Digital Technologies Research and Applications
http://ojs.ukscip.com/index.php/dtra

Article

Programming Techniques for Considering *m* Desired Conditions from *n* Possible Conditions

Surapon Riyana [1,*], Nigran Homdoung [1,2] and Kittikorn Sasujit [1,2]

[1] Maejo University, Sansai, Chiangmai, 50290, Thailand
[2] School of Renewable Energy, Maejo University, Sansai, Chiangmai, 50290, Thailand

Received: 15 August 2022; **Accepted:** 6 September 2022; **Published:** 19 September 2022

Abstract: The performance of computer programs (or the hardware that can be programmed such as IoTs, embedded computers, and PLCs) is generally based on the complexity of the particular program development technique. High complexity often uses more execution times and system resources. For this reason, the computer program is less computational complexity to be desired. The conditional statements tell computers what certain information is a major cause of computer program complexities, e.g., considering *m* desired conditions from *n* possible conditions. To achieve this aim in computer programs, the data combination is often utilized. However, it is high complexity. Moreover, they cannot give that one condition takes precedence over others. To rid these vulnerabilities of combined conditions, a simple programming technique for considering *m* desired conditions from *n* possible conditions is proposed in this work, which is based on the summation of the condition weights. It only has the complexity of search spaces and data constructions to be *O(n)* and each condition can be set to be different precedence from the others. Furthermore, the proposed technique is evaluated by extensive experiments. From the experimental results, they indicate that the proposed technique is more effective and efficient than the comparative technique.

Keywords: condition weights; weighted summations; data combinations; data considerations; programming techniques

1. Introduction

The introduction should briefly place the study in a broad context and highlight why it is important, in particular, in relation to the current state of research in the field. Finally, it can conclude with a brief statement of the aim of the work and a comment about whether that aim was achieved.

A hardness problem in computer programming is how to specify *m* desired system conditions from n possible system conditions. To rid this problem, the data combination technique is applied. For example, if *a*, *b*, *c*, and *d* are the possible system conditions that must be considered such that they are based on the system limitation that must have two of four system conditions that are satisfied. Thus, there are six combined conditions, i.e., *ab*, *ac*, *ad*, *bc*, *bd*, and *cd*, that must be considered in the system. The infographic of the combined system conditions with this example is shown in Figure 1.

From the example, we can claim that the search spec for considering the desired system conditions can be calculated by Equation (1).

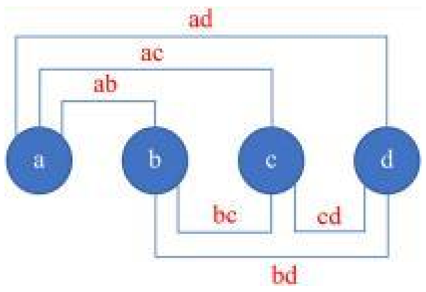


Figure 1. The infographic of the 2-size of the combined system conditions as *a*, *b*, *c*, and *d*.

$$S(n, m) = \frac{n!}{m!(n-m)!} \quad (1)$$

where,

- *n* is the number of all possible conditions,
- *m* is the number of the desired conditions that must be satisfied, and
- $n \geq m$.

In addition, if we use the program iteration [1] to construct all *m*-size combined conditions from *n* possible conditions, the complexity of constructing all *m*-size combined conditions from *n* possible conditions can be calculated by Equation (2). Thus, the complexity of constructing all 2-size combined conditions of a, b, c, and d is based on the program iteration to be 20, i.e., $G(4,2) = [4!−(4−2)!]+[3!−(3−2)!]+[2!−(2−2)!] = (4 * 3) + (3 * 2) + (2 * 1) = 20$.

$$\begin{aligned} G(n,m) = {} & [n! - (n-m)!] + [(n-1)! - ((n-1)-m)!] \\ & + \cdots + [(n-(a-1))! - ((n-(a-1))-m)!] \quad (2) \\ & + [(n-a)! - ((n-a)-m)! \end{aligned}$$

In a real-world case, we suppose that we are developing a smart farm system that uses seven related smart soil moisture sensors, which are independently installed on the farm for controlling the water pump. Moreover, this smart farm system is due to be based on the system limitation that if two of seven soil moisture sensors respond to turn on the water pump, the water pump is turned on by the system. In this situation, if the considerably desired condition processor of the system is based on the brute force technique [2], i.e., data combinations [3–5], there are twenty-one combined system conditions, $S(7,2) =21$, that must be considered. However, if the system limitation for turning on/off the water pump is changed to be three of seven system conditions that must be satisfied. We can see that there are thirty-five combined system conditions, $S(3, 7) = 35$, that must be considered. Furthermore, if a new soil moisture sensor is added to the system (i.e., the system uses eight soil moisture sensors), we can see that there are fifty-six combined system conditions, $S(3,8) = 56$, that must be considered. In addition, aside from the search space cost, the cost of constructing all possible combined conditions must also be considered in data combinations. The cost of constructing all possible combined conditions can be calculated by Equation (2) . From these examples, it is clear that the number of the considered system conditions very much affects the search space for considering the desired system conditions. Moreover, we can observe that this considerably desired system condition processor further has an important vulnerability that must be improved,i.e., it cannot give that one system condition takes precedence over the other system conditions.

To rid of the mentioned vulnerabilities in computer programming, this work proposes a simple, effective, and efficient technique for considering *m* desired system conditions from *n* possible system conditions. That is, the search space for considering *m* desired system conditions from n possible system conditions is reduced to only be n. Moreover, the proposed technique allows the developer who can set the priority of the system conditions to be different precedence. To achieve these aims of the proposed technique, the weighted conditions and their summation are applied.

Related Work

Currently, the various smart technologies (e.g., computers, Internet of Things (IoT) or Internet of Everything (IoE) [6,7], smart homes [8], smart offices [9], smartphones [10,11], smartwatches [12,13], and others) play in the daily life of humans. With smart technologies, we observe that aside from smart devices, the software for controlling them is also very important. Or we can say that the efficiency and effectiveness of smart technologies are often based on their controlled intelligence software. To achieve the intelligence software of smart technologies, there are several essential software development and management techniques to be proposed such as agile development methodology [14–17], scrum development methodologies [18–20], waterfall development methods [21,22], rapid application developments [23,24], feature-driven developments, and extreme programming (XP) [25,26]. Generally, the efficiency and effectiveness of software depend on software development and management techniques. For this reason, software development and management techniques are in place to ensure efficient operations across the software. Aside from software development and management techniques, data security [27–29], data privacy [30–34], and data complexity [35,36] must also be considered. The complexity of software generally directs to affect the software performances [37–39] and the usage resources [40]. To the best of our knowledge about software complexities, conditional statements tell a computer what to do with certain information, it is a major cause of computer software complexity. A conditional statement is often available in computer software, it is considering *m* desired conditions from *n* possible conditions. To achieve this aim in computer software, data combinations [3–5] are often applied. However, data combinations generally have the high complexity. That is, it has the search space to be $O(S(n,m))$, and they further have the complexity of constructing all possibly combined conditions to be $O(G(n,m))$. Moreover, data combinations cannot give that one condition takes precedence over the other conditions. To rid these vulnerabilities of data combinations, a simple, effective, and efficient programming technique for considering *m* desired conditions from *n* possible conditions is proposed in this work, it is based on condition weights and the summation of condition weights. For this reason, the complexity of the proposed technique is only $O(n)$. Moreover, each condition is available in programs, it can be set to be different precedence.

2. Materials and Methods

2.1. Problem Definitions

Definition 1 (Possible system conditions): Let $C = \{c_1, c_2,..., c_n\}$ be the set of all possible conditions that are available in the system such that every $c_z \in C$, where $1 \leq z \leq n$, is represented by a system condition that is in the form as $condition_{vz} \Delta c_z$, where Δ is a particularly compared operation and $condition_{vz}$ is the specified value that is used to compare c_z. For this reason, the compared answer between $condition_{vz}$ and c_z, with using Δ is either "true" or "false".

Example 1 (Possible system conditions): Let C be constructed from c_1, c_2, c_3, and c_4 such that they are represented by the numeric as 5, 2, 3, and 5 respectively. Let v_{c1}, v_{c2}, v_{c3}, and v_{c4} be the compared value for c_1, c_2, c_3, and c_4 respectively such that they are represented by the numeric as 5, 4, 3, and 1 respectively. Let $c_1 = v_{c1}$, $c_2 \geq v_{c2}$, $c_3 > v_{c3}$, and $c_4 > v_{c4}$ be the given compared system conditions. The answer of these compared system conditions is *true*, *false*, *false*, *true* respectively.

Definition 2 (System condition weights): Let $W = \{w \quad w \quad w_{c_n}\}$, where $w_{c1}, w_{c2}, \ldots, w_{c_n} \in N \cup 0$, be the set of the system condition weights or the system condition priorities for c_1, c_2,..., and c_n respectively. Let w_{cz1} , w_{cz2} ew be the specified system condition weight for c_{z1} , c_{z2} e C respectively. If $w_{cz1} > w_{cz2}$, the mean of system conditions is that c_{z1} is the higher priority than c_{z2} .

Definition 3 (System condition sequences): Let $C_{seq} = (c_{Z1}, c_{Z2}, \ldots, c_{Zn})$, where $c_{Z1}, c_{Z2}, \ldots, c_{Zn}$ eC, be the se - quence of$C$ such that they are satisfied by the limitations that are c_{Z1} u c_{Z2} u ... uc_{Zn} = C, c_{Z1} n c_{Z2} n ... n c_{Zn} = D, and $w_{c_{Z1}}$ $w_{c_{Z2}}$ ... w_{cZn}.

From the Example 1, we suppose that the developer needs to set the system condition priority that the system condition c_1 is the highest priority. Moreover, the priority of c_3 is second. Both remaining conditions, c_2 and c_4, are set to be the lowest priority. For this situation, the developer could set the weight of c_1 to be 3. The weight of c_3 is 2. And the weight of c_2 and c_4 is set to be 1. Therefore, the sequence of the system conditions c_1, c_2, c_3, and c_4 is satisfied by the limitation of Definition 3 to be (c_1, c_3, c_2, c_4) or (c_1, c_3, c_4, c_2).

Definition 4 (Desired system conditions): Let D = fd_1, d_2, ..., d_m}, where D C ,be the set of the desired system conditions such that the compared answer between every system condition d_y e D and its compared value v_{dy} is only true.

With Example 1, the system conditions can satisfy the limitation of Definition 4, they are only c_1 and c_4 because they have the compared answer to be true.

2.2. (*n*, *m*)-Conditions Based on Data Combinations

This section is devoted to presenting the (n,m)-Condition technique that based on data combinations. Before this (n,m)-Condition technique will be presented, an important definition of data combinations is defined.

Definition 5 (Desirably combined system conditions): Let m be a positive integer. Let P(C) denote the set of all subsets of C. Let COMB = {comb| comb e P(C) Λ |comb| = m} be the desirably combined system conditions. That is, *COMB* is the set of the subset of C such that every element $comb \in COMB$ has the size to be equal and greater than m and the compared answer between it and its particularly compared value is further true.

This (n,m)-Condition technique is simple and straightforward. It generally uses the program iteration to specify m desired system conditions from n possible system conditions. A (n,m)-Condition algorithm is shown in Algorithm 1. To specify m desired system conditions from n possible system conditions, the size of C is firstly investigated. If the size of C is less than m, the algorithm returns Failure. Because m desired system conditions from C cannot be possible. If not, the second step of the algorithm is enabled, i.e., all possible combinations of C are generated. Finally, the algorithm finds m desired system conditions. If the algorithm discovers m desired system conditions such that m system conditions that have the state to be *True*, the algorithm returns *True*. If not, the algorithm returns *False*.

Algorithm 1: (n,m)-Conditions based on data combinations.

```
Let Cbe the set of all possible conditions;
Let m be the number of the system conditions that must be satisfied;
Let COMB be the set of all possible combinations of C;
if |C|<m then
  return Failure;
  end if
  COMB:=COMBINATION(C);
  while COMB do
    if comb1 ,..., comb_m are True, where comb_1,...,comb_m eCOMB, then
      return True;
    end if
  end while
  return False;
```

With the Algorithm 1, we can claim that the lower bound, the best case, of specifying m desired system conditions from n possible system conditions by using this (n,m)-Condition technique is 1, $\Omega(1)$, and the upper bound, the worst case, of this (n,m)-Condition technique is $G(n,m)$,i.e., $O(G(n,m))$. In the best case, it can occur when the system is extremely fortunate, i.e., the first element of *COMB* is considered to satisfy the specified system conditions. The worst case can occur when the satisfied system conditions are the latest element of *COMB* or the system limitation does not appear in *COMB*.

2.3. (n, m)-Conditions Based on the Summation of the System Condition Weights

This section is devoted to proposing a simple, effective, and efficient (n,m)-Condition technique that is based on the system condition weights and the summation of the system condition weights. Before it will be presented, an important definition is defined.

Definition 6 (Summed weight (n,m)-Conditions): Let *DW* be represented by a positive integer or zero such that it is the referred system condition weight. If $\sum_{y=1}^{m} d_y \gtrsim DW$, where $d_y \in D$, the system is satisfied by the system conditions, otherwise, the system cannot satisfy the system conditions.

To achieve *m* desired system conditions from *n* possible system conditions by using this (n,m)-Condition technique, the weight for every system condition is defined by a positive integer or zero in the first step. For example, we suppose that the system has four conditions c_1, c_2, c_3, and c_4 that must be considered. The system conditions only have the state to be true, which can affect the process(es) of the system. Furthermore, we assume that the priority of these system conditions is different, i.e., c_1 is higher priority than c_2, c_3, and c_4. Moreover, c_2, c_3, and c_4 have the same of the priority. For this situation, the developer who could set the weight for c_1 to be *2*, i.e., w_{c1}=2. The weight for c_2, c_3, and c_4 could be set to be 1,i.e., w_{c2}= 1, w_{c3}= 1, and w_{c4}= 1. In the second step, the system conditions are resorted (re-sequenced) by their weights by descending order. Thus, a resorted data version of the system conditions that is given in this example to be $c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4$. The referred system condition weight *DW* is defined in the third step. We suppose that the referred system condition weight is set to be 3. In the fourth step, the state of these system conditions is investigated, and their weights are summed. Moreover, the summed weight of these system conditions, $\sum_{y=1}^{4} c_y$, and the referred system condition weight DW are compared. If $\sum_{y=1} c_y$ is equal to or greater than *DW* or all possible system conditions are investigated completely, the system investigation processor is ended and sets the returned system state. Finally, the state of the system is returned, i.e., when $\sum_{y=1}^{4} c_y \gtrsim DW$ the returned system state is true, otherwise, the returned system state is false.

The (n,m)-Condition processor based on the condition weights and their summation to be shown in Algorithm 2. With this algorithm, we can claim that the lower bound, the best case, of specifying *m* desired system conditions from *n* possible system conditions by using the summed weight (n,m)-Condition is also 1, $\Omega(1)$. The best case of this (n,m)-Condition technique can occur when the system is extremely fortunate, i.e., the weight of the first considered system condition can satisfy the referred system condition weight. With the upper bound, the worst case, of this (n,m)-Condition technique is *n* or $|C|$, i.e., $O(n)$ or $O(|C|)$, because every system condition can only be considered to be at most one time. The worst case of using the summed weight (n,m)-Condition can occur that the satisfied system limitation can be available when the system must consider all possible system conditions or the specified system limitation cannot be impossible.

Algorithm 2: (n,m)-Conditions based on the condition weights and their summation.

```
Let $c_2$, $c_3$ , … , and $c_n$ be all possible system conditions;
Let $W_{c1}$, $W_{c2}$,..., and $W_{cn}$ be the weight for $c_1$, $c_2$, ..., and $c_n$ respectively;
Let $m$ be the number of system conditions that must be satisfied;
Let $DW$ be the referred summation weight;
Let $SUM$ be the summed weight of the satisfied system conditions;
Let $C_{z1}$, $C_{z2}$,..., and $C_{zn}$ be the resorted data version of $c_1$, $c_2$,..., and $c_n$ by descending order; $SUM$: = 0;
    for g: = 1 to $n$ do
        if $C_{zg}$ is True then
            $SUM$: = $SUM$+$C_{zg}$ ;
            if SUM $\geq$ $DW$ then
                return True;
            end if
        end if
    end for
    return False;
```

With Algorithm 1 and Algorithm 2, they are clear that the summed weight (n,m)-Condition is simple, effective, and efficient than the (n,m)-Condition that based on data combinations.

3. Results and Discussion

In this section, it is proposed to evaluate the effectiveness and efficiency of the summed weight (n,m) -Condition and the (n,m)-Condition that based on data combinations.

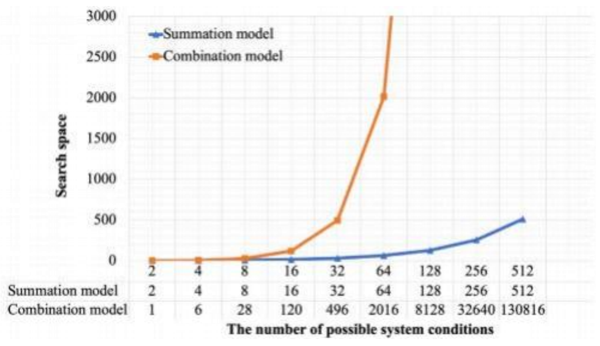
3.1. Experimental Setup

All experiments are proposed in this work, they are conducted on four Intel(R) Xeon(R) Gold 5318H@2.50 GHz CPUs with 512 GB memory and 10 TB NVMe HDD running Windows Server 20222 Standard. Their implementations are built and executed on Microsoft Visual Studio 2022 Community Edition.

3.2. Effectiveness

This section is devoted to evaluating the effectiveness of the summed weight (*n* ,*m*)-Condition by comparing it with the (*n*,*m*)-Condition that based on data combinations.

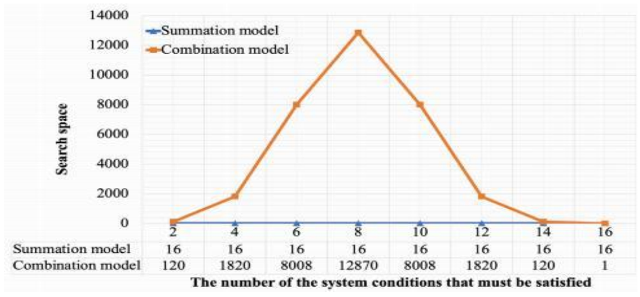
The experimental results are shown in Figure 2. They are proposed to evaluate the effect of search spaces that based on *n*. For experiments, the value of *m*, the number of the desired system conditions, is fixed to be 2. The value of *n*, the number of all possible system conditions, is varied from 2 to 512. In addition, with the summed weight (*n*,*m*)-Condition, every condition's weight is fixed to be 1. Furthermore, we give the system conditions (the desired system conditions) that satisfy the system limitation to appear at the end of the system conditions. The experimental results show that the summed weight (*n*,*m*)-Condition is more effective than the (*n*,*m*)-Condition that based on data combinations. Especially as the number of possible system conditions is increased. The cause of these effects is that the summed weight (*n*,*m*)-Condition takes a linear time, or *O*(*n*). But the complexity of the combined (*n*,*m*)-Condition is factorial, O(*S* (*n* ,*m*)),i.e., the search space of the combined (*n*,*m*)-Condition can be calculated by Equation (1).



	2	4	8	16	32	64	128	256	512
Summation model	2	4	8	16	32	64	128	256	512
Combination model	1	6	28	120	496	2016	8128	32640	130816

Figure 2. The effect of search spaces is based on n.

Another experiment is proposed in this section, it is proposed to evaluate the effect of search spaces that are based on m. For experiments, the value of n, the number of the possible system conditions, is fixed to be 16. The value of m, the number of the desired system conditions, is varied from 2 to 16. In addition, with the summed weight (n,m)-Condition, every condition's weight is fixed to be 1. Furthermore, we also give the system conditions (the desired system conditions) that satisfy the system limitation to appear at the end of the system conditions. From the experimental results as shown in Figure 3, we can also observe that the summed weight (n,m)-Condition is more effective than the (n,m)-Condition that based on data combinations. Moreover, the experimental results of the summed weight (n ,m)-Condition are stable,i.e., the search space of every experimental result of the summed weight (*n*,*m*)-Condition is only 16. Form the experimental results that are shown in Figures 2 and 3, we can conclude that only the number of possible system conditions, *n*, can affect the search space of the summed weight (*n*,*m*)-Condition. Moreover, we observe that the experimental results of the combined (*n*,*m*)-Condition are in the form of bell curves or normal distributions. That is when the number of the desired system conditions, *m*, is increased in the range between 2 and 8, the number of the combined system conditions is also increased after that the number of the combined system conditions have decreased in the range between 8 and 16. This kind of plot is a trend of data combinations.



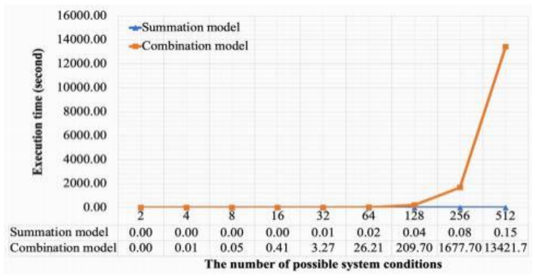
	2	4	6	8	10	12	14	16
Summation model	16	16	16	16	16	16	16	16
Combination model	120	1820	8008	12870	8008	1820	120	1

Figure 3. The effect of search spaces is based on m.

3.3 Efficiency

This section is devoted to evaluating the efficiency of the summed weight (*n,m*)-Condition by comparing it with the (*n,m*)-Condition that based on data combinations.

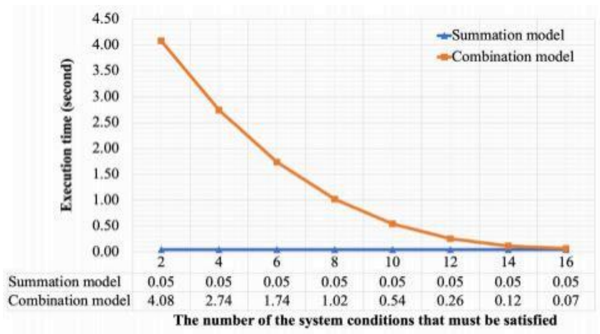
The experimental results are shown in Figure 4. They are proposed to evaluate the effect of execution times that based on *n*. For experiments, the value of *m*, the number of the desired system conditions, is fixed to be 2. The value of *n*, the number of the possible system conditions, is varied from 2 to 512. In addition, with the summed weight (*n*,*m*)-Condition, every condition's weight is fixed to be 1. Furthermore, we give the system conditions (the desired system conditions) that satisfy the system limitation to appear at the end of the system conditions. From the experimental results, we observe that when the number of the possible system conditions, *n*, is increased, both algorithms are using more execution times or less efficiency. That is because the number of the possible system conditions directly influences the efficiency of both experimental algorithms. Moreover, we observe that the summed weight (*n*,*m*) -Condition is more efficient than the (*n*,*m*)-Condition that based on data combinations. Especially as the number of possible system conditions is increased. These experimental results are in accordance with the complexity of them that is presented in Section 3.2 and 3.3. That is, the data construction complexity of the summed weight (*n*,*m*)-Condition is only *O*(*n*) but the data construction complexity of the combined (*n*,*m*)-Condition is *O*(*G* (*n*,*m*)).



	2	4	8	16	32	64	128	256	512
Summation model	0.00	0.00	0.00	0.00	0.01	0.02	0.04	0.08	0.15
Combination model	0.00	0.01	0.05	0.41	3.27	26.21	209.70	1677.70	13421.7

Figure 4. The effect of execution times is based on m.

Another experiment is also proposed to evaluate the efficiency of the summed weight (n,m)-Condition and the (n,m)-Condition that based on data combinations. It is shown in Figure 5. For experiments, the value of n, the number of the possible system conditions, is fixed to be 16. The value of m, the number of the desired system conditions, is varied from 2 to 16. In addition, with the summed weight (n,m)-Condition, every condition's weight is fixed to be 1. Furthermore, we also give the system conditions (the desired system conditions) that satisfy the system limitation to appear at the end of the system conditions. From the experimental results as shown in Figure 5, we can also observe that the summed weight (n,m)-Condition is more efficient than the (n,m)-Condition that based on data combinations. With the summed weight (n,m)-Condition, we observe that its experimental results are stable, or we can say that the number of the desired system conditions, m, does not affect the execution time, efficiency, of the summed weight (n,m)-Condition. With the combined (n,m)-Condition, we observe that the execution time is decreased when the number of the desired system conditions, m, is increased. That is because when increasing the number of the desired system conditions, m, the number of program iterations for constructing the system condition candidates is decreased.



	2	4	6	8	10	12	14	16
Summation model	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Combination model	4.08	2.74	1.74	1.02	0.54	0.26	0.12	0.07

Figure 5. The effect of execution times is based on n.

4. Conclusions

This work is devoted to proposing a simple, effective, and efficient programming technique that is used to specify m desired conditions from n possible conditions. Aside from specifying the desired conditions, the precedence of the conditions that are available in the program, is also considered by the proposed technique. To achieve these aims of the proposed technique, the condition weights and their summation are applied. That is, m desired conditions from n possible conditions can be found when the summation of the desired conditions is equal to or greater than the referred condition summation. From the experimental results, they indicate that the proposed technique is more effective and efficient than the compared technique that is based on data combinations. That is, the complexity of search spaces and data constructions of the proposed model is only $O(n)$ but the compared model has the complexity of search spaces and data constructions to be $O(S(n,m))$ and $O(G(n,m))$ respectively.

Conflicts of Interest

There is no conflict of interest.

References

1. Chentsov, A., 2017. The program iteration method in a game problem of guidance. Proceedings of the Steklov Institute of Mathematics. 297(1), 43–61.
2. Heule, M.J., Kullmann, O., 2017. The science of brute force. Communications of the ACM. 60(8), 70–79.
3. Itai, A., 2001. Generating permutations and combinations in lexicographical order. Journal of the Brazilian Computer Society. 7(3), 65–68.
4. Shen, M.K., 1962. On the generation of permutations and combinations. BIT Numerical Mathematics. 2(4), 228–231.
5. Karp, R.M., 1975. On the computational complexity of combinatorial problems. Networks. 5(1), 45–68. DOI: https://doi.org/10.1002/net.1975.5.1.45.
6. Laghari, A., Wu, K., Laghari, R., et al., 2021. A review and state of art of Internet of Things (IoT). Archives of Computational Methods in Engineering. DOI: https://doi.org/10.1007/s11831-021-09622-6.
7. Suresh, P., Daniel, J.V., Parthasarathy, V., et al., 2014. A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. 2014International Conference on Science Engineering and Management Research (ICSEMR). pp. 1–8. DOI:https://doi.org/10.1109/ICSEMR.2014.7043637.
8. De Silva, L.C., Morikawa, C., Petra, I.M., 2012.State of the art of smart homes. Engineering Applications of Artificial Intelligence. 25(7), 1313–1321. Advanced issues in Artificial Intelligence and Pattern Recognition for Intelligent Surveillance System in Smart Home Environment. DOI: https://doi.org/10.1016/j.engappai.2012.05.002.
9. Papagiannidis, S., Marikyan, D., 2020. Smart offices: A productivity and well-being perspective. International Journal of Information Management. 51(C). DOI: https://doi.org/10.1016/j.ijinfomgt.2019.10.012
10. Deng, T., Kanthawala, S., Meng, J., et al., 2019. Measuring smartphone usage and task switching with log tracking and self-reports. Mobile Media & Communication. 7(1), 3–23. DOI: https://doi.org/10.1177/2050157918761491.

11. Wilmer, H.H., Sherman, L.E., Chein, J.M., 2017. Smartphones and cognition: A review of research exploring the links between mobile technology habits and cognitive functioning. Frontiers in Psychology. 8. DOI: https://doi.org/10.3389/fpsyg.2017.00605.
12. Al-Maroof, R.S., Alhumaid, K., Alhamad, A.Q., et al., 2021. User acceptance of smart watch formedical purposes: An empirical study. Future Internet. 13(5). DOI: https://doi.org/10.3390/fi13050127.
13. Niknejad, N., Ismail, W.B., Mardani,A., et al., 2020. A comprehensive overview of smart wearables: The state of the art literature, recent advances, and future challenges. Engineering Applications of Artificial Intelligence. 90, 103529. DOI: https://doi.org/10.1016/j.engappai.2020.103529
14. Shore, J., Warden, S., 2021. The art of agile development. O'Reilly Media, Inc.
15. Rigby, D.K., Sutherland, J., Noble,A., 2018. Agile at scale. Harvard Business Review. 96(3), 88-96.
16. Abrahamsson, P., Salo, O., Ronkainen, J., et al., 2017. Agile software development methods: Review and analysis. DOI: https://doi.org/10.48550/ARXIV.1709.08439.
17. Abrahamsson, P., Warsta, J., Siponen, M., et al., 2003. New directions on agile methods: A comparative analysis. 25th International Conference on Software Engineering. Proceedings. pp. 244-254. DOI: https://doi.org/10.1109/ICSE.2003.1201204
18. Streule, T., Miserini, N., Bartlome, O., et al., 2016. Implementation of scrum in the construction industry. Procedia Engineering. Selected papers from Creative Construction Conference 2016. 164, 269–276. DOI: https://doi.org/10.1016/j.proeng.2016.11.619
19. Sharma, S., Hasteer, N., 2016. A comprehensive study on state of scrum development. 2016 International Conference on Computing, Communication and Automation (ICCCA). pp. 867-872. DOI: https://doi.org/10.1109/CCAA.2016.7813837.
20. Kniberg, H., 2015. Scrum and XP from the Trenches. Lulu.com.
21. Chari, K., Agrawal, M., 2018. Impact of incorrect and new requirements on waterfall software project outcomes. Empirical Software Engineering. 23(1), 165- 185.
22. Bassil, Y., 2012.A simulation model for the waterfall software development lifecycle. DOI:https://doi.org/10.48550/ARXIV.1205.6904.
23. Chrismanto, A.R., Santoso, H., Wibowo, A., et al., 2019. Developing agriculture land mapping using rapid application development (rad): A case study from Indonesia. International Journal of Advanced Computer Science and Applications (IJACSA). 10(10).
24. Martin, J., 1991. Rapid application development. Macmillan Publishing Co., Inc.
25. Budoya, C., Kissaka, M., Mtebe, J., 2019. Instructional design enabled agile method using addie modeland feature driven development method. International Journal of Education and Development using ICT. 15(1).
26. Nawaz, Z., Aftab, S., Anwer, F., 2017. Simplified fdd process model. International Journal of Modern Education & Computer Science. 9(9).
27. Amoroso, E., 2018. Recent progress in software security. IEEE Software. 35(2), 11–13.
28. Barnum, S., McGraw, G., 2005. Knowledge for software security. IEEE Security & Privacy. 3(2), 74–78.
29. McGraw, G., 2004. Software security. IEEE Security & Privacy. 2(2), 80–83.
30. Riyana, S., Riyana, N., Nanthachumphu, S., 2017. Enhanced (k, e)-anonymous for categorical data. Proceedings of the 6th International Conference on Software and Computer Applications. pp. 62–67.
31. Riyana, S., Nanthachumphu, S., Riyana, N., 2020. Achieving privacy preservation constraints in missing-value datasets. SN Computer Science. 1(4), 1–10.
32. Riyana, S., 2021. (lp1, . . . ,lpn)-privacy: privacy preservation models for numerical quasi-identifiers and multiple sensitive attributes. Journal of Ambient Intelligence and Humanized Computing. pp. 1–17.
33. Riyana, N., Riyana, S., Nanthachumphu, S., et al., 2020. Privacy violation issues in republication of modification datasets. International Conference on Intelligent Computing & Optimization. pp. 938-953. Springer, Cham.
34. Riyana, S., Riyana, N., 2021. A privacy preservation model for rfid data collections is highly secure and more efficient than lkc-privacy. The 12th International Conference on Advances in Information Technology. pp. 1–11.
35. Duran, R., Sorva, J., Leite, S., 2018. Towards an analysis of program complexity from a cognitive perspective. Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER. Association for Computing Machinery, New York, NY, USA. 18, 21–30. DOI: https://doi.org/10.1145/3230977.3230986
36. Wolf-Branigin, M., 2013. Using complexity theory for research and program evaluation. Oxford University Press.
37. Daly, D., Brown, W., Ingo, H., et al., 2020. The use of change point detection to identify software performance regressions in a continuous integration system. ICPE. Association for Computing Machinery, New York, NY, USA. 20, 67–75. DOI: https://doi.org/10.1145/3358960.3375791
38. Aleti, A., Trubiani, C., van Hoorn, A., et al., 2018. An efficient method for uncertainty propagation in robust software performance estimation. Journal of Systems and Software. 138, 222–235. DOI: https://doi.org/10.1016/j.jss.2018.01.010
39. Grechanik, M., Luo,Q., Poshyvanyk, D., et al., 2016. Enhancing rules for cloud resource provisioning via learned software performance models. ICPE. Association for Computing Machinery, New York, NY, USA. 16, 209–214. DOI: https://doi.org/10.1145/2851553.2851568
40. Catuogno, L., Galdi, C., Pasquino, N., 2018. An effective methodology for measuring software resource usage. IEEE Transactions on Instrumentation and Measurement. 67(10), 2487–2494. DOI: https://doi.org/10.1109/TIM.2018.2815431.

Copyright © 2022 by the author(s). Published by UK Scientific Publishing Limited. This is an open access article under the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).