An Algorithm to Support and Improve Breadboard Wiring

Antoine Bossard

Graduate School of Science, Kanagawa University, Yokohama, Japan
Email: abossard@kanagawa-u.ac.jp
Manuscript received December 16, 2025; accepted March 9, 2026; published August 17, 2026

***Abstract*—Prototyping Internet of Things (IoT) devices often involves connecting electronic components by means of a breadboard since it allows for solder-less circuit design and testing. Breadboards share a common basic connection network topology: for example, holes in a same column are directly connected electrically, and there is a gap in the middle of the breadboard that halves hole columns, that is, that cuts the electric connection between the holes of a same column. Furthermore, connections on a breadboard are possible with wires, which is a very flexible approach since it enables the circuit designer to connect components with almost no restriction other than the breadboard dimensions. Here, two types of wires can be distinguished: jumper wires, which are conventional electric wires bending over the breadboard, and what we call hereinafter "flat" wires, which lie on the breadboard, being parallel to its surface. Under such circumstances, the implementation of a circuit is not trivial and often requires a lengthy design process. Besides, a modification, even minor, such as the addition of a resistor or a LED, frequently forces the redesign of the circuit implementation on the breadboard. In this paper, we describe and evaluate an algorithm to support breadboard circuit design and implementation. From a set of electronic components, precisely component pin pairs to be connected, the algorithm generates, if any, a possible design. The obtained result is quantitatively evaluated so that in the case of several possible designs, the best one is retained.**

***Keywords*—IoT, connection, network, matrix, routing, synchronisation**

I. Introduction

Wireless technologies, such as 5G, have enabled the wide adoption of the Internet of Things (IoT), which has in turn been key in the development of physical computing [1, 2]. The design of IoT devices often resorts to building prototypes by means of what is called a "breadboard": electronic components are connected in a flexible manner with temporary wires instead of soldering. The advantages are obvious: wires can be plugged in and out at will. As a result, the pins of electronic components are frequently designed to be compatible with breadboards, or the components come with adapters (also known as breakouts) to fit in the breadboard holes. A breadboard is often coupled with a microcontroller device, such as an Arduino Uno board or a Raspberry system on a chip (SoC) [3]. It can also directly involve a microcontroller, like a PIC [4].

Breadboard wiring, in order to mutually connect the pins of electronic components, involves two main sorts of wires: jumper wires, which enable to freely connect two holes by making an arch above the breadboard, and what we call "flat" wires, which are plugged into the breadboard and lie parallel to it, as if the wires were literally applied and stuck onto it.

Notwithstanding the multiple benefits of such a solder-less breadboard, breadboard wiring can however rapidly become a headache: although relying only on jumper wires is possible, it is not desirable for two main reasons. First, the breadboard would rapidly become completely obfuscated by the wire arches, and second, jumper wires being arcs are necessarily longer than flat connections, which worsens the clock skew effect, for instance when relying on the I^2C communication protocol: synchronisation between I^2C devices is lost, which in the worst case is irrecuperable, and at least requires re-pairing the involved I^2C devices. Hence, a negative impact on both reliability and performance.

It is thus easy to understand that relying on "flat" wires is highly desirable. Electric connection based on flat wires is however not trivial since, by essence, flat wires go along the surface of the breadboard, thus in practice masking, and eventually blocking, the holes that are passed over.

In this paper, we propose an algorithm and the corresponding system to automatically design, that is realise, the breadboard connections requested by the user, such that jumper wires are avoided as much as possible in favour of flat wires.

The rest of this paper is organised as follows: first, related works are discussed in Section II. Then, preliminaries, including notations and definitions, are given in Section III. Next, the proposed breadboard wiring algorithm is described in detail in Section IV. This algorithm is exemplified with a real-world example in Section V and it is experimentally evaluated in Section VI. Finally, this paper is concluded in Section VII.

II. Previous and Related Works

Regarding previous works, one can note the existence of a breadboard simulator, or virtual breadboard, introduced as a pedagogical tool to enable remote teaching and experiments, notably including a circuit checking feature [5]. Wiring nevertheless remains the responsibility of the user (student in this case).

The authors of AutoFritz, another related work, have proposed a complete, pedagogical support system for circuit design centred on a breadboard. AutoFritz provides an autocomplete mechanism to suggest possible breadboard wiring solutions. However, breadboard design support thereof is only partial in that breadboard connections in AutoFritz are simplified by allowing curved wires, which is undesirable, or even impossible, with flat wires [6]. (AutoFritz seemingly does not distinguish between jumper wires and flat wires.)

More generally, circuit design is a notoriously difficult issue, and as such it has become the motivation of, for instance, the crossing number problem of graph theory [7–9]: the aim is to reduce the number of wires crossing to a minimum, since it is an obstacle for printed circuit board design. A similar issue arises with breadboard circuit design. It is important to note that the crossing number problem is

NP-hard [10].

III. Preliminaries

In general and in this research, a breadboard consists of two halves, separated by a gap. Each half is a matrix of holes (the hole separation is conventionally set at 2.54 mm), with each column of the matrix electrically connecting the holes of the column. On the contrary, the holes of a same row of the matrix are not electrically connected [11]. Some breadboards additionally feature lines on the exterior of the two halves: because those special side lines comparatively pose few wiring problems, they are ignored here.

One half of a breadboard typically consists of five rows and a few dozens of columns: because this can vary, the row and column number are two input parameters of the proposed algorithm (see Section IV).

Given this configuration, for a breadboard of R rows per half and C columns, we define a node u as a triple (h, c, r), with $h \in \{0, 1\}$ the breadboard half of u, c $(0 \leq c < C)$ its column number and r $(0 \leq r < R)$ its row number. Furthermore, let $h(u)$, $r(u)$ and $c(u)$ respectively be the half, row number and column number of node u. An illustration is given in Fig. 1.

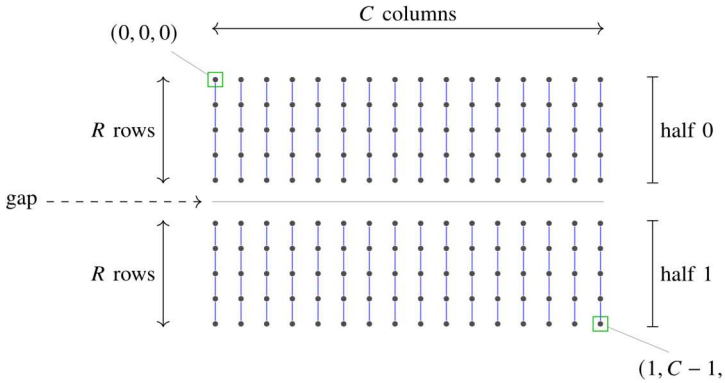

Fig. 1. The breadboard nodes (grey dots) and the implicit connections (hidden wires, in blue). The variables and notations used hereinafter are shown for reference, with the top left and bottom right node triples detailed. (Here $C = 16$ and $R = 5$.)

Photographs of sample, actual designs of breadboards are shown in Fig. 2: except on the minimally sized breadboards, the gap separating the two halves is clearly visible.

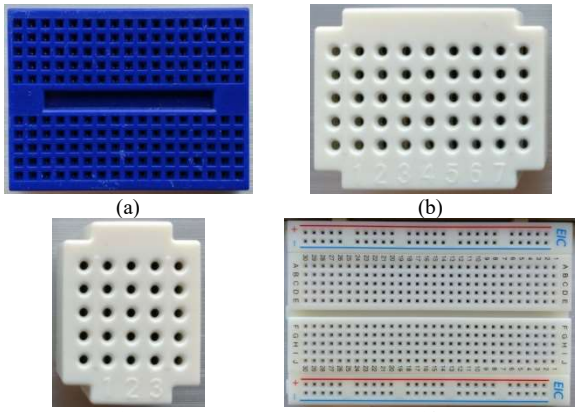

Fig. 2. Photographs of sample, actual designs of breadboards: except on the minimally sized breadboards ((b) and (c)), the gap separating the two halves is clearly visible ((a) and (d)).

The length of flat wires corresponds to a fixed and exact number of breadboard holes: flat wires are thus in most cases used (laid) either vertically or horizontally with respect to the breadboard. The unit of distance used in this work is therefore expressed in breadboard holes.

Flat wires are in most cases designed so that their length corresponds to an exact number of breadboard holes. Hence, we assume that a flat wire is either vertical or horizontal with respect to the breadboard matrix (i.e. parallel either to a matrix row or column). Furthermore, a wire, both flat and jumper, is represented by the pair of the two distinct nodes it connects.

Definition 1: A wire $\{u, v\}$ (it is recalled that $u \neq v$) passes over a node w if and only if, assuming without loss of generality that $c(u) \leq c(v)$ and $r(u) \leq r(v)$, one of the following mutually exclusive logic propositions holds:

- $h(u) = h(v) \wedge r(u) = r(v) \wedge h(w) = h(u) \wedge r(w) = r(u) \wedge c(u) \leq c(w) \leq c(v)$.
- $h(u) = h(v) \wedge c(u) = c(v) \wedge h(w) = h(u) \wedge c(w) = c(u) \wedge r(u) \leq r(w) \leq r(v)$.
- $h(u) \neq h(v) \wedge c(w) = c(u) \wedge \big((h(w) = 0 \wedge r(u) \leq r(w)) \vee (h(w) = 1 \wedge r(w) \leq r(v))\big)$.

Finally, we say that two wires cross each other if and only if there exists a node that is passed over (cf. Definition 1) by both wires.

IV. Methodology

A. Main Idea of the Algorithm

In addition to the breadboard number of rows per half and number of columns, the nodes to be connected, given as node pairs, are part of the algorithm input data. Although it remains optional since irrelevant for wiring, the sets of nodes that make electronic components can be considered as well, notably to visualise the breadboard wiring result.

The existence of an optimal solution, that is the connection of all the node pairs with no jumper wire, only flat wires, depends on the algorithm input: beside the pairs themselves (i.e. what to connect), the size of the breadboard (i.e. R and C) and the number of node pairs are critical. Because the proposed algorithm tentatively connects node pairs one by one, we maximise the probability of finding an optimal solution by considering the node pairs as one sequence of pairs and trying all the permutations of that sequence. This is reasonable with respect to time complexity as first a breadboard typically consists of $R = 5$ rows per half and at most $C = 63$ columns [11] and second, the number of

electronic components plugged, and thus of node pairs, remains low due to the breadboard design: a component cannot be setup with several pins in one same column (unless those pins can or have to be interconnected, which would be unusual to say the least).

Connection patterns of four flat wires or more to connect one node pair are not realistic: the aggregated wire length would likely surpass that of a jumper wire and the breadboard design itself would rapidly be impractical: the more the flat wires to connect one single pair, the fewer the remaining nodes (holes) to connect other pairs, and thus the lower the probability to connect other pairs with flat wires. The solutions produced by the proposed algorithm thus limit the connection of a node pair to at most three flat wires. Besides, for the solutions obtained by the proposed algorithm, node pairs that fail to be linked with flat wires under the aforementioned conditions default to a jumper wire connection.

Two main cases are distinguished: whether the pair to connect has its nodes in one same breadboard half or not.

B. *Case 1: The Nodes of the Pair are in the Same Half*

Let $\{u, v\}$ be the node pair to connect. It is recalled that in this case $h(u) = h(v)$. If, as a particular case as explained later in Section IV-D, $u = v$ holds, there is nothing to do to realise node connection. Otherwise, that is $u \neq v$ holds, we proceed as follows.

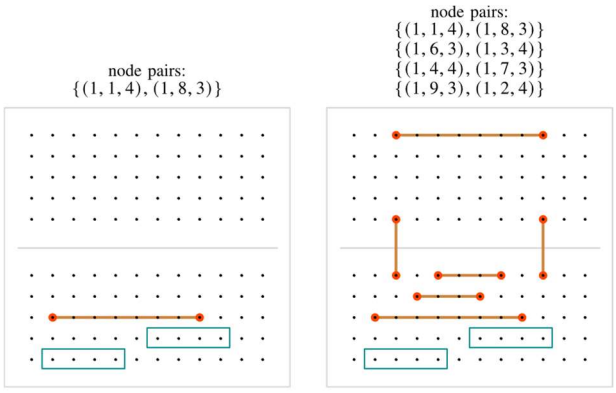

Fig. 3. Case 1: the nodes of the pair are in the same half. (a) Direct connection in the node half. (b) A configuration where indirect connection in the opposite half is necessary.

We first try to connect u and v "directly" with one single horizontal wire in $h(u)$ $(= h(v))$ the half of nodes u and v. In other words, we try to find a row r $(0 \leq r < R)$ in half $h(u)$ $(= h(v))$ such that a flat wire starting and ending at columns $c(u)$ and $c(v)$, precisely connecting nodes $(h(u), c(u), r)$ and $(h(u), c(v), r)$, neither crosses a flat wire nor passes over a node of an electronic component. If such a row r exists, the connection of u and v is finished. An illustration of this subcase is given in Fig. 3a.

Otherwise, we try to complete the connection of u and v "indirectly", that is in the opposite half of the breadboard, as follows. First, check if we can reach h' the opposite half (i.e. $h' \equiv h(u) + 1 \bmod 2$) from both u and v. In practice, this is about checking for node u (resp. v) whether column $c(u)$ (resp. $c(v)$) of h' is available, that is, whether that column includes a node of an electronic component, and whether the two nodes $(0, c(u), R-1)$ and $(1, c(u), 0)$ (resp. $(0, c(v), R-1)$ and $(1, c(v), 0)$) are not passed over by a flat wire and are not included in an electronic component. If one of these two conditions is not satisfied, then connection fails. Otherwise, we need to check one last condition: whether there exists a row r in half h' such that a flat wire starting and ending at columns $c(u)$ and $c(v)$, precisely connecting nodes $(h', c(u), r)$ and $(h', c(v), r)$, neither crosses a flat wire nor passes over a node of an electronic component. If such a row r exists, the connection of u and v is finished, and fails otherwise. An illustration of this subcase is given in Fig. 3b.

C. *Case 2: The Nodes of the Pair are in Different Halves*

Let $\{u, v\}$ be the node pair to connect. It is recalled that in this case $h(u) \neq h(v)$. We first try to reach $h(v)$ the half opposite of node u from u. Concretely, this is about checking for node u whether column $c(u)$ of h' is available (cf. Section IV-B) and whether the two nodes $(0, c(u), R-1)$ and $(1, c(u), 0)$ are not passed over by a flat wire and are not included in an electronic component. If node u can so be connected to the opposite half h', we next check for the existence of a row r in half h' such that a flat wire starting and ending at columns $c(u)$ and $c(v)$, precisely connecting nodes $(h', c(u), r)$ and $(h', c(v), r)$, neither crosses a flat wire nor passes over a node of an electronic component. If such a row r exists, the connection of u and v is finished.

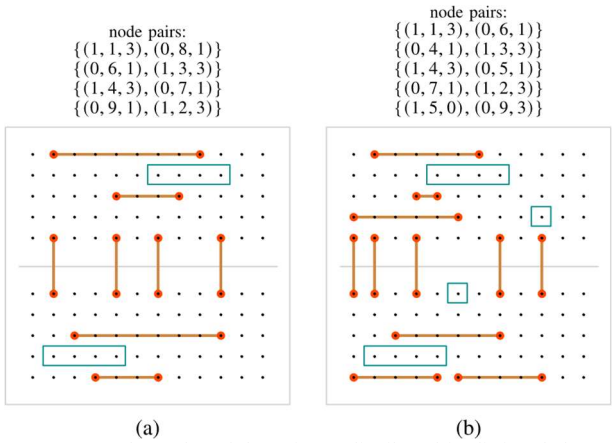

Fig. 4. Case 2: the nodes of the pair are distributed in distinct halves. (a) Connection in two flat wires. (b) A configuration where, amongst connections in two flat wires, connection in three flat wires is necessary.

Otherwise, we try the opposite: reaching $h(u)$ the half opposite of node v from v. This can be done as just described in the previous paragraph after exchanging the roles of nodes u and v, so for the sake of conciseness details are omitted here. An illustration of these two symmetric subcases is given in Fig. 4a.

When both of these two approaches, which each involve two flat wires, fail, we try another connection pattern that involves three flat wires as follows. Find a column c $(0 \leq c < C)$ by iteration such that the following conditions are all satisfied:

- there exists a row r_u in half $h(u)$ such that a flat wire starting and ending at columns $c(u)$ and c, precisely connecting nodes $(h(u), c(u), r_u)$ and $(h(u), c, r_u)$, neither crosses a flat wire nor passes over a node of an electronic component;
- we can reach half $h(v)$ from column c of half $h(u)$. Concretely, this is about checking for node u whether column c of $h(v)$ is available (cf. Section IV-B) and

whether the two nodes $(0, c, R-1)$ and $(1, c, 0)$ are not passed over by a flat wire and are not included in an electronic component;

- there exists a row r_v in half $h(v)$ such that a flat wire starting and ending at columns c and $c(v)$, precisely connecting the two nodes $(h(v), c, r_v)$ and $(h(v), c(v), r_v)$, neither crosses a flat wire nor passes over a node of an electronic component.

If these three conditions are all satisfied, then nodes u and v can be connected with a flat wire between $(h(u), c(u), r_u)$ and $(h(u), c, r_u)$, one between $(0, c, R-1)$ and $(1, c, 0)$ and a last one between $(h(v), c, r_v)$ and $(h(v), c(v), r_v)$. An illustration is given in Fig. 4b.

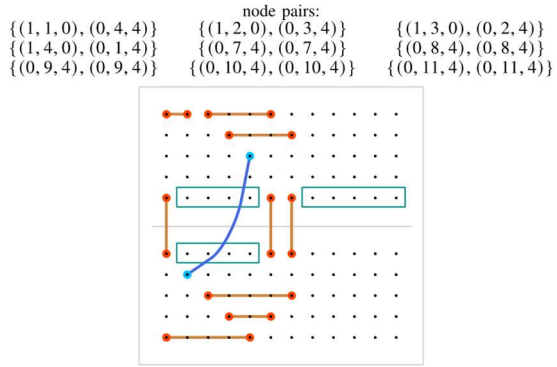

Fig. 5. A sample configuration matching Case 2 and whose best solution involves one jumper wire (curved line, in blue).

In both Case 1 and Case 2, the algorithm is completed by connecting the node pairs for which no flat wire could be set. This is simply done by setting a jumper wire connecting the two columns corresponding to the pair nodes. If no available row remains in those two columns, that is a row that is neither passed over by a flat wire nor includes another jumper wire end node, connection fails (i.e. both flat wire and jumper wire could not be set). An illustration is given in Fig. 5.

D. Additional Algorithmic Details

When trying to find an available row to connect two columns as done several times in both cases above (e.g. rows r, r_u and r_v), it is important to prioritise rows that are farthest from the breadboard gap. On the contrary, if the row that is closest to the gap is used, it would block all gap crossings for the columns passed over by that flat wire, and thus lower the probability of finding an optimal wiring solution.

Similarly, when crossing the gap at a particular column, the two nodes (in that column) of the two rows on each side of the gap are connected with a flat wire: using rows that are farther from the gap for crossing it would block additional rows and again reduce the probability of obtaining an optimal wiring solution.

Finally, in the case there are electronic components that have pins that are not to be connected to anything, every such pin, say node u, is paired with itself, that is $\{u, u\}$, so that it is not passed over by a flat wire. (Hence, it can be noted from this assumption that a node pair is not necessarily made of two distinct nodes, and therefore the pair notation $\{,\}$ denotes a multiset rather than a set.) And in such a configuration, the number of node pair permutations can be reduced by removing the node pairs of the form $\{u, u\}$ beforehand as such node pairs obviously do not impact wiring.

E. Pseudo-code and Time Complexity Analysis

Simplified pseudo-code for the described algorithm is given in Algorithm 1 in a recursive form (the time complexity of subroutines is given in comments). The worst-case time complexity $T(|P|)$ of the Connect (P) function can be estimated as follows:

$$T(k) = \begin{cases} O(R\,|P|^2) & \text{if } k = 0 \\ T(k-1) + O(R\,k) & \text{if } k \geq 1\,. \end{cases}$$

Since this function is called for each permutation of P as previously described, we obtain a total worst-case time complexity of $O(|P|!\ R\ |P|^2)$ for the proposed algorithm. This high worst-case time complexity is nonetheless mitigated by the small values of R and $|P|$ as explained in Section IV-A, and by reduction of the number of node pair permutations as explained in Section IV-D. It is thus important to also evaluate the proposal empirically: this is the purpose of the following Sections V and VI.

V. A Real-world Example

An additional, real-world breadboard example is shown in this section: an Arduino Nano board controlling an external EEPROM memory with the I²C protocol.

The configuration of the electronic components is as follows: an Arduino Nano board is plugged into the breadboard together with an external EEPROM memory (Microchip Technology 24FC1025 PDIP)–a PDIP integrated circuit with eight pins distributed on two opposing sides. The Arduino Nano board has fifteen pins distributed on two opposing sides. Both components are plugged into the breadboard across its gap.

The external EEPROM memory is controlled with the I²C protocol, which relies on two lines: SDA the data line and SCL the clock line. The Arduino Nano board provides the I²C lines at its A4 and A5 pins. The other six pins of the external EEPROM memory are connected either to the earth or to a positive tension (+5 V). Earth and positive tension are provided directly by the Arduino Nano board (two earth pins, one on each side of the board, and one positive tension (+5 V) pin are available). Finally, the Arduino Nano board itself is powered externally by a Mini-B USB cable. So, in total, there are eight node pairs defined, with pairs involving earth being initially defined on the same side of the two electronic components (e.g. the earth pin at the top of the Arduino Nano board is paired with the earth pin at the top of the external EEPROM memory).

The obtained wiring result is shown in Fig. 6: the best solution found involves three jumper wires, with all the node pairs successfully connected. (On the Arduino Nano board, the earth pins are labelled GND and the positive tension pin VCC.)

```
Function Connect (P):
  Input: A list P of node pairs.
  Output: Flat wires F, jumper wires J and unconnected node pairs U.
  if P = ∅ then                                          // Flat wires completed: find holes for jumper wires.
    foreach (p : {u, v}) ∈ X do                          // for each pair not yet connected
      r_u ← find-row(h(u), c(u)); r_v ← find-row(h(v), c(v));   // O(R|P|)   // available rows
      if r_u ∧ r_v then J ← J ∪ {p};                     // one new jumper wire
      else U ← U ∪ {p};                                  // new unconnected pair
  else                                                   // Otherwise, continue wiring with flat wires.
    (p : {u, v}) : P' ← P;                               // get the current node pair
    if u = v then Connect(P');
    else if h(u) = h(v) then                             // u and v located in the same half
      r ← find-crossing-row(h(u), c(u), c(v));           // O(R|P|)   // available row between u, v
      if r then F ← F ∪ {p}; Connect(P');                // one new flat wire (horizontal)
      else
        δ_u : (d_u, a_u) ← find-departure-arrival(u);    // O(|P|)   // two nodes on both...
        δ_v : (d_v, a_v) ← find-departure-arrival(v);    // O(|P|)   // ... gap sides for u, v
        if δ_u ∧ δ_v then
          r ← find-crossing-row(h̄(u), c(u), c(v));       // O(R|P|)
          if r then                                      // three new flat wires
            F ← F ∪ {{d_u, d_v}, {a_u, a_v}, {(h̄(u), c(u), r), (h̄(v), c(v), r)}};
            Connect(P');
          else X ← X ∪ {p}; Connect(P');                 // connection failed
        else X ← X ∪ {p}; Connect(P');                   // connection failed
      else                                               // u and v located in distinct halves
        δ_u : (d_u, a_u) ← find-departure-arrival(u);     // O(|P|)
        if δ_u then
          r ← find-crossing-row(h(v), c(u), c(v));       // O(R|P|)
          if r then F ← F ∪ {{d_u, a_u}, {(h(v), c(u), r), (h(v), c(v), r)}};   // two new flat wires
        if not r then
          δ_v : (d_v, a_v) ← find-departure-arrival(v);   // O(|P|)
          if δ_v then
            r ← find-crossing-row(h(u), c(u), c(v));     // O(R|P|)
            if r then F ← F ∪ {{d_v, a_v}, {(h(u), c(v), r), (h(u), c(u), r)}};   // two new flat wires
        if not r then
          foreach column c do                            // for each column of the breadboard
            r_u ← find-crossing-row(h(u), c(u), c);      // O(R|P|)
            if r_u then
              δ_c : (d_c, a_c) ← find-departure-arrival((h(u), c, r_u));   // O(|P|)
              if δ_c then
                r_v ← find-crossing-row(h(v), c, c(v));  // O(R|P|)
                if r_v then                              // three new flat wires
                  F' ← {{(h(u), c(u), r_u), (h(u), c, r_u)}, {d_c, a_c}, {(h(v), c, r_v), (h(v), c(v), r_v)}};
                  break;
          if F' then F ← F ∪ F' else X ← X ∪ {p};
        Connect(P');
```

Algorithm 1. Simplified pseudo-code of the proposed wiring algorithm. The time complexity of subroutines is given in comments.

node pairs:

{(1, 7, 0), (0, 21, 4)}	{(1, 8, 0), (0, 20, 4)}	{(0, 11, 0), (0, 19, 4)}	{(1, 13, 0), (1, 18, 0)}	{(1, 13, 0), (1, 19, 0)}	{(1, 13, 0), (1, 21, 0)}
{(1, 11, 0), (0, 18, 4)}	{(1, 11, 0), (1, 20, 0)}	{(0, 0, 4), (0, 0, 4)}	{(0, 1, 4), (0, 1, 4)}	{(0, 2, 4), (0, 2, 4)}	{(0, 3, 4), (0, 3, 4)}
{(0, 4, 4), (0, 4, 4)}	{(0, 5, 4), (0, 5, 4)}	{(0, 6, 4), (0, 6, 4)}	{(0, 7, 4), (0, 7, 4)}	{(0, 8, 4), (0, 8, 4)}	{(0, 9, 4), (0, 9, 4)}
{(0, 10, 4), (0, 10, 4)}	{(0, 12, 4), (0, 12, 4)}	{(0, 13, 4), (0, 13, 4)}	{(0, 14, 4), (0, 14, 4)}	{(1, 0, 0), (1, 0, 0)}	{(1, 1, 0), (1, 1, 0)}
{(1, 2, 0), (1, 2, 0)}	{(1, 3, 0), (1, 3, 0)}	{(1, 4, 0), (1, 4, 0)}	{(1, 5, 0), (1, 5, 0)}	{(1, 6, 0), (1, 6, 0)}	{(1, 9, 0), (1, 9, 0)}
{(1, 10, 0), (1, 10, 0)}	{(1, 12, 0), (1, 12, 0)}	{(1, 14, 0), (1, 14, 0)}			

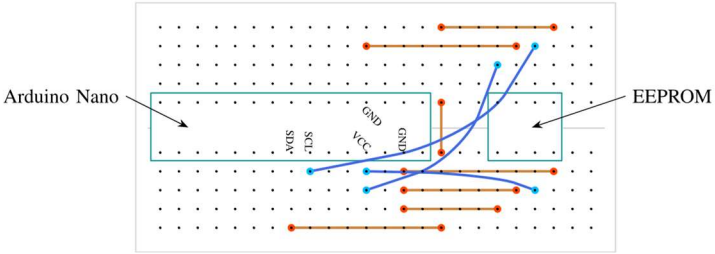

Fig. 6. The obtained wiring result in the real-world scenario of an Arduino Nano board controlling an external EEPROM memory with the I²C protocol.

VI. Experimental Evaluation

We have conducted an experimental evaluation of the proposed system in order to quantitatively measure the time required to produce a solution under various conditions and to compare the quality of the results. To this end, we have reused the electronic component configuration of Section V.

Besides, the proposed algorithm has been implemented with the Racket programming language and framework; the previously shown wiring results (and their respective graphical renderings) have been directly generated from this implementation.

First, we have applied the proposed algorithm in a single pass fashion: the node pairs are taken in the order set at call time and the obtained result is output. No other node pair sequence is tried. As a result, the solution involving 9 flat wires, 2 jumper wires and 1 unconnected node pair shown in Fig. 7 was obtained. The execution time of the proposed algorithm in this experiment, not including graphical rendering, whose duration is however negligible, was 3 ms

(real time), which is to be compared with the execution times obtained in the next experiments. Besides, regarding the generated solution, it can be noted that in this case, the unconnected node pair concerns connection of the external EEPROM memory to the earth.

node pairs:

{(1, 7, 0), (0, 21, 4)}	{(1, 8, 0), (0, 20, 4)}	{(0, 11, 0), (0, 19, 4)}
{(1, 13, 0), (1, 18, 0)}	{(1, 13, 0), (1, 19, 0)}	{(1, 13, 0), (1, 21, 0)}
{(1, 11, 0), (0, 18, 4)}	{(1, 11, 0), (1, 20, 0)}	{(0, 0, 4), (0, 0, 4)}
{(0, 1, 4), (0, 1, 4)}	{(0, 2, 4), (0, 2, 4)}	{(0, 3, 4), (0, 3, 4)}
{(0, 4, 4), (0, 4, 4)}	{(0, 5, 4), (0, 5, 4)}	{(0, 6, 4), (0, 6, 4)}
{(0, 7, 4), (0, 7, 4)}	{(0, 8, 4), (0, 8, 4)}	{(0, 9, 4), (0, 9, 4)}
{(0, 10, 4), (0, 10, 4)}	{(0, 12, 4), (0, 12, 4)}	{(0, 13, 4), (0, 13, 4)}
{(0, 14, 4), (0, 14, 4)}	{(1, 0, 0), (1, 0, 0)}	{(1, 1, 0), (1, 1, 0)}
{(1, 2, 0), (1, 2, 0)}	{(1, 3, 0), (1, 3, 0)}	{(1, 4, 0), (1, 4, 0)}
{(1, 5, 0), (1, 5, 0)}	{(1, 6, 0), (1, 6, 0)}	{(1, 9, 0), (1, 9, 0)}
{(1, 10, 0), (1, 10, 0)}	{(1, 12, 0), (1, 12, 0)}	{(1, 14, 0), (1, 14, 0)}

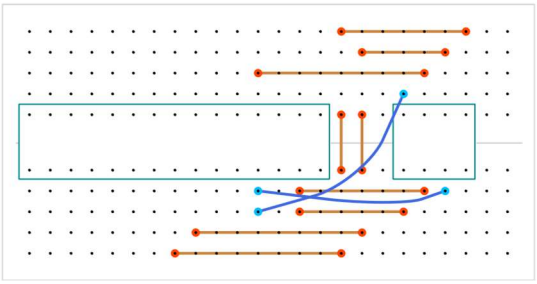

Fig. 7. Result output after one single pass of the proposed algorithm: 9 flat wires, 2 jumper wires and 1 unconnected node pair.

Second, we have run the implementation so that the first solution with zero unconnected node pairs and zero jumper wire found by the algorithm is returned. The result is that of Fig. 6: 7 flat wires, 3 jumper wires and 0 unconnected pairs.

In this third experiment, the optimal solution of possibly several is retained. A solution is treated as better than another when 1) it has fewer unconnected pairs than the other, or 2) it has a smaller number of jumper wires, or 3) it has a smaller number of flat wires. These three conditions are checked in this precise order: for instance, the number of unconnected pairs has priority over the other two criteria when comparing two solutions. Concretely, in an exhaustive manner, to every permutation of the node pairs is applied the algorithm implementation. The result is once again that of Fig. 6: 7 flat wires, 3 jumper wires and 0 unconnected pairs. In other words, no better solution was found when exhaustively trying all the node pair permutations compared with the previous non-exhaustive experiment.

Then, we have measured the execution time of the algorithm (again, not including graphical rendering, whose duration is however negligible), with two variants as follows: 1) every single node pair of the configuration is considered when generating the node pair permutations, and 2) the $\{u, u\}$ node pairs (cf. Section IV-D), that is node pairs connecting a node with itself, are excluded from the permutation generation. Variant 2 thus corresponds to an optimised, faster version. This time, the electronic component configuration used is that of Fig. 5. The measured execution times–real time, in milliseconds–are shown in Table 1.

Table 1. The measured execution times of the proposed wiring algorithm under various experimental conditions (real time, in milliseconds).

Solution search method	Variant 1 (ms) (i.e. non optimised)	Variant 2 (ms) (i.e. optimised)
non-exhaustive	149109	12
exhaustive	150210	13

The relatively small time difference between the non-exhaustive and exhaustive implementation is most likely induced by the particular electronic component configuration used: the solution found in the non-exhaustive search is induced by a permutation that is generated towards the end of the permutation generation process.

As for the optimised version, excluding the $\{u, u\}$ node pairs from the permutation generation task has an obvious impact on the algorithm execution time: in this particular electronic component configuration, there are five such $\{u, u\}$ pairs and four "regular" node pairs. Thus, the optimised approach reduces the number of permutations from $9! = 362880$ to $4! = 24$, hence the significantly reduced execution times.

For reference, in the electronic component configuration described in Section V, Variant 2, with thus $8! = 40320$ node pair permutations, induces an execution time of 74113 ms in the case of a non-exhaustive solution search, and 73869 ms in the case of an exhaustive solution search: there is no significant time difference between the exhaustive and non-exhaustive search method in this situation.

VII. Conclusions

Allowing for solder-less prototyping, breadboards are a very common sighting in IoT device design. Nevertheless, the breadboard electric connection network topology imposes restrictions on circuit design. Those restrictions make the prototyping of circuits a long, tedious process, that in addition often has to be started over in the case of an even minor change to the circuit. In this paper, aiming to tackle this issue, we have described and evaluated an algorithm that produces possible wire connections to implement a given design. To these solutions is assigned a score, used to select the optimal circuit implementation solution.

Although the proposed algorithm has to take into account many details, its approach is rather simple. It is thus all the more interesting to note that the obtained solutions are sometimes smarter than, or at least different from what a human could manually design. The experimental results show that the proposed optimisation significantly reduces the execution time of the algorithm for both the exhaustive and non-exhaustive solution search methods.

Regarding future developments of this research, as explained, the length of wires is often fixed: all wire lengths are not always available. So, we could further refine this work by adding as supplementary parameter the available wire lengths. In addition, although we have considered connection of node pairs independently of others, in some cases several different pins of an electronic component provide the same function (e.g. earth, typically), which could thus allow several node pair candidates, like $\{u, v\}$, $\{u, w\}$ with v and w being different pins but both earth pins, to realise the needed connection. Enabling such pair candidates for greater connection flexibility is thus one possible improvement to the proposal. Finally, as briefly explained, some breadboards have "side" lines, on the outside of the two halves, typically for earth and tension; such side lines have not been taken into consideration here. This is yet another possible future work.

Conflict of Interest

The author declares no conflict of interest.

ACKNOWLEDGEMENTS

The author is sincerely thankful towards the reviewers for their insightful comments and suggestions.

REFERENCES

[1] M. Przybylla and R. Romeike, "Physical computing and its scope–Towards a constructionist computer science curriculum with physical computing," *Informatics in Education*, vol. 13, no. 2, pp. 225–240, Oct. 2014.

[2] J. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.

[3] A. López-Vargas, M. Fuentes, and M. Vivar, "IoT application for real-time monitoring of solar home systems based on Arduino™ with 3G connectivity," *IEEE Sensors Journal*, vol. 19, no. 2, pp. 679–691, Oct. 2019.

[4] A. Bossard, "An optimised PIC programmer also having a high pedagogical value," in *Proc. 4th International Conference on Computer and Communication Engineering (CCCE; Oslo, Norway, 24–26 May)*, 2024, pp. 158–170.

[5] L. L. Scarlatos, A. R. Pratama, and T. T. Tchoubar, "The virtual breadboard: Helping students to learn electrical engineering at a distance," in *Proc. Future Technologies Conference (FTC; Vancouver, Canada, 29–30 November)*, 2017, pp. 610–617.

[6] J.-Y. Lo, D.-Y. Huang, T.-S. Kuo, C.-K. Sun, J. Gong, T. Seyed, X.-D. Yang, and B.-Y. Chen, "AutoFritz: Autocomplete for prototyping virtual breadboard circuits," in *Proc. Conference on Human Factors in Computing Systems (CHI; Glasgow, UK, 4–9 May)*, 2019, pp. 1–13.

[7] A. Bossard, K. Kaneko, and F. C. Harris Jr, "On the crossing number of a torus network," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E106.A, no. 1, pp. 35–44, Jan. 2023.

[8] F. T. Leighton, "New lower bound techniques for VLSI," *Mathematical Systems Theory*, vol. 17, no. 1, pp. 47–70, Dec. 1984.

[9] O. Sýkora and I. Vrt'o, "On VLSI layouts of the star graph and related networks," *Integration, the VLSI Journal*, vol. 17, no. 1, pp. 83–93, Aug. 1994.

[10] M. R. Garey and D. S. Johnson, "Crossing number is NP-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 4, no. 3, pp. 312–316, Sep. 1983.

[11] Cixi Zhongyi Electronics, "Specification for approval – 830 points solderless breadboard," Aug. 2012, item no. ZYJ-102.