Graphenbasierte Modellierung

Einleitung

In CAD Systemen möchte man nicht nur die geometrischen Objekte definieren, sondern man möchte auch funktionale Abhängigkeiten zwischen den geometrischen Objekten bzw. zwischen sonstigen Variablen, z.B. Materialkonstanten oder Belastungen, und den geometrischen Objekten definieren. Dies geschieht mit Hilfe von Constraints. Diese können z.B. festlegen, dass der Abstand von 2 Punkten ein veränderliches Maß haben soll, oder das 2 Linien gleichlang oder parallel sein sollen. Die Techniken zur Lösung dieser Geometrischen Constraint Systeme können in 3 Klassen eingeteilt werden:

- Gleichungsbasierte Methoden.
  Hier werden die Constraints durch i.A. nichtlineare Gleichungen zwischen den Variablen dargestellt. Für das nichtlineare Gleichungssystem wird entweder mit Hilfe des Newton -Raphson Verfahrens eine iterative Lösung gesucht [Light& Gossard, 1982] oder es wird eine algebraische Methode mit Hilfe von Gröbner Basen benutzt [Hoffmann, 1989] [Kondo, 1992].
- Konstruktive Techniken.
  Hier arbeitet man entweder mit Regelbasierten [Verroust, Schonek, Roller, 1992] oder Graphbasierten Techniken.
  Regelbasierte Techniken führen zur Konstruktion von Control Sets aus Längen und Winkeln. Diese Controls Sets definieren Dreiecke oder Vierecke, die dann zur Gesamtlösung zusammengefasst werden.
  Graphbasierte Techniken modellieren die **Constraints** als **Hyperkanten** zwischen den **geometrischen** oder **sonstigen Variablen**. Statt mit einem Hypergraphen, kann man auch mit einem **bipartiten Graphen** arbeiten, beidem jeweils **Constraints** und **Variablen** durch zunächst **ungerichtete Kanten** verbunden sind. Entsprechend der Anzahl der Constraints können nun aus einer gegebenen Teilmenge der Variablen, die übrigen berechnet werden.
  Man will nun nach Erstellung eines Modells frei wählen können, welche Variablen vorgegeben und welche berechnet werden. Z.B. soll nicht nur ein Punkt als Schnittpunkt von 2 Linien berechnet werden, sondern bei einer Verschiebung des Schnittpunktes sollen die Linien sich adäquat bewegen. Um aus dem Modell und den festgelegten Variablen eine Berechnungsfolge zu konstruieren wird eine Analyse der Freiheitsgrade benutzt. Ziel ist es aus dem **ungerichteten** Constraint- Graphen einen **gerichteten** zu erzeugen. Ist eine **sequentielle Berechnung** möglich, soll auch ein **azyklischer Graph** entstehen. In manchen Fällen, wo zyklische Abhängigkeiten bestehen, kann jedoch nur ein Graph mit Zyklen erzeugt werden. Sind zu viele Variablen festgelegt, d.h. das System ist überbestimmt ist keine Orientierung möglich. Ansätze in dieser Richtung liefern [Berling, 1996], [Fudos, Hoffmann, 1993], [Hsu, Brüderlin, 1997], [Fudos 2000].

Hier wird der Ansatz von [Berling, 1996] weiter geführt. Es wird jedoch eine reichere Auswahl von Constraint Klassen betrachtet. Ferner wird bei der Analyse der Freiheitsgrade auch berücksichtigt, dass manche Constraints nicht unabhängig von einander sind. So hat ein Punkt im 2D zwar 2 Freiheitsgrade, die jedoch nicht durch 2 Constraints abgedeckt werden können, die beide nur die X oder Y Koordinate

beeinflussen. Ferner dürfen die durch die Constraints gegebenen Ortskurven nicht Kreise mit dem gleichen Mittelpunkt oder parallele Linien sein. Des Weiteren sollen auch stark unterbestimmte Systeme ein sinnvolles Verhalten zeigen. Daher haben die Constraint nicht nur auf die direkt berührten Punkten Auswirkungen, sondern sie werden auch auf entfernter liegende Punkte propagiert.

Geometrische Modellierung

Bei der Erstellung des geometrischen 2D-Modells werden folgende Variablen benutzt:

- Punkt: 2 Freiheitsgrade (X, Y Koordinaten)
- Kreis 3 Freiheitsgrade(X, Y Mittelpunkt, Radius)
- skalaren Wert: 1 Freiheitsgrad(X Wert)

Ein Kreis wird abgeleitet von Punkt, damit kann der Mittelpunkt, wie ein normaler Punkt behandelt werden.
Ein Skalar kann z.B. die Distanz von 2 Punkten oder Radius eines Kreises sein.

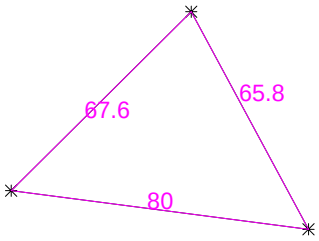
Die Freiheitsgrade der Variablen werden eingeschränkt durch die zu erfüllenden Constraints. Das Constraint, das den Abstand zweier Punkte bestimmt, nimmt z.B. einen Freiheitsgrad weg. Eine Konstruktion kann also maximal so lange Constraints erfüllen, bis durch sie alle Freiheitsgrade der Variablen gebunden sind.

Beispiel:

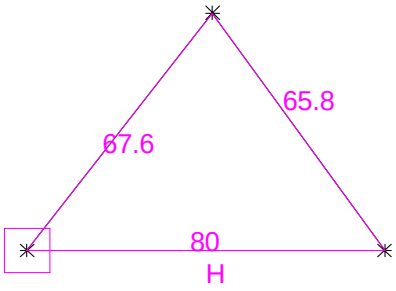
- Eine Linie gegeben durch 2 Endpunkte.
- Freiheitsgrade: 4 (X1,Y1,X2,Y2)
- Gibt man die Länge vor hat man noch 3 Freiheitsgrade.(Verschieben in X und Y Richtung und Drehung)
- Fordert man, dass die Linie waagrecht ist, so bleiben noch 2 Freiheitsgrade (Verschieben in X und Y Richtung).

Man kann jetzt noch die Koordinaten eines Punktes festlegen oder die X-Koordinate des einen und die Y-Koordinate des anderen Punktes. Man kann jedoch nicht die X oder Y Koordinaten beider Punkte festlegen, da diese durch die vorherigen Constraints bereits von einander abhängig sind. Man muss daher bei dem Verfolgen der Constraints nach halten, auf welche Koordinaten der Constraint Einfluss hat. Der Constraint Waagrecht beeinflusst nur die Y-Koordinaten.

Ferner muss man berücksichtigen, dass manche Constraints die Lage der Punkte untereinander beeinflussen andere die Lage und Ausrichtung der Gesamtkonstruktion.
So beeinflusst der Abstand zweier Punkte oder der Constraint, der festlegt, dass 3 Punkte einen rechten Winkel bilden, nur die Lage der Punkte zu einander. Für diese Lage zu einander, d.h. die kongruente Form, gibt es bei n Punkten jedoch nur 2n-3 Freiheitsgrade. Daher kann man bei einem Dreieck, bei 3 Seitenlängen, nicht noch einen Winkel bestimmen.

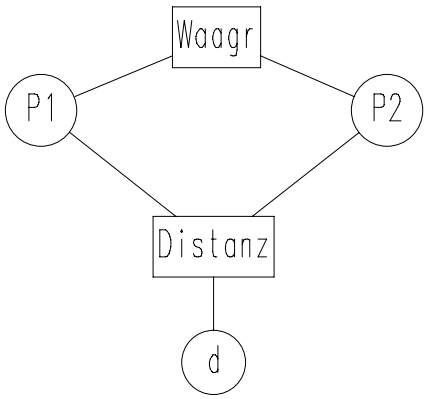


Man kann aber die Koordinaten eines Punktes festlegen und zusätzlich fordern, dass die untere Linie waagrecht ist.



Die Punkte, Kreise, Skalarvariablen und Constraint bilden die Knoten eines bipartiten Graphen. Die eine Knotenklasse bilden die Punkte, Kreise und die skalaren Variablen (diese Klasse wird im Folgenden Variablen genannt), die andere die Constraints. Die Kanten stellen die Beziehungen zwischen den Constraints und den Variablen dar.

Beispiel: 2 Punkte mit Abstand und Waagrechter Ausrichtung.



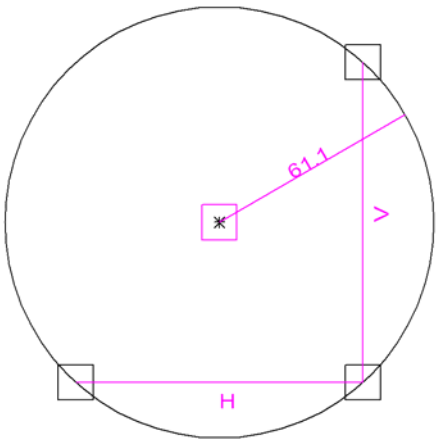
Die Variablen werden später im gerichteten Graphen mit maximal so vielen einlaufenden Kanten verbunden, wie sie Freiheitsgrade haben. Ist der Wert der Variablen festgelegt, werden die adjazenten Kanten später alle als auslaufend orientiert. Wird eine Skalarvariable für mehrere Constraints genutzt, kann nur maximal eine Kante einlaufend sein.

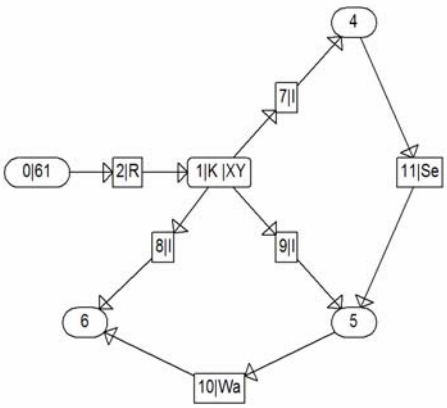
Die Constraint Knoten haben eine Kantenverbindung zu jeder Variablen, die durch sie beeinflusst wird. In der Literatur wird i.A. mit Constraints gearbeitet, die eine feste Anzahl von Freiheitsgraden binden. Ein Constraint, das den Abstand von 2 Punkten bestimmt, wird mit 2 Punkten und einer Skalar Variablen verknüpft. Entweder bestimmen die beiden Punkte den Abstand oder der Abstand und ein Punkt legt für den anderen Punkt eine Ortskurve fest. Man sagt dann der Abstand- Constraint konsumiert einen Freiheitsgrad. In der hier betrachteten Modellierung haben die Constraints jedoch einen vorgegebenen Freiheitsgrad. Der Constraint Distanz hat 2 Freiheitsgrade, d.h. maximal 2 der adjazenten Kanten können einlaufend orientiert sein. Dies könnte wieder sein:

- Die beiden Punkte: Dann wird der Abstand d festgelegt.
- Ein Punkt und der Abstand d: Dann wird für den anderen Punkt eine Ortskurve bestimmt, hier Kreis um den ersten Punkt mit dem Radius d.

Ein Constraint konsumiert so viele Freiheitsgrade der Variablen, wie er auslaufende Kanten hat, d.h. Anzahl der adjazenten Kanten – Freiheitsgrad des Constraint.
Hier können einige Constraints eine variable Anzahl von adjazenten Kanten haben. Der Constraint Waagrecht z.B. bestimmt, dass eine Reihe von Punkten die gleichen Y-Koordinaten haben sollen. Es kann nur eine Kante einlaufend orientiert werden. Der adjazente Punkt bestimmt die Y-Koordinate der anderen Punkte. Dieses Constraint hat somit einen Freiheitsgrad. Sollen n Punkte waagrecht ausgerichtet sein, so werden demnach n-1 Freiheitsgrade bei den Variablen gebunden.

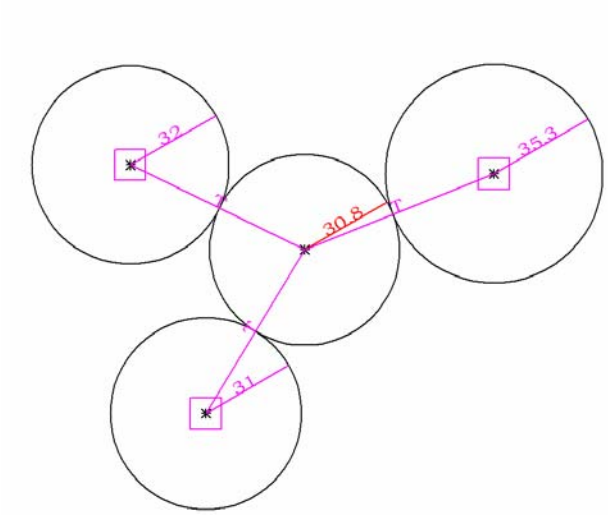
Ein Kreis kann durch jede Kombination von 3 vorgegebenen Elementen festgelegt werden.



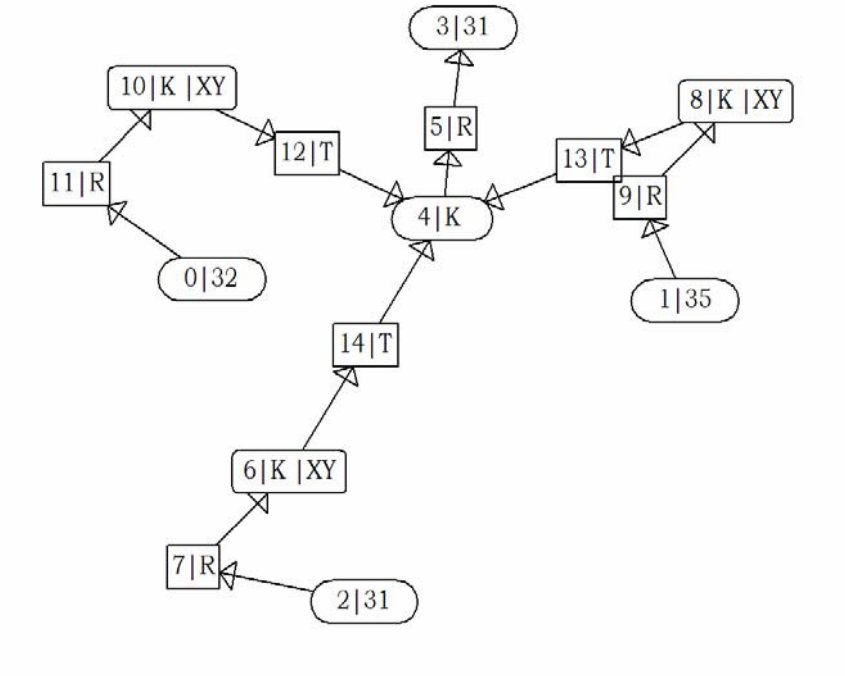


Hier ist der Kreis durch Mittelpunkt und Radius gegeben. Für die Punkte, die auf dem Kreis liegen, ist ein Freiheitsgrad gebunden. Der andere kann noch genutzt werden, z.B. für die waagrechte bzw. senkrechte Ausrichtung. Von den 3 Punkten könnte noch einer in einer Koordinate verschoben werden.

Ein Kreis kann auch dadurch bestimmt werden, dass er tangential zu 3 anderen Kreisen liegt.



Wird einer der 3 Kreise verschoben, so werden für Mittelpunkt und Radius des inneren Kreises die Werte angepasst.



Es ist jedoch nicht immer möglich, den Graphen kreisfrei zu orientieren. Dann werden in dem Graphen die starken Komponenten ermittelt. Das ist eine Aufteilung der Knotenmenge in Teilmengen so, dass die Zyklen alle innerhalb der starken Komponenten liegen. Die starken Komponenten mit mehr als einem Knoten werden im Allgemeinen durch eine Iteration berechnet. Dies führt jedoch nicht immer zu einem Ergebnis. Daher werden später Verfahren vorgestellt, die für einige häufig vorkommende Fälle eine direkte Lösung erlauben. Die Reihenfolge der Berechnung der starken Komponenten kann nach der topologischen Sortierung erfolgen.

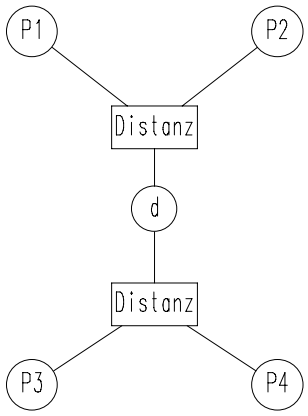
Beschreibung der Constraints

	Freiheits-grad	Adjazente Knoten	Beschreibung	Bestimmte Koordinate
Twaagrecht	1	Punkte, die waagrecht ausgerichtet werden	Ein Punkt bestimmt die Y-Koordinate der anderen Punkte	Y
Tsenkrecht	1	Punkte, die senkrecht ausgerichtet werden	Ein Punkt bestimmt die X-Koordinate der anderen Punkte	X
Tdistanz	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	XY
Tdistanzx	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	X
Tdistanzy	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	Y
Twinkel	2	2 oder mehr Punkte und ein Skalar	2 Punkte bestimmen einen Winkel oder der Winkel und ein Punkt bestimmen die anderen Punkte	XY
Twinkel3P	3	3 Punkte und ein Skalar	3 Punkte bestimmen einen von ihnen aufgespannten Winkel oder der Winkel und 2 Punkte bestimmen den 3. Punkt	XY

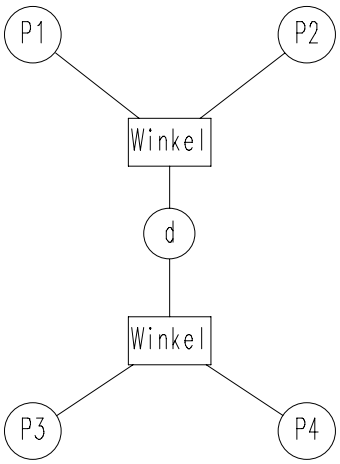
Trechtwinklig	2	3 Punkte	Wie Winkel3P, jedoch ist der Winkel auf 90° festgelegt	XY
Tparallele	3	4 Punkte und ein Skalar	Die durch die Punkte P1,P2 bzw. P3,P4 gebildeten Linien sind Parallel mit dem Abstand d	XY
Tlinie	2	2 oder mehr Punkte, die auf einer Geraden liegen	2 Punkte bestimmen die Gerade auf der alle Punkte liegen.	XY
Tgleichmx	2	2 oder mehr Punkte, die horizontal gleichmäßig verteilt werden	Entsprechend den X-Koordinaten von 2 Punkten, werden die restlichen Punkte horizontal gleich verteilt	X
Tgleichmy	2	2 oder mehr Punkte, die vertikal gleichmäßig verteilt werden	Entsprechend den Y-Koordinaten von 2 Punkten, werden die restlichen Punkte vertikal gleich verteilt	Y
Tdistanzlp	3	Skalar (Abstand) und 2 Punkte, die eine Gerade definieren und ein Punkt, dessen Abstand bestimmt wird	3 Punkte bestimmen den Abstand oder Abstand und 2 Punkte bestimmen den dritten Punkt	XY
Toperator	2	2 Skalare als Operanden und ein Skalar als Ergebnis	Die Operanden bestimmen das Ergebnis oder ein Oparand und das Ergebnis bestimmen den anderen Operanden	XYR
Tradius	1	Kreis, Radius (Skalar)	Der Kreis hat den Radius des Skalar	XYR

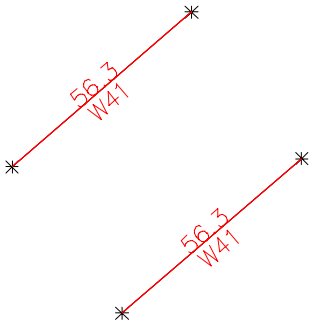
Tinzident	1	Kreis, Punkt	Der Punkt liegt auf dem Kreis	XYR
Ttangential	1	2 Kreise	2 Kreise sind tangential	XYR
Ttangentialkl	2	Kreis,2 Punkte	Die Gerade, die durch die Punkte geht ist tangential an den Kreis	XYR
Ttangente2c	2	2 Kreise 2 Punkte	Die Gerade, die durch die Punkte geht ist eine Tangente an die beiden Kreise	XYR
Ttangente2cwagrecht	1	2 Kreise 2 Punkte	Die Gerade, die durch die Punkte geht ist eine waagrechte Tangente an die beiden Kreise	XYR

Andere Constraints lassen sich aus diesen zusammensetzen. Sollen 2 Linien, die gleiche Länge haben, werden 2 Distanz Constraints definiert, deren Abstand identifiziert wird. Im Beispiel soll der Abstand zwischen P1 und P2 gleich dem zwischen P3 und P4 sein.

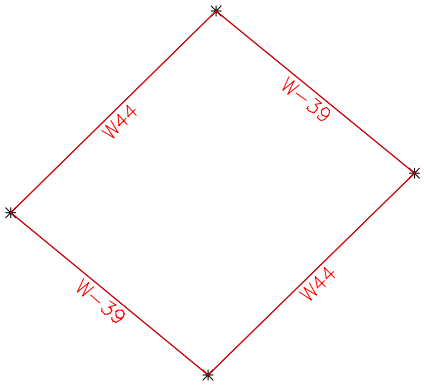


Analog kann definiert werden, wenn 2 Linien parallel sein sollen. Der Constraint Winkel bestimmt den Winkel zwischen den beiden Punkten. Ähnlich wie bei der gleichen Länge haben hier die Winkel den gleichen Wert.

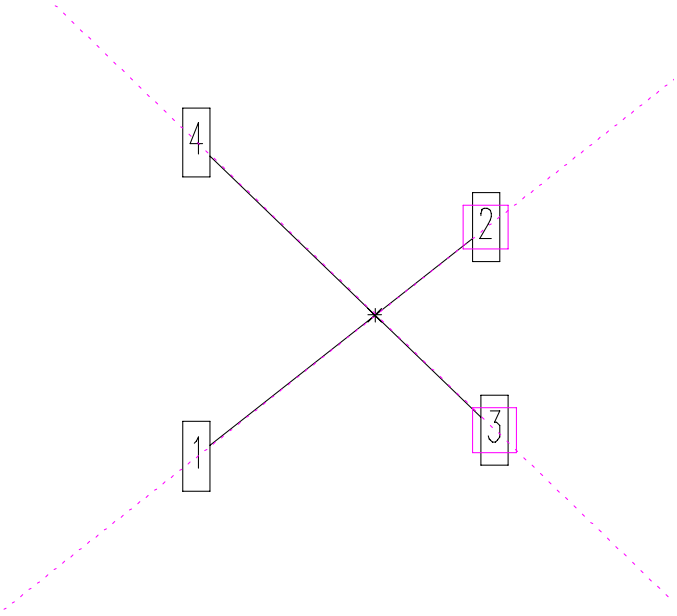


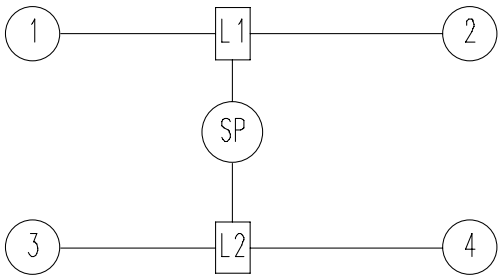


Damit kann z.B. ein Parallelogramm definiert werden.

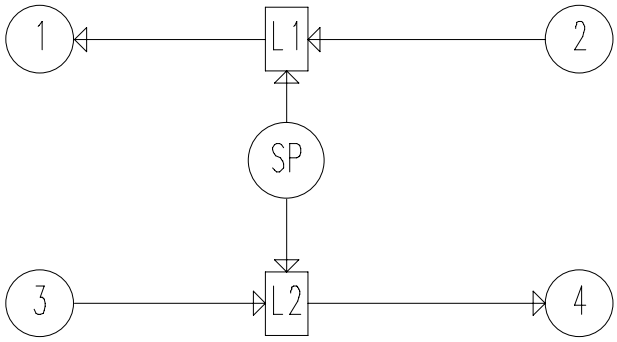


Ein Punkt kann durch zwei Linien Constraints als Schnittpunkt von 2 Linien definiert werden.



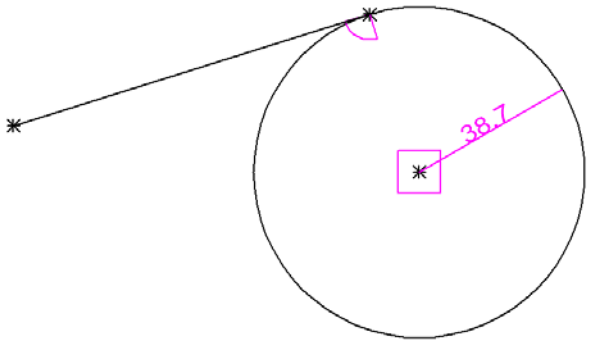


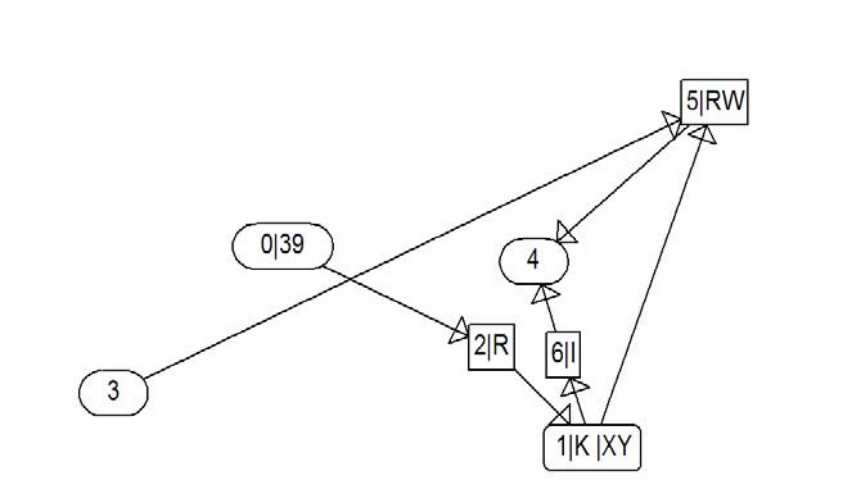
Der Schnittpunkt muss auf den Linien der Punkte 1-2 und der Punkte 3-4 sein. Wird der Schnittpunkt verschoben, so ändern sich die Punkte 1 und 4, da 2 und 3 fixiert sind. Die Orientierung des Graphen ist dann:



Ändern sich die Punkte 1-4 ändert sich analog der Schnittpunkt.

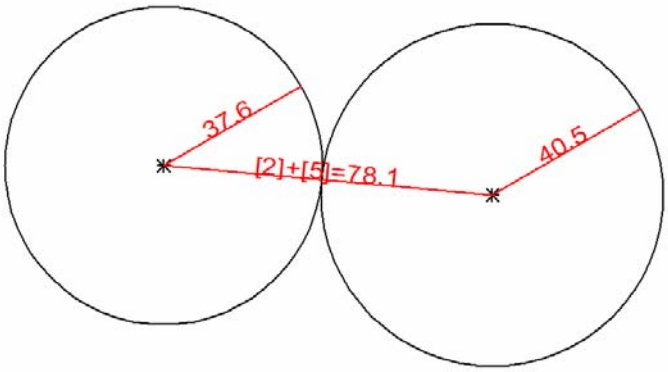
Auch eine Linie, die von einem Punkt aus tangential an einen Kreis geht, kann über das Constraints Rechtwinklig definiert werden.

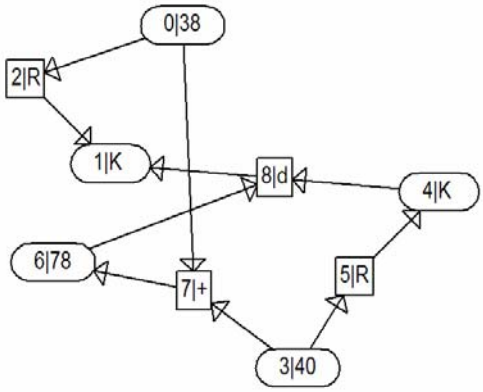




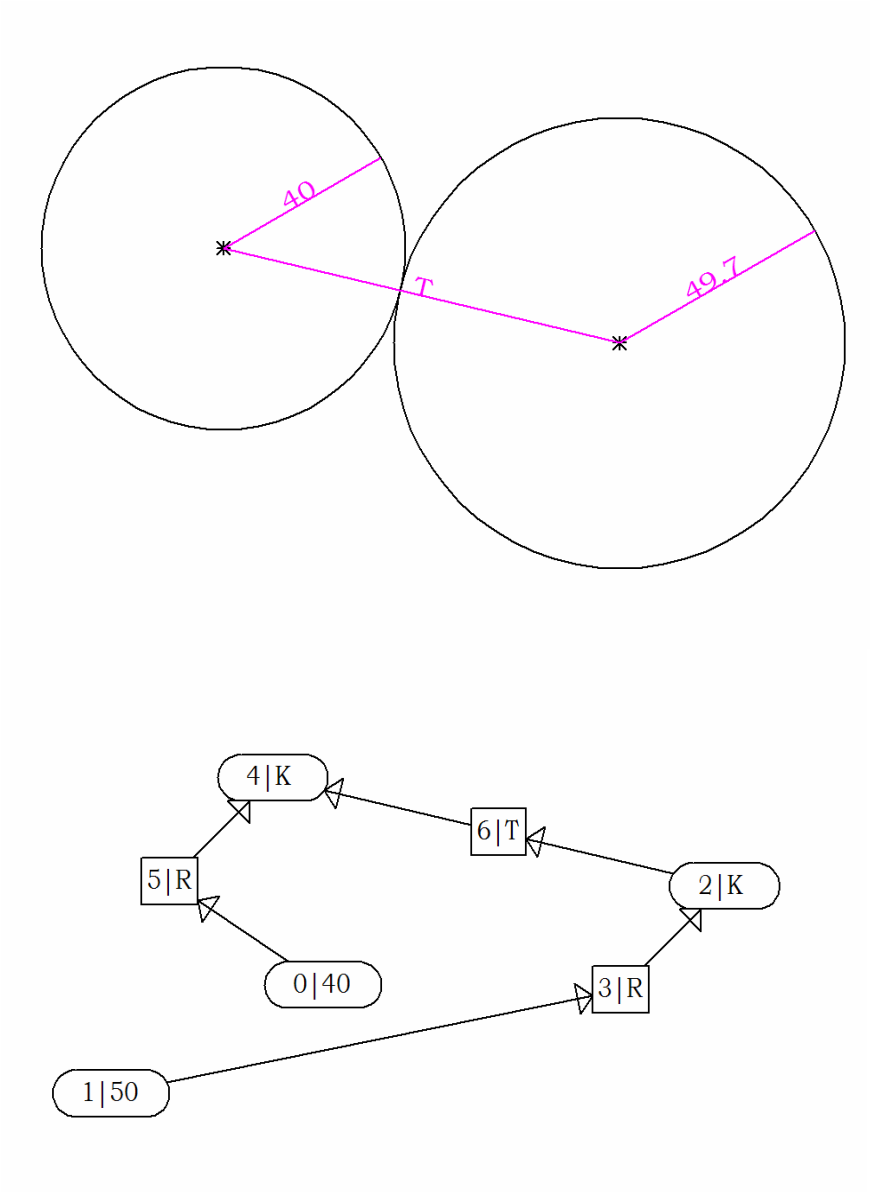
der Punkt von dem die Tangente gezeichnet wird, der Mittelpunkt und der Tangentialpunkt bilden einen rechten Winkel. Werden Punkt, Mittelpunkt und Radius festgelegt, so muss der Tangentialpunkt auf dem Kreis liegen und einen rechten Winkle bilden.

Sollen 2 Kreise sich tangential berühren, so kann dies über eine Abstandsbeziehung der Mittelpunkte definiert werden. Der Abstand der Mittelpunkt muss die Summe bzw. die Differenz der Radien ergeben.





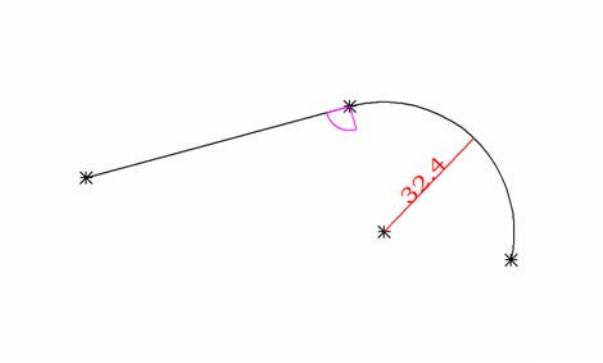
Hat man aber mehrere tangentiale Beziehungen zwischen Kreisen, so führt diese Form der Modellierung sehr leicht zu Zyklen. Einfacher ist es daher ein spezielles Constraint zu definieren, das die tangentiale Beziehung zwischen 2 Kreisen festlegt.

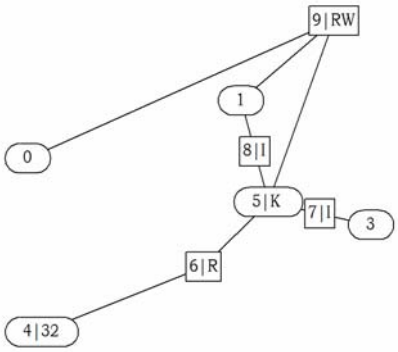


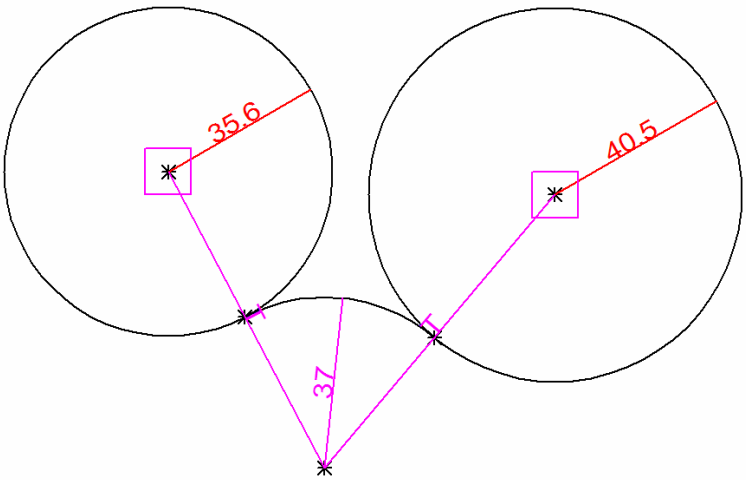
Sind die Radien festgelegt, so ist auch der Abstand der Mittelpunkte bestimmt. Wird ein Mittelpunkt verschoben, wird der andere nachgezogen.

Durch entsprechende Orientierung des Constraint Graphen können sowohl bei einer Änderung der Radien die neuen Mittelpunkte berechnet werden, als auch bei einer Änderung der Mittelpunkte neue Radien ermittelt werden. Die Konstruktion hat insgesamt 6 Freiheitsgrade (2 Kreise). Tangential hat einen Freiheitsgrad und 2 adjazente Kanten, bindet also einen Freiheitsgrad. Insgesamt bleiben also 5 Freiheitsgrade übrig, z.B. Koordinate eines Mittelpunktes, 2 Radien und Richtung des anderen Mittelpunktes oder auch 2 Mittelpunkte und ein Radius.

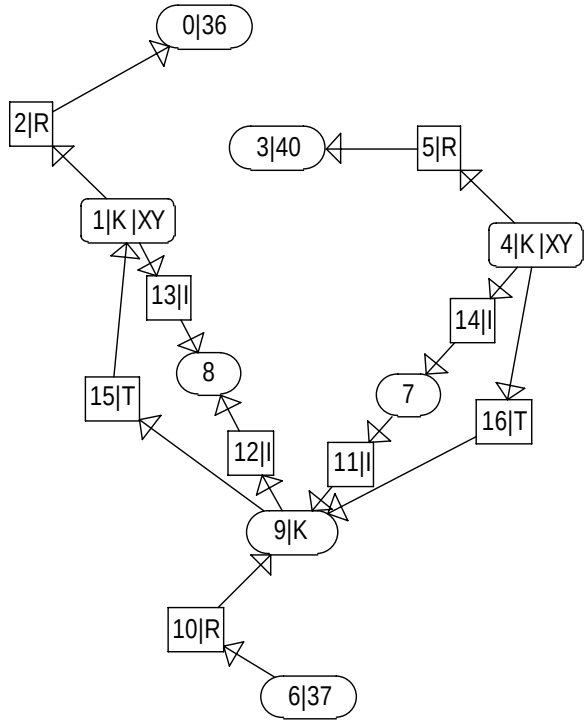
Soll ein Bogen tangential an eine Linie anschließen, so muss der Endpunkt der Linie, der Anfangspunkt des Bogens sein, ferner muss er tangential sein, d.h. die beiden Linienpunkte und der Kreismittelpunkt bilden einen rechten Winkel.





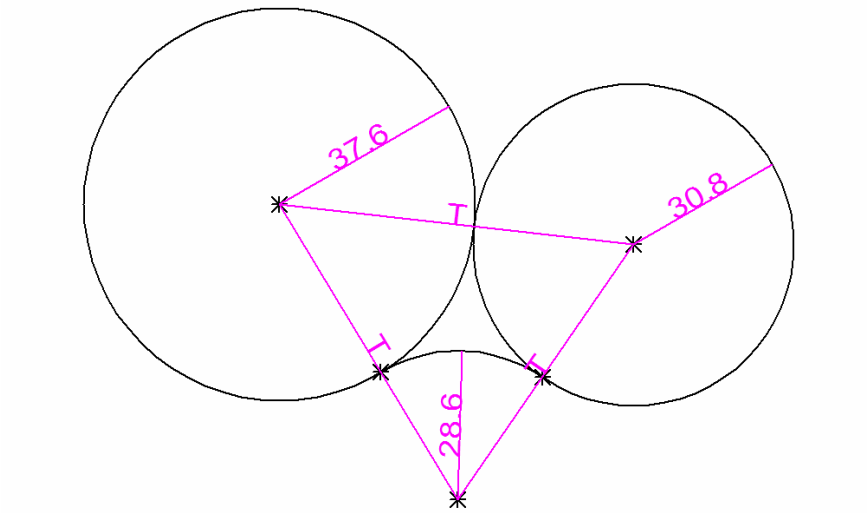


Der Bogen sei tangential an den beiden Kreisen. Das kann wieder über das Constraint Tangential ausgedrückt werden (Constraints 15 und 16).



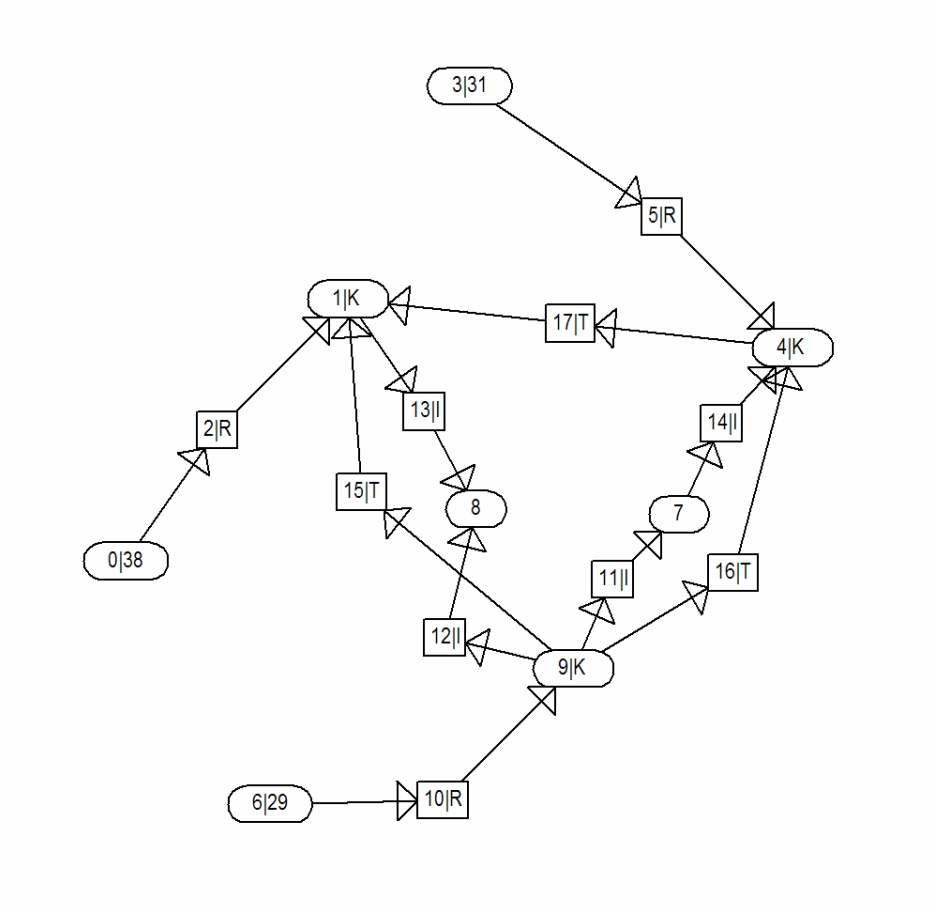
Im obigen Beispiel sind die Mittelpunkte der Kreise und der Radius des Bogens vorgegeben. Wird ein Kreis verschoben, so passen sich der Bogen und die Radien der Kreise jeweils an. Der orientierte Graph ist sogar zyklenfrei.

Man kann nun noch die beiden Kreise tangential machen und die Radien fixieren



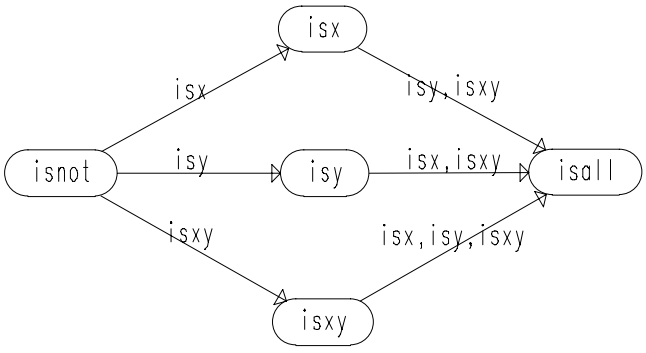
Für die 3 Kreise hat man 9 Freiheitsgrade. Davon werden durch die Cpmstraints belegt:
3x Tangential
3x Radius
es bleiben noch 3 Freiheitsgrade übrig, diese werden durch die Lage bestimmt. Die Form ist durch die 6 belegten Freiheitsgrade bestimmt.

Man kann daher z.B. den Mittelpunkt des Bogens verschieben. Der gerichtete Graph hat dann folgende Form:



Orientierung des Constraintgraphen

Durch einlaufende Kanten wird der Freiheitsgrad der Variablen reduziert. Ist der Freiheitsgrad auf 0 reduziert, müssen alle weiteren Kanten auslaufend orientiert werden. Wie oben gezeigt beeinflussen manche Constraint jedoch nur die X bzw. Y Koordinate eines Punktes. Daher kann der Freiheitsgrad nicht einfach runtergezählt werden, sondern es muss folgende Transitionsmatrix benutzt werden:



Ein Punkt hat zunächst 2 Freiheitsgrade, was dem Zustand isnot entspricht. Wird durch ein Constraint die X-Koordinate bestimmt, geht er in den Zustand isx über. Nun kann er nur noch ein Constraint erfüllen, was auch die Y-Koordinate verändert. Dies kann entweder ein Constraint sein, das nur die Y-Koordinate bestimmt (isy) oder ein Constraint, was eine Ortskurve festlegt (isxy). Insgesamt kann ein Punkt also folgende Constraintpaare erfüllen:

- `Isx, isy`
- `Isx, isxy`
- `Isy, isxy`
- `Isxy, isxy`

Für die Kreise ist dies noch etwas komplizierter, da ein Kreis zusätzlich auch noch durch den Radius bestimmt wird. Jedoch nicht alle Constraints, können durch den Radius abgedeckt werden, sondern nur: tradius, tinzident und ttangential.
Ein Kreis hat kann zusätzlich noch Constraints durch seinen Radius erfüllen.
type tfix=(isnot,isx,isy,isxy,isall,isr,isxr,isyr,isxyr,is2xy);

Auch die Funktion, die für einen Punkt, die noch vorhandenen Freiheitsgrade berechnet, hat als Parameter, was dadurch zusätzlich fixiert werden soll:

```
function tnod.sdf(afix:tfix):integer;
var wasx,wasy,wasxy,wasxy2:boolean;
 i,res:integer;
 fix:tfix;
 function moeglich:boolean;
 begin
 result:=false;
 if (afix=isx) and wasx then exit
 else if (afix=isy) and wasy then exit
 else if (afix=isxy) and wasxy2 then exit;
 result:=true;
 end;
begin
wasx:=false;wasy:=false;wasxy:=false;wasxy2:=false;
result:=0;
res:=2;
case fix2 of
 isx:begin wasx:=true;res:=1;end;
 isy:begin wasy:=true;res:=1;end;
 isxy,isall:exit;
end;
if not moeglich then exit;
for i:=0 to count-1 do if ori[i]=isin then begin
 fix:=constr[i].fix;
 res:=res-1;
 if fix=isx then begin
  if wasx then exit;wasx:=true;
 end else if fix=isy then begin
  if wasy then exit;wasy:=true;
 end;
 if not moeglich then exit;
```

```
end;
if res>0 then result:=res else result:=0;
end;
```

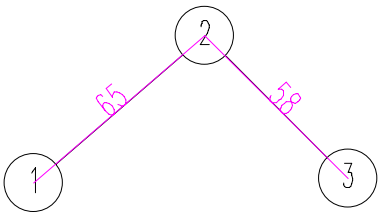
`fix2` gibt den derzeitigen Zustand des Punktes an. Ist z.B. die X Koordinate bereits fixiert (`isx`), so hat er für ein Constraint, das auch nur die X Koordinate fixiert, keinen Freiheitsgrad mehr, für andere Constraints (`isy,isxy`), die auch die Y Koordinate beeinflussen, jedoch noch einen Freiheitsgrad.

Bei einem Kreis muss zusätzlich noch isr, isxr, isyr, isxyr berücksichtigt werden.

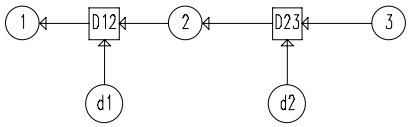
```
function tkreis.sdf(afix:tfix):integer;
var fix0,fix1:integer;
 wasx,wasy,wasr,wasxy,wasxy2:boolean;
 i,res:integer;
 fix:tfix;
 function moeglich:boolean;
 begin
 result:=false;
 if (afix=isx) and wasx then exit
 else if (afix=isy) and wasx then exit
 else if (afix=isr) and wasr then exit
 else if (afix=isxy) and wasxy2 then exit;
 result:=true;
 end;
begin
wasx:=false;wasy:=false;wasr:=false;wasxy:=false;wasxy2:=false
;
result:=0;
res:=3;
case fix2 of
 isx:begin wasx:=true;res:=2;end;
 isy:begin wasy:=true;res:=2;end;
 isr:begin wasr:=true;res:=2;end;
 isxy,isall,is2xy:begin
  wasx:=true;wasy:=true;wasxy:=true;wasxy2:=true;res:=1;
 end;
end;
if not moeglich then exit;
for i:=0 to count-1 do if ori[i]=isin then begin
 fix:=constr[i].fix;
 res:=res-1;
 if fix=isx then begin
  if wasx then exit;wasx:=true;
 end else if fix=isy then begin
  if wasy then exit;wasy:=true;
 end else if fix=isxy then begin
  if wasxy2 then exit;if wasxy then wasxy2:=true else
wasxy:=true;
 end else if fix=isr then begin
  if wasr then exit;wasr:=true;
 end;
```

```
 if not moeglich then exit;
end;
result:=res;
end;
```

Ist es möglich, einen Constraint Graphen zyklenfrei zu orientieren, kann diese Orientierung auch gefunden werden. Eine möglich zyklenfreie Orientierung, kann jedoch nicht immer berechnet werden.
Beispiel: Gegeben seien 3 Punkte, die durch 2 Linien mit je einem Abstand Constraint verbunden sind.



Wird der Punkt 3 soweit verschoben, dass sein Abstand vom Punkt 1 größer als die Summe der Abstände ist, wird man für den Punkt 2 keine Lösung mehr finden. Falls der Punkt 1 nicht festgelegt ist muss also die Verschiebung von 3 auch nach 1 propagiert werden, damit 1 entsprechend nachgezogen wird. Der orientierte Constraint Graph hat dann folgende Form:



Für den Punkt 2 gibt es dann nur die Ortskurve Kreis um 3 mit Radius d2. Von der bisherigen Position von 2 wird dann ein Lot auf diesen Kreis gefällt. Das gleiche geschieht bei Punkt 1. Verschiebt man Punkt 3 so werden die Punkt 2 und 1 nachgezogen, wobei sich die Anordnung immer mehr einer geraden Linie nähert.

Algorithmus zur Orientierung des Constraintgraphen

Bevor versucht wird den ungerichteten Graphen zu orientieren, wird zunächst getestet, ob die Konstruktion nicht überbestimmt ist.

Für die einzelnen Constraints muss untersucht werden, ob sie die Form, die Lage oder die Ausrichtung bestimmen.

Die Constraints tdistanz, twinkel3p, tdistanzlp, tparallele verringern jeweils die Formfreiheitsgrade um eins, wenn ihr Wert vorgegeben ist, bzw. wenn der gleiche Wert schon in einem anderen Constraint benutzt wurde.
Trechtwinklig verringert der Formfreiheitsgrad jeweils um eins, tlinie und tgleichm um die Anzahl der Punkte-2.
Die Constraints tinzident und tradius verringern die Formfreiheitsgrade jeweils um eins.
Bei den Constraints twaagrecht, tsenkrecht, twinkel bestimmen der Wert und die ersten beiden Punkte die Ausrichtung, alle weiteren Punkte und alle weiteren Constraints verringern die Formfreiheitsgrade.

Der erste fixierte Punkt bestimmt die Lage der Konstruktion. Weitere Fixierungen reduzieren die Formfreiheitsgrade. Ferner können Fixierungen von Punkten und Constraints, z.B. über Abstände zu Abhängigkeiten führen, was die Konstruktion ebenfall überbestimmt macht.

Für die Abstände zwischen Punkten gilt:

```
dx(P1-P2) , dx(P2-P3) -> dx(P1,P3)
dy(P1-P2) , dy(P2-P3) -> dy(P1,P3)
dx(P1-P2) , dy(P1,P2) -> d(P1,P2)
```

Werden Punkte fixiert gilt:

```
Fixx(P1) , Fixx(P2) -> dx(P1,P2)
Fixy(P1) , Fixy(P2) -> dy(P1,P2)
Fixx(P1),Fixy(P1),Fixx(P2),Fixy(P2)-> d(P1,P2)
```

Diese Bedingungen werden durch Matrizen überprüft

```
Md[n,m]=true  <-> d(Pn,Pm)  fixiert
Mdx[n,m]=true <-> dx(Pn,Pm) fixiert
Mdy[n,m]=true <-> dy(Pn,Pm) fixiert

Fixx[n]=true  <-> x Koordinate von Pn fixiert
Fixy[n]=true  <-> y Koordinate von Pn fixiert
```

Für jeden Abstand- Constraint und jede Punkt Fixierung wird die entsprechende Matrix gesetzt.
Für **Mdx** und **Mdy** wird jeweils die Hülle gebildet

```
Md := Md + Mdx*Mdy
Fixx[n] and Fixx[m] -> Mdx[n,m]=true
Fixy[n] and Fixy[m] -> Mdy[n,m]=true
Fixx[n] and Fixy[n] and Fixx[m] and Fixy[m] ->
Md[n,m]=true
```

Ist die entsprechende Position in der Matrix schon gesetzt, ist die Konstruktion überbestimmt

Schließlich muss noch überprüft werden, ob n Constraints von Winkel zwischen 3 Punkten (trechtwinklig oder twinkel3p) ein geschlossenes Polygon mit n Punkten bilden. Dann ist die Konstruktion überbestimmt, da in einem geschlossenen Polygon von n Punkten nur n-1 Winkel unabhängig sind.
Ergibt die Analyse der Freiheitsgrade nicht, dass die Konstruktion überbestimmt ist, wird eine möglichst zyklenfreie Orientierung des Graphen gesucht.

Im ersten Schritt werden alle vorgegeben Fixierungen erfüllt. Dies können sein, Fixierungen der Werte bzw. Koordinaten durch sich durch vorgegebene Maße oder Punkte ergeben oder Fixierungen, die sich aus der Operation ergeben. Soll ein Punkt an eine bestimmte X, Y Koordinate verschoben werden ist er fixiert, d.h. im Zustand `isall`.

Für die Orientierung werden folgende Funktionen benutzt:
`v.sdf(c.fix)` = noch vorhandener Freiheitsgrad in Knoten v für das Constraint c
`c.sdf` = noch vorhandener Freiheitsgrad in Constraint c, das ist der Freiheitsgrad – Anzahl der einlaufenden Kanten.

Bevor jeweils neue Kanten orientiert werden, wird geprüft, welche Kanten zwangsweise orientiert werden müssen:
Für jede Variable v und jeden adjazenten Constraint c, mit noch ungerichteter Kante:

```
if (v.sdf(c.fix)=0)then begin
 if (c.sdf>0)then begin
  kantenrichtung(v,c);change:=true;
 end else begin
  result:=false;exit; //Kante kann nicht orientiert werden
 end;
end;
```

Für jeden Constraint c und jede adjazente Variable v, mit noch ungerichteter Kante:

```
if (c.sdf=0)and(v.sdf(c.fix)>0) then begin
 kantenrichtung(c,v);change:=true;
end else begin
  result:=false;exit;// kante kann nicht orientiert werden
 end;
end;
```

Für alle Constraints c, wo die Anzahl der noch ungerichteten Kanten = der Anzahl der noch vorhandenen Freiheitsgrade ist;

```
j:=c.fnedges(isnone); // Anzahl der noch ungerichteten Kanten
if (j>0)and(j<=c.sdf) then begin
 for j:=0 to c.count-1 do if c.ori[j]=isnone then begin
  v:=c.vars[j];
  kantenrichtung(v,c);change:=true;
 end;
end;
```

Von den Variablen, d.h. den Punkten oder Skalaren, die mit der Operation geändert werden sollen, ausgehend, wird eine Breitensuche im Constraint Graphen durchgeführt. Entsprechend der Reihenfolge, in der die Variablen -Knoten bei der Breitensuche durchlaufen wurden, wird eine Reihenfolge festgelegt.

In dieser Reihenfolge werden die noch ungerichteten Kanten der Variablen auslaufend orientiert. Nach jeder einzelnen Orientierung werden wieder die Zwangsorientierungen geprüft.

Hiermit werden die Constraints von den zu verändernden Variablen ausgehend propagiert.

Beispiel:

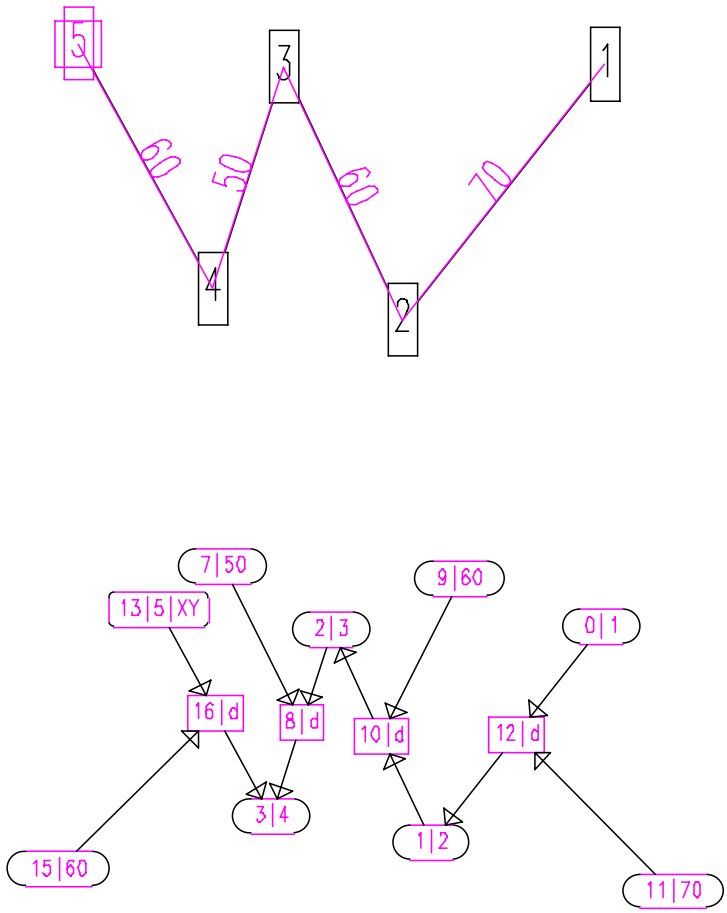


Abbildung 9

Der Knoten 1 wird verschoben. Die Änderung wird bis zum Knoten 4 propagiert, der seine 2. Ortskurve über den Abstand zum Knoten 5 erhält, dessen Position fixiert ist.

Kann der Graph mit diesem Verfahren zyklenfrei orientiert werden, wird diese Orientierung hiernach auch benutzt. Bleiben nach dem obigen Algorithmus noch ungerichtete Kanten übrig, oder enthält die Orientierung Zyklen, wird versucht die Lösung mit der minimalen Anzahl von Zyklen zu finden:

Dazu gibt es 2 Möglichkeiten:

1. Methode: Matching erweitern

Kann eine Kante nicht orientiert werden, so ist an beiden adjazenten Knoten kein Freiheitsgrad mehr vorhanden. Um die Kante zu orientieren, muss also an einem adjazenten Knoten eine einlaufende Kante in eine auslaufende verwandelt werden. Jedoch muss dann der dritte Knoten, der mit dieser Kante adjazent ist, noch einen Freiheitsgrad haben. Ist dies nicht der Fall, muss man an diesem Knoten auch wieder eine Kantenrichtung ändern. Dies wird solange wiederholt, bis man an einen Knoten mit einem noch vorhandenen Freiheitsgrad stößt, oder man gelangt auf dem Weg zu einem Knoten der schon auf dem Weg ist. Hat man einen Knoten mit Freiheitsgrad gefunden, werden alle Kanten auf dem Weg umorientiert, andernfalls wird die Suche abgebrochen. Diese Suche, die auch als Bestimmung eines maximalen Matchings in bipartiten Graphen (McHugh, 1990, Seite 200ff) bekannt ist, wird hier durch die rekursive Funktion `suchen` realisiert.

```
function suchen(a,a1:tany;l:tanylist):boolean;
// Es wird von a1 kommend der Weg über a hinaus verlängert
var v:tvar;
 c:tconstr;
 i:integer;
label 1,99;
begin
result:=false;
if a.mark2 then exit; //schon besucht
a.mark2:=true;
if a is tvar then begin
 v:=a as tvar;c:=a1 as tconstr;
 if v.sdf(c.fix)>0 then goto 99 // a hat noch Freiheitsgrad
 else begin
  1:
  for i:=0 to a.count-1 do if a.ori[i]=isin then begin
   if suchen(a.kanten[i],a,l)then goto 99;
  end;
 end;
 exit;
end else begin
 c:=a as tconstr;
 if c.sdf>0 then goto 99 else goto 1;
end;
99:result:=true;l.add(a);
// l enthält den Weg von hinten nach vorne
end;
```

War das suchen mit einem adjazenten Knoten der noch ungerichteten Kante erfolgreich, werden alle Kanten auf dem Weg umorientiert und dann kann die Kante bei diesem Knoten einlaufend orientiert werden.

Kann bei den beiden adjazenten Knoten, der ungerichteten Kante, kein Weg zu einem Knoten mit noch vorhandenen Freiheitsgrad gefunden werden, so ist das Constraint System überdefiniert und es existiert keine Lösung.

Hierbei wird eine Lösung gefunden, die nicht optimal sein muss, d.h. die evtl. nicht die wenigsten Zyklen hat.

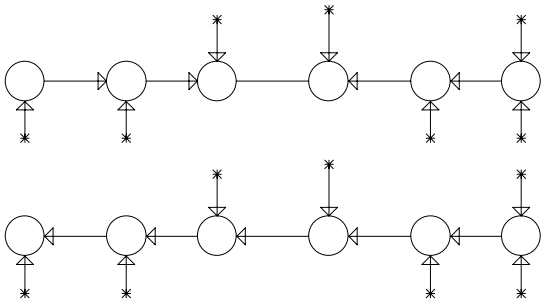


Abbildung 10

2. Methode: Brute Force:
Es werden systematisch alle möglichen Orientierungen untersucht, und die ausgewählt, die am wenigsten Knoten auf einem Zyklus hat.

Der Aufwand ist exponentiell mit der Anzahl der Wahlmöglichkeiten. Dies ist aber weniger als die Anzahl der Kanten, da nach einer Wahl mit dem obigen Algorithmus weitere Kanten orientiert werden. Zu einem werden nach einer Orientierung alle Zwangsorientierungen vorgenommen. Zusätzlich werden für alle Variablen, bei denen noch alle noch ungerichteten Kanten einlaufend orientiert werden können, diese auch als einlaufend orientiert. In der ersten Phase wird dies absichtlich nicht gemacht, da hierdurch die Propagierung in der Reihenfolge der Breitensuche durchbrochen werden kann.
Wird eine zyklenfreie Orientierung gefunden, wird aufgehört, sonst wird die Anzahl der Knoten in den Zyklen ermittelt. Es wird die Lösung ausgewählt, die die minimale Anzahl von Knoten in Zyklen hat. Wird bei der Lösungssuche die bisher beste Anzahl überschritten, wird dieser Weg nicht mehr verfolgt (Branch and bound).

Durch die vorherige Prüfung der Freiheitsgrade ist der Fall, dass keine Lösung existiert, und damit auf jeden Fall, der ganze Lösungsbaum durchlaufen wird, ausgeschlossen.

Evaluierung des Constraint- Graphen

Der nach der Orientierung erhaltene gerichtete Graph, kann nun jedoch Zyklen erhalten. Deswegen werden zunächst die starken Komponenten ermittelt. Hier werden Knoten zwischen denen ein Zyklus existiert, zu einer Komponente

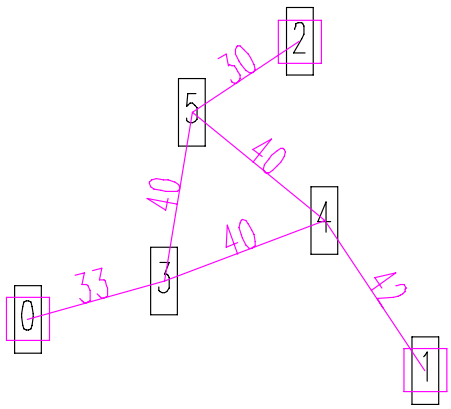
zusammengefasst. Hat der Graph keine Zyklen besteht jede Komponente aus genau einem Knoten.

Die Ermittlung der starken Komponenten erfolgt durch eine rekursive Tiefensuche. Findet man hierbei einen Knoten der bereits auf dem Weg liegt hat man einem Kreis gefunden und die Knoten auf dem Kreis gehören zur gleichen starken Komponente. Der Graph der starken Komponenten bildet einen DAG. Daher können die starken Komponenten in der Reihenfolge der topologischen Sortierung evaluiert werden.

Die Evaluierung einer starken Komponente mit mehr als einem Knoten, ist jedoch nicht sequentiell möglich.
Hier kann ein Iterationsverfahren benutzt werden. Hierbei muss jedoch nicht ein Gleichungssystem über alle Knoten in dem Graph der starken Komponente gelöst werden. Es wird vielmehr eine Variablenmenge V gesucht, die alle Kreise der Komponente dominiert.
Dies nennt man die Feedback-Vertex- Menge F. Gesucht wird eine minimale Menge F. Dieses Problem ist NP-vollständig, es kann jedoch wieder mit rekursiver Tiefensuche eine Näherungslösung gefunden werden. Wählt der Algorithmus für F einen Constraintknoten C aus, wird er ersetzt durch den Variablenknoten V, der auch auf dem Kreis liegt, und der von C aus erreicht wird. Da der Graph bipartit ist, kann en Kreis nicht nur Constraintknoten enthalten.
In vielen Fällen bilden ein oder mehrere skalare Wertknoten bereits eine Feedback-Menge. Ist dies der Fall, kann eine Intervallschachtelung (Regula Falsi) benutzt werden. Dabei wird jeweils getestet, ob die Differenzen zwischen zwei Iterationen verschieden Vorzeichen haben. Wenn ja, wir der Mittelwert zwischen den beiden letzten Iterationen als neue Näherung genommen.

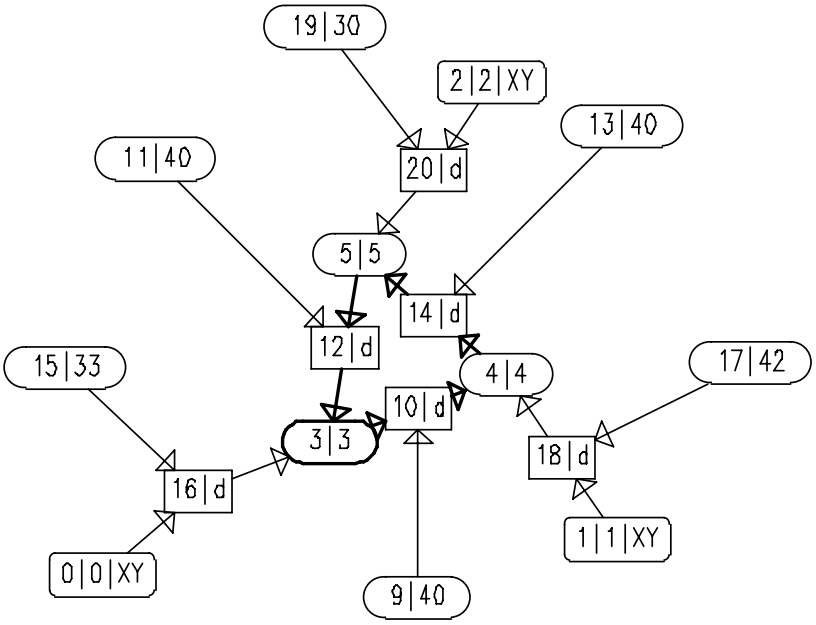
Bei der Iteration werden in einer Schleife die Knoten aus F mit topologischer Sortierung berechnet. Wird dabei ein anderer Knoten aus F oder der gleiche Knoten benötigt, wird der alte Wert genommen. Dies wird solange gemacht, bis sich die Werte der Knoten in F nicht mehr ändern oder eine Maximalzahl von Durchläufen erreicht ist.

Dies funktioniert auch, wenn mehrere Punkte in einer starken Komponente sind, wie in folgendem Beispiel:



Die zugehörige Constraint Graph ist:

C:\cad95\2dsiecke-gra.KOG



Die Knoten 3,4 und 5 mit ihren Abständen müssen gleichzeitig gelöst werden. Wird z.B. der Punkt 2 verschoben oder der Abstand 20 geändert, erhält man für Punkt 5 eine neue Ortskurve und damit neue Koordinaten. Mit diesen Koordinaten wird Punkt 3 geändert und dann Punkt 4. Damit stimmt aber der Abstand zu 5 nicht mehr. Daher wird dieser Zyklus wiederholt bis die Punkte 3,4 und 5 eine stabile Lage bekommen.

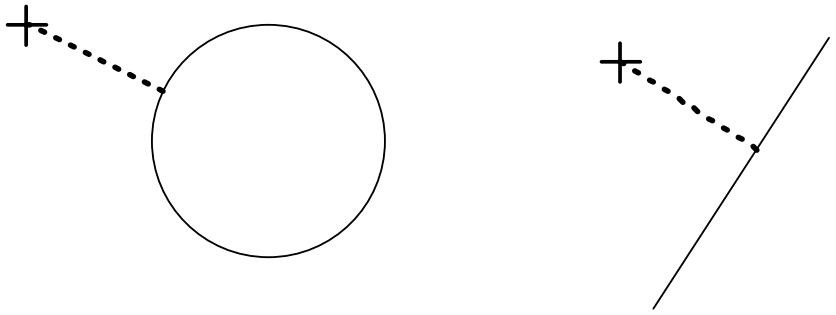
Ist die Evaluierung für den Gesamtgraphen nicht erfolgreich, wird zunächst der längste Kreis in dem gerichteten Graphen ermittelt. Auf diesem Kreis werden alle Kanten umgedreht. Die Zahl der ein- und auslaufenden Kanten bleibt dabei für jeden Knoten gleich. Auf den so geänderten Graphen wird dann noch mal das gleiche Verfahren angewandt.

Bei der Evaluierung des kreisfreien Graphen der starken Komponenten wird für die Komponenten, die nur aus einem Knoten bestehen, jeweils nachdem alle Knoten, von denen Kanten einlaufen berechnet sind, der Knoten berechnet. Dazu wird für den Knoten die Methode `compute` aufgerufen.
Bei den Variablenknoten (tv, tnod, tkreis) wird wie folgt verfahren:

`tv.compute:` keine Aktion. Der Wert ist durch den mit einer einlaufenden Kante verbundenen Constraint-Knoten schon berechnet.
`Tnod.compute:` Durch die mit einer einlaufenden Kante verbundenen Constraint-Knoten sind für den Punktknoten maximal 2 Ortskurven bestimmt. Dies können sein:

- X-Koordinate
- Y-Koordinate
- Linie
- Kreis

Eine Funktion `computexy` berechnet aus den bisherigen Koordinaten und den geforderten Ortskurven die neuen Koordinaten, falls dies möglich ist. Die Berechnung kann z.B. nicht möglich sein, wenn z.B. ein Kreis keinen Schnittpunkt mit der der anderen Ortskurve hat.
Gibt es für einen Punkt nur eine Ortskurve, wird das Lot von der bisherigen Position auf diesen Kreis bzw. diese Linie gebildet.



Für einen Kreis können durch die Constraints, von denen Kanten einlaufen, folgende Bedingungen festgelegt sein:

- Isx: X Koordinate des Mittelpunktes
- Isy: Y Koordinate des Mittelpunktes
- Isc: Kreis auf dem der Mittelpunkt liegt
- Isl: Linie auf der der Mittelpunkt liegt
- Ist: Linie zu der der Kreis tangential ist
- Isp: Punkt der auf dem Kreis liegt
- Istan: Kreis zu dem der Kreis tangential liegt
- Isr: Radius des Kreises

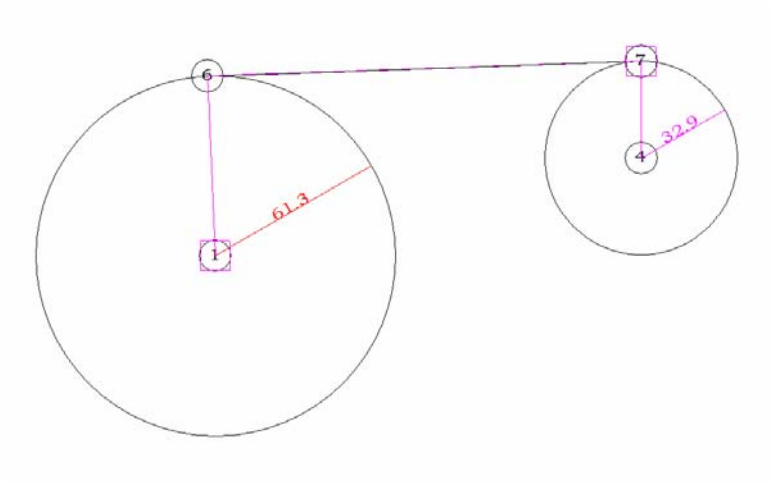
Für ein bis 3 solcher Bedingungen ist jeweils die Berechnung des Kreises in **`tkreis.compute`** festgelegt.

Bei den Constraint- Knoten existiert für jede Klasse eine spezielle Methode. Diese Methoden sollen für einige Klasen beschrieben werden.

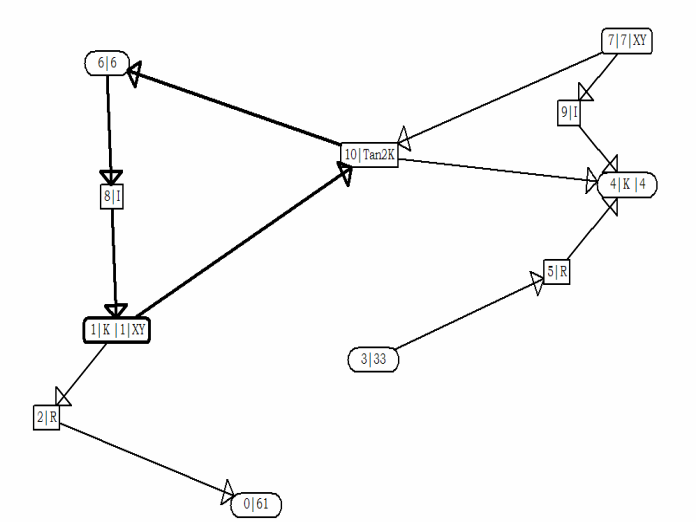
Twaagrecht: Der Knoten hat eine einlaufende Kante. Die Y- Koordinate dieses Punktes bestimmt die Y- Koordinate aller Knoten, der auslaufenden Kanten.
Tsenkrecht: Analog.
Tlinie: Die 2 Knoten der einlaufenden Kanten, bestimmen die Hesse-Normalform (a, b, c) der definierten Linie. Diese Linie wird für die Knoten der auslaufenden Kanten eine Ortskurve.
Tdistanz: Dieser Knoten hat 3 adjazente Variablen- Knoten. Der Wertnoten enthält den Abstand, die beiden anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den beiden Punkten der Abstand berechnet. Man hat dann ein gesteuertes Maß. Der Wert des Maßes wird bei einer Änderung jeweils nachgeführt. Ist er einlaufend, bestimmt er und ein Punkt einen Kreis, auf dem der andere Punkt liegen muss. Dann hat man ein steuerndes Maß.
Tdistanzx, Tdistanzy: Analog.

Twinkel: Dieser Knoten hat 3 adjazente Variablen- Knoten. Der Wertknoten enthält den Winkel, die beiden anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den beiden Punkten der Winkel berechnet. Ist er einlaufend, bestimmt er und ein Punkt einen Linie, auf der der andere Punkt liegen muss.
Twinkel3P: Dieser Knoten hat 4 adjazente Variablen- Knoten. Der Wertknoten enthält den Abstand, die 3 anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den 3 Punkten der Winkel berechnet. Ist er einlaufend, bestimmt er und zwei Punkte eine Ortskurve, auf der der dritte Punkt liegen muss.
Tdistanzlp: Dieser Knoten hat 4 adjazente Variablen- Knoten. Der Wertknoten enthält den Abstand, die 3 anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den 3 Punkten der Abstand berechnet. Ist er einlaufend, bestimmt er und zwei Punkte eine Ortskurve, auf der der dritte Punkt liegen muss.
Trechtwinklig: Analog wie Twinkel3P, jedoch ist der Winkel auf 90° festgelegt.
Tgleichmx: 2 Punkte, die durch einlaufende Kanten verbunden sind, bestimmen die X- Koordinaten der übrigen Punkte.
Tgleichmy: Analog.
Toperator: Aus 2 einlaufenden Wertknoten wird der Wert des anderen Knotens berechnet. Entweder aus 2 Operanden das Ergebnis oder aus dem Ergebnis und einem Operanden den anderen Operanden.
Ttangential: Für den Kreis an der auslaufenden Kante wird der andere Kreis als tangential festgelegt. Für die Kreise wird dann jeweils eine entsprechende Funktion, z.B. Kreis tangential an 3 Kreise, aufgerufen.
Ttangentialkl: Der Knoten ist adjazent zu einem Kreis und 2 Punkten. Sind die Kanten von beiden Punkten einlaufend so ist für den Kreis die Bedingung ist zur Geraden, die durch die Punkte definiert wird, gegeben. Ist einer der Punkte durch eine auslaufende Kante verbunden, so muss dieser Punkt auf der Geraden, die durch die Tangente vom anderen Punkt an den Kreis gebildet wird, liegen
Tradius: Ist die Kanten vom Kreis einlaufend so wird der Radius festgelegt, ist sie auslaufend so bestimmt der Skalar den Radius des Kreises.
Tinzident: Dieser Knoten ist adjazent zu einem Kreis und einem Punkt. Ist die Kante vom Kreis einlaufend, so muss der Punkt auf dem Kreis liegen. Sonst ist für den Kreis die Bedingung isp gegeben.

Liegt ein Zyklus vor, konvergiert die Iteration jedoch häufig nicht. Daher wird versucht für einige häufig vorkommende Fälle eine direkte Lösung zu finden.



Es soll der Punkt 6 in Y-Richtung verschoben werden. Man erhält folgenden Constraint Graphen:



Man hat einen Zykel bestehend aus Kreis, Tangente2K, Punkt, Inzident. Man kann diesen Zykel aufbrechen, indem man Tan2k berechnet. Dabei werden aus dem Kreismittelpunkt 1 und dem Punkt 7 für den Punkt 6 und den Kreis 4 die Ortslinien berechnet.

Literatur

[Berling96] Berling R. 'Eine Constraint-basierte Modellierung für Geometrische Objekte', Dissertation.

[Du, Rosendahl, Berling 1993] Du C, Rosendahl M ,Berling R, 'Variation of Geometry and Parametric Design', Proc. 3rd. international conference on CAD and computer graphics,
Beijing, Aug. 23-26, 1993, pp 400-405,international academic publishers, 1993
Du,Rosendahl,Berling1993

[Fudos, Hoffman, 1993] I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. Technical Report 93-076, Purdue University, Computer Science, 1993.
Correctness Proof of a Geometric Constraint Solver

[Hoffmann89] C.M. Hoffmann. Solid and Geometric Modeling. Morgan Kaufmann, 1989.

[Hsu, Brüderlin 1997] C. Hsu, B. Brüderlin. "A Hybrid Constraint Solver Using Exact and Iterative Geometric Constructions," in CAD Systems Development -- Tools and Methods, D. Roller, P. Brunet, eds., Springer Verlag, 1997.

[Kondo, 1992] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. Computer Aided Design, 24(3):141--147, March 1992.

[Light, Gossard 1982] R. Light and D. Gossard. Modification of geometric models through variational geometry. Computer Aided Design, 14:209--214, July 1982

[Verroust, Schonek, Roller, 1992] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized Computer-aided design. Computer Aided Design, 24 (10):531-540,
October 1992.