Inhaltsverzeichnis

1 Auswahl der Stützpunkte und Gradienten **5**

1.1 Nutzerdokumentation 5

1.1.1 Aufgabenstellung 5

1.1.2 Lösungsverfahren - theoretische Grundlagen 6

1.1.3 Umgebungsbedingungen 9

1.1.4 Beschreibung der Ein- und Ausgabe 9

1.1.5 Robustheit und Fehlermeldungen 15

1.1.6 Fallbeispiele 16

1.2 Modulstruktur der Auswahl 25

1.3 Beschreibung der Module 25

1.3.1 Funktion Auswahl 25

1.3.1.1 Aufgabenstellung und Begründung der Umsetzung . 25

1.3.1.2 Schnittstellen 27

1.3.1.3 Beschreibung der Funktion Auswahl(..) 30

1.3.1.4 Lokale Variablen und ihre Bedeutung 32

1.3.2 Funktion Flaechevek 33

1.3.2.1 Aufgabenstellung und Begründung der Umsetzung . 33

1.3.2.2 Schnittstellen 33

1.3.2.3 Beschreibung der Funktion Flaechevek(..) 35

1.3.2.4 Lokale Variablen und ihre Bedeutung 36

1.3.3 Funktion Vergleich 36

1.3.3.1 Aufgabenstellung und Begründung der Umsetzung . 36

1.3.3.2 Schnittstellen 37

1.3.3.3 Beschreibung der Funktion Vergleich(..) 37

1.3.3.4 Lokale Variablen und ihre Bedeutung 38

1.3.4 Funktion Anfuegen 38

1.3.4.1 Aufgabenstellung und Begründung der Umsetzung . 38

1.3.4.2 Schnittstellen 38

1.3.4.3 Lokale Variablen und ihre Bedeutung 38

1.3.5 Funktionen Octant,Quadrant 39

1.3.5.1 Aufgabenstellung und Begründung der Umsetzung . 39

1.3.5.2 Schnittstellen 39

1.3.5.3 Lokale Variablen und ihre Bedeutung 39

1.3.6 Funktion comp 39

1.3.6.1 Aufgabenstellung und Begründung der Umsetzung . 39

1.3.6.2 Beschreibung der Funktion comp(..) 39

1.4 Quelltext zur Auswahl – `auswahl5.c` 40

Abbildungsverzeichnis

1.1 Lokale Nachbarschaftssuche 5
1.2 Reichweite im Variogramm bzw. im Korrelogramm 7
1.3 Verringerung von Redundanzen durch Sektorensuche 8
1.4 Stützstellensuche in relevanten Gebieten (links) 9
1.5 Rechenzeitverbrauch in Abhängigkeit vom Sortierraster (rechts) . . . 9
1.6 Menü `Modellparameter/Searching parameters` 10
1.7 Logarithmische Darstellung der Stützstellenabstände 11
1.8 Menü `Modellparameter/Searching param./Stützstellen` 12
1.9 Übersicht der Abhängigkeiten im Auswahlalgorithmus 14
1.10 Auswahlbeispiel 1 16
1.11 Auswahlbeispiel 2 17
1.12 Auswahlbeispiel 3 17
1.13 Auswahlbeispiel 4 18
1.14 Auswahlbeispiel 5 18
1.15 Auswahlbeispiel 6 19
1.16 Auswahlbeispiel 7 19
1.17 Auswahlbeispiel 8 20
1.18 Auswahlbeispiel 9 20
1.19 Auswahlbeispiel 10 21
1.20 Suche der absolut nächsten Punkte (Kalilagerstätte) 22
1.21 Suche in Quadranten (Kalilagerstätte) 23
1.22 Suche in Oktanten (Kalilagerstätte) 24
1.23 Rechenzeitverbrauch des Auswahlalgorithmusses 25
1.24 Bezeichnung der Sektoren bei Quadranten und Oktanteneinteilung . 30
1.25 Numerierung und Orientierung des Sortierrasters 34
1.26 Abarbeitungsreihenfolge bei der Suche nach relevanten Flächen . . . 35

Kapitel 1

Auswahl der Stützpunkte und Gradienten bei der Vorhersage

1.1 Nutzerdokumentation

1.1.1 Aufgabenstellung

Eine Grundaufgabe bei der geostatistischen Modellierung ist es, aus einer größeren Menge von Stützstellen, die für eine Vorhersagestelle relevanten Werte herauszusuchen und diese für die Berechnung bereitzustellen. Es können verschiedene Kriterien, nach denen die Auswahl erfolgen soll, formuliert werden. An dieser Stelle sollen nur zwei grundlegende Unterscheidungsmerkmale festgelegt werden:

- Bei der richtungsunabhängigen lokalen Nachbarschaftssuche wird nach den absolut nächsten zur Vorhersagestelle liegenden Punkten gesucht (Abb. 1.1, links).
- Bei der richtungsabhängigen lokalen Nachbarschaftssuche wird nach den nächsten in den Sektoren liegenden Punkten gesucht, wobei die Belegung der Sektoren Vorrang vor der Distanz zur Vorhersagestelle hat (Abb. 1.1, rechts).

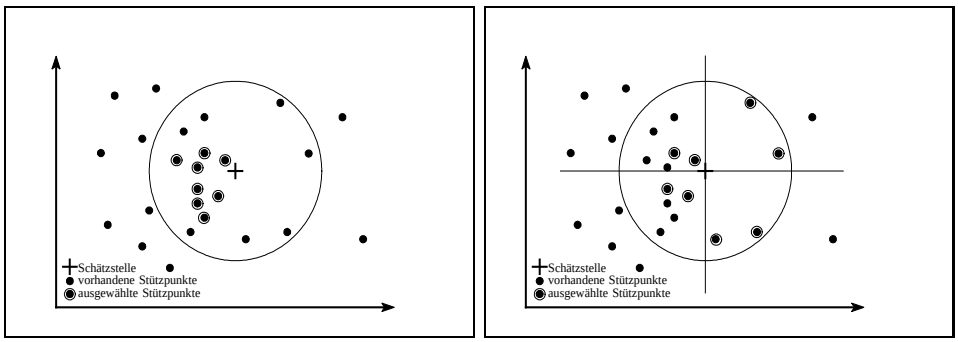

Abbildung 1.1: Lokale Nachbarschaftssuche

Es lassen sich weitere Kriterien finden, diese sollen im vorliegenden Programm jedoch nur Einschränkungen von jeweils einem der oben beschriebenen Unterscheidungsmerkmale bzw. Mischformen aus ihnen sein. Im Abschnitt 1.1.6 S. 16 werden

diese Varianten ausführlicher beschrieben. Bei der praktischen Nutzung findet die Vorhersage in dichten Rastern, die mit einer hohen Anzahl von Schätzstellen verbunden sind ($\geq$ 1000 Punkte), statt. Deshalb steht bei der Implementierung des Suchalgorithmusses die Anforderung nach einer möglichst hohen Effizienz. Das Ergebnis der Auswahlfunktion soll ein eindimensionales Feld sein, dessen Elemente eine Struktur im Sinne der C-Programmierung sind. Das einzelne Strukturelement enthält alle in den Dateien verfügbaren Informationen zur jeweils ausgewählten Stelle.

1.1.2 Lösungsverfahren - theoretische Grundlagen

Die meisten Kriging- und Simulationsmethoden berücksichtigen eine limitierte Anzahl von nächstliegenden Stützpunkten. Ein Grund liegt in der begrenzten Leistung der CPU und des Speichers bei Computern. Das bedeutet:

- Die CPU-Zeit steigt proportional zur dritten Potenz der Datenanzahl, d.h. eine Verdopplung der Daten führt zu einer achtfachen Zunahme der CPU-Zeit.
- Der Speicherplatz der Kovarianzmatrix steigt proportional zur zweiten Potenz der Datenanzahl, d.h. z.B. die doppelte Anzahl von Daten führt zu einer vierfachen Erhöhung des Speicherplatzbedarfs.

Neben der progammtechnischen Seite existiert auch vom geostatistischen Standpunkt eine Notwendigkeit zur Auswahl. Die wichtigste grundlegende theoretische Betrachtung ist die nach der Beziehung zwischen der Annahme des Modells stationärer Zufallsfunktionen und der Relevanz der ausgewählten Punkte. Für die Nutzung von Zufallsfunktionsmodellen ist Stationarität notwendig. Gilt die Stationarität für einen ganzen Bereich der Lagerstätte (bzw. Vorhersagemerkmal), so kann man folglich auch Stationarität (in der Praxis heißt das: Gültigkeit der ermittelten empirischen Kovarianzfunktion) für diesen Teilbereich annehmen. Die Annahme, daß eine partielle Stützpunktauswahl eine Realisierung der stat. Zufallsfunktion ist, ist streng gebunden an eine zweite Annahme; nämlich der, daß Daten zusammengefaßt werden können [IS89].

Beide sind im vorliegenden Programm genutzt worden, sie können jedoch quantitativ nicht überprüft werden. Es sollte aber eine Beurteilung der qualitativen Information möglich sein. Es ist natürlich immer eine subjektive Entscheidung, ob die Stützpunkte einem Homogenitätsbereich zugehören oder nicht. Sie hängt von einem erfolgreichen Literaturstudium und von dem Erfahrungsschatz der zuständigen Personen (z.B. Geologe) ab. Störungen (Faltungen, Verschiebungen) teilen zum Beispiel in Gruppen von Stützpunkten (Homogenitätsbereich). Die Definition von Homogenitätsgrenzen kann wichtiger sein als die Wahl des Schätzverfahrens und sollte bei jeder Schätzung der erste Schritt sein.

Darüber hinaus wäre eine Prüfung der Relevanz der für die Schätzung in einem Punkt herangezogenen Meßstellen an jeder Schätzstelle für eine gewissenhafte Vorhersage angemessen. Es ist leider gängige Praxis, eine unveränderte Suchstrategie über das gesamte Feld anzuwenden. „Black–box“ - Computerprogramme machen es einem besonders einfach, sich einer von Fall-zu-Fall-Bestimmung zu enthalten. Es sollten aber Programme bevorzugt werden, die mehr Interaktion (Voransicht der ausgewählten Stützpunkte, Histogramme etc.) erlauben. Crossvalidationuntersuchungen erweisen sich bei der Entscheidung, ob bestimmte Gebiete eine gesonderte Suchstrategie erfordern, in der Vorbereitung der Schätzung als nützlich.

Die Annahme einer globalen Suchnachbarschaft erfordert die Kenntnis der Kovarianz über die weiteste Distanz zwischen den Daten. In der praktischen Anwendung ist es aber nun so, daß man die Kovarianz meist aus den vorhandenen Meßpunkten

bestimmt. Die Anpassung der stationären Zufallsfunktion kann daher bei Strecken über 1/3 bis 1/2 der Feldesgröße im Allgemeinen als sehr schlecht eingeschätzt werden [DJ92]. Die lokale Nachbarschaftssuche dagegen erfordert keine Kenntnis der Kovarianzen über den Durchmesser des Einwirkungsbereichs hinaus. *„Die Annahme der Stationarität bezieht sich auf das Modell und hat nichts mit der realen Situation zu tun. Von der Perspektive des Modells verbessern zusätzliche Stützpunkte ungeachtet der Distanz zum Schätzpunkt immer die Schätzung. Das muß aber nicht bedeuten, daß das Gleiche von der Perspektive der Realität wahr ist“* [IS89]. Stattdessen ist es oft so, daß durch die Restriktion auf eine geringere Nachbarschaftsbeeinflussung die statistischen Eigenschaften, die theoretisch durch das Modell zu erwarten sind, weniger von den konformen, realen abweichen als bei großem Suchbereich. Je geringer die Anpassung des Modells an die Realität ist, desto weniger erfüllen die aktuellen Schätzer solche Eigenschaften wie Unverzerrtheit und minimale Schätzvarianz. Die

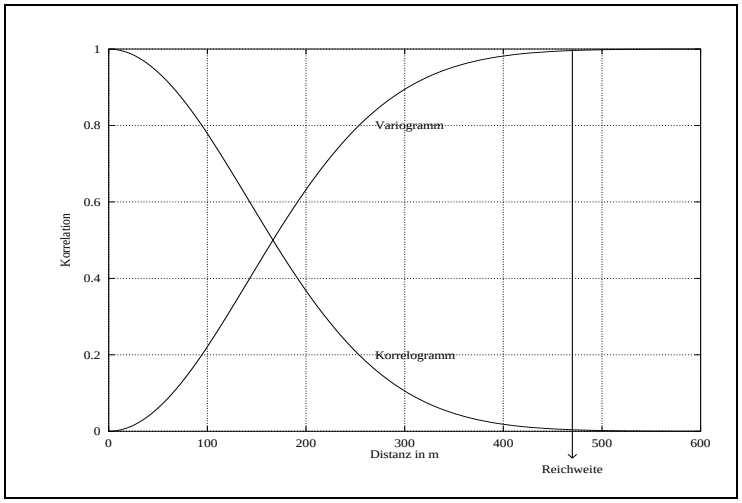

Abbildung 1.2: Reichweite im Variogramm bzw. im Korrelogramm

Benutzung der gewichteten Linearkombination, deren Summe der Gewichte Eins sind, garantieren noch nicht eine Verzerrung vom Betrage Null.

Bei der Wahl des Einwirkungsbereiches sollte außerdem darauf geachtet werden, daß er die Größe der Reichweite des Variogramms bzw. Korrelogramms nicht überschreitet. Dieser Bereich (*engl.: range of variogram*) ist in Abb. 1.2 an der Distanz, von der an die Korrelation konstant bleibt, ablesbar. In der täglichen Anwendung werden Plots von der Schätzung mit möglichst vielen und weniger Punkten gemacht. Wenn sich die Ergebnisse in den unteren Bereichen der Punktanzahl entscheidend (hier sichtbar) verändern, nimmt man an, daß das Optimum erreicht ist [IS89].
Obwohl beim ordinary Kriging, das dem Programm *grille* letztendlich zugrunde liegt, Redundanzen von Stützstellen im Verfahren berücksichtigt werden, soll trotzdem kurz auf diese Problematik eingegangen werden. Wenn zwei oder mehr Stützpunkte in Abhängigkeit von der Entfernung zum Schätzpunkt sehr eng zusammenliegen, ist die Summe der Gewichte beim ordinary Kriging annähernd so groß wie das einer einzelnen Stützstelle in gleicher mittlerer Entfernung [AS88]. Es ist deshalb unter dem Aspekt der widrigen Clustereffekte ratsam, diese Redundanzen durch Zusammenfassen der Punkte zu beseitigen. Darüber hinaus sollten nach [IS89] alle Punkte über der Reichweite (*range of variogram*) richtungsabhängig ebenfalls dieser Prozedur unterzogen werden, wenn der Einwirkungsbereich größer gewählt wurde. Außerdem ist diesem Vorgehen der Vorteil des geringeren Rechenaufwandes involviert.

Eine weitere Methode, Redundanzen zu vermindern, ist die Sektorensuche. In Abb. 1.3 S. 8 ist beispielhaft die Wirkung der richtungsabhängigen Suche in Sektoren demonstriert. In der Praxis erfolgt die Messung häufig in Profilen. Der stark gezeichnete Kreis ist der vereinbarte Einwirkungsbereich, während der schwächere jeweils als Orientierungshilfe gedacht ist. Der Mittelpunkt des Kreises die Schätzstelle dar. Bei der richtungsunabhängigen Suche in Abb. 1.3 a werden alle Punkte in nur einer Profilrichtung berücksichtigt. Die Struktur in dieser Richtung wäre dadurch z.B. ungerechtfertigt überrepräsentiert, wenn der Schätzpunkt neben dem Profil liegen würde. Bei Quadranteneinteilung sollen in jedem Sektor maximal zwei Stützstellen fallen. Diese Maßnahme erzwingt auch Stützstellen, die neben dem gemessenen Profil, das durch die Schätzstelle führt, liegen. In Abb. 1.3 c ist maximal eine Stützstelle pro Oktant zugelassen. Vor allem bei dreidimensionalen Vorhersagen kann durch Sektorensuche verhindert werden, daß alle ausgewählten Stützstellen aus einem Bohrloch stammen.

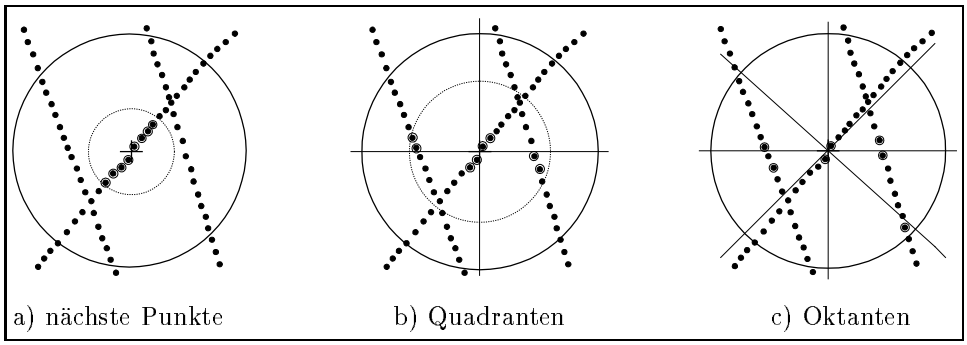

Abbildung 1.3: Verringerung von Redundanzen durch Sektorensuche

Nach den Betrachtungen über die maximale Begrenzung des Einwirkungsbereiches, soll noch kurz die gegenteilige Frage erörtert werden, ob ausreichend Stützstellen im Suchfenster liegen. Der minimale Einwirkungsradius richtet sich nach der Geometrie der Daten im Raum. Er muß groß genug sein, um einige Punkte finden zu können. Bei regulärem bzw. pseudoregulärem Raster der Meßdaten wird der Suchbereich so gewählt, daß immer die nächsten vier Punkte erfaßt werden. Bei einem Grid in Rechtecken wäre das die längere Seite. Bei irregulärem Raster sollte der Suchbereich leicht größer als der durchschnittliche Abstand zwischen den Punkten sein. Er wird nach [IS89] wie folgt berechnet:

$$\bar{s} = \sqrt{\frac{A}{i}} \qquad \begin{array}{l} \bar{s} \ldots\ldots \text{durchschnittlicher Stützpunktabstand} \\ A \text{ . durch Außenpolygon umschlossene Fläche} \\ i \ldots\ldots\ldots\ldots\ldots\ldots \text{Anzahl der Stützpunkte} \end{array} \tag{1.1}$$

Zum Abschluß der theoretischen Betrachtungen über die Auswahl bei der Interpolation seien noch einige Aspekte zum im Programm *grille* realisierten Algorithmus erläutert. Der zu implementierende Suchalgorithmus ist das Kernstück, der mit verschiedenen Parametern mehrfach aufgerufen wird und im Verhältnis zu den übrigen Prozeduren die meiste Rechenzeit in Anspruch nimmt (siehe Abb. 1.23 S. 25).

Es wird eine Art abgewandeltes Hashing verwendet, dieses Verfahren wird in Abschnitt 1.1.4 S. 10 ausführlicher beschrieben. Es ist hier kein reines Hashing, da ständig wie im eben genannten Abschnitt beschrieben die Schlüssel wechseln. Aus den Koordinaten der Schätzstelle und dem Einwirkunksradius werden die Rechteckflächen berechnet, die primär relevante Daten enthalten (Abb. 1.4). Das ist in diesem Falle die Hashfunktion. Mit den Flächennummern lassen sich aus der Tabelle der Indizes der Anfangs- und Endindex der Daten im Feld der sortierten Datensätze

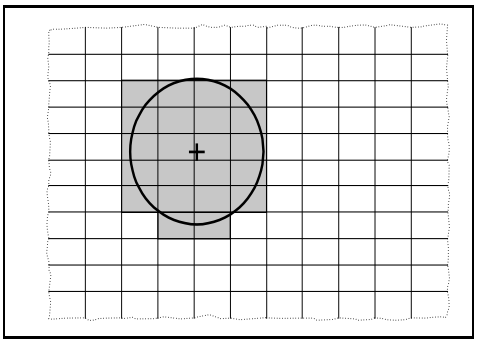

Abbildung 1.4: Stützstellensuche in relevanten Gebieten (links)

Abbildung 1.5: Rechenzeitverbrauch in Abhängigkeit vom Sortierraster (rechts)

ermitteln. Eine große Zahl von Daten wird dadurch eliminiert, weil sie in Flächen definiert sind, die jenseits der Suchgrenzen liegen. Die Wirkung der Vorabbehandlung der Daten auf die Rechenzeit ist in Abhängigkeit vom Sortierrasterabstand in Abb. 1.5 veranschaulicht. Die Zeiten beziehen sich auf einen kontrollierten Ablauf des Programms. Mit Hilfe des UNIX-Kommandos *gprof* wird die Datei *gmon.out* ausgewertet, in die durch Verwendung der C-Compiler-Option *-p* beim Programmablauf das Laufzeitverhalten protokolliert wird. Die „wahre“ absolute Abarbeitungsdauer wird sich deshalb von der in der Grafik zu sehenden unterscheiden. Die Werte entstanden durch Vorhersage an 251001 Schätzstellen. Dabei wurden die jeweils 30 nächsten Punkte aus einem Datenfeld von 233 Datensätzen gesucht. Der Suchradius betrug 500 m. Es ist eine deutliche Abnahme der Rechenzeit mit geringer werdendem Sortierrasterabstand zu verzeichnen. Sortierrasterabstände im Bereich des Suchradius sind am günstigsten. Abstände kleiner als der EWB ergeben keine Rechenzeiteinsparung mehr, da die notwendige Zeit für die Flächenbestimmung wegen der erhöhten Zahl der Flächen zunimmt.

In der Softwarebibliothek „GS-LIB“ wird ein ähnliches Verfahren verwendet, dort wird diese Methode „super block search“ genannt [DJ92].

Erste Aufgabe und Voraussetzung zum Krigen im vorliegenden Programm ist die Sortierung und die Erstellung einer Indexübersicht der Flächen im Menü `Dateiarbeit/Sortierung`. Das Rechteckraster ist unabhängig von den späteren Vorhersagerastern. Sinnvollerweise ist das Flächenraster größer. Die Netzlinienabstände richten sich vor allem nach der Anzahl der Stützpunkte und der Ausdehnung des Vorhersagemerkmals (z.B. Markscheide der Lagerstätte). Bei vielen Meßstellen sollte das Raster engmaschiger sein. Bei wenigen Daten kann das Gitter im Extremfall aus einer Masche bestehen, so daß an jedem Schätzpunkt nur die sequentielle Suche mit allen Stützpunkten stattfindet.

1.1.3 Umgebungsbedingungen

Im Modul *Auswahl* werden keine fremden Bibliotheken bzw. Programme genutzt. Auf Systemaufrufe von der Betriebssystemebene wurde ebenfalls verzichtet.

1.1.4 Beschreibung der Ein- und Ausgabe

Im Programm *grille* werden drei verschiedene Arten von Stützstellen zur Berechnung herangezogen:

1. Stützpunkte des zu schätzenden Merkmals (z.B. Schichtgrenzen, Gehalte),
2. Stützpunkte eines zweiten Merkmals, das mit dem ersten korreliert ist und
3. Gradienten, die die Struktur des Schätzmerkmals beschreiben.

Die Daten müssen in einem programmspezifischen Sortierformat vorliegen. Bevor die eigentliche Schätzung mit `Vorhersage` stattfinden kann, müssen die Ursprungsdaten mit dem Menü `Dateiarbeit/Sortieren` behandelt worden sein. Als Ergebnis dessen liegen Dateien ein und desselben Wortstammes mit der Endung ∼.srt bzw. ∼.inf vor (siehe Abschnitt ??). Die Daten müssen im Menü `Dateiarbeit/Einlesen` eingelesen werden. Jedes Merkmal (Schichtgrenzen sowie Gradienten) stehen in einer eigenen Datei. Im Höchstfall sollen also aus drei Feldern, die für die jeweilige Schätzstelle relevanten Daten herausgesucht werden. Im Programm *grille* ist es möglich, für alle drei Merkmale verschiedene Suchkriterien festzulegen.

Beschreibung des Auswahlmenüs

Im Menü `Modellparameter/Searching Parameters` findet sich das in Abb. 1.6 zu sehende Menü.

Unter den Menüpunkten (1) bis (3) können für die Lagerstättenmerkmale sowie

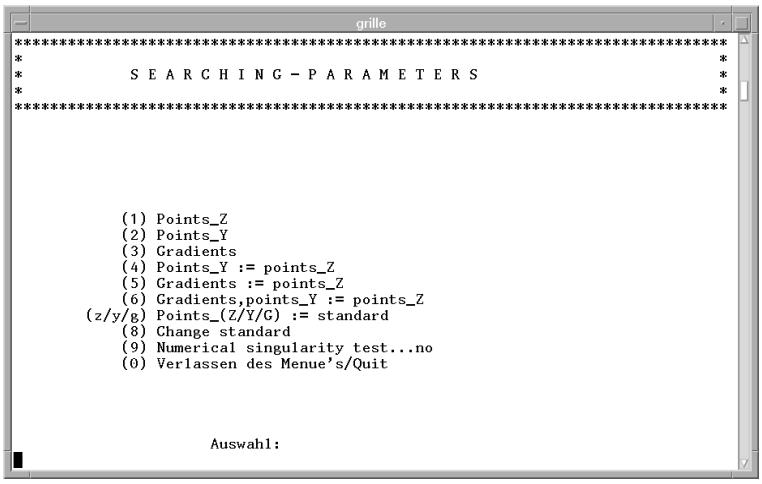

Abbildung 1.6: Menü `(2) Festlegen Modellparameter/ (5) Searching parameters`

für die Gradienten jeweils spezifische Suchkriterien festgelegt werden. Unter diesen Punkten liegt ein weiteres Untermenü. Die Menüpunkte (4) bis (8) erleichtern lediglich die Handhabung des Programms, da Standardeinstellungen mit einem einzigen Tastendruck wiederhergestellt werden bzw. neue Standardeinstellungen definiert werden können. Mit dem Schalter (9) lassen sich Stützstellen in Abhängigkeit von einer zu definierenden Funktion zusammenfassen. Im einzelnen bedeuten die Punkte folgendes:

`(1) Points_Z` Suchkriterien für das zu schätzende Merkmal werden festgelegt (z.B. zu schätzende Schichtgrenze).

`(2) Points_Y` Suchkriterien für kreuzkorreliertes Merkmal (z.B. eine zweite Schichtgrenze, die ähnlich der zu schätzenden schwingt).

`(3) Gradients` Suchkriterien für die vorhandenen Gradienten werden festgelegt.

`(4) Points_Y := Points_Z` Die Suchparameter des Schätzmerkmals werden per Tastendruck an die Suchparameter des kreuzkorrelierten Merkmals übertragen.

(5) `Gradients := Points_Z` Die Suchparameter des Schätzmerkmals werden per Tastendruck an die Suchparameter der Gradienten übertragen.

(6) `Gradients,Points_Y := Points_Z` Die Suchparameter des Schätzmerkmals werden per Tastendruck an die Suchparameter des kreuzkorrelierten Merkmals und der Gradienten übertragen.

(z/y/g) `Points_(Z/Y/G) := standard` unterteilt sich in:

- **(z)** Durch Wahl von **(z)** werden die in der Initialisierungsdatei *.grille.aus* definierten Standardwerte für das Schätzmerkmal zugeordnet.
- **(y)** Durch Wahl von **(y)** werden die in der Initialisierungsdatei *.grille.aus* definierten Standardwerte für das kreuzkorrelierte Merkmal zugeordnet.
- **(g)** Durch Wahl von **(g)** werden die in der Initialisierungsdatei *.grille.aus* definierten Standardwerte für die Gradienten zugeordnet.

(8) `Change standard` Es können die Standardeinstellungen in der Datei *.grille.aus* geändert werden. Durch Wahl von **(8)** werden die augenblickliche Einstellungen für alle Merkmale in die Datei *.grille.aus* geschrieben und somit zum gültigen Standard gemacht.

(9) `Numerical singularity test` Dieser Punkt ist ein Schalter zwischen der Zeichenkette **no** und einem numerischen Wert. Er bietet

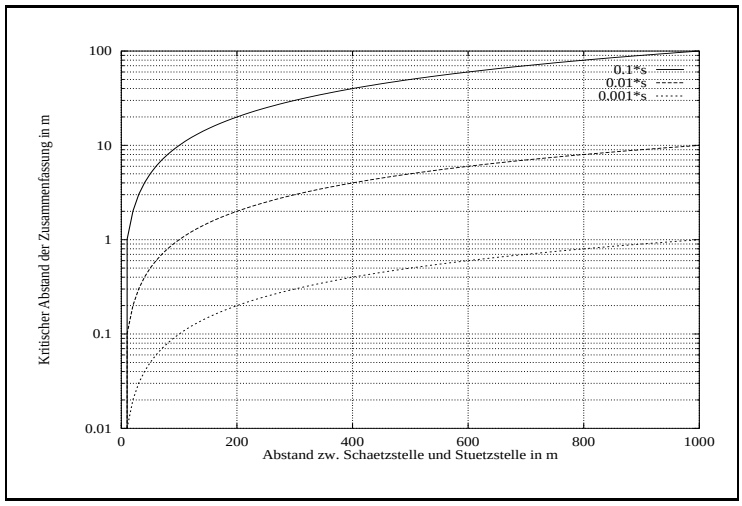

Abbildung 1.7: Logarithmische Darstellung der Stützstellenabstände, ab denen zusammengefaßt wird, in Abhängigkeit vom Schätzstellenabstand

die Möglichkeit, in Abhängigkeit von einer definierten Funktion Stützstellen (inklusive Gradienten) nach erfolgter Auswahl zusammenzufassen. Ein Grund für eine Zusammenfassung (Mittelbildung) von Stützstellen sind numerische Singularitäten bzw. Redundanzen im zu lösenden Gleichungssystem durch zu eng beieinanderliegende Punkte (ausführlich in Abschnitt 1.1.2). In diesem Fall werden die Punkte in Abhängigkeit von der Entfernung zum Stützpunkt gemittelt. Der numerische Wert von **(9)** ist ein Proportionalfaktor in dieser Funktion. In der jetzigen Form bedeutet das, wenn

$$s_{ij} < PF \bullet s_0 \qquad \begin{array}{l} s_{ij} \ldots\ldots\ldots\ldots\ldots\ldots \text{Stützstellenabstand} \\ s_0 \text{ . Abstand zw. Schätzstelle und den Stützstellen} \\ PF \ldots\ldots\ldots\ldots\ldots\ldots \text{Proportionalfaktor} \end{array} \quad (1.2)$$

ist, dann werden die Stützpunkte bzw. Gradienten zusammengefaßt. Je größer PF ist, desto weiter auseinanderliegende Punkte werden zusammengefaßt. In Abb. 1.7 ist eine Kurvenschar für verschiedene PF dargestellt. Die Ordinate ist logarithmisch eingeteilt. Steht der Schalter auf `no` wird dieser Algorithmus nicht abgearbeitet. Die Entscheidung gilt in jedem Fall für alle Merkmale, so sie in die Berechnung eingehen.

`(0) Quit/ESC` Das Menü kann mit `(0)` bzw. Esc verlassen werden.

Beschreibung der möglichen Suchkriterien
Das in Abb. 1.8 dargestellte Menü `P O I N T__Z - S E A R C H - P A R A M E T E R` ist hinter den übergeordneten Menüpunkten `(1) Points_Z`, `(2) Points_Y` und `(3) Gradients` verborgen.
Für alle drei Merkmale können die gleichen Suchkriterien mit jeweils unterschied-

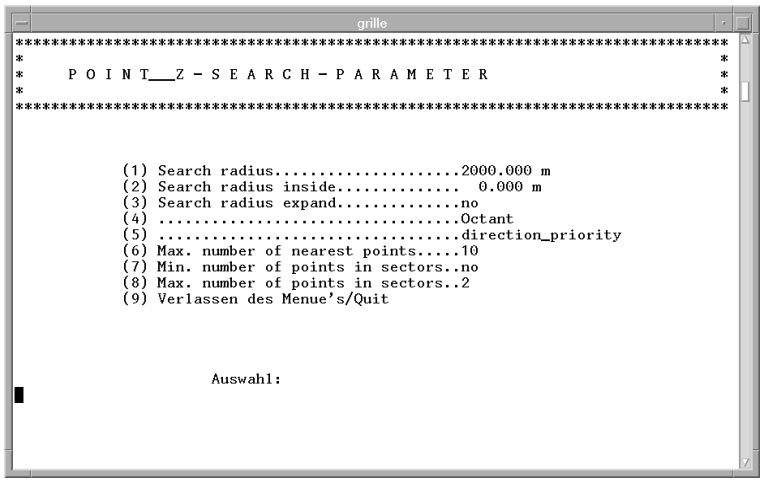

Abbildung 1.8: Menü `(2) Festlegen Modellparameter/ (5) Searching parameters/ (1) Points_Z`

lichen Werten belegt werden. Das im folgenden Beschriebene trifft demzufolge für alle Merkmale zu. Mit den Menüpunkten `(1)` bis `(8)` läßt sich die Suchstrategie steuern. An dieser Stelle werden nur die möglichen Werte, die die einzelnen Steuerungsparameter annehmen können beschrieben. Das Zusammenwirken und die Abhängigkeiten werden in Abschnitt 1.1.6 mit Fallbeispielen ausführlich erläutert.

`(1) Search radius` Das ist der Radius eines Kreises, innerhalb dessen nach Stützstellen gesucht wird. Es ist in jedem Fall ein in Meter einzugebener numerischer Wert, und er muß größer als Null sein. Punkte außerhalb dieses Suchkreises werden nicht zur Schätzung herangezogen.

`(2) Search radius inside` Mit diesem Radius kann ein Einwirkungsring, innerhalb dessen die relevanten Stützstellen liegen sollen, definiert werden. Er kann numerische Werte von 0.0 m bis zu einem Wert, der kleiner als der „eigentliche“ äußere Suchradius ist, annehmen.

`(3) Search radius expand` Durch ihn ist es möglich, den äußeren Suchradius zu erweitern, wenn im durch `(1)` festgelegten `Search radius` keine Stützstelle gefunden wurde. Es wird solange innerhalb des `search radius expand` gesucht bis der allernächste gefunden bzw. bis in den Sektoren der durch `(7) Min. number of points in sectors` festgelegte Wert erreicht ist. Der Wert

von `Search radius expand` ist `no` oder ein numerischer Wert in Meter, der größer als `Search radius` sein muß.

`(4)` Dieser Schalter hat die Werte:

`without sectors` Es werden die in Menü `(6)` vereinbarten nächsten, in unmittelbarer Nachbarschaft liegenden Stützstellen unabhängig davon, in welcher Richtung sie liegen, herausgesucht.

`Quadrant`

`Octant` Es wird richtungsabhängig in Quadranten und Oktanten entsprechend der in `(7)` und `(8)` gewählten Werte gesucht.

`(5)` Der Schalter hat die Werte:

`direction_priority` Es wird unabhängig von der Richtung bzw. richtungsabhängig in Sektoren gesucht; jenachdem wie der Schalter `(4)` steht. Das sollte die Standardeinstellung sein.

`mixed_priority` Diese Schalterstellung bedeutet, daß zunächst die nächsten Punkte richtungsunabhängig mit dem Wert von Menüpunkt `(6)` gesucht werden. Danach erfolgt eine Prüfung, ob alle Sektoren, mit der Mindestanzahl aus `(7) Min. number of points in sectors` belegt sind. Das bedeutet, daß unter `(4)` der Schalter auf `Quadrant` bzw. `Octant` stehen muß, obwohl zuerst nach den nächsten Stellen richtungsunabhängig gesucht wurde. Wenn nach der Belegungsprüfung alle Sektoren ausreichend besetzt sind, bricht der Suchalgorithmus ab. Wenn ein oder mehrere Sektoren unterbelegt sind, werden nur diese Sektoren bis zur Minimalbelegung nach `(7)` mit Stützstellen gefüllt. Notfalls kann eine Bereichserweiterung mit `Search radius expand` erfolgen.

`(6) Max. number of nearest points` Durch diese Variable wird die Stützstellenanzahl der lokalen Nachbarschaftssuche auf diesen numerischen Wert begrenzt.

`(7) Min. number of points in sectors` Bei richtungsabhängiger Suche in Sektoren kann eine minimale Anzahl von Stützstellen festgelegt werden, die auf jeden Fall in einem Sektor vorhanden sein müssen. Werden zuwenig Stützstellen in einem oder mehreren Sektoren gefunden, erfolgt an dieser Stelle keine Vorhersage bzw. es kann durch Bereichserweiterung mit dem Menüpunkt `(3) Search radius expand` solange gesucht werden, bis innerhalb dieses neuen Bereiches die minimale Stützstellenanzahl erreicht ist. Der Wert kann auf `no` bzw. einem numerischen Wert kleiner als der Wert von `Max. number of nearest points` stehen.

`(8) Max. number of points in sectors` Dies ist die maximale Anzahl an nächstgelegenen Stützstellen je Sektor. Er ist in jedem Fall ein numerischer Wert größer Null.

`Quit/ESC` Verlassen des Menüs mit `(9)` oder Esc.

Mit den beschriebenen Schaltern ist es möglich, eine Vielzahl von Suchstrategien zu kombinieren. Die hierarchische Struktur und die Abhängigkeiten der Schalter untereinander sind in Abb. 1.9 dargestellt. Sie soll hier nicht näher erläutert werden, da sie vor allem als Hilfe bei der Arbeit gedacht ist und die Zusammenhänge sich während dieser erschließen werden. Logische Fehleingaben werden vom Programm getestet, wenn mit dem Schalter `(9)` das Menü `P O I N T_Z - S E A R C H - P A R A M E T E R` verlassen werden soll. Es erscheint eine Fehlermeldung, die mit einem Tastendruck bestätigt werden muß. Man bleibt solange im selben Menü, bis alle Fehleingaben entsprechend den Vorgaben beseitigt worden sind. Anwendungsbeispiele zu einzelnen Suchvarianten werden in Abschnitt 1.1.6 S. 16 vorgestellt.

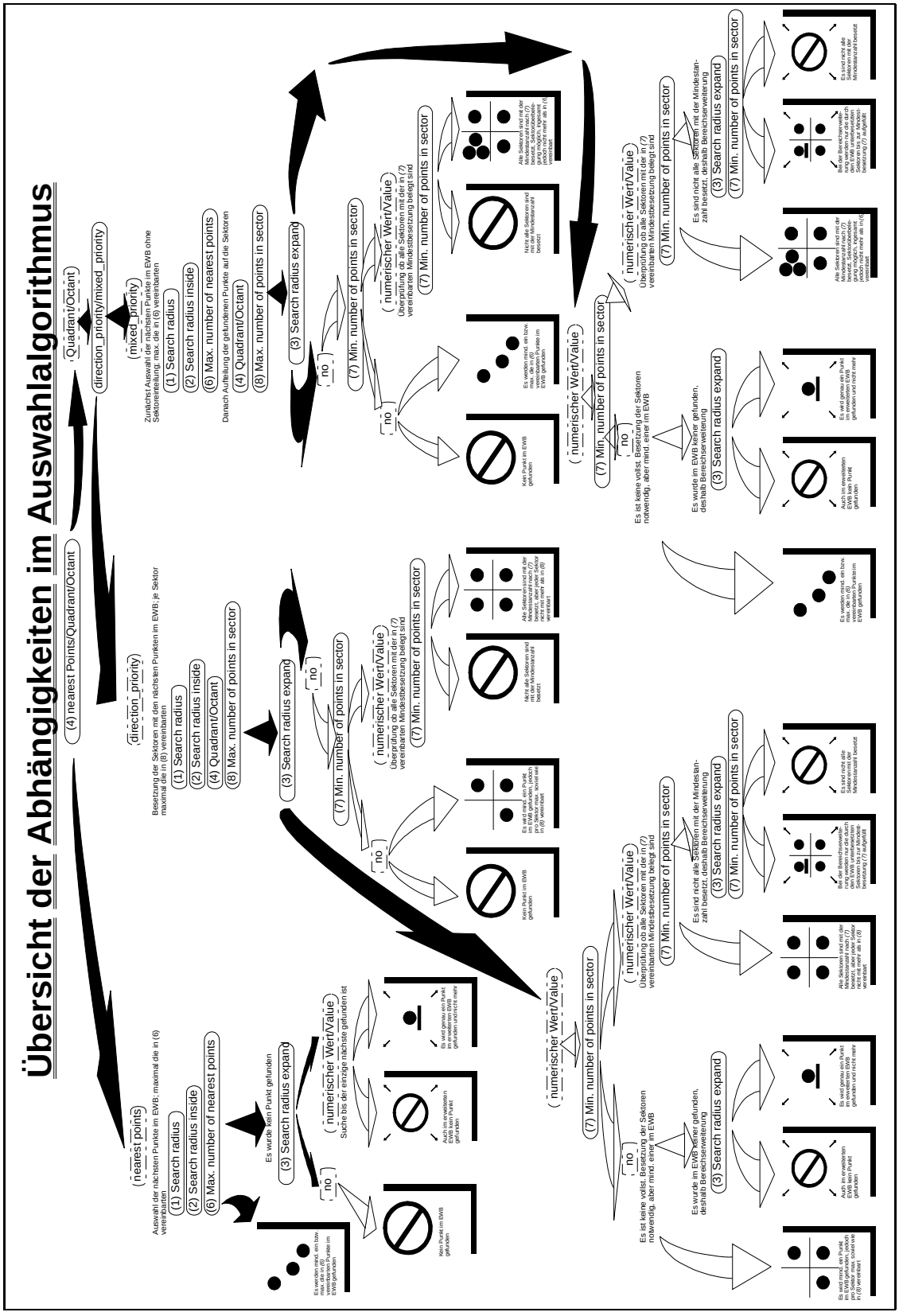

Abbildung 1.9: Übersicht der Abhängigkeiten im Auswahlalgorithmus

1.1.5 Robustheit und Fehlermeldungen

Mögliche Fehleingaben werden unmittelbar nach der Eingabe überprüft. Sobald man das Menü `P O I N T_Z - S E A R C H - P A R A M E T E R` über `(9)/ESC` verlassen will, werden die folgenden Tests nacheinander durchgeführt und ergeben die entsprechenden Fehlermeldungen. Jede Fehlermeldung muß mit einem beliebigen Tastendruck bestätigt werden. Fehlermeldungen entstehen bei der Verneinung der folgenden Fragen:

1. `Search radius` > `Search radius inside ?`
   Fehlermeldung: *Search radius inside too large!*
   Beseitigung: `Search radius inside` verkleinern bzw. `Search radius` ver-. größern

2. `Search radius expand` > `Search radius ?`
   Fehlermeldung: *Search radius expand too small!*
   Beseitigung: `Search radius` verkleinern bzw. `Search radius expand`. vergrößern

3. `Search radius` > Null ?
   Fehlermeldung: *Search radius must be greater than zero!*
   Beseitigung: `Search radius` muß größer als Null und positiv sein.

4. `Max. number of nearest points` > Null ?
   Fehlermeldung: *Maximum number of nearest points must be greater than zero!*
   Beseitigung: Eine ganze Zahl größer als Null muß eingegeben werden.

5. `Max. number of nearest points` ≤ programmbedingte Höchstanzahl an Stützstellen ?
   Fehlermeldung: *Maximum number of nearest points must be lower than wert*
   Beseitigung: Eine kleinere als die als *wert* angebene ganze Zahl eintippen.

6. `Min. number of points in sector` ≥ Null ?
   Fehlermeldung: *Minimum number of points per Sector must be zero or greater!*
   Beseitigung: Eine Null bzw. eine positive ganze Zahl, die kleiner als `Max. number of points in sector` ist, eingeben.

7. `Min. number of points in sector` ≤ `Max. number of points in sector ?`
   Fehlermeldung: *Minimum number of points per sector must be lower than wert*
   Beseitigung: Eine kleinere als die als *wert* angebene ganze Zahl eintippen.

8. `Max. number of points in sector` > Null ?
   Fehlermeldung: *Maximum number of points per Sector must be greater than zero!*
   Beseitigung: Eine ganze Zahl größer als Null muß eingegeben werden.

9. `Max. number of points in sector` ≤ programmbedingte Höchstanzahl an Stützstellen per Sektor ?
   Fehlermeldung: *Maximum number of points per sector must be lower than wert*
   Beseitigung: Eine kleinere als die als *wert* angebene ganze Zahl eintippen.

Neben dem unmittelbaren Abfangen von Fehleingaben sind im Programm auch Tests während der Abarbeitung des Suchalgorithmus eingebaut. Es sollen dabei vor allem Fehler erkannt werden, deren Ursache noch nicht bekannt ist und deshalb im Vorfeld noch nicht berücksichtigt werden konnten. Wenn nach der Prüfung der Sektorzugehörigkeit kein entsprechender Sektor gefunden wurde, erscheint die Meldung: *„Warning: It's not possible to assign a point to a sector!“* Bei der Berechnung der

relevanten, im Suchbereich liegenden Flächen des Sortierrasters werden die Flächennummern zurückgegeben. Mit diesen Flächennummern wird direkt auf die Anfangs- und Endindizes der rastersortierten Datensätze zugegriffen. Zusätzlich steht über den Indizes zur Kontrolle die Flächennummer. Die in der Berechnung ermittelte Flächennummer und die im Indexfeld stehende werden miteinander auf Deckungsgleichheit verglichen. Sollte sie ungleich sein, wird die Fehlermeldung: *„Warning: Here is the computed number wert not equal to the index number wert!"* Wenn eine Stützstelle nicht in dem durch die vorherige Sortierung in `Dateiarbeit/Sortierung` erzeugten Raster (Abb. 6) liegt, erscheint die Warnung: *„Warning: HW:...RW:... point don't lie in sort grid!"* Es werden in diesem Fall alle Datensätze zur weiteren Suche herangezogen. Meist liegt jedoch ein Eingabefehler bei der Festlegung der Koordinaten für die Schätzstellen vor, die im Menü `Vorhersage/ Einzel-, Profil- oder Rastervorhersage` gemacht wurden. Die Ausgabe von Warnungen wird durch ein akustisches Signal begleitet, und außerdem wird die Abarbeitung des Programms für vier Sekunden unterbrochen. Danach geht das Programm ohne Interaktion selbständig weiter.

1.1.6 Fallbeispiele

Anhand von konstruierten Beispielen und einer gut erkundeten Kalilagerstätte sollen die Wirkungsweise sowie die Abhängigkeitsverhältnisse der Auswahloptionen im Menü `Modellparameter/Searching parameters/Points_*` zum besseren Verständnis illustriert werden. Die in den folgenden Beispielen stark hervorgehobenen Optionswerte sind für das jeweils aktuelle Beispiel besonders relevant.

Beispiel 1 - nächste Punkte

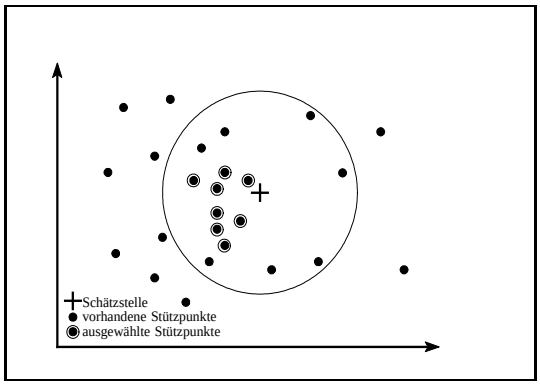

Abbildung 1.10: Auswahlbeispiel 1

Hier sollen die bezüglich der Schätzstelle nächstliegenden acht Stützstellen im Suchbereich gefunden werden. Es werden keine Redundanzen ausgeschalten.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. no
(4)................................... nearest points
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 0
(8) Max. number of points in sectors. 5
```

Beispiel 2 - nächste Punkte mit Bereichserweiterung

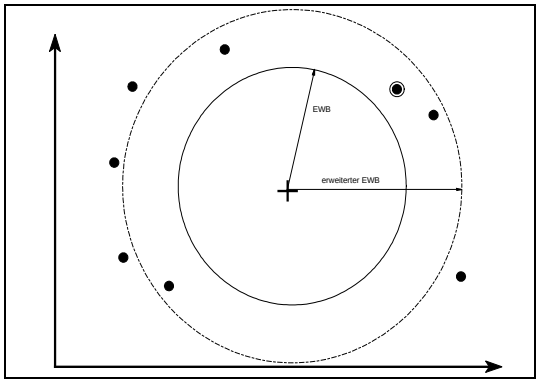

Abbildung 1.11: Auswahlbeispiel 2

Soll in jedem Fall eine Schätzung an den durch Koordinaten angegeben Stellen stattfinden, so muß ein genügend großer Suchbereich festgelegt werden, damit immer wenigstens eine Stützstelle gefunden wird. Wenn keine Stützstelle im Suchbereich gefunden wurde, wird die nächste im erweiterten Suchbereich gesucht. Diese Option ist nur dann sinnvoll, wenn wegen der ermittelten Modellparameter ein kleiner Suchbereich festgelegt werden muß. Ist an manchen Orten zu dünn beprobt worden, würden „Löcher“ bei der Vorhersage entstehen.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. 250 m
(4)................................... nearest points
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 0
(8) Max. number of points in sectors. 5
```

Beispiel 3 - Richtungsabhängige Suche

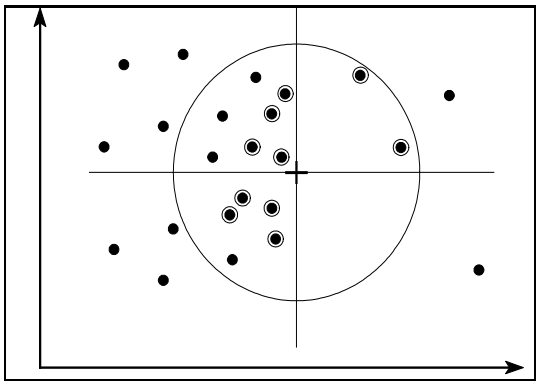

Abbildung 1.12: Auswahlbeispiel 3

Bei der richtungsabhängigen Suche in Oktanten oder Quadranten werden in jedem Sektor die nächsten Punkte bis zu einer Maximalanzahl gesucht. In dieser Konstellation können auch einzelne Sektoren unter- bzw. unbesetzt bleiben. Ist kein Sektor mit mindestens einer Stützstelle besetzt, erfolgt keine Schätzung.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. no
(4)................................... Quadrant
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 0
(8) Max. number of points in sectors. 4
```

Beispiel 4 - Richtungsabhängige Suche mit Mindestbesetzung

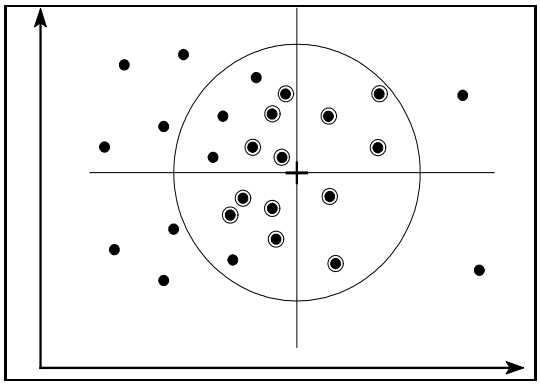

Abbildung 1.13: Auswahlbeispiel 4

Um z.B. eine Extrapolation zu verhindern, kann man eine Mindestbesetzung jedes Sektors mit Zwei vereinbart werden. Ist ein Sektor unterbelegt, findet keine Schätzung statt.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. no
(4)................................... Quadrant
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 2
(8) Max. number of points in sectors. 4
```

Beispiel 5 - Richtungspriorität mit Mindestbesetzung und Bereichserweiterung

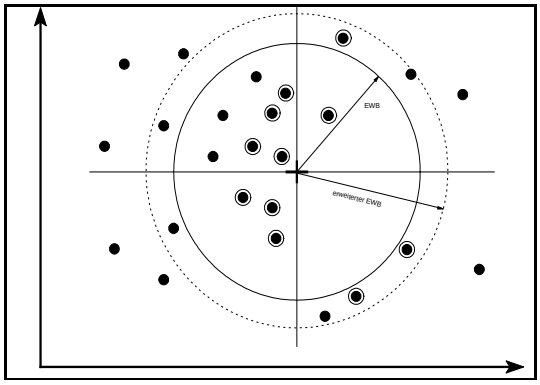

Abbildung 1.14: Auswahlbeispiel 5

Um die in Beispiel 2 erwähnten „Löcher“ auch für Sektorensuche zu vermeiden, kann auch hier eine Bereichserweiterung in Betracht gezogen werden. Die Suche in der Bereichserweiterung erfolgt nur in den Sektoren, in denen nicht die Mindestanzahl im Suchbereich erreicht wurde. Es wird nur solange in der Erweiterung gesucht, bis die Mindestanzahl an nächsten Punkten gefunden wurde. Dabei zählen die schon im Suchbereich gefundenen mit.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. 250 m
(4)................................... Quadrant
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 2
(8) Max. number of points in sectors. 4
```

Beispiel 6 - Richtungspriorität mit Mindestbesetzung und Bereichserweiterung

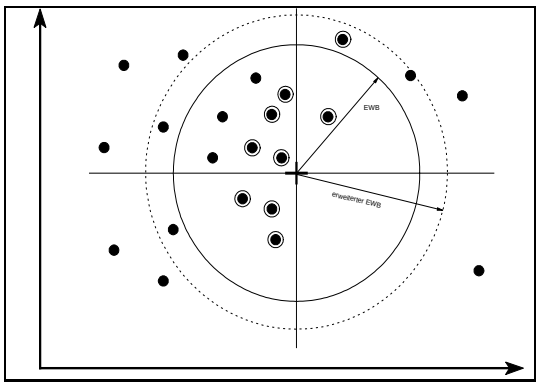

Abbildung 1.15: Auswahlbeispiel 6

In diesem Beispiel findet durch die Forderung nach Mindestbesetzung und dem Nichtvorhandensein von Stützstellen im rechten unteren Sektor selbst im erweiterten EWB keine Schätzung statt.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. 250 m
(4)................................... Quadrant
(5)................................... direction_priority
(6) Max. number of nearest points.... 8
(7) Min. number of points in sectors. 2
(8) Max. number of points in sectors. 4
```

Beispiel 7 - Gemischte Priorität zwischen Distanz und Richtung

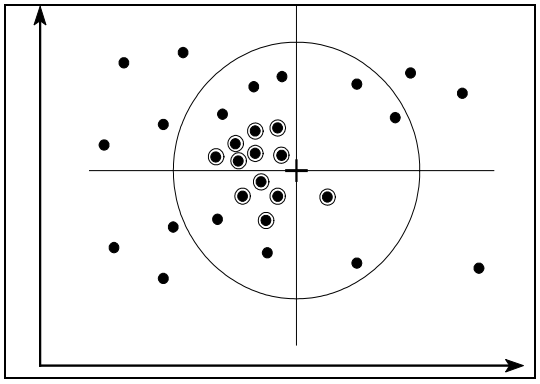

Abbildung 1.16: Auswahlbeispiel 7

Einen Kompromiß aus richtungsunabhängiger Suche und Richtungsabhängigkeit stellen die folgenden Beispiele dar. Hier wird richtungsunabhängig nach den nächsten 12 Stützstellen gesucht und da keine Mindestbelegung der vorerst unberücksichtigten Sektoren gefordert ist, entspricht dieser Fall dem Beispiel 1.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. no
(4)................................... Quadrant
(5)................................... mixed_priority
(6) Max. number of nearest points.... 12
(7) Min. number of points in sectors. 0
(8) Max. number of points in sectors. 4
```

Beispiel 8 - Gemischte Priorität zwischen Distanz und Richtung sowie Mindestbesetzung

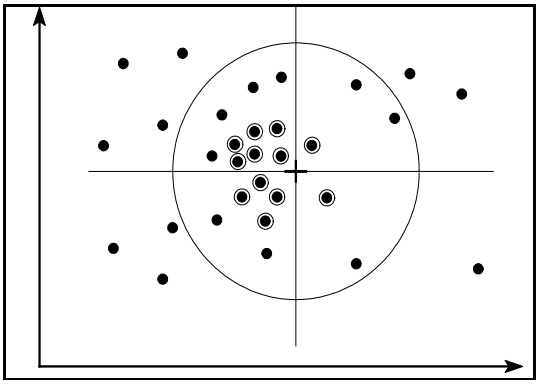

Abbildung 1.17: Auswahlbeispiel 8

Dieser Fall unterscheidet sich vom vorhergehenden lediglich in der Forderung nach der Mindestsektorbesetzung von Eins. Wenn ein oder mehrere Sektoren unterbelegt sind, findet keine Schätzung statt.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. no
(4)................................... Quadrant
(5)................................... mixed_priority
(6) Max. number of nearest points.... 12
(7) Min. number of points in sectors. 1
(8) Max. number of points in sectors. 4
```

Beispiel 9 - Gemischte Priorität zwischen Distanz und Richtung sowie Bereichserweiterung

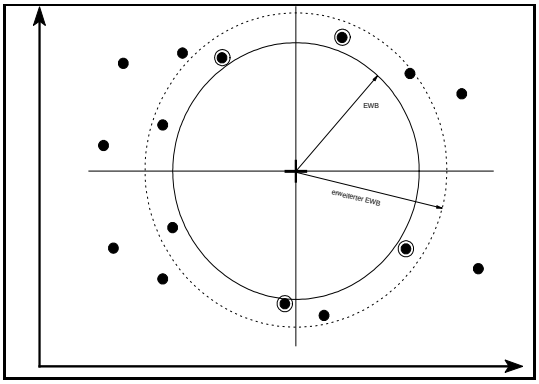

Abbildung 1.18: Auswahlbeispiel 9

Wenn im Suchradius keine Stützstelle gefunden wurde, wird im erweiterten Suchbereich in jedem Sektor die geforderte Mindestanzahl gesucht. Ist keine Mindestbelegung vereinbart, wird die einzige nächste Stelle gesucht.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. 250 m
(4)................................... Quadrant
(5)................................... mixed_priority
(6) Max. number of nearest points.... 12
(7) Min. number of points in sectors. 1
(8) Max. number of points in sectors. 4
```

Beispiel 10 - Gemischte Priorität zwischen Distanz und Richtung sowie Bereichserweiterung

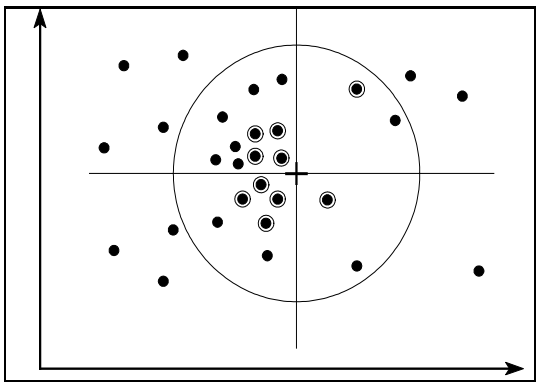

Abbildung 1.19: Auswahlbeispiel 10

Wenn in diesem Falle keine Bereichserweiterung festgelegt wäre, würde keine Vorhersage stattfinden. In der ersten Phase werden die nächsten 12 Stützstellen gesucht und danach getestet, ob alle Sektoren belegt sind. Die nächsten 12 Stützstellen sind in Abb. 1.16 S. 19 gekennzeichnet. Der rechte obere Sektor wäre also unterbelegt. Das Ergebnis der Bereichserweiterung sind mit höchstens der Maximalzahl gefüllte Sektoren. Die Belegung kann aber auch zwischen dem Minimum und dem Maximum für jeden Sektor schwanken. Es werden nur die Unterbelegten bis zur Mindestzahl gefüllt.

```
(1) Search radius..................... 200 m
(2) Search radius inside.............. 0 m
(3) Search radius expand.............. 250 m
(4)................................... Quadrant
(5)................................... mixed_priority
(6) Max. number of nearest points.... 12
(7) Min. number of points in sectors. 1
(8) Max. number of points in sectors. 4
```

Nach den soeben beschriebenen synthetischen Fällen soll am Beispiel eines Ausschnittes aus einer real existierenden Kalilagerstätte die verschiedener Auswahlstrategien illustriert werden. Die Isolinienendarstellungen des Liegenden in den Abb. 1.20 bis 1.22 sind durch Co-Kriging erzeugt worden. Das kreuzkorrelierte Merkmal ist das Hangende der Lagerstätte, das durch, in den Abbildungen mit Dreiecken signalisierten, Bohrungen beschrieben wird. Im Liegenden werden nur neun Stützstellen als bekannt vorausgesetzt. Auf der jeweils rechten Seite sieht man Plots der Differenzen zur Vergleichsfläche. Diese ist vorher mit Hilfe des Ordinary Kriging unter Einbeziehung aller bekannten Liegendstützstellen (an denselben Stellen wie im Hangenden) hergestellt worden. Die verschiedenen Co-Krigingvorhersagen sind durch die Variation der Suchstrategie entstanden. Die Suchstrategie ist über das gesamte Gebiet konstant geblieben. Die Suche wurde so gestaltet, daß immer 32 Stützpunkte an jeder Schätzstelle ausgewählt werden:

Abb. 1.20 es werden die absolut nächsten 32 Stützstellen ausgewählt

Abb. 1.21 in den Quadranten werden die jeweils nächsten 8 Stützstellen ausgewählt

Abb. 1.22 in den Oktanten werden die jeweils nächsten 4 Stützstellen ausgewählt

Beim Vergleich der Isoliniendarstellungen stellt sich heraus, daß sich allein durch die Anordnung der ausgewählten Stützstellen die Schätzergebnisse im 10-m-Bereich variieren. Dabei ist für die Qualität des Gesamtergebnisses keine Methode zu favourisieren. Man sieht anhand der Differenzen-Plots, daß sich in den verschiedenen Teilgebieten auch unterschiedliche Suchstrategien als die besseren im Sinne der Sollfläche herausstellen. Das bestätigt die Aussage aus Abschnitt 1.1.2, daß für eine sorgfältige Vorhersage an jeder Schätzstelle eine eigene, angepaßte Suchstrategie angemessen ist. Es gibt im Allgemeinen keine optimale Lösung für alle Teilgebiete in

einer bewegten Lagerstätte, sondern nur einen Kompromiß, der mehr oder weniger gut ausfällt.

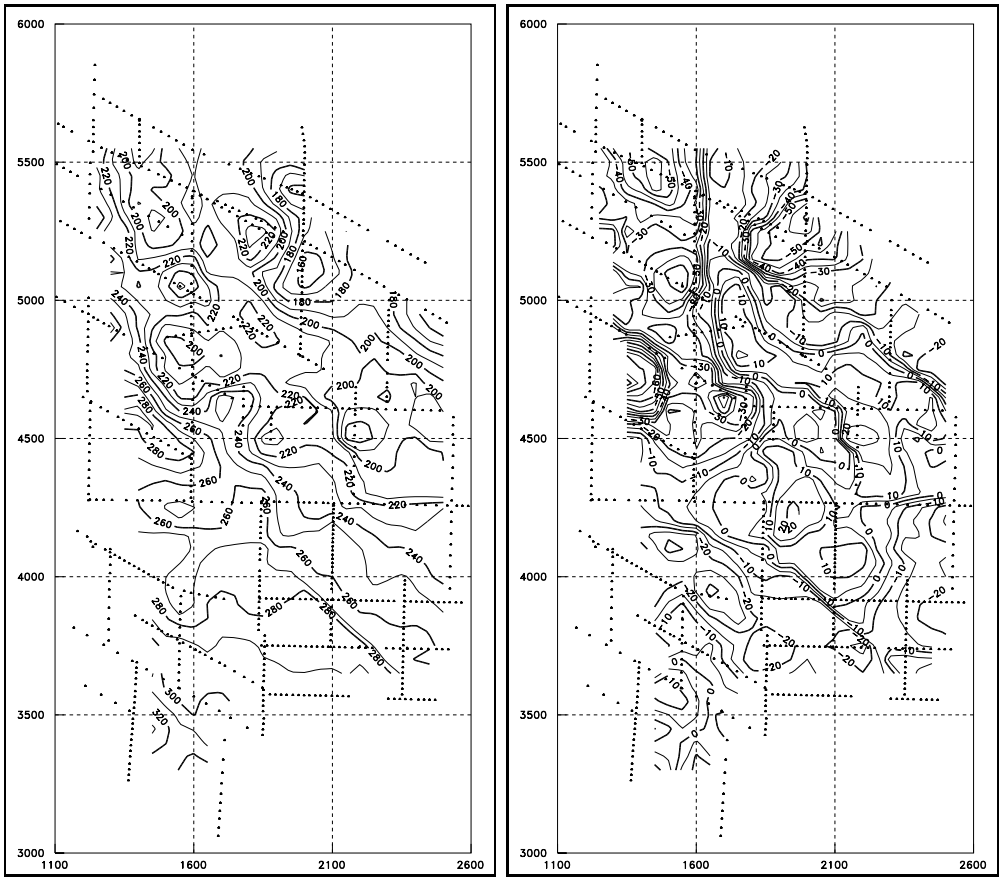

Abbildung 1.20: Suche der absolut nächsten Punkte (Kalilagerstätte)

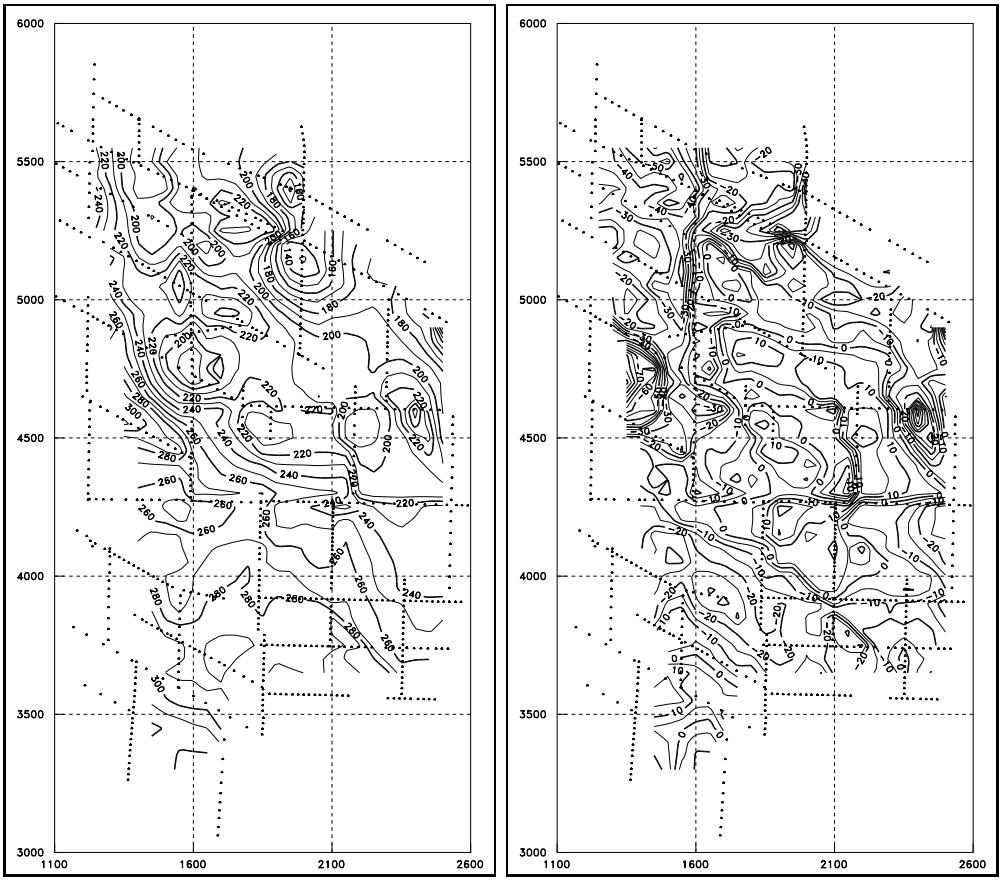

Abbildung 1.21: Suche in Quadranten (Kalilagerstätte)

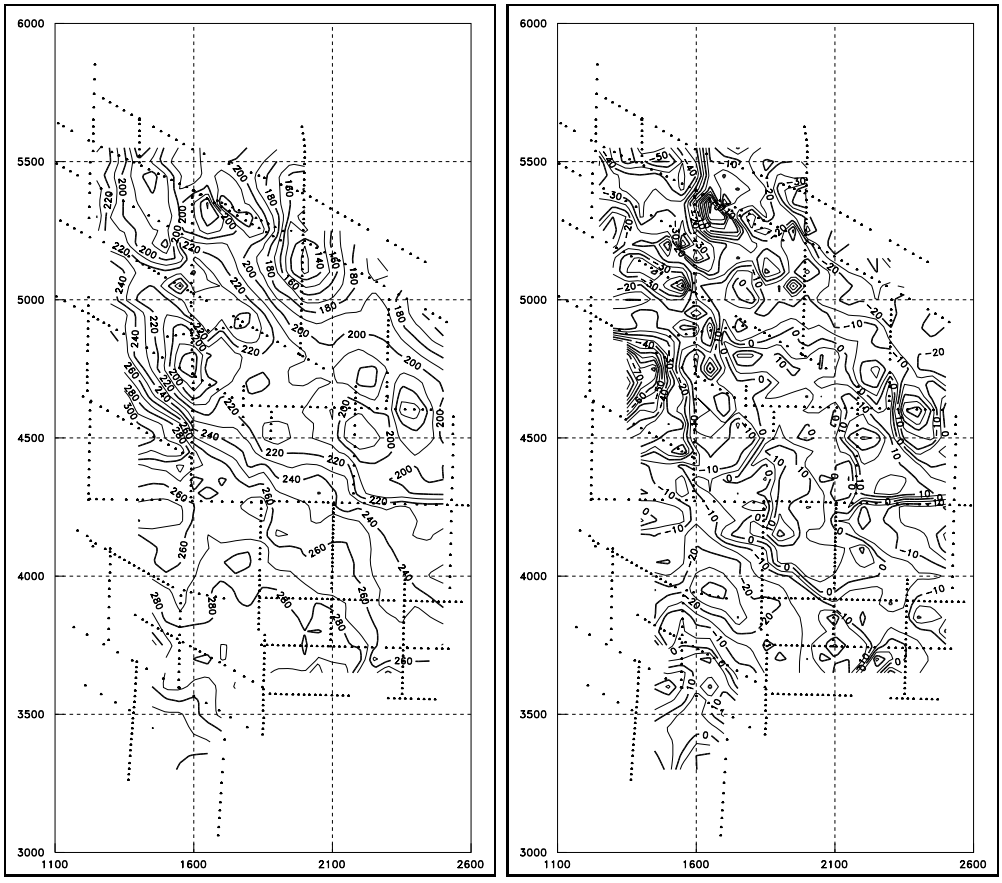

Abbildung 1.22: Suche in Oktanten (Kalilagerstätte)

1.2 Modulstruktur der Auswahl

Der Programmodul *Auswahl* ruft folgende Funktionen auf:

- Anfuegen
- Flaechevek
- Octant
- Quadrant
- Vergleich
- comp.

Diese Funktionen werden alle von dem Modul *Auswahl* mit unterschiedlichen Parametern aufgerufen. Den Ablauf der Auswahl steuern dabei die im Abschnitt 1.1.4 vorgestellten Eingabeparameter.

1.3 Beschreibung der Module des Auswahlalgorithmusses

1.3.1 Funktion Auswahl

1.3.1.1 Aufgabenstellung und Begründung der Umsetzung

Diese Funktion soll aus einem Feld bzw. aus einer Datei diejenigen Datensätze heraussuchen, die für die Schätzung an einer Vorhersagestelle relevant sein sollen. Das Ergebnis dieser Funktion ist ein eindimensionales Feld, dessen Elemente Strukturen sind, die alle Informationen zu der Meßstelle enthalten sowie die Anzahl der Elemente des Feldes. Innerhalb der Funktion findet keine Interaktion statt. Sie wird von der Funktion `Berech` mehrmals, an jeder Schätzstelle jedoch maximal dreimal, aufgerufen.

Im Programmbaustein `Auswahl` ist der zu implementierende Suchalgorithmus das Kernstück, der mit verschiedenen Parametern mehrfach aufgerufen wird und im Verhältnis zu den übrigen Prozeduren die meiste Rechenzeit in Anspruch nimmt (siehe Abb. 1.23).

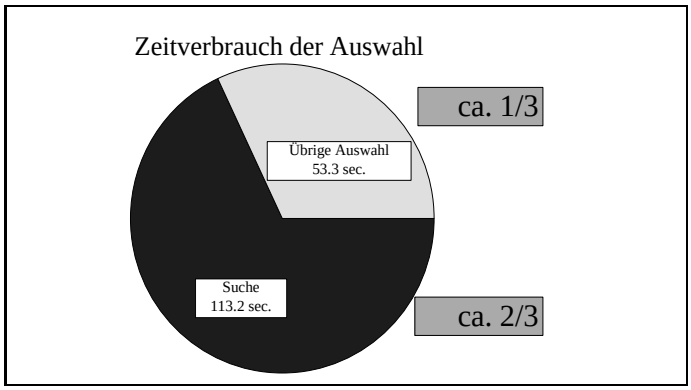

Abbildung 1.23: Rechenzeitverbrauch des Auswahlalgorithmusses

Aus diesem Grunde sei ein etwas ausführlicherer Exkurs zu dieser grundlegenden Operation und der damit verbundenen Entscheidung für einen bestimmten Algorithmus erlaubt. Suchen ist Bestandteil vieler Berechnungsverfahren, deshalb sind die bekanntesten und am meisten gebrauchten Algorithmen sehr gründlich untersucht worden. In der Literatur sind diese Methoden mit ihren Einsatzbedingungen, Fähigkeiten und Leistungsparametern (Zeitfaktor) ausführlich beschrieben [Sed92] . Im weiteren werden die Begriffe Schlüssel für das Suchmuster bzw. Suchkriterium und Datensatz (Feldelement), der die Informationen für das bezeichnete Objekt enthält, verwendet. Felder sollen in diesem Zusammenhang nicht nur Felder im Sinne der Programmierung sein, sondern jegliche adressierbare Strukturen (z.B. Dateien). Bei den grundlegenden Verfahren, auf denen höherentwickelte aufbauen, werden:

- Felder mit Schlüsselvergleichen durchsucht,
- Datensätze mit dem Schlüsselwort indiziert oder
- Strukturen, die durch die Werte der Schlüssel definiert sind, aufgebaut.

SEDGEWICK unterteilt in die folgenden Suchalgorithmen:

- sequentielle Suche
- binäre Suche
- Suche in einem Binärbaum
- Suche in ausgeglichenen Bäumen
- Hashing
- digitale Suche
- externe Suche

Die Wahl des Verfahrens hängt im wesentlichen vom Aufbau und der Länge der Datensatzfelder ab. Die im Programm zu durchsuchenden Felder sind in ihrem Aufbau im Kapitel 1 genau beschrieben. Die Daten sind nach Rechteckflächen sortiert und sequentiell angeordnet. Aus den von SEDGEWICK angebotenen Verfahren lassen sich unter Berücksichtigung des Datenaufbaus die sequentielle Suche und mit Einschränkungen das Hashing anwenden.

Die Länge der Felder kann in Abhängigkeit von der zu lösenden Aufgabenstellung sehr unterschiedlich sein. Bei kleinen Datenmengen wäre die sequentielle Suche die effektivste Methode. Wobei hier anzumerken ist, daß bei dieser Art der Suche in der Literatur von einer Sortierung in Abhängigkeit von der Größe des Schlüssels ausgegangen wird. Das heißt der Algorithmus wird abgebrochen, wenn ein Element größer oder kleiner als der Schlüssel ist. Unter diesen Voraussetzung wird der notwendige Aufwand wie folgt angegeben:

- bei erfolgloser Suche (Feld mit N Datensätze immer N+1 Vergleiche
- bei erfolgreicher Suche durchschnittlich N/2 Vergleiche

Beim Kriging ändert sich jedoch mit der Schätzstelle auch jedesmal der Schlüssel der Strukturelemente nach denen sortiert werden muß. Der Schlüssel ist der söhlige Abstand zwischen Meß- und Vorhersagestelle, der sich in den meisten Fällen während der Suche ebenfalls ändern wird sowie der gewählte Suchbereich. Eine ständige Neusortierung ist natürlich zu aufwendig, deshalb müssen an jedem Vorhersagepunkt N+1 Vergleiche durchgeführt werden.

Für sehr lange Felder, wie sie bei Vorhersagen in ausgedehnten und gut erkundeten Lagerstätten vorkommen können, ist die sequentielle Suche aufgrund des hohen Zeitaufwandes inakzeptabel.
Deshalb ist die Hashing-Methode auf ihre Anwendbarkeit zu untersuchen. Das komplementäre Extrem zur sequentiellen Suche wäre ein einmaliger Zugriff auf den Datensatz in der Tabelle. Hätte man einen Speicher zur Verfügung, könnte der Schlüssel als Speicheradresse verwendet werden, so daß nur ein Zugriff notwendig ist. Das Hashing ist ein Suchalgorithmus aus dem Kompromiß zwischen dem Minimum an Speicherplatzerfordernis bei der sequentiellen Suche und dem schnellen Direktzugriff über Schlüssel. Zwischen beiden soll ein Gleichgewicht gefunden werden. Die Methode beruht auf einer direkten Bezugnahme auf Datensätze einer Tabelle (Feld) durch Ausführung arithmetischer Funktionen, die Schlüssel in Tabellenadressen umwandeln. Diese arithmetische Funktion nennt man Hashfunktion. Im Programm *grille* wird kein reines Hashing verwendet, da ständig wie oben beschrieben die Schlüssel wechseln. Aus den Koordinaten der Schätzstelle und dem Suchradius werden die Rechteckflächen berechnet, die primär relevante Daten enthalten (Abb. 1.4 S. 9). Das ist in diesem Falle die Hashfunktion. Mit den Flächennummern lassen sich aus der Tabelle der Indizes der Anfangs- und Endindex der Daten im Feld der sortierten Datensätze ermitteln. Eine große Zahl von Daten wird dadurch eliminiert, weil sie in Flächen definiert sind, die jenseits der Suchgrenzen liegen.

Die Wirkung der Vorabbehandlung der Daten auf die Rechenzeit ist in Abhängigkeit vom Sortierrasterabstand in Abb. 1.5 veranschaulicht. Die Zeiten beziehen sich auf einen kontrollierten Ablauf des Programms. Mit Hilfe des UNIX-Kommandos *gprof* wird die Datei *gmon.out* ausgewertet, in die durch Verwendung der C-Compiler-Option *-p* beim Programmablauf das Laufzeitverhalten protokolliert wird. Die „wahre“ absolute Abarbeitungsdauer wird sich deshalb von der in der Grafik zu sehenden unterscheiden. Die Werte entstanden durch Vorhersage an 251001 Schätzstellen. Dabei wurden die jeweils 30 nächsten Punkte aus einem Datenfeld von 233 Datensätzen gesucht. Der Suchradius betrug 500 m. Es ist eine deutliche Abnahme der Rechenzeit mit geringer werdenden Sortierrasterabstand zu verzeichnen. Sortierrasterabstände im Bereich des Suchradius sind am günstigsten. Abstände kleiner als der EWB ergeben keine Rechenzeiteinsparung mehr, da die notwendige Zeit für die Flächenbestimmung wegen der erhöhten Zahl der Flächen zunimmt.

1.3.1.2 Schnittstellen

Globale Variablen:

- extern float rechtsVor: Rechtswert der Schätzstelle
- extern float hochVor: Hochwert der Schätzstelle
- extern struct dat urdZ[MAXURDAT],urdY[MAXURDAT],urdG[MAXGRAD]: Felder mit den eingelesenen Daten aus den sortierten Dateien (∼.srt, sogenannte ursprüngliche Daten), wobei bei jedem Aufruf von `Auswahl` in Abhängigkeit vom Funktionsparameter `*urd` nur ein Feld aktuell ist; aus diesen Feldern werden die relevanten Stützstellen ausgewählt; diese Variablen brauchen nur in der aufrufenden Funktion bekannt (extern-Vereinbarung) zu sein
- extern int inffeldZ[MAXINF],inffeldY[MAXINF],inffeldG[MAXINF]: korrespondierende Anfangs– und Endindizes der Daten in `urd*` für die jeweilige Sortierfläche in aufsteigender Reihenfolge sowie die Nummer der Fläche; wobei bei jedem Aufruf von `Auswahl` in Abhängigkeit von Funktionsparameter `*inffeld` nur ein Feld aktuell ist; diese Variablen brauchen nur in der aufrufenden Funktion bekannt (extern-Vereinbarung) sein

- extern double inffeldoZ[6],inffeldoY[6],inffeldoG[6]: Inhalt des Dateikopfes aus ∼.inf; Geometrie des Sortierrasters; wobei bei jedem Aufruf von `Auswahl` in Abhängigkeit von Funktionsparameter `*infeldo` nur ein Feld aktuell ist; diese Variablen brauchen nur in der aufrufenden Funktion bekannt (extern-Vereinbarung) sein

  0. Hochwert links oben
  1. Rechtswert links unten
  2. Anzahl der Spalten
  3. Anzahl der Zeilen
  4. Abstand im Hochwert
  5. Abstand im Rechtswert

- extern int anz_DatZ, anz_DatY, anz_DatG: Anzahl der in `urd*` enthaltenen Dateien; bei jedem Aufruf von `Auswahl` in Abhängigkeit von Funktionsparameter `int anzDa` ist nur eine Variable aktuell; diese Variablen brauchen nur in der aufrufenden Funktion bekannt (extern-Vereinbarung) sein

- extern char filegloZ[10],filegloY[10],filegloG[10]: Stamm des eingelesenen, sortierten Files mit dem „.“ und ohne Extension; in Abhängigkeit vom Funktionsparameter s werden die Filenamen ∼.srt und ∼.inf erzeugt, um bei Überbelegung von `urd*` mit Dateien arbeiten zu können (`datei` wäre dann auf 1 gesetzt)

- extern int datei: Werte 0 (Arbeit mit Feld urd*) oder 1 (Arbeit mit der Datei ∼.srt)

- extern int infja: Werte 0 (Arbeit mit Feldern `inffeld*` und `inffeldo*`) oder 1 (Arbeit mit der Datei ∼.inf)

- extern struct dat ewrZ[MAX_EWR+1], ewrY[MAX_EWR+1], ewrG[MAXEW_GR+1]: Felder in denen die ausgewählten Stützstellen gespeichert sind, sie werden zum Aufbau des Gleichungssystems genutzt; bei jedem Aufruf von `Auswahl` ist in Abhängigkeit vom Funktionsparameter `*ewr` nur ein Feld aktuell ist; diese Variablen brauchen nur in der aufrufenden Funktion bekannt (extern-Vereinbarung) sein

- extern KONSTANTE MAXURDAT: Feldesgröße der sortierten Daten (Stützpunkte)

- extern KONSTANTE MAXGRAD: Feldesgröße der sortierten Daten (Gradienten)

- extern KONSTANTE MAXINF: Feldesgröße der Indexdatei für sortierte Daten; 3*Anzahl der sort. Flächen + 1

- extern KONSTANTE MAX_EWR: max. Sektorbelegung + 1 für Stützpunkte

- extern KONSTANTE MAX_GR: max. Sektorbelegung + 1 für Gradienten

- extern KONSTANTE MAX_SZG: Feldgröße von okt[1-8],sie muß mind. um eins größer sein als die gewünschte max. Sektorbelegung, ist die größere von den KONSTANTEN MAX_EWR und MAX_GR

- extern KONSTANTE LAENAME: Länge der Vektoren für die Filenamen, sie sollten alle die gleiche Länge haben

- extern KONSTANTE ADDTEIL: Anzahl der Iterationschritte bei der Zerlegung des Erweiterungsradius
- extern KONSTANTE DIV: siehe Flaechevek(..)

Funktionsparameter:

- int return-wert (A): Anzahl der ausgewählten Stützpunkte (nicht größer als MaxNear*, 4*Maxsect* oder 8*Maxsect*) oder -1, wenn keine Stützstelle bzw. kein Gradient gefunden wurde.
- struct dat *urd (E): Zeiger auf vorher genannte Felder urdZ,urdY oder urdG; kein Veränderung des Feldes in `Auswahl`
- struct dat *ewr (A): Zeiger auf vorher genannte Felder ewrZ, ewrY oder ewrG
- int s (E): Schalter mit den Werten 1 (Stützpunkte des Vorhersagemerkmals Z), 2 (Stützpunkte des kreuzkorrelierten Merkmals Y) oder 3 (Gradienten G)
- int anzDa (E): Anzahl der eingelesenen Daten in `*urd`
- int EwR (E): Einwirkungsradius in m, in dessen Grenzen nach Stützstellen gesucht werden soll, außerdem wird das Abbild dieser Variablen bei der Bereichserweiterung zur Übergabe an Funktion `Vergleich(..)` schrittweise verändert
- int AwR (E): Radius als innere Begrenzung eines Einwirkungsringes in m, innerhalb dessen nach Stützstellen gesucht werden soll
- int GEwR (E): Erweiterung des EwR; Radius in m und größer als EwR oder bei Nichterweiterung GEwR = -1
- int Richsect (E):
  - 0:richtungsunabhängige Suche nach den absolut nächsten Punkten
  - 4:richtungsabhängige Suche nach den nächsten Punkten in Quadranten
  - 8:richtungsabhängige Suche nach den nächsten Punkten in Oktanten
- int NearRie (E): 0 für absolute Richtungspriorität (direction_priority) und 1 für mixed_priority
- int MaxNear (E): Maximum der nächsten Punkte für Richsect=0 die im EwR ausgewählt werden soll (> 0)
- int Bessect (E): Minimum der nächsten Punkte für Richsect=4,8 die in den Sektoren ausgewählt werden soll ($<$Maxsect und ≥ 0)
- int Maxsect (E): Maximum der nächsten Punkte für Richsect=4,8 die in den Sektoren ausgewählt werden soll (> 0)
- int *inffeld (E): Zeiger auf Indexfeld (s.o.)
- double *infeldo (E): Zeiger auf Feld mit den Kopfdaten der Indexdatei (s.o.)
- int (*Sector)(float *hi,float *ri) (E): Zeiger auf eine Funktion, die beim Aufruf von `Auswahl` angegeben wird und in diesem auswahl*.c-File bekannt sein muß (extern oder hier deklariert); hat bis jetzt die Werte Octant() oder Quadrant()

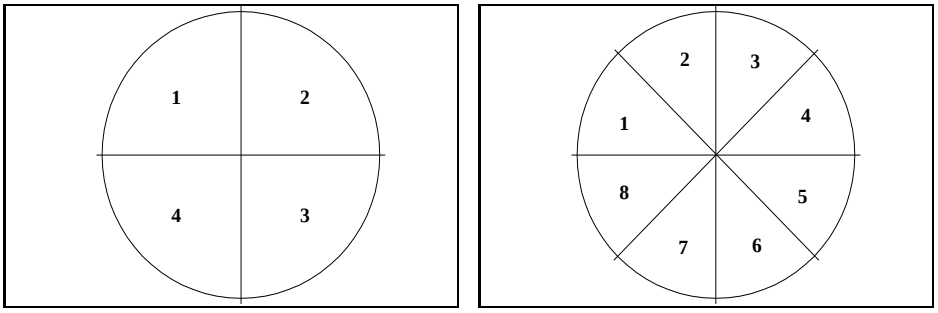

Abbildung 1.24: Bezeichnung der Sektoren bei Quadranten und Oktanteneinteilung

1.3.1.3 Beschreibung der Funktion Auswahl(..)

In der Funktion *Auswahl* wird über die übergebenen Steuerungsparameter die in Übersicht 1.9 S. 14 dargestellte Hierarchie organisiert. Die eigentliche Entscheidung, ob eine Meßstelle in den Suchbereich fällt und an welche Stelle in der Reihenfolge der Punkte diese Meßstelle nach ihrem Abstand zur Vorhersagestelle eingeordnet wird, wird in der Funktion `Vergleich` getroffen. Am Anfang wird in Abhängigkeit vom Übergabeparameter s mit der globalen Variable `fileglo[Z/Y/G]`, die in `Dateiarbeit/Sortierung` oder `Dateiarbeit/Einlesen` belegt wird, die Filenamen ∼.inf und ∼.srt erzeugt. Die Filenamen sind in den char-Feldern `finf und fsrt` abgelegt. Die Zeigervariablen vom Typ `char *finf und *fisrt` zeigen auf diese Felder und dienen lediglich zur Namenserzeugung mit den Standardbibliotheksfunktionen.
Alle Zweige der Auswahlhierarchie sind in ihrem Grundprinzip gleich aufgebaut. Zuerst wird mit der Funktion `Flaechevek` festgestellt, welche Flächennummern für diesen Suchbereich relevant sind. An `int anzflae` wird die Anzahl der Flächen zurückgegeben, im Vektor `int flaevek[28]` stehen die relevanten Flächennummern. Wenn `datei` bzw. `infja` auf 1 stehen, werden die Files mit den in `char fsrt` bzw. `finf` gespeicherten Namen zum Lesen geöffnet. in einer äußeren Schleife mit der Laufvariablen `i` wird von jeder Fläche der Anfangs– und Endindex der darin enthaltenen Datensätze ermittelt und den Variablen `int von` und `bis` zugeordnet. Wenn `datei=1` ist, dann muß auf die Variable `long offset` geachtet werden. `Offset` ist die Zahl der Bytes, an die der Filezeiger `FILE *fpinf` im Bezug zum Fileanfang mit `fseek()` positioniert wird (56 = Kopf der Datei; 16 = Größe einer Zeile). Wenn in `Dateiarbeit/Sortierung` oder `Dateiarbeit/Einlesen` etwas beim Format zum Schreiben geändert wird, sind auch hier andere Zahlen einzusetzen. Wenn `flaevek[i] != flaenr` ist, wird eine Fehlermeldung ausgegeben, meist sind Positionierfehler vorhanden (z.B. durch Veränderung der ∼.inf-Datei per Hand). Sollte `anzflae > 25` sein, wird das gesamte Feld bzw. die gesamte Datei sequentiell durchsucht: `von=0, bis=anzDa-1`. Da es auch leere Flächen gibt, deren Indizes -1 sind, wird vor der Abarbeitung der inneren Schleife mit der Laufvariablen `j` getestet, ob der Anfangsindex `von` gleich -1 ist. Ist das der Fall, wird zur nächsten Fläche übergangen: `for(i=0;i<anzflae;i++)`. Ist die Fläche besetzt, läuft die innere Schleife mit `for(j=von;j<=bis;j++)`. In dieser Schleife wird der Variablen `struct dat pkt` nacheinander die Datensätze aus der gerade aktuellen Sortierfläche zugeordnet (wenn `datei=1` ist, gilt für die Variable `offset` gilt das gleiche wie vorhin; 10 = 1.Zeile in Datei und 75 = eine Datenzeile). Die Variable `pkt` wird als aktueller Vergleichspunkt an `Vergleich` übergeben. Nach Beendigung beider Schleifen werden gegebenenfalls die Dateien geschlossen und die grundlegenden Operationen der Auswahl sind beendet. An dieser Stelle sollte der Abschnitt zur Beschreibung der Funktion `Vergleich(...)`, S. 36 gelesen werden.

Im Zweig `nearest points (Richsect=0)` wird erst, genau wie eben beschrieben, der allgemeine Ablauf abgearbeitet. Das Besondere hier ist, daß `Vergleich` als Übergabeparameter sofort das Feld, auf das `struct dat *ewr` zeigt, benutzt. Es ist größer als die Felder für die Sektorenvergleiche `okt[1-8][MAX_SZG]` und ist das Ergebnisfeld der Funktion `Auswahl`. Damit ist auch der Eingangsparameter `MaxNear` für die maximal mögliche Besetzung im Suchbereich verbunden. Wenn kein Punkt im Suchbereich gefunden wurde (`dimewr<0`) und Bereichserweiterung erlaubt ist, wird die nächste Sequenz abgearbeitet. Sie besteht aus den gleichen Befehlen wie zuvor, nur daß `Vergleich mit schra2=1` für die maximal mögliche Besetzung im Suchbereich und mit vergrößertem Suchradius `EwR` aufgerufen wird. Um Zeit zu sparen, wird nicht sofort im gesamten Erweiterungsbereich gesucht, sondern der Suchradius wird iterativ vergrößert. Die Iterationsschrittweite `addewr` erhält man mit der Teilung der Differenz zwischen Erweiterungsradius `GEwR*` und eigentlichem Suchradius `EwR*` durch `ADDTEIL`. Die Iteration erfolgt mit einer `do{....}while`-Schleife solange, bis ein Punkt gefunden wurde oder `GEwR*` erreicht wurde.

Die weiteren Erklärungen werden zur besseren Orientierung innerhalb der `if...else....`-Anweisungen des Programmquelltextes stattfinden.

```
/* Anfang Zweig fuer Richtungsprioritaet und gemischte Auswahl */
else
      {
      if(NearRie==0) /*Zweig fuer Richtungsprioritaet;
            erstes Besetzen ohne Bereichserweiterung*/
        {
```

In diesem Bereich werden die einzelnen Sektoren besetzt, wenn ausschließliche Richtungspriorität gewählt wurde. Bis zur Zuordnung der Datensätze zur Variablen `struct dat pkt` ist der Ablauf mit dem vorher Beschriebenen deckungsgleich. Mit dem Aufruf der Funktion `(*Sector)(&pkt.h,&pkt.r)` wird über einen Zeiger `Octant oder Quadrant` (abhängig vom Aufruf in `Auswahl(...)`) aufgerufen. `int sec` enthält dann die Sektorennummer. Der Unterschied zum vorangegangenen Zweig ist, daß `Vergleich` in Abhängigkeit von `int sec` mit anderen Parametern aufgerufen wird: die Felder für die Sektorenvergleiche sind okt[1-8][MAX_SZG] und die maximal mögliche Besetzung dieser Felder ist `int Maxsect` (Max. Anzahl pro Sektor im Suchradius).

```
        }
         else /* Mixed-Zweig; nur die Besetzung mit den nearest
points
            und deren nachtraegliche Sortierung in Sectoren ohne
            Richtungsprioritaet */
        {
```

Wenn `mixed_priority (NearRie=1)` gewählt wurde, wird `*ewr`, wie zuerst beschrieben, besetzt und danach auf Sektoren aufgeteilt. Die Besetzung von `*ewr` bleibt unverändert, da nach der Prüfung, ob alle Sektoren mit der Mindestanzahl belegt sind bzw. wenn diese 0, ist `*ewr` an die Funktion `Berech` zurückgegeben wird. Die Sektoren werden nur bis zur max. möglichen Anzahl besetzt. Mögliche überzählige Punkte bleiben unberücksichtigt.

```
        }
/* Ende der Abarbeitg der ersten Schicht (Richtgsprioritaet auf der
einen und mixed auf der anderen Seite; aber beide ohne
Bereichserweiterung bisher*/
/* gemeinsame Abarbeitung von Richtungsprioritaet und mixed fuer die
jeweils weiterfuehrenden Zweige 'ohne Bereichserweit- erung' */
        if(GEwR==(-1))
        {
```

Hier werden alle Fälle abgearbeitet, bei denen keine Bereichserweiterung stattfindet. Es gibt deshalb mehrere `return`-Anweisungen. Nähere Hinweise finden sich direkt im Quelltext. Die Funktion `Anfuegen` speichert alle Teilfelder `okt[1-8]` in das Rückgabefeld von `Auswahl(..)`. Wenn eine Mindestbesetzung gefordert ist, wird das über die Schranke `int schra1 = Bessect-1` geprüft.

```
        }
/* Ende ohne Bereichserweiterung */
/*Bearbeitung der bereichserweiterung---------------*/
/*gemeinsame Bearbeitung der Zweige mit Ber.ewr. aber ohne
vollstaendige Besetzung (mixed und Rich.prioritaet)*/
        if((GEwR!=(-1))&&(Bessect==0))
        {
```

Alle Möglichkeiten, die in den Zweigen mixed_- und direction_priority und keine Forderung nach Mindestbesetzung haben sowie bei denen Bereichserweiterung gefordert ist, werden durchlaufen. Zuerst wird getestet, ob schon mindestens eine Stützstelle gefunden wurde. Ist das der Fall, wird die vorhandene Besetzung zurückgegeben, wobei bei `mixed_priority` ungeachtet der möglichen Überfüllung eines Sektors sofort `*ewr` von `Auswahl(..)` per Zeiger übergeben wird. Ist noch kein Punkt gefunden worden, wird nach dem einzigen nächsten gesucht. Die Methode ist mit der für `nearest points` deckungsgleich. Wird auch in der Erweiterung nichts gefunden, wird -1 returniert.

```
        {
/* Ende fuer unvollstaendige Besetzung fuer mixed und
Richtungsprioritaet*/
 /*Zweig fuer Bereichserweitrg., Vollstaendige Besetzung mit Bessect

und ausschliesslicher Richtungsprioritaet */
        doja=1;
        if((GEwR!=(-1))&&(Bessect!=0))
        {
```

Hier werden für mixed_ und direction_priority bei Forderung nach Bereichserweiterung und Forderung nach Mindestbesetzung die möglichen Fälle bearbeitet. Es sind für dieses Stück viele Erklärungen im Quelltext vorhanden. Am Anfang wird überprüft, ob alle Sektoren die Mindestbesetzung erreichen, bei Erfolg kann sofort abgebrochen werden. Bei `mixed_priority` werden die unvollständigen auf -1 gesetzt und bei `doja=1` (Durchlauf der do-while-Schleife ohne Bereichserweiterung) neu besetzt. Die Sektoren werden nur belegt, wenn sie unvollständig waren. Als Besonderheit dieses Zweiges sei genannt, daß bei `mixed_priority` eventuell vorher überbesetzte Sektoren nur mit der Maximalanzahl `Maxsect` besetzt sind.

```
        }
      }
/*Ende Zweig fuer ausschliessliche Richtungsprioritaet und mixed
Priority */
```

1.3.1.4 Lokale Variablen und ihre Bedeutung

- int anzflae: Anzahl der relevanten Flächen
- int flaevek[28]: Vektor, der die Nummern der relevanten Flächen enthält (siehe Funktion `Vergleich`)
- int i,j: Laufvariablen der Schleifen
- int von,bis: Anfangs– und Endindex der Datensätze in den einzelnen relevanten Flächen

- int flaenr: Flächennummer, die aus Datei oder Indexfeld gelesen wird (Kontrolle)
- int dimokt1,dimokt2,dimokt3,dimokt4,dimokt5,dimokt6,dimokt7,dimokt8: aktuelle Dimensionen der Felder `okt[1-8]`
- int dimewr: Dimension des Feldes auf den der Zeiger `*ewr` zeigt; return-wert
- int schra2: Obergrenze (Dimension), bis zu der das struct dat Feld der Funktion `Vergleich` gefüllt sein darf (Maxsect,1,Bessect)
- int schra1: Synonym für Mindestanzahl pro Sektor (hier Dimension für die Felder der Sektoren, Bessect-1)
- int sec: Nummer des Sektors, um den Punkt einem Sektorfeld zuordnen zu können (Quadrant 1-4 und Oktant 1-8, siehe selbige Funktionen)
- int doja: Flag, das erst 0 gesetzt wird, und nach der ersten Abarbeitung der letzten do-while-Schleife gleich 1 ist
- char *fiinf,finf[LAENAME]: Zeiger und Name des sortierten Datenfiles
- char *fisrt,fsrt[LAENAME]: Zeiger und Name des zugehörigen Indexfeldes
- struct dat okt1[MAX_SZG],okt2[MAX_SZG],okt3[MAX_SZG],okt4[MAX_SZG], okt5[MAX_SZG],okt6[MAX_SZG],okt7[MAX_SZG],okt8[MAX_SZG]: temporäre Felder für Datensätze in den Sektoren
- struct dat pkt: jeweils der Datensatz (Stützstelle), der an `Vergleich(..)` zur Überprüfung übergeben wird
- FILE *fpinf: Filedescriptor der Indexdatei
- FILE *fpsrt: Filedescriptor der sortierten Datendatei
- long offset: Byteanzahl für Positionierung der Filedescriptoren
- float addewr: Iterationsschrittweite in den do-while-Schleifen

1.3.2 Funktion Flaechevek

1.3.2.1 Aufgabenstellung und Begründung der Umsetzung

Diese Funktion hat die Aufgabe, aus einem definierten Flächenraster, diejenigen Flächen herauszusuchen, die in den Suchbereich `EwR` fallen. Die Flächennummern werden im Vektor `*flaevektt` an `Auswahl(..)` zurückgeben (siehe Abb. 1.4, S. 9). Außerdem wird die Anzahl der relevanten Flächen zurückgegeben. Die Routine soll wahlweise mit einer Indexdatei oder einem Indexfeld arbeiten können. Die bevorzugte Methode sollte die Nutzung des Feldes sein. Wenn mehr als 25 Flächen in den Suchradius fallen, wird der Wert 28 (wenn mehr als 25 Fl'ache relevant sind, wird die gesamte Datei (Feld) durchsucht) zurückgegeben.

1.3.2.2 Schnittstellen

Globale Variablen:

- extern int infja: Werte 0 (Arbeit mit Feldern inffeld* und inffeldo*) oder 1 (Arbeit mit der Datei ∼.inf)
- extern float rechtsVor: Rechtswert der Schätzstelle

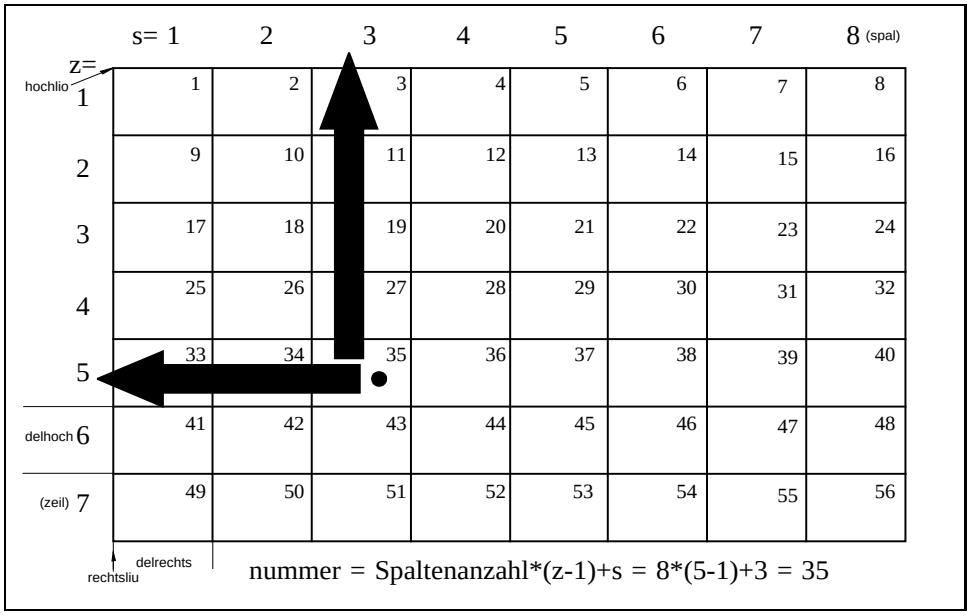

Abbildung 1.25: Numerierung und Orientierung des Sortierrasters

- extern float hochVor: Hochwert der Schätzstelle

- extern KONSTANTE DIV: um wieviel EwR/DIV darf der EwR über eine Rechteckfläche hinausragen, damit die benachbarte Rechteckfläche nicht als relevant gezählt wird

Funktionsparameter:

- char *filename (E): Zeiger auf Filename der Indexdatei ∼.inf

- int *flaevektt (A): Zeiger auf das Feld, das im aufrufenden Programm die Flächennummern enthalten soll

- int EwR (E): Suchradius in m

- double *inffeldo (E): Zeiger auf Feld, das die Kopfdaten des Sortierrasters enthält (siehe Funktion `Auswahl(..)`), unverändert zurück

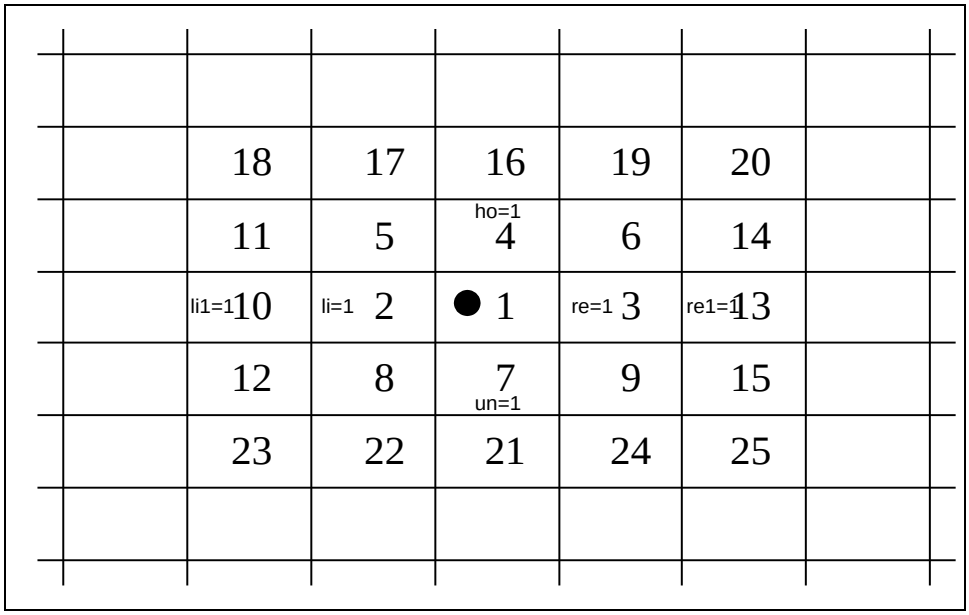

Abbildung 1.26: Abarbeitungsreihenfolge bei der Suche nach relevanten Flächen

1.3.2.3 Beschreibung der Funktion Flaechevek(..)

Zuerst werden in Abhängigkeit von `infja` die Kopfdaten der Indexdatei aus dem File mit dem Namen `char *filename` (`infja==1`) gelesen oder vom übergebenen Feld `int *flaevektt` an die entsprechenden Variablen übergeben (Belegung des Feldes in Abschnitt 1.3.1.2, S. 27 f). `s` und `z` sind die Spalten– bzw. Zeilenflächennummern im Sortierraster, dessen Ursprung links oben ist und dessen +y-Achse nach unten gerichtet ist. `s` und `z` werden für die Fläche ermittelt, in der die Stützstelle liegt. Wenn der Punkt irgendwo außerhalb des Sortierrasters liegt, wird jeweils 28 zurückgegeben, damit alle Punkte zur Suche herangezogen werden. Die Ermittlung von `nummer` ist in Abb. 1.25 zu sehen. `nummer` ist das erste Element des Flächennummernvektors und die Fläche, in dem die Vorhersagestelle liegt. Sie wird immer zurückgegeben, die anderen Flächen hängen von der Größe des Suchradius ab. Dann werden die Rechtswerte der linken und rechten Begrenzung (`rechtsre, rechtsli`) sowie die Hochwerte der oberen und unteren Begrenzung (`hochob, hochun`) der ersten relevanten Fläche bestimmt. Sie dienen im weiteren zur Entscheidung, ob der Suchradius über die jeweilige Grenze hinausragt, und so die nächstliegende Fläche in dieser Richtung ebenfalls relevant ist. Durch die Konstante `DIV` wird erreicht, daß der Suchradius einen definierten Anteil von diesem über die Begrenzung hinausragen darf, ohne daß die benachbarte Fläche als relevant gilt. Die Reihenfolge der Abfrage der benachbarten Flächen ist in Abb. 1.26 mit den entsprechenden Schaltern dargestellt. Die Schalter werden auf 1 gesetzt, wenn die entsprechende Fläche als relevant gezählt wird. Bei den Flächen 2, 3, 4 und 7 wird nur getestet, ob der Suchradius in die Flächen hineinreicht. Außerdem, und das gilt auch für die Flächen 10, 13, 16 und 21 ebenfalls, wird überprüft, ob bereits der Rand des Sortierrasters erreicht ist. Wenn die Schalter `li` bzw. `re` auf ein stehen, werden die Flächen 5, 6, 8 und 9 als relevant gezählt. Im Anschluß daran werden noch die äußeren Flächen ausgetestet. Dabei wird zur Zeitersparnis jeweils die übergeordnete Frage gestellt, ob der anderthalbfache Suchradius größer als die Sortierrasterabstände `delrechts` und `delhoch` sind. Das eben beschriebene System wird auch für die äußeren Flächen angewendet. Sollte der Suchradius größer als das Zweieinhalbfache von `del` sein, dann wird 28 zurückgegeben und alle Datensätze zum Vergleichen genutzt.

1.3.2.4 Lokale Variablen und ihre Bedeutung

- int i: am Ende Anzahl der Flächen, sonst Laufvariable des Feldes für die Flächennummern
- int ii: Laufvariable
- int li: Schalter, ob Fläche 2 relevant ist (Abb. 1.26)
- int re: Schalter, ob Fläche 3 relevant ist (Abb. 1.26)
- int ho: Schalter, ob Fläche 4 relevant ist (Abb. 1.26)
- int un: Schalter, ob Fläche 7 relevant ist (Abb. 1.26)
- int li1: Schalter, ob Fläche 10 relevant ist (Abb. 1.26)
- int re1: Schalter, ob Fläche 13 relevant ist (Abb. 1.26)
- double s: Spaltennummer der Fläche 1 (`nummer`)
- double z: Zeilennummer der Fläche 1 (`nummer`)
- double nummer: Flächennummer der Fläche. in der die Stützstelle liegt
- double hochlio: Hochwert links oben im Sortierraster
- double rechtsliu: Rechtswert links liu im Sortierraster
- double delhoch: Abstand des Sortierrasters im Hochwert
- double delrechts: Abstand des Sortierrasters im Rechtswert
- double del: der kleinere Abstand des Sortierrasters
- double rechtsli: Rechtswert der linken Begrenzung der Fläche mit `nummer`
- double rechtsre: Rechtswert der rechten Begrenzung der Fläche mit `nummer`
- double hochob: Hochwert der oberen Begrenzung der Fläche mit `nummer`
- double hochun: Hochwert der unteren Begrenzung der Fläche mit `nummer`
- double spal: Spaltenanzahl des Sortierrasters
- double zeil: Zeilenanzahl des Sortierrasters
- double flaevekt[28]: lokaler Vektor für die Flächennummer (aber im Gegensatz zum Zeiger `*flaevekt`: double zwecks der Berechnung)
- FILE *fp: Filedescriptor auf das Indexfeld

1.3.3 Funktion Vergleich

1.3.3.1 Aufgabenstellung und Begründung der Umsetzung

Diese Funktion ist das Kernstück des Auswahlalgorithmusses. Sie führt die Vergleiche durch:

1. liegt die Stelle im Suchbereich,
2. liegt die Stelle näher als bereits vorher untersuchte Stellen an der Vorhersagestelle,

3. Sortierung der relevanten Stellen nach dem Abstand zur Vorhersagestelle sobald das Feld mit der maximalen Anzahl gefällt ist und Einordung der folgenden Stellen in diese Reihenfolge,

4. Rückgabe der Dimension des Feldes mit den relevanten Stützstellen sowie des Feldes selbst.

1.3.3.2 Schnittstellen

Globale Variablen:

- keine

Funktionsparameter:

- int return: Dimension des Feldes, auf das `struct dat *okt` zeigt
- struct dat *okt (E/A): Zeiger auf ein Feld, das nach dem Durchlauf von `Vergleich(..)` die relevanten Datensätze enthält
- struct dat pkt (E): die zu untersuchende Stelle
- int dimokt (E): Dimension des Feldes, auf das `struct dat *okt` zeigt, bevor `Vergleich(..)` durchlaufen wurde
- int sch (E): Schalter, ob Stützpunkte Z, Y (1, 2) oder Gradienten G (3) untersucht werden (Stützpunkte haben eine Streuung, die bei der Priorität von Stützstellen eine Rolle spielt)
- int constt (E): Anzahl der maximalen Besetzung des Feldes (MaxNear, Maxsect)
- int EwR (E): Suchradius
- int AwR (E): innerer Radius eins möglichen Suchringes

1.3.3.3 Beschreibung der Funktion Vergleich(..)

Zuerst wird für den späteren Abstandsvergleich der Abstand zwischen Vorhersagepunkt und der aktuellen Stützstelle berechnet und `pkt.fre3` zugeordnet. Dann erfolgt der Test, ob die Stützstelle in den Suchbereich fällt. Bei Erweiterung des Programmes kann hier eine beliebige Funktion mit anderen Suchgrenzen z.B. dreidimensionale Suchellipsoid eingefügt werden. Der Algorithmus spaltet sich in zwei Teile auf, bei dem:

- das Feld der relevanten Stützstellen noch nicht voll besetzt ist
- das Feld der relevanten Stützstellen bereits voll besetzt ist.

In diesen Zweigen werden noch jeweils Extrabehandlungen für Stützpunkte Z, Y einerseits und Gradienten G andererseits gemacht. Wenn das Feld noch nicht voll besetzt ist, wird es vorerst unsortiert bis zur Maximalfüllung besetzt. Bei den Stützpunkten Z, Y wird das erste Element mit der Streuung 0 immer auf das Feldelement mit dem Index 0 gesetzt. Bei allen weiteren Elementen mit der Streuung 0 werden die Abstände zur Vorhersagestützstelle mit dem, des auf dem Feldelement mit dem Index 0 stehen, verglichen. Wenn er kleiner ist kommt das neue auf diesen Platz, ansonsten wird das neue wie „normale“ Stützstellen behandelt. Beim Austausch auf „Platz 0“ verschiebt sich die Besetzung um ein Element nach hinten. Sobald die vollständige Füllung erreicht ist, wird das Feld nach dem Quicksort-Algorithmus

mit größer werdenden Abständen sortiert. (da es kurze Felder sind, ist wahrscheinlich ein anderer Sortieralg. angebracht) Wenn das erste Element die Streuung 0 hat, wird ab dem Zweiten sortiert, da eine Besetzung mit genau einem Element der Streuung 0 Vorrang hat [Har90].

Wenn das Feld voll besetzt ist, wird weiterhin versucht das erste Feldelement bei Stützpunkten Z, Y mit Elementen der Streuung 0 zu besetzen. Wobei, wenn noch keines da war, das letzte, am weitesten entfernt liegende Element herausfallen würde, oder es wird, wenn schon eines da war, der Abstandsvergleich gemacht. Wenn das neue Element näher liegt, kommt es auf „Platz 0“, ansonsten wird es wie eine „normale“ Stützstelle behandelt. Diese wird mit dem jeweils letzten, am weitesten weg liegenden Element verglichen und wenn sie näher als dieses zur Vorhersagestelle liegen, sofort in die abfallende Reihenfolge eingeordnet. Als return-wert wird die Dimension des Feldes zurückgegeben.

1.3.3.4 Lokale Variablen und ihre Bedeutung

- int i: Laufvariable
- unsigned width: Größe der Struktur `struct dat`
- struct dat temp: temporäre Variable zur Zwischenablage beim Einsortieren von Stützstellen

1.3.4 Funktion Anfuegen

1.3.4.1 Aufgabenstellung und Begründung der Umsetzung

Diese Funktion soll zwei Felder zusammenfügen in der Art, das das Zweite an das Erste gehangen wird und das neue Feld den Namen des Ersten hat. Die Dimensionen der Felder müssen bekannt sein. Als return-wert wird die Dimension des neuen Feldes oder , wenn beide leer sind, -1 zurückgegeben.

1.3.4.2 Schnittstellen

Globale Variablen:

- keine

Funktionsparameter:

- int return-wert: neue Dimension des (gesamten) 1. Feldes
- int dim (E): Dimension des 1. Feldes
- int dimokt (E): Dimension des anzuhängenden 2. Feldes
- struct dat *okt (E): Zeiger auf das 2. Feld
- struct dat *ewrr (E/A): Zeiger auf das 1. Feld

1.3.4.3 Lokale Variablen und ihre Bedeutung

- int i: Laufvariable für 1. Feld
- int j: Laufvariable für 2. Feld

1.3.5 Funktionen Octant,Quadrant

1.3.5.1 Aufgabenstellung und Begründung der Umsetzung

Sie sollen die Sektorennummer zurückgeben, in welchem sich ein Punkt bezogen auf die Vorhersagestelle als Zentrum befindet. Die Bezeichnung der Sektoren sieht man in Abb. 1.24, S. 30.

1.3.5.2 Schnittstellen

Globale Variablen:

- extern float hochVor: Hochwert der Vorhersagestelle
- extern float rechtsVor: Rechtswert der Vorhersagestelle

Funktionsparameter:

- int return-wert: Sektorennummer
- float *hi: Hochwert des Punktes
- float *ri: Rechtswert des Punktes

1.3.5.3 Lokale Variablen und ihre Bedeutung

- keine

1.3.6 Funktion comp

1.3.6.1 Aufgabenstellung und Begründung der Umsetzung

Es wird eine Vergleichsfunktion für die Standardbibliotheksfunktion `qsort(..)` für den konkreten Variablentyp `struct dat` bereitgestellt [KR90].

1.3.6.2 Beschreibung der Funktion comp(..)

Von zwei Punkten wird der Abstand zur Vorhersagestelle verglichen und die return's sind so eingerichtet, daß gemäß der Funktion `qsort(..)` das Feld vom kleinsten Element (Abstand) beginnend sortiert wird [KR90].

1.4 Quelltext zur Auswahl – auswahl5.c

```
/* Headerdatei fuer auswahl5.c---------------------------*/

#include "typ_defi.h"
#include "exdaarb5.h"
#include "exmod2.h"
#include <math.h>
#define DIV 5
#define ADDTEIL 5

extern float rechtsVor,hochVor;

int comp(struct dat *,struct dat *);
int Octant(float *,float *);
int Quadrant(float *,float *);
int Flaechevek(char *,int * ,int ,double *);
int Anfuegen(int ,int ,struct dat *,struct dat *);
int Auswahl(struct dat *,struct dat *,int ,int ,
int ,int ,int ,int ,int ,int , int ,int ,int *,double *,
int (*Sector)(float *,float *));
int Vergleich(struct dat *,struct dat , int ,int ,int ,
int ,int );
int Singueberpr(struct dat *, int *,int ,float ,
                        float (*Snum)(float *,float ));
float Snumi(float *,float );


/*----------  auswahl5.c ---------------------------------*/
#include "headausw5.h"


int Auswahl(struct dat *urd,struct dat *ewr,int s,int anzDa,
int EwR,int AwR,int GEwR,int Richsect,int NearRie,int MaxNear,
int Bessect,int Maxsect,int *inffeld,double *infeldo,
int (*Sector)(float *hi,float *ri))
{
int anzflae,flaevek[28],i,von,bis,flaenr,j,dimokt8;
int dimokt1,dimokt2,dimokt3,dimokt4,dimokt5,dimokt6,dimokt7;
int dimewr,schra2,schra1,sec,doja;
char *fiinf,finf[LAENAME],*fisrt,fsrt[LAENAME];
struct dat okt1[MAX_SZG],okt2[MAX_SZG],okt3[MAX_SZG];
struct dat okt4[MAX_SZG],okt5[MAX_SZG],okt6[MAX_SZG];
struct dat okt7[MAX_SZG],okt8[MAX_SZG],pkt;
FILE *fpinf,*fpsrt;
long offset;
float addewr;

dimokt1=dimokt2=dimokt3=dimokt4=dimokt5=dimokt6=dimokt7=dimokt8=(-1);
dimewr=(-1);

/*Erzeugung Filenamen,Eroeffnen *.inf u. *.srt  */
for(i=0;i<=LAENAME;i++)
  {
  finf[i]='\0';
  fsrt[i]='\0';
  }
fiinf = finf,fisrt = fsrt;
if(s==1)
  {
  strcat(finf,filegloZ);
  strcat(fsrt,filegloZ);
  }
if(s==2)
  {
  strcat(finf,filegloY);
```

```
  strcat(fsrt,filegloY);
  }
if(s==3)
  {
  strcat(finf,filegloG);
  strcat(fsrt,filegloG);
  }
strcat(fiinf,"inf");
strcat(fisrt,"srt");
/* Ende Erzeugung Filenamen,Eroeffnen *.inf u. *.srt  */

/* Anfang fuer nearest points (ausschliesslich) */
if(Richsect==0)
  {
  anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
  if(infja==1)
    fpinf=fopen(finf,"r");
  if(datei==1)
    fpsrt = fopen(fsrt,"r");
  for(i=0;i<(anzflae);i++)
    {
    if(anzflae<26)
     {
     if(infja==1)
      {
      offset= 54 + 16*(flaevek[i]-1);
      setbuf(fpinf,NULL);
      fseek(fpinf,offset,SEEK_SET);
      fscanf(fpinf,"%d",&flaenr);
      fscanf(fpinf,"%d",&von);
      fscanf(fpinf,"%d",&bis);
      }
     else
      {
      flaenr = inffeld[3*(flaevek[i]-1)+6];
      von = inffeld[3*(flaevek[i]-1)+7];
      bis = inffeld[3*(flaevek[i]-1)+8];
      }
      if(flaevek[i]!=flaenr)
        {
        printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
        putchar('\007');
        sleep(4);
        }
     }
    else
      von=0,bis=(anzDa-1),i=anzflae;
    if(von != (-1))
      for(j=von;j<=bis;j++)
        {
        if(datei==0)  /* Nutzung des Feldes urd  */
         pkt = urd[j];
        else  /* Nutzung des Files *.srt    */
          {
          offset = 75*j+10;
          fseek(fpsrt,offset,SEEK_SET);
          fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
    &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
          }
        dimewr=Vergleich(ewr,pkt,dimewr,s,MaxNear,EwR,AwR);
     }  /* fore j */
    }    /* fore i */
  if(infja==1)
    fclose(fpinf);
  if(datei==1)
    fclose(fpsrt);
```

```
 /* Bereicherweiterung, wenn noch kein Punkt im EWB liegt und wenn
 Bereichserweiterung erlaubt ist */
 if((dimewr<0)&&(GEwR!=(-1)))
   {
   addewr = (GEwR-EwR)/ADDTEIL;
   do{
     EwR += addewr;
     schra2=1;
     anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
     if(infja==1)
       fpinf=fopen(finf,"r");
     if(datei==1)
       fpsrt = fopen(fsrt,"r");
     for(i=0;i<(anzflae);i++)
       {
       if(anzflae<26)
        {
if(infja==1)
 {
         offset= 54 + 16*(flaevek[i]-1);
         setbuf(fpinf,NULL);
         fseek(fpinf,offset,SEEK_SET);
         fscanf(fpinf,"%d",&flaenr);
         fscanf(fpinf,"%d",&von);
         fscanf(fpinf,"%d",&bis);
 }
        else
         {
         flaenr = inffeld[3*(flaevek[i]-1)+6];
         von = inffeld[3*(flaevek[i]-1)+7];
         bis = inffeld[3*(flaevek[i]-1)+8];
         }
         if(flaevek[i]!=flaenr)
           {
       printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
           putchar('\007');
           sleep(4);
           }
         }
       else
         von=0,bis=(anzDa-1),i=anzflae;
       if(von != (-1))
         for(j=von;j<=bis;j++)
           {
           if(datei==0)  /* Nutzung des Feldes urd  */
            pkt = urd[j];
           else  /* Nutzung des Files *.srt    */
             {
             offset = 75*j+10;
             fseek(fpsrt,offset,SEEK_SET);
             fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
             &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
             }
           dimewr=Vergleich(ewr,pkt,dimewr,s,schra2,EwR,AwR);
        }  /* fore j */
       }    /* fore i */
     if(infja==1)
       fclose(fpinf);
     if(datei==1)
       fclose(fpsrt);
     }
   while((EwR<(GEwR-addewr/2))&&(dimewr==(-1)));
   }
 if(dimewr==(-1))
   return (-1);
 else
   return(dimewr+1);
```

```
  }
/* Ende Zweig fuer nearest points ------------------------------*/


/* Anfang Zweig fuer Richtungsprioritaet und gemischte Auswahl */
else
  {
  if(NearRie==0) /*Zweig fuer Richtungsprioritaet;
                   erstes Besetzen ohne Bereichserweiterung*/
    {
    anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
    if(infja==1)
      fpinf=fopen(finf,"r");
    if(datei==1)
      fpsrt = fopen(fsrt,"r");
    for(i=0;i<(anzflae);i++)
      {
      if(anzflae<26)
       {
       if(infja==1)
{
        offset= 54 + 16*(flaevek[i]-1);
        setbuf(fpinf,NULL);
        fseek(fpinf,offset,SEEK_SET);
        fscanf(fpinf,"%d",&flaenr);
        fscanf(fpinf,"%d",&von);
        fscanf(fpinf,"%d",&bis);
}
       else
        {
        flaenr = inffeld[3*(flaevek[i]-1)+6];
        von = inffeld[3*(flaevek[i]-1)+7];
        bis = inffeld[3*(flaevek[i]-1)+8];
        }
        if(flaevek[i]!=flaenr)
          {
        printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
          putchar('\007');
          sleep(4);
          }
        }
      else
        von=0,bis=(anzDa-1),i=anzflae;
      if(von != (-1))
        for(j=von;j<=bis;j++)
          {
          if(datei==0)  /* Nutzung des Feldes urd  */
           pkt = urd[j];
          else  /* Nutzung des Files *.srt    */
            {
            offset = 75*j+10;
            fseek(fpsrt,offset,SEEK_SET);
            fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
     &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
            }

          sec=(*Sector)(&pkt.h,&pkt.r);
          switch(sec)
            {
            case 1 :
              dimokt1=Vergleich(okt1,pkt,dimokt1,s,Maxsect,EwR,AwR);
              break;
            case 2 :
              dimokt2=Vergleich(okt2,pkt,dimokt2,s,Maxsect,EwR,AwR);
              break;
            case 3 :
              dimokt3=Vergleich(okt3,pkt,dimokt3,s,Maxsect,EwR,AwR);
```

```
            break;
          case 4 :
            dimokt4=Vergleich(okt4,pkt,dimokt4,s,Maxsect,EwR,AwR);
            break;
          case 5 :
            dimokt5=Vergleich(okt5,pkt,dimokt5,s,Maxsect,EwR,AwR);
            break;
          case 6 :
            dimokt6=Vergleich(okt6,pkt,dimokt6,s,Maxsect,EwR,AwR);
            break;
          case 7 :
            dimokt7=Vergleich(okt7,pkt,dimokt7,s,Maxsect,EwR,AwR);
            break;
          case 8 :
            dimokt8=Vergleich(okt8,pkt,dimokt8,s,Maxsect,EwR,AwR);
            break;
          default :
            printf("\nIt is not possible point assign to a sector\n");
            putchar('\007');
            sleep(4);
          }

     }  /* fore j */
    }    /* fore i */
  if(infja==1)
    fclose(fpinf);
  if(datei==1)
    fclose(fpsrt);
  }
 else  /* Mixed-Zweig; nur die Besetzung mit den nearest points
        und deren nachtraegliche Sortierung in Sectoren ohne
        Richtungsprioritaet */
  {
  anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
  if(infja==1)
    fpinf=fopen(finf,"r");
  if(datei==1)
    fpsrt = fopen(fsrt,"r");
  for(i=0;i<(anzflae);i++)
    {
    if(anzflae<26)
     {
     if(infja==1)
{
      offset= 54 + 16*(flaevek[i]-1);
      setbuf(fpinf,NULL);
      fseek(fpinf,offset,SEEK_SET);
      fscanf(fpinf,"%d",&flaenr);
      fscanf(fpinf,"%d",&von);
      fscanf(fpinf,"%d",&bis);
}
     else
      {
      flaenr = inffeld[3*(flaevek[i]-1)+6];
      von = inffeld[3*(flaevek[i]-1)+7];
      bis = inffeld[3*(flaevek[i]-1)+8];
      }
      if(flaevek[i]!=flaenr)
        {
      printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
        putchar('\007');
        sleep(4);
        }
      }
    else
      von=0,bis=(anzDa-1),i=anzflae;
    if(von != (-1))
```

```
    for(j=von;j<=bis;j++)
      {
      if(datei==0)  /* Nutzung des Feldes urd  */
       pkt = urd[j];
      else  /* Nutzung des Files *.srt    */
        {
        offset = 75*j+10;
        fseek(fpsrt,offset,SEEK_SET);
        fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
   &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
}
      dimewr=Vergleich(ewr,pkt,dimewr,s,MaxNear,EwR,AwR);
      }  /* fore j */
  }    /* fore i */
if(infja==1)
  fclose(fpinf);
if(datei==1)
  fclose(fpsrt);

/* Aufteilung der nearest points auf die Sektoren */
for(i=0;i<=dimewr;i++)
  {
  sec=(*Sector)(&ewr[i].h,&ewr[i].r);
  switch(sec)
    {
    case 1 :
      if(dimokt1<Maxsect)
        {
okt1[dimokt1+1]=ewr[i];
dimokt1++;
        }
      break;
    case 2 :
      if(dimokt2<Maxsect)
        {
okt2[dimokt2+1]=ewr[i];
dimokt2++;
        }
      break;
    case 3 :
      if(dimokt3<Maxsect)
        {
okt3[dimokt3+1]=ewr[i];
dimokt3++;
        }
      break;
    case 4 :
      if(dimokt4<Maxsect)
        {
okt4[dimokt4+1]=ewr[i];
dimokt4++;
        }
      break;
    case 5 :
      if(dimokt5<Maxsect)
        {
okt5[dimokt5+1]=ewr[i];
dimokt5++;
        }
      break;
    case 6 :
      if(dimokt6<Maxsect)
        {
okt6[dimokt6+1]=ewr[i];
dimokt6++;
        }
      break;
```

```
        case 7 :
          if(dimokt7<Maxsect)
            {
    okt7[dimokt7+1]=ewr[i];
    dimokt7++;
            }
          break;
        case 8 :
          if(dimokt8<Maxsect)
            {
    okt8[dimokt8+1]=ewr[i];
    dimokt8++;
            }
          break;
        default :
          printf("\nIt is not possible point assign to a sector\n");
          putchar('\007');
          sleep(4);
        }
      }
    }
/* Ende der Abarbeitung der ersten Schicht (Richtungsprioritaet auf
der einen und mixed auf der anderen Seite; aber beide ohne
Bereichserweiterung bisher*/


/* gemeinsame Abarbeitung von Richtungsprioritaet und mixed
fuer die jeweils weiterfuehrenden Zweige 'ohne Bereichserweit-
erung' */
  if(GEwR==(-1))
    {
    if(Bessect==0)
      {
      if(NearRie==0) /* gilt fuer direction_priority, ansonsten Rueckgabe */
{            /* vom ganzen *ewr */
dimewr = (-1);
        dimewr=Anfuegen(dimewr,dimokt1,okt1,ewr);
        dimewr=Anfuegen(dimewr,dimokt2,okt2,ewr);
        dimewr=Anfuegen(dimewr,dimokt3,okt3,ewr);
        dimewr=Anfuegen(dimewr,dimokt4,okt4,ewr);
        dimewr=Anfuegen(dimewr,dimokt5,okt5,ewr);
        dimewr=Anfuegen(dimewr,dimokt6,okt6,ewr);
        dimewr=Anfuegen(dimewr,dimokt7,okt7,ewr);
        dimewr=Anfuegen(dimewr,dimokt8,okt8,ewr);
}
      if(dimewr==(-1))   /* gilt fuer NearRie==0 und NearRie!=0 */
return(-1);
      else               /* gilt fuer NearRie==0 und NearRie!=0 */
return(dimewr+1);
      }
    else /* vollstaendige Besetzung ist notwendig*/
      {
      schra1=Bessect-1;
      if(Richsect==4)
{
        if((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
  ||(dimokt4<schra1))
          return(-1);
        else
  {
  if(NearRie==0)
  {
          dimewr = (-1);
          dimewr=Anfuegen(dimewr,dimokt1,okt1,ewr);
          dimewr=Anfuegen(dimewr,dimokt2,okt2,ewr);
          dimewr=Anfuegen(dimewr,dimokt3,okt3,ewr);
          dimewr=Anfuegen(dimewr,dimokt4,okt4,ewr);
```

```
          return(dimewr+1);
  }
  else
  return(dimewr+1); /* wenn mixed priority gefordert ist */
  }
        }
      if(Richsect==8)
{
        if((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
           ||(dimokt4<schra1)||(dimokt5<schra1)||(dimokt6<schra1)
           ||(dimokt7<schra1)||(dimokt8<schra1))
          return(-1);
        else
  {
  if(NearRie==0)
  {
          dimewr = (-1);
          dimewr=Anfuegen(dimewr,dimokt1,okt1,ewr);
          dimewr=Anfuegen(dimewr,dimokt2,okt2,ewr);
          dimewr=Anfuegen(dimewr,dimokt3,okt3,ewr);
          dimewr=Anfuegen(dimewr,dimokt4,okt4,ewr);
          dimewr=Anfuegen(dimewr,dimokt5,okt5,ewr);
          dimewr=Anfuegen(dimewr,dimokt6,okt6,ewr);
          dimewr=Anfuegen(dimewr,dimokt7,okt7,ewr);
          dimewr=Anfuegen(dimewr,dimokt8,okt8,ewr);
          return(dimewr+1);
  }
  else
  return(dimewr+1);
          }
        }
      }
    }
/* Ende ohne Bereichserweiterung */


/*Bearbeitung der Bereichserweiterung---------------*/

/*gemeinsame Bearbeitung der Zweige mit Ber.ewr. aber
ohne vollstaendige Besetzung (mixed und Rich.prioritaet*/
  if((GEwR!=(-1))&&(Bessect==0))
    {
    addewr = (GEwR-EwR)/ADDTEIL;
    do{
      EwR += addewr;
      schra2=1;
      if(NearRie==0) /* nur Richtungsprioritaet*/
      {
      dimewr = (-1);
      dimewr=Anfuegen(dimewr,dimokt1,okt1,ewr);
      dimewr=Anfuegen(dimewr,dimokt2,okt2,ewr);
      dimewr=Anfuegen(dimewr,dimokt3,okt3,ewr);
      dimewr=Anfuegen(dimewr,dimokt4,okt4,ewr);
      dimewr=Anfuegen(dimewr,dimokt5,okt5,ewr);
      dimewr=Anfuegen(dimewr,dimokt6,okt6,ewr);
      dimewr=Anfuegen(dimewr,dimokt7,okt7,ewr);
      dimewr=Anfuegen(dimewr,dimokt8,okt8,ewr);
      }
      else        /* mixed priority */
      {
      if(dimewr != (-1))
return(dimewr+1);
      }
      if(dimewr!=(-1))
 return(dimewr+1);
      else      /* es wird nach dem einzigsten naechsten gesucht */
{
```

```
        anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
if(infja==1)
          fpinf=fopen(finf,"r");
if(datei==1)
          fpsrt = fopen(fsrt,"r");
        for(i=0;i<(anzflae);i++)
          {
          if(anzflae<26)
           {
   if(infja==1)
    {
            offset= 54 + 16*(flaevek[i]-1);
            setbuf(fpinf,NULL);
            fseek(fpinf,offset,SEEK_SET);
            fscanf(fpinf,"%d",&flaenr);
            fscanf(fpinf,"%d",&von);
            fscanf(fpinf,"%d",&bis);
    }
           else
            {
            flaenr = inffeld[3*(flaevek[i]-1)+6];
            von = inffeld[3*(flaevek[i]-1)+7];
            bis = inffeld[3*(flaevek[i]-1)+8];
            }
            if(flaevek[i]!=flaenr)
              {
       printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
              putchar('\007');
              sleep(4);
              }
            }
          else
            von=0,bis=(anzDa-1),i=anzflae;
          if(von != (-1))
            for(j=von;j<=bis;j++)
              {
              if(datei==0)  /* Nutzung des Feldes urd  */
               pkt = urd[j];
              else  /* Nutzung des Files *.srt    */
                {
                offset = 75*j+10;
                fseek(fpsrt,offset,SEEK_SET);
                fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
               &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
             }
              dimewr=Vergleich(ewr,pkt,dimewr,s,schra2,EwR,AwR);
              }  /* fore j */
          }    /* fore i */
        if(infja==1)
          fclose(fpinf);
        if(datei==1)
          fclose(fpsrt);
}  /* Ende-es wird nach dem einzigsten naechsten gesucht */
      }
    while((EwR<(GEwR-addewr/2))&&(dimewr==(-1)));
    if(dimewr==(-1))
      return(-1);
    else
      return(dimewr+1);
    }
/* Ende fuer unvollstaendige Besetzung fuer mixed und
   Richtungsprioritaet*/

/*Zweig fuer Bereichserweitrg., Vollstaendige Besetzung
mit Bessect und ausschliesslicher Richtungsprioritaet */
  doja=0;
  if((GEwR!=(-1))&&(Bessect!=0))
```

```
    {
    schra1=Bessect-1;
    /* es fogt in "if" Ueberpruefung, ob alle Sektoren ausreichend
    besetzt sind, getrennt nach Oktant und Quadrant */
    if((((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
        ||(dimokt4<schra1)||(dimokt5<schra1)||(dimokt6<schra1)
        ||(dimokt7<schra1)||(dimokt8<schra1))&&(Richsect==8))||
        (((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
        ||(dimokt4<schra1))&&(Richsect==4)))
      {
      /* wenn mixed, dann Ueberpruefung der einzelnen Sektoren und bei
      Unterbesetzung Setzen der Dimension des Teifeldes auf -1 */
      if(NearRie==1)
        {
        schra2=Bessect;
        if(dimokt1<schra1)
          dimokt1=(-1);
        if(dimokt2<schra1)
          dimokt2=(-1);
        if(dimokt3<schra1)
          dimokt3=(-1);
        if(dimokt4<schra1)
          dimokt4=(-1);
        if(dimokt5<schra1)
          dimokt5=(-1);
        if(dimokt6<schra1)
          dimokt6=(-1);
        if(dimokt7<schra1)
          dimokt7=(-1);
        if(dimokt8<schra1)
          dimokt8=(-1);
        }
      else /* bei Richtungsprioritaet */
        schra2=Maxsect;
      /* ab jezt wieder gemeinsam mixed und Richtung */
      addewr = (GEwR-EwR)/ADDTEIL;
      do{
        if((NearRie==1)&&(doja==0));
/* Besetzung des der nicht vollen (sie stehen mittlerweile auf
-1) mit den eigentlich schon ausgewaehlten und im zweiten
Durchlauf ( wenn doja==1) geht die Bereichserweiterung los*/
        else
/* bei Richtung sofort Bereichserweiterung */
          {
          AwR = EwR;
  EwR += addewr;
          }
        anzflae=Flaechevek(finf,flaevek,EwR,infeldo);
if(infja==1)
          fpinf=fopen(finf,"r");
if(datei==1)
          fpsrt = fopen(fsrt,"r");
        for(i=0;i<(anzflae);i++)
          {
          if(anzflae<26)
           {
   if(infja==1)
    {
            offset= 54 + 16*(flaevek[i]-1);
            setbuf(fpinf,NULL);
            fseek(fpinf,offset,SEEK_SET);
            fscanf(fpinf,"%d",&flaenr);
            fscanf(fpinf,"%d",&von);
            fscanf(fpinf,"%d",&bis);
    }
           else
            {
```

```
          flaenr = inffeld[3*(flaevek[i]-1)+6];
          von = inffeld[3*(flaevek[i]-1)+7];
          bis = inffeld[3*(flaevek[i]-1)+8];
          }
          if(flaevek[i]!=flaenr)
            {
      printf("\nWarning: Here is the computed number %d not equal the index number %d\n",flaevek[i],flaenr);
          putchar('\007');
          sleep(4);
          }
        }
      else
        von=0,bis=(anzDa-1),i=anzflae;
      if(von != (-1))
        for(j=von;j<=bis;j++)
          {
          if(datei==0)  /* Nutzung des Feldes urd  */
           pkt = urd[j];
          else  /* Nutzung des Files *.srt    */
            {
            offset = 75*j+10;
            fseek(fpsrt,offset,SEEK_SET);
            fscanf(fpsrt,"%s%lf%lf%lf%lf%lf%lf",pkt.nr,&pkt.h,&pkt.r,
     &pkt.hor,&pkt.hh,&pkt.fre1,&pkt.fre2);
            }

          sec=(*Sector)(&pkt.h,&pkt.r);
          switch(sec)
            {
            case 1 :
/* Bedeutung der "if"-Ausdruecke: wenn NearRie==0,
dann wird der folgende Ausdruck immer abgearbeitet;
wenn NearRie==1 (mixed), dann wird er nur
abgearbeitet, solnage Bessect (Minimalbesetzung) noch
nicht erreicht ist */
              if(!((NearRie==1)&&(dimokt1>=schra1)))
               dimokt1=Vergleich(okt1,pkt,dimokt1,s,schra2,EwR,AwR);
              break;
            case 2 :
              if(!((NearRie==1)&&(dimokt2>=schra1)))
               dimokt2=Vergleich(okt2,pkt,dimokt2,s,schra2,EwR,AwR);
              break;
            case 3 :
              if(!((NearRie==1)&&(dimokt3>=schra1)))
               dimokt3=Vergleich(okt3,pkt,dimokt3,s,schra2,EwR,AwR);
              break;
            case 4 :
              if(!((NearRie==1)&&(dimokt4>=schra1)))
               dimokt4=Vergleich(okt4,pkt,dimokt4,s,schra2,EwR,AwR);
              break;
            case 5 :
              if(!((NearRie==1)&&(dimokt5>=schra1)))
               dimokt5=Vergleich(okt5,pkt,dimokt5,s,schra2,EwR,AwR);
              break;
            case 6 :
              if(!((NearRie==1)&&(dimokt6>=schra1)))
               dimokt6=Vergleich(okt6,pkt,dimokt6,s,schra2,EwR,AwR);
              break;
            case 7 :
              if(!((NearRie==1)&&(dimokt7>=schra1)))
               dimokt7=Vergleich(okt7,pkt,dimokt7,s,schra2,EwR,AwR);
              break;
            case 8 :
              if(!((NearRie==1)&&(dimokt8>=schra1)))
               dimokt8=Vergleich(okt8,pkt,dimokt8,s,schra2,EwR,AwR);
              break;
            default :
```

```
                printf("\nIt is not possible point assign to a sector\n");
                putchar('\007');
                sleep(4);
              }
          }  /* fore j */
        }    /* fore i */
      if(infja==1)
        fclose(fpinf);
      if(datei==1)
        fclose(fpsrt);
      doja=1;
      }
    while(((((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
      ||(dimokt4<schra1)||(dimokt5<schra1)||(dimokt6<schra1)
      ||(dimokt7<schra1)||(dimokt8<schra1))&&(Richsect==8))||
      (((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
      ||(dimokt4<schra1))&&(Richsect==4)))&&(EwR<(GEwR-addewr/2)));
    }
  if((((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
      ||(dimokt4<schra1)||(dimokt5<schra1)||(dimokt6<schra1)
      ||(dimokt7<schra1)||(dimokt8<schra1))&&(Richsect==8))||
      (((dimokt1<schra1)||(dimokt2<schra1)||(dimokt3<schra1)
      ||(dimokt4<schra1))&&(Richsect==4)))
    return(-1); /* wenn auch die Bereichserweiterung kein Erfolg ist */
  else
    {
    if((NearRie==1)&&(doja==0)) /* wenn mixed und Ber.erw nicht durchlaufen*/
     return(dimewr+1);
    else
    {
    dimewr = (-1);
    dimewr=Anfuegen(dimewr,dimokt1,okt1,ewr);
    dimewr=Anfuegen(dimewr,dimokt2,okt2,ewr);
    dimewr=Anfuegen(dimewr,dimokt3,okt3,ewr);
    dimewr=Anfuegen(dimewr,dimokt4,okt4,ewr);
    dimewr=Anfuegen(dimewr,dimokt5,okt5,ewr);
    dimewr=Anfuegen(dimewr,dimokt6,okt6,ewr);
    dimewr=Anfuegen(dimewr,dimokt7,okt7,ewr);
    dimewr=Anfuegen(dimewr,dimokt8,okt8,ewr);
    return(dimewr+1);
    }
    }
  }
 }
/*Ende Zweig fuer ausschliessliche Richtungsprioritaet
und mixed Priority */
}

/*-------------------------------------------------------*/

int Anfuegen(int dim,int dimokt,struct dat *okt,struct dat *ewrr)
{
int i,j;
if((dim==(-1))&&(dimokt==(-1)))
  return(-1);
if((dim!=(-1))&&(dimokt==(-1)))
  return(dim);
for(i=(dim+1),j=0;i<=(dim+dimokt+1);i++,j++)
   ewrr[i]=okt[j];
return(--i);
}
/*-------------------------------------------------------*/

int Flaechevek(char *filename,int *flaevektt,int EwR,double *inffeldo)
{
int i,ii,li,re,ho,un,li1,re1;
double s,z,nummer,hochlio,rechtsliu,del,delhoch,delrechts;
```

```
double rechtsli,rechtsre,hochob,hochun,spal,zeil,flaevekt[28];
FILE *fp;
if(infja==1)
{
fp=fopen(filename,"r");
fscanf(fp,"%lf%lf%lf%lf%lf%lf",&hochlio,&rechtsliu,&spal,&zeil,
       &delhoch,&delrechts);
fclose(fp);
}
else
{
hochlio=inffeldo[0],rechtsliu=inffeldo[1],spal=inffeldo[2],zeil=inffeldo[3];
delhoch=inffeldo[4],delrechts=inffeldo[5];
}

if(delhoch>= delrechts)  /* del ist das kleinere delta */
  del=delrechts;
else
  del=delhoch;


s=ceil((rechtsVor-rechtsliu)/(delrechts));  /* Spalte */ /*ceil=Abrunden*/
z=ceil((hochlio-hochVor)/(delhoch));    /* Zeile  */
if(s<=0 || z<=0)
  {
  printf("\nHW:%9.3f, RW:%9.3f dont lie in sort grid!\n",hochVor,rechtsVor);
  putchar('\007');
  return(28); /* alle Punkte werden bei der Auswahl genutzt */
  }
if((rechtsliu +spal*delrechts)<= rechtsVor)
   return(28);
if((hochlio -zeil*delhoch)>= hochVor)
   return(28);
z--;
nummer = spal*z+s;

flaevekt[0]=nummer;
rechtsre = (s*delrechts)+rechtsliu;
rechtsli = rechtsre-delrechts;
hochob = hochlio-(z*delhoch);
hochun = hochob-delhoch;

i=1,re=li=0;
if(((rechtsVor-rechtsli+EwR/DIV)<EwR)&&(fmod((nummer-1),spal)!=0))
  {
  flaevekt[i]=nummer-1;
  i++,li=1;
  }
if(((rechtsre-rechtsVor+EwR/DIV)<EwR)&&(fmod(nummer,spal)!=0))
  {
  flaevekt[i]=nummer+1;
  i++,re=1;
  }

if(((hochob-hochVor+EwR/DIV)<EwR)&&((nummer-spal)>0))
  {
  flaevekt[i]=nummer-spal;
  i++,ho=1;
  if(li==1)
    {
    flaevekt[i]=nummer-spal-1;
    i++;
    }
  if(re==1)
    {
    flaevekt[i]=nummer-spal+1;
    i++;
```

```
    }
  }
if(((hochVor-hochun+EwR/DIV)<EwR)
              &&((nummer+spal)<=(spal*zeil)))
  {
  flaevekt[i]=nummer+spal;
  i++,un=1;
  if(li==1)
    {
    flaevekt[i]=nummer+spal-1;
    i++;
    }
  if(re==1)
    {
    flaevekt[i]=nummer+spal+1;
    i++;
    }
  }

if(EwR > (1.5*delrechts))
  {
  if(((rechtsVor-rechtsli+delrechts+EwR/DIV)<EwR)
                   &&(fmod((nummer-2),spal)!=0))
    {
    flaevekt[i]=nummer-2;
    i++,li1=1;
    if(ho==1)
      {
      flaevekt[i]=nummer-2-spal;
      i++;
      }
    if(un==1)
      {
      flaevekt[i]=nummer-2+spal;
      i++;
      }
    }
  if(((rechtsre-rechtsVor+delrechts+EwR/DIV)<EwR)
&&((fmod((nummer+1),spal)!=0)&&(fmod(nummer,spal)!=0)))
    {
    flaevekt[i]=nummer+2;
    i++,re1=1;
    if(ho==1)
      {
      flaevekt[i]=nummer+2-spal;
      i++;
      }
    if(un==1)
      {
      flaevekt[i]=nummer+2+spal;
      i++;
      }
    }
  }

if(EwR>(1.5*delhoch))
{
if(((hochob-hochVor+delhoch+EwR/DIV)<EwR)
                      &&((nummer-2*spal)>0))
  {
  flaevekt[i]=nummer-2*spal;
  i++;
  if(li==1)
    {
    flaevekt[i]=nummer-2*spal-1;
    i++;
    if(li1==1)
```

```
      {
      flaevekt[i]=nummer-2*spal-2;
      i++;
      }
    }
  if(re==1)
    {
    flaevekt[i]=nummer-2*spal+1;
    i++;
    if(re1==1)
      {
      flaevekt[i]=nummer-2*spal+2;
      i++;
      }
    }
  }
if(((hochVor-hochun+delhoch+EwR/DIV)<EwR)
          &&((nummer+2*spal)<=(spal*zeil)))
  {
  flaevekt[i]=nummer+2*spal;
  i++;
  if(li==1)
    {
    flaevekt[i]=nummer+2*spal-1;
    i++;
    if(li1==1)
      {
      flaevekt[i]=nummer+2*spal-2;
      i++;
      }
    }
  if(re==1)
    {
    flaevekt[i]=nummer+2*spal+1;
    i++;
    if(re1==1)
      {
      flaevekt[i]=nummer+2*spal+2;
      i++;
      }
    }
  }
}  /*     ewr >1.5*delhoch und delrechts  */
  for(ii=0;ii<i;ii++)          /* Zuweisung double an int-Zeiger */
    flaevektt[ii]=flaevekt[ii];

if(EwR > (2.5*del))
  return(28);

return(i);
}
/*-------------------------------------------------------------*/

int Octant(float *hi,float *ri)
{
if(*hi >= hochVor)   /* obere Haelfte(1-4) */
  if(*ri <= rechtsVor) /* (1,2)  */
    if((*hi-hochVor) <= (rechtsVor-*ri))
      return(1);
    else
      return(2);
  else
    if((*hi-hochVor) >= (*ri-rechtsVor))
      return(3);
    else
      return(4);
```

```
else      /* untere Haelfte (5-8) */
  if(*ri > rechtsVor)
    if((hochVor-*hi) <= (*ri-rechtsVor))
      return(5);
    else
      return(6);
  else
    if((hochVor-*hi) >= (rechtsVor - *ri))
      return(7);
    else
      return(8);
}
/*-----------------------------------------------------------*/


int Quadrant(float *hi,float *ri)
{
if(*hi >= hochVor)   /* obere Haelfte(1,2) */
  if(*ri <= rechtsVor) /* (1)  */
    return(1);
  else
    return(2);

else      /* untere Haelfte (3,4) */
  if(*ri > rechtsVor)
    return(3);
  else
    return(4);
}
/*-----------------------------------------------------------*/

int Vergleich(struct dat *okt,struct dat pkt, int dimokt,
       int sch,int constt,int EwR,int AwR)
{
int i;
unsigned width;
struct dat temp;

/* Berechnung Strecke Pkt. -  Vor  und Zuordnung fre3 */
pkt.fre3=sqrt((rechtsVor-pkt.r)*(rechtsVor-pkt.r)
+(hochVor-pkt.h)*(hochVor-pkt.h));

/* Test auf Enthaltensein im Suchbereich */
if((pkt.fre3>= EwR)||(pkt.fre3<AwR))
  return(dimokt);

if(dimokt<(constt-1)) /* Feld ist noch nicht voll besetzt*/
  {
  if((sch==1)||(sch==2)) /* Stuetzpunkte Z, Y */
    {
    if((pkt.hh==0)&&(dimokt<0)) /* ist Streuung gleich NULL
     bei erster Besetzung  */
      {
      okt[0]=pkt;
      dimokt++;
      return(dimokt);
      }
    if((pkt.hh==0)&&((okt[0].hh != 0)||((okt[0].hh==0)&&
                  (okt[0].fre3 > pkt.fre3))))
    /* ist Streuung des 1. und aktuellen Elementes gleich NULL
    ab zweiter Besetzung */
      {
      for(i=(dimokt+1);i>0;i--)
okt[i]=okt[i-1];
      okt[0]=pkt;
      }
    else
```

```
      okt[dimokt+1]=pkt;
    dimokt++;
    }      /* Ende Stuetzpunkte Z, Y */
  else     /* Gradienten G */
    {
    if(dimokt<0)
      okt[0]=pkt;
    else
      okt[dimokt+1]=pkt;
    dimokt++;
    }
  /*  Sortierung, weil okt voll besetzt ist */
  if(dimokt == (constt-1))
    {
    width=sizeof(struct dat);
    if((okt[0].hh==0)&&((sch==1)||(sch==2)))
      qsort(&okt[1],constt-1,width,comp);
    else
      qsort(okt,constt,width,comp);
    }
  }
else  /* dimokt >= constt, Feld ist bereites voll besetzt */
  {
  if((sch==1)||(sch==2)) /* Stuetzpunkte Z, Y */
    if((pkt.hh==0)&&((okt[0].hh!=0)||((okt[0].hh==0)&&
       (okt[0].fre3>pkt.fre3))))
      {
      for(i=dimokt;i>0;i--)
okt[i]=okt[i-1];
      okt[0]=pkt;
      }
    else
      {
      if(okt[constt-1].fre3>pkt.fre3)
{
i=constt-1;
        okt[constt-1]=pkt;
while((okt[i-1].fre3>okt[i].fre3)&&
   (i>((okt[0].hh==0) ? 1 : 0)))
         {
  temp=okt[i-1];
  okt[i-1]=okt[i];
  okt[i]=temp;
  i--;
  }
       }
     }
  else  /* Gradienten G */
    {
      if(okt[constt-1].fre3>pkt.fre3)
{
i=constt-1;
        okt[constt-1]=pkt;
while((okt[i-1].fre3>okt[i].fre3)&& (i> 0))
         {
  temp=okt[i-1];
  okt[i-1]=okt[i];
  okt[i]=temp;
  i--;
  }
       }
     }
  }
return(dimokt);
}
/*-----------------------------------------------------------*/
```

```
int comp(struct dat *i,struct dat *j)
{
if(i->fre3 > j->fre3)
  return(1);
if(i->fre3 < j->fre3)
  return(-1);
else
  return(0);
}
/*---------------------------------------------------------*/



int Singueberpr(struct dat *ewr, int *enddim,int sch,float k,
      float (*Snum)(float *s,float k))
{
int weg[MAX_SZG+2],ohwei,i,j;
float dh,dr,sij,ss,s0;

for(i=0;i<=MAX_SZG;i++)
  weg[i]=1;

ohwei=0; /* Erhoehung bei Streichen von Punkten */

i=0;
while(i<= *enddim)
  {
  if(weg[i]==1)
  {
  j=i+1;
  while(j<= *enddim)
    {
    if(weg[j]==1)
      {
      dh=ewr[i].h-ewr[j].h;
      dr=ewr[i].r-ewr[j].r;
      sij=sqrt(dh*dh+dr*dr);
      s0=(ewr[i].fre3+ewr[j].fre3)/2;
      ss= (*Snum)(&s0,k);
      if(sij<ss)
{
        ewr[i].r=(ewr[i].r+ewr[j].r)/2;
        ewr[i].h=(ewr[i].h+ewr[j].h)/2;
        ewr[i].hh=(ewr[i].hh+ewr[j].hh)/2;
        ewr[i].hor=(ewr[i].hor+ewr[j].hor)/2;
        ewr[i].fre3=(ewr[i].fre3+ewr[j].fre3)/2;
if(sch==3)
  {
          ewr[i].fre2=(ewr[i].fre2+ewr[j].fre2)/2;
          ewr[i].fre1=(ewr[i].fre1+ewr[j].fre1)/2;
  }
       weg[i]=0,ohwei++;
}
      }
    j++;
    }
  }
  i++;
  }


/* Streichen der Punkte mit weg == 0 */
if(ohwei != 0)
  {
  i=0,j=0;
  while(i<= *enddim)
    {
```

```
    if(weg[i+j] == 1)
      {
      ewr[i]=ewr[i+j];
      i++;
      }
    else
      j++,(*enddim)--;
    }
  }
return(ohwei);
}


float Snumi(float *s,float k)
{
return(k);
}
```

Literaturverzeichnis

[AS88] Hikmet Akin and Heinrich Siemes. *Praktische Geostatistik.* Springer–Verlag, Berlin, Heidelberg, 1988.

[DJ92] Clayton V. Deutsch and Andre G. Journel. *GSLIB - Geostatistical Software Libary and User's Guide.* Oxford University Press, New York, Oxford, 1992.

[Har90] Lutz Harnos. Extrapolation im Großtagebau Welzow Süd. Master's thesis, Bergakademie Freiberg, 1990.

[IS89] Edward H. Isaaks and R. Mohan Srivastava. *Applied Geostatistics.* Oxford University Press, New York, Oxford, 1989.

[KR90] Brian W. Kernighan and Dennis M. Ritchie. *Programmieren in C.* Carl Hanser Verlag and Prentice-Hall Inc., München, Wien and London, 1990.

[Sed92] Robert Sedgewick. *Algorithmen in C.* Addison–Wesley (Deutschland) GmbH, Bonn, München, 1992.