Optimizing Design Cost of System by using Dynamic Programming

[1,] K Indira Priyadarsini, [2,] Amanulla Mohammad
[1,] *M.TECH (SE)* [2,] *Assistant Professor (CSE)*
G.I.E.T, RAJAHMUNDRY

ABSTRACT

Search based software engineering is an emerging discipline that optimizing the cost of system design by using algorithmic search techniques. For example in the design article intelligence system that help to crime investigation designs. Use the minimal amount of computing between to reduce weight and lost while supporting training and reorganization task running on board. Hard ware and software (system) design is a process which indentify software and hardware knapsack. Dynamic programming is a problem solving technique which solves the optimization design cost. This paper provide how cost. Constrained problem can be modeled as set of two dimensional knapsack problems and provides a frame work and algorithm to optimize design cost of system. An experimental result showing that results reaches the maximum of optimization solution value.

I. INTRODUCTION:

Software development organizations survive in a competitive market by profiting from the conversion of developers' effort to useful and successful software products. To build such products, the organization usually follows a process that divides the development effort into several activities. Each of these activities requires specific characteristics (e.g., such as skills, capabilities, and experience). Most of these characteristics are sought in human resources assigned to accomplish the activities. Search-based software engineering is an emerging discipline that aims to decrease the cost of optimizing system design by using algorithmic search techniques, such as genetic algorithms or simulated annealing, to automate the design search. In this paradigm, rather than performing the search manually, designers iteratively produce a design by using a search technique to find designs that optimize a specific system quality while adhering to design constraints. Each time a new design is produced, designers can use the knowledge they have gleaned from the new design solution to craft more precise constraints to guide the next design search. Search-based software engineering has been applied to the design of a number of software engineering aspects, ranging from generating test data to project management and staffing to software security. A common theme in domains where search-based software engineering is applied is that the design solution space is so large and tightly constrained that the time required to find an optimal solution grows at an exponential rate with the problem size. These vast and constrained solutions spaces make it hard for designers to derive good solutions manually. One domain with solution spaces that exhibit these challenging characteristics is hardware/software co-design. A key problem in these design scenarios is that they create a complex cost-constrained producer/consumer problem involving the software and hardware design. The hardware design determines the resources, such as processing power and memory, that are available to the software. Likewise, the hardware consumes a portion of the project budget and thus reduces resources remaining for the software (assuming a fixed budget). The software also consumes a portion of the budget and the resources produced by the hardware configuration. The perceived value of system comes from the attributes of the software design, *e.g.*, image processing accuracy in the satellite example. The intricate dependencies between the hardware and software's production and consumption of resources, cost, and value make the design solution space so large and complex that finding an optimal and valid design configuration is hard.

II. MOTIVATING EXAMPLE:

This section presents a satellite design example to motivate the need to expand search-based software engineering techniques to encompass cost- constrained hardware/software producer consumer co-design problems. Designing satellites, such as the satellite for NASA's Magnetospheric Multiscale (MMS) mission [11], requires carefully balancing hardware/software design subject to tight budgets Figure 1 shows a satellite with a number of possible variations in software and hardware design.

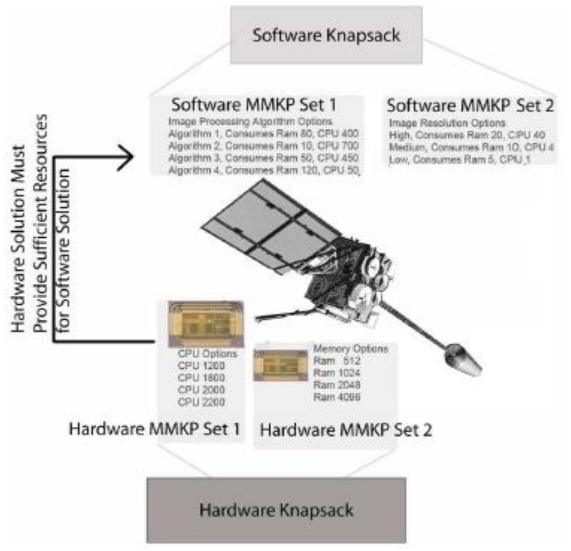

Software Knapsack
Software MMKP Set 1
Image Processing Algorithm Options
Algorithm 1, Consumes Ram 80, CPU 400
Algorithm 2, Consumes Ram 10, CPU 700
Algorithm 3, Consumes Ram 50, CPU 450
Algorithm 4, Consumes Ram 120, CPU 50
Software MMKP Set 2
Image Resolution Options
High, Consumes Ram 20, CPU 40
Medium, Consumes Ram 10, CPU 4
Low, Consumes Ram 5, CPU 1
Hardware Solution Must Provide Sufficient Resources for Software Solution
CPU Options
CPU 1200
CPU 1800
CPU 2000
CPU 2200
Memory Options
Ram 512
Ram 1024
Ram 2048
Ram 4096
Hardware MMKP Set 1
Hardware MMKP Set 2
Hardware Knapsack

Fig .1

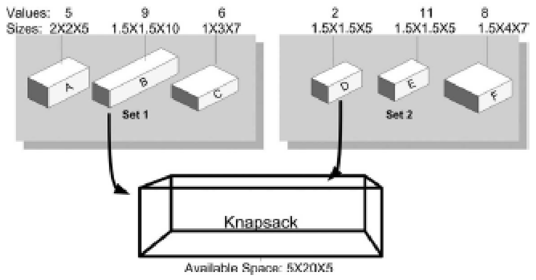


Fig.2

The software and hardware design problems are hard to solve individually. Each design problem consists of a number of design variability points that can be implemented by exactly one design option, such as a specific image processing algorithm. Each design option has an associated resource consumption, such as cost, and value associated with it. Moreover, the design options cannot be arbitrarily chosen because there is a limited amount of each resource available to consume.

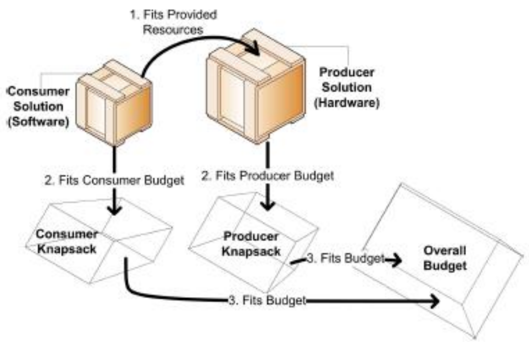


Fig.3

Figure 2 shows an example MMKP problem where two sets contain items of different sizes and values. At most one of the items A,B, and C can be put into the knapsack. Likewise, only one of the items D, E, and F can be put into the knapsack. The goal is to find the combination of two items, where one item is chosen from each set, that fits into the knapsack and maximizes the overall value.

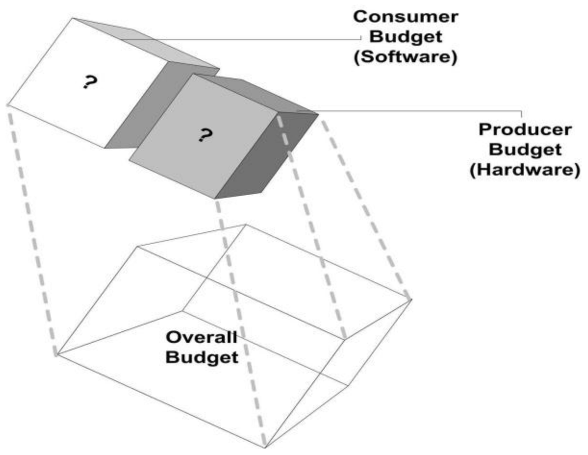
III. PROBLEM DESCRIPTION:

MMKP co-design problem, CoP, as an 8-tuple:
CoP =< Pr,Co, S1, S2, S,R,Uc(x, k),Up(x, k) >
where:
• Pr is the producer MMKP problem (*e*.g., the hardware choices).
• Co is the consumer MMKP problem (*e*.g., the softwarechoices).
• S1 is the size of the producer, Pr, knapsack.
• S2 is the size of the consumer, Co, knapsack.
• R is the set of resource types (*e*.g., RAM, CPU, etc.) that can be produced and consumed by Pr and Co, respectively.
• S is the total allowed combined size of the two knapsacks for Pr and Co (*e*.g., total budget).
• Uc(x, k) is a function which calculates the amount of the resource k ⊂ R **c**onsumed by an item x ⊂ Co (*e*.g., RAM consumed).
• Up(x, k) is a function which calculates the amount of the the resource k ⊂ R **p**roduced by an item x ⊂ Pr (*e*.g., RAM provided).

Let a solution to the MMKP co-design problem be defined as a 2-tuple, < p, c >, where p ⊂ Pr is the set of items chosen from the producer MMKP problem and c ⊂ Co is the set of items chosen from the consumer MMKP problem. A visualization of a solution tuple is shown in Figure 3. value of the solution as the sum of the values of the elements in the consumer solution:

$$V = \sum_{0}^{j} valueof(c_j)$$

where j is the total number of items in c, cj is the jth item in c, and value of() is a unction that returns the value of an item in the consumer solution.

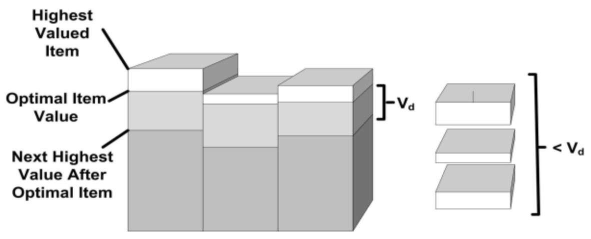


IV. ALGORITHM OVERVIEW:

Inputs:
CoP = < Pr,Co, S1, S2, S,R,Uc(x, k),Up(x, k) >
D = stepsize
Algorithm:
1) For int i = 0 to ⌊ S/D⌋ , set S1 = i ∗ D and S2 =
S − S1
2) For each set of values for S1 and S2:
2.1) Solve for a solution, tc, to Co, given S2
2.2) Calculate a resource consumption heuristic
V r(k) for the resource in r ∈ R:
V r(r) = P|tc|
j=0 Uc(tcj , k)
P|R|
j=0P|tc|
k=0 Uc(tcj , k)

2.3) Solve for a solution, p, to Pr that maximizes the
sum of the values of the items selected for the
knapsack, P|p|
k=0 V alue(pk), where the value of
the kth item is calculated as:
V alue(pk) =
|R|
X
j=0
V r(rj) ∗ Up(pk, rj)
2.4) For each resources rj ∈ R, calculate the amount
of that resource, P(r), produced by the items I p:
P(r) = Up(p0, rj)+Up(p1, rj) . . .Up(p|p|−1, rj)
2.5) Create a new multidimensional knapsack problem,
Cmo, from Co, such that the maximum size
of each dimension of the new knapsack is defined
by the vector:
Sm2 = (S2, r0, r1, . . . r|R|−1)
2.6) Solve for a solution, c, to Cmo and add a solution
tuple < p, c > to the list of candidate solutions,
lc, for CoP
3) Sort the potential solutions, lc, of CoP and output
both the highest valued solution and the list of other
potential solutions.

V. RESULTS:



results are applicable to systems that have hard real time timing constraints and resource consumption characteristics. In particular, resources, such as CPU utilization, must have fixed limits. Moreover, the calculations are based on static worst-case bounds on resource consumption that must be known at design time. The results do not apply to systems where design decisions need to be based on dynamically changing resource consumption profiles. Each experiment used a total of 100 budget iterations (T = 100). We also used the M-HEU MMKP approximation algorithm as our MMKP solver. All experiments were conducted on an Apple MacBook Pro with a 2.4 GHz Intel Core 2 Duo processor, 2 gigabyes of RAM, running OS X version 10.4.11, and a 1.5 Java Virtual Machine (JVM) run in client mode. The JVM was launched with a maximum heap size of 64mb (-Xmx=64m).

VI. EXPERIMENT RESULTS WITH RANDOM DATA:

When compared the algorithms on a series of problems that were completely randomly generated. For these problems, we did not know the true optimal value. We generated 100 problems with 50 sets per MMKP problem and 15 items per set. This yielded a solution space size of 15100. In order to ensure that we generated tractable problem instances, we set extremely loose resource constraints on the problems to create a high probability that a solution existed.

VII. CONCLUSION:

Designing hardware and software in tandem to maximize a system capability can be an NP-hard activity. Search-based software engineering is a promising approach that can be used to leverage algorithmic techniques during system co-design.This paper presented a polynomial-time search-based software 14 engineering technique, called *Allocation-baSed Configuration Exploration Technique* (ASCENT), for finding near optimal hardware/software co-design solutions

This paper also provided how cost. Constrained problem could be modeled as set of two dimensional knapsack problems and provides a frame work and algorithm to optimize design cost of the system.

REFERENCES:

[1] M. Abdelhalim and S.-D. Habib. Modeling Communication Cost and Hardware Alternatives in PSO Based HW/SW Partitioning. In *Proceedings of the International Conference on Microelectronics*, pages 111–114, Dec. 2007.

[2] M. Akbar, E. Manning, G. Shoja, and S. Khan. Heuristic Solutions for the Multiple-Choice Multi-dimension Knapsack Problem. In *Proceedings of the International Conference on Computational Science-Part II*, pages 659–668. Springer-Verlag London, UK, 2001.

[3] A. Barreto, M. Barros, and C. Werner. Staffing a Software Project: A Constraint Satisfaction and Optimization-based Approach. *Computers and Operations Research*, 35(10):3073–3089, 2008.

[4] T. Wiangtong, P. Cheung, and W. Luk. Comparing three heuristic search methods for functional partitioning inhardware–software codesign. *Design Automation for Embedded Systems*, 6(4):425–449, 2002.

[5] K. Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, 2000.

[6] T. Wiangtong, P. Cheung, and W. Luk. Comparing three heuristic search methods for functional partitioning inhardware–software codesign. *Design Automation for Embedded Systems*, 6(4):425–449, 2002.

[7] J. Clark and J. Jacob. Protocols are Programs Too: the Meta-heuristic Search for Security Protocols.*Information and Software Technology*, 43(14):891–904, 2001.

[8] J. Gosling. *Introductory Statistics*. Pascal Press, Glebe, Australia, 1995. [9] M. Harman. The Current State and Future of Search Based Software Engineering. *International Conference on Software Engineering, Minneapolis, MN*, pages 342–357, May 2007.

[9] P.-A. Hsiung, P.-H. Lu, and C.-W. Liu. Energy efficient co-scheduling in dynamically reconfigurable systems. In *Proceedings of the International Conference on Hardware/software Codesign and System Synthesis, Salzburg, Austria*, pages 87–92, October 2007.

[10] G. Antoniol, M. Di Penta, and M. Harman. A RobustSearch-based Approach to Project Management in the Presence of Abandonment,Rework.*Proceedings of the International Symposium on Software Metrics*, pages 172–183, Sept. 2004.

[11 L. Chung. *Non-Functional equirements in SoftwareEngineering*. Springer, 2000.

[12] M. Harman and B. Jones.Search-based software engineering. *Information and Software Technology*,43(14):833–839, 2001.

[13] D. Vanderster, N. Dimopoulos, and R. Sobie. Metascheduling multiple resource types using the MMKP. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain*, pages 231–237, Sept. 2006.

[14] P. McMinn. Search-based software test data generation: a survey. *Software Testing, Verification & Reliability*,14(2):105–156, 2004.

[15] M. Harman. The Current State and Future of Search Based Software engineering. *International Conference on SoftwareEngineering, Minneapolis, MN*, pages 342–357, May 2007