Derandomization with Conditional Expectations

Emin Karayel

March 17, 2025

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* (λ*x*. (*f x*, *g x*)) *A* (*B*×*C*) *A* ≠ {} *finite A*
  shows *map-pmf f* (*pmf-of-set A*) = *pmf-of-set B* (**is** *?L* = *?R*)
proof −
  have *B* × *C* ≠ {} **using** *assms*(*1*,*2*) **unfolding** *bij-betw-def* **by** *auto*
  hence *0*: *B* ≠ {} *C* ≠ {} **by** *auto*
  have *finite* (*B* × *C*)
    unfolding *bij-betw-imp-surj-on*[*OF assms*(*1*), *symmetric*] **by** (*intro finite-imageI assms*(*3*))
  hence *1*: *finite B finite C*
    using *0 finite-cartesian-productD1 finite-cartesian-productD2* **by** *auto*

  have *?L* = *map-pmf fst* (*map-pmf* (λ*x*. (*f x*, *g x*)) (*pmf-of-set A*))
    unfolding *map-pmf-comp* **by** *simp*
  also have ... = *map-pmf fst* (*pmf-of-set* (*B* × *C*))
    by (*intro arg-cong2*[**where** *f*=*map-pmf*] *map-pmf-of-set-bij-betw assms refl*)
  also have ... = *pmf-of-set B*
    using *0 1* **by** (*subst pmf-of-set-prod-eq*) (*auto simp add*:*map-fst-pair-pmf*)
  finally show *?thesis* **by** *simp*
qed

lemma *integral-bind-pmf*:
  fixes *f* :: - ⇒ *real*
  assumes ⋀*x*. *x* ∈ *set-pmf* (*bind-pmf p q*) ⟹ |*f x*| ≤ *M*
  shows (∫ *x*. *f x* ∂*bind-pmf p q*) = (∫ *x*. ∫ *y*. *f y* ∂*q x* ∂*p*) (**is** *?L* = *?R*)
proof −
  define *clamp* **where** *clamp x* = (*if* |*x*| > *M then 0 else x*) **for** *x*

  obtain *x* **where** *x* ∈ *set-pmf* (*bind-pmf p q*) **using** *set-pmf-not-empty* **by** *fast*
  hence *M-ge-0*: *M* ≥ *0* **using** *assms* **by** *fastforce*

  have *a*:⋀*x y*. *x* ∈ *set-pmf p* ⟹ *y* ∈ *set-pmf* (*q x*) ⟹ ¬|*f y*| > *M*
    using *assms* **by** *fastforce*
  hence (∫ *x*. *f x* ∂*bind-pmf p q*) = (∫ *x*. *clamp* (*f x*) ∂*bind-pmf p q*)
    unfolding *clamp-def* **by** (*intro integral-cong-AE AE-pmfI*) *auto*
  also have ... = (∫ *x*. ∫ *y*. *clamp* (*f y*) ∂*q x* ∂*p*) **unfolding** *measure-pmf-bind*
    by (*subst integral-bind*[**where** *K*=*count-space UNIV* **and** *B′*=*1* **and** *B*=*M*])
      (*simp-all add*:*measure-subprob clamp-def M-ge-0*)
  also have ... = *?R* **unfolding** *clamp-def* **using** *a* **by** (*intro integral-cong-AE AE-pmfI*) *simp-all*
  finally show *?thesis* **by** *simp*
qed

lemma *pmf-of-set-un*:
  fixes *A B* :: ′*x set*
  assumes *A* ∪ *B* ≠ {} *A* ∩ *B* = {} *finite* (*A* ∪ *B*)
  defines *p* ≡ *real* (*card A*) / *real* (*card A* + *card B*)
  shows *pmf-of-set* (*A* ∪ *B*) = *do* {*c* ← *bernoulli-pmf p*; *pmf-of-set* (*if c then A else B*)}
    (**is** *?L* = *?R*)

proof (*rule pmf-eqI*)
  fix *x* :: *'x*
  have *p-range*: *0* ≤ *p p* ≤ *1* **unfolding** *p-def* **by** (*auto simp*: *divide-le-eq*)
  have *card A* + *card B* > *0* **using** *assms*(*1*,*2*,*3*) **by** *auto*
  hence *a*: *1*−*p* = *real* (*card B*) / *real* (*card A* + *card B*)
    unfolding *p-def* **by** (*auto simp*:*divide-simps*)
  have *b*: *of-bool* (*x* ∈ *T*) = *pmf* (*pmf-of-set T*) *x* ∗ *real* (*card T*) **if** *finite T* **for** *T*
    using *that* **by** (*cases T* ≠ {}) *auto*

  have *pmf ?L x* = *indicator* (*A* ∪ *B*) *x* / *card* (*A* ∪ *B*) **using** *assms* **by** *simp*
  also have ... = (*of-bool* (*x*∈*A*) + *of-bool* (*x*∈*B*)) /(*card A*+*card B*) **using** *assms*(*1*−*3*)
    by (*intro arg-cong2*[**where** *f*=(/)] *arg-cong*[**where** *f*=*real*] *card-Un-disjoint*) *auto*
  also have ... = (*pmf* (*pmf-of-set A*) *x* ∗ *card A* + *pmf* (*pmf-of-set B*) *x* ∗ *card B*) /(*card A*+*card B*)
    using *assms*(*3*) **by** (*intro arg-cong2*[**where** *f*=(/)] *arg-cong2*[**where** *f*=(+)] *refl b*) *auto*
  also have ... = *pmf* (*pmf-of-set A*) *x* ∗ *p* + *pmf* (*pmf-of-set B*) *x* ∗ (*1* − *p*)
    unfolding *a* **unfolding** *p-def* **by** (*simp add*:*divide-simps*)
  also have ... = *pmf ?R x* **using** *p-range* **by** (*simp add*:*pmf-bind*)
  finally show *pmf ?L x* = *pmf ?R x* **by** *simp*
qed

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* (*measure-pmf M*) *f*
  assumes *measure-pmf.expectation M f* ≥ (*c*::*real*)
  shows ∃ *x* ∈ *set-pmf M*. *f x* ≥ *c*
proof (*rule ccontr*)
  assume ¬ (∃ *x*∈*set-pmf M*. *c* ≤ *f x*)
  hence *AE x in M*. *f x* < *c* **by** (*intro AE-pmfI*) *auto*
  thus *False* **using** *measure-pmf.expectation-less*[*OF assms*(*1*)] *assms*(*2*) *not-less* **by** *auto*
qed

1.2 On Convexity

A translation rule for convexity.

lemma *convex-on-shift*:
  fixes *f* :: (*'b* :: *real-vector*) ⇒ *real*
  assumes *convex-on S f convex S*
  shows *convex-on* {*x*. *x* + *a* ∈ *S*} (λ*x*. *f* (*x*+*a*))
proof −
  have *f* (((*1* − *t*) $*_R$ *x* + *t* $*_R$ *y*) + *a*) ≤ (*1*−*t*) ∗ *f* (*x*+*a*) + *t* ∗ *f* (*y*+*a*) (**is** *?L* ≤ *?R*)
    if *0* < *t t* < *1 x* ∈ {*x*. *x* + *a* ∈ *S*} *y* ∈ {*x*. *x* + *a* ∈ *S*} **for** *x y t*
  proof −
    have *?L* = *f* ((*1*−*t*) $*_R$ (*x*+*a*) + *t* $*_R$ (*y*+*a*)) **by** (*simp add*:*algebra-simps*)
    also have ... ≤ (*1*−*t*) ∗ *f* (*x*+*a*) + *t* ∗ *f* (*y*+*a*) **using** *that* **by** (*intro convex-onD*[*OF assms*(*1*)]) *auto*
    finally show *?thesis* **by** *auto*
  qed
  moreover have {*x*. *x* + *a* ∈ *S*} = (λ*x*. *x* − *a*) ' *S* **by** (*auto simp*:*image-iff algebra-simps*)
  hence *convex* {*x*. *x* + *a* ∈ *S*} **using** *assms*(*2*) **by** *auto*
  ultimately show *?thesis* **using** *assms* **by** (*intro convex-onI*) *auto*
qed

1.3 On *subseq* and *strict-subseq*

lemma *strict-subseq-imp-shorter*: *strict-subseq x y* ⟹ *length x* < *length y*

unfolding *strict-subseq-def* **by** (*meson linorder-neqE-nat not-subseq-length subseq-same-length*)

lemma *subseq-distinct*: *subseq x y* $\Longrightarrow$ *distinct y* $\Longrightarrow$ *distinct x*
by (*metis distinct-nthsI subseq-conv-nths*)

lemma *strict-subseq-imp-distinct*: *strict-subseq x y* $\Longrightarrow$ *distinct y* $\Longrightarrow$ *distinct x*
using *subseq-distinct* **unfolding** *strict-subseq-def* **by** *auto*

lemma *subseq-set*: *subseq xs ys* $\Longrightarrow$ *set xs* $\subseteq$ *set ys*
unfolding *strict-subseq-def* **by** (*metis set-nths-subset subseq-conv-nths*)

lemma *strict-subseq-set*: *strict-subseq x y* $\Longrightarrow$ *set x* $\subseteq$ *set y*
unfolding *strict-subseq-def* **by** (*intro subseq-set*) *simp*

lemma *subseq-induct*:
assumes $\bigwedge$*ys*. ($\bigwedge$*zs*. *strict-subseq zs ys* $\Longrightarrow$ *P zs*) $\Longrightarrow$ *P ys*
shows *P xs*
proof (*induction length xs arbitrary*:*xs rule*: *nat-less-induct*)
case *1*
have *P ys* **if** *strict-subseq ys xs* **for** *ys*
proof −
have *length ys* < *length xs* **using** *strict-subseq-imp-shorter that* **by** *auto*
thus *P ys* **using** *1* **by** *simp*
qed
thus *?case* **using** *assms* **by** *blast*
qed

lemma *subseq-induct′*:
assumes *P* []
assumes $\bigwedge$*y ys*. ($\bigwedge$*zs*. *strict-subseq zs* (*y*#*ys*) $\Longrightarrow$ *P zs*) $\Longrightarrow$ *P* (*y*#*ys*)
shows *P xs*
proof (*induction xs rule*: *subseq-induct*)
case (*1 ys*)
show *?case*
proof (*cases ys*)
case *Nil* **thus** *?thesis* **using** *assms*(*1*) **by** *simp*
next
case (*Cons ysh yst*)
show *?thesis* **using** *1* **unfolding** *Cons* **by** (*rule assms*(*2*)) *auto*
qed
qed

lemma *strict-subseq-remove1*:
assumes *w* $\in$ *set x*
shows *strict-subseq* (*remove1 w x*) *x*
proof −
have *subseq* (*remove1 w x*) *x* **by** (*induction x*) *auto*
moreover have *remove1 w x* $\neq$ *x* **using** *assms* **by** (*simp add*: *remove1-split*)
ultimately show *?thesis* **unfolding** *strict-subseq-def* **by** *auto*
qed

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 −

have *?L = map-pmf fst (map-pmf (partition P) (pmf-of-set (permutations-of-set S)))*
by *(simp add:map-pmf-comp)*
also have *... = map-pmf fst (pair-pmf ?R (pmf-of-set (permutations-of-set {x ∈ S. ¬ P x})))*
by *(simp add:partition-random-permutations[OF assms(1)])*
also have *... = ?R* **by** *(simp add:map-fst-pair-pmf)*
finally show *?thesis* **by** *simp*
qed

lemma *permutations-of-set-prefix*:
assumes *finite S v ∈ S*
shows *measure (pmf-of-set (permutations-of-set S)) {xs. prefix [v] xs} = 1/real (card S)*
(**is** *?L = ?R*)
proof −
have *S-ne*: *S ≠ {}* **using** *assms(2)* **by** *auto*
have *?L = (∫ vs. indicator {vs. prefix [v] vs} vs ∂pmf-of-set (permutations-of-set S))* **by** *simp*
also have *... = (∫ h. of-bool (v = h) ∂pmf-of-set S)*
unfolding *random-permutation-of-set[OF assms(1) S-ne]*
apply *(subst integral-bind-pmf[***where** *M=1], simp)*
apply *(subst integral-bind-pmf[***where** *M=1], simp)*
by *(simp add:indicator-def)*
also have *... = (∫ h. indicator {v} h ∂pmf-of-set S)* **by** *(simp add:indicator-def eq-commute)*
also have *... = measure (pmf-of-set S) {v}* **by** *simp*
also have *... = 1/card S* **using** *assms(1,2) S-ne* **by** *(subst measure-pmf-of-set) auto*
finally show *?thesis* **by** *simp*
qed

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 −
have *0 < card (permutations-of-set V)* **using** *finV* **by** *(subst card-permutations-of-set) auto*
thus *permutations-of-set V ≠ {} finite (permutations-of-set V)* **using** *card-gt-0-iff* **by** *blast+*
qed

lemma *cond-perm-non-empty-finite*:
cond-perm V p ≠ {} finite (cond-perm V p)
proof −
have *0 < card (permutations-of-set (V − set p))*
using *finV* **by** *(subst card-permutations-of-set) auto*
also have *... = card (cond-perm V p)*
unfolding *cond-perm-def* **by** *(intro card-image[symmetric] inj-onI) auto*
finally have *card (cond-perm V p) > 0* **by** *simp*
thus *cond-perm V p ≠ {} finite (cond-perm V p)* **using** *card-ge-0-finite* **by** *auto*
qed

lemma *cond-perm-alt*:
assumes *distinct p set p ⊆ V*
shows *cond-perm V p = {xs ∈ permutations-of-set V. prefix p xs}*
proof −
have *p@xs ∈ permutations-of-set V* **if** *xs ∈ permutations-of-set (V−set p)* **for** *xs*
using *permutations-of-setD[OF that] assms* **by** *(intro permutations-of-setI) auto*
moreover have *xs ∈ cond-perm V p* **if** *xs ∈ permutations-of-set V* **and** *a:prefix p xs* **for** *xs*
proof −

obtain *ys* **where** *xs-def*:*xs* = *p*@*ys* **using** *a prefixE* **by** *auto*
have *0*:*distinct* (*p*@*ys*) *set* (*p*@*ys*) = *V*
using *permutations-of-setD*[*OF that*(*1*)] **unfolding** *xs-def* **by** *auto*
hence *set ys* = *V* − *set p* **by** *auto*
moreover have *distinct ys* **using** *0* **by** *auto*
ultimately have *ys* ∈ *permutations-of-set* (*V* − *set p*) **by** (*intro permutations-of-setI*)
thus *?thesis* **unfolding** *cond-perm-def xs-def* **by** *simp*
qed
ultimately show *?thesis* **by** (*auto simp*:*cond-perm-def*)
qed

lemma *cond-permD*:
assumes *distinct p set p* ⊆ *V xs* ∈ *cond-perm V p*
shows *distinct xs set xs* = *V*
using *assms*(*3*) *permutations-of-setD* **unfolding** *cond-perm-alt*[*OF assms*(*1*,*2*)] **by** *auto*

1.5 On Finite Simple Graphs

lemma *degree-sum*: ($\sum$ *v* ∈ *V*. *degree v*) = *2* ∗ *card E* (**is** *?L* = *?R*)
proof −
have *?L* = ($\sum$ *v* ∈ *V*. ($\sum$ *e* ∈ *E*. *of-bool*(*v* ∈ *e*)))
using *fin-edges finV* **unfolding** *alt-degree-def incident-edges-def vincident-def*
by (*simp add*:*of-bool-def sum.If-cases Int-def*)
also have ... = ($\sum$ *e* ∈ *E*. *card* (*e* ∩ *V*))
using *fin-edges finV* **by** (*subst sum.swap*) (*simp add*:*of-bool-def sum.If-cases Int-commute*)
also have ... = ($\sum$ *e* ∈ *E*. *card e*)
using *wellformed* **by** (*intro sum.cong arg-cong*[**where** *f*=*card*] *Int-absorb2*) *auto*
also have ... = *2*∗*card E* **using** *two-edges* **by** *simp*
finally show *?thesis* **by** *simp*
qed

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

definition *environment* **where** *environment S* = *S* ∪ {*v*. ∃ *s* ∈ *S*. *vert-adj v s*}

lemma *finite-environment*:
assumes *finite S*
shows *finite* (*environment S*)
proof −
have *environment S* ⊆ *S* ∪ *V* **unfolding** *environment-def* **using** *vert-adj-imp-inV* **by** *auto*
thus *?thesis* **using** *assms finite-Un finV finite-subset* **by** *auto*
qed

lemma *environment-mono*: *S* ⊆ *T* ⟹ *environment S* ⊆ *environment T*
unfolding *environment-def* **by** *auto*

lemma *environment-sym*: *x* ∈ *environment* {*y*} ⟷ *y* ∈ *environment* {*x*}
unfolding *environment-def vert-adj-def* **by** (*auto simp*:*insert-commute*)

lemma *environment-self*: *S* ⊆ *environment S* **unfolding** *environment-def* **by** *auto*

lemma *environment-sym-2*: *A* ∩ *environment B* = {} ⟷ *B* ∩ *environment A* = {}
proof −
have *False* **if** *B* ∩ *environment A* = {} *x* ∈ *A* ∩ *environment B* **for** *x A B*
proof (*cases x* ∈ *B*)
case *True* **thus** *?thesis* **using** *that environment-self* **by** *auto*
next
case *False*
hence *x* ∈ {*x*. ∃ *v* ∈ *B*. *vert-adj x v*} **using** *that*(*2*) **unfolding** *environment-def* **by** *auto*

then obtain v **where** v-def: v ∈ B x ∈ environment {v} **unfolding** environment-def **by** auto
have v ∈ environment A **using** environment-mono that(2) environment-sym v-def(2) **by** blast
then show ?thesis **using** v-def(1) that(1) **by** auto
qed
thus ?thesis **by** auto
qed

lemma environment-range: S ⊆ V ⟹ environment S ⊆ V
unfolding environment-def **using** vert-adj-imp-inV **by** auto

lemma environment-union: environment (S ∪ T) = environment S ∪ environment T
unfolding environment-def **by** auto

lemma card-environment: card (environment {v}) = 1 + degree v (**is** ?L = ?R)
proof −
have ?L = card (insert v {x. {x, v} ∈ E}) **unfolding** environment-def vert-adj-def **by** simp
also have ... = Suc (card {x. {x, v} ∈ E})
by (intro card-insert-disjoint finite-subset[OF - finV])
(auto simp:singleton-not-edge wellformed-alt-fst)
also have ... = Suc (card (neighborhood v)) **unfolding** neighborhood-def vert-adj-def
by (intro arg-cong[**where** f=λx. Suc (card x)])
(auto simp:wellformed-alt-fst insert-commute)
also have ... = Suc (degree v)
unfolding alt-degree-def card-incident-sedges-neighborhood **by** simp
finally show ?thesis **by** simp
qed

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-size C = card {e ∈ E. e ∩ C ≠ {} ∧ e − C ≠ {}}

lemma eval-cond-edge:
assumes L ⊆ U finite U e ∈ E
shows (∫ C. of-bool (e∩C≠{} ∧ e−C≠{}) ∂pmf-of-set {C. L⊆C∧C⊆U}) =
((if e ⊆ −U ∪ L then of-bool(e ∩ L ≠{} ∧ e ∩ −U ≠{})::real else 1/2))
(**is** ?L = ?R)
proof −
obtain e1 e2 **where** e-def: e = {e1,e2} e1 ≠ e2 **using** two-edges[OF assms(3)]
by (meson card-2-iff)

let ?sing-iff = (λx e. (if x then {e} else {}))

define *R1* **where** *R1* = (*if e1* ∈ *L then* {*True*} *else* (*if e1* ∈ *U* − *L then* {*False*, *True*} *else* {*False*}))
define *R2* **where** *R2* = (*if e2* ∈ *L then* {*True*} *else* (*if e2* ∈ *U* − *L then* {*False*, *True*} *else* {*False*}))

have *bij*: *bij-betw* (λ*x*. ((*e1* ∈ *x*,*e2* ∈*x*),*x*−{*e1*,*e2*})) {*C*. *L* ⊆ *C* ∧ *C* ⊆ *U*}
((*R1* × *R2*) × {*C*. *L*−{*e1*,*e2*} ⊆ *C* ∧ *C* ⊆ *U*−{*e1*,*e2*}})
unfolding *R1-def R2-def* **using** *e-def*(*2*) *assms*(*1*)
by (*intro bij-betwI*[**where** *g*=(λ((*a*,*b*),*x*). *x* ∪ *?sing-iff a e1* ∪ *?sing-iff b e2*)])
(*auto split*:*if-split-asm*)

have *r*: *map-pmf* (λ*x*. (*e1* ∈ *x*, *e2* ∈ *x*)) (*pmf-of-set* {*C*. *L* ⊆ *C* ∧ *C* ⊆ *U*}) = *pmf-of-set* (*R1* × *R2*)
using *assms*(*1*,*2*) *map-pmf-of-set-bij-betw-2*[*OF bij*] **by** *auto*

have *?L* = ∫ *C*. *of-bool* ((*e1* ∈ *C*) ≠ (*e2* ∈ *C*)) ∂(*pmf-of-set* {*C*. *L* ⊆ *C* ∧ *C* ⊆ *U*})
unfolding *e-def*(*1*) **using** *e-def*(*2*) **by** (*intro integral-cong-AE AE-pmfI*) *auto*
also have ... = ∫ *x*. *of-bool*(*fst x* ≠ *snd x*) ∂*pmf-of-set* (*R1* × *R2*)
unfolding *r*[*symmetric*] **by** *simp*
also have ... = (*if* {*e1*,*e2*} ⊆ −*U* ∪ *L then of-bool*({*e1*,*e2*} ∩ *L* ≠{}∧{*e1*,*e2*} ∩− *U* ≠{}) *else 1*/*2*)
unfolding *R1-def R2-def e-def*(*1*) **using** *e-def*(*2*) *assms*(*1*)
by (*auto simp add*:*integral-pmf-of-set split*:*if-split-asm*)
also have ... = *?R* **unfolding** *e-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

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*:
(∫ *C*. *real* (*cut-size C*) ∂*pmf-of-set* (*Pow V*)) = *real* (*card E*) / *2* (**is** *?L* = *?R*)
proof −
have *a*:*False* **if** *x* ∈ *E x* ⊆ −*V* **for** *x*
proof −
have *x* = {} **using** *wellformed*[*OF that*(*1*)] *that*(*2*) **by** *auto*
thus *False* **using** *two-edges*[*OF that*(*1*)] **by** *simp*
qed

have *?L* = (∫ *C*. (∑ *e* ∈ *E*. *of-bool* (*e* ∩ *C* ≠ {} ∧ *e* − *C* ≠ {})) ∂*pmf-of-set* (*Pow V*))
using *fin-edges* **by** (*simp-all add*:*of-bool-def cut-size-def sum.If-cases Int-def*)
also have ... = (∑ *e* ∈ *E*. (∫ *C*. *of-bool* (*e* ∩ *C* ≠ {} ∧ *e* − *C* ≠ {}) ∂*pmf-of-set* (*Pow V*)))
using *finV* **by** (*intro Bochner-Integration.integral-sum integrable-measure-pmf-finite*)
(*simp add*: *Pow-not-empty*)
also have ... = (∑ *e* ∈ *E*. (∫ *C*. *of-bool* (*e*∩*C*≠{} ∧ *e* − *C* ≠ {}) ∂*pmf-of-set* {*C*. {} ⊆ *C* ∧ *C* ⊆ *V*}))
unfolding *Pow-def* **by** *simp*
also have ... = (∑ *e* ∈ *E*. (*if e* ⊆ − *V* ∪ {} *then of-bool* (*e* ∩ {} ≠{}∧*e* ∩−*V* ≠{}) *else 1* / *2*))
by (*intro sum.cong eval-cond-edge finV*) *auto*
also have ... = (∑ *e* ∈ *E*. *1*/*2*) **using** *a* **by** (*intro sum.cong*) *auto*
also have ... = *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed

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

lemma *exists-cut*: ∃ *C* ⊆ *V*. *real* (*cut-size C*) ≥ *card E*/*2*

proof −
have $\exists x \in$*set-pmf* (*pmf-of-set* (*Pow V*)). *card E* / *2* ≤ *cut-size x* **using** *finV exp-cut-size*[*symmetric*]
by (*intro exists-point-above-expectation integrable-measure-pmf-finite*)(*auto simp*:*Pow-not-empty*)
moreover have *set-pmf* (*pmf-of-set* (*Pow V*)) = *Pow V*
using *finV Pow-not-empty* **by** (*intro set-pmf-of-set*) *auto*
ultimately show *?thesis* **by** *auto*
qed

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* ∈ *E*. *v* ∈ *e* ∧ *e* ∩ *R* ≠ {}} ≥ *card* {*e* ∈ *E*. *v* ∈ *e* ∧ *e* ∩ *B* ≠ {}} *then*
derandomized-max-cut vs R (*B* ∪ {*v*}) *E*
else
derandomized-max-cut vs (*R* ∪ {*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* = (∫ *C*. *real* (*cut-size C*) ∂*pmf-of-set* {*C*. *R* ⊆ *C* ∧ *C* ⊆ *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* ⊆ *V B* ⊆ *V R* ∩ *B* = {} *v* ∈ *V* −*R*−*B*
shows *cond-exp R B* = (*cond-exp* (*R* ∪ {*v*}) *B* + *cond-exp R* (*B* ∪ {*v*}))/*2* (**is** *?L* = *?R*)
proof −
let *?A* = {*C*. *R*∪{*v*}⊆*C*∧*C*⊆*V*−*B*}
let *?B* = {*C*. *R*⊆*C*∧*C*⊆*V*−(*B*∪{*v*})}
define *p* **where** *p* = *real* (*card ?A*) / (*card ?A* + *card ?B*)

have *a*: {*C*. *R*⊆*C*∧*C*⊆*V*−*B*} = *?A* ∪ *?B* **using** *assms* **by** *auto*
have *b*: *?A* ∩ *?B* = {} **using** *assms* **by** *auto*
have *c*: *finite* (*?A* ∪ *?B*) **using** *finV* **by** *auto*
have *R* ∪{*v*} ⊆ *V*−*B* **using** *assms* **by** *auto*
hence *g*: *?A* ≠ {} **by** *auto*
hence *d*: *?A* ∪ *?B* ≠ {} **by** *simp*
have *e*: *real* (*cut-size x*) ≤ *real* (*card E*) **for** *x*
unfolding *cut-size-def* **by** (*intro of-nat-mono card-mono fin-edges*) *auto*

have *card ?A = card ?B* **using** *assms(1−4)*
by (*intro bij-betw-same-card*[**where** *f=λx. x − {v}*] *bij-betwI*[**where** *g=insert v*]) *auto*
moreover have *card ?A > 0* **using** *g c card-gt-0-iff* **by** *auto*
ultimately have *p-val*: *p = 1/2* **unfolding** *p-def* **by** *auto*
have *?L = (∫ b.(∫ C. real (cut-size C) ∂pmf-of-set (if b then ?A else ?B)) ∂bernoulli-pmf p)*
using *e* **unfolding** *cond-exp-def a pmf-of-set-un*[*OF d b c*] *p-def*
by (*subst integral-bind-pmf*[**where** *M=card E*]) *auto*
also have ... = ((∫ *C. real*(*cut-size C*) ∂*pmf-of-set ?A*)+(∫ *C. real*(*cut-size C*) ∂*pmf-of-set ?B*))/*2*
unfolding *p-val* **by** (*subst integral-bernoulli-pmf*) *simp-all*
also have ... = *?R* **unfolding** *cond-exp-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *cond-exp-cut-size*:
assumes *R ⊆ V B ⊆ V R ∩ 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 −
have *a*:*finite {C. R ⊆ C ∧ C ⊆ V − B} {C. R ⊆ C ∧ C ⊆ V − B} ≠ {}* **using** *finV assms* **by** *auto*

have *b*:*e ⊆ −V ∪ B ∪ R* **if** *cthat*: *e ∈ E e ∩ R ≠ {} e ∩ B ≠ {}* **for** *e*
proof −
obtain *e1* **where** *e1*: *e1 ∈ e ∩ R* **using** *cthat(2)* **by** *auto*
obtain *e2* **where** *e2*: *e2 ∈ e ∩ B* **using** *cthat(3)* **by** *auto*
have *e1 ≠ e2* **using** *e1 e2 assms(3)* **by** *auto*
hence *card {e1,e2} = 2* **by** *auto*
hence *e = {e1,e2}* **using** *two-edges*[*OF cthat(1)*] *e1 e2*
by (*intro card-seteq*[*symmetric*]) (*auto intro*!:*card-ge-0-finite*)
thus *?thesis* **using** *e1 e2* **by** *simp*
qed

have *?L = (∫ C. (∑ e ∈ E. of-bool (e ∩ C ≠ {} ∧ e − C ≠ {})) ∂pmf-of-set {C. R ⊆ C ∧ C ⊆ V−B})*
unfolding *cond-exp-def* **using** *fin-edges*
by (*simp-all add*:*of-bool-def cut-size-def sum.If-cases Int-def*)
also have ... = (∑ *e ∈ E.* (∫ *C. of-bool* (*e∩C ≠ {} ∧ e−C ≠ {}*) ∂*pmf-of-set {C. R⊆C ∧ C⊆V−B}*))
using *a* **by** (*intro Bochner-Integration.integral-sum integrable-measure-pmf-finite*) *auto*
also have ... = (∑ *e ∈ E.* ((*if e ⊆ −(V−B) ∪ R then of-bool*(*e∩R≠{}∧e∩−(V−B)≠{}*)::*real else 1/2*)))
using *finV assms(1,3)* **by** (*intro sum.cong eval-cond-edge*) *auto*
also have ... = *real* (*card {e∈E. e⊆−V∪B∪R∧e∩R≠{}∧e∩−(V−B)≠{}}*) + *real* (*card {e∈E. ¬ e ⊆−V∪B∪ R}*) / *2*
using *fin-edges* **by** (*simp add*: *sum.If-cases of-bool-def Int-def*)
also have ... = *?R* **using** *wellformed assms b*
by (*intro arg-cong*[**where** *f=card*] *arg-cong2*[**where** *f=(+)*] *arg-cong*[**where** *f=real*]
arg-cong2[**where** *f=(/)*] *refl Collect-cong order-antisym*) *auto*
finally show *?thesis* **by** *simp*
qed

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 –
  define R :: 'a set where R = {}
  define B :: 'a set where B = {}
  have a:cut-size (derandomized-max-cut vs R B E) ≥ cond-exp R B ∧
      (derandomized-max-cut vs R B E) ⊆ V
    if distinct vs set vs ∩ R = {} set vs ∩ B = {} R ∩ B = {} ⋃{set vs,R,B}= V
    using that
  proof (induction vs arbitrary: R B)
    case Nil
    have cond-exp R B = real (card {e∈E. e∩R≠{}∧e∩B≠{}}) + real (card {e∈E. e∩V−R−B
≠ {}}) / 2
      using Nil by (intro cond-exp-cut-size) auto
     also have ... = real (card {e∈E. e∩R≠{}∧e∩B≠{}})+real (card ({}::'a set set ))/2 using
Nil
      by (intro arg-cong[where f=card] arg-cong2[where f=(+)]  arg-cong2[where f=(/)]
          arg-cong[where f=real]) auto
    also have ... = real (card {e∈E. e∩R≠{}∧e∩B≠{}}) by simp
    also have ... = real (cut-size R) using Nil wellformed unfolding cut-size-def
      by (intro arg-cong[where f=card] arg-cong2[where f=(+)] arg-cong[where f=real]) auto
    finally have cond-exp R B = real (cut-size R) by simp
    thus ?case using Nil by auto
  next
    case (Cons vh vt)
    let ?NB = {e ∈ E. vh ∈ e ∧ e ∩ B ≠ {}}
    let ?NR = {e ∈ E. vh ∈ e ∧ e ∩ R ≠ {}}
    define t where t = real (card {e ∈ E. e ∩ V − R − (B ∪ {vh}) ≠ {}}) / 2
    have t-alt: t = real (card {e ∈ E. e ∩ V − (R ∪ {vh}) − B ≠ {}}) / 2
      unfolding t-def by (intro arg-cong[where f=λx. real (card x) /2]) auto

    have cond-exp R (B∪{vh})−card ?NR = real(card {e∈E. e∩R≠{}∧e∩(B∪{vh})≠{}})−(card
?NR)+t
      using Cons(2−6) unfolding t-def by (subst cond-exp-cut-size) auto
    also have ... = real(card {e∈E. e∩R≠{}∧e∩(B∪{vh})≠{}}−card ?NR)+t
      using fin-edges by (intro of-nat-diff[symmetric] arg-cong2[where f=(+)] card-mono) auto
    also have ... = real(card ({e∈E. e∩R≠{}∧e∩(B∪{vh})≠{}}− ?NR))+t
       using fin-edges by (intro arg-cong[where f=(λx. real x+t)] card-Diff-subset[symmetric])
auto
    also have ... = real(card ({e∈E. e∩(R∪{vh})≠{}∧e∩B≠{}}− ?NB))+t
      by (intro arg-cong[where f=(λx. real (card x) + t)] ) auto
    also have ... = real(card {e∈E. e∩(R∪{vh})≠{}∧e∩B≠{}}−card ?NB)+t
      using fin-edges by (intro arg-cong[where f=(λx. real x+t)] card-Diff-subset) auto
    also have ... = real(card {e∈E. e∩(R∪{vh})≠{}∧e∩B≠{}})−(card ?NB)+t
      using fin-edges by (intro of-nat-diff arg-cong2[where f=(+)] card-mono) auto
    also have ... = cond-exp (R∪{vh}) B −card ?NB
      using Cons(2−6) unfolding t-alt by (subst cond-exp-cut-size) auto
    finally have d:cond-exp R (B∪{vh}) − cond-exp (R∪{vh}) B = real (card ?NR) − card ?NB
      by (simp add:ac-simps)

    have split: cond-exp R B = (cond-exp (R ∪ {vh}) B + cond-exp R (B ∪ {vh})) / 2
      using Cons(2−6) by (intro cond-exp-split) auto

    have dvt: distinct vt using Cons(2) by simp
    show ?case
    proof (cases card ?NR ≥ card ?NB)
      case True
      have 0:set vt∩R={} set vt∩(B∪{vh})={} R∩(B∪{vh})={} ⋃{set vt,R,B∪{vh}}=V
        using Cons(2−6) by auto
```

```
    have cond-exp R B ≤ cond-exp R (B ∪ {vh}) unfolding split using d True by simp
    thus ?thesis using True Cons(1)[OF dvt 0] by simp
  next
    case False
    have 0:set vt∩(R∪{vh})={} set vt∩B={} (R∪{vh})∩B={} ⋃{set vt,R∪{vh},B}=V
      using Cons(2−6) by auto
    have cond-exp R B ≤ cond-exp (R ∪ {vh}) B unfolding split using d False by simp
    thus ?thesis using False Cons(1)[OF dvt 0] by simp
  qed
 qed
 moreover have e ∩ V ≠ {} if e ∈ E for e
  using Int-absorb2[OF wellformed[OF that]] two-edges[OF that] by auto
 hence {e ∈ E. e ∩ V ≠ {}} = E by auto
 hence cond-exp {} {} = graph-size / 2 by (subst cond-exp-cut-size) auto
 ultimately show C ⊆ V 2 ∗ cut-size C ≥ card E
  unfolding C-def R-def B-def using permutations-of-setD[OF assms(1)] by auto
qed

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 (λw. {v,w} ∉ E) vt) E

context fin-sgraph
begin

lemma indep-set-range: subseq (indep-set p E) p
proof (induction p rule:subseq-induct')
 case 1 thus ?case by simp
next
 case (2 ph pt)
 have subseq (filter (λw. {ph, w} ∉ E) pt) pt by simp
 also have strict-subseq ... (ph#pt) unfolding strict-subseq-def by auto
 finally have strict-subseq (filter (λw. {ph, w} ∉ E) pt) (ph # pt) by simp
 hence subseq (indep-set (ph # pt) E) (ph#filter (λw. {ph, w} ∉ E) pt)
  unfolding indep-set.simps by (intro 2 subseq-Cons2)
```

also have *subseq ...* (*ph#pt*) **by** *simp*
finally show *?case* **by** *simp*
qed

lemma *is-independent-set-insert*:
assumes *is-independent-set A x* ∈ *V* − *environment A*
shows *is-independent-set* (*insert x A*)
using *assms* **unfolding** *is-independent-alt vert-adj-def environment-def*
by (*simp add*:*insert-commute singleton-not-edge*)

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 −
show *distinct* (*indep-set p E*) **using** *indep-set-range assms*(*1*) *subseq-distinct* **by** *auto*
show *set* (*indep-set p E*) ⊆ *V* **using** *indep-set-range assms*(*2*)
by (*metis* (*full-types*) *list-emb-set subset-code*(*1*))

show *is-independent-set* (*set* (*indep-set p E*))
using *assms*(*1*,*2*)
proof (*induction p rule*:*subseq-induct′*)
case *1*
then show *?case* **by** (*auto simp add*:*is-independent-set-def all-edges-def*)
next
case (*2 y ys*)
have *subseq* (*filter* (*λw.* {*y*, *w*} ∉ *E*) *ys*) *ys* **by** *simp*
also have *strict-subseq ...* (*y#ys*) **by** (*simp add*: *list-emb-Cons strict-subseq-def*)
finally have *strict-subseq* (*filter* (*λw.* {*y*, *w*} ∉ *E*) *ys*) (*y* # *ys*) **by** *simp*
moreover have *False* **if** *y*∈*environment* (*set* (*indep-set* (*filter* (*λw.* {*y*, *w*} ∉ *E*) *ys*) *E*))
proof −
have *y* ∈ *environment* (*set* (*filter* (*λw.* {*y*,*w*} ∉ *E*) *ys*))
using *that environment-mono subseq-set*[*OF indep-set-range*] **by** *blast*
hence ∃ *z* ∈ (*set* (*filter* (*λw.* {*y*,*w*} ∉ *E*) *ys*)). {*z*,*y*} ∈ *E*
using *2*(*2*) **unfolding** *environment-def vert-adj-def* **by** *simp*
then show *?thesis* **by** (*simp add*:*insert-commute*)
qed
ultimately have *is-independent-set* (*insert y* (*set* (*indep-set* (*filter* (*λw.* {*y*, *w*} ∉ *E*) *ys*) *E*)))

using *2*(*2*,*3*) **by** (*intro is-independent-set-insert 2*) *auto*
thus *?case* **by** *simp*
qed
qed

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* (*λy. 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 −
let *?f* = (*λy. y* ∈ *environment* {*v*})

```
  obtain q1 q2 where q-def: q = q1@v#q2 using assms(4) by (meson split-list)
  obtain p1 p2 where p-def: p = p1@p2 subseq q1 p1 subseq (v#q2) p2
    using assms(3) list-emb-appendD unfolding q-def by blast

  have v ∈ set p2 using p-def(3) list-emb-set by force
  hence 0:v ∉ set p1 using assms(2) unfolding p-def(1) by auto
  have filter ?f p1 = []
  proof (cases filter ?f p1)
    case Nil thus ?thesis by simp
  next
    case (Cons p1h p2h)
    hence p1h = v using assms(1) unfolding is-first-def p-def(1) by simp
    hence False using 0 Cons by (metis filter-eq-ConsD in-set-conv-decomp)
    then show ?thesis by simp
  qed
  hence filter ?f q1 = [] using p-def(2) by (metis (full-types) filter-empty-conv list-emb-set)
  moreover have v ∈ environment {v} unfolding environment-def by simp
  ultimately show ?thesis unfolding q-def is-first-def by simp
qed

lemma is-first-imp-in-set:
  assumes is-first v p
  shows v ∈ set p
proof −
  have v ∈ set (filter (λy. y ∈ environment {v}) p)
    using assms unfolding is-first-def by (meson prefix-imp-subseq subseq-singleton-left)
  thus ?thesis by simp
qed
```

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)
  using assms
proof (induction p rule:subseq-induct)
  case (1 ys)
  hence i:v ∈ set (indep-set zs E) if is-first v zs strict-subseq zs ys for zs
    using strict-subseq-imp-distinct strict-subseq-set that by (intro 1(1)) blast+

  define ysh yst where ysht-def: ysh = hd ys yst = tl ys
  have split-ys: ys = ysh#yst if ys ≠ [] using that unfolding ysht-def by auto

  consider (a) ys = [] | (b) ys ≠ [] hd ys = v | (c) ys ≠ [] hd ys ≠ v by auto
  then show ?case
  proof (cases)
    case a then show ?thesis using 1(4) by (simp add:is-first-def)
  next
    case b then show ?thesis unfolding split-ys[OF b(1)] by simp
  next
    case c
    have 0:subseq (filter (λw. {ysh, w} ∉ E) yst) ys unfolding split-ys[OF c(1)] by auto
    have v ∈ set ys using 1(4) is-first-imp-in-set by auto
    hence v ∈ set yst using c unfolding split-ys[OF c(1)] by simp
    moreover have ysh ≠ v using c(2) split-ys[OF c(1)] by simp
    hence ysh ∉ environment {v} using 1(4) unfolding is-first-def split-ys[OF c(1)] by auto
    hence {ysh,v} ∉ E unfolding environment-def vert-adj-def by auto
```

ultimately have *v* ∈ *set* (*filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst*) **by** *simp*
hence *is-first v* (*filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst*) **by** (*intro is-first-subseq*[*OF 1*(*4*)] *0 1*(*2*))
moreover have *length yst* < *length ys* **using** *split-ys*[*OF c*(*1*)] **by** *auto*
hence *length* (*filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst*) < *length ys*
using *length-filter-le dual-order.strict-trans2* **by** *blast*
hence *filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst* ≠ *ys* **by** *auto*
hence *strict-subseq* (*filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst*) *ys*
using *0* **unfolding** *strict-subseq-def* **by** *auto*
ultimately have *v* ∈ *set* (*indep-set* (*filter* (λ*w*. {*ysh*, *w*} ∉ *E*) *yst*) *E*) **by** (*intro i*)
then show *?thesis* **unfolding** *split-ys*[*OF c*(*1*)] **by** *simp*
qed
qed

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 −
let *?perm* = (λ*x*. *pmf-of-set* (*permutations-of-set x*))
have *a*:*finite* (*set-pmf* Ω) **unfolding** Ω*-def* **using** *perm-non-empty-finite* **by** *simp*
have *b*:*distinct y set y* ⊆ *V* **if** *y* ∈ *set-pmf* Ω **for** *y*
using *that perm-non-empty-finite permutations-of-setD* **unfolding** Ω*-def* **by** *auto*

have *?R* = (∑ *v*∈*V*. *1* / *real* (*card* (*environment* {*v*}))) **unfolding** *card-environment* **by** *simp*
also have ... = (∑ *v*∈*V*. *measure* (*?perm* (*environment* {*v*})) {*vs*. *prefix*[*v*] *vs*})
using *finite-environment environment-self* **by** (*intro sum.cong permutations-of-set-prefix*[*symmetric*]) *auto*
also have ... = (∑ *v* ∈ *V*. (∫ *vs*. *indicator* {*vs*. *prefix* [*v*] *vs*} *vs* ∂*?perm* (*environment* {*v*}∩*V*)))
using *Int-absorb2*[*OF environment-range*] **by** (*intro sum.cong refl*) *simp*
also have ...=(∑ *v*∈*V*.(∫ *vs*. *of-bool*(*prefix*[*v*]*vs*) ∂*map-pmf* (*filter* (λ*x*. *x* ∈ *environment* {*v*})) Ω))
unfolding Ω*-def filter-permutations-of-set-pmf*[*OF finV*]
by (*intro sum.cong arg-cong2*[**where** *f*=*measure-pmf.expectation*])
(*simp-all add*:*Int-def conj-commute of-bool-def indicator-def*)
also have ... = (∑ *v* ∈ *V*. (∫ *vs*. *of-bool*(*is-first v vs*) ∂Ω))
unfolding *is-first-def* **by** (*intro sum.cong*) *simp-all*
also have ... = (∫ *vs*. (∑ *v* ∈ *V*. *of-bool*(*is-first v vs*)) ∂Ω)
by (*intro integral-sum*[*symmetric*] *integrable-measure-pmf-finite*[*OF a*])
also have ... ≤ (∫ *vs*. *real* (*card* (*set* (*indep-set vs E*))) ∂Ω)
using *finV b* **by** (*intro integral-mono-AE AE-pmfI integrable-measure-pmf-finite*[*OF a*])
(*auto intro*!:*card-mono set-indep-set*)
also have ... ≤ *?L*
by (*intro integral-mono-AE AE-pmfI integrable-measure-pmf-finite*[*OF a*] *of-nat-mono card-length*)
finally show *?thesis* **by** *simp*
qed

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 −
have *convex-on* {*x*. *x* + *1* ∈ {*0*<..}} (λ*x*. *inverse* (*x*+*1*::*real*))
by (*intro convex-on-shift*[*OF convex-on-inverse*]) *auto*
moreover have {*x*. (*0*::*real*) < *x* + *1*} = {−*1*<..} **by** (*auto simp*:*algebra-simps*)
ultimately show *?thesis* **by** (*simp add*:*inverse-eq-divide*)
qed

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

proof −
  have *card V* / (*2*∗*card E* / *card V* + *1*) = *card V*∗ (*1* / (((*2*∗*card E*)::*real*) / *card V* + *1*))
by *simp*
  also have ... = *card V*∗ (*1* / (($\sum$ *v* ∈ *V*. (*1* / *real* (*card V*)) $*_R$ *degree v*) + *1*))
    unfolding *degree-sum*[*symmetric*] **by** (*simp add*:*sum-divide-distrib*)
  also have ... ≤ *card V* ∗ ($\sum$ *v* ∈ *V*. (*1* / *card V*) ∗ (*1*/ (*degree v*+(*1*::*real*))))
  proof (*cases V* = {})
    case *True* **thus** *?thesis* **by** *simp*
  next
    case *False* **thus** *?thesis*
      using *finV* **by** (*intro mult-left-mono convex-on-sum*[*OF - - inverse-x-plus-1-convex*] *finV*)
auto
  qed
  also have ... = ($\sum$ *v* ∈ *V*. *1*/ (*degree v*+*1*))
    using *finV* **unfolding** *sum-distrib-left* **by** (*intro sum.cong refl*) *auto*
  finally show *?thesis* **by** *simp*
qed

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 −
  let *?Ω* = *pmf-of-set* (*permutations-of-set V*)
  let *?w* = *real* (*card V*) / (*real* (*2*∗*card E*) / *card V* + *1*)

  have *a*:*finite* (*set-pmf ?Ω*) **using** *perm-non-empty-finite* **by** *simp*

  have ($\int$ *vs. real* (*length* (*indep-set vs E*)) ∂*?Ω*) ≥ *?w*
    using *exp-indep-set caro-wei-aux* **by** *simp*
  then obtain *vs* **where** *vs-def*: *vs* ∈ *set-pmf ?Ω real* (*length* (*indep-set vs E*)) ≥ *?w*
    using *exists-point-above-expectation integrable-measure-pmf-finite*[*OF a*] **by** *blast*
  define *S* **where** *S* = *set* (*indep-set vs E*)

  have *vs-range*: *distinct vs set vs* ⊆ *V*
    using *vs-def*(*1*) *perm-non-empty-finite permutations-of-setD* **by** *auto*

  have *b*:*S* ⊆ *V is-independent-set S* **and** *c*: *distinct* (*indep-set vs E*)
    unfolding *S-def* **using** *indep-set-correct*[*OF vs-range*] **by** *auto*

  have *real* (*card S*) = *length* (*indep-set vs E*) **using** *c distinct-card* **unfolding** *S-def* **by** *auto*
  also have ... ≥ *?w* **using** *vs-def*(*2*) **by** *auto*
  finally have *real* (*card S*) ≥ *?w* **by** *simp*
  thus *?thesis* **using** *b c* **by** *auto*
qed

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*. $\sum$ *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* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *set set* $\Rightarrow$ $'a$ *list*
  where
    derandomized-indep-set [] *p E* = *indep-set p E* |
    derandomized-indep-set (*vh#vt*) *p E* = (
      let node-deg = (λv. *real* (*card* $\{e \in E.\ v \in e\}$));
          is-indep = (λv. *list-all* (λw. $\{v,w\} \notin E$) *p*);
          env = (λv. *filter is-indep* (*v#filter* (λw. $\{v,w\} \in E$) (*vh#vt*)));
          cost = (λv. ($\sum w \leftarrow$ *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*)
  by *pat-completeness auto*

termination
proof (*relation Wellfounded.measure* (λx. *length*(*fst x*)))
  fix *cost* :: $'a \Rightarrow$ *real* **and** *w vh* :: $'a$ **and** *p vt* :: $'a$ *list* **and** *E* :: $'a$ *set set*
  define *v* **where** *v* = *vh#vt*
  assume *w* = *arg-min-list cost* (*vh # vt*)
  hence *w* $\in$ *set v* **unfolding** *v-def* **using** *arg-min-list-in* **by** *blast*
  thus ((*remove1 w v*, *p* @ [*w*], *E*), *v*, *p*, *E*) $\in$ *Wellfounded.measure* (λx. *length* (*fst x*))
    unfolding *in-measure* **by** (*simp add*:*length-remove1*) (*simp add*: *v-def*)
qed *auto*

context *fin-sgraph*
begin

lemma *is-first-append-1*:
  assumes *v* $\notin$ *environment* (*set p*)
  shows *is-first v* (*p*@*q*) = *is-first v q*
proof −
  have *environment* $\{v\} \cap$ *set p* = {} **using** *environment-sym-2 assms* **by** *auto*
  hence *filter* (λy. *y* $\in$ *environment* $\{v\}$) *p* = [] **unfolding** *filter-empty-conv* **by** *auto*
  thus *?thesis* **unfolding** *is-first-def* **by** *simp*
qed

lemma *is-first-append-2*:
  assumes *v* $\in$ *environment* (*set p*)
  shows *is-first v* (*p*@*q*) = *is-first v p*
proof −
  obtain *u* **where** *u* $\in$ *set p v* $\in$ *environment* $\{u\}$
    using *assms* **unfolding** *environment-def* **by** *auto*
  hence *filter* (λy. *y* $\in$ *environment* $\{v\}$) *p* $\neq$ []
    using *environment-sym* **unfolding** *filter-empty-conv* **by** *meson*
  thus *?thesis* **unfolding** *is-first-def* **by** (*cases filter* (λy. *y* $\in$ *environment* $\{v\}$) *p*) *auto*
qed

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

definition *p-estimator* **where**
  p-estimator p = ($\int$ *vs*. ($\sum v \in V$. *of-bool*(*is-first v vs*)) ∂*pmf-of-set* (*cond-perm V p*))

lemma *p-estimator-split*:
  assumes *V* − *set p* $\neq$ {}
  shows *p-estimator p* = ($\sum v \in V-$*set p*. *p-estimator* (*p*@[*v*])) / *real* (*card* (*V*−*set p*)) (**is** *?L* =

?R)
proof –
let *?q* = *λx. pmf-of-set* (*permutations-of-set* (*V*−*set p*−{*x*}))
have *0*:*finite* (*V* − *set p*) *V* − *set p* ≠ {} **using** *finV assms* **by** *auto*

have *?L* = (∫ *vs.* (∑ *v*∈*V*. *of-bool* (*is-first v* (*p*@*vs*))) ∂*pmf-of-set* (*permutations-of-set* (*V*−*set p*)))
using *finV* **unfolding** *p-estimator-def cond-perm-def*
by (*subst map-pmf-of-set-inj*[*symmetric*]) (*auto intro*:*inj-onI*)
also have ...=(∑ *x*∈*V*−*set p*.(∫ *vs*.(∑ *v*∈*V*. *of-bool*(*is-first v*(*p*@*x*#*vs*)))∂*?q x*))/*real*(*card* (*V*−*set p*))
using *0* **unfolding** *random-permutation-of-set*[*OF 0*] **by** (*subst pmf-expectation-bind-pmf-of-set*)
(*simp-all add*:*map-pmf-def*[*symmetric*] *inverse-eq-divide sum-divide-distrib*)
also have ... = (∑ *x*∈*V*−*set p*. *p-estimator* (*p*@[*x*])) /*real*(*card* (*V*−*set p*))
using *finV Diff-insert* **unfolding** *p-estimator-def cond-perm-def*
by (*subst map-pmf-of-set-inj*[*symmetric*]) (*auto intro*:*inj-onI simp flip*:*Diff-insert*)
finally show *?thesis* **by** *simp*
qed

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 –
let *?p* = *pmf-of-set* (*cond-perm V p*)
let *?q* = *pmf-of-set* (*permutations-of-set* (*V* − *set p*))
define *Q* **where** *Q* = *environment* (*set p*) − *P*

have *P* ⊆ *V* **using** *assms*(*2*) *is-first-imp-in-set* **unfolding** *P-def* **by** *auto*
moreover have *environment* (*set p*) ⊆ *V* **using** *environment-range assms*(*2*) **by** *auto*
ultimately have *V-split*: *V* = *P* ∪ *Q* ∪ *R* **unfolding** *R-def Q-def* **by** *auto*

have *P* ⊆ *environment* (*set p*) **using** *environment-def P-def is-first-imp-in-set* **by** *auto*
hence *0*: (*P* ∪ *Q*) ∩ *R* = {} *P* ∩ *Q* = {} **unfolding** *R-def Q-def* **by** *auto*

have *1*: *finite P finite R finite* (*P* ∪ *Q*) **using** *V-split finV* **by** *auto*

have *a*: *is-first v* (*p*@*vs*) **if** *v* ∈ *P* **for** *v vs*
using *that* **unfolding** *P-def is-first-def* **by** *auto*

have *b*: ¬*is-first v* (*p*@*vs*) **if** *v* ∈ *Q* **for** *v vs*
using *that* **unfolding** *Q-def P-def* **by** (*subst is-first-append-2*) *auto*

have *c*: (∫ *vs*. *of-bool* (*is-first v* (*p*@*vs*)) ∂*?q*) = *1* / (*degree v* + *1*::*real*) (**is** *?L1* = *?R1*)
if *v-range*:*v* ∈ *R* **for** *v*
proof –
have *set p* ∩ *environment* {*v*} = {} **using** *that environment-sym-2* **unfolding** *R-def* **by** *auto*
moreover have *environment* {*v*} ⊆ *V*
using *v-range* **unfolding** *R-def* **by** (*intro environment-range*) *auto*
ultimately have *d*:{*x*∈*V*−*set p*. *x*∈*environment*{*v*}} = *environment*{*v*} **by** *auto*

have *?L1* = (∫ *vs*. *indicator* {*vs*. *is-first v* (*p*@*vs*)} *vs* ∂*?q*) **by** (*simp add*:*indicator-def*)
also have ... = *measure ?q* {*vs*. *is-first v* (*p*@*vs*)} **by** *simp*
also have ... = *measure ?q* {*vs*. *is-first v vs*}

using *that* **unfolding** *R-def*
by (*intro arg-cong2*[**where** *f=measure*] *Collect-cong is-first-append-1*) *auto*
also have ... = *measure* (*map-pmf* (*filter* ($\lambda x.\ x \in$ *environment* $\{v\}$)) *?q*) $\{$*vs. prefix* [*v*] *vs*$\}$
unfolding *is-first-def* **by** *simp*
also have ... =
measure (*pmf-of-set* (*permutations-of-set* $\{x \in V - set\ p.\ x \in environment\{v\}\}$)) $\{$*vs. prefix* [*v*] *vs*$\}$
using *finV* **by** (*subst filter-permutations-of-set-pmf*) *auto*
also have ... = *1* / *real* (*card* (*environment* $\{v\}$)) **unfolding** *d*
using *finite-environment environment-self* **by** (*subst permutations-of-set-prefix*) *auto*
also have ... = *?R1* **unfolding** *card-environment* **by** *simp*
finally show *?thesis* **by** *simp*
qed

have *?L* = ($\int$ *vs. real* ($\sum v \in V$. *of-bool* (*is-first v vs*)) ∂ *?p*)
unfolding *p-estimator-def* **using** *cond-perm-non-empty-finite cond-permD*[*OF assms*(*1*,*2*)]
by (*intro integral-cong-AE AE-pmfI arg-cong*[**where** *f=real*]) *auto*
also have ... = ($\int$ *vs.* ($\sum v \in V$. *of-bool* (*is-first v vs*)) ∂ *?p*) **by** *simp*
also have ... = ($\sum v \in V$. ($\int$ *vs. of-bool* (*is-first v vs*) ∂ *?p*))
by (*intro integral-sum finite-measure.integrable-const-bound*[**where** *B=1*] *AE-pmfI*) *auto*
also have ... = ($\sum v \in V$. ($\int$ *vs. of-bool* (*is-first v vs*) ∂*map-pmf* ((@) *p*) *?q*))
unfolding *cond-perm-def* **by** (*subst map-pmf-of-set-inj*) (*auto intro*:*inj-onI finV*)
also have ... = ($\sum v \in V$. ($\int$ *vs. of-bool* (*is-first v* (*p*@*vs*)) ∂*?q*)) **by** *simp*
also have ... = *real* (*card P*) + ($\sum v \in R$. ($\int$ *vs. of-bool* (*is-first v* (*p*@*vs*)) ∂*?q*))
unfolding *V-split* **using** *0 1 a b* **by** (*simp add*: *sum.union-disjoint*)
also have ... = *?R* **by** (*simp add*:*c cong*:*sum.cong*)
finally show *?thesis* **by** *simp*
qed

lemma *p-estimator-step*:
assumes *distinct* (*p*@[*v*]) *set* (*p*@[*v*]) $\subseteq V$
shows *p-estimator* (*p*@[*v*]) − *p-estimator p* = *of-bool*(*environment* $\{v\} \cap$ *set p* = $\{\}$)
− ($\sum w \in environment\ \{v\} - environment(set\ p).\ 1\ /\ (degree\ w+1::real)$)
proof −
let *?d* = λv. *1*/(*degree v* + *1*::*real*)
let *?e* = λx. *environment x*
define τ :: *nat* **where** τ = *of-bool*(*environment* $\{v\} \cap$ *set p* = $\{\}$)
have *real-tau*: *of-bool*(*environment* $\{v\} \cap$ *set p* = $\{\}$) = *real* τ **unfolding** τ*-def* **by** *simp*
have *v-range*: $v \in V$ **using** *assms*(*2*) **by** *auto*

have *3*: *finite* (*set* (*p*@[*v*])) **by** *simp*
have *4*: *is-first w* (*p* @ [*v*]) $\longleftrightarrow$ *is-first w p* **if** $w \neq v$ **for** *w*
using *that* **unfolding** *is-first-def* **by** *auto*
have *7*:$v \notin$ *set p* **using** *assms*(*1*) **by** *simp*
hence *5*:$w \neq v$ **if** *is-first w p* **for** *w* **using** *is-first-imp-in-set*[*OF that*] **by** *auto*

have *environment* $\{v\} \cap$ *set p* = $\{\}$ $\longleftrightarrow$ *is-first v* (*p*@[*v*]) (**is** *?L1* $\longleftrightarrow$ *?R1*)
proof
assume *?L1*
hence $x \notin$ *environment* $\{v\}$ **if** $x \in$ *set p* **for** *x* **using** *that* **by** *auto*
moreover have $v \in$ *environment* $\{v\}$ **unfolding** *environment-def* **by** *auto*
ultimately show *?R1* **unfolding** *is-first-def* **by** (*simp add:filter-empty-conv*)
next
assume *?R1*
moreover have $v \notin$ *set p* **using** *assms*(*1*) **by** *auto*
hence $\neg$*prefix* [*v*] (*filter* ($\lambda y.\ y \in$ *environment* $\{v\}$) *p*)
by (*meson filter-is-subset prefix-imp-subseq subseq-singleton-left subset-code*(*1*))
ultimately have *filter* ($\lambda y.\ y \in$ *environment* $\{v\}$) *p* = []

unfolding *is-first-def filter-append* **by** (*cases filter* (λ*y*. *y* ∈ *environment* {*v*}) *p*) *auto*
thus *?L1* **unfolding** *filter-empty-conv* **by** *auto*
qed
hence *6*: τ = *of-bool* (*is-first v* (*p*@[*v*])) **unfolding** *τ-def* **by** *simp*

have *card* {*w*. *is-first w*(*p*@[*v*])}=*card* {*w*. *is-first w*(*p*@[*v*])∧*w*≠*v*}+*card* {*w*. *is-first v*(*p*@[*v*])∧*w*=*v*}
using *is-first-imp-in-set* **by** (*subst card-Un-disjoint*[*symmetric*])
(*auto intro*:*finite-subset*[*OF* - *3*] *arg-cong*[**where** *f*=*card*])
also have ... = *card* {*w*. *is-first w p* ∧*w*≠*v*} + *of-bool* (*is-first v* (*p*@[*v*]))
using *4* **by** (*intro arg-cong2*[**where** *f*=(+)] *arg-cong*[**where** *f*=*card*] *Collect-cong*) *auto*
also have ... = *card* {*w*. *is-first w p*} + τ
using *5 6* **by** (*intro arg-cong2*[**where** *f*=(+)] *arg-cong*[**where** *f*=*card*] *Collect-cong*) *auto*
finally have *2*:*card* {*w*. *is-first w* (*p*@[*v*])} = *card* {*w*. *is-first w p*} + τ **by** *simp*

have *?e* {*v*} ⊆ *V* **using** *v-range environment-range* **by** *auto*
hence *V*−*?e* (*set* (*p*@[*v*])) ∪ (*?e* {*v*}−*?e* (*set p*)) = *V* − *?e* (*set p*)
unfolding *set-append environment-union* **by** *auto*
moreover have *?e* {*v*} ⊆ *?e* (*set* (*p*@[*v*])) **unfolding** *environment-def* **by** *auto*
hence (*V*−*?e* (*set* (*p*@[*v*]))) ∩ (*?e* {*v*}−*?e* (*set p*)) = {} **by** *blast*
moreover have *finite* (*?e* {*v*}) **by** (*intro finite-environment*) *auto*
ultimately have *3*:
(∑ *v*∈*V*−*?e* (*set* (*p*@[*v*])). *?d v*)+ (∑ *v*∈*?e* {*v*}−*?e* (*set p*). *?d v*) = (∑ *v*∈*V*−*?e* (*set p*). *?d v*)
using *finV* **by** (*subst sum.union-disjoint*[*symmetric*]) *auto*

show *?thesis*
using *assms 2 3* **unfolding** *real-tau* **by** (*subst* (*1 2*) *p-estimator*) *auto*
qed

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*))
using *assms*
proof (*induction p1 arbitrary*: *p2 rule*:*subseq-induct′*)
case *1*
hence *distinct* (*indep-set p2 E*) ∧ *is-independent-set* (*set* (*indep-set p2 E*))
using *permutations-of-setD* **by** (*intro conjI indep-set-correct*) *auto*
thus *?case* **by** *simp*
next
case (*2 p1h p1t*)
define *p1* **where** *p1* = *p1h*#*p1t*
define *node-deg* **where** *node-deg* = (λ*v*. *real* (*card* {*e* ∈ *E*. *v* ∈ *e*}))
define *is-indep* **where** *is-indep* = (λ*v*. *list-all* (λ*w*. {*v*,*w*} ∉ *E*) *p2*)
define *env* **where** *env* = (λ*v*. *filter is-indep* (*v*#*filter* (λ*w*. {*v*,*w*} ∈ *E*) (*p1h*#*p1t*)))
define *cost* **where** *cost* = (λ*v*. (∑ *w* ← *env v*. *1* /(*node-deg w*+*1*)) − *of-bool*(*is-indep v*))
define *w* **where** *w* = *arg-min-list cost p1*
have *w-set*: *w* ∈ *set p1* **unfolding** *w-def p1-def* **using** *arg-min-list-in* **by** *blast*
have *perm*: *p1*@*p2* ∈ *permutations-of-set V* **using** *2*(*2*) *p1-def* **by** *auto*
have *dist*: *distinct p1 distinct p2 set p1* ∩ *set p2* = {} *set p1* ∪ *set p2* = *V*
set p1 = *V* − *set p2* **using** *permutations-of-setD*[*OF perm*] **by** *auto*

have *a*: *set* (*remove1 w p1* @ *p2* @ [*w*]) = *V* **using** *w-set dist*(*4*) **by** (*auto simp*:*set-remove1-eq*[*OF dist*(*1*)])

have *b*: *distinct* (*remove1 w p1* @ *p2* @ [*w*]) **using** *dist*(*1*,*2*,*3*) *w-set* **by** *auto*
have *c*: *strict-subseq* (*remove1 w p1*) *p1* **by** (*intro strict-subseq-remove1 w-set*)

have distinct (derandomized-indep-set (remove1 w (p1h # p1t)) (p2 @ [w]) E) ∧
is-independent-set (set (derandomized-indep-set (remove1 w (p1h # p1t)) (p2 @ [w]) E))
using a b c **unfolding** p1-def **by** (intro 2 permutations-of-setI) simp-all
thus ?case
unfolding p1-def derandomized-indep-set.simps node-deg-def[symmetric] is-indep-def[symmetric]
by (simp del:remove1.simps add:Let-def cost-def p1-def env-def w-def)
qed

lemma derandomized-indep-set-length-aux:
assumes p1@p2 ∈ permutations-of-set V
shows length (derandomized-indep-set p1 p2 E) ≥ p-estimator p2
using assms
proof (induction p1 arbitrary: p2 rule:subseq-induct′)
case 1
have a:set p2 − environment (set p2) = {} **using** environment-self **by** auto
have p-estimator p2 = card {v. is-first v p2}
using permutations-of-setD[OF 1] **by** (subst p-estimator) (auto simp:a)
also have ... ≤ card (set (indep-set p2 E))
using permutations-of-setD[OF 1] set-indep-set **by** (intro of-nat-mono card-mono) auto
also have ... ≤ length (indep-set p2 E) **using** card-length **by** auto
also have ... = length (derandomized-indep-set [] p2 E) **using** 1 **by** simp
finally show ?case **by** simp
next
case (2 p1h p1t)
define p1 **where** p1 = p1h#p1t
define node-deg **where** node-deg = (λv. real (card {e ∈ E. v ∈ e}))
define is-indep **where** is-indep = (λv. list-all (λw. {v,w} ∉ E) p2)
define env **where** env = (λv. filter is-indep (v#filter (λw. {v,w} ∈ E) (p1h#p1t)))
define cost **where** cost = (λv. (∑ w ← env v. 1 /(node-deg w+1)) − of-bool(is-indep v))
define w **where** w = arg-min-list cost p1

let ?e = environment
have perm: p1@p2 ∈ permutations-of-set V **using** 2(2) p1-def **by** auto
have dist: distinct p1 distinct p2 set p1 ∩ set p2 = {} set p1 ∪ set p2 = V
set p1 = V − set p2 set p2 = V − set p1
using permutations-of-setD[OF perm] **by** auto

have w-set: w ∈ set p1 **unfolding** w-def p1-def **using** arg-min-list-in **by** blast
have v-notin-p2: v ∉ set p2 **if** v ∈ set p1 **for** v **using** dist(5) that **by** auto

have is-indep: is-indep v = (environment {v} ∩ set p2 = {}) **if** v ∈ set p1 **for** v
unfolding is-indep-def list-all-iff environment-def vert-adj-def **using** v-notin-p2[OF that]
by (auto simp add:insert-commute)

have cost-correct: cost v = p-estimator p2 − p-estimator (p2@[v])
(**is** ?L = ?R) **if** v ∈ set p1 **for** v
proof −
have set (env v) = {x ∈ {v} ∪ {x ∈ set p1. {v, x} ∈ E}. is-indep x}
unfolding env-def p1-def[symmetric] **by** auto
also have ... = {x ∈ environment {v} ∩ set p1. is-indep x}
using that **unfolding** environment-def vert-adj-def **by** (auto simp:insert-commute)
also have ... = {x ∈ environment {v} ∩ set p1. set p2 ∩ environment {x} = {}}
using is-indep **by** auto
also have ... = environment {v} ∩ set p1 − environment (set p2)
by (subst environment-sym-2) auto
also have ... = environment {v} ∩ (V − set p2) − environment (set p2)
using environment-range dist(1−4) that
by (intro arg-cong2[**where** f=(−)] arg-cong2[**where** f=(∩)] refl) auto

also have ... = *environment* {*v*} ∩ *V* − *set p2* − *environment* (*set p2*) **by** *auto*
also have ... = *environment* {*v*} ∩ *V* − *environment* (*set p2*) **using** *environment-self* **by** *auto*
also have ... = *environment* {*v*} − *environment* (*set p2*)
using *that dist*(*4*) **by** (*intro arg-cong2*[**where** *f*=(−)] *refl Int-absorb2 environment-range*) *auto*
finally have *env-v*: *set* (*env v*) = *environment* {*v*} − *environment* (*set p2*) **by** *simp*

have {*v*,*v*} ∉ *E* **by** (*simp add*: *singleton-not-edge*)
hence *v* ∉ *set* (*filter* (λ*w*. {*v*, *w*} ∈ *E*) *p1*) **by** *simp*
hence *distinct* (*v* # *filter* (λ*w*. {*v*, *w*} ∈ *E*) *p1*) **using** *dist*(*1*) **by** *simp*
hence *dist-env-v*: *distinct* (*env v*)
unfolding *env-def p1-def*[*symmetric*] **using** *distinct-filter* **by** *blast*

have *?L* = (∑ *w*←*env v*. *1* / (*node-deg w* + *1*)) − *of-bool* (*is-indep v*)
unfolding *cost-def* **by** *simp*
also have ... = (∑ *w*←*env v*. *1* / (*node-deg w* + *1*)) − *of-bool*(*environment* {*v*} ∩ *set p2* = {})
by (*simp add*: *is-indep*[*OF that*])
also have ... = (∑ *w*←*env v*. *1* / (*degree w* + *1*)) − *of-bool*(*environment* {*v*} ∩ *set p2* = {})
unfolding *node-deg-def alt-degree-def incident-edges-def vincident-def* **by** (*simp add*:*ac-simps*)
also have ... = (∑ *v*∈*?e* {*v*}−*?e* (*set p2*). *1*/(*degree v*+*1*))−*of-bool*(*?e* {*v*} ∩ *set p2* = {})
by (*subst sum-list-distinct-conv-sum-set*[*OF dist-env-v*]) (*simp add*:*env-v*)
also have ... = − (*of-bool*(*?e* {*v*}∩*set p2*={})−(∑ *v*∈*?e* {*v*}−*?e* (*set p2*). *1*/(*degree v*+*1*)))
by (*simp add*:*algebra-simps*)
also have ... = − (*p-estimator* (*p2*@[*v*]) − *p-estimator* (*p2*))
using *that dist*(*2*−*4*) **by** (*intro arg-cong*[**where** *f*=λ*x*. −*x*] *p-estimator-step*[*symmetric*]) *auto*

also have ... = *?R* **by** (*simp add*:*algebra-simps*)
finally show *?thesis* **by** *simp*
qed

have *p1-ne*: *p1* ≠ [] **using** *p1-def* **by** *simp*

have *card* (*set p1*) ∗ *Min* (*cost* ‘ *set p1*) = (∑ *v* ∈ *set p1*. *Min* (*cost* ‘ *set p1*)) **by** *simp*
also have ... ≤ (∑ *v* ∈ *set p1*. *cost v*) **by** (*intro sum-mono*) *simp*
also have ... = (∑ *v* ∈ *set p1*. *p-estimator p2* − *p-estimator* (*p2*@[*v*]))
by (*intro sum.cong cost-correct refl*)
also have ... = (∑ *v* ∈ *V*−*set p2*. *p-estimator p2* − *p-estimator* (*p2*@[*v*]))
using *dist*(*1*−*4*) **by** (*intro sum.cong*) *auto*
also have ... = *card* (*V*−*set p2*) ∗ *p-estimator p2* − (∑ *v* ∈ *V*−*set p2*. *p-estimator* (*p2*@[*v*]))
unfolding *sum-subtractf* **by** *simp*
also have ... = *0* **using** *dist*(*5*)[*symmetric*] *p1-ne* **by** (*subst p-estimator-split*) *auto*
finally have *Min* (*cost* ‘ *set p1*) ≤ *0* **using** *p1-ne* **by** (*simp add*: *mult-le-0-iff*)
hence *cost-w-nonpos*: *cost w* ≤ *0* **unfolding** *w-def f-arg-min-list-f*[*OF p1-ne*] **by** *argo*

have *a*: *set* (*remove1 w p1* @ *p2* @ [*w*]) = *V*
using *w-set dist*(*4*) **by** (*auto simp*:*set-remove1-eq*[*OF dist*(*1*)])

have *b*: *distinct* (*remove1 w p1* @ *p2* @ [*w*])
using *dist*(*1*,*2*,*3*) *v-notin-p2*[*OF w-set*] **by** *auto*

have *c*: *strict-subseq* (*remove1 w p1*) *p1* **by** (*intro strict-subseq-remove1 w-set*)

have *p-estimator p2* ≤ *p-estimator p2* − *cost w* **using** *cost-w-nonpos* **by** *simp*
also have ... = *p-estimator* (*p2*@[*w*]) **unfolding** *cost-correct*[*OF w-set*] **by** *simp*
also have ... ≤ *length* (*derandomized-indep-set* (*remove1 w p1*) (*p2*@[*w*]) *E*)
using *c* **by** (*intro 2 a b permutations-of-setI*) (*auto simp*:*p1-def*)
also have ... = *real* (*length* (*derandomized-indep-set p1 p2 E*))

unfolding *p1-def derandomized-indep-set.simps node-deg-def*[*symmetric*] *is-indep-def*[*symmetric*]
by (*simp del*:*remove1.simps add*:*Let-def cost-def p1-def env-def w-def*)
finally show *?case* **by** (*simp add*:*p1-def*)
qed

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 $p \in$ *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*) $\geq$ ($\sum$ *v* $\in$ *V*. *1*/ (*degree v*+*1*))
length (*derandomized-indep-set p* [] *E*) $\geq$ *card V* / (*2*∗*card E* / *card V* + *1*)
proof −
let *?res* = *derandomized-indep-set p* [] *E*
show *is-independent-set* (*set ?res*) **using** *assms derandomized-indep-set-correct-aux* **by** *auto*
show *distinct ?res* **using** *assms derandomized-indep-set-correct-aux* **by** *auto*

have ($\sum$ *v* $\in$ *V*. *1*/ (*degree v*+*1*)) $\leq$ *p-estimator* []
by (*subst p-estimator*) (*simp-all add*:*environment-def is-first-def ac-simps*)
also have ... $\leq$ *length ?res* **using** *assms derandomized-indep-set-length-aux* **by** *auto*
finally show *a*: ($\sum$ *v* $\in$ *V*. *1*/ (*degree v*+*1*)) $\leq$ *length ?res* **by** *auto*
thus *card V* / (*2*∗*card E* / *card V* + *1*) $\leq$ *length ?res* **using** *caro-wei-aux* **by** *simp*
qed

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.