Verified Approximation Algorithms

Robin Eßmann, Tobias Nipkow, Simon Robillard, Ujkan Sulejmani

December 14, 2021

Abstract

We present the first formal verifications of approximation algorithms for NP-complete optimization problems: vertex cover, set cover, independent set, center selection, load balancing, and bin packing. The proofs correct incompletenesses in existing proofs and improve the approximation ratio in one case. A detailed description of our work (excluding center selection) has been published in the proceedings of *IJCAR 2020* [3].

Contents

1 Vertex Cover

theory *Approx-VC-Hoare*
imports *HOL−Hoare.Hoare-Logic*
begin

The algorithm is classical, the proof is based on and augments the one by Berghammer and Müller-Olm [1].

1.1 Graph

A graph is simply a set of edges, where an edge is a 2-element set.

definition *vertex-cover* :: $'a$ *set set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
vertex-cover E C = ($\forall e \in E.\ e \cap C \neq \{\}$)

abbreviation *matching* :: $'a$ *set set* $\Rightarrow$ *bool* **where**
matching $M \equiv$ *pairwise disjnt* M

lemma *card-matching-vertex-cover*:
$[\![$ *finite* C; *matching* M; $M \subseteq E$; *vertex-cover* E C $]\!] \Longrightarrow$ *card* $M \leq$ *card* C
apply(*erule card-le-if-inj-on-rel*[**where** $r = \lambda e\ v.\ v \in e$])
apply (*meson disjnt-def disjnt-iff vertex-cover-def subsetCE*)
by (*meson disjnt-iff pairwise-def*)

1.2 The Approximation Algorithm

Formulated using a simple(!) predefined Hoare-logic. This leads to a streamlined proof based on standard invariant reasoning.

The nondeterministic selection of an element from a set F is simulated by *SOME* $x.\ x \in F$. The *SOME* operator is built into HOL: *SOME* $x.\ P\ x$ denotes some x that satisfies P if such an x exists; otherwise it denotes an arbitrary element. Note that there is no actual nondeterminism involved: *SOME* $x.\ P\ x$ is some fixed element but in general we don't know which one. Proofs about *SOME* are notoriously tedious. Typically it involves showing first that $\exists x.\ P\ x$. Then $\exists x.\ ?P\ x \Longrightarrow\ ?P$ (*SOME* $x.\ ?P\ x$) implies P (*SOME* $x.\ P\ x$). There are a number of (more) useful related theorems: just click on $\exists x.\ ?P\ x \Longrightarrow\ ?P$ (*SOME* $x.\ ?P\ x$) to be taken there.

Convenient notation for choosing an arbitrary element from a set:

abbreviation *some A* $\equiv$ *SOME x. x* $\in$ *A*

locale *Edges* =
  fixes *E* :: *'a set set*
  assumes *finE*: *finite E*
  assumes *edges2*: *e* $\in$ *E* $\Longrightarrow$ *card e* = *2*
begin

The invariant:

definition *inv-matching C F M* =
  (*matching M* $\wedge$ *M* $\subseteq$ *E* $\wedge$ *card C* $\leq$ *2* $*$ *card M* $\wedge$ ($\forall$ *e* $\in$ *M*. $\forall$ *f* $\in$ *F*. *e* $\cap$ *f* = {}))

definition *invar* :: *'a set* $\Rightarrow$ *'a set set* $\Rightarrow$ *bool* **where**
invar C F = (*F* $\subseteq$ *E* $\wedge$ *vertex-cover* (*E*−*F*) *C* $\wedge$ *finite C* $\wedge$ ($\exists$ *M*. *inv-matching C F M*))

Preservation of the invariant by the loop body:

lemma *invar-step*:
  assumes *F* $\neq$ {} *invar C F*
  shows *invar* (*C* $\cup$ *some F*) (*F* − {*e'* $\in$ *F*. *some F* $\cap$ *e'* $\neq$ {}})
proof −
  from *assms*(*2*) **obtain** *M* **where** *F* $\subseteq$ *E* **and** *vc*: *vertex-cover* (*E*−*F*) *C* **and** *fC*: *finite C*
    and *m*: *matching M M* $\subseteq$ *E* **and** *card*: *card C* $\leq$ *2* $*$ *card M*
    and *disj*: $\forall$ *e* $\in$ *M*. $\forall$ *f* $\in$ *F*. *e* $\cap$ *f* = {}
  by (*auto simp*: *invar-def inv-matching-def*)
  let *?e* = *SOME e. e* $\in$ *F*
  have *?e* $\in$ *F* **using** ‹*F* $\neq$ {}› **by** (*simp add*: *some-in-eq*)
  hence *fe'*: *finite ?e* **using** ‹*F* $\subseteq$ *E*› *edges2* **by**(*intro card-ge-0-finite*) *auto*
  have *?e* $\notin$ *M* **using** *edges2* ‹*?e* $\in$ *F*› *disj* ‹*F* $\subseteq$ *E*› **by** *fastforce*
  have *card'*: *card* (*C* $\cup$ *?e*) $\leq$ *2* $*$ *card* (*insert ?e M*)
      using ‹*?e* $\in$ *F*› ‹*?e* $\notin$ *M*› *card-Un-le*[*of C ?e*] ‹*F* $\subseteq$ *E*› *edges2 card finite-subset*[*OF m*(*2*) *finE*]
    by *fastforce*
  let *?M* = *M* $\cup$ {*?e*}
  have *vc'*: *vertex-cover* (*E* − (*F* − {*e'* $\in$ *F*. *?e* $\cap$ *e'* $\neq$ {}})) (*C* $\cup$ *?e*)
    using *vc* **by**(*auto simp*: *vertex-cover-def*)
  have *m'*: *inv-matching* (*C* $\cup$ *?e*) (*F* − {*e'* $\in$ *F*. *?e* $\cap$ *e'* $\neq$ {}}) *?M*
    using *m card'* ‹*F* $\subseteq$ *E*› ‹*?e* $\in$ *F*› *disj*
    by(*auto simp*: *inv-matching-def Int-commute disjnt-def pairwise-insert*)
  show *?thesis* **using** ‹*F* $\subseteq$ *E*› *vc' fC fe' m'* **by**(*auto simp add*: *invar-def Let-def*)
qed

lemma *approx-vertex-cover*:
VARS C F
  {*True*}

$C := \{\}$;
$F := E$;
WHILE $F \neq \{\}$
INV $\{invar\ C\ F\}$
DO $C := C \cup some\ F$;
$F := F - \{e' \in F.\ some\ F \cap e' \neq \{\}\}$
OD
$\{vertex\text{-}cover\ E\ C \wedge (\forall C'.\ finite\ C' \wedge vertex\text{-}cover\ E\ C' \longrightarrow card\ C \leq 2 * card\ C')\}$
proof (*vcg*, *goal-cases*)
case (*1 C F*)
have *inv-matching* {} *E* {} **by** (*auto simp add*: *inv-matching-def*)
with *1* **show** *?case* **by** (*auto simp add*: *invar-def vertex-cover-def*)
next
case (*2 C F*)
thus *?case* **using** *invar-step*[*of F C*] **by**(*auto simp*: *Let-def*)
next
case (*3 C F*)
then obtain M :: *'a set set* **where**
post: *vertex-cover E C matching M* $M \subseteq E$ *card* $C \leq 2 * card\ M$
by(*auto simp*: *invar-def inv-matching-def*)

have *opt*: $card\ C \leq 2 * card\ C'$ **if** C': *finite* C' *vertex-cover* $E\ C'$ **for** C'
proof −
note *post*(*4*)
also have $2 * card\ M \leq 2 * card\ C'$
using *card-matching-vertex-cover*[*OF* C'(*1*) *post*(*2*,*3*) C'(*2*)] **by** *simp*
finally show $card\ C \leq 2 * card\ C'$ **.**
qed

show *?case* **using** *post*(*1*) *opt* **by** *auto*
qed

end

1.3 Version for Hypergraphs

Almost the same. We assume that the degree of every edge is bounded.

locale *Bounded-Hypergraph* =
fixes E :: *'a set set*
fixes k :: *nat*
assumes *finE*: *finite E*
assumes *edge-bnd*: $e \in E \Longrightarrow finite\ e \wedge card\ e \leq k$
assumes *E1*: $\{\} \notin E$
begin

definition *inv-matching* $C\ F\ M$ =
$(matching\ M \wedge M \subseteq E \wedge card\ C \leq k * card\ M \wedge (\forall e \in M.\ \forall f \in F.\ e \cap f = \{\}))$

definition *invar* :: *'a set* $\Rightarrow$ *'a set set* $\Rightarrow$ *bool* **where**
invar C F = ($F \subseteq E \land$ *vertex-cover* $(E-F)$ $C \land$ *finite* $C \land (\exists M.$ *inv-matching C F M*))

lemma *invar-step*:
  assumes $F \neq \{\}$ *invar C F*
  shows *invar* ($C \cup$ *some F*) ($F - \{e' \in F.$ *some* $F \cap e' \neq \{\}\}$)
proof −
  from *assms*(*2*) **obtain** *M* **where** $F \subseteq E$ **and** *vc*: *vertex-cover* $(E-F)$ *C* **and** *fC*: *finite C*
    and *m*: *matching M* $M \subseteq E$ **and** *card*: *card* $C \leq k *$ *card M*
    and *disj*: $\forall e \in M. \forall f \in F. e \cap f = \{\}$
  by (*auto simp*: *invar-def inv-matching-def*)
  let *?e* = *SOME e*. $e \in F$
  have $?e \in F$ **using** ‹$F \neq \{\}$› **by** (*simp add*: *some-in-eq*)
  hence *fe'*: *finite ?e* **using** ‹$F \subseteq E$› *assms*(*2*) *edge-bnd* **by** *blast*
  have $?e \notin M$ **using** *E1* ‹$?e \in F$› *disj* ‹$F \subseteq E$› **by** *fastforce*
  have *card'*: *card* ($C \cup ?e$) $\leq k *$ *card* (*insert ?e M*)
    using ‹$?e \in F$› ‹$?e \notin M$› *card-Un-le*[*of C ?e*] ‹$F \subseteq E$› *edge-bnd card finite-subset*[*OF m*(*2*) *finE*]
    by *fastforce*
  let $?M = M \cup \{?e\}$
  have *vc'*: *vertex-cover* ($E - (F - \{e' \in F. ?e \cap e' \neq \{\}\})$) ($C \cup ?e$)
    using *vc* **by**(*auto simp*: *vertex-cover-def*)
  have *m'*: *inv-matching* ($C \cup ?e$) ($F - \{e' \in F. ?e \cap e' \neq \{\}\}$) *?M*
    using *m card'* ‹$F \subseteq E$› ‹$?e \in F$› *disj*
    by(*auto simp*: *inv-matching-def Int-commute disjnt-def pairwise-insert*)
  show *?thesis* **using** ‹$F \subseteq E$› *vc' fC fe' m'* **by**(*auto simp add*: *invar-def Let-def*)
qed

lemma *approx-vertex-cover-bnd*:
VARS C F
  {*True*}
  $C := \{\}$;
  $F := E$;
  WHILE $F \neq \{\}$
  INV {*invar C F*}
  DO $C := C \cup$ *some F*;
    $F := F - \{e' \in F.$ *some* $F \cap e' \neq \{\}\}$
  OD
  {*vertex-cover E C* $\land (\forall C'.$ *finite* $C' \land$ *vertex-cover E* $C' \longrightarrow$ *card* $C \leq k *$ *card* $C')$}
proof (*vcg*, *goal-cases*)
  case (*1 C F*)
  have *inv-matching* {} *E* {} **by** (*auto simp add*: *inv-matching-def*)
  with *1* **show** *?case* **by** (*auto simp add*: *invar-def vertex-cover-def*)
next

```
  case (2 C F)
  thus ?case using invar-step[of F C] by(auto simp: Let-def)
next
  case (3 C F)
  then obtain M :: 'a set set where
    post: vertex-cover E C matching M M ⊆ E card C ≤ k ∗ card M
    by(auto simp: invar-def inv-matching-def)

  have opt: card C ≤ k ∗ card C' if C': finite C' vertex-cover E C' for C'
  proof −
    note post(4)
    also have k ∗ card M ≤ k ∗ card C'
    using card-matching-vertex-cover[OF C'(1) post(2,3) C'(2)] by simp
    finally show card C ≤ k ∗ card C' .
  qed

  show ?case using post(1) opt by auto
qed

end

end
```

2 Set Cover

```
theory Approx-SC-Hoare
imports
  HOL−Hoare.Hoare-Logic
  Complex-Main
begin
```

This is a formalization of the set cover algorithm and proof in the book by Kleinberg and Tardos [4].

```
definition harm :: nat ⇒ 'a :: real-normed-field where
  harm n = (∑k=1..n. inverse (of-nat k))


locale Set-Cover =
  fixes w :: nat ⇒ real
    and m :: nat
    and S :: nat ⇒ 'a set
  assumes S-finite: ∀ i ∈ {1..m}. finite (S i)
      and w-nonneg: ∀ i. 0 ≤ w i
begin

definition U :: 'a set where
  U = (⋃ i ∈ {1..m}. S i)

lemma S-subset: ∀ i ∈ {1..m}. S i ⊆ U
```

using *U-def* **by** *blast*

lemma *U-finite*: *finite U*
unfolding *U-def* **using** *S-finite* **by** *blast*

lemma *empty-cover*: $m = 0 \Longrightarrow U = \{\}$
using *U-def* **by** *simp*

definition $sc :: nat\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
$sc\ C\ X \longleftrightarrow C \subseteq \{1..m\} \wedge (\bigcup i \in C.\ S\ i) = X$

definition $cost :: 'a\ set \Rightarrow nat \Rightarrow real$ **where**
$cost\ R\ i = w\ i\ /\ card\ (S\ i \cap R)$

lemma *cost-nonneg*: $0 \le cost\ R\ i$
using *w-nonneg* **by** (*simp add*: *cost-def*)

$cost\ R\ i = 0$ if $card\ (S\ i \cap R) = 0$! Needs to be accounted for separately in *min-arg*.

fun $min\text{-}arg :: 'a\ set \Rightarrow nat \Rightarrow nat$ **where**
$min\text{-}arg\ R\ 0 = 1$
$|\ min\text{-}arg\ R\ (Suc\ x) =$
$(let\ j = min\text{-}arg\ R\ x$
$in\ if\ S\ j \cap R = \{\} \vee (S\ (Suc\ x) \cap R \neq \{\} \wedge cost\ R\ (Suc\ x) < cost\ R\ j)\ then\ (Suc\ x)\ else\ j)$

lemma *min-in-range*: $k > 0 \Longrightarrow min\text{-}arg\ R\ k \in \{1..k\}$
by (*induction k*) (*force simp*: *Let-def*)+

lemma *min-empty*: $S\ (min\text{-}arg\ R\ k) \cap R = \{\} \Longrightarrow \forall\, i \in \{1..k\}.\ S\ i \cap R = \{\}$
proof (*induction k*)
case (*Suc k*)
from *Suc.prems* **have** *prem*: $S\ (min\text{-}arg\ R\ k) \cap R = \{\}$ **by** (*auto simp*: *Let-def split*: *if-splits*)
with *Suc.IH* **have** *IH*: $\forall\, i \in \{1..k\}.\ S\ i \cap R = \{\}$ **.**
show *?case* **proof fix** *i* **assume** $i \in \{1..Suc\ k\}$ **show** $S\ i \cap R = \{\}$
proof (*cases* ‹$i = Suc\ k$›)
case *True* **with** *Suc.prems prem* **show** *?thesis* **by** *simp*
next
case *False* **with** *IH* ‹$i \in \{1..Suc\ k\}$› **show** *?thesis* **by** *simp*
qed
qed
qed *simp*

lemma *min-correct*: $[\![\ i \in \{1..k\};\ S\ i \cap R \neq \{\}\ ]\!] \Longrightarrow cost\ R\ (min\text{-}arg\ R\ k) \le cost\ R\ i$
proof (*induction k*)
case (*Suc k*)
show *?case* **proof** (*cases* ‹$i = Suc\ k$›)

case *True* **with** *Suc.prems* **show** *?thesis* **by** (*auto simp*: *Let-def*)
next
case *False* **with** *Suc.prems Suc.IH* **have** *IH*: $cost\ R\ (min\text{-}arg\ R\ k) \leq cost\ R\ i$ **by** *simp*
from *Suc.prems False min-empty*[*of R k*] **have** $S\ (min\text{-}arg\ R\ k) \cap R \neq \{\}$ **by** *force*
with *IH* **show** *?thesis* **by** (*auto simp*: *Let-def*)
qed
qed *simp*

Correctness holds quite trivially for both m = 0 and m > 0 (assuming a set cover can be found at all, otherwise algorithm would not terminate).

lemma *set-cover-correct*:
$VARS\ (R :: 'a\ set)\ (C :: nat\ set)\ (i :: nat)$
$\{True\}$
$R := U;\ C := \{\};$
$WHILE\ R \neq \{\}\ INV\ \{R \subseteq U \wedge sc\ C\ (U - R)\}\ DO$
$i := min\text{-}arg\ R\ m;$
$R := R - S\ i;$
$C := C \cup \{i\}$
OD
$\{sc\ C\ U\}$
proof (*vcg*, *goal-cases*)
case *2* **show** *?case* **proof** (*cases m*)
case *0*
from *empty-cover*[*OF this*] *2* **show** *?thesis* **by** (*auto simp*: *sc-def*)
next
case *Suc* **then have** $m > 0$ **by** *simp*
from *min-in-range*[*OF this*] *2* **show** *?thesis* **using** *S-subset* **by** (*auto simp*: *sc-def*)
qed
qed (*auto simp*: *sc-def*)

definition *c-exists* :: $nat\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
$c\text{-}exists\ C\ R = (\exists c.\ sum\ w\ C = sum\ c\ (U - R) \wedge (\forall i.\ 0 \leq c\ i)$
$\wedge (\forall k \in \{1..m\}.\ sum\ c\ (S\ k \cap (U - R))$
$\leq (\sum j = card\ (S\ k \cap R) + 1..card\ (S\ k).\ inverse\ j) * w\ k))$

definition *inv* :: $nat\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
$inv\ C\ R \longleftrightarrow sc\ C\ (U - R) \wedge R \subseteq U \wedge c\text{-}exists\ C\ R$

lemma *invI*:
assumes $sc\ C\ (U - R)\ R \subseteq U$
$\exists c.\ sum\ w\ C = sum\ c\ (U - R) \wedge (\forall i.\ 0 \leq c\ i)$
$\wedge (\forall k \in \{1..m\}.\ sum\ c\ (S\ k \cap (U - R))$
$\leq (\sum j = card\ (S\ k \cap R) + 1..card\ (S\ k).\ inverse\ j) * w\ k)$
shows *inv C R* **using** *assms* **by** (*auto simp*: *inv-def c-exists-def*)

lemma *invD*:

```
  assumes inv C R
  shows sc C (U − R) R ⊆ U
        ∃ c. sum w C = sum c (U − R) ∧ (∀ i. 0 ≤ c i)
      ∧ (∀ k ∈ {1..m}. sum c (S k ∩ (U − R))
                    ≤ (∑ j = card (S k ∩ R) + 1..card (S k). inverse j) ∗ w k)
  using assms by (auto simp: inv-def c-exists-def)

lemma inv-init: inv {} U
proof (rule invI, goal-cases)
  case 3
  let ?c = (λ-. 0) :: 'a ⇒ real
  have sum w {} = sum ?c (U − U) by simp
  moreover {
    have ∀ k ∈ {1..m}. 0 ≤ (∑ j = card (S k ∩ U) + 1..card (S k). inverse j) ∗
w k
      by (simp add: sum-nonneg w-nonneg)
    then have (∀ k∈{1..m}. sum ?c (S k ∩ (U − U))
           ≤ (∑ j = card (S k ∩ U) + 1..card (S k). inverse j) ∗ w k) by simp
  }
  ultimately show ?case by blast
qed (simp-all add: sc-def)

lemma inv-step:
  assumes inv C R R ≠ {}
  defines [simp]: i ≡ min-arg R m
  shows inv (C ∪ {i}) (R − (S i))
proof (cases m)
  case 0
  from empty-cover[OF this] invD(2)[OF assms(1)] have R = {} by blast
  then show ?thesis using assms(2) by simp
next
  case Suc then have 0 < m by simp
  note hyp = invD[OF assms(1)]
  show ?thesis proof (rule invI, goal-cases)
      — Correctness
    case 1 have i ∈ {1..m} using min-in-range[OF ‹0 < m›] by simp
    with hyp(1) S-subset show ?case by (auto simp: sc-def)
  next
    case 2 from hyp(2) show ?case by auto
  next
    case 3
      — Set Cover grows
    have ∃ i ∈ {1..m}. S i ∩ R ≠ {}
      using assms(2) U-def hyp(2) by blast
    then have S i ∩ R ≠ {} using min-empty by auto
    then have 0 < card (S i ∩ R)
      using S-finite min-in-range[OF ‹0 < m›] by auto

      — Proving properties of cost function
```

from *hyp*(*3*) **obtain** *c* **where** $sum\ w\ C = sum\ c\ (U - R)$ $\forall i.\ 0 \leq c\ i$ **and**
SUM: $\forall k\in\{1..m\}.\ sum\ c\ (S\ k \cap (U - R))$
$\leq (\sum j = card\ (S\ k \cap R) + 1..card\ (S\ k).\ inverse\ j) * w\ k$ **by** *blast*
let $?c = (\lambda x.\ if\ x \in S\ i \cap R\ then\ cost\ R\ i\ else\ c\ x)$

— Proof of Lemma 11.9
have $finite\ (U - R)$ $finite\ (S\ i \cap R)$ $(U - R) \cap (S\ i \cap R) = \{\}$
using *U-finite S-finite min-in-range*[*OF* ‹$0 < m$›] **by** *auto*
then have $sum\ ?c\ (U - R \cup (S\ i \cap R)) = sum\ ?c\ (U - R) + sum\ ?c\ (S\ i \cap R)$
by (*rule sum.union-disjoint*)
moreover have *U-split*: $U - (R - S\ i) = U - R \cup (S\ i \cap R)$ **using** *hyp*(*2*) **by** *blast*
moreover {
have $sum\ ?c\ (S\ i \cap R) = card\ (S\ i \cap R) * cost\ R\ i$ **by** *simp*
also have ... $= w\ i$ **unfolding** *cost-def* **using** ‹$0 < card\ (S\ i \cap R)$› **by** *simp*
finally have $sum\ ?c\ (S\ i \cap R) = w\ i$ **.**
}
ultimately have $sum\ ?c\ (U - (R - S\ i)) = sum\ ?c\ (U - R) + w\ i$ **by** *simp*
moreover {
have $C \cap \{i\} = \{\}$ **using** *hyp*(*1*) ‹$S\ i \cap R \neq \{\}$› **by** (*auto simp*: *sc-def*)
from *sum.union-disjoint*[*OF - - this*] **have** $sum\ w\ (C \cup \{i\}) = sum\ w\ C + w\ i$
using *hyp*(*1*) **by** (*auto simp*: *sc-def intro*: *finite-subset*)
}
ultimately have *1*: $sum\ w\ (C \cup \{i\}) = sum\ ?c\ (U - (R - S\ i))$ — Lemma 11.9
using ‹$sum\ w\ C = sum\ c\ (U - R)$› **by** *simp*

have *2*: $\forall i.\ 0 \leq ?c\ i$ **using** ‹$\forall i.\ 0 \leq c\ i$› *cost-nonneg* **by** *simp*

— Proof of Lemma 11.10
have *3*: $\forall k\in\{1..m\}.\ sum\ ?c\ (S\ k \cap (U - (R - S\ i)))$
$\leq (\sum j = card\ (S\ k \cap (R - S\ i)) + 1..card\ (S\ k).\ inverse\ j) * w\ k$
proof
fix *k* **assume** $k \in \{1..m\}$
let $?rem = S\ k \cap R$ — Remaining elements to be covered
let $?add = S\ k \cap S\ i \cap R$ — Elements that will be covered in this step
let $?cov = S\ k \cap (U - R)$ — Covered elements

— Transforming left and right sides
have $sum\ ?c\ (S\ k \cap (U - (R - S\ i))) = sum\ ?c\ (S\ k \cap (U - R \cup (S\ i \cap R)))$
unfolding *U-split* **..**
also have ... $= sum\ ?c\ (?cov \cup ?add)$
by (*simp add*: *Int-Un-distrib Int-assoc*)
also have ... $= sum\ ?c\ ?cov + sum\ ?c\ ?add$
by (*rule sum.union-disjoint*) (*insert S-finite* ‹$k \in$ -›, *auto*)
finally have *lhs*:

sum ?c $(S\ k \cap (U - (R - S\ i)))$ = *sum ?c ?cov* + *sum ?c ?add* **.**
have $S\ k \cap (R - S\ i)$ = *?rem* − *?add* **by** *blast*
then have *card* $(S\ k \cap (R - S\ i))$ = *card* (*?rem* − *?add*) **by** *simp*
also have ... = *card ?rem* − *card ?add*
using *S-finite* ‹$k \in$ -› **by** (*auto intro*: *card-Diff-subset*)
finally have *rhs*:
card $(S\ k \cap (R - S\ i))$ + *1* = *card ?rem* − *card ?add* + *1* **by** *simp*

— The apparent complexity of the remaining proof is deceiving. Much of this is just about convincing Isabelle that these sum transformations are allowed.
have *sum ?c ?add* = *card ?add* ∗ *cost R i* **by** *simp*
also have ... ≤ *card ?add* ∗ *cost R k*
proof (*cases ?rem* = {})
case *True*
then have *card ?add* = *0* **by** (*auto simp*: *card-eq-0-iff*)
then show *?thesis* **by** *simp*
next
case *False*
from *min-correct*[*OF* ‹$k \in$ -› *this*] **have** *cost R i* ≤ *cost R k* **by** *simp*
then show *?thesis* **by** (*simp add*: *mult-left-mono*)
qed
also have ... = *card ?add* ∗ *inverse* (*card ?rem*) ∗ *w k*
by (*simp add*: *cost-def divide-inverse-commute*)
also have ... = ($\sum$*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse* (*card ?rem*)) ∗ *w k*
proof −
have *card ?add* ≤ *card ?rem*
using *S-finite* ‹$k \in$ -› **by** (*blast intro*: *card-mono*)
then show *?thesis* **by** (*simp add*: *sum-distrib-left*)
qed
also have ... ≤ ($\sum$*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse j*) ∗ *w k*
proof −
have ∀*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse* (*card ?rem*) ≤ *inverse j*
by *force*
then have ($\sum$*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse* (*card ?rem*))
≤ ($\sum$*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse j*)
by (*blast intro*: *sum-mono*)
with *w-nonneg* **show** *?thesis* **by** (*blast intro*: *mult-right-mono*)
qed
finally have *sum ?c ?add*
≤ ($\sum$*j* ∈ {*card ?rem* − *card ?add* + *1* .. *card ?rem*}. *inverse j*) ∗ *w k* **.**
moreover from *SUM* **have** *sum ?c ?cov*
≤ ($\sum$*j* ∈ {*card ?rem* + *1* .. *card* (*S k*)}. *inverse j*) ∗ *w k*
using ‹$k \in$ {*1..m*}› **by** *simp*
ultimately have *sum ?c* $(S\ k \cap (U - (R - S\ i)))$

$\leq ((\sum j \in \{card\ ?rem - card\ ?add + 1\ ..\ card\ ?rem\}.\ inverse\ j) +$
$(\sum j \in \{card\ ?rem + 1\ ..\ card\ (S\ k)\}.\ inverse\ j)) * w\ k$
unfolding *lhs* **by** *argo*
also have ... $= (\sum j \in \{card\ ?rem - card\ ?add + 1\ ..\ card\ (S\ k)\}.\ inverse\ j) * w\ k$
proof $-$
have *sum-split*: $b \in \{a\ ..\ c\} \Longrightarrow sum\ f\ \{a\ ..\ c\} = sum\ f\ \{a\ ..\ b\} + sum\ f\ \{Suc\ b\ ..\ c\}$
for $f :: nat \Rightarrow real$ **and** $a\ b\ c :: nat$
proof $-$
assume $b \in \{a\ ..\ c\}$
then have $\{a\ ..\ b\} \cup \{Suc\ b\ ..\ c\} = \{a\ ..\ c\}$ **by** *force*
moreover have $\{a\ ..\ b\} \cap \{Suc\ b\ ..\ c\} = \{\}$
using ‹$b \in \{a\ ..\ c\}$› **by** *auto*
ultimately show *?thesis* **by** (*metis finite-atLeastAtMost sum.union-disjoint*)
qed

have $(\sum j \in \{card\ ?rem - card\ ?add + 1\ ..\ card\ (S\ k)\}.\ inverse\ j)$
$= (\sum j \in \{card\ ?rem - card\ ?add + 1\ ..\ card\ ?rem\}.\ inverse\ j)$
$+ (\sum j \in \{card\ ?rem + 1\ ..\ card\ (S\ k)\}.\ inverse\ j)$
proof (*cases* ‹$?add = \{\}$›)
case *False*
then have $0 < card\ ?add\ 0 < card\ ?rem$
using *S-finite* ‹$k \in$ -› **by** *fastforce+*
then have $Suc\ (card\ ?rem - card\ ?add) \leq card\ ?rem$ **by** *simp*
moreover have $card\ ?rem \leq card\ (S\ k)$
using *S-finite* ‹$k \in$ -› **by** (*simp add*: *card-mono*)
ultimately show *?thesis* **by** (*auto intro*: *sum-split*)
qed *simp*
then show *?thesis* **by** *algebra*
qed
finally show $sum\ ?c\ (S\ k \cap (U - (R - S\ i)))$
$\leq (\sum j \in \{card\ (S\ k \cap (R - S\ i)) + 1\ ..\ card\ (S\ k)\}.\ inverse\ j) * w\ k$
unfolding *rhs* **.**
qed

from *1 2 3* **show** *?case* **by** *blast*
qed
qed

lemma *cover-sum*:
fixes $c :: 'a \Rightarrow real$
assumes $sc\ C\ V\ \forall i.\ 0 \leq c\ i$
shows $sum\ c\ V \leq (\sum i \in C.\ sum\ c\ (S\ i))$
proof $-$
from *assms*(*1*) **have** *finite C* **by** (*auto simp*: *sc-def finite-subset*)
then show *?thesis* **using** *assms*(*1*)
proof (*induction C arbitrary*: *V rule*: *finite-induct*)
case (*insert i C*)

have *V-split*: $(\bigcup (S \text{ ` } insert\ i\ C)) = (\bigcup (S \text{ ` } C)) \cup S\ i$ **by** *auto*
have *finite*: *finite* $(\bigcup (S \text{ ` } C))$ *finite* $(S\ i)$
using *insert S-finite* **by** (*auto simp*: *sc-def*)

have $sum\ c\ (S\ i) - sum\ c\ (\bigcup (S \text{ ` } C) \cap S\ i) \leq sum\ c\ (S\ i)$
using *assms*(*2*) **by** (*simp add*: *sum-nonneg*)
then have $sum\ c\ (\bigcup (S \text{ ` } insert\ i\ C)) \leq sum\ c\ (\bigcup (S \text{ ` } C)) + sum\ c\ (S\ i)$
unfolding *V-split* **using** *sum-Un*[*OF finite*, *of c*] **by** *linarith*
moreover have $(\sum i{\in}insert\ i\ C.\ sum\ c\ (S\ i)) = (\sum i \in C.\ sum\ c\ (S\ i)) + sum\ c\ (S\ i)$
by (*simp add*: *insert.hyps*)
ultimately show *?case* **using** *insert* **by** (*fastforce simp*: *sc-def*)
qed (*simp add*: *sc-def*)
qed

abbreviation H :: *nat* $\Rightarrow$ *real* **where** $H \equiv harm$

definition *d-star* :: *nat* (d^*) **where** $d^* \equiv Max\ (card \text{ ` } (S \text{ ` } \{1..m\}))$

lemma *set-cover-bound*:
assumes *inv* C $\{\}$ *sc* C' U
shows $sum\ w\ C \leq H\ d^* * sum\ w\ C'$
proof −
from *invD*(*3*)[*OF assms*(*1*)] **obtain** *c* **where**
$sum\ w\ C = sum\ c\ U\ \forall i.\ 0 \leq c\ i$ **and** *H-bound*:
$\forall k \in \{1..m\}.\ sum\ c\ (S\ k) \leq H\ (card\ (S\ k)) * w\ k$ — Lemma 11.10
by (*auto simp*: *harm-def Int-absorb2 S-subset*)

have $\forall k \in \{1..m\}.\ card\ (S\ k) \leq d^*$ **by** (*auto simp*: *d-star-def*)
then have $\forall k \in \{1..m\}.\ H\ (card\ (S\ k)) \leq H\ d^*$ **by** (*auto simp*: *harm-def intro*!: *sum-mono2*)
with *H-bound* **have** $\forall k \in \{1..m\}.\ sum\ c\ (S\ k) \leq H\ d^* * w\ k$
by (*metis atLeastAtMost-iff atLeastatMost-empty-iff empty-iff mult-right-mono w-nonneg*)
moreover have $C' \subseteq \{1..m\}$ **using** *assms*(*2*) **by** (*simp add*: *sc-def*)
ultimately have $\forall i \in C'.\ sum\ c\ (S\ i) \leq H\ d^* * w\ i$ **by** *blast*
then have $(\sum i \in C'.\ sum\ c\ (S\ i)) \leq H\ d^* * sum\ w\ C'$
by (*auto simp*: *sum-distrib-left intro*: *sum-mono*)

have $sum\ w\ C = sum\ c\ U$ **by** *fact* — Lemma 11.9
also have ... $\leq (\sum i \in C'.\ sum\ c\ (S\ i))$ **by** (*rule cover-sum*[*OF assms*(*2*)]) *fact*
also have ... $\leq H\ d^* * sum\ w\ C'$ **by** *fact*
finally show *?thesis* **.**
qed

theorem *set-cover-approx*:
VARS $(R :: 'a\ set)\ (C :: nat\ set)\ (i :: nat)$
$\{True\}$
$R := U;\ C := \{\};$

$WHILE\ R \neq \{\}\ INV\ \{inv\ C\ R\}\ DO$
$i := min\text{-}arg\ R\ m;$
$R := R - S\ i;$
$C := C \cup \{i\}$
OD
$\{sc\ C\ U \wedge (\forall C'.\ sc\ C'\ U \longrightarrow sum\ w\ C \leq H\ d^* * sum\ w\ C')\}$
proof (*vcg*, *goal-cases*)
case *1* **show** *?case* **by** (*rule inv-init*)
next
case *2* **thus** *?case* **using** *inv-step* **..**
next
case (*3 R C i*)
then have *sc C U* **unfolding** *inv-def* **by** *auto*
with *3* **show** *?case* **by** (*auto intro*: *set-cover-bound*)
qed

end

end

3 Independent Set

theory *Approx-MIS-Hoare*
imports
HOL−Hoare.Hoare-Logic
HOL−Library.Disjoint-Sets
begin

The algorithm is classical, the proofs are inspired by the ones by Berghammer and Müller-Olm [1]. In particular the approximation ratio is improved from Δ+*1* to Δ.

3.1 Graph

A set set is simply a set of edges, where an edge is a 2-element set.

definition *independent-vertices* :: $'a\ set\ set \Rightarrow\ 'a\ set \Rightarrow bool$ **where**
$independent\text{-}vertices\ E\ S \longleftrightarrow S \subseteq \bigcup E \wedge (\forall v1\ v2.\ v1 \in S \wedge v2 \in S \longrightarrow \{v1, v2\} \notin E)$

locale *Graph-E* =
fixes E :: $'a\ set\ set$
assumes *finite-E*: *finite E*
assumes *edges2*: $e \in E \Longrightarrow card\ e = 2$
begin

fun *vertices* :: $'a\ set\ set \Rightarrow\ 'a\ set$ **where**
$vertices\ G = \bigcup G$

abbreviation V :: $'a\ set$ **where**
$V \equiv vertices\ E$

definition *approximation-miv* :: $nat \Rightarrow 'a\ set \Rightarrow bool$ **where**
$approximation\text{-}miv\ n\ S \longleftrightarrow independent\text{-}vertices\ E\ S \wedge (\forall S'.\ independent\text{-}vertices\ E\ S' \longrightarrow card\ S' \leq card\ S * n)$

fun *neighbors* :: $'a \Rightarrow 'a\ set$ **where**
$neighbors\ v = \{u.\ \{u,v\} \in E\}$

fun *degree-vertex* :: $'a \Rightarrow nat$ **where**
$degree\text{-}vertex\ v = card\ (neighbors\ v)$

abbreviation Δ :: *nat* **where**
$\Delta \equiv Max\{degree\text{-}vertex\ u | u.\ u \in V\}$

lemma *finite-edges*: $e \in E \Longrightarrow finite\ e$
  using *card-ge-0-finite* **and** *edges2* **by** *force*

lemma *finite-V*: *finite V*
  using *finite-edges* **and** *finite-E* **by** *auto*

lemma *finite-neighbors*: *finite* (*neighbors u*)
  using *finite-V* **and** *rev-finite-subset* [*of V neighbors u*] **by** *auto*

lemma *independent-vertices-finite*: $independent\text{-}vertices\ E\ S \Longrightarrow finite\ S$
  by (*metis rev-finite-subset independent-vertices-def vertices.simps finite-V*)

lemma *edge-ex-vertices*: $e \in E \Longrightarrow \exists\, u\ v.\ u \neq v \wedge e = \{u, v\}$
proof −
  assume $e \in E$
  then have $card\ e = Suc\ (Suc\ 0)$ **using** *edges2* **by** *auto*
  then show $\exists\, u\ v.\ u \neq v \wedge e = \{u, v\}$
    by (*metis card-eq-SucD insertI1*)
qed

lemma Δ-*pos* [*simp*]: $E = \{\} \vee 0 < \Delta$
proof *cases*
  assume $E = \{\}$
  then show $E = \{\} \vee 0 < \Delta$ **by** *auto*
next
  assume 1: $E \neq \{\}$
  then have $V \neq \{\}$ **using** *edges2* **by** *fastforce*
  moreover have $finite\ \{degree\text{-}vertex\ u\ | u.\ u \in V\}$
    by (*metis finite-V finite-imageI Setcompr-eq-image*)
  ultimately have 2: $\Delta \in \{degree\text{-}vertex\ u\ | u.\ u \in V\}$ **using** *Max-in* **by** *auto*
  have $\Delta \neq 0$
  proof
    assume $\Delta = 0$

with *2* **obtain** u **where** *3*: $u \in V$ **and** *4*: *degree-vertex* $u = 0$ **by** *auto*
from *3* **obtain** e **where** *5*: $e \in E$ **and** $u \in e$ **by** *auto*
moreover with *4* **have** $\forall v.\ \{u,\ v\} \neq e$ **using** *finite-neighbors insert-absorb* **by** *fastforce*
ultimately show *False* **using** *edge-ex-vertices* **by** *auto*
qed
then show $E = \{\} \vee 0 < \Delta$ **by** *auto*
qed

lemma Δ*-max-degree*: $u \in V \Longrightarrow$ *degree-vertex* $u \leq \Delta$
proof −
assume H: $u \in V$
have *finite* $\{$*degree-vertex* $u\ |u.\ u \in V\}$
by (*metis finite-V finite-imageI Setcompr-eq-image*)
with H **show** *degree-vertex* $u \leq \Delta$ **using** *Max-ge* **by** *auto*
qed

3.2 Wei's algorithm: $(\Delta+1)$-approximation

The 'functional' part of the invariant, used to prove that the algorithm produces an independent set of vertices.

definition *inv-iv* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow bool$ **where**
inv-iv $S\ X \longleftrightarrow$ *independent-vertices* $E\ S$
$\wedge X \subseteq V$
$\wedge (\forall v1 \in (V - X).\ \forall v2 \in S.\ \{v1,\ v2\} \notin E)$
$\wedge S \subseteq X$

Strenghten the invariant with an approximation ratio r:

definition *inv-approx* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow nat \Rightarrow bool$ **where**
inv-approx $S\ X\ r \longleftrightarrow$ *inv-iv* $S\ X \wedge$ *card* $X \leq$ *card* $S * r$

Preservation of the functional invariant:

lemma *inv-preserv*:
fixes S :: $'a\ set$
and X :: $'a\ set$
and x :: $'a$
assumes *inv*: *inv-iv* $S\ X$
and *x-def*: $x \in V - X$
shows *inv-iv* (*insert* $x\ S$) $(X \cup$ *neighbors* $x \cup \{x\})$
proof −
have *inv1*: *independent-vertices* $E\ S$
and *inv2*: $X \subseteq V$
and *inv3*: $S \subseteq X$
and *inv4*: $\forall v1\ v2.\ v1 \in (V - X) \wedge v2 \in S \longrightarrow \{v1,\ v2\} \notin E$
using *inv* **unfolding** *inv-iv-def* **by** *auto*
have *finite-S*: *finite* S **using** *inv1* **and** *independent-vertices-finite* **by** *auto*
have *S1*: $\forall y \in S.\ \{x,\ y\} \notin E$ **using** *inv4* **and** *x-def* **by** *blast*
have *S2*: $\forall x \in S.\ \forall y \in S.\ \{x,\ y\} \notin E$ **using** *inv1* **unfolding** *independent-vertices-def* **by** *metis*

```
have S3: v1 ∈ insert x S ⟹ v2 ∈ insert x S ⟹ {v1, v2} ∉ E for v1 v2
  proof –
    assume v1 ∈ insert x S and v2 ∈ insert x S
    then consider
        (a) v1 = x and v2 = x
      | (b) v1 = x and v2 ∈ S
      | (c) v1 ∈ S and v2 = x
      | (d) v1 ∈ S and v2 ∈ S
      by auto
    then show {v1, v2} ∉ E
    proof cases
      case a then show ?thesis using edges2 by force
    next
      case b then show ?thesis using S1 by auto
    next
      case c then show ?thesis using S1 by (metis doubleton-eq-iff)
    next
      case d then show ?thesis using S2 by auto
    qed
  qed

have independent-vertices E (insert x S)
  using S3 and inv1 and x-def unfolding independent-vertices-def by auto

moreover have X ∪ neighbors x ∪ {x} ⊆ V
proof
  fix xa
  assume xa ∈ X ∪ neighbors x ∪ {x}
  then consider (a) xa ∈ X | (b) xa ∈ neighbors x | (c) xa = x by auto
  then show xa ∈ V
  proof cases
    case a
    then show ?thesis using inv2 by blast
  next
    case b
    then show ?thesis by auto
  next
    case c
    then show ?thesis using x-def by blast
  qed
qed

moreover have insert x S ⊆ X ∪ neighbors x ∪ {x} using inv3 by auto

moreover have v1 ∈ V − (X ∪ neighbors x ∪ {x}) ⟹ v2 ∈ insert x S ⟹
{v1, v2} ∉ E for v1 v2
proof –
  assume H: v1 ∈ V − (X ∪ neighbors x ∪ {x}) and v2 ∈ insert x S
  then consider (a) v2 = x | (b) v2 ∈ S by auto
```

then show $\{v1, v2\} \notin E$
proof *cases*
case *a*
with *H* **have** *v1* $\notin$ *neighbors v2* **by** *blast*
then show *?thesis* **by** *auto*
next
case *b*
from *H* **have** *v1* $\in V - X$ **by** *blast*
with *b* **and** *inv4* **show** *?thesis* **by** *blast*
qed
qed

ultimately show *inv-iv* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$) **unfolding** *inv-iv-def* **by** *blast*
qed

lemma *inv-approx-preserv*:
assumes *inv*: *inv-approx S X* ($\Delta + 1$)
and *x-def*: $x \in V - X$
shows *inv-approx* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$) ($\Delta + 1$)
proof −
have *finite-S*: *finite S* **using** *inv* **and** *independent-vertices-finite*
unfolding *inv-approx-def inv-iv-def* **by** *auto*
have *Sx*: $x \notin S$ **using** *inv* **and** *x-def* **unfolding** *inv-approx-def inv-iv-def* **by** *blast*

from *inv* **have** *inv-iv S X* **unfolding** *inv-approx-def* **by** *auto*
with *x-def* **have** *inv-iv* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$)
proof (*intro inv-preserv*, *auto*) **qed**

moreover have *card* ($X \cup$ *neighbors x* $\cup \{x\}$) $\leq$ *card* (*insert x S*) $*$ ($\Delta + 1$)
proof −
have *degree-vertex x* $\leq \Delta$ **using** Δ*-max-degree* **and** *x-def* **by** *auto*
then have *card* (*neighbors x* $\cup \{x\}$) $\leq \Delta + 1$ **using** *card-Un-le* [*of neighbors x* $\{x\}$] **by** *auto*
then have *card* ($X \cup$ *neighbors x* $\cup \{x\}$) $\leq$ *card X* $+ \Delta + 1$ **using** *card-Un-le* [*of X neighbors x* $\cup \{x\}$] **by** *auto*
also have ... $\leq$ *card S* $*$ ($\Delta + 1$) $+ \Delta + 1$ **using** *inv* **unfolding** *inv-approx-def* **by** *auto*
also have ... $=$ *card* (*insert x S*) $*$ ($\Delta + 1$) **using** *finite-S* **and** *Sx* **by** *auto*
finally show *?thesis* **.**
qed

ultimately show *inv-approx* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$) ($\Delta + 1$)
unfolding *inv-approx-def* **by** *auto*
qed

lemma *inv-approx*: *independent-vertices E S* $\Longrightarrow$ *card V* $\leq$ *card S* $*$ *r* $\Longrightarrow$ *ap-*

```
proximation-miv r S
proof -
  assume 1: independent-vertices E S and 2: card V ≤ card S ∗ r
  have independent-vertices E S′ ⟹ card S′ ≤ card S ∗ r for S′
  proof -
    assume  independent-vertices E S′
    then have S′ ⊆ V unfolding independent-vertices-def by auto
    then have card S′ ≤ card V using finite-V and card-mono by auto
    also have ... ≤ card S ∗ r using 2 by auto
    finally show card S′ ≤ card S ∗ r .
  qed
  with 1 show approximation-miv r S unfolding approximation-miv-def by auto
qed

theorem wei-approx-Δ-plus-1:
VARS (S :: 'a set) (X :: 'a set) (x :: 'a)
  { True }
  S := {};
  X := {};
  WHILE X ≠ V
  INV { inv-approx S X (Δ + 1) }
  DO x := (SOME x. x ∈ V − X);
     S := insert x S;
     X := X ∪ neighbors x ∪ {x}
  OD
  { approximation-miv (Δ + 1) S }
proof (vcg, goal-cases)
  case (1 S X x)
   then show ?case unfolding inv-approx-def inv-iv-def independent-vertices-def
by auto
next
  case (2 S X x)

  let ?x = (SOME x. x ∈ V − X)
  have V − X ≠ {} using 2 unfolding inv-approx-def inv-iv-def by blast
  then have ?x ∈ V − X using some-in-eq by metis
  with 2 show ?case using inv-approx-preserv by auto
next
  case (3 S X x)
 then show ?case using inv-approx unfolding inv-approx-def inv-iv-def by auto
qed
```

3.3 Wei's algorithm: Δ-approximation

The previous approximation uses very little information about the optimal solution (it has at most as many vertices as the set itself). With some extra effort we can improve the ratio to Δ instead of $\Delta+1$. In order to do that we must show that among the vertices removed in each iteration, at most Δ could belong to an optimal solution. This requires carrying around a set P

(via a ghost variable) which records the vertices deleted in each iteration.

definition *inv-partition* :: *'a set* ⇒ *'a set* ⇒ *'a set set* ⇒ *bool* **where**
inv-partition S X P ⟷ *inv-iv S X*
$\wedge \bigcup P = X$
$\wedge$ ($\forall p \in P.\ \exists s \in V.\ p = \{s\} \cup$ *neighbors s*)
$\wedge$ *card P* = *card S*
$\wedge$ *finite P*

lemma *inv-partition-preserv*:
assumes *inv*: *inv-partition S X P*
and *x-def*: $x \in V - X$
shows *inv-partition* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$) (*insert* ($\{x\} \cup$ *neighbors x*) *P*)
proof −
have *finite-S*: *finite S* **using** *inv* **and** *independent-vertices-finite*
unfolding *inv-partition-def inv-iv-def* **by** *auto*
have *Sx*: $x \notin S$ **using** *inv* **and** *x-def* **unfolding** *inv-partition-def inv-iv-def* **by** *blast*

from *inv* **have** *inv-iv S X* **unfolding** *inv-partition-def* **by** *auto*
with *x-def* **have** *inv-iv* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$)
proof (*intro inv-preserv*, *auto*) **qed**

moreover have $\bigcup$(*insert* ($\{x\} \cup$ *neighbors x*) *P*) = $X \cup$ *neighbors x* $\cup \{x\}$
using *inv* **unfolding** *inv-partition-def* **by** *auto*

moreover have ($\forall p \in$*insert* ($\{x\} \cup$ *neighbors x*) *P*. $\exists s \in V.\ p = \{s\} \cup$ *neighbors s*)
using *inv* **and** *x-def* **unfolding** *inv-partition-def* **by** *auto*

moreover have *card* (*insert* ($\{x\} \cup$ *neighbors x*) *P*) = *card* (*insert x S*)
proof −
from *x-def* **and** *inv* **have** $x \notin \bigcup P$ **unfolding** *inv-partition-def* **by** *auto*
then have $\{x\} \cup$ *neighbors x* $\notin P$ **by** *auto*
then have *card* (*insert* ($\{x\} \cup$ *neighbors x*) *P*) = *card P* + *1* **using** *inv* **unfolding** *inv-partition-def* **by** *auto*
moreover have *card* (*insert x S*) = *card S* + *1* **using** *Sx* **and** *finite-S* **by** *auto*
ultimately show *?thesis* **using** *inv* **unfolding** *inv-partition-def* **by** *auto*
qed

moreover have *finite* (*insert* ($\{x\} \cup$ *neighbors x*) *P*)
using *inv* **unfolding** *inv-partition-def* **by** *auto*

ultimately show *inv-partition* (*insert x S*) ($X \cup$ *neighbors x* $\cup \{x\}$) (*insert* ($\{x\}$ $\cup$ *neighbors x*) *P*)
unfolding *inv-partition-def* **by** *auto*
qed

lemma *card-Union-le-sum-card*:

fixes U :: $'a$ *set set*
assumes $\forall u \in U.$ *finite* u
shows *card* $(\bigcup U) \leq$ *sum card* U
proof (*cases finite* U)
case *False*
then show *card* $(\bigcup U) \leq$ *sum card* U
using *card-eq-0-iff finite-UnionD* **by** *auto*
next
case *True*
then show *card* $(\bigcup U) \leq$ *sum card* U
proof (*induct* U *rule*: *finite-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert* x F)
then have *card*$(\bigcup$(*insert* x F)$) \leq$ *card*(x) + *card* $(\bigcup F)$ **using** *card-Un-le* **by** *auto*
also have ... $\leq$ *card*(x) + *sum card* F **using** *insert.hyps* **by** *auto*
also have ... = *sum card* (*insert* x F) **using** *sum.insert-if* **and** *insert.hyps* **by** *auto*
finally show *?case* **.**
qed
qed

lemma *sum-card*:
fixes U :: $'a$ *set set*
and n :: *nat*
assumes $\forall S \in U.$ *card* $S \leq n$
shows *sum card* $U \leq$ *card* $U * n$
proof *cases*
assume *infinite* $U \vee U = \{\}$
then have *sum card* $U = 0$ **using** *sum.infinite* **by** *auto*
then show *sum card* $U \leq$ *card* $U * n$ **by** *auto*
next
assume $\neg$(*infinite* $U \vee U = \{\}$)
with *assms* **have** *finite* U **and** $U \neq \{\}$**and** $\forall S \in U.$ *card* $S \leq n$ **by** *auto*
then show *sum card* $U \leq$ *card* $U * n$
proof (*induct* U *rule*: *finite-ne-induct*)
case (*singleton* x)
then show *?case* **by** *auto*
next
case (*insert* x F)
assume $\forall S \in$*insert* x $F.$ *card* $S \leq n$
then have *1*:*card* $x \leq n$ **and** *2*:*sum card* $F \leq$ *card* $F * n$ **using** *insert.hyps* **by** *auto*
then have *sum card* (*insert* x F) = *card* x + *sum card* F **using** *sum.insert-if* **and** *insert.hyps* **by** *auto*
also have ... $\leq n +$ *card* $F * n$ **using** *1* **and** *2* **by** *auto*

also have ... = *card* (*insert* x F) $*$ n **using** *card-insert-if* **and** *insert.hyps* **by** *auto*
finally show *?case* **.**
qed
qed

lemma *x-or-neighbors*:
fixes P :: *'a set set*
and S :: *'a set*
assumes *inv*: $\forall p{\in}P.\ \exists s \in V.\ p = \{s\} \cup$ *neighbors* s
and *ivS*: *independent-vertices* E S
shows $\forall p \in P.$ *card* $(S \cap p) \leq \Delta$
proof
fix p
assume $p \in P$
then obtain s **where** *1*: $s \in V \wedge p = \{s\} \cup$ *neighbors* s **using** *inv* **by** *blast*
then show *card* $(S \cap p) \leq \Delta$
proof *cases*
assume $s \in S$
then have $S \cap$ *neighbors* $s = \{\}$ **using** *ivS* **unfolding** *independent-vertices-def* **by** *auto*
then have $S \cap p \subseteq \{s\}$ **using** *1* **by** *auto*
then have *2*: *card* $(S \cap p) \leq 1$ **using** *subset-singletonD* **by** *fastforce*
consider (*a*) $E = \{\}$ | (*b*) $0 < \Delta$ **using** Δ*-pos* **by** *auto*
then show *card* $(S \cap p) \leq \Delta$
proof *cases*
case *a*
then have $S = \{\}$ **using** *ivS* **unfolding** *independent-vertices-def* **by** *auto*
then show *?thesis* **by** *auto*
next
case *b*
then show *?thesis* **using** *2* **by** *auto*
qed
next
assume $s \notin S$
with *1* **have** $S \cap p \subseteq$ *neighbors* s **by** *auto*
then have *card* $(S \cap p) \leq$ *degree-vertex* s **using** *card-mono* **and** *finite-neighbors* **by** *auto*
then show *card* $(S \cap p) \leq \Delta$ **using** *1* **and** Δ*-max-degree* [*of* s] **by** *auto*
qed
qed

lemma *inv-partition-approx*: *inv-partition* S V $P \Longrightarrow$ *approximation-miv* Δ S
proof $-$
assume *H1*: *inv-partition* S V P
then have *independent-vertices* E S **unfolding** *inv-partition-def inv-iv-def* **by** *auto*

moreover have *independent-vertices* $E\ S' \Longrightarrow$ *card* $S' \leq$ *card* $S * \Delta$ **for** S'
proof −
let *?I* = $\{S' \cap p \mid p.\ p \in P\}$

assume *H2*: *independent-vertices* $E\ S'$
then have $S' \subseteq V$ **unfolding** *independent-vertices-def* **using** *vertices.simps* **by** *blast*
with *H1* **have** $S' = S' \cap \bigcup P$ **unfolding** *inv-partition-def* **by** *auto*
then have $S' = (\bigcup p \in P.\ S' \cap p)$ **using** *Int-Union* **by** *auto*
then have $S' = \bigcup$ *?I* **by** *blast*
moreover have *finite* S' **using** *H2* **and** *independent-vertices-finite* **by** *auto*
then have $p \in P \Longrightarrow$ *finite* $(S' \cap p)$ **for** p **by** *auto*
ultimately have *card* $S' \leq$ *sum card ?I* **using** *card-Union-le-sum-card* [*of ?I*] **by** *auto*
also have ... $\leq$ *card ?I* $* \Delta$
using *x-or-neighbors* [*of P S'*]
and *sum-card* [*of ?I* Δ]
and *H1* **and** *H2* **unfolding** *inv-partition-def* **by** *auto*
also have ... $\leq$ *card* $P * \Delta$
proof −
have *finite P* **using** *H1* **unfolding** *inv-partition-def* **by** *auto*
then have *card ?I* $\leq$ *card P*
using *Setcompr-eq-image* [*of* $\lambda p.\ S' \cap p$ *P*]
and *card-image-le* **unfolding** *inv-partition-def* **by** *auto*
then show *?thesis* **by** *auto*
qed
also have ... = *card* $S * \Delta$ **using** *H1* **unfolding** *inv-partition-def* **by** *auto*
ultimately show *card* $S' \leq$ *card* $S * \Delta$ **by** *auto*
qed
ultimately show *approximation-miv* Δ *S* **unfolding** *approximation-miv-def* **by** *auto*
qed

theorem *wei-approx-*Δ:
VARS $(S :: {'a}\ set)$ $(X :: {'a}\ set)$ $(x :: {'a})$
{ *True* }
$S := \{\}$;
$X := \{\}$;
WHILE $X \neq V$
INV { $\exists P.$ *inv-partition S X P* }
DO $x :=$ (*SOME* $x.\ x \in V - X$);
$S :=$ *insert x S*;
$X := X \cup$ *neighbors* $x \cup \{x\}$
OD
{ *approximation-miv* Δ *S* }
proof (*vcg*, *goal-cases*)
case (*1 S X x*)

have *inv-partition* {} {} {} **unfolding** *inv-partition-def inv-iv-def independent-vertices-def*

by *auto*
  then show *?case* **by** *auto*
next
  case (*2 S X x*)

  let *?x* = (*SOME x. x* ∈ *V* − *X*)
  from *2* **obtain** *P* **where** *I*: *inv-partition S X P* **by** *auto*
  then have *V* − *X* ≠ {} **using** *2* **unfolding** *inv-partition-def* **by** *auto*
  then have *?x* ∈ *V* − *X* **using** *some-in-eq* **by** *metis*

  with *I* **have** *inv-partition* (*insert ?x S*) (*X* ∪ *neighbors ?x* ∪ {*?x*}) (*insert* ({*?x*} ∪ *neighbors ?x*) *P*)
    using *inv-partition-preserv* **by** *blast*
  then show *?case* **by** *auto*
next
  case (*3 S X x*)
  then show *?case* **using** *inv-partition-approx* **unfolding** *inv-approx-def* **by** *auto*
qed

3.4 Wei's algorithm with dynamically computed approximation ratio

In this subsection, we augment the algorithm with a variable used to compute the effective approximation ratio of the solution. In addition, the vertex of smallest degree is picked. With this heuristic, the algorithm achieves an approximation ratio of $(\Delta+2)/3$, but this is not proved here.

definition *vertex-heuristic* :: *'a set* ⇒ *'a* ⇒ *bool* **where**
vertex-heuristic X v = (∀ *u* ∈ *V* − *X*. *card* (*neighbors v* − *X*) ≤ *card* (*neighbors u* − *X*))

lemma *ex-min-finite-set*:
  fixes *S* :: *'a set*
    and *f* :: *'a* ⇒ *nat*
    shows *finite S* ⟹ *S* ≠ {} ⟹ ∃ *x*. *x* ∈ *S* ∧ (∀ *y* ∈ *S*. *f x* ≤ *f y*)
        (**is** *?P1* ⟹ *?P2* ⟹ ∃ *x*. *?minf S x*)
proof (*induct S rule*: *finite-ne-induct*)
  case (*singleton x*)
  have *?minf* {*x*} *x* **by** *auto*
  then show *?case* **by** *auto*
next
  case (*insert x F*)
  from *insert*(*4*) **obtain** *y* **where** *Py*: *?minf F y* **by** *auto*
  show ∃ *z*. *?minf* (*insert x F*) *z*
  proof *cases*
    assume *f x* < *f y*
    then have *?minf* (*insert x F*) *x* **using** *Py* **by** *auto*
    then show *?case* **by** *auto*

next
assume $\neg f\ x < f\ y$
then have *?minf* (*insert* x F) y **using** *Py* **by** *auto*
then show *?case* **by** *auto*
qed
qed

lemma *inv-approx-preserv2*:
fixes S :: *'a set*
and X :: *'a set*
and s :: *nat*
and x :: *'a*
assumes *inv*: *inv-approx* S X s
and *x-def*: $x \in V - X$
shows *inv-approx* (*insert* x S) ($X \cup$ *neighbors* $x \cup \{x\}$) (*max* (*card* (*neighbors* $x \cup \{x\} - X$)) s)
proof −
have *finite-S*: *finite* S **using** *inv* **and** *independent-vertices-finite* **unfolding** *inv-approx-def inv-iv-def* **by** *auto*
have *Sx*: $x \notin S$ **using** *inv* **and** *x-def* **unfolding** *inv-approx-def inv-iv-def* **by** *blast*

from *inv* **have** *inv-iv* S X **unfolding** *inv-approx-def* **by** *auto*
with *x-def* **have** *inv-iv* (*insert* x S) ($X \cup$ *neighbors* $x \cup \{x\}$)
proof (*intro inv-preserv*, *auto*) **qed**

moreover have *card* ($X \cup$ *neighbors* $x \cup \{x\}$) $\leq$ *card* (*insert* x S) $*$ *max* (*card* (*neighbors* $x \cup \{x\} - X$)) s
proof −
let $?N$ = *neighbors* $x \cup \{x\} - X$
have *card* ($X \cup ?N$) $\leq$ *card* X + *card* $?N$ **using** *card-Un-le* [*of* X $?N$] **by** *auto*
also have ... $\leq$ *card* $S * s$ + *card* $?N$ **using** *inv* **unfolding** *inv-approx-def* **by** *auto*
also have ... $\leq$ *card* S $*$ *max* (*card* $?N$) s + *card* $?N$ **by** *auto*
also have ... $\leq$ *card* S $*$ *max* (*card* $?N$) s + *max* (*card* $?N$) s **by** *auto*
also have ... = *card* (*insert* x S) $*$ *max* (*card* $?N$) s **using** *Sx* **and** *finite-S* **by** *auto*
finally show *?thesis* **by** *auto*
qed

ultimately show *inv-approx* (*insert* x S) ($X \cup$ *neighbors* $x \cup \{x\}$) (*max* (*card* (*neighbors* $x \cup \{x\} - X$)) s)
unfolding *inv-approx-def* **by** *auto*
qed

theorem *wei-approx-min-degree-heuristic*:
VARS (S :: *'a set*) (X :: *'a set*) (x :: *'a*) (r :: *nat*)
{ *True* }
$S := \{\}$;

$X := \{\}$;
$r := 0$;
WHILE $X \neq V$
INV { *inv-approx S X r* }
DO $x := (SOME\ x.\ x \in V - X \wedge vertex\text{-}heuristic\ X\ x)$;
$S := insert\ x\ S$;
$r := max\ (card\ (neighbors\ x \cup \{x\} - X))\ r$;
$X := X \cup neighbors\ x \cup \{x\}$
OD
{ *approximation-miv r S* }
proof (*vcg*, *goal-cases*)
case (*1 S X x r*)
then show *?case* **unfolding** *inv-approx-def inv-iv-def independent-vertices-def* **by** *auto*
next
case (*2 S X x r*)

let $?x = (SOME\ x.\ x \in V - X \wedge vertex\text{-}heuristic\ X\ x)$
have $V - X \neq \{\}$ **using** *2* **unfolding** *inv-approx-def inv-iv-def* **by** *blast*
moreover have *finite* $(V - X)$ **using** *2* **and** *finite-V* **by** *auto*
ultimately have $\exists x.\ x \in V - X \wedge vertex\text{-}heuristic\ X\ x$
using *ex-min-finite-set* [**where** $?f = \lambda x.\ card\ (neighbors\ x - X)$]
unfolding *vertex-heuristic-def* **by** *auto*
then have *x-def*: $?x \in V - X \wedge vertex\text{-}heuristic\ X\ ?x$
using *someI-ex* [**where** $?P = \lambda x.\ x \in V - X \wedge vertex\text{-}heuristic\ X\ x$] **by** *auto*
with *2* **show** *?case* **using** *inv-approx-preserv2* **by** *auto*
next
case (*3 S X x r*)
then show *?case* **using** *inv-approx* **unfolding** *inv-approx-def inv-iv-def* **by** *auto*
qed

end
end

4 Load Balancing

theory *Approx-LB-Hoare*
imports *Complex-Main HOL−Hoare.Hoare-Logic*
begin

This is a formalization of the load balancing algorithms and proofs in the book by Kleinberg and Tardos [4].

hide-const (**open**) *sorted*

lemma *sum-le-card-Max*: $[\![\ finite\ A;\ A \neq \{\}\ ]\!] \Longrightarrow sum\ f\ A \leq card\ A * Max\ (f\ `\ A)$
proof(*induction A rule*: *finite-ne-induct*)

case (*singleton x*)
then show *?case* **by** *simp*
next
case (*insert x F*)
then show *?case* **by** (*auto simp*: *max-def order.trans*[*of sum f F card F* $*$ *Max* (*f* ‘ *F*)]])
qed

lemma *Max-const*[*simp*]: ⟦ *finite A*; $A \neq \{\}$ ⟧ $\Longrightarrow$ *Max* ((λ-. *c*) ‘ *A*) = *c*
using *Max-in image-is-empty* **by** *blast*

abbreviation Max_0 :: *nat set* ⇒ *nat* **where**
Max_0 *N* ≡ (*if N*={} *then 0 else Max N*)

fun *f-*Max_0 :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat* **where**
f-Max_0 *f 0* = *0*
| *f-*Max_0 *f* (*Suc x*) = *max* (*f* (*Suc x*)) (*f-*Max_0 *f x*)

lemma *f-*Max_0*-equiv*: *f-*Max_0 *f n* = Max_0 (*f* ‘ {*1..n*})
by (*induction n*) (*auto simp*: *not-le atLeastAtMostSuc-conv*)

lemma *f-*Max_0*-correct*:
∀ *x* ∈ {*1..m*}. *T x* ≤ *f-*Max_0 *T m*
m > *0* ⟹ ∃ *x* ∈ {*1..m*}. *T x* = *f-*Max_0 *T m*
apply (*induction m*)
apply *simp-all*
apply (*metis atLeastAtMost-iff le-Suc-eq max.cobounded1 max.coboundedI2*)
subgoal for *m* **by** (*cases* ‹*m* = *0*›) (*auto simp*: *max-def*)
done

lemma *f-*Max_0*-mono*:
y ≤ *T x* ⟹ *f-*Max_0 (*T* (*x* := *y*)) *m* ≤ *f-*Max_0 *T m*
T x ≤ *y* ⟹ *f-*Max_0 *T m* ≤ *f-*Max_0 (*T* (*x* := *y*)) *m*
by (*induction m*) *auto*

lemma *f-*Max_0*-out-of-range* [*simp*]:
x ∉ {*1..k*} ⟹ *f-*Max_0 (*T* (*x* := *y*)) *k* = *f-*Max_0 *T k*
by (*induction k*) *auto*

lemma *fun-upd-f-*Max_0:
assumes *x* ∈ {*1..m*} *T x* ≤ *y*
shows *f-*Max_0 (*T* (*x* := *y*)) *m* = *max y* (*f-*Max_0 *T m*)
using *assms* **by** (*induction m*) *auto*

locale *LoadBalancing* =
fixes *t* :: *nat* ⇒ *nat*
and *m* :: *nat*
and *n* :: *nat*
assumes *m-gt-0*: *m* > *0*

begin

4.1 Formalization of a Correct Load Balancing

4.1.1 Definition

definition *lb* :: $(nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat\ set) \Rightarrow nat \Rightarrow bool$ **where**
$lb\ T\ A\ j = ((\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\})$ — No job is assigned to more than one machine
$\wedge\ (\bigcup x \in \{1..m\}.\ A\ x) = \{1..j\}$ — Every job is assigned
$\wedge\ (\forall x \in \{1..m\}.\ (\sum j \in A\ x.\ t\ j) = T\ x)$ — The processing times sum up to the correct load)

abbreviation *makespan* :: $(nat \Rightarrow nat) \Rightarrow nat$ **where**
$makespan\ T \equiv f\text{-}Max_0\ \ T\ m$

lemma *makespan-def'*: $makespan\ T = Max\ (T\ `\ \{1..m\})$
using *m-gt-0* **by** (*simp add*: $f\text{-}Max_0\text{-}equiv$)

lemma *makespan-correct*:
$\forall x \in \{1..m\}.\ T\ x \leq makespan\ T$
$\exists x \in \{1..m\}.\ T\ x = makespan\ T$
using $f\text{-}Max_0\text{-}correct$ *m-gt-0* **by** *auto*

lemma *lbE*:
assumes $lb\ T\ A\ j$
shows $\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\}$
$(\bigcup x \in \{1..m\}.\ A\ x) = \{1..j\}$
$\forall x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y) = T\ x$
using *assms* **unfolding** *lb-def* **by** *blast+*

lemma *lbI*:
assumes $\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\}$
$(\bigcup x \in \{1..m\}.\ A\ x) = \{1..j\}$
$\forall x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y) = T\ x$
shows $lb\ T\ A\ j$ **using** *assms* **unfolding** *lb-def* **by** *blast*

lemma *A-lb-finite* [*simp*]:
assumes $lb\ T\ A\ j\ x \in \{1..m\}$
shows *finite* $(A\ x)$
by (*metis lbE*(*2*) *assms finite-UN finite-atLeastAtMost*)

If $A\ x$ is pairwise disjoint for all $x \in \{1..m\}$, then the the sum over the sums of the individual $A\ x$ is equal to the sum over the union of all $A\ x$.

lemma *sum-sum-eq-sum-Un*:
fixes $A :: nat \Rightarrow nat\ set$
assumes $\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\}$
and $\forall x \in \{1..m\}.\ finite\ (A\ x)$
shows $(\sum x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y)) = (\sum x \in (\bigcup y \in \{1..m\}.\ A\ y).\ t\ x)$

using *assms*
proof (*induction m*)
case (*Suc m*)
have *FINITE*: $\mathit{finite}\ (\bigcup x \in \{1..m\}.\ A\ x)\ \mathit{finite}\ (A\ (\mathit{Suc}\ m))$
using *Suc.prems*(*2*) **by** *auto*
have $\forall x \in \{1..m\}.\ A\ x \cap A\ (\mathit{Suc}\ m) = \{\}$
using *Suc.prems*(*1*) **by** *simp*
then have *DISJNT*: $(\bigcup x \in \{1..m\}.\ A\ x) \cap (A\ (\mathit{Suc}\ m)) = \{\}$ **using** *Union-disjoint* **by** *blast*
have $(\sum x \in (\bigcup y \in \{1..m\}.\ A\ y).\ t\ x) + (\sum x \in A\ (\mathit{Suc}\ m).\ t\ x)$
$= (\sum x \in ((\bigcup y \in \{1..m\}.\ A\ y) \cup A\ (\mathit{Suc}\ m)).\ t\ x)$
using *sum.union-disjoint*[*OF FINITE DISJNT*, *symmetric*] **.**
also have ... $= (\sum x \in (\bigcup y \in \{1..\mathit{Suc}\ m\}.\ A\ y).\ t\ x)$
by (*metis UN-insert image-Suc-lessThan image-insert inf-sup-aci*(*5*) *lessThan-Suc*)
finally show *?case* **using** *Suc* **by** *auto*
qed *simp*

If T and A are a correct load balancing for j jobs and m machines, then the sum of the loads has to be equal to the sum of the processing times of the jobs

lemma *lb-impl-job-sum*:
assumes $\mathit{lb}\ T\ A\ j$
shows $(\sum x \in \{1..m\}.\ T\ x) = (\sum x \in \{1..j\}.\ t\ x)$
proof −
note *lbrules* = *lbE*[*OF assms*]
from *assms* **have** *FINITE*: $\forall x \in \{1..m\}.\ \mathit{finite}\ (A\ x)$ **by** *simp*
have $(\sum x \in \{1..m\}.\ T\ x) = (\sum x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y))$
using *lbrules*(*3*) **by** *simp*
also have ... $= (\sum x \in \{1..j\}.\ t\ x)$
using *sum-sum-eq-sum-Un*[*OF lbrules*(*1*) *FINITE*]
unfolding *lbrules*(*2*) **.**
finally show *?thesis* **.**
qed

4.1.2 Lower Bounds for the Makespan

If T and A are a correct load balancing for j jobs and m machines, then the processing time of any job $x \in \{1..j\}$ is a lower bound for the load of some machine $y \in \{1..m\}$

lemma *job-lower-bound-machine*:
assumes $\mathit{lb}\ T\ A\ j\ x \in \{1..j\}$
shows $\exists y \in \{1..m\}.\ t\ x \le T\ y$
proof −
note *lbrules* = *lbE*[*OF assms*(*1*)]
have $\exists y \in \{1..m\}.\ x \in A\ y$ **using** *lbrules*(*2*) *assms*(*2*) **by** *blast*
then obtain *y* **where** *y-def*: $y \in \{1..m\}\ x \in A\ y$ **..**
moreover have $\mathit{finite}\ (A\ y)$ **using** *assms*(*1*) *y-def*(*1*) **by** *simp*
ultimately have $t\ x \le (\sum x \in A\ y.\ t\ x)$ **using** *lbrules*(*1*) *member-le-sum* **by** *fast*

also have ... = *T y* **using** *lbrules(3) y-def(1)* **by** *blast*
finally show *?thesis* **using** *y-def(1)* **by** *blast*
qed

As the load of any machine is a lower bound for the makespan, the processing time of any job $x \in \{1..j\}$ has to also be a lower bound for the makespan. Follows from *job-lower-bound-machine* and *makespan-correct*.

lemma *job-lower-bound-makespan*:
assumes $lb\ T\ A\ j\ x \in \{1..j\}$
shows $t\ x \leq makespan\ T$
by (*meson job-lower-bound-machine*[*OF assms*] *makespan-correct*(*1*) *le-trans*)

The makespan over j jobs is a lower bound for the makespan of any correct load balancing for j jobs.

lemma *max-job-lower-bound-makespan*:
assumes $lb\ T\ A\ j$
shows $Max_0\ (t\ `\ \{1..j\}) \leq makespan\ T$
using *job-lower-bound-makespan*[*OF assms*] **by** *fastforce*

lemma *job-dist-lower-bound-makespan*:
assumes $lb\ T\ A\ j$
shows $(\sum x \in \{1..j\}.\ t\ x)\ /\ m \leq makespan\ T$
proof −
have $(\sum x \in \{1..j\}.\ t\ x) \leq m * makespan\ T$
using *assms lb-impl-job-sum*[*symmetric*]
and *sum-le-card-Max*[*of* {*1..m*}] *m-gt-0* **by** (*simp add*: *makespan-def'*)
then have $real\ (\sum x \in \{1..j\}.\ t\ x) \leq real\ m * real\ (makespan\ T)$
using *of-nat-mono* **by** *fastforce*
then show *?thesis* **by** (*simp add*: *field-simps m-gt-0*)
qed

4.2 The Greedy Approximation Algorithm

This function will perform a linear scan from $k \in \{1..m\}$ and return the index of the machine with minimum load assuming $m > 0$

fun *min-arg* :: $(nat \Rightarrow nat) \Rightarrow nat \Rightarrow nat$ **where**
min-arg T 0 = 1
| *min-arg T (Suc x) =*
(*let k = min-arg T x*
in if T (Suc x) < T k then (Suc x) else k)

lemma *min-correct*:
$\forall x \in \{1..m\}.\ T\ (min\text{-}arg\ T\ m) \leq T\ x$
by (*induction m*) (*auto simp*: *Let-def le-Suc-eq*, *force*)

lemma *min-in-range*:
$k > 0 \Longrightarrow (min\text{-}arg\ T\ k) \in \{1..k\}$
by (*induction k*) (*auto simp*: *Let-def*, *force+*)

lemma *add-job*:
  assumes $lb\ T\ A\ j\ x \in \{1..m\}$
  shows $lb\ (T\ (x := T\ x + t\ (Suc\ j)))\ (A\ (x := A\ x \cup \{Suc\ j\}))\ (Suc\ j)$
    (**is** ‹$lb\ ?T\ ?A$ -›)
proof −
  note $lbrules = lbE[OF\ assms(1)]$

  — Rule 1: $A(x := A\ x \cup \{Suc\ j\})$ pairwise disjoint
  have $NOTIN$: $\forall i \in \{1..m\}.\ Suc\ j \notin A\ i$ **using** $lbrules(2)\ assms(2)$ **by** *force*
  with $lbrules(1)$ **have** $\forall i \in \{1..m\}.\ i \neq x \longrightarrow A\ i \cap (A\ x \cup \{Suc\ j\}) = \{\}$
    using $assms(2)$ **by** *blast*
  then have 1: $\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow ?A\ x \cap ?A\ y = \{\}$
    using $lbrules(1)$ **by** *simp*

  — Rule 2: $A(x := A\ x \cup \{Suc\ j\})$ contains all jobs
  have $(\bigcup y \in \{1..m\}.\ ?A\ y) = (\bigcup y \in \{1..m\}.\ A\ y) \cup \{Suc\ j\}$
    using *UNION-fun-upd* $assms(2)$ **by** *auto*
  also have ... $= \{1..Suc\ j\}$ **unfolding** $lbrules(2)$ **by** *auto*
  finally have 2: $(\bigcup y \in \{1..m\}.\ ?A\ y) = \{1..Suc\ j\}$ **.**

  — Rule 3: $A(x := A\ x \cup \{Suc\ j\})$ sums to $T(x := T\ x + t\ (Suc\ j))$
  have $(\sum i \in ?A\ x.\ t\ i) = (\sum i \in A\ x \cup \{Suc\ j\}.\ t\ i)$ **by** *simp*
  moreover have $A\ x \cap \{Suc\ j\} = \{\}$ **using** $NOTIN\ assms(2)$ **by** *blast*
  moreover have $finite\ (A\ x)\ finite\ \{Suc\ j\}$ **using** *assms* **by** *simp+*
  ultimately have $(\sum i \in ?A\ x.\ t\ i) = (\sum i \in A\ x.\ t\ i) + (\sum i \in \{Suc\ j\}.\ t\ i)$
    using *sum.union-disjoint* **by** *simp*
  also have ... $= T\ x + t\ (Suc\ j)$ **using** $lbrules(3)\ assms(2)$ **by** *simp*
  finally have $(\sum i \in ?A\ x.\ t\ i) = ?T\ x$ **by** *simp*
  then have 3: $\forall i \in \{1..m\}.\ (\sum j \in ?A\ i.\ t\ j) = ?T\ i$
    using $lbrules(3)\ assms(2)$ **by** *simp*

  from $lbI[OF\ 1\ 2\ 3]$ **show** *?thesis* **.**
qed

lemma *makespan-mono*:
  $y \leq T\ x \Longrightarrow makespan\ (T\ (x := y)) \leq makespan\ T$
  $T\ x \leq y \Longrightarrow makespan\ T \leq makespan\ (T\ (x := y))$
  using $f\text{-}Max_0\text{-}mono$ **by** *auto*

lemma *smaller-optimum*:
  assumes $lb\ T\ A\ (Suc\ j)$
  shows $\exists T'\ A'.\ lb\ T'\ A'\ j \wedge makespan\ T' \leq makespan\ T$
proof −
  note $lbrules = lbE[OF\ assms]$
  have $\exists x \in \{1..m\}.\ Suc\ j \in A\ x$ **using** $lbrules(2)$ **by** *auto*
  then obtain x **where** $x\text{-}def$: $x \in \{1..m\}\ Suc\ j \in A\ x$ **..**
  let $?T = T\ (x := T\ x - t\ (Suc\ j))$
  let $?A = A\ (x := A\ x - \{Suc\ j\})$

— Rule 1: $A(x := A\ x - \{Suc\ j\})$ pairwise disjoint
from *lbrules*(*1*) **have** $\forall\, i \in \{1..m\}.\ i \neq x \longrightarrow A\ i \cap (A\ x - \{Suc\ j\}) = \{\}$
using *x-def*(*1*) **by** *blast*
then have *1*: $\forall\, x \in \{1..m\}.\ \forall\, y \in \{1..m\}.\ x \neq y \longrightarrow ?A\ x \cap ?A\ y = \{\}$
using *lbrules*(*1*) **by** *auto*

— Rule 2: $A(x := A\ x - \{Suc\ j\})$ contains all jobs
have *NOTIN*: $\forall\, i \in \{1..m\}.\ i \neq x \longrightarrow Suc\ j \notin A\ i$ **using** *lbrules*(*1*) *x-def* **by** *blast*
then have $(\bigcup y \in \{1..m\}.\ ?A\ y) = (\bigcup y \in \{1..m\}.\ A\ y) - \{Suc\ j\}$
using *UNION-fun-upd x-def* **by** *auto*
also have ... $= \{1..j\}$ **unfolding** *lbrules*(*2*) **by** *auto*
finally have *2*: $(\bigcup y \in \{1..m\}.\ ?A\ y) = \{1..j\}$ **.**

— Rule 3: $A(x := A\ x - \{Suc\ j\})$ sums to $T(x := T\ x - t\ (Suc\ j))$
have $(\sum i \in A\ x - \{Suc\ j\}.\ t\ i) = (\sum i \in A\ x.\ t\ i) - t\ (Suc\ j)$
by (*simp add*: *sum-diff1-nat x-def*(*2*))
also have ... $= T\ x - t\ (Suc\ j)$ **using** *lbrules*(*3*) *x-def*(*1*) **by** *simp*
finally have $(\sum i \in ?A\ x.\ t\ i) = ?T\ x$ **by** *simp*
then have *3*: $\forall\, i \in \{1..m\}.\ (\sum j \in ?A\ i.\ t\ j) = ?T\ i$
using *lbrules*(*3*) *x-def*(*1*) **by** *simp*

— *makespan* is not larger
have *lb ?T ?A j* $\wedge$ *makespan ?T* $\leq$ *makespan T*
using *lbI*[*OF 1 2 3*] *makespan-mono*(*1*) **by** *force*
then show *?thesis* **by** *blast*
qed

If the processing time y does not contribute to the makespan, we can ignore it.

lemma *remove-small-job*:
assumes *makespan* $(T\ (x := T\ x + y)) \neq T\ x + y$
shows *makespan* $(T\ (x := T\ x + y)) =$ *makespan* T
proof −
let $?T = T\ (x := T\ x + y)$
have *NOT-X*: *makespan* $?T \neq ?T\ x$ **using** *assms*(*1*) **by** *simp*
then have $\exists\, i \in \{1..m\}.$ *makespan* $?T = ?T\ i \wedge i \neq x$
using *makespan-correct*(*2*) **by** *metis*
then obtain *i* **where** *i-def*: $i \in \{1..m\}$ *makespan* $?T = ?T\ i\ i \neq x$ **by** *blast*
then have $?T\ i = T\ i$ **using** *NOT-X* **by** *simp*
moreover from *this* **have** *makespan* $T = T\ i$
by (*metis i-def*(*1*,*2*) *antisym-conv le-add1 makespan-mono*(*2*) *makespan-correct*(*1*))
ultimately show *?thesis* **using** *i-def*(*2*) **by** *simp*
qed

lemma *greedy-makespan-no-jobs* [*simp*]:
makespan $(\lambda\text{-}.\ 0) = 0$
using *m-gt-0* **by** (*simp add*: *makespan-def'*)

lemma *min-avg*: $m * T\ (\textit{min-arg}\ T\ m) \le (\sum i \in \{1..m\}.\ T\ i)$
  (**is** ‹- $*$ $?T \le\ ?S$›)
proof −
  have $(\sum \text{-} \in \{1..m\}.\ ?T) \le\ ?S$
    using *sum-mono*[*of* ‹$\{1..m\}$› ‹λ-. $?T$› T]
      and *min-correct* **by** *blast*
  then show *?thesis* **by** *simp*
qed

definition $\textit{inv}_1 :: (\textit{nat} \Rightarrow \textit{nat}) \Rightarrow (\textit{nat} \Rightarrow \textit{nat set}) \Rightarrow \textit{nat} \Rightarrow \textit{bool}$ **where**
  $\textit{inv}_1\ T\ A\ j = (\textit{lb}\ T\ A\ j \land j \le n \land (\forall T'\ A'.\ \textit{lb}\ T'\ A'\ j \longrightarrow \textit{makespan}\ T \le 2 * \textit{makespan}\ T'))$

lemma $\textit{inv}_1 E$:
  assumes $\textit{inv}_1\ T\ A\ j$
  shows $\textit{lb}\ T\ A\ j$ $j \le n$
    $\textit{lb}\ T'\ A'\ j \Longrightarrow \textit{makespan}\ T \le 2 * \textit{makespan}\ T'$
  using *assms* **unfolding** $\textit{inv}_1$-*def* **by** *blast+*

lemma $\textit{inv}_1 I$:
  assumes $\textit{lb}\ T\ A\ j$ $j \le n$ $\forall T'\ A'.\ \textit{lb}\ T'\ A'\ j \longrightarrow \textit{makespan}\ T \le 2 * \textit{makespan}\ T'$
  shows $\textit{inv}_1\ T\ A\ j$ **using** *assms* **unfolding** $\textit{inv}_1$-*def* **by** *blast*

lemma $\textit{inv}_1$-*step*:
  assumes $\textit{inv}_1\ T\ A\ j$ $j < n$
  shows $\textit{inv}_1\ (T\ ((\textit{min-arg}\ T\ m) := T\ (\textit{min-arg}\ T\ m) + t\ (\textit{Suc}\ j)))$
      $(A\ ((\textit{min-arg}\ T\ m) := A\ (\textit{min-arg}\ T\ m) \cup \{\textit{Suc}\ j\}))\ (\textit{Suc}\ j)$
    (**is** ‹$\textit{inv}_1\ ?T\ ?A$ -›)
proof −
  note $\textit{invrules} = \textit{inv}_1 E[\textit{OF assms}(1)]$
  — Greedy is correct
  have *LB*: $\textit{lb}\ ?T\ ?A\ (\textit{Suc}\ j)$
    using *add-job*[*OF invrules*(*1*) *min-in-range*[*OF m-gt-0*]] **by** *blast*
  — Greedy maintains approximation factor
  have *MK*: $\forall T'\ A'.\ \textit{lb}\ T'\ A'\ (\textit{Suc}\ j) \longrightarrow \textit{makespan}\ ?T \le 2 * \textit{makespan}\ T'$
  proof *rule+*
    fix $T_1\ A_1$ **assume** $\textit{lb}\ T_1\ A_1\ (\textit{Suc}\ j)$
    from *smaller-optimum*[*OF this*]
    obtain $T_0\ A_0$ **where** $\textit{lb}\ T_0\ A_0\ j$ $\textit{makespan}\ T_0 \le \textit{makespan}\ T_1$ **by** *blast*
    then have *IH*: $\textit{makespan}\ T \le 2 * \textit{makespan}\ T_1$
      using *invrules*(*3*) **by** *force*
    show $\textit{makespan}\ ?T \le 2 * \textit{makespan}\ T_1$
    proof (*cases* ‹$\textit{makespan}\ ?T = T\ (\textit{min-arg}\ T\ m) + t\ (\textit{Suc}\ j)$›)
      case *True*
      have $m * T\ (\textit{min-arg}\ T\ m) \le (\sum i \in \{1..m\}.\ T\ i)$ **by** (*rule min-avg*)
      also have $\ldots = (\sum i \in \{1..j\}.\ t\ i)$ **by** (*rule lb-impl-job-sum*[*OF invrules*(*1*)])
      finally have $\textit{real}\ m * T\ (\textit{min-arg}\ T\ m) \le (\sum i \in \{1..j\}.\ t\ i)$
        by (*auto dest*: *of-nat-mono*)

with *m-gt-0* **have** T (*min-arg* T m) $\leq$ ($\sum i \in \{1..j\}.\ t\ i$) / m
by (*simp add*: *field-simps*)
then have T (*min-arg* T m) $\leq$ *makespan* T_1
using *job-dist-lower-bound-makespan*[*OF* ‹*lb* T_0 A_0 j›]
and ‹*makespan* $T_0 \leq$ *makespan* T_1› **by** *linarith*
moreover have t (*Suc* j) $\leq$ *makespan* T_1
using *job-lower-bound-makespan*[*OF* ‹*lb* T_1 A_1 (*Suc* j)›] **by** *simp*
ultimately show *?thesis* **unfolding** *True* **by** *simp*
next
case *False* **show** *?thesis* **using** *remove-small-job*[*OF False*] *IH* **by** *simp*
qed
qed
from *inv*$_1$*I*[*OF LB - MK*] **show** *?thesis* **using** *assms*(*2*) **by** *simp*
qed

lemma *simple-greedy-approximation*:
VARS T A i j
{*True*}
$T := (\lambda\text{-}.\ 0)$;
$A := (\lambda\text{-}.\ \{\})$;
$j := 0$;
WHILE $j < n$ *INV* {*inv*$_1$ T A j} *DO*
$i :=$ *min-arg* T m;
$j :=$ (*Suc* j);
$A := A\ (i := A(i) \cup \{j\})$;
$T := T\ (i := T(i) + t\ j)$
OD
$\{lb\ T\ A\ n \wedge (\forall T'\ A'.\ lb\ T'\ A'\ n \longrightarrow makespan\ T \leq 2 * makespan\ T')\}$
proof (*vcg*, *goal-cases*)
case (*1 T A i j*)
then show *?case* **by** (*simp add*: *lb-def inv*$_1$*-def*)
next
case (*2 T A i j*)
then show *?case* **using** *inv*$_1$*-step* **by** *simp*
next
case (*3 T A i j*)
then show *?case* **unfolding** *inv*$_1$*-def* **by** *force*
qed

definition *sorted* :: *nat* $\Rightarrow$ *bool* **where**
sorted $j = (\forall x \in \{1..j\}.\ \forall y \in \{1..x\}.\ t\ x \leq t\ y)$

lemma *sorted-smaller* [*simp*]: $[\![$ *sorted* j; $j \geq j'$ $]\!] \Longrightarrow$ *sorted* j'
unfolding *sorted-def* **by** *simp*

lemma *j-gt-m-pigeonhole*:
assumes *lb* T A j $j > m$
shows $\exists x \in \{1..j\}.\ \exists y \in \{1..j\}.\ \exists z \in \{1..m\}.\ x \neq y \wedge x \in A\ z \wedge y \in A\ z$
proof –

have $\forall x \in \{1..j\}.\ \exists y \in \{1..m\}.\ x \in A\ y$
using $lbE(2)[OF\ assms(1)]$ **by** *blast*
then have $\exists f.\ \forall x \in \{1..j\}.\ x \in A\ (f\ x) \wedge f\ x \in \{1..m\}$ **by** *metis*
then obtain f **where** $f\text{-}def$: $\forall x \in \{1..j\}.\ x \in A\ (f\ x) \wedge f\ x \in \{1..m\}$ **..**
then have $card\ (f\ `\ \{1..j\}) \leq card\ \{1..m\}$
by (*meson card-mono finite-atLeastAtMost image-subset-iff*)
also have $... < card\ \{1..j\}$ **using** $assms(2)$ **by** *simp*
finally have $card\ (f\ `\ \{1..j\}) < card\ \{1..j\}$ **.**
then have $\neg\ inj\text{-}on\ f\ \{1..j\}$ **using** *pigeonhole* **by** *blast*
then have $\exists x \in \{1..j\}.\ \exists y \in \{1..j\}.\ x \neq y \wedge f\ x = f\ y$
unfolding *inj-on-def* **by** *blast*
then show *?thesis* **using** *f-def* **by** *metis*
qed

If T and A are a correct load balancing for j jobs and m machines with $j > m$, and the jobs are sorted in descending order, then there exists a machine $x \in \{1..m\}$ whose load is at least twice as large as the processing time of job j.

lemma *sorted-job-lower-bound-machine*:
assumes $lb\ T\ A\ j\ j > m\ sorted\ j$
shows $\exists x \in \{1..m\}.\ 2 * t\ j \leq T\ x$
proof −
— Step 1: Obtaining the jobs
note $lbrules = lbE[OF\ assms(1)]$
obtain $j_1\ j_2\ x$ **where** $*$:
$j_1 \in \{1..j\}\ j_2 \in \{1..j\}\ x \in \{1..m\}\ j_1 \neq j_2\ j_1 \in A\ x\ j_2 \in A\ x$
using $j\text{-}gt\text{-}m\text{-}pigeonhole[OF\ assms(1,2)]$ **by** *blast*

— Step 2: Jobs contained in sum
have $finite\ (A\ x)$ **using** $assms(1)\ *(3)$ **by** *simp*
then have SUM: $(\sum i \in A\ x.\ t\ i) = t\ j_1 + t\ j_2 + (\sum i \in A\ x - \{j_1\} - \{j_2\}.\ t\ i)$
using $*(4-6)$ **by** (*simp add*: *sum.remove*)

— Step 3: Proof of lower bound
have $t\ j \leq t\ j_1\ \ t\ j \leq t\ j_2$
using $assms(3)\ *(1-2)$ **unfolding** *sorted-def* **by** *auto*
then have $2 * t\ j \leq t\ j_1 + t\ j_2$ **by** *simp*
also have $... \leq (\sum i \in A\ x.\ t\ i)$ **unfolding** SUM **by** *simp*
finally have $2 * t\ j \leq T\ x$ **using** $lbrules(3)\ *(3)$ **by** *simp*
then show *?thesis* **using** $*(3)$ **by** *blast*
qed

Reasoning analogous to *job-lower-bound-makespan*.

lemma *sorted-job-lower-bound-makespan*:
assumes $lb\ T\ A\ j\ j > m\ sorted\ j$
shows $2 * t\ j \leq makespan\ T$
proof −
obtain x **where** $x\text{-}def$: $x \in \{1..m\}\ 2 * t\ j \leq T\ x$

using *sorted-job-lower-bound-machine*[*OF assms*] **..**
with *makespan-correct*(*1*) **have** $T\ x \le makespan\ T$ **by** *blast*
with *x-def*(*2*) **show** *?thesis* **by** *simp*
qed

lemma *min-zero*:
assumes $x \in \{1..k\}$ $T\ x = 0$
shows $T\ (min\text{-}arg\ T\ k) = 0$
using *assms*(*1*)
proof (*induction k*)
case (*Suc k*)
show *?case* **proof** (*cases* ‹$x = Suc\ k$›)
case *True*
then show *?thesis* **using** *assms*(*2*) **by** (*simp add*: *Let-def*)
next
case *False*
with *Suc* **have** $T\ (min\text{-}arg\ T\ k) = 0$ **by** *simp*
then show *?thesis* **by** *simp*
qed
qed *simp*

lemma *min-zero-index*:
assumes $x \in \{1..k\}$ $T\ x = 0$
shows $min\text{-}arg\ T\ k \le x$
using *assms*(*1*)
proof (*induction k*)
case (*Suc k*)
show *?case* **proof** (*cases* ‹$x = Suc\ k$›)
case *True*
then show *?thesis* **using** *min-in-range*[*of Suc k*] **by** *simp*
next
case *False*
with *Suc.prems* **have** $x \in \{1..k\}$ **by** *simp*
from *min-zero*[*OF this*, *of T*] *assms*(*2*) *Suc.IH*[*OF this*]
show *?thesis* **by** *simp*
qed
qed *simp*

definition inv_2 :: $(nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat\ set) \Rightarrow nat \Rightarrow bool$ **where**
$inv_2\ T\ A\ j = (lb\ T\ A\ j \wedge j \le n$
$\wedge\ (\forall\ T'\ A'.\ lb\ T'\ A'\ j \longrightarrow makespan\ T \le 3\ /\ 2 * makespan\ T')$
$\wedge\ (\forall\ x > j.\ T\ x = 0)$
$\wedge\ (j \le m \longrightarrow makespan\ T = Max_0\ (t\ `\ \{1..j\})))$

lemma inv_2E:
assumes $inv_2\ T\ A\ j$
shows $lb\ T\ A\ j$ $j \le n$
$lb\ T'\ A'\ j \Longrightarrow makespan\ T \le 3\ /\ 2 * makespan\ T'$
$\forall\ x > j.\ T\ x = 0$ $j \le m \Longrightarrow makespan\ T = Max_0\ (t\ `\ \{1..j\})$

using *assms* **unfolding** inv_2*-def* **by** *blast+*

lemma $inv_2 I$:
assumes *lb T A j* $j \leq n$
$\forall T'\ A'.\ lb\ T'\ A'\ j \longrightarrow makespan\ T \leq 3\ /\ 2 * makespan\ T'$
$\forall x > j.\ T\ x = 0$
$j \leq m \Longrightarrow makespan\ T = Max_0\ (t\ `\ \{1..j\})$
shows $inv_2\ T\ A\ j$
unfolding inv_2*-def* **using** *assms* **by** *blast*

lemma inv_2*-step*:
assumes *sorted n* $inv_2\ T\ A\ j$ $j < n$
shows $inv_2\ (T\ (min\text{-}arg\ T\ m := T(min\text{-}arg\ T\ m) + t(Suc\ j)))$
$(A\ (min\text{-}arg\ T\ m := A(min\text{-}arg\ T\ m) \cup \{Suc\ j\}))\ (Suc\ j)$
(**is** ‹inv_2 *?T ?A -*›)
proof (*cases* ‹*Suc j* > *m*›)
case *True* **note** *invrules* = $inv_2 E$[*OF assms*(*2*)]
— Greedy is correct
have *LB*: *lb ?T ?A* (*Suc j*)
using *add-job*[*OF invrules*(*1*) *min-in-range*[*OF m-gt-0*]] **by** *blast*
— Greedy maintains approximation factor
have *MK*: $\forall T'\ A'.\ lb\ T'\ A'\ (Suc\ j) \longrightarrow makespan\ ?T \leq 3\ /\ 2 * makespan\ T'$
proof *rule+*
fix T_1 A_1 **assume** *lb* T_1 A_1 (*Suc j*)
from *smaller-optimum*[*OF this*]
obtain T_0 A_0 **where** *lb* T_0 A_0 *j makespan* $T_0 \leq$ *makespan* T_1 **by** *blast*
then have *IH*: *makespan* $T \leq 3\ /\ 2 *$ *makespan* T_1
using *invrules*(*3*) **by** *force*
show *makespan ?T* $\leq 3\ /\ 2 *$ *makespan* T_1
proof (*cases* ‹*makespan ?T* = *T* (*min-arg T m*) + *t* (*Suc j*)›)
case *True*
have $m * T\ (min\text{-}arg\ T\ m) \leq (\sum i \in \{1..m\}.\ T\ i)$ **by** (*rule min-avg*)
also have ... $= (\sum i \in \{1..j\}.\ t\ i)$ **by** (*rule lb-impl-job-sum*[*OF invrules*(*1*)])
finally have *real* $m * T\ (min\text{-}arg\ T\ m) \leq (\sum i \in \{1..j\}.\ t\ i)$
by (*auto dest*: *of-nat-mono*)
with *m-gt-0* **have** $T\ (min\text{-}arg\ T\ m) \leq (\sum i \in \{1..j\}.\ t\ i)\ /\ m$
by (*simp add*: *field-simps*)
then have *T* (*min-arg T m*) $\leq$ *makespan* T_1
using *job-dist-lower-bound-makespan*[*OF* ‹*lb* T_0 A_0 *j*›]
and ‹*makespan* $T_0 \leq$ *makespan* T_1› **by** *linarith*
moreover have $2 * t\ (Suc\ j) \leq$ *makespan* T_1
using *sorted-job-lower-bound-makespan*[*OF* ‹*lb* T_1 A_1 (*Suc j*)› ‹*Suc j* > *m*›]
and *assms*(*1*,*3*) **by** *simp*
ultimately show *?thesis* **unfolding** *True* **by** *simp*
next
case *False* **show** *?thesis* **using** *remove-small-job*[*OF False*] *IH* **by** *simp*
qed
qed
have $\forall x > Suc\ j.\ ?T\ x = 0$

using *invrules*(*4*) *min-in-range*[*OF m-gt-0*, *of T*] *True* **by** *simp*
with $inv_2 I$[*OF LB - MK*] **show** *?thesis* **using** *assms*(*3*) *True* **by** *simp*
next
case *False*
then have *IN-RANGE*: *Suc j* ∈ {*1..m*} **by** *simp*
note *invrules* = $inv_2 E$[*OF assms*(*2*)]
then have *T* (*Suc j*) = *0* **by** *blast*

— Greedy is correct
have *LB*: *lb ?T ?A* (*Suc j*)
using *add-job*[*OF invrules*(*1*) *min-in-range*[*OF m-gt-0*]] **by** *blast*

— Greedy is trivially optimal
from *IN-RANGE* ‹*T* (*Suc j*) = *0*› **have** *min-arg T m* ≤ *Suc j*
using *min-zero-index* **by** *blast*
with *invrules*(*4*) **have** *EMPTY*: ∀ *x* > *Suc j*. *?T x* = *0* **by** *simp*
from *IN-RANGE* ‹*T* (*Suc j*) = *0*› **have** *T* (*min-arg T m*) = *0*
using *min-zero* **by** *blast*
with *fun-upd-f-*Max_0[*OF min-in-range*[*OF m-gt-0*]] *invrules*(*5*) *False*
have *TRIV*: *makespan ?T* = Max_0 (*t* ‘ {*1..Suc j*}) **unfolding** *f-*Max_0*-equiv*[*symmetric*]
by *simp*
have *MK*: ∀ *T′ A′*. *lb T′ A′* (*Suc j*) ⟶ *makespan ?T* ≤ *3* / *2* ∗ *makespan T′*
by (*auto simp*: *TRIV*[*folded f-*Max_0*-equiv*]
dest!: *max-job-lower-bound-makespan*[*folded f-*Max_0*-equiv*])

from $inv_2 I$[*OF LB - MK EMPTY TRIV*] **show** *?thesis* **using** *assms*(*3*) **by** *simp*
qed

lemma *sorted-greedy-approximation*:
sorted n ⟹ *VARS T A i j*
{*True*}
T := (λ-. *0*);
A := (λ-. {});
j := *0*;
WHILE j < *n INV* {inv_2 *T A j*} *DO*
i := *min-arg T m*;
j := (*Suc j*);
A := *A* (*i* := *A*(*i*) ∪ {*j*});
T := *T* (*i* := *T*(*i*) + *t j*)
OD
{*lb T A n* ∧ (∀ *T′ A′*. *lb T′ A′ n* ⟶ *makespan T* ≤ *3* / *2* ∗ *makespan T′*)}
proof (*vcg*, *goal-cases*)
case (*1 T A i j*)
then show *?case* **by** (*simp add*: *lb-def* inv_2*-def*)
next
case (*2 T A i j*)
then show *?case* **using** inv_2*-step* **by** *simp*
next
case (*3 T A i j*)

then show *?case* **unfolding** inv_2*-def* **by** *force*
qed

end

end

5 Bin Packing

theory *Approx-BP-Hoare*
imports *Complex-Main HOL−Hoare.Hoare-Logic HOL−Library.Disjoint-Sets*
begin

The algorithm and proofs are based on the work by Berghammer and Reuter [2].

5.1 Formalization of a Correct Bin Packing

Definition of the unary operator $[\![\cdot]\!]$ from the article. B will only be wrapped into a set if it is non-empty.

definition *wrap* :: $'a$ *set* $\Rightarrow$ $'a$ *set set* **where**
wrap B = (*if* $B = \{\}$ *then* $\{\}$ *else* $\{B\}$)

lemma *wrap-card*:
card (*wrap* B) $\leq$ *1*
unfolding *wrap-def* **by** *auto*

If M and N are pairwise disjoint with V and not yet contained in V, then the union of M and N is also pairwise disjoint with V.

lemma *pairwise-disjnt-Un*:
assumes *pairwise disjnt* $(\{M\} \cup \{N\} \cup V)$ $M \notin V$ $N \notin V$
shows *pairwise disjnt* $(\{M \cup N\} \cup V)$
using *assms* **unfolding** *pairwise-def* **by** *auto*

A Bin Packing Problem is defined like in the article:

locale *BinPacking* =
fixes U :: $'a$ *set* — A finite, non-empty set of objects
and w :: $'a \Rightarrow$ *real* — A mapping from objects to their respective weights (positive real numbers)
and c :: *nat* — The maximum capacity of a bin (a natural number)
and S :: $'a$ *set* — The set of *small* objects (weight no larger than *1/2* of c)
and L :: $'a$ *set* — The set of *large* objects (weight larger than *1/2* of c)
assumes *weight*: $\forall u \in U.\ 0 < w(u) \wedge w(u) \leq c$
and *U-Finite*: *finite* U
and *U-NE*: $U \neq \{\}$
and *S-def*: $S = \{u \in U.\ w(u) \leq c\ /\ 2\}$
and *L-def*: $L = U - S$
begin

In the article, this is defined as w as well. However, to avoid ambiguity, we will abbreviate the weight of a bin as W.

abbreviation $W :: {}'a\ set \Rightarrow real$ **where**
$W\ B \equiv (\sum u \in B.\ w(u))$

P constitutes as a correct bin packing if P is a partition of U (as defined in *partition-on-def*) and the weights of the bins do not exceed their maximum capacity c.

definition $bp :: {}'a\ set\ set \Rightarrow bool$ **where**
$bp\ P \longleftrightarrow partition\text{-}on\ U\ P \wedge (\forall B \in P.\ W(B) \leq c)$

lemma *bpE*:
assumes $bp\ P$
shows $pairwise\ disjnt\ P\ \{\} \notin P\ \bigcup P = U\ \forall B \in P.\ W(B) \leq c$
using *assms* **unfolding** *bp-def partition-on-def* **by** *blast+*

lemma *bpI*:
assumes $pairwise\ disjnt\ P\ \{\} \notin P\ \bigcup P = U\ \forall B \in P.\ W(B) \leq c$
shows $bp\ P$
using *assms* **unfolding** *bp-def partition-on-def* **by** *blast*

Although we assume the S and L sets as given, manually obtaining them from U is trivial and can be achieved in linear time. Proposed by the article [2].

lemma *S-L-set-generation*:
$VARS\ S\ L\ W\ u$
$\{True\}$
$S := \{\};\ L := \{\};\ W := U;$
$WHILE\ W \neq \{\}$
$INV\ \{W \subseteq U \wedge S = \{v \in U - W.\ w(v) \leq c\ /\ 2\} \wedge L = \{v \in U - W.\ w(v) > c\ /\ 2\}\}\ DO$
$u := (SOME\ u.\ u \in W);$
$IF\ 2 * w(u) \leq c$
$THEN\ S := S \cup \{u\}$
$ELSE\ L := L \cup \{u\}\ FI;$
$W := W - \{u\}$
OD
$\{S = \{v \in U.\ w(v) \leq c\ /\ 2\} \wedge L = \{v \in U.\ w(v) > c\ /\ 2\}\}$
by *vcg* (*auto simp*: *some-in-eq*)

5.2 The Proposed Approximation Algorithm

5.2.1 Functional Correctness

According to the article, inv_1 holds if $P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}$ is a correct solution for the bin packing problem [2]. However, various assumptions made in the article seem to suggest that more information is demanded from this invariant and, indeed, mere correctness (as defined in *bp-def*) does not appear to suffice. To amend this, four additional conjuncts have been added to this invariant, whose necessity will be explained in the following proofs. It should be noted that there may be other (shorter) ways to amend this invariant. This approach, however, makes for rather straight-forward proofs, as these conjuncts can be utilized and proved in relatively few steps.

definition inv_1 :: $'a\ set\ set \Rightarrow\ 'a\ set\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set \Rightarrow bool$ **where**
$inv_1\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$ — A correct solution to the bin packing problem
$\wedge \bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$ — The partial solution does not contain objects that have not yet been assigned
$\wedge\ B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$ — B_1 is distinct from all the other bins
$\wedge\ B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$ — B_2 is distinct from all the other bins
$\wedge\ (P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$ — The first and second partial solutions are disjoint from each other.

lemma inv_1E:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
shows $bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
and $\bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$
and $B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$
and $B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$
and $(P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$
using *assms* **unfolding** inv_1-*def* **by** *auto*

lemma inv_1I:
assumes $bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
and $\bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$
and $B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$
and $B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$
and $(P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$
shows $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
using *assms* **unfolding** inv_1-*def* **by** *blast*

lemma *wrap-Un* [*simp*]: $wrap\ (M \cup \{x\}) = \{M \cup \{x\}\}$ **unfolding** *wrap-def* **by** *simp*
lemma *wrap-empty* [*simp*]: $wrap\ \{\} = \{\}$ **unfolding** *wrap-def* **by** *simp*
lemma *wrap-not-empty* [*simp*]: $M \neq \{\} \longleftrightarrow wrap\ M = \{M\}$ **unfolding** *wrap-def* **by** *simp*

If inv_1 holds for the current partial solution, and the weight of an object $u \in V$ added to B_1 does not exceed its capacity, then inv_1 also holds if B_1 and $\{u\}$ are replaced by $B_1 \cup \{u\}$.

lemma inv_1-*stepA*:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W(B_1) + w(u) \leq c$
shows $inv_1\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (V - \{u\})$

proof –
note *invrules* = inv_1E[*OF assms*(*1*)] **and** *bprules* = *bpE*[*OF invrules*(*1*)]

In the proof for *Theorem 3.2* of the article it is erroneously argued that if $P_1 \cup$ *wrap* $B_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V\}$ is a partition of U, then the same holds if B_1 is replaced by $B_1 \cup \{u\}$. This is, however, not necessarily the case if B_1 or $\{u\}$ are already contained in the partial solution. Suppose P_1 contains the non-empty bin B_1, then $P_1 \cup$ *wrap* B_1 would still be pairwise disjoint, provided P_1 was pairwise disjoint before, as the union simply ignores the duplicate B_1. Now, if the algorithm modifies B_1 by adding an element from V such that B_1 becomes some non-empty $B_1{}'$ with $B_1 \cap B_1{}' \neq \emptyset$ and $B_1{}' \notin P_1$, one can see that this property would no longer be preserved. To avoid such a situation, we will use the first additional conjunct in inv_1 to ensure that $\{u\}$ is not yet contained in the partial solution, and the second additional conjunct to ensure that B_1 is not yet contained in the partial solution.

have *NOTIN*: $\forall M \in P_1 \cup$ *wrap* $B_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}.\ u \notin M$
using *invrules*(*2*) *assms*(*2*) **by** *blast*
have $\{\{v\} \mid v.\ v \in V\} = \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$
using *assms*(*2*) **by** *blast*
then have *pairwise disjnt* ($P_1 \cup$ *wrap* $B_1 \cup P_2 \cup$ *wrap* $B_2 \cup (\{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$)
using *bprules*(*1*) *assms*(*2*) **by** *simp*
then have *pairwise disjnt* (*wrap* $B_1 \cup \{\{u\}\} \cup P_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$) **by** (*simp add*: *Un-commute*)
then have *assm*: *pairwise disjnt* (*wrap* $B_1 \cup \{\{u\}\} \cup (P_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$) **by** (*simp add*: *Un-assoc*)
have *pairwise disjnt* ($\{B_1 \cup \{u\}\} \cup (P_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$)
proof (*cases* ‹$B_1 = \{\}$›)
case *True* **with** *assm* **show** *?thesis* **by** *simp*
next
case *False*
with *assm* **have** *assm*: *pairwise disjnt* ($\{B_1\} \cup \{\{u\}\} \cup (P_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$) **by** *simp*
from *NOTIN* **have** $\{u\} \notin P_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$ **by** *blast*
from *pairwise-disjnt-Un*[*OF assm - this*] *invrules*(*2*,*3*) **show** *?thesis*
using *False* **by** *auto*
qed
then have *1*: *pairwise disjnt* ($P_1 \cup$ *wrap* $(B_1 \cup \{u\}) \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$)
unfolding *wrap-Un* **by** *simp*

— Rule 2: No empty sets
from *bprules*(*2*) **have** *2*: $\{\} \notin P_1 \cup$ *wrap* $(B_1 \cup \{u\}) \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$

unfolding *wrap-def* **by** *simp*

— Rule 3: Union preserved
from *bprules*(*3*) **have** $\bigcup$ $(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}) = U$
using *assms*(*2*) **by** *blast*
then have *3*: $\bigcup$ $(P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}) = U$
unfolding *wrap-def* **by** *force*

— Rule 4: Weights below capacity
have $0 < w\ u$ **using** *weight assms*(*2*) *bprules*(*3*) **by** *blast*
have *finite* B_1 **using** *bprules*(*3*) *U-Finite* **by** (*cases* ‹$B_1 = \{\}$›) *auto*
then have $W\ (B_1 \cup \{u\}) \leq W\ B_1 + w\ u$ **using** ‹$0 < w\ u$› **by** (*cases* ‹$u \in B_1$›) (*auto simp*: *insert-absorb*)
also have ... $\leq c$ **using** *assms*(*3*) **.**
finally have $W\ (B_1 \cup \{u\}) \leq c$ **.**
then have $\forall B \in wrap\ (B_1 \cup \{u\}).\ W\ B \leq c$ **unfolding** *wrap-Un* **by** *blast*
moreover have $\forall B{\in}P_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \leq c$
using *bprules*(*4*) **by** *blast*
ultimately have *4*: $\forall B{\in}P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \leq c$ **by** *blast*
from *bpI*[*OF 1 2 3 4*] **have** *1*: $bp\ (P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$ **.**

— Auxiliary information is preserved
have $u \in U$ **using** *assms*(*2*) *bprules*(*3*) **by** *blast*
then have *R*: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have *L*: $\bigcup$ $(P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2 \cup wrap\ B_2) = \bigcup$ $(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) \cup \{u\}$
unfolding *wrap-def* **using** *NOTIN* **by** *auto*
have *2*: $\bigcup$ $(P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2 \cup wrap\ B_2) = U - (V - \{u\})$
unfolding *L R invrules*(*2*) **..**
have *3*: $B_1 \cup \{u\} \notin P_1 \cup P_2 \cup wrap\ B_2$
using *NOTIN* **by** *auto*
have *4*: $B_2 \notin P_1 \cup wrap\ (B_1 \cup \{u\}) \cup P_2$
using *invrules*(*4*) *NOTIN* **unfolding** *wrap-def* **by** *fastforce*
have *5*: $(P_1 \cup wrap\ (B_1 \cup \{u\})) \cap (P_2 \cup wrap\ B_2) = \{\}$
using *invrules*(*5*) *NOTIN* **unfolding** *wrap-Un* **by** *auto*

from inv_1I[*OF 1 2 3 4 5*] **show** *?thesis* **.**
qed

If inv_1 holds for the current partial solution, and the weight of an object $u \in V$ added to B_2 does not exceed its capacity, then inv_1 also holds if B_2 and $\{u\}$ are replaced by $B_2 \cup \{u\}$.

lemma inv_1*-stepB*:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W\ B_2 + w\ u \leq c$
shows $inv_1\ (P_1 \cup wrap\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (V - \{u\})$

proof –
note *invrules* = $inv_1 E$[*OF assms*(*1*)] **and** *bprules* = *bpE*[*OF invrules*(*1*)]

The argumentation here is similar to the one in inv_1-*stepA* with B_1 replaced with B_2 and using the first and third additional conjuncts of inv_1 to amend the issue, instead of the first and second.

have *NOTIN*: $\forall M \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}.\ u \notin M$
using *invrules*(*2*) *assms*(*2*) **by** *blast*
have $\{\{v\} \mid v.\ v \in V\} = \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$
using *assms*(*2*) **by** *blast*
then have *pairwise disjnt* $(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$
using *bprules*(*1*) *assms*(*2*) **by** *simp*
then have *assm*: *pairwise disjnt* $(wrap\ B_2 \cup \{\{u\}\} \cup (P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$
by (*simp add*: *Un-assoc Un-commute*)
have *pairwise disjnt* $(\{B_2 \cup \{u\}\} \cup (P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$
proof (*cases* ‹$B_2 = \{\}$›)
case *True* **with** *assm* **show** *?thesis* **by** *simp*
next
case *False*
with *assm* **have** *assm*: *pairwise disjnt* $(\{B_2\} \cup \{\{u\}\} \cup (P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$ **by** *simp*
from *NOTIN* **have** $\{u\} \notin P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$ **by** *blast*
from *pairwise-disjnt-Un*[*OF assm - this*] *invrules*(*2*,*4*) **show** *?thesis*
using *False* **by** *auto*
qed
then have *1*: *pairwise disjnt* $(P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$
unfolding *wrap-Un* **by** *simp*

— Rule 2: No empty sets
from *bprules*(*2*) **have** *2*: $\{\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$
unfolding *wrap-def* **by** *simp*

— Rule 3: Union preserved
from *bprules*(*3*) **have** $\bigcup\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\}) = U$
using *assms*(*2*) **by** *blast*
then have *3*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\} \mid v.\ v \in V - \{u\}\}) = U$
unfolding *wrap-def* **by** *force*

— Rule 4: Weights below capacity
have $0 < w\ u$ **using** *weight assms*(*2*) *bprules*(*3*) **by** *blast*

have *finite* B_2 **using** *bprules*(*3*) *U-Finite* **by** (*cases* ‹$B_2 = \{\}$›) *auto*
then have $W\ (B_2 \cup \{u\}) \leq W\ B_2 + w\ u$ **using** ‹$0 < w\ u$› **by** (*cases* ‹$u \in B_2$›) (*auto simp*: *insert-absorb*)
also have ... $\leq c$ **using** *assms*(*3*) **.**
finally have $W\ (B_2 \cup \{u\}) \leq c$ **.**
then have $\forall B \in wrap\ (B_2 \cup \{u\}).\ W\ B \leq c$ **unfolding** *wrap-Un* **by** *blast*
moreover have $\forall B{\in}P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \leq c$
using *bprules*(*4*) **by** *blast*
ultimately have *4*: $\forall B{\in}P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \leq c$
by *auto*
from *bpI*[*OF 1 2 3 4*] **have** *1*: *bp* $(P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$ **.**

— Auxiliary information is preserved
have $u \in U$ **using** *assms*(*2*) *bprules*(*3*) **by** *blast*
then have *R*: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have *L*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\})) = \bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ B_2) \cup \{u\}$
unfolding *wrap-def* **using** *NOTIN* **by** *auto*
have *2*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\})) = U - (V - \{u\})$
unfolding *L R* **using** *invrules*(*2*) **by** *simp*
have *3*: $\{\} \notin P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ (B_2 \cup \{u\})$
using *bpE*(*2*)[*OF 1*] **by** *simp*
have *4*: $B_2 \cup \{u\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2$
using *NOTIN* **by** *auto*
have *5*: $(P_1 \cup wrap\ B_1 \cup wrap\ \{\}) \cap (P_2 \cup wrap\ (B_2 \cup \{u\})) = \{\}$
using *invrules*(*5*) *NOTIN* **unfolding** *wrap-empty wrap-Un* **by** *auto*

from inv_1I[*OF 1 2 3 4 5*] **show** *?thesis* **.**
qed

If inv_1 holds for the current partial solution, then inv_1 also holds if B_1 and B_2 are added to P_1 and P_2 respectively, B_1 is emptied and B_2 initialized with $u \in V$.

lemma inv_1-*stepC*:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V$
shows $inv_1\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$
proof −
note *invrules* = inv_1E[*OF assms*(*1*)]
— Rule 1-4: Correct Bin Packing
have $P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
$= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
by (*metis* (*no-types*, *lifting*) *Un-assoc Un-empty-right insert-not-empty wrap-empty wrap-not-empty*)
also have ... $= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}$
using *assms*(*2*) **by** *auto*

finally have EQ: $P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
$= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}$ **.**
from $invrules(1)$ **have** 1: $bp\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$
unfolding EQ **.**

— Auxiliary information is preserved
have $NOTIN$: $\forall M \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ u \notin M$
using $invrules(2)\ assms(2)$ **by** *blast*
have $u \in U$ **using** $assms(2)\ bpE(3)[OF\ invrules(1)]$ **by** *blast*
then have R: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have L: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}) = \bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2)) \cup \{u\}$
unfolding *wrap-def* **using** $NOTIN$ **by** *auto*
have 2: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}) = U - (V - \{u\})$
unfolding $L\ R$ **using** $invrules(2)$ **by** *auto*
have 3: $\{\} \notin P_1 \cup wrap\ B_1 \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}$
using $bpE(2)[OF\ 1]$ **by** *simp*
have 4: $\{u\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2)$
using $NOTIN$ **by** *auto*
have 5: $(P_1 \cup wrap\ B_1 \cup wrap\ \{\}) \cap (P_2 \cup wrap\ B_2 \cup wrap\ \{u\}) = \{\}$
using $invrules(5)\ NOTIN$ **unfolding** *wrap-def* **by** *force*

from $inv_1 I[OF\ 1\ 2\ 3\ 4\ 5]$ **show** *?thesis* **.**
qed

A simplified version of the bin packing algorithm proposed in the article. It serves as an introduction into the approach taken, and, while it does not provide the desired approximation factor, it does ensure that P is a correct solution of the bin packing problem.

lemma *simple-bp-correct*:
$VARS\ P\ P_1\ P_2\ B_1\ B_2\ V\ u$
$\{True\}$
$P_1 := \{\};\ P_2 := \{\};\ B_1 := \{\};\ B_2 := \{\};\ V := U;$
$WHILE\ V \cap S \neq \{\}\ INV\ \{inv_1\ P_1\ P_2\ B_1\ B_2\ V\}\ DO$
$u := (SOME\ u.\ u \in V);\ V := V - \{u\};$
$IF\ W(B_1) + w(u) \leq c$
$THEN\ B_1 := B_1 \cup \{u\}$
$ELSE\ IF\ W(B_2) + w(u) \leq c$
$THEN\ B_2 := B_2 \cup \{u\}$
$ELSE\ P_2 := P_2 \cup wrap\ B_2;\ B_2 := \{u\}\ FI;$
$P_1 := P_1 \cup wrap\ B_1;\ B_1 := \{\}\ FI$
$OD;$
$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |\ v.\ v \in V\}$
$\{bp\ P\}$
proof (*vcg*, *goal-cases*)

case $(1\ P\ P_1\ P_2\ B_1\ B_2\ V\ u)$
show *?case*
unfolding *bp-def partition-on-def pairwise-def wrap-def* inv_1*-def*
using *weight* **by** *auto*
next
case $(2\ P\ P_1\ P_2\ B_1\ B_2\ V\ u)$
then have *INV*: $inv_1\ P_1\ P_2\ B_1\ B_2\ V$ **..**
from *2* **have** $V \neq \{\}$ **by** *blast*
then have *IN*: $(SOME\ u.\ u \in V) \in V$ **by** (*simp add*: *some-in-eq*)
from inv_1*-stepA*[*OF INV IN*] inv_1*-stepB*[*OF INV IN*] inv_1*-stepC*[*OF INV IN*]
show *?case* **by** *blast*
next
case $(3\ P\ P_1\ P_2\ B_1\ B_2\ V\ u)$
then show *?case* **unfolding** inv_1*-def* **by** *blast*
qed

5.2.2 Lower Bounds for the Bin Packing Problem

lemma *bp-bins-finite* [*simp*]:
assumes *bp P*
shows $\forall B \in P.\ finite\ B$
using *bpE*(*3*)[*OF assms*] *U-Finite* **by** (*meson Sup-upper finite-subset*)

lemma *bp-sol-finite* [*simp*]:
assumes *bp P*
shows *finite P*
using *bpE*(*3*)[*OF assms*] *U-Finite* **by** (*simp add*: *finite-UnionD*)

If P is a solution of the bin packing problem, then no bin in P may contain more than one large object.

lemma *only-one-L-per-bin*:
assumes $bp\ P\ B \in P$
shows $\forall x \in B.\ \forall y \in B.\ x \neq y \longrightarrow x \notin L \vee y \notin L$
proof (*rule ccontr*, *simp*)
assume $\exists x{\in}B.\ \exists y{\in}B.\ x \neq y \wedge x \in L \wedge y \in L$
then obtain $x\ y$ **where** $*$: $x \in B\ y \in B\ x \neq y\ x \in L\ y \in L$ **by** *blast*
then have $c < w\ x + w\ y$ **using** *L-def S-def* **by** *force*
have *finite B* **using** *assms* **by** *simp*
have $y \in B - \{x\}$ **using** $*(2,3)$ **by** *blast*
have $W\ B = W\ (B - \{x\}) + w\ x$
using $*(1)$ ‹*finite B*› **by** (*simp add*: *sum.remove*)
also have ... $= W\ (B - \{x\} - \{y\}) + w\ x + w\ y$
using ‹$y \in B - \{x\}$› ‹*finite B*› **by** (*simp add*: *sum.remove*)
finally have $*$: $W\ B = W\ (B - \{x\} - \{y\}) + w\ x + w\ y$ **.**
have $\forall u \in B.\ 0 < w\ u$ **using** *bpE*(*3*)[*OF assms*(*1*)] *assms*(*2*) *weight* **by** *blast*
then have $0 \leq W\ (B - \{x\} - \{y\})$ **by** (*smt DiffD1 sum-nonneg*)
with $*$ **have** $c < W\ B$ **using** ‹$c < w\ x + w\ y$› **by** *simp*
then show *False* **using** *bpE*(*4*)[*OF assms*(*1*)] *assms*(*2*) **by** *fastforce*
qed

If P is a solution of the bin packing problem, then the amount of large objects is a lower bound for the amount of bins in P.

lemma *L-lower-bound-card*:
  assumes *bp P*
  shows *card* $L \leq$ *card* P
proof −
  have $\forall x \in L. \exists B \in P. x \in B$
    using *bpE*(*3*)[*OF assms*] *L-def* **by** *blast*
  then obtain f **where** *f-def*: $\forall u \in L. u \in f\ u \wedge f\ u \in P$ **by** *metis*
  then have *inj-on f L*
    unfolding *inj-on-def* **using** *only-one-L-per-bin*[*OF assms*] **by** *blast*
  then have *card-eq*: *card* $L =$ *card* $(f$ ‘ $L)$ **by** (*simp add*: *card-image*)
  have f ‘ $L \subseteq P$ **using** *f-def* **by** *blast*
  moreover have *finite P* **using** *assms* **by** *simp*
  ultimately have *card* $(f$ ‘ $L) \leq$ *card* P **by** (*simp add*: *card-mono*)
  then show *?thesis* **unfolding** *card-eq* **.**
qed

If P is a solution of the bin packing problem, then the amount of bins of a subset of P in which every bin contains a large object is a lower bound on the amount of large objects.

lemma *subset-bp-card*:
  assumes *bp P* $M \subseteq P$ $\forall B \in M. B \cap L \neq \{\}$
  shows *card* $M \leq$ *card* L
proof −
  have $\forall B \in M. \exists u \in L. u \in B$ **using** *assms*(*3*) **by** *fast*
  then have $\exists f. \forall B \in M. f\ B \in L \wedge f\ B \in B$ **by** *metis*
  then obtain f **where** *f-def*: $\forall B \in M. f\ B \in L \wedge f\ B \in B$ **..**
  have *inj-on f M*
  proof (*rule ccontr*)
    assume $\neg$ *inj-on f M*
    then have $\exists x \in M. \exists y \in M. x \neq y \wedge f\ x = f\ y$ **unfolding** *inj-on-def* **by** *blast*
    then obtain $x\ y$ **where** $*$: $x \in M$ $y \in M$ $x \neq y$ $f\ x = f\ y$ **by** *blast*
    then have $\exists u. u \in x \wedge u \in y$ **using** *f-def* **by** *metis*
    then have $x \cap y \neq \{\}$ **by** *blast*
    moreover have *pairwise disjnt M* **using** *pairwise-subset*[*OF bpE*(*1*)[*OF assms*(*1*)] *assms*(*2*)] **.**
    ultimately show *False* **using** $*$ **unfolding** *pairwise-def disjnt-def* **by** *simp*
  qed
  moreover have *finite L* **using** *L-def U-Finite* **by** *blast*
  moreover have f ‘ $M \subseteq L$ **using** *f-def* **by** *blast*
  ultimately show *?thesis* **using** *card-inj-on-le* **by** *blast*
qed

If P is a correct solution of the bin packing problem, inv_1 holds for the partial solution, and every bin in $P_1 \cup$ *wrap* B_1 contains a large object, then the amount of bins in $P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\}$ is a lower bound for the amount of bins in P.

lemma *L-bins-lower-bound-card*:

assumes *bp P inv*$_1$ P_1 P_2 B_1 B_2 V $\forall B \in P_1 \cup$ *wrap* B_1. $B \cap L \neq \{\}$
shows *card* $(P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\}) \leq$ *card P*
proof −
note *invrules* = *inv*$_1$*E*[*OF assms*(*2*)]
have $\forall B \in \{\{v\} \mid v.\ v \in V \cap L\}.\ B \cap L \neq \{\}$ **by** *blast*
with *assms*(*3*) **have**
$P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\} \subseteq P_1 \cup$ *wrap* $B_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\}$ $\mid v.\ v \in V\}$
$\forall B \in P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\}.\ B \cap L \neq \{\}$ **by** *blast+*
from *subset-bp-card*[*OF invrules*(*1*) *this*] **show** *?thesis*
using *L-lower-bound-card*[*OF assms*(*1*)] **by** *linarith*
qed

If P is a correct solution of the bin packing problem, then the sum of the weights of the objects is equal to the sum of the weights of the bins in P.

lemma *sum-Un-eq-sum-sum*:
assumes *bp P*
shows $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
proof −
have *FINITE*: $\forall B \in P.$ *finite B* **using** *assms* **by** *simp*
have *DISJNT*: $\forall A \in P.\ \forall B \in P.\ A \neq B \longrightarrow A \cap B = \{\}$
using *bpE*(*1*)[*OF assms*] **unfolding** *pairwise-def disjnt-def* **.**
have $(\sum u \in (\bigcup P).\ w\ u) = (\sum B \in P.\ W\ B)$
using *sum.Union-disjoint*[*OF FINITE DISJNT*] **by** *auto*
then show *?thesis* **unfolding** *bpE*(*3*)[*OF assms*] **.**
qed

If P is a correct solution of the bin packing problem, then the sum of the weights of the items is a lower bound of amount of bins in P multiplied by their maximum capacity.

lemma *sum-lower-bound-card*:
assumes *bp P*
shows $(\sum u \in U.\ w\ u) \leq c *$ *card P*
proof −
have $*$: $\forall B \in P.\ 0 < W\ B \wedge W\ B \leq c$
using *bpE*(*2*−*4*)[*OF assms*] *weight* **by** (*metis UnionI assms bp-bins-finite sum-pos*)
have $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
using *sum-Un-eq-sum-sum*[*OF assms*] **.**
also have ... $\leq (\sum B \in P.\ c)$ **using** *sum-mono* $*$ **by** *fastforce*
also have ... $= c *$ *card P* **by** *simp*
finally show *?thesis* **.**
qed

lemma *bp-NE*:
assumes *bp P*
shows $P \neq \{\}$
using *U-NE bpE*(*3*)[*OF assms*] **by** *blast*

lemma *sum-Un-ge*:
  fixes f :: - $\Rightarrow$ *real*
  assumes *finite M finite N* $\forall B \in M \cup N.\ 0 < f\ B$
  shows *sum f M* $\leq$ *sum f* $(M \cup N)$
proof $-$
  have $0 \leq$ *sum f N* $-$ *sum f* $(M \cap N)$
    using *assms* **by** (*smt DiffD1 inf.cobounded2 UnCI sum-mono2*)
  then have *sum f M* $\leq$ *sum f M* $+$ *sum f N* $-$ *sum f* $(M \cap N)$
    by *simp*
  also have ... $=$ *sum f* $(M \cup N)$
    using *sum-Un*[*OF assms*(*1*,*2*), *symmetric*] **.**
  finally show *?thesis* **.**
qed

If *bij-exists* holds, one can obtain a function which is bijective between the bins in P and the objects in V such that an object returned by the function would cause the bin to exceed its capacity.

definition *bij-exists* :: *'a set set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool* **where**
  bij-exists $P\ V = (\exists f.$ *bij-betw* $f\ P\ V \wedge (\forall B \in P.\ W\ B + w\ (f\ B) > c))$

If P is a functionally correct solution of the bin packing problem, inv_1 holds for the partial solution, and such a bijective function exists between the bins in P_1 and the objects in $P_2 \cup$ *wrap* B_2, the following strict lower bound can be shown:

lemma P_1*-lower-bound-card*:
  assumes *bp P* inv_1 P_1 P_2 B_1 B_2 V *bij-exists* P_1 $(\bigcup (P_2 \cup$ *wrap* $B_2))$
  shows *card* $P_1 + 1 \leq$ *card P*
proof (*cases* ‹$P_1 = \{\}$›)
  case *True*
  have *finite P* **using** *assms*(*1*) **by** *simp*
  then have $1 \leq$ *card P* **using** *bp-NE*[*OF assms*(*1*)]
   by (*metis Nat.add-0-right Suc-diff-1 Suc-le-mono card-gt-0-iff le0 mult-Suc-right nat-mult-1*)
  then show *?thesis* **unfolding** *True* **by** *simp*
next
  note *invrules* $=$ inv_1E[*OF assms*(*2*)]
  case *False*
  obtain *f* **where** *f-def*: *bij-betw f* P_1 $(\bigcup (P_2 \cup$ *wrap* $B_2))$ $\forall B \in P_1.\ W\ B + w\ (f\ B) > c$
    using *assms*(*3*) **unfolding** *bij-exists-def* **by** *blast*
  have *FINITE*: *finite* P_1 *finite* $(P_2 \cup$ *wrap* $B_2)$ *finite* $(P_1 \cup P_2 \cup$ *wrap* $B_2)$ *finite* (*wrap* $B_1 \cup \{\{v\}\ |v.\ v \in V\}$)
    using inv_1E(*1*)[*OF assms*(*2*)] *bp-sol-finite* **by** *blast+*

  have *F*: $\forall B \in P_2 \cup$ *wrap* B_2. *finite B* **using** *invrules*(*1*) **by** *simp*
  have *D*: $\forall A \in P_2 \cup$ *wrap* $B_2.\ \forall B \in P_2 \cup$ *wrap* $B_2.\ A \neq B \longrightarrow A \cap B = \{\}$
    using *bpE*(*1*)[*OF invrules*(*1*)] **unfolding** *pairwise-def disjnt-def* **by** *auto*
  have *sum-eq*: $W\ (\bigcup\ (P_2 \cup$ *wrap* $B_2)) = (\sum B \in P_2 \cup$ *wrap* $B_2.\ W\ B)$

using *sum.Union-disjoint*[*OF F D*] **by** *auto*

have $\forall B \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\}.\ 0 < W\ B$
using *bpE*(*2*,*3*)[*OF invrules*(*1*)] *weight* **by** (*metis* (*no-types*, *lifting*) *UnionI bp-bins-finite invrules*(*1*) *sum-pos*)
then have $(\sum B \in P_1 \cup P_2 \cup wrap\ B_2.\ W\ B) \le (\sum B \in P_1 \cup P_2 \cup wrap\ B_2 \cup (wrap\ B_1 \cup \{\{v\} \mid v.\ v \in V\}).\ W\ B)$
using *sum-Un-ge*[*OF FINITE*(*3*,*4*), *of W*] **by** *blast*
also have ... $= (\sum B \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\}.\ W\ B)$ **by** (*smt Un-assoc Un-commute*)
also have ... $= W\ U$ **using** *sum-Un-eq-sum-sum*[*OF invrules*(*1*), *symmetric*] **.**
finally have $*$: $(\sum B \in P_1 \cup P_2 \cup wrap\ B_2.\ W\ B) \le W\ U$ **.**

— This follows from the fourth and final additional conjunct of inv_1 and is necessary to combine the sums of the bins of the two partial solutions. This does not inherently follow from the union being a correct solution, as this need not be the case if P_1 and $P_2 \cup wrap\ B_2$ happened to be equal.
have *DISJNT*: $P_1 \cap (P_2 \cup wrap\ B_2) = \{\}$ **using** *invrules*(*5*) **by** *blast*

— This part of the proof is based on the proof on page 72 of the article [2].
have $c * card\ P_1 = (\sum B \in P_1.\ c)$ **by** *simp*
also have ... $< (\sum B \in P_1.\ W\ B + w\ (f\ B))$
using *f-def*(*2*) *sum-strict-mono*[*OF FINITE*(*1*) *False*] **by** *fastforce*
also have ... $= (\sum B \in P_1.\ W\ B) + (\sum B \in P_1.\ w\ (f\ B))$
by (*simp add*: *Groups-Big.comm-monoid-add-class.sum.distrib*)
also have ... $= (\sum B \in P_1.\ W\ B) + W\ (\bigcup\ (P_2 \cup wrap\ B_2))$ **unfolding** *sum.reindex-bij-betw*[*OF f-def*(*1*), *of w*] **..**
also have ... $= (\sum B \in P_1.\ W\ B) + (\sum B \in P_2 \cup wrap\ B_2.\ W\ B)$ **unfolding** *sum-eq* **..**
also have ... $= (\sum B \in P_1 \cup P_2 \cup wrap\ B_2.\ W\ B)$ **using** *sum.union-disjoint*[*OF FINITE*(*1*,*2*) *DISJNT*, *of W*] **by** (*simp add*: *Un-assoc*)
also have ... $\le (\sum u \in U.\ w\ u)$ **using** $*$ **.**
also have ... $\le c * card\ P$ **using** *sum-lower-bound-card*[*OF assms*(*1*)] **.**
finally show *?thesis* **by** (*meson discrete nat-mult-less-cancel-disj of-nat-less-imp-less*)
qed

As $card\ (wrap\ ?B) \le 1$ holds, it follows that the amount of bins in $P_1 \cup wrap\ B_1$ are a lower bound for the amount of bins in P.

lemma P_1-B_1-*lower-bound-card*:
assumes *bp P* $inv_1\ P_1\ P_2\ B_1\ B_2\ V$ *bij-exists* $P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
shows $card\ (P_1 \cup wrap\ B_1) \le card\ P$
proof −
have $card\ (P_1 \cup wrap\ B_1) \le card\ P_1 + card\ (wrap\ B_1)$
using *card-Un-le* **by** *blast*
also have ... $\le card\ P_1 + 1$ **using** *wrap-card* **by** *simp*
also have ... $\le card\ P$ **using** P_1-*lower-bound-card*[*OF assms*] **.**
finally show *?thesis* **.**
qed

If inv_1 holds, there are at most half as many bins in P_2 as there are objects in P_2, and we can again obtain a bijective function between the bins in P_1 and the objects of the second partial solution, then the amount of bins in the second partial solution are a strict lower bound for half the bins of the first partial solution.

lemma P_2-B_2-*lower-bound*-P_1:
  assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V$ *2* $*$ *card* $P_2 \le$ *card* $(\bigcup P_2)$ *bij-exists* P_1 $(\bigcup (P_2 \cup$ *wrap* $B_2))$
  shows *2* $*$ *card* $(P_2 \cup$ *wrap* $B_2) \le$ *card* P_1 + *1*
proof −
  note *invrules* = inv_1E[*OF assms*(*1*)] **and** *bprules* = *bpE*[*OF invrules*(*1*)]

  have *pairwise disjnt* $(P_2 \cup$ *wrap* $B_2)$
    using *bprules*(*1*) *pairwise-subset* **by** *blast*
  moreover have $B_2 \notin P_2$ **using** *invrules*(*4*) **by** *simp*
  ultimately have *DISJNT*: $\bigcup P_2 \cap B_2 = \{\}$
    by (*auto*, *metis* (*no-types*, *opaque-lifting*) *sup-bot.right-neutral Un-insert-right disjnt-iff mk-disjoint-insert pairwise-insert wrap-Un*)

  have *finite* $(\bigcup P_2)$ **using** *U-Finite bprules*(*3*) **by** *auto*
  have *finite* B_2 **using** *bp-bins-finite*[*OF invrules*(*1*)] *wrap-not-empty* **by** *blast*
  have *finite* P_2 *finite* (*wrap* B_2) **using** *bp-sol-finite*[*OF invrules*(*1*)] **by** *blast+*
  have *DISJNT2*: $P_2 \cap$ *wrap* $B_2 = \{\}$ **unfolding** *wrap-def* **using** ‹$B_2 \notin P_2$› **by** *auto*
  have *card* (*wrap* B_2) $\le$ *card* B_2
  proof (*cases* ‹$B_2 = \{\}$›)
    case *False*
    then have *1* $\le$ *card* B_2 **by** (*simp add*: *leI* ‹*finite* B_2›)
    then show *?thesis* **using** *wrap-card*[*of* B_2] **by** *linarith*
  qed *simp*

  — This part of the proof is based on the proof on page 73 of the article [2].
  from *assms*(*2*) **have** *2* $*$ *card* P_2 + *2* $*$ *card* (*wrap* B_2) $\le$ *card* $(\bigcup P_2)$ + *card* (*wrap* B_2) + *1*
    using *wrap-card*[*of* B_2] **by** *linarith*
  then have *2* $*$ (*card* P_2 + *card* (*wrap* B_2)) $\le$ *card* $(\bigcup P_2)$ + *card* B_2 + *1*
    using ‹*card* (*wrap* B_2) $\le$ *card* B_2› **by** *simp*
  then have *2* $*$ (*card* $(P_2 \cup$ *wrap* $B_2)) \le$ *card* $(\bigcup P_2 \cup B_2)$ + *1*
    using *card-Un-disjoint*[*OF* ‹*finite* $(\bigcup P_2)$› ‹*finite* B_2› *DISJNT*]
      and *card-Un-disjoint*[*OF* ‹*finite* P_2› ‹*finite* (*wrap* B_2)› *DISJNT2*] **by** *argo*
  then have *2* $*$ (*card* $(P_2 \cup$ *wrap* $B_2)) \le$ *card* $(\bigcup (P_2 \cup$ *wrap* $B_2))$ + *1*
    by (*cases* ‹$B_2 = \{\}$›) (*auto simp*: *Un-commute*)
  then show *2* $*$ (*card* $(P_2 \cup$ *wrap* $B_2)) \le$ *card* P_1 + *1*
    using *assms*(*3*) *bij-betw-same-card* **unfolding** *bij-exists-def* **by** *metis*
qed

5.2.3 Proving the Approximation Factor

We define inv_2 as it is defined in the article. These conjuncts allow us to prove the desired approximation factor.

definition inv_2 :: $'a\ set\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
  $inv_2\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow inv_1\ P_1\ P_2\ B_1\ B_2\ V$ — inv_1 holds for the partial solution
  $\wedge\ (V \cap L \neq \{\} \longrightarrow (\forall B \in P_1 \cup wrap\ B_1.\ B \cap L \neq \{\}))$ — If there are still large objects left, then every bin of the first partial solution must contain a large object
  $\wedge\ bij\text{-}exists\ P_1\ (\bigcup(P_2 \cup wrap\ B_2))$ — There exists a bijective function between the bins of the first partial solution and the objects of the second one
  $\wedge\ (2 * card\ P_2 \leq card\ (\bigcup P_2))$ — There are at most twice as many bins in P_2 as there are objects in P_2

lemma inv_2E:
  assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ V$
  shows $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
    and $V \cap L \neq \{\} \Longrightarrow \forall B \in P_1 \cup wrap\ B_1.\ B \cap L \neq \{\}$
    and $bij\text{-}exists\ P_1\ (\bigcup(P_2 \cup wrap\ B_2))$
    and $2 * card\ P_2 \leq card\ (\bigcup P_2)$
  using *assms* **unfolding** inv_2*-def* **by** *blast+*

lemma inv_2I:
  assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
    and $V \cap L \neq \{\} \Longrightarrow \forall B \in P_1 \cup wrap\ B_1.\ B \cap L \neq \{\}$
    and $bij\text{-}exists\ P_1\ (\bigcup(P_2 \cup wrap\ B_2))$
    and $2 * card\ P_2 \leq card\ (\bigcup P_2)$
  shows $inv_2\ P_1\ P_2\ B_1\ B_2\ V$
  using *assms* **unfolding** inv_2*-def* **by** *blast*

If P is a correct solution of the bin packing problem, inv_2 holds for the partial solution, and there are no more small objects left to be distributed, then the amount of bins of the partial solution is no larger than $3\ /\ 2$ of the amount of bins in P. This proof strongly follows the proof in *Theorem 4.1* of the article [2].

lemma *bin-packing-lower-bound-card*:
  assumes $V \cap S = \{\}\ inv_2\ P_1\ P_2\ B_1\ B_2\ V\ bp\ P$
  shows $card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}) \leq 3\ /\ 2 * card\ P$
proof $(cases\ \langle V = \{\}\rangle)$
  note $invrules = inv_2E[OF\ assms(2)]$
  case *True*
  then have $card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
      $= card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2)$ **by** *simp*
  also have ... $\leq card\ (P_1 \cup wrap\ B_1) + card\ (P_2 \cup wrap\ B_2)$
    using $card\text{-}Un\text{-}le[of\ \langle P_1 \cup wrap\ B_1\rangle]$ **by** (*simp add*: *Un-assoc*)
  also have ... $\leq card\ P + card\ (P_2 \cup wrap\ B_2)$

using P_1-B_1-*lower-bound-card*[*OF assms*(*3*) *invrules*(*1*,*3*)] **by** *simp*
also have ... $\leq$ *card P* + *card P* / *2*
using P_2-B_2-*lower-bound*-P_1[*OF invrules*(*1*,*4*,*3*)]
and P_1-*lower-bound-card*[*OF assms*(*3*) *invrules*(*1*,*3*)] **by** *linarith*
finally show *?thesis* **by** *linarith*
next
note *invrules* = inv_2E[*OF assms*(*2*)]
case *False*
have $U = S \cup L$ **using** *S-def L-def* **by** *blast*
then have $*$: $V = V \cap L$
using *bpE*(*3*)[*OF* inv_1E(*1*)[*OF invrules*(*1*)]]
and *assms*(*1*) **by** *blast*
with *False* **have** *NE*: $V \cap L \neq \{\}$ **by** *simp*
have *card* ($P_1 \cup$ *wrap* $B_1 \cup P_2 \cup$ *wrap* $B_2 \cup \{\{v\} \mid v.\ v \in V\}$)
= *card* ($P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\} \cup P_2 \cup$ *wrap* B_2)
using $*$ **by** (*simp add*: *Un-commute Un-assoc*)
also have ... $\leq$ *card* ($P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\}$) + *card* ($P_2 \cup$ *wrap* B_2)
using *card-Un-le*[*of* ‹$P_1 \cup$ *wrap* $B_1 \cup \{\{v\} \mid v.\ v \in V \cap L\}$›] **by** (*simp add*: *Un-assoc*)
also have ... $\leq$ *card P* + *card* ($P_2 \cup$ *wrap* B_2)
using *L-bins-lower-bound-card*[*OF assms*(*3*) *invrules*(*1*) *invrules*(*2*)[*OF NE*]] **by** *linarith*
also have ... $\leq$ *card P* + *card P* / *2*
using P_2-B_2-*lower-bound*-P_1[*OF invrules*(*1*,*4*,*3*)]
and P_1-*lower-bound-card*[*OF assms*(*3*) *invrules*(*1*,*3*)] **by** *linarith*
finally show *?thesis* **by** *linarith*
qed

We define inv_3 as it is defined in the article. This final conjunct allows us to prove that the invariant will be maintained by the algorithm.

definition inv_3 :: *'a set set* $\Rightarrow$ *'a set set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool* **where**
$inv_3\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow inv_2\ P_1\ P_2\ B_1\ B_2\ V \wedge B_2 \subseteq S$

lemma inv_3E:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V$
shows $inv_2\ P_1\ P_2\ B_1\ B_2\ V$ **and** $B_2 \subseteq S$
using *assms* **unfolding** inv_3-*def* **by** *blast*+

lemma inv_3I:
assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ V$ **and** $B_2 \subseteq S$
shows $inv_3\ P_1\ P_2\ B_1\ B_2\ V$
using *assms* **unfolding** inv_3-*def* **by** *blast*

lemma *loop-init*:
$inv_3\ \{\}\ \{\}\ \{\}\ \{\}\ U$
proof −
have $*$: $inv_1\ \{\}\ \{\}\ \{\}\ \{\}\ U$
unfolding *bp-def partition-on-def pairwise-def wrap-def* inv_1-*def*

using *weight* **by** *auto*
have *bij-exists* {} ($\bigcup$ ({} $\cup$ *wrap* {}))
using *bij-betwI'* **unfolding** *bij-exists-def* **by** *fastforce*
from inv_2I[*OF* ∗ - *this*] **have** inv_2 {} {} {} {} U **by** *auto*
from inv_3I[*OF this*] **show** *?thesis* **by** *blast*
qed

If B_1 is empty and there are no large objects left, then inv_3 will be maintained if B_1 is initialized with $u \in V \cap S$.

lemma *loop-stepA*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V\ B_1 = \{\}\ V \cap L = \{\}\ u \in V \cap S$
shows $inv_3\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$
proof –
note *invrules* = inv_2E[*OF* $inv_3E(1)$[*OF assms*(1)]]
have *WEIGHT*: $W\ B_1 + w\ u \leq c$ **using** *S-def assms*(2,4) **by** *simp*
from *assms*(4) **have** $u \in V$ **by** *blast*
from inv_1-*stepA*[*OF invrules*(1) *this WEIGHT*] *assms*(2) **have** 1: $inv_1\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$ **by** *simp*
have 2: $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ \{u\}.\ B \cap L \neq \{\}$ **using** *assms*(3) **by** *blast*
from inv_2I[*OF 1 2*] *invrules* **have** $inv_2\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$ **by** *blast*
from inv_3I[*OF this*] **show** *?thesis* **using** $inv_3E(2)$[*OF assms*(1)] **.**
qed

If B_1 is empty and there are large objects left, then inv_3 will be maintained if B_1 is initialized with $u \in V \cap L$.

lemma *loop-stepB*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V\ B_1 = \{\}\ u \in V \cap L$
shows $inv_3\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$
proof –
note *invrules* = inv_2E[*OF* $inv_3E(1)$[*OF assms*(1)]]
have *WEIGHT*: $W\ B_1 + w\ u \leq c$ **using** *L-def weight assms*(2,3) **by** *simp*
from *assms*(3) **have** $u \in V$ **by** *blast*
from inv_1-*stepA*[*OF invrules*(1) *this WEIGHT*] *assms*(2) **have** 1: $inv_1\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$ **by** *simp*
have $\forall B{\in}P_1.\ B \cap L \neq \{\}$ **using** *assms*(3) *invrules*(2) **by** *blast*
then have 2: $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ \{u\}.\ B \cap L \neq \{\}$
using *assms*(3) **by** (*metis Int-iff UnE empty-iff insertE singletonI wrap-not-empty*)
from inv_2I[*OF 1 2*] *invrules* **have** $inv_2\ P_1\ P_2\ \{u\}\ B_2\ (V - \{u\})$ **by** *blast*
from inv_3I[*OF this*] **show** *?thesis* **using** $inv_3E(2)$[*OF assms*(1)] **.**
qed

If B_1 is not empty and $u \in V \cap S$ does not exceed its maximum capacity, then inv_3 will be maintained if B_1 and $\{u\}$ are replaced with $B_1 \cup \{u\}$.

lemma *loop-stepC*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V\ B_1 \neq \{\}\ u \in V \cap S\ W\ B_1 + w(u) \leq c$
shows $inv_3\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (V - \{u\})$
proof –
note *invrules* = inv_2E[*OF* $inv_3E(1)$[*OF assms*(1)]]

from *assms*(*3*) **have** $u \in V$ **by** *blast*
from *inv*$_1$*-stepA*[*OF invrules*(*1*) *this assms*(*4*)] **have** *1*: inv_1 P_1 P_2 $(B_1 \cup \{u\})$ B_2 $(V - \{u\})$ **.**
have $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ B_1.\ B \cap L \neq \{\}$ **using** *invrules*(*2*) **by** *blast*
then have *2*: $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ (B_1 \cup \{u\}).\ B \cap L \neq \{\}$
by (*metis Int-commute Un-empty-right Un-insert-right assms*(*2*) *disjoint-insert*(*2*) *insert-iff wrap-not-empty*)
from *inv*$_2$*I*[*OF 1 2*] *invrules* **have** inv_2 P_1 P_2 $(B_1 \cup \{u\})$ B_2 $(V - \{u\})$ **by** *blast*
from *inv*$_3$*I*[*OF this*] **show** *?thesis* **using** *inv*$_3$*E*(*2*)[*OF assms*(*1*)] **.**
qed

If B_1 is not empty and $u \in V \cap S$ does exceed its maximum capacity but not the capacity of B_2, then inv_3 will be maintained if B_1 is added to P_1 and emptied, and B_2 and $\{u\}$ are replaced with $B_2 \cup \{u\}$.

lemma *loop-stepD*:
assumes inv_3 P_1 P_2 B_1 B_2 V $B_1 \neq \{\}$ $u \in V \cap S$ $W\ B_1 + w(u) > c$ $W\ B_2 + w(u) \leq c$
shows inv_3 $(P_1 \cup wrap\ B_1)$ P_2 $\{\}$ $(B_2 \cup \{u\})$ $(V - \{u\})$
proof −
note *invrules* = *inv*$_2$*E*[*OF inv*$_3$*E*(*1*)[*OF assms*(*1*)]]
from *assms*(*3*) **have** $u \in V$ **by** *blast*
from *inv*$_1$*-stepB*[*OF invrules*(*1*) *this assms*(*5*)] **have** *1*: inv_1 $(P_1 \cup wrap\ B_1)$ P_2 $\{\}$ $(B_2 \cup \{u\})$ $(V - \{u\})$ **.**

have *2*: $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ B_1 \cup wrap\ \{\}.\ B \cap L \neq \{\}$
using *invrules*(*2*) **unfolding** *wrap-empty* **by** *blast*

from *invrules*(*3*) **obtain** *f* **where** *f-def*: *bij-betw* f P_1 $(\bigcup\ (P_2 \cup wrap\ B_2))$ $\forall B{\in}P_1.\ c < W\ B + w\ (f\ B)$ **unfolding** *bij-exists-def* **by** *blast*
have $B_1 \notin P_1$ **using** *inv*$_1$*E*(*3*)[*OF invrules*(*1*)] **by** *blast*
have $u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$ **using** *inv*$_1$*E*(*2*)[*OF invrules*(*1*)] *assms*(*3*) **by** *blast*
then have $(\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\}))) = (\bigcup\ (P_2 \cup wrap\ B_2 \cup \{\{u\}\}))$
by (*metis Sup-empty Un-assoc Union-Un-distrib ccpo-Sup-singleton wrap-empty wrap-not-empty*)
also have ... $= (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **by** *simp*
finally have *UN*: $(\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\}))) = (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **.**
have $wrap\ B_1 = \{B_1\}$ **using** *wrap-not-empty*[*of* B_1] *assms*(*2*) **by** *simp*
let $?f = f\ (B_1 := u)$
have *BIJ*: *bij-betw* $?f$ $(P_1 \cup wrap\ B_1)$ $(\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\})))$
unfolding *wrap-empty* ‹$wrap\ B_1 = \{B_1\}$› *UN* **using** *f-def*(*1*) ‹$B_1 \notin P_1$› ‹$u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$›
by (*metis* (*no-types*, *lifting*) *bij-betw-cong fun-upd-other fun-upd-same notIn-Un-bij-betw3*)
have $c < W\ B_1 + w\ (?f\ B_1)$ **using** *assms*(*4*) **by** *simp*
then have $(\forall B{\in}P_1 \cup wrap\ B_1.\ c < W\ B + w\ (?f\ B))$
unfolding ‹$wrap\ B_1 = \{B_1\}$› **using** *f-def*(*2*) **by** *simp*

with *BIJ* **have** *bij-betw ?f* $(P_1 \cup$ *wrap* $B_1)$ $(\bigcup$ $(P_2 \cup$ *wrap* $(B_2 \cup \{u\})))$
$\wedge$ $(\forall B{\in}P_1 \cup$ *wrap* $B_1.\ c < W\ B + w\ (?f\ B))$ **by** *blast*
then have *3*: *bij-exists* $(P_1 \cup$ *wrap* $B_1)$ $(\bigcup$ $(P_2 \cup$ *wrap* $(B_2 \cup \{u\})))$
unfolding *bij-exists-def* **by** *blast*
from inv_2I[*OF 1 2 3*] **have** inv_2 $(P_1 \cup$ *wrap* $B_1)$ P_2 $\{\}$ $(B_2 \cup \{u\})$ $(V - \{u\})$
using *invrules*(*4*) **by** *blast*

from inv_3I[*OF this*] **show** *?thesis* **using** inv_3E(*2*)[*OF assms*(*1*)] *assms*(*3*) **by** *blast*
qed

If the maximum capacity of B_2 is exceeded by $u \in V \cap S$, then B_2 must contain at least two objects.

lemma B_2*-at-least-two-objects*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V$ $u \in V \cap S$ $W\ B_2 + w(u) > c$
shows $2 \leq card\ B_2$
proof (*rule ccontr*, *simp add*: *not-le*)
have *FINITE*: *finite* B_2 **using** inv_1E(*1*)[*OF* inv_2E(*1*)[*OF* inv_3E(*1*)[*OF assms*(*1*)]]]
by (*metis* (*no-types*, *lifting*) *Finite-Set.finite.simps U-Finite Union-Un-distrib bpE*(*3*) *ccpo-Sup-singleton finite-Un wrap-not-empty*)
assume $card\ B_2 < 2$
then consider (*0*) $card\ B_2 = 0$ | (*1*) $card\ B_2 = 1$ **by** *linarith*
then show *False* **proof** *cases*
case *0* **then have** $B_2 = \{\}$ **using** *FINITE* **by** *simp*
then show *?thesis* **using** *assms*(*2*,*3*) *S-def* **by** *simp*
next
case *1* **then obtain** v **where** $B_2 = \{v\}$
using *card-1-singletonE* **by** *auto*
with inv_3E(*2*)[*OF assms*(*1*)] **have** $2 * w\ v \leq c$ **using** *S-def* **by** *simp*
moreover from ‹$B_2 = \{v\}$› **have** $W\ B_2 = w\ v$ **by** *simp*
ultimately show *?thesis* **using** *assms*(*2*,*3*) *S-def* **by** *simp*
qed
qed

If B_1 is not empty and $u \in V \cap S$ exceeds the maximum capacity of both B_1 and B_2, then inv_3 will be maintained if B_1 and B_2 are added to P_1 and P_2 respectively, emptied, and B_2 initialized with u.

lemma *loop-stepE*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ V$ $B_1 \neq \{\}$ $u \in V \cap S$ $W\ B_1 + w(u) > c$ $W\ B_2 + w(u) > c$
shows inv_3 $(P_1 \cup$ *wrap* $B_1)$ $(P_2 \cup$ *wrap* $B_2)$ $\{\}$ $\{u\}$ $(V - \{u\})$
proof −
note *invrules* = inv_2E[*OF* inv_3E(*1*)[*OF assms*(*1*)]]
from *assms*(*3*) **have** $u \in V$ **by** *blast*
from inv_1*-stepC*[*OF invrules*(*1*) *this*] **have** *1*: inv_1 $(P_1 \cup$ *wrap* $B_1)$ $(P_2 \cup$ *wrap* $B_2)$ $\{\}$ $\{u\}$ $(V - \{u\})$ **.**

have *2*: $(V - \{u\}) \cap L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup$ *wrap* $B_1 \cup$ *wrap* $\{\}.\ B \cap L \neq \{\}$
using *invrules*(*2*) **unfolding** *wrap-empty* **by** *blast*

from *invrules*(*3*) **obtain** *f* **where** *f-def*: *bij-betw f* P_1 ($\bigcup$ (P_2 ∪ *wrap* B_2)) ∀ $B \in P_1$. $c <$ *W B* + *w* (*f B*) **unfolding** *bij-exists-def* **by** *blast*
have $B_1 \notin P_1$ **using** $inv_1E(3)$[*OF invrules*(*1*)] **by** *blast*
have $u \notin$ ($\bigcup$ (P_2 ∪ *wrap* B_2)) **using** $inv_1E(2)$[*OF invrules*(*1*)] *assms*(*3*) **by** *blast*
have ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ *wrap* {*u*})) = ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ {{*u*}})) **unfolding** *wrap-def* **by** *simp*
also have ... = ($\bigcup$ (P_2 ∪ *wrap* B_2)) ∪ {*u*} **by** *simp*
finally have *UN*: ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ *wrap* {*u*})) = ($\bigcup$ (P_2 ∪ *wrap* B_2)) ∪ {*u*} **.**
have *wrap* B_1 = {B_1} **using** *wrap-not-empty*[*of* B_1] *assms*(*2*) **by** *simp*
let *?f* = *f* (B_1 := *u*)
have *BIJ*: *bij-betw ?f* (P_1 ∪ *wrap* B_1) ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ *wrap* {*u*}))
unfolding *wrap-empty* ‹*wrap* B_1 = {B_1}› *UN* **using** *f-def*(*1*) ‹$B_1 \notin P_1$› ‹$u \notin$ ($\bigcup$ (P_2 ∪ *wrap* B_2))›
by (*metis* (*no-types*, *lifting*) *bij-betw-cong fun-upd-other fun-upd-same notIn-Un-bij-betw3*)
have $c <$ *W* B_1 + *w* (*?f* B_1) **using** *assms*(*4*) **by** *simp*
then have (∀ $B \in P_1$ ∪ *wrap* B_1. $c <$ *W B* + *w* (*?f B*))
unfolding ‹*wrap* B_1 = {B_1}› **using** *f-def*(*2*) **by** *simp*
with *BIJ* **have** *bij-betw ?f* (P_1 ∪ *wrap* B_1) ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ *wrap* {*u*}))
∧ (∀ $B \in P_1$ ∪ *wrap* B_1. $c <$ *W B* + *w* (*?f B*)) **by** *blast*
then have *3*: *bij-exists* (P_1 ∪ *wrap* B_1) ($\bigcup$ (P_2 ∪ *wrap* B_2 ∪ *wrap* {*u*}))
unfolding *bij-exists-def* **by** *blast*

have *4*: *2* ∗ *card* (P_2 ∪ *wrap* B_2) ≤ *card* ($\bigcup$ (P_2 ∪ *wrap* B_2))
proof −
note *bprules* = *bpE*[*OF* $inv_1E(1)$[*OF invrules*(*1*)]]
have *pairwise disjnt* (P_2 ∪ *wrap* B_2)
using *bprules*(*1*) *pairwise-subset* **by** *blast*
moreover have $B_2 \notin P_2$ **using** $inv_1E(4)$[*OF invrules*(*1*)] **by** *simp*
ultimately have *DISJNT*: $\bigcup P_2 \cap B_2$ = {}
by (*auto*, *metis* (*no-types*, *opaque-lifting*) *sup-bot.right-neutral Un-insert-right disjnt-iff mk-disjoint-insert pairwise-insert wrap-Un*)
have *finite* ($\bigcup P_2$) **using** *U-Finite bprules*(*3*) **by** *auto*
have *finite* B_2 **using** $inv_1E(1)$[*OF invrules*(*1*)] *bp-bins-finite wrap-not-empty* **by** *blast*

have *2* ∗ *card* (P_2 ∪ *wrap* B_2) ≤ *2* ∗ (*card* P_2 + *card* (*wrap* B_2))
using *card-Un-le*[*of* P_2 ‹*wrap* B_2›] **by** *simp*
also have ... ≤ *2* ∗ *card* P_2 + *2* **using** *wrap-card* **by** *auto*
also have ... ≤ *card* ($\bigcup$ P_2) + *2* **using** *invrules*(*4*) **by** *simp*
also have ... ≤ *card* ($\bigcup$ P_2) + *card* B_2 **using** B_2*-at-least-two-objects*[*OF assms*(*1*,*3*,*5*)] **by** *simp*
also have ... = *card* ($\bigcup$ (P_2 ∪ {B_2})) **using** *DISJNT card-Un-disjoint*[*OF* ‹*finite* ($\bigcup P_2$)› ‹*finite* B_2›] **by** (*simp add*: *Un-commute*)
also have ... = *card* ($\bigcup$ (P_2 ∪ *wrap* B_2)) **by** (*cases* ‹B_2 = {}›) *auto*
finally show *?thesis* **.**
qed

from $inv_2 I[OF\ 1\ 2\ 3\ 4]$ **have** $inv_2\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$ **.**

from $inv_3 I[OF\ this]$ **show** *?thesis* **using** *assms(3)* **by** *blast*
qed

The bin packing algorithm as it is proposed in the article [2]. P will not only be a correct solution of the bin packing problem, but the amount of bins will be a lower bound for *3 / 2* of the amount of bins of any correct solution Q, and thus guarantee an approximation factor of *3 / 2* for the optimum.

lemma *bp-approx*:
$VARS\ P\ P_1\ P_2\ B_1\ B_2\ V\ u$
$\{True\}$
$P_1 := \{\};\ P_2 := \{\};\ B_1 := \{\};\ B_2 := \{\};\ V := U;$
$WHILE\ V \cap S \neq \{\}\ INV\ \{inv_3\ P_1\ P_2\ B_1\ B_2\ V\}\ DO$
$IF\ B_1 \neq \{\}$
$THEN\ u := (SOME\ u.\ u \in V \cap S)$
$ELSE\ IF\ V \cap L \neq \{\}$
$THEN\ u := (SOME\ u.\ u \in V \cap L)$
$ELSE\ u := (SOME\ u.\ u \in V \cap S)\ FI\ FI;$
$V := V - \{u\};$
$IF\ W(B_1) + w(u) \leq c$
$THEN\ B_1 := B_1 \cup \{u\}$
$ELSE\ IF\ W(B_2) + w(u) \leq c$
$THEN\ B_2 := B_2 \cup \{u\}$
$ELSE\ P_2 := P_2 \cup wrap\ B_2;\ B_2 := \{u\}\ FI;$
$P_1 := P_1 \cup wrap\ B_1;\ B_1 := \{\}\ FI$
$OD;$
$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\}$
$\{bp\ P \wedge (\forall Q.\ bp\ Q \longrightarrow card\ P \leq 3\ /\ 2 * card\ Q)\}$
proof (*vcg*, *goal-cases*)
case $(1\ P\ P_1\ P_2\ B_1\ B_2\ V\ u)$
then show *?case* **by** (*simp add*: *loop-init*)
next
case $(2\ P\ P_1\ P_2\ B_1\ B_2\ V\ u)$
then have *INV*: $inv_3\ P_1\ P_2\ B_1\ B_2\ V$ **..**
let $?s = SOME\ u.\ u \in V \cap S$
let $?l = SOME\ u.\ u \in V \cap L$
have *LIN*: $V \cap L \neq \{\} \Longrightarrow ?l \in V \cap L$ **using** *some-in-eq* **by** *metis*
then have *LWEIGHT*: $V \cap L \neq \{\} \Longrightarrow w\ ?l \leq c$ **using** *L-def weight* **by** *blast*
from *2* **have** $V \cap S \neq \{\}$ **..**
then have *IN*: $?s \in V \cap S$ **using** *some-in-eq* **by** *metis*
then have $w\ ?s \leq c$ **using** *S-def* **by** *simp*

then show *?case*
using *LWEIGHT loop-stepA[OF INV - - IN] loop-stepB[OF INV - LIN] loop-stepC[OF INV - IN]*
and *loop-stepD[OF INV - IN] loop-stepE[OF INV - IN]* **by** (*cases* $\langle B_1 = \{\}\rangle$,

cases ‹$V \cap L = \{\}$›) *auto*
next
  case ($3\ P\ P_1\ P_2\ B_1\ B_2\ V\ u$)
  then have *INV*: $inv_3\ P_1\ P_2\ B_1\ B_2\ V$ **and** *EMPTY*: $V \cap S = \{\}$ **by** *blast+*
  from $inv_1 E(1)[OF\ inv_2 E(1)[OF\ inv_3 E(1)[OF\ INV]]]$ **and** *bin-packing-lower-bound-card*[OF $EMPTY\ inv_3 E(1)[OF\ INV]$]
    show *?case* **by** *blast*
qed

end

5.3 The Full Linear Time Version of the Proposed Algorithm

Finally, we prove the Algorithm proposed on page 78 of the article [2]. This version generates the S and L sets beforehand and uses them directly to calculate the solution, thus removing the need for intersection operations, and ensuring linear time if we can perform *insertion*, *removal*, *and selection of an element*, *the union of two sets*, *and the emptiness test in constant time* [2].

locale *BinPacking-Complete* =
  fixes U :: *'a set* — A finite, non-empty set of objects
    and w :: *'a* ⇒ *real* — A mapping from objects to their respective weights (positive real numbers)
    and c :: *nat* — The maximum capacity of a bin (as a natural number)
  assumes *weight*: $\forall u \in U.\ 0 < w(u) \wedge w(u) \leq c$
    and *U-Finite*: *finite* U
    and *U-NE*: $U \neq \{\}$
begin

The correctness proofs will be identical to the ones of the simplified algorithm.

abbreviation W :: *'a set* ⇒ *real* **where**
  $W\ B \equiv (\sum u \in B.\ w(u))$

definition *bp* :: *'a set set* ⇒ *bool* **where**
  bp $P \longleftrightarrow$ *partition-on* $U\ P \wedge (\forall B \in P.\ W(B) \leq c)$

lemma *bpE*:
  assumes *bp* P
  shows *pairwise disjnt* P $\{\} \notin P$ $\bigcup P = U$ $\forall B \in P.\ W(B) \leq c$
  using *assms* **unfolding** *bp-def partition-on-def* **by** *blast+*

lemma *bpI*:
  assumes *pairwise disjnt* P $\{\} \notin P$ $\bigcup P = U$ $\forall B \in P.\ W(B) \leq c$
  shows *bp* P
  using *assms* **unfolding** *bp-def partition-on-def* **by** *blast*

definition inv_1 :: *'a set set* ⇒ *'a set set* ⇒ *'a set* ⇒ *'a set* ⇒ *'a set* ⇒ *bool* **where**

$inv_1\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$ — A correct solution to the bin packing problem

$\wedge \bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$ — The partial solution does not contain objects that have not yet been assigned

$\wedge B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$ — B_1 is distinct from all the other bins

$\wedge B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$ — B_2 is distinct from all the other bins

$\wedge (P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$ — The first and second partial solutions are disjoint from each other.

lemma inv_1E:
  assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
  shows $bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
    and $\bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$
    and $B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$
    and $B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$
    and $(P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$
  using *assms* **unfolding** inv_1*-def* **by** *auto*

lemma inv_1I:
  assumes $bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
    and $\bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$
    and $B_1 \notin (P_1 \cup P_2 \cup wrap\ B_2)$
    and $B_2 \notin (P_1 \cup wrap\ B_1 \cup P_2)$
    and $(P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$
  shows $inv_1\ P_1\ P_2\ B_1\ B_2\ V$
  using *assms* **unfolding** inv_1*-def* **by** *blast*

lemma *wrap-Un* [*simp*]: $wrap\ (M \cup \{x\}) = \{M \cup \{x\}\}$ **unfolding** *wrap-def* **by** *simp*
lemma *wrap-empty* [*simp*]: $wrap\ \{\} = \{\}$ **unfolding** *wrap-def* **by** *simp*
lemma *wrap-not-empty* [*simp*]: $M \neq \{\} \longleftrightarrow wrap\ M = \{M\}$ **unfolding** *wrap-def* **by** *simp*

lemma inv_1*-stepA*:
  assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W(B_1) + w(u) \leq c$
  shows $inv_1\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (V - \{u\})$
proof –
  note $invrules = inv_1E[OF\ assms(1)]$ **and** $bprules = bpE[OF\ invrules(1)]$

  — Rule 1: Pairwise Disjoint
  have $NOTIN$: $\forall M \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ u \notin M$
    using $invrules(2)\ assms(2)$ **by** *blast*
  have $\{\{v\}\ |v.\ v \in V\} = \{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
    using $assms(2)$ **by** *blast*
  then have $pairwise\ disjnt\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup (\{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}))$

using *bprules*(*1*) *assms*(*2*) **by** *simp*
then have *pairwise disjnt* (*wrap* B_1 $\cup$ $\{\{u\}\}$ $\cup$ P_1 $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\})$ **by** (*simp add*: *Un-commute*)
then have *assm*: *pairwise disjnt* (*wrap* B_1 $\cup$ $\{\{u\}\}$ $\cup$ (P_1 $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$)) **by** (*simp add*: *Un-assoc*)
have *pairwise disjnt* ($\{B_1 \cup \{u\}\}$ $\cup$ (P_1 $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$))
proof (*cases* ‹$B_1 = \{\}$›)
case *True* **with** *assm* **show** *?thesis* **by** *simp*
next
case *False*
with *assm* **have** *assm*: *pairwise disjnt* ($\{B_1\}$ $\cup$ $\{\{u\}\}$ $\cup$ (P_1 $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$)) **by** *simp*
from *NOTIN* **have** $\{u\} \notin P_1 \cup P_2 \cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$ **by** *blast*
from *pairwise-disjnt-Un*[*OF assm - this*] *invrules*(*2*,*3*) **show** *?thesis*
using *False* **by** *auto*
qed
then have *1*: *pairwise disjnt* (P_1 $\cup$ *wrap* ($B_1 \cup \{u\}$) $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$)
unfolding *wrap-Un* **by** *simp*

— Rule 2: No empty sets
from *bprules*(*2*) **have** *2*: $\{\} \notin P_1 \cup$ *wrap* ($B_1 \cup \{u\}$) $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$
unfolding *wrap-def* **by** *simp*

— Rule 3: Union preserved
from *bprules*(*3*) **have** $\bigcup$ (P_1 $\cup$ *wrap* B_1 $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{u\}\}$ $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$) $= U$
using *assms*(*2*) **by** *blast*
then have *3*: $\bigcup$ (P_1 $\cup$ *wrap* ($B_1 \cup \{u\}$) $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$) $= U$
unfolding *wrap-def* **by** *force*

— Rule 4: Weights below capacity
have $0 < w\ u$ **using** *weight assms*(*2*) *bprules*(*3*) **by** *blast*
have *finite* B_1 **using** *bprules*(*3*) *U-Finite* **by** (*cases* ‹$B_1 = \{\}$›) *auto*
then have W ($B_1 \cup \{u\}$) $\leq W\ B_1 + w\ u$ **using** ‹$0 < w\ u$› **by** (*cases* ‹$u \in B_1$›) (*auto simp*: *insert-absorb*)
also have ... $\leq c$ **using** *assms*(*3*) **.**
finally have W ($B_1 \cup \{u\}$) $\leq c$ **.**
then have $\forall B \in$ *wrap* ($B_1 \cup \{u\}$). $W\ B \leq c$ **unfolding** *wrap-Un* **by** *blast*
moreover have $\forall B{\in}P_1 \cup P_2 \cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$. $W\ B \leq c$
using *bprules*(*4*) **by** *blast*
ultimately have *4*: $\forall B{\in}P_1 \cup$ *wrap* ($B_1 \cup \{u\}$) $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$. $W\ B \leq c$ **by** *blast*
from *bpI*[*OF 1 2 3 4*] **have** *1*: *bp* (P_1 $\cup$ *wrap* ($B_1 \cup \{u\}$) $\cup$ P_2 $\cup$ *wrap* B_2 $\cup$ $\{\{v\}$ $|v.$ $v \in V - \{u\}\}$) **.**

— Auxiliary information is preserved
have $u \in U$ **using** *assms*(*2*) *bprules*(*3*) **by** *blast*
then have R: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have L: $\bigcup (P_1 \cup \textit{wrap}\ (B_1 \cup \{u\}) \cup P_2 \cup \textit{wrap}\ B_2) = \bigcup (P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \textit{wrap}\ B_2) \cup \{u\}$
unfolding *wrap-def* **using** *NOTIN* **by** *auto*
have *2*: $\bigcup (P_1 \cup \textit{wrap}\ (B_1 \cup \{u\}) \cup P_2 \cup \textit{wrap}\ B_2) = U - (V - \{u\})$
unfolding L R *invrules*(*2*) **..**
have *3*: $B_1 \cup \{u\} \notin P_1 \cup P_2 \cup \textit{wrap}\ B_2$
using *NOTIN* **by** *auto*
have *4*: $B_2 \notin P_1 \cup \textit{wrap}\ (B_1 \cup \{u\}) \cup P_2$
using *invrules*(*4*) *NOTIN* **unfolding** *wrap-def* **by** *fastforce*
have *5*: $(P_1 \cup \textit{wrap}\ (B_1 \cup \{u\})) \cap (P_2 \cup \textit{wrap}\ B_2) = \{\}$
using *invrules*(*5*) *NOTIN* **unfolding** *wrap-Un* **by** *auto*

from $\textit{inv}_1 I$[*OF 1 2 3 4 5*] **show** *?thesis* **.**
qed

lemma $\textit{inv}_1$*-stepB*:
assumes $\textit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W\ B_2 + w\ u \leq c$
shows $\textit{inv}_1\ (P_1 \cup \textit{wrap}\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (V - \{u\})$
proof −
note $\textit{invrules} = \textit{inv}_1 E$[*OF assms*(*1*)] **and** *bprules* = *bpE*[*OF invrules*(*1*)]

have *NOTIN*: $\forall M \in P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \textit{wrap}\ B_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}.\ u \notin M$
using *invrules*(*2*) *assms*(*2*) **by** *blast*
have $\{\{v\} \mid v.\ v \in V\} = \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$
using *assms*(*2*) **by** *blast*
then have *pairwise disjnt* $(P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \textit{wrap}\ B_2 \cup \{\{u\}\} \cup \{\{v\} \mid v.\ v \in V - \{u\}\})$
using *bprules*(*1*) *assms*(*2*) **by** *simp*
then have *assm*: *pairwise disjnt* $(\textit{wrap}\ B_2 \cup \{\{u\}\} \cup (P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$
by (*simp add*: *Un-assoc Un-commute*)
have *pairwise disjnt* $(\{B_2 \cup \{u\}\} \cup (P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$
proof (*cases* ‹$B_2 = \{\}$›)
case *True* **with** *assm* **show** *?thesis* **by** *simp*
next
case *False*
with *assm* **have** *assm*: *pairwise disjnt* $(\{B_2\} \cup \{\{u\}\} \cup (P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}))$ **by** *simp*
from *NOTIN* **have** $\{u\} \notin P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \{\{v\} \mid v.\ v \in V - \{u\}\}$ **by** *blast*
from *pairwise-disjnt-Un*[*OF assm - this*] *invrules*(*2*,*4*) **show** *?thesis*
using *False* **by** *auto*
qed

then have *1*: *pairwise disjnt* $(P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$
unfolding *wrap-Un* **by** *simp*

— Rule 2: No empty sets
from *bprules*(*2*) **have** *2*: $\{\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
unfolding *wrap-def* **by** *simp*

— Rule 3: Union preserved
from *bprules*(*3*) **have** $\bigcup\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}) = U$
using *assms*(*2*) **by** *blast*
then have *3*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\}) = U$
unfolding *wrap-def* **by** *force*

— Rule 4: Weights below capacity
have $0 < w\ u$ **using** *weight assms*(*2*) *bprules*(*3*) **by** *blast*
have *finite* B_2 **using** *bprules*(*3*) *U-Finite* **by** (*cases* ‹$B_2 = \{\}$›) *auto*
then have $W\ (B_2 \cup \{u\}) \le W\ B_2 + w\ u$ **using** ‹$0 < w\ u$› **by** (*cases* ‹$u \in B_2$›) (*auto simp*: *insert-absorb*)
also have ... $\le c$ **using** *assms*(*3*) **.**
finally have $W\ (B_2 \cup \{u\}) \le c$ **.**
then have $\forall B \in wrap\ (B_2 \cup \{u\}).\ W\ B \le c$ **unfolding** *wrap-Un* **by** *blast*
moreover have $\forall B \in P_1 \cup wrap\ B_1 \cup P_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \le c$
using *bprules*(*4*) **by** *blast*
ultimately have *4*: $\forall B \in P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ W\ B \le c$
by *auto*
from *bpI*[*OF 1 2 3 4*] **have** *1*: *bp* $(P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\}) \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$ **.**

— Auxiliary information is preserved
have $u \in U$ **using** *assms*(*2*) *bprules*(*3*) **by** *blast*
then have *R*: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have *L*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\})) = \bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ B_2) \cup \{u\}$
unfolding *wrap-def* **using** *NOTIN* **by** *auto*
have *2*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2 \cup wrap\ (B_2 \cup \{u\})) = U - (V - \{u\})$
unfolding *L R* **using** *invrules*(*2*) **by** *simp*
have *3*: $\{\} \notin P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ (B_2 \cup \{u\})$
using *bpE*(*2*)[*OF 1*] **by** *simp*
have *4*: $B_2 \cup \{u\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup P_2$
using *NOTIN* **by** *auto*
have *5*: $(P_1 \cup wrap\ B_1 \cup wrap\ \{\}) \cap (P_2 \cup wrap\ (B_2 \cup \{u\})) = \{\}$
using *invrules*(*5*) *NOTIN* **unfolding** *wrap-empty wrap-Un* **by** *auto*

from $inv_1 I[OF\ 1\ 2\ 3\ 4\ 5]$ **show** *?thesis* **.**
qed

lemma inv_1-*stepC*:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V$
shows $inv_1\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$
proof −
note $invrules = inv_1 E[OF\ assms(1)]$
— Rule 1-4: Correct Bin Packing
have $P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
$= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{u\}\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
by (*metis* (*no-types*, *lifting*) *Un-assoc Un-empty-right insert-not-empty wrap-empty wrap-not-empty*)
also have ... $= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}$
using *assms*(*2*) **by** *auto*
finally have *EQ*: $P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\}$
$= P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V\}$ **.**
from *invrules*(*1*) **have** *1*: $bp\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\} \cup \{\{v\}\ |v.\ v \in V - \{u\}\})$
unfolding *EQ* **.**

— Auxiliary information is preserved
have *NOTIN*: $\forall M \in P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in V - \{u\}\}.\ u \notin M$
using *invrules*(*2*) *assms*(*2*) **by** *blast*
have $u \in U$ **using** *assms*(*2*) *bpE*(*3*)[*OF invrules*(*1*)] **by** *blast*
then have *R*: $U - (V - \{u\}) = U - V \cup \{u\}$ **by** *blast*
have *L*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}) = \bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2)) \cup \{u\}$
unfolding *wrap-def* **using** *NOTIN* **by** *auto*
have *2*: $\bigcup\ (P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}) = U - (V - \{u\})$
unfolding *L R* **using** *invrules*(*2*) **by** *auto*
have *3*: $\{\} \notin P_1 \cup wrap\ B_1 \cup (P_2 \cup wrap\ B_2) \cup wrap\ \{u\}$
using *bpE*(*2*)[*OF 1*] **by** *simp*
have *4*: $\{u\} \notin P_1 \cup wrap\ B_1 \cup wrap\ \{\} \cup (P_2 \cup wrap\ B_2)$
using *NOTIN* **by** *auto*
have *5*: $(P_1 \cup wrap\ B_1 \cup wrap\ \{\}) \cap (P_2 \cup wrap\ B_2 \cup wrap\ \{u\}) = \{\}$
using *invrules*(*5*) *NOTIN* **unfolding** *wrap-def* **by** *force*

from $inv_1 I[OF\ 1\ 2\ 3\ 4\ 5]$ **show** *?thesis* **.**
qed

From this point onward, we will require a different approach for proving lower bounds. Instead of fixing and assuming the definitions of the S and L sets, we will introduce the abbreviations S_U and L_U for any occurrences of the original S and L sets. The union of S and L can be interpreted as

V. As a result, occurrences of $V \cap S$ become $(S \cup L) \cap S = S$, and $V \cap L$ become $(S \cup L) \cap L = L$. Occurrences of these sets will have to be replaced appropriately.

abbreviation S_U **where**
  $S_U \equiv \{u \in U.\ w\ u \leq c\ /\ 2\}$

abbreviation L_U **where**
  $L_U \equiv \{u \in U.\ c\ /\ 2 < w\ u\}$

As we will remove elements from S and L, we will only be able to show that they remain subsets of S_U and L_U respectively.

abbreviation *SL* **where**
  $SL\ S\ L \equiv S \subseteq S_U \wedge L \subseteq L_U$

lemma *bp-bins-finite* [*simp*]:
  assumes *bp P*
  shows $\forall B \in P.$ *finite B*
  using *bpE*(*3*)[*OF assms*] *U-Finite* **by** (*meson Sup-upper finite-subset*)

lemma *bp-sol-finite* [*simp*]:
  assumes *bp P*
  shows *finite P*
  using *bpE*(*3*)[*OF assms*] *U-Finite* **by** (*simp add*: *finite-UnionD*)

lemma *only-one-L-per-bin*:
  assumes $bp\ P\ B \in P$
  shows $\forall x \in B.\ \forall y \in B.\ x \neq y \longrightarrow x \notin L_U \vee y \notin L_U$
proof (*rule ccontr*, *simp*)
  assume $\exists x{\in}B.\ \exists y{\in}B.\ x \neq y \wedge y \in U \wedge x \in U \wedge real\ c < w\ x * 2 \wedge real\ c < w\ y * 2$
  then obtain *x y* **where** $*$: $x \in B\ y \in B\ x \neq y\ x \in L_U\ y \in L_U$ **by** *auto*
  then have $c < w\ x + w\ y$ **by** *force*
  have *finite B* **using** *assms* **by** *simp*
  have $y \in B - \{x\}$ **using** $*(2,3)$ **by** *blast*
  have $W\ B = W\ (B - \{x\}) + w\ x$
    using $*(1)$ ‹*finite B*› **by** (*simp add*: *sum.remove*)
  also have ... $= W\ (B - \{x\} - \{y\}) + w\ x + w\ y$
    using ‹$y \in B - \{x\}$› ‹*finite B*› **by** (*simp add*: *sum.remove*)
  finally have $*$: $W\ B = W\ (B - \{x\} - \{y\}) + w\ x + w\ y$ **.**
  have $\forall u \in B.\ 0 < w\ u$ **using** *bpE*(*3*)[*OF assms*(*1*)] *assms*(*2*) *weight* **by** *blast*
  then have $0 \leq W\ (B - \{x\} - \{y\})$ **by** (*smt DiffD1 sum-nonneg*)
  with $*$ **have** $c < W\ B$ **using** ‹$c < w\ x + w\ y$› **by** *simp*
  then show *False* **using** *bpE*(*4*)[*OF assms*(*1*)] *assms*(*2*) **by** *fastforce*
qed

lemma *L-lower-bound-card*:
  assumes *bp P*
  shows $card\ L_U \leq card\ P$
proof −

have $\forall x \in L_U.\ \exists B \in P.\ x \in B$
using *bpE(3)[OF assms]* **by** *blast*
then obtain *f* **where** *f-def*: $\forall u \in L_U.\ u \in f\ u \wedge f\ u \in P$ **by** *metis*
then have *inj-on f* L_U
unfolding *inj-on-def* **using** *only-one-L-per-bin[OF assms]* **by** *blast*
then have *card-eq*: *card* L_U = *card* (*f* ‘ L_U) **by** (*simp add*: *card-image*)
have *f* ‘ $L_U \subseteq P$ **using** *f-def* **by** *blast*
moreover have *finite P* **using** *assms* **by** *simp*
ultimately have *card* (*f* ‘ L_U) $\leq$ *card P* **by** (*simp add*: *card-mono*)
then show *?thesis* **unfolding** *card-eq* **.**
qed

lemma *subset-bp-card*:
assumes *bp P* $M \subseteq P$ $\forall B \in M.\ B \cap L_U \neq \{\}$
shows *card M* $\leq$ *card* L_U
proof −
have $\forall B \in M.\ \exists u \in L_U.\ u \in B$ **using** *assms(3)* **by** *fast*
then have $\exists f.\ \forall B \in M.\ f\ B \in L_U \wedge f\ B \in B$ **by** *metis*
then obtain *f* **where** *f-def*: $\forall B \in M.\ f\ B \in L_U \wedge f\ B \in B$ **..**
have *inj-on f M*
proof (*rule ccontr*)
assume ¬ *inj-on f M*
then have $\exists x \in M.\ \exists y \in M.\ x \neq y \wedge f\ x = f\ y$ **unfolding** *inj-on-def* **by** *blast*
then obtain *x y* **where** $*$: $x \in M$ $y \in M$ $x \neq y$ $f\ x = f\ y$ **by** *blast*
then have $\exists u.\ u \in x \wedge u \in y$ **using** *f-def* **by** *metis*
then have $x \cap y \neq \{\}$ **by** *blast*
moreover have *pairwise disjnt M* **using** *pairwise-subset[OF bpE(1)[OF assms(1)] assms(2)]* **.**
ultimately show *False* **using** $*$ **unfolding** *pairwise-def disjnt-def* **by** *simp*
qed
moreover have *finite* L_U **using** *U-Finite* **by** *auto*
moreover have *f* ‘ $M \subseteq L_U$ **using** *f-def* **by** *blast*
ultimately show *?thesis* **using** *card-inj-on-le* **by** *blast*
qed

lemma *L-bins-lower-bound-card*:
assumes *bp P* inv_1 P_1 P_2 B_1 B_2 $(S \cup L)$ $\forall B \in P_1 \cup wrap\ B_1.\ B \cap L_U \neq \{\}$
and *SL-def*: *SL S L*
shows *card* $(P_1 \cup wrap\ B_1 \cup \{\{v\} \mid v.\ v \in L\}) \leq$ *card P*
proof −
note *invrules* = inv_1E*[OF assms(2)]*
have $\forall B \in \{\{v\} \mid v.\ v \in L\}.\ B \cap L_U \neq \{\}$ **using** *SL-def* **by** *blast*
with *assms(3)* **have**
$P_1 \cup wrap\ B_1 \cup \{\{v\} \mid v.\ v \in L\} \subseteq P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in S \cup L\}$
$\forall B \in P_1 \cup wrap\ B_1 \cup \{\{v\} \mid v.\ v \in L\}.\ B \cap L_U \neq \{\}$ **by** *blast+*
from *subset-bp-card[OF invrules(1) this]* **show** *?thesis*
using *L-lower-bound-card[OF assms(1)]* **by** *linarith*
qed

lemma *sum-Un-eq-sum-sum*:
assumes *bp P*
shows $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
proof −
have *FINITE*: $\forall B \in P.$ *finite B* **using** *assms* **by** *simp*
have *DISJNT*: $\forall A \in P.\ \forall B \in P.\ A \neq B \longrightarrow A \cap B = \{\}$
using *bpE*(*1*)[*OF assms*] **unfolding** *pairwise-def disjnt-def* **.**
have $(\sum u \in (\bigcup P).\ w\ u) = (\sum B \in P.\ W\ B)$
using *sum.Union-disjoint*[*OF FINITE DISJNT*] **by** *auto*
then show *?thesis* **unfolding** *bpE*(*3*)[*OF assms*] **.**
qed

lemma *sum-lower-bound-card*:
assumes *bp P*
shows $(\sum u \in U.\ w\ u) \leq c * card\ P$
proof −
have $*$: $\forall B \in P.\ 0 < W\ B \wedge W\ B \leq c$
using *bpE*(*2*−*4*)[*OF assms*] *weight* **by** (*metis UnionI assms bp-bins-finite sum-pos*)
have $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
using *sum-Un-eq-sum-sum*[*OF assms*] **.**
also have ... $\leq (\sum B \in P.\ c)$ **using** *sum-mono* $*$ **by** *fastforce*
also have ... $= c * card\ P$ **by** *simp*
finally show *?thesis* **.**
qed

lemma *bp-NE*:
assumes *bp P*
shows $P \neq \{\}$
using *U-NE bpE*(*3*)[*OF assms*] **by** *blast*

lemma *sum-Un-ge*:
fixes *f* :: - $\Rightarrow$ *real*
assumes *finite M finite N* $\forall B \in M \cup N.\ 0 < f\ B$
shows $sum\ f\ M \leq sum\ f\ (M \cup N)$
proof −
have $0 \leq sum\ f\ N - sum\ f\ (M \cap N)$
using *assms* **by** (*smt DiffD1 inf.cobounded2 UnCI sum-mono2*)
then have $sum\ f\ M \leq sum\ f\ M + sum\ f\ N - sum\ f\ (M \cap N)$
by *simp*
also have ... $= sum\ f\ (M \cup N)$
using *sum-Un*[*OF assms*(*1*,*2*), *symmetric*] **.**
finally show *?thesis* **.**
qed

definition *bij-exists* :: $'a\ set\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
bij-exists $P\ V = (\exists f.\ bij\text{-}betw\ f\ P\ V \wedge (\forall B \in P.\ W\ B + w\ (f\ B) > c))$

lemma P_1*-lower-bound-card*:
assumes *bp P inv*$_1$ P_1 P_2 B_1 B_2 $(S \cup L)$ *bij-exists* P_1 $(\bigcup(P_2 \cup \mathit{wrap}\ B_2))$
shows *card* $P_1 + 1 \leq$ *card P*
proof (*cases* ‹$P_1 = \{\}$›)
case *True*
have *finite P* **using** *assms*(*1*) **by** *simp*
then have $1 \leq$ *card P* **using** *bp-NE*[*OF assms*(*1*)]
by (*metis Nat.add-0-right Suc-diff-1 Suc-le-mono card-gt-0-iff le0 mult-Suc-right nat-mult-1*)
then show *?thesis* **unfolding** *True* **by** *simp*
next
note *invrules* = *inv*$_1$*E*[*OF assms*(*2*)]
case *False*
obtain *f* **where** *f-def*: *bij-betw f* P_1 $(\bigcup(P_2 \cup \mathit{wrap}\ B_2))$ $\forall B \in P_1.\ W\ B + w\ (f\ B) > c$
using *assms*(*3*) **unfolding** *bij-exists-def* **by** *blast*
have *FINITE*: *finite* P_1 *finite* $(P_2 \cup \mathit{wrap}\ B_2)$ *finite* $(P_1 \cup P_2 \cup \mathit{wrap}\ B_2)$ *finite* $(\mathit{wrap}\ B_1 \cup \{\{v\} \mid v.\ v \in S \cup L\})$
using *inv*$_1$*E*(*1*)[*OF assms*(*2*)] *bp-sol-finite* **by** *blast+*

have *F*: $\forall B \in P_2 \cup \mathit{wrap}\ B_2.$ *finite B* **using** *invrules*(*1*) **by** *simp*
have *D*: $\forall A \in P_2 \cup \mathit{wrap}\ B_2.\ \forall B \in P_2 \cup \mathit{wrap}\ B_2.\ A \neq B \longrightarrow A \cap B = \{\}$
using *bpE*(*1*)[*OF invrules*(*1*)] **unfolding** *pairwise-def disjnt-def* **by** *auto*
have *sum-eq*: $W\ (\bigcup\ (P_2 \cup \mathit{wrap}\ B_2)) = (\sum B \in P_2 \cup \mathit{wrap}\ B_2.\ W\ B)$
using *sum.Union-disjoint*[*OF F D*] **by** *auto*

have $\forall B{\in}P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2 \cup \{\{v\} \mid v.\ v \in S \cup L\}.\ 0 < W\ B$
using *bpE*(*2*,*3*)[*OF invrules*(*1*)] *weight* **by** (*metis* (*no-types*, *lifting*) *UnionI bp-bins-finite invrules*(*1*) *sum-pos*)
then have $(\sum B \in P_1 \cup P_2 \cup \mathit{wrap}\ B_2.\ W\ B) \leq (\sum B \in P_1 \cup P_2 \cup \mathit{wrap}\ B_2 \cup (\mathit{wrap}\ B_1 \cup \{\{v\} \mid v.\ v \in S \cup L\}).\ W\ B)$
using *sum-Un-ge*[*OF FINITE*(*3*,*4*), *of W*] **by** *blast*
also have ... $= (\sum B \in P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2 \cup \{\{v\} \mid v.\ v \in S \cup L\}.\ W\ B)$ **by** (*smt Un-assoc Un-commute*)
also have ... $= W\ U$ **using** *sum-Un-eq-sum-sum*[*OF invrules*(*1*), *symmetric*] **.**
finally have $*$: $(\sum B \in P_1 \cup P_2 \cup \mathit{wrap}\ B_2.\ W\ B) \leq W\ U$ **.**
have *DISJNT*: $P_1 \cap (P_2 \cup \mathit{wrap}\ B_2) = \{\}$ **using** *invrules*(*5*) **by** *blast*

— This part of the proof is based on the proof on page 72 of the article [2].
have $c * \mathit{card}\ P_1 = (\sum B \in P_1.\ c)$ **by** *simp*
also have ... $< (\sum B \in P_1.\ W\ B + w\ (f\ B))$
using *f-def*(*2*) *sum-strict-mono*[*OF FINITE*(*1*) *False*] **by** *fastforce*
also have ... $= (\sum B \in P_1.\ W\ B) + (\sum B \in P_1.\ w\ (f\ B))$
by (*simp add*: *Groups-Big.comm-monoid-add-class.sum.distrib*)
also have ... $= (\sum B \in P_1.\ W\ B) + W\ (\bigcup\ (P_2 \cup \mathit{wrap}\ B_2))$ **unfolding** *sum.reindex-bij-betw*[*OF f-def*(*1*), *of w*] **..**
also have ... $= (\sum B \in P_1.\ W\ B) + (\sum B \in P_2 \cup \mathit{wrap}\ B_2.\ W\ B)$ **unfolding** *sum-eq* **..**
also have ... $= (\sum B \in P_1 \cup P_2 \cup \mathit{wrap}\ B_2.\ W\ B)$ **using** *sum.union-disjoint*[*OF*

FINITE(*1*,*2*) *DISJNT*, *of W*] **by** (*simp add*: *Un-assoc*)
also have ... $\leq$ ($\sum u \in U.\ w\ u$) **using** $*$ **.**
also have ... $\leq c * card\ P$ **using** *sum-lower-bound-card*[*OF assms*(*1*)] **.**
finally show *?thesis* **by** (*meson discrete nat-mult-less-cancel-disj of-nat-less-imp-less*)
qed

lemma P_1-B_1-*lower-bound-card*:
assumes *bp P* $inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)$ *bij-exists* P_1 ($\bigcup(P_2 \cup wrap\ B_2)$)
shows *card* ($P_1 \cup wrap\ B_1) \leq card\ P$
proof −
have *card* ($P_1 \cup wrap\ B_1) \leq card\ P_1 + card\ (wrap\ B_1)$
using *card-Un-le* **by** *blast*
also have ... $\leq card\ P_1 + 1$ **using** *wrap-card* **by** *simp*
also have ... $\leq card\ P$ **using** P_1-*lower-bound-card*[*OF assms*] **.**
finally show *?thesis* **.**
qed

lemma P_2-B_2-*lower-bound*-P_1:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)\ 2 * card\ P_2 \leq card\ (\bigcup P_2)$ *bij-exists* P_1 ($\bigcup(P_2 \cup wrap\ B_2)$)
shows $2 * card\ (P_2 \cup wrap\ B_2) \leq card\ P_1 + 1$
proof −
note *invrules* = inv_1E[*OF assms*(*1*)] **and** *bprules* = *bpE*[*OF invrules*(*1*)]

have *pairwise disjnt* ($P_2 \cup wrap\ B_2$)
using *bprules*(*1*) *pairwise-subset* **by** *blast*
moreover have $B_2 \notin P_2$ **using** *invrules*(*4*) **by** *simp*
ultimately have *DISJNT*: $\bigcup P_2 \cap B_2 = \{\}$
by (*auto*, *metis* (*no-types*, *opaque-lifting*) *sup-bot.right-neutral Un-insert-right disjnt-iff mk-disjoint-insert pairwise-insert wrap-Un*)

have *finite* ($\bigcup P_2$) **using** *U-Finite bprules*(*3*) **by** *auto*
have *finite* B_2 **using** *bp-bins-finite*[*OF invrules*(*1*)] *wrap-not-empty* **by** *blast*
have *finite* P_2 *finite* (*wrap* B_2) **using** *bp-sol-finite*[*OF invrules*(*1*)] **by** *blast+*
have *DISJNT2*: $P_2 \cap wrap\ B_2 = \{\}$ **unfolding** *wrap-def* **using** ‹$B_2 \notin P_2$› **by** *auto*
have *card* (*wrap* B_2) $\leq card\ B_2$
proof (*cases* ‹$B_2 = \{\}$›)
case *False*
then have $1 \leq card\ B_2$ **by** (*simp add*: *leI* ‹*finite* B_2›)
then show *?thesis* **using** *wrap-card*[*of* B_2] **by** *linarith*
qed *simp*

— This part of the proof is based on the proof on page 73 of the article [2].
from *assms*(*2*) **have** $2 * card\ P_2 + 2 * card\ (wrap\ B_2) \leq card\ (\bigcup P_2) + card\ (wrap\ B_2) + 1$
using *wrap-card*[*of* B_2] **by** *linarith*
then have $2 * (card\ P_2 + card\ (wrap\ B_2)) \leq card\ (\bigcup P_2) + card\ B_2 + 1$
using ‹$card\ (wrap\ B_2) \leq card\ B_2$› **by** *simp*

then have $2 * (card\ (P_2 \cup wrap\ B_2)) \leq card\ (\bigcup P_2 \cup B_2) + 1$
using $card\text{-}Un\text{-}disjoint[OF\ \langle finite\ (\bigcup P_2)\rangle\ \langle finite\ B_2\rangle\ DISJNT]$
and $card\text{-}Un\text{-}disjoint[OF\ \langle finite\ P_2\rangle\ \langle finite\ (wrap\ B_2)\rangle\ DISJNT2]$ **by** *argo*
then have $2 * (card\ (P_2 \cup wrap\ B_2)) \leq card\ (\bigcup (P_2 \cup wrap\ B_2)) + 1$
by $(cases\ \langle B_2 = \{\}\rangle)\ (auto\ simp\colon Un\text{-}commute)$
then show $2 * (card\ (P_2 \cup wrap\ B_2)) \leq card\ P_1 + 1$
using $assms(3)$ *bij-betw-same-card* **unfolding** *bij-exists-def* **by** *metis*
qed

We add $SL\ S\ L$ to inv_2 to ensure that the S and L sets only contain objects with correct weights.

definition $inv_2 :: 'a\ set\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow bool$ **where**
$inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L \longleftrightarrow inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)$ — inv_1 holds for the partial solution
$\wedge\ (L \neq \{\} \longrightarrow (\forall B \in P_1 \cup wrap\ B_1.\ B \cap L_U \neq \{\}))$ — If there are still large objects left, then every bin of the first partial solution must contain a large object
$\wedge\ bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$ — There exists a bijective function between the bins of the first partial solution and the objects of the second one
$\wedge\ (2 * card\ P_2 \leq card\ (\bigcup P_2))$ — There are at most twice as many bins in P_2 as there are objects in P_2
$\wedge\ SL\ S\ L$ — S and L are subsets of S_U and L_U

lemma inv_2E:
assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$
shows $inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)$
and $L \neq \{\} \Longrightarrow \forall B \in P_1 \cup wrap\ B_1.\ B \cap L_U \neq \{\}$
and $bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
and $2 * card\ P_2 \leq card\ (\bigcup P_2)$
and $SL\ S\ L$
using *assms* **unfolding** $inv_2\text{-}def$ **by** *blast+*

lemma inv_2I:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)$
and $L \neq \{\} \Longrightarrow \forall B \in P_1 \cup wrap\ B_1.\ B \cap L_U \neq \{\}$
and $bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
and $2 * card\ P_2 \leq card\ (\bigcup P_2)$
and $SL\ S\ L$
shows $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$
using *assms* **unfolding** $inv_2\text{-}def$ **by** *blast*

lemma *bin-packing-lower-bound-card*:
assumes $S = \{\}\ inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L\ bp\ P$
shows $card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in S \cup L\}) \leq 3\ /\ 2 * card\ P$
proof $(cases\ \langle L = \{\}\rangle)$
note $invrules = inv_2E[OF\ assms(2)]$

case *True*
then have $card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in S \cup L\})$
$= card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2)$ **using** $assms(1)$ **by** *simp*
also have $... \leq card\ (P_1 \cup wrap\ B_1) + card\ (P_2 \cup wrap\ B_2)$
using $card\text{-}Un\text{-}le[of\ \langle P_1 \cup wrap\ B_1\rangle]$ **by** $(simp\ add{:}\ Un\text{-}assoc)$
also have $... \leq card\ P + card\ (P_2 \cup wrap\ B_2)$
using $P_1\text{-}B_1\text{-}lower\text{-}bound\text{-}card[OF\ assms(3)\ invrules(1,3)]$ **by** *simp*
also have $... \leq card\ P + card\ P\ /\ 2$
using $P_2\text{-}B_2\text{-}lower\text{-}bound\text{-}P_1[OF\ invrules(1,4,3)]$
and $P_1\text{-}lower\text{-}bound\text{-}card[OF\ assms(3)\ invrules(1,3)]$ **by** *linarith*
finally show *?thesis* **by** *linarith*
next
note $invrules = inv_2E[OF\ assms(2)]$
case *False*
have $card\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}\ |v.\ v \in S \cup L\})$
$= card\ (P_1 \cup wrap\ B_1 \cup \{\{v\}\ |v.\ v \in L\} \cup P_2 \cup wrap\ B_2)$
using $assms(1)$ **by** $(simp\ add{:}\ Un\text{-}commute\ Un\text{-}assoc)$
also have $... \leq card\ (P_1 \cup wrap\ B_1 \cup \{\{v\}\ |v.\ v \in L\}) + card\ (P_2 \cup wrap\ B_2)$
using $card\text{-}Un\text{-}le[of\ \langle P_1 \cup wrap\ B_1 \cup \{\{v\}\ |v.\ v \in L\}\rangle]$ **by** $(simp\ add{:}\ Un\text{-}assoc)$
also have $... \leq card\ P + card\ (P_2 \cup wrap\ B_2)$
using $L\text{-}bins\text{-}lower\text{-}bound\text{-}card[OF\ assms(3)\ invrules(1)\ invrules(2)[OF\ False]$
$invrules(5)]$ **by** *linarith*
also have $... \leq card\ P + card\ P\ /\ 2$
using $P_2\text{-}B_2\text{-}lower\text{-}bound\text{-}P_1[OF\ invrules(1,4,3)]$
and $P_1\text{-}lower\text{-}bound\text{-}card[OF\ assms(3)\ invrules(1,3)]$ **by** *linarith*
finally show *?thesis* **by** *linarith*
qed

definition $inv_3 :: 'a\ set\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow$
$bool$ **where**
$inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L \longleftrightarrow inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L \wedge B_2 \subseteq S_U$

lemma inv_3E:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L$
shows $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$ **and** $B_2 \subseteq S_U$
using *assms* **unfolding** $inv_3\text{-}def$ **by** *blast+*

lemma inv_3I:
assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$ **and** $B_2 \subseteq S_U$
shows $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L$
using *assms* **unfolding** $inv_3\text{-}def$ **by** *blast*

lemma *loop-init*:
$inv_3\ \{\}\ \{\}\ \{\}\ \{\}\ S_U\ L_U$
proof −
have $S_U \cup L_U = U$ **by** *auto*
then have $*$: $inv_1\ \{\}\ \{\}\ \{\}\ \{\}\ (S_U \cup L_U)$
unfolding $bp\text{-}def\ partition\text{-}on\text{-}def\ pairwise\text{-}def\ wrap\text{-}def\ inv_1\text{-}def$
using *weight* **by** *auto*

have *bij-exists* $\{\}$ $(\bigcup$ $(\{\} \cup$ *wrap* $\{\}))$
using *bij-betwI′* **unfolding** *bij-exists-def* **by** *fastforce*
from inv_2I[*OF* $*$ - *this*] **have** inv_2 $\{\}$ $\{\}$ $\{\}$ $\{\}$ S_U L_U **by** *auto*
from inv_3I[*OF this*] **show** *?thesis* **by** *blast*
qed

lemma *loop-stepA*:
assumes inv_3 P_1 P_2 B_1 B_2 S L $B_1 = \{\}$ $L = \{\}$ $u \in S$
shows inv_3 P_1 P_2 $\{u\}$ B_2 $(S - \{u\})$ L
proof −
note *invrules* = inv_2E[*OF* inv_3E(*1*)[*OF assms*(*1*)]]
have *WEIGHT*: W B_1 + w $u \leq c$ **using** *invrules*(*5*) *assms*(*2*,*4*) **by** *fastforce*
from *assms*(*4*) **have** $u \in S \cup L$ **by** *blast*
from inv_1-*stepA*[*OF invrules*(*1*) *this WEIGHT*] *assms*(*2*,*3*) **have** *1*: inv_1 P_1 P_2 $\{u\}$ B_2 $(S - \{u\} \cup L)$ **by** *simp*
have *2*: $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup$ *wrap* $\{u\}$. $B \cap L_U \neq \{\}$ **using** *assms*(*3*) **by** *blast*
from inv_2I[*OF 1 2*] *invrules* **have** inv_2 P_1 P_2 $\{u\}$ B_2 $(S - \{u\})$ L **by** *blast*
from inv_3I[*OF this*] **show** *?thesis* **using** inv_3E(*2*)[*OF assms*(*1*)] **.**
qed

lemma *loop-stepB*:
assumes inv_3 P_1 P_2 B_1 B_2 S L $B_1 = \{\}$ $u \in L$
shows inv_3 P_1 P_2 $\{u\}$ B_2 S $(L - \{u\})$
proof −
note *invrules* = inv_2E[*OF* inv_3E(*1*)[*OF assms*(*1*)]]
have *WEIGHT*: W B_1 + w $u \leq c$ **using** *weight invrules*(*5*) *assms*(*2*,*3*) **by** *fastforce*

— This observation follows from the fact that the S and L sets have to be disjoint from each other, and allows us to reuse our proofs of the preservation of inv_1 by simply replacing V with $S \cup L$
have $*$: $S \cup L - \{u\} = S \cup (L - \{u\})$ **using** *invrules*(*5*) *assms*(*3*) **by** *force*
from *assms*(*3*) **have** $u \in S \cup L$ **by** *blast*
from inv_1-*stepA*[*OF invrules*(*1*) *this WEIGHT*] *assms*(*2*) $*$ **have** *1*: inv_1 P_1 P_2 $\{u\}$ B_2 $(S \cup (L - \{u\}))$ **by** *simp*
have $\forall B{\in}P_1$. $B \cap L_U \neq \{\}$ $\{u\} \cap L_U \neq \{\}$ **using** *assms*(*3*) *invrules*(*2*,*5*) **by** *blast+*
then have *2*: $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup$ *wrap* $\{u\}$. $B \cap L_U \neq \{\}$
using *assms*(*3*) **by** (*metis* (*full-types*) *Un-iff empty-iff insert-iff wrap-not-empty*)
from inv_2I[*OF 1 2*] *invrules* **have** inv_2 P_1 P_2 $\{u\}$ B_2 S $(L - \{u\})$ **by** *blast*
from inv_3I[*OF this*] **show** *?thesis* **using** inv_3E(*2*)[*OF assms*(*1*)] **.**
qed

lemma *loop-stepC*:
assumes inv_3 P_1 P_2 B_1 B_2 S L $B_1 \neq \{\}$ $u \in S$ W $B_1 + w(u) \leq c$
shows inv_3 P_1 P_2 $(B_1 \cup \{u\})$ B_2 $(S - \{u\})$ L
proof −
note *invrules* = inv_2E[*OF* inv_3E(*1*)[*OF assms*(*1*)]]

— Same approach, but removing $\{u\}$ from S instead of L
have $*$: $S \cup L - \{u\} = (S - \{u\}) \cup L$ **using** *invrules(5) assms(3)* **by** *force*
from *assms(3)* **have** $u \in S \cup L$ **by** *blast*
from *inv₁-stepA[OF invrules(1) this assms(4)]* $*$ **have** *1*: $inv_1\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (S - \{u\} \cup L)$ **by** *simp*
have $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ B_1.\ B \cap L_U \neq \{\}$ **using** *invrules(2)* **by** *blast*
then have *2*: $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ (B_1 \cup \{u\}).\ B \cap L_U \neq \{\}$
by (*smt Int-insert-left Un-empty-right Un-iff Un-insert-right assms(2) insert-not-empty singletonD singletonI wrap-def*)
from inv_2I*[OF 1 2] invrules* **have** $inv_2\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (S - \{u\})\ L$ **by** *blast*
from inv_3I*[OF this]* **show** *?thesis* **using** inv_3E*(2)[OF assms(1)]* **.**
qed

lemma *loop-stepD*:
assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L\ B_1 \neq \{\}\ u \in S\ W\ B_1 + w(u) > c\ W\ B_2 + w(u) \leq c$
shows $inv_3\ (P_1 \cup wrap\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (S - \{u\})\ L$
proof −
note *invrules* = inv_2E*[OF* inv_3E*(1)[OF assms(1)]]*
have $*$: $S \cup L - \{u\} = (S - \{u\}) \cup L$ **using** *invrules(5) assms(3)* **by** *force*
from *assms(3)* **have** $u \in S \cup L$ **by** *blast*
from *inv₁-stepB[OF invrules(1) this assms(5)]* $*$ **have** *1*: $inv_1\ (P_1 \cup wrap\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (S - \{u\} \cup L)$ **by** *simp*

have *2*: $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup wrap\ B_1 \cup wrap\ \{\}.\ B \cap L_U \neq \{\}$
using *invrules(2)* **unfolding** *wrap-empty* **by** *blast*

from *invrules(3)* **obtain** f **where** *f-def*: $bij\text{-}betw\ f\ P_1\ (\bigcup\ (P_2 \cup wrap\ B_2))\ \forall B{\in}P_1.\ c < W\ B + w\ (f\ B)$ **unfolding** *bij-exists-def* **by** *blast*
have $B_1 \notin P_1$ **using** inv_1E*(3)[OF invrules(1)]* **by** *blast*
have $u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$ **using** inv_1E*(2)[OF invrules(1)] assms(3)* **by** *blast*
then have $(\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\}))) = (\bigcup\ (P_2 \cup wrap\ B_2 \cup \{\{u\}\}))$
by (*metis Sup-empty Un-assoc Union-Un-distrib ccpo-Sup-singleton wrap-empty wrap-not-empty*)
also have ... $= (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **by** *simp*
finally have *UN*: $(\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\}))) = (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **.**
have $wrap\ B_1 = \{B_1\}$ **using** *wrap-not-empty[of* B_1*] assms(2)* **by** *simp*
let $?f = f\ (B_1 := u)$
have *BIJ*: $bij\text{-}betw\ ?f\ (P_1 \cup wrap\ B_1)\ (\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\})))$
unfolding *wrap-empty* ‹$wrap\ B_1 = \{B_1\}$› *UN* **using** *f-def(1)* ‹$B_1 \notin P_1$› ‹$u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$›
by (*metis (no-types, lifting) bij-betw-cong fun-upd-other fun-upd-same notIn-Un-bij-betw3*)
have $c < W\ B_1 + w\ (?f\ B_1)$ **using** *assms(4)* **by** *simp*
then have $(\forall B{\in}P_1 \cup wrap\ B_1.\ c < W\ B + w\ (?f\ B))$
unfolding ‹$wrap\ B_1 = \{B_1\}$› **using** *f-def(2)* **by** *simp*
with *BIJ* **have** $bij\text{-}betw\ ?f\ (P_1 \cup wrap\ B_1)\ (\bigcup\ (P_2 \cup wrap\ (B_2 \cup \{u\})))$
$\wedge\ (\forall B{\in}P_1 \cup wrap\ B_1.\ c < W\ B + w\ (?f\ B))$ **by** *blast*

then have *3*: *bij-exists* $(P_1 \cup \mathit{wrap}\ B_1)\ (\bigcup\ (P_2 \cup \mathit{wrap}\ (B_2 \cup \{u\})))$
unfolding *bij-exists-def* **by** *blast*
from $\mathit{inv}_2 I[\mathit{OF}\ 1\ 2\ 3]$ **have** $\mathit{inv}_2\ (P_1 \cup \mathit{wrap}\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (S - \{u\})\ L$ **using** *invrules*(*4*,*5*) **by** *blast*

from $\mathit{inv}_3 I[\mathit{OF}\ \mathit{this}]$ **show** *?thesis* **using** $\mathit{inv}_3 E(2)[\mathit{OF}\ \mathit{assms}(1)]$ *assms*(*3*) *invrules*(*5*) **by** *blast*
qed

lemma B_2*-at-least-two-objects*:
assumes $\mathit{inv}_3\ P_1\ P_2\ B_1\ B_2\ S\ L\ u \in S\ W\ B_2 + w(u) > c$
shows $2 \leq \mathit{card}\ B_2$
proof (*rule ccontr*, *simp add*: *not-le*)
have *FINITE*: *finite* B_2 **using** $\mathit{inv}_1 E(1)[\mathit{OF}\ \mathit{inv}_2 E(1)[\mathit{OF}\ \mathit{inv}_3 E(1)[\mathit{OF}\ \mathit{assms}(1)]]]$
by (*metis* (*no-types*, *lifting*) *Finite-Set.finite.simps U-Finite Union-Un-distrib bpE*(*3*) *ccpo-Sup-singleton finite-Un wrap-not-empty*)
assume $\mathit{card}\ B_2 < 2$
then consider (*0*) $\mathit{card}\ B_2 = 0$ | (*1*) $\mathit{card}\ B_2 = 1$ **by** *linarith*
then show *False* **proof** *cases*
case *0* **then have** $B_2 = \{\}$ **using** *FINITE* **by** *simp*
then show *?thesis* **using** *assms*(*2*,*3*) $\mathit{inv}_2 E(5)[\mathit{OF}\ \mathit{inv}_3 E(1)[\mathit{OF}\ \mathit{assms}(1)]]$ **by** *force*
next
case *1* **then obtain** v **where** $B_2 = \{v\}$
using *card-1-singletonE* **by** *auto*
with $\mathit{inv}_3 E(2)[\mathit{OF}\ \mathit{assms}(1)]$ **have** $2 * w\ v \leq c$ **using** $\mathit{inv}_2 E(5)[\mathit{OF}\ \mathit{inv}_3 E(1)[\mathit{OF}\ \mathit{assms}(1)]]$ **by** *simp*
moreover from ‹$B_2 = \{v\}$› **have** $W\ B_2 = w\ v$ **by** *simp*
ultimately show *?thesis* **using** *assms*(*2*,*3*) $\mathit{inv}_2 E(5)[\mathit{OF}\ \mathit{inv}_3 E(1)[\mathit{OF}\ \mathit{assms}(1)]]$ **by** *force*
qed
qed

lemma *loop-stepE*:
assumes $\mathit{inv}_3\ P_1\ P_2\ B_1\ B_2\ S\ L\ B_1 \neq \{\}\ u \in S\ W\ B_1 + w(u) > c\ W\ B_2 + w(u) > c$
shows $\mathit{inv}_3\ (P_1 \cup \mathit{wrap}\ B_1)\ (P_2 \cup \mathit{wrap}\ B_2)\ \{\}\ \{u\}\ (S - \{u\})\ L$
proof −
note $\mathit{invrules} = \mathit{inv}_2 E[\mathit{OF}\ \mathit{inv}_3 E(1)[\mathit{OF}\ \mathit{assms}(1)]]$
have $*$: $S \cup L - \{u\} = (S - \{u\}) \cup L$ **using** *invrules*(*5*) *assms*(*3*) **by** *force*
from *assms*(*3*) **have** $u \in S \cup L$ **by** *blast*
from inv_1*-stepC*[*OF invrules*(*1*) *this*] $*$ **have** *1*: $\mathit{inv}_1\ (P_1 \cup \mathit{wrap}\ B_1)\ (P_2 \cup \mathit{wrap}\ B_2)\ \{\}\ \{u\}\ (S - \{u\} \cup L)$ **by** *simp*

have *2*: $L \neq \{\} \Longrightarrow \forall B{\in}P_1 \cup \mathit{wrap}\ B_1 \cup \mathit{wrap}\ \{\}.\ B \cap L_U \neq \{\}$
using *invrules*(*2*) **unfolding** *wrap-empty* **by** *blast*

from *invrules*(*3*) **obtain** f **where** *f-def*: $\mathit{bij\text{-}betw}\ f\ P_1\ (\bigcup\ (P_2 \cup \mathit{wrap}\ B_2))\ \forall B{\in}P_1.\ c < W\ B + w\ (f\ B)$ **unfolding** *bij-exists-def* **by** *blast*

have $B_1 \notin P_1$ **using** $inv_1E(3)[OF\ invrules(1)]$ **by** *blast*
have $u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$ **using** $inv_1E(2)[OF\ invrules(1)]$ $assms(3)$ **by** *blast*
have $(\bigcup\ (P_2 \cup wrap\ B_2 \cup wrap\ \{u\})) = (\bigcup\ (P_2 \cup wrap\ B_2 \cup \{\{u\}\}))$ **unfolding** *wrap-def* **by** *simp*
also have ... $= (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **by** *simp*
finally have UN: $(\bigcup\ (P_2 \cup wrap\ B_2 \cup wrap\ \{u\})) = (\bigcup\ (P_2 \cup wrap\ B_2)) \cup \{u\}$ **.**
have $wrap\ B_1 = \{B_1\}$ **using** *wrap-not-empty*$[of\ B_1]$ $assms(2)$ **by** *simp*
let $?f = f\ (B_1 := u)$
have BIJ: *bij-betw* $?f\ (P_1 \cup wrap\ B_1)\ (\bigcup\ (P_2 \cup wrap\ B_2 \cup wrap\ \{u\}))$
unfolding *wrap-empty* ‹$wrap\ B_1 = \{B_1\}$› UN **using** *f-def*(1) ‹$B_1 \notin P_1$› ‹$u \notin (\bigcup\ (P_2 \cup wrap\ B_2))$›
by (*metis* (*no-types*, *lifting*) *bij-betw-cong fun-upd-other fun-upd-same notIn-Un-bij-betw3*)
have $c < W\ B_1 + w\ (?f\ B_1)$ **using** $assms(4)$ **by** *simp*
then have $(\forall B{\in}P_1 \cup wrap\ B_1.\ c < W\ B + w\ (?f\ B))$
unfolding ‹$wrap\ B_1 = \{B_1\}$› **using** *f-def*(2) **by** *simp*
with BIJ **have** *bij-betw* $?f\ (P_1 \cup wrap\ B_1)\ (\bigcup\ (P_2 \cup wrap\ B_2 \cup wrap\ \{u\}))$
$\wedge (\forall B{\in}P_1 \cup wrap\ B_1.\ c < W\ B + w\ (?f\ B))$ **by** *blast*
then have 3: *bij-exists* $(P_1 \cup wrap\ B_1)\ (\bigcup\ (P_2 \cup wrap\ B_2 \cup wrap\ \{u\}))$
unfolding *bij-exists-def* **by** *blast*

have 4: $2 * card\ (P_2 \cup wrap\ B_2) \leq card\ (\bigcup\ (P_2 \cup wrap\ B_2))$
proof −
note $bprules = bpE[OF\ inv_1E(1)[OF\ invrules(1)]]$
have *pairwise disjnt* $(P_2 \cup wrap\ B_2)$
using *bprules*(1) *pairwise-subset* **by** *blast*
moreover have $B_2 \notin P_2$ **using** $inv_1E(4)[OF\ invrules(1)]$ **by** *simp*
ultimately have $DISJNT$: $\bigcup P_2 \cap B_2 = \{\}$
by (*auto*, *metis* (*no-types*, *opaque-lifting*) *sup-bot.right-neutral Un-insert-right disjnt-iff mk-disjoint-insert pairwise-insert wrap-Un*)
have *finite* $(\bigcup P_2)$ **using** *U-Finite bprules*(3) **by** *auto*
have *finite* B_2 **using** $inv_1E(1)[OF\ invrules(1)]$ *bp-bins-finite wrap-not-empty* **by** *blast*

have $2 * card\ (P_2 \cup wrap\ B_2) \leq 2 * (card\ P_2 + card\ (wrap\ B_2))$
using *card-Un-le*$[of\ P_2$ ‹$wrap\ B_2$›$]$ **by** *simp*
also have ... $\leq 2 * card\ P_2 + 2$ **using** *wrap-card* **by** *auto*
also have ... $\leq card\ (\bigcup\ P_2) + 2$ **using** *invrules*(4) **by** *simp*
also have ... $\leq card\ (\bigcup\ P_2) + card\ B_2$ **using** B_2*-at-least-two-objects*$[OF\ assms(1,3,5)]$ **by** *simp*
also have ... $= card\ (\bigcup\ (P_2 \cup \{B_2\}))$ **using** $DISJNT$ *card-Un-disjoint*$[OF$ ‹*finite* $(\bigcup P_2)$› ‹*finite* B_2›$]$ **by** (*simp add*: *Un-commute*)
also have ... $= card\ (\bigcup\ (P_2 \cup wrap\ B_2))$ **by** (*cases* ‹$B_2 = \{\}$›) *auto*
finally show *?thesis* **.**
qed
from $inv_2I[OF\ 1\ 2\ 3\ 4]$ **have** $inv_2\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (S - \{u\})\ L$
using *invrules*(5) **by** *blast*

from $inv_3I[OF\ this]$ **show** *?thesis* **using** *assms(3) invrules(5)* **by** *blast*
qed

The bin packing algorithm as it is proposed on page 78 of the article [2]. P will not only be a correct solution of the bin packing problem, but the amount of bins will be a lower bound for *3 / 2* of the amount of bins of any correct solution Q, and thus guarantee an approximation factor of *3 / 2* for the optimum.

lemma *bp-approx*:
$VARS\ P\ P_1\ P_2\ B_1\ B_2\ V\ S\ L\ u$
$\{True\}$
$S := \{\};\ L{:=}\ \{\};\ V := U;$
$WHILE\ V \neq \{\}\ INV\ \{V \subseteq U \wedge S = \{u \in U - V.\ w(u) \leq c\ /\ 2\} \wedge L = \{u \in U - V.\ c\ /\ 2 < w(u)\}\}\ DO$
$u := (SOME\ u.\ u \in V);$
$IF\ w(u) \leq c\ /\ 2$
$THEN\ S := S \cup \{u\}$
$ELSE\ L := L \cup \{u\}\ FI;$
$V := V - \{u\}$
$OD;$
$P_1 := \{\};\ P_2 := \{\};\ B_1 := \{\};\ B_2 := \{\};$
$WHILE\ S \neq \{\}\ INV\ \{inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L\}\ DO$
$IF\ B_1 \neq \{\}$
$THEN\ u := (SOME\ u.\ u \in S);\ S := S - \{u\}$
$ELSE\ IF\ L \neq \{\}$
$THEN\ u := (SOME\ u.\ u \in L);\ L := L - \{u\}$
$ELSE\ u := (SOME\ u.\ u \in S);\ S := S - \{u\}\ FI\ FI;$
$IF\ W(B_1) + w(u) \leq c$
$THEN\ B_1 := B_1 \cup \{u\}$
$ELSE\ IF\ W(B_2) + w(u) \leq c$
$THEN\ B_2 := B_2 \cup \{u\}$
$ELSE\ P_2 := P_2 \cup wrap\ B_2;\ B_2 := \{u\}\ FI;$
$P_1 := P_1 \cup wrap\ B_1;\ B_1 := \{\}\ FI$
$OD;$
$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2;\ V := L;$
$WHILE\ V \neq \{\}$
$INV\ \{S = \{\} \wedge inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L \wedge V \subseteq L \wedge P = P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}|v.\ v \in L - V\}\}\ DO$
$u := (SOME\ u.\ u \in V);\ P := P \cup \{\{u\}\};\ V := V - \{u\}$
OD
$\{bp\ P \wedge (\forall Q.\ bp\ Q \longrightarrow card\ P \leq 3\ /\ 2 * card\ Q)\}$
proof (*vcg*, *goal-cases*)
case $(1\ P\ P_1\ P_2\ B_1\ B_2\ V\ S\ L\ u)$
then show *?case* **by** *blast*
next
case $(2\ P\ P_1\ P_2\ B_1\ B_2\ V\ S\ L\ u)$
then show *?case* **by** (*auto simp*: *some-in-eq*)
next

case (*3 P* P_1 P_2 B_1 B_2 *V S L u*)
then show *?case* **using** *loop-init* **by** *force*
next
case (*4 P* P_1 P_2 B_1 B_2 *V S L u*)
then have *INV*: inv_3 P_1 P_2 B_1 B_2 *S L* **..**
let *?s* = *SOME u. u* ∈ *S*
let *?l* = *SOME u. u* ∈ *L*
note *SL-def* = $inv_2E(5)$[*OF* $inv_3E(1)$[*OF INV*]]
have *LIN*: *L* ≠ {} ⟹ *?l* ∈ *L* **using** *some-in-eq* **by** *metis*
then have *LWEIGHT*: *L* ≠ {} ⟹ *w ?l* ≤ *c* **using** *weight SL-def* **by** *blast*
from *4* **have** *S* ≠ {} **..**
then have *IN*: *?s* ∈ *S* **using** *some-in-eq* **by** *metis*
then have *w ?s* ≤ *c* **using** *SL-def* **by** *auto*
then show *?case*
using *LWEIGHT loop-stepA*[*OF INV - - IN*] *loop-stepB*[*OF INV - LIN*] *loop-stepC*[*OF INV - IN*]
and *loop-stepD*[*OF INV - IN*] *loop-stepE*[*OF INV - IN*] **by** (*cases* ‹B_1 = {}›, *cases* ‹*L* = {}›) *auto*
next
case (*5 P* P_1 P_2 B_1 B_2 *V S L u*)
then show *?case* **by** *blast*
next
case (*6 P* P_1 P_2 B_1 B_2 *V S L u*)
then have ∗: (*SOME u. u* ∈ *V*) ∈ *V* (*SOME u. u* ∈ *V*) ∈ *L* **by** (*auto simp add*: *some-in-eq*)
then have P_1 ∪ *wrap* B_1 ∪ P_2 ∪ *wrap* B_2 ∪ {{*v*} |*v. v* ∈ *L* − (*V* − {*SOME u. u* ∈ *V*})}
= P_1 ∪ *wrap* B_1 ∪ P_2 ∪ *wrap* B_2 ∪ {{*v*} |*v. v* ∈ *L* − *V* ∪ {*SOME u. u* ∈ *V*}}
by *blast*
with *6* ∗ **show** *?case* **by** *blast*
next
case (*7 P* P_1 P_2 B_1 B_2 *V S L u*)
then have ∗: inv_2 P_1 P_2 B_1 B_2 *S L*
using $inv_3E(1)$ **by** *blast*
from $inv_1E(1)$[*OF* $inv_2E(1)$[*OF* ∗]] *7*
have *bp P* **by** *fastforce*
with *bin-packing-lower-bound-card*[*OF* - ∗] *7*
show *?case* **by** *fastforce*
qed

end

end

6 Center Selection

theory *Center-Selection*
imports *Complex-Main HOL−Hoare.Hoare-Logic*

begin

The Center Selection (or metric k-center) problem. Given a set of *sites* S in a metric space, find a subset $C \subseteq S$ that minimizes the maximal distance from any $s \in S$ to some $c \in C$. This theory presents a verified 2-approximation algorithm. It is based on Section 11.2 in the book by Kleinberg and Tardos [4]. In contrast to the proof in the book, our proof is a standard invariant proof.

locale *Center-Selection* =
  fixes S :: ('a :: *metric-space*) *set*
    and k :: *nat*
  assumes *finite-sites*: *finite* S
  and *non-empty-sites*: $S \neq \{\}$
and *non-zero-k*: $k > 0$
begin

definition *distance* :: ('a::*metric-space*) *set* ⇒ ('a::*metric-space*) ⇒ *real* **where**
distance C s = *Min* (*dist* s ' C)

definition *radius* :: ('a :: *metric-space*) *set* ⇒ *real* **where**
radius C = *Max* (*distance* C ' S)

lemma *distance-mono*:
assumes $C_1 \subseteq C_2$ **and** $C_1 \neq \{\}$ **and** *finite* C_2
shows *distance* C_1 $s \geq$ *distance* C_2 s
by (*simp add*: *Min.subset-imp assms distance-def image-mono*)

lemma *finite-distances*: *finite* (*distance* C ' S)
  using *finite-sites* **by** *simp*

lemma *non-empty-distances*: *distance* C ' $S \neq \{\}$
  using *non-empty-sites* **by** *simp*

lemma *radius-contained*: *radius* $C \in$ *distance* C ' S
  using *finite-distances non-empty-distances Max-in radius-def* **by** *simp*

lemma *radius-def2*: $\exists s \in S$. *distance* C s = *radius* C
  using *radius-contained image-iff* **by** *metis*

lemma *dist-lemmas-aux*:
  assumes *finite* C
    and $C \neq \{\}$
  shows *finite* (*dist* s ' C)
   and *finite* (*dist* s ' C) ⟹ *distance* C $s \in$ *dist* s ' C
   and *distance* C $s \in$ *dist* s ' C ⟹ $\exists c \in C$. *dist* s c = *distance* C s
and $\exists c \in C$. *dist* s c = *distance* C s ⟹ *distance* C $s \geq 0$
proof
  show *finite* C **using** *assms*(*1*) **by** *simp*
next

assume *finite* (*dist s ‘ C*)
then show *distance C s* $\in$ *dist s ‘ C* **using** *distance-def eq-Min-iff assms*(*2*) **by** *blast*
next
assume *distance C s* $\in$ *dist s ‘ C*
then show $\exists\, c \in C$. *dist s c* = *distance C s* **by** *auto*
next
assume $\exists\, c \in C$. *dist s c* = *distance C s*
then show *distance C s* $\geq$ *0* **by** (*metis zero-le-dist*)
qed

lemma *dist-lemmas*:
assumes *finite C*
and $C \neq \{\}$
shows *finite* (*dist s ‘ C*)
and *distance C s* $\in$ *dist s ‘ C*
and $\exists\, c \in C$. *dist s c* = *distance C s*
and *distance C s* $\geq$ *0*
using *dist-lemmas-aux assms* **by** *auto*

lemma *radius-max-prop*: ($\forall\, s \in S$. *distance C s* $\leq r$) $\Longrightarrow$ (*radius C* $\leq r$)
by (*metis image-iff radius-contained*)

lemma *dist-ins*:
assumes $\forall\, c_1 \in C.\ \forall\, c_2 \in C.\ c_1 \neq c_2 \longrightarrow x <$ *dist* $c_1\ c_2$
and *distance C s* $> x$
and *finite C*
and $C \neq \{\}$
shows $\forall\, c_1 \in (C \cup \{s\}).\ \forall\, c_2 \in (C \cup \{s\}).\ c_1 \neq c_2 \longrightarrow x <$ *dist* $c_1\ c_2$
proof (*rule+*)
fix $c_1\ c_2$
assume *local-assms*: $c_1 \in C \cup \{s\}$ $c_2 \in C \cup \{s\}$ $c_1 \neq c_2$
then have $c_1 \in C \ \wedge c_2 \in C \ \vee c_1 \in C \ \wedge c_2 \in \{s\} \vee c_2 \in C \ \wedge c_1 \in \{s\} \vee c_1 \in \{s\} \wedge c_2 \in \{s\}$ **by** *auto*
then show $x <$ *dist* $c_1\ c_2$
proof (*elim disjE*)
assume $c_1 \in C \ \wedge c_2 \in C$
then show *?thesis* **using** *assms*(*1*) *local-assms*(*3*) **by** *simp*
next
assume *case-assm*: $c_1 \in C \wedge c_2 \in \{s\}$
have $x <$ *distance C* c_2 **using** *assms*(*2*) *case-assm* **by** *simp*
also have ... $\leq$ *dist* $c_2\ c_1$
using *Min.coboundedI distance-def assms*(*3*,*4*) *dist-lemmas*(*1*, *2*) *case-assm* **by** *simp*
also have ... = *dist* $c_1\ c_2$ **using** *dist-commute* **by** *metis*
finally show *?thesis* **.**
next
assume *case-assm*: $c_2 \in C \wedge c_1 \in \{s\}$
have $x <$ *distance C* c_1 **using** *assms*(*2*) *case-assm* **by** *simp*

also have ... $\leq$ *dist* c_1 c_2
using *Min.coboundedI distance-def assms*(*3*,*4*) *dist-lemmas*(*1*, *2*) *case-assm* **by** *simp*
finally show *?thesis* **.**
next
assume $c_1 \in \{s\} \wedge c_2 \in \{s\}$
then have *False* **using** *local-assms* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

6.1 A Preliminary Algorithm and Proof

This subsection verifies an auxiliary algorithm by Kleinberg and Tardos. Our proof of the main algorithm does not does not rely on this auxiliary algorithm at all but we do reuse part off its invariant proof later on.

definition *inv* :: ($'a$ :: *metric-space*) *set* $\Rightarrow$ ($'a$ :: *metric-space set*) $\Rightarrow$ *real* $\Rightarrow$ *bool*
where
inv S' C r =
$((\forall s \in (S - S').\ \mathit{distance}\ C\ s \leq 2{*}r) \wedge S' \subseteq S \wedge C \subseteq S \wedge$
$(\forall c \in C.\ \forall s \in S'.\ S' \neq \{\} \longrightarrow \mathit{dist}\ c\ s > 2 * r) \wedge (S' = S \vee C \neq \{\}) \wedge$
$(\forall c_1 \in C.\ \forall c_2 \in C.\ c_1 \neq c_2 \longrightarrow \mathit{dist}\ c_1\ c_2 > 2 * r))$

lemma *inv-init*: *inv* S $\{\}$ r
unfolding *inv-def non-empty-sites* **by** *simp*
lemma *inv-step*:
assumes $S' \neq \{\}$
and *IH*: *inv* S' C r
defines[*simp*]: $s \equiv (\mathit{SOME}\ s.\ s \in S')$
shows *inv* $(S' - \{s'\ .\ s' \in S' \wedge \mathit{dist}\ s\ s' \leq 2{*}r\})$ $(C \cup \{s\})$ r
proof −
have *s-def*: $s \in S'$ **using** *assms*(*1*) *some-in-eq* **by** *auto*

have *finite* $(C \cup \{s\})$ **using** *IH finite-subset*[*OF - finite-sites*] **by** (*simp add*: *inv-def*)

moreover

have $(\forall s' \in (S - (S' - \{s'\ .\ s' \in S' \wedge \mathit{dist}\ s\ s' \leq 2{*}r\})).\ \mathit{distance}\ (C \cup \{s\})\ s' \leq 2{*}r)$
proof
fix s''
assume $s'' \in S - (S' - \{s'\ .\ s' \in S' \wedge \mathit{dist}\ s\ s' \leq 2{*}r\})$
then have $s'' \in S - S' \vee s'' \in \{s'\ .\ s' \in S' \wedge \mathit{dist}\ s\ s' \leq 2{*}r\}$ **by** *simp*
then show *distance* $(C \cup \{s\})$ $s'' \leq 2 * r$
proof (*elim disjE*)
assume *local-assm*: $s'' \in S - S'$
have $S' = S \vee C \neq \{\}$ **using** *IH* **by** (*simp add*: *inv-def*)
then show *?thesis*

proof (*elim disjE*)
assume $S' = S$
then have $s'' \in \{\}$ **using** *local-assm* **by** *simp*
then show *?thesis* **by** *simp*
next
assume *C-not-empty*: $C \neq \{\}$
have *finite C* **using** *IH finite-subset*[*OF - finite-sites*] **by** (*simp add*: *inv-def*)
then have *distance* $(C \cup \{s\})$ $s'' \leq$ *distance* C s''
using *distance-mono C-not-empty* **by** (*meson Un-upper1 calculation*)
also have ... $\leq 2 * r$ **using** *IH local-assm inv-def* **by** *simp*
finally show *?thesis* **.**
qed
next
assume *local-assm*: $s'' \in \{s' .\ s' \in S' \land dist\ s\ s' \leq 2{*}r\}$
then have *distance* $(C \cup \{s\})$ $s'' \leq dist\ s''\ s$
using *Min.coboundedI distance-def dist-lemmas calculation* **by** *auto*
also have ... $\leq 2 * r$ **using** *local-assm* **by** (*smt dist-self dist-triangle2 mem-Collect-eq*)
finally show *?thesis* **.**
qed
qed

moreover

have $S' - \{s' .\ s' \in S' \land dist\ s\ s' \leq 2{*}r\} \subseteq S$ **using** *IH* **by** (*auto simp*: *inv-def*)

moreover
{
have $s \in S$ **using** *IH inv-def s-def* **by** *auto*
then have $C \cup \{s\} \subseteq S$ **using** *IH* **by** (*simp add*: *inv-def*)
}
moreover

have $(\forall\, c{\in}C \cup \{s\}.\ \forall\, c_2{\in}C \cup \{s\}.\ c \neq c_2 \longrightarrow 2 * r < dist\ c\ c_2)$
proof (*rule+*)
fix c_1 c_2
assume *local-assms*: $c_1 \in C \cup \{s\}$ $c_2 \in C \cup \{s\}$ $c_1 \neq c_2$
then have $(c_1 \in C \land c_2 \in C) \lor (c_1 = s \land c_2 \in C) \lor (c_1 \in C \land c_2 = s) \lor (c_1 = s \land c_2 = s)$
using *assms* **by** *auto*
then show $2 * r < dist\ c_1\ c_2$
proof (*elim disjE*)
assume $c_1 \in C \land c_2 \in C$
then show $2 * r < dist\ c_1\ c_2$ **using** *IH inv-def local-assms* **by** *simp*
next
assume *case-assm*: $c_1 = s \land c_2 \in C$
have $(\forall\, c \in C.\ \forall\, s{\in}S'.\ S' \neq \{\} \longrightarrow 2 * r < dist\ c\ s)$ **using** *IH inv-def* **by** *simp*
then show *?thesis* **by** (*smt case-assm s-def assms*(*1*) *dist-self dist-triangle3*

singletonD)
  next
    assume *case-assm*: $c_1 \in C \wedge c_2 = s$
    have $(\forall c \in C. \forall s{\in}S'. S' \neq \{\} \longrightarrow 2 * r < dist\ c\ s)$ **using** *IH inv-def* **by** *simp*
    then show *?thesis* **by** (*smt case-assm s-def assms*(*1*) *dist-self dist-triangle3 singletonD*)
  next
    assume $c_1 = s \wedge c_2 = s$
    then have *False* **using** *local-assms*(*3*) **by** *simp*
    then show *?thesis* **by** *simp*
  qed
qed

moreover

have $(\forall c{\in}C \cup \{s\}. \forall s'' \in S' - \{s' \in S'. dist\ s\ s' \leq 2 * r\}.$
  $S' - \{s' \in S'. dist\ s\ s' \leq 2 * r\} \neq \{\} \longrightarrow 2 * r < dist\ c\ s'')$
  using *IH inv-def* **by** *fastforce*

moreover

have $(S' - \{s' \in S'. dist\ s\ s' \leq 2 * r\} = S \vee C \cup \{s\} \neq \{\})$ **by** *simp*

ultimately show *?thesis* **unfolding** *inv-def* **by** *blast*
qed

lemma *inv-last-1*:
  assumes $\forall s \in (S - S'). distance\ C\ s \leq 2{*}r$
    and $S' = \{\}$
  shows $radius\ C \leq 2{*}r$
  by (*metis Diff-empty assms image-iff radius-contained*)

lemma *inv-last-2*:
  assumes *finite C*
  and $card\ C > n$
  and $C \subseteq S$
  and $\forall c_1 \in C. \forall c_2 \in C. c_1 \neq c_2 \longrightarrow dist\ c_1\ c_2 > 2{*}r$
  shows $\forall C'. card\ C' \leq n \wedge card\ C' > 0 \longrightarrow radius\ C' > r$ (**is** *?P*)
proof (*rule ccontr*)
  assume ¬ *?P*
  then obtain C' **where** *card-C'*: $card\ C' \leq n \wedge card\ C' > 0$ **and** *radius-C'*: $radius\ C' \leq r$ **by** *auto*
  have $\forall c \in C. (\exists c'. c' \in C' \wedge dist\ c\ c' \leq r)$
  proof
    fix *c*
    assume $c \in C$
    then have $c \in S$ **using** *assms*(*3*) **by** *blast*
    then have $distance\ C'\ c \leq radius\ C'$ **using** *finite-distances* **by** (*simp add*:

radius-def)
  then have *distance C′ c ≤ r* **using** *radius-C′* **by** *simp*
  then show *∃ c′. c′ ∈ C′ ∧ dist c c′ ≤ r* **using** *dist-lemmas*
   by (*metis card-C′ card-gt-0-iff*)
 qed
 then obtain *f* **where** *f*: *∀ c∈C. f c ∈ C′ ∧ dist c (f c) ≤ r* **by** *metis*
 have *¬inj-on f C*
 proof
  assume *inj-on f C*
  then have *card C′ ≥ card C* **using** ‹*inj-on f C*› *card-inj-on-le card-ge-0-finite card-C′ f* **by** *blast*
  then show *False* **using** *card-C′* ‹*n < card C*› **by** *linarith*
 qed
 then obtain *c1 c2* **where** *defs*: *c1 ∈ C ∧ c2 ∈ C ∧ c1 ≠ c2 ∧ f c1 = f c2* **using** *inj-on-def* **by** *blast*
 then have ∗: *dist c1 (f c1) ≤ r ∧ dist c2 (f c1) ≤ r* **using** *f* **by** *auto*

 have *2 ∗ r < dist c1 c2* **using** *assms defs* **by** *simp*
 also have ... *≤ dist c1 (f c1) + dist (f c1) c2* **by**(*rule dist-triangle*)
 also have ... *= dist c1 (f c1) + dist c2 (f c1)* **using** *dist-commute* **by** *simp*
 also have ... *≤ 2 ∗ r* **using** ∗ **by** *simp*
 finally show *False* **by** *simp*
qed

lemma *inv-last*:
 assumes *inv {} C r*
 shows (*card C ≤ k ⟶ radius C ≤ 2∗r*) ∧ (*card C > k ⟶* (*∀ C′. card C′ > 0 ∧ card C′ ≤ k ⟶ radius C′ > r*))
 using *assms inv-def inv-last-1 inv-last-2 finite-subset*[*OF - finite-sites*] **by** *auto*

theorem *Center-Selection-r*:
 VARS (*S′* :: (*′a* :: *metric-space*) *set*) (*C* :: (*′a* :: *metric-space*) *set*) (*r* :: *real*) (*s* :: *′a*)
 {*True*}
 S′ := *S*;
 C := {};
 WHILE S′ ≠ {} INV {inv S′ C r} DO
  s := (*SOME s. s ∈ S′*);
  C := *C ∪ {s}*;
  S′ := *S′ − {s′ . s′ ∈ S′ ∧ dist s s′ ≤ 2∗r}*
  OD
 {(*card C ≤ k ⟶ radius C ≤ 2∗r*) ∧ (*card C > k ⟶* (*∀ C′. card C′ > 0 ∧ card C′ ≤ k ⟶ radius C′ > r*))}
proof (*vcg*, *goal-cases*)
 case (*1 S′ C r*)
 then show *?case* **using** *inv-init* **by** *simp*
next
 case (*2 S′ C r*)
 then show *?case* **using** *inv-step* **by** *simp*

next
  case $(3\ S'\ C\ r)$
  then show *?case* **using** *inv-last* **by** *blast*
qed

6.2 The Main Algorithm

definition *invar* :: $('a$ :: *metric-space*$)$ *set* $\Rightarrow$ *bool* **where**
invar $C = (C \neq \{\} \wedge$ *card* $C \leq k \wedge C \subseteq S \wedge$
  $(\forall\, C'.\ (\forall\, c_1 \in C.\ \forall\, c_2 \in C.\ c_1 \neq c_2 \longrightarrow$ *dist* $c_1\ c_2 > 2 *$ *radius* $C')$
    $\vee\ (\forall\, s \in S.$ *distance* $C\ s \leq 2 *$ *radius* $C')))$

abbreviation *some* **where** *some* $A \equiv$ (*SOME* $s.\ s \in A$)

lemma *invar-init*: *invar* $\{$*some* $S\}$
proof $-$
  let *?s* = *some* S
  have *s-in-S*: *?s* $\in S$ **using** *some-in-eq non-empty-sites* **by** *blast*

  have $\{$*?s*$\} \neq \{\}$ **by** *simp*

  moreover

  have $\{$*SOME* $s.\ s \in S\} \subseteq S$ **using** *s-in-S* **by** *simp*

  moreover

  have *card* $\{$*SOME* $s.\ s \in S\} \leq k$ **using** *non-zero-k* **by** *simp*

  ultimately show *?thesis* **by** (*auto simp*: *invar-def*)
qed

abbreviation *furthest-from* **where**
furthest-from $C \equiv$ (*SOME* $s.\ s \in S \wedge$ *distance* $C\ s =$ *Max* (*distance* C ‘ S))

lemma *invar-step*:
assumes *invar* C
and *card* $C < k$
shows *invar* $(C \cup \{$*furthest-from* $C\})$
proof $-$
  have *furthest-from-C-props*: *furthest-from* $C \in S \wedge$ *distance* C (*furthest-from* C) = *radius* C
    using *someI-ex*[*of* $\lambda x.\ x \in S \wedge$ *distance* $C\ x =$ *radius* C] *radius-def2 radius-def* **by** *auto*
  have *C-props*: *finite* $C \wedge C \neq \{\}$
    using *finite-subset*[*OF* - *finite-sites*] *assms*(*1*) **unfolding** *invar-def* **by** *blast*
  {
    have *card* $(C \cup \{$*furthest-from* $C\}) \leq$ *card* $C + 1$
      using *assms*(*1*) *C-props* **unfolding** *invar-def* **by** (*simp add*: *card-insert-if*)

then have *card* $(C \cup \{\mathit{furthest\text{-}from}\ C\}) < k + \mathit{1}$ **using** *assms*$(\mathit{2})$ **by** *simp*
then have *card* $(C \cup \{\mathit{furthest\text{-}from}\ C\}) \leq k$ **by** *simp*
}
moreover

have $C \cup \{\mathit{furthest\text{-}from}\ C\} \neq \{\}$ **by** *simp*

moreover

have $(C \cup \{\mathit{furthest\text{-}from}\ C\}) \subseteq S$ **using** *assms*$(\mathit{1})$ *furthest-from-C-props* **unfolding** *invar-def* **by** *simp*

moreover

have $\forall C'. (\forall s \in S.\ \mathit{distance}\ (C \cup \{\mathit{furthest\text{-}from}\ C\})\ s \leq \mathit{2} * \mathit{radius}\ C')$
$\vee (\forall c_1 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ \forall c_2 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ c_1 \neq c_2 \longrightarrow \mathit{2} * \mathit{radius}\ C' < \mathit{dist}\ c_1\ c_2)$
proof
fix C'
have *distance* C $(\mathit{furthest\text{-}from}\ C) > \mathit{2} * \mathit{radius}\ C' \vee \mathit{distance}\ C\ (\mathit{furthest\text{-}from}\ C) \leq \mathit{2} * \mathit{radius}\ C'$ **by** *auto*
then show $(\forall s \in S.\ \mathit{distance}\ (C \cup \{\mathit{furthest\text{-}from}\ C\})\ s \leq \mathit{2} * \mathit{radius}\ C')$
$\vee (\forall c_1 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ \forall c_2 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ c_1 \neq c_2 \longrightarrow \mathit{2} * \mathit{radius}\ C' < \mathit{dist}\ c_1\ c_2)$
proof (*elim disjE*)
assume *asm*: *distance* C $(\mathit{furthest\text{-}from}\ C) > \mathit{2} * \mathit{radius}\ C'$
then have $\neg(\forall s \in S.\ \mathit{distance}\ C\ s \leq \mathit{2} * \mathit{radius}\ C')$ **using** *furthest-from-C-props* **by** *force*
then have *IH*: $\forall c_1 \in C.\ \forall c_2 \in C.\ c_1 \neq c_2 \longrightarrow \mathit{2} * \mathit{radius}\ C' < \mathit{dist}\ c_1\ c_2$
using *assms*$(\mathit{1})$ **unfolding** *invar-def* **by** *blast*
have $(\forall c_1 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ (\forall c_2 \in C \cup \{\mathit{furthest\text{-}from}\ C\}.\ c_1 \neq c_2 \longrightarrow \mathit{2} * \mathit{radius}\ C' < \mathit{dist}\ c_1\ c_2))$
using *dist-ins*[*of* C *2* $*$ *radius* C' *furthest-from* C] *IH C-props asm* **by** *simp*
then show *?thesis* **by** *simp*
next
assume *main-assm*: $\mathit{2} * \mathit{radius}\ C' \geq \mathit{distance}\ C\ (\mathit{furthest\text{-}from}\ C)$
have $(\forall s \in S.\ \mathit{distance}\ (C \cup \{\mathit{furthest\text{-}from}\ C\})\ s \leq \mathit{2} * \mathit{radius}\ C')$
proof
fix s
assume *local-assm*: $s \in S$
then show *distance* $(C \cup \{\mathit{furthest\text{-}from}\ C\})\ s \leq \mathit{2} * \mathit{radius}\ C'$
proof $-$
have *distance* $(C \cup \{\mathit{furthest\text{-}from}\ C\})\ s \leq \mathit{distance}\ C\ s$
using *distance-mono*[*of* C $C \cup \{\mathit{furthest\text{-}from}\ C\}$] *C-props* **by** *auto*
also have $\ldots \leq \mathit{distance}\ C\ (\mathit{furthest\text{-}from}\ C)$
using *Max.coboundedI local-assm finite-distances radius-def furthest-from-C-props* **by** *auto*
also have $\ldots \leq \mathit{2} * \mathit{radius}\ C'$ **using** *main-assm* **by** *simp*
finally show *?thesis* **.**

    qed
   qed
   then show *?thesis* **by** *blast*
  qed
 qed

 ultimately show *?thesis* **unfolding** *invar-def* **by** *blast*
qed

lemma *invar-last*:
assumes *invar* C **and** $\neg$*card* $C < k$
shows *card* $C = k$ **and** *card* $C' > 0 \wedge$ *card* $C' \leq k \longrightarrow$ *radius* $C \leq 2 *$ *radius* C'
proof −
 show *card* $C = k$ **using** *assms*(*1*, *2*) **unfolding** *invar-def* **by** *simp*
next
 have *C-props*: *finite* $C \wedge C \neq \{\}$ **using** *finite-sites assms*(*1*) **unfolding** *invar-def* **by** (*meson finite-subset*)
 show *card* $C' > 0 \wedge$ *card* $C' \leq k \longrightarrow$ *radius* $C \leq 2 *$ *radius* C'
 proof (*rule impI*)
  assume C'*-assms*: $0 <$ *card* (C' :: *'a set*) $\wedge$ *card* $C' \leq k$
  let *?r* = *radius* C'
  have ($\forall c_1 \in C. \forall c_2 \in C.\ c_1 \neq c_2 \longrightarrow 2 *$ *?r* $<$ *dist* $c_1\ c_2$) $\vee$ ($\forall s \in S.$ *distance* $C\ s \leq 2 *$ *?r*)
   using *assms*(*1*) **unfolding** *invar-def* **by** *simp*
  then show *radius* $C \leq 2 *$ *?r*
  proof
   assume *case-assm*: $\forall c_1{\in}C. \forall c_2{\in}C.\ c_1 \neq c_2 \longrightarrow 2 *$ *?r* $<$ *dist* $c_1\ c_2$
   obtain s **where** *s-def*: *radius* $C =$ *distance* $C\ s \wedge s \in S$ **using** *radius-def2* **by** *metis*
   show *?thesis*
   proof (*rule ccontr*)
    assume *contr-assm*: $\neg$ *radius* $C \leq 2 *$ *?r*
    then have *s-prop*: *distance* $C\ s > 2 *$ *?r* **using** *s-def* **by** *simp*
    then have ‹$\forall c_1 \in C \cup \{s\}. \forall c_2 \in C \cup \{s\}.\ c_1 \neq c_2 \longrightarrow$ *dist* $c_1\ c_2 > 2 *$ *?r*›
     using *C-props dist-ins*[*of C 2*∗*?r s*] *case-assm* **by** *blast*
    moreover
    {
     have $s \notin C$
     proof
      assume $s \in C$
      then have *distance* $C\ s \leq$ *dist* $s\ s$ **using** *Min.coboundedI*[*of distance C* ‘ *S dist s s*]
       by (*simp add*: *distance-def C-props*)
      also have ... $= 0$ **by** *simp*
      finally have *distance* $C\ s = 0$ **using** *dist-lemmas*(*4*) **by** (*smt C-props*)
      then have *radius-le-zero*: $2 *$ *?r* < 0 **using** *contr-assm s-def* **by** *simp*
      obtain x **where** *x-def*: *?r* = *distance* $C'\ x$ **using** *radius-def2* **by** *metis*
      obtain l **where** *l-def*: *distance* $C'\ x =$ *dist* $x\ l$ **using** *dist-lemmas*(*3*) **by**

(*metis* C'-*assms card-gt-0-iff*)
then have *dist x l* = *?r* **by** (*simp add*: *x-def*)
also have ... < *0* **using** C'-*assms radius-le-zero* **by** *simp*
finally show *False* **by** *simp*
qed
then have *card* ($C \cup \{s\}$) > *k* **using** *assms*(*1*,*2*) *C-props* **unfolding** *invar-def* **by** *simp*
}
moreover
have $C \cup \{s\} \subseteq S$ **using** *assms*(*1*) *s-def* **unfolding** *invar-def* **by** *simp*
moreover
have *finite* ($C \cup \{s\}$) **using** *calculation*(*3*) *finite-subset finite-sites* **by** *auto*
ultimately have $\forall$ *C*. *card C* $\leq$ *k* $\wedge$ *card C* > *0* $\longrightarrow$ *radius C* > *?r* **using** *inv-last-2* **by** *metis*
then have *?r* > *?r* **using** C'-*assms* **by** *blast*
then show *False* **by** *simp*
qed
next
assume $\forall$ *s*$\in$*S*. *distance C s* $\leq$ *2* $*$ *radius* C'
then show *?thesis* **by** (*metis image-iff radius-contained*)
qed
qed
qed

theorem *Center-Selection*:
VARS (*C* :: ('*a* :: *metric-space*) *set*) (*s* :: ('*a* :: *metric-space*))
{*k* $\leq$ *card S*}
C := {*some S*};
WHILE card C < *k INV* {*invar C*} *DO*
C := $C \cup$ {*furthest-from C*}
OD
{*card C* = *k* $\wedge$ ($\forall$ C'. *card* C' > *0* $\wedge$ *card* C' $\leq$ *k* $\longrightarrow$ *radius C* $\leq$ *2* $*$ *radius* C')}
proof (*vcg*, *goal-cases*)
case (*1 C s*)
show *?case* **using** *invar-init* **by** *simp*
next
case (*2 C s*)
then show *?case* **using** *invar-step* **by** *blast*
next
case (*3 C s*)
then show *?case* **using** *invar-last* **by** *blast*
qed

end
end

References

[1] R. Berghammer and M. Müller-Olm. Formal development and verification of approximation algorithms using auxiliary variables. In M. Bruynooghe, editor, *Logic Based Program Synthesis and Transformation, LOPSTR 2003*, volume 3018 of *LNCS*, pages 59–74. Springer, 2003.

[2] R. Berghammer and F. Reuter. A linear approximation algorithm for bin packing with absolute approximation factor 3/2. *Sci. Comput. Program.*, 48(1):67–80, 2003.

[3] R. Eßmann, T. Nipkow, and S. Robillard. Verified approximation algorithms. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning (IJCAR 2020)*, volume 12167 of *LNCS*, page 12167. Springer, 2020. https://doi.org/10.1007/978-3-030-51054-1_17.

[4] J. M. Kleinberg and É. Tardos. *Algorithm Design.* Addison-Wesley, 2006.