

# Derandomization with Conditional Expectations

Emin Karayel

February 6, 2026

## 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

<b>1</b>	<b>Some Preliminary Results</b>	<b>2</b>
1.1	On Probability Theory . . . . .	2
1.2	On Convexity . . . . .	3
1.3	On <i>subseq</i> and <i>strict-subseq</i> . . . . .	3
1.4	On Random Permutations . . . . .	4
1.5	On Finite Simple Graphs . . . . .	6
<b>2</b>	<b>Method of Conditional Expectations: Large Cuts</b>	<b>7</b>
<b>3</b>	<b>Method of Pessimistic Estimators: Independent Sets</b>	<b>12</b>

# 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**  $\text{bij-betw } (\lambda x. (f x, g x)) A (B \times C) A \neq \{\}$  *finite A*  
**shows**  $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } B$  (**is**  $?L = ?R$ )

**proof** –

**have**  $B \times C \neq \{\}$  **using** *assms(1,2)* **unfolding** *bij-betw-def* **by** *auto*

**hence**  $0: B \neq \{\}$   $C \neq \{\}$  **by** *auto*

**have** *finite*  $(B \times C)$

**unfolding** *bij-betw-imp-surj-on*[*OF assms(1), symmetric*] **by** (*intro finite-imageI assms(3)*)

**hence**  $1: \text{finite } B \text{ finite } C$

**using**  $0$  *finite-cartesian-productD1 finite-cartesian-productD2* **by** *auto*

**have**  $?L = \text{map-pmf } \text{fst } (\text{map-pmf } (\lambda x. (f x, g x)) (\text{pmf-of-set } A))$

**unfolding** *map-pmf-comp* **by** *simp*

**also have**  $\dots = \text{map-pmf } \text{fst } (\text{pmf-of-set } (B \times C))$

**by** (*intro arg-cong2[where f=map-pmf] map-pmf-of-set-bij-betw assms refl*)

**also have**  $\dots = \text{pmf-of-set } B$

**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**  $f :: - \Rightarrow \text{real}$

**assumes**  $\bigwedge x. x \in \text{set-pmf } (\text{bind-pmf } p q) \implies |f x| \leq M$

**shows**  $(\int x. f x \partial \text{bind-pmf } p q) = (\int x. \int y. f y \partial q x \partial p)$  (**is**  $?L = ?R$ )

**proof** –

**define** *clamp* **where**  $\text{clamp } x = (\text{if } |x| > M \text{ then } 0 \text{ else } x)$  **for**  $x$

**obtain**  $x$  **where**  $x \in \text{set-pmf } (\text{bind-pmf } p q)$  **using** *set-pmf-not-empty* **by** *fast*

**hence**  $M \geq 0$  **using** *assms* **by** *fastforce*

**have**  $a: \bigwedge x y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } (q x) \implies \neg |f y| > M$

**using** *assms* **by** *fastforce*

**hence**  $(\int x. f x \partial \text{bind-pmf } p q) = (\int x. \text{clamp } (f x) \partial \text{bind-pmf } p q)$

**unfolding** *clamp-def* **by** (*intro integral-cong-AE AE-pmfI*) *auto*

**also have**  $\dots = (\int x. \int y. \text{clamp } (f y) \partial q x \partial p)$  **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**  $\dots = ?R$  **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**  $A B :: 'x \text{ set}$

**assumes**  $A \cup B \neq \{\}$   $A \cap B = \{\}$  *finite*  $(A \cup B)$

**defines**  $p \equiv \text{real } (\text{card } A) / \text{real } (\text{card } A + \text{card } B)$

**shows**  $\text{pmf-of-set } (A \cup B) = \text{do } \{c \leftarrow \text{bernoulli-pmf } p; \text{pmf-of-set } (\text{if } c \text{ then } A \text{ else } B)\}$

(**is**  $?L = ?R$ )

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

**have**  $\text{pmf } ?L x = \text{indicator } (A \cup B) x / \text{card } (A \cup B)$  **using**  $\text{assms}$  **by** *simp*  
**also have**  $\dots = (\text{of-bool } (x \in A) + \text{of-bool } (x \in B)) / (\text{card } A + \text{card } B)$  **using**  $\text{assms}(1-3)$   
**by** (*intro arg-cong2[where f=(/)] arg-cong[where f=real] card-Un-disjoint*) *auto*  
**also have**  $\dots = (\text{pmf } (\text{pmf-of-set } A) x * \text{card } A + \text{pmf } (\text{pmf-of-set } B) x * \text{card } B) / (\text{card } A + \text{card } B)$   
**using**  $\text{assms}(3)$  **by** (*intro arg-cong2[where f=(/)] arg-cong2[where f=(+)] refl b*) *auto*  
**also have**  $\dots = \text{pmf } (\text{pmf-of-set } A) x * p + \text{pmf } (\text{pmf-of-set } B) x * (1 - p)$   
**unfolding**  $a$  **unfolding**  $p\text{-def}$  **by** (*simp add: divide-simps*)  
**also have**  $\dots = \text{pmf } ?R x$  **using**  $p\text{-range}$  **by** (*simp add: pmf-bind*)  
**finally show**  $\text{pmf } ?L x = \text{pmf } ?R x$  **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  $\geq$  (c::real)*  
**shows**  $\exists x \in \text{set-pmf } M. f x \geq c$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\exists x \in \text{set-pmf } M. c \leq f x)$   
**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**  $f :: ('b :: \text{real-vector}) \Rightarrow \text{real}$   
**assumes** *convex-on S f convex S*  
**shows** *convex-on {x. x + a  $\in$  S} ( $\lambda x. f (x+a)$ )*  
**proof** –  
**have**  $f (((1-t) *_R x + t *_R y) + a) \leq (1-t) * f (x+a) + t * f (y+a)$  (**is**  $?L \leq ?R$ )  
**if**  $0 < t < 1$   $x \in \{x. x + a \in S\}$   $y \in \{x. x + a \in S\}$  **for**  $x y t$   
**proof** –  
**have**  $?L = f ((1-t) *_R (x+a) + t *_R (y+a))$  **by** (*simp add: algebra-simps*)  
**also have**  $\dots \leq (1-t) * f (x+a) + t * f (y+a)$  **using** *that* **by** (*intro convex-onD[OF assms(1)]*)  
*auto*  
**finally show** *?thesis* **by** *auto*  
**qed**  
**moreover have**  $\{x. x + a \in S\} = (\lambda x. x - a) ' S$  **by** (*auto simp: image-iff algebra-simps*)  
**hence** *convex {x. x + a  $\in$  S}* **using**  $\text{assms}(2)$  **by** *auto*  
**ultimately show** *?thesis* **using**  $\text{assms}$  **by** (*intro convex-onI*) *auto*  
**qed**

## 1.3 On subseq and strict-subseq

**lemma** *strict-subseq-imp-shorter*: *strict-subseq x y  $\implies$  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  $\implies$  distinct y  $\implies$  distinct x*  
**by** (*metis distinct-nthsI subseq-conv-nths*)

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

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

**lemma** *subseq-induct*:

**assumes**  $\bigwedge ys. (\bigwedge zs. \text{strict-subseq } zs \ ys \implies P \ zs) \implies P \ ys$   
**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**  $\bigwedge y \ ys. (\bigwedge zs. \text{strict-subseq } zs \ (y\#\ ys) \implies P \ zs) \implies P \ (y\#\ ys)$

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

**lemma** *permutations-of-set-prefix*:

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

**proof** –

**have**  $S \neq \{\}$  **using**  $\text{assms}(2)$  **by**  $\text{auto}$   
**have**  $?L = (\int vs. \text{indicator } \{vs. \text{prefix } [v] \ vs\} \ vs \ \partial \text{pmf-of-set } (\text{permutations-of-set } S))$  **by**  $\text{simp}$   
**also have**  $\dots = (\int h. \text{of-bool } (v = h) \ \partial \text{pmf-of-set } S)$   
**unfolding**  $\text{random-permutation-of-set}[OF \text{ assms}(1) \ S \text{-ne}]$   
**apply**  $(\text{subst integral-bind-pmf}[\text{where } M=1], \text{simp})$   
**apply**  $(\text{subst integral-bind-pmf}[\text{where } M=1], \text{simp})$   
**by**  $(\text{simp add:indicator-def})$   
**also have**  $\dots = (\int h. \text{indicator } \{v\} \ h \ \partial \text{pmf-of-set } S)$  **by**  $(\text{simp add:indicator-def eq-commute})$   
**also have**  $\dots = \text{measure } (\text{pmf-of-set } S) \ \{v\}$  **by**  $\text{simp}$   
**also have**  $\dots = 1 / \text{card } S$  **using**  $\text{assms}(1,2) \ S \text{-ne}$  **by**  $(\text{subst measure-pmf-of-set}) \ \text{auto}$   
**finally show**  $?thesis$  **by**  $\text{simp}$

**qed**

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

**definition** *cond-perm* **where**  $\text{cond-perm } V \ p = (@) \ p \ \text{'permutations-of-set } (V - \text{set } p)$

**context** *fin-sgraph*

**begin**

**lemma** *perm-non-empty-finite*:

$\text{permutations-of-set } V \neq \{\}$  *finite*  $(\text{permutations-of-set } V)$

**proof** –

**have**  $0 < \text{card } (\text{permutations-of-set } V)$  **using**  $\text{fin } V$  **by**  $(\text{subst card-permutations-of-set}) \ \text{auto}$   
**thus**  $\text{permutations-of-set } V \neq \{\}$  *finite*  $(\text{permutations-of-set } V)$  **using**  $\text{card-gt-0-iff}$  **by**  $\text{blast+}$   
**qed**

**lemma** *cond-perm-non-empty-finite*:

$\text{cond-perm } V \ p \neq \{\}$  *finite*  $(\text{cond-perm } V \ p)$

**proof** –

**have**  $0 < \text{card } (\text{permutations-of-set } (V - \text{set } p))$   
**using**  $\text{fin } V$  **by**  $(\text{subst card-permutations-of-set}) \ \text{auto}$   
**also have**  $\dots = \text{card } (\text{cond-perm } V \ p)$   
**unfolding**  $\text{cond-perm-def}$  **by**  $(\text{intro card-image}[symmetric] \ \text{inj-on } I) \ \text{auto}$   
**finally have**  $\text{card } (\text{cond-perm } V \ p) > 0$  **by**  $\text{simp}$   
**thus**  $\text{cond-perm } V \ p \neq \{\}$  *finite*  $(\text{cond-perm } V \ p)$  **using**  $\text{card-ge-0-finite}$  **by**  $\text{auto}$   
**qed**

**lemma** *cond-perm-alt*:

**assumes**  $\text{distinct } p \ \text{set } p \subseteq V$

**shows**  $\text{cond-perm } V \ p = \{xs \in \text{permutations-of-set } V. \text{prefix } p \ xs\}$

**proof** –

**have**  $p @ xs \in \text{permutations-of-set } V$  **if**  $xs \in \text{permutations-of-set } (V - \text{set } p)$  **for**  $xs$   
**using**  $\text{permutations-of-set } D[OF \ \text{that}] \ \text{assms}$  **by**  $(\text{intro permutations-of-set } I) \ \text{auto}$   
**moreover have**  $xs \in \text{cond-perm } V \ p$  **if**  $xs \in \text{permutations-of-set } V$  **and**  $a:\text{prefix } p \ xs$  **for**  $xs$   
**proof** –

**obtain**  $ys$  **where**  $xs-def:xs = p@ys$  **using**  $a\ prefix E$  **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**  $ys \in permutations-of-set (V - set\ p)$  **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**  $distinct\ p$   $set\ p \subseteq V$   $xs \in cond-perm\ V\ p$   
**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**  $degree-sum$ :  $(\sum v \in V. degree\ v) = 2 * card\ E$  **(is**  $?L = ?R$ **)**  
**proof** –  
**have**  $?L = (\sum v \in V. (\sum e \in E. of-bool(v \in e)))$   
**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**  $... = (\sum e \in E. card\ (e \cap V))$   
**using**  $fin-edges\ finV$  **by**  $(subst\ sum.swap)$   $(simp\ add:of-bool-def\ sum.If-cases\ Int-commute)$   
**also** **have**  $... = (\sum e \in E. card\ e)$   
**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**  $environment\ S = S \cup \{v. \exists s \in S. vert-adj\ v\ s\}$

**lemma**  $finite-environment$ :  
**assumes**  $finite\ S$   
**shows**  $finite\ (environment\ S)$   
**proof** –  
**have**  $environment\ S \subseteq S \cup V$  **unfolding**  $environment-def$  **using**  $vert-adj-imp-inV$  **by**  $auto$   
**thus**  $?thesis$  **using**  $assms\ finite-Un\ finV\ finite-subset$  **by**  $auto$   
**qed**

**lemma**  $environment-mono$ :  $S \subseteq T \implies environment\ S \subseteq environment\ T$   
**unfolding**  $environment-def$  **by**  $auto$

**lemma**  $environment-sym$ :  $x \in environment\ \{y\} \longleftrightarrow y \in environment\ \{x\}$   
**unfolding**  $environment-def\ vert-adj-def$  **by**  $(auto\ simp:insert-commute)$

**lemma**  $environment-self$ :  $S \subseteq environment\ S$  **unfolding**  $environment-def$  **by**  $auto$

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

**proof** –  
**have**  $False$  **if**  $B \cap environment\ A = \{\}$   $x \in A \cap environment\ B$  **for**  $x\ A\ B$   
**proof**  $(cases\ x \in B)$   
**case**  $True$  **thus**  $?thesis$  **using**  $that\ environment-self$  **by**  $auto$   
**next**  
**case**  $False$   
**hence**  $x \in \{x. \exists v \in B. vert-adj\ x\ v\}$  **using**  $that(2)$  **unfolding**  $environment-def$  **by**  $auto$

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

**lemma**  $\text{environment-range}$ :  $S \subseteq V \implies \text{environment } S \subseteq V$   
**unfolding**  $\text{environment-def}$  **using**  $\text{vert-adj-imp-inV}$  **by**  $\text{auto}$

**lemma**  $\text{environment-union}$ :  $\text{environment } (S \cup T) = \text{environment } S \cup \text{environment } T$   
**unfolding**  $\text{environment-def}$  **by**  $\text{auto}$

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

**proof** –

**have**  $?L = \text{card } (\text{insert } v \ \{x. \{x, v\} \in E\})$  **unfolding**  $\text{environment-def}$   $\text{vert-adj-def}$  **by**  $\text{simp}$   
**also have**  $\dots = \text{Suc } (\text{card } \{x. \{x, v\} \in E\})$   
**by** ( $\text{intro } \text{card-insert-disjoint}$   $\text{finite-subset}[OF - \text{finV}]$ )  
 $(\text{auto } \text{simp}:\text{singleton-not-edge}$   $\text{wellformed-alt-fst})$   
**also have**  $\dots = \text{Suc } (\text{card } (\text{neighborhood } v))$  **unfolding**  $\text{neighborhood-def}$   $\text{vert-adj-def}$   
**by** ( $\text{intro } \text{arg-cong}[\text{where } f = \lambda x. \text{Suc } (\text{card } x)]$ )  
 $(\text{auto } \text{simp}:\text{wellformed-alt-fst}$   $\text{insert-commute})$   
**also have**  $\dots = \text{Suc } (\text{degree } v)$   
**unfolding**  $\text{alt-degree-def}$   $\text{card-incident-sedges-neighborhood}$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$

**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**  $\text{Derandomization-Conditional-Expectations-Cut}$   
**imports**  $\text{Derandomization-Conditional-Expectations-Preliminary}$   
**begin**

**context**  $\text{fin-sgraph}$   
**begin**

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

**lemma**  $\text{eval-cond-edge}$ :

**assumes**  $L \subseteq U$   $\text{finite } U$   $e \in E$   
**shows**  $(\int C. \text{of-bool } (e \cap C \neq \{\} \wedge e - C \neq \{\}) \ \partial \text{pmf-of-set } \{C. L \subseteq C \wedge C \subseteq U\}) =$   
 $((\text{if } e \subseteq -U \cup L \text{ then } \text{of-bool}(e \cap L \neq \{\} \wedge e \cap -U \neq \{\}) :: \text{real else } 1/2))$   
**(is**  $?L = ?R$ )

**proof** –

**obtain**  $e1 \ e2$  **where**  $e$ -def:  $e = \{e1, e2\}$   $e1 \neq e2$  **using**  $\text{two-edges}[OF \ \text{assms}(3)]$   
**by** ( $\text{meson } \text{card-2-iff}$ )

**let**  $?sing\text{-iff} = (\lambda x \ e. (\text{if } x \text{ then } \{e\} \text{ else } \{\}))$

**define**  $R1$  **where**  $R1 = (if\ e1 \in L\ then\ \{True\}\ else\ (if\ e1 \in U - L\ then\ \{False, True\}\ else\ \{False\}))$

**define**  $R2$  **where**  $R2 = (if\ e2 \in L\ then\ \{True\}\ else\ (if\ e2 \in U - L\ then\ \{False, True\}\ else\ \{False\}))$

**have**  $bij$ :  $bij\ betw\ (\lambda x. ((e1 \in x, e2 \in x), x - \{e1, e2\}))\ \{C. L \subseteq C \wedge C \subseteq U\}$

$((R1 \times R2) \times \{C. L - \{e1, e2\} \subseteq C \wedge C \subseteq U - \{e1, e2\}\})$

**unfolding**  $R1\ def\ R2\ def$  **using**  $e\ def(2)$   $assms(1)$

**by**  $(intro\ bij\ betwI[\mathbf{where}\ g = (\lambda((a, b), x). x \cup ?sing\ iff\ a\ e1 \cup ?sing\ iff\ b\ e2)])$

$(auto\ split:if\ split\ asm)$

**have**  $r$ :  $map\ pmf\ (\lambda x. (e1 \in x, e2 \in x))\ (pmf\ of\ set\ \{C. L \subseteq C \wedge C \subseteq U\}) = pmf\ of\ set\ (R1 \times R2)$

**using**  $assms(1, 2)$   $map\ pmf\ of\ set\ bij\ betw\ 2[OF\ bij]$  **by**  $auto$

**have**  $?L = \int C. of\ bool\ ((e1 \in C) \neq (e2 \in C))\ \partial(pmfmf\ of\ set\ \{C. L \subseteq C \wedge C \subseteq U\})$

**unfolding**  $e\ def(1)$  **using**  $e\ def(2)$  **by**  $(intro\ integral\ cong\ AE\ AE\ pmfI)\ auto$

**also** **have**  $\dots = \int x. of\ bool(fst\ x \neq snd\ x)\ \partial pmf\ of\ set\ (R1 \times R2)$

**unfolding**  $r[symmetric]$  **by**  $simp$

**also** **have**  $\dots = (if\ \{e1, e2\} \subseteq -U \cup L\ then\ of\ bool(\{e1, e2\} \cap L \neq \{\} \wedge \{e1, e2\} \cap -U \neq \{\})\ else\ 1/2)$

**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**  $\dots = ?R$  **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**  $exp\ cut\ size$ :

$(\int C. real\ (cut\ size\ C)\ \partial pmf\ of\ set\ (Pow\ V)) = real\ (card\ E) / 2$  **(is**  $?L = ?R)$

**proof** –

**have**  $a:False$  **if**  $x \in E\ x \subseteq -V$  **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**  $?L = (\int C. (\sum e \in E. of\ bool\ (e \cap C \neq \{\} \wedge e - C \neq \{\}))\ \partial pmf\ of\ set\ (Pow\ V))$

**using**  $fin\ edges$  **by**  $(simp\ all\ add:of\ bool\ def\ cut\ size\ def\ sum.If\ cases\ Int\ def)$

**also** **have**  $\dots = (\sum e \in E. (\int C. of\ bool\ (e \cap C \neq \{\} \wedge e - C \neq \{\}))\ \partial pmf\ of\ set\ (Pow\ V))$

**using**  $finV$  **by**  $(intro\ Bochner\ Integration.integral\ sum\ integrable\ measure\ pmf\ finite)$

$(simp\ add: Pow\ not\ empty)$

**also** **have**  $\dots = (\sum e \in E. (\int C. of\ bool\ (e \cap C \neq \{\} \wedge e - C \neq \{\}))\ \partial pmf\ of\ set\ \{C. \{\} \subseteq C \wedge C \subseteq V\})$

**unfolding**  $Pow\ def$  **by**  $simp$

**also** **have**  $\dots = (\sum e \in E. (if\ e \subseteq -V \cup \{\}\ then\ of\ bool\ (e \cap \{\} \neq \{\} \wedge e \cap -V \neq \{\})\ else\ 1 / 2))$

**by**  $(intro\ sum.cong\ eval\ cond\ edge\ finV)\ auto$

**also** **have**  $\dots = (\sum e \in E. 1/2)$  **using**  $a$  **by**  $(intro\ sum.cong)\ auto$

**also** **have**  $\dots = ?R$  **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**  $exists\ cut$ :  $\exists C \subseteq V. real\ (cut\ size\ C) \geq card\ E / 2$

**proof** –

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

**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  $\Rightarrow$  'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set set  $\Rightarrow$  '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**  $\text{cond-exp } R B = (\int C. \text{real} (\text{cut-size } C) \partial \text{pmf-of-set} \{C. R \subseteq C \wedge C \subseteq V - B\})$

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**  $R \subseteq V \ B \subseteq V \ R \cap B = \{v\} \ v \in V - R - B$

**shows**  $\text{cond-exp } R B = (\text{cond-exp } (R \cup \{v\}) B + \text{cond-exp } R (B \cup \{v\})) / 2$  (**is**  $?L = ?R$ )

**proof** –

**let**  $?A = \{C. R \cup \{v\} \subseteq C \wedge C \subseteq V - B\}$

**let**  $?B = \{C. R \subseteq C \wedge C \subseteq V - (B \cup \{v\})\}$

**define**  $p$  **where**  $p = \text{real} (\text{card } ?A) / (\text{card } ?A + \text{card } ?B)$

**have**  $a: \{C. R \subseteq C \wedge C \subseteq V - B\} = ?A \cup ?B$  **using** *assms* **by**  $\text{auto}$

**have**  $b: ?A \cap ?B = \{v\}$  **using** *assms* **by**  $\text{auto}$

**have**  $c: \text{finite} (?A \cup ?B)$  **using**  $\text{fin } V$  **by**  $\text{auto}$

**have**  $R \cup \{v\} \subseteq V - B$  **using** *assms* **by**  $\text{auto}$

**hence**  $g: ?A \neq \{v\}$  **by**  $\text{auto}$

**hence**  $d: ?A \cup ?B \neq \{v\}$  **by** *simp*

**have**  $e: \text{real} (\text{cut-size } x) \leq \text{real} (\text{card } E)$  **for**  $x$

**unfolding** *cut-size-def* **by**  $(\text{intro of-nat-mono card-mono fin-edges}) \text{ auto}$

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

**lemma**  $\text{cond-exp-cut-size}$ :

**assumes**  $R \subseteq V \ B \subseteq V \ R \cap B = \{\}$   
**shows**  $\text{cond-exp } R \ B = \text{real } (\text{card } \{e \in E. e \cap R \neq \{\} \wedge e \cap B \neq \{\}\}) + \text{real } (\text{card } \{e \in E. e \cap V - R - B \neq \{\}\}) / 2$   
**(is**  $?L = ?R$ **)**

**proof** –

**have**  $a:\text{finite } \{C. R \subseteq C \wedge C \subseteq V - B\} \ \{C. R \subseteq C \wedge C \subseteq V - B\} \neq \{\}$  **using**  $\text{finV } \text{assms}$   
**by**  $\text{auto}$

**have**  $b:e \subseteq -V \cup B \cup R$  **if**  $\text{cthat}: e \in E \ e \cap R \neq \{\} \ e \cap B \neq \{\}$  **for**  $e$

**proof** –

**obtain**  $e1$  **where**  $e1: e1 \in e \cap R$  **using**  $\text{cthat}(2)$  **by**  $\text{auto}$

**obtain**  $e2$  **where**  $e2: e2 \in e \cap B$  **using**  $\text{cthat}(3)$  **by**  $\text{auto}$

**have**  $e1 \neq e2$  **using**  $e1 \ e2 \ \text{assms}(3)$  **by**  $\text{auto}$

**hence**  $\text{card } \{e1, e2\} = 2$  **by**  $\text{auto}$

**hence**  $e = \{e1, e2\}$  **using**  $\text{two-edges}[OF \ \text{cthat}(1)] \ e1 \ e2$

**by** ( $\text{intro } \text{card-seteq}[\text{symmetric}]$ ) ( $\text{auto } \text{intro}!: \text{card-ge-0-finite}$ )

**thus**  $?thesis$  **using**  $e1 \ e2$  **by**  $\text{simp}$

**qed**

**have**  $?L = (\int C. (\sum e \in E. \text{of-bool } (e \cap C \neq \{\} \wedge e - C \neq \{\})) \ \partial \text{pmf-of-set } \{C. R \subseteq C \wedge C \subseteq V - B\})$

**unfolding**  $\text{cond-exp-def}$  **using**  $\text{fin-edges}$

**by** ( $\text{simp-all } \text{add:of-bool-def } \text{cut-size-def } \text{sum.If-cases } \text{Int-def}$ )

**also have**  $\dots = (\sum e \in E. (\int C. \text{of-bool } (e \cap C \neq \{\} \wedge e - C \neq \{\})) \ \partial \text{pmf-of-set } \{C. R \subseteq C \wedge C \subseteq V - B\})$

**using**  $a$  **by** ( $\text{intro } \text{Bochner-Integration.integral-sum } \text{integrable-measure-pmf-finite}$ )  $\text{auto}$

**also have**  $\dots = (\sum e \in E. ((\text{if } e \subseteq -(V - B) \cup R \text{ then } \text{of-bool}(e \cap R \neq \{\} \wedge e \cap -(V - B) \neq \{\}))::\text{real } \text{else } 1/2))$

**using**  $\text{finV } \text{assms}(1,3)$  **by** ( $\text{intro } \text{sum.cong } \text{eval-cond-edge}$ )  $\text{auto}$

**also have**  $\dots = \text{real } (\text{card } \{e \in E. e \subseteq -V \cup B \cup R \wedge e \cap R \neq \{\} \wedge e \cap -(V - B) \neq \{\}\}) + \text{real } (\text{card } \{e \in E. \neg e \subseteq -V \cup B \cup R\}) / 2$

**using**  $\text{fin-edges}$  **by** ( $\text{simp } \text{add: } \text{sum.If-cases } \text{of-bool-def } \text{Int-def}$ )

**also have**  $\dots = ?R$  **using**  $\text{wellformed } \text{assms } b$

**by** ( $\text{intro } \text{arg-cong}[\text{where } f=\text{card}] \ \text{arg-cong2}[\text{where } f=(+)] \ \text{arg-cong}[\text{where } f=\text{real}]$

$\text{arg-cong2}[\text{where } f=(/)] \ \text{refl } \text{Collect-cong } \text{order-antisym}$ )  $\text{auto}$

**finally show**  $?thesis$  **by**  $\text{simp}$

**qed**

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

**theorem**  $\text{derandomized-max-cut}$ :

**assumes**  $vs \in \text{permutations-of-set } V$

**defines**  $C \equiv \text{derandomized-max-cut } vs \ \{\} \ \{\} \ E$

**shows**  $C \subseteq V \ 2 * \text{cut-size } C \geq \text{card } E$

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  $R B \leq \text{cond-exp } R (B \cup \{vh\})$  unfolding split using  $d$  True by simp
thus ?thesis using True Cons(1)[OF dvt 0] by simp
next
  case False
  have  $0:\text{set } vt \cap (R \cup \{vh\}) = \{\}$  set  $vt \cap B = \{\}$   $(R \cup \{vh\}) \cap B = \{\} \cup \{\text{set } vt, R \cup \{vh\}, B\} = V$ 
    using Cons(2-6) by auto
  have cond-exp  $R B \leq \text{cond-exp } (R \cup \{vh\}) B$  unfolding split using  $d$  False by simp
  thus ?thesis using False Cons(1)[OF dvt 0] by simp
qed
qed
moreover have  $e \cap V \neq \{\}$  if  $e \in E$  for  $e$ 
  using Int-absorb2[OF wellformed[OF that]] two-edges[OF that] by auto
hence  $\{e \in E. e \cap V \neq \{\}\} = E$  by auto
hence cond-exp  $\{\} \{\} = \text{graph-size} / 2$  by (subst cond-exp-cut-size) auto
ultimately show  $C \subseteq V \ 2 * \text{cut-size } C \geq \text{card } E$ 
  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  $\Rightarrow$  'a set set  $\Rightarrow$  'a list
  where
    indep-set []  $E = []$  |
    indep-set ( $v\#vt$ )  $E = v\#\text{indep-set } (\text{filter } (\lambda w. \{v,w\} \notin E) vt) E$ 

```

```

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  $(\lambda w. \{ph, w\} \notin E) pt$ )  $pt$  by simp

```

```

  also have strict-subseq ... ( $ph\#pt$ ) unfolding strict-subseq-def by auto

```

```

  finally have strict-subseq (filter  $(\lambda w. \{ph, w\} \notin E) pt$ ) ( $ph \# pt$ ) by simp

```

```

  hence subseq (indep-set ( $ph \# pt$ )  $E$ ) ( $ph\#\text{filter } (\lambda w. \{ph, w\} \notin E) pt$ )

```

```

    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  $\exists z \in (set (filter (\lambda w. \{y, w\} \notin E) ys)). \{z, y\} \in E$

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**  $q\text{-def}: q = q1@v\#q2$  **using**  $assms(4)$  **by** (*meson split-list*)  
**obtain**  $p1\ p2$  **where**  $p\text{-def}: p = p1@p2\ subseq\ q1\ p1\ subseq\ (v\#q2)\ p2$   
**using**  $assms(3)$  *list-emb-appendD* **unfolding**  $q\text{-def}$  **by** *blast*  
  
**have**  $v \in set\ p2$  **using**  $p\text{-def}(3)$  *list-emb-set* **by** *force*  
**hence**  $0:v \notin set\ p1$  **using**  $assms(2)$  **unfolding**  $p\text{-def}(1)$  **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*  $p\text{-def}(1)$  **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**  $p\text{-def}(2)$  **by** (*metis (full-types) filter-empty-conv list-emb-set*)  
**moreover have**  $v \in environment\ \{v\}$  **unfolding** *environment-def* **by** *simp*  
**ultimately show**  $?thesis$  **unfolding**  $q\text{-def}\ is\text{-first}\text{-def}$  **by** *simp*  
**qed**

**lemma** *is-first-imp-in-set:*

**assumes** *is-first*  $v\ p$   
**shows**  $v \in set\ p$

**proof** –

**have**  $v \in set\ (filter\ (\lambda y. y \in environment\ \{v\})\ p)$   
**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*  $p\ set\ p \subseteq V\ is\text{-first}\ v\ p$   
**shows**  $v \in set\ (indep\text{-set}\ p\ E)$   
**using**  $assms$

**proof** (*induction p rule:subseq-induct*)

**case** ( $1\ ys$ )

**hence**  $i:v \in set\ (indep\text{-set}\ zs\ E)$  **if**  $is\text{-first}\ v\ zs\ strict\text{-subseq}\ zs\ ys$  **for**  $zs$   
**using** *strict-subseq-imp-distinct strict-subseq-set that* **by** (*intro 1(1)*) *blast+*

**define**  $ysh\ yst$  **where**  $ysht\text{-def}: ysh = hd\ ys\ yst = tl\ ys$

**have**  $split\text{-ys}: ys = ysh\#\ yst$  **if**  $ys \neq []$  **using** *that* **unfolding**  $ysht\text{-def}$  **by** *auto*

**consider** ( $a$ )  $ys = []$  | ( $b$ )  $ys \neq []\ hd\ ys = v$  | ( $c$ )  $ys \neq []\ hd\ ys \neq v$  **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**  $split\text{-ys}[OF\ b(1)]$  **by** *simp*

**next**

**case**  $c$

**have**  $0:subseq\ (filter\ (\lambda w. \{ysh, w\} \notin E)\ yst)\ ys$  **unfolding**  $split\text{-ys}[OF\ c(1)]$  **by** *auto*

**have**  $v \in set\ ys$  **using**  $1(4)$  *is-first-imp-in-set* **by** *auto*

**hence**  $v \in set\ yst$  **using**  $c$  **unfolding**  $split\text{-ys}[OF\ c(1)]$  **by** *simp*

**moreover have**  $ysh \neq v$  **using**  $c(2)$   $split\text{-ys}[OF\ c(1)]$  **by** *simp*

**hence**  $ysh \notin environment\ \{v\}$  **using**  $1(4)$  **unfolding** *is-first-def*  $split\text{-ys}[OF\ c(1)]$  **by** *auto*

**hence**  $\{ysh, v\} \notin E$  **unfolding** *environment-def vert-adj-def* **by** *auto*

**ultimately have**  $v \in \text{set } (\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst})$  **by** *simp*  
**hence** *is-first*  $v$   $(\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst})$  **by**  $(\text{intro } \text{is-first-subseq}[OF 1(4)] 0 1(2))$   
**moreover have**  $\text{length } \text{yst} < \text{length } \text{ys}$  **using** *split-ys* $[OF c(1)]$  **by** *auto*  
**hence**  $\text{length } (\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst}) < \text{length } \text{ys}$   
**using** *length-filter-le dual-order.strict-trans2* **by** *blast*  
**hence**  $\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst} \neq \text{ys}$  **by** *auto*  
**hence** *strict-subseq*  $(\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst}) \text{ ys}$   
**using** *0 unfolding strict-subseq-def* **by** *auto*  
**ultimately have**  $v \in \text{set } (\text{indep-set } (\text{filter } (\lambda w. \{ysh, w\} \notin E) \text{ yst}) E)$  **by**  $(\text{intro } i)$   
**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**  $\Omega \equiv \text{pmf-of-set } (\text{permutations-of-set } V)$   
**shows**  $(\int \text{vs. real } (\text{length } (\text{indep-set } \text{vs } E)) \partial\Omega) \geq (\sum v \in V. 1 / (\text{degree } v + 1::\text{real}))$   
**(is**  $?L \geq ?R$ **)**

**proof** –

**let**  $?perm = (\lambda x. \text{pmf-of-set } (\text{permutations-of-set } x))$   
**have**  $a:\text{finite } (\text{set-pmf } \Omega)$  **unfolding**  $\Omega\text{-def}$  **using** *perm-non-empty-finite* **by** *simp*  
**have**  $b:\text{distinct } y \text{ set } y \subseteq V$  **if**  $y \in \text{set-pmf } \Omega$  **for**  $y$   
**using** *that perm-non-empty-finite permutations-of-setD* **unfolding**  $\Omega\text{-def}$  **by** *auto*  
  
**have**  $?R = (\sum v \in V. 1 / \text{real } (\text{card } (\text{environment } \{v\})))$  **unfolding** *card-environment* **by** *simp*  
**also have**  $\dots = (\sum v \in V. \text{measure } (?perm (\text{environment } \{v\})) \{vs. \text{prefix}[v] \text{vs}\})$   
**using** *finite-environment environment-self* **by**  $(\text{intro } \text{sum.cong permutations-of-set-prefix}[symmetric])$   
*auto*  
**also have**  $\dots = (\sum v \in V. (\int \text{vs. indicator } \{vs. \text{prefix } [v] \text{vs}\} \text{vs } \partial ?perm (\text{environment } \{v\} \cap V)))$   
**using** *Int-absorb2* $[OF \text{environment-range}]$  **by**  $(\text{intro } \text{sum.cong refl})$  *simp*  
**also have**  $\dots = (\sum v \in V. (\int \text{vs. of-bool}(\text{prefix}[v] \text{vs}) \partial \text{map-pmf } (\text{filter } (\lambda x. x \in \text{environment } \{v\})))$   
 $\Omega)$   
**unfolding**  $\Omega\text{-def}$  *filter-permutations-of-set-pmf* $[OF \text{fin } V]$   
**by**  $(\text{intro } \text{sum.cong arg-cong2}[\text{where } f = \text{measure-pmf.expectation}])$   
*(simp-all add:Int-def conj-commute of-bool-def indicator-def)*  
**also have**  $\dots = (\sum v \in V. (\int \text{vs. of-bool}(\text{is-first } v \text{vs}) \partial\Omega))$   
**unfolding** *is-first-def* **by**  $(\text{intro } \text{sum.cong})$  *simp-all*  
**also have**  $\dots = (\int \text{vs. } (\sum v \in V. \text{of-bool}(\text{is-first } v \text{vs}) \partial\Omega)$   
**by**  $(\text{intro } \text{integral-sum}[symmetric])$  *integrable-measure-pmf-finite* $[OF a]$   
**also have**  $\dots \leq (\int \text{vs. real } (\text{card } (\text{set } (\text{indep-set } \text{vs } E))) \partial\Omega)$   
**using** *fin V b* **by**  $(\text{intro } \text{integral-mono-AE AE-pmfI integrable-measure-pmf-finite}[OF a])$   
*(auto intro!:card-mono set-indep-set)*  
**also have**  $\dots \leq ?L$   
**by**  $(\text{intro } \text{integral-mono-AE AE-pmfI integrable-measure-pmf-finite}[OF a])$  *of-nat-mono card-length*  
**finally show** *?thesis* **by** *simp*  
**qed**

The function  $\lambda x. 1 / (x + 1)$  is convex.

**lemma** *inverse-x-plus-1-convex*: *convex-on*  $\{-1 < ..\}$   $(\lambda x. 1 / (x+1::\text{real}))$

**proof** –

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

**lemma** *caro-wei-aux*:  $\text{card } V / (2 * \text{card } E / \text{card } V + 1) \leq (\sum v \in V. 1 / (\text{degree } v + 1))$

**proof** –

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

$\exists S \subseteq V. \text{is-independent-set } S \wedge \text{card } S \geq \text{card } V / (2 * \text{card } E / \text{card } V + 1)$

**proof** –

**let**  $? \Omega = \text{pmf-of-set } (\text{permutations-of-set } V)$   
**let**  $?w = \text{real } (\text{card } V) / (\text{real } (2 * \text{card } E) / \text{card } V + 1)$   
  
**have** *a:finite* (*set-pmf*  $? \Omega$ ) **using** *perm-non-empty-finite* **by** *simp*  
  
**have**  $(\int vs. \text{real } (\text{length } (\text{indep-set } vs \ E)) \ \partial ? \Omega) \geq ?w$   
**using** *exp-indep-set caro-wei-aux* **by** *simp*  
**then obtain** *vs* **where** *vs-def*:  $vs \in \text{set-pmf } ? \Omega$   $\text{real } (\text{length } (\text{indep-set } vs \ E)) \geq ?w$   
**using** *exists-point-above-expectation integrable-measure-pmf-finite[OF a]* **by** *blast*  
**define** *S* **where**  $S = \text{set } (\text{indep-set } vs \ E)$   
  
**have** *vs-range*:  $\text{distinct } vs \ \text{set } vs \subseteq V$   
**using** *vs-def(1)* *perm-non-empty-finite permutations-of-setD* **by** *auto*  
  
**have**  $b:S \subseteq V$  *is-independent-set* *S* **and** *c*:  $\text{distinct } (\text{indep-set } vs \ E)$   
**unfolding** *S-def* **using** *indep-set-correct[OF vs-range]* **by** *auto*  
  
**have**  $\text{real } (\text{card } S) = \text{length } (\text{indep-set } vs \ E)$  **using** *c* *distinct-card* **unfolding** *S-def* **by** *auto*  
**also have**  $\dots \geq ?w$  **using** *vs-def(2)* **by** *auto*  
**finally have**  $\text{real } (\text{card } S) \geq ?w$  **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

$\text{measure-pmf.expectation } (\text{pmf-of-set } (\text{permutations-of-set } V)) (\lambda vs. \sum v \in V. \text{of-bool } (\text{is-first } v \ vs))$

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 = ( $\lambda v$ . real (card {e  $\in$  E. v  $\in$  e}));
      is-indep = ( $\lambda v$ . list-all ( $\lambda w$ . {v,w  $\notin$  E} p));
      env = ( $\lambda v$ . filter is-indep (v#filter ( $\lambda w$ . {v,w  $\in$  E} (vh#vt))));
      cost = ( $\lambda v$ . ( $\sum w \leftarrow \text{env } v$ . 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 ( $\lambda x$ . 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 ( $\lambda x$ . 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 ( $\lambda y$ . 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 ( $\lambda y$ . y  $\in$  environment {v}) p  $\neq$  []
    using environment-sym unfolding filter-empty-conv by meson
  thus ?thesis unfolding is-first-def by (cases filter ( $\lambda y$ . 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**

$$p\text{-estimator } p = (\int vs. (\sum v \in V. \text{of-bool}(\text{is-first } v \text{ vs})) \partial \text{pmf-of-set } (\text{cond-perm } V \text{ } p))$$

**lemma** *p-estimator-split*:

```

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

```

?R)

**proof** –

**let** ?q =  $\lambda x. \text{pmf-of-set } (\text{permutations-of-set } (V\text{-set } p\text{-}\{x\}))$

**have** 0:finite (V – set p) V – set p  $\neq \{\}$  **using** finV assms **by** auto

**have** ?L = ( $\int vs. (\sum v \in V. \text{of-bool } (\text{is-first } v (p@vs))) \partial \text{pmf-of-set } (\text{permutations-of-set } (V\text{-set } p))$ )

**using** finV **unfolding** p-estimator-def cond-perm-def

**by** (subst map-pmf-of-set-inj[symmetric]) (auto intro:inj-onI)

**also have** ... = ( $\sum x \in V\text{-set } p. (\int vs. (\sum v \in V. \text{of-bool } (\text{is-first } v (p@x\#vs))) \partial ?q x) / \text{real}(\text{card } (V\text{-set } p))$ )

**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** ... = ( $\sum x \in V\text{-set } p. \text{p-estimator } (p@[x]) / \text{real}(\text{card } (V\text{-set } p))$ )

**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  $\subseteq V$

**defines** P  $\equiv \{v. \text{is-first } v p\}$

**defines** R  $\equiv V - \text{environment } (\text{set } p)$

**shows** p-estimator p = card P + ( $\sum v \in R. 1 / (\text{degree } v + 1 :: \text{real})$ )

(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  $\subseteq V$  **using** assms(2) is-first-imp-in-set **unfolding** P-def **by** auto

**moreover have** environment (set p)  $\subseteq V$  **using** environment-range assms(2) **by** auto

**ultimately have** V-split: V = P  $\cup$  Q  $\cup$  R **unfolding** R-def Q-def **by** auto

**have** P  $\subseteq$  environment (set p) **using** environment-def P-def is-first-imp-in-set **by** auto

**hence** 0: (P  $\cup$  Q)  $\cap$  R =  $\{\}$  P  $\cap$  Q =  $\{\}$  **unfolding** R-def Q-def **by** auto

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

**have** a: is-first v (p@vs) **if** v  $\in$  P **for** v vs

**using** that **unfolding** P-def is-first-def **by** auto

**have** b:  $\neg$ is-first v (p@vs) **if** v  $\in$  Q **for** v vs

**using** that **unfolding** Q-def P-def **by** (subst is-first-append-2) auto

**have** c: ( $\int vs. \text{of-bool } (\text{is-first } v (p@vs)) \partial ?q$ ) = 1 / (degree v + 1 :: real) (is ?L1 = ?R1)

**if** v-range:v  $\in$  R **for** v

**proof** –

**have** set p  $\cap$  environment {v} =  $\{\}$  **using** that environment-sym-2 **unfolding** R-def **by** auto

**moreover have** environment {v}  $\subseteq V$

**using** v-range **unfolding** R-def **by** (intro environment-range) auto

**ultimately have** d: {x  $\in$  V – set p. x  $\in$  environment {v}} = environment {v} **by** auto

**have** ?L1 = ( $\int vs. \text{indicator } \{vs. \text{is-first } v (p@vs)\} vs \partial ?q$ ) **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 (λx. x ∈ environment {v})) ?q) {vs. prefix [v] vs}*  
**unfolding** *is-first-def* **by** *simp*  
**also have** ... =  
*measure (pmf-of-set (permutations-of-set {x∈V-set p. x∈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* = (*∫ vs. real (∑ v ∈ 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** ... = (*∫ vs. (∑ v ∈ V. of-bool (is-first v vs)) ∂ ?p*) **by** *simp*  
**also have** ... = (*∑ v ∈ V. (∫ 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** ... = (*∑ v ∈ V. (∫ 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** ... = (*∑ v ∈ V. (∫ vs. of-bool (is-first v (p@vs)) ∂?q*) **by** *simp*  
**also have** ... = *real (card P) + (∑ v ∈ R. (∫ 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]) ⊆ 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** -

**let** *?d* = *λv. 1/(degree v + 1::real)*  
**let** *?e* = *λx. environment x*  
**define** *τ* :: *nat* **where** *τ = of-bool(environment {v} ∩ set p = {})*  
**have** *real-tau: of-bool(environment {v} ∩ set p = {}) = real τ* **unfolding** *τ-def* **by** *simp*  
**have** *v-range: v ∈ V* **using** *assms(2)* **by** *auto*

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

**have** *environment {v} ∩ set p = {} ↔ is-first v (p@[v])* (**is** *?L1* *↔* *?R1*)

**proof**

**assume** *?L1*  
**hence** *x ∉ environment {v} if x ∈ set p for x* **using** *that* **by** *auto*  
**moreover have** *v ∈ 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 ∉ set p* **using** *assms(1)* **by** *auto*  
**hence** *¬prefix [v] (filter (λy. y ∈ environment {v}) p)*  
**by** (*meson filter-is-subset prefix-imp-subseq subseq-singleton-left subset-code(1)*)  
**ultimately have** *filter (λy. y ∈ environment {v}) p = []*

**unfolding** *is-first-def filter-append* **by** (*cases filter* ( $\lambda y. y \in \text{environment } \{v\}$ ) *p*) *auto*  
**thus** ?L1 **unfolding** *filter-empty-conv* **by** *auto*  
**qed**  
**hence** 6:  $\tau = \text{of-bool } (\text{is-first } v \ (p@[v]))$  **unfolding**  $\tau$ -*def* **by** *simp*  
**have**  $\text{card } \{w. \text{is-first } w \ (p@[v])\} = \text{card } \{w. \text{is-first } w \ (p@[v]) \wedge w \neq v\} + \text{card } \{w. \text{is-first } v \ (p@[v]) \wedge w = v\}$   
**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**  $\dots = \text{card } \{w. \text{is-first } w \ p \wedge w \neq v\} + \text{of-bool } (\text{is-first } v \ (p@[v]))$   
**using** 4 **by** (*intro arg-cong2[where f=(+)] arg-cong[where f=card] Collect-cong*) *auto*  
**also have**  $\dots = \text{card } \{w. \text{is-first } w \ p\} + \tau$   
**using** 5 6 **by** (*intro arg-cong2[where f=(+)] arg-cong[where f=card] Collect-cong*) *auto*  
**finally have** 2:  $\text{card } \{w. \text{is-first } w \ (p@[v])\} = \text{card } \{w. \text{is-first } w \ p\} + \tau$  **by** *simp*

**have**  $?e \ \{v\} \subseteq V$  **using** *v-range environment-range* **by** *auto*  
**hence**  $V - ?e \ (\text{set } (p@[v])) \cup (?e \ \{v\} - ?e \ (\text{set } p)) = V - ?e \ (\text{set } p)$   
**unfolding** *set-append environment-union* **by** *auto*  
**moreover have**  $?e \ \{v\} \subseteq ?e \ (\text{set } (p@[v]))$  **unfolding** *environment-def* **by** *auto*  
**hence**  $(V - ?e \ (\text{set } (p@[v]))) \cap (?e \ \{v\} - ?e \ (\text{set } p)) = \{\}$  **by** *blast*  
**moreover have** *finite*  $(?e \ \{v\})$  **by** (*intro finite-environment*) *auto*  
**ultimately have** 3:  
 $(\sum_{v \in V - ?e \ (\text{set } (p@[v]))} ?d \ v) + (\sum_{v \in ?e \ \{v\} - ?e \ (\text{set } p)} ?d \ v) = (\sum_{v \in V - ?e \ (\text{set } p)} ?d \ v)$   
**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**  $p1 @ p2 \in \text{permutations-of-set } V$   
**shows**  $\text{distinct } (\text{derandomized-indep-set } p1 \ p2 \ E) \wedge$   
 $\text{is-independent-set } (\text{set } (\text{derandomized-indep-set } p1 \ p2 \ E))$   
**using** *assms*

**proof** (*induction p1 arbitrary: p2 rule:subseq-induct'*)

**case** 1  
**hence**  $\text{distinct } (\text{indep-set } p2 \ E) \wedge \text{is-independent-set } (\text{set } (\text{indep-set } p2 \ E))$   
**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**  $\text{node-deg} = (\lambda v. \text{real } (\text{card } \{e \in E. v \in e\}))$   
**define** *is-indep* **where**  $\text{is-indep} = (\lambda v. \text{list-all } (\lambda w. \{v, w\} \notin E) \ p2)$   
**define** *env* **where**  $\text{env} = (\lambda v. \text{filter } \text{is-indep } (v \# \text{filter } (\lambda w. \{v, w\} \in E) \ (p1h \# p1t)))$   
**define** *cost* **where**  $\text{cost} = (\lambda v. (\sum w \leftarrow \text{env } v. 1 / (\text{node-deg } w + 1)) - \text{of-bool}(\text{is-indep } v))$   
**define** *w* **where**  $w = \text{arg-min-list } \text{cost } p1$   
**have** *w-set*:  $w \in \text{set } p1$  **unfolding** *w-def p1-def* **using** *arg-min-list-in* **by** *blast*  
**have** *perm*:  $p1 @ p2 \in \text{permutations-of-set } V$  **using** 2(2) *p1-def* **by** *auto*  
**have** *dist*:  $\text{distinct } p1 \ \text{distinct } p2 \ \text{set } p1 \cap \text{set } p2 = \{\} \ \text{set } p1 \cup \text{set } p2 = V$   
 $\text{set } p1 = V - \text{set } p2$  **using** *permutations-of-setD[OF perm]* **by** *auto*

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

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

**have** *distinct* (derandomized-indep-set (remove1 w (p1h # p1t)) (p2 @ [w]) E)  $\wedge$   
*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  $\in$  permutations-of-set V  
**shows** length (derandomized-indep-set p1 p2 E)  $\geq$  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** ...  $\leq$  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** ...  $\leq$  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 = ( $\lambda v$ . real (card {e  $\in$  E. v  $\in$  e}))

**define** is-indep **where** is-indep = ( $\lambda v$ . list-all ( $\lambda w$ . {v,w}  $\notin$  E) p2)

**define** env **where** env = ( $\lambda v$ . filter is-indep (v#filter ( $\lambda w$ . {v,w}  $\in$  E) (p1h#p1t)))

**define** cost **where** cost = ( $\lambda v$ . ( $\sum w \leftarrow$  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  $\in$  permutations-of-set V **using** 2(2) p1-def **by** auto

**have** dist: distinct p1 distinct p2 set p1  $\cap$  set p2 = {} set p1  $\cup$  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  $\in$  set p1 **unfolding** w-def p1-def **using** arg-min-list-in **by** blast

**have** v-notin-p2: v  $\notin$  set p2 **if** v  $\in$  set p1 **for** v **using** dist(5) **that** **by** auto

**have** is-indep: is-indep v = (environment {v}  $\cap$  set p2 = {}) **if** v  $\in$  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  $\in$  set p1 **for** v

**proof** –

**have** set (env v) = {x  $\in$  {v}  $\cup$  {x  $\in$  set p1. {v, x}  $\in$  E}. is-indep x}

**unfolding** env-def p1-def[symmetric] **by** auto

**also have** ... = {x  $\in$  environment {v}  $\cap$  set p1. is-indep x}

**using** that **unfolding** environment-def vert-adj-def **by** (auto simp:insert-commute)

**also have** ... = {x  $\in$  environment {v}  $\cap$  set p1. set p2  $\cap$  environment {x} = {}}

**using** is-indep **by** auto

**also have** ... = environment {v}  $\cap$  set p1 – environment (set p2)

**by** (subst environment-sym-2) auto

**also have** ... = environment {v}  $\cap$  (V – set p2) – environment (set p2)

**using** environment-range dist(1-4) **that**

**by** (intro arg-cong2[where f=(-)] arg-cong2[where f=( $\cap$ )] refl) auto

**also have** ... = *environment*  $\{v\} \cap V - \text{set } p2 - \text{environment } (\text{set } p2)$  **by** *auto*  
**also have** ... = *environment*  $\{v\} \cap V - \text{environment } (\text{set } p2)$  **using** *environment-self* **by** *auto*  
**also have** ... = *environment*  $\{v\} - \text{environment } (\text{set } p2)$   
**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**  $\{v, v\} \notin E$  **by** (*simp add: singleton-not-edge*)  
**hence**  $v \notin \text{set } (\text{filter } (\lambda w. \{v, w\} \in E) p1)$  **by** *simp*  
**hence** *distinct (v # filter (λw. {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**  $?L = (\sum w \leftarrow \text{env } v. 1 / (\text{node-deg } w + 1)) - \text{of-bool } (\text{is-indep } v)$   
**unfolding** *cost-def* **by** *simp*  
**also have** ... =  $(\sum w \leftarrow \text{env } v. 1 / (\text{node-deg } w + 1)) - \text{of-bool}(\text{environment } \{v\} \cap \text{set } p2 = \{\})$   
 $\{\}$   
**by** (*simp add: is-indep[OF that]*)  
**also have** ... =  $(\sum w \leftarrow \text{env } v. 1 / (\text{degree } w + 1)) - \text{of-bool}(\text{environment } \{v\} \cap \text{set } p2 = \{\})$   
**unfolding** *node-deg-def alt-degree-def incident-edges-def vincident-def* **by** (*simp add: ac-simps*)  
**also have** ... =  $(\sum v \in ?e \{v\} - ?e (\text{set } p2). 1 / (\text{degree } v + 1)) - \text{of-bool} (?e \{v\} \cap \text{set } p2 = \{\})$   
**by** (*subst sum-list-distinct-conv-sum-set[OF dist-env-v]*) (*simp add: env-v*)  
**also have** ... =  $-(\text{of-bool} (?e \{v\} \cap \text{set } p2 = \{\}) - (\sum v \in ?e \{v\} - ?e (\text{set } p2). 1 / (\text{degree } v + 1)))$   
**by** (*simp add: algebra-simps*)  
**also have** ... =  $-(p\text{-estimator } (p2@[v]) - p\text{-estimator } (p2))$   
**using** *that dist(2-4)* **by** (*intro arg-cong[where f=λx. -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** ... ≤  $(\sum v \in \text{set } p1. \text{cost } v)$  **by** (*intro sum-mono*) *simp*  
**also have** ... =  $(\sum v \in \text{set } p1. p\text{-estimator } p2 - p\text{-estimator } (p2@[v]))$   
**by** (*intro sum.cong cost-correct refl*)  
**also have** ... =  $(\sum v \in V - \text{set } p2. p\text{-estimator } p2 - p\text{-estimator } (p2@[v]))$   
**using** *dist(1-4)* **by** (*intro sum.cong*) *auto*  
**also have** ... =  $\text{card } (V - \text{set } p2) * p\text{-estimator } p2 - (\sum v \in V - \text{set } p2. p\text{-estimator } (p2@[v]))$   
**unfolding** *sum-subtractf* **by** *simp*  
**also have** ... = 0 **using** *dist(5)[symmetric]* *p1-ne* **by** (*subst p-estimator-split*) *auto*  
**finally have**  $\text{Min } (\text{cost ' set } p1) \leq 0$  **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**  $p\text{-estimator } p2 \leq p\text{-estimator } p2 - \text{cost } w$  **using** *cost-w-nonpos* **by** *simp*  
**also have** ... =  $p\text{-estimator } (p2@[w])$  **unfolding** *cost-correct[OF w-set]* **by** *simp*  
**also have** ... ≤  $\text{length } (\text{derandomized-indep-set } (\text{remove1 } w \text{ p1}) (p2@[w]) E)$   
**using** *c* **by** (*intro 2 a b permutations-of-setI*) (*auto simp:p1-def*)  
**also have** ... =  $\text{real } (\text{length } (\text{derandomized-indep-set } p1 \text{ p2 } E))$

```

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**  $p \in \text{permutations-of-set } V$

**shows**

*is-independent-set* (set (derandomized-indep-set  $p \ \square \ E$ ))

*distinct* (derandomized-indep-set  $p \ \square \ E$ )

$\text{length}$  (derandomized-indep-set  $p \ \square \ E$ )  $\geq (\sum v \in V. 1 / (\text{degree } v+1))$

$\text{length}$  (derandomized-indep-set  $p \ \square \ E$ )  $\geq \text{card } V / (2 * \text{card } E / \text{card } V + 1)$

**proof** –

**let** ?res = derandomized-indep-set  $p \ \square \ E$

**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**  $(\sum v \in V. 1 / (\text{degree } v+1)) \leq p\text{-estimator} \ \square$

**by** (*subst p-estimator*) (*simp-all add:environment-def is-first-def ac-simps*)

**also have** ...  $\leq \text{length}$  ?res **using** *assms derandomized-indep-set-length-aux* **by** *auto*

**finally show**  $a: (\sum v \in V. 1 / (\text{degree } v+1)) \leq \text{length}$  ?res **by** *auto*

**thus**  $\text{card } V / (2 * \text{card } E / \text{card } V + 1) \leq \text{length}$  ?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.