Suche, Planung und Scheduling

Jürgen Dorn

Tel. 58801-18426
Email: dorn@dbai.tuwien.ac.at
http://www.dbai.tuwien.ac.at/staff/dorn/

Sprechstunde: Donnerstag 16–17 Uhr

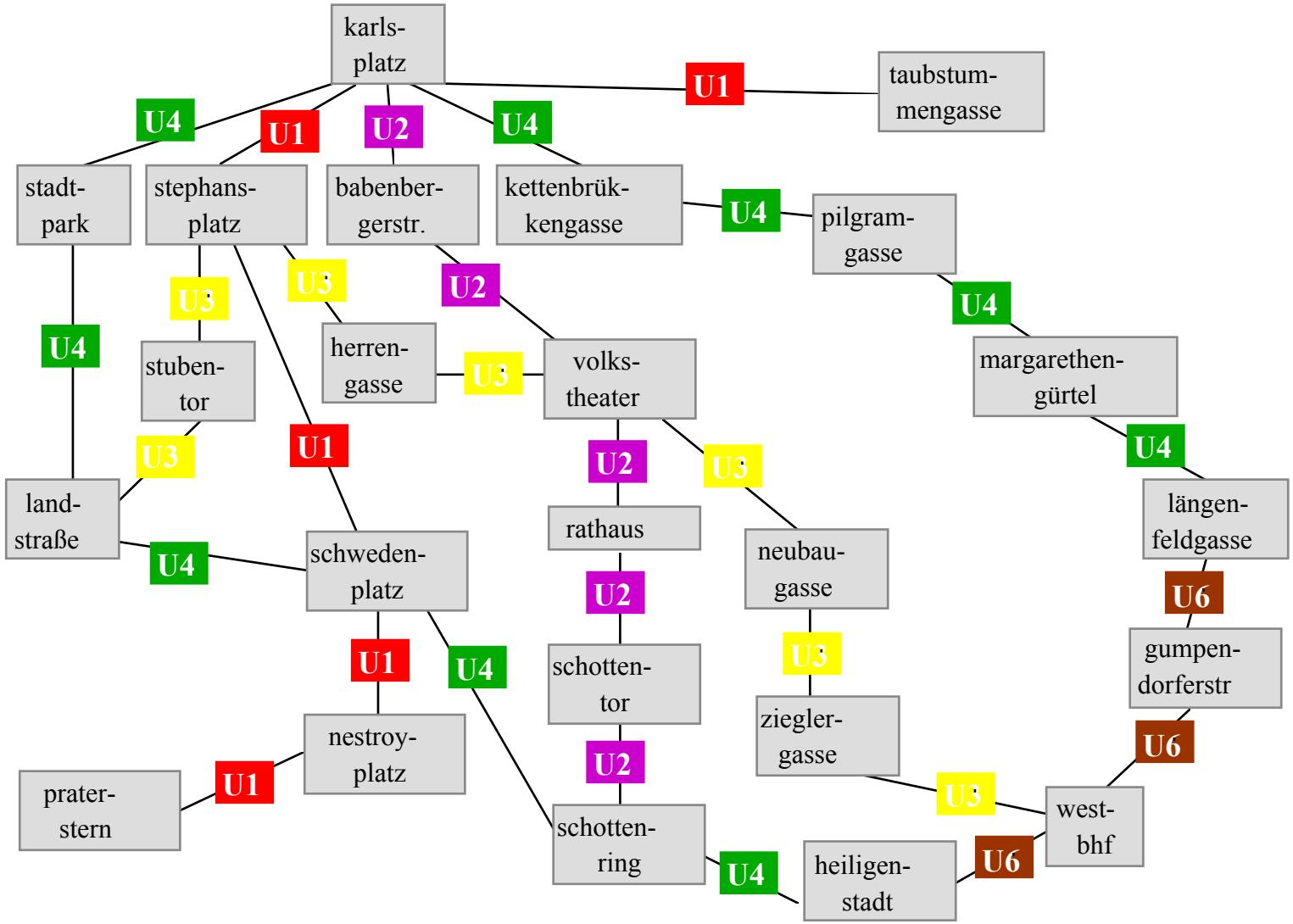
Gliederung

- Wissensbasierte Suche
  - uninformierte Suche
  - A* und IDA* Algorithmus
  - Suche in Und/Oder-Graphen
- Wissensbasierte Planung
  - Wissensrepräsentation
  - lineare Planung
  - Erweiterungen
- Wissensbasiertes Scheduling
  - Wissensrepräsentation
  - Weiche Einschränkungen
  - Tabusuche
  - Anwendung im Edelstahlwerk

Wissensbasierte Suche

- Suche in (expliziten oder impliziten) Graphen
- Anwendung z.B. Wegsuche
  - Punkt bzw. Knoten (z.B. ein Ort)
  - Kante (direkte Verbindung zwischen zwei Orten)
  - Verzweigungsfaktor (wieviele Wege gehen von einem Punkt weg)
  - Kanten können Kosten zugeordnet sein (Dauer einer Fahrt, Länge der Strecke, Fahrpreis, …)
- Problem: finde kostengünstigsten Weg zwischen zwei Punkten

U-Bahn Wegegraph



Ein allgemeiner Suchalgorithmus

- Start- und Zielpunkt: p_s, p_z
- Suchgraph: Graph und Suchbaum: Tree
- bereits untersuchte Punkte: Closed
- als nächstes zu untersuchende Punkte: Open
- Sortieren von Open: heuristic_sort

```
function search(p_S, p_Z, Path) : boolean;
   begin
     Graph:=p_S; Tree:= p_S; Open := p_S; Closed := ∅; p_i := p_S;
     while p_Z ∉ Open and Open ° ∅ do
       p_i := first(Open);
       for all p_j := suc(p_i) do
         Graph := add(Graph, edge(p_i, p_j));
         if not p_j in Open and not p_j in Closed then
             Tree := add(Tree, edge(p_i, p_j));
             Open := Open + p_j
         end if
       end for all;
       Open := Open - p_i;
       Closed := Closed + p_i;
       heuristic_sort(Open);
     end while;
     IF p_i = p_Z then
        Path := p_Z;
        p_i := p_Z;
        repeat
          p_i = pre(p_i, Tree);
          Path := p_i + Path
        until p_i = p_S;
        return search := true
       else search := false
     end if
   end function search;
```

uninformierte Suchstrategien

- Tiefensuche und Backtracking
  - tiefere Punkte nach vorn in Open Liste
  - möglicherweise lange Suchzeit
  - Tiefenschranke notwendig
- Breitensuche
  - höhere Punkte nach vorn in Open Liste
  - kürzester Weg (Anzahl der Kanten), aber großer Speicherbedarf
- Suche mit iterativer Vertiefung
  - erster Schritt, Suche mit Backtracking bis zur Tiefe 1
  - die Suchtiefe wird dann beim nächsten Versuch um 1 erhöht
  - wiederholen der Backtrackingsuche bis Weg gefunden
  - Kombination der Vorteile von Breitensuche und Backtracking
  - kürzester Weg wird gefunden und Speicheraufwand wird verringert

informierte heuristische Suchstrategien

- Punkte, für die kleinere Kosten berechnet werden, stehen weiter vorne in der Liste „Open”
- die Güte eines Weges vom Startpunkt p_s zu einem Zielpunkt p_z über einen Zwischenpunkt wird geschätzt
  - Kostenfunktion für eine Kante $k(p_i, p_j)$
  - $h^*(p)$ sind die optimalen Kosten von p nach p_z
  - $g^*(p)$ sind die optimalen Kosten von p_s nach p
  - $f^*(p) = g^*(p) + h^*(p)$
  - diese Kosten sind aber unbekannt
  - deswegen werden dafür Schätzfunktionen eingeführt
  - $h(p)$ ist die Schätzfunktion für die Kosten von p nach p_z
  - $g(p)$ ist die Schätzfunktion für die Kosten von p_s nach p
  - $f(p) = g(p) + h(p)$

Algorithmus A*

- **Monotonie**
  - Monotoniebedingung von A*: $0 < h(p) \leq h^*(p)$
  - h(p) ist monoton, wenn $h(p_i) - h(p_j) \leq \text{kosten}(p_i, p_j)$
- **Zulässigkeit**
  - ein Suchalgorithmus ist zulässig, wenn zugesichert ist, dass er einen Weg findet, soweit dieser existiert
  - A* ist zulässig, wenn der Graph endlich ist
- **Effizienz**
  - die Auswahl von h(p) ist entscheidend für die Effektivität der heuristischen Suche
  - h(p) = 0 und einheitliche Kosten der Kanten → Breitensuche
  - Komplexität im Extremfall wie Breitensuche

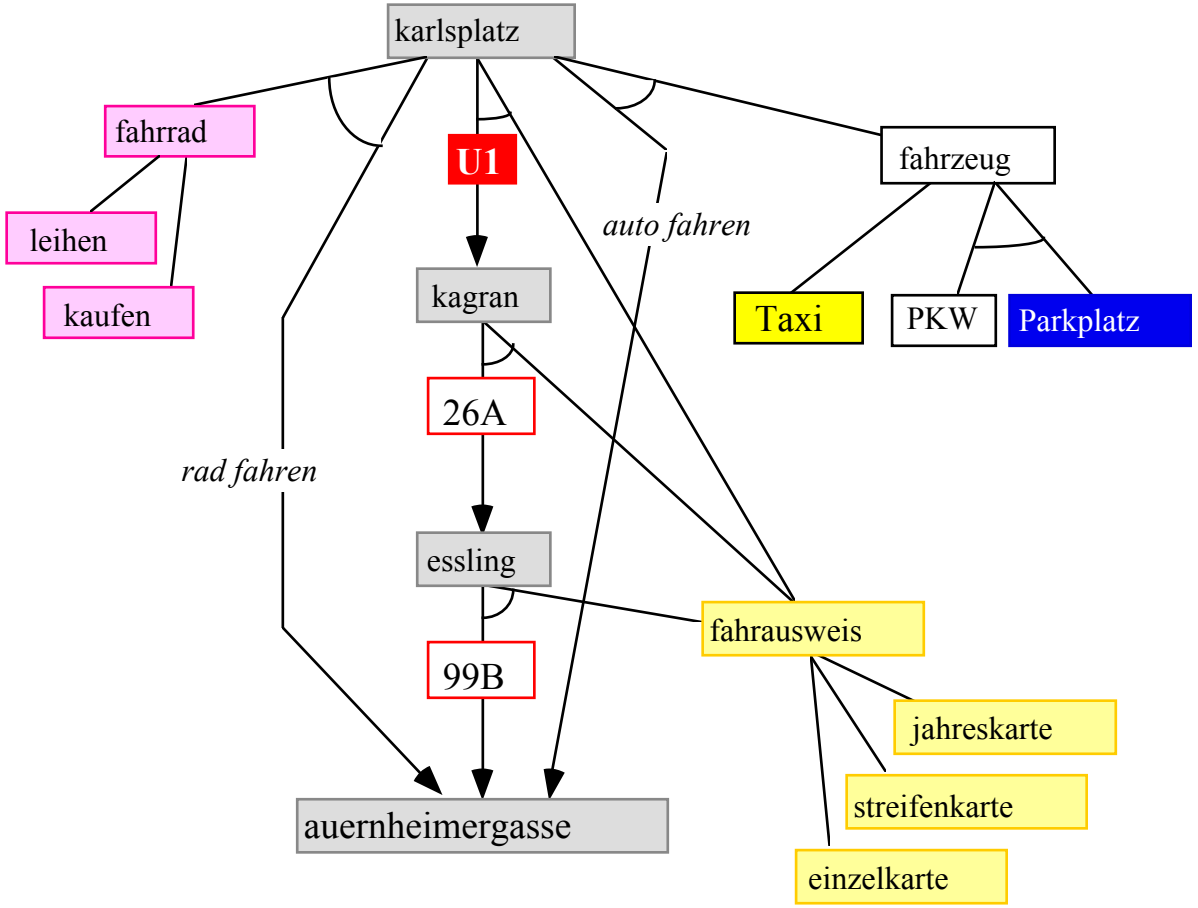
IDA*

- Kombination von Algorithmus A* und Backtracking
- Backtracking benutzt die geschätzten Kosten als Tiefenschranke
- die geschätzten Kosten zwischen p_s und p_z sind Tiefenschranke
- mit Backtracking nach Lösung suchen
- wird keine Lösung gefunden, erhöhen der Tiefenschranke auf den minimalen Wert, der beim Backtracking zum Abbruch führte
- Vorteil: linearer Speicheraufwand im Gegensatz zum exponentiellem Aufwand bei A*
- der Zeitaufwand ist theoretisch größer als bei A*
- praktisch wurde jedoch nachgewiesen, dass der Zeitaufwand nicht viel größer ist

Und/Oder Graphen

- bisher vorgestellte Graphen sind Oder-Graphen
- wenn verschiedene Bedingungen gleichzeitig erfüllt sein sollen
  → Repräsentation mit Und/Oder-Graphen
- Und/Oder-Graphen sind Hypergraphen
  - es existieren Kanten zwischen Mengen von Punkten
  - Lösungen sind nicht mehr Lösungswege sondern Lösungsgraphen
  - das Ziel kann eine Menge von Punkten sein
- der Algorithmus A0* benutzt einen vielversprechendsten Lösungsgraphen GP

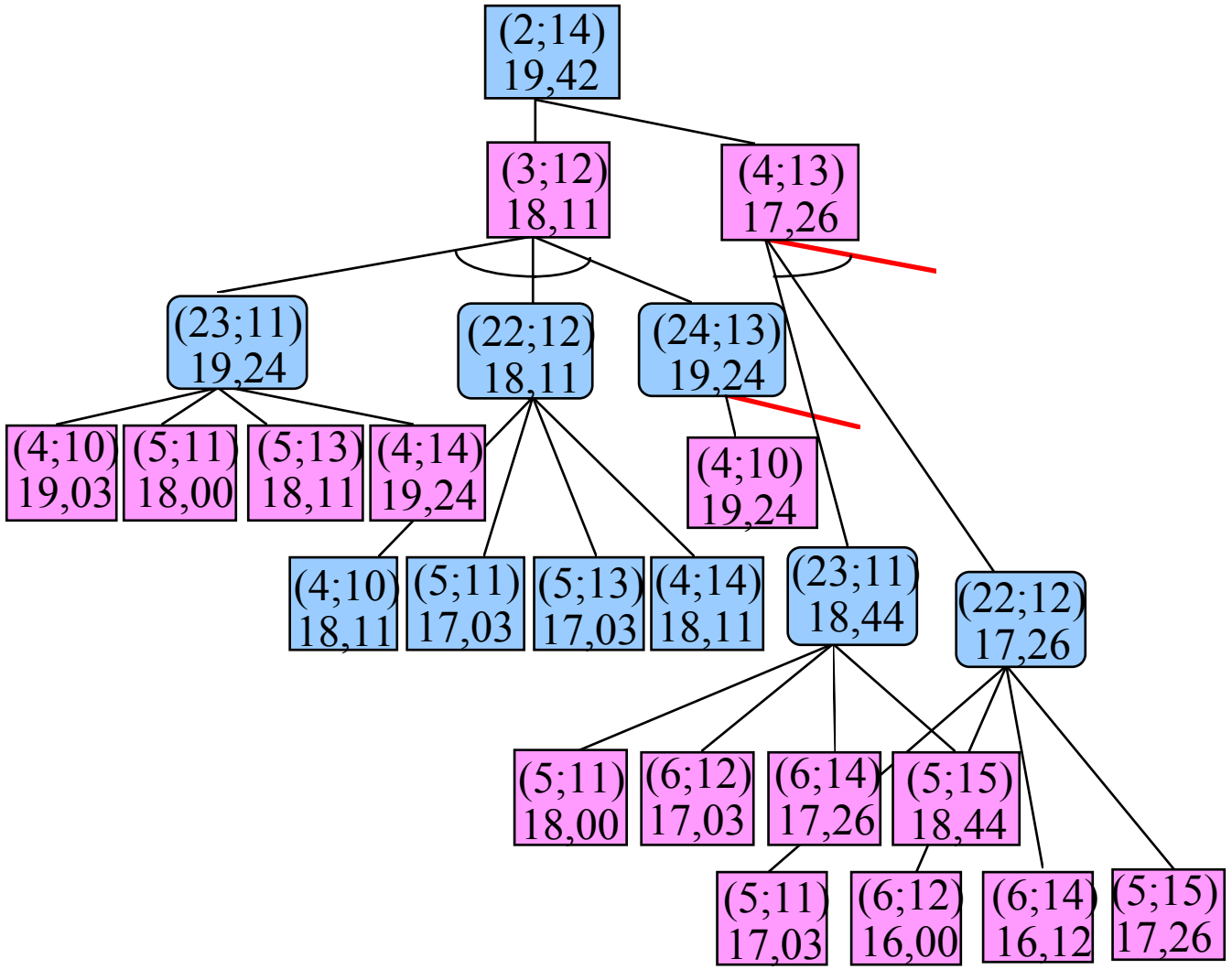
Beispiel


karlsplatz
fahrrad
U1
fahrzeug
leihen
kaufen
auto fahren
kagran
Taxi
PKW
Parkplatz
26A
rad fahren
essling
fahrausweis
99B
jahreskarte
streifenkarte
auernheimergasse
einzelkarte

Entscheidungen bei Zielkonflikten

- Suche in Nullsummenspielen
- der Gewinn des einen Spielers ist gleich hoch wie der Verlust des anderen Spielers (z.B. Schach)
  - wenn die Gegner nun abwechselnd eine Aktion ausführen, muss Spieler A immer damit rechnen, dass der Gegenspieler B, die für Spieler A ungünstigste Alternative wählt
  - der sogenannte Spielbaum wird wie ein Und/Oder-Graph repräsentiert
- die Suche wird jedoch anders durchgeführt
- Expandierung bis zum Suchhorizont
- Bewertung der Blätter
- Heraufpropagierung mit MinMax Strategie
- möglicherweise α-β Beschneidung

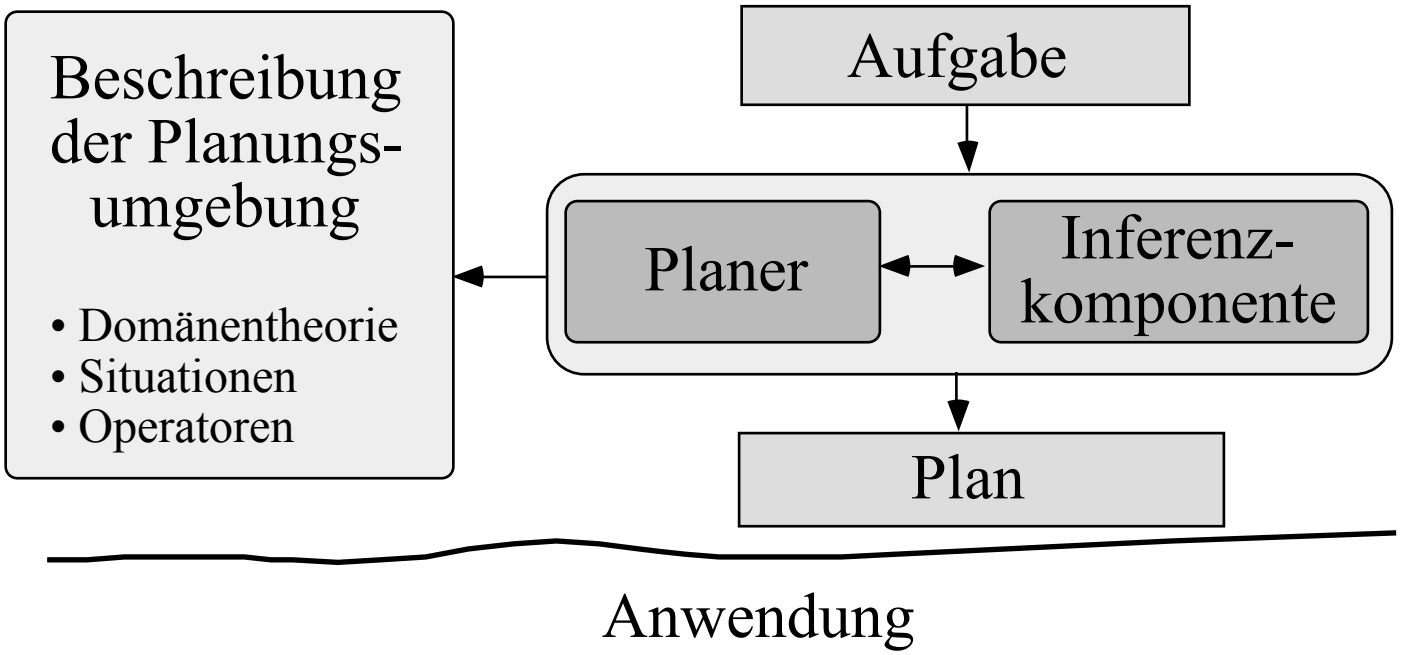
Spielbaum


(2;14)
19,42
(3;12)
18,11
(4;13)
17,26
(23;11)
19,24
(22;12)
18,11
(24;13)
19,24
(4;10)
19,03
(5;11)
18,00
(5;13)
18,11
(4;14)
19,24
(4;10)
19,24
(4;10)
18,11
(5;11)
17,03
(5;13)
17,03
(4;14)
18,11
(23;11)
18,44
(22;12)
17,26
(5;11)
18,00
(6;12)
17,03
(6;14)
17,26
(5;15)
18,44
(5;11)
17,03
(6;12)
16,00
(6;14)
16,12
(5;15)
17,26

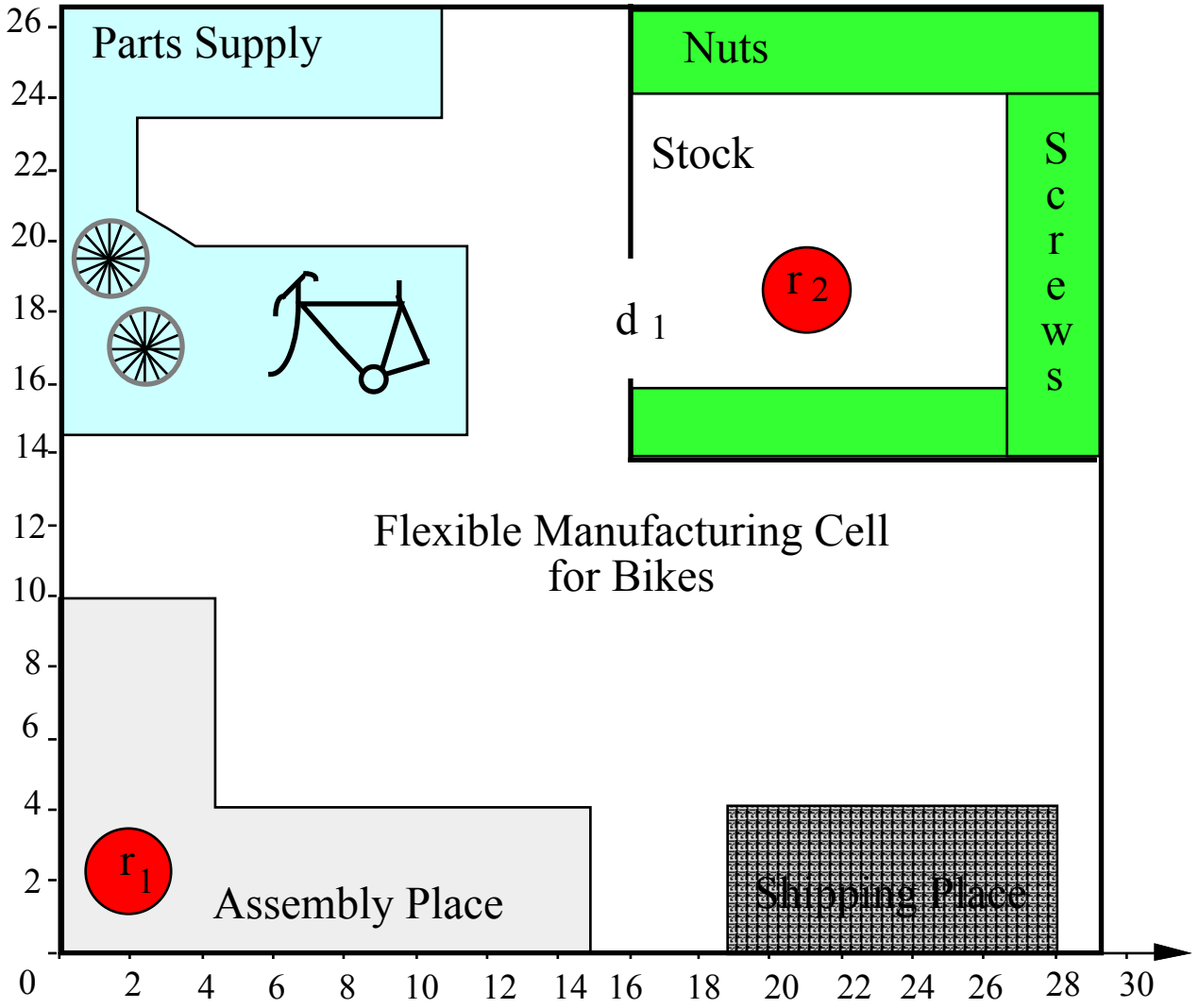
Wissensbasierte Planung

- Handlungsplanung
- Was muss getan werden, damit in der Zukunft eine Aussage gilt (z.B. ein Fahrrad produziert ist)
- hier Einschränkung auf statische Probleme
- Dynamische Probleme (Echtzeitplanung)
  - Echtzeitreaktion
  - während der Planung verändert sich die Umgebung
  - Adaptieren von alten Plänen
  - Zuverlässigkeit und Sicherheit

Architektur



Beispiel


Parts Supply
Nuts
Stock
Screws
r 2
d 1
Flexible Manufacturing Cell
for Bikes
r 1
Assembly Place
Shipping Place
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30

Domänentheorie

- Objektklassen und Instanzen

  frame(robot, agent, [capacity, load]).
  frame(assembly_robot, robot, []).
  exist(r_2, mobile_robot, [capacity : 1500]).
  frame(place, self, [region, [x, y]]).
  place(parts_supply, [cell, [3, 29])].

- Zugriffsfunktionen

  r_2 is_a robot. r_2 at parts_supply.
  r_2 has bike_frame. r_2 in cell.
  bike parts [bike_frame, front_wheel, rear_wheel].
  screws contains screw.

- Domänentheorie

  imposs(A_1 at O_1, A_2 at O_2) :- A_1 == A_2, O_1 \= O_2.
  is_a(O, object) :-
  frame(O, object, _),
  exist(_, container, A),
  member(contains : O, A).

Situationen

- Situationskalkül
  - Menge von Aussagen die gleichzeitig existieren
  - Startsituation
  - Zielsituation
  - Situationsübergänge
  - Planen als Suche nach adäquaten Situationsübergängen
  - Reihenfolgeproblem
- Beispiel einer Situation

  r_1 at assembly_place.
  r_2 at stock.
  bike_frame at parts_supply.
  front wheel at parts_supply.
  rear_wheel at parts_supply.
  open d_1.

Planungsoperatoren

- vier Listen:
  - Einschränkungen (Bedingungen an die Instantiierung der Variablen)
  - Vorbedingungen (was muss gelten, damit der Operator ausgeführt werden kann)
  - Hinzufügungen (was gilt nach der Ausführung)
  - Aufhebungen (was gilt nicht mehr nach der Ausführung)
- Beispiel

```
operator(move(R, P1, P2) :: [R is_a mobile_robot, P2 is_a place,
  P2 in Ro, Ro is_a room, P1 in Ro, P1 is_a place] ::
  [R in Ro, R at P1] :: [R at P2] :: [R at P1]).
```

- Sieben Operatoren für Beispiel
  - move(R, P_1, P_2)
  - enter(R, R_1, R_2)
  - open(R, D)
  - take_from(R, C, O)
  - take(R, O, P)
  - put(R, O, P)
  - assemble(R, O)

Planungstrategie

- Means-End-Analyse
  1. wende Operator an, der die Differenz zwischen Ziel und Startsituation verringert

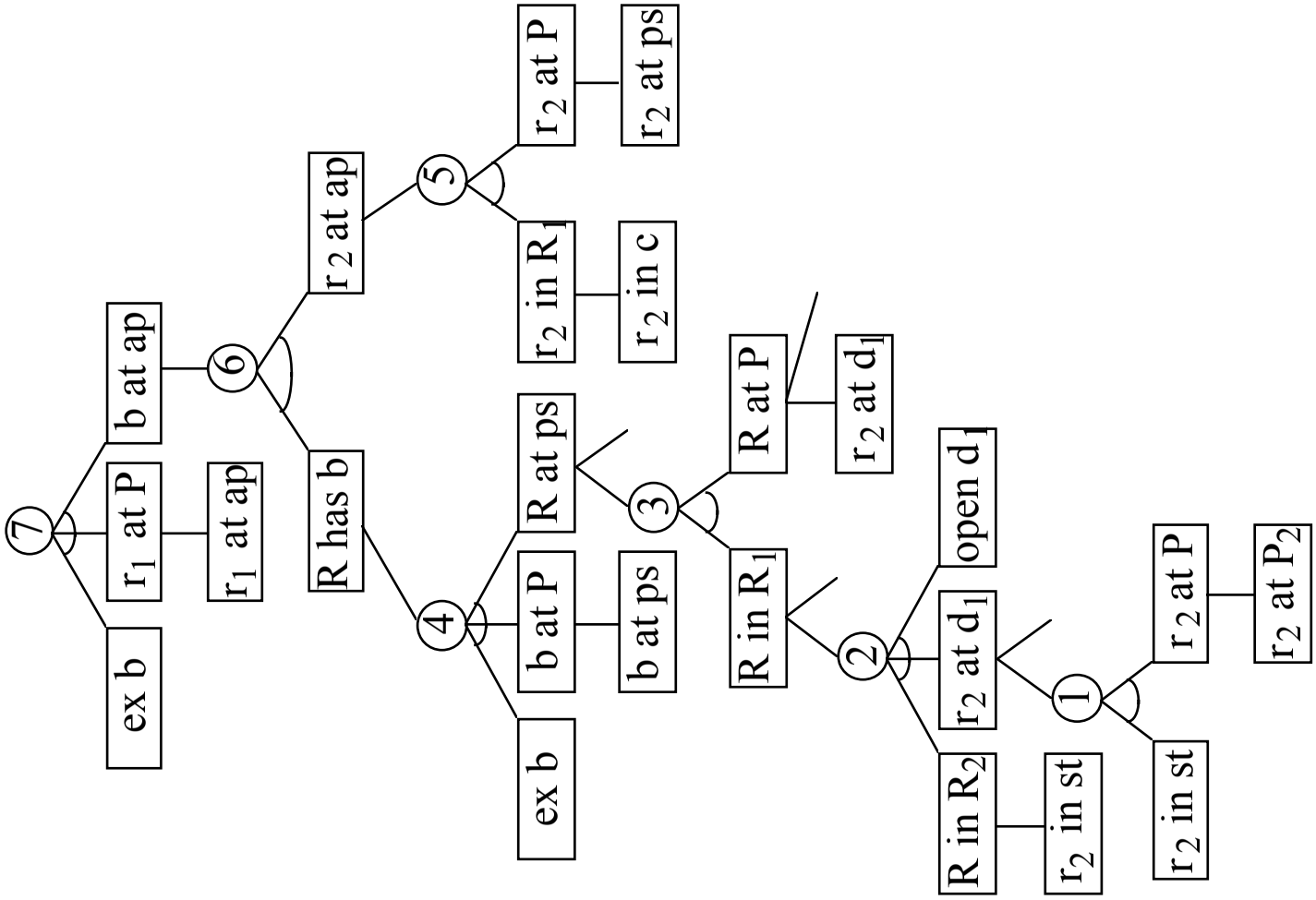
     welcher Operator fügt Ziele ein
  2. wende Operator an, der die Differenz zwischen Vorbedingungen des Operators und Startsituation verringert
- Zielsituation: `r1 has bike_frame`
- gefundenener Operator: `take(r1, bike_frame)`
- Vorbedingungen: `exist bike_frame, r1 at P,bike_frame at P`
- Endgültiger Plan: `move(r2,stock,d1) => enter(r2,stock,cell) => move(r2,d1,parts_supply) => take(r2,bike_frame) => move(r2,parts_supply,assembly_place) => put(r2,bike_frame) => take(r1,bike_frame)`

„nicht-deterministische" Entscheidungen

1. welches Teilziel einer Liste wird zuerst gelöst
   - bisher Reihenfolge der Liste, dann Permutation
2. unterschiedliche anwendbare Schemata
   - Heuristik: jeweils spezifischeren Operator auswählen
3. Unifikation der Literale (Zielaussagen)
   - möglichst zuerst die instantiieren, die weniger Alternativen bieten

- die Stellen der „Nichtdeterminiertheit" sind potentielle Punkte für Backtracking
- Entscheidung durch Heuristiken
- entsprechend der Strategie der geringsten Festlegung (least-commitment strategy)

Planen als lineare Tiefensuche



Verbesserungen der linearen Planung

- Nichtlineare Planung
  - einzelne Teilpläne werden ineinander kombiniert
  - regressive Planung (Einschub von Operatoren in geeignete Stellen des Plans)
- Einschränkungsbasierte Planung (constraint-based)
  - frühzeitige Reduzierung der Wertebereiche von Variablen
- Hierarchische Planung
  - Situationsabstraktion durch Bewertung von Aussagen von Situationen und Vorbedingungen
  - Operatorabstraktion

Wissensbasiertes Scheduling

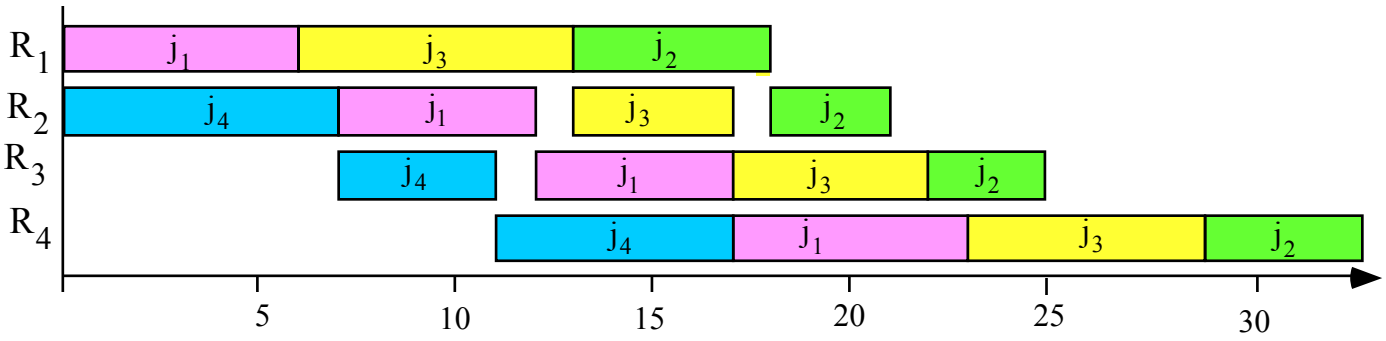
- die sog. Prozesspläne sind bekannt
  (also wie produziere ich ein bestimmtes Fahrrad)
- Problem
  - eine Menge von Aufträgen für unterschiedliche Produkte, die unterschiedliche Bearbeitungen erfordern
  - verfügbare Ressourcen sollen optimal genutzt werden
    - Roboter soll immer beschäftigt sein
    - Lager soll immer möglichst klein sein (wg. Kapitalbindung)
  - Aufträge haben geforderte Fertigstellungszeiten, die möglichst gut eingehalten werden sollen
  - in realen Umgebungen meist noch viele weitere harte technologische und weiche organisatorische Einschränkungen

Beispiel

- vier Fahrradtypen
  - Fahrrad „Lifestyle” (j_1), Fahrrad „Öko” (j_2)
  - Fahrrad „Lion King” (j_3), Fahrrad „Giro” (j_4)
- vier Ressourcen
  - Schweißroboter (R_1), Sprühroboter (R_2), Transportroboter (R_3), Montageroboter (R_4)
- Prozesspläne

Jobs	R_1	R_2	R_3	R_4
j_1	6	5	5	6
j_2	5	3	3	4
j_3	0	7	4	6
j_4	7	4	5	6

- Plan



Verfahren

Problem: Komplexität der Suche und der Beschreibung

- Zwei Teilprobleme: Bestimmung der Reihenfolge und das Festlegen von Zeiten
- meist Suche nach semi-aktivem Schedule (nach der Bestimmung der Reihenfolge, Operationen so früh wie möglich einplanen)

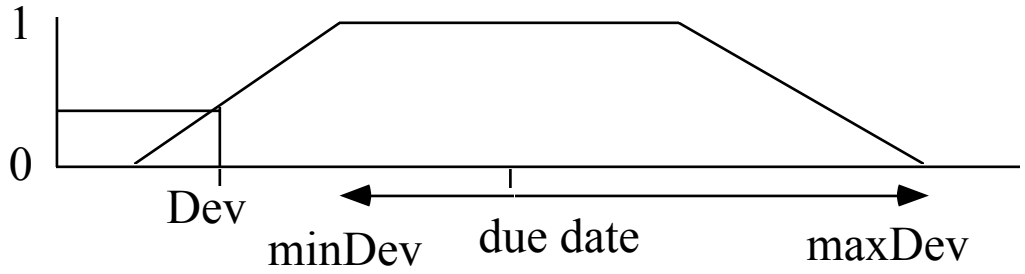
- Konstruktive Verfahren (kein Backtracking, linearer Aufwand)
- explizite Enumeration (Aufzählung aller Lösungen)
- implizite Enumeration (Beschneidung der Suche)
- Integer oder Mixed Integer Programmierung
- Heuristisches Dispatching
- Regelbasiertes Scheduling
- Einschränkungsbasiertes Scheduling
- Iterative Optimierung

Scheduling als Constraint Satisfaction Problem

- Darstellung der Beschränktheit von Ressourcen
  (Zeit, Kapazität, Menge von Ressourcen, …)
  - Gesetzmäßigkeiten zwischen Objekten
  - Menge von Variablen V_i (z.B. Start der Operation $o_{i,j}$)
  - Bereich D_i (explizit aufzählbar $\{2, 3, 4\}$ oder durch Grenzen gegeben $[2, 4]$)
  - Einschränkungen (constraints) c_i bzw. c_{ij} zwischen Variablen V_i und V_j
  - Lieferzeit Ende von Job $J_i < 20$
  - Ende der Operation $o_{i,j} \leq$ Start der Operation $o_{i,j+1}$
  - Ende der Operation $o_{i,j} \leq$ Start der Operation $o_{i+1,j} \vee$ Ende der Operation $o_{i+1,j} \leq$ Start der Operation $o_{i,j}$
- gesucht sind Tupeln $(w_1, \ldots, w_n)$, wobei w_i eine gültige Belegung von V_i ist

Scheduling als Constraint Optimization Problem

- viele Einschränkungen sind weich
- manche Einschränkungen bzw. Ziele widersprechen sich
- Einschränkungen haben unterschiedliche Wichtigkeit bzw. Nutzen
- Aufträge haben unterschiedliche Priorität
- Weiche Einschränkungen
  - Satisfaction(C_i) ∈ [0, 1]
  - Weight(C_i) ∈ [0, 1]
  - Type(C_i) ∈ {tardiness, idleTime, flowTime, makeSpan, compatibility, …}
- z.B. Lieferzeit



Tabusuche

- erster Plan wird zufällig, heuristisch oder sonst wie konstruiert
- Suche nach einem besseren Nachbarschedule
  - Anwendung unterschiedlicher Operatoren
  - z.B. Vertauschen von zwei Operationen oder Jobs
- Heuristik: Reparatur der größten Verletzung einer Einschränkung
- Tabusuche nimmt beste Nachbarschaftslösung
- diese wird versucht weiter iterativ zu verbessern
- akzeptiert auch schlechtere Nachbarn, wenn keine besseren existieren
- um Zyklen zu vermeiden, wird eine Tabuliste eingeführt, die „alte“ Lösungen verbietet

Tabuliste

- in einer Tabuliste werden Attribute alter Lösungen oder der angewandten Operatoren gespeichert
  - wird die Konstellation, dass die Jobs J_4 und J_1 hintereinanderstehen, gespeichert, dann sind alle neuen Lösungen verboten, bei denen diese Jobs hintereinanderstehen
  - oder werden die Jobs J_3 und J_2 vertauscht, wird diese Kombination gespeichert, und ein Vertauschen von J_2 und J_3 verboten
- die Tabulistenlänge hat einen Einfluss auf das Suchverhalten
- nach n Schritten wird der Tabustatus wieder aufgehoben (z.B. n=4)
- manchmal wird erlaubt, dass der Tabustatus ignoriert wird (z.B. wenn neuer bester Plan gefunden wird)
- auch mehrere Tabulisten machen Sinn
  - lange Liste mit Hashfunktionswerten
  - kurze Liste mit Reihenfolgekombinationen

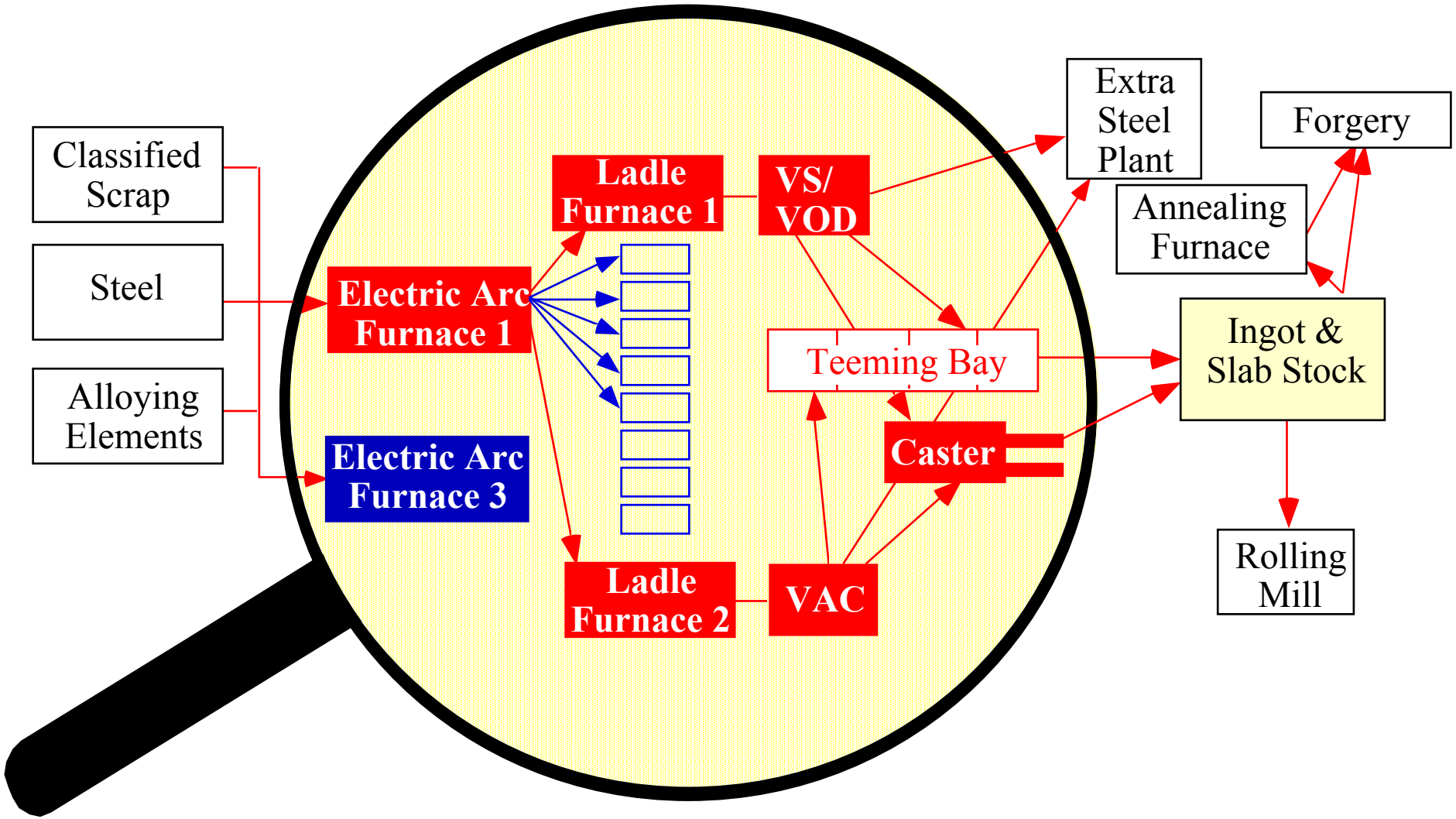
Intensifizierung und Diversifikation

- Idee: wenn lange in eingegrenztem Gebiet des Problemlösungsraums gesucht wurde, sollten die Parameter der Suche so geändert werden, dass man in ein anderes Gebiet kommt
  - Tabuliste wird als Kurzzeitgedächtnis interpretiert
  - Langzeitgedächtnis kann dann Parameter der Tabusuche dynamisch verändern
    - die Länge der Tabuliste
    - die Bestimmung der Nachbarschaft (Auswahl anderer Operatoren, Erweiterung der Nachbarschaft durch weitere Operatoren oder Verringerung)
    - Aspekte der Bewertungsfunktion
- Verbesserungen lassen sich auch durch ein Lernen der Länge erreichen
- auch stochastische Komponenten können Verbesserungen bringen (z.B. Operatoren, die Einfluss auf größte Verletzung haben und ein paar zufällige Operatoren)

Reaktives Scheduling

- während der Ausführung des Plans ändert sich die Anwendung (Maschinen fallen aus, neue wichtige Aufträge, Fehlproduktion)
- Plan muss adaptiert werden
- Ziele
  1. Möglichst wenig Änderungen am Plan
  2. Möglichst robuste Pläne, die keine Änderungen erfordern
  3. Möglichst erst dann reagieren, wenn es auch nötig ist
- unser Ansatz in Refresh
  - Robustheit ist Teil der Bewertungsfunktion
  - Änderungen als Verletzung von Einschränkungen
  - Reparatur von neu verletzten Einschränkungen

Feinplanung im Edelstahlwerk

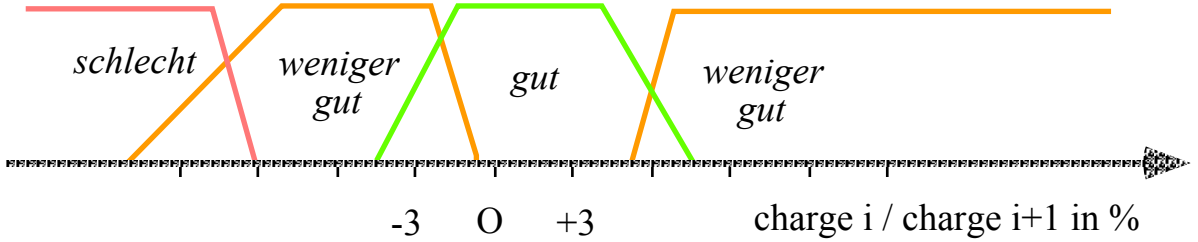


Wirtschaftliche Ziele

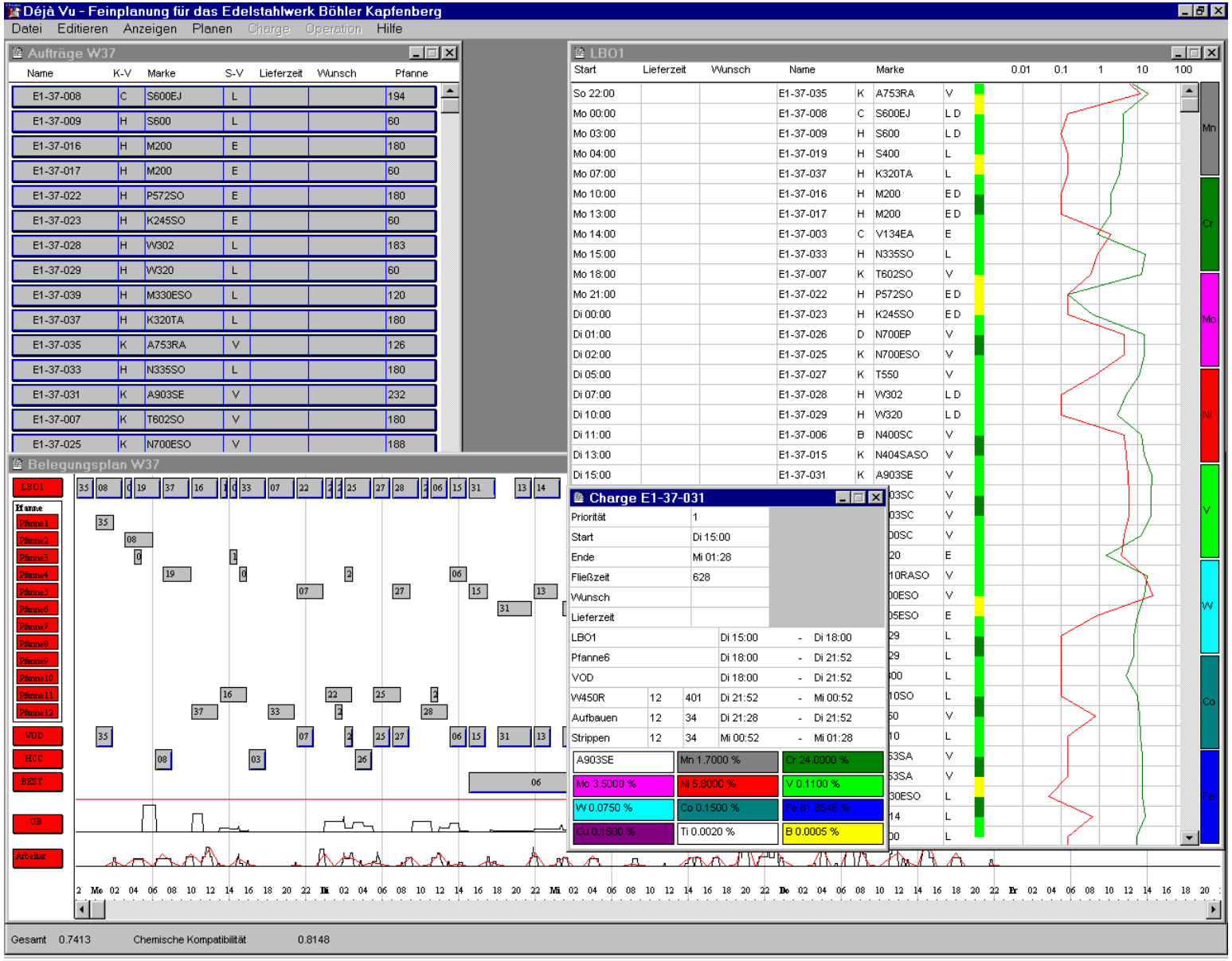
- Qualität (Vermeidung der Verunreinigung durch Vorgängerchargen)
- schlanke Produktion, kleine Lager
- zeitgerechte Fertigung
- Minimierung Energieeinsatz (Pfannentechnologie, nicht unnötiges Erwärmen)
- Gleichmäßige Auslastung der Grubenmannschaft

Lösung

- Repräsentation der graduellen Erfüllung von Einschränkungen
  - die Kompatibilität der chemischen Elemente



- bevorzugt werden Kombinationen, bei denen eine sehr gute Kompatibilität gegeben ist (robust)
- Schwellwert, der erreicht werden muss, damit Plan legal
- multikriterielle Bewertung des Plans mit weichen Einschränkungen
- Optimierung des Plans mit Tabusuche


Déjà Vu - Feinplanung für das Edelstahlwerk Böhler Kapfenberg
Datei Editieren Anzeigen Planen Charge Operation Hilfe
Aufträge W37
Name K-V Marke S-V Lieferzeit Wunsch Pfanne
E1-37-008 C S600EJ L 194
E1-37-009 H S600 L 60
E1-37-016 H M200 E 180
E1-37-017 H M200 E 60
E1-37-022 H P572SO E 180
E1-37-023 H K245SO E 60
E1-37-028 H W302 L 183
E1-37-029 H W320 L 60
E1-37-039 H M330ESO L 120
E1-37-037 H K320TA L 180
E1-37-035 K A753RA V 126
E1-37-033 H N335SO L 180
E1-37-031 K A903SE V 232
E1-37-007 K T602SO V 180
E1-37-025 K N700ESO V 188
LBO1
Start Lieferzeit Wunsch Name Marke
So 22:00 E1-37-035 K A753RA V
Mo 00:00 E1-37-008 C S600EJ L D
Mo 03:00 E1-37-009 H S600 L D
Mo 04:00 E1-37-019 H S400 L
Mo 07:00 E1-37-037 H K320TA L
Mo 10:00 E1-37-016 H M200 E D
Mo 13:00 E1-37-017 H M200 E D
Mo 14:00 E1-37-003 C V134EA E
Mo 15:00 E1-37-033 H N335SO L
Mo 18:00 E1-37-007 K T602SO V
Mo 21:00 E1-37-022 H P572SO E D
Di 00:00 E1-37-023 H K245SO E D
Di 01:00 E1-37-026 D N700EP V
Di 02:00 E1-37-025 K N700ESO V
Di 05:00 E1-37-027 K T550 V
Di 07:00 E1-37-028 H W302 L D
Di 10:00 E1-37-029 H W320 L D
Di 11:00 E1-37-006 B N400SC V
Di 13:00 E1-37-015 K N404SASO V
Di 15:00 E1-37-031 K A903SE V
0.01 0.1 1 10 100
Mn Cr Mo Ni V W Co Fe
Belegungsplan W37
LBO1
Pfanne
Pfanne1 Pfanne2 Pfanne3 Pfanne4 Pfanne5 Pfanne6 Pfanne7 Pfanne8 Pfanne9 Pfanne10 Pfanne11 Pfanne12
VOD
HCC
BEST
GB
Arbeiter
Charge E1-37-031
Priorität 1
Start Di 15:00
Ende Mi 01:28
Fließzeit 628
Wunsch
Lieferzeit
LBO1 Di 15:00 - Di 18:00
Pfanne6 Di 18:00 - Di 21:52
VOD Di 18:00 - Di 21:52
W450R 12 401 Di 21:52 - Mi 00:52
Aufbauen 12 34 Di 21:28 - Di 21:52
Strippen 12 34 Mi 00:52 - Mi 01:28
A903SE
Mn 1.7000 %
Cr 24.0000 %
Mo 3.5000 %
Ni 5.8000 %
V 0.1100 %
W 0.0750 %
Co 0.1500 %
Fe 61.8546 %
Cu 0.1500 %
Ti 0.0020 %
B 0.0005 %
Gesamt 0.7413
Chemische Kompatibilität 0.8148

Weiterführende Veranstaltungen

- Wissensbasierte Systeme, Th. Eiter u. Mitarbeiter
- VO2 und UE2 Wissensbasiertes Planen, Jürgen Dorn
- DÉJÀ VU Applikationsframework für Feinplanungssysteme
  - Fokus: Produktions(fein-)planung
  - Anwendungen in der Stahlindustrie
  - Transportprobleme (Logistik)
  - Integration der Planung in einer Zulieferkette (Integration E-Commerce und Produktionsplanung)
- FWF-Projekt REFRESH – Reaktives Scheduling
- EC3 – Electronic Commerce Competence Center