Design and Use of Mobile Apps to Support Problem Transformation Techniques in Algorithmic Thinking

S.R. Subramanya

School of Engineering and Computing, National University, San Diego, USA

Abstract.
Computation / Information technologies are ubiquitous and are the enablers of almost all domains in the real–world, such as manufacturing, transportation, engineering, sciences, healthcare, banking and finance, retail, public administration, commerce, etc. Algorithms are at the core of all computing. Students and professionals in the Computing disciplines (Computer Science, Computer Engineering, Data Science, Information Systems, Artificial Intelligence, etc.) need to have a good grasp of algorithmic techniques and algorithmic thinking skills for effectively solving information–rich, information–driven, real–world and societal problems. Problem transformation techniques are known to enhance algorithmic thinking skills (based on several years of teaching experience), and are important and useful for students and professionals alike. Despite the plethora of mobile Apps, their availability as learning supplements in higher education has been scarce. This paper presents – (1) examples of problem transformations to enhance algorithmic thinking skills, and (2) design aspects of mobile Apps as supplements to support the transformations techniques of (1). These are expected to provide a rich and effective learning experience in algorithmic thinking.

Keywords: algorithmic thinking, transformation techniques, learning supplements, mobile Apps

1. Introduction

The ultimate objectives of all the various facets of computing are about solving real–world problems in minimum times, using minimum computing resources. Algorithms are at the core of the above endeavour. In the increasingly information and computation driven world, a good grasp of algorithmic techniques is extremely important for the development of elegant and efficient solutions to problems in various domains. Also, algorithmic thinking is considered one of the 21st century skills and is important across several workplaces. Students and professionals in the Computing disciplines (Computer Science, Computer Engineering, Data Science, Information Systems, Artificial Intelligence, etc.) need to have a good grasp of algorithmic techniques and algorithmic thinking skills for effectively solving information–rich, information–driven, real–world and societal problems. Even for technical professionals from other disciplines, it is beneficial to have a good grasp of algorithmic techniques which they can later apply effectively to solve problems in their chosen domains of specializations.

The term computational thinking was introduced in (Wing, 2006), where computational thinking is described as consisting of a variety of elements drawn from Computer Science. Among other things, it involves solving problems, designing systems, reformulating a seemingly difficult problem into one we know how to solve, thinking recursively, using abstraction and decomposition, choosing an appropriate representation for a problem or modelling the relevant aspects of a problem to make it tractable, using heuristic reasoning to discover a solution, and several others. A variant, algorithmic thinking is presented in (Futschek, 2006) and describes that algorithmic thinking (a key ability in informatics) can be developed independently from learning programming, by the use of problems that are not easy to solve but have an easily understandable problem definition.

There has been an explosive growth in the number of available mobile Apps spanning a wide diversity of areas and applications. The have been having significant impacts in various aspects of our work and lives. However, there has been a paucity of Apps as supplements to learning in the higher education domain. Their potential and positive impacts in the domain of higher education and learning are yet to be fully realized.

The smartphone and tablet Apps would be very effective supplements to traditional lectures and text books. This is based on the following observations: (a) well–designed smartphone / tablet Apps could incorporate rich media and thus facilitate better retention; (b) the Apps for learning could be made interactive and engaging (using gaming and similar techniques) to hold the users' attention; (c) the Apps could facilitate customization/personalization to cater to different learning styles; (d) they enable 'any–time', 'any–where', 'any how' learning due to their inherent mobility, personalization, and choices; (e) the Apps could cater to short attention spans by presenting a logical unit of content one screen at a time; (f) they could be designed to make learning exploratory by incorporating numerous related content, and providing support for non-linear navigation. This paper presents the general benefits of mobile Apps in learning, and the design aspects which must be considered in such an App to support problem transformation techniques used to enhance algorithmic thinking.

2. Background

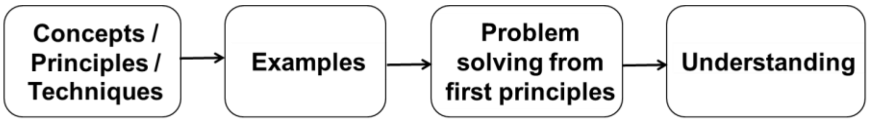
There have been several novel efforts in the development of techniques to facilitate and enhance algorithmic thinking. For example, there have been several examples of work in the areas of teaching computational/algorithmic thinking and programming without the use of computers, but by several paper–and–pencil methods and/or hands-on gadgets/tactile methods. A major aspect of the Australian Informatics Competition (AIC), which has a core focus on algorithms, is a pen–and–paper event, which is described in (Burton, 2010). A new technique for implementing educational programming languages using tangible interface technology, which makes use of inexpensive and durable parts with no embedded electronics or power supplies, is described in (Horn & Jacob, 2007).

Several hands–on gadgets (primarily made of wood / cardboard) which have been developed to facilitate novices in programming and computing to improve the comprehension of a few representative algorithms, and which serve as stepping stones to algorithmic thinking and programming are presented in (Subramanya, 2014), (Subramanya, 2016). These gadgets are designed around several classical problems such as 0/1 knapsack problem, 2D packing problem, sorting, towers of Hanoi problem, etc. These gadgets also act as motivations for persons who may not have mathematical or computing background, but are interested in problem solving and developing programs. Informal qualitative feedback has shown the effectiveness of the hands–on gadgets for the beginners, in facilitating a good understanding of the problems, and in the development of the required solutions.

2.1 Problem transformations in algorithmic thinking

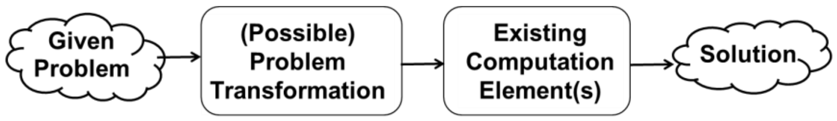
In traditional classrooms and in textbooks for teaching algorithms, the content is delivered in a 'linear' manner, whose flow is shown in the figure below.

Figure 1: Linear exposition of content during traditional learning


Concepts / Principles / Techniques
Examples
Problem solving from first principles
Understanding

In the common 'linear' exposition in teaching algorithms, the principle of the algorithm technique is first described, then a few examples of the problems and their solution using the technique are given, followed by having the learners solve problems using the technique learn from first principles. To enhance the algorithmic thinking process, several novel techniques would be beneficial. These can be viewed as 'non–linear' techniques in the sense that they are not straight–forward delivery of content with concepts and examples, rather they facilitate viewing problems and thinking about solutions from a different angle, and to make connections between seemingly different problems, but sharing a common solution structure. These techniques have been derived from several years of experience in teaching courses on algorithm design. One of the techniques in this category is problem transformation, whose outline is shown in Fig. 2.

Figure 2: Problem transformation for making use of existing computation elements


Given Problem
(Possible) Problem Transformation
Existing Computation Element(s)
Solution

This is one of the important skills in algorithmic thinking and in developing algorithms. Most of the time, the solution to a problem need not be developed from scratch, rather use of existing components to solve problems. However, to make use of the preexisting 'computation element(s)' the original problem needs to be transformed such that it could be given as input expected from the existing computation element(s). This leads to considerable savings in time and effort.

2.2 Benefits of mobile Apps as supplements in learning

While the value of mobile devices, software, and applications in enhancing learning has been recognized, there have not been many studies into what aspects of mobility provide value to learning. Some guidelines for learning in a mobile environment are given in terms of a flexible model is given in (O'Malley, et. al., 2003). These guidelines enable developers, tutors, and learners to identify learning practices and effective pedagogies, and identifying key elements unique to mobile learning. The effectiveness and usefulness of mobile technologies in augmenting traditional teaching methods and in increasing the student engagement, motivation, and interaction is given in (Callaghan, et. al., 2006).

The recent surge in the development of Apps for smartphones has given a new impetus to several mobile learning related initiatives. These have spanned several areas such as (a) development of mobile content and applications, (b) research into pedagogical issues in using mobile devices/applications in learning, (c) study and evaluation of the effectiveness of mobile Apps in learning, etc. A brief summarization of some of the literature related to mobile learning is given in (Cobcroft, et. al., 2006).

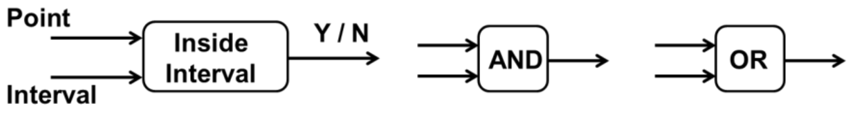
3. Problem transformation in order to use of existing components as an important part of algorithmic thinking

One of the key skills in 'building' algorithms for several applications is not to start from scratch, but to use known and well–tested, pre-existing gadgets / computing elements / components / blocks (design patterns, library routines / functions, etc.) or combinations of those in meaningful ways to obtain the solution to the problem. This requires skills in transforming the original problem in such a way that the pre-existing computing elements could be used. This approach saves tremendous amounts of time and effort. In this section, we present descriptions of a few problems, the descriptions of the computing blocks available, and the task of combining the available computing blocks to build an algorithm to solve a given problem.

3.1 Use of a gadget which determines if a point is in an interval

The following gadgets / "computation units" shown in Figure 3 are already available. The "Inside Interval" box takes a point and an interval as inputs and outputs an Yes (No) depending on whether the point is inside the interval (or not). The AND and OR are the standard logical operations.

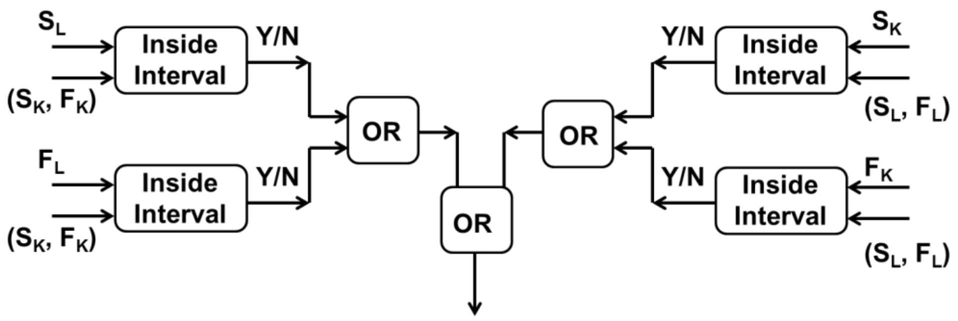
Figure 3: "Computational units" for the interval overlap problem



Using only these existing (predefined) components, it is required to solve the interval overlap problem – to determine whether two intervals (s_k, f_k) and (s_L, f_L) overlap or not. One may use multiple units but it is desirable to use the minimum number possible.

It must be observed that for two intervals to overlap, at least one of the end points of one interval (say K) must be contained inside the other interval (say L). This logic is 'implemented' using combinations of the above blocks is shown in Figure 4 below.

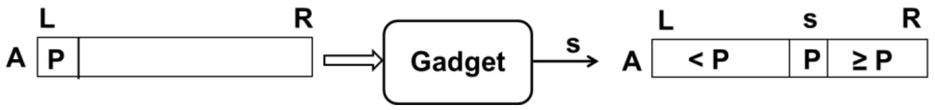
Figure 4: Determining interval overlap using the pre-existing blocks



3.2 Use of a "partition" gadget

Suppose a gadget shown in Figure 5 below is available which takes as input any array A[L..R] of numbers, and partitions A and returns the index (new position of) s of P, the pivot element A[L]. Note that after partition, all elements to the left of index s are less than A[L] and all the elements to the right of s are greater than or equal to A[L]. Given A[0 .. N], an array of size N + 1 containing N numbers in A[1] through A[N], and a number (key) Q, it is required to use the gadget just once and find the number of elements of A which are ≥ Q.

Figure 5: The partition gadget

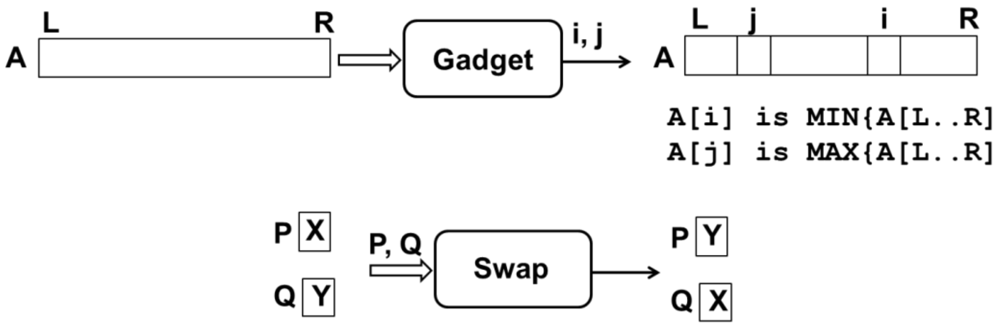


The leftmost element of A, A[L] is set to key Q: A[0] = Q. This array is then input to the "partition gadget". The number of elements ≥ Q is easily seen to be (R – s).

3.3 Use of a gadget that locates the minimum and maximum in an array

A gadget that as input any array A[L..R] of numbers, and locates and returns the indices *i* and *j* of the minimum and maximum elements in the array, and a gadget which swaps the contents of two memory locations given as inputs, are shown in Figure 6 below. Using these gadgets alone, it is required to sort a given array A[0 .. N-1] in non–decreasing order.

Figure 6: Gadgets that locates the minimum and maximum in an array and swaps the contents of two locations



By using the gadget on an input array A[L..R] once, it is clear that A[i] and A[j] would be the minimum and maximum values among A[L] to A[R]. Then, the pairs (A[i], A[L]) and (A[j], A[R]) are swapped using the "Swap" gadget. After this, A[L} and A[R] will have the smallest and largest of the values among A[L] to A[R]. The gadget is next used on A[L+1 .. R-1] followed by swapping, and this is repeated as long as L < R. When the process terminates, the array A will be sorted. The use of the gadgets can also be given in the following sequence of statements.

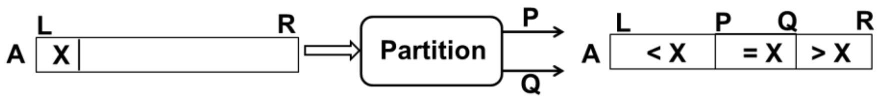
```
while (L < R) do
  (i,j) ← GADGET(A[L..R]);
  Swap (A[L], A[i]);
  Swap (A[R], A[j]);
  L = L + 1; R = R – 1;
endwhile
```

3.4 Use of a "partition" gadget to sort an array

The gadget shown in Figure 7 below takes as input any array A[L..R] of numbers, and produces three partitions. It locates and returns the indices *P* and *Q* of A such that (a) all elements to the left of P are < X, (b) all elements A[P] to A[Q] (inclusive) are equal to X, and (c) all elements to the right of Q are < X (where X = A[L] which is referred to as the pivot).

Using this gadget repeatedly, it is required to sort a given array A[0 .. N-1] in non–decreasing order.

Figure 7: A partition gadget that produces three partitions (values <X, =X, >X, where X=A[L]



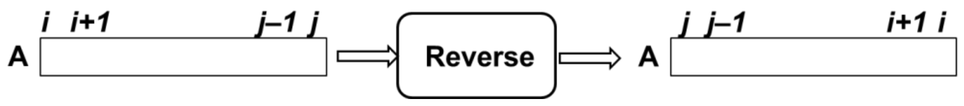
Note that after using the gadget once, the elements A[P] to A[Q] (inclusive) are in their final positions in the final sorted array. Subsequently, the gadget needs to be applied to the two partitions A[L .. P-1] and A[Q+1 .. R], repeatedly as long as P > L and Q < R. The repeated use of this gadget as described is captured in the following sequence of statements.

```
SORT_WITH_GADGET (A[L..R])
begin
  if (L == R)return;
   (P, Q) ← GADGET (A[L..R]);
  SORT_WITH_GADGET (A[L..P-1]);
  SORT_WITH_GADGET (A[Q+1..R]);
end
```

3.5 Use of a gadget that reverses part of a given array

The gadget shown in Figure 8 below takes as input any (part of) array A[i..j] of numbers, and reverses the elements of the array in the index range from *i* to *j*. The task is to use the gadget minimum number of times to rotate left a given array A[1 .. N] by K positions.

Figure 8: A gadget that reverses portion of a given array



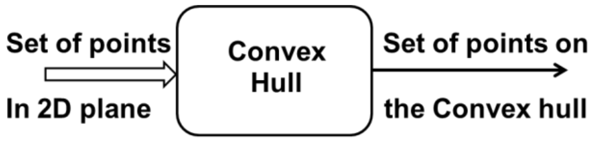
Note that after rotation by left by K positions, the first element of the original array will be at index (N-K+1) MOD N, the second element of the original array at index (N-K+2) MOD N, etc. To accomplish this, the gadget can be used a minimum of three times: (i) first reverse the elements in the index range (1, K), (ii) then reverse the elements in the index range (K+1, N), and (iii) lastly, reverse the elements in the range (1, N). This can easily be verified by writing out the indices after each step. The following sequence of gadget use with the indices shown will accomplish the task.

```
REVERSE (A[1..K]);
REVERSE (A[K+1..N]);
REVERSE (A[1..N]);
```

3.6 Use of a gadget that computes the convex hull of a set of points in 2D plane

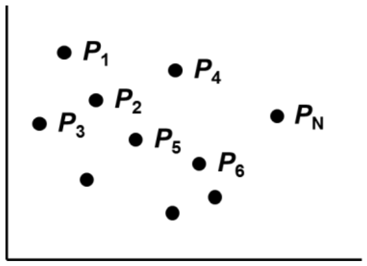
Suppose a gadget (‘computation box’) is available, as shown in Figure 9, which takes as input a set of points in 2D plane, and computes and returns the set of points on the convex hull corresponding to the input.

Figure 9: A gadget that computes the convex hull



Now, given a set of *N* points P_1, P_2, P_3 … … P_N in 2D plane (as shown in Figure 10), it is required to find the farthest pair of points using only the gadget shown in Fig. 9.

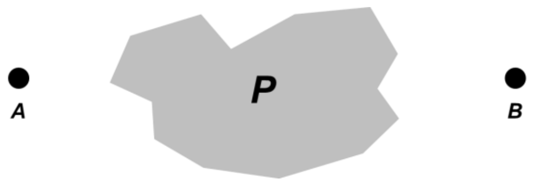
Figure 10: A set of points in 2D plane



Supply the *N* points as input to the gadget (of Fig. 9). Check all pairs in the output to find a farthest pair. Note that the output would have the points on the convex hull corresponding to the input set of points. In general, there are only a constant number of points (independent of *N*) on the convex hull.

As an example of another problem, consider a robot currently at position *A* which needs to move to position *B* by circumventing obstacle (modeled as polygon *P*) in between, as shown in Figure 11. The coordinates of *A*, *B*, and of the vertices of *P* are known. It is required to determine the path of minimum length the robot should take from *A* to *B* avoiding the obstacle, using the gadget of Fig. 9.

Figure 11: Obstacle modelled by a polygon P that need to be circumvented by a robot going from A to B

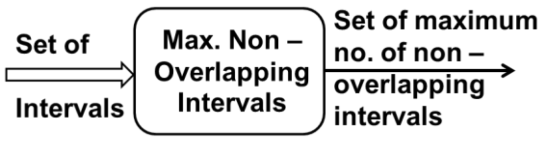


Form the set of points *S* = {*P*} U {*A*} U {*B*}, consisting of A and B, and the vertices of P. Supply *S* as input to the gadget (of Fig. 11). Note that there would be exactly two points from the extremities of the points in *P* 'connecting' to *A* and *B* in the output, and the shorter of the two paths from *A* to *B* which is the desired shortest path.

3.7 Use of a gadget that computes the set of maximum number of non–overlapping intervals

An interval is a tuple (S_i, F_i), where S_i and F_i are real numbers representing the starting and ending points of the interval (temporal or spatial). Suppose a gadget (software routine / design pattern) is available, as shown in Figure 12, which takes as input a set of intervals, and computes and outputs the set of maximum number of non–overlapping intervals. Given a set of intervals {(S_1, F_1), (S_2, F_2), … , (S_N, F_N)}, the start and finish times of N meetings, it is required to schedule the *maximum* number of (compatible / non–overlapping / non–conflicting) meetings.

Figure 12: Gadget which computes the set of maximum number of non–overlapping intervals


Set of Intervals
Max. Non – Overlapping Intervals
Set of maximum no. of non – overlapping intervals

Since each meeting is an intervals, and the gadget readily computes the set of maximum number of non–overlapping intervals, we can directly give the set of meetings as input to the gadget to obtain the required maximum number of non–conflicting meetings.

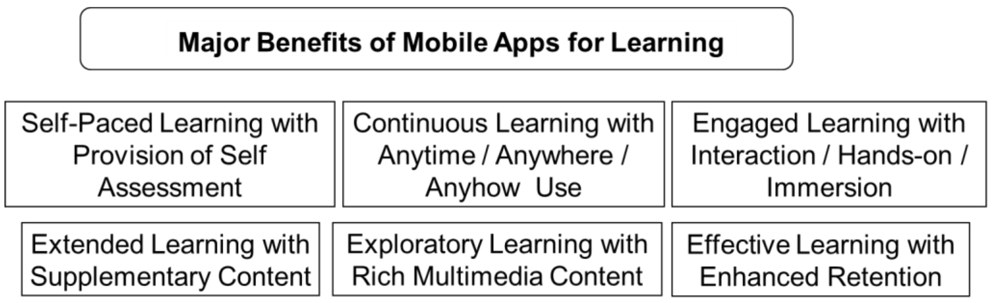
4. Design and use of mobile apps to aid transformation techniques in algorithmic thinking

The mobile Apps have tremendous potential to enhance the effectiveness of learning. In the following subsections, we present (a) the benefits of mobile Apps in learning, (b) the desired features of a mobile App used to support transformation techniques used in algorithmic thinking, and (c) a few general user experience (UX) factors desirable in a mobile App used as supplement to learning.

5.1 Benefits of mobile Apps as supplements in learning

Mobile Apps have tremendous potential as supplements to class lectures and textbooks. The major benefits of using mobile Apps as supplements in the learning aspects are shown in Figure 13.

Figure 13: Major benefits of mobile apps for learning

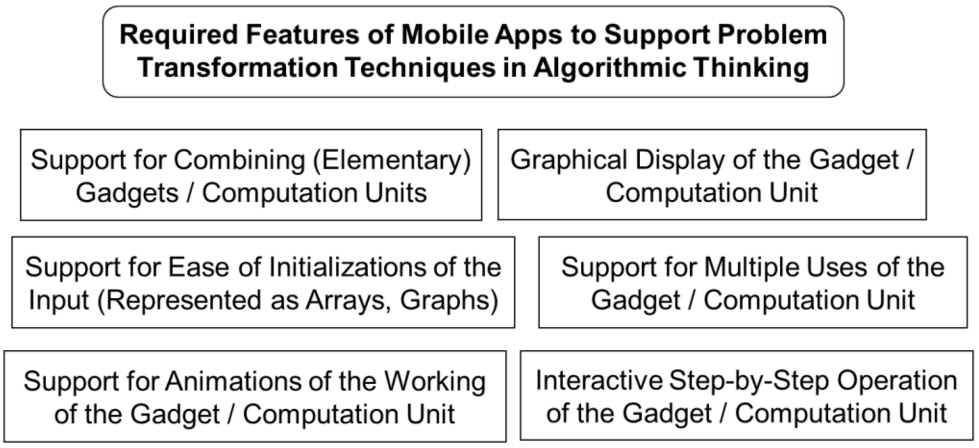

Major Benefits of Mobile Apps for Learning
Self-Paced Learning with Provision of Self Assessment
Continuous Learning with Anytime / Anywhere / Anyhow Use
Engaged Learning with Interaction / Hands-on / Immersion
Extended Learning with Supplementary Content
Exploratory Learning with Rich Multimedia Content
Effective Learning with Enhanced Retention

Mobile Apps as learning supplements facilitate ‘any-time’, ‘any-where’ learning due to their inherent mobility and Web access, which support continuous learning. Use of assessment which is personalized to the level of the learner supports self–paced learning experience. Thoughtfully designed Apps can support customization/personalization to cater to different learning styles. Use of interactions with animations greatly reinforces learning. The Apps for learning could be made interactive and engaging by using judicious combinations of text, graphics, audio, video, and animation (multimedia). In addition, they may use elements of games to provide highly engaging and virtual hands–on learning experience. Judicious use of rich multimedia content will also lead to enhanced retention. The Apps could be designed to be exploratory (as opposed to being limited to a narrow scope) by incorporating numerous related content, and by providing support for non–linear navigation, cross–referencing, and even making cross–disciplinary connections, as appropriate.

5.2 Desired salient features of mobile Apps to support problem transformation techniques in algorithmic thinking

The mobile Apps intended to support problem transformation in algorithmic thinking should have a set of features in order for them to be effective. These are shown in Figure 2, and their brief descriptions are given in Figure 14 below.

Figure 14: Desired salient features of mobile apps to support problem transformation in algorithmic thinking


Required Features of Mobile Apps to Support Problem Transformation Techniques in Algorithmic Thinking
Support for Combining (Elementary) Gadgets / Computation Units
Graphical Display of the Gadget / Computation Unit
Support for Ease of Initializations of the Input (Represented as Arrays, Graphs)
Support for Multiple Uses of the Gadget / Computation Unit
Support for Animations of the Working of the Gadget / Computation Unit
Interactive Step-by-Step Operation of the Gadget / Computation Unit

The App should support easy combinations of elementary/basic computational elements. Facilities to initialize/specify the inputs to the units must be easy. The intermediary and final output(s) while combining the basic units should be clearly output. Well–designed icons/graphical elements representing the computational elements, with clarity and unambiguity would benefit the users in using the basic units and building 'higher–level' units. Interactivity and help guides would ease the users' tasks. Judicious use of animations aid the users in clear understanding of the computational units' behavior and working.

5.3 Major user experience (UX) factors

The effective design of user interface (UI) and user experience (UX) have been gaining importance in the context of mobile devices. These are very crucial to enable the widespread use of mobile Apps. For our purposes, we define UX to be a set of properties of the mobile App which provide a level of abstraction, reducing the (unnecessary) cognitive overload for the user, and enable the efficient, effective use of the App by the user, resulting in an overall pleasurable experience. Some of the major content-related issues that need to be addressed in mobile environments are presented in (Subramanya & Yi, 2005). A model and some issues for the consumption experience of digital educational content have been proposed in (Subramanya, 2012). The proposed model is expected to aid the designers and developers of digital educational content and services to facilitate the improvements of the content consumption experience. Several user experience factors which need to be considered in the design of mobile Apps which are intended to be used as supplements to lectures and text books is presented in (Subramanya, 2014), which are briefly summarized below.

An intuitive and easy–to–use interface would hide the underlying complexities of the system to enable the effective use the App by the uses. The interactions and navigations should be consistent across tasks and functionalities. The average number of (atomic) user actions (ex. key clicks / taps) to accomplish a task should be minimal.

An affordance is a property of an object, or an environment, which allows an individual to perform an action. In the context of human–computer interaction (HCI) it indicates the easy discoverability of possible actions. In the case of mobile Apps, the affordances should be unambiguous to enable selection of actions to achieve intended operations, and eliminate/minimize the number of backtracking.

The average number of navigation levels to accomplish a given task/functionality contributes to the cognitive load. Minimal cognitive load is extremely important for mobile Apps used as supplement to learning. With a low cognitive load, the user can focus more on the content and the learning aspects.

App with low cognitive overload and with a high level of interaction and immersion would be engaging and would sustain the interest of the user to the point of finishing the tasks. This is especially important considering the fact that the attention spans of mobile users are rather short.

5. Conclusion

Algorithms are at the core of all computing. Algorithmic thinking is a very important skill for students and professionals in the Computing disciplines (Computer Science, Computer Engineering, Data Science, Information Systems, Artificial Intelligence, etc.). Mobile Apps can be used as effective supplements for learning by students and professionals in the Computing disciplines. Problem transformation techniques are known to enhance algorithmic thinking skills. This paper presented some representative examples of problem transformations to enhance algorithmic thinking skills, and the design aspects of mobile Apps to be used as supplements to support the transformations techniques. These are expected to facilitate enhancement of algorithmic thinking skills.

References

[1] Wing, J. (2006). “Computational thinking”, *Communication of the ACM*, 49(3), pp.33–35.

[2] Futschek, G. (2006). “Algorithmic Thinking: The Key For Understanding Computer Science”, *Proceedings of the 2nd International Conference on Informatics in Secondary Schools: Evolution and Perspectives* (ISSEP2006), pp.159–168.

[3] Burton, B.A. (2010). “Encouraging Algorithmic Thinking Without a Computer”, *Olympiads in Informatics*, Vol. 4, pp.3–14.

[4] Horn, M.S. and Jacob, R.J.K. (2007). “Designing tangible programming languages for classroom use”, *Proceedings of the 1st international conference on Tangible and Embedded Interaction* (TEI’07), pp.159–162.

[5] Subramanya, S.R. (2014). “Hands–on Gadgets to Facilitate Algorithmic Thinking for Beginners”, *International Journal of Advanced Research in Computer Science & Technology* (eISSN: 2347–8446; pISSN: 2347–9817), Vol. 2, Issue 2, pp.521–526.

[6] Subramanya, S.R. (2016). “Hands–on Gadgets to Facilitate and Augment Algorithmic Thinking and Problem Solving for Beginners”, *International Journal of Software & Hardware Research in Engineering* (ISSN: 2347–4890), Vol. 4, Issue 6, pp.42–50.

[7] O'Malley, C., *et. al*. "Guidelines for Learning/Teaching/Tutoring in a Mobile Environment", *Report of MOBILearn Project* (UoN, UoB, OU/D4.1/1.0) (www.mobilearn.org), pp.1–57.

[8] Callaghan, M.J., *et. al.* (2006). "Technology Supported Pedagogy in a Flexible, Mobile, Learning Environment", *Proc. Association of Pacific Rim Universities Conference on Online Distance Learning and Internet*, Tokyo.

[9] Cobcroft, R. *et. al.* (2006)."Mobile Learning in Review: Opportunities and Challenges for Learners, Teachers, and Institutions", *Proc. Online Learning and Teaching Conference,* Brisbane, pp.21–30.

[10] Subramanya, S.R. and Yi, B.K. (2005). "Mobile Content Provisioning – Major Issues", *Wireless World Research Forum*, Paris, December.

[11] Subramanya, S.R. (2012). "Enhancing Digital Educational Content Consumption Experience", *The Journal of Research in Innovative Teaching*, 5(1), pp.106–115.

[12] Subramanya, S.R. (2014). "Mobile Apps as Supplementary Educational Resources", *International Journal of Advances in Management, Technology & Engineering Sciences* (ISSN:2249–7455), Vol. III, Issue 9 (II), 2014, pp.38–43.