Verified Approximation Algorithms

Robin Eßmann, Tobias Nipkow, Simon Robillard, Ujkan Sulejmani

March 17, 2025

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**
$vertex\text{-}cover\ E\ C = (\forall\, e \in E.\ e \cap C \neq \{\})$

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

lemma *card-matching-vertex-cover*:
$[\![\ finite\ C;\ \ matching\ M;\ \ M \subseteq E;\ \ vertex\text{-}cover\ E\ C\ ]\!] \Longrightarrow card\ M \leq card\ C$
$\langle proof \rangle$

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* ⇒ *'a set set* ⇒ *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*⟩

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-cover* $E\ C \wedge (\forall C'.$ *finite* $C' \wedge$ *vertex-cover* $E\ C' \longrightarrow$ *card* $C \leq 2 *$ *card* $C')$}
⟨*proof*⟩

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 ⇒ 'a set set ⇒ bool* **where**
$invar\ C\ F = (F \subseteq E \wedge vertex\text{-}cover\ (E{-}F)\ C \wedge finite\ C \wedge (\exists M.\ inv\text{-}matching\ C\ F\ M))$

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*⟩

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
 $\{vertex\text{-}cover\ E\ C \wedge (\forall C'.\ finite\ C' \wedge vertex\text{-}cover\ E\ C' \longrightarrow card\ C \leq k * card\ C')\}$
⟨*proof*⟩

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**
 $harm\ n = (\sum k{=}1..n.\ inverse\ (of\text{-}nat\ k))$

locale *Set-Cover* =

fixes *w* :: *nat* ⇒ *real*
and *m* :: *nat*
and *S* :: *nat* ⇒ *'a set*
assumes *S-finite*: $\forall\, i \in \{1..m\}.\ \mathit{finite}\ (S\ i)$
and *w-nonneg*: $\forall\, i.\ 0 \leq w\ i$
begin

definition *U* :: *'a set* **where**
$U = (\bigcup i \in \{1..m\}.\ S\ i)$

lemma *S-subset*: $\forall\, i \in \{1..m\}.\ S\ i \subseteq U$
⟨*proof*⟩

lemma *U-finite*: *finite U*
⟨*proof*⟩

lemma *empty-cover*: $m = 0 \Longrightarrow U = \{\}$
⟨*proof*⟩

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

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

lemma *cost-nonneg*: $0 \leq cost\ R\ i$
⟨*proof*⟩

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

fun *min-arg* :: *'a set* ⇒ *nat* ⇒ *nat* **where**
min-arg R 0 = 1
| *min-arg R (Suc x) =*
(*let j = min-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*: $k > 0 \Longrightarrow \textit{min-arg}\ R\ k \in \{1..k\}$
⟨*proof*⟩

lemma *min-empty*: $S\ (\textit{min-arg}\ R\ k) \cap R = \{\} \Longrightarrow \forall\, i \in \{1..k\}.\ S\ i \cap R = \{\}$
⟨*proof*⟩

lemma *min-correct*: $[\![\ i \in \{1..k\};\ S\ i \cap R \neq \{\}\ ]\!] \Longrightarrow cost\ R\ (\textit{min-arg}\ R\ k) \leq cost\ R\ i$
⟨*proof*⟩

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 ≠ {} *INV* {*R* ⊆ *U* ∧ *sc C* (*U* − *R*)} *DO*
  i := *min-arg R m*;
  R := *R* − *S i*;
  C := *C* ∪ {*i*}
  OD
  {*sc C U*}
⟨*proof*⟩

definition *c-exists* :: *nat set* ⇒ *'a set* ⇒ *bool* **where**
  c-exists C R = (∃ *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*))

definition *inv* :: *nat set* ⇒ *'a set* ⇒ *bool* **where**
  inv C R ⟷ *sc C* (*U* − *R*) ∧ *R* ⊆ *U* ∧ *c-exists C R*

lemma *invI*:
  assumes *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*)
   shows *inv C R* ⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *inv-init*: *inv* {} *U*
⟨*proof*⟩

lemma *inv-step*:
  assumes *inv C R R* ≠ {}
  defines [*simp*]: *i* ≡ *min-arg R m*
  shows *inv* (*C* ∪ {*i*}) (*R* − (*S i*))
⟨*proof*⟩

lemma *cover-sum*:
  fixes *c* :: *'a* ⇒ *real*
  assumes *sc C V* ∀ *i*. *0* ≤ *c i*
  shows *sum c V* ≤ (∑ *i* ∈ *C*. *sum c* (*S i*))
⟨*proof*⟩

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

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

lemma *set-cover-bound*:
  assumes *inv C* {} *sc C′ U*
  shows *sum w C* $\leq$ *H* d^* $*$ *sum w C′*
⟨*proof*⟩

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-arg R m*;
  R := *R* − *S i*;
  C := *C* ∪ {*i*}
  OD
  {*sc C U* ∧ (∀ *C′*. *sc C′ U* ⟶ *sum w C* $\leq$ *H* d^* $*$ *sum w C′*)}
⟨*proof*⟩

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* ⇒ *′a set* ⇒ *bool* **where**
independent-vertices E S ⟷ *S* ⊆ ⋃ *E* ∧ (∀ *v1 v2*. *v1* ∈ *S* ∧ *v2* ∈ *S* ⟶ {*v1*, *v2*} ∉ *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* ⇒ *'a set* **where**
$vertices\ G = \bigcup G$

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

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

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

fun *degree-vertex* :: *'a* ⇒ *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*: $e \in E \Longrightarrow finite\ e$
⟨*proof*⟩

lemma *finite-V*: *finite V*
⟨*proof*⟩

lemma *finite-neighbors*: *finite* (*neighbors u*)
⟨*proof*⟩

lemma *independent-vertices-finite*: $independent\text{-}vertices\ E\ S \Longrightarrow finite\ S$
⟨*proof*⟩

lemma *edge-ex-vertices*: $e \in E \Longrightarrow \exists u\ v.\ u \neq v \land e = \{u, v\}$
⟨*proof*⟩

lemma Δ*-pos* [*simp*]: $E = \{\} \lor 0 < \Delta$
⟨*proof*⟩

lemma Δ*-max-degree*: $u \in V \Longrightarrow degree\text{-}vertex\ u \leq \Delta$
⟨*proof*⟩

3.2 Wei's algorithm: (Δ+*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* ⇒ *'a set* ⇒ *bool* **where**
$inv\text{-}iv\ S\ X \longleftrightarrow independent\text{-}vertices\ E\ S$

$\wedge\ X \subseteq V$
$\wedge\ (\forall\, v1 \in (V - X).\ \forall\, v2 \in S.\ \{v1,\ v2\} \notin E)$
$\wedge\ S \subseteq X$

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*⟩

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*⟩

lemma *inv-approx*: *independent-vertices E S* $\Longrightarrow$ *card V* $\leq$ *card S* $*$ *r* $\Longrightarrow$ *approximation-miv r S*
⟨*proof*⟩

theorem *wei-approx-*Δ*-plus-1*:
VARS (S :: *'a set*) (X :: *'a set*) (x :: *'a*)
{ *True* }
$S := \{\}$;
$X := \{\}$;
WHILE $X \neq V$
INV { *inv-approx S X* $(\Delta + 1)$ }
DO $x :=$ (*SOME* $x.\ x \in V - X$);
$S :=$ *insert x S*;
$X := X \cup$ *neighbors x* $\cup \{x\}$
OD
{ *approximation-miv* $(\Delta + 1)$ S }
⟨*proof*⟩

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

lemma *inv-partition-preserv*:
assumes *inv*: $inv\text{-}partition\ S\ X\ P$
and *x-def*: $x \in V - X$
shows $inv\text{-}partition\ (insert\ x\ S)\ (X \cup neighbors\ x \cup \{x\})\ (insert\ (\{x\} \cup neighbors\ x)\ P)$
⟨*proof*⟩

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*⟩

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*⟩

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*: $independent\text{-}vertices\ E\ S$
shows $\forall p \in P.\ card\ (S \cap p) \leq \Delta$
⟨*proof*⟩

lemma *inv-partition-approx*: $inv\text{-}partition\ S\ V\ P \Longrightarrow approximation\text{-}miv\ \Delta\ S$
⟨*proof*⟩

theorem *wei-approx-Δ*:
$VARS\ (S :: 'a\ set)\ (X :: 'a\ set)\ (x :: 'a)$
$\{\ True\ \}$
$S := \{\};$
$X := \{\};$
$WHILE\ X \neq V$

INV { ∃ *P*. *inv-partition S X P* }
DO x := (*SOME x*. *x* ∈ *V* − *X*);
S := *insert x S*;
X := *X* ∪ *neighbors x* ∪ {*x*}
OD
{ *approximation-miv* Δ *S* }
⟨*proof*⟩

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*⟩

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* ∈ *V* − *X*
shows *inv-approx* (*insert x S*) (*X* ∪ *neighbors x* ∪ {*x*}) (*max* (*card* (*neighbors x* ∪ {*x*} − *X*)) *s*)
⟨*proof*⟩

theorem *wei-approx-min-degree-heuristic*:
VARS (*S* :: *'a set*) (*X* :: *'a set*) (*x* :: *'a*) (*r* :: *nat*)
{ *True* }
S := {};
X := {};
r := *0*;
WHILE X ≠ *V*
INV { *inv-approx S X r* }
DO x := (*SOME x*. *x* ∈ *V* − *X* ∧ *vertex-heuristic X x*);
S := *insert x S*;

$r := max\ (card\ (neighbors\ x \cup \{x\} - X))\ r;$
$X := X \cup neighbors\ x \cup \{x\}$
OD
$\{\ approximation\text{-}miv\ r\ S\ \}$
⟨*proof*⟩

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*⟩

lemma *Max-const*[*simp*]: $[\![\ finite\ A;\ A \neq \{\}\ ]\!] \Longrightarrow Max\ ((\lambda\text{-}.\ c)\ `\ A) = c$
⟨*proof*⟩

abbreviation $Max_0 :: nat\ set \Rightarrow nat$ **where**
$Max_0\ N \equiv (if\ N{=}\{\}\ then\ 0\ else\ Max\ N)$

fun $f\text{-}Max_0 :: (nat \Rightarrow nat) \Rightarrow nat \Rightarrow nat$ **where**
$f\text{-}Max_0\ f\ 0 = 0$
$|\ f\text{-}Max_0\ f\ (Suc\ x) = max\ (f\ (Suc\ x))\ (f\text{-}Max_0\ f\ x)$

lemma $f\text{-}Max_0\text{-}equiv$: $f\text{-}Max_0\ f\ n = Max_0\ (f\ `\ \{1..n\})$
⟨*proof*⟩

lemma $f\text{-}Max_0\text{-}correct$:
$\forall x \in \{1..m\}.\ T\ x \leq f\text{-}Max_0\ T\ m$
$m > 0 \Longrightarrow \exists x \in \{1..m\}.\ T\ x = f\text{-}Max_0\ T\ m$
⟨*proof*⟩

lemma $f\text{-}Max_0\text{-}mono$:
$y \leq T\ x \Longrightarrow f\text{-}Max_0\ (T\ (x := y))\ m \leq f\text{-}Max_0\ T\ m$
$T\ x \leq y \Longrightarrow f\text{-}Max_0\ T\ m \leq f\text{-}Max_0\ (T\ (x := y))\ m$
⟨*proof*⟩

lemma $f\text{-}Max_0\text{-}out\text{-}of\text{-}range$ [*simp*]:

$x \notin \{1..k\} \Longrightarrow \textit{f-Max}_0\ (T\ (x := y))\ k = \textit{f-Max}_0\ T\ k$
⟨*proof*⟩

lemma *fun-upd-f-Max*$_0$:
assumes $x \in \{1..m\}$ $T\ x \leq y$
shows $\textit{f-Max}_0\ (T\ (x := y))\ m = \textit{max}\ y\ (\textit{f-Max}_0\ T\ m)$
⟨*proof*⟩

locale *LoadBalancing* =
fixes $t :: \textit{nat} \Rightarrow \textit{nat}$
and $m :: \textit{nat}$
and $n :: \textit{nat}$
assumes *m-gt-0*: $m > 0$
begin

4.1 Formalization of a Correct Load Balancing

4.1.1 Definition

definition $\textit{lb} :: (\textit{nat} \Rightarrow \textit{nat}) \Rightarrow (\textit{nat} \Rightarrow \textit{nat set}) \Rightarrow \textit{nat} \Rightarrow \textit{bool}$ **where**
$\textit{lb}\ T\ A\ j = ((\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\})$ — 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 $\textit{makespan} :: (\textit{nat} \Rightarrow \textit{nat}) \Rightarrow \textit{nat}$ **where**
$\textit{makespan}\ T \equiv \textit{f-Max}_0\ T\ m$

lemma *makespan-def'*: $\textit{makespan}\ T = \textit{Max}\ (T\ `\ \{1..m\})$
⟨*proof*⟩

lemma *makespan-correct*:
$\forall x \in \{1..m\}.\ T\ x \leq \textit{makespan}\ T$
$\exists x \in \{1..m\}.\ T\ x = \textit{makespan}\ T$
⟨*proof*⟩

lemma *lbE*:
assumes $\textit{lb}\ T\ A\ j$
shows $\forall x \in \{1..m\}.\ \forall y \in \{1..m\}.\ x \neq y \longrightarrow A\ x \cap A\ y = \{\}$
$(\bigcup x \in \{1..m\}.\ A\ x) = \{1..j\}$
$\forall x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y) = T\ x$
⟨*proof*⟩

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 $\textit{lb}\ T\ A\ j$ ⟨*proof*⟩

lemma *A-lb-finite* [*simp*]:
assumes $lb\ T\ A\ j\ x \in \{1..m\}$
shows $finite\ (A\ x)$
⟨*proof*⟩

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 $(\sum x \in \{1..m\}.\ (\sum y \in A\ x.\ t\ y)) = (\sum x \in (\bigcup y \in \{1..m\}.\ A\ y).\ t\ x)$
⟨*proof*⟩

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 $lb\ T\ A\ j$
shows $(\sum x \in \{1..m\}.\ T\ x) = (\sum x \in \{1..j\}.\ t\ x)$
⟨*proof*⟩

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 $lb\ T\ A\ j\ x \in \{1..j\}$
shows $\exists y \in \{1..m\}.\ t\ x \leq T\ y$
⟨*proof*⟩

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 $t\ x \leq makespan\ T$
⟨*proof*⟩

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 $Max_0\ (t\ `\ \{1..j\}) \leq makespan\ T$
⟨*proof*⟩

lemma *job-dist-lower-bound-makespan*:
  assumes *lb T A j*
  shows $(\sum x \in \{1..j\}.\ t\ x)\ /\ m \leq makespan\ T$
⟨*proof*⟩

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**
  $min\text{-}arg\ T\ 0 = 1$
| $min\text{-}arg\ T\ (Suc\ x) =$
   $(let\ k = min\text{-}arg\ T\ x$
    $in\ if\ T\ (Suc\ x) < T\ k\ then\ (Suc\ x)\ else\ k)$

lemma *min-correct*:
  $\forall x \in \{1..m\}.\ T\ (min\text{-}arg\ T\ m) \leq T\ x$
  ⟨*proof*⟩

lemma *min-in-range*:
  $k > 0 \Longrightarrow (min\text{-}arg\ T\ k) \in \{1..k\}$
  ⟨*proof*⟩

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*⟩

lemma *makespan-mono*:
  $y \leq T\ x \Longrightarrow makespan\ (T\ (x := y)) \leq makespan\ T$
  $T\ x \leq y \Longrightarrow makespan\ T \leq makespan\ (T\ (x := y))$
  ⟨*proof*⟩

lemma *smaller-optimum*:
  assumes $lb\ T\ A\ (Suc\ j)$
  shows $\exists\ T'\ A'.\ lb\ T'\ A'\ j \wedge makespan\ T' \leq makespan\ T$
⟨*proof*⟩

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*⟩

lemma *greedy-makespan-no-jobs* [*simp*]:
  $makespan\ (\lambda\text{-}.\ 0) = 0$
  ⟨*proof*⟩

lemma *min-avg*: $m * T\ (\mathit{min\text{-}arg}\ T\ m) \leq (\sum i \in \{1..m\}.\ T\ i)$
  (**is** ‹- ∗ ?T ≤ ?S›)
⟨*proof*⟩

definition inv_1 :: $(nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat\ set) \Rightarrow nat \Rightarrow bool$ **where**
  $inv_1\ T\ A\ j = (lb\ T\ A\ j \wedge j \leq n \wedge (\forall T'\ A'.\ lb\ T'\ A'\ j \longrightarrow makespan\ T \leq 2 * makespan\ T'))$

lemma $inv_1 E$:
  assumes $inv_1\ T\ A\ j$
  shows $lb\ T\ A\ j$ $j \leq n$
    $lb\ T'\ A'\ j \Longrightarrow makespan\ T \leq 2 * makespan\ T'$
  ⟨*proof*⟩

lemma $inv_1 I$:
  assumes $lb\ T\ A\ j$ $j \leq n$ $\forall T'\ A'.\ lb\ T'\ A'\ j \longrightarrow makespan\ T \leq 2 * makespan\ T'$
  shows $inv_1\ T\ A\ j$ ⟨*proof*⟩

lemma inv_1*-step*:
  assumes $inv_1\ T\ A\ j$ $j < n$
  shows $inv_1\ (T\ ((\mathit{min\text{-}arg}\ T\ m) := T\ (\mathit{min\text{-}arg}\ T\ m) + t\ (Suc\ j)))$
      $(A\ ((\mathit{min\text{-}arg}\ T\ m) := A\ (\mathit{min\text{-}arg}\ T\ m) \cup \{Suc\ j\}))\ (Suc\ j)$
    (**is** ‹inv_1 ?T ?A -›)
⟨*proof*⟩

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*
  $i := \mathit{min\text{-}arg}\ T\ m$;
  $j := (Suc\ j)$;
  $A := A\ (i := A(i) \cup \{j\})$;
  $T := T\ (i := T(i) + t\ j)$
OD
$\{lb\ T\ A\ n \wedge (\forall T'\ A'.\ lb\ T'\ A'\ n \longrightarrow makespan\ T \leq 2 * makespan\ T')\}$
⟨*proof*⟩

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*]: $[\![\ sorted\ j;\ j \geq j'\ ]\!] \Longrightarrow sorted\ j'$
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *min-zero*:
assumes $x \in \{1..k\}\ T\ x = 0$
shows $T\ (min\text{-}arg\ T\ k) = 0$
⟨*proof*⟩

lemma *min-zero-index*:
assumes $x \in \{1..k\}\ T\ x = 0$
shows $min\text{-}arg\ T\ k \leq x$
⟨*proof*⟩

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

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

lemma inv_2I:
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 $inv_2\ T\ A\ j$
⟨*proof*⟩

lemma *inv*$_2$*-step*:
  assumes *sorted n inv*$_2$ *T A j j < n*
  shows *inv*$_2$ (*T* (*min-arg T m* := *T*(*min-arg T m*) + *t*(*Suc j*)))
    (*A* (*min-arg T m* := *A*(*min-arg T m*) ∪ {*Suc j*})) (*Suc j*)
  (**is** ‹*inv*$_2$ *?T ?A -*›)
⟨*proof*⟩

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*⟩

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 ⟦·⟧ from the article. B will only be wrapped into a set if it is non-empty.

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

lemma *wrap-card*:
  card (*wrap B*) ≤ *1*
  ⟨*proof*⟩

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* $(\{M \cup N\} \cup V)$
⟨*proof*⟩

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* ⇒ *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* ⇒ *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* ⇒ *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* $\{\} \notin P$ $\bigcup P = U$ $\forall B \in P.\ W(B) \leq c$
⟨*proof*⟩

lemma *bpI*:
assumes *pairwise disjnt P* $\{\} \notin P$ $\bigcup P = U$ $\forall B \in P.\ W(B) \leq c$
shows *bp P*
⟨*proof*⟩

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
$\{S = \{v \in U.\ w(v) \leq c\ /\ 2\} \wedge L = \{v \in U.\ w(v) > c\ /\ 2\}\}$
⟨*proof*⟩

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 $(P_1 \cup wrap\ B_1) \cap (P_2 \cup wrap\ B_2) = \{\}$
⟨*proof*⟩

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 $\bigcup(P_1 \cup \textit{wrap}\ B_1 \cup P_2 \cup \textit{wrap}\ B_2) = U - V$
and $B_1 \notin (P_1 \cup P_2 \cup \textit{wrap}\ B_2)$
and $B_2 \notin (P_1 \cup \textit{wrap}\ B_1 \cup P_2)$
and $(P_1 \cup \textit{wrap}\ B_1) \cap (P_2 \cup \textit{wrap}\ B_2) = \{\}$
shows $\textit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V$
⟨*proof*⟩

lemma *wrap-Un* [*simp*]: $\textit{wrap}\ (M \cup \{x\}) = \{M \cup \{x\}\}$ ⟨*proof*⟩
lemma *wrap-empty* [*simp*]: $\textit{wrap}\ \{\} = \{\}$ ⟨*proof*⟩
lemma *wrap-not-empty* [*simp*]: $M \neq \{\} \longleftrightarrow \textit{wrap}\ M = \{M\}$ ⟨*proof*⟩

If $\textit{inv}_1$ 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 $\textit{inv}_1$ also holds if B_1 and $\{u\}$ are replaced by $B_1 \cup \{u\}$.

lemma $\textit{inv}_1$*-stepA*:
assumes $\textit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W(B_1) + w(u) \leq c$
shows $\textit{inv}_1\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (V - \{u\})$
⟨*proof*⟩

If $\textit{inv}_1$ 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 $\textit{inv}_1$ also holds if B_2 and $\{u\}$ are replaced by $B_2 \cup \{u\}$.

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*⟩

If $\textit{inv}_1$ holds for the current partial solution, then $\textit{inv}_1$ 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 $\textit{inv}_1$*-stepC*:
assumes $\textit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V$
shows $\textit{inv}_1\ (P_1 \cup \textit{wrap}\ B_1)\ (P_2 \cup \textit{wrap}\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$
⟨*proof*⟩

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 $P\ P_1\ P_2\ B_1\ B_2\ V\ u$
$\{\textit{True}\}$
$P_1 := \{\};\ P_2 := \{\};\ B_1 := \{\};\ B_2 := \{\};\ V := U;$
WHILE $V \cap S \neq \{\}$ *INV* $\{\textit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\}$ *DO*
$u := (\textit{SOME}\ u.\ u \in V);\ V := V - \{u\};$
IF $W(B_1) + w(u) \leq c$
THEN $B_1 := B_1 \cup \{u\}$
ELSE IF $W(B_2) + w(u) \leq c$

$THEN\ B_2 := B_2 \cup \{u\}$
$ELSE\ P_2 := P_2 \cup wrap\ B_2;\ B_2 := \{u\}\ FI;$
$P_1 := P_1 \cup wrap\ B_1;\ B_1 := \{\}\ FI$
$OD;$
$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\}$
$\{bp\ P\}$
⟨*proof*⟩

5.2.2 Lower Bounds for the Bin Packing Problem

lemma *bp-bins-finite* [*simp*]:
assumes $bp\ P$
shows $\forall B \in P.\ finite\ B$
⟨*proof*⟩

lemma *bp-sol-finite* [*simp*]:
assumes $bp\ P$
shows $finite\ P$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $card\ (P_1 \cup wrap\ B_1 \cup \{\{v\}\ |v.\ v \in V \cap L\}) \leq card\ P$
$\langle proof \rangle$

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 $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
$\langle proof \rangle$

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 $(\sum u \in U.\ w\ u) \leq c * card\ P$
$\langle proof \rangle$

lemma *bp-NE*:
assumes $bp\ P$
shows $P \neq \{\}$
$\langle proof \rangle$

lemma *sum-Un-ge*:
fixes $f :: \text{-} \Rightarrow real$
assumes $finite\ M\ finite\ N\ \forall B \in M \cup N.\ 0 < f\ B$
shows $sum\ f\ M \leq sum\ f\ (M \cup N)$
$\langle proof \rangle$

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

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 $bp\ P\ inv_1\ P_1\ P_2\ B_1\ B_2\ V\ bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
shows $card\ P_1 + 1 \leq card\ P$
$\langle proof \rangle$

As $card\ (wrap\ ?B) \leq 1$ 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 $bp\ P\ inv_1\ P_1\ P_2\ B_1\ B_2\ V\ bij\text{-}exists\ P_1\ (\bigcup(P_2 \cup wrap\ B_2))$
shows $card\ (P_1 \cup wrap\ B_1) \le card\ P$
$\langle proof \rangle$

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 $P_2\text{-}B_2\text{-}lower\text{-}bound\text{-}P_1$:
assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ V\ 2 * card\ P_2 \le card\ (\bigcup P_2)\ bij\text{-}exists\ P_1\ (\bigcup(P_2 \cup wrap\ B_2))$
shows $2 * card\ (P_2 \cup wrap\ B_2) \le card\ P_1 + 1$
$\langle proof \rangle$

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
$\wedge\ (2 * card\ P_2 \le card\ (\bigcup P_2))$ — 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 $2 * card\ P_2 \le card\ (\bigcup P_2)$
$\langle proof \rangle$

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 $2 * card\ P_2 \le card\ (\bigcup P_2)$
shows $inv_2\ P_1\ P_2\ B_1\ B_2\ V$
$\langle proof \rangle$

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*⟩

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 ⇒ 'a set set ⇒ 'a set ⇒ 'a set ⇒ 'a set ⇒ bool* **where**
$inv_3\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow inv_2\ P_1\ P_2\ B_1\ B_2\ V \land B_2 \subseteq S$

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** $B_2 \subseteq S$
⟨*proof*⟩

lemma inv_3I:
assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ V$ **and** $B_2 \subseteq S$
shows $inv_3\ P_1\ P_2\ B_1\ B_2\ V$
⟨*proof*⟩

lemma *loop-init*:
$inv_3\ \{\}\ \{\}\ \{\}\ \{\}\ U$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $inv_3\ (P_1 \cup wrap\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (V - \{u\})$
$\langle proof \rangle$

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 $2 \leq card\ B_2$
$\langle proof \rangle$

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 $inv_3\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$
$\langle proof \rangle$

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;

$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\}$
$\{bp\ P \land (\forall Q.\ bp\ Q \longrightarrow card\ P \leq 3\ /\ 2 * card\ Q)\}$
⟨*proof*⟩

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*: $\forall u \in U.\ 0 < w(u) \land w(u) \leq c$
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**
$bp\ P \longleftrightarrow partition\text{-}on\ U\ P \land (\forall B \in P.\ W(B) \leq c)$

lemma *bpE*:
assumes *bp P*
shows $pairwise\ disjnt\ P\ \{\} \notin P\ \bigcup P = U\ \forall B \in P.\ W(B) \leq c$
⟨*proof*⟩

lemma *bpI*:
assumes $pairwise\ disjnt\ P\ \{\} \notin P\ \bigcup P = U\ \forall B \in P.\ W(B) \leq c$
shows *bp P*
⟨*proof*⟩

definition inv_1 :: *'a set set* ⇒ *'a set set* ⇒ *'a set* ⇒ *'a set* ⇒ *'a set* ⇒ *bool* **where**
$inv_1\ P_1\ P_2\ B_1\ B_2\ V \longleftrightarrow bp\ (P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\} \mid v.\ v \in V\})$ — A correct solution to the bin packing problem
$\land \bigcup(P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2) = U - V$ — The partial solution does not contain objects that have not yet been assigned

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

lemma $\mathit{inv}_1 E$:
assumes $\mathit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V$
shows $\mathit{bp}\ (P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
and $\bigcup(P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2) = U - V$
and $B_1 \notin (P_1 \cup P_2 \cup \mathit{wrap}\ B_2)$
and $B_2 \notin (P_1 \cup \mathit{wrap}\ B_1 \cup P_2)$
and $(P_1 \cup \mathit{wrap}\ B_1) \cap (P_2 \cup \mathit{wrap}\ B_2) = \{\}$
⟨*proof*⟩

lemma $\mathit{inv}_1 I$:
assumes $\mathit{bp}\ (P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2 \cup \{\{v\}\ |v.\ v \in V\})$
and $\bigcup(P_1 \cup \mathit{wrap}\ B_1 \cup P_2 \cup \mathit{wrap}\ B_2) = U - V$
and $B_1 \notin (P_1 \cup P_2 \cup \mathit{wrap}\ B_2)$
and $B_2 \notin (P_1 \cup \mathit{wrap}\ B_1 \cup P_2)$
and $(P_1 \cup \mathit{wrap}\ B_1) \cap (P_2 \cup \mathit{wrap}\ B_2) = \{\}$
shows $\mathit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V$
⟨*proof*⟩

lemma *wrap-Un* [*simp*]: $\mathit{wrap}\ (M \cup \{x\}) = \{M \cup \{x\}\}$ ⟨*proof*⟩
lemma *wrap-empty* [*simp*]: $\mathit{wrap}\ \{\} = \{\}$ ⟨*proof*⟩
lemma *wrap-not-empty* [*simp*]: $M \neq \{\} \longleftrightarrow \mathit{wrap}\ M = \{M\}$ ⟨*proof*⟩

lemma inv_1*-stepA*:
assumes $\mathit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W(B_1) + w(u) \leq c$
shows $\mathit{inv}_1\ P_1\ P_2\ (B_1 \cup \{u\})\ B_2\ (V - \{u\})$
⟨*proof*⟩

lemma inv_1*-stepB*:
assumes $\mathit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V\ W\ B_2 + w\ u \leq c$
shows $\mathit{inv}_1\ (P_1 \cup \mathit{wrap}\ B_1)\ P_2\ \{\}\ (B_2 \cup \{u\})\ (V - \{u\})$
⟨*proof*⟩

lemma inv_1*-stepC*:
assumes $\mathit{inv}_1\ P_1\ P_2\ B_1\ B_2\ V\ u \in V$
shows $\mathit{inv}_1\ (P_1 \cup \mathit{wrap}\ B_1)\ (P_2 \cup \mathit{wrap}\ B_2)\ \{\}\ \{u\}\ (V - \{u\})$
⟨*proof*⟩

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 $\forall B \in P.\ finite\ B$
⟨*proof*⟩

lemma *bp-sol-finite* [*simp*]:
assumes *bp P*
shows *finite P*
⟨*proof*⟩

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*⟩

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

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*⟩

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 $card\ (P_1 \cup wrap\ B_1 \cup \{\{v\}\ |v.\ v \in L\}) \leq card\ P$
⟨*proof*⟩

lemma *sum-Un-eq-sum-sum*:
assumes *bp P*
shows $(\sum u \in U.\ w\ u) = (\sum B \in P.\ W\ B)$
⟨*proof*⟩

lemma *sum-lower-bound-card*:
  assumes *bp P*
  shows $(\sum u \in U.\ w\ u) \leq c * card\ P$
⟨*proof*⟩

lemma *bp-NE*:
  assumes *bp P*
  shows $P \neq \{\}$
  ⟨*proof*⟩

lemma *sum-Un-ge*:
  fixes f :: - ⇒ *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*⟩

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

lemma P_1*-lower-bound-card*:
  assumes $bp\ P\ inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)\ bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
  shows $card\ P_1 + 1 \leq card\ P$
⟨*proof*⟩

lemma P_1-B_1*-lower-bound-card*:
  assumes $bp\ P\ inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)\ bij\text{-}exists\ P_1\ (\bigcup (P_2 \cup wrap\ B_2))$
  shows $card\ (P_1 \cup wrap\ B_1) \leq card\ P$
⟨*proof*⟩

lemma P_2-B_2*-lower-bound-*P_1:
  assumes $inv_1\ P_1\ P_2\ B_1\ B_2\ (S \cup L)\ 2 * card\ P_2 \leq card\ (\bigcup P_2)\ bij\text{-}exists\ P_1$ $(\bigcup (P_2 \cup wrap\ B_2))$
  shows $2 * card\ (P_2 \cup wrap\ B_2) \leq card\ P_1 + 1$
⟨*proof*⟩

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* ⇒ ′*a set set* ⇒ ′*a set* ⇒ ′*a set* ⇒ ′*a set* ⇒ ′*a set* ⇒ *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 $inv_2 E$:
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 $SL\ S\ L$
⟨*proof*⟩

lemma $inv_2 I$:
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 $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$
⟨*proof*⟩

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*⟩

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 $inv_3 E$:
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** $B_2 \subseteq S_U$
⟨*proof*⟩

lemma $inv_3 I$:
assumes $inv_2\ P_1\ P_2\ B_1\ B_2\ S\ L$ **and** $B_2 \subseteq S_U$
shows $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L$
⟨*proof*⟩

lemma *loop-init*:
$inv_3\ \{\}\ \{\}\ \{\}\ \{\}\ S_U\ L_U$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma B_2*-at-least-two-objects*:
  assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L\ u \in S\ W\ B_2 + w(u) > c$
  shows $2 \leq card\ B_2$
⟨*proof*⟩

lemma *loop-stepE*:
  assumes $inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L\ B_1 \neq \{\}\ u \in S\ W\ B_1 + w(u) > c\ W\ B_2 + w(u) > c$
  shows $inv_3\ (P_1 \cup wrap\ B_1)\ (P_2 \cup wrap\ B_2)\ \{\}\ \{u\}\ (S - \{u\})\ L$
⟨*proof*⟩

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 $P\ P_1\ P_2\ B_1\ B_2\ V\ S\ L\ u$
  $\{True\}$
  $S := \{\};\ L := \{\};\ V := U;$
  $WHILE\ V \neq \{\}\ INV\ \{V \subseteq U \wedge S = \{u \in U - V.\ w(u) \leq c\ /\ 2\} \wedge L = \{u \in U - V.\ c\ /\ 2 < w(u)\}\}\ DO$
    $u := (SOME\ u.\ u \in V);$
    $IF\ w(u) \leq c\ /\ 2$
    $THEN\ S := S \cup \{u\}$
    $ELSE\ L := L \cup \{u\}\ FI;$
    $V := V - \{u\}$
  $OD;$
  $P_1 := \{\};\ P_2 := \{\};\ B_1 := \{\};\ B_2 := \{\};$
  $WHILE\ S \neq \{\}\ INV\ \{inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L\}\ DO$
    $IF\ B_1 \neq \{\}$
    $THEN\ u := (SOME\ u.\ u \in S);\ S := S - \{u\}$
    $ELSE\ IF\ L \neq \{\}$

$THEN\ u := (SOME\ u.\ u \in L);\ L := L - \{u\}$
$ELSE\ u := (SOME\ u.\ u \in S);\ S := S - \{u\}\ FI\ FI;$
$IF\ W(B_1) + w(u) \leq c$
$THEN\ B_1 := B_1 \cup \{u\}$
$ELSE\ IF\ W(B_2) + w(u) \leq c$
$THEN\ B_2 := B_2 \cup \{u\}$
$ELSE\ P_2 := P_2 \cup wrap\ B_2;\ B_2 := \{u\}\ FI;$
$P_1 := P_1 \cup wrap\ B_1;\ B_1 := \{\}\ FI$
$OD;$
$P := P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2;\ V := L;$
$WHILE\ V \neq \{\}$
$INV\ \{S = \{\} \land inv_3\ P_1\ P_2\ B_1\ B_2\ S\ L \land V \subseteq L \land P = P_1 \cup wrap\ B_1 \cup P_2 \cup wrap\ B_2 \cup \{\{v\}|v.\ v \in L - V\}\}\ DO$
$u := (SOME\ u.\ u \in V);\ P := P \cup \{\{u\}\};\ V := V - \{u\}$
OD
$\{bp\ P \land (\forall\ Q.\ bp\ Q \longrightarrow card\ P \leq 3\ /\ 2 * card\ Q)\}$
$\langle proof \rangle$

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* $\Rightarrow$ $('a$::*metric-space*$)$ $\Rightarrow$ *real* **where**
distance $C\ s = Min\ (dist\ s\ \text{‘}\ C)$

definition *radius* :: $('a$ :: *metric-space*$)$ *set* $\Rightarrow$ *real* **where**
radius $C = Max\ (distance\ C\ \text{‘}\ S)$

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
⟨*proof*⟩

lemma *finite-distances*: *finite* (*distance* C ‘ S)
⟨*proof*⟩

lemma *non-empty-distances*: *distance* C ‘ $S \neq \{\}$
⟨*proof*⟩

lemma *radius-contained*: *radius* $C \in$ *distance* C ‘ S
⟨*proof*⟩

lemma *radius-def2*: $\exists\, s \in S.$ *distance* $C\ s =$ *radius* C
⟨*proof*⟩

lemma *dist-lemmas-aux*:
assumes *finite* C
and $C \neq \{\}$
shows *finite* (*dist* s ‘ C)
and *finite* (*dist* s ‘ C) $\Longrightarrow$ *distance* $C\ s \in$ *dist* s ‘ C
and *distance* $C\ s \in$ *dist* s ‘ $C \Longrightarrow \exists\, c \in C.$ *dist* $s\ c =$ *distance* $C\ s$
and $\exists\, c \in C.$ *dist* $s\ c =$ *distance* $C\ s \Longrightarrow$ *distance* $C\ s \geq 0$
⟨*proof*⟩

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* $C\ s \geq 0$
⟨*proof*⟩

lemma *radius-max-prop*: $(\forall\, s \in S.$ *distance* $C\ s \leq r) \Longrightarrow$ (*radius* $C \leq r$)
⟨*proof*⟩

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*⟩

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

lemma *inv-init*: $inv\ S\ \{\}\ r$
⟨*proof*⟩
lemma *inv-step*:
assumes $S' \neq \{\}$
and *IH*: $inv\ S'\ C\ r$
defines[*simp*]: $s \equiv (SOME\ s.\ s \in S')$
shows $inv\ (S' - \{s'\ .\ s' \in S' \wedge dist\ s\ s' \leq 2{*}r\})\ (C \cup \{s\})\ r$
⟨*proof*⟩

lemma *inv-last-1*:
assumes $\forall s \in (S - S').\ distance\ C\ s \leq 2{*}r$
and $S' = \{\}$
shows $radius\ C \leq 2{*}r$
⟨*proof*⟩

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*⟩

lemma *inv-last*:
assumes $inv\ \{\}\ C\ r$
shows $(card\ C \leq k \longrightarrow radius\ C \leq 2{*}r) \wedge (card\ C > k \longrightarrow (\forall C'.\ card\ C' > 0 \wedge card\ C' \leq k \longrightarrow radius\ C' > r))$
⟨*proof*⟩

theorem *Center-Selection-r*:
$VARS\ (S' :: ('a :: metric\text{-}space)\ set)\ (C :: ('a :: metric\text{-}space)\ set)\ (r :: real)\ (s :: 'a)$
$\{True\}$
$S' := S;$
$C := \{\};$
$WHILE\ S' \neq \{\}\ INV\ \{inv\ S'\ C\ r\}\ DO$
$s := (SOME\ s.\ s \in S');$

$C := C \cup \{s\};$
$S' := S' - \{s'.\ s' \in S' \wedge dist\ s\ s' \leq 2{*}r\}$
OD
$\{(card\ C \leq k \longrightarrow radius\ C \leq 2{*}r) \wedge (card\ C > k \longrightarrow (\forall C'.\ card\ C' > 0 \wedge card\ C' \leq k \longrightarrow radius\ C' > r))\}$
⟨*proof*⟩

6.2 The Main Algorithm

definition *invar* :: (*'a* :: *metric-space*) *set* ⇒ *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*⟩

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

lemma *invar-step*:
assumes *invar C*
and $card\ C < k$
shows $invar\ (C \cup \{furthest\text{-}from\ C\})$
⟨*proof*⟩

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*⟩

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*
$C := C \cup \{furthest\text{-}from\ C\}$
OD
$\{card\ C = k \wedge (\forall C'.\ card\ C' > 0 \wedge card\ C' \leq k \longrightarrow radius\ C \leq 2 * radius\ C')\}$
⟨*proof*⟩

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.