*vh*#*vt*)) (*p*@[*w*]) *E*)
⟨*proof*⟩

termination
⟨*proof*⟩

context *fin-sgraph*
begin

lemma *is-first-append-1*:
assumes *v* ∉ *environment* (*set p*)
shows *is-first v* (*p*@*q*) = *is-first v q*
⟨*proof*⟩

lemma *is-first-append-2*:
assumes *v* ∈ *environment* (*set p*)
shows *is-first v* (*p*@*q*) = *is-first v p*
⟨*proof*⟩

The conditional expectation of the pessimistic estimator for a given prefix of the ordering of the vertices.

definition *p-estimator* **where**
p-estimator p = (∫ *vs*. (∑ *v* ∈ *V*. *of-bool*(*is-first v vs*)) ∂*pmf-of-set* (*cond-perm V p*))

lemma *p-estimator-split*:
assumes *V* − *set p* ≠ {}
shows *p-estimator p* = (∑ *v*∈*V*−*set p*. *p-estimator* (*p*@[*v*])) / *real* (*card* (*V*−*set p*)) (**is** *?L* = *?R*)
⟨*proof*⟩

The fact that the pessimistic estimator can be computed efficiently is the reason we can apply this method:

lemma *p-estimator*:
assumes *distinct p set p* ⊆ *V*
defines *P* ≡ {*v*. *is-first v p*}
defines *R* ≡ *V* − *environment* (*set p*)
shows *p-estimator p* = *card P* + (∑ *v*∈*R*. *1*/(*degree v* +*1*::*real*))
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *p-estimator-step*:
assumes *distinct* (*p*@[*v*]) *set* (*p*@[*v*]) ⊆ *V*
shows *p-estimator* (*p*@[*v*]) − *p-estimator p* = *of-bool*(*environment* {*v*} ∩ *set p* = {})
− (∑ *w*∈*environment* {*v*}−*environment*(*set p*). *1* / (*degree w*+*1*::*real*))
⟨*proof*⟩

lemma *derandomized-indep-set-correct-aux*:
assumes *p1@p2* ∈ *permutations-of-set V*
shows *distinct* (*derandomized-indep-set p1 p2 E*) ∧
is-independent-set (*set* (*derandomized-indep-set p1 p2 E*))
⟨*proof*⟩

lemma *derandomized-indep-set-length-aux*:
assumes *p1@p2* ∈ *permutations-of-set V*
shows *length* (*derandomized-indep-set p1 p2 E*) ≥ *p-estimator p2*
⟨*proof*⟩

The main result of this section the algorithm *derandomized-indep-set* obtains an independent set meeting the Caro-Wei bound in polynomial time.

theorem *derandomized-indep-set*:
assumes *p* ∈ *permutations-of-set V*
shows
is-independent-set (*set* (*derandomized-indep-set p* [] *E*))
distinct (*derandomized-indep-set p* [] *E*)
length (*derandomized-indep-set p* [] *E*) ≥ (∑ *v* ∈ *V*. *1*/ (*degree v*+*1*))
length (*derandomized-indep-set p* [] *E*) ≥ *card V* / (*2*∗*card E* / *card V* + *1*)
⟨*proof*⟩

end

end

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. John Wiley & Sons, Ltd, 2000.

[2] S. Jukna. *Extremal Combinatorics: With Applications in Computer Science*, chapter Derandomization, pages 307–318. Springer, Berlin, Heidelberg, 2001.

[3] O. Murphy. Lower bounds on the stability number of graphs computed in terms of degrees. *Discrete Mathematics*, 90(2):207–211, 1991.

[4] S. P. Vadhan. Pseudorandomness. *Foundations and Trends®in Theoretical Computer Science*, 7(1-3):1–336, 2012.