Algorithm 5.2 Pseudo-code for BBA\*

Input:

Graph TaskGraph

Int NumProcessors

Double BestCurrentSolutionValue = List Scheduling Heuristic Output

Int NumTasks = n

Task CurrentTask = null

Int CurrentProc = −1

Task PreviousTask = null

Int PreviousProc = -1

Int NumFreeTasks(s) = free(s).length

Int depth = 0

ScheduleState state = {}

ScheduleState bestFoundState = {}

Output: Optimal Schedule represented by ScheduleState state

BBA\*\_Start():

BBA\*\_Recursion(CurrentTask, CurrentProc, PreviousTask, PreviousProc, NumFreeTasks(s), depth, state,

BestFoundState, BestCurrentSolutionValue );

Print optimal solution BestFoundState and exit

Method:

BBA\*\_Recursion(CurrentTask, CurrentProc, PreviousTask, PreviousProc, NumFreeTasks(s), depth, state,

BestFoundState, BestCurrentSolutionValue, NumProcessors ){

done = 0

If(!free(state).isEmpty){ //if there exists free tasks for state

For {i = 1, i <= NumFreeTasks(state), i++}

For {j = 1, j<=NumProcessors, j++}{

depth = depth + 1

Sanitise\_schedule(); // Removes scheduled entries when backtracked

//Schedule a picked task t from free(state) onto proc j. Add it to state.

PreviousTask = CurrentTask, PreviousProc = CurrentProc

CurrentTask = free(s)[i], CurrentProc = j

state.add(CurrentTask, CurrentProcessor)

If(HeuristicFunction(state) ≤ BestCurrentSolutionValue AND depth = NumTasks)

BestFoundState = state

BestCurrentSolutionValue = HeuristicFunction(state)

If(HeuristicFunction(state) ≤ BestCurrentSolutionValue AND depth ≤ NumTasks) {

done = BBA\*\_Recursion(CurrentTask, CurrentProc, PreviousTask,

PreviousProc, NumFreeTasks(state), depth, state, BestFoundState

BestCurrentSolutionValue );

}

If(done = 0){ //Recursion exits and backtrack search tree

depth = depth − 1

}

}

}

Return done

}

}

Possible preprocessing:

Structures (like hashmaps) to quickly add to/calculate free(State)

strong initial lower bound/schedule