Build Order Optimization in StarCraft

David Churchill and Michael Buro

Daniel Federau

Universität Basel

19. November 2015

Motivation

- planning can be used in real-time strategy games (RTS), e.g.
  - pathfinding of units
  - strategical planning
  - tactical assault planning
- in this paper: finding an optimal build order for the game StarCraft

StarCraft

- created by Blizzard Entertainment in 1998
- one of the most popular RTS-games
- the goal is to destroy all enemy buildings
- the player gathers resources, builds production buildings and combat units
- consumable resources: minerals, gas and supply
- building dependencies are saved in tech tree

StarCraft


8053
8975
34/49
Protoss Probe
Kills: 0
20/20 20/20
MENU
Warp in Photon Cannon
150

Build Order Optimization

- build order is the order in which units/buildings are built
- optimal build order reaches a given goal as fast as possible (minimize makespan)
- goal: build number of units/buildings/resources

Overview

- definition of the search space is needed for search
- every unit, building and consumable is considered a resource
- every action has preconditions and produces resources

Action - Definition

action $a = (\delta, r, b, c, p)$

- δ: duration measured in frames
- r: required resources, need to be present in order to execute action
- b: borrowed resources, will be available again after action finishes (e.g. production buildings)
- c: consumed resources, become unavailable after executing action (e.g. minerals, gas)
- p: produced resources after action finishes

Action - Example

action $a =$ "Build Terran unit Firebat"

- δ: 576 frames (24 seconds)
- $r = \{\text{Academy}\}$
- $b = \{\text{Barracks}\}$
- $c = \{50$ Minerals, 25 Gas, 1 Supply$\}$
- $p = \{1$ Firebat$\}$

States

state $S = (t, R, P, I)$

- t: current game time
- R: vector with every resource available
- P: actions currently in progress
- I: worker income data (10 gather minerals, 3 gather gas) $\rightarrow$ used for abstraction

Abstractions

used to reduce search space and increase the performance of the planner:

1. fixed income rate per worker per frame (0.045 minerals, 0.07 gas)
2. assign 3 workers to a refinery when it finishes
3. add 4 seconds to the game time whenever a building is constructed

Action Legality

- difference between executable and legal actions
- an action a is legal in state S if:
  1. required or borrowed resources are currently available, borrowed or under construction
  2. consumable resources are currently available or will be in the future without executing an action

State Transition

3 functions for the definition of the transition function for a given state S:

- $S' \leftarrow \text{Sim}(S, \delta)$: simulates progression from S during δ without actions $\rightarrow$ increases resource count and finishes actions
- $\delta \leftarrow \text{When}(S, R)$: returns duration δ when resources R are available
- $S' \leftarrow \text{Do}(S, a)$: execute action a in state S if resources are available (does not increase time of S)

transition function $T : S' = Do(Sim(S, When(S, a)), a)$

Search Algorithm

- depth-first branch and bound algorithm
- recursive algorithm
- possible to stop at any time to return best solution so far
- heuristic functions for pruning nodes
- search algorithm is optimal if heuristic is admissible

High-level Algorithm

DFBB(*S*)

- return best solution so far if time runs out
- update bound whenever a better solution is found
- expand children:
  - heuristic evaluation of children
  - prune child if cost so far and heuristic is bigger than bound

Heuristics

maximum of the two heuristics is used for lower bound:

- LandmarkLowerBound(S, G)
  - uses landmarks (vital actions for achieving a goal)
  - landmarks can be obtained from tech tree
  - sum of duration of all non-concurrent landmark actions
- ResourceGoalBound(S, G)
  - sum of all consumed resources needed to build all units/buildings in goal G
  - duration that is needed to gather this amount with current worker count

Macro Actions

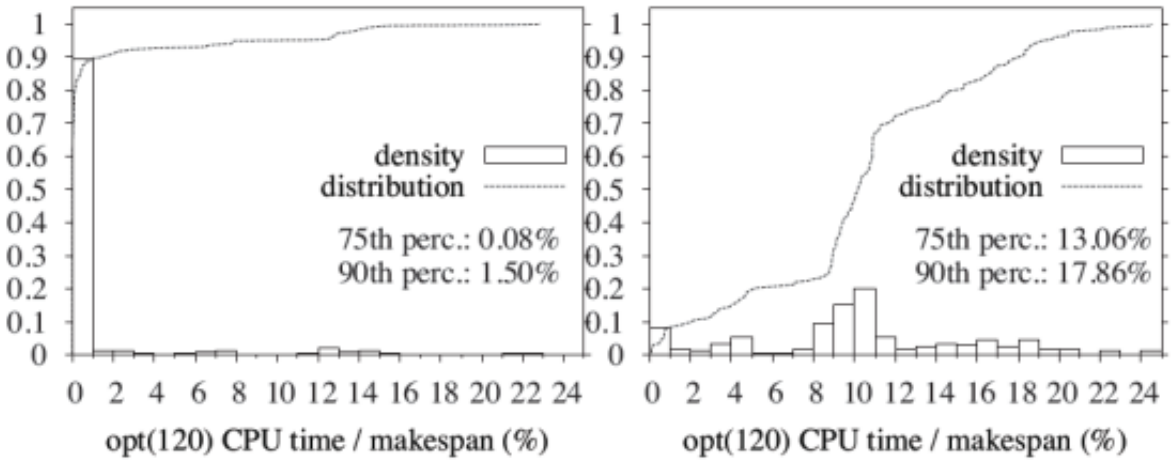
- manually implemented
- double existing actions
- every action has a repetition value K
- defines how often an action has to be executed in a row
- decreases depth of search but produces non-optimal solutions

Comparison

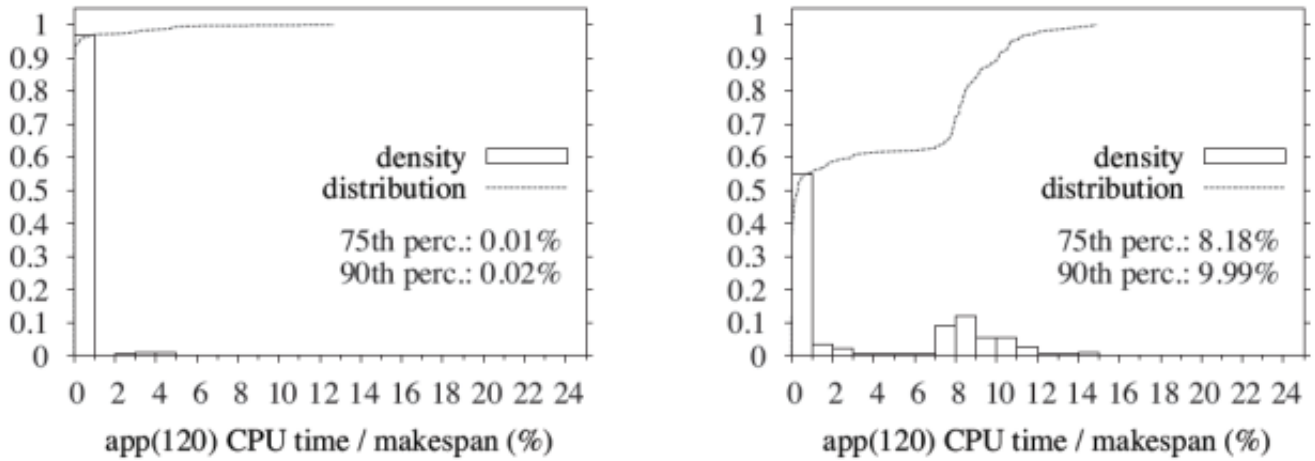
- produced build orders were compared to ones from professional players
- build orders were extracted manually from replays
  - save sequence of all actions that produce resources
  - every 500 frames from beginning of the game
  - until 10000 frames (7 min) or until one of the units dies
- goals were extracted with $GetGoal(B, t_s, t_e)$
  - build order B, start time t_s, end time t_e
  - every resource produced by actions issued between t_s and t_e

Results: CPU-Usage

A) CPU time statistics for search without macro actions:

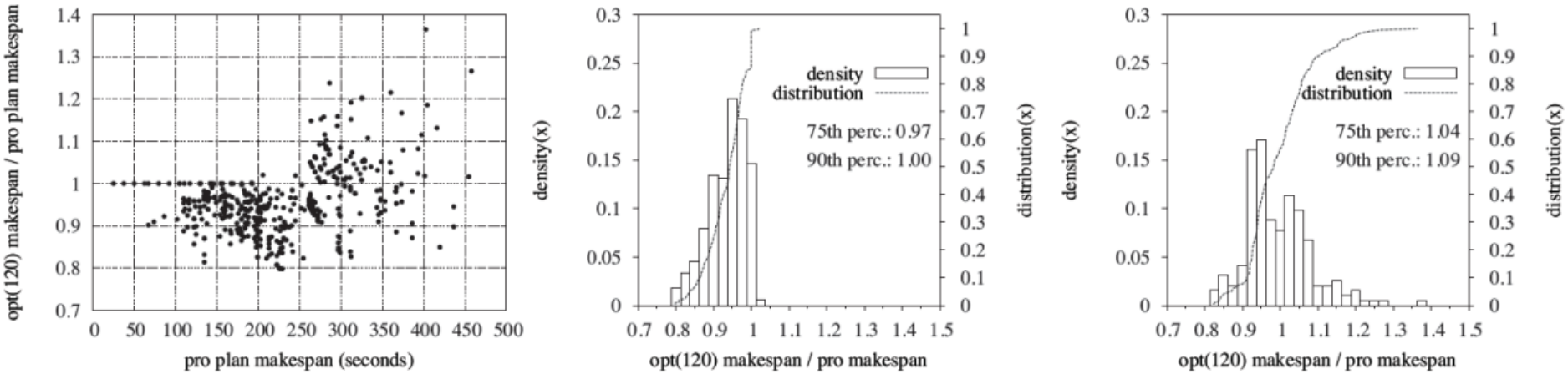


B) CPU time statistics for search with macro actions:

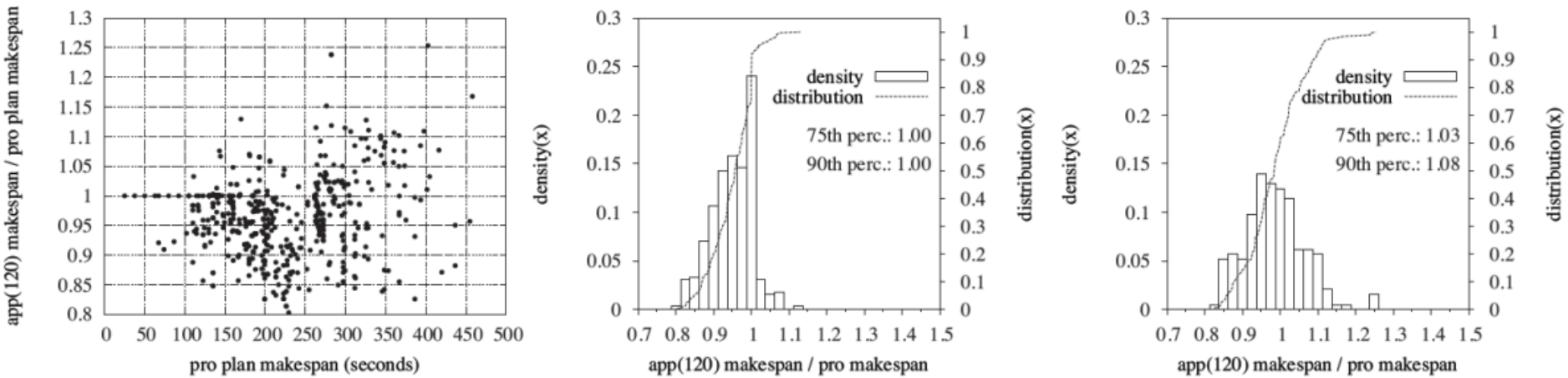


Results: Comparison with professional replays

A) Planning without macro actions, 120s increments, plan quality relative to professional player makespans [opt(120)]



B) Planning with macro actions, 120s increments, plan quality relative to professional player plan makespans [app(120)]



Conclusion

- possible to compute build orders in real time
- results are close to professional build orders
- abstractions greatly reduce search time but can lead to non-optimal solution

Discussion

- comparison in favour of the planner:
  - professional player also has to control units
  - player can change his goal during his build order
- planner can not detect unit loss

Image Sources

- Frame 4: `http://s3.vidimg02.popscreen.com/original/31/NTQ2MDU5MjUz_o_lets-play-starcraft-brood-war---03-legacy-of-the-xelna.jpg`
- Frame 6: `http://www.teamliquid.net/forum/brood-war/226892-techtree-pictures`

Search Algorithm

Algorithm 1 Depth-First Branch & Bound

Require: goal G, state S, time limit t, bound b

```
 1: procedure DFBB(S)
 2:     if TimeElapsed ≥ t then
 3:         return
 4:     end if
 5:     if S safisfies G then
 6:         b ← min(b, S_t)                          ▷ update bound
 7:         bestSolution ← solutionPath(S)
 8:     else
 9:         while S has more children do
10:             S' ← S.nextChild
11:             S'.parent ← S
12:             h ← eval(S')                         ▷ heuristic evaluation
13:             if S'_t + h < b then
14:                 DFBB(S')
15:             end if
16:         end while
17:     end if
18: end procedure
```

Compare Algorithm

Require: BuildOrder B, time limit t, Increment Time i
procedure CompareBuildOrder(B, t, i)
$S \leftarrow$ Initial StarCraft State
SearchPlan $\leftarrow$ DFBB(S,GetGoal($B, 0, \infty$), t)
if SearchPlan.timeElapsed $\leq t$ **then**
return MakeSpan(SearchPlan)/MakeSpan(B)
else
inc $\leftarrow i$
SearchPlan $\leftarrow \emptyset$
while inc $\leq$ MakeSpan(B) **do**
IncPlan $\leftarrow$ DFBB(S,GetGoal(B,inc$-i$,inc),t)
if IncPlan.timeElapsed $\geq t$ **then**
return failure
else
SearchPlan.append(IncPlan)
$S \leftarrow S$.execute(IncPlan)
inc $\leftarrow$ inc $+i$
end if
end while
return MakeSpan(SearchPlan)/MakeSpan(B)
end if
end procedure