Gleichmäßige Flächenaufteilung von Polygonen

Sebastian Loder Jan Steffen Jendrny

Januar 2022

In dieser Arbeit wird ein konkretes Problem der Polygonzerlegung, das sogenannte *Problem der verankerten Flächenaufteilung* (eng. *anchored area partition problem*) vorgestellt, welches von Susan Hert und Vladimir Lumelsky [1] veröffentlicht wurde. Die vorliegende Arbeit basiert maßgebend auf dieser Veröffentlichung und beschreibt, wie dieses Problem mittels *sweepline*- und *divide-and-conquer*-Techniken effizient gelöst werden kann. Die Lösung erfolgt zunächst für konvexe Polygone und wird anschließend auf nicht konvexe, nicht einfache Polygone erweitert.

Stichwörter: Polygonzerlegung, Flächenaufteilung, Flächenerkundung, Roboterplanung

1 Einleitung

Die Polygonzerlegung ist eines der zentralen Probleme in der algorithmischen Geometrie und hat viele Anwendungsfälle, wie beispielsweise in der Kartographie, Bildverarbeitung oder in der Computergrafik. In vielen Fällen wird die Polygonzerlegung benötigt, um aus einem beliebigen Polygon eine Menge von Teilpolygonen mit bestimmten Eigenschaften zu berechnen. Als Beispiel einer vielfach verwendeten Polygonzerlegung kann die Triangulation genannt werden, bei welcher ein gegebenes Polygon in eine Menge von Dreiecken zerlegt wird. Für die so berechnete Menge von Dreiecken stehen dann effiziente Algorithmen zur Lösung von Problemen zur Verfügung. Anschließend können die Lösungen der Teilpolygone zu einer Lösung für das Ausgangspolygon zusammengefasst werden.

Bei dem hier vorgestellten Problem der verankerten Flächenaufteilung ist die Anforderung an die resultierenden Teilpolygone nicht durch eine bestimmte Geometrie (z. B. ein Dreieck), sondern durch die Lage und Fläche der Teilpolygone gegeben. Bezüglich der Lage besteht die Anforderung darin, dass ein gegebener Punkt (Standort genannt) auf dem resultierenden Polygon liegen muss. Jeder Standort weist als Eigenschaft eine Flächenanforderung auf, welche durch die Größe des Teilpolygons erfüllt werden soll. Die Flächenanforderung kann je Standort den gleichen Wert aufweisen, kann aber auch unter den Standorten variieren. Somit bezieht sich das hier beschriebene Problem sowohl auf eine gleichmäßige, als auch eine ungleichmäßige Flächenaufteilung. Das beschriebene Problem ist unter anderem durch die Flächenerkundung von Robotern motiviert:

Auf einem Polygon werden n Roboter auf Standorten S_i, $i = 1, ..., n$ positioniert, welche die Aufgabe erhalten, zusammen die gesamte Fläche des Polygons zu erkunden. Hierzu muss jede Position innerhalb des Polygons von einem der n Roboter abgefahren werden. Um die Arbeit unter den Robotern aufzuteilen, ist es sinnvoll, jedem Roboter einen Polygonteil zuzuweisen, der jeweils zu bearbeiten ist. Die Teilpolygone sollen sich nicht überlappen, um ein mehrfaches Überfahren zu vermeiden. Bei der Flächenaufteilung muss berücksichtigt werden, dass der Startpunkt eines jeden Roboters auf dem zugewiesenen Teilpolygon beziehungsweise in diesem liegt. Eine unterschiedliche Leistung der Roboter kann über die Flächenanforderung je Standort berücksichtigt werden.

Zur formalen Beschreibung des Problems sind als Eingangsdaten ein Polygon P sowie eine (nicht leere) Liste von Standorten $S(P)$ gegeben. Für jeden der n Standorte S_i , $i = 1, ..., n$ ist der benö-

```
// Input: Convex polygon CP
Function ConvexDivide(CP)
    if Length(S(CP)) == 1 then return CP
    // Here, the postion of L has to be calculated
    PrL, PlL = Cut(CP, L)    // CP is cut into two pieces PrL and PlL
    ConvexDivide(PrL)    // recursive PrL
    ConvexDivide(PlL)    // recursive PlL
 End ConvexDivide()
```

Listing 1: Die Grundidee hinter dem Algorithmus *ConvexDivide*

tigte Flächenanteil c_i, $i = 1, ..., n$ mit $0 < c_i < 1$ gegeben, sodass $\sum_{i=1}^{n} c_i = 1.0$ gilt. Das Polygon P soll in n, nicht überlappende Polygone zerlegt werden, sodass jeder Standort S_i auf dem Teilpolygon P_i mit Fläche $c_i * Area(P)$ liegt. Aus der Fläche des Polygons P kann für jeden der n Standorte die benötigte Fläche mit $c_i * Area(P)$ bestimmt werden.

In der Praxis werden Flächenaufteilungen beispielsweise bei der Einteilung von Zustellbezirken der Post oder Einsatzgebieten von Rettungskräften verwendet. Die zuvor beschriebene, konkrete Problembeschreibung der verankerten Flächenaufteilung kann in Bezug auf die Roboterplanung beispielsweise auf Saug- oder Mähroboter übertragen werden, welche inzwischen in einigen Haushalten zu finden sind.

Hinweis: Die in dieser Arbeiten genutzten Begriffe und Abkürzungen sind aus der Veröffentlichung von Hert und Lumelsky [1] übernommen und werden in Kapitel 6 erläutert. Die Listings wurden gegenüber der Veröffentlichung [1] zur besseren Lesbarkeit verändert.

2 Aufteilung eines einfachen, konvexen Polygons

2.1 Grundidee

Bei der nachfolgend beschriebenen Lösung des Problems wird ein konvexes Eingabepolygon CP mithilfe von Liniensegmenten schrittweise zerlegt. Jedes Liniensegment L ist hierbei vom Startpunkt L_s zum Endpunkt L_e orientiert, wobei beide Punkte auf dem Rand von CP liegen. Wenn für L_s und L_e eine Position gefunden wurde, erfolgt eine Zerlegung in zwei Teilpolygone. Die bei jeder Teilung entstehende Teilpolygone erhalten entsprechend ihrer Lage zum Liniensegment L die Bezeichnungen P_L^r für das rechts und P_L^l für das links des Liniensegments liegenden Polygons, wobei angenommen wird, dass hierbei die Blickrichtung von L_s nach L_e orientiert ist. Die Liniensegmente (bzw. L_s und L_e) werden so positioniert, dass die Fläche von P_L^r der benötigten Fläche der auf dem Rand von P_L^r liegenden Standorte entspricht (P_L^l analog). Die Zerlegung wird für jedes Teilpolygon P_L^r und P_L^l rekursiv aufgerufen, bis nur noch 1-Standort Polygone vorliegen. Hierbei sei betont, dass aus einer Zerlegung eines konvexen Polygons durch ein Liniensegment immer zwei konvexe Polygone und insbesondere kein nicht konvexes Polygon resultiert.

2.2 Aufteilung eines einfachen, konvexen Polygons

Aus CP entstehende, konvexe Polygone werden mit CP_i notiert. Mit den genannten Überlegungen lässt sich ein rekursiver *divide-and-conquer* - Algorithmus zur Flächenaufteilung eines konvexen Polygons - basierend auf n Standorten - wie in Listing 1 skizzieren.

Bei jedem Aufruf von *ConvexDivide*(CP) wird zunächst geprüft, ob das übergebene Polygon nur noch einen Standort enthält. Falls ja, ist der Zielzustand für dasübergebene Polygon erreicht und es ist keine weitere Flächenaufteilung erforderlich. Falls das Polygon mehrere Standorte enthält, erfolgt eine weitere Aufteilung des Polygons in zwei Teilpolygone P_L^r und P_L^l, welche dann rekursiv mit *ConvexDivide* aufgerufen werden.

Abbildung 1 zeigt ein Beispiel für eine Zerlegung eines 4-Standort-Polygons in 4 Teilpolygone mit jeweils einem Standort. In (a) wird P_L^r mit einer Fläche von 3.4 FE abgetrennt und dem Standort

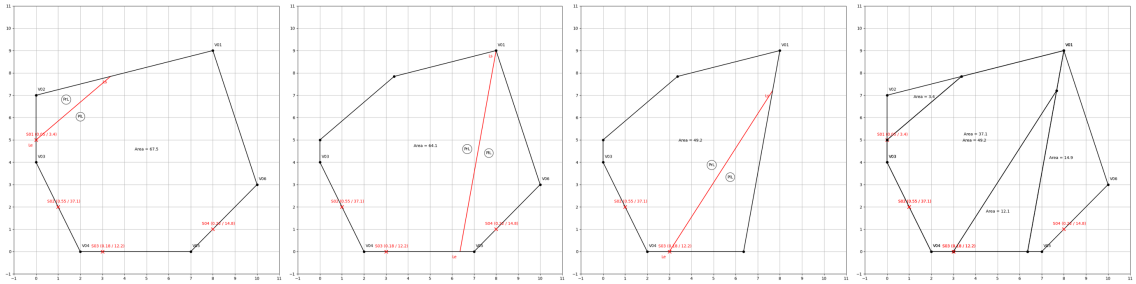

Abbildung 1: Zerlegung eines konvexen Polygons CP in vier konvexe Polygone $CP_1, ..., CP_4$

$S01$ zugeordnet. Die in (b) entstehenden Teilpolygone P_L^r und P_L^l weisen eine Fläche von 49.3 FE beziehungsweise 14.8 FE auf. Letzteres wird dem Standort $S04$ zugeordnet und ist als 1-Standort-Polygon fertig bearbeitet. P_L^r wird in (c) erneut aufgeteilt, sodass die Flächenanforderung von $S02$ und $S03$ jeweils erfüllt werden. (d) zeigt die resultierende Aufteilung mit 4 Teilpolygonen.

2.3 Positionierung der Schnittlinie

Aus vorangegangenem Kapitel bleibt offen, wie genau die Aufteilung eines konvexen Polygons CP in die Polygone P_L^r und P_L^l erfolgt, sodass anschließend $Area(P_L^r) = AreaRequired(S_1, ..., S_i)$ und $Area(P_L^l) = AreaRequired(S_{i+1}, ..., S_n)$ gilt. Konkret ist zu klären, wie Anfangs- und Endpunkt der Schnittlinien positioniert werden (siehe Listing 1, Zeile 4). Zunächst werden L_s und L_e beim Aufruf von *ConvexDivide* wie folgt initialisiert:

- Der Startpunkt L_s der Linie L wird mit den Koordinaten des ersten Punkts der Liste $W()$ initialisiert, wobei dieser nach Definition ein Polygonpunkt (und kein Standort) ist. Es gilt daher $w_1 \in V()$.
- Der Endpunkt L_e wird mit den Koordinaten des ersten Standorts in $W()$ initialisiert und mit w_k notiert, wobei k der Index in $W()$ ist, bei dem der erste Standort liegt. Da die Standorte nach ihrem Vorkommen auf dem Weg von v_1 nach v_l geordnet sind, ist bei einem konvexen Polygon sichergestellt, dass die Standorte $S_2, \ldots, S_n$ alle links der Linie L liegen.

Bei einer Zerlegung mit einer so initialisierten Linie würde $S(P_L^r) = S_1$ und $S(P_L^l) = S_2, ..., S_n$ gelten, wobei S_1 in einer Ecke von P_L^r liegen würde. Je nach Fläche von P_L^r und $AreaRequired(S_1)$ werden folgende Fälle unterschieden:

Fall 1: $Area(P_L^r) > AreaRequired(S_1)$
Nach der Initialisierung der Linie L wird festgestellt, dass die Fläche von P_L^r größer ist als die benötigte Fläche von S_1. In diesem Fall erfolgt eine Verkleinerung von $Area(P_L^r)$ unter Beibehaltung von $S(P_L^r) = S_1$. Dies geschieht, indem L_e als Drehpunkt fungiert und L_s inkrementell gegen den Uhrzeigersinn entlang des Polygons verschoben wird, bis $Area(P_L^r) = AreaRequired(S_1)$ gilt. Zur Verdeutlichung dieser Vorgehensweise sollen folgende Punkte nochmals herausgestellt werden:

- Durch die Initialisierung kann auf dem Weg von w_1 zu w_k kein weiterer Standort liegen, d.h. $AreaRequired(S(P_L^r))$ ist konstant.
- $Area(P_L^r)$ wird mit Verschiebung von L_s stetig kleiner. Bei $L_s = S_1$ gilt $Area(P_L^r) = 0$.
- L_e ist fest, d.h. S_1 ist stets Teil von P_L^r.

Wenn die Bedingung $Area(P_L^r) = AreaRequired(S_1)$ eintritt, erfolgt eine Polygonzerlegung. Für P_L^r erfolgt keine weitere Zerlegung beim Aufruf von *ConvexDivide* (siehe Listing 1, Zeile 3), da nur S_1 auf dessen Rand liegt. Falls auf dem Rand von P_L^l mehr als ein Standort verbleibt, erfolgt eine

erneute Zerlegung beim Aufruf von *ConvexDivide* (P_L^l) (siehe Listing 1, Zeile 7).

Fall 2: $Area(P_L^r) < AreaRequired(S_1)$
Nach der Initialisierung der Linie L wird festgestellt, dass die Fläche von P_L^r kleiner ist als die benötigte Fläche von S_1. In diesem Fall erfolgt eine Vergrößerung von $Area(P_L^r)$ mit dem Ziel, die Flächenanforderung zu erfüllen. Hierbei fungiert L_s als Drehpunkt und L_e wird auf den nächsten in $W()$ vorkommenden Polygonpunkt oder Standort w_{k+1} gesetzt. Die Anforderung wird erneut geprüft. In diesem Schritt wird L_e von Polygonpunkt zu Polygonpunkt verschoben. Eine inkrementelle Verschiebung entlang des Polygons erfolgt dann unter Fall 2.1 beziehungsweise 2.2.

Hierbei kann es nun vorkommen, dass L_e auf die Koordinaten eines Punktes w_j in $W()$ gesetzt wird, welcher ein Standort $\neq S_1$ ist. Dieser Standort wird dann beim nächsten Vorrücken (also bei w_{j+1}) zur benötigten Fläche von P_L^r hinzugenommen. Bei Fall 2 kann $AreaRequired(P_L^r)$ demnach ansteigen, sodass ein Vorücken von L_e zwar zu einer größeren Fläche von P_L^r, nicht aber unbedingt zu einem günstigeren Verhältnis aus $Area(P_L^r)/AreaRequired(S(P_L^r))$ führt.
L_e wird so oft verschoben, bis eine der folgenden Bedingungen eintritt:

- $Area(P_L^r) > AreaRequired(S(P_L^r))$
- $L_e = S_n$

Je nachdem, wie weit L_e vorrückt und wie groß die Fläche von P_L^r im Vergleich zur Flächenanforderung von $S(P_L^r)$ ist, werden nun weiter zwei Fälle unterschieden:

Fall 2.1: $L_e = S_n$ **und** $Area(P_L^r) > AreaRequired(S(P_L^r))$
In diesem Fall wird der Endpunkt L_e inkrementell im Uhrzeigersinn entlang des Polygons bewegt, bis $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt. Hinweis: Angenommen die Ausgangsposition von L_e ist w_j, dann muss es zwischen w_j und w_{j-1} eine Position geben, bei der $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt, da beim Vorrücken $Area(P_L^r)$ bei w_{j-1} zu klein und bei w_j zu groß war. Dieser Zwischenpunkt wird durch Interpolation gefunden.

Fall 2.2: $L_e = S_n$ **und** $Area(P_L^r) < AreaRequired(S(P_L^r))$
In diesem Fall wird der Anfangspunkt L_s inkrementell im Uhrzeigersinn entlang des Polygons bewegt, bis $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt.
Diese Vorgehensweise entspricht im Wesentlichen Fall 1, wobei L_s (initialisiert mit w_1) nun nicht gegen den Uhrzeigersinn zum ersten Standort S_1, sondern im Uhrzeigersinn zum letzten Standort S_n bewegt wird. Vergleiche auch Abbildung 2, Fall (c) und (i).

In der Abbildung 2 werden die unterschiedlichen Fälle gezeigt, die durch angepasste Flächenanforderungen von S01/S02 hervorgerufen werden (Fall 1: 0.20/0.80, Fall 2.1: 0.70/0.30, Fall 2.2: 0.95/0.05). Die Serie (a) - (c) bezieht sich hierbei auf Fall 1. Nach der Initialisierung (a) liegt bereits $Area(P_L^r) > AreaRequired(S(P_L^r))$ vor, sodass L_e nicht gegen den Uhrzeigersinn bewegt werden muss und fest bleibt, siehe (b). In (c) wird L_s dann inkrementell gegen den Uhrzeigersinn bewegt, bis $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt. Fall 2.1 wird in Serie (d) - (f) dargestellt, bei welchem nach der Initialisierung (d) zunächst $Area(P_L^r) < AreaRequired(S(P_L^r))$ gilt. Nach einer schrittweisen Verschiebung von L_e gegen den Uhrzeigersinn (e) folgt letztendlich eine Interpolation in (f), sodass $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt. Die Serie (g) - (i) bezieht sich auf Fall 2.2, wobei L_e in (h) zuerst bis zum letzten Standort wandert, bevor anschließend eine inkrementelle Verschiebung von L_s in (i) erfolgt, sodass $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt. Hierbei lässt sich gut die Analogie zwischen Fall (c) und (i) erkennen.
In Listing 2 wurde der Algorithmus *ConvexDivide* nun durch die beschriebenen Fällen erweitert.
Abbildung 5 zeigt ein Beispiel einer Zerlegung eines nicht konvexen Polygons.

3 Verallgemeinerung: Aufteilung eines nicht einfachen, nicht konvexen Polygons

Es wurde gezeigt, dass ein einfaches, konvexes Polygon rekursiv in n 1-Standort-Polygone aufgeteilt werden kann. Dieses Kapitel dient dazu einen verallgemeinerten Algorithmus zu skizzieren, damit

```
// Input: Convex polygon CP
Function ConvexDivide(CP)
    if Length(S(CP)) == 1 then return CP
    Ls = W(1), Le = W(k)    // k = index of first Site in W
    PrL, PlL = Cut(CP, L)    // partitioning, returns PrL and PlL
    while Area(PrL) < AreaRequired(S(PrL)) and Le != Sn do
        if W(k-1) != S1 and W(k-1) in S then
            S(PrL) = S(PrL) + W(k-1)    // add previous Site to S(PrL)
        k += 1
        Le = W(k)    // move Le to next point in W
        PrL, PlL = Cut(CP, L)
    if Area(PrL) > AreaRequired(S(PrL)) and Le == S1 then
        move Le CCW until Area(PrL) == AreaRequired(S(PrL))
    else if Area(PrL) < AreaRequired(S(PrL))
        if Le != Sn then
            interpolate Le that Area(PrL) == AreaRequired(S(PrL))
        else if Le == Sn then
            move Ls CW until Area(PrL) == AreaRequired(S(PrL))
    PrL, PlL = Cut(CP, L)    // CP is cut into two pieces PrL and PlL
    ConvexDivide(PrL)    // recursive PrL
    ConvexDivide(PlL)    // recursive PlL
End ConvexDivide()
```

Listing 2: Der vollständige Algorithmus *ConvexDivide*

auch für nicht einfache, nicht konvexe Polygone (siehe beispielsweise Abbildung 3) das Problem der verankerten Flächenaufteilung gelöst werden kann.

Bevor dieser Algorithmus vorgestellt wird, soll die Beschreibung der dahinterliegenden Grundidee einen Überblick über die Vorgehensweise verschaffen. Danach werden die vorbereitenden Schritte vorgestellt und die Aufteilung des Polygons erläutert. Ein Beispiel dient anschließend zur Veranschaulichung des vorgestellten Algorithmus und zum Schluss des Kapitels wird der Sonderfall geschildert, bei dem Standorte im Inneren des Polygons liegen.

3.1 Grundidee

In Kapitel 2 wurde bereits erläutert, wie ein einfaches, konvexes Polygon aufgeteilt werden kann. Dieses Vorgehen kann auch bei der Aufteilung nicht konvexer Polygone verwendet werden, muss jedoch in einigen Punkten erweitert werden.
Als Voraussetzung wird angenommen, dass ein nicht einfaches, nicht konvexes Polygon P bereits in konvexe Teilpolygone $CP_1, ..., CP_p$ zerlegt wurde. Im ersten Schritt werden die Teilpolygone mithilfe einer Tiefensuche neu geordnet, um eine feste Bearbeitungsfolge für das weitere Vorgehen zu erhalten. Anschließend werden die Teilpolygone rekursiv aufgeteilt, wie es ähnlich bereits in Kapitel 2 gezeigt wurde. Allerdings können nun Sonderfälle auftreten, die bei der Zerlegung eines einfachen, konvexen Polygons nicht vorkommen können. Einerseits kann CP_i weniger Fläche ausfüllen, als durch $AreaRequired(S(CP_i))$ gefordert ist. In diesem Fall ist CP_i Flächen-unvollständig und muss Flächen von anderen Teilpolygonen übernehmen. Andererseits kann es sein, dass einzelne Teilpolygone keinen Standort enthalten oder weniger Fläche ausfüllen, als durch $AreaRequired(S(CP_i))$ gefordert ist. In diesem Fall ist CP_i Standort-unvollständig und andere Teilpolygone müssen Flächen von CP_i übernehmen.
Die Neuordnung wird innerhalb der Prozedur *OrderPieces* umgesetzt und die Aufteilung inklusive der Sonderfallbehandlung wird durch die beiden Methoden *NonConvexDivide* und *DetachAndAssign* umgesetzt, die sich gegenseitig rekursiv aufrufen, bis ein n-Standort Polygon in n 1-Standort Polygone aufgeteilt wurde.

3.2 Aufteilung in konvexe Teilpolygone

Als Voraussetzung für die gleichmäßige Aufteilung eines nicht einfachen, nicht konvexen Polygons wird angenommen, dass das Polygon bereits in konvexe Teilpolygone aufgeteilt wurde. In verschie-

```
// Input: Nj - Node of the connectivity Graph
Function OderPieces(Nj)
    if Nj has not beeing marked then
        if Nj is a leaf node then
            Mark(Nj)
            Output(CPj)
            for each Nk in Neighbors(Nj)
                OrderPieces(Nk)
        else
            Mark(Nj)
            for each Nk in Neighbors(Nj)
                OrderPieces(Nk)
            Output(CPj)
End OrderPieces(Nj)
```

Listing 3: Der Algorithmus *OrderPieces*

denen Werken werden Möglichkeiten einer solchen Aufteilung vorgestellt. Ein Vorgehen wäre zum Beispiel, eine Triangulation eines Polygons zu erzeugen. In diesem Fall würde jedoch eine hohe Anzahl von Teilpolygonen entstehen. Um Teilpolygone zusammenzufassen, könnten nacheinander Kanten der Triangulation entfernt werden, solange das dadurch entstehende Teilpolygon weiterhin konvex ist [2]. Hieraus wird ersichtlich, dass es verschiedene Möglichkeiten gibt, ein Polygon in konvexe Teilpolygone aufzuteilen. Zum Schluss dieser Arbeit wird besprochen, welche Auswirkungen diese vorbereitenden Schritte auf den Verlauf des vorgestellten Algorithmus haben können.

3.3 Neuordnung der konvexen Teilpolygone

Es kann nun davon ausgegangen werden, dass das Polygon P bereits in konvexe Teilpolygone $CP_1, ..., CP_p$ zerlegt wurde. Die Teilpolygone können willkürlich geordnet sein und die Indizes treffen keine Aussage über die tatsächliche Anordnung im Polygon P. Aus diesem Grund werden die Teilpolygone zuerst neu geordnet, was für den anschließend dargestellten Algorithmus erforderlich ist. Dazu wird ein Verbindungsgraph G gebildet und anhand dessen mittels einer Tiefensuche eine Ordnung erzeugt. Abbildung 4 zeigt ein in konvexe Teilpolygone zerlegtes Polygon und dessen Verbindungsgraph.

Jedes Teilpolygon CP_i wird durch einen Knoten N_i in G abgebildet. Für jeden Nachbarn $CP_k (i \neq k)$ des Teilpolygons CP_i wird eine Kante zum jeweils korrespondierenden Knoten N_k eingefügt. Wir definieren einen Knoten N_i in G als Blatt, wenn N entweder nur einen Nachbarn hat oder alle Nachbarn von N_i als besucht markiert wurden. Die Prozedur *OrderPieces* beschreibt nun die Neuordnung der Teilpolygone. *OrderPieces* wird mit einem beliebigen Knoten N_i von G initial aufgerufen. Zuerst wird geprüft, ob N_i bereits markiert wurde. Ist dies der Fall, kann der Aufruf zurückkehren. Falls N_i noch nicht markiert wurde, wird geprüft, ob N_i nach obiger Definition ein Blatt ist. Falls N_i kein Blatt ist, dann wird der Knoten markiert und für alle Nachbarn N_k von N_i rekursiv *OrderPieces* aufgerufen. Nach dem Rücksprung der Aufrufe aller Nachbarn von CP_i wird CP_i ausgegeben. Falls N_i ein Blatt ist, dann wird N_i markiert und CP_i ausgegeben. Anschließend wird für alle Nachbarn N_k von N_i rekursiv *OrderPieces* aufgerufen. Die neue Ordnung der CP_i ist nun die Reihenfolge, in der die Teilpolygone ausgegeben wurden.

3.4 Aufteilung eines nicht einfachen, nicht konvexen Polygons

Für die Aufteilung wird jedes Teilpolygon $CP_1, ..., CP_p$ betrachtet. Konkret wird das Polygon $PredPoly(CP_i)$ so aufgeteilt, dass ein Teilstück einem Standort in CP_i (falls vorhanden) zugeordnet wird und der Rest dem Polygon $PredPoly(CP_k)$ mit $k > i$ angehangen wird. Diese Aufteilung wird durch die sich gegenseitig rekursiv aufrufenden Prozeduren *NonConvexDivide* und *DetachAndAssign* erreicht. Erstere erzeugt ein Liniensegment, welches $PredPoly(CP_i)$ in zwei Teile aufteilt und letztere ordnet die Teile entweder einem Standort zu oder teilt sie erneut auf.

Zuerst wird die Prozedur *NonConvexDivide* beschrieben, die die Teilpolygone in zwei Teile aufteilt. Listing 4 beschreibt diese Prozedur. Als Eingabe dient ein konvexes Teilpolygon, beschrie-

```
// Input: Convex polygon CP
Function NonConvexDivide(CP)
    Ls = W(1), Le = W(k)   // k = index of first Site CCW from w1 in W
    PrL, PlL = Cut(CP, L)   // partitioning, returns PrL and PlL
    while Area(PrL) < AreaRequired(S(PrL)) and Le != wm do
       if W(k-1) != S1 and W(k-1) in S then
          S(CPrL) = S(CPrL) + W(k-1)   // add previous Site to S(CPrL)
       k += 1
       Le = W(k)   // move Le to next point in W
       PrL, PlL = Cut(CP, L)
    if Area(PrL) > AreaRequired(S(CPrL)) then
        if Le == Si then
            k1 = 1
            while Area(PrL) > AreaRequired(S(PrL)) do
                k1 = k1 + 1
                Ls = w(k1)
            L1 = (w(k1), Le)
            T(t1,t2,t3) = (w(k1), w(k1 - 1), Le)
        else
            L1 = (Ls, w(k-1))
            T(t1,t2,t3) = (w(k - 1), w(k1), Ls)
        if Area(PrL1+T) > AreaRequired(S(CPrL)) then
            interpolate point t on (t1,t2) until
                Area(PrL+T) == AreaRequired(S(CPrL))
            T(t1,t2,t3) = (t1, t, t3)
            DetachAndAssign(PrL1 + T)
            DetachAndAssign(PlL1 - T - PredPoly(CP,(t1,t))
        else if Area(PrL1+PredPoly(CP,(t1,t2)) < AreaRequired(S(CPrL)) then
            interpolate point t on (t1,t2) until
                Area(PrL+T) == AreaRequired(S(CPrL))
            T(t1,t2,t3) = (t1, t, t3)
            DetachAndAssign(PrL1 + T)
            DetachAndAssign(PlL1 - T - PredPoly(CP, (t1,t))
        else
            PS = interiorPoint(t1,t2)   //PS is new Pseudosite
            T(t1,t2,t3) = (t1,PS,t3)
            AreaRequired(PS) = AreaRequired(S(CPrL) - Area(PrL1+T)
            Order(W(PredPoly(CP,(t1,t2))   //such that w1 = PS if Le! = Si and
                wm = PS if Le == Si
            DetachAndAssign(PredPoly(CP, (t1,t2))
            DetachAndAssign(PrL1 + T)
            DetachAndAssign(PlL1 + T)
    else
        t = InteriorPoint(wm, w1)
        L1 = (t,Si)
        DetachAndAssign(PrL1)
        DetachAndAssign(PlL1)
End NonConvexDivide()
```

Listing 4: Der Algorithmus *NonConvexDivide*

ben durch die Liste $W(CP_i)$ (mit $w_k, k = 1, \ldots, m$) mit allen Polygonpunkten inklusive Steiner-Punkten und die Liste $S(CP_i)$ mit den Standorten des Teilpolygons inklusive der jeweils benötigten Fläche. Anders als bei *ConvexDivide* aus Kapitel 2 ist für die Bearbeitung relevant, welche Punkte w_1 und w_m in $W(CP_i)$ sind. Die Kante, die durch die Polygonpunkte (w_m, w_1) erzeugt wird, sei nun die Kante zu $NextNeighbor(CP_i)$. Hat CP_i keinen nächsten Nachbarn, muss w_m gleich einem Standort sein. Weiterhin gilt, dass $W(CP_i)$ wieder gegen den Uhrzeigesinn geordnet ist.

Wie es auch schon bei *ConvexDivide* der Fall war, lässt die Prozedur erneut ein Liniensegment L gegen den Uhrzeigersinn durch das Polygon CP_i wandern, wobei L_s als Drehpunkt dient. L wird durch $(L_s, L_e) = (w_1, S_i)$ initialisiert, wobei S_i der erste Standort aus $S(CP_i)$ ist.
Nun können zwei Fälle eintreten, in denen die Schleife stoppt:

- Die Fläche rechts der Linie L ist größer oder gleich der benötigten Fläche der Standorte, die sich in diesem Gebiet befinden. Es gilt: $Area(P_L^r) \geq AreaRequired(S(CP_L^r))$
- Das Ende des Polygons wird erreicht, also $L_e = w_m$.

Durch die Bearbeitung von vorherigen Teilpolygonen kann es sein, dass nicht zugewiesene Teile dieser Polygone in die Aufteilung von CP_i miteinbezogen werden müssen. Außerdem kann nun der Fall eintreten, dass die Fläche des Teilpolygons kleiner ist als $AreaRequired(S(CP_i))$. Aus diesem Grund müssen die oberen beiden Fälle noch feingranularer aufgeteilt werden.

Fall 1: Wie auch in der Prozedur *ConvexDivide* wird ein Ende der Linie L entlang des Polygons bewegt, um die Fläche $Area(P_L^r)$ zu verkleinern. Hierbei unterscheiden wir zwei Fälle. Falls $L_e = S_i$ für einen beliebigen Wert für i gilt, dann wird der Startpunkt L_s gegen den Uhrzeigersinn bewegt, ansonsten wird der Endpunkt L_e im Uhrzeigersinn bewegt.
Nun seien L_1 und L_2 zwei Liniensegmente, die einen gemeinsamen, festen Endpunkt haben. Dieser gemeinsame Endpunkt ist für beide Linien entweder L_s oder L_e und damit das Gegenstück zum oben bestimmten Punkt, welcher entlang des Polygons bewegt wird. Die Linien sind so positioniert, dass $Area(P_{L_1}^r) < AreaRequired(S(CP_{L_1}^r))$ und $Area(P_{L_2}^r) > AreaRequired(S(CP_{L_2}^r))$ gilt. Die Linie L_2 wird demnach durch (w_1, w_k) und die Linie L_1 durch (w_1, w_{k-1}) beschrieben. Dadurch entsteht ein Dreieck $T = (t_1, t_2, t_3)$, das die Differenz von $CP_{L_1}^r$ und $CP_{L_2}^r$ bildet. Außerdem sei (t_1, t_2) das Liniensegment von CP_i, das L_1 und L_2 verbindet. Der gemeinsame Endpunkt von L_1 und L_2 ist demnach t_3.
Nun muss $CP_{L_1}^r$ mit einer Teilfäche des Dreiecks T und gegebenenfalls mit Teilflächen der Reste der Vorgängerpolygone vereinigt werden, damit die Flächenanforderungen der Standorte in $CP_{L_1}^r$ erfüllt werden. Dabei entstehen 3 Fälle:

- $Area(P_{L_1}^r + T) > AreaRequired(S(CP_L^r))$
  Die Flächenanforderung der Standorte kann durch die Fläche rechts von L_1 und T vollständig gedeckt werden. Insbesondere wird kein Flächenanteil von $PredPoly(CP, (t_1, t2))$ benötigt.

- $Area(P_{L_1}^r + T) \leq AreaRequired(S(CP_L^r))$ und
  $Area(P_{L_1}^r + PredPoly(CP, (t_1, t_2))) < AreaRequired(S(CP_L^r))$ $(*)$
  Die Flächen von $P_{L_1}^r$ und T reichen zusammen nicht $(<)$ oder exakt $(=)$ aus, um die Flächenanforderung der Standorte von CP_L^r zu erfüllen (1. Bedingung). Weiterhin liegt der Fall vor, dass die Fläche von $P_{L_1}^r$ in Kombination mit dem Vorgängerpolygon $PredPoly(CP, (t_1, t_2))$ kleiner als die geforderte Fläche ist (2. Bedingung).

- $Area(P_{L_1}^r + T) \leq AreaRequired(S(CP_L^r))$ und
  $Area(P_{L_1}^r + PredPoly(CP, (t_1, t_2))) \geq AreaRequired(S(CP_L^r))$
  Bedingung 1 ist analog zu Fall 1.2. Weiterhin liegt nun jedoch der Fall vor, dass die Fläche von $P_{L_1}^r$ in Kombination mit dem Vorgängerpolygon $PredPoly(CP, (t_1, t_2))$ zur Erfüllung der Anforderung genügt (2. Bedingung).

Die je nach Fall entstehenden Polygone werden anschließend an die Prozedur *DetachAndAssign* übergeben und dort entweder Standorten zugewiesen oder durch einen rekursiven Aufruf von *NonConvexDivide* erneut aufgeteilt.

Fall 1.1: Die Flächenanforderung von $S(CP_L^r)$ kann durch das Polygon $P_{L_1}^r$ zusammen mit dem Dreieck T erfüllt werden. In diesem Fall reicht es aus, mittels linearer Interpolation einen Punkt t zwischen t_1 und t_2 zu finden, sodass für das Dreieck $T' = (t_1, t, t_3)$ gilt:

$$Area(P_{L_1}^r + T' - PredPoly(CP, (t_1, t))) = AreaRequired(S(CP_L^r))$$

Durch diese Aufteilung entstehen die beiden Polygone $(P_{L_1}^r + T' - PredPoly(CP, (t_1, t))$ und $(P_{L_1}^r - T')$, die der Prozedur *DetachAndAssign* übergeben werden.

Fall 1.2: Damit die Flächenanforderung erfüllt werden kann, wird zunächst das Vorgängerpolygon $PredPoly(CP, (t_1, t_2))$ hinzugenommen und dieses um die Fläche $P_{L_1}^r$ und einen Teil des Dreiecks T erweitert. Erneut wird durch lineare Interpolation der Punkt t gefunden und wie oben das Dreieck T' gebildet, sodass die Flächenanforderung erfüllt ist. Das Dreieck T' kann durch die strikte Ungleichung $(*)$ nicht kollabieren. Somit entstehen die beiden Polygone $(P_{L_1}^r + T')$ und $(P_{L_1}^r - T' - PredPoly(CP, (t_1, t))$, die der Prozedur *DetachAndAssign* übergeben werden.

Fall 1.3: Die Fläche der Vorgängerpolygone ist größer als die Flächenanforderung der Standorte. In diesem Fall muss die Fläche der Vorgängerpolygone wiederrum aufgeteilt werden. Ein Teil wird zur Erfüllung der Flächenanforderung genutzt und $S(CP_{L_1}^r)$ zugeordnet. Der übrige Teil wird im Weiteren durch einen sogenannten Pseudostandort PS abgebildet. PS wird auf der Kante zwischen t_1 und t_2 willkürlich hinzugefügt, sodass das Dreieck $T' = (t_1, PS, t_3)$ entsteht. Es gilt:

$$AreaRequired(PS) = AreaRequired(S(CP_L^r) - Area(P_{L_1}^r + T'))$$

PS bekommt also die fehlende Flächenanforderung zugewiesen. Dadurch kann $PredPoly(CP, (t_1, PS))$ ebenfalls durch *NonConvexDivide* aufgeteilt und ein Teilpolygon PS zugewiesen werden. Das zugewiesene Teilpolygon kann dann dem Polygon $(P_{L_1}^r + T')$ hinzugefügt werden. Dieses Polygon und das Polygon $(P_{L_1}^l - T')$ werden dann an *DetachAndAssign* übergeben.

Weiterhin muss der Fall betrachtet werden, bei dem das Liniensegment L das Polygon einmal komplett durchlaufen hat.
Fall 2: Dieser Fall tritt ein, wenn ein Teilpolygon und die Reste der Vorgängerpolygone weniger Fläche enthalten, als die Standorte beanspruchen. In diesem Fall ist CP_i Flächen-unvollständig und es muss für mindestens einen Standort aus $S(CP_L^r)$ ein Pseudostandort erzeugt werden. Dazu wird ein Punkt t auf der Kante (w_m, w_1) erzeugt, also der Kante zu $NextNeighbor(CP_i)$. Nun sei $L = (t, S_i)$, wobei S_i der erste Standort in $S(CP_i)$ gegen den Uhrzeigersinn von w_1 aus ist. Nun wird $W(P_L^r)$ so geordnet, dass $t = w_1$ gilt. Anschließend werden P_L^r und P_L^l an *DetachAndAssign* übergeben. Hierbei entsteht entweder auf dem Liniensegment (w_1, w_2) ein Pseudostandort oder ein Teil von P_L^r wird dem Standort S_i zugeordnet. Wenn ein Pseudostandort entsteht, dann wird diesem die Flächenanforderung von S_i abzüglich der Fläche von P_L^r zugeordnet und das Polygon P_L^r von CP_i entfernt. Mit den restlichen Standorten von CP_i wird dasselbe Verfahren angewandt. Die Pseudostandorte werden nun bei der Aufteilung von $NextNeighbor(CP_i)$ behandelt. Wenn den Pseudostandorten hierbei ein Polygon zugeteilt wird, dann wird dieses Polygon auf die korrespondierenden Standorte übertragen.
Zusammenfassend wird durch den Algorithmus von *NonConvexDivide* ein q-Standort-Polygon entweder in ein q_1-Standort-Polygon und ein q_2-Standort-Polygon mit $q_1, q_2 > 0$ und $q_1 + q_2 = q$ aufgeteilt oder es wird ein 1-Standort Polygon abgetrennt und es bleibt ein q'-Standort Polygon mit $q' = q - 1$ übrig.

3.5 Der Algorithmus DetachAndAssign

Die Prozedur *DetachAndAssign* teilt ein Polygon einem Standort zu oder teilt ein Teilpolygon erneut mittels *NonConvexDivide* auf. *DetachAndAssign* ist durch Listing 5 beschrieben.
Beim Aufruf von *DetachAndAssign*(*Poly*(*CP*)) können 3 Fälle auftreten. Das Polygon

- $PredPoly(CP)$ ist Flächen-vollständig
- $PredPoly(CP)$ ist Flächen-unvollständig
- $PredPoly(CP)$ ist Standort-unvollständig

```
// Input: Poly(CP) - Polygon rooted at convex piece CP
Function DetachAndAssign(Poly(CP))
    if Length(S(CP)) == 0 then return
    if PredPoly(CP) is AreaComplete then
        if S(CP) == {Si} then   //for some i
            Assign PredPoly(CP) to Si
            Detach PredPoly(CP) from Poly(CP)
        else
            Detach PredPoly(CP) from Poly(CP)
            Order(W(CP))   //such that wm = Si for some i
            NonConvexeDivide(CP)
    else if PredPoly(CP) is areaIncomplete then
        if S(CP) == {Si} then   //for some i
             Assign PredPoly(CP) to Si
             Detach PredPoly(CP) from Poly(CP)
             PS = interiorPoint(w(j),w(k))   //with (w(j),w(k)) is edge to
                 NextNeighbor(CP)
        else
            Order(W(CP))   //such that edge (w(m),w(1)) is edge to
                NextNeighbor(CP)
            NonConvexeDivide(CP)
    else
        Order(W(CP))   //such that edge (w(m),w(1)) is edge to NextNeighbor(CP)
        NonConvexeDivide(CP)
End DetachAndAssign()
```

Listing 5: Der Algorithmus *DetachAndAssign*

Im ersten Fall kann es sein, dass $PredPoly(CP)$ nur einen Standort besitzt. Dann kann $PredPoly(CP)$ vom Polygon $Poly(CP)$ getrennt (*Detach*) und komplett diesem Standort zugeteilt (*Assign*) werden. Falls $PredPoly(CP)$ mehrere Standorte enthält, wird $PredPoly(CP)$ von $Poly(CP)$ getrennt und rekursiv mittels *NonConvexDivide* aufgeteilt.
Im zweiten Fall treten die gleichen zwei Unterfälle auf. Falls $PredPoly(CP)$ nur einen Standort S_i hat, kann $PredPoly(CP)$ von $Poly(CP)$ getrennt und dem Standort zugeteilt werden. Da $PredPoly(CP)$ Flächen-unvollständig ist, muss nun ein Pseudostandort auf der Kante zu $NextNeighbor(CP)$ erzeugt werden, der die restliche Flächenanforderung von S_i enthält. Flächen, die im weiteren Verlauf dem Pseudostandort zugeteilt werden, werden so mittelbar dem Standort S_i zugeteilt. Falls $PredPoly(CP)$ mehrere Standorte hat, dann wird $PredPoly(CP)$ zunächst neu geordnet, sodass (w_m, w_1) die Kante zu $NextNeighbor(CP)$ ist. Anschließend erfolgt wiederum ein rekursiver Aufruf von *NonConvexDivide*, da nicht bekannt ist, durch welchen Standort die Flächen-Unvollständigkeit resultiert.
Im dritten Fall hat $PredPoly(CP)$ mehr Fläche, als die Standorte von CP benötigen. In diesem Fall wird $PredPoly(CP)$ ebenfalls neu geordnet, sodass (w_m, w_1) die Kante zu $NextNeighbor(CP)$ ist. Mit diesem Polygon erfolgt ein Aufruf von *NonConvexDivide*.

Abbildung 6 zeigt ein Beispiel einer Zerlegung eines nicht konvexen Polygons.

3.6 Behandlung innen liegender Standorte

Für innen liegende Standorte muss die in Kapitel 3.2 beschriebene Zerlegung in konvexe Teilpolygone so erfolgen, dass jeder Standort anschließend auf einer Kante liegt. Ist dies nicht direkt möglich, können für die Standorte auch weitere Kanten eingefügt werden und die Aufteilung in konvexe Teilpolygone wird etwas detaillierter. Die neuen Kanten laufen durch die innenliegenden Standorte. Für den korrekten Ablauf des Algorithmus spielt diese Art der Zerlegung keine Rolle.

4 Komplexitätsanalyse

4.1 Konvexes Polygon

Der Algorithmus *ConvexDivide* benötigt lineare Zeit bezogen auf die Anzahl der Elemente der Liste $W(P)$, um einen einzelnen Schnitt durchzuführen. Der dabei erforderliche Aufbau der Polygone P_L^r beziehungsweise P_L^l sowie die Ermittlung der Fläche ist in konstanter Zeit möglich. Das Finden der Punkte, bei denen $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt, kann ebenso in konstanter Zeit erfolgen. Der Algorithmus *ConvexDivide* benötigt daher $O(n+v)$ Zeit (n = Anzahl der Standorte, v = Anzahl an Polygonpunkten).
Im ungünstigsten Fall trennt *ConvexDivide* von einem konvexen Polygon mit q Standorten nur ein Dreieck ($v = 3$, $n = 1$) ab. Neben dem Dreieck verbleibt dann ein konvexes Polygon mit $q - 1$ Standorten und $v + 1$ Polygonpunkten. Für dieses Polygon gilt wiederum selbes. Um eine gesamte Flächenzerlegung eines konvexen Polygons zu berechnen, wird $O((n-1)(n+v))$ Zeit benötigt. Hierbei sei angemerkt, dass dieser Algorithmus stets terminiert, da die Anzahl an Standorten konstant ist und die Anzahl der Teilpolygone je Schnitt um 1 erhöht wird. Nach $n - 1$ Schnitten entspricht die Anzahl der Teilpolygone der Anzahl der Standorte.

4.2 Nicht konvexes Polygon

Der Algorithmus *OrderPieces* besucht jeden Knoten im Nachbarschaftsgraphen maximal zwei Mal, sodass die Ordnung in linearer Zeit $O(p)$ bezogen auf die Anzahl p der konvexen Teile erfolgen kann. Der Algorithmus *NonConvexDivide* benötigt $O(pn^2 + nv)$ Zeit, um alle p konvexen Teile unter den n Standorten aufzuteilen. Im ungünstigsten Fall wird jedes der p konvexen Teile in n Polygone zerlegt, wobei jedes der Polygone einen Teil eines Standorts abbildet.
Der Algorithmus *DetachAndAssign* wird getrennt für den Teil *Detach* beziehungsweise *Assign* betrachtet.
Um ein Polygon nach einem Schnitt zu lösen (*Detach*), müssen alle Zeiger auf Nachbarpolygone aktualisiert werden. Dieser Vorgang ist in Zeit $O(v_j)$ für ein konvexes Polygon mit v_j Polygonpunkten möglich. Der ungünstigste Fall beziehungsweise die maximale Anzahl an zu aktualisierenden Zeigern besteht dann, wenn je Zerlegung ein Dreieck ($v = 3$) und ein Polygon mit $v_j + 1$ Polygonpunkten abgetrennt werden. Für alle Teilpolygone ergibt sich dann eine maximale Zeit von $O(v + pn^2)$.
Der Algorithmus *Assign* übernimmt die Zuweisung von Flächen zu Standorten, wobei eine Fläche aus einem Set von konvexen Teilen besteht. Die Vereinigung der konvexen Teile zu einem Polygon benötigt lineare Zeit bezogen auf die Anzahl der Polygonpunkte aller Teile, also maximal $O(pn+v)$. Die Vereinigung aller n-Standort-Polygone benötigt daher $O(pn^2+vn)$ Zeit. Zusammenfassend ergibt sich aus den Laufzeiten für *OrderPieces*, *NonConvexDivide* und *DetachAndAssign* eine Gesamtlaufzeit von $O(pn^2 + vn)$.

5 Schluss und Ausblick

In dieser Arbeit wurde das *Problem der verankerten Flächenaufteilung* vorgestellt und anhand von Beispielen für konvexe und nicht konvexe Polygone erläutert. Sowohl der beschriebene Algorithmus für ein konvexes Polygon, als auch die verallgemeinerte Version für ein nicht konvexes, nicht einfaches Polygon liefern für ein gegebenes Problem eine Lösung.

Weitere Untersuchungsmöglichkeiten für die Anpassung und Optimierung des Algorithmus liegen beispielsweise für den konvexen Fall in der Reihenfolge der Punkte des Eingabepolygons beziehungsweise für den nicht konvexen Fall in der Art der Aufteilung des Polygons in konvexe Teilpolygone CP_i. Weitergehende Forschung könnte den Einfluss dieser Parameter auf die Form der entstehenden Polygone untersuchen. Unter anderem könnte es für die Roboterplanung von Vorteil sein, wenn die resultierenden Flächen möglichst einfach und kompakt (im Sinne des kleinesten umschließenden Durchmessers) sind, da Roboter auf solchen Flächen typischerweise effizienter arbeiten.

6 Abkürzungs- und Begriffsverzeichnis

Abkürzung/Begriff	Bedeutung
c	Geforderter Flächenanteil des Standorts S am zugehörigen Polygon P mit $0 < c < 1$
CP	Ein konvexes Polygon
FE	Flächeneinheiten
L	Liniensegment, orientiert von L_s nach L_e
L_s	Startpunkt des Liniensegments L
L_e	Endpunkt des Liniensegments L
P	Polygon
CP_L^r	Teilpolygon rechts des Liniensegments L
CP_L^l	Teilpolygon rechts des Liniensegments L
S	Standort
Steiner-Punkt	Polygonpunkt, welcher nicht Teil des Eingangspolygons ist und während der Lösung eines geometrischen Problems hinzugefügt wird
q-Standort-Polygon	Polygon mit insgesamt q Standorten
$V(P)$	Liste der Polygonpunkte von P, inklusive aller Steiner-Punkte, gegen den Uhrzeigersinn geordnet
$S(P)$	Liste der zu P zugeordneten Standorte S_i, in der Reihenfolge ihres Vorkommens gegen den Uhrzeigersinn geordnet, beginnend mit v_1
$W(P)$	Liste der Polygonpunkte und Standorte von P, d.h. $S(P)$+$V(P)$, wobei $w_1 \in V(P)$ bzw. $w_1 \notin S(P)$ gilt
$Area(P)$	Flächeninhalt des Polygons P
$AreaRequired(S(P))$	Benötigte Fläche der Standorte im Polygon P, berechnet mit $c * Area(P)$
Flächen-vollständig	Wenn für ein Polygon P gilt: $Area(P) = AreaRequired(P)$
Flächen-unvollständig	Wenn für ein Polygon P gilt: $Area(P) < AreaRequired(P)$
Standort-unvollständig	Wenn für ein Polygon P gilt: $Area(P) > AreaRequired(P)$
Nachbar	Zwei Teilpolygone, die eine gemeinsame Kante haben, werden als Nachbarn bezeichnet
Vorgänger von CP_i	Alle Teilpolygone CP_k, die eine kleinere Ordnung als CP_i haben und es einen Weg im Verbindungsgraph von CP_i nach CP_k gibt
Nachfolger	Alle Teilpolygone CP_k, die eine größere Ordnung als CP_i haben und es einen Weg im Verbindungsgraph von CP_i nach CP_k gibt
$NextNeighbor(CP_i)$	Ein Nachbarpolygon des Teilpolygons CP_i, das ein Nachfolger von CP_i ist und die kleinste Ordnung hat

Abkürzung/Begriff	Bedeutung
$PredPoly(CP_i)$	CP_i und alle von CP_i aus erreichbaren Vorgänger, ohne dabei einen Nachfolger zu schneiden
$PredPoly(CP_i, (e))$	CP_i und alle von CP_i aus über die Kante e erreichbaren Vorgänger, ohne dabei einen Nachfolger zu schneiden
$Poly(CP_i)$	CP_i und alle von CP_i aus erreichbaren Teilpolygone
P_L^r	CP_L^r und $PredPoly(CP_i, (e))$, für alle Kanten, die rechts der Linie L verlaufen oder einen Endpunkt auf L haben
P_L^l	CP_L^l und $PredPoly(CP_i, (e))$, für alle Kanten, die links der Linie L verlaufen oder einen Endpunkt auf L haben

Literatur

[1] S. Hert and V. Lumelsky. Polygon area decomposition for multiple-robot workspace division. *International Journal of Computational Geometry & Applications*, 08(04):437–466, 1998.

[2] Schachter. Decomposition of polygons into convex sets. *IEEE Transactions on Computers*, C-27(11):1078–1082, 1978.

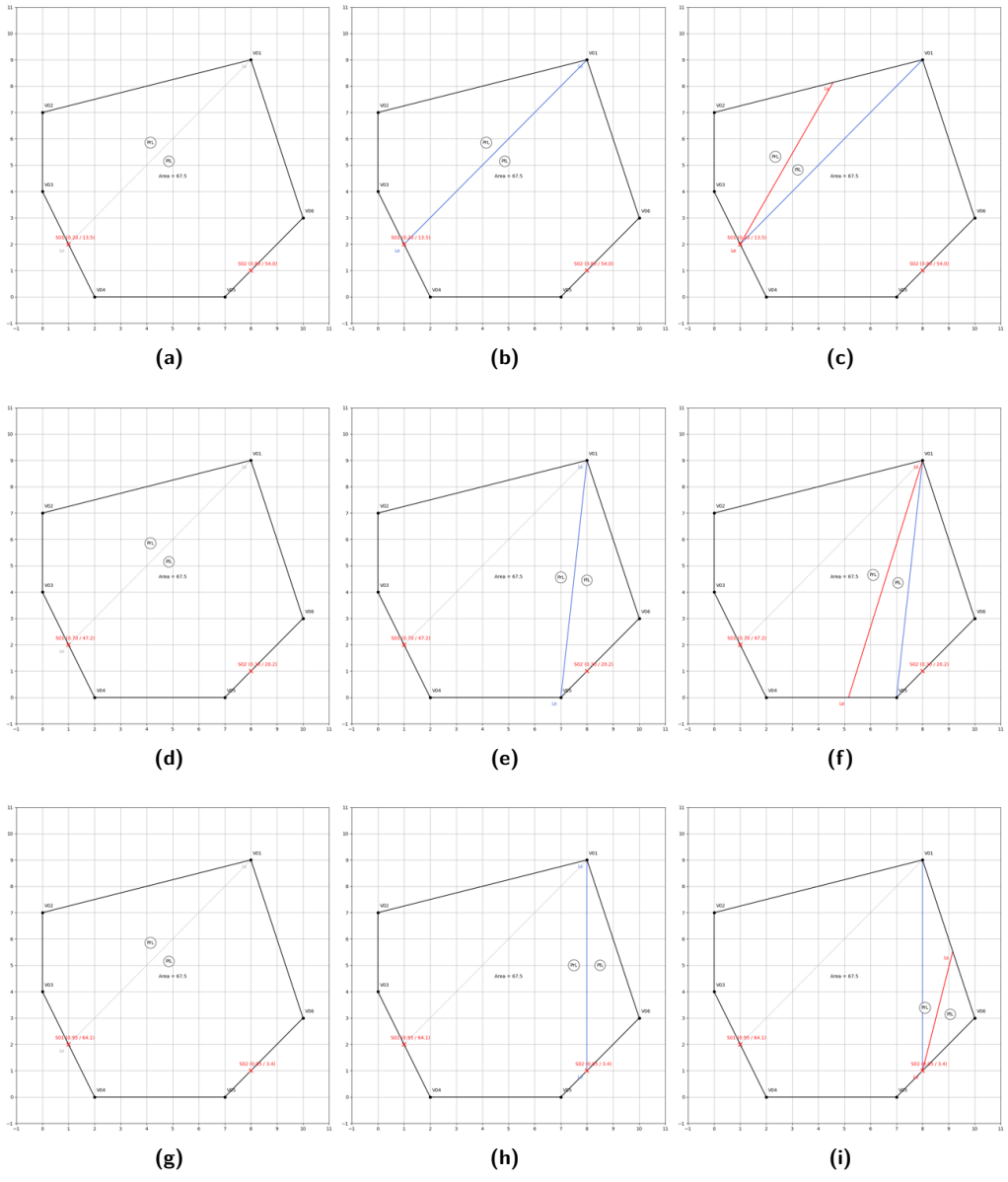

Abbildung 2: Fall 1, 2.1 und 2.2 inklusive der jeweiligen Zwischenschritte im Algorithmus *ConvexDivide*.

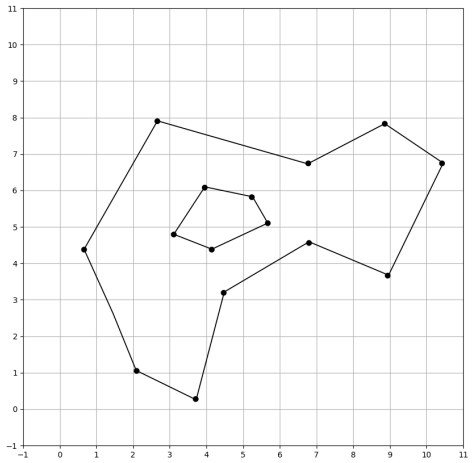

Abbildung 3: Beispiel eines nicht einfachen, nicht konvexen Polygons

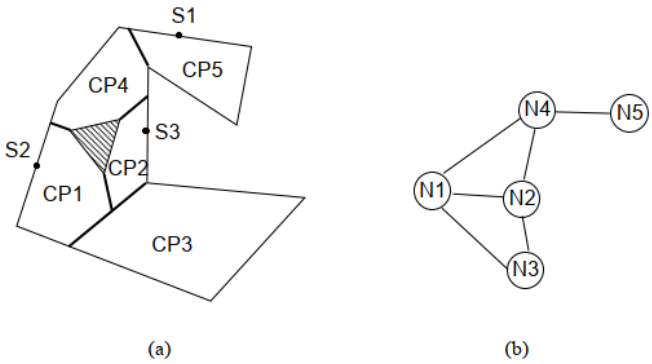

Abbildung 4: Ein in konvexe Teilpolygone zerlegtes Polygon und dessen Verbindungsgraph. Wenn *OrderPieces* mit CP_1 als Eingabe aufgerufen wird, dann werden die verschiedenen Teilpolygone wiefolgt ausgegeben: $CP_5, CP_3, CP_2, CP_4, CP_1$. Das Beispiel ist übernommen aus [1].

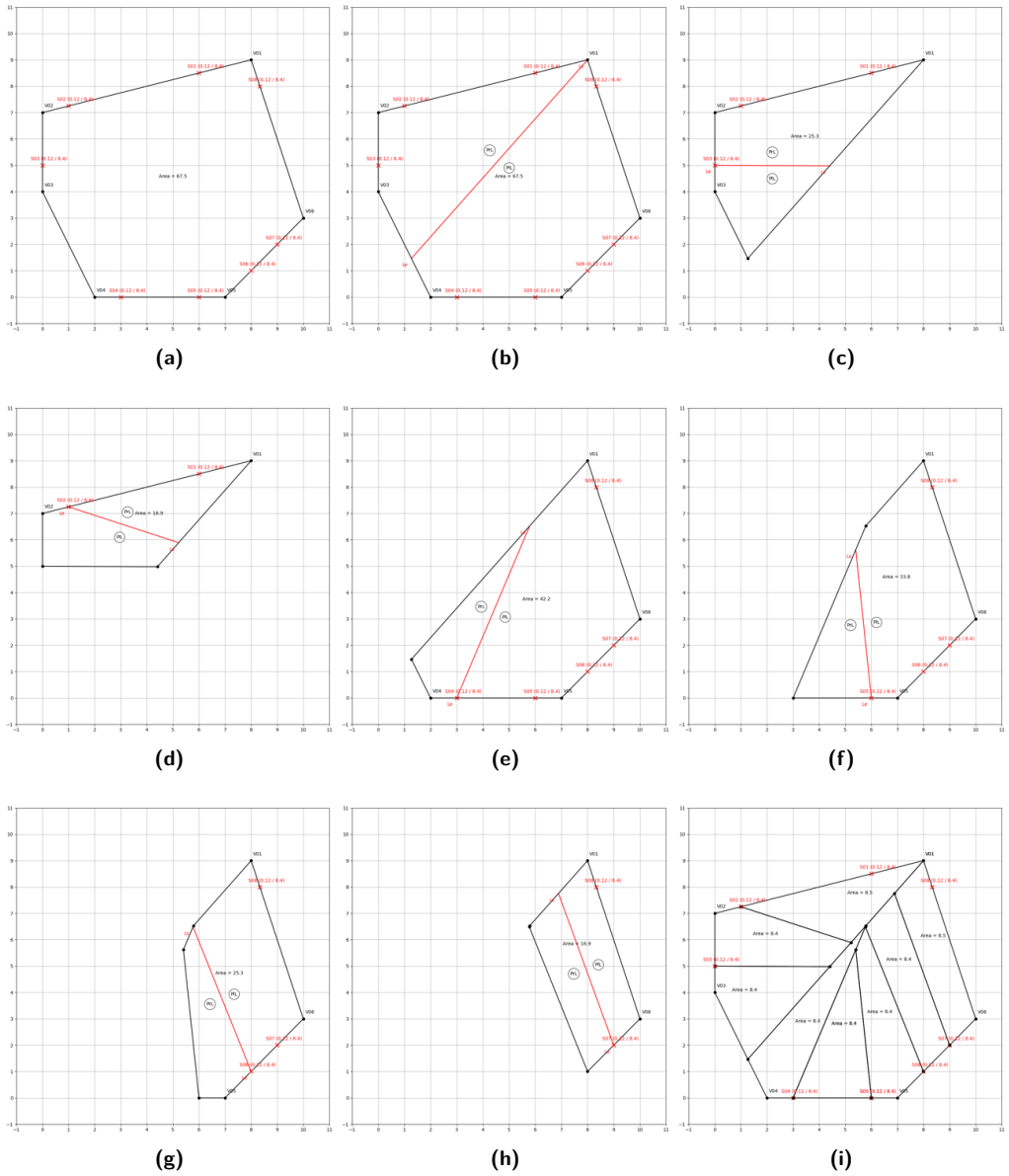

Abbildung 5: Beispiel einer gleichmäßigen Aufteilung eines konvexen Polygons mit acht Standorten und einem jeweils benötigen Flächenanteil von $12,5\%$. (a) zeigt hierbei das Ausgangspolygon CP mit dem ersten Polygonpunkt bei Koordinate $(8,9)$ und den im Gegenuhrzeigersinn geordneten Standorten $S01$ bis $S08$. In (b) liegt zunächst Fall 2.1 vor, bei welchem L_e bis $V04$ bewegt wird, sodass $Area(P_L^r) > AreaRequired(S(P_L^r))$ gilt. $S(P_L^r)$ enthält in diesem Zustand die Standorte $S01$ bis $S03$. Für L_e wird anschließend eine Interpolation zwischen $V04$ und $V03$ durchgeführt, sodass $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt und es wird ein Schnitt durchgeführt. Für das rechts der Schnittlinie entstandene Polygon wird in (c) erneut *ConvexDivide* aufgerufen. Hierbei tritt Fall 2.2 ein. L_e erreicht bei der Verschiebung $S03$, welcher letzter Standort in S_i st. L_s wird anschließend im Uhrzeigersinn inkrementell bewegt, bis $Area(P_L^r) = AreaRequired(S(P_L^r))$ gilt. Erneut muss das rechts der Schnittlinie entstandene Polygon aufgeteilt werden, siehe (d). L_e wandert wieder bis zum letzten Standort in $S()$, sodass erneut Fall 2.2 vorliegt. Dass $Area(P_L^r)$ bei $L_e = S_n$ eine Fläche von 0 aufweist, stellt für den Algorithmus kein Problem dar. In (e) wird nun das in (b) links der Schnittlinie entstandene Polygon weiter aufgeteilt. Es liegt Fall 1 vor, da nach der Initialisierung mit $L_e = S_1$ bereits $Area(P_L^r) > AreaRequired(S(P_L^r))$ gilt. L_s wird anschließend gegen den Uhrzeigersinn verschoben, bis $Area(P_L^r) = AreaRequired(S(P_L^r))$ vorliegt. In den nächsten Schritten (f)-(h) gilt analog Fall 1. Speziell in (f) soll darauf hingewiesen werden, dass das inkrementelle Verschieben des Punktes L_s auch über eine Polygonecke hinaus möglich sein muss. (i) zeigt die resultierende Flächenaufteilung aus den Schritten (b) - (h) mit Teilflächen zu je $12,5\%$.

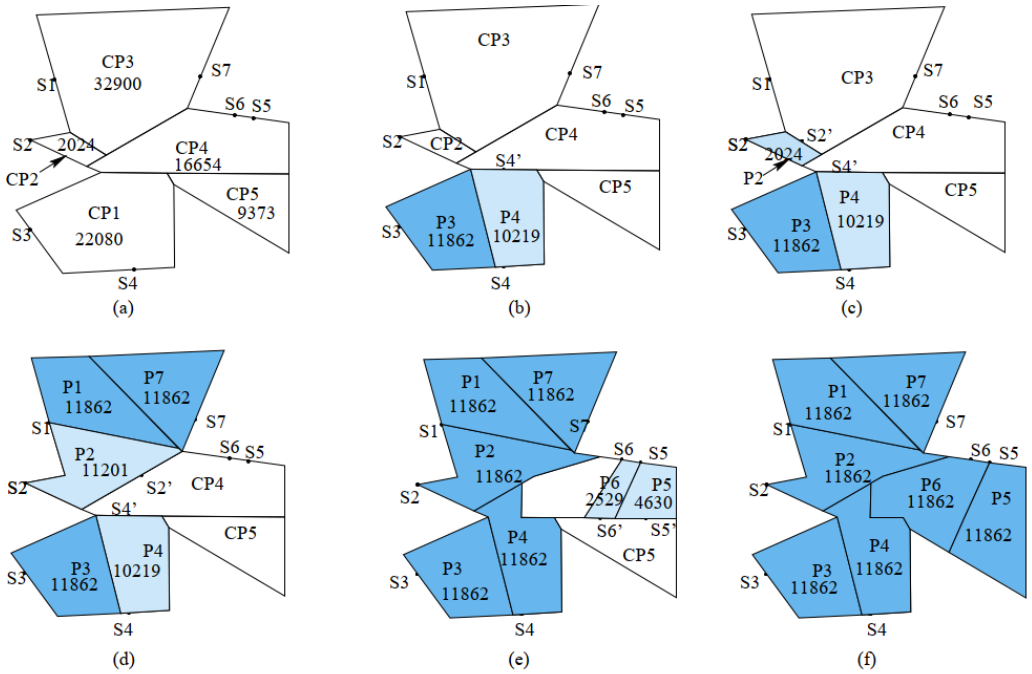

Abbildung 6: Beispiel einer Aufteilung eines nicht konvexen Polygons, entnommen aus [1] Abbildung 17

Gezeigt sind hier die verschiedenen Stadien der gleichmäßigen Aufteilung eines nicht konvexen Polygons mit 12 Ecken und sieben Standorten. (a) zeigt die initiale Aufteilung des Polygons in 5 konvexe Teilpolygone $CP1, \ldots, CP5$. In (b) – (f) werden die Teilpolygone, die bereits einem Standort zugeteilt sind, dunkelblau markiert. Die Teilpolygone, die bereits einem Standort zugeteilt, aber noch Flächen-unvollständig sind, werden hellblau markiert.

In (b) wird das Teilpolygon $CP1$ bearbeitet und dabei in zwei Teilpolygone aufgeteilt. $P3$ wird dem Standort $S3$ zugeordnet und $P4$ dem Standort $S4$. $P4$ erfüllt die Flächenanforderung von $S4$ nicht vollständig, weshalb ein Pseudostandort $S'4$ an der Kante zu $NextNeighbor(CP1)$ erzeugt wird. Dieser Pseudostandort wird zu einem späteren Zeitpunkt bearbeitet.

(c) zeigt den Zustand nach der Bearbeitung von Teilpolygon $CP2$. Hier tritt erneut der Fall auf, dass die Fläche, die dem Standort $S2$ zugeteilt wird, zu klein ist. Aus diesem Grund wird der Pseudostandort $S'2$ an der Kante zu $NextNeighbor(CP2)$ erzeugt.

In (d) wird der Zustand nach der Bearbeitung von Teilpolygon $CP3$ gezeigt. Dort wird das Teilpolygon $P7$ dem Standort $S7$ zugeteilt und das Teilpolygon $P1$ dem Standort $S1$. Der Rest von $CP3$ wird mit $P2$ vereint und daher ebenfalls $S2$ zugewiesen. Da die Fläche von $P2$ weiterhin nicht groß genug ist, um der Flächenanforderung von $S2$ zu genügen, wird ein neuer Pseudostandort $S'2$ an der Kante zu $NextNeighbor(CP3)$ erzeugt.

(e) zeigt den Zustand nach der Bearbeitung von $CP4$. Dort hat das Liniensegment das Ende von $CP4$ erreicht, ohne dass zuvor die Flächenanforderung der Standorte erfüllt werden konnte. Aus diesem Grund wird $P5$ und der Pseudostandort $S'5$ erzeugt und diese $S5$ zugeordnet. Anschließend wird derselbe Schritt für $S6$ wiederholt. Nach Abziehen der Flächen $P5$ und $P6$ von $CP4$ werden Teile von $CP4$ den Pseudostandorten $S'2$ und $S4$ zugeordnet und mit den Teilpolygonen $P2$ und $P4$ vereint.

Im letzten Schritt (f) wird $CP5$ und der verbliebene Teil von $CP4$ den Pseudostandorten $S5$ und $S6$ zugeordnet. Die Abbildung zeigt das in gleichmäßige Teilpolygone $P1, \ldots, P7$ zerlegte Polygon.