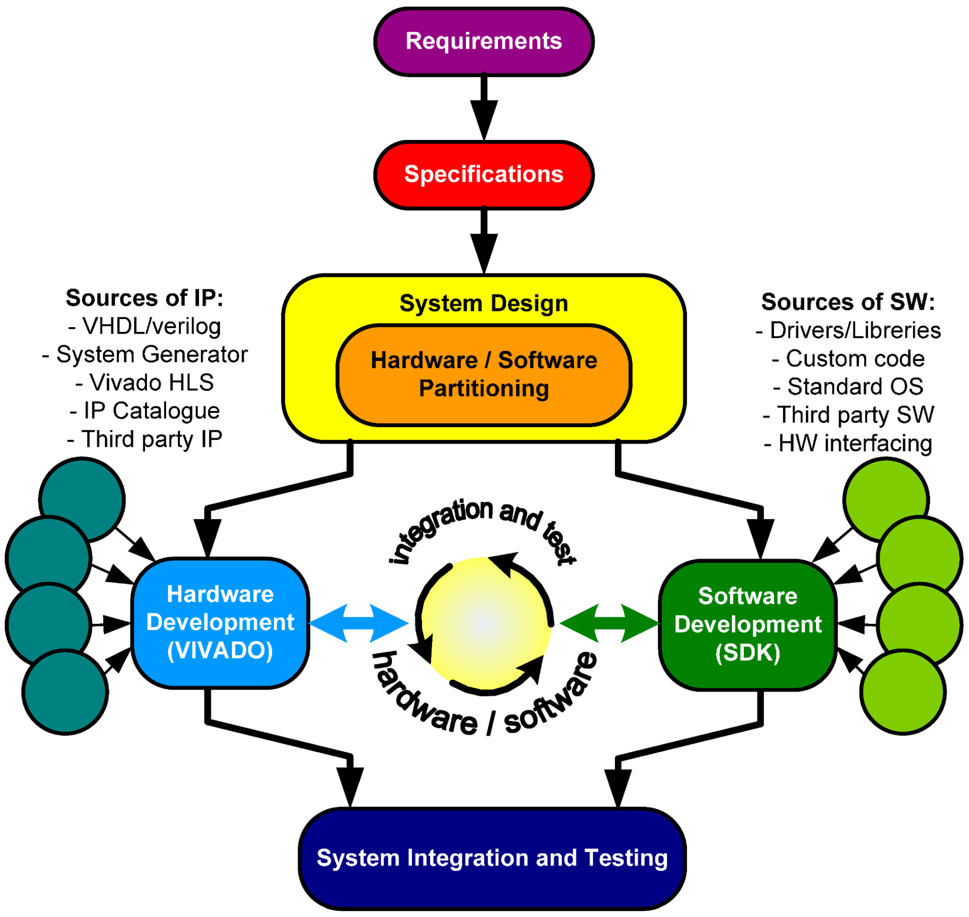


AIFA Final Report

Topic : Heuristic Search Methods for Functional partitioning in Hardware-Software codesign

07.11.2021

Group members

Abhisek Mohanty (18EC3AI20)

Aishik Mandal (18EC3AI21)

Hardhik Mohanty (18EE3AI26)

Samarth (18EC35051)

Github Link: https://github.com/AbhisekM2000/AIFA-term-project.git

You Tube Demonstration Video Link: https://youtu.be/u8bdyL_1zNM

Problem Statement

I. Problem statement

Task partitioning and task scheduling are required in many applications, for instance codesign systems, parallel processor systems, and reconfigurable systems. Sub-tasks extracted from the input description should be implemented in the right place (using the partitioner) at the right time (using the scheduler). It is well known that such scheduling and partitioning problems are NP-complete and are therefore intractable. Furthermore, in the case of reconfigurable hardware, such problems are considered NP-hard. Optimization techniques based on heuristic methods are generally employed to explore the search space so that feasible and near-optimal solutions can be obtained.

Here we focus on algorithms involving heuristic search to solve hardware software partitioning problems. Although heuristic methods do not guarantee the optimum point, they can produce acceptable solutions within a reasonable amount of time. Common heuristic-based algorithms include Depth First Branch & Bound (DFBB), simulated annealing (SA), and genetic algorithm (GA). We are planning to model the problem as a CSP and then apply Genetic Algorithm (GA) and Simulated annealing(SA) to get the optimal scheduling.

II. Formal Problem statement

Given : A task graph with each node T_i has hardware area cost h_i and software execution time cost s_i

We take the binary decision variable as y_i.

y_i =1, if T_i is implemented by Software, else if y_i=0, then it is implemented by hardware.

Optimization objective:

$$\sum_i (1 - y_i) h_i$$

Subject to constraint:

$$\sum_i y_i . s_i \leq D$$

Where, D denotes the total Software resources available

AI modelling

The problem we are trying to solve has the constraint of execution time and we can not go beyond certain timing constraints for scheduling. At the same time we are also trying to minimize the hardware resources used in terms of its total area in order to reduce the size of our system.

Given the above scenario we can model the problem as a CSP and use the CSP solver. We can also use search to get a solution. But the optimal solution is to search the whole state space using some advanced search technique. This can be time as well as resource consuming. Thus we use a Heuristic based Search. It is useful for pruning when overhead for optimal solution is unacceptable. Depending upon the heuristic function we can produce solutions close to the optimal solution.

Solution Approach

Programming Language: Python

Branch & Bound(BB)

- State space tree expansion starts at a state when all tasks are unallocated.
- Make a move by choosing one of the tasks
- At a given step, we have two move options
    - Allocate task for hardware execution
    - Allocate task for software execution
    - Branch on all possible options on the state reached through the move iff,
        - Current best solution cost < Upper Bound at that state.
        - Otherwise, prune or kill the search of the subsequent sub-tree.

Depth First Branch & Bound (DFBB)

- Initially, the algorithm maintains a stack (LIFO structure) for newly generated states of the state space tree.
- A goal state holds a valid allocation (HW/SW) for every task.
- The optimal goal state is the one that has the minimal HW area cost.
- There is an upper bound function f(n) in every state: f(n) = g(n) + h(n)
    - g(n) = Actual HW cost of tasks already allocated to SW
    - g'(n) = Actual SW execution time cost at current state

- h(n) = Upper bound on estimated HW cost (of software tasks) of reaching goal state from present state

$$h(n) = \sum_{j=l1}^{lk} h_j + (h_{lk+1}/s_{lk+1})^* \left(D - g'(n) - \sum_{j=l1}^{lk} s_j \right)$$

such that the following holds,

$$\sum_{j=l1}^{lk} s_j \leq D - g^{'}(n) \leq \sum_{j=l1}^{lk+1} s_j$$

Our implementation of this can be found here

Demonstration with an Example

We take an example in order to demonstrate our implementation. We consider an example set with 4 tasks whose software and hardware resources used are as follows:

	Task 1	Task 2	Task 3	Task 4
H_i	10	10	12	18
S_i	2	4	6	9
H_i/S_i	5	2.5	2	2

The outputs obtained through our codes are

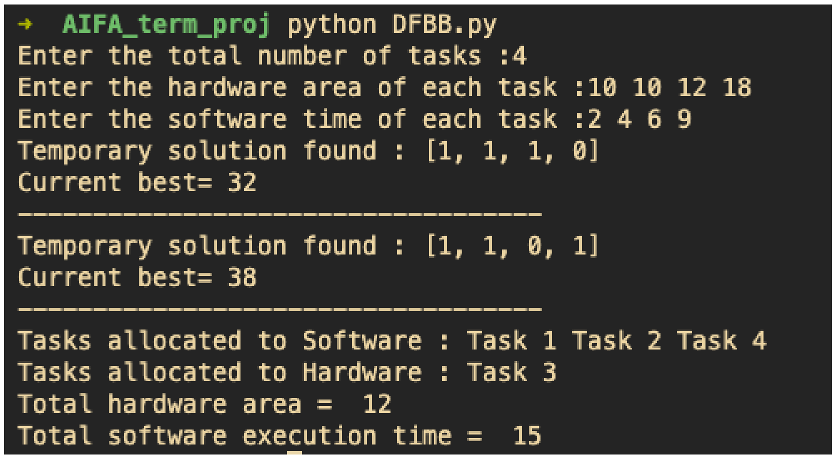
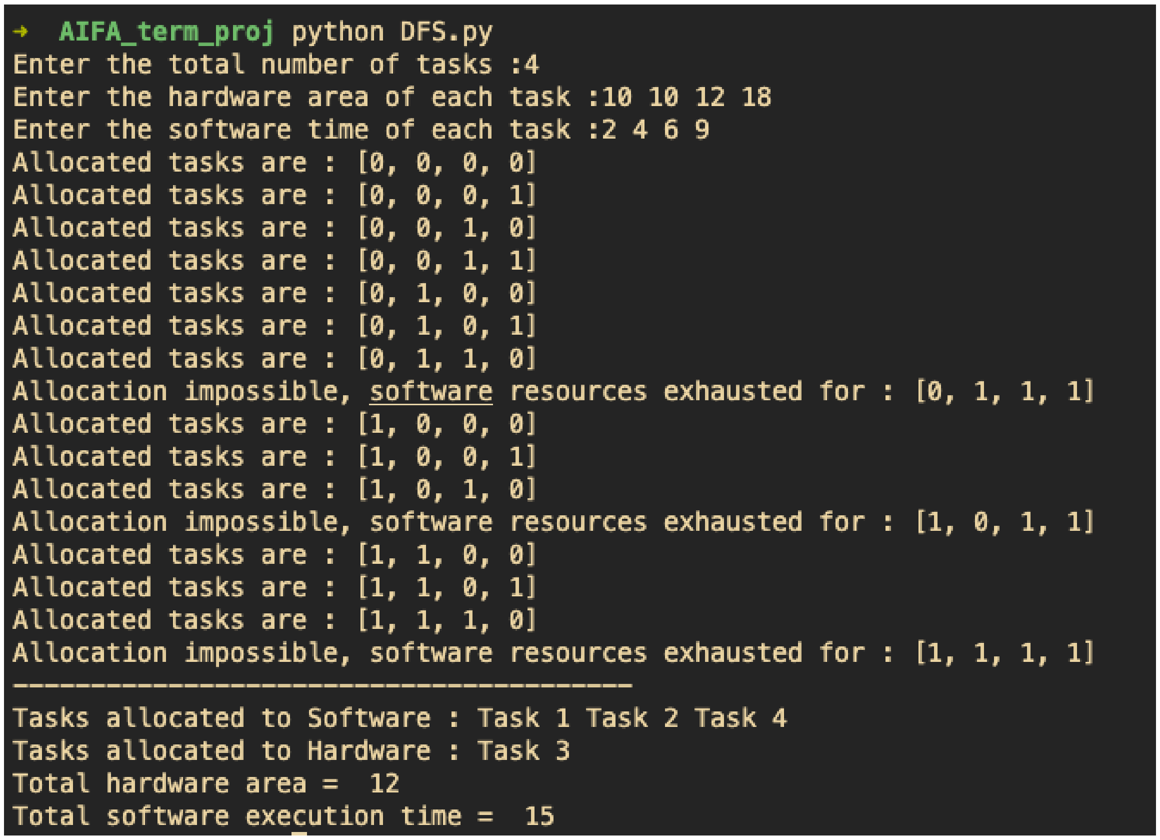

```
➜  AIFA_term_proj python DFBB.py
Enter the total number of tasks :4
Enter the hardware area of each task :10 10 12 18
Enter the software time of each task :2 4 6 9
Temporary solution found : [1, 1, 1, 0]
Current best= 32
-----------------------------------
Temporary solution found : [1, 1, 0, 1]
Current best= 38
-----------------------------------
Tasks allocated to Software : Task 1 Task 2 Task 4
Tasks allocated to Hardware : Task 3
Total hardware area =  12
Total software execution time =  15
```

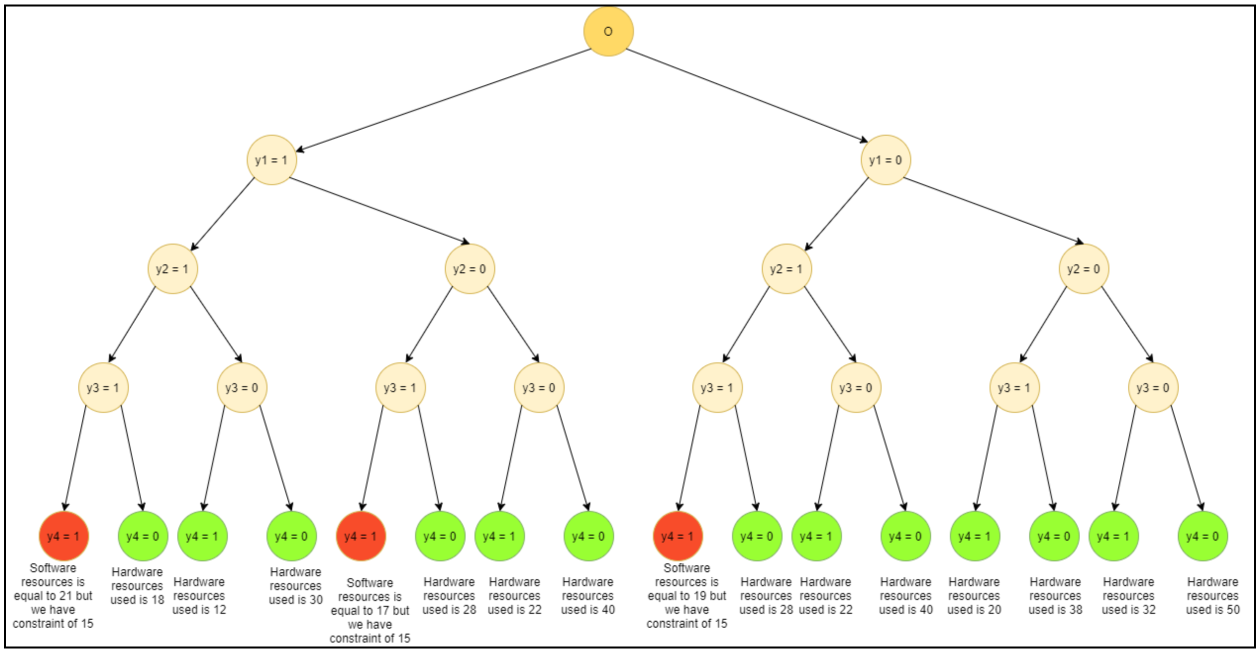
Figure 1. Output obtained with Depth First Branch and Bound. The heuristics used were mentioned in the solution approach



```
➜  AIFA_term_proj python DFS.py
Enter the total number of tasks :4
Enter the hardware area of each task :10 10 12 18
Enter the software time of each task :2 4 6 9
Allocated tasks are : [0, 0, 0, 0]
Allocated tasks are : [0, 0, 0, 1]
Allocated tasks are : [0, 0, 1, 0]
Allocated tasks are : [0, 0, 1, 1]
Allocated tasks are : [0, 1, 0, 0]
Allocated tasks are : [0, 1, 0, 1]
Allocated tasks are : [0, 1, 1, 0]
Allocation impossible, software resources exhausted for : [0, 1, 1, 1]
Allocated tasks are : [1, 0, 0, 0]
Allocated tasks are : [1, 0, 0, 1]
Allocated tasks are : [1, 0, 1, 0]
Allocation impossible, software resources exhausted for : [1, 0, 1, 1]
Allocated tasks are : [1, 1, 0, 0]
Allocated tasks are : [1, 1, 0, 1]
Allocated tasks are : [1, 1, 1, 0]
Allocation impossible, software resources exhausted for : [1, 1, 1, 1]
----------------------------------------
Tasks allocated to Software : Task 1 Task 2 Task 4
Tasks allocated to Hardware : Task 3
Total hardware area =  12
Total software execution time =  15
```

Figure 2. Output obtained with Depth First Search without using any heuristics

The graphical representation of depth first search without any heuristics is given below



o
y1 = 1
y1 = 0
y2 = 1
y2 = 0
y2 = 1
y2 = 0
y3 = 1
y3 = 0
y3 = 1
y3 = 0
y3 = 1
y3 = 0
y3 = 1
y3 = 0
y4 = 1
Software resources is equal to 21 but we have constraint of 15
y4 = 0
Hardware resources used is 18
y4 = 1
Hardware resources used is 12
y4 = 0
Hardware resources used is 30
y4 = 1
Software resources is equal to 17 but we have constraint of 15
y4 = 0
Hardware resources used is 28
y4 = 1
Hardware resources used is 22
y4 = 0
Hardware resources used is 40
y4 = 1
Software resources is equal to 19 but we have constraint of 15
y4 = 0
Hardware resources used is 28
y4 = 1
Hardware resources used is 22
y4 = 0
Hardware resources used is 40
y4 = 1
Hardware resources used is 20
y4 = 0
Hardware resources used is 38
y4 = 1
Hardware resources used is 32
y4 = 0
Hardware resources used is 50

Figure 3. Graphical representation of Depth First Search

As we can see above all the states are explored in depth first search as a result the complexity is exponential. To improve upon this we use heuristics. The explanation and graphical representation of depth first branch and bound is shown below

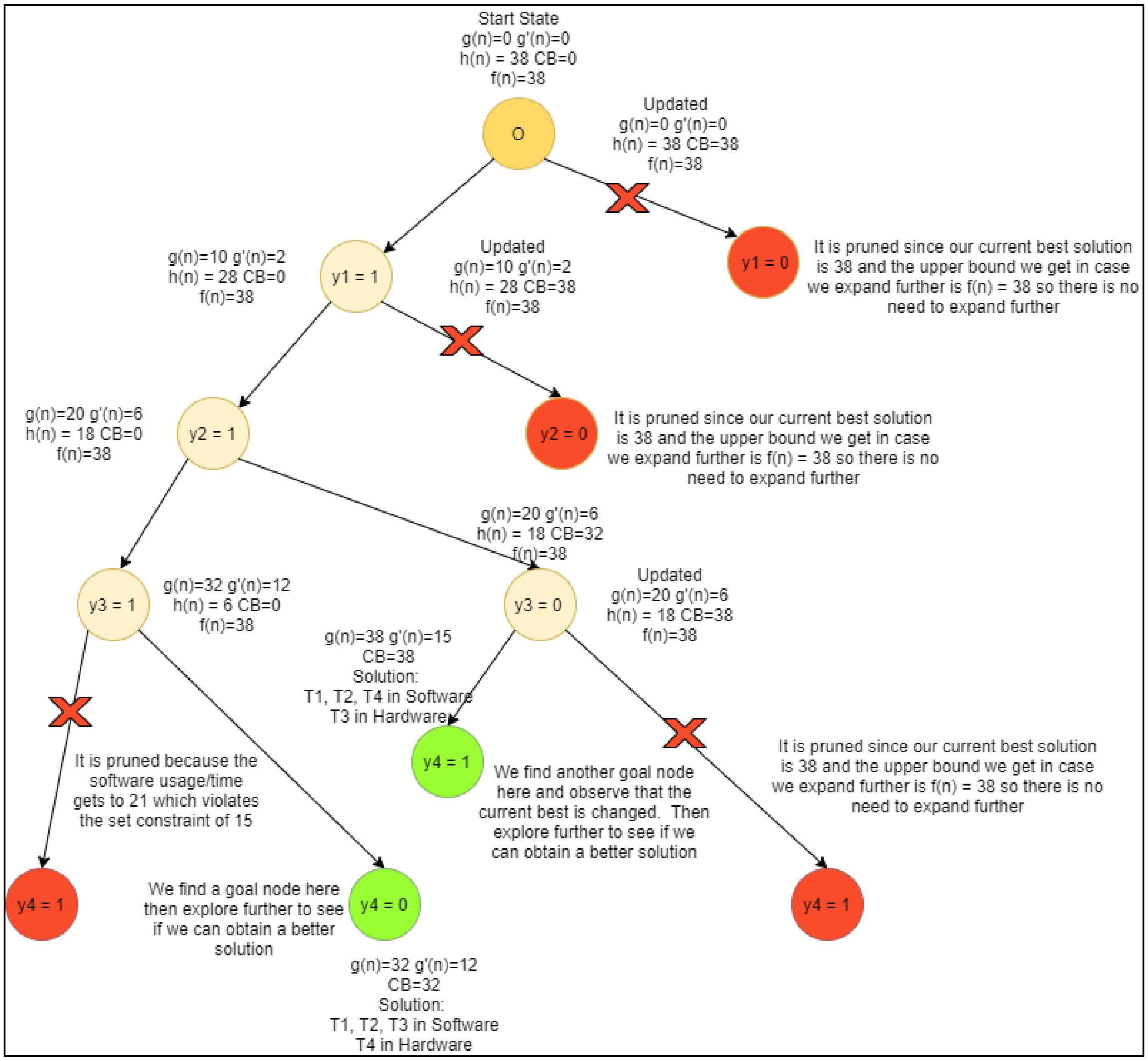
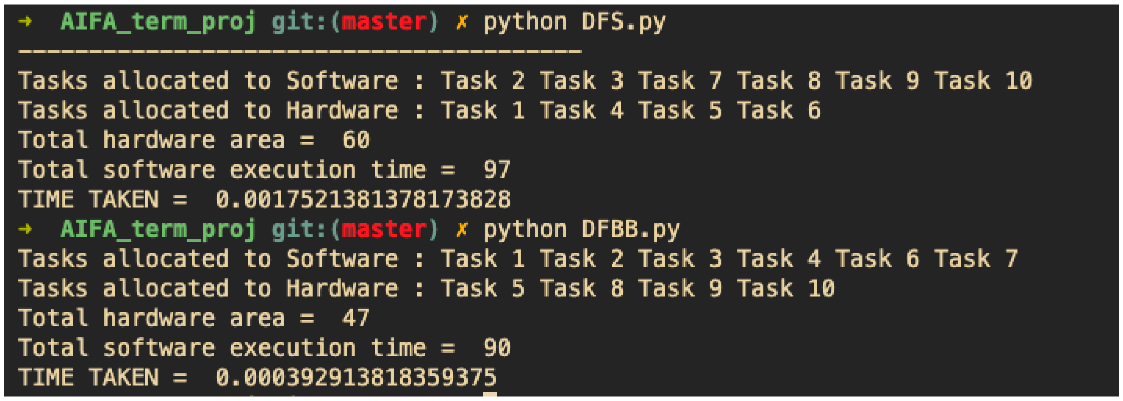

Start State
g(n)=0 g'(n)=0
h(n) = 38 CB=0
f(n)=38
O
Updated
g(n)=0 g'(n)=0
h(n) = 38 CB=38
f(n)=38
y1 = 0
It is pruned since our current best solution is 38 and the upper bound we get in case we expand further is f(n) = 38 so there is no need to expand further
g(n)=10 g'(n)=2
h(n) = 28 CB=0
f(n)=38
y1 = 1
Updated
g(n)=10 g'(n)=2
h(n) = 28 CB=38
f(n)=38
y2 = 0
It is pruned since our current best solution is 38 and the upper bound we get in case we expand further is f(n) = 38 so there is no need to expand further
g(n)=20 g'(n)=6
h(n) = 18 CB=0
f(n)=38
y2 = 1
g(n)=20 g'(n)=6
h(n) = 18 CB=32
f(n)=38
y3 = 1
g(n)=32 g'(n)=12
h(n) = 6 CB=0
f(n)=38
y3 = 0
Updated
g(n)=20 g'(n)=6
h(n) = 18 CB=38
f(n)=38
g(n)=38 g'(n)=15
CB=38
Solution:
T1, T2, T4 in Software
T3 in Hardware
y4 = 1
We find another goal node here and observe that the current best is changed. Then explore further to see if we can obtain a better solution
It is pruned since our current best solution is 38 and the upper bound we get in case we expand further is f(n) = 38 so there is no need to expand further
It is pruned because the software usage/time gets to 21 which violates the set constraint of 15
y4 = 1
We find a goal node here then explore further to see if we can obtain a better solution
y4 = 0
g(n)=32 g'(n)=12
CB=32
Solution:
T1, T2, T3 in Software
T4 in Hardware
y4 = 1

Figure 4. Graphical representation of Depth First Branch and Bound

As we can see above some of the branches are pruned according to our heuristics thus reducing the search complexity. The specific reason for each pruning is given beside each pruned branch. Due to this pruning the complexity required to reach the optimal goal node is less on using depth first branch and bound.

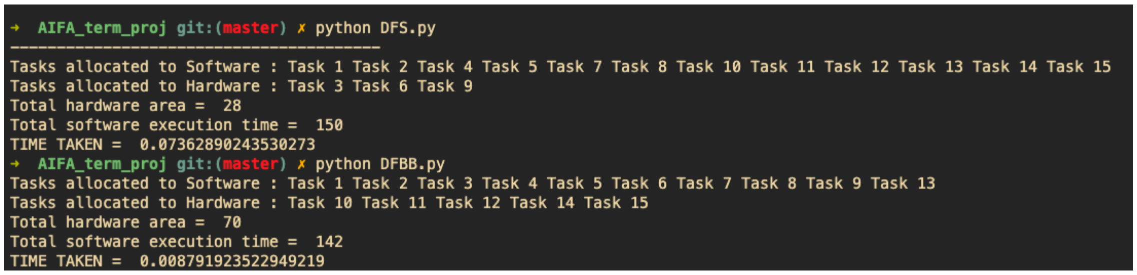
Time Complexity Comparison

We compare the time complexity of DFS and DFBB. We can clearly see the DFBB is faster DFS and as we increase the number of tasks the improvement is even more significant. This is because in DFBB we need to calculate the heuristics and sort our inputs according to H_i/S_i but they are not required in case of DFS. But when there are huge number of tasks we need to explore 2^n number of leaf nodes in DFS which is significantly reduced by using DFBB.



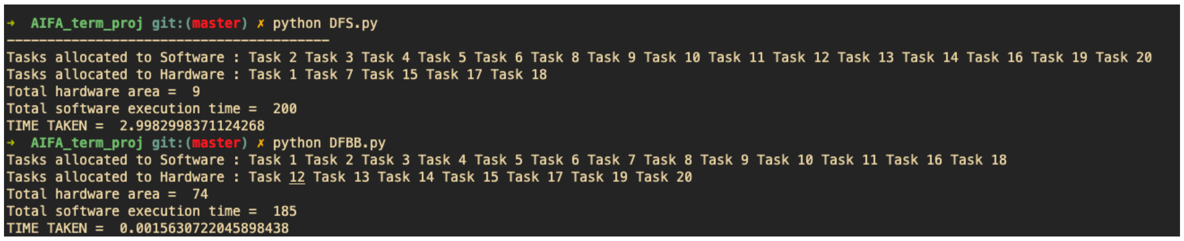
```
➜  AIFA_term_proj git:(master) ✗ python DFS.py
----------------------------------------
Tasks allocated to Software : Task 2 Task 3 Task 7 Task 8 Task 9 Task 10
Tasks allocated to Hardware : Task 1 Task 4 Task 5 Task 6
Total hardware area =  60
Total software execution time =  97
TIME TAKEN =  0.0017521381378173828
➜  AIFA_term_proj git:(master) ✗ python DFBB.py
Tasks allocated to Software : Task 1 Task 2 Task 3 Task 4 Task 6 Task 7
Tasks allocated to Hardware : Task 5 Task 8 Task 9 Task 10
Total hardware area =  47
Total software execution time =  90
TIME TAKEN =  0.0003929138183593750
```

Figure 5. Comparison using 10 Tasks



```
➜  AIFA_term_proj git:(master) ✗ python DFS.py
----------------------------------------
Tasks allocated to Software : Task 1 Task 2 Task 4 Task 5 Task 7 Task 8 Task 10 Task 11 Task 12 Task 13 Task 14 Task 15
Tasks allocated to Hardware : Task 3 Task 6 Task 9
Total hardware area =  28
Total software execution time =  150
TIME TAKEN =  0.07362890243530273
➜  AIFA_term_proj git:(master) ✗ python DFBB.py
Tasks allocated to Software : Task 1 Task 2 Task 3 Task 4 Task 5 Task 6 Task 7 Task 8 Task 9 Task 13
Tasks allocated to Hardware : Task 10 Task 11 Task 12 Task 14 Task 15
Total hardware area =  70
Total software execution time =  142
TIME TAKEN =  0.008791923522949219
```

Figure 6. Comparison using 15 Tasks



```
➜  AIFA_term_proj git:(master) ✗ python DFS.py
----------------------------------------
Tasks allocated to Software : Task 2 Task 3 Task 4 Task 5 Task 6 Task 8 Task 9 Task 10 Task 11 Task 12 Task 13 Task 14 Task 16 Task 19 Task 20
Tasks allocated to Hardware : Task 1 Task 7 Task 15 Task 17 Task 18
Total hardware area =  9
Total software execution time =  200
TIME TAKEN =  2.9982998371124268
➜  AIFA_term_proj git:(master) ✗ python DFBB.py
Tasks allocated to Software : Task 1 Task 2 Task 3 Task 4 Task 5 Task 6 Task 7 Task 8 Task 9 Task 10 Task 11 Task 16 Task 18
Tasks allocated to Hardware : Task 12 Task 13 Task 14 Task 15 Task 17 Task 19 Task 20
Total hardware area =  74
Total software execution time =  185
TIME TAKEN =  0.0015630722045898438
```

Figure 7. Comparison using 20 Tasks

The solution returned by them are different since the software and hardware resources were randomly allocated to each task. As both the codes were ran independently the solution does not remain the same. Since we just wanted to compare the time complexities of the two algorithms the solution need not be the same we only need to allocate the same number of tasks.

Demonstration Video

We also provide a video demonstrating our project. The youtube video link for the same is https://youtu.be/u8bdyL_1zNM

References

1. An Effective Heuristic-Based Approach for Partitioning
2. Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware–Software Codesign
3. What is hardware/software partitioning?
4. Hardware-software partitioning in embedded system design