Runtime Extensions for BMSSP

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

November 5, 2026

Abstract

This entry extends the AFP formalization *Correctness and Complexity of BMSSP in Isabelle/HOL* [2], which formalizes the bounded multi-source shortest path algorithm of Duan, Mao, Mao, Shu, and Yin [1]. The extension adds a charged direct-insert relation that records both the range charged by the recurrence and the semantic output returned by each recursive child. It proves that least costs are realized by verified-correct runs, exhibits a closed branching instance, and records a machine-checked totality obstruction for the older diagonal exact-concrete path-family relation. Its main theorem, `path_nonvac_runtime_bigo_card_V_unconditional`, proves without run-existence, cost-bound, or closed-bound assumptions that the least charged cost on the growing unit-weight path family is $O(|V_n| \log^{2/3} |V_n|)$ when the internal schedule is evaluated at n^2. This is a cost-relation theorem, not a machine-step bound for the SML executable from the base entry.

AI assistance was used for proof engineering.

Contents

1 Relationship to the Published Entry **2**

2 The Unconditional Size-Parametric Headline **3**

3 Verified Runtime and the Obstruction Record **3**

4 Scope **4**

5 Checked Source **4**

6 Runtime Bound for a Verified-Correct BMSSP Run **24**

7 A Second, Structurally Different Non-Vacuous Instance **26**

8 A Size-Parametric BMSSP Running-Time Bound over a Growing Graph Family 28
8.1 The path family 28
8.2 Structural and cardinality lemmas 28
8.3 Easy locale obligations, uniform in k 29
8.4 The unconditional cost-recurrence bound (FLOOR) 29
8.5 The forced-degenerate (empty-pivot) regime is provably avoided 29
8.6 Unique shortest walks on the path family 30
8.7 The exact-concrete algorithm-level running-time bound 33
8.8 A charged path-family theorem waiting on charged cost bounds 34

9 On the Totality of the Exact-Concrete Cost Relation 35

10 An unconditional, non-vacuous size-parametric BMSSP running-time bound 53
10.1 Elementary analytic facts 53
10.2 A reusable "the source is the sole pivot" brick 53
10.3 The genuine shortest-path distance on the path family 54
10.4 A schedule that dominates the path size (non-vacuous regime) 54
10.5 The bucket cap dominates the path size at the inflated schedule 68
10.6 The inflated schedule index stays below the graph size 68
10.7 The inflated schedule is genuine and keeps the polylog in $\Theta(ln\ n)$ 69
10.8 The decoupled charged running time of the path family . . . 69
10.9 The decoupled charged headline (conditional skeleton for **B3**) 71
10.10 Dropping the run-existence premise (B1, unconditional in runs) 73
10.11 Charged-cost upper bound via the uncharged refinement (B2, uncharged route) 75
10.12 The unconditional non-vacuous size-parametric running-time bound 77
10.13 Charged closed-bound engine (B2): the charged level recurrence solver 78

1 Relationship to the Published Entry

The base entry [2] supplies finite weighted digraph semantics, BMSSP pre- and postconditions, the recursive and exact-concrete cost relations, logarithmic schedule bounds, a bucketed partition model, and executable correctness. This entry is a child session of `BMSSP_Correctness`. Its first three theories add only the new charged relation and lemmas required by the follow-up; all other theories import those extensions and the published AFP theories.

Theory	Added surface
`BMSSP_Direct_Insert_Extensions`	Charged direct-insert run and loop relations, correctness, existence, and child-cost accounting.
`BMSSP_Strict_Tie_Breaking_Extensions`	Charged progress and level-bound closers under strict tie breaking.
`BMSSP_Top_Level_Bounds_Extensions`	Charged top-level run, cost, least-time, correctness, and refined-bound interfaces.
`BMSSP_Verified_Runtime`	Joins the existing asymptotic least-cost theorem with correctness of a run realizing that least cost.
`BMSSP_Runtime_Instance_Branching`	A closed out-star instance of the verified-runtime theorem.
`BMSSP_Path_Family_Extended`	The growing unit-weight path family, the unconditional cost-expression floor, and conditional diagonal runtime statements.
`BMSSP_Cost_Totality`	The exact-concrete diagonal totality obstruction.
`BMSSP_NonVacuous_Family`	The decoupled charged family and unconditional non-vacuous runtime headline.

2 The Unconditional Size-Parametric Headline

For $n \in \mathbb{N}$, let P_n be the directed unit-weight path with vertices $0, \ldots, n$. The development proves

`path_nonvac_runtime_bigo_card_V_unconditional`

with the asymptotic meaning

$$T(n) \in O\Big(|V_n|\big(\ln(|V_n|+2)\big)^{2/3}\Big),$$

where $T(n) =$ `path_nonvac_time` n is the least charged direct-insert top-level cost for P_n at internal schedule n^2.

The statement has no assumptions. Its non-vacuity is explicit: `eventually_path_nonvac_charged_cost_exists` proves that charged runs exist for all but finitely many n, and `path_k_charged_cheap_imp_nonvac_time_le_bound` constructs a charged cost below the refined graph-time budget. The square schedule leaves the target logarithmic class unchanged because $\ln(n^2) = 2\ln n$, while making the level cap eventually large enough to cover the path.

3 Verified Runtime and the Obstruction Record

The theorem `verified_bmssp_runtime_bigo_target` strengthens the base entry's least-cost asymptotic statement by proving that the least cost is eventually realized by an exact-concrete top-level run whose completed set is the whole graph and whose output satisfies `sssp_correct`. The theory

`BMSSP_Runtime_Instance_Branching` closes this statement on an out-star whose source has two incomparable outgoing shortest paths.

Separately, `BMSSP_Cost_Totality` proves the structural obstruction in the older diagonal exact-concrete loop relation: a genuine step is forced to the initial prefix under a sound label, and no genuine step can begin at a positive lower bound. The conditional diagonal family theorem is therefore not used as evidence for non-vacuity. The unconditional headline instead uses the charged relation and decouples graph size from the internal schedule.

4 Scope

The imported base entry remains the source of executable `bmssp_distances` and its functional-correctness theorem. This entry does not claim that generated SML executes within the asymptotic cost bound. The runtime statements concern the formal cost relations and their graph-size-parametric instances.

5 Checked Source

theory *BMSSP-Direct-Insert-Extensions*
imports *BMSSP-Correctness.BMSSP-Direct-Insert-Costed*
begin

context *unique-shortest-digraph*
begin

inductive *charged-direct-insert-costed-partition-loop-state*
and *charged-direct-insert-costed-bmssp* **where**
Charged-Direct-Insert-State-Done:
keys-of $D = \{\} \Longrightarrow$
bound-le (*Fin a*) $B \Longrightarrow$
complete-on d' (*bound-tree* P B) $\Longrightarrow$
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B* d' D *a*
[] [] B [*range-tree* P *a* B] []
(*bound-tree* P (*Fin a*) $\cup \bigcup$ (*set* [*range-tree* P *a* B])) *0* []
| *Charged-Direct-Insert-State-Stop*:
bound-le (*Fin a*) $B \Longrightarrow$
complete-on d' (*bound-tree* P (*Fin a*)) $\Longrightarrow$
$k * cap \leq$ *card* (*bound-tree* P (*Fin a*)) $\Longrightarrow$
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B* d' D *a*
[] [] (*Fin a*) [*range-tree* P *a* (*Fin a*)] []
(*bound-tree* P (*Fin a*) $\cup \bigcup$ (*set* [*range-tree* P *a* (*Fin a*)])) *0* []
| *Charged-Direct-Insert-State-Step*:
pull-separates D (*M-of l*) *Bmax S-pull beta D-pull* $\Longrightarrow$
bound-le (*Fin beta*) $B \Longrightarrow$
bmssp-pre-full d S-pull (*Fin beta*) $\Longrightarrow$

$S\text{-}pull = split\text{-}below\ d\ P\ beta \Longrightarrow$
$(\forall\, x{\in}S\text{-}pull.\ reachable\ s\ x) \Longrightarrow$
$a \leq b \Longrightarrow$
$bound\text{-}le\ (Fin\ a)\ B' \Longrightarrow$
$complete\text{-}on\ d'\ (bound\text{-}tree\ P\ (Fin\ a)) \Longrightarrow$
$card\ (bound\text{-}tree\ P\ (Fin\ a)) < k * cap \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\text{-}pull\ (Fin\ beta)$
$d\text{-}child\ (Fin\ b)\ U\text{-}child\ c\text{-}child \Longrightarrow$
$charged\text{-}child = range\text{-}tree\ P\ a\ (Fin\ b) \Longrightarrow$
$complete\text{-}preserved\ d\text{-}child\ d'\ U\text{-}child \Longrightarrow$
$direct\text{-}edge\text{-}batch =$
$edge\text{-}relaxation\text{-}pairs\text{-}in\text{-}bound\ d\text{-}child\ charged\text{-}child\ beta\ B \Longrightarrow$
$lower\text{-}edge\text{-}batch =$
$edge\text{-}relaxation\text{-}pairs\text{-}between\ d\text{-}child\ charged\text{-}child\ b\ beta \Longrightarrow$
$source\text{-}batch = label\text{-}pairs\text{-}between\ d\ S\text{-}pull\ b\ beta \Longrightarrow$
$batch = direct\text{-}edge\text{-}batch\ @\ lower\text{-}edge\text{-}batch\ @\ source\text{-}batch \Longrightarrow$
$D\text{-}next = batch\text{-}min\text{-}update\ D\text{-}pull\ batch \Longrightarrow$
$partition\text{-}pull\text{-}cost\text{-}bound\ c\text{-}pull\ S\text{-}pull \Longrightarrow$
$partition\text{-}batch\text{-}cost\text{-}bound\ c\text{-}direct\ t\ direct\text{-}edge\text{-}batch \Longrightarrow$
$partition\text{-}batch\text{-}cost\text{-}bound\ c\text{-}lower\ h\ lower\text{-}edge\text{-}batch \Longrightarrow$
$partition\text{-}batch\text{-}cost\text{-}bound\ c\text{-}sources\ h\ source\text{-}batch \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'$
$D\text{-}next\ b\ betas\ bs\ B'\ charged\text{-}tail\ child\text{-}outputs\text{-}tail\ U\text{-}tail\ c\text{-}tail$
$child\text{-}costs\text{-}tail \Longrightarrow$
$c = c\text{-}pull + c\text{-}direct + c\text{-}lower + c\text{-}sources + c\text{-}child + c\text{-}tail \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$(beta\ \#\ betas)\ (b\ \#\ bs)\ B'$
$(charged\text{-}child\ \#\ charged\text{-}tail)\ (U\text{-}child\ \#\ child\text{-}outputs\text{-}tail)$
$(bound\text{-}tree\ P\ (Fin\ a) \cup \bigcup (set\ (charged\text{-}child\ \#\ charged\text{-}tail)))\ c$
$(c\text{-}child\ \#\ child\text{-}costs\text{-}tail)$

| *Charged-Direct-Insert-Base*:
$S = \{x\} \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ 0\ d\ S\ B$
$(\lambda v.\ if\ v \in base\text{-}case\text{-}vertices\ k\ x\ B\ then\ dist\ s\ v\ else\ d\ v)$
$(base\text{-}case\text{-}bound\ k\ x\ B)$
$(base\text{-}case\text{-}vertices\ k\ x\ B)$
$(base\text{-}case\text{-}scan\text{-}cost\ \Delta\ k\ x\ B)$

| *Charged-Direct-Insert-Step*:
$D = label\text{-}partition\text{-}view$
$(find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B)$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B) \Longrightarrow$
$partition\text{-}initial\text{-}insert\text{-}cost\text{-}bound\ c\text{-}insert\ t$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B) \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l$
$(find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B)$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B)\ B\ d'\ D\ a\ betas\ bs\ B'$
$charged\text{-}Us\ child\text{-}outputs\ U\text{-}loop\ c\text{-}loop\ child\text{-}costs\text{-}loop \Longrightarrow$
$complete\text{-}on\ d'$
$\{v \in bound\text{-}tree\ S\ B'.\ find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B\ v = dist\ s\ v\} \Longrightarrow$

$U = U\text{-}loop \cup$
$\{v \in bound\text{-}tree\ S\ B'.\ find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B\ v = dist\ s\ v\} \Longrightarrow$
$c = fp\text{-}iter\text{-}capped\text{-}scan\text{-}cost\ k\ cap\ d\ S\ S\ B + c\text{-}insert + c\text{-}loop \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ (Suc\ l)\ d\ S\ B\ d'\ B'\ U\ c$

lemma *charged-direct-insert-partition-loop-state-step-done-exists*:
fixes *a b beta Bmax* :: *real*
and *B* :: *bound*
and *D D-pull D-next* :: *'a partition-view*
and *M-of* :: *nat* ⇒ *nat*
and Δ *t h k cap l c-pull c-direct c-lower c-sources c-child* :: *nat*
and *d d' d-child* :: *'a* ⇒ *real*
and *P S-pull U-child* :: *'a set*
assumes *pull*: *pull-separates D* (*M-of l*) *Bmax S-pull beta D-pull*
and *beta-bound*: *bound-le* (*Fin beta*) *B*
and *pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
and *split*: *S-pull* = *split-below d P beta*
and *reaches*: ∀ *x*∈*S-pull*. *reachable s x*
and *a-le-b*: *a* ≤ *b*
and *a-bound*: *bound-le* (*Fin a*) *B*
and *complete-a*: *complete-on d'* (*bound-tree P* (*Fin a*))
and *small*: *card* (*bound-tree P* (*Fin a*)) < *k* ∗ *cap*
and *child*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-pull*
(*Fin beta*) *d-child* (*Fin b*) *U-child c-child*
and *preserved*: *complete-preserved d-child d' U-child*
and *D-next*:
D-next = *batch-min-update D-pull*
(*edge-relaxation-pairs-in-bound d-child* (*range-tree P a* (*Fin b*)) *beta B* @
edge-relaxation-pairs-between d-child (*range-tree P a* (*Fin b*)) *b beta* @
label-pairs-between d S-pull b beta)
and *c-pull*: *partition-pull-cost-bound c-pull S-pull*
and *c-direct*:
partition-batch-cost-bound c-direct t
(*edge-relaxation-pairs-in-bound d-child* (*range-tree P a* (*Fin b*)) *beta B*)
and *c-lower*:
partition-batch-cost-bound c-lower h
(*edge-relaxation-pairs-between d-child* (*range-tree P a* (*Fin b*)) *b beta*)
and *c-sources*:
partition-batch-cost-bound c-sources h (*label-pairs-between d S-pull b beta*)
and *tail-empty*: *keys-of D-next* = {}
and *b-bound*: *bound-le* (*Fin b*) *B*
and *complete-B*: *complete-on d'* (*bound-tree P B*)
shows
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B*
d' D a [*beta*] [*b*] *B* [*range-tree P a* (*Fin b*), *range-tree P b B*]
[*U-child*]
(*bound-tree P* (*Fin a*) ∪
⋃(*set* [*range-tree P a* (*Fin b*), *range-tree P b B*]))

$(c\text{-}pull + c\text{-}direct + c\text{-}lower + c\text{-}sources + c\text{-}child + 0)\ [c\text{-}child]$
⟨*proof*⟩

theorem *direct-insert-costed-refines-charged-direct-insert*:
$direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ Us\ U\ c\ child\text{-}costs \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ Us\ (range\text{-}tree\text{-}child\text{-}list\ P\ a\ bs)\ U\ c\ child\text{-}costs$
and *direct-insert-costed-bmssp-refines-charged-direct-insert*:
$direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\ B\ d'\ B'\ U\ c \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\ B\ d'\ B'\ U\ c$
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-child-call-charged-ranges*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs \Longrightarrow$
$length\ child\text{-}costs = length\ bs \wedge$
$length\ child\text{-}outputs = length\ bs \wedge$
$list\text{-}all2$
$(\lambda c\text{-}child\ U\text{-}child.\ \exists\ S\text{-}child\ B\text{-}child\ d\text{-}child\ B\text{-}child'.$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\text{-}child\ B\text{-}child$
$d\text{-}child\ B\text{-}child'\ U\text{-}child\ c\text{-}child \wedge$
$bmssp\text{-}pre\text{-}full\ d\ S\text{-}child\ B\text{-}child \wedge$
$(\forall x{\in}S\text{-}child.\ reachable\ s\ x) \wedge$
$card\ S\text{-}child \leq M\text{-}of\ l \wedge$
$(\forall x{\in}S\text{-}child.\ below\text{-}bound\ (d\ x)\ B\text{-}child))$
$child\text{-}costs\ child\text{-}outputs \wedge$
$list\text{-}all2$
$(\lambda c\text{-}child\ charged\text{-}child.\ \exists\ U\text{-}child\ S\text{-}child\ B\text{-}child\ d\text{-}child\ B\text{-}child'.$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\text{-}child\ B\text{-}child$
$d\text{-}child\ B\text{-}child'\ U\text{-}child\ c\text{-}child \wedge$
$bmssp\text{-}pre\text{-}full\ d\ S\text{-}child\ B\text{-}child \wedge$
$(\forall x{\in}S\text{-}child.\ reachable\ s\ x) \wedge$
$card\ S\text{-}child \leq M\text{-}of\ l \wedge$
$(\forall x{\in}S\text{-}child.\ below\text{-}bound\ (d\ x)\ B\text{-}child))$
$child\text{-}costs\ (range\text{-}tree\text{-}child\text{-}list\ P\ a\ bs) \wedge$
$charged\text{-}Us = range\text{-}tree\text{-}chain\text{-}list\ P\ a\ bs\ B'$
and *charged-direct-insert-costed-bmssp-child-call-charged-ranges-trivial*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\ B\ d'\ B'\ U\ c \Longrightarrow True$
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-trace*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs \Longrightarrow$
$sound\text{-}label\ d \Longrightarrow$
$bmssp\text{-}pre\text{-}full\ d\ P\ B \Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow reachable\ s\ x) \Longrightarrow$
$concrete\text{-}partition\text{-}loop\text{-}trace\ P\ B\ a\ bs\ d'\ B'\ charged\text{-}Us\ U$
and *charged-direct-insert-costed-bmssp-correct*:

charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d′ B′ U c* $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d S B $\Longrightarrow$
$(\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*$) \Longrightarrow$
bmssp-post-full d S B d′ B′ U
⟨*proof*⟩

lemma *charged-direct-insert-costed-bmssp-step-with-insert-cost*:
assumes *init*:
D = *label-partition-view*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*)
and *insert*:
partition-initial-insert-cost-bound c-insert t
(*find-pivots-pivots-capped k cap d S B*)
and *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′ D a betas bs B′*
charged-Us child-outputs U-loop c-loop child-costs-loop
and *complete*:
complete-on d′
{*v* ∈ *bound-tree S B′. find-pivots-label-capped k cap d S B v* = *dist s v*}
and *U*:
U = *U-loop* ∪
{*v* ∈ *bound-tree S B′. find-pivots-label-capped k cap d S B v* = *dist s v*}
and *c*:
c = *fp-iter-capped-scan-cost k cap d S S B* + *c-insert* + *c-loop*
shows *charged-direct-insert-costed-bmssp* Δ *M-of t h k cap* (*Suc l*) *d S B d′ B′ U c*
⟨*proof*⟩

lemma *charged-direct-insert-costed-bmssp-Suc-exists-from-initial-loop-at-bound*:
assumes *loop*:
$\exists$ *d′ a betas bs charged-Us child-outputs U-loop c-loop child-costs-loop.*
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′*
(*label-partition-view*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*))
a betas bs B′ charged-Us child-outputs U-loop c-loop child-costs-loop ∧
complete-on d′
{*v* ∈ *bound-tree S B′.*
find-pivots-label-capped k cap d S B v = *dist s v*} ∧
partition-initial-insert-cost-bound c-insert t
(*find-pivots-pivots-capped k cap d S B*)
shows $\exists$ *d′ U c.*
charged-direct-insert-costed-bmssp Δ *M-of t h k cap* (*Suc l*) *d S B d′ B′ U c*

⟨*proof*⟩

lemma *charged-direct-insert-costed-bmssp-Suc-exists-from-initial-loop-at-bound-with-insert*:
assumes *loop*:
∃ *d′ a betas bs charged-Us child-outputs U-loop c-loop*
child-costs-loop c-insert.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′*
(*label-partition-view*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*)) *a betas bs B′ charged-Us child-outputs U-loop c-loop child-costs-loop* ∧
complete-on d′
{*v* ∈ *bound-tree S B′*.
find-pivots-label-capped k cap d S B v = *dist s v*} ∧
partition-initial-insert-cost-bound c-insert t
(*find-pivots-pivots-capped k cap d S B*)
shows ∃ *d′ U c*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap* (*Suc l*) *d S B d′ B′ U c*
⟨*proof*⟩

lemma *charged-direct-insert-costed-bmssp-Suc-exists-from-initial-loop*:
assumes *loop*:
∃ *d′ a betas bs B′ charged-Us child-outputs U-loop c-loop child-costs-loop*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′*
(*label-partition-view*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*)) *a betas bs B′ charged-Us child-outputs U-loop c-loop child-costs-loop* ∧
complete-on d′
{*v* ∈ *bound-tree S B′*.
find-pivots-label-capped k cap d S B v = *dist s v*} ∧
partition-initial-insert-cost-bound c-insert t
(*find-pivots-pivots-capped k cap d S B*)
shows ∃ *d′ B′ U c*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap* (*Suc l*) *d S B d′ B′ U c*
⟨*proof*⟩

lemma *charged-direct-insert-partition-loop-state-done-exists*:
assumes *empty*: *keys-of D* = {}
and *a-bound*: *bound-le* (*Fin a*) *B*
and *complete*: *complete-on d′* (*bound-tree P B*)
shows ∃ *betas bs B′ charged-Us child-outputs U c child-costs*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs

⟨*proof*⟩

lemma *charged-direct-insert-partition-loop-state-stop-exists*:
assumes *a-bound*: *bound-le* (*Fin a*) *B*
and *complete*: *complete-on d′* (*bound-tree P* (*Fin a*))
and *threshold*: *k* ∗ *cap* ≤ *card* (*bound-tree P* (*Fin a*))
shows ∃ *betas bs B′ charged-Us child-outputs U c child-costs*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs
⟨*proof*⟩

lemma *charged-direct-insert-partition-loop-state-step-exists*:
assumes *pull*: *pull-separates D* (*M-of l*) *Bmax S-pull beta D-pull*
and *beta-bound*: *bound-le* (*Fin beta*) *B*
and *child-pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
and *S-pull-def*: *S-pull* = *split-below d P beta*
and *S-pull-reaches*: ∀ *x*∈*S-pull*. *reachable s x*
and *a-le-b*: *a* ≤ *b*
and *a-bound*: *bound-le* (*Fin a*) *B′*
and *complete-prefix*: *complete-on d′* (*bound-tree P* (*Fin a*))
and *below-threshold*: *card* (*bound-tree P* (*Fin a*)) < *k* ∗ *cap*
and *child*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-pull*
(*Fin beta*) *d-child* (*Fin b*) *U-child c-child*
and *preserved*: *complete-preserved d-child d′ U-child*
and *tail*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B d′
(*batch-min-update D-pull*
(*edge-relaxation-pairs-in-bound d-child* (*range-tree P a* (*Fin b*)) *beta B* @
edge-relaxation-pairs-between d-child (*range-tree P a* (*Fin b*)) *b beta* @
label-pairs-between d S-pull b beta))
b betas bs B′ charged-tail child-outputs-tail U-tail c-tail
child-costs-tail
shows ∃ *c child-costs*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B d′ D a (*beta* # *betas*) (*b* # *bs*) *B′*
(*range-tree P a* (*Fin b*) # *charged-tail*)
(*U-child* # *child-outputs-tail*)
(*bound-tree P* (*Fin a*) ∪
⋃(*set* (*range-tree P a* (*Fin b*) # *charged-tail*))) *c child-costs*
⟨*proof*⟩

lemma *charged-direct-insert-partition-loop-state-cases-exists-with-complete*:
fixes *W* :: *bound* ⇒ *′a set*
assumes *cases*:
(*keys-of D* = {} ∧
bound-le (*Fin a*) *B* ∧
complete-on d′ (*bound-tree P B*) ∧

```
      complete-on d' (W B)) ∨
     (bound-le (Fin a) B ∧
      complete-on d' (bound-tree P (Fin a)) ∧
      k * cap ≤ card (bound-tree P (Fin a)) ∧
      complete-on d' (W (Fin a))) ∨
     (∃ S-pull beta D-pull B'.
      pull-separates D (M-of l) Bmax S-pull beta D-pull ∧
      bound-le (Fin beta) B ∧
      bmssp-pre-full d S-pull (Fin beta) ∧
      S-pull = split-below d P beta ∧
      (∀ x∈S-pull. reachable s x) ∧
      complete-on d' (bound-tree P (Fin a)) ∧
      card (bound-tree P (Fin a)) < k * cap ∧
      complete-on d' (W B') ∧
      (∃ d-child b U-child c-child betas bs charged-tail
          child-outputs-tail U-tail c-tail child-costs-tail.
        a ≤ b ∧
        bound-le (Fin a) B' ∧
        charged-direct-insert-costed-bmssp Δ M-of t h k cap l d
          S-pull (Fin beta) d-child (Fin b) U-child c-child ∧
        complete-preserved d-child d' U-child ∧
        charged-direct-insert-costed-partition-loop-state Δ M-of t h k cap l
          d P B d'
          (batch-min-update D-pull
            (edge-relaxation-pairs-in-bound d-child
              (range-tree P a (Fin b)) beta B @
             edge-relaxation-pairs-between d-child
              (range-tree P a (Fin b)) b beta @
             label-pairs-between d S-pull b beta))
          b betas bs B' charged-tail child-outputs-tail U-tail c-tail
          child-costs-tail))
  shows ∃ betas bs B' charged-Us child-outputs U c child-costs.
    charged-direct-insert-costed-partition-loop-state Δ M-of t h k cap l
      d P B d' D a betas bs B' charged-Us child-outputs U c child-costs ∧
    complete-on d' (W B')
⟨proof⟩

lemma charged-direct-insert-costed-bmssp-Suc-exists-if-pivots-empty-same-bound:
  assumes a-bound: bound-le (Fin a) B
    and pivots-empty:
      find-pivots-pivots-capped k cap d S B = {}
  shows ∃ d' U c.
    charged-direct-insert-costed-bmssp Δ M-of t h k cap (Suc l) d S B
      d' B U c
⟨proof⟩

lemma charged-direct-insert-costed-bmssp-Suc-exists-if-initial-threshold-bound:
  assumes a-bound: bound-le (Fin a) B
    and threshold:
```

$k * cap \leq$
$card\ (bound\text{-}tree\ (find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B)\ (Fin\ a))$
shows $\exists\, d'\ U\ c.$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ (Suc\ l)\ d\ S\ B$
$d'\ (Fin\ a)\ U\ c$
⟨*proof*⟩

theorem *finite-initial-label-charged-direct-insert-costed-top-level-correct*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow reachable\ s\ v$
and *run*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l$
$finite\text{-}initial\text{-}label\ \{s\}\ Infinity\ d'\ Infinity\ U\ c$
shows *sssp-correct* d'
⟨*proof*⟩

theorem *finite-initial-label-charged-direct-insert-costed-top-level-reachable-correct*:
assumes *run*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l$
$finite\text{-}initial\text{-}label\ \{s\}\ Infinity\ d'\ Infinity\ U\ c$
shows *sssp-correct* d'
⟨*proof*⟩

inductive-cases *charged-direct-insert-costed-bmssp-zeroE*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ 0\ d\ S\ B\ d'\ B'\ U\ c$

inductive-cases *charged-direct-insert-costed-bmssp-SucE*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ (Suc\ l)\ d\ S\ B\ d'\ B'\ U\ c$

lemma *charged-direct-insert-costed-partition-loop-state-mono*:
assumes *run*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs$
shows *nondecreasing-from a bs*
⟨*proof*⟩

lemma *charged-direct-insert-costed-partition-loop-state-beta-bounds*:
assumes *run*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs$
shows *bounds-le B betas*
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-success-or-threshold*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
$betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs \Longrightarrow$
$sound\text{-}label\ d \Longrightarrow$
$bmssp\text{-}pre\text{-}full\ d\ P\ B \Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow reachable\ s\ x) \Longrightarrow$
$B' = B \vee k * cap \leq card\ U$

and *charged-direct-insert-costed-bmssp-success-or-threshold-trivial*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d′ B′ U c* $\Longrightarrow$ *True*
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-cost-bound-by-child-sources-and-edge-ranges*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs* $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d P B $\Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*$) \Longrightarrow$
$\exists$ *child-sources.*
length child-sources = *length bs* $\wedge$
list-all2
(λ*S-child charged-child.* $\exists$ *c-child U-child B-child d-child B-child′.*
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child d-child B-child′ U-child c-child* $\wedge$
bmssp-pre-full d S-child B-child $\wedge$
$(\forall x \in$*S-child. reachable s x*$)$ $\wedge$
card S-child $\leq$ *M-of l* $\wedge$
$(\forall x \in$*S-child. below-bound* $(d\ x)$ *B-child*$)$)
child-sources (*range-tree-child-list P a bs*) $\wedge$
c $\leq$ *sum-list child-costs* + *sum-list* (*map card child-sources*) +
t $*$ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
h $*$ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*)) +
h $*$ *sum-list* (*map card child-sources*)
and *charged-direct-insert-costed-bmssp-cost-child-sources-and-edge-ranges-trivial*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d′ B′ U c* $\Longrightarrow$ *True*
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-child-cost-bounds-with-invariants*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs* $\Longrightarrow$
P $\subseteq$ *V* $\Longrightarrow$
k $*$ *card P* $\leq$ *cap* $\Longrightarrow$
tree-antichain P $\Longrightarrow$
$(\bigwedge$*c-child charged-child U-child S-child B-child d-child B-child′.*
$[\![$*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child d-child B-child′ U-child c-child*;
bmssp-pre-full d S-child B-child;
$\bigwedge x.\ x \in$ *S-child* $\Longrightarrow$ *reachable s x*;
card S-child $\leq$ *M-of l*;
$\bigwedge x.\ x \in$ *S-child* $\Longrightarrow$ *below-bound* $(d\ x)$ *B-child*;
k $*$ *card S-child* $\leq$ *cap*;
tree-antichain S-child$]\!]$
$\Longrightarrow$ *c-child* $\leq$ *level-range-cost-bound A R L charged-child*$) \Longrightarrow$

list-all2 (λ*c-child charged-child*.
c-child ≤ *level-range-cost-bound A R L charged-child*)
child-costs (*range-tree-child-list P a bs*)
and *charged-direct-insert-costed-bmssp-child-cost-bounds-with-invariants-trivial*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d′ B′ U c* ⟹ *True*
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-closes-level-bound-from-child-bound-with-invariants-and-edge-ranges*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*
betas bs B′ charged-Us child-outputs U c child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: ⋀*x*. *x* ∈ *P* ⟹ *reachable s x*
and *P-k-cap*: *k* ∗ *card P* ≤ *cap*
and *P-anti*: *tree-antichain P*
and *child-bound*:
⋀*c-child charged-child U-child S-child B-child d-child B-child′*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child;
bmssp-pre-full d S-child B-child;
⋀*x*. *x* ∈ *S-child* ⟹ *reachable s x*;
card S-child ≤ *M-of l*;
⋀*x*. *x* ∈ *S-child* ⟹ *below-bound* (*d x*) *B-child*;
k ∗ *card S-child* ≤ *cap*;
tree-antichain S-child⟧
⟹ *c-child* ≤ *level-range-cost-bound A R L charged-child*
and *source-factor*: *Suc h* ≤ *A*
and *source-progress*:
⋀*child-sources*.
list-all2
(λ*S-child charged-child*. ∃ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ *x*∈*S-child*. *reachable s x*) ∧
card S-child ≤ *M-of l* ∧
(∀ *x*∈*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) ⟹
sum-list (*map card child-sources*) ≤
card (*range-tree-chain P a bs B′*)
shows *c* ≤
A ∗ *Suc L* ∗ *card U* + *R* ∗ *card* (*outgoing-edges U*) +
t ∗ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
h ∗ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*))
⟨*proof*⟩

end

end
theory *BMSSP-Strict-Tie-Breaking-Extensions*
imports *BMSSP-Correctness.BMSSP-Strict-Tie-Breaking*
BMSSP-Direct-Insert-Extensions
begin

context *strict-tie-breaking-digraph*
begin

theorem *charged-direct-insert-costed-step-seen-success*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′ D a betas bs B′*
charged-Us child-outputs U-loop c-loop child-costs
and *U-def*:
U = *U-loop* $\cup$
$\{v \in$ *bound-tree S B′*. *find-pivots-label-capped k cap d S B v* =
dist s v$\}$
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* (*d x*) *B*
and *success*: $B' = B$
shows *card* (*find-pivots-seen-capped k cap d S B*) $\leq$ *card U*
⟨*proof*⟩

theorem *charged-direct-insert-costed-capped-step-threshold-if-not-success*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d′ D a betas bs B′*
charged-Us child-outputs U-loop c-loop child-costs
and *U-def*:
U = *U-loop* $\cup$
$\{v \in$ *bound-tree S B′*. *find-pivots-label-capped k cap d S B v* =
dist s v$\}$
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *not-success*: $B' \neq B$
shows $k * cap \leq$ *card U*
⟨*proof*⟩

theorem *charged-direct-insert-costed-capped-step-scan-insert-budget-from-scaled-seen-or-threshold*:

assumes *degree*: *edge-outdegree-le* Δ
and *degree-factor*: $\Delta \leq A$
and *insert-factor*: $t \leq A * k$
and *insert-scaled-factor*: $t \leq A\text{-}insert * k$
and *seen-scaled-factor*: $k * \Delta + A\text{-}insert \leq 2 * A$
and *insert*:
partition-initial-insert-cost-bound c-insert t
(*find-pivots-pivots-capped k cap d S B*)
and *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
(*find-pivots-label-capped k cap d S B*)
(*find-pivots-pivots-capped k cap d S B*) *B d' D a betas bs B'*
charged-Us child-outputs U-loop c-loop child-costs
and *U-def*:
$U = U\text{-}loop \cup$
$\{v \in$ *bound-tree S B'*. *find-pivots-label-capped k cap d S B v* =
dist s v$\}$
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *S-k-cap*: $k * card\ S \leq cap$
and *anti*: *tree-antichain S*
and *k-pos*: $0 < k$
and *seen-success*:
$B' = B \Longrightarrow$
card (*find-pivots-seen-capped k cap d S B*) $\leq$ *card U*
shows *fp-iter-capped-scan-cost k cap d S S B* + *c-insert* $\leq$
$2 * A * card\ U$
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-closes-level-bound-from-child-bound-with-invariants-and-edge-budget*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d' D a betas bs B' charged-Us child-outputs U c child-costs*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
and *P-k-cap*: $k * card\ P \leq cap$
and *P-anti*: *tree-antichain P*
and *child-bound*:
$\bigwedge$*c-child charged-child U-child S-child B-child d-child B-child'*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child d-child B-child' U-child c-child*;
bmssp-pre-full d S-child B-child;
$\bigwedge x.\ x \in$ *S-child* $\Longrightarrow$ *reachable s x*;
card S-child $\leq$ *M-of l*;
$\bigwedge x.\ x \in$ *S-child* $\Longrightarrow$ *below-bound* (*d x*) *B-child*;
$k * card\ S\text{-}child \leq cap$;

tree-antichain S-child⟧
⟹ *c-child* ≤ *level-range-cost-bound A R L charged-child*
and *source-factor*: *Suc h* ≤ *A*
and *source-progress*:
⋀*child-sources*.
list-all2
(λ*S-child charged-child*. ∃ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ *x*∈*S-child*. *reachable s x*) ∧
card S-child ≤ *M-of l* ∧
(∀ *x*∈*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) ⟹
sum-list (*map card child-sources*) ≤
card (*range-tree-chain P a bs B′*)
and *edge-budget*:
t ∗ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
h ∗ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*))
≤ *H* ∗ *card* (*outgoing-edges U*)
shows *c* ≤ *A* ∗ *Suc L* ∗ *card U* + (*R* + *H*) ∗ *card* (*outgoing-edges U*)
⟨*proof*⟩

theorem *charged-direct-insert-costed-nonbase-step-closes-level-bound-from-child-bound-with-invariants-and-edge-budget*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*
betas bs B′ charged-Us child-outputs U-loop c-loop child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: ⋀*x*. *x* ∈ *P* ⟹ *reachable s x*
and *P-k-cap*: *k* ∗ *card P* ≤ *cap*
and *P-anti*: *tree-antichain P*
and *child-bound*:
⋀*c-child charged-child U-child S-child B-child d-child B-child′*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child;
bmssp-pre-full d S-child B-child;
⋀*x*. *x* ∈ *S-child* ⟹ *reachable s x*;
card S-child ≤ *M-of l*;
⋀*x*. *x* ∈ *S-child* ⟹ *below-bound* (*d x*) *B-child*;
k ∗ *card S-child* ≤ *cap*;
tree-antichain S-child⟧
⟹ *c-child* ≤ *level-range-cost-bound A R L charged-child*
and *source-factor*: *Suc h* ≤ *A*
and *source-progress*:
⋀*child-sources*.

list-all2
(λ*S-child charged-child*. ∃ *c-child U-child B-child d-child B-child'*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child' U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ *x*∈*S-child*. *reachable s x*) ∧
card S-child ≤ *M-of l* ∧
(∀ *x*∈*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) ⟹
sum-list (*map card child-sources*) ≤
card (*range-tree-chain P a bs B'*)
and *edge-budget*:
t ∗ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
h ∗ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*))
≤ *H* ∗ *card* (*outgoing-edges U-loop*)
and *U-def*: *U* = *U-loop* ∪ *W*
and *finite-U*: *finite U*
and *scan-insert*: *c-scan-insert* ≤ *A* ∗ *card U*
and *c-def*: *c* = *c-scan-insert* + *c-loop*
shows *c* ≤ *A* ∗ *Suc* (*Suc L*) ∗ *card U* +
(*R* + *H*) ∗ *card* (*outgoing-edges U*)
⟨*proof*⟩

end

end
theory *BMSSP-Top-Level-Bounds-Extensions*
imports *BMSSP-Correctness.BMSSP-Top-Level-Bounds*
BMSSP-Strict-Tie-Breaking-Extensions
begin

context *strict-tie-breaking-digraph*
begin

theorem *charged-direct-insert-algorithm-correct*:
assumes *all-reachable*: ⋀*v*. *v* ∈ *V* ⟹ *reachable s v*
and *run*:
charged-direct-insert-costed-bmssp Δ (*bmssp-level-cap k t*) *t h k*
(*bmssp-level-cap k t l*) *l*
finite-initial-label {*s*} *Infinity d' Infinity U c*
shows *U* = *V* ∧ *sssp-correct d'*
⟨*proof*⟩

theorem *charged-direct-insert-algorithm-reachable-correct*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ (*bmssp-level-cap k t*) *t h k*
(*bmssp-level-cap k t l*) *l*

finite-initial-label $\{s\}$ *Infinity* d' *Infinity* U c
shows *sssp-correct* d'
⟨*proof*⟩

definition *charged-direct-insert-top-level-run* ::
$nat \Rightarrow nat \Rightarrow ('a \Rightarrow real) \Rightarrow 'a\ set \Rightarrow nat \Rightarrow bool$ **where**
charged-direct-insert-top-level-run $D\ N\ d'\ U\ c \longleftrightarrow$
(*let* p = *sssp-log-one-third-param* N;
q = *sssp-log-two-thirds-param* N
in charged-direct-insert-costed-bmssp D (*bmssp-level-cap* $p\ q$) $q\ p\ p$
(*bmssp-level-cap* $p\ q\ p$) p
finite-initial-label $\{s\}$ *Infinity* d' *Infinity* U c)

definition *charged-direct-insert-top-level-cost* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
charged-direct-insert-top-level-cost $D\ N\ c \longleftrightarrow$
($\exists\ d'\ U.$ *charged-direct-insert-top-level-run* $D\ N\ d'\ U\ c$)

definition *charged-direct-insert-top-level-time* :: $nat \Rightarrow nat \Rightarrow nat$ **where**
charged-direct-insert-top-level-time $D\ N =$
(*LEAST* $c.$ *charged-direct-insert-top-level-cost* $D\ N\ c$)

definition *charged-direct-insert-closed-refined-bound-log-params-fixed-degree* ::
$nat \Rightarrow nat \Rightarrow bool$ **where**
charged-direct-insert-closed-refined-bound-log-params-fixed-degree $D\ N \longleftrightarrow$
(*let* p = *sssp-log-one-third-param* N;
q = *sssp-log-two-thirds-param* N
in $\forall\ d'\ U\ c.$
charged-direct-insert-costed-bmssp D (*bmssp-level-cap* $p\ q$) $q\ p\ p$
(*bmssp-level-cap* $p\ q\ p$) p
finite-initial-label $\{s\}$ *Infinity* d' *Infinity* U $c \longrightarrow$
$c \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc* $D * p$) (λ-. q) (λ-. p)
(λ-. p) (λ-. q) (λ-. *edge-count*) *vertex-count*)

theorem *charged-direct-insert-top-level-run-correct*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
and *run*: *charged-direct-insert-top-level-run* $D\ N\ d'\ U\ c$
shows $U = V \wedge$ *sssp-correct* d'
⟨*proof*⟩

lemma *charged-direct-insert-top-level-cost-exists-from-initial-loop*:
fixes $D\ N\ p\ q\ cap\ l :: nat$
and *d-fp* :: $'a \Rightarrow real$
and $P :: 'a\ set$
defines *p-def*: $p \equiv$ *sssp-log-one-third-param* N
and *q-def*: $q \equiv$ *sssp-log-two-thirds-param* N
and *cap-def*: *cap* $\equiv$ *bmssp-level-cap* $p\ q\ p$
and *l-def*: $l \equiv p - 1$
and *d-fp-def*:

d-fp ≡ *find-pivots-label-capped p cap finite-initial-label* {*s*}
Infinity
and *P-def*:
P ≡ *find-pivots-pivots-capped p cap finite-initial-label* {*s*}
Infinity
assumes *loop*:
∃ *d′ a betas bs charged-Us child-outputs U-loop c-loop*
child-costs-loop c-insert.
charged-direct-insert-costed-partition-loop-state D
(*bmssp-level-cap p q*) *q p p cap l d-fp P Infinity d′*
(*label-partition-view d-fp P*) *a betas bs Infinity charged-Us*
child-outputs U-loop c-loop child-costs-loop ∧
complete-on d′
{*v* ∈ *bound-tree* {*s*} *Infinity*. *d-fp v* = *dist s v*} ∧
partition-initial-insert-cost-bound c-insert q P
shows ∃ *c*. *charged-direct-insert-top-level-cost D N c*
⟨*proof*⟩

lemma *charged-direct-insert-top-level-time-witness*:
assumes *charged-direct-insert-top-level-cost D N c*
shows *charged-direct-insert-top-level-cost D N*
(*charged-direct-insert-top-level-time D N*)
⟨*proof*⟩

lemma *charged-direct-insert-top-level-time-le*:
assumes *charged-direct-insert-top-level-cost D N c*
shows *charged-direct-insert-top-level-time D N* ≤ *c*
⟨*proof*⟩

theorem *charged-direct-insert-algorithm-correct-and-refined-graph-bound-log-params-fixed-degree-from-closed-bound*:
fixes *N D p q* :: *nat*
defines *p-def*: *p* ≡ *sssp-log-one-third-param N*
and *q-def*: *q* ≡ *sssp-log-two-thirds-param N*
assumes *all-reachable*: ⋀*v*. *v* ∈ *V* ⟹ *reachable s v*
and *closed-bound*:
charged-direct-insert-closed-refined-bound-log-params-fixed-degree D N
and *run*:
charged-direct-insert-costed-bmssp D (*bmssp-level-cap p q*) *q p p*
(*bmssp-level-cap p q p*) *p*
finite-initial-label {*s*} *Infinity d′ Infinity U c*
shows *U* = *V* ∧ *sssp-correct d′* ∧
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc D* ∗ *p*) (λ-. *q*) (λ-. *p*)
(λ-. *p*) (λ-. *q*) (λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

theorem *charged-direct-insert-top-level-run-correct-and-refined-bound-log-params-fixed-degree-from-closed-bound*:

assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
and *closed-bound*:
charged-direct-insert-closed-refined-bound-log-params-fixed-degree $D\ N$
and *run*: *charged-direct-insert-top-level-run* $D\ N\ d'\ U\ c$
shows $U = V \wedge$ *sssp-correct* $d' \wedge$
$c \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc* $D *$ *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

theorem *charged-direct-insert-top-level-cost-refined-bound-log-params-fixed-degree-from-closed-bound*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
and *closed-bound*:
charged-direct-insert-closed-refined-bound-log-params-fixed-degree $D\ N$
and *cost*: *charged-direct-insert-top-level-cost* $D\ N\ c$
shows $c \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc* $D *$ *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

theorem *charged-direct-insert-top-level-cost-refined-bound-log-params-fixed-degree*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
and *degree*: *edge-outdegree-le* D
and *closed-bound*:
charged-direct-insert-closed-refined-bound-log-params-fixed-degree $D\ N$
and *cost*: *charged-direct-insert-top-level-cost* $D\ N\ c$
shows $c \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc* $D *$ *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-one-third-param* N)
(λ-. *sssp-log-two-thirds-param* N)
(λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

theorem *charged-direct-insert-top-level-time-refined-bound-log-params-fixed-degree-from-closed-bound*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
and *closed-bound*:
charged-direct-insert-closed-refined-bound-log-params-fixed-degree $D\ N$

and *run-exists*: *charged-direct-insert-top-level-cost D N c*
shows *charged-direct-insert-top-level-time D N* ≤
bmssp-refined-graph-time-bound
(λ-. *Suc D* ∗ *sssp-log-one-third-param N*)
(λ-. *sssp-log-two-thirds-param N*)
(λ-. *sssp-log-one-third-param N*)
(λ-. *sssp-log-one-third-param N*)
(λ-. *sssp-log-two-thirds-param N*)
(λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

lemma *charged-direct-insert-top-level-cost-exists-if-source-pivot-finishes*:
fixes *D N p q cap l* :: *nat*
and *d-fp* :: *'a* ⇒ *real*
and *P* :: *'a set*
and *Bmax* :: *real*
defines *p-def*: *p* ≡ *sssp-log-one-third-param N*
and *q-def*: *q* ≡ *sssp-log-two-thirds-param N*
and *cap-def*: *cap* ≡ *bmssp-level-cap p q p*
and *l-def*: *l* ≡ *p* − *1*
and *d-fp-def*:
d-fp ≡ *find-pivots-label-capped p cap finite-initial-label* {*s*}
Infinity
and *P-def*:
P ≡ *find-pivots-pivots-capped p cap finite-initial-label* {*s*}
Infinity
assumes *P-source*: *P* = {*s*}
and *source-below*: *d-fp s* < *Bmax*
and *Bmax-nonneg*: *0* ≤ *Bmax*
and *child-pre*: *bmssp-pre-full d-fp* {*s*} (*Fin Bmax*)
and *below-threshold*: *card* (*bound-tree P* (*Fin 0*)) < *p* ∗ *cap*
and *child*:
charged-direct-insert-costed-bmssp D (*bmssp-level-cap p q*) *q p p*
cap l d-fp {*s*} (*Fin Bmax*) (*dist s*) (*Fin Bmax*) *U-child c-child*
and *direct-empty*:
edge-relaxation-pairs-in-bound (*dist s*) (*range-tree P 0* (*Fin Bmax*))
Bmax Infinity = []
shows ∃ *c*. *charged-direct-insert-top-level-cost D N c*
⟨*proof*⟩

definition *charged-direct-insert-source-pivot-finishes* :: *nat* ⇒ *nat* ⇒ *bool* **where**
charged-direct-insert-source-pivot-finishes D N ⟷
(*let p* = *sssp-log-one-third-param N*;
q = *sssp-log-two-thirds-param N*;
cap = *bmssp-level-cap p q p*;
l = *p* − *1*;
d-fp = *find-pivots-label-capped p cap finite-initial-label* {*s*}
Infinity;
P = *find-pivots-pivots-capped p cap finite-initial-label* {*s*}

```
      Infinity
    in ∃ Bmax U-child c-child.
      P = {s} ∧
      d-fp s < Bmax ∧
      0 ≤ Bmax ∧
      bmssp-pre-full d-fp {s} (Fin Bmax) ∧
      card (bound-tree P (Fin 0)) < p * cap ∧
      charged-direct-insert-costed-bmssp D (bmssp-level-cap p q) q p p
        cap l d-fp {s} (Fin Bmax) (dist s) (Fin Bmax)
        U-child c-child ∧
      edge-relaxation-pairs-in-bound (dist s) (range-tree P 0 (Fin Bmax))
        Bmax Infinity = [])

lemma charged-direct-insert-top-level-cost-exists-if-source-pivot-finishes-pred:
  assumes charged-direct-insert-source-pivot-finishes D N
  shows ∃ c. charged-direct-insert-top-level-cost D N c
⟨proof⟩

lemma eventually-charged-direct-insert-top-level-cost-if-source-pivot-finishes:
  assumes
    eventually (λN. charged-direct-insert-source-pivot-finishes D N) at-top
  shows eventually (λN. ∃ c. charged-direct-insert-top-level-cost D N c)
    at-top
  ⟨proof⟩

lemma charged-direct-insert-top-level-cost-exists-if-top-pivots-empty:
  assumes pivots-empty:
    find-pivots-pivots-capped (sssp-log-one-third-param N)
      (bmssp-level-cap (sssp-log-one-third-param N)
        (sssp-log-two-thirds-param N) (sssp-log-one-third-param N))
      finite-initial-label {s} Infinity = {}
  shows ∃ c. charged-direct-insert-top-level-cost D N c
⟨proof⟩

lemma eventually-charged-direct-insert-top-level-cost-from-empty-top-pivots:
  eventually (λN. ∃ c. charged-direct-insert-top-level-cost D N c) at-top
  ⟨proof⟩

end

end
theory BMSSP-Verified-Runtime
  imports BMSSP-Top-Level-Bounds-Extensions
begin
```

6 Runtime Bound for a Verified-Correct BMSSP Run

The top-level theory *BMSSP-Top-Level-Bounds* proves two facts that, taken separately, leave a gap a reader is right to worry about.

- *exact-concrete-top-level-run-correct-and-refined-bound-log-params-fixed-degree* shows that *one* costed run is simultaneously correct and cost-bounded: for a single *exact-concrete-top-level-run* producing output label d', completed set U and cost c, we have $U = V$, *sssp-correct* d', and a closed-form bound on c. This is a per-run statement in closed form.

- *bmssp-runtime-bigo-target* lifts the closed-form bound to the asymptotic headline $O(m * (ln\ n)\ powr\ (2/3))$, but it is stated about *T-bmssp* D alone — the *least cost* over valid runs. As a number, *T-bmssp* $D\ N$ carries no evidence that the run achieving it computes shortest paths.

  The asymptotic headline therefore speaks about a least-cost integer, while the correctness theorem speaks about an individual run; nothing yet says the run whose cost realises the asymptotic bound is the same run that is proved correct. This theory closes that gap. Its central observation is that *T-bmssp* $D\ N$, being a least cost, is realised by an *actual* run (by *exact-concrete-top-level-time-witness*), and that this realising run is governed by the per-run correctness theorem. Consequently the asymptotic running-time bound is a statement about runs that are themselves verified to return exact single-source shortest-path distances.

context *strict-tie-breaking-digraph*
begin

First, the least-cost value *T-bmssp* $D\ N$ is realised by a concrete top-level run, and that run satisfies full single-source shortest-path correctness. The hypothesis is exactly the one already used throughout the top-level development: at least one valid run exists for the given size.

theorem *T-bmssp-realised-by-verified-run*:
  assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable* $s\ v$
    and *degree*: *edge-outdegree-le* D
    and *run-exists*: *exact-concrete-top-level-cost* $D\ N\ c$
  shows $\exists\, d'\ U.$
      exact-concrete-top-level-run $D\ N\ d'\ U$ (*T-bmssp* $D\ N$) $\wedge$
      $U = V \wedge$ *sssp-correct* d'
⟨*proof*⟩

The realising run exists for all but finitely many sizes: the same eventual-existence fact that powers the asymptotic proof (*eventually-exact-concrete-*

top-level-cost-from-empty-top-pivots) guarantees a valid run, hence a verified realising run, eventually in N.

theorem *eventually-T-bmssp-realised-by-verified-run*:
  assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable s v*
    and *degree*: *edge-outdegree-le D*
  shows *eventually*
    $(\lambda N.\ \exists\, d'\ U.$
        exact-concrete-top-level-run D N d′ U (*T-bmssp D N*) $\wedge$
        $U = V \wedge$ *sssp-correct d′*) *at-top*
⟨*proof*⟩

end

The public headline of this theory lives in the same asymptotic locale as *bmssp-runtime-headline-instance.bmssp-runtime-bigo-target*. It states the deterministic BMSSP running-time bound together with the witness that the running time belongs to runs returning exact shortest-path distances: for all but finitely many sizes the checked running time *T-bmssp N* is the cost of a concrete run whose output is *sssp-correct* and whose completed set is the whole vertex set.

context *bmssp-runtime-headline-instance*
begin

theorem *verified-bmssp-runtime-bigo-target*:
  shows $(\lambda n.$ *real* (*T-bmssp n*)) $\in$
      $O(\lambda n.$ *real* (*m n*) $*$ (*ln* (*real n* + *2*)) *powr* (*2* / *3*)) $\wedge$
    eventually
      $(\lambda N.\ \exists\, d'\ U.$
          bounded.exact-concrete-top-level-run D N d′ U (*T-bmssp N*) $\wedge$
          $U = V \wedge$ *bounded.sssp-correct d′*) *at-top*
⟨*proof*⟩

Read together, the two conjuncts of *verified-bmssp-runtime-bigo-target* say that the $O(\lambda n.$ *real* (*m n*) $*$ *ln* (*real n* + *2*) *powr* (*2* / *3*)) running-time bound is not a statement about an abstract least cost in isolation: the cost it bounds is, for all but finitely many sizes, achieved by an explicit BMSSP run that provably computes the exact single-source shortest-path distance function on every reachable vertex. This is the sense in which the entry delivers a single algorithm that is at once correct and within the deterministic sorting-barrier running time.

end

end
theory *BMSSP-Runtime-Instance-Branching*
  imports *BMSSP-Verified-Runtime*
    BMSSP-Correctness.BMSSP-Executable-Base-Case
begin

7 A Second, Structurally Different Non-Vacuous Instance

The path instance in *BMSSP-Runtime-Instance* certifies the runtime locale on a single chain. To show that inhabitation does not rely on the degenerate linear shape, this theory discharges the same locale on a structurally different graph: the out-star $1 \leftarrow 0 \rightarrow 2$ with two leaves, in which the source has outdegree two and there are two incomparable shortest paths. The proof structure mirrors the path instance: the verified simple-walk enumerator pins down the unique shortest walk to each vertex.

definition *star-vs* :: *nat list* **where**
star-vs = [*0*, *1*, *2*]

definition *star-es* :: (*nat* × *nat*) *list* **where**
star-es = [(*0*, *1*), (*0*, *2*)]

definition *star-V* :: *nat set* **where**
star-V = *set star-vs*

definition *star-E* :: (*nat* × *nat*) *set* **where**
star-E = *set star-es*

definition *star-weight* :: *nat* ⇒ *nat* ⇒ *real* **where**
star-weight u v = (*if* (*u*, *v*) ∈ *star-E then 1 else 0*)

lemma *star-vs-V*: *set star-vs* = *star-V*
⟨*proof*⟩

lemma *star-es-E*: *set star-es* = *star-E*
⟨*proof*⟩

lemma *star-positive-weight*: (*u*, *v*) ∈ *star-E* ⟹ *0* < *star-weight u v*
⟨*proof*⟩

lemma *star-finite-weighted-digraph*:
finite-weighted-digraph star-V star-E star-weight 0
⟨*proof*⟩

interpretation *sfw*: *finite-weighted-digraph star-V star-E star-weight 0*
⟨*proof*⟩

lemma *star-walk-lists*:
exec-simple-walks-betw star-vs star-es 0 0 = [[*0*]]
exec-simple-walks-betw star-vs star-es 0 (*Suc 0*) = [[*0*, *1*]]
exec-simple-walks-betw star-vs star-es 0 2 = [[*0*, *2*]]
⟨*proof*⟩

lemma *star-simple-walk-sets*:

$\{p.\ sfw.simple\text{-}walk\text{-}betw\ 0\ p\ 0\} = \{[0]\}$
$\{p.\ sfw.simple\text{-}walk\text{-}betw\ 0\ p\ (Suc\ 0)\} = \{[0,\ 1]\}$
$\{p.\ sfw.simple\text{-}walk\text{-}betw\ 0\ p\ 2\} = \{[0,\ 2]\}$
⟨*proof*⟩

lemma *star-walk-vertex*:
assumes *sfw.simple-walk-betw 0 p v*
shows $v \in$ *star-V*
⟨*proof*⟩

lemma *star-unique-shortest-walk*:
assumes *sfw.shortest-walk 0 p v* **and** *sfw.shortest-walk 0 q v*
shows $p = q$
⟨*proof*⟩

lemma *star-all-reachable*:
assumes $v \in$ *star-V*
shows *sfw.reachable 0 v*
⟨*proof*⟩

interpretation *susd*: *unique-shortest-digraph star-V star-E star-weight 0*
⟨*proof*⟩

lemma *star-edge-outdegree*: *susd.edge-outdegree-le 2*
⟨*proof*⟩

interpretation *sbr*: *bounded-reduced-positive-instance star-V star-E star-weight 0 2*
⟨*proof*⟩

As with the path, the out-star yields the closed, assumption-free running-time bound and its verified-correct strengthening. The two instances share no structural feature beyond the locale assumptions, so together they witness that the BMSSP runtime headline holds across genuinely different graph shapes.

lemmas *star-runtime-bigo-target* =
sbr.runtime-headline.bmssp-runtime-bigo-target

lemmas *star-verified-runtime-bigo-target* =
sbr.runtime-headline.verified-bmssp-runtime-bigo-target

end
theory *BMSSP-Path-Family-Extended*
imports *BMSSP-Correctness.BMSSP-Path-Family BMSSP-Verified-Runtime*
begin

8 A Size-Parametric BMSSP Running-Time Bound over a Growing Graph Family

The headline *bmssp-runtime-headline-instance.bmssp-runtime-bigo-target* is stated for a graph fixed by the locale, with the asymptotic variable feeding only the internal logarithmic schedule; the edge-count factor in its target is a constant. This theory removes that limitation by exhibiting a genuine *family* of graphs whose size grows without bound: the unit-weight directed path P_k on vertices $0, \ldots, k$ with edges $i \to i + 1$. For this family $card\ (E_k) = k$ and $card\ (V_k) = k + 1$ both grow with k, so k is a true graph-size variable, and we prove the BMSSP running time is $O(k \cdot (ln\ k)^{2/3})$.

8.1 The path family

definition *path-k-V* :: *nat* ⇒ *nat set* **where**
path-k-V k = {*0*..*k*}

definition *path-k-E* :: *nat* ⇒ (*nat* × *nat*) *set* **where**
path-k-E k = (λ*i*. (*i*, *Suc i*)) ‘ {*0*..<*k*}

definition *path-k-weight* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *real* **where**
path-k-weight k u v = (*if* (*u*, *v*) ∈ *path-k-E k then 1 else 0*)

8.2 Structural and cardinality lemmas

lemma *path-k-V-eq*: *path-k-V k* = {*0*..*k*}
⟨*proof*⟩

lemma *path-k-E-iff*: (*u*, *v*) ∈ *path-k-E k* ⟷ *v* = *Suc u* ∧ *u* < *k*
⟨*proof*⟩

lemma *path-k-no-incoming-0*: (*u*, *0*) ∉ *path-k-E k*
⟨*proof*⟩

lemma *path-k-incoming-unique*:
(*u*, *j*) ∈ *path-k-E k* ⟷ (*0* < *j* ∧ *j* ≤ *k* ∧ *u* = *j* − *1*)
⟨*proof*⟩

lemma *path-k-card-V*: *card* (*path-k-V k*) = *Suc k*
⟨*proof*⟩

lemma *path-k-card-E*: *card* (*path-k-E k*) = *k*
⟨*proof*⟩

lemma *path-k-finite-V*: *finite* (*path-k-V k*)
⟨*proof*⟩

lemma *path-k-finite-E*: *finite* (*path-k-E k*)

⟨*proof*⟩

8.3 Easy locale obligations, uniform in k

lemma *path-k-finite-weighted-digraph*:
finite-weighted-digraph (*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0*
⟨*proof*⟩

lemma *path-k-positive-weight*:
(*u*, *v*) ∈ *path-k-E k* ⟹ *0* < *path-k-weight k u v*
⟨*proof*⟩

8.4 The unconditional cost-recurrence bound (FLOOR)

The function *path-size-cost* is exactly the closed cost expression that bounds a BMSSP run on P_k at schedule parameter k, with the graph cardinalities substituted: edge slot k and vertex slot *Suc k*. Its asymptotic bound holds unconditionally — no run existence and no shortest-walk uniqueness are required — and already exhibits a genuine size-parametric $O(k \cdot (ln\ k)^{2/3})$ growth in which both the edge count k and the vertex count *Suc k* grow with the asymptotic variable.

definition *path-size-cost* :: *nat* ⇒ *nat* **where**
path-size-cost k =
bmssp-refined-graph-time-bound
(λ-. *Suc 1* ∗ *sssp-log-one-third-param k*)
(λ-. *sssp-log-two-thirds-param k*)
(λ-. *sssp-log-one-third-param k*)
(λ-. *sssp-log-one-third-param k*)
(λ-. *sssp-log-two-thirds-param k*)
(λ-. *k*) (*Suc k*)

theorem *path-size-cost-bigo-size*:
(λ*k*. *real* (*path-size-cost k*))
∈ *O*(λ*k*. *real k* ∗ (*ln* (*real k* + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

8.5 The forced-degenerate (empty-pivot) regime is provably avoided

The only unconditional source of run existence in the development, *strict-tie-breaking-digraph.eventually-exact-concrete-top-level-cost-from-empty-top-pivots*, fires through *unique-shortest-digraph.find-pivots-pivots-capped-singleton-empty-if-params-exceed-vertex-count*, whose hypothesis requires the schedule parameter to *exceed* the vertex count. On P_k at schedule k this hypothesis is *Suc k* < *sssp-log-one-third-param k*, which fails for all large k because the one-third logarithmic factor grows far slower than

k. Hence the empty-pivot no-op cannot be the run whose cost the family bound governs: any such run does genuine top-level work.

lemma *sssp-log-one-third-param-le-linear*:
eventually (λk. *sssp-log-one-third-param* $k \leq$ *Suc* k) *at-top*
⟨*proof*⟩

theorem *path-family-not-forced-degenerate*:
eventually
(λk. $\neg$ (*card* (*path-k-V* k) $<$ *sssp-log-one-third-param* k)) *at-top*
⟨*proof*⟩

8.6 Unique shortest walks on the path family

context fixes k :: *nat* **begin**

interpretation *pk*: *finite-weighted-digraph*
path-k-V k *path-k-E* k *path-k-weight* k *0*
⟨*proof*⟩

lemma *path-k-canonical-walk*:
$j \leq k \Longrightarrow$ *pk.simple-walk-betw 0* [*0..<Suc j*] *j*
⟨*proof*⟩

lemma *path-k-all-reachable*:
$v \in$ *path-k-V* $k \Longrightarrow$ *pk.reachable 0 v*
⟨*proof*⟩

lemma *path-k-simple-walk-unique*:
pk.simple-walk-betw 0 p j $\Longrightarrow$ $j \leq k \Longrightarrow p =$ [*0..<Suc j*]
⟨*proof*⟩

lemma *path-k-unique-shortest-walk*:
pk.shortest-walk 0 p v $\Longrightarrow$ *pk.shortest-walk 0 q v* $\Longrightarrow p = q$
⟨*proof*⟩

lemma *path-k-bounded-instance*:
bounded-reduced-positive-instance (*path-k-V* k) (*path-k-E* k)
(*path-k-weight* k) *0 1*
⟨*proof*⟩

interpretation *bpi*: *bounded-reduced-positive-instance*
path-k-V k *path-k-E* k *path-k-weight* k *0 1*
⟨*proof*⟩

lemma *path-k-charged-top-level-cost-exists-if-root-run*:
assumes *root*:
$\exists\, d'\ U\ c.$
bpi.charged-direct-insert-costed-bmssp (*Suc 0*)
(*bmssp-level-cap* (*sssp-log-one-third-param* k)

(*sssp-log-two-thirds-param k*))
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*)
(*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
(*sssp-log-one-third-param k*)
pk.finite-initial-label {*0*} *Infinity d' Infinity U c*
shows ∃ *c. strict-tie-breaking-digraph.charged-direct-insert-top-level-cost*
(*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0 1 k c*
⟨*proof*⟩

lemma *path-k-charged-root-run-exists-if-initial-loop*:
assumes *loop*:
∃ *d' a betas bs charged-Us child-outputs U-loop c-loop*
child-costs-loop c-insert.
bpi.charged-direct-insert-costed-partition-loop-state
(*Suc 0*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*))
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*)
(*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
(*sssp-log-one-third-param k − 1*)
(*bpi.find-pivots-label-capped* (*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
pk.finite-initial-label {*0*} *Infinity*)
(*bpi.find-pivots-pivots-capped* (*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
pk.finite-initial-label {*0*} *Infinity*)
Infinity d'
(*bpi.label-partition-view*
(*bpi.find-pivots-label-capped* (*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
pk.finite-initial-label {*0*} *Infinity*)
(*bpi.find-pivots-pivots-capped* (*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))
pk.finite-initial-label {*0*} *Infinity*))
a betas bs Infinity charged-Us child-outputs U-loop c-loop
child-costs-loop ∧
pk.complete-on d'
{*v* ∈ *pk.bound-tree* {*0*} *Infinity.*
bpi.find-pivots-label-capped (*sssp-log-one-third-param k*)
(*bmssp-level-cap* (*sssp-log-one-third-param k*)
(*sssp-log-two-thirds-param k*) (*sssp-log-one-third-param k*))

```
        pk.finite-initial-label {0} Infinity v = pk.dist 0 v} ∧
      bpi.partition-initial-insert-cost-bound c-insert
       (sssp-log-two-thirds-param k)
       (bpi.find-pivots-pivots-capped (sssp-log-one-third-param k)
        (bmssp-level-cap (sssp-log-one-third-param k)
         (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
        pk.finite-initial-label {0} Infinity)
  shows ∃ d' U c.
    bpi.charged-direct-insert-costed-bmssp (Suc 0)
     (bmssp-level-cap (sssp-log-one-third-param k)
      (sssp-log-two-thirds-param k))
     (sssp-log-two-thirds-param k) (sssp-log-one-third-param k)
     (sssp-log-one-third-param k)
     (bmssp-level-cap (sssp-log-one-third-param k)
      (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
     (sssp-log-one-third-param k)
     pk.finite-initial-label {0} Infinity d' Infinity U c
⟨proof⟩

lemma path-k-charged-top-level-cost-exists-if-initial-loop:
  assumes loop:
     ∃ d' a betas bs charged-Us child-outputs U-loop c-loop
        child-costs-loop c-insert.
       bpi.charged-direct-insert-costed-partition-loop-state
        (Suc 0)
        (bmssp-level-cap (sssp-log-one-third-param k)
         (sssp-log-two-thirds-param k))
        (sssp-log-two-thirds-param k) (sssp-log-one-third-param k)
        (sssp-log-one-third-param k)
        (bmssp-level-cap (sssp-log-one-third-param k)
         (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
        (sssp-log-one-third-param k − 1)
        (bpi.find-pivots-label-capped (sssp-log-one-third-param k)
         (bmssp-level-cap (sssp-log-one-third-param k)
          (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
         pk.finite-initial-label {0} Infinity)
        (bpi.find-pivots-pivots-capped (sssp-log-one-third-param k)
         (bmssp-level-cap (sssp-log-one-third-param k)
          (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
         pk.finite-initial-label {0} Infinity)
        Infinity d'
        (bpi.label-partition-view
         (bpi.find-pivots-label-capped (sssp-log-one-third-param k)
          (bmssp-level-cap (sssp-log-one-third-param k)
           (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
          pk.finite-initial-label {0} Infinity)
         (bpi.find-pivots-pivots-capped (sssp-log-one-third-param k)
          (bmssp-level-cap (sssp-log-one-third-param k)
           (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
```

```
      pk.finite-initial-label {0} Infinity))
    a betas bs Infinity charged-Us child-outputs U-loop c-loop
    child-costs-loop ∧
  pk.complete-on d'
    {v ∈ pk.bound-tree {0} Infinity.
      bpi.find-pivots-label-capped (sssp-log-one-third-param k)
        (bmssp-level-cap (sssp-log-one-third-param k)
          (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
        pk.finite-initial-label {0} Infinity v = pk.dist 0 v} ∧
  bpi.partition-initial-insert-cost-bound c-insert
    (sssp-log-two-thirds-param k)
    (bpi.find-pivots-pivots-capped (sssp-log-one-third-param k)
      (bmssp-level-cap (sssp-log-one-third-param k)
        (sssp-log-two-thirds-param k) (sssp-log-one-third-param k))
      pk.finite-initial-label {0} Infinity)
shows ∃ c. strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
  (path-k-V k) (path-k-E k) (path-k-weight k) 0 1 k c
⟨proof⟩

end
```

8.7 The exact-concrete algorithm-level running-time bound

Unlike the unconditional floor *path-size-cost-bigo-size*, which bounds the closed cost expression with the graph cardinalities substituted, this theorem bounds the least exact-concrete top-level cost *path-T-bmssp*. Its hypothesis is intentionally explicit: for all but finitely many k, at least one exact-concrete top-level run must exist on P_k. The separate totality-obstruction theory shows that this hypothesis is not dischargeable for the exact-concrete relation on this family; the theorem is therefore retained as a conditional comparison result, not as a non-vacuous algorithm-level claim.

```
definition path-T-bmssp :: nat ⇒ nat where
  path-T-bmssp k = strict-tie-breaking-digraph.T-bmssp
    (path-k-V k) (path-k-E k) (path-k-weight k) 0 1 k

theorem eventually-path-family-charged-direct-insert-top-level-cost-if-source-pivot-finishes:
  assumes source-pivot-finishes:
    eventually
      (λk. strict-tie-breaking-digraph.charged-direct-insert-source-pivot-finishes
        (path-k-V k) (path-k-E k) (path-k-weight k) 0 1 k) at-top
  shows eventually
    (λk. ∃ c. strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
      (path-k-V k) (path-k-E k) (path-k-weight k) 0 1 k c) at-top
  ⟨proof⟩

theorem bmssp-path-family-runtime-bigo-size:
  assumes runs-exist:
```

eventually (λk. ∃ c. *strict-tie-breaking-digraph.exact-concrete-top-level-cost*
(*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0 1 k c*) *at-top*
shows (λk. *real* (*path-T-bmssp k*))
∈ *O*(λk. *real k* ∗ (*ln* (*real k* + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

8.8 A charged path-family theorem waiting on charged cost bounds

The charged direct-insert top-level relation is the intended successor for the path-family integration. On this branch we can already state and check the exact conditional bridge needed by downstream JAR-facing documentation: if charged top-level runs eventually exist on P_k and every such charged cost is bounded by the same refined graph-time expression, then the least charged cost has the size-parametric target. The second assumption is the charged analogue of the refined-bound theorem still expected from the runtime proof branch.

definition *path-charged-direct-insert-time* :: *nat* ⇒ *nat* **where**
path-charged-direct-insert-time k =
strict-tie-breaking-digraph.charged-direct-insert-top-level-time
(*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0 1 k*

theorem *bmssp-path-family-charged-direct-insert-runtime-bigo-size-if-cost-bounded*:
assumes *charged-runs-exist*:
eventually
(λk. ∃ c.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0 1 k c*) *at-top*
and *charged-cost-bound*:
eventually
(λk. ∀ c.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V k*) (*path-k-E k*) (*path-k-weight k*) *0 1 k c* ⟶
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param k*)
(λ-. *sssp-log-two-thirds-param k*)
(λ-. *sssp-log-one-third-param k*)
(λ-. *sssp-log-one-third-param k*)
(λ-. *sssp-log-two-thirds-param k*)
(λ-. *k*) (*Suc k*)) *at-top*
shows (λk. *real* (*path-charged-direct-insert-time k*))
∈ *O*(λk. *real k* ∗ (*ln* (*real k* + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

end
theory *BMSSP-Cost-Totality*
imports *BMSSP-Path-Family-Extended*
begin

9 On the Totality of the Exact-Concrete Cost Relation

This theory records a precise, machine-checked structural fact about the exact-concrete cost relation *unique-shortest-digraph.exact-concrete-bmssp* and its companion loop relation *unique-shortest-digraph.exact-concrete-partition-loop-state*.

The intended goal was to discharge the run-existence hypothesis *runs-exist* of *bmssp-path-family-runtime-bigo-size* unconditionally, by constructing a terminating partition-loop witness for the path family. The construction reduces, via *unique-shortest-digraph.exact-concrete-bmssp-Suc-exists-from-finite-initial-loop-cases*, to building a partition-loop state whose loop iterates over the successive half-open distance ranges of the pivot tree.

The lemmas below show why that loop cannot iterate more than once under a sound label. The single inductive rule *unique-shortest-digraph.exact-concrete-partition-loop-state* uses for a genuine step (Exact-Concrete-State-Step) carries the hypothesis that the recursive child output equals the incremental range tree *range-tree P a* (*Fin b*). But child correctness, available whenever the carried label is sound, pins that same child output to the cumulative bounded tree *bound-tree P* (*Fin b*). Since *range-tree P a* (*Fin b*) is by definition the set difference *bound-tree P* (*Fin b*) − *bound-tree P* (*Fin a*), the two can agree only when the lower prefix *bound-tree P* (*Fin a*) is empty. For a reachable singleton source rooted at s this prefix is empty exactly when the lower bound is zero, because the source itself has distance zero and therefore lies in every prefix with a strictly positive bound.

The consequence is that a sound exact-concrete partition loop admits a genuine step only at lower bound zero; any second genuine step, which would necessarily occur at a strictly positive lower bound, is impossible. A complete top-level run, whose returned bound is *Infinity*, requires the loop to terminate in the Done case with an emptied partition; with non-empty pivots that demands the single permitted step to leave no residual frontier, i.e. its one child must already solve the entire bounded tree. These observations isolate the obstruction to the unconditional totality result precisely in the shape of the State-Step rule.

Consequence for the path-family proof route. On the unit-weight directed path P_k, a multi-step exact-concrete construction would require a second genuine loop step at a positive lower bound. The two rigidity lemmas below rule out precisely that step. We therefore retain *bmssp-path-family-runtime-bigo-size* only as a conditional theorem and do not use it as evidence for a non-vacuous path-family runtime claim.

The unconditional floor *path-size-cost-bigo-size*, which bounds the closed cost expression with the graph cardinalities substituted, is unaffected. The non-vacuous theorem in `BMSSP_NonVacuous_Family.thy` instead uses the

charged relation and a decoupled schedule. Relaxing the exact-concrete State-Step rule to mirror the more permissive operational loop would be a separate change to the imported cost relation and is not undertaken here.

context *unique-shortest-digraph*
begin

Step rigidity. In a sound exact-concrete loop step the recursive child output is the cumulative bounded tree, so equating it with the incremental range tree forces the lower prefix tree to be empty.

lemma *exact-concrete-step-forces-empty-prefix*:
assumes *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P* (*Fin Bmax*)
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
and *pull-split*: *S-pull* = *split-below d P beta*
and *beta-bound*: *bound-le* (*Fin beta*) (*Fin Bmax*)
and *child-pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
and *a-le-b*: $a \leq b$
and *child*:
exact-concrete-bmssp Δ *M-of t h k cap l d S-pull* (*Fin beta*)
d-child (*Fin b*) *U-child c-child*
and *U-child-eq*: *U-child* = *range-tree P a* (*Fin b*)
shows *bound-tree P* (*Fin a*) = {}
⟨*proof*⟩

end

Two locale-free interface facts about *pull-separates*. The first is cap monotonicity: weakening the pull capacity preserves a pull. The second is the existence of a singleton pull at any capacity $M \geq 1$, given a key whose value strictly dominates every other key's value. The residual queue in the existential is annotated *'k partition-view* so that the record extension slot is fixed to *unit*; without the annotation the concrete record witness fails to unify with the fresh extension type variable that the bare existential introduces.

lemma *pull-separates-mono-cap*:
assumes *pull-separates D M B S x D'* **and** $M \leq M'$
shows *pull-separates D M' B S x D'*
⟨*proof*⟩

lemma *pull-separates-singleton-exists-abstract*:
assumes *fin*: *finite* (*keys-of D*)
and *u*: $u \in$ *keys-of D*
and *umin*: $\bigwedge v.\ v \in$ *keys-of D* $\Longrightarrow v \neq u \Longrightarrow$ *value-of D u* $<$ *value-of D v*
and *M1*: $1 \leq M$
and *upper*: *value-of D u* $< B$
shows $\exists$ *beta* (*D'*::*'k partition-view*).
pull-separates (*D*::*'k partition-view*) *M B* {*u*} *beta D'*
⟨*proof*⟩

context *strict-tie-breaking-digraph*
begin

Source persistence. The source lies in every prefix tree with a strictly positive bound, because its true distance is zero.

lemma *source-in-positive-prefix*:
  assumes *a-pos*: $0 < a$
  shows $s \in$ *bound-tree* $\{s\}$ (*Fin a*)
⟨*proof*⟩

The obstruction, combining step rigidity with source persistence. For the singleton source a genuine exact-concrete loop step is impossible at any strictly positive lower bound. Equivalently every genuine step must occur at lower bound zero, so a sound loop performs at most one genuine step.

lemma *no-genuine-exact-concrete-step-at-positive-lower-bound*:
  assumes *sound*: *sound-label d*
    and *pre*: *bmssp-pre-full d* $\{s\}$ (*Fin Bmax*)
    and *pull-split*: *S-pull* = *split-below d* $\{s\}$ *beta*
    and *beta-bound*: *bound-le* (*Fin beta*) (*Fin Bmax*)
    and *child-pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
    and *a-pos*: $0 < a$
    and *a-le-b*: $a \leq b$
    and *child*:
      exact-concrete-bmssp Δ *M-of t h k cap l d S-pull* (*Fin beta*)
        d-child (*Fin b*) *U-child c-child*
    and *U-child-eq*: *U-child* = *range-tree* $\{s\}$ *a* (*Fin b*)
  shows *False*
⟨*proof*⟩

The relaxed (charged) model: isolating the remaining obligation. The rigidity analysis above explains why the *exact-concrete* relation admits no genuine step at a positive lower bound, and hence why its size-parametric running-time bound is vacuous on the path family. The permissive *charged-direct-insert-costed-bmssp* relation removes that rigidity — its State-Step rule (rule *Charged-Direct-Insert-Step*) leaves the child output *U-child* free and only records the incremental range tree *range-tree P a* (*Fin b*) as the cost-accounting slice *charged-child*. Runs therefore exist; the open question is whether the charged per-level *cost* bound goes through.

That bound is gated on a single arithmetic obligation, the charged *source-progress*: for each loop child the pulled source set must be dominated in cardinality by its range-tree slice, *card* (*split-below d P beta*) $\leq$ *card* (*range-tree P a* (*Fin b*)). Summed along the loop via *sum-card-dominated-by-range-tree-child-list-le-chain* this is exactly what bounds the cumulative pull cost by the chain cardinality, i.e. linearly in the vertex count.

The direct-insert development discharges this for free only because its rigid rule equates the child output with the slice; by the same rigidity that route is vacuous (the lemmas above). In the charged model the child output

is the cumulative *bound-tree P* (*Fin b*) (no lower bound), so the slice bound is genuine new information. The lemmas below reduce it to a single named side-condition — *pivot settledness*: the pulled sources avoid the already-settled prefix *bound-tree P* (*Fin a*) below the step's lower bound *a*. Geometrically: a pivot pulled at the band starting at *a* has true distance at least *a*.

This side-condition is *not* derivable from the local State-Step hypotheses (*complete-on d′* constrains the output label d', not the input label d that selects the pulled sources; a settled pivot below *a* carrying a small input label could be pulled). Establishing it requires a frontier/monotonicity invariant threaded top-down through the charged inductive relation — the same band-monotonicity that underlies the correctness proof, but re-expressed on the separate cost relation. That is the design step the introduction to this theory records as not undertaken. The reduction below makes the obligation precise, single, and reusable: once pivot settledness is available as a loop invariant, the charged top-level bound becomes unconditional by mirroring the direct-insert chain.

lemma *split-below-card-le-range-tree-if-settled*:
  assumes *within*: *split-below d P beta* $\subseteq$ *bound-tree P* (*Fin b*)
    and *settled*: *split-below d P beta* $\cap$ *bound-tree P* (*Fin a*) = {}
  shows *card* (*split-below d P beta*) $\leq$ *card* (*range-tree P a* (*Fin b*))
⟨*proof*⟩

The within-bound inclusion is the standard source-subset fact: pivots whose input label is correct and below the returned bound *b* lie in the child's bounded tree, hence (by monotonicity of *bound-tree* in the source set) in *bound-tree P* (*Fin b*).

lemma *split-below-within-bound-tree-if-complete-label-below*:
  assumes *complete*: *complete-on d* (*split-below d P beta*)
    and *reaches*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *reachable s x*
    and *below*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *d x* $<$ *b*
  shows *split-below d P beta* $\subseteq$ *bound-tree P* (*Fin b*)
⟨*proof*⟩

The tight reduction. The charged per-child source-card domination follows from the standard pivot-completeness and label bound (both available in the direct-insert template) together with the single genuinely-new pivot settledness side-condition. This is the charged counterpart of the direct *source-card-le* step and feeds the existing chain summation.

lemma *split-below-card-le-range-tree-if-complete-and-settled*:
  assumes *complete*: *complete-on d* (*split-below d P beta*)
    and *reaches*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *reachable s x*
    and *below*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *d x* $<$ *b*
    and *settled*: *split-below d P beta* $\cap$ *bound-tree P* (*Fin a*) = {}
  shows *card* (*split-below d P beta*) $\leq$ *card* (*range-tree P a* (*Fin b*))
  ⟨*proof*⟩

Reducing the settledness side-condition to threaded invariants.

The lemmas above take pivot settledness *split-below d P beta* ∩ *bound-tree P* (*Fin a*) = {} as a hypothesis. The authors' note (above) flags that settledness is *not* derivable from the local State-Step hypotheses alone: it needs a frontier/monotonicity invariant threaded top-down through the recursion. The next three lemmas isolate *exactly* that obligation by reducing settledness to two recognizable, modular invariants of the input label d at a loop step:

- **label-floor** $\forall x{\in}P.\ a \le d\ x$ — the current pivot labels lie at or above the step's lower band a (the structural fact that earlier bands have already consumed the sources below a), and
- **prefix-exactness** $\forall v{\in}$*tree-set P*. *local.dist s v* $< a \longrightarrow d\ v =$ *local.dist s v* — d is exact on the settled tree-set prefix below a (the band-monotonicity invariant the descending recursion maintains). Under those two, a pulled pivot with *local.dist s x* $< a$ would have exact label $d\ x =$ *local.dist s x* and hence $d\ x < a$, contradicting the label floor; so the pull is settled. This turns the opaque settledness side-condition into the single precise invariant a top-down threading must establish, and feeds the existing card / chain machinery unchanged.

lemma *split-below-inter-bound-tree-empty-if-label-ge-exact*:
assumes *ge*: $\bigwedge x.\ x \in P \Longrightarrow a \le d\ x$
and *exact*: $\bigwedge v.\ v \in$ *tree-set P* $\Longrightarrow$ *dist s v* $< a \Longrightarrow d\ v =$ *dist s v*
shows *split-below d P beta* ∩ *bound-tree P* (*Fin a*) = {}
⟨*proof*⟩

lemma *split-below-card-le-range-tree-if-complete-label-ge-exact*:
assumes *complete*: *complete-on d* (*split-below d P beta*)
and *reaches*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow d\ x < b$
and *ge*: $\bigwedge x.\ x \in P \Longrightarrow a \le d\ x$
and *exact*: $\bigwedge v.\ v \in$ *tree-set P* $\Longrightarrow$ *dist s v* $< a \Longrightarrow d\ v =$ *dist s v*
shows *card* (*split-below d P beta*) ≤ *card* (*range-tree P a* (*Fin b*))
⟨*proof*⟩

lemma *split-below-subset-range-tree-if-complete-label-ge-exact*:
assumes *complete*: *complete-on d* (*split-below d P beta*)
and *reaches*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in$ *split-below d P beta* $\Longrightarrow d\ x < b$
and *ge*: $\bigwedge x.\ x \in P \Longrightarrow a \le d\ x$
and *exact*: $\bigwedge v.\ v \in$ *tree-set P* $\Longrightarrow$ *dist s v* $< a \Longrightarrow d\ v =$ *dist s v*
shows *split-below d P beta* ⊆ *range-tree P a* (*Fin b*)
⟨*proof*⟩

Strong base case of the charged totality induction. At level *0* the charged relation can fire only through the *Charged-Direct-Insert-Base* rule,

which is restricted to singleton source sets. For such a singleton we exhibit the complete base-case witness: a charged run exists, the returned bound does not exceed the input bound, the output set is exactly the bounded tree of the returned bound, and the returned label is complete on that output. This is the leaf in which the level induction for charged-run existence (obligation *O1* of the path-family bridge) bottoms out. The completeness certificate is read straight off *base-case-result-correct*; note we reduce that fact with *base-case-result-def* and *prod.case only* – a full simplification would orient the $U = \textit{bound-tree}\ S\ B'$ conjunct into a rewrite rule and corrupt the label witness, which itself mentions *base-case-vertices*.

lemma *charged-bmssp-zero-total*:
  assumes $S = \{x\}$
  shows $\exists\, d'\ B'\ U\ c.$
    charged-direct-insert-costed-bmssp Δ *M-of t h k cap 0 d S B d′ B′ U c* $\wedge$
    bound-le $B'\ B$ $\wedge$
    $U =$ *bound-tree* $S\ B'$ $\wedge$
    complete-on $d'\ U$
⟨*proof*⟩

Full coverage at the leaf. When the bounded tree of the input bound has at most k vertices the truncating base case keeps *all* of them, so the level-*0* charged run returns the input bound itself ($B' = B$) and its output set is the entire bounded tree *bound-tree S B*. This is the sharpened leaf used by the coverage half of the level induction: a small enough source band is completely solved in one base call without losing the residual frontier.

lemma *charged-bmssp-zero-full-cover*:
  assumes *S*: $S = \{x\}$
    and *small*: *card* (*bound-tree S B*) $\leq k$
  shows $\exists\, d'\ U\ c.$
    charged-direct-insert-costed-bmssp Δ *M-of t h k cap 0 d S B d′ B U c* $\wedge$
    $U =$ *bound-tree* $S\ B$ $\wedge$ *complete-on* $d'\ U$
⟨*proof*⟩

The output-label collapse $d' =$ *dist s*. Fixing the run's output label to the true distance *local.dist s* trivialises every completeness obligation in the charged relation: *complete-on* (*local.dist s*) *?U* gives *complete-on* (*local.dist s*) *U* for all *U*, and the next fact gives *complete-preserved d* (*local.dist s*) *U* for all *d*, *U*. Since *charged-direct-insert-top-level-cost* existentially quantifies the output label, choosing *local.dist s* is legitimate (the empty-pivot witness already does so), and it removes the band-monotonicity exactness invariant from the run-existence problem entirely, leaving a purely structural induction.

lemma *complete-preserved-dist*: *complete-preserved d* (*dist s*) *U*
  ⟨*proof*⟩

The $d' =$ *local.dist s* specialisation of the band-loop driver *charged-direct-insert-partition-loop-state-cases-exists-with-complete*: with the output label

pinned to *local.dist s* every *complete-on* and *complete-preserved* side-condition is discharged automatically, so a loop state exists as soon as the current band admits a Done, Stop, or Step case (the Step case still carrying a child run and a recursive tail loop state).

lemma *charged-loop-state-exists-dist*:
fixes *a Bmax* :: *real*
assumes *cases*:
(*keys-of D* = {} ∧ *bound-le* (*Fin a*) *B*) ∨
(*bound-le* (*Fin a*) *B* ∧ *k* ∗ *cap* ≤ *card* (*bound-tree P* (*Fin a*))) ∨
(∃ *S-pull beta D-pull B′*.
pull-separates D (*M-of l*) *Bmax S-pull beta D-pull* ∧
bound-le (*Fin beta*) *B* ∧
bmssp-pre-full d S-pull (*Fin beta*) ∧
S-pull = *split-below d P beta* ∧
(∀ *x*∈*S-pull*. *reachable s x*) ∧
card (*bound-tree P* (*Fin a*)) < *k* ∗ *cap* ∧
(∃ *d-child b U-child c-child betas bs charged-tail*
child-outputs-tail U-tail c-tail child-costs-tail.
a ≤ *b* ∧
bound-le (*Fin a*) *B′* ∧
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d*
S-pull (*Fin beta*) *d-child* (*Fin b*) *U-child c-child* ∧
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B (*dist s*)
(*batch-min-update D-pull*
(*edge-relaxation-pairs-in-bound d-child*
(*range-tree P a* (*Fin b*)) *beta B* @
edge-relaxation-pairs-between d-child
(*range-tree P a* (*Fin b*)) *b beta* @
label-pairs-between d S-pull b beta))
b betas bs B′ charged-tail child-outputs-tail U-tail c-tail
child-costs-tail))
shows ∃ *betas bs B′ charged-Us child-outputs U c child-costs*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l*
d P B (*dist s*) *D a betas bs B′ charged-Us child-outputs U c child-costs*
⟨*proof*⟩

One genuine step, then Done, under $d' = \mathit{local.dist}\ s$**.** Specialising the combined step/Done loop witness *charged-direct-insert-partition-loop-state-step-done-exists* to the true-distance output label discharges all three completeness side-conditions automatically (via *complete-on* (*local.dist s*) *?U* and *complete-preserved ?d* (*local.dist s*) *?U*) and fixes the four partition-operation costs to zero. What is left is purely structural: a pull of the current band, a recursive child run, the resulting residual partition view being empty, and the band/threshold bookkeeping. The loop returned terminates at the input bound B – its output bound is B itself – so it is exactly the loop shape consumed by the top-level existence reduction when $B = \mathit{Infinity}$.

lemma *charged-loop-single-step-then-done-exists-dist*:

fixes *a b beta Bmax* :: *real* **and** *B* :: *bound* **and** *D D-pull* :: *'a partition-view*
assumes *pull*: *pull-separates D* (*M-of l*) *Bmax S-pull beta D-pull*
and *beta-bound*: *bound-le* (*Fin beta*) *B*
and *pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
and *split*: *S-pull* = *split-below d P beta*
and *reaches*: ∀ *x*∈*S-pull*. *reachable s x*
and *a-le-b*: *a* ≤ *b*
and *a-bound*: *bound-le* (*Fin a*) *B*
and *small*: *card* (*bound-tree P* (*Fin a*)) < *k* ∗ *cap*
and *child*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-pull* (*Fin beta*)
d-child (*Fin b*) *U-child c-child*
and *tail-empty*:
keys-of (*batch-min-update D-pull*
(*edge-relaxation-pairs-in-bound d-child* (*range-tree P a* (*Fin b*)) *beta B* @
edge-relaxation-pairs-between d-child (*range-tree P a* (*Fin b*)) *b beta* @
label-pairs-between d S-pull b beta)) = {}
and *b-bound*: *bound-le* (*Fin b*) *B*
shows ∃ *betas bs charged-Us child-outputs U c child-costs*.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B*
(*dist s*) *D a betas bs B charged-Us child-outputs U c child-costs*
⟨*proof*⟩

Cost-exposing form of the single-step-then-Done loop. Identical to *charged-loop-single-step-then-done-exists-dist* but returning the concrete loop state with its cost read off as the lone child cost *c-child* (all four partition costs and the empty Done tail are zero). This is the brick the cheap charged top-level run on the path family threads level by level, so that the whole run cost telescopes to a sum of the FindPivots scans plus the single base-case scan, with no edge-range or chain terms.

lemma *charged-loop-single-step-then-done-costed-dist*:
fixes *a b beta Bmax* :: *real* **and** *B* :: *bound* **and** *D D-pull* :: *'a partition-view*
assumes *pull*: *pull-separates D* (*M-of l*) *Bmax S-pull beta D-pull*
and *beta-bound*: *bound-le* (*Fin beta*) *B*
and *pre*: *bmssp-pre-full d S-pull* (*Fin beta*)
and *split*: *S-pull* = *split-below d P beta*
and *reaches*: ∀ *x*∈*S-pull*. *reachable s x*
and *a-le-b*: *a* ≤ *b*
and *a-bound*: *bound-le* (*Fin a*) *B*
and *small*: *card* (*bound-tree P* (*Fin a*)) < *k* ∗ *cap*
and *child*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-pull* (*Fin beta*)
d-child (*Fin b*) *U-child c-child*
and *tail-empty*:
keys-of (*batch-min-update D-pull*
(*edge-relaxation-pairs-in-bound d-child* (*range-tree P a* (*Fin b*)) *beta B* @
edge-relaxation-pairs-between d-child (*range-tree P a* (*Fin b*)) *b beta* @
label-pairs-between d S-pull b beta)) = {}
and *b-bound*: *bound-le* (*Fin b*) *B*

shows *charged-direct-insert-costed-partition-loop-state* Δ *M-of t h k cap l d P B*
(dist s) D a [beta] [b] B
(range-tree P a (Fin b) # [range-tree P b B]) [U-child]
(bound-tree P (Fin a) ∪ ⋃(set (range-tree P a (Fin b) # [range-tree P b B])))
c-child [c-child]
⟨*proof*⟩

One genuine step, then an arbitrary tail, under $d' = local.dist\ s$**.** This is the multi-step companion of *charged-loop-single-step-then-done-exists-dist*: rather than forcing the residual partition to be empty (Done) it consumes a ready-made loop state for the band $[b, B')$ and stacks one more *Charged-Direct-Insert-State-Step* on top, pulling the band $[a, b)$ with a recursive child run. All three completeness obligations and the four partition costs collapse via *complete-on (local.dist s) ?U / complete-preserved ?d (local.dist s) ?U*. Iterating this brick is how the path band is covered step by step: each step fans out one level-l child of at most $k * cap$ vertices, so a whole-path cover follows by chaining the cons over the frontier.

lemma *charged-loop-step-cons-exists-dist*:
fixes *a b beta Bmax* :: *real* **and** *B B'* :: *bound* **and** *D D-pull* :: *'a partition-view*
assumes *pull*: *pull-separates D (M-of l) Bmax S-pull beta D-pull*
and *beta-bound*: *bound-le (Fin beta) B*
and *pre*: *bmssp-pre-full d S-pull (Fin beta)*
and *split*: *S-pull = split-below d P beta*
and *reaches*: ∀ *x*∈*S-pull. reachable s x*
and *a-le-b*: $a \leq b$
and *a-bound*: *bound-le (Fin a) B'*
and *small*: *card (bound-tree P (Fin a)) < k * cap*
and *child*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-pull (Fin beta) d-child (Fin b) U-child c-child*
and *D-next*:
D-next = batch-min-update D-pull
(edge-relaxation-pairs-in-bound d-child (range-tree P a (Fin b)) beta B @
edge-relaxation-pairs-between d-child (range-tree P a (Fin b)) b beta @
label-pairs-between d S-pull b beta)
and *tail*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B (dist s) D-next b betas bs B' charged-tail child-outputs-tail U-tail c-tail child-costs-tail*
shows *charged-direct-insert-costed-partition-loop-state* Δ *M-of t h k cap l d P B*
(dist s) D a (beta # betas) (b # bs) B'
(range-tree P a (Fin b) # charged-tail) (U-child # child-outputs-tail)
(bound-tree P (Fin a) ∪ ⋃(set (range-tree P a (Fin b) # charged-tail)))
(0 + 0 + 0 + 0 + c-child + c-tail) (c-child # child-costs-tail)
⟨*proof*⟩

Charged per-call progress: the success-or-threshold dichotomy. The next five facts are the charged-cost analogue of the direct-insert source-card chain. For every charged run the number of sources is bounded by the

cardinality of the run's *own* output U: either the bound was not advanced (success, $B' = B$), so the sources sit inside their own bounded tree, or the FindPivots cap is saturated (threshold, $k * cap \leq card\ U$). This is the stranding-free progress step. It transfers the direct-insert dichotomy to the permissive charged relation *without* the rigid *U-child = range-tree P a* (*Fin b*) step rule, and is therefore non-vacuous on the path family. Note, however, that the bound is against the run's *cumulative* own output, not the disjoint range slice; summing it over the loop children does not on its own telescope to the linear chain (see the loop-level companion below).

PIECE A: charged source-card chain (amortized, stranding-free via the success-or-threshold dichotomy).

theorem *charged-direct-insert-costed-bmssp-source-card-le-if-sound-label-below-output*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d' B' U c*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* (*d x*) *B'*
shows *card S* $\leq$ *card U*
⟨*proof*⟩

theorem *charged-direct-insert-costed-bmssp-Suc-success-or-threshold*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap* (*Suc l*) *d S B d' B' U c*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
shows $B' = B \vee k * cap \leq card\ U$
⟨*proof*⟩

theorem *charged-direct-insert-costed-bmssp-zero-source-card-le-from-label-below*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap 0 d S B d' B' U c*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* (*d x*) *B*
and *k-pos*: $0 < k$
shows *card S* $\leq$ *card U*
⟨*proof*⟩

theorem *charged-direct-insert-costed-bmssp-Suc-source-card-le-from-label-below*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap* (*Suc l*) *d S B d' B' U c*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*

and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable* $s\ x$
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* $(d\ x)\ B$
and *S-cap*: *card* $S \leq$ *cap*
and *k-pos*: $0 < k$
shows *card* $S \leq$ *card* U
⟨*proof*⟩

theorem *charged-direct-insert-costed-bmssp-source-card-le-from-label-below-all-levels*:
assumes *run*:
charged-direct-insert-costed-bmssp Δ *M-of* $t\ h\ k$ *cap* $l\ d\ S\ B\ d'\ B'\ U\ c$
and *sound*: *sound-label* d
and *pre*: *bmssp-pre-full* $d\ S\ B$
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable* $s\ x$
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* $(d\ x)\ B$
and *S-cap*: *card* $S \leq$ *cap*
and *k-pos*: $0 < k$
shows *card* $S \leq$ *card* U
⟨*proof*⟩

Loop-level companion: the charged partition-loop cost is bounded by the linear range-tree chain cardinality as soon as the pulled child sources land in their range slices (the transparent no-stranding/settledness invariant).

lemma *list-all2-card-mono-if-finite-right*:
assumes *list-all2* $(\subseteq)$ *Xs Ys*
and $\bigwedge Y.\ Y \in$ *set Ys* $\Longrightarrow$ *finite* Y
shows *list-all2* $(\lambda X\ Y.$ *card* $X \leq$ *card* $Y)$ *Xs Ys*
⟨*proof*⟩

lemma *finite-range-tree-child-list-elem*:
$Y \in$ *set* (*range-tree-child-list* $S\ a\ bs$) $\Longrightarrow$ *finite* Y
⟨*proof*⟩

lemma *list-all2-subset-imp-card-le-range-tree-child-list*:
assumes *list-all2* $(\subseteq)$ *Xs* (*range-tree-child-list* $S\ a\ bs$)
shows *list-all2* $(\lambda X\ R.$ *card* $X \leq$ *card* $R)$ *Xs* (*range-tree-child-list* $S\ a\ bs$)
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-source-sum-le-chain-if-settled*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of* $t\ h\ k$ *cap* $l\ d\ P\ B\ d'\ D\ a$
betas bs B' *charged-Us child-outputs* $U\ c$ *child-costs*
and *sound*: *sound-label* d
and *pre*: *bmssp-pre-full* $d\ P\ B$
and *reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable* $s\ x$
shows $\exists$ *child-sources*.
length child-sources = *length bs* $\wedge$
list-all2

(λS-child charged-child. ∃ c-child U-child B-child d-child B-child′.
charged-direct-insert-costed-bmssp Δ M-of t h k cap l d S-child B-child
d-child B-child′ U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ x∈S-child. reachable s x) ∧
card S-child ≤ M-of l ∧
(∀ x∈S-child. below-bound (d x) B-child))
child-sources (range-tree-child-list P a bs) ∧
c ≤ sum-list child-costs + sum-list (map card child-sources) +
t ∗ sum-list
(map card (range-tree-child-direct-edge-range-list P B a betas bs)) +
h ∗ sum-list
(map card (range-tree-child-edge-range-list P a betas bs)) +
h ∗ sum-list (map card child-sources) ∧
(list-all2 (⊆) child-sources (range-tree-child-list P a bs) ⟶
sum-list (map card child-sources) ≤ card (range-tree-chain P a bs B′))
⟨proof⟩

Explicit linear per-loop cost. Substituting the settled source bound into the cost decomposition turns the only non-telescoping source term (*1* + *h*) ∗ *sum-list* (*map card child-sources*) into a linear function of the produced range-tree chain. Under the transparent settledness invariant the charged partition-loop cost is therefore bounded by the recursive child costs plus (*1* + *h*) ∗ *card* (*range-tree-chain P a bs B′*) plus the edge-range terms (which telescope linearly through the shared *range-tree-child-list* machinery, exactly as in the direct-insert level bound). This is the cost half of the charged per-level closer; the only remaining obligation to discharge the charged top-level bound is the settledness hypothesis itself, i.e. the global amortized no-stranding invariant.

theorem *charged-direct-insert-costed-partition-loop-state-cost-le-linear-if-settled*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*
betas bs B′ charged-Us child-outputs U c child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *reaches*: ⋀x. x ∈ P ⟹ *reachable s x*
shows ∃ *child-sources*.
length child-sources = *length bs* ∧
list-all2
(λS-child charged-child. ∃ c-child U-child B-child d-child B-child′.
charged-direct-insert-costed-bmssp Δ M-of t h k cap l d S-child B-child
d-child B-child′ U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ x∈S-child. reachable s x) ∧
card S-child ≤ M-of l ∧
(∀ x∈S-child. below-bound (d x) B-child))
child-sources (range-tree-child-list P a bs) ∧
(list-all2 (⊆) child-sources (range-tree-child-list P a bs) ⟶

$c \leq$ *sum-list child-costs*
$+ (1 + h) *$ *card* (*range-tree-chain P a bs B′*)
$+ t *$ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*))
$+ h *$ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*)))
⟨*proof*⟩

Fully-linear per-loop local budget under settledness. The two edge-range sums telescope to *edge-count* through the shared *range-tree-child-edge-range-list* / *range-tree-child-direct-edge-range-list* disjointness lemmas (the same ones the direct-insert level bound uses), so under the settledness invariant the charged partition-loop cost reduces to the canonical local budget *sum-list child-costs* $+ (1 + h) *$ *card* (*range-tree-chain P a bs B′*) $+ (t + h) *$ *edge-count*: recursive child costs, a linear function of the produced range-tree chain, and a linear function of the edge count. This is precisely the per-level local-budget shape consumed by the direct-insert linchpin induction; the charged top-level bound now reduces to threading the single settledness hypothesis through that induction.

theorem *charged-direct-insert-costed-partition-loop-state-cost-local-budget-if-settled*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
shows ∃ *child-sources*.
length child-sources = *length bs* ∧
list-all2
(λ*S-child charged-child*. ∃ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child d-child B-child′ U-child c-child* ∧
bmssp-pre-full d S-child B-child ∧
(∀ *x*∈*S-child*. *reachable s x*) ∧
card S-child ≤ *M-of l* ∧
(∀ *x*∈*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) ∧
(*list-all2* (⊆) *child-sources* (*range-tree-child-list P a bs*) ⟶
c ≤ *sum-list child-costs*
$+ (1 + h) *$ *card* (*range-tree-chain P a bs B′*)
$+ (t + h) *$ *edge-count*)
⟨*proof*⟩

Increment monotonicity bridge. A child run's cost is bounded by its *own* range-tree increment *range-tree* (*split-below d P beta*) *a B*; because the pulled source *split-below d P beta* sits inside the parent pivot tree, that increment is contained in the parent slice *range-tree P a B*, and *level-range-cost-bound* is monotone in the set argument. Hence the intrinsic child bound

transfers to the slice bound the loop closer consumes — using only the easy containment direction, with no cut-equality obligation.

lemma *tree-set-mono*: $X \subseteq Y \Longrightarrow$ *tree-set* $X \subseteq$ *tree-set* Y
⟨*proof*⟩

lemma *range-tree-mono-tree-set*:
tree-set $S \subseteq$ *tree-set* $P \Longrightarrow$ *range-tree* $S\ a\ B \subseteq$ *range-tree* $P\ a\ B$
⟨*proof*⟩

lemma *range-tree-split-below-subset*:
range-tree (*split-below* $d\ P\ beta$) $a\ B \subseteq$ *range-tree* $P\ a\ B$
⟨*proof*⟩

lemma *level-range-cost-bound-mono*:
assumes *sub*: $X \subseteq Y$ **and** *finY*: *finite* Y
shows *level-range-cost-bound* $A\ R\ L\ X \leq$ *level-range-cost-bound* $A\ R\ L\ Y$
⟨*proof*⟩

lemma *level-range-cost-bound-range-tree-split-below-le*:
assumes *finP*: *finite* (*range-tree* $P\ a\ B$)
shows *level-range-cost-bound* $A\ R\ L$ (*range-tree* (*split-below* $d\ P\ beta$) $a\ B$)
$\leq$ *level-range-cost-bound* $A\ R\ L$ (*range-tree* $P\ a\ B$)
⟨*proof*⟩

Increment-faithful charged loop closer. The same per-loop accounting as *charged-direct-insert-costed-partition-loop-state-closes-level-bound-from-child-bound-with-invariants-and-edge-ranges*, but the produced bound is kept against the range-tree *chain range-tree-chain* $P\ a\ bs\ B'$ — the freshly-settled increment of the loop — rather than being loosened to the loop output U. Keeping the chain is what lets the bound telescope across the recursion: a child's loop charges only its own increment, and sibling increments are disjoint. The source term *Suc* $h *$ *card* (*range-tree-chain* $P\ a\ bs\ B'$) is absorbed into the vertex coefficient through *Suc* $h \leq A$.

theorem *charged-direct-insert-costed-partition-loop-state-closes-increment-bound-from-child-bound-with-invariants*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of* $t\ h\ k\ cap\ l\ d\ P\ B\ d'\ D\ a$
betas bs B' *charged-Us child-outputs* $U\ c$ *child-costs*
and *sound*: *sound-label* d
and *pre*: *bmssp-pre-full* $d\ P\ B$
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable* $s\ x$
and *P-k-cap*: $k *$ *card* $P \leq cap$
and *P-anti*: *tree-antichain* P
and *child-bound*:
$\bigwedge$*c-child charged-child U-child S-child B-child d-child B-child'*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of* $t\ h\ k\ cap\ l\ d$ *S-child B-child*
d-child B-child' U-child c-child;
bmssp-pre-full d *S-child B-child*;

```
      ⋀x. x ∈ S-child ⟹ reachable s x;
      card S-child ≤ M-of l;
      ⋀x. x ∈ S-child ⟹ below-bound (d x) B-child;
      k * card S-child ≤ cap;
      tree-antichain S-child⟧
    ⟹ c-child ≤ level-range-cost-bound A R L charged-child
 and source-factor: Suc h ≤ A
 and source-progress:
  ⋀child-sources.
   list-all2
    (λS-child charged-child. ∃ c-child U-child B-child d-child B-child'.
      charged-direct-insert-costed-bmssp Δ M-of t h k cap l d S-child B-child
        d-child B-child' U-child c-child ∧
      bmssp-pre-full d S-child B-child ∧
      (∀ x∈S-child. reachable s x) ∧
      card S-child ≤ M-of l ∧
      (∀ x∈S-child. below-bound (d x) B-child))
    child-sources (range-tree-child-list P a bs) ⟹
   sum-list (map card child-sources) ≤
    card (range-tree-chain P a bs B')
shows c ≤
  A * Suc L * card (range-tree-chain P a bs B') +
  R * card (outgoing-edges (range-tree-chain P a bs B')) +
  t * sum-list
    (map card (range-tree-child-direct-edge-range-list P B a betas bs)) +
  h * sum-list
    (map card (range-tree-child-edge-range-list P a betas bs))
⟨proof⟩
```

Charged amortized accounting (B2). The charged counterpart of the direct-insert amortized chain (*direct-insert-costed-partition-loop-state-closes-amortized-bound-from-child-costs*). The pulled child sources are absorbed into the produced range-tree chain via the per-loop *source-progress* settledness oracle, and the edge-range sums telescope through the shared disjointness lemmas exactly as in the direct-insert development. These reduce the charged top-level refined bound to the single settledness side-condition.

Increment-monotonicity bridge for the amortized budget. The amortized cost budget is monotone in its vertex-set argument (all three of *card*, *outgoing-edges*, and *outgoing-edges-range* are), so a child run's intrinsic amortized bound against its own range-tree increment *range-tree* (*split-below d P beta*) *a B′* transfers to the parent slice *range-tree P a B′* that the loop closer consumes — using only the easy containment direction *range-tree-split-below-subset*.

```
lemma bmssp-amortized-cost-bound-mono:
 assumes sub: X ⊆ Y and finY: finite Y
 shows bmssp-amortized-cost-bound A R h t q L B X
     ≤ bmssp-amortized-cost-bound A R h t q L B Y
```

⟨*proof*⟩

lemma *bmssp-amortized-cost-bound-range-tree-split-below-le*:
assumes *finP*: *finite* (*range-tree P a B′*)
shows *bmssp-amortized-cost-bound A R h t q L B*
(*range-tree* (*split-below d P beta*) *a B′*)
≤ *bmssp-amortized-cost-bound A R h t q L B* (*range-tree P a B′*)
⟨*proof*⟩

theorem *charged-cost-from-child-source-and-amortized-bounds*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*
betas bs B′ charged-Us child-outputs U c child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: ⋀*x*. *x* ∈ *P* ⟹ *reachable s x*
and *child-cost-bounds*:
list-all2 (λ*c-child UB*. *case UB of* (*U-child*, *B-child*) ⇒
c-child ≤ *bmssp-amortized-cost-bound A R h t q L B-child U-child*)
child-costs (*range-tree-child-bound-pair-list P a betas bs*)
and *source-progress*:
⋀*child-sources*.
list-all2
(λ*S-child charged-child*. ∃ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child ∧
bmssp-pre-full d S-child B-child ∧
(∀ *x*∈*S-child*. *reachable s x*) ∧
card S-child ≤ *M-of l* ∧
(∀ *x*∈*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) ⟹
sum-list (*map card child-sources*) ≤
card (*range-tree-chain P a bs B′*)
shows *c* ≤
A ∗ *L* ∗ *card* (*range-tree-chain P a bs B′*) +
(*R* + *q* ∗ *h*) ∗ *card* (*outgoing-edges* (*range-tree-chain P a bs B′*)) +
t ∗ *sum-list*
(*map card* (*range-tree-child-zero-edge-range-list P a betas bs*)) +
t ∗ *sum-list*
(*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
h ∗ *sum-list*
(*map card* (*range-tree-child-edge-range-list P a betas bs*)) +
(*Suc h*) ∗ *card* (*range-tree-chain P a bs B′*)
⟨*proof*⟩

theorem *charged-direct-insert-costed-partition-loop-state-closes-amortized-bound-from-child-costs*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*

betas bs B′ charged-Us child-outputs U c child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
and *child-cost-bounds*:
list-all2 (λ*c-child UB. case UB of* (*U-child*, *B-child*) $\Rightarrow$
c-child $\leq$ *bmssp-amortized-cost-bound A R h t q L B-child U-child*)
child-costs (*range-tree-child-bound-pair-list P a betas bs*)
and *source-factor*: *Suc h* $\leq$ *A*
and *source-progress*:
$\bigwedge$*child-sources*.
list-all2
(λ*S-child charged-child*. $\exists$ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child $\wedge$
bmssp-pre-full d S-child B-child $\wedge$
($\forall$ *x*$\in$*S-child. reachable s x*) $\wedge$
card S-child $\leq$ *M-of l* $\wedge$
($\forall$ *x*$\in$*S-child. below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) $\Longrightarrow$
sum-list (*map card child-sources*) $\leq$
card (*range-tree-chain P a bs B′*)
shows *c* $\leq$ *bmssp-amortized-cost-bound A R h t* (*Suc q*) (*Suc L*) *B U*
⟨*proof*⟩

theorem *charged-direct-insert-costed-nonbase-step-closes-amortized-bound-from-child-costs*:
assumes *loop*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a*
betas bs B′ charged-Us child-outputs U-loop c-loop child-costs
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
and *child-cost-bounds*:
list-all2 (λ*c-child UB. case UB of* (*U-child*, *B-child*) $\Rightarrow$
c-child $\leq$ *bmssp-amortized-cost-bound A R h t q L B-child U-child*)
child-costs (*range-tree-child-bound-pair-list P a betas bs*)
and *source-factor*: *Suc h* $\leq$ *A*
and *source-progress*:
$\bigwedge$*child-sources*.
list-all2
(λ*S-child charged-child*. $\exists$ *c-child U-child B-child d-child B-child′*.
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S-child B-child*
d-child B-child′ U-child c-child $\wedge$
bmssp-pre-full d S-child B-child $\wedge$
($\forall$ *x*$\in$*S-child. reachable s x*) $\wedge$
card S-child $\leq$ *M-of l* $\wedge$
($\forall$ *x*$\in$*S-child. below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) $\Longrightarrow$

sum-list (*map card child-sources*) ≤
card (*range-tree-chain P a bs B′*)
and *U-def*: *U* = *U-loop* ∪ *W*
and *finite-U*: *finite U*
and *scan-insert*: *c-scan-insert* ≤ *A* ∗ *card U*
and *c-def*: *c* = *c-scan-insert* + *c-loop*
shows *c* ≤
bmssp-amortized-cost-bound A R h t (*Suc q*) (*Suc* (*Suc L*)) *B U*
⟨*proof*⟩

Charged amortized child-cost propagator. The amortized counterpart of *charged-direct-insert-costed-partition-loop-state-child-cost-bounds-with-invariants*: it threads an amortized per-child budget (stated against the disjoint range-tree slice *charged-child*) across the partition loop, producing the list the amortized loop closer consumes. In a recursion-level induction the per-child budget is discharged by the inductive hypothesis against the child increment and the increment->slice bridge *bmssp-amortized-cost-bound-range-tree-split-below-le*.

theorem *charged-direct-insert-costed-partition-loop-state-child-amortized-cost-bounds-with-invariants*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs* ⟹
P ⊆ *V* ⟹
k ∗ *card P* ≤ *cap* ⟹
tree-antichain P ⟹
(⋀*c-child charged-child U-child S-child B-child d-child B-child′*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child d-child B-child′ U-child c-child*;
bmssp-pre-full d S-child B-child;
⋀*x*. *x* ∈ *S-child* ⟹ *reachable s x*;
card S-child ≤ *M-of l*;
⋀*x*. *x* ∈ *S-child* ⟹ *below-bound* (*d x*) *B-child*;
k ∗ *card S-child* ≤ *cap*;
tree-antichain S-child⟧
⟹ *c-child* ≤ *bmssp-amortized-cost-bound A R h t q L B-child charged-child*)
⟹
list-all2 (λ*c-child UB*. *case UB of* (*U-child*, *B-child*) ⇒
c-child ≤ *bmssp-amortized-cost-bound A R h t q L B-child U-child*)
child-costs (*range-tree-child-bound-pair-list P a betas bs*)
and *charged-direct-insert-costed-bmssp-child-amortized-cost-bounds-with-invariants-trivial*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d′ B′ U c* ⟹ *True*
⟨*proof*⟩

end

end
theory *BMSSP-NonVacuous-Family*
imports *BMSSP-Path-Family-Extended BMSSP-Cost-Totality*

begin

10 An unconditional, non-vacuous size-parametric BMSSP running-time bound

The conditional charged headline *bmssp-path-family-charged-direct-insert-runtime-bigo-size-if-cost-bounded* runs the path P_k at the *same* index k that also fixes the internal logarithmic schedule. On that diagonal the top-level bucket cap $cap\ k = p\ k \cdot 2^{p\ k \cdot q\ k}$ with $p\ k =$ *sssp-log-one-third-param* k and $q\ k =$ *sssp-log-two-thirds-param* k is sublinear in k, so it eventually fails to cover the $k + 1$ path vertices and no charged top-level run can exist — the diagonal family is vacuous.

This theory *decouples* the graph size from the schedule index: it keeps the genuine growing graph P_n (with $card\ (path\text{-}k\text{-}V\ n) = Suc\ n \to \infty$) but runs it at the *inflated* schedule index $n \cdot n$. Because $cap\ (n \cdot n) \geq (n^2 + 2)^{ln\ 2} \geq n^{2\ ln\ 2}$ and $2\ ln\ 2 > 1$, the cap now covers the path for all but finitely many n, while $ln\ (n \cdot n) = 2\ ln\ n$ keeps the schedule polylog in $\Theta(ln\ n)$. This is milestone **B0**: the arithmetic that de-risks the whole construction.

10.1 Elementary analytic facts

A power with exponent > 1 eventually dominates the successor function.

lemma *real-Suc-le-powr-eventually*:
fixes c :: *real*
assumes *c1*: $1 < c$
shows *eventually* (λn::*nat*. *real* (*Suc n*) $\leq$ (*real n*) *powr c*) *at-top*
⟨*proof*⟩

The slope $2\ ln\ 2$ obtained from squaring the schedule exceeds 1.

lemma *one-less-two-ln-two*: $1 < 2 * ln\ (2$::*real*$)$
⟨*proof*⟩

10.2 A reusable "the source is the sole pivot" brick

The dual of *unique-shortest-digraph.find-pivots-pivots-capped-singleton-empty-if-params-exceed-vertex-count*: when the capped scan does *not* overflow and the source's own tree slice already holds at least k seen vertices, the singleton FindPivots call returns exactly the source. This is the $P = \{s\}$ half of the source-pivot witness, and it is precisely the regime of the decoupled path family: the cap covers the path, yet the source subtree is large.

context *unique-shortest-digraph*
begin

lemma *find-pivots-pivots-capped-singleton-source-if-large-tree*:

assumes *no-overflow*: *card* (*find-pivots-seen-capped k cap d* $\{s\}$ *B*) $\leq$ *cap*
and *large*: $k \leq$ *card* (*tree-of s* $\cap$ *find-pivots-seen-capped k cap d* $\{s\}$ *B*)
shows *find-pivots-pivots-capped k cap d* $\{s\}$ *B* $= \{s\}$
⟨*proof*⟩

The source's shortest-path tree is exactly its reachable set: the source is the head of every shortest walk, hence lies on every shortest path. Combined with the FindPivots brick this turns the $P = \{s\}$ obligation on the path family into a pure cardinality count of the capped scan.

lemma *tree-path-source-iff-reachable*:
tree-path s v $\longleftrightarrow$ *reachable s v*
⟨*proof*⟩

lemma *tree-of-source-eq*:
tree-of s $= \{v \in V.$ *reachable s v*$\}$
⟨*proof*⟩

Generic per-round bound on the capped FindPivots scan cost. Each round of *fp-iter-capped-scan-cost* contributes *card* (*outgoing-edges F*) for the current frontier *F*, and every frontier's outgoing-edge set is a subset of the whole edge set, so each round costs at most *edge-count*. Hence the *n*-round scan costs at most $n *$ *edge-count*, uniformly in the schedule arguments. This is the only handle we need on the FindPivots cost: the cheap charged top-level run on the path family pays exactly one such scan per recursion level (all four partition costs being zero), so its whole cost is a sum of these scans plus a single base-case scan.

lemma *fp-iter-capped-scan-cost-le-rounds*:
fp-iter-capped-scan-cost n cap d F W B $\leq n *$ *edge-count*
⟨*proof*⟩

end

10.3 The genuine shortest-path distance on the path family

The unit-weight path P_k has the identity distance *dist 0 v* $= v$: the canonical walk $0 \to 1 \to \ldots \to v$ is the unique simple walk to v (*path-k-simple-walk-unique*) and carries v unit edges. This is the keystone distance fact underlying every later path-tree, bound-tree and FindPivots characterisation on the decoupled family.

10.4 A schedule that dominates the path size (non-vacuous regime)

For the genuinely non-vacuous result we run the growing path P_k at a schedule $\sigma\ k$ chosen so that the FindPivots round count $p =$ *sssp-log-one-third-param* $(\sigma\ k)$ strictly exceeds the *Suc k* path vertices. Such a schedule

exists because *sssp-log-one-third-param* diverges (*sssp-log-one-third-param-at-top*). In this regime the top pivots are empty and a charged top-level run exists unconditionally, while $card\ (path\text{-}k\text{-}V\ k) = Suc\ k \to \infty$.

definition *path-sched* :: *nat* ⇒ *nat* **where**
  path-sched k = (*LEAST N*. *Suc k* < *sssp-log-one-third-param N*)

lemma *path-sched-param-gt*:
  Suc k < *sssp-log-one-third-param* (*path-sched k*)
⟨*proof*⟩

context fixes *k* :: *nat*
begin

interpretation *pk*: *finite-weighted-digraph*
  path-k-V k path-k-E k path-k-weight k 0
  ⟨*proof*⟩

lemma *path-k-walk-weight-upt*:
  a + *j* ≤ *k* ⟹ *pk.walk-weight* [*a*..<*Suc* (*a* + *j*)] = *real j*
⟨*proof*⟩

lemma *path-k-dist-eq*:
  assumes *v* ≤ *k*
  shows *pk.dist 0 v* = *real v*
⟨*proof*⟩

interpretation *bpi*: *bounded-reduced-positive-instance*
  path-k-V k path-k-E k path-k-weight k 0 1
  ⟨*proof*⟩

The source's shortest-path tree is the whole path: every path vertex is reachable from *0*, and *0* heads every shortest walk. Hence the source-tree slice in the FindPivots count is the entire vertex set.

lemma *path-k-tree-of-source*:
  bpi.tree-of 0 = *path-k-V k*
⟨*proof*⟩

The capped FindPivots scan stays inside the graph, so its seen set never exceeds the *Suc k* path vertices. This discharges the no-overflow half of the singleton source-pivot brick whenever the cap covers the path, i.e. eventually at the inflated schedule (see the cap-domination theorem below).

lemma *path-k-find-pivots-seen-card-le*:
  card (*bpi.find-pivots-seen-capped K cap d* {*0*} *Infinity*) ≤ *Suc k*
⟨*proof*⟩

Empty top pivots when the schedule dominates the path. If the FindPivots round count K already strictly exceeds the *Suc k* path vertices (and the cap covers the path, so the capped scan does not overflow), then no

vertex roots a K-large source-tree slice — the whole source tree has only *Suc* $k < K$ vertices — so the capped FindPivots call from $\{0\}$ returns the *empty* pivot set. This is the genuinely *non-vacuous* regime: with empty pivots the top-level partition loop is immediately Done, so a charged top-level run exists unconditionally, with no recourse to the (walled) multi-step band walk.

lemma *path-k-top-pivots-empty*:
assumes *Kk*: *Suc k* $<$ *K* **and** *cap*: *Suc k* $\leq$ *cap*
shows *bpi.find-pivots-pivots-capped K cap d* $\{0\}$ *Infinity* $= \{\}$
⟨*proof*⟩

Non-vacuous run existence (empty-pivots regime). Whenever the schedule parameter $p =$ *sssp-log-one-third-param N* exceeds the *Suc k* path vertices, the top pivots are empty (the cap $p \cdot 2^{p \cdot q} \geq p$ covers the path), so a charged top-level cost exists by the immediate-Done route *bpi.charged-direct-insert-top-level-cost-exists-if-top-pivots-empty*. This discharges run-existence unconditionally on the path in the schedule-dominates-graph regime, the regime where the BMSSP top-level call terminates in a single sweep.

lemma *path-k-charged-cost-exists-if-param-large*:
assumes *Kk*: *Suc k* $<$ *sssp-log-one-third-param N*
shows $\exists$ *c*. *bpi.charged-direct-insert-top-level-cost 1 N c*
⟨*proof*⟩

Unconditional run existence at the dominating schedule. Specialising the empty-pivots route at $N =$ *path-sched k* discharges run-existence on P_k with *no* side condition: at this schedule the FindPivots round count exceeds the path, so a charged top-level cost always exists.

lemma *path-k-charged-cost-exists-at-path-sched*:
$\exists$ *c*. *bpi.charged-direct-insert-top-level-cost 1* (*path-sched k*) *c*
⟨*proof*⟩

Unit out-degree and a linear FindPivots scan bound. Each path vertex has at most one out-edge, so a singleton frontier stays a singleton along the capped FindPivots iteration; hence the scan cost is bounded by the round count times the frontier size — here *linearly* in the frontier, with *no* cap factor. This is the per-call ingredient for the genuine $O(|V|)$ cost of the empty-pivots run (the general library bound carries a spurious *cap* factor).

lemma *path-k-outdegree-le*: *bpi.outdegree-le 1*
⟨*proof*⟩

lemma *path-k-fp-iter-capped-scan-cost-le-card-frontier*:
assumes *F* $\subseteq$ *path-k-V k*
shows *bpi.fp-iter-capped-scan-cost n cap d F W B* $\leq n *$ *card F*
⟨*proof*⟩

Every prefix vertex $v \leq K \leq k$ is a short-tight witness for the K-round FindPivots scan from the source: the canonical walk $0 \to 1 \to \ldots \to v$ is the (unique) shortest walk to v (*path-k-canonical-walk*, *path-k-dist-eq*), hence

successively tight, starts at the complete source, stays below the *Infinity* bound and uses exactly $v \leq K$ edges.

lemma *path-k-short-tight-witness-prefix*:
assumes *vK*: $v \leq K$ **and** *Kk*: $K \leq k$
shows *bpi.short-tight-witness K pk.finite-initial-label* $\{0\}$ *Infinity v*
⟨*proof*⟩

The seen *lower* bound (deliverable (G2) for the $P = \{s\}$ obligation): whenever the round count K is at most the path length and the cap covers the path, the K-round capped scan from $\{0\}$ reaches at least K vertices. The prefix $\{0..K\}$ are short-tight witnesses, hence sit inside the completed (uncapped) seen set (*bpi.find-pivots-completes-short-vertices*); the no-overflow regime collapses the capped scan to the uncapped one (*bpi.fp-iter-capped-eq-fp-iter-if-final-within-cap*).

lemma *path-k-seen-card-ge*:
assumes *Kk*: $K \leq k$ **and** *cover*: *Suc k* $\leq$ *cap*
shows $K \leq$ *card* (*bpi.find-pivots-seen-capped K cap pk.finite-initial-label* $\{0\}$ *Infinity*)
⟨*proof*⟩

A bound-generic, label-generic version of the short-tight prefix witness: any source-complete finite label ($d\ 0 = 0$) makes every prefix vertex $v \leq K \leq k$ a short-tight witness for the K-round capped scan, at any bound covering the path. This frees the seen lower bound from the specific initial label, which is what the level-l child runs need (their FindPivots input label is the previous level's capped label, not *finite-initial-label*).

lemma *path-k-short-tight-witness-prefix-fin*:
assumes *vK*: $v \leq K$ **and** *Kk*: $K \leq k$ **and** *d0*: $d\ 0 = 0$
shows *bpi.short-tight-witness K d* $\{0\}$ (*Fin* (*real* (*Suc k*))) *v*
⟨*proof*⟩

The bound-/label-generic seen lower bound: at a bound covering the path, the capped K-round scan from $\{0\}$ reaches at least K vertices for any source-complete sound label, provided the cap covers the path.

lemma *path-k-seen-card-ge-fin*:
assumes *Kk*: $K \leq k$ **and** *cover*: *Suc k* $\leq$ *cap* **and** *d0*: $d\ 0 = 0$
and *sound*: *pk.sound-label d*
shows $K \leq$ *card* (*bpi.find-pivots-seen-capped K cap d* $\{0\}$ (*Fin* (*real* (*Suc k*))))
⟨*proof*⟩

lemma *path-k-source-is-sole-pivot-fin*:
assumes *Kk*: $K \leq k$ **and** *cover*: *Suc k* $\leq$ *cap* **and** *d0*: $d\ 0 = 0$
and *sound*: *pk.sound-label d*
shows *bpi.find-pivots-pivots-capped K cap d* $\{0\}$ (*Fin* (*real* (*Suc k*))) = $\{0\}$
⟨*proof*⟩

The source-is-sole-pivot gateway (deliverable (G3), conjunct $P = \{s\}$): under the same two hypotheses, the singleton FindPivots call on the path

returns exactly the source. This combines the seen lower bound above (the K qualifying vertices) with the no-overflow upper bound (*path-k-find-pivots-seen-card-le*) through the reusable brick *bpi.find-pivots-pivots-capped-singleton-source-if-large-tree*; the source tree is the whole path (*path-k-tree-of-source*), so the tree-slice intersection is the entire seen set.

lemma *path-k-source-is-sole-pivot*:
  assumes *Kk*: $K \leq k$ **and** *cover*: $Suc\ k \leq cap$
  shows *bpi.find-pivots-pivots-capped K cap pk.finite-initial-label* $\{0\}$ *Infinity* = $\{0\}$
⟨*proof*⟩

The remaining structural conjuncts of *strict-tie-breaking-digraph.charged-direct-insert-source-pivot-finishes* on the path family, discharged once and for all. Together with the source-is-sole-pivot gateway above they reduce the predicate to its single charged child-run obligation (conjunct *6*): a full-cover level-$p-1$ run of the path inside the bound $Fin\ (real\ (Suc\ k))$.

Conjunct *2*: the source's FindPivots label never rises above *0*.

lemma *path-k-fp-label-source-le-zero*:
  bpi.find-pivots-label-capped K cap pk.finite-initial-label $\{0\}$ *Infinity* $0 \leq 0$
⟨*proof*⟩

Conjunct *5*: the zero-bounded source tree is empty.

lemma *path-k-bound-tree-source-Fin0-empty*:
  pk.bound-tree $\{0\}$ $(Fin\ 0) = \{\}$
⟨*proof*⟩

Conjunct *4*: the FindPivots source precondition holds at any finite bound.

lemma *path-k-fp-source-pre-full*:
  pk.bmssp-pre-full
    (*bpi.find-pivots-label-capped K cap pk.finite-initial-label* $\{0\}$ *Infinity*)
    $\{0\}$ $(Fin\ B)$
⟨*proof*⟩

Conjunct *7*: with the bound $Fin\ (real\ (Suc\ k))$ chosen to cover the whole path, the source band is the entire path, so its outgoing boundary edges all relax to a value $\leq k < Suc\ k$ and the direct-edge batch is empty.

lemma *path-k-direct-batch-empty*:
  bpi.edge-relaxation-pairs-in-bound $(pk.dist\ 0)$
    (*bpi.range-tree* $\{0\}$ *0* $(Fin\ (real\ (Suc\ k))))$ $(real\ (Suc\ k))$ *Infinity* = []
⟨*proof*⟩

Milestone B1 (deliverable (G3)), the decoupled charged reduction. Putting conjuncts *1–5* and *7* together, the explicit-assumption top-level builder *bpi.charged-direct-insert-top-level-cost-exists-if-source-pivot-finishes* turns the existence of the single full-cover level-$p-1$ child run (conjunct *6*) into the existence of a charged top-level cost. This is the schedule-aware reduction of (G3) to the lone charged child-run obligation: once that

run is built for the inflated schedule N, the charged top-level cost on P_k exists unconditionally.

lemma *path-k-charged-cost-exists-if-child*:
assumes *pk-le*: *sssp-log-one-third-param N* $\leq$ *k*
and *cover*: *Suc k* $\leq$ *bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)
and *child*:
∃ *U-child c-child*.
bpi.charged-direct-insert-costed-bmssp (*Suc 0*) (*bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*)) (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*) (*bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)) (*sssp-log-one-third-param N* − *1*) (*bpi.find-pivots-label-capped* (*sssp-log-one-third-param N*) (*bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)) *pk.finite-initial-label* {*0*} *Infinity*) {*0*} (*Fin* (*real* (*Suc k*))) (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U-child c-child*
shows ∃ *c*. *bpi.charged-direct-insert-top-level-cost 1 N c*
⟨*proof*⟩

Cover lift (*l* → *Suc l*). If a full-cover charged run of the whole path exists at level l for the FindPivots label of a source-complete sound input label d, then one exists at level *Suc l* for d: the FindPivots call on the path returns just the source (cap covers the path), one pull separates the source from the band, the level-l cover discharges the band, and the residual loop terminates immediately. Stacking this from the level-1 base reaches the level $p-1$ required by conjunct 6.

lemma *path-k-cover-lift*:
fixes *p q cap l* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *pk-le*: *p* $\leq$ *k* **and** *cover*: *Suc k* $\leq$ *cap*
and *ppos*: *0* < *p*
and *sound*: *pk.sound-label d* **and** *d0*: *d 0* = *0*
and *pre0*: *pk.bmssp-pre-full d* {*0*} *Infinity*
and *child*: ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*) (*bmssp-level-cap p q*) *q p p cap l* (*bpi.find-pivots-label-capped p cap d* {*0*} (*Fin* (*real* (*Suc k*)))) {*0*} (*Fin* (*real* (*Suc k*))) (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
shows ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*) (*bmssp-level-cap p q*) *q p p cap* (*Suc l*) *d* {*0*} (*Fin* (*real* (*Suc k*))) (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
⟨*proof*⟩

Level chain. Stacking the cover lift from a level-1 base, a whole-path cover exists at every level $l \geq 1$ for any source-complete sound input

label. Specialising at $l = p - 1$ supplies conjunct 6 of *strict-tie-breaking-digraph.charged-direct-insert-source-pivot-finishes* unconditionally once the level-1 base is available.

lemma *path-k-cover-ge1*:
fixes *p q cap* :: *nat*
assumes *pk-le*: $p \le k$ **and** *cover*: *Suc k* $\le$ *cap* **and** *ppos*: $0 < p$
and *base1*: $\bigwedge d.$ *pk.sound-label d* $\Longrightarrow$ *d 0 = 0* $\Longrightarrow$ *pk.bmssp-pre-full d* {*0*} *Infinity*
$\Longrightarrow \exists U\ c.$ *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap* (*Suc 0*) *d* {*0*} (*Fin* (*real* (*Suc k*)))
(*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
shows $\bigwedge d.$ *pk.sound-label d* $\Longrightarrow$ *d 0 = 0* $\Longrightarrow$ *pk.bmssp-pre-full d* {*0*} *Infinity* $\Longrightarrow$
$\exists U\ c.$ *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*) (*bmssp-level-cap p q*) *q p p*
cap (*Suc l*) *d* {*0*} (*Fin* (*real* (*Suc k*))) (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
⟨*proof*⟩

The path band tree. Every vertex $a \le v \le k$ lies on the unique shortest walk from the source, so the source band starting at a below the finite bound *Fin r* is exactly the prefix $\{a..k\}$ truncated by r.

lemma *path-k-through-iff*:
assumes $v \le k$
shows *pk.through* {*a*} *v* $\longleftrightarrow$ $a \le v$
⟨*proof*⟩

lemma *path-k-bound-tree-band*:
pk.bound-tree {*a*} (*Fin r*) = {*v*. $a \le v \wedge v \le k \wedge$ *real v* $< r$}
⟨*proof*⟩

The base-case order on a path source band $\{a\}$ below *Suc k* is just the increasing run $[a, \ldots, k]$: distances equal indices, so the unique distinct distance-sorted listing of $\{a..k\}$ is $[a..<$*Suc k*$]$.

lemma *path-k-base-order*:
shows *pk.base-case-order a* (*Fin* (*real* (*Suc k*))) = [*a*..<*Suc k*]
⟨*proof*⟩

One base step. A level-0 base case on a singleton path band $\{a\}$ truncated just above a fully settles a: the bounded tree is $\{a\}$, of card *1* $\le p$, so the base call keeps it whole. Witnesses are given explicitly (the truncating label, returned bound, vertex set) to avoid any search in the large path simpset.

lemma *path-k-base-step*:
fixes *a p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *ak*: $a \le k$ **and** *ppos*: $0 < p$
shows $\exists d'\ c.$ *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap 0 d* {*a*} (*Fin* (*real* (*Suc a*)))
d' (*Fin* (*real* (*Suc a*))) {*a*} *c*

⟨*proof*⟩

Top base step. A level-0 base case on the whole source band $\{0\}$ below *Fin* (*real* (*Suc k*)) settles exactly the prefix $\{0..<p\}$ and advances the returned bound to *Fin* (*real p*): the path order is $[0..<Suc\ k]$, the capped scan stops at the p-th vertex p, and only strictly-smaller vertices survive. This is the per-block brick that the multi-step loop chains.

Singleton frontier pull. A one-element bucket $\{a\}$ with label view d is pulled whole (fits under the fanout $0 < p$), leaving the empty view; the separator threshold is the loop bound *Bmax*. Each multi-step block iteration uses this pull.

lemma *path-k-pull-singleton*:
fixes *a p* :: *nat* **and** *Bmax* :: *real* **and** *d* :: *nat* ⇒ *real*
assumes *ppos*: *0* < *p*
shows *pull-separates* (*bpi.label-partition-view d* {*a*}) *p Bmax* {*a*} *Bmax*
(*bpi.label-partition-view d* {} :: *nat partition-view*)
⟨*proof*⟩

Block range tree. The slice of the source tree at heights $[a,\ a{+}p)$ is the block $\{a..<a+p\}$; this is the charged child the loop discharges at each step.

lemma *path-k-range-block*:
assumes *apk*: *a* + *p* ≤ *k*
shows *bpi.range-tree* {*a*} (*real a*) (*Fin* (*real* (*a* + *p*))) = {*a*..<*a*+*p*}
⟨*proof*⟩

Source-pivot range slice. Read off the source tree $\{0\}$ the path range $[a,\ b)$ is exactly the integer block $\{a..<b\}$ whenever $b \le Suc\ k$. Unlike *path-k-range-block* (which slices from the band start $\{a\}$), the pivot here is the *global* source $\{0\}$ that the top-level partition loop carries, which is what the loop's relaxation batches *bpi.range-tree* $\{0\}$ *a B* use.

lemma *path-k-range-zero*:
assumes *bk*: *b* ≤ *Suc k*
shows *bpi.range-tree* {*0*} (*real a*) (*Fin* (*real b*)) = {*a*..<*b*}
⟨*proof*⟩

The source-pivot pull on the path. Because the source carries the genuine zero label, the loop's threshold pull *bpi.split-below d* $\{0\}$ *beta* of the sole pivot set $\{0\}$ is exactly $\{0\}$ below a positive threshold and empty otherwise. This pins down the only pivot the top-level loop can ever pull.

lemma *path-k-split-below-zero*:
assumes *d0*: *d* (*0*::*nat*) = *0*
shows *bpi.split-below d* {*0*} *beta* = (*if 0* < *beta then* {*0*} *else* {})
⟨*proof*⟩

Pivot settledness at the source band (the B2 cost brick). At the loop's initial band $a = 0$ the already-settled prefix *pk.bound-tree* $\{0\}$ (*Fin 0*) is empty, so the source pull is trivially settled, and lands inside its own

range slice $\{0..<b\}$ whenever the child advanced the bound ($0 < b \leq Suc\ k$). This is the per-step ingredient of the no-stranding invariant of *bpi.charged-direct-insert-costed-partition-loop-state-cost-local-budget-if-settled*, specialised to the unit-out-degree path where the loop runs at band *0*.

lemma *path-k-pull-settled-at-zero*:
bpi.split-below d {0} beta ∩ pk.bound-tree {0} (Fin 0) = {}
⟨*proof*⟩

lemma *path-k-pull-in-first-slice*:
assumes *bpos*: *0 < b* **and** *bk*: *b ≤ Suc k*
shows *bpi.split-below d {0} beta ⊆ bpi.range-tree {0} (real 0) (Fin (real b))*
⟨*proof*⟩

The relaxation never re-creates the source. Every relaxation target on the path is the head *Suc u* of a forward edge $(u, Suc\ u)$, hence strictly positive. So a loop step's batch keys avoid *0*: once the sole pivot *0* is pulled out of the partition it is never pushed back. This is the frontier-monotonicity fact behind pivot settledness on the unit-out-degree path — the loop can advance the frontier but can never return it to the source band.

lemma *path-k-relax-in-bound-keys-pos*:
0 ∉ fst ‘ set (bpi.edge-relaxation-pairs-in-bound d U L B′)
⟨*proof*⟩

The same frontier-positivity for the *between* relaxation batch: every pushed key is an edge target *Suc u > 0*.

lemma *path-k-relax-between-keys-pos*:
0 ∉ fst ‘ set (bpi.edge-relaxation-pairs-between d U L H)
⟨*proof*⟩

Block frontier. Relaxing the block $\{a..<a+p\}$ across its single boundary edge $(a+p-1,\ a+p)$ pushes exactly the next frontier vertex $a+p$; the lower and source batches are empty. Hence the partition advances $\{a\} \mapsto \{a+p\}$.

lemma *path-k-block-next-keys*:
assumes *apk*: *a + p ≤ k* **and** *ppos*: *0 < p*
shows *keys-of (batch-min-update (bpi.label-partition-view (pk.dist 0) {} :: nat partition-view)*
(bpi.edge-relaxation-pairs-in-bound (pk.dist 0)
(bpi.range-tree {a} (real a) (Fin (real (a+p)))) (real (a+p)) (Fin (real (Suc k))) @
bpi.edge-relaxation-pairs-between (pk.dist 0)
(bpi.range-tree {a} (real a) (Fin (real (a+p)))) (real (a+p)) (real (a+p)) @
bpi.label-pairs-between (pk.dist 0) {a} (real (a+p)) (real (a+p)))) = {a+p}
⟨*proof*⟩

lemma *path-k-base-top*:
fixes *p q cap* :: *nat* **and** *d* :: *nat ⇒ real*
assumes *pk-le*: *p ≤ k* **and** *ppos*: *0 < p*

shows *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap 0 d* {*0*} (*Fin* (*real* (*Suc k*)))
(λ*v*. *if v* ∈ *pk.base-case-vertices p 0* (*Fin* (*real* (*Suc k*))) *then pk.dist 0 v else d v*)
(*Fin* (*real p*)) {*0*..<*p*}
(*bpi.base-case-scan-cost* (*Suc 0*) *p 0* (*Fin* (*real* (*Suc k*))))
⟨*proof*⟩

Generic block base step. From any band start a with $a + p \leq k$, the level-0 base call on $\{a\}$ below the full bound settles exactly the block $\{a..<a+p\}$ and advances the returned bound to *Fin* (*real* $(a + p)$). Iterating this brick over $a = 0,\ p,\ 2p,\ \ldots$ is the multi-step frontier walk.

lemma *path-k-base-block*:
fixes *a p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *apk*: *a* + *p* ≤ *k* **and** *ppos*: *0* < *p*
shows *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap 0 d* {*a*} (*Fin* (*real* (*Suc k*)))
(λ*v*. *if v* ∈ *pk.base-case-vertices p a* (*Fin* (*real* (*Suc k*))) *then pk.dist 0 v else d v*)
(*Fin* (*real* (*a* + *p*))) {*a*..<*a*+*p*}
(*bpi.base-case-scan-cost* (*Suc 0*) *p a* (*Fin* (*real* (*Suc k*))))
⟨*proof*⟩

Terminal block. When the residual band $\{a..k\}$ fits within p, the base call below the full bound covers it whole and keeps the bound, settling the loop.

lemma *path-k-base-last*:
fixes *a p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *ak*: *a* ≤ *k* **and** *ple*: *Suc k* − *a* ≤ *p*
shows *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap 0 d* {*a*} (*Fin* (*real* (*Suc k*)))
(λ*v*. *if v* ∈ *pk.base-case-vertices p a* (*Fin* (*real* (*Suc k*))) *then pk.dist 0 v else d v*)
(*Fin* (*real* (*Suc k*))) {*a*..*k*}
(*bpi.base-case-scan-cost* (*Suc 0*) *p a* (*Fin* (*real* (*Suc k*))))
⟨*proof*⟩

Level-0 whole-path cover (fanout regime). When the FindPivots round count p already dominates the *Suc* k path vertices, a single level-0 base call on $\{0\}$ below *Fin* (*real* (*Suc* k)) settles the entire band $\{0..k\}$: the bounded tree has card *Suc* $k \leq p$, so the capped scan keeps it whole and the bound is preserved. This is the genuine level-0 child the cover lift needs.

lemma *path-k-zero-cover*:
fixes *p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *small*: *Suc k* ≤ *p*
shows ∃ *d' U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap 0 d* {*0*} (*Fin* (*real* (*Suc k*)))
d' (*Fin* (*real* (*Suc k*))) *U c*

⟨*proof*⟩

Level-1 base (fanout regime). Lifting the level-0 whole cover by one pull gives the level-1 whole-path cover required as the base of *path-k-cover-ge1*, whenever p dominates the path.

lemma *path-k-base1*:
  fixes *p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
  assumes *pk-le*: $p \le k$ **and** *cover*: *Suc k* ≤ *cap* **and** *ppos*: $0 < p$
    and *small*: *Suc k* ≤ *p*
    and *sound*: *pk.sound-label d* **and** *d0*: *d 0* = *0*
    and *pre0*: *pk.bmssp-pre-full d* {*0*} *Infinity*
  shows ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
      (*bmssp-level-cap p q*) *q p p cap* (*Suc 0*) *d* {*0*} (*Fin* (*real* (*Suc k*)))
      (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
⟨*proof*⟩

Level-1 base in the genuine $p \le k$ regime (empty-band / free-prefix construction). Unlike *path-k-base1*, which needs the round count to dominate the path (*Suc k* ≤ *p*), this builds a level-1 whole-path cover for *any* $p \le k$. The permissive charged loop is started at the band $a = p$ (claiming the already-settled prefix {*0*..<*p*} via the true-distance output label), pulls the sole pivot {*0*}, runs a single level-0 base child that returns *Fin* (*real p*), and finishes over the *empty* band {*p*..<*p*}: with $a = b = p$ the range tree is empty, so the relaxation pushes nothing and the residual partition is empty, giving an immediate *Done* back at the input bound *Fin* (*real* (*Suc k*)). This is the level-1 base the cover chain *path-k-cover-ge1* consumes.

lemma *path-k-base1-general*:
  fixes *p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
  assumes *pk-le*: $p \le k$ **and** *cover*: *Suc k* ≤ *cap* **and** *ppos*: $0 < p$
    and *capgt*: *1* < *cap*
    and *sound*: *pk.sound-label d* **and** *d0*: *d 0* = *0*
    and *pre0*: *pk.bmssp-pre-full d* {*0*} *Infinity*
  shows ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
      (*bmssp-level-cap p q*) *q p p cap* (*Suc 0*) *d* {*0*} (*Fin* (*real* (*Suc k*)))
      (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
⟨*proof*⟩

Milestone B1, discharged: a charged top-level cost exists on P_k at the inflated schedule. Stacking the empty-band level-1 base *path-k-base1-general* through the cover chain *path-k-cover-ge1* to level $p - 1$ supplies the single charged child run (conjunct 6) that the source-pivot reduction *path-k-charged-cost-exists-if-child* turns into a charged top-level cost — with *no* run-existence assumption, in the genuine $2 \le p \le k$ regime where the cap covers the path.

theorem *path-k-charged-cost-exists*:
  assumes *pk-le*: *sssp-log-one-third-param N* ≤ *k*
    and *cover*: *Suc k* ≤ *bmssp-level-cap* (*sssp-log-one-third-param N*)

(*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)
and *p2*: *2* ≤ *sssp-log-one-third-param N*
shows ∃ *c*. *bpi.charged-direct-insert-top-level-cost 1 N c*
⟨*proof*⟩

Cost-bounded level-1 base. The cost-tracking twin of *path-k-base1-general*: the level-1 charged run built by the empty-band single-step-then-Done loop has cost exactly one FindPivots scan plus one base-case scan, hence at most *p* ∗ *bpi.edge-count* + *p*. All four partition costs and the Done tail are zero, so no edge-range or chain terms enter — this is what keeps the cheap run polylogarithmic.

lemma *path-k-base1-costed*:
fixes *p q cap* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *pk-le*: *p* ≤ *k* **and** *cover*: *Suc k* ≤ *cap* **and** *ppos*: *0* < *p*
and *capgt*: *1* < *cap*
and *sound*: *pk.sound-label d* **and** *d0*: *d 0* = *0*
and *pre0*: *pk.bmssp-pre-full d* {*0*} *Infinity*
shows ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap* (*Suc 0*) *d* {*0*} (*Fin* (*real* (*Suc k*)))
(*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
∧ *c* ≤ *p* ∗ *bpi.edge-count* + *p*
⟨*proof*⟩

Cost-bounded cover lift (*l* → *Suc l*). The cost-tracking twin of *path-k-cover-lift*: lifting a level-*l* whole-path cover of cost at most *Cb* to level *Suc l* adds exactly one FindPivots scan (cost ≤ *p* · *edge-count*); all four partition costs and the Done tail are zero. Stacking this from the level-1 base accumulates one scan per level, giving the polylogarithmic cheap-run cost.

lemma *path-k-cover-lift-costed*:
fixes *p q cap l Cb* :: *nat* **and** *d* :: *nat* ⇒ *real*
assumes *pk-le*: *p* ≤ *k* **and** *cover*: *Suc k* ≤ *cap*
and *ppos*: *0* < *p*
and *sound*: *pk.sound-label d* **and** *d0*: *d 0* = *0*
and *pre0*: *pk.bmssp-pre-full d* {*0*} *Infinity*
and *child*: ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap l*
(*bpi.find-pivots-label-capped p cap d* {*0*} (*Fin* (*real* (*Suc k*))))
{*0*} (*Fin* (*real* (*Suc k*))) (*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
∧ *c* ≤ *Cb*
shows ∃ *U c*. *bpi.charged-direct-insert-costed-bmssp* (*Suc 0*)
(*bmssp-level-cap p q*) *q p p cap* (*Suc l*) *d* {*0*} (*Fin* (*real* (*Suc k*)))
(*pk.dist 0*) (*Fin* (*real* (*Suc k*))) *U c*
∧ *c* ≤ *Cb* + *p* ∗ *bpi.edge-count*
⟨*proof*⟩

Cost-bounded level chain. Stacking the cost-bounded cover lift from the cost-bounded level-1 base, a whole-path cover of cost at most (*Suc l*) · (*p* · *edge-count*) + *p* exists at every level *Suc l*: each level adds one FindPivots

scan. Specialising at $l = p - 1$ bounds the single charged child run feeding the top-level reduction by $p \cdot (p \cdot \textit{edge-count}) + p$, which is polylogarithmic in the schedule.

lemma *path-k-cover-ge1-costed*:
fixes *p q cap* :: *nat*
assumes *pk-le*: $p \leq k$ **and** *cover*: $\textit{Suc}\ k \leq \textit{cap}$ **and** *ppos*: $0 < p$
shows $\bigwedge d.\ \textit{pk.sound-label}\ d \Longrightarrow d\ 0 = 0 \Longrightarrow \textit{pk.bmssp-pre-full}\ d\ \{0\}\ \textit{Infinity}$
$\Longrightarrow$
$\exists\, U\ c.\ \textit{bpi.charged-direct-insert-costed-bmssp}\ (\textit{Suc}\ 0)\ (\textit{bmssp-level-cap}\ p\ q)\ q\ p\ p$
$\textit{cap}\ (\textit{Suc}\ l)\ d\ \{0\}\ (\textit{Fin}\ (\textit{real}\ (\textit{Suc}\ k)))\ (\textit{pk.dist}\ 0)\ (\textit{Fin}\ (\textit{real}\ (\textit{Suc}\ k)))\ U\ c$
$\wedge\ c \leq \textit{Suc}\ l * (p * \textit{bpi.edge-count}) + p$
⟨*proof*⟩

Cost-bounded top-level reduction. The cost-tracking twin of *path-k-charged-cost-exists-if-child*: the source-pivot top-level run pulls the sole source pivot once, runs the bounded level-$p-1$ child of cost at most Cb, and terminates (the direct batch is empty on the path). Its cost is one more FindPivots scan on top of the child, hence at most $Cb + p \cdot \textit{edge-count}$.

lemma *path-k-charged-cost-le-if-child*:
fixes *Cb* :: *nat*
assumes *pk-le*: $\textit{sssp-log-one-third-param}\ N \leq k$
and *cover*: $\textit{Suc}\ k \leq \textit{bmssp-level-cap}\ (\textit{sssp-log-one-third-param}\ N)$
$(\textit{sssp-log-two-thirds-param}\ N)\ (\textit{sssp-log-one-third-param}\ N)$
and *ppos*: $0 < \textit{sssp-log-one-third-param}\ N$
and *child*:
$\exists\, \textit{U-child}\ \textit{c-child}.$
$\textit{bpi.charged-direct-insert-costed-bmssp}\ (\textit{Suc}\ 0)$
$(\textit{bmssp-level-cap}\ (\textit{sssp-log-one-third-param}\ N)\ (\textit{sssp-log-two-thirds-param}\ N))$
$(\textit{sssp-log-two-thirds-param}\ N)\ (\textit{sssp-log-one-third-param}\ N)$
$(\textit{sssp-log-one-third-param}\ N)$
$(\textit{bmssp-level-cap}\ (\textit{sssp-log-one-third-param}\ N)\ (\textit{sssp-log-two-thirds-param}\ N)$
$(\textit{sssp-log-one-third-param}\ N))$
$(\textit{sssp-log-one-third-param}\ N - 1)$
$(\textit{bpi.find-pivots-label-capped}\ (\textit{sssp-log-one-third-param}\ N)$
$(\textit{bmssp-level-cap}\ (\textit{sssp-log-one-third-param}\ N)\ (\textit{sssp-log-two-thirds-param}\ N)$
$(\textit{sssp-log-one-third-param}\ N))$
$\textit{pk.finite-initial-label}\ \{0\}\ \textit{Infinity})$
$\{0\}\ (\textit{Fin}\ (\textit{real}\ (\textit{Suc}\ k)))\ (\textit{pk.dist}\ 0)\ (\textit{Fin}\ (\textit{real}\ (\textit{Suc}\ k)))\ \textit{U-child}\ \textit{c-child}$
$\wedge\ \textit{c-child} \leq \textit{Cb}$
shows $\exists\, c.\ \textit{bpi.charged-direct-insert-top-level-cost}\ 1\ N\ c$
$\wedge\ c \leq \textit{Cb} + \textit{sssp-log-one-third-param}\ N * \textit{bpi.edge-count}$
⟨*proof*⟩

The cheap charged top-level cost is polylogarithmic. Assembling the cost-bounded level chain (to level $p-1$) and the cost-bounded top-level

reduction: in the genuine $2 \le p \le k$ pivot regime where the cap covers the path, a charged top-level cost exists that is bounded by $p \cdot (p \cdot \textit{edge-count}) + p$ — a single *square-of-the-pivot-rounds* times edge factor. No band-containment or amortized engine is needed: the cheap run pays one FindPivots scan per recursion level plus one base scan.

theorem *path-k-charged-cost-le*:
  assumes *pk-le*: *sssp-log-one-third-param N* $\le$ *k*
    and *cover*: *Suc k* $\le$ *bmssp-level-cap* (*sssp-log-one-third-param N*)
      (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)
    and *p2*: *2* $\le$ *sssp-log-one-third-param N*
  shows $\exists$ *c*. *bpi.charged-direct-insert-top-level-cost 1 N c*
      $\wedge$ *c* $\le$ *sssp-log-one-third-param N* $*$ (*sssp-log-one-third-param N* $*$ *bpi.edge-count*)
          + *sssp-log-one-third-param N*
⟨*proof*⟩

Charged-cost structural facts. The next facts record permissive-relation structure on the path: a returned bound that covers the path forces the whole vertex set into the bounded tree, a level-0 base settles at most $kk \le k$ vertices, and (the harder fact) a source-pivot loop whose data structure already excludes the source but is still nonempty can never finish. They are reusable ingredients for bounding charged costs.

A covering returned bound forces the whole path into the bounded tree.

lemma *path-k-cover-imp-V-subset*:
  assumes *cov*: *bound-le* (*Fin* (*real* (*Suc k*))) *B′*
  shows *path-k-V k* $\subseteq$ *pk.bound-tree* {*0*} *B′*
⟨*proof*⟩

Base of the no-coverage induction. A level-0 charged base run from the source settles at most $kk \le k$ vertices, so its returned bound cannot cover the *Suc k* path vertices.

lemma *path-k-charged-base-noncover*:
  fixes *kk* :: *nat*
  assumes *kk-le*: *kk* $\le$ *k*
    and *run*: *bpi.charged-direct-insert-costed-bmssp* Δ *M-of t h kk cap 0 d* {*0*} *B d′ B′ U c*
    and *sound*: *pk.sound-label d*
    and *pre*: *pk.bmssp-pre-full d* {*0*} *B*
  shows ¬ *bound-le* (*Fin* (*real* (*Suc k*))) *B′*
⟨*proof*⟩

Frontier persistence (the undone band-monotonicity invariant, on the charged cost relation). On the source-pivot path a partition loop whose current data structure already excludes the source *0* but is still nonempty can never finish: the only pullable pivot is *0* (every *split-below d* {*0*} lies in {*0*}), so each remaining step is a null pull, and the relaxation only ever pushes strictly positive frontier vertices (*path-k-relax-in-bound-keys-pos*,

path-k-relax-between-keys-pos). Hence the nonempty, source-free frontier is preserved forever and neither *Done* (empty keys) nor *Stop* (cap threshold, excluded by $Suc\ k < kk \cdot cap$) can fire.

lemma *path-k-loop-frontier-persist*:
  bpi.charged-direct-insert-costed-partition-loop-state Δ *M-of t h kk cap l d P B d′*
    D a betas bs B′ charged-Us child-outputs U c child-costs $\Longrightarrow$
    $P = \{0\} \Longrightarrow Suc\ k < kk * cap \Longrightarrow (0::nat) \notin keys\text{-}of\ D \Longrightarrow keys\text{-}of\ D \neq \{\}$
    $\Longrightarrow$ *False*
  and *bpi.charged-direct-insert-costed-bmssp* Δ *M-of t h kk cap l d S B d′ B′ U c*
    $\Longrightarrow$ *True*
⟨*proof*⟩

end

10.5 The bucket cap dominates the path size at the inflated schedule

abbreviation *cap-at* :: *nat* $\Rightarrow$ *nat* **where**
  cap-at N $\equiv$ *bmssp-level-cap* (*sssp-log-one-third-param N*)
    (*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)

lemma *param-one-third-ge-factor*:
  sssp-log-factor-one-third N $\leq$ *real* (*sssp-log-one-third-param N*)
⟨*proof*⟩

lemma *param-two-thirds-ge-factor*:
  sssp-log-factor N $\leq$ *real* (*sssp-log-two-thirds-param N*)
⟨*proof*⟩

lemma *ln-le-param-product*:
  ln (*real N* + *2*)
    $\leq$ *real* (*sssp-log-one-third-param N*) $*$ *real* (*sssp-log-two-thirds-param N*)
⟨*proof*⟩

lemma *powr-ln2-le-cap*:
  (*real N* + *2*) *powr ln 2* $\leq$ *real* (*cap-at N*)
⟨*proof*⟩

The decisive arithmetic fact (milestone **B0**, deliverable (G2)): at the inflated schedule $n \cdot n$ the top-level bucket cap eventually covers all $Suc\ n$ path vertices.

theorem *eventually-card-path-le-cap-inflated*:
  eventually (λn. *card* (*path-k-V n*) $\leq$ *cap-at* ($n * n$)) *at-top*
⟨*proof*⟩

10.6 The inflated schedule index stays below the graph size

Growth upper bound (used so that the prefix $\{0..p\}$ of short-tight witnesses fits inside the path): at the inflated schedule $k \cdot k$ the FindPivots round

parameter p = *sssp-log-one-third-param* $(k \cdot k)$ is eventually at most the path length k, because $p \sim (ln\ (k^2 + 2))^{1/3}$ grows far slower than k.

lemma *path-param-le-self*:
eventually (λk::*nat*. *sssp-log-one-third-param* $(k * k) \leq k$) *at-top*
⟨*proof*⟩

10.7 The inflated schedule is genuine and keeps the polylog in $\Theta(ln\ n)$

lemma *nat-le-mult-self*: (n::*nat*) $\leq n * n$
⟨*proof*⟩

theorem *inflated-schedule-at-top*:
filterlim (λn::*nat*. $n * n$) *at-top at-top*
⟨*proof*⟩

theorem *ln-inflated-schedule*:
assumes *1* $\leq n$
shows *ln* (*real* $(n * n)$) = *2* $*$ *ln* (*real* n)
⟨*proof*⟩

Deliverable (G1): the graph size itself is unbounded.

theorem *path-card-V-at-top*:
filterlim (λn. *card* (*path-k-V* n)) *at-top at-top*
⟨*proof*⟩

The inflated-schedule pivot regime (B1 staging). Three facts hold together for all but finitely many n: the FindPivots round count p = *sssp-log-one-third-param* $(n \cdot n)$ has grown past *1* (so the level chain $p - 1 \geq 1$ is non-degenerate), it stays below the path length n, and the inflated bucket cap covers the *Suc n* path vertices. These are exactly the hypotheses $p \leq k$, *Suc* $k \leq$ *cap*, $0 < p$ of the path cover bricks at $k = n$, packaged as a single eventual statement to feed the (G3) reduction.

theorem *eventually-inflated-pivot-regime*:
eventually (λn. *2* $\leq$ *sssp-log-one-third-param* $(n * n)$
$\wedge$ *sssp-log-one-third-param* $(n * n) \leq n$
$\wedge$ *Suc* $n \leq$ *cap-at* $(n * n)$) *at-top*
⟨*proof*⟩

10.8 The decoupled charged running time of the path family

definition *path-nonvac-time* :: *nat* ⇒ *nat* **where**
path-nonvac-time n =
strict-tie-breaking-digraph.charged-direct-insert-top-level-time
(*path-k-V* n) (*path-k-E* n) (*path-k-weight* n) *0 1* $(n * n)$

The decoupled charged reduction (milestone **B1**, deliverable (G3)): whenever the source pivot finishes on P_n at the inflated schedule $n \cdot n$, a charged

top-level run — hence a charged top-level cost — exists. This is the schedule-decoupled analogue of *eventually-path-family-charged-direct-insert-top-level-cost-if-source-pivot-finishes*.

theorem *eventually-path-nonvac-charged-cost-exists-if-source-pivot-finishes*:
assumes *source-pivot-finishes*:
eventually
(λ*n*. *strict-tie-breaking-digraph.charged-direct-insert-source-pivot-finishes*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*)) *at-top*
shows *eventually*
(λ*n*. ∃ *c*. *strict-tie-breaking-digraph.charged-direct-insert-top-level-cost*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c*) *at-top*
⟨*proof*⟩

Milestone B1, fully discharged. Dropping the *source-pivot-finishes* hypothesis: at the inflated schedule the genuine $2 \le p \le n$ pivot regime holds eventually (*eventually-inflated-pivot-regime*), so the unconditional charged-cost existence *path-k-charged-cost-exists* applies for all but finitely many n. A charged top-level run on the decoupled family therefore exists eventually, with *no* assumption.

theorem *eventually-path-nonvac-charged-cost-exists*:
eventually
(λ*n*. ∃ *c*. *strict-tie-breaking-digraph.charged-direct-insert-top-level-cost*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c*) *at-top*
⟨*proof*⟩

Status of the unconditional discharge of *source-pivot-finishes* **(G3).** The reduction above turns (G3) into the obligation

- *eventually* (λ*n*. *charged-direct-insert-source-pivot-finishes* (P_n) *0 1* (*n* · *n*)) *at-top*. Its first conjunct $P = \{s\}$ is supplied by the reusable brick *unique-shortest-digraph.find-pivots-pivots-capped-singleton-source-if-large-tree*, whose two premises specialise to the path as follows. The genuine distance *path-k-dist-eq* and the source-tree identity *path-k-tree-of-source* collapse the source-tree slice to the capped scan, and the no-overflow premise is *path-k-find-pivots-seen-card-le* together with the cap domination below. What remains for $P = \{s\}$ is the seen *lower* bound $p \le$ *card* (*find-pivots-seen-capped* ...), i.e. that the p-round bounded scan from $\{0\}$ reaches at least p path vertices; with *path-k-dist-eq* every prefix edge is tight, so the prefix $\{0..p\}$ are short-tight witnesses inside the scan. The remaining conjuncts of *source-pivot-finishes* (the band child run and the empty direct-edge batch) are then a multi-step loop-coverage construction that pulls the path band by band, each child closed by the empty-pivots route on a sub-band that fits below p, and is the substance of milestone **B1**.

10.9 The decoupled charged headline (conditional skeleton for B3)

Mirroring the diagonal conditional *bmssp-path-family-charged-direct-insert-runtime-bigo-size-if-cost-bounded*, but on the *decoupled* family P_n run at schedule $n \cdot n$: if charged top-level runs eventually exist and every charged cost is bounded by the refined graph-time budget at the inflated schedule, the least charged cost has the genuine size-parametric target $O(n \cdot (ln\ n)^{2/3})$. Unlike the diagonal version, the two hypotheses here are *not* obstructed: the cap covers the path (see *eventually-card-path-le-cap-inflated*), so the goal of milestones **B1**/**B2** is to discharge them unconditionally.

theorem *path-nonvac-runtime-bigo-if-runs-and-cost-bound*:
assumes *charged-runs-exist*:
eventually
(λn. ∃ c. *strict-tie-breaking-digraph.charged-direct-insert-top-level-cost*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c*) *at-top*
and *charged-cost-bound*:
eventually
(λn. ∀ c.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c* ⟶
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *n*) (*Suc n*)) *at-top*
shows (λn. *real* (*path-nonvac-time n*))
∈ *O*(λn. *real n* ∗ (*ln* (*real n* + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

The same conditional headline phrased in deliverable (G5)'s exact form, in the *genuine vertex count card* (*path-k-V n*). This is the precise statement that milestones **B1**/**B2** must make unconditional.

theorem *path-nonvac-runtime-bigo-card-V-if-runs-and-cost-bound*:
assumes *charged-runs-exist*:
eventually
(λn. ∃ c. *strict-tie-breaking-digraph.charged-direct-insert-top-level-cost*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c*) *at-top*
and *charged-cost-bound*:
eventually
(λn. ∀ c.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c* ⟶
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))

```
      (λ-. sssp-log-one-third-param (n * n))
      (λ-. sssp-log-one-third-param (n * n))
      (λ-. sssp-log-two-thirds-param (n * n))
      (λ-. n) (Suc n)) at-top
  shows (λn. real (path-nonvac-time n))
    ∈ O(λn. real (card (path-k-V n)) *
        (ln (real (card (path-k-V n)) + 2)) powr (2 / 3))
⟨proof⟩
```

Factoring the cost hypothesis through the library cost bound (B2 scaffolding). The verbatim *bmssp-refined-graph-time-bound* hypothesis of the headline above is exactly what the proven library theorem *strict-tie-breaking-digraph.charged-direct-insert-top-level-cost-refined-bound-log-params-fixed-degree* delivers from the canonical closed-bound predicate *strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-params-fixed-degree* on the decoupled family (the path is fully reachable and unit out-degree, so the two side conditions are automatic). Discharging *charged-cost-bound* therefore reduces to the canonical closed-bound predicate at the inflated schedule, which is the single remaining obligation of milestone **B2**.

```
lemma eventually-path-nonvac-cost-bound-if-closed-bound:
  assumes closed-bound:
    eventually
      (λn. strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-
params-fixed-degree
        (path-k-V n) (path-k-E n) (path-k-weight n) 0 1 (n * n)) at-top
  shows eventually
    (λn. ∀ c.
      strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
        (path-k-V n) (path-k-E n) (path-k-weight n) 0 1 (n * n) c ⟶
      c ≤ bmssp-refined-graph-time-bound
        (λ-. Suc 1 * sssp-log-one-third-param (n * n))
        (λ-. sssp-log-two-thirds-param (n * n))
        (λ-. sssp-log-one-third-param (n * n))
        (λ-. sssp-log-one-third-param (n * n))
        (λ-. sssp-log-two-thirds-param (n * n))
        (λ-. n) (Suc n)) at-top
  ⟨proof⟩
```

The headline phrased against the canonical obligations: top-level runs exist and the closed refined bound holds, both eventually. This trims **B3** to the two library-shaped premises that **B1**/**B2** must make unconditional.

```
theorem path-nonvac-runtime-bigo-card-V-if-runs-and-closed-bound:
  assumes charged-runs-exist:
    eventually
      (λn. ∃ c. strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
        (path-k-V n) (path-k-E n) (path-k-weight n) 0 1 (n * n) c) at-top
    and closed-bound:
```

eventually
(λn. *strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-params-fixed-degree*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*)) *at-top*
shows (λn. *real* (*path-nonvac-time n*))
∈ *O*(λn. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2 / 3*))
⟨*proof*⟩

Milestone B1 applied to the headline. The run-existence premise of the headline is now genuinely *discharged* (not merely bypassed by the vacuous least-cost constant): *eventually-path-nonvac-charged-cost-exists* proves a charged top-level run exists for all but finitely many n, so the headline reduces to the single canonical cost obligation *closed-bound* (**B2**). This makes the runtime claim genuinely non-vacuous: the bounded quantity is the cost of a run that provably exists.

theorem *path-nonvac-runtime-bigo-card-V-if-closed-bound-runs-discharged*:
assumes *closed-bound*:
eventually
(λn. *strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-params-fixed-degree*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*)) *at-top*
shows (λn. *real* (*path-nonvac-time n*))
∈ *O*(λn. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2 / 3*))
⟨*proof*⟩

10.10 Dropping the run-existence premise (B1, unconditional in runs)

lemma *Least-le-or-const*:
fixes *P* :: *nat* ⇒ *bool* **and** *B* :: *nat*
assumes ⋀c. *P c* ⟹ *c* ≤ *B*
shows (*LEAST c*. *P c*) ≤ *B* + (*LEAST c*::*nat*. ¬ *True*)
⟨*proof*⟩

lemma *path-refined-bound-bigo*:
(λn. *real* (*bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *n*) (*Suc n*)))
∈ *O*(λn. *sssp-time-target* (λn. *n*) *n*)
⟨*proof*⟩

lemma *sssp-time-target-self-ge*:
eventually (λn::*nat*. *real K* ≤ *sssp-time-target* (λn. *n*) *n*) *at-top*

⟨*proof*⟩

lemma *const-bigo-target*:
(λ*n*::*nat*. *real K*) ∈ *O*(λ*n*. *sssp-time-target* (λ*n*. *n*) *n*)
⟨*proof*⟩

Milestone B1, unconditional in run-existence. The charged running time of the decoupled path family is size-parametric O of the SSSP time target as soon as every charged top-level cost is bounded by the refined graph-time budget — with *no* assumption that a charged run exists. When a run exists the least cost is bounded by the budget; when none exists the least-cost is a fixed constant, which the divergent target dominates.

theorem *path-nonvac-runtime-bigo-if-cost-bound*:
assumes *charged-cost-bound*:
eventually
(λ*n*. ∀ *c*.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c* ⟶
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *n*) (*Suc n*)) *at-top*
shows (λ*n*. *real* (*path-nonvac-time n*))
∈ *O*(λ*n*. *real n* ∗ (*ln* (*real n* + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

The same statement in the genuine vertex count, and reduced to the canonical closed refined bound: both now *without* the run-existence premise.

theorem *path-nonvac-runtime-bigo-card-V-if-cost-bound*:
assumes *charged-cost-bound*:
eventually
(λ*n*. ∀ *c*.
strict-tie-breaking-digraph.charged-direct-insert-top-level-cost
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*) *c* ⟶
c ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *n*) (*Suc n*)) *at-top*
shows (λ*n*. *real* (*path-nonvac-time n*))
∈ *O*(λ*n*. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

theorem *path-nonvac-runtime-bigo-card-V-if-closed-bound*:
assumes *closed-bound*:
eventually
(λ*n*. *strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-params-fixed-degree*
(*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0 1* (*n* ∗ *n*)) *at-top*
shows (λ*n*. *real* (*path-nonvac-time n*))
∈ *O*(λ*n*. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

10.11 Charged-cost upper bound via the uncharged refinement (B2, uncharged route)

The decoupled charged running time is bounded by the refined graph-time budget as soon as a single *uncharged* top-level run exists. The charged closed-bound predicate bounds *every* charged run, which on the path is genuinely too strong: the permissive charged loop admits runs at non-zero source bands whose slice accounting breaks band-containment. The headline only needs the *least* charged cost, so it suffices to exhibit one bounded run. Every uncharged run refines to a charged run of equal cost (*unique-shortest-digraph.direct-insert-costed-bmssp-refines-charged-direct-insert*), and the uncharged top-level cost is bounded unconditionally by the proven library bound *strict-tie-breaking-digraph.finite-initial-label-direct-insert-costed-top-level-correct-and-closed-bmssp-refined-graph-time-bound-level-cap-amortized* (no band-containment side condition). Hence the least charged cost — i.e. the decoupled running time — is bounded by the refined budget whenever an uncharged run exists, reducing **B2** to the genuinely *constructible* obligation of uncharged top-level run existence on the path.

context fixes *k* :: *nat*
begin
interpretation *pk*: *finite-weighted-digraph*
path-k-V k path-k-E k path-k-weight k 0
⟨*proof*⟩
interpretation *bpi*: *bounded-reduced-positive-instance*
path-k-V k path-k-E k path-k-weight k 0 1
⟨*proof*⟩

lemma *path-k-uncharged-top-run-imp-charged-cost*:
assumes *run*:
bpi.direct-insert-costed-bmssp (*1*::*nat*)
(*bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*))
(*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)
(*sssp-log-one-third-param N*) (*cap-at N*) (*sssp-log-one-third-param N*)
pk.finite-initial-label {*0*} *Infinity d*′ *Infinity U c*
shows *bpi.charged-direct-insert-top-level-cost 1 N c*

⟨*proof*⟩

lemma *path-k-charged-cost-imp-nonvac-time-le*:
assumes *cost*: *bpi.charged-direct-insert-top-level-cost 1* $(k * k)$ *c*
shows *path-nonvac-time* $k \leq c$
⟨*proof*⟩

lemma *path-k-uncharged-run-cost-bound*:
fixes N :: *nat*
assumes *p2*: $2 \leq$ *sssp-log-one-third-param N*
and *run*:
bpi.direct-insert-costed-bmssp (*1*::*nat*)
(*bmssp-level-cap* (*sssp-log-one-third-param N*) (*sssp-log-two-thirds-param N*))
(*sssp-log-two-thirds-param N*) (*sssp-log-one-third-param N*)
(*sssp-log-one-third-param N*) (*cap-at N*) (*sssp-log-one-third-param N*)
pk.finite-initial-label {*0*} *Infinity d' Infinity U c*
shows *c* $\leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* $*$ *sssp-log-one-third-param N*) (λ-. *sssp-log-two-thirds-param N*)
(λ-. *sssp-log-one-third-param N*) (λ-. *sssp-log-one-third-param N*)
(λ-. *sssp-log-two-thirds-param N*) (λ-. *k*) (*Suc k*)
⟨*proof*⟩

lemma *path-k-uncharged-run-imp-nonvac-time-le-bound*:
assumes *p2*: $2 \leq$ *sssp-log-one-third-param* $(k * k)$
and *run*:
bpi.direct-insert-costed-bmssp (*1*::*nat*)
(*bmssp-level-cap* (*sssp-log-one-third-param* $(k * k)$) (*sssp-log-two-thirds-param* $(k * k)$))
(*sssp-log-two-thirds-param* $(k * k)$) (*sssp-log-one-third-param* $(k * k)$)
(*sssp-log-one-third-param* $(k * k)$) (*cap-at* $(k * k)$) (*sssp-log-one-third-param* $(k * k)$)
pk.finite-initial-label {*0*} *Infinity d' Infinity U c*
shows *path-nonvac-time* $k \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* $*$ *sssp-log-one-third-param* $(k * k)$) (λ-. *sssp-log-two-thirds-param* $(k * k)$)
(λ-. *sssp-log-one-third-param* $(k * k)$) (λ-. *sssp-log-one-third-param* $(k * k)$)
(λ-. *sssp-log-two-thirds-param* $(k * k)$) (λ-. *k*) (*Suc k*)
⟨*proof*⟩

The cheap charged running time is below the refined graph-time budget. The genuinely *constructible* analogue of *path-k-uncharged-run-imp-nonvac-time-le-bound*: rather than assuming an (in the genuine regime non-existent) uncharged top-level run, we use the cheap charged top-level cost *path-k-charged-cost-le*, whose value $p \cdot (p \cdot \textit{edge-count}) + p$ is dominated termwise by the refined budget — the edge factor absorbs $p^2 \cdot \textit{edge-count}$ and the vertex factor absorbs the additive p. Hence the least charged cost, i.e. the decoupled running time, is below the refined budget, with *no* run-existence or cost-bound assumption.

lemma *path-k-charged-cheap-imp-nonvac-time-le-bound*:
assumes *pk-le*: *sssp-log-one-third-param* $(k * k) \leq k$
and *cover*: *Suc* $k \leq$ *bmssp-level-cap* (*sssp-log-one-third-param* $(k * k)$)
(*sssp-log-two-thirds-param* $(k * k)$) (*sssp-log-one-third-param* $(k * k)$)
and *p2*: $2 \leq$ *sssp-log-one-third-param* $(k * k)$
shows *path-nonvac-time* $k \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* $(k * k)$) (λ-. *sssp-log-two-thirds-param* $(k * k)$)
(λ-. *sssp-log-one-third-param* $(k * k)$) (λ-. *sssp-log-one-third-param* $(k * k)$)
(λ-. *sssp-log-two-thirds-param* $(k * k)$) (λ-. k) (*Suc* k)
⟨*proof*⟩

end

theorem *path-nonvac-runtime-bigo-card-V-if-time-le-bound*:
assumes *tle*:
eventually (λn. *path-nonvac-time* $n \leq$ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* $(n * n)$) (λ-. *sssp-log-two-thirds-param* $(n * n)$)
(λ-. *sssp-log-one-third-param* $(n * n)$) (λ-. *sssp-log-one-third-param* $(n * n)$)
(λ-. *sssp-log-two-thirds-param* $(n * n)$) (λ-. n) (*Suc* n)) *at-top*
shows (λn. *real* (*path-nonvac-time* n))
$\in O(\lambda n.$ *real* (*card* (*path-k-V* n)) ∗
(*ln* (*real* (*card* (*path-k-V* n)) + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

10.12 The unconditional non-vacuous size-parametric running-time bound

Milestones B1/B2/B3, fully discharged. The decoupled charged running time of the path family is *unconditionally* in the genuine size-parametric class $O(card\ V_n \cdot (ln\ (card\ V_n))^{2/3})$, with *no* run-existence, cost-bound, or closed-bound hypothesis. Combining *eventually-inflated-pivot-regime* (the genuine $2 \leq p \leq n$ pivot regime where the cap covers the path holds eventually) with *path-k-charged-cheap-imp-nonvac-time-le-bound* (the cheap charged top-level cost lies below the refined graph-time budget) discharges the time hypothesis of *path-nonvac-runtime-bigo-card-V-if-time-le-bound*.

This is the unconditional form of the conditional headline *path-nonvac-runtime-bigo-card-V-if-runs-and-closed-bound*: both of its premises (eventual charged-run existence and the canonical closed refined bound) are now dropped. The running time bounded here is genuinely non-vacuous: it is the *least* charged cost of the decoupled family, a cost of a charged top-level run that provably exists and is provably small.

theorem *path-nonvac-runtime-bigo-card-V-unconditional*:
(λn. *real* (*path-nonvac-time* n))
$\in O(\lambda n.$ *real* (*card* (*path-k-V* n)) ∗
(*ln* (*real* (*card* (*path-k-V* n)) + *2*)) *powr* (*2* / *3*))

⟨*proof*⟩

The same unconditional bound restated against the two library-shaped premises of the conditional headline, witnessing that both have been discharged: the charged runtime is in the size-parametric class with neither the run-existence premise nor the closed refined bound assumed.

theorem *path-nonvac-runtime-bigo-card-V-runs-and-closed-bound-discharged*:
(λ*n*. *real* (*path-nonvac-time n*))
∈ *O*(λ*n*. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

definition *path-uncharged-top-run* :: *nat* ⇒ (*nat* ⇒ *real*) ⇒ *nat set* ⇒ *nat* ⇒ *bool* **where**
path-uncharged-top-run n d′ U c ⟷
unique-shortest-digraph.direct-insert-costed-bmssp (*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0*
(*1*::*nat*) (*bmssp-level-cap* (*sssp-log-one-third-param* (*n* ∗ *n*)) (*sssp-log-two-thirds-param* (*n* ∗ *n*)))
(*sssp-log-two-thirds-param* (*n* ∗ *n*)) (*sssp-log-one-third-param* (*n* ∗ *n*))
(*sssp-log-one-third-param* (*n* ∗ *n*)) (*cap-at* (*n* ∗ *n*)) (*sssp-log-one-third-param* (*n* ∗ *n*))
(*finite-weighted-digraph.finite-initial-label* (*path-k-V n*) (*path-k-E n*) (*path-k-weight n*) *0*)
{*0*} *Infinity d′ Infinity U c*

lemma *eventually-path-nonvac-time-le-bound-if-uncharged-runs*:
assumes *ex*: *eventually* (λ*n*. ∃ *d′ U c*. *path-uncharged-top-run n d′ U c*) *at-top*
shows *eventually* (λ*n*. *path-nonvac-time n* ≤ *bmssp-refined-graph-time-bound*
(λ-. *Suc 1* ∗ *sssp-log-one-third-param* (*n* ∗ *n*)) (λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*))
(λ-. *sssp-log-one-third-param* (*n* ∗ *n*)) (λ-. *sssp-log-one-third-param* (*n* ∗ *n*))
(λ-. *sssp-log-two-thirds-param* (*n* ∗ *n*)) (λ-. *n*) (*Suc n*)) *at-top*
⟨*proof*⟩

theorem *path-nonvac-runtime-bigo-card-V-if-uncharged-runs*:
assumes *ex*: *eventually* (λ*n*. ∃ *d′ U c*. *path-uncharged-top-run n d′ U c*) *at-top*
shows (λ*n*. *real* (*path-nonvac-time n*))
∈ *O*(λ*n*. *real* (*card* (*path-k-V n*)) ∗
(*ln* (*real* (*card* (*path-k-V n*)) + *2*)) *powr* (*2* / *3*))
⟨*proof*⟩

10.13 Charged closed-bound engine (B2): the charged level recurrence solver

The charged top-level closed-bound predicate *strict-tie-breaking-digraph.charged-direct-insert-closed-refined-bound-log-params-fixed-degree* bounds *every* charged top-level run. The uncharged analogue is already

solved by *strict-tie-breaking-digraph.direct-insert-costed-bmssp-level-bound-from-local-budgets-with-invariants-and-edge-budget*; here we build the charged analogue, threading the charged per-step closer *strict-tie-breaking-digraph.charged-direct-insert-costed-nonbase-step-closes-level-bound-from-child-bound-with-invariants-and-edge-budget* through the recursion. The two semantic obligations the charged closer carries — the loop edge budget and the source-progress (no-stranding) invariant — are discharged generically here (edge budget) and on the path below (source progress).

context *strict-tie-breaking-digraph*
begin

Charged analogue of *direct-insert-costed-partition-loop-state-trivial-edge-budget*: the separated direct/lower edge-range sums of a charged loop step are dominated by the outgoing-edge count of the produced loop output, with combined factor $t + h$.

lemma *charged-direct-insert-costed-partition-loop-state-trivial-edge-budget*:
assumes *run*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d P B*
and *P-reaches*: $\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*
shows $t *$ *sum-list* (*map card* (*range-tree-child-direct-edge-range-list P B a betas bs*)) +
$h *$ *sum-list* (*map card* (*range-tree-child-edge-range-list P a betas bs*))
$\leq (t + h) *$ *card* (*outgoing-edges U*)
⟨*proof*⟩

Per-child amortized output → slice converter (B2 engine). The charged amortized loop closer *charged-direct-insert-costed-partition-loop-state-closes-amortized-bound-from-child-costs* consumes per-child amortized bounds stated against the *disjoint range-tree slices*. A recursion-level induction instead produces per-child bounds against the child *outputs*. Whenever each output sits inside its slice (the band-containment maintained by the descending recursion), the amortized budget's monotonicity *bmssp-amortized-cost-bound-mono* lifts the output bounds to the slice bounds the closer needs.

lemma *list-all2-bmssp-amortized-cost-bound-lift-subset*:
assumes *outb*:
list-all2 (λ*c-child UB*.
c-child ≤ *bmssp-amortized-cost-bound A R h t q L* (*snd UB*) (*fst UB*))
child-costs output-pairs
and *contain*:
list-all2 (λ*OB SB*.
fst OB ⊆ *fst SB* ∧ *snd OB* = *snd SB* ∧ *finite* (*fst SB*))
output-pairs slice-pairs
shows *list-all2* (λ*c-child UB*.
c-child ≤ *bmssp-amortized-cost-bound A R h t q L* (*snd UB*) (*fst UB*))

child-costs slice-pairs
⟨*proof*⟩

Band containment (B2 engine): a settled child output sits inside its slice. The descending recursion maintains, at a loop step with lower band a, two modular invariants of the working label d over the pivot set S: a *label floor* $a \leq d\ x$ and *prefix exactness* below a. Under those a pulled child's whole produced tree *bound-tree S B* lies at or above the band, hence inside the band-range slice *range-tree S a B* — using only *tree-path-dist-le* and the floor/exactness reductions of the note preceding *split-below-inter-bound-tree-empty-if-label-ge-exact*.

lemma *bound-tree-subset-range-tree-if-floor-exact*:
assumes *floor*: $\bigwedge x.\ x \in S \Longrightarrow a \leq d\ x$
and *exact*: $\bigwedge u.\ u \in S \Longrightarrow dist\ s\ u < a \Longrightarrow d\ u = dist\ s\ u$
shows *bound-tree S B* $\subseteq$ *range-tree S a B*
⟨*proof*⟩

lemma *bound-tree-split-below-subset-range-tree-if-floor-exact*:
assumes *floor*: $\bigwedge x.\ x \in P \Longrightarrow a \leq d\ x$
and *exact*: $\bigwedge u.\ u \in P \Longrightarrow dist\ s\ u < a \Longrightarrow d\ u = dist\ s\ u$
shows *bound-tree* (*split-below d P beta*) *B* $\subseteq$ *range-tree P a B*
⟨*proof*⟩

Charged per-child amortized slice-bound propagator (B2 engine). The charged amortized loop closer *charged-direct-insert-costed-partition-loop-state-closes-amortized-bound-from-child-costs* consumes per-child amortized bounds stated against the disjoint range-tree slices *range-tree-child-bound-pair-list P a betas bs*. A recursion-level induction supplies per-child bounds against the child *outputs*; whenever each output sits inside its slice — a *list-all2* band-containment that the descending recursion threads as data (and which is trivial on the source band $a = 0$, where the slice is the full bounded tree) — the amortized budget's monotonicity lifts the output bounds to the slice bounds. Unlike the unusable library closers, the per-child budget here is stated against *outputs*, so it is dischargeable by an ordinary inductive hypothesis.

lemma *charged-child-amortized-slice-bounds-from-output-and-containment*:
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l d P B d′ D a betas bs B′ charged-Us child-outputs U c child-costs* $\Longrightarrow$
$P \subseteq V \Longrightarrow$
$k * card\ P \leq cap \Longrightarrow$
tree-antichain P $\Longrightarrow$
$(\bigwedge$*c-child U-child S-child B-child d-child B-child′*.
⟦*charged-direct-insert-costed-bmssp* Δ *M-of t h k cap l d S-child B-child d-child B-child′ U-child c-child*;
bmssp-pre-full d S-child B-child;
$\bigwedge x.\ x \in$ *S-child* $\Longrightarrow$ *reachable s x*;
card S-child $\leq$ *M-of l*;

$\bigwedge x.\ x \in S\text{-}child \Longrightarrow below\text{-}bound\ (d\ x)\ B\text{-}child;$
$k * card\ S\text{-}child \leq cap;$
$tree\text{-}antichain\ S\text{-}child\rrbracket$
$\Longrightarrow c\text{-}child \leq bmssp\text{-}amortized\text{-}cost\text{-}bound\ A\ R\ h\ t\ q\ L\ B\text{-}child\ U\text{-}child) \Longrightarrow$
$list\text{-}all2\ (\lambda U\text{-}out\ SB.\ U\text{-}out \subseteq fst\ SB)\ child\text{-}outputs$
$(range\text{-}tree\text{-}child\text{-}bound\text{-}pair\text{-}list\ P\ a\ betas\ bs) \Longrightarrow$
$list\text{-}all2\ (\lambda c\text{-}child\ SB.$
$case\ SB\ of\ (U\text{-}child,\ B\text{-}child) \Rightarrow$
$c\text{-}child \leq bmssp\text{-}amortized\text{-}cost\text{-}bound\ A\ R\ h\ t\ q\ L\ B\text{-}child\ U\text{-}child)$
$child\text{-}costs\ (range\text{-}tree\text{-}child\text{-}bound\text{-}pair\text{-}list\ P\ a\ betas\ bs)$
and *charged-direct-insert-costed-bmssp-child-amortized-slice-bounds-trivial*:
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}bmssp\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l\ d\ S\ B\ d'\ B'\ U\ c \Longrightarrow True$
⟨*proof*⟩

Charged amortized level recurrence (B2 engine). The charged analogue of *direct-insert-costed-bmssp-amortized-bound-from-local-budgets-with-invariants*: a strong-induction on the recursion level closes the charged run cost against the amortized graph budget *bmssp-amortized-cost-bound* of the produced output U. The per-loop scan/insert budget, the source-progress (settledness) oracle, and the band-containment of child outputs are taken as the modular threaded side-conditions *step-budget*, *source-progress* and *child-containment*; each is discharged on the decoupled path below, where the source band $a = 0$ makes them transparent.

lemma *charged-direct-insert-costed-bmssp-amortized-bound-from-budgets-progress-containment*:
assumes *base-budget*: $\Delta \leq A$
and *R-pos*: $0 < R$
and *source-factor*: $Suc\ h \leq A$
and *k-pos*: $0 < k$
and *M-cap*: $\bigwedge i.\ i \leq l \Longrightarrow M\text{-}of\ i \leq cap$
and *step-budget*:
$\bigwedge l\ d\ S\ B\ D\ c\text{-}insert\ d'\ a\ betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\text{-}loop$
$c\text{-}loop\ child\text{-}costs\ U.$
$D = label\text{-}partition\text{-}view$
$(find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B)$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B) \Longrightarrow$
$partition\text{-}initial\text{-}insert\text{-}cost\text{-}bound\ c\text{-}insert\ t$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B) \Longrightarrow$
$charged\text{-}direct\text{-}insert\text{-}costed\text{-}partition\text{-}loop\text{-}state\ \Delta\ M\text{-}of\ t\ h\ k\ cap\ l$
$(find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B)$
$(find\text{-}pivots\text{-}pivots\text{-}capped\ k\ cap\ d\ S\ B)\ B\ d'\ D\ a\ betas\ bs\ B'$
$charged\text{-}Us\ child\text{-}outputs\ U\text{-}loop\ c\text{-}loop\ child\text{-}costs \Longrightarrow$
$complete\text{-}on\ d'$
$\{v \in bound\text{-}tree\ S\ B'.$
$find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B\ v = dist\ s\ v\} \Longrightarrow$
$U = U\text{-}loop \cup$
$\{v \in bound\text{-}tree\ S\ B'.$
$find\text{-}pivots\text{-}label\text{-}capped\ k\ cap\ d\ S\ B\ v = dist\ s\ v\} \Longrightarrow$

sound-label d $\Longrightarrow$
bmssp-pre-full d S B $\Longrightarrow$
$(\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*$) \Longrightarrow$
$(\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* $(d\ x)\ B) \Longrightarrow$
$k * card\ S \leq cap \Longrightarrow$
tree-antichain S $\Longrightarrow$
fp-iter-capped-scan-cost k cap d S S B $+$ *c-insert* $\leq A * card\ U$
and *source-progress*:
$\bigwedge l'\ d\ P\ B\ d'\ D\ a\ betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs$
child-sources.
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l' d P B*
d' D a betas bs B' charged-Us child-outputs U c child-costs $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d P B $\Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*$) \Longrightarrow$
list-all2
(λ*S-child charged-child.* $\exists$ *c-child U-child B-child d-child B-child'.*
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l' d S-child B-child*
d-child B-child' U-child c-child $\wedge$
bmssp-pre-full d S-child B-child $\wedge$
$(\forall\, x{\in}$*S-child. reachable s x*$)\ \wedge$
card S-child $\leq$ *M-of l'* $\wedge$
$(\forall\, x{\in}$*S-child. below-bound* $(d\ x)$ *B-child*$)$)
child-sources (*range-tree-child-list P a bs*) $\Longrightarrow$
sum-list (*map card child-sources*) $\leq$ *card* (*range-tree-chain P a bs B'*)
and *child-containment*:
$\bigwedge l'\ d\ P\ B\ d'\ D\ a\ betas\ bs\ B'\ charged\text{-}Us\ child\text{-}outputs\ U\ c\ child\text{-}costs.$
charged-direct-insert-costed-partition-loop-state Δ *M-of t h k cap l' d P B*
d' D a betas bs B' charged-Us child-outputs U c child-costs $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d P B $\Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*$) \Longrightarrow$
list-all2 (λ*U-out SB. U-out* $\subseteq$ *fst SB*) *child-outputs*
(*range-tree-child-bound-pair-list P a betas bs*)
and *run*:
charged-direct-insert-costed-bmssp Δ *M-of t h k cap l d S B d' B' U c*
and *sound*: *sound-label d*
and *pre*: *bmssp-pre-full d S B*
and *S-reaches*: $\bigwedge x.\ x \in S \Longrightarrow$ *reachable s x*
and *below*: $\bigwedge x.\ x \in S \Longrightarrow$ *below-bound* $(d\ x)\ B$
and *S-k-cap*: $k * card\ S \leq cap$
and *S-anti*: *tree-antichain S*
shows $c \leq$ *bmssp-amortized-cost-bound A R h t l* $(2 * l + 1)$ *B U*
⟨*proof*⟩

Charged top-level refined-graph-time bound (B2), conditional on the two threaded settledness invariants. The charged analogue of *finite-initial-label-direct-insert-costed-top-level-correct-and-closed-bmssp-refined-graph-time-bound-level-cap-amortized*: every charged top-level run is

bounded by the refined graph-time budget, with the per-loop scan/insert budget discharged by the bucketed step-cost lemma and the two recursion-threaded oracles — source progress and child-output band-containment — left as the only side-conditions. On the decoupled path family below both reduce to the source-band settledness already isolated for the family, so this delivers the canonical closed refined bound unconditionally there.

theorem *finite-initial-label-charged-top-level-refined-bound-amortized*:
assumes *all-reachable*: $\bigwedge v.\ v \in V \Longrightarrow$ *reachable s v*
and *degree*: *edge-outdegree-le* Δ
and *degree-factor*: $\Delta \leq A$
and *R-pos*: $0 < R$
and *insert-factor*: $t \leq A * k$
and *insert-scaled-factor*: $t \leq$ *A-insert* $* k$
and *seen-scaled-factor*: $k * \Delta +$ *A-insert* $\leq 2 * A$
and *source-factor*: *Suc h* $\leq 2 * A$
and *k-pos*: $0 < k$
and *source-progress*:
$\bigwedge l'\ d\ P\ B\ d'\ D\ a$ *betas bs* B' *charged-Us child-outputs U c child-costs child-sources.*
charged-direct-insert-costed-partition-loop-state Δ (*bmssp-level-cap k t*) *t h k* (*bmssp-level-cap k t l*) $l'\ d\ P\ B$ $d'\ D\ a$ *betas bs* B' *charged-Us child-outputs U c child-costs* $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d P B $\Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*$) \Longrightarrow$
list-all2
(λ*S-child charged-child*. $\exists$ *c-child U-child B-child d-child B-child'*.
charged-direct-insert-costed-bmssp Δ (*bmssp-level-cap k t*) *t h k* (*bmssp-level-cap k t l*) $l'\ d$ *S-child B-child d-child B-child'* *U-child c-child* $\wedge$
bmssp-pre-full d S-child B-child $\wedge$
($\forall\, x \in$*S-child*. *reachable s x*) $\wedge$
card S-child $\leq$ *bmssp-level-cap k t* $l' \wedge$
($\forall\, x \in$*S-child*. *below-bound* (*d x*) *B-child*))
child-sources (*range-tree-child-list P a bs*) $\Longrightarrow$
sum-list (*map card child-sources*) $\leq$ *card* (*range-tree-chain P a bs* B')
and *child-containment*:
$\bigwedge l'\ d\ P\ B\ d'\ D\ a$ *betas bs* B' *charged-Us child-outputs U c child-costs.*
charged-direct-insert-costed-partition-loop-state Δ (*bmssp-level-cap k t*) *t h k* (*bmssp-level-cap k t l*) $l'\ d\ P\ B$ $d'\ D\ a$ *betas bs* B' *charged-Us child-outputs U c child-costs* $\Longrightarrow$
sound-label d $\Longrightarrow$
bmssp-pre-full d P B $\Longrightarrow$
$(\bigwedge x.\ x \in P \Longrightarrow$ *reachable s x*$) \Longrightarrow$
list-all2 (λ*U-out SB*. *U-out* $\subseteq$ *fst SB*) *child-outputs* (*range-tree-child-bound-pair-list P a betas bs*)
and *run*:
charged-direct-insert-costed-bmssp Δ (*bmssp-level-cap k t*) *t h k*

(*bmssp-level-cap k t l*) *l*
finite-initial-label {*s*} *Infinity d' Infinity U c*
shows $c \leq$ *bmssp-refined-graph-time-bound* (λ-. *A*) (λ-. *R*) (λ-. *h*)
(λ-. *l*) (λ-. *t*) (λ-. *edge-count*) *vertex-count*
⟨*proof*⟩

end

end

References

[1] R. Duan, J. Mao, X. Mao, X. Shu, and L. Yin. Breaking the sorting barrier for directed single-source shortest paths. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 36–44, 2025. DOI: `https://doi.org/10.1145/3717823.3718179`.

[2] A. F. Ramos, D. B. Hulak, and R. J. G. B. de Queiroz. Correctness and complexity of BMSSP in Isabelle/HOL. *Archive of Formal Proofs*, June 2026. `https://isa-afp.org/entries/BMSSP_Correctness.html`, Formal proof development.