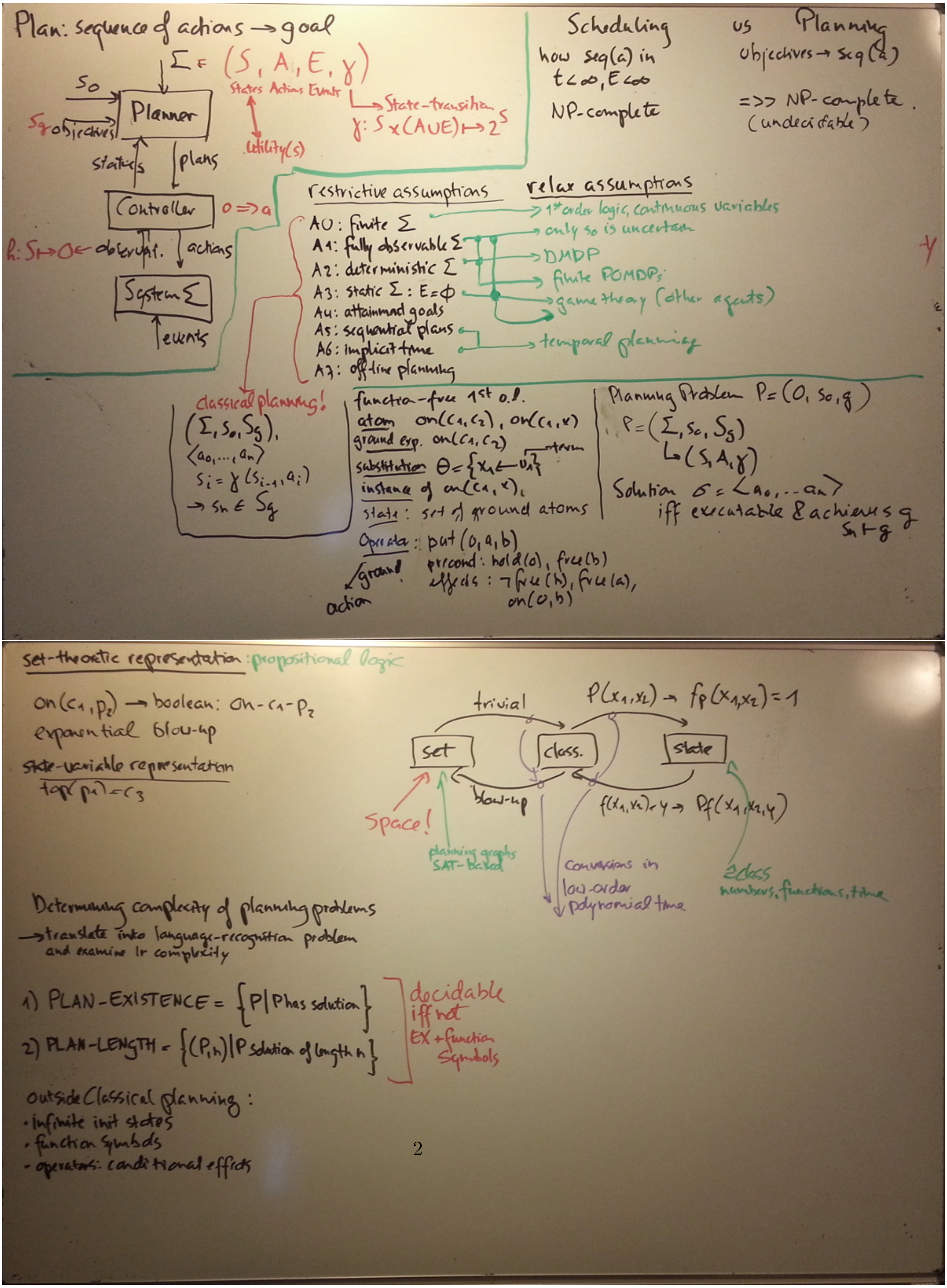
Summary Planning and Scheduling

Alexander Hagg
Bonn-Rhein-Sieg University of Applied Sciences
Master Course on Autonomous Systems
Grantham-Allee 20, 53757 Sankt Augustin
Germany

Sankt Augustin, Decmber 5, 2013



Plan: sequence of actions → goal
Σ = (S, A, E, γ)
States Actions Events
State-transition
γ: S x (A∪E) ↦ 2^S
Utility(s)
s0
Sg, objectives
Planner
status
plans
Controller
o ⇒ a
h: S ↦ O ← observat.
actions
System Σ
events
Scheduling
how seq(a) in
t<∞, E<∞
NP-complete
vs Planning
objectives → seq(a)
=>> NP-complete
(undecidable)
restrictive assumptions
relax assumptions
A0: finite Σ
A1: fully observable Σ
A2: deterministic Σ
A3: static Σ: E=Φ
A4: attainment goals
A5: sequential plans
A6: implicit time
A7: off-line planning
1st order logic, continuous variables
only s0 is uncertain
DMDP
finite POMDPs
game theory (other agents)
temporal planning
classical planning!
(Σ, S0, Sg)
⟨a0, ..., an⟩
si = γ(si-1, ai)
→ sn ∈ Sg
function-free 1st o.l.
atom on(c1, c2), on(c1, x)
ground exp. on(c1, c2)
substitution Θ = {x1 ← v1} term
instance of on(c1, x)
state: set of ground atoms
Operator: put(o, a, b)
precond: hold(o), free(b)
effects: ¬free(b), free(a), on(o, b)
ground
action
Planning Problem P = (O, s0, g)
P = (Σ, s0, Sg)
(S, A, γ)
Solution σ = ⟨a0, .. an⟩
iff executable & achieves g
sn ⊢ g
set-theoretic representation propositional logic
on(c1, p2) → boolean: on-c1-p2
exponential blow-up
state-variable representation
top(p1) = c3
trivial
P(x1, x2) → fP(x1, x2) = 1
set
class.
state
blow-up
f(x1, x2) = y → Pf(x1, x2, y)
Space!
planning graphs
SAT-based
conversions in
low-order
polynomial time
≥ class
numbers, functions, time
Determining complexity of planning problems
→ translate into language-recognition problem
and examine its complexity
1) PLAN-EXISTENCE = {P | P has solution}
2) PLAN-LENGTH = {(P, n) | P solution of length n}
decidable
iff not
EX + function
symbols
outside Classical planning:
· infinite init states
· function symbols
· operators: conditional effects

Figure 1:

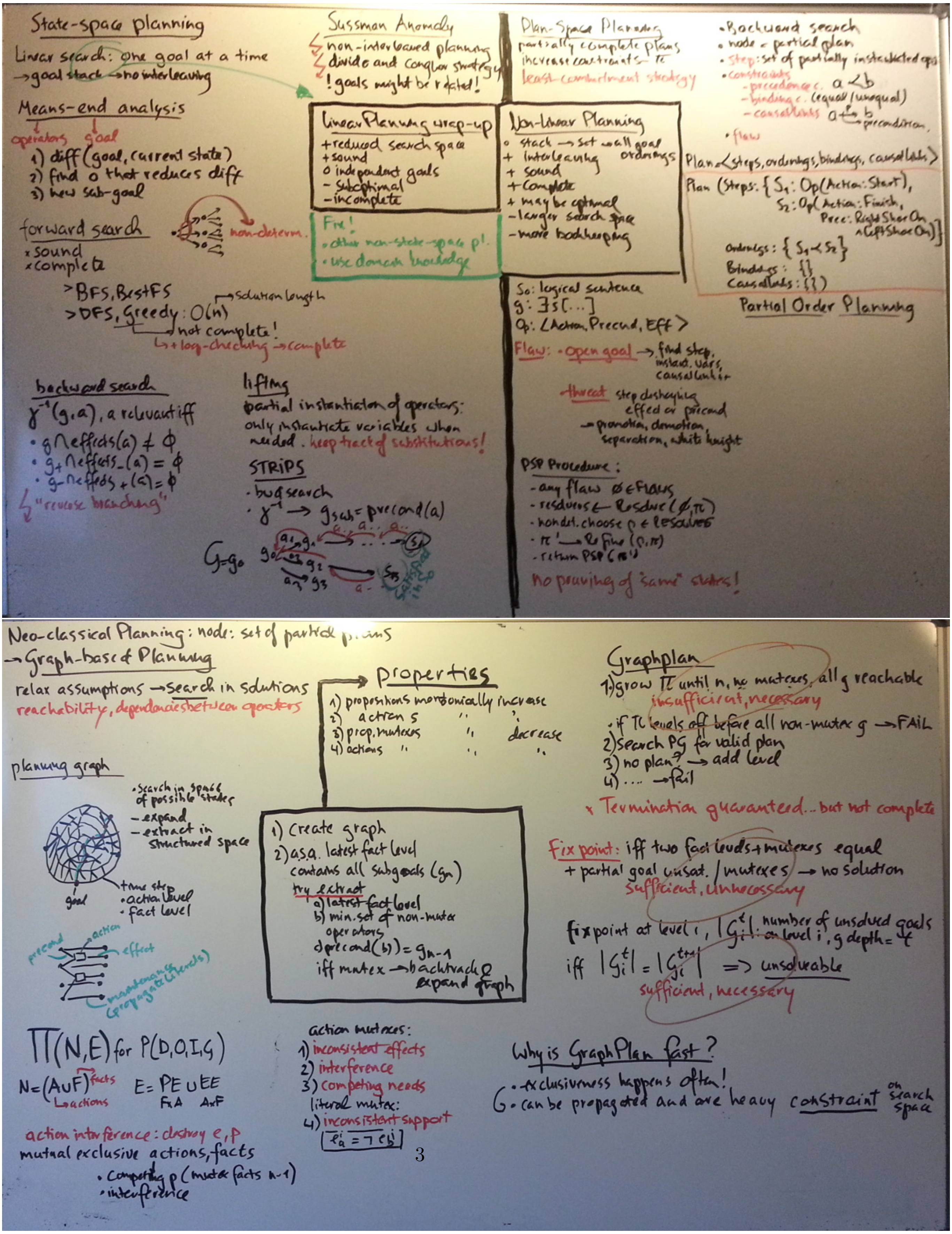
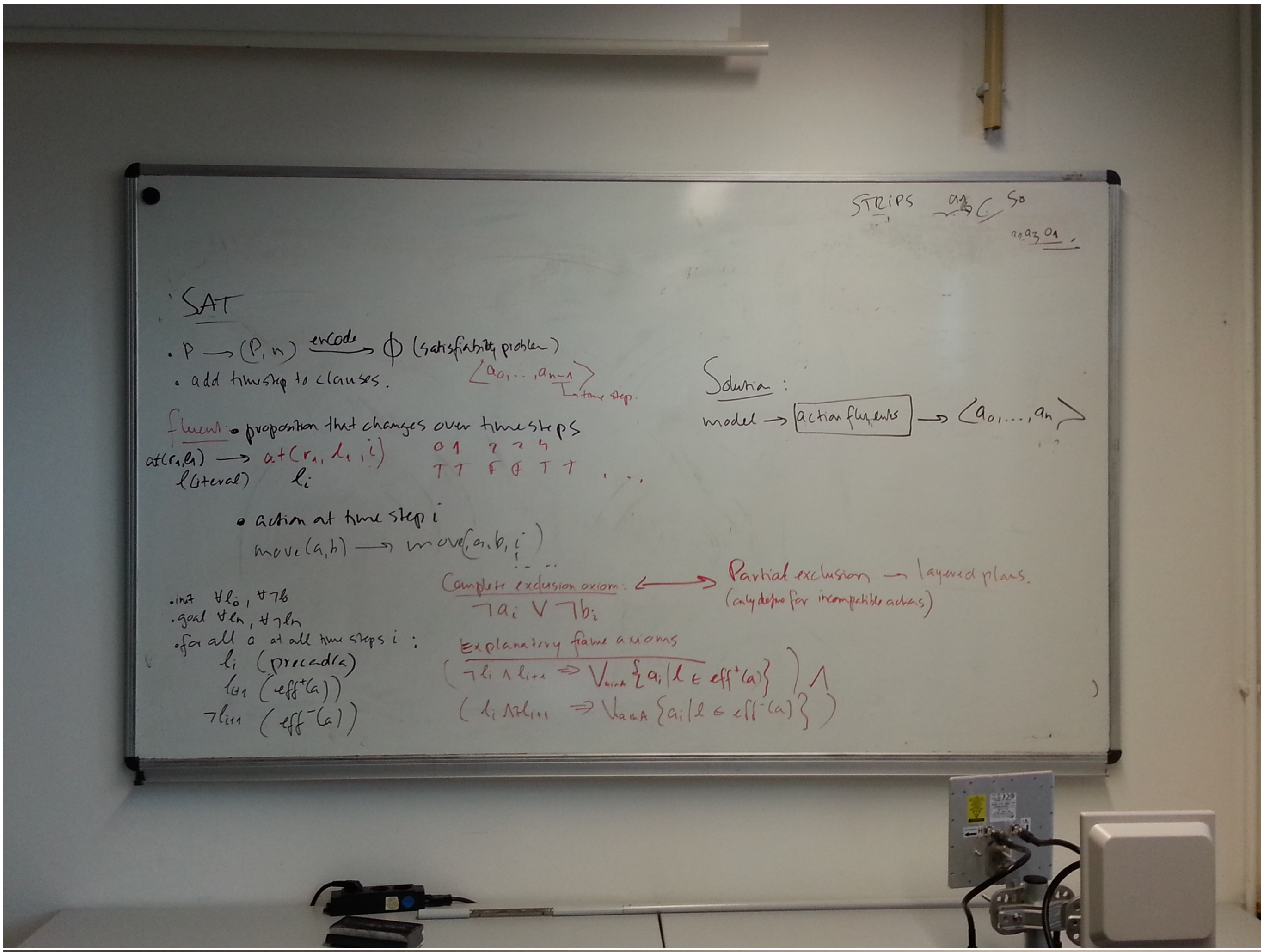
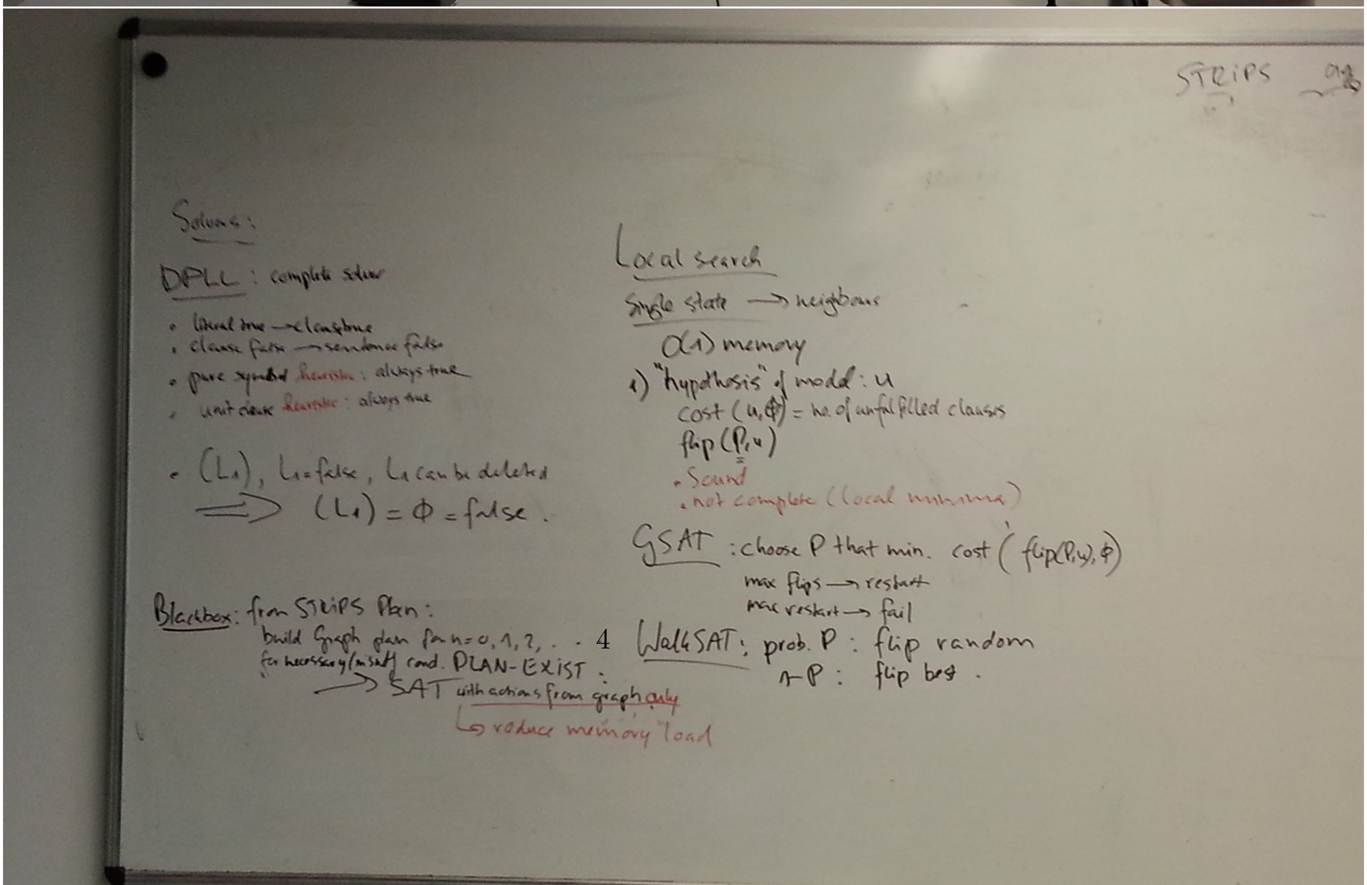

State-space planning
Linear search: one goal at a time
→goal stack →no interleaving
Means-end analysis
operators goal
1) diff (goal, current state)
2) find o that reduces diff
3) new sub-goal
forward search
non-determ.
× sound
× complete
>BFS, BestFS
>DFS, Greedy: O(n)
solution length
not complete!
+ loop-checking → complete
backward search
γ⁻¹(g,a), a relevant iff
• g ∩ effects(a) ≠ ϕ
• g+ ∩ effects−(a) = ϕ
• g− ∩ effects+(a) = ϕ
"reverse branching"
lifting
partial instantiation of operators:
only instantiate variables when needed. keep track of substitutions!
STRIPS
• bwd search
• γ⁻¹ → gsub = precond(a)
G=g₀
Sussman Anomaly
non-interleaved planning
divide and conquer strategy
! goals might be related!
Linear Planning wrap-up
+ reduced search space
+ sound
o independent goals
− suboptimal
− incomplete
Fix!
• other non-state-space pl.
• use domain knowledge
Plan-Space Planning
partially complete plans
increase constraints → π
least-commitment strategy
Non-Linear Planning
o stack → set → all goal orderings
+ interleaving
+ sound
+ complete
+ may be optimal
− larger search space
− more bookkeeping
S₀: logical sentence
g: ∃s[...]
Op: ⟨Action, Precond, Eff⟩
Flaw: • open goal → find step, instant. vars, causal link
• threat step destroying effect or precond
→ promotion, demotion, separation, white knight
PSP Procedure:
− any flaw ϕ ∈ Flaws
− resolvers ← Resolve(ϕ, π)
− nondet. choose ρ ∈ resolvers
− π' ← Refine(ρ, π)
− return PSP(π')
no pruning of "same" states!
• Backward search
• node = partial plan
• step: set of partially instantiated ops
• constraints
− precedence c. a ≺ b
− binding c. (equal / unequal)
− causal links a →ᵖ b precondition
• flaw
Plan ⟨steps, orderings, bindings, causal links⟩
Plan (Steps: {S₁: Op(Action: Start),
S₂: Op(Action: Finish,
Prec: RightShoeOn ∧ LeftShoeOn)}
Orderings: {S₁ ≺ S₂}
Bindings: {}
Causal links: {})
Partial Order Planning
Neo-classical Planning: node: set of partial plans
→ Graph-based Planning
relax assumptions → search in solutions
reachability, dependencies between operators
planning graph
• search in space of possible states
− expand
− extract in structured space
goal
time step
• action level
• fact level
precond
action
effect
maintenance (propagate literals)
properties
1) propositions monotonically increase
2) actions " "
3) prop. mutexes " decrease
4) actions " "
1) Create graph
2) a.s.a. latest fact level contains all subgoals (gₙ)
try extract
a) latest fact level
b) min. set of non-mutex operators
d) precond(b) = gₙ₋₁
iff mutex → backtrack & expand graph
Graphplan
1) grow Π until n, no mutexes, all g reachable
insufficient, necessary
• if Π levels off before all non-mutex g → FAIL
2) Search PG for valid plan
3) no plan? → add level
4) ... → fail
× Termination guaranteed... but not complete
Fix point: iff two fact levels + mutexes equal
+ partial goal unsat. / mutexes → no solution
sufficient, unnecessary
fixpoint at level i, |Gᵢᵗ|: number of unsolved goals on level i, g depth = t
iff |Gᵢᵗ| = |Gᵢᵗ⁺¹| ⇒ unsolvable
sufficient, necessary
Π(N,E) for P(D,O,I,G)
N = (A ∪ F) facts, locations
E = PE ∪ EE
action interference: destroy e, p
mutual exclusive actions, facts
• competing p (mutex facts n−1)
• interference
action mutexes:
1) inconsistent effects
2) interference
3) competing needs
literal mutex:
4) inconsistent support
Why is GraphPlan fast?
• exclusiveness happens often!
• can be propagated and are heavy constraint on search space

Figure 2: ..


STRIPS
SAT
• P → (P, n) encode → Φ (satisfiability problem)
• add timestep to clauses.
⟨a0, … , an−1⟩
time step.
fluent • proposition that changes over timesteps
at(r1,l1) → at(r1, l1, i)
l(iteral) li
0 1 2 3 4
T T F F T T . . .
• action at time step i
move(a,b) → move(a,b,i)
Solution:
model → action fluents → ⟨a0, … , an⟩
Complete exclusion axiom: ¬ai ∨ ¬bi
Partial exclusion → layered plans.
(only defined for incompatible actions)
.init ∀l0, ∀¬l0
.goal ∀ln, ∀¬ln
.for all a at all time steps i:
li (precond(a))
li+1 (eff+(a))
¬li+1 (eff−(a))
Explanatory frame axioms
(¬li ∧ li+1 ⇒ ∨a∈A {ai | l ∈ eff+(a)}) ∧
(li ∧ ¬li+1 ⇒ ∨a∈A {ai | l ∈ eff−(a)})


STRIPS
Solvers:
DPLL: complete solver
• literal true → clause true
• clause false → sentence false
• pure symbol heuristic: always true
• unit clause heuristic: always true
• (L1), L1 = false, L1 can be deleted
⟹ (L1) = Φ = false.
Blackbox: from STRIPS Plan:
build Graph plan for n = 0, 1, 2, . .
for necessary (not suff) cond. PLAN-EXIST:
→ SAT with actions from graph only
reduce memory load
Local search
Single state → neighbours
O(1) memory
1) "hypothesis" of model: u
cost (u, Φ) = no. of unfulfilled clauses
flip (P, u)
• Sound
• not complete (local minima)
GSAT: choose P that min. cost (flip(P,u), Φ)
max flips → restart
max restart → fail
WalkSAT: prob. P: flip random
1−P: flip best.

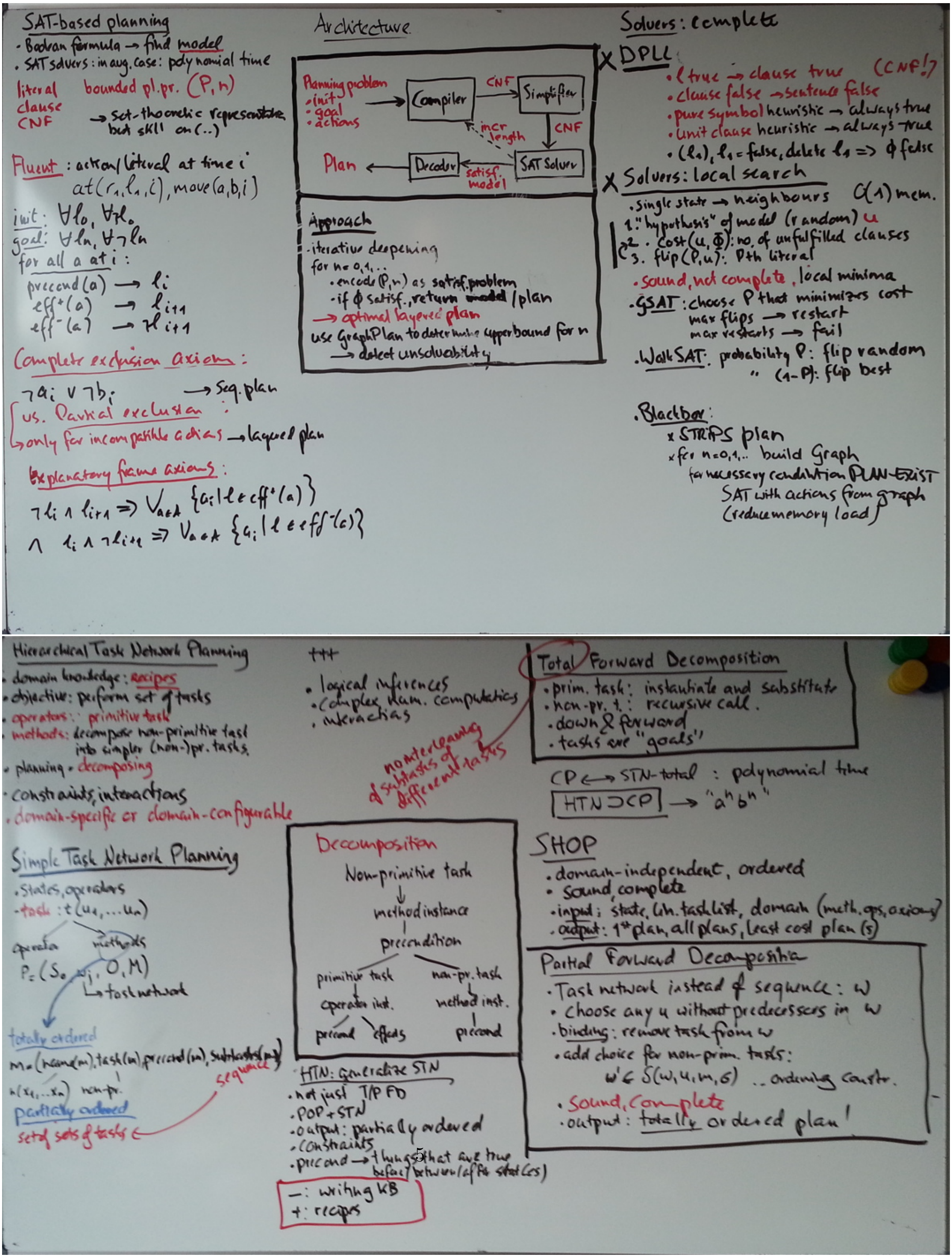

SAT-based planning
· Boolean formula → find model
· SAT solvers: in avg. case: polynomial time
literal
clause
CNF
bounded pl.pr. (P,n)
→ set-theoretic representation but still on(..)
Fluent: action/literal at time i
at(r1,l1,i), move(a,b,i)
init: ∀l0, ∀¬l0
goal: ∀ln, ∀¬ln
for all a at i:
precond(a) → li
eff+(a) → li+1
eff-(a) → ¬li+1
Complete exclusion axiom:
¬ai ∨ ¬bi → seq. plan
vs. Partial exclusion:
only for incompatible actions → layered plan
Explanatory frame axioms:
Architecture
Planning problem
·init
·goal
·actions
Compiler
CNF
Simplifier
incr. length
CNF
SAT Solver
satisf. model
Decoder
Plan
Approach
·iterative deepening
for n = 0,1,..
·encode (P,n) as satisf. problem
·if ϕ satisf., return model / plan
→ optimal layered plan
use GraphPlan to determine upper bound for n
→ detect unsolvability
Solvers: complete
DPLL
·l true → clause true
(CNF!)
·clause false → sentence false
·pure symbol heuristic → always true
·unit clause heuristic → always true
·(l1), l1 = false, delete l1 ⇒ ϕ false
Solvers: local search
·single state → neighbours
O(1) mem.
1. "hypothesis" of model (random) u
2. cost(u, Φ): no. of unfulfilled clauses
3. flip(P,u): Pth literal
·sound, not complete, local minima
·GSAT: choose P that minimizes cost
max flips → restart
max restarts → fail
·WalkSAT: probability P: flip random
" (1-P): flip best
·Blackbox:
x STRIPS plan
x for n=0,1,.. build Graph
for necessary condition PLAN-EXIST
SAT with actions from graph
(reduce memory load)
Hierarchical Task Network Planning
·domain knowledge: recipes
·objective: perform set of tasks
·operators: primitive task
·methods: decompose non-primitive task into simpler (non-)pr. tasks
·planning = decomposing
·constraints, interactions
·domain-specific or domain-configurable
Simple Task Network Planning
·states, operators
·task: t(u1,...un)
operator
methods
P = (S0, wi, O, M)
task network
totally ordered
partially ordered
set of sets of tasks
sequence
+++
·logical inferences
·complex ham. computations
·interactions
non interleaving of subtasks of different tasks
Decomposition
Non-primitive task
method instance
precondition
primitive task
non-pr. task
operator inst.
method inst.
precond
effects
precond
HTN: Generalize STN
·not just T/P FD
·POP + STN
·output: partially ordered
·constraints
–: writing KB
+: recipes
Total Forward Decomposition
·prim. task: instantiate and substitute
·non-pr. t.: recursive call
·down & forward
·tasks are "goals"
CP ⟷ STN-total : polynomial time
HTN ⊃ CP → "aⁿbⁿ"
SHOP
·domain-independent, ordered
·sound, complete
·input: state, ini. tasklist, domain (meth., ops, axioms)
·output: 1st plan, all plans, least cost plan(s)
Partial Forward Decomposition
·Task network instead of sequence: w
·choose any u without predecessors in w
·binding: remove task from w
·add choice for non-prim. tasks:
w' ∈ δ(w,u,m,σ) .. ordering constr.
·sound, complete
·output: totally ordered plan!

Figure 4: ..

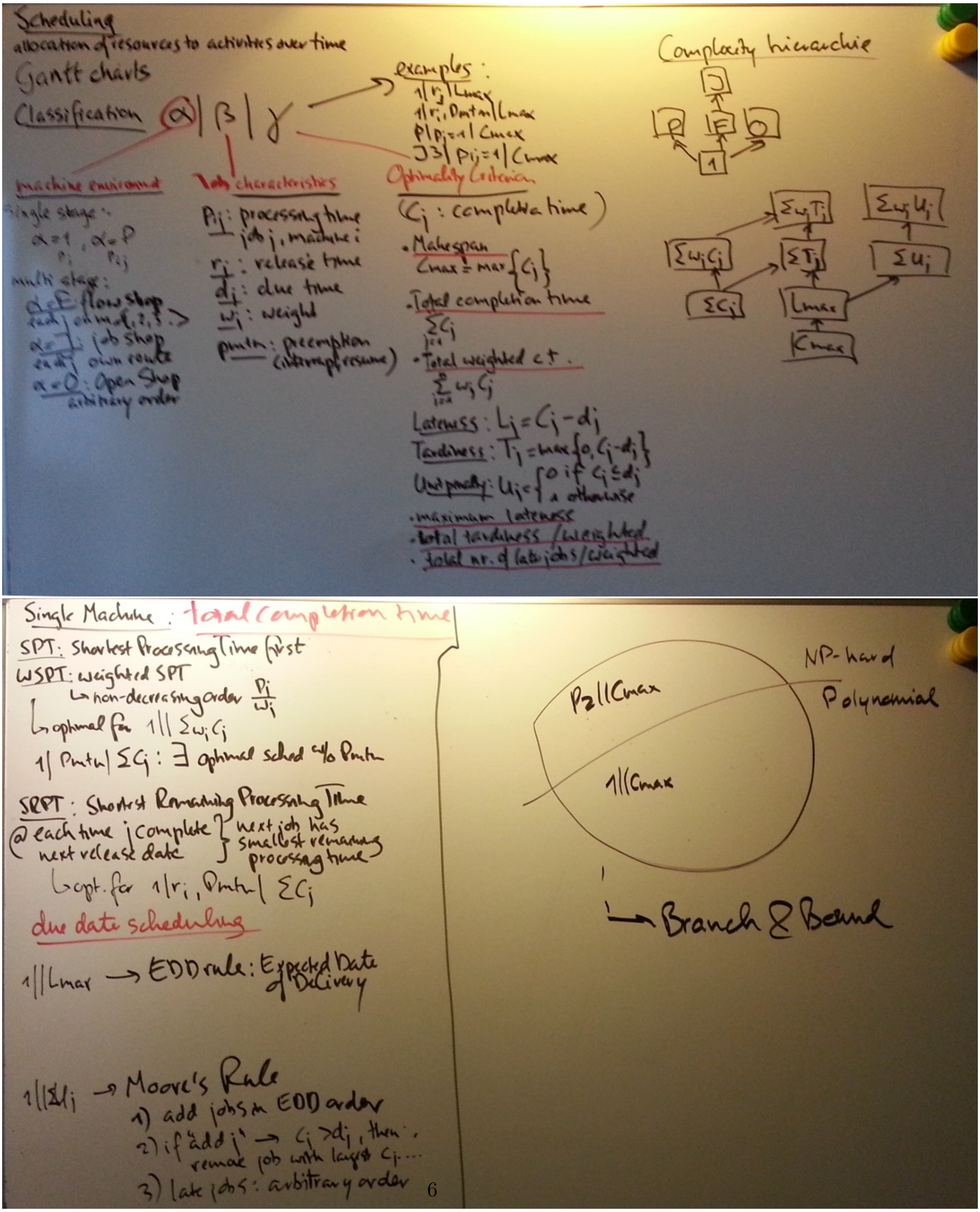

Scheduling
allocation of resources to activities over time
Gantt charts
Classification
examples:
machine environment
Job characteristics
Optimality Criterion
single stage:
multi stage:
flow shop
job shop
each j own route
Open Shop
arbitrary order
processing time
job j, machine i
release time
due time
weight
preemption
(interrupt/resume)
completion time
Makespan
Total completion time
Total weighted c.t.
Lateness
Tardiness
0 if
otherwise
maximum lateness
total tardiness /weighted
total nr. of late jobs/weighted
Complexity hierarchie
Single Machine
total completion time
SPT: Shortest Processing Time first
WSPT: weighted SPT
non-decreasing order
optimal for
optimal sched w/o Pmtn
SRPT: Shortest Remaining Processing Time
@ each time
j complete
next release date
next job has
smallest remaining
processing time
opt. for
due date scheduling
EDD rule: Expected Date of Delivery
Moore's Rule
1) add jobs in EDD order
2) if add j
then
remove job with largest
3) late jobs: arbitrary order
NP-hard
Polynomial
Branch & Bound

Figure 5: ..