A Tight $(1.5 + \epsilon)$-Approximation for Unsplittable Capacitated Vehicle Routing on Trees

Claire Mathieu
CNRS Paris, France

Hang Zhou
École Polytechnique, Institut Polytechnique de Paris, France

Abstract

In the unsplittable capacitated vehicle routing problem (UCVRP) on trees, we are given a rooted tree with edge weights and a subset of vertices of the tree called terminals. Each terminal is associated with a positive demand between 0 and 1. The goal is to find a minimum length collection of tours starting and ending at the root of the tree such that the demand of each terminal is covered by a single tour (i.e., the demand cannot be split), and the total demand of the terminals in each tour does not exceed the capacity of 1.

For the special case when all terminals have equal demands, a long line of research culminated in a quasi-polynomial time approximation scheme [Jayaprakash and Salavatipour, TALG 2023] and a polynomial time approximation scheme [Mathieu and Zhou, TALG 2023].

In this work, we study the general case when the terminals have arbitrary demands. Our main contribution is a polynomial time $(1.5 + \epsilon)$-approximation algorithm for the UCVRP on trees. This is the first improvement upon the 2-approximation algorithm more than 30 years ago. Our approximation ratio is essentially best possible, since it is NP-hard to approximate the UCVRP on trees to better than a 1.5 factor.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases approximation algorithms, capacitated vehicle routing, graph algorithms, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.91

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2202.05691

Funding This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

1 Introduction

In the *unsplittable capacitated vehicle routing problem (UCVRP)* on *trees*, we are given a rooted tree with edge weights and a subset of vertices of the tree called *terminals*. Each terminal is associated with a positive *demand* between 0 and 1. The root of the tree is called the *depot*. The goal is to find a minimum length collection of tours starting and ending at the depot such that the demand of each terminal is covered by a *single* tour (i.e., the demand cannot be split), and the total demand of the terminals in each tour does not exceed the *capacity* of 1.

The UCVRP on trees has been well studied in the special setting when all terminals have *equal* demands:[1] Hamaguchi and Katoh [17] gave a polynomial time 1.5-approximation; the approximation ratio was improved to 1.35078 by Asano, Katoh, and Kawashima [3] and

[1] Up to scaling, the equal demand setting is equivalent to the *unit demand* version of the capacitated vehicle routing problem in which each terminal has unit demand, and the capacity of each tour is a positive integer k.

was further reduced to 4/3 by Becker [4]; Becker and Paul [5] gave a bicriteria polynomial time approximation scheme; and very recently, Jayaprakash and Salavatipour [18] gave a quasi-polynomial time approximation scheme, based on which Mathieu and Zhou [21] designed a polynomial time approximation scheme.

In this work, we study the UCVRP on trees in the general setting when the terminals have *arbitrary* demands. Our main contribution is a polynomial time $(1.5+\epsilon)$-approximation algorithm (Theorem 1). This is the first improvement upon the 2-approximation algorithm of Labbé, Laporte, and Mercure [20] more than 30 years ago. Our approximation ratio is essentially best possible, since it is NP-hard to approximate the UCVRP on trees to better than a 1.5 factor [14].

▶ **Theorem 1.** *For any $\epsilon > 0$, there is a polynomial time $(1.5+\epsilon)$-approximation algorithm for the unsplittable capacitated vehicle routing problem on trees.*

The UCVRP on trees generalizes the UCVRP on *paths*. The latter problem has been studied extensively due to its applications in scheduling, see Section 1.1. As an immediate corollary of Theorem 1, we obtain the first polynomial time $(1.5+\epsilon)$-approximation algorithm for the UCVRP on paths. This ratio is essentially best possible, since it is NP-hard to approximate the UCVRP on paths to better than a 1.5 factor.

1.1 Related Work

Originally introduced by Dantzig and Ramser in 1959 [10], the UCVRP generalizes the *traveling salesman problem*, and is one of the most basic problems in Operations Research.

UCVRP on general metrics

The classical tour partitioning algorithm [16] introduced by Haimovich and Rinnooy Kan in 1985 was proved to be a constant-factor approximation on general metrics [2]. Very recently, Blauth, Traub, and Vygen [6] achieved the first improvement upon the tour partitioning algorithm. The best-to-date approximation ratio for general metrics stands at roughly 3.194 due to Friggstad, Mousavi, Rahgoshay, and Salavatipour [13].

UCVRP on paths

The UCVRP on paths is equivalent to the scheduling problem of minimizing the makespan on a single batch processing machine with non-identical job sizes [26]. Many heuristics have been proposed and evaluated empirically, e.g., [26, 12, 22, 9, 19, 24, 7, 1, 23]. Prior to our work, the best approximation ratio for the UCVRP on paths was 1.6 due to Wu and Lu [27].

The UCVRP on paths has also been studied in special cases. For example, when the optimal value is at least $\Omega(1/\epsilon^6)$ times the maximum distance between any terminal and the depot, asymptotic polynomial time approximation schemes are known [11, 25, 8].[2] In contrast, the algorithm in Theorem 1 applies to *any* path instance.

UCVRP in the Euclidean plane

In the two-dimensional Euclidean plane, the UCVRP admits a $(2+\epsilon)$-approximation [15].

[2] The UCVRP on paths is called the *train delivery problem* in [11, 25, 8].

2 Overview of Techniques

To prove Theorem 1, at a high level, our approach is to modify the problem and add enough structural constraints so that the structured problem contains a $(1.5 + O(\epsilon))$-approximate solution and can be solved in polynomial time by dynamic programming.

2.1 Preprocessing

We start by some preprocessing as in [21]. We assume without loss of generality that every vertex in the tree has two children, and the terminals are the leaf vertices of the tree [21]. Furthermore, we assume that the tree has bounded distances (Section 3.2). Next, we decompose the tree into *components* (Figure 1 and Section 3.3).

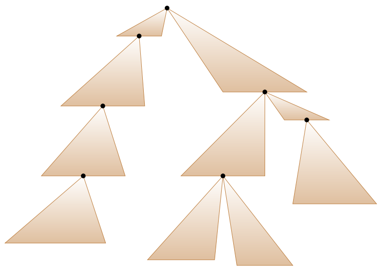


Figure 1 Decomposition of the tree into *components*. Figure extracted from [21]. Each brown triangle represents a *component*. Each component has a *root* vertex and at most one *exit* vertex.

2.2 Solutions Within Each Component

A significant difficulty is to compute solutions within each component. It would be natural to attempt to extend the approach in the setting when all terminals have equal demands [21]. In that setting, the *demands of the subtours*[3] in each component are among a polynomial number of values; since the component is visited by a constant number of tours in a near-optimal solution, that solution inside the component can be computed exactly in polynomial time using a simple dynamic program. However, when the terminals have arbitrary demands, the demands of the subtours in each component might be among an exponential number of values.[4] Indeed, unless $P = NP$, we cannot compute in polynomial time a better-than-1.5 approximate solution inside a component, since that problem generalizes the bin packing problem.

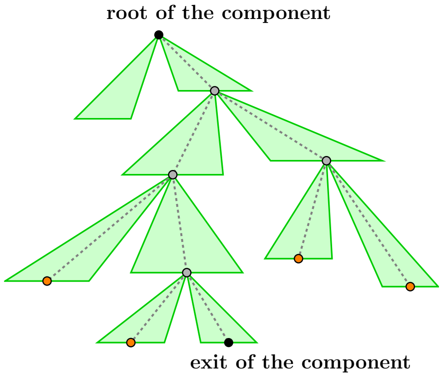
To compute in polynomial time good approximate solutions within each component, at a high level, we simplify the solution structure in each component, so that the demands of the subtours in that component are among a *constant* $O_\epsilon(1)$ number of values,[5] while increasing the cost of the solution by at most a multiplicative factor $1.5 + O(\epsilon)$.

Where does the 1.5 factor come from? Intuitively, our construction creates an additional subtour to cover a selected subset of terminals, charging each edge on that subtour to *two* existing subtours using that edge, thus adding a 0.5 factor to the cost.

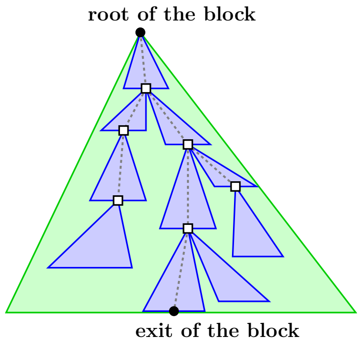
In the rest of this section, we explain our approach in more details.

---

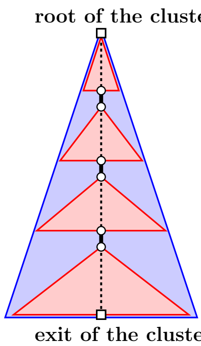
3 The *demand of a subtour* is the total demand of the terminals visited by that subtour.
4 For example, consider a component that is a star graph with $\Theta(n)$ leaves, where the i^{th} leaf has demand $1/2^i$.
5 The notation $O_\epsilon(1)$ stands for $O(f(\epsilon))$ where $f(\epsilon)$ is any function on ϵ.



(a) Decomposition of a component into *blocks*. The orange nodes represent the *big* terminals in the component. The black nodes represent the *root* and the *exit* vertices of the component (defined in Lemma 6). The gray nodes are the branching vertices in the subtree spanning the orange and the black nodes. Splitting the component at the orange, the black, and the gray nodes results in a set of *blocks*, represented by green triangles. Each block has a *root* vertex and at most one *exit* vertex. See Section 4.1.



(b) Decomposition of a block into *clusters*. The green triangle represents a block. Each blue triangle represents a *cluster*. Each cluster has a *root* vertex and at most one *exit* vertex. A cluster is *passing* if it has an exit vertex, and is *ending* otherwise. Each passing cluster has a *spine* (dashed). See Section 4.2.



(c) Decomposition of a passing cluster into *cells*. The blue triangle represents a passing cluster. Removing the thick edges from the cluster results in a set of at most $1/\epsilon$ *cells*. Each red triangle represents a cell. Each of those cells has a *root* vertex, an *exit* vertex, and a *spine* (dashed). See Section 4.3.

Figure 2 Three-level decomposition of a component.

2.2.1 Multi-Level Decomposition (Section 4)

We partition each component into $O_\epsilon(1)$ parts using a *multi-level decomposition*.

In the first level, the component is decomposed into $O_\epsilon(1)$ *blocks* so that all terminals strictly inside a block are *small* (we distinguish *big* and *small* terminals depending on their demands). See Figure 2a and Section 4.1.

In the second level, each block is decomposed into $O_\epsilon(1)$ *clusters* so that the overall demand of each cluster is roughly an ϵ fraction of the demand of a component. See Figure 2b and Section 4.2.

In the third level, each cluster is decomposed into $O_\epsilon(1)$ *cells* so that the *spine* of each cell is roughly an ϵ fraction of the *spine* of the cluster, where the *spine* of a cell (resp. a cluster) is the path traversing that cell (resp. that cluster). See Figure 2c and Section 4.3.

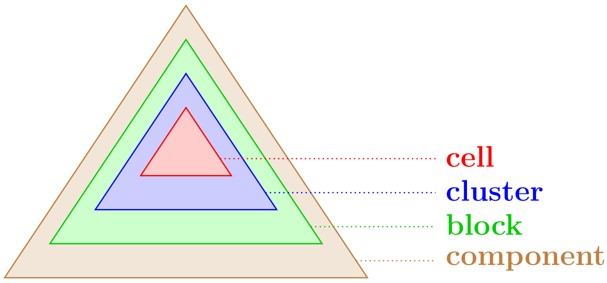


Figure 3 Relation of the multiple levels in the decomposition.

Comparison with the decomposition in [21]

The distinction between big and small terminals plays an important role in UCVRP. This distinction does not exist in the equal demand setting in [21]. In the current paper, the decomposition into blocks is new and enables us to deal with big and small terminals separately; the decomposition into clusters is similar to the decomposition in [21]; the decomposition into cells is a main novelty (see the usage of cells in Section 2.2.2).

2.2.2 Simplifying the Local Solution (Section 5)

The main technical contribution in this paper is the Local Theorem (Theorem 13), which simplifies a local solution inside a component so that, *in each cell, a single subtour visits all small terminals*, while increasing the cost of the local solution by at most a multiplicative factor $1.5 + O(\epsilon)$. The Local Theorem builds upon techniques from [5, 21] together with substantial new ideas.

A first attempt is to reassign all small terminals of each cluster to a single subtour. However, there are two obstacles. First, in order to maintain the connectivity of the resulting subtours, we need to pay for an extra copy of the spines of the clusters, which is expensive. Secondly, using a lemma of Becker and Paul [5], the resulting subtours exceed their capacities slightly. To reduce the demands of the subtours exceeding capacities, an extra cost of only an ϵ fraction of the solution cost is sufficient in the equal demand setting [21], but this is no longer achievable in the arbitrary demand setting.

To overcome those obstacles, we decompose each cluster into *cells* and we reassign all small terminals of each cell to a single subtour. In the analysis, we introduce the technical concept of *threshold cells* (Figure 4a), and we ensure that each cluster contains *at most* *one* threshold cell. In order to maintain the connectivity of the resulting subtours, we only need to pay for an extra copy of *the spines of the threshold cells* (Figure 4b), whose cost is negligible.

To reduce the demand of each resulting subtour exceeding capacity, we select some cells from that subtour, and we remove all pieces in that subtour belonging to those cells. We show that each removed piece is connected to the root through at least *two* subtours in the solution (Lemma 20). That property is a main technical novelty in this paper. It enables us to reconnect all removed pieces with an extra cost of at most *half* of the solution cost (Lemma 21), hence an approximation ratio of $1.5 + O(\epsilon)$.

2.3 Postprocessing

We modify the tree of components using the techniques in [21] so that the new tree has only $O_\epsilon(1)$ levels of components. Consider a near-optimal solution in the new tree. We apply the Local Theorem (Theorem 13) to simplify the local solutions in all components. Then we combine the simplified local solutions into a global solution. The combination requires particular care to deal with the additional subtour in each component created in the Local Theorem.

Next, we apply the *adaptive rounding* technique to the resulting global solution. The adaptive rounding technique for capacitated vehicle routing was first used by Jayaprakash and Salavatipour [18] in their design of a QPTAS in the equal demand setting. This technique enables us reduce the number of subtour demands in each subtree to a constant $O_\epsilon(1)$.

Finally, we design a polynomial time dynamic program to compute the best solution that satisfies the structural constraints established previously. The computed solution is a $(1.5 + O(\epsilon))$-approximation.

This completes the proof of Theorem 1. See the full version of the paper for more details.

▶ Remark 2. When the overall cost of all edges in the tree is fixed, letting W denote this cost, it is possible to adapt our analysis to obtain an *asymptotic polynomial time approximation scheme*. To that end, we observe that in the proof of the Local Theorem (Theorem 13), the extra cost to connect all removed pieces in a component is at most twice the overall cost of all edges in that component, so the overall extra cost over all components is at most $2W$. Thus the cost of the computed solution is at most $1 + O(\epsilon)$ times the optimal cost plus $2W$.

3 Preliminaries

3.1 Formal Problem Description and Notations

Let T be a rooted tree (V, E) with edge weights $w(u, v) \geq 0$ for all $(u, v) \in E$. Let n denote the number of vertices in V. The *cost* of a tour (resp. a subtour) t, denoted by $\text{cost}(t)$, is the overall weight of the edges on t. For a set S of tours (resp. subtours), the *cost* of S, denoted by $\text{cost}(S)$, is $\sum_{t \in S} \text{cost}(t)$.

▶ **Definition 3** (UCVRP on trees). *An instance of the* unsplittable capacitated vehicle routing problem (UCVRP) *on* trees *consists of*

- *an edge weighted* tree $T = (V, E)$ *with* root $r \in V$ *representing the* depot*,*
- *a set* $V' \subseteq V$ *of* terminals*,*
- *for each terminal* $v \in V'$*, a* demand *of* v*, denoted by* $\text{demand}(v)$*, which belongs to* $(0, 1]$*.*

A feasible solution is a set of tours *such that*

- *each tour starts and ends at* r*,*

- *the demand of each terminal is covered by a* single *tour, i.e., the demand cannot be split,*
- *the total demand of the terminals covered by each tour does not exceed the* capacity *of 1.*

The goal is to find a feasible solution of minimum cost.

For any two vertices $u, v \in V$, let $\text{dist}(u, v)$ denote the distance between u and v in the tree T.

We say that a tour (resp. a subtour) *visits* a terminal if it covers the demand of that terminal. For technical reasons, we allow *dummy* terminals of appropriate demands to be included. The *demand* of a tour (resp. a subtour) t, denoted by $\text{demand}(t)$, is defined to be the total demand of all terminals (including dummy terminals) visited by t.

3.2 Reduction to Instances of Bounded Distances

▶ **Definition 4** (bounded distances, Definition 2.1 in [21]). *Let $D_{\min}$ (resp. $D_{\max}$) denote the minimum (resp. maximum) distance between the depot and any terminal in the tree T. We say that T has* bounded distances *if $D_{\max} < (1/\epsilon)^{(1/\epsilon)-1} \cdot D_{\min}$.*

The next theorem (Theorem 5) enables us to assume without loss of generality that the tree T has bounded distances.

▶ **Theorem 5** (Theorem 2.3 and Section 9 in [21]). *For any $\rho \geq 1$, if there is a polynomial time ρ-approximation algorithm for the UCVRP on trees with* bounded distances*, then there is a polynomial time $(1 + 5\epsilon)\rho$-approximation algorithm for the UCVRP on trees with* general distances.

3.3 Decomposition Into Components

The next lemma decomposes the tree T into *components*.

▶ **Lemma 6** (Lemma 4.2 in [21]). *Let $\Gamma = 12/\epsilon$. There is a polynomial time algorithm to compute a partition of the edges of the tree T into a set $\mathcal{C}$ of* components *(see Figure 1), such that all of the following properties are satisfied:*

1. *Every component $c \in \mathcal{C}$ is a connected subgraph of T; the* root *vertex of the component c, denoted by r_c, is the vertex in c that is closest to the depot.*
2. *A component c shares vertices with other components at vertex r_c and possibly at one other vertex, called the* exit *vertex of the component c and denoted by e_c. We say that c is an* internal *component if c has an exit vertex, and is a* leaf *component otherwise.*
3. *The total demand of the terminals in each component $c \in \mathcal{C}$ is at most 2Γ.*
4. *The number of components in $\mathcal{C}$ is at most $\max\{1, 3 \cdot \text{demand}(T)/\Gamma\}$, where $\text{demand}(T)$ denotes the total demand of the terminals in the tree T.*

▶ **Definition 7** (Definition 4.4 in [21]). *Let $c \in \mathcal{C}$ be any component. A* subtour in component c *is a path t that starts and ends at the root r_c of component c, and such that every vertex on t is in component c. We say that a subtour t is a* passing subtour *if c has an exit vertex and that vertex belongs to t, and is an* ending subtour *otherwise.*

4 Multi-Level Decomposition in a Component

Let $c \in \mathcal{C}$ be any component. We partition c using a *multi-level decomposition*: first, c is decomposed into *blocks* (Section 4.1); next, each block is decomposed into *clusters* (Section 4.2); and finally, each cluster is decomposed into *cells* (Section 4.3).

We introduce some notations. Let z denote any block (resp. any cluster or any cell). Then z has a *root* vertex and at most one *exit* vertex. We say that a terminal v is *strictly* inside z if v belongs to z and v is different from the root vertex and the exit vertex of z. The *demand* of z is defined as the total demand of all terminals *strictly* inside z. If z has no exit vertex, then z is called *ending*; otherwise z is called *passing*, and the path between the root vertex and the exit vertex of z is called the *spine* of z.

We distinguish *big* and *small* terminals depending on their demands.

▶ **Definition 8** (big and small terminals). *Let* $\alpha = \epsilon^{(1/\epsilon)+1}$. *Let* $\Gamma' = \epsilon \cdot \alpha/\Gamma$, *where* Γ *is defined in Lemma 6. We say that a terminal* v *is* big *if* $\text{demand}(v) > \Gamma'$ *and* small *otherwise.*

4.1 Decomposition of a Component Into Blocks (Figure 2a)

Let c be a component. Let $U \subseteq V$ denote the set of vertices consisting of the big terminals in c, the *root* vertex of c, and possibly the *exit* vertex of c if c is an *internal* component (see Lemma 6 for definitions). Let T_U denote the subtree of c spanning the vertices in U. We say that a vertex in T_U is a *key* vertex if either it belongs to U or it has two children in T_U. We define a *block* to be a maximally connected subgraph of component c in which any key vertex has degree 1; in other words, blocks are obtained by splitting the component at the key vertices. Note that any terminal strictly inside a block is small. The blocks form a partition of the edges of component c.

4.2 Decomposition of a Block Into Clusters (Figure 2b)

As an adaptation from Lemma 6, we decompose a block into clusters in Lemma 9.

▶ **Lemma 9.** *Let* b *be any block. There is a polynomial time algorithm to compute a partition of the edges of the block* b *into a set of* clusters*, such that all of the following properties are satisfied:*

1. *Every cluster* x *is a connected subgraph of* b*; the* root *vertex of the cluster* x*, denoted by* r_x*, is the vertex in* x *that is closest to the depot.*
2. *A cluster* x *shares vertices with other clusters at vertex* r_x *and possibly at one other vertex, called the* exit *vertex of the cluster* x *and denoted by* e_x*. If block* b *has an exit vertex* e_b*, then there is a cluster* x *in* b *such that* $e_x = e_b$*.*
3. *The demand of each cluster in* b *is at most* $2\Gamma'$*.*
4. *The number of clusters in* b *is at most* $3 \cdot (\text{demand}(b)/\Gamma' + 1)$*.*

4.3 Decomposition of a Cluster Into Cells (Figure 2c)

Let x be any cluster.

Case 1: x **is an ending cluster.** The decomposition of x consists of a single *cell*, which is the entire cluster x.

Case 2: x **is a passing cluster.** Let ℓ denote the cost of the spine of cluster x. If $\ell = 0$, the decomposition of x consists of a single *cell*, which is the entire cluster x. Next, we assume that $\ell > 0$. For each integer $i \in [1, (1/\epsilon) - 1]$, there exists a unique edge (u, v) on the spine of cluster x satisfying $\min(\text{dist}(r_x, u), \text{dist}(r_x, v)) \leq i \cdot \epsilon \cdot \ell < \max(\text{dist}(r_x, u), \text{dist}(r_x, v))$; let e_i denote that edge. Removing the edges $e_1, e_2, \ldots, e_{(1/\epsilon)-1}$ from cluster x results in at most $1/\epsilon$ connected subgraphs; each subgraph is called a *cell*. Observe that those cells form a partition of the vertices of cluster x.

The (unique) cell inside an ending cluster is an ending cell, and any cell inside a passing cluster is a passing cell. Fact 10 follows directly from the construction.

▶ **Fact 10.** *Let x be a passing cluster. The cost of the spine of any cell in x is at most an ϵ fraction of the cost of the spine of x.*

▶ **Fact 11.** *In any component c, the number of cells and the number of big terminals are both $O_\epsilon(1)$.*

Proof. By Lemma 6, the total demand of the terminals in component c is at most 2Γ. Since the demand of a big terminal is at least Γ', there are at most $2\Gamma/\Gamma' = O_\epsilon(1)$ big terminals in c.

From the construction in Section 4.1, the set U consists of at most $2 + 2\Gamma/\Gamma'$ vertices. Since each vertex in c has at most two children, the number of blocks in c is at most $2|U| \leq 4 + 4\Gamma/\Gamma'$. From the construction in Section 4.2, each block b is partitioned into at most $3 \cdot (\text{demand}(b)/\Gamma' + 1)$ clusters, where demand(b) is at most the total demand of the terminals in component c, which is at most 2Γ. From the construction in Section 4.3, each cluster is partitioned into at most $1/\epsilon$ cells. So the number of cells in c is at most $(4 + 4\Gamma/\Gamma') \cdot (3 \cdot (2\Gamma/\Gamma' + 1)) \cdot (1/\epsilon) = O_\epsilon(1)$. ◀

▶ **Definition 12** (Adaptation from Definition 7). *A* subtour in a cluster (resp. cell) *is a path t that starts and ends at the root of that cluster (resp. cell), and such that every vertex on t is in that cluster (resp. cell). We say that a subtour t is a* passing subtour *if that cluster (resp. cell) has an exit vertex and that vertex belongs to t, and is an* ending subtour *otherwise. The* spine subtour *in a passing cluster (resp. passing cell) consists of the spine of that cluster (resp. cell) in both directions.*

5 Simplifying the Local Solution

In this section, we prove the Local Theorem (Theorem 13).

▶ **Theorem 13** (Local Theorem). *Let c be any component. Let S_c denote a set of at most $(2\Gamma/\alpha) + 1$ subtours in component c visiting all terminals in c. Then there exists a set S_c^* of subtours in component c visiting all terminals in c, such that all of the following properties hold:*

1. *For each cell in c, a single subtour in S_c^* visits all small terminals in that cell;*
2. *S_c^* contains one particular subtour $\bar{t}$ of demand at most 1, and the subtours in $S_c^* \setminus \{\bar{t}\}$ are in one-to-one correspondence with the subtours in S_c, such that for every subtour t in S_c and its corresponding subtour t^* in $S_c^* \setminus \{\bar{t}\}$, the demand of t^* is at most the demand of t, and in addition, if t is a passing subtour in c, then t^* is also a passing subtour in c;*
3. *The cost of S_c^* is at most $1.5 + 2\epsilon$ times the cost of S_c.*

▶ Remark 14. Note that the cost to connect the newly generated subtour $\bar{t}$ to the depot is negligible thanks to the properties of the components.

5.1 Construction of S_c^*

The construction of S_c^* starts from S_c and proceeds in 5 steps. In particular, Step 2 uses a new concept of *threshold cells* and is the main novelty in the construction.

The following lemma due to Becker and Paul [5] will be used in Step 1 and Step 3.

▶ **Lemma 15** (Assignment Lemma, Lemma 1 in [5]). *Let $G = (V[G], E[G])$ be an edge-weighted bipartite graph with vertex set $V[G] = A \uplus B$ and edge set $E[G] \subseteq A \times B$, such that each edge $(a, b) \in E[G]$ has a weight $w(a, b) \geq 0$. For each vertex $b \in B$, let $N(b)$ denote the set of vertices $a \in A$ such that $(a, b) \in E[G]$. We assume that $N(b) \neq \emptyset$ and the weight $w(b)$ of the vertex b satisfies $0 \leq w(b) \leq \sum_{a \in N(b)} w(a, b)$. Then there exists a function $f : B \to A$ such that each vertex $b \in B$ is assigned to a vertex $a \in N(b)$ and, for each vertex $a \in A$, we have*

$$\sum_{b \in B \mid f(b)=a} w(b) - \sum_{b \in B \mid (a,b) \in E[G]} w(a, b) \leq \max_{b \in B} \{w(b)\}.$$

Step 1: Combining ending subtours within each cluster

Let A_0 denote S_c. We define a weighted bipartite graph G in which the vertices in one part represent the subtours in A_0 and the vertices in the other part represent the clusters in c.[6] There is an edge in G between a subtour $a \in A_0$ and a cluster x in c if and only if a contains an ending subtour t in x; the weight of the edge is defined to be demand(t). For each cluster x in c, we define the weight of x in G to be the sum of the weights of its incident edges in G. We apply the Assignment Lemma (Lemma 15) to the graph G (deprived of the vertices of degree 0) and obtain a function f that maps each cluster x in c to some subtour $a \in A_0$ such that (a, x) is an edge in G.

We construct a set of subtours A_1 as follows: for every cluster x in c and for every subtour $a \in A_0$ containing an ending subtour t in x, the subtour t is removed from a and added to the subtour $f(x)$. Observe that each resulting subtour in A_1 is connected. From the construction, *for each cluster x, at most one subtour in A_1 has an ending subtour in x.* In particular, for any *ending cell*, which is equivalent to an ending cluster, a single subtour in A_1 visits all small terminals in that cell.

Step 2: Extending ending subtours within threshold cells

Let x be any passing cluster in c such that there is a subtour in A_1 containing an ending subtour in x. From Step 1 of the construction, such a subtour in A_1 is unique; let t_e denote the corresponding ending subtour in x.

We define the **threshold cell** of cluster x to be the deepest cell in x containing vertices of t_e. See Figure 4a.

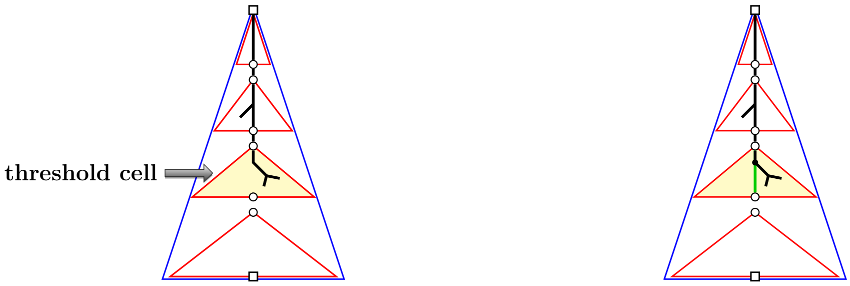
Then we add to t_e the part of the *spine subtour in the threshold cell of x* that does not belong to t_e, resulting in a subtour $\tilde{t}_e$; see Figure 4b.

Let A_2 denote the resulting set of subtours in c after the extension within all threshold cells. From the construction, *for each passing cell s, all subtours in s that are contained in A_2 are passing subtours in s.*

Step 3: Combining passing subtours within each passing cell

We define a weighted bipartite graph G' in which the vertices in one part represent the subtours in A_2 and the vertices in the other part represent the passing cells in c.[7] There is an edge in G' between a subtour $a \in A_2$ and a passing cell s in c if and only if a contains a non-spine passing subtour t in s; the weight of the edge is defined to be the total demand of the small terminals on t. For each passing cell s in c, we define the weight of s in G' to

[6] With a slight abuse, we identify a vertex in G with either a subtour in A_0 or a cluster in c.
[7] With a slight abuse, we identify a vertex in G' with either a subtour in A_2 or a passing cell in c.



(a) Subtour before extension. **(b)** Subtour after extension.

Figure 4 The threshold cell and the extension of an ending subtour. The outermost triangle in blue represents a cluster x. In Figure 4a, the black segments represent the ending subtour t_e in x. The *threshold cell* of cluster x is the *deepest* cell visited by t_e and is represented by the yellow triangle. In Figure 4b, subtour t_e is extended within the threshold cell: the green segment represents the part of the *spine subtour of the threshold cell* that is added to t_e, resulting in a subtour $\tilde{t}_e$.

be the sum of the weights of its incident edges in G'. We apply the Assignment Lemma (Lemma 15) to the graph G' (deprived of the vertices of degree 0) and obtain a function f' that maps each passing cell s in c to some subtour $a \in A_2$ such that (a, s) is an edge in G'.

We construct a set of subtours A_3 as follows: for every passing cell s in c and for every subtour $a \in A_2$ containing a non-spine passing subtour t in s, the subtour t is removed from a except for the spine subtour of s; the removed part is added to the subtour $f'(s)$. Observe that each resulting subtour in A_3 is connected. From the construction, *for each passing cell s, a single subtour in A_3 visits all small terminals in s.*

Step 4: Correcting subtour capacities

For each subtour t_3 in A_3, let t_0 denote the corresponding subtour in A_0. As soon as the demand of t_3 is greater than the demand of t_0, we repeatedly modify t_3 as follows: find a terminal v that is *visited by t_3 but not visited by t_0*; let s denote the cell containing v and let t_s denote the subtour of t_3 in cell s; if s is an ending cell, then remove t_s from t_3; and if s is a passing cell, then remove t_s from t_3 except for the spine subtour of s.

Let A_4 denote the resulting set of modified subtours. Observe that each subtour in A_4 is connected. From the construction, *the demand of each subtour in A_4 is at most the demand of the corresponding subtour in A_0.* Note that the big terminals in each subtour in A_4 are the same as the big terminals in the corresponding subtour in A_0.[8]

Let $\mathcal{R}$ denote the set of the removed pieces. We claim that the total demand of the pieces in $\mathcal{R}$ is at most 1 (Lemma 22).

Step 5: Creating an additional subtour

We connect all pieces in $\mathcal{R}$ by a single subtour $\bar{t}$, which is the minimal subtour in component c that connects all pieces in $\mathcal{R}$ to the root of component c.

Finally, let S_c^* denote $A_4 \cup \{\bar{t}\}$.

[8] Any big terminal cannot be removed, since it is the exit vertex of some cell, thus belongs to the spine of that cell.

5.2 Analysis on the Cost of S_c^*

From the construction of S_c^*, we observe that the cost of S_c^* equals the cost of S_c plus the extra costs in Step 2 and in Step 5 of the construction, denoted by W_2 and W_5, respectively.

To analyze the extra costs, first, in a preliminary lemma (Lemma 16), we bound the overall cost of the spines of the threshold cells. Lemma 16 will be used to analyze both W_2 (Corollary 17) and W_5 (Lemma 21).

▶ **Lemma 16.** *The overall cost of the spines of all threshold cells in the component c is at most $(\epsilon/2) \cdot \mathrm{cost}(S_c)$.*

Proof. Consider any threshold cell s. Let x be the passing cluster that contains s. By Fact 10, the cost of the spine of cell s is at most an ϵ fraction of the cost of the spine of x. Since x is a passing cluster, at least one subtour in S_c contains a passing subtour in x; let t_x denote that passing subtour in x. Observe that t_x contains each edge of the spine of cluster x in both directions (Definition 12), so the cost of the spine of x is at most $\mathrm{cost}(t_x)/2$. Thus the cost of the spine of s is at most $(\epsilon/2) \cdot \mathrm{cost}(t_x)$. We *charge* the cost of the spine of s to t_x.

From the construction, each cluster contains at most one threshold cell. Thus the costs of the spines of all threshold cells are charged to disjoint parts of S_c. The claim follows. ◀

Observe that the extra cost in Step 2 of the construction is at most the overall cost of the spine subtours in all threshold cells in the component c, which equals twice the overall cost of the spines of those cells by Definition 12.

▶ **Corollary 17.** *The extra cost W_2 in Step 2 of the construction is at most $\epsilon \cdot \mathrm{cost}(S_c)$.*

Next, we bound the extra cost in Step 5 of the construction.

▶ **Fact 18.** *Let t denote any subtour in S_c. Let x denote any cluster in c. Let r_c and r_x denote the root vertices of component c and of cluster x, respectively; let e_x denote the exit vertex of cluster x. If the r_c-to-r_x path (resp. the r_c-to-e_x path) belongs to t, then that path belongs to the corresponding subtour of t throughout the construction in Section 5.1.*

▶ **Definition 19** (nice edges). *We say that an edge e in component c is* nice *if e belongs to* at least two *subtours in A_2.*

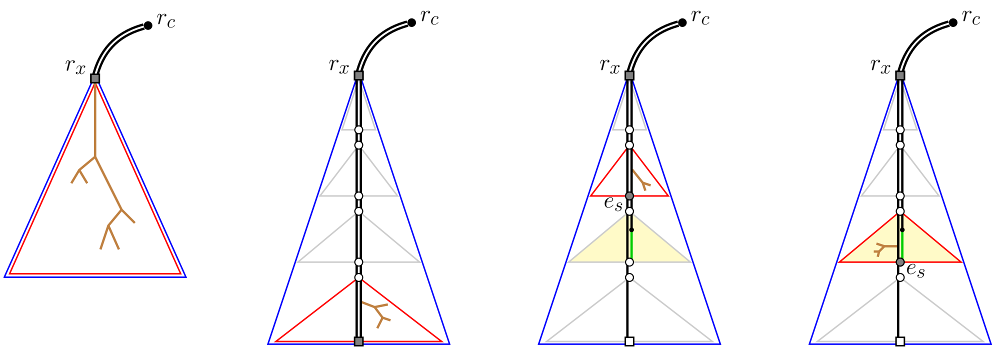
The next Lemma (Lemma 20) is the main novelty in the analysis.

▶ **Lemma 20.** *Any piece in $\mathcal{R}$ is connected to the root r_c of component c through nice edges in c.*

Proof. Consider any piece $q \in \mathcal{R}$. Let s be the cell containing q. Let x be the cluster containing q. See Figure 5. Let r_s and r_x denote the root vertices of cell s and of cluster x, respectively. Observe that the terminals in x are visited by at least two subtours in S_c. This is because, if all terminals in cluster x are visited by a single subtour in S_c, then those terminals belong to the corresponding subtour throughout the construction, thus none of those terminals belongs to a piece in $\mathcal{R}$, contradiction. Thus the r_c-to-r_x path belongs to at least two subtours in S_c. By Fact 18, the r_c-to-r_x path belongs to at least two subtours in A_2, thus every edge on the r_c-to-r_x path is nice. It suffices to show the following Claim:

Piece q is connected to vertex r_x through nice edges in c. (*)

There are two cases:



(a) Case 1. **(b)** Subcase 2(i). **(c)** Subsubcase 2(ii)(α). **(d)** Subsubcase 2(ii)(β).

Figure 5 Illustrations for the different cases in the proof of Lemma 20. A piece $q \in \mathcal{R}$ is in brown. The cell s containing that piece is represented by the triangle in red; the cluster x containing that piece is represented by the outermost triangle in blue. The black node r_c is the root of component c. In Figure 5a, x is an ending cluster. In Figure 5b, x is a passing cluster, and the solution S_c contains two passing subtours in x. In Figures 5c and 5d, x is a passing cluster, and the solution S_c contains a unique passing subtour in x; the yellow triangle represents the threshold cell of x. In the case when q belongs to the threshold cell (Figure 5d), q is connected to r_c through at least two subtours, thanks to the extension of the ending subtour within the threshold cell.

Case 1: x is an ending cluster. See Figure 5a. From the decomposition in Section 4.3, s is an ending cell and s equals x. Piece q is an ending subtour in x and in particular contains r_x. Claim (*) follows trivially.

Case 2: x is a passing cluster. Let e_s and e_x denote the exit vertices of cell s and of cluster x, respectively. Observe that at least one subtour in S_c contains a passing subtour in x. There are two subcases.

Subcase 2(i): At least two subtours in S_c contain passing subtours in x.
See Figure 5b. Then the r_c-to-e_x path belongs to at least two subtours in S_c. By Fact 18, the r_c-to-e_x path belongs to at least two subtours in A_2, thus each edge on the spine of x is nice. Since piece q contains a vertex on the spine of x, Claim (*) follows.

Subcase 2(ii): Exactly one subtour in S_c contains a passing subtour in x.
See Figures 5c and 5d. Let t_p denote that passing subtour in x. As observed previously, at least two subtours in S_c visit terminals in x, so there must be at least one subtour in S_c that contains an ending subtour in x. Let $t_e^1, \dots, t_e^m$ (for some $m \geq 1$) denote the ending subtours in x contained in the subtours in S_c. In Step 1 of the construction, the m ending subtours are combined into a single ending subtour, denoted by t_e (recall that the threshold cell of x is defined with respect to t_e); and in Step 2 of the construction, subtour t_e is extended to a subtour $\tilde{t}_e$ (Figure 4). Note that the passing subtour t_p remains unchanged in Steps 1 and 2 of the construction. We observe that cell s is either above or equal to the threshold cell of x. This is because, if cell s is below the threshold cell of x, then all terminals in s are visited by a single subtour in S_c, i.e., the subtour t_p, so those terminals belong to the corresponding subtour of t_p throughout the construction, thus none of those terminals belongs to a piece in $\mathcal{R}$, contradiction. Hence the following two subsubcases.

Subsubcase 2(ii)(α): s is above the threshold cell of x. See Figure 5c. Each edge on the r_x-to-e_s path belongs to both subtours t_p and t_e, hence is nice. Since q contains some vertex on the spine of s, Claim (*) follows.

Subsubcase 2(ii)(β): s equals the threshold cell of x. See Figure 5d. Observe that each edge on the r_x-to-e_s path belongs to $\tilde{t}_e$ due to the extension of the ending subtour t_e within the threshold cell (Step 2 of the construction). Thus each edge on the r_x-to-e_s path belongs to both subtours t_p and $\tilde{t}_e$, hence is nice. Since q contains some vertex on the spine of s, Claim (*) follows. ◀

▶ **Lemma 21.** *The extra cost W_5 in Step 5 of the construction is at most $(0.5+\epsilon)\cdot \text{cost}(S_c)$.*

Proof. Let W_{nice} denote the overall cost of the nice edges in c. We show that $W_5 \leq 2\cdot W_{\text{nice}}$. Let H be the multi-subgraph in c that consists of the pieces in $\mathcal{R}$ and two copies of each nice edge in c (one copy for each direction). Since any piece in $\mathcal{R}$ is connected to the root r_c of component c through nice edges (Lemma 20), H induces a connected subtour in c. So $W_5 \leq 2\cdot W_{\text{nice}}$.

Next, we analyze W_{nice}. From the construction, any nice edge e in c is of at least one of the two cases:

Case 1: e belongs to at least two subtours in S_c. Then e has at least 4 copies in S_c, since each subtour to which e belongs contains 2 copies of e (one for each direction). Thus the overall cost of the edges e in this case is at most $0.25\cdot \text{cost}(S_c)$.

Case 2: e belongs to the spine of a threshold cell in component c. By Lemma 16, the overall cost of the edges e in this case is at most $(\epsilon/2)\cdot\text{cost}(S_c)$.

Hence the overall cost W_{nice} of the nice edges is at most $(0.25+\epsilon/2)\cdot\text{cost}(S_c)$. Therefore, $W_5 \leq 2\cdot W_{\text{nice}} \leq (0.5+\epsilon)\cdot\text{cost}(S_c)$. ◀

From Corollary 17 and Lemma 21, we conclude that

$$\text{cost}(S_c^*) = \text{cost}(S_c) + W_2 + W_5 \leq (1.5+2\epsilon)\cdot\text{cost}(S_c).$$

Hence the third property of the claim in the Local Theorem (Theorem 13).

5.3 Feasibility

From the construction, S_c^* is a set of subtours in c visiting all terminals in c. The first property of the claim in the Local Theorem (Theorem 13) follows from the construction. The second property of the claim follows from the construction, Fact 18, and the following Lemma 22.

▶ **Lemma 22.** *The total demand of the pieces in $\mathcal{R}$ is at most 1.*

Proof. Observe that the pieces in $\mathcal{R}$ are removed from subtours in A_3. Let t_3 denote any subtour in A_3. Let t_0, t_1, t_2, and t_4 denote the corresponding subtours of t_3 in A_0, A_1, A_2, and A_4, respectively. Let Δ denote the overall demand of the pieces that are removed from t_3 in Step 4 of the construction. Observe that $\Delta = \text{demand}(t_3) - \text{demand}(t_4)$. To bound Δ, first, by Step 1 of the construction and the Assignment Lemma (Lemma 15), the demand of each subtour in A_0 is increased by at most the maximum demand of a cluster. Thus $\text{demand}(t_1) - \text{demand}(t_0)$ is at most the maximum demand of a cluster, which is at most $2\Gamma'$ by the definition of clusters (Section 4.2). By Step 2 of the construction, $\text{demand}(t_2) = \text{demand}(t_1)$. By Step 3 of the construction and the Assignment Lemma (Lemma 15), the demand of each subtour in A_2 is increased by at most the maximum demand of a cell. Thus $\text{demand}(t_3) - \text{demand}(t_2)$ is at most the maximum demand of a cell, which is at most $2\Gamma'$ by the definition of cells (Section 4.3). By Step 4 of the construction, $\text{demand}(t_0) - \text{demand}(t_4)$ is at most the maximum demand of a cell, which is at most $2\Gamma'$. Combining, we have $\Delta = \text{demand}(t_3) - \text{demand}(t_4) \leq 6\Gamma'$.

The number of subtours in A_3 equals the number of subtours in S_c, which is at most $(2\Gamma/\alpha)+1$ by assumption. Thus total demand of the pieces in $\mathcal{R}$ is at most $6\Gamma' \cdot ((2\Gamma/\alpha)+1) < 13\epsilon < 1$, assuming $\epsilon < 1/13$. ◀

This completes the proof of the Local Theorem (Theorem 13).

References

1 Muhammad Al-Salamah. Constrained binary artificial bee colony to minimize the makespan for single machine batch processing with non-identical job sizes. *Applied Soft Computing*, 29:379–385, 2015.

2 Kemal Altinkemer and Bezalel Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

3 Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.

4 Amariah Becker. A tight 4/3 approximation for capacitated vehicle routing in trees. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems*, volume 116, pages 3:1–3:15, 2018.

5 Amariah Becker and Alice Paul. A framework for vehicle routing approximation schemes in trees. In *Workshop on Algorithms and Data Structures*, pages 112–125. Springer, 2019.

6 Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. *Mathematical Programming*, pages 1–47, 2022.

7 Huaping Chen, Bing Du, and George Q. Huang. Scheduling a batch processing machine with non-identical job sizes: a clustering perspective. *International Journal of Production Research*, 49(19):5755–5778, 2011.

8 Jing Chen, He Guo, Xin Han, and Kazuo Iwama. The train delivery problem revisited. In *International Symposium on Algorithms and Computation*, pages 601–611. Springer, 2013.

9 Purushothaman Damodaran, Praveen Kumar Manjeshwar, and Krishnaswami Srihari. Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics*, 103(2):882–891, 2006.

10 George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

11 Aparna Das, Claire Mathieu, and Shay Mozes. The train delivery problem-vehicle routing meets bin packing. In *International Workshop on Approximation and Online Algorithms*, pages 94–105. Springer, 2010.

12 Lionel Dupont and Clarisse Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers & Operations Research*, 29(7):807–819, 2002.

13 Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Improved approximations for capacitated vehicle routing with unsplittable client demands. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 251–261. Springer, 2022.

14 Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

15 Fabrizio Grandoni, Claire Mathieu, and Hang Zhou. Unsplittable Euclidean Capacitated Vehicle Routing: A $(2+\epsilon)$-Approximation Algorithm. In *Innovations in Theoretical Computer Science (ITCS)*, volume 251 of *LIPIcs*, pages 63:1–63:13, 2023.

16 Mordecai Haimovich and Alexander H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

17 Shin-ya Hamaguchi and Naoki Katoh. A capacitated vehicle routing problem on a tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

18 Aditya Jayaprakash and Mohammad R. Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. *ACM Transactions on Algorithms (TALG)*, 19(2), 2023.

19 Ali Husseinzadeh Kashan, Behrooz Karimi, and Fariborz Jolai. Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. *International Journal of Production Research*, 44(12):2337–2360, 2006.

20 Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

21 Claire Mathieu and Hang Zhou. A PTAS for capacitated vehicle routing on trees. *ACM Transactions on Algorithms (TALG)*, 19(2), 2023.

22 Sharif Melouk, Purushothaman Damodaran, and Ping-Yu Chang. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, 87(2):141–147, 2004.

23 İbrahim Muter. Exact algorithms to minimize makespan on single and parallel batch processing machines. *European Journal of Operational Research*, 285(2):470–483, 2020.

24 N. Rafiee Parsa, Behrooz Karimi, and Ali Husseinzadeh Kashan. A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers & Operations Research*, 37(10):1720–1730, 2010.

25 Thomas Rothvoß. The entropy rounding method in approximation algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 356–372. SIAM, 2012.

26 Reha Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *The International Journal of Production Research*, 32(7):1615–1635, 1994.

27 Yuanxiao Wu and Xiwen Lu. Capacitated vehicle routing problem on line with unsplittable demands. *Journal of Combinatorial Optimization*, pages 1–11, 2020.