An efficient algorithm for maximum cliques problem on IoT devices

Zine El Abidine Bouneb

Department of Mathematics and Computer Science,
University of Oum El Bouaghi,
B.P. 358 route de Constantine, Oum El Bouaghi 04000, Algeria
Email: bounebzineelabidine@gmail.com

Abstract: This work describes how the maximum clique problem (MCP) algorithm can be performed on microcontrollers in a dynamic environment. In practice, many problems can be formalised using MCP and graphs where our problem is considered in the context of a dynamic environment. MCP is, however, a tricky problem NP-Complete for which suitable solutions must be designed for microcontrollers. Microcontrollers are built for specific purposes and optimised to meet different constraints, such as timing, nested recursion depth limitation, or no recursion at all due to recursion stack limitation, power, and RAM limitation. On another side, graph representation and all the algorithms mentioned in the literature to solve the MCP problem, which is recursive, consumes memory and is designed specifically for computers rather than a microcontroller.

Keywords: maximal clique enumeration; MCE; maximum clique problem; MCP; microcontrollers; internet of things; IoT; agent coalition; symbolic computation; n queens completion problem; MicroPython.

Reference to this paper should be made as follows: Bouneb, Z.E.A. (2023) 'An efficient algorithm for maximum cliques problem on IoT devices', *Int. J. Computational Science and Engineering*, Vol. 26, No. 1, pp.1–11.

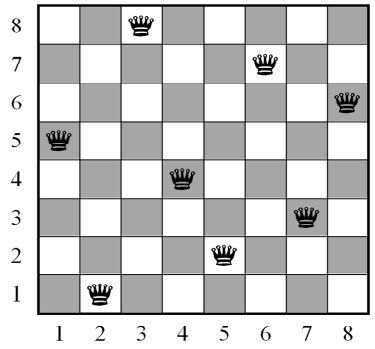
Biographical notes: Zine El Abidine Bouneb received his PhD in Computer Science from Constantine University, Algeria in 2011 with the collaboration of the International Institute of Software Engineering IIST in Macau, SAR of China. His thesis projects focused on developing symbolic distributed algorithms for model checking based on maximality semantics. His primary interest is formal methods and distributed systems. He is working on the application of formal methods to the internet of things.

1 Introduction

Engineers and scientists always build mathematical models with particular abstraction for solving real problems or harnessing specific physical laws for building new genius products. Where nowadays, our science is a stack of abstractions. For example, we can go from Maxwell equations to Ohms law to digital gates to software via many abstractions. We produce a specific genius product going the reverse way by refinement. Refinement is the dual word of abstraction. At a certain level of abstraction, sometimes a problem can be seen from a different angle where scientists can have multiple-choice to use many mathematical models. For example, resource allocation problems can be modelled and solved using integer programming, linear programming, or graph theory. In this paper, we address the issue of the maximum clique on a dynamic graph. By definition, a clique is a complete subgraph, and a maximal clique is the set of all cliques in a given graph.

In contrast, the maximum clique is the largest in the graph, i.e., the maximum cardinality set in the maximal clique. The problem considered here is a variant of the maximum clique problem (MCP) that arises from a dynamic environment. In this paper, we compute the maximum clique that includes specific vertices in the graph. This last can be a particular case of the MCP. To better explain this fact, let us consider the problem of eight queens. This problem was first posed by a German chess player in 1848. The n-queens problem is a generalisation of the above problem: placing n non-attacking queens on n × n chessboard
Figure 1.

Figure 1 Eight queens problem



If we consider the complement of queen graph (Bell and Stevens, 2009), the problem can be solved by enumerating all maximal cliques with *n* cells or vertices, i.e., MCE of an n-clique problem, an NP-complete problem even in the approximate case (Khot, 2001).

If we assume that *m* queens are already positioned on the chessboard, what are the possible solutions for the (*n*-*m*) remaining queens? What is the maximum number of queens we can add to the chessboard if there is no solution? This is what has been considered as n queen completion problem (Gent et al., 2017). The complexity of this problem was first studied in Gent et al. (2017). By analogy to *the n* queens' problem, this paper tries to solve the maximal clique completion problem MCCP for short, a particular case of the MCP. For that reason, we propose a symbolic graph representation suitable for microcontrollers to overcome RAM limitation and a new iterative algorithm for overcoming the stack limitation of recursively calls of functions or no stack at all. For example, the PIC microcontroller 16F877A does not support recursive calls. Programming languages designed for microcontrollers like the MicroPython impose specific maximum recursive depths. For instance, according to our programming experience, MicroPython on the Chinese IoT microcontroller esp8266 has 19 nested recursive calls as a limit. For solving the eight queen problem, formalised as a maximum clique, we cannot use any of the algorithms in the literature due to recursion and the complexity of the problem. The author will review the entire algorithms mentioned in the literature for solving the MCP problem in the next section. Furthermore, we will explain why microcontrollers are unable to support the literature algorithms for solving the problem considered in this paper. After that, the author proposes a new representation of graphs suitable for microcontrollers and a new algorithm dedicated to solving the MCP on this representation. The last sections are an assessment of the algorithm on the queen's problem using python language. Moreover, accurate implementation of the algorithm using MicroPython on the NodeMCU with MAX7219 8×8 LED display can be realised.

2 Background

Today microcontrollers are used heavily in IoT systems. IoT devices, in general, have limited memory and computing power, where challenges are raised to the research community in this field (Nikoui et al., 2021). For that reason, efficiency is a pertinent concern for IoT systems. Many are the suggested propositions in the literature to fulfil this goal. Researchers started looking for a lightweight protocol with good performance to use it as the base protocol for the broker where the IBM protocol MQTT used in the petroleum rigs fits this subject (Kanakaris and Papakostas, 2020).

Furthermore, the growing number of IoT devices and the tremendous volume of daily data make security another crucial objective to consider in IoT systems. Efficiency is the essential criterion, i.e., without efficiency, any proposed solution for a given problem in IoT systems cannot work (Ramu et al., 2020; Xiao et al., 2021). For example, in Zhao et al. (2021), the authors suggest an edge streaming data processing framework for reducing bandwidth resources and transmission delay during data transmission.

Moreover, the environment in IoT is more dynamic (Yang et al., 2021). An IoT device can join the environment as it can leave. For example, in the joining, specific resources are requested, or connection links are established. For instance, in Hu et al. (2021), the authors suggest a probabilistic graph for community detection in a social network where the jargon community is similar to a clique in our context. In contrast, the solution proposed in this paper is an exact solution versus the approximative solution of Hu et al. (2021). We recall that the NP-complete problem can be solved by distribution for parallelisation; for example, if one hundred people want to cut their hair, we have one barber. If the barber takes 20 minutes for each client, the operation needs two days. But if we hire 100 barber, the process spends 20 minutes only. The second approach used approximative techniques based on machine learning. For example, for the graph colouring problem VCP the authors Kashani et al. (2020) use fuzzy irregular cellular automaton (FICA) for finding a near-optimal solution. As we have explained before, the mathematical models used in engineering are related. For example, the VCP is related to the maximal clique's dual problem, the maximum independent set. We draw attention to that the issue presented here can be formalised using other mathematical models used in scheduling. Either for decreasing waiting time by increasing concurrency or reducing energy consumption, we can cite Daid et al. (2021), where the authors use machine learning for facilitating the execution time to reduce energy consumption. The motivation for computing the exact solution in our approach is that mutual exclusion is a critical property. In general, for critical systems, engineers use formal methods. For example, in Ahamad et al. (2021) the authors use formal verification to warranty that the proposed protocol of authentification complies with security properties since security is critical.

IoT infrastructure can add intelligence to our systems. For example, a cell phone takes excellent pictures that are physically impossible with its cheap and tiny camera sensor, but coupling this sensor with the suitable algorithms for image processing can take wonderful quality photos. For example, in Dong et al. (2021), the authors present an excellent solution for on-the-fly electricity bill generation, which is impossible without the internet of things (IoT).

Table 1 PIC MCU Family

	Base line	*Mid-range*	*Enhanced mid-range*	*PIC18*
No. of pins	6–40	8–64	8–64	18–100
Program memory	Up to 3 KB	Up to 14 KB	Up to 28 KB	Up to 128 KB
Data memory	Up to 134 bytes	Up to 368 bytes	Up to 1.5 KB	Up to 4 KB
Instruction length	12-bit	14-bit	14-bit	16-bit
No. of instruction set	33	35	49	83
Speed	5 MIPS*	5 MIPS	8 MIPS	Up to 16 MIPS
Feature	• Comparator • 8-bit ADC • Data memory • Internal oscillator	• In addition of baseline • SPI • I2C • UART • PWM • 10-bit ADC • OP-Amps	• In addition of mid-range • High performance • Multiple communication peripherals	• In addition of enhanced mid-range • CAN • LIN • USB • Ethernet • 12-bit ADC
Deep stack call	0–2 level	2–8 level	2–8 level	Up to 31 level
Families	PIC10, PIC12, PIC16	PIC12, PIC16	PIC12F1XXX, PIC16F1XXX	PIC18

Similarly, in Bouneb and Saidouni (2021), the authors explain the importance of IoT technology in plumbing systems, where the plumbing system will be intelligent due to the software infrastructure behind IoT technology. The authors suggest a house plumbing system with fault tolerance and only one hot and cold water pipe. The paper showed that the concept of scenario alone could not ensure the integrity of the group mutual exclusion property. The idea of a group was first introduced by Joung (2000), where the resource considered is unbound, or we can say a state-based resource.

Despite all of this IoT enhancement, technology cannot handle resource sharing with maximal concurrency. Moreover, the underlying IoT infrastructure cannot handle resource management automatically (Sangaiah et al., 2020). This fact motivated the research community to develop a novel blockchain for automatisation; for example, in Zhang et al. (2021), the authors suggest a novel blockchain for social networks to preserve privacy where social IoT is a particular case of a social network.

Similarly, the algorithms mentioned in the literature for solving group mutual exclusion algorithms, to name a few Luo et al. (2013), Park et al. (2017) and Bashiri et al. (2018), miss automatisation for IoT systems because they need to compute a graph and give the groups manually. After all, the resources considered in those algorithms are unbound.

In Bouneb (2021), the author considers the concept of the group on *bound resources*. Each resource has one instance. In this context, IoT devices participate in computing IoT groups to increase concurrency, as in the case of resources or computing groups based on sustained connection. Group formation is an essential issue in the IoT environment. Many features can be used to assess the correct notion of a group in a given setting. How is the quality of a group configuration measured so that one can say that a grouping of IoT devices in coalitions is better for this IoT device or the whole system than another? In this situation, the author shows that if we use graphs as a mathematical model for computing groups, the problem will be seen as a maximal or MCP. The distributed algorithm presented in Bouneb (2021) computes maximal clique distributively without recursion. All the algorithms presented in the literature, like Das et al. (2020) and Blanuša et al. (2020), use recursion and shared memory. The cloud as shared memory works fine, but those algorithms do not work speciously for critical systems without the internet. In Bouneb (2021), the graph of resources is already coded. Furthermore, for any non-resources graph, the distributed algorithm cannot be applied. In addition, the algorithm cannot be used on standalone IoT devices.

Other approximate approaches have been proposed for the MCP using heuristics (Smith et al., 2019; Babkin et al., 2018). We recall that the distributed algorithms are generally used for the exact solution.

The algorithm presented in this paper can efficiently solve the MCP and its variant maximum clique completion problem. We draw attention that all the algorithms presented in the literature are recursive. IoT devices are, in general, microcontrollers, which are a single on-chip computer or a system on chip (soc) that includes a processor core, data and code memory, Gpio, many on-chip peripherals, and communication interfaces. But in general, it cannot deal well with recursion. Recursion is curried out using the stack data structure. The stack is implemented on the SRAM memory. Hence the stack size will vary during runtime. To store the current fill level of the Stack, the CPU

contains a special register called the stack pointer (SP) in most microcontrollers is 16 bits, which points to the top of the stack. Stacks typically grow 'down', from the higher memory addresses to the lower addresses. So the SP generally starts at the end of the data memory. SP decremented with every push and incremented with every pop. Data space in some implementations of microcontrollers is so tiny that only the stack pointer lows register 8 bits are needed. Hence each function call takes 2 bytes on the stack to store the caller's address and the number and size of local variables, including passed arguments that are also stored on the stack. This fact can determine the number of recursive call depth limits for each microcontroller. For some microcontrollers, recursion is not allowed at all. For that reason, the application helps in the selection of a suitable micro-controller. For example, in the table below, the PIC MCU Family characteristics are divided into four categories: base line, mid-range, enhanced mid-range, PIC18.

3 Maximal clique enumeration problem (MCE)

This section describes the main algorithms proposed in the literature for computing Maximal cliques in a graph.

1 The algorithm from the article by Bron and Kerbosh (1973) is the first algorithm created to find all the maximal cliques of a graph. It works with three sets:
   a A set C which contains the partial clique built at time t.
   b A set T which contains the candidate nodes to enlarge the partial clique.
   c A set D, which contains the nodes that have already been visited to construct a clique.

Algorithm 1

Procedure MCE (C, T, D)
  if $T \cup D = \emptyset$ Then
    report C as a maximal clique
  end if
  choose a pivot vertex p in $T \cup D$
  for each vertex v in $T * N(p)$ Do
    $T := T - v$
    $C := v$
Procedure MCE (C, $T \cup N(v)$, $D \cup N(v)$)
    $C := C - v$
    $D := v$
  end for

The algorithm, therefore, initialises T with all the nodes V of the graph while C and D are empty. Then, the algorithm performs several iterations. During each iteration, a pivot is selected. This fact facilitates the choice of the nodes that can be added to the current clique. The selection of the pivot is random. Therefore, the experiments do not always take the same time nor return the cliques in the same order. The procedure is written below in algorithm 1.

2 Tomita: Tomita et al. (2004) Tomita's algorithm is now the algorithm of reference. Its complexity is of the order of $O(3^{n/3})$. It differs from Bron and Kerbosh's algorithm by choosing the pivot, which generally allows pruning more efficiently. He also uses three sets:
   - A set Q which contains the partial clique which is constructed at a time t.
   - A *SUBG* set and a *CAND* set allow you to select the best possible pivot, then the best candidate, to expand the clique.

   The algorithm initialises the set Q empty while *SUBG* and *CAND* are initialised with the whole nodes V. Pruning is done by choosing the pivot. This choice allows a faster search because it maximises the size of the entire *CAND*. Moreover, if the *CAND* set is empty, but *SUBG* is not empty, we backtrack and return C because the current generated clique C can be a subset of the maximal clique. The procedure is depicted in Algorithm 2.

Algorithm 2

Procedure Tomita (Q, $SUBG$, $CAND$)
  if $SUBG = \emptyset$ Then
    report Q as a maximal clique
  end if
  choose a pivot vertex $u \in SUBG$ which maximise $|CAND \cap N(u)|$
  while $CAND - N(u) \neq \emptyset$ Do
    choose $q \in CAND - N(u)$
    $Q := Q + q$
Procedure Tomita (Q, $SUBG \cap N(v)$, $CAND \cap N(v)$)
    $Q := Q - v$
    $CAND := CAND - q$
  end while

3 Eppstein: Eppstein and Strash (2011) proposed an improvement of Tomita's algorithm. Before executing Tomita, they suggest using the degeneracy of the graph.
   - *Definition 1.1 The degeneracy of a graph G is the smallest number k such that each sub-graph $S \in G$ contains a vertex of degree at most k.* For each graph *G*, we can calculate its order of degeneracy, which is a linear order of vertices such that each vertex has at most *d* neighbours later in order. The degeneracy of a graph and its order can be calculated in linear time. The algorithm uses the same sets as Tomita; however, they are not initialised in the same way (see Algorithm 3).

Algorithm 3

Procedure Tomita (Q, $SUBG$, $CAND$)

for each vertex v_i in a degeneracy ordering v_0, v_1, … of G do

$SUBG$: = $N(v_i) \cap \{v_i + 1, \ldots, v_{n-1}\}$

$CAND$: = $N(v_i) \cap \{v_0, \ldots, v_{i-1}\}$

Procedure Tomita (v_i, $SUBG$, $CAND$)

end FOR

4 MCP

We present in this part a list of algorithms proposed in the literature to solve the MCP in a graph.

1 MC and derivatives: MC is an exact algorithm that finds the clique maximum in a simple graph. This algorithm has several extensions (see Prosser, 2012). In the following, we present the basic principle of MC as well as its main variants.

The first algorithm, MC, uses two sets:

- A set C of nodes, initially empty, which contains the partial clique.

  Constructed at time t and which contains the maximum clique at the end of the algorithm.
- A set P, initialised with all the graph nodes, contains the candidate nodes to enlarge the partial clique. The algorithm also uses a max variable which contains the size of the maximum clique found. The detailed procedure is presented in Algorithm 4.

Algorithm 4

Procedure-MC(C, P, max)

while $P \neq \varnothing$ do

if $|C| + |P| <$ max: return

choose v the last element of P

C: = $C + v$

if $|P \cap \Gamma(v)| = 0$ and $|C| >$ max

Save C as the biggest clique

max: = $|C|$

end if

if $|P \cap \Gamma(v)| \neq 0$

Procedure-MC(C, $P \cap \Gamma(v)$, max)

end if

C: = $C - v$

P: = $P - v$

end while

The first modification to save a little time is the MC0 procedure. We only consider the neighbours whose identifier is less than the selected pivot node v, avoiding duplicates. In Prosser (2012), presenting different versions of the MC algorithm: MCQ, MCS, and BBMC. MCS itself is available in 2 versions MCSa and MCSb. According to the sorting previously carried out by a chosen colouring procedure, all these algorithms have in common to colour the graph.

2 MCF: Pattabiraman et al.(2013) propose an exact algorithm and a heuristic for the MCP applied to large graphs.

The authors propose pruning methods during the algorithm. This pruning makes it possible to avoid exploring specific routes which would not provide better solutions. The heuristic proposed by the authors differs from the standard algorithm since it does not test each node of the set U but takes the one with the maximum degree. The algorithm, however, consists of exploring the nodes whose degree is greater than the size of the maximum clique found (see details in Algorithm 5.1, and its sub-function 5.2)

Algorithm 5.1

MCF()

max: = 0 for each $v \in |E|$ do

if deg(v) ≥ max then = > pruning 1

U: = ∅

for each $n \in \Gamma(v)$

if id(v) > id(n) = > pruning 2

if deg(n) ≥ max then = > pruning 3

U: = $U + n$

end if

end if

end for

CLIQUE(U, 1, max)

end if

end for

Algorithm 5.2

CLIQUE(U, *size*, max)

if $U = \varnothing$ then

if *size* > max then

max: = *size*

return

end if

end if

while |U| > 0 do

if (*size* + $|U|$) ≤ max: return

randomly choose $u \in U$

U: = $U - u$

$\Gamma'(u) = \{w | w \in \Gamma(u)$ and deg(w) ≥ max$\}$

CLIQUE($U \cap \Gamma'(u)$, *size* + 1, max)

end while

In the context of microcontrollers, the algorithms proposed above cannot work as a standalone application executed on

them due to their limited stack of the recursive call and little RAM. Hence we suggest a new iterative (non-recursive) algorithm for maximal and MCPs, which can be adapted easily for the maximum clique completion problem. The algorithm is based on partition refinement of the graph. Where the representation of this graph is a subset of natural numbers only, furthermore, this representation is a symbolic representation that is compact and hides the graph's structure inside it.

There are different ways to optimally represent a graph, depending on the density of its edges, type of operations to be performed, and ease of use. Here are the most commonly used representations to represent a graph.

1 Adjacency matrix: Adjacency matrix is a sequential representation. It is used to represent which nodes are adjacent to each other, i.e., is there any edge connecting nodes to a graph? In this representation, we have to construct a square matrix A of n^2 element. If there is an edge from a vertex i to vertex j, the corresponding element of A, $a_{ij} = 1$, otherwise $a_{ij} = 0$. If there is any weighted graph, then instead of 1*s* and 0*s*, we can store the weight of the edge.

2 Representation and modelling by the incidence matrix: In incidence matrix representation, a graph can be, represented using a matrix A of n rows and m columns such that:

- A(p; k) = 0 if and only if p is not adjacent to the edge j
- A(j; k) = 1 if and only if j is the end of edge k
- A(i; k) = –1 if and only if i is the origin of the edge k

This matrix is the most intensive in memory space.

3 Adjacency list: is generally the most used representation, where we have the most efficient algorithms. A graph can be, represented using a dictionary: it is a simple table entry where each line corresponds to a vertex and includes the linked list of successors (or predecessors) of this vertex.

5 Contribution

5.1 Basic definitions

5.1.1 Definition 1

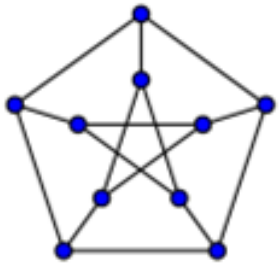
Let V = V = {b_1, b_2, …, b_m} an ordered finite set of m elements, the set P(V) is the set of all subset of V denoted by V and ranged over by {v_1, v_2, …, v_k}. Geometrically, we define a graph as a set of points (vertices) in space interconnected by a set of lines (edges). For graph G, we denote the vertex-set by V and the edge-set by E and write G = (V, E). An edge $e_k = (v_i, v_j) \in E$ iff $v_i \cap v_j = \varnothing$ We make the notation $v_i \sim v_j$. Furthermore $v_i, v_j \in V$ s.t $i \neq j => v_i \neq v_j$. if we consider each bi as a binary digit, it means $b_i \in B = \{0, 1\}$. We have $b_j \in v_i => b_j = 1$ else $b_j = 0$. Each vertex v_i will be coded that can be mapped to a number in N, hence m should be greater or equal to n.

Our proposed symbolic representation is based on Kneser graph (Kneser, 1955). Kneser's graph $K(m, k)$, is the graph whose vertices correspond to the k element subsets of a set of m elements, and where two vertices are adjacent if and only if the two corresponding sets are disjoint. In this paper, we use a particular case of Kneser's graph as defined in definition 1, where the definition of the graph will be just a subset of natural numbers.

We draw attention to the problem of finding a representation for a given graph is a 2-SAT problem with many linear algorithms in the literature Even et al. (1976). But for our context here of microcontroller, we draw attention that most of the algorithm in the literature for solving a 2-SAT or SAT problem is based on recursive backtracking. To overcome this problem, another approach is proposed. If we have a graph with n vertices, instead of coding a vertex on n bits, we will code it at most on 2n bits for avoiding recursive calls see Algorithm 6. The disadvantage of this approach is that it is not efficient in memory space. We recall that the problem considered here is more challenging due to a dynamic environment like the IoT. An advantage of our representation is that it concealed the structure of the graph inside the coding.

Example 1

Figure 2 Petersen graph (see online version for colours)



The Petersen graph in Figure 2 can be represented by: G = {773, 138, 52, 73, 178, 42, 113, 193, 900, 524}

Algorithm 6

Data: $G = (V, E)$, $|V| = N$

Result: $V = \{n_1, n_2, \ldots, n_N\}$

Begin

$Co := N$

for each $v_i \in V$: /* initialisation*/

$n_i := 2^{i-1}$ /* i start from 1 */

for each e = (v_i, v_j) not in E $i < j$:

$co := co + 1$

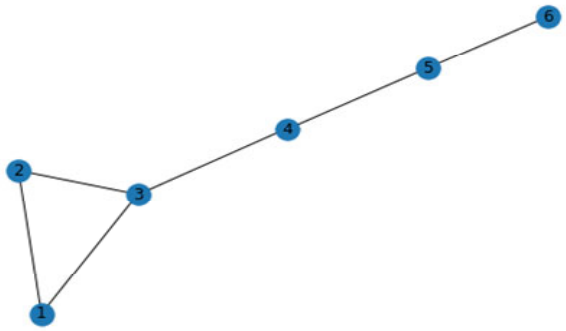
$n_i := n_i + 2^{co}$

$n_j := n_j + 2^{co}$

end

Example 2 Let's consider the graph G in Figure 3, G = (V, E), V = {1, 2, 3, 4, 5, 6}, E = {(1, 2), (1, 3), (2, 3), (3, 4), (4, 5), (5, 6)}

Figure 3 Graph G (see online version for colours)



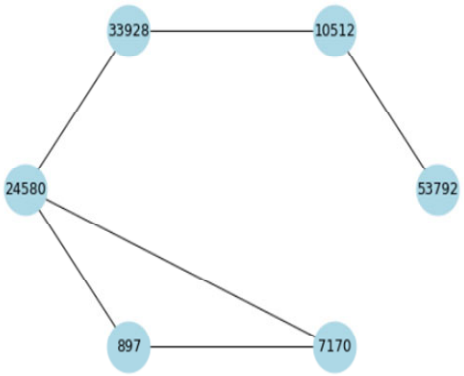
In the initialisation we have V = {n_1 = 1, n_2 = 2, n_3 = 4, n_4 = 8, n_5 = 16, n_6 = 32}, co = 6

For example

(1, 4) ∉ E => co = 7, n_1 = n_1 + 64, n_4 = n_4 + 64. After termination we will get the coding:

$$V = \begin{Bmatrix} V_1 = 897, V_2 = 7{,}170, V_3 = 24{,}580, \\ V_4 = 33{,}928, V_5 = 10{,}512, V_6 = 53{,}792 \end{Bmatrix}$$

Figure 4 Coding for G (see online version for colours)



This coding can be optimised using permutations. A possible optimised coding using Microsoft z3 sat solver is: v = {v5 = 19, v1 = 33, v3 = 20, v6 = 44, v2 = 10, v4 = 40}. The problem with our algorithm is that when the graph is large, the natural number used for the coding gets larger. Natural numbers on microcontrollers are limited using unsigned integers; to overcome this problem, we use tuples for a large number. Python has no limitation on the internal representation of integers. As you can see, the graph is just simple natural numbers that can be loaded easily in the microcontroller RAM, flash memory, or microSD card. Moreover, the algorithm presented here has linear complexity.

We draw attention to that our representation is called symbolic representation. There is a lot of symbolic representation in the literature like binary decision diagram (BDD) and its variant ZDD (zero suppressed BDD)…, etc. (Knuth, 2009). The main goal of BDD is fighting the state space explosion problem, .i.e., reducing spatial complexity. In our context, the objective of the symbolic representation is to avoid recursion and, it does not assume that the number of vertices in the graph is known. In BDD, authors suppose that the number of vertices in a dynamic graph is known beforehand for educational purposes only (Saidouni and Labbani, 2003; Chaki and Gurfinkel, 2018). But in practice, they use the algebra of BDD. Since the problem of knowing the number of vertices beforehand is uncomputable (Catt and Norrish, 2021). In addition, our representation is not concurrent to BDD but can be combined with ZDD (Minato, 1993) for better performance. The problem with the packages for BDDs and its variant in the literature is recursion, which is not convenient in our application. Because the actual tendency is to avoid recursion where the work of Sølvsten et al. (2021) can be used in our context, we let this for future research. Moreover, the symbolic representation presented here is very close to the spatial complexity of BDD.

5.1.2 Algorithm for MCE, MCP and MCCP

Algorithm 7 is similar to Hopcroft (1971) for minimisation of automata, i.e., based on partitions refinement. The algorithm is optimised for a microcontroller, where the *refvec* represents the coding of the vertices, and the cliques are represented with binary coding. For example, the clique c = {897, 7,170, 24,580} is represented symbolically as c = 111,000 regarding the *refvec*, i.e., if bit *i* is set to 1, the vertex *refvec*[*i*] is included in the cliques, 0 if not included. The algorithm is a real implementation using python three, where the function *symboliclique* carries out the coding of the cliques as in San Segundo et al. (2018). The function *ensclique* carries out the inverse operation, transforming the coded clique to a standard set. The function *splitt*2:

$$(z1, o1) = splitt2(refvec.index(mi), c, z1, o1, treated)$$

Split a given coded clique into two parts regarding the vertex mi:

Z1 The vertices *i* in *c* where $mi\&I = 0$; means adjacent to *mi*.

O1 The vertices *i* in c where $mi\&I \neq 0$ means not adjacent to *mi*.

Treated represent the set of the vertices already treated, and splitting has been carried out for the existing cliques relatively to them.

Algorithm 7 at the first partition $\pi_0 = [c_0]$ starts by the whole set of vertices as a clique c_0 and chooses one vertex v_1 and splits c_0 into two cliques c_{11} and c_{12}. After this splitting the set treated will contain v_1, i.e., the partition will be $\pi_1 = [c_{11}, c_{12}]$, treated = $\{v_1\}$. The second iteration continues the operation of splitting for c_{11} and c_{12} regarding v_2. c_{11} will be c_{21} and c_{22}, c_{12} will be c_{23} and c_{24}. We will get the partition: $\pi_1 = [c_{21}, c_{22}, c_{23}, c_{24}]$, treated = $\{v_1, v_2\}$. We draw attention that for all c_{ij} and c_{kl} in π_i c_{ij} and $c_{kl} \neq c_{ij}$ this property represents the notion of maximality in maximal clique problem. We call it in mathematics the anti-chain property

(Clements, 1974). This property needs an anti-chain function not included in our program here for simplicity. The algorithm terminates when he visits all the vertices in the graph.

```
Algorithm 7
Input: a coded graph V = {n1, n2, …, nN}
                                ID = [vi] ⊂ V
                                Output: all maximum cliques completed ID /* of course we can list only one*/
                                refvec = l
                                n = len(refvec)
                                M = [symboliclique(set(refvec))]
                                mi = refvec[0]
                                ens = set([]) #[vi, vj]
                                idvec = Bitset(settovector(ens, refvec))
                                i = 0
                                treated = set([])
while (len(treated) ! = n):
    if (mi not in treated):
        treated.add(mi)
        temp = []
        for c in M:
            r = ensclique(c)
            if mi in r:
                z1 = []
                o1 = []
                (z1, o1) = splitt2(refvec.index(mi), c, z1, o1, treated)
                if idvec and z1 = = idvec:
                    if z1 not in temp:
                        if CountSetBits(z1) > = n:
                            temp.append(z1)
                if idvec and o1 == idvec:
                    if o1 not in temp:
                        if CountSetBits(o1 )> = n:
                            temp.append(o1)
            else:
                if idvec and c == idvec:
                    if c not in temp:
                            temp.append(c)
    else:
        i = i + 1
        mi = refvec[i]
        M = temp #antichain(temp)
print(i)
Mi = list(set(M))
co = 0
for e in range(len(Mi)):
                                co = co + 1
                                print(ensclique(Mi[e]))
                                print(co)
```

If we want to compute maximal cliques MCE, we start with an empty set and delete the green lines in the algorithm above. *countSetBits* is a function that counts the cardinality of a given clique since cliques are represented symbolically; hence this function counts the bits set to one. If the green lines are not deleted, the algorithm solves MCP. If a non-empty list starts the Blue line, the algorithm computes MCCP. We can speed up this algorithm using the same approach as BronKerbosch and its variant by degeneracy ordering and pivoting. The main advantages of the algorithm presented here for this context versus the algorithms in the literature are:

1 the algorithm in this paper is incremental where the algorithms in the literature are not

2 iterative

3 it can be applied in a dynamic environment

4 it can be parallelised by the map and reduce without sharing anything

5 it can be executed on microcontrollers

6 solve the MCCP problem on IoT devices where the algorithms in the literature cannot.

Example After executing the above algorithm on the precedent graph in Figure 3 we get:

```
iteration N = 0 treated = {}
partition = 0 = [897, 7,170, 24,580, 33,928, 10,512, 53,792]
partition = 0 = [{53,792, 897, 7,170, 24,580, 33,928, 10,512}]
iteration N = 1 treated = {897}
partition 1 = [56, 31]
[{897, 7,170, 24,580}, {53,792, 7,170, 24,580, 33,928, 10,512}]
iteration N = 2 treated = {897, 7,170}
partition 2 = [56, 15]
[{897, 7,170, 24,580}, {33,928, 10,512, 53,792, 24,580}]
iteration N = 3 treated = {897, 7,170, 24,580}
partition 3 = [56, 12, 7]
[{897, 7170, 24580}, {33928, 24580}, {33928, 10512, 53792}]
iteration N = 4 treated = {33,928, 897, 7,170, 24,580}
partition 4 = [56, 12, 6, 3]
[{897, 7,170, 24,580}, {33,928, 24,580}, {33,928, 10,512}, {10,512, 53,792}]
iteration N = 5 treated = {897, 7,170, 24,580, 33,928, 10,512}
partition 5 = [56, 12, 6, 3]
[{897, 7,170, 24,580}, {33,928, 24,580}, {33,928, 10,512}, {10,512, 53,792}]
```

iteration N = 6 treated = {53,792, 897, 7,170, 24,580, 33,928, 10,512}

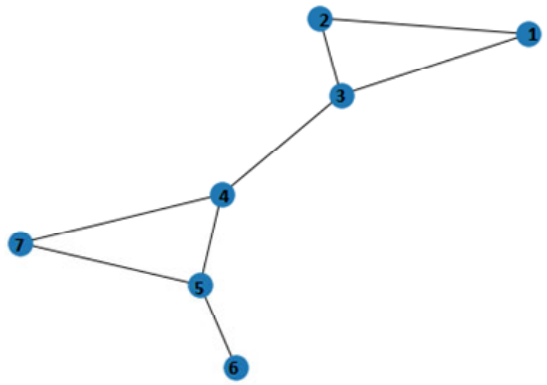
partition 6 = [56, 12, 6, 3]

[{897, 7,170, 24,580}, {33,928, 24,580}, {33,928, 10,512}, {10,512, 53,792}]

5.1.3 Dynamic environment

Let's assume that a new agent is coming and establish links with a few other agents. Each vertex on the graph represents an agent with a microcontroller as its primary intelligence computing unit. The new graph is depicted in Figure 5

Figure 5 Graph G with new links (see online version for colours)



The new edges add to the graph are Ne = {(4, 7), (5, 7)}. For the new coding, we simply use the algorithm of new vertex bellow:

Algorithm new vertex

Data: V = {n_1, n_2, …, n_N}, Ne
Result: V = {n_1, n_2 , …, n_N, n_{N+1}}
Begin
    Co: = N
    n_{N+1}: = 2^N / * i start from 1 */
  for each e = (v_i, v_{N+1})not in Ne:
        co: = co+1
        n_i: = n_i + 2^{co}
        n_{N+1}: = n_{N+1} + 2^{co}
end

Hence the new coding will be V = {3,840, 61,441, 458,754, 528,644, 74,248, 1,721,360, 1,345,568}.

To compute a new set of cliques, we simply add one more iteration to the last partitioning relative to the new vertex V6 coded as 1,345,568. This will be done by initialisation of the first partition M by the last one computed in the precedent execution with the new coding of vertices, and we will get:

Initialisation

Lp [{3840, 61441, 458754}, {458754, 528644}, {74248, 528644}, {74248, 1721360}]

Refvec [3,840, 61,441, 458,754, 528,644, 74,248, 1,721,360, 1,345,568]

M []

For each e n Lp:

M.append(*symboliclique*(*e*))

Treated {3,840, 61,441, 458,754, 528,644, 74,248, 1,721,360}

Iteration N 7 treated = {3,840, 61,441, 458,754, 1,345,568, 528,644, 74,248, 1,721,360}

Partition 7 [112, 24, 13, 6] [{3,840, 61,441, 458,754}, {458,754, 528,644}, {74,248, 1,345,568, 528,644}, {74,248, 1,721,360}]

You can easily see that this algorithm is suitable for microcontrollers in a dynamic environment. Its characteristics are incremental, where the entire algorithms mentioned in the literature cannot solve this problem. Few algorithms in the literature considering the dynamicity of the environment (Yang et al., 2021).

6 Use case

This section will apply the proposed algorithm above for MCE on n queen's problem to compare its speed with the BronKerbosch algorithm endowed with degeneracy ordering and pivoting (Eppstein and Strash, 2011). We draw attention to that the number of iteration in our algorithm equals the number of vertices in the graph. Moreover, our algorithm can have many enhancements, like degeneracy ordering and pivoting see Table 2.

Table 2 Maximal clique enumeration

N queens	*Paper algorithm*	*BronKerbosch*	*Depth recursive call*
3	1ms	997µs	14
4	6ms	3ms	51
5	31ms	5ms	172
6	113ms	27ms	910
7	913ms	106ms	4,526
8	8s	605ms	24,891

Table 3 MCCP with degeneracy ordering but without pivoting

|ID|	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Time	7.95s	593ms	52ms	17ms	18ms	5ms	5ms	9ms	6ms

It is clear from Table 3 that for the maximum clique completion problem, time decrease relative to the cardinality of the set ID, which represents the queens already positioned on the chessboard. This fact proves that the time will decrease for parallelisation than what is shown in Table 2. The parallelisation in the context of microcontrollers will be carried out by distribution. For

example, in the context of IoT systems, each vertex on the graph can represent a microcontroller where each microcontroller computes its group, representing a clique in the graph. The main disadvantage for our algorithm is that for the eight queen graph, we need 4,211 bytes for coding the graph, which is approximately 4.1 kb. Where the optimised coding of the graph using Microsoft z3 SAT solver needs only 436 bytes. If the graph is not dynamic, we can compute the coding beforehand and load it to EEPROM or Flash memory. We let the optimisation of the algorithm for graph coding to future research.

7 Conclusions

This paper proposes an incremental symbolic algorithm for solving the MCP and MCCP where the last problem is a particular case. Due to the assessment in this paper via the *n* queen problem, we believe that the algorithm of the article is the best for microcontrollers where no other algorithm in the literature can fulfil the requirement imposed by microcontrollers. As a perspective, we will try to enhance the spatial complexity of the encoding algorithm using ZDD. The algorithm presented in the paper is compatible with MicroPython because MicroPython is a very compact implementation of Python 3 for embedded systems, and there is no limitation on integers. Moreover, for parallelisation, writing the proposed algorithm in Erlang language with a map and reduce in mind can increase the performance of the proposed algorithm. Erlang is a functional language with no limitation on integers and communication sequential process (CSP) as a style of programming

References

Ahamad, S.S. and Pathan, A-S.K. (2021) 'A formally verified authentication protocol in secure framework for mobile healthcare during COVID-19-like pandemic', *Connection Science*, Vol. 33, No. 3, pp.532–554, DOI: 10.1080/09540091.2020.1854180.

Babkin, E., Babkina, T. and Demidovskij, A. (2018) 'Hybrid neural network and bi-criteria tabu-machine: comparison of new approaches to maximum clique problem', *International Journal of Big Data Intelligence* (*IJBDI*), Vol. 5, No. 3, pp.143–155.

Bashiri, M. et al. (2018) 'PAIM: platoon-based autonomous intersection management', in *21st International Conference on Intelligent Transportation Systems* (*ITSC*) Maui, Hawaii, USA, 4–7 November.

Bell, J. and Stevens, B. (2009) 'A survey of known results and research areas for n-queens, *Discrete Mathematics*, Vol. 309, No. 2009, pp.1–31.

Blanuša, J., Stoica, R., Ienne, P. and Atasu, K. (2020) Manycore clique enumeration with fast set intersections, pp.2676–2690. DOI: https://doi.org/10.14778/3407790.3407853.

Bouneb, Z.E.A (2021) 'A distributed algorithm for computing groups in IoT systems', *International Journal of Software Science and Computational Intelligence* (*IJSSCI*), Vol. 14, No. 1.

Bouneb, Z.E.A. and Saidouni, D.E. (2021) 'Toward an IoT-based software-defined plumbing network system with fault tolerance', *International Journal of Hyperconnectivity and the Internet of Things* (IJHIoT), Vol. 6, No. 1, pp.1–18, DOI: 10.4018/IJHIoT.285587.

Bron, C. and Kerbosch, J. (1973) 'Algorithm 457: finding all cliques of an undirected graph', *CACM*, Vol. 16, No. 9, pp.575–577.

Catt, E. and Norrish, M. (2021) 'On the formalisation of Kolmogorov complexity', in *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and proofs* (*CPP* 2021), Association for Computing Machinery, New York, NY, USA, pp.291–299, https://doi.org/10.1145/3437992.3439921.

Chaki, S. and Gurfinkel, A. (2018) 'BDD-based symbolic model checking', in Clarke, E., Henzinger, T., Veith, H. and Bloem, R. (Eds.): *Handbook of Model Checking*, Springer, Cham, https://doi.org/10.1007/978-3-319-10575-8_8.

Clements, G.F. (1974) 'Sperner's theorem with constraints', *Discrete Math*, Vol. 10, pp.235–255, https://doi.org/10.1016/0012-365X(74)90120-4, DOI: 10.1016/0012-365X(74)90120-4.

Daid, R., Kumar, Y., Hu, Y-C. and Chen, W-L. (2021) 'An effective scheduling in data centres for efficient CPU usage and service level agreement fulfilment using machine learning, *Connection Science*, Vol. 33, No. 4, pp.954–974, DOI: 10.1080/09540091.2021.1926929.

Das, A., Sanei-Mehri, S-V. and Tirthapura, S. (2020) 'Shared-memory parallel maximal clique enumeration', *ACM Trans. Parallel Comput.*, Article 5, Vol. 7, No. 1, https://doi.org/10.1145/3380936 (accessed 28 March 2020).

Dong, Y., Shen, J., Ji, S., Qi, R. and Liu, S. (2021) 'A novel appliance-based secure data aggregation scheme for bill generation and demand management in smart grids', *Connection Science*, Vol. 33, No. 4, pp.1116–1137, DOI: 10.1080/09540091.2021.1882389.

Eppstein, D. and Strash, D. (2011) *Listing All Maximal Cliques in Large Sparse Real-World Graphs*, DOI: 10.1007/978-3-642-20662-7_31.

Even, S., Itai, A. and Shamir, A. (1976) 'On the complexity of timetable and multicommodity flow problems', *SIAM Journal on Computing*, Vol. 5, No. 4, pp.691–703.

Gent, I.P., Jefferson, C. and Nightingale, P. (2017) 'Complexity of n-queens completion', *Journal of Artificial Intelligence Research*, Vol. 59, No. 2017, pp.815–848.

Hopcroft, J. (1971) 'An n log n algorithm for minimizing states in a finite automaton, theory of machines and computations', *Proc. Internat. Sympos.*, *Technion*, Haifa, pp.189–196.

Hu, J., Wang, Z., Chen, J. and Dai, Y. (2021) 'A community partitioning algorithm based on network enhancement', *Connection Science*, Vol. 33, No. 1, pp.42–61, DOI: 10.1080/09540091.2020.1753172.

Joung, Y-J. (2000) 'Asynchronous group mutual exclusion', *Distributed Computing*. Vol. 13, pp.189–206, DOI: 10.1007/PL00008918.

Kanakaris, V. and Papakostas, G.A. (2020) 'Internet of things protocols – a survey', *International Journal of Humanitarian Technology*, Vol. 1, No. 2, pp.101–117, https://doi.org/10.1504/IJHT.2020.112449.

Kashani, M., Gorgin, S. and Shojaedini, S.V. (2020) 'A fuzzy irregular cellular automata-based method for the vertex colouring problem', *Connection Science*, Vol. 32, No. 1, pp.37–52, DOI: 10.1080/09540091.2019.1650329.

Khot, S. (2001) 'Improved inapproximability results for max clique, chromatic number, and approximate graph coloring', pp.600–609, DOI: 10.1109/SFCS.2001.959936.

Kneser, M. (1955) 'Aufgabe 360', *Jahresbericht der Deutschen Mathematiker-Vereinigung*, Vol. 58, No. 2, p.27.

Knuth, D.E. (2009) 'The art of computer programming', *Bitwise Tricks & Techniques Binary Decision Diagrams*, 12th ed., Fascicle 1, Vol. 4, Addison-Wesley Professional, Stanford University.

Luo, A., Wu, W., Cao, J. and Raynal, M. (2013) 'A generalized mutual exclusion problem and its algorithm', *42nd International Conference on Parallel Processing*, pp.300–309, DOI: 10.1109/ICPP.2013.39.

Minato, S-I. (1993) 'Zero-suppressed BDDs for set manipulation in combinatorial problems', in *Proceedings of the 30th international Design Automation Conference* (*DAC* '93), Association for Computing Machinery, New York, NY, USA, pp.272–277, https://doi.org/10.1145/157485.164890.

Nikoui, T.S., Rahmani, A.M., Balador, A. and Javadi, H.H.S. (2021) 'Internet of things architecture challenges: a systematic review', *Int. J. Commun. Syst.*; Vol. 34, p.e4678. https://doi.org/10.1002/dac.4678.

Park, S.H., Young, B. and Kim, Y.K. (2017) 'Group mutual exclusion algorithm for intersection traffic control of autonomous vehicle', in *International Conference Grid, Cloud & Cluster Computing GCC'17*, pp.55–58.

Pattabiraman, B., Patwary, M.M.A., Gebremedhin, A.H., Liao, W. and Choudhary A. (2013) 'Fast algorithms for the maximum clique problem on massive sparse graphs', in Bonato, A., Mitzenmacher, M. and Prałat, P. (Eds.): *Algorithms and Models for the Web Graph. WAW 2013. Lecture Notes in Computer Science*, Vol. 8305, Springer, Cham, https://doi.org/10.1007/978-3-319-03536-9_13

Prosser, P. (2012) 'Exact algorithms for maximum clique: a computational study', *Algorithms*, Vol. 5, No. 4, pp.545–587.

Ramu, G., Mishra, Z., Singh, P. and Acharya, B. (2020) Performance optimized architectures of piccolo block cipher for low resource IoT applications', *International Journal of High-Performance Systems Architecture*, Vol. 9, No. 1, pp.49–57, https://doi.org/10.1504/IJHPSA.2020.107175.

Saidouni, D. and Labbani, O. (2003) 'Maximality-based symbolic model checking', in *ACS/IEEE International Conference on Computer Systems and Applications*, Book of Abstracts, p.98, DOI: 10.1109/AICCSA.2003.1227530.

San Segundo, P., Artieda, J. and Strash, D. (2018) 'Efficiently enumerating all maximal cliques with bit-parallelism', *Computers & Operations Research*, Vol. 92, pp.37–46, DOI: 10.1016/j.cor.2017.12.006.

Sangaiah, A.K., Hosseinabadi, A.A.R., Shareh, M.B., Rad, S.Y.B., Zolfagharian, A. and Chilamkurti, N., (2020) 'IoT resource allocation and optimization based on heuristic algorithm', *Sensors*, Vol. 20. p.539, DOI: 10.3390/s20020539.

Smith, D.H., Perkins, S. and Montemanni, R. (2019) 'Solving the maximum clique problem with a hybrid algorithm', *International Journal of Metaheuristics*, Vol. 7, No. 2, pp.152–175, https://doi.org/10.1504/IJMHEUR.2019.098270.

Sølvsten, S.C., van de Pol, J., Jakobsen, A.B. and Thomasen, M.W.B. (2021) *Efficient Binary Decision Diagram Manipulation in External Memory*, arXiv: 2104.12101.

Tomita, E., Tanaka, A. and Takahashi, H., (2004) 'The worst-case time complexity for generating all maximal cliques, pp.161–170, DOI: 10.1007/978-3-540-27798-9_19.

Xiao, L., Xie, S. Han, D., Liang, W., Guo, J. and Chou, W-K. (2021) 'A lightweight authentication scheme for telecare medical information system', *Connection Science*, Vol. 33, No. 3, pp.769–785, DOI: 10.1080/09540091.2021.1889976.

Yang, Y., Hao, F., Pang, B. et al. (2021) 'Dynamic maximal cliques detection and evolution management in social internet of things: a formal concept analysis approach', *IEEE Transactions on Network Science and Engineering*, https://doi.org/10.1109/NSE.2021.3067939.

Zhang, S., Yao, T., Sandor, V.K.A., Weng, T-H., Liang, W. and Su, J. (2021) 'A novel blockchain-based privacy-preserving framework for online social networks', *Connection Science*, Vol. 33, No. 3, pp.555–575, DOI: 10.1080/09540091.2020.1854181.

Zhao, H., Yao, L.B., Zeng, Z.X., Li, D.H., Xie, J.L., Zhu, W.L. and Tang, J. (2021) 'An edge streaming data processing framework for autonomous driving', *Connection Science*, Vol. 33, No. 2, pp.173–200, DOI: 10.1080/09540091.2020.1782840.