Maximum Cardinality Matching

Christine Rizkallah

December 14, 2021

Abstract

A *matching* in a graph G is a subset M of the edges of G such that no two share an endpoint. A matching has maximum cardinality if its cardinality is at least as large as that of any other matching. An *odd-set cover* OSC of a graph G is a labeling of the nodes of G with integers such that every edge of G is either incident to a node labeled 1 or connects two nodes labeled with the same number $i \geq 2$.

Theorem 1 (Edmonds [2])**.** Let M be a matching in a graph G and let OSC be an odd-set cover of G. For any $i \geq 0$, let n_i be the number of nodes labeled i. If

$$|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

then M is a maximum cardinality matching.

We provide an Isabelle proof of Edmonds theorem. For an explanation of the proof see [1].

Contents

theory *Matching*
imports *Main*
begin

type-synonym *label* = *nat*

1 Definitions

definition *finite-graph* :: *'v set* => *('v * 'v) set* ⇒ *bool* **where**
finite-graph V E = (*finite V* ∧ *finite E* ∧
(∀ *e* ∈ *E*. *fst e* ∈ *V* ∧ *snd e* ∈ *V* ∧ *fst e* ~= *snd e*))

definition *degree* :: *('v * 'v) set* ⇒ *'v* ⇒ *nat* **where**
degree E v = *card* {*e* ∈ *E*. *fst e* = *v* ∨ *snd e* = *v*}

definition *edge-as-set* :: *('v * 'v)* ⇒ *'v set* **where**
edge-as-set e = {*fst e*, *snd e*}

definition *N* :: *'v set* ⇒ *('v* ⇒ *label)* ⇒ *nat* ⇒ *nat* **where**
N V L i = *card* {*v* ∈ *V*. *L v* = *i*}

definition *weight*:: *label set* ⇒ *(label* ⇒ *nat)* ⇒ *nat* **where**
weight LV f = *f 1* + (∑ *i*∈*LV*. *(f i) div 2*)

definition *OSC* :: *('v* ⇒ *label)* ⇒ *('v * 'v) set* ⇒ *bool* **where**
OSC L E = (∀ *e* ∈ *E*. *L* (*fst e*) = *1* ∨ *L* (*snd e*) = *1* ∨
L (*fst e*) = *L* (*snd e*) ∧ *L* (*fst e*) > *1*)

definition *disjoint-edges* :: *('v * 'v)* ⇒ *('v * 'v)* ⇒ *bool* **where**
disjoint-edges e1 e2 = (*fst e1* ≠ *fst e2* ∧ *fst e1* ≠ *snd e2* ∧
snd e1 ≠ *fst e2* ∧ *snd e1* ≠ *snd e2*)

definition *matching* :: *'v set* ⇒ *('v * 'v) set* ⇒ *('v * 'v) set* ⇒ *bool* **where**
matching V E M = (*M* ⊆ *E* ∧ *finite-graph V E* ∧
(∀ *e1* ∈ *M*. ∀ *e2* ∈ *M*. *e1* ≠ *e2* ⟶ *disjoint-edges e1 e2*))

definition *matching-i* :: *nat* ⇒ *'v set* ⇒ *('v * 'v) set* ⇒ *('v * 'v) set* ⇒
('v ⇒ *label)* ⇒ *('v * 'v) set* **where**
matching-i i V E M L = {*e* ∈ *M*. *i=1* ∧ (*L* (*fst e*) = *i* ∨ *L* (*snd e*) = *i*)
∨ *i>1* ∧ *L* (*fst e*) = *i* ∧ *L* (*snd e*) = *i*}

definition *V-i*:: *nat* ⇒ *'v set* ⇒ *('v * 'v) set* ⇒ *('v * 'v) set* ⇒
('v ⇒ *label)* ⇒ *'v set* **where**
V-i i V E M L = ⋃ (*edge-as-set* ‘ *matching-i i V E M L*)

definition *endpoint-inV* :: *'v set* ⇒ *('v * 'v)* ⇒ *'v* **where**
endpoint-inV V e = (*if fst e* ∈ *V then fst e else snd e*)

definition *relevant-endpoint* :: *('v* ⇒ *label)* ⇒ *'v set* ⇒
*('v * 'v)* ⇒ *'v* **where**
relevant-endpoint L V e = (*if L* (*fst e*) = *1 then fst e else snd e*)

2 Lemmas

lemma *definition-of-range*:

endpoint-inV V1 ‘ matching-i 1 V E M L =
{ v. ∃ e ∈ matching-i 1 V E M L. endpoint-inV V1 e = v } **by** *auto*

lemma *matching-i-edges-as-sets*:
edge-as-set ‘ matching-i i V E M L =
{ e1. ∃ (u, v) ∈ matching-i i V E M L. edge-as-set (u, v) = e1} **by** *auto*

lemma *matching-disjointness*:
assumes *matching V E M*
assumes *e1 ∈ M*
assumes *e2 ∈ M*
assumes *e1 ≠ e2*
shows *edge-as-set e1 ∩ edge-as-set e2 = {}*
using *assms*
by (*auto simp add*: *edge-as-set-def disjoint-edges-def matching-def*)

lemma *expand-set-containment*:
assumes *matching V E M*
assumes *e ∈ M*
shows *e ∈ E*
using *assms*
by (*auto simp add*:*matching-def*)

theorem *injectivity*:
assumes *is-osc*: *OSC L E*
assumes *is-m*: *matching V E M*
assumes *e1-in-M1*: *e1 ∈ matching-i 1 V E M L*
and *e2-in-M1*: *e2 ∈ matching-i 1 V E M L*
assumes *diff*: (*e1 ≠ e2*)
shows *endpoint-inV {v ∈ V. L v = 1} e1 ≠ endpoint-inV {v ∈ V. L v = 1} e2*
proof −
from *e1-in-M1* **have** *e1 ∈ M* **by** (*auto simp add*: *matching-i-def*)
moreover
from *e2-in-M1* **have** *e2 ∈ M* **by** (*auto simp add*: *matching-i-def*)
ultimately
have *disjoint-edge-sets*: *edge-as-set e1 ∩ edge-as-set e2 = {}*
using *diff is-m matching-disjointness* **by** *fast*
then show *?thesis* **by** (*auto simp add*: *edge-as-set-def endpoint-inV-def*)
qed

2.1 $|M1| \leq n1$

lemma *card-M1-le-NVL1*:
assumes *matching V E M*
assumes *OSC L E*
shows *card* (*matching-i 1 V E M L*) ≤ (*N V L 1*)
proof −
let *?f = endpoint-inV {v ∈ V. L v = 1}*

```
  let ?A = matching-i 1 V E M L
  let ?B = {v ∈ V. L v = 1}
  have inj-on ?f ?A using assms injectivity
    unfolding inj-on-def by blast
  moreover have ?f ‘ ?A ⊆ ?B
  proof −
    {
      fix e assume e ∈ matching-i 1 V E M L
      then have endpoint-inV {v ∈ V. L v = 1} e ∈ {v ∈ V. L v = 1}
        using assms
        by (auto simp add: endpoint-inV-def matching-def
          matching-i-def OSC-def finite-graph-def definition-of-range)
    }
    then show ?thesis using assms definition-of-range by blast
  qed
  moreover have finite ?B using assms
    by (simp add: matching-def finite-graph-def)
  ultimately show ?thesis unfolding N-def by (rule card-inj-on-le)
qed

lemma edge-as-set-inj-on-Mi:
  assumes matching V E M
  shows inj-on edge-as-set (matching-i i V E M L)
  using assms
  unfolding inj-on-def edge-as-set-def matching-def
    disjoint-edges-def matching-i-def
  by blast

lemma card-Mi-eq-card-edge-as-set-Mi:
  assumes matching V E M
  shows card (matching-i i V E M L) = card (edge-as-set‘ matching-i i V E M L)
  (is card ?Mi = card (?f ‘ -))
proof −
  from assms have bij-betw ?f ?Mi (?f ‘ ?Mi)
    by (simp add: bij-betw-def matching-i-edges-as-sets edge-as-set-inj-on-Mi)
  then show ?thesis by (rule bij-betw-same-card)
qed

lemma card-edge-as-set-Mi-twice-card-partitions:
  assumes OSC L E ∧ matching V E M ∧ i > 1
  shows 2 ∗ card (edge-as-set‘matching-i i V E M L)
  = card (V-i i V E M L) (is 2 ∗ card ?C = card ?Vi)
proof −
  from assms have 1: finite (⋃ ?C)
    by (auto simp add: matching-def finite-graph-def
      matching-i-def edge-as-set-def finite-subset)
  show ?thesis unfolding V-i-def
  proof (rule card-partition)
    show finite ?C using 1 by (rule finite-UnionD)
```

```
  next
    show finite (⋃ ?C) using 1 .
  next
    fix c assume c ∈ ?C then show card c = 2
    proof (rule imageE)
      fix x
      assume 2: c = edge-as-set x and 3: x ∈ matching-i i V E M L
      with assms have x ∈ E
        unfolding matching-i-def matching-def by blast
      then have fst x ≠ snd x using assms 3
        by (auto simp add: matching-def finite-graph-def)
      with 2 show ?thesis by (auto simp add: edge-as-set-def)
    qed
  next
    fix x1 x2
    assume 4: x1 ∈ ?C and 5: x2 ∈ ?C and 6: x1 ≠ x2
    {
      fix e1 e2
      assume 7: x1 = edge-as-set e1 e1 ∈ matching-i i V E M L
        x2 = edge-as-set e2 e2 ∈ matching-i i V E M L
      from assms have matching V E M by simp
      moreover
      from 7 assms have e1 ∈ M and e2 ∈ M
        by (simp-all add: matching-i-def)
      moreover from 6 7 have e1 ≠ e2 by blast
      ultimately have x1 ∩ x2 = {} unfolding 7
        by (rule matching-disjointness)
    }
    with 4 5 show x1 ∩ x2 = {} by clarsimp
  qed
qed

lemma card-Mi-twice-card-Vi:
  assumes OSC L E ∧ matching V E M ∧ i > 1
  shows 2 * card (matching-i i V E M L) = card (V-i i V E M L)
proof −
  from assms have finite (V-i i V E M L)
    by (auto simp add: edge-as-set-def finite-subset
      matching-def finite-graph-def V-i-def matching-i-def )
  with assms show ?thesis
    by (simp add: card-Mi-eq-card-edge-as-set-Mi
      card-edge-as-set-Mi-twice-card-partitions V-i-def)
qed

lemma card-Mi-le-floor-div-2-Vi:
  assumes OSC L E ∧ matching V E M ∧ i > 1
  shows card (matching-i i V E M L) ≤ (card (V-i i V E M L)) div 2
  using card-Mi-twice-card-Vi[OF assms]
  by arith
```

lemma *card-Vi-le-NVLi*:
assumes *i>1* $\wedge$ *matching V E M*
shows *card* (*V-i i V E M L*) $\leq$ *N V L i*
unfolding *N-def*
proof (*rule card-mono*)
show *finite* $\{v \in V.\ L\ v = i\}$ **using** *assms*
by (*simp add*: *matching-def finite-graph-def*)
next
let *?A* = *edge-as-set* ‘ *matching-i i V E M L*
let *?C* = $\{v \in V.\ L\ v = i\}$
show *V-i i V E M L* $\subseteq$ *?C* **using** *assms* **unfolding** *V-i-def*
proof (*intro Union-least*)
fix *X* **assume** *X* $\in$ *?A*
with *assms* **have** $\exists x \in$ *matching-i i V E M L. edge-as-set x* = *X*
by (*simp add*: *matching-i-edges-as-sets*)
with *assms* **show** *X* $\subseteq$ *?C*
unfolding *finite-graph-def matching-def*
matching-i-def edge-as-set-def **by** *blast*
qed
qed

2.2 $|Mi| \leq \lfloor ni/2 \rfloor$

lemma *card-Mi-le-floor-div-2-NVLi*:
assumes *OSC L E* $\wedge$ *matching V E M* $\wedge$ *i* > *1*
shows *card* (*matching-i i V E M L*) $\leq$ (*N V L i*) *div 2*
proof −
from *assms* **have** *card* (*V-i i V E M L*) $\leq$ (*N V L i*)
by (*simp add*: *card-Vi-le-NVLi*)
then have *card* (*V-i i V E M L*) *div 2* $\leq$ (*N V L i*) *div 2*
by *simp*
moreover from *assms* **have**
card (*matching-i i V E M L*) $\leq$ *card* (*V-i i V E M L*) *div 2*
by (*intro card-Mi-le-floor-div-2-Vi*)
ultimately show *?thesis* **by** *auto*
qed

2.3 $|M| \leq \sum |Mi|$

lemma *card-M-le-sum-card-Mi*:
assumes *matching V E M* **and** *OSC L E*
shows *card M* $\leq$ ($\sum$ *i* $\in$ *L‘V*. *card* (*matching-i i V E M L*))
(**is** *card* - $\leq$ *?CardMi*)
proof −
let *?UnMi* = $\bigcup x \in$ *L‘V*. *matching-i x V E M L*
from *assms* **have** *1*: *finite ?UnMi*
by (*auto simp add*: *matching-def*
finite-graph-def matching-i-def finite-subset)
{

```
    fix e assume e-inM: e ∈ M
    let ?v = relevant-endpoint L V e
    have 1: e ∈ matching-i (L ?v) V E M L using assms e-inM
     proof cases
      assume L (fst e) = 1
      thus ?thesis using assms e-inM
       by (simp add: relevant-endpoint-def matching-i-def)
     next
      assume a: L (fst e) ≠ 1
      have L (fst e) = 1 ∨ L (snd e) = 1
       ∨ (L (fst e) = L (snd e) ∧ L (fst e) >1)
       using assms e-inM unfolding OSC-def
       by (blast intro: expand-set-containment)
      thus ?thesis using assms e-inM a
       by (auto simp add: relevant-endpoint-def matching-i-def)
     qed
     have 2: ?v ∈ V using assms e-inM
      by (auto simp add: matching-def
       relevant-endpoint-def matching-i-def finite-graph-def)
     then have ∃ v ∈ V. e ∈ matching-i (L v) V E M L using assms 1 2
      by (intro bexI)
    }
    with assms have M ⊆ ?UnMi by (auto)
    with assms and 1 have card M ≤ card ?UnMi by (intro card-mono)
    moreover from assms have card ?UnMi = ?CardMi
    proof (intro card-UN-disjoint)
     show finite (L‘V) using assms
      by (simp add: matching-def finite-graph-def)
    next
     show ∀ i∈L‘V. finite (matching-i i V E M L) using assms
      unfolding matching-def finite-graph-def matching-i-def
      by (blast intro: finite-subset)
    next
     show ∀ i ∈ L‘V. ∀ j ∈ L‘V. i ≠ j ⟶
      matching-i i V E M L ∩ matching-i j V E M L = {} using assms
      by (auto simp add: matching-i-def)
    qed
  ultimately show ?thesis by simp
qed

theorem card-M-le-weight-NVLi:
  assumes matching V E M and OSC L E
  shows card M ≤ weight {i ∈ L ‘ V. i > 1} (N V L) (is - ≤ ?W)
proof −
  let ?M01 = ∑ i| i ∈ L‘V ∧ (i=1 ∨ i=0). card (matching-i i V E M L)
  let ?Mgr1 = ∑ i| i ∈ L‘V ∧ 1 < i. card (matching-i i V E M L)
  let ?Mi = ∑ i∈L‘V. card (matching-i i V E M L)
  have card M ≤ ?Mi using assms by (rule card-M-le-sum-card-Mi)
  moreover
```

```
have ?Mi ≤ ?W
proof −
  let ?A = {i ∈ L ‘ V. i = 1 ∨ i = 0}
  let ?B = {i ∈ L ‘ V. 1 < i}
  let ?g = λ i. card (matching-i i V E M L)
  let ?set01 = { i. i : L ‘ V & (i = 1 | i = 0)}
  have a: L ‘ V = ?A ∪ ?B using assms by auto
  have finite V using assms
    by (simp add: matching-def finite-graph-def)
  have b: sum ?g (?A ∪ ?B) = sum ?g ?A + sum ?g ?B
    using assms ‹finite V› by (auto intro: sum.union-disjoint)
  have 1: ?Mi = ?M01+ ?Mgr1 using assms a b
    by (simp add: matching-def finite-graph-def)
  moreover
  have 0: card (matching-i 0 V E M L) = 0 using assms
    by (simp add: matching-i-def)
    have 2: ?M01 ≤ N V L 1
    proof cases
      assume a: 1 ∈ L‘V
      have ?M01 = card (matching-i 1 V E M L)
      proof cases
        assume b: 0 ∈ L‘V
        with a assms have  ?set01 = {0, 1} by blast
        thus ?thesis using assms 0 by simp
      next
        assume b: 0 ∉ L‘V
        with a have ?set01 = {1} by (auto simp del:One-nat-def)
        thus ?thesis by simp
      qed
      thus ?thesis using assms a
        by (simp del: One-nat-def, intro card-M1-le-NVL1)
    next
      assume a: 1 ∉ L‘V
      show ?thesis
      proof cases
        assume b: 0 ∈ L‘V
        with a assms have  ?set01 = {0} by (auto simp del:One-nat-def)
        thus ?thesis using assms 0 by auto
      next
        assume b: 0 ∉ L‘V
        with a have ?set01 = {} by (auto simp del:One-nat-def)
          then have ?M01 = (∑ i∈{}. card (matching-i i V E M L)) by auto
          thus ?thesis by simp
        qed
      qed
    moreover
    have 3: ?Mgr1 ≤ (∑ i|i∈L‘V ∧ 1 < i. N V L i div 2) using assms
      by (intro sum-mono card-Mi-le-floor-div-2-NVLi, simp)
  ultimately
```

show *?thesis* **using** *1 2 3 assms* **by** (*simp add*: *weight-def*)
qed
ultimately show *?thesis* **by** *simp*
qed

3 Final Theorem

The following theorem is due to Edmond [2]:

theorem *maximum-cardinality-matching*:
assumes *matching V E M* **and** *OSC L E*
and *card M* = *weight* {*i* ∈ *L* ‘ *V*. *i* > *1*} (*N V L*)
and *matching V E M'*
shows *card M'* ≤ *card M*
using *assms card-M-le-weight-NVLi*
by *simp*

The widely used algorithmic library LEDA has a certifying algorithm for maximum cardinality matching. This Isabelle proof is part of the work done to verify the checker of this certifying algorithm. For more information see [1].

end

References

[1] E. Alkassar, S. Böhme, K. Mehlhorn, and C. Rizkallah. Verification of certifying computations. In *Proceedings of the 23rd International Conference on Computer Aided Verfication (CAV2011)*, Cliff Lodge, Snowbird, Utah, USA, 2011. To Appear.

[2] J. Edmonds. Maximum matching and a polyhedron with 0,1 - vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.