Also available at http://amc.imfm.si
ISSN 1855-3966 (printed edn.), ISSN 1855-3974 (electronic edn.)
ARS MATHEMATICA CONTEMPORANEA 2 (2009) 121–136

Deciding the deterministic property for soliton graphs

Miklós Bartha [*]
Memorial University of Newfoundland, St. John's, Canada

Miklós Krész [†]
University of Szeged, Szeged, Hungary

Received 1 October 2007, accepted 29 March 2009, published online 20 July 2009

Abstract

Soliton automata are a graph theoretic model for electronic switching at the molecular level. In the design of soliton circuits, the deterministic property of the corresponding automata is of primary importance. The underlying graphs of such automata, called deterministic soliton graphs, are characterized in terms of graphs not having even-length cycles and graphs having a unique perfect matching. On the basis of this characterization, a modification of the currently most efficient unique perfect matching algorithm is worked out to decide in $\mathcal{O}(m \log^4 n)$ time if a graph with n vertices and m edges defines a deterministic soliton automaton. A yet more efficient $\mathcal{O}(m)$ algorithm is given for the special case of chestnut and elementary soliton graphs. All of these algorithms are capable of constructing a state for the corresponding soliton automaton, and the general algorithm can also be used to simplify the automaton to an isomorphic elementary one.

Keywords: Graphs, matchings, soliton graphs, soliton automata.

Math. Subj. Class.: 05C70

[*] Work partially supported by Natural Science and Engineering Research Council of Canada, Discovery Grant #170493-03.

[†] Work partially supported by the Hungarian-Slovenian Intergovernmental Programme for 2008-2009 (project number *BI-HU/08-09-012*) and by the MÖB-DAAD Hungarian-German Researcher Exchange Program (project No. 3).

E-mail addresses: bartha@cs.mun.ca (Miklós Bartha), kresz@jgypk.u-szeged.hu (Miklós Krész)

Copyright © 2009 DMFA

1 Introduction

Soliton switching as a physical phenomenon has first been described in [7] through several examples of a soliton wave traveling along chains of alternating single and double bonds in hydrocarbon molecules, which typically contain a system of such chemical bonds. The corresponding mathematical model was introduced in [8] by the name soliton automaton.

Since the early paper [8], theoretical research on soliton automata has taken two separate routes. One approach tries to characterize the transition monoids of deterministic soliton automata directly as appropriate groups. Some of the most important results in this direction can be found in [8], [9]and [10]. The other approach, highlighted by the works [1], [2], [3], [4],[15], [17]and [18] analyzes the internal structure of the underlying graphs of soliton automata on the grounds of matching theory. The two approaches have the same final goal: determine the computational power of soliton automata. Other more practical aspects of soliton switching have been studied in [13].

The present paper follows the second route outlined above. Based on the characterization of deterministic soliton graphs obtained in [4], we provide an efficient algorithm to decide if an arbitrary graph G defines a deterministic soliton automaton. On a graph with n vertices and m edges, the algorithm runs in $\mathcal{O}(m \log^4 n)$ time. We also present a linear-time decision algorithm for two special classes of deterministic soliton graphs: chestnuts and elementary graphs. Our algorithms can be used in other applications as well. One of these applications aims at finding a flexible perfect matching for a graph equipped with a so-called grace set of vertices [5]. See also [11] and [12] for the same problem in graphs without the grace feature. Another application in computational biology concerns the description of an automated system that successfully predicts RNA structure [20].

2 Soliton graphs and automata

By a graph $G = (V(G), E(G))$, throughout the paper, we mean a finite undirected graph with multiple edges and loops allowed. Edges in $E(G)$ are denoted by pairs $(u, v) \in V(G) \times V(G)$ with no direction between u and v implied. Our notation and terminology will follow [19]. For a vertex $v \in V(G)$, $d(v)$ denotes the *degree* of v. Vertex v is called *external* if $d(v) = 1$ or $d(v) = 0$, and *internal* if $d(v) \geq 2$. External edges are those of $E(G)$ that are incident with at least one external vertex. All other edges are called internal. Graph G is called *open* if it has at least one external vertex and *closed* if it has none.

For a set of vertices $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by X, that is, the graph which has as vertices X, and as edges those of $E(G)$ that connect two vertices in X. If S is a subgraph of G, then $G[S] = G[V(S)]$. In this way, $[\,]$ becomes an idempotent operation on subgraphs of G. Since in our discussion external vertices play a distinguished role, we do not want to allow that subgraphs of G have external vertices other than those of G. Therefore, in every subgraph of G, we shall automatically add a “protective” loop around each vertex with degree one or zero that is not external in G.

A *walk* in graph G is an alternating sequence of vertices and edges, which starts and ends with a vertex, and in which each edge is incident with the vertex immediately preceding it and the vertex immediately following it. The *length* of a walk is the number of occurrences of edges in it. A *path* is a walk in which all vertices are distinct, and a *cycle* is a walk which can be decomposed into a path and an edge connecting the two endpoints of the path.

A *matching* M of graph G is a subset of $E(G)$ such that no vertex of G occurs more than once as an endpoint of some edge in M. It is understood by this definition that loops do not participate in any matching. The endpoints of the edges contained in M are said to be *covered* by M. A *perfect matching* is a matching that covers all vertices in $V(G)$, while a *perfect internal matching* is one that covers all of the internal vertices. An open graph having a perfect internal matching is called a *soliton graph*. To keep our discussion coherent, we shall deal with perfect matchings as perfect internal matchings of closed graphs. To this end it is sufficient to introduce a loop around each unwanted external vertex, thus turning these vertices internal. It is clear that no generality is lost by this simple maneuver.

Let G be a graph and M be a matching of G. An edge $e \in E(G)$ is said to be *M-positive* (*M-negative*) if $e \in M$ (respectively, $e \notin M$). An *M-alternating path (cycle)* in G is a path (respectively, even-length cycle) stepping on M-positive and M-negative edges in an alternating way. Let us agree that, if the matching M is understood or irrelevant in a particular context, then it will not be explicitly indicated in these terms. An alternating path is *positive* (*negative*) if it is such at its internal endpoint(s), meaning that the edges incident with those endpoints are positive (respectively, negative). An *external alternating path* is one that has an external endpoint. If both endpoints of the path are external, then it is called a *crossing*. An *alternating unit* is either a crossing or an alternating cycle.

Let M be a perfect internal matching of a soliton graph G. *Switching* on an M-alternating unit amounts to changing the sign of each edge along the unit with respect to M. It is easy to see that the operation of switching on an M-alternating unit α creates a new perfect internal matching for G, denoted $S(M,\alpha)$.

Now we generalize the M-alternating property for walks α starting from an external vertex. In parallel we also generalize the concept of switching on such an *external alternating walk* α, which process creates a set of edges $S(M,\alpha)$. The definition is as follows:

(i) The walk $\alpha = v_0 e v_1$, where $e = (v_0, v_1)$ for an external $v_0 \in V(G)$, is an external M-alternating walk and $S(M,\alpha) = M\Delta\{e\}$, where Δ denotes symmetric difference of sets.

(ii) If $\alpha = v_0 e_1 \ldots e_n v_n$ is an external M-alternating walk ending at an internal vertex v_n, and $e_{n+1} = (v_n, v_{n+1})$ is such that $e_{n+1} \in S(M,\alpha)$ iff $e_n \in S(M,\alpha)$, then $\alpha' = \alpha e_{n+1} v_{n+1}$ is an external M-alternating walk and $S(M,\alpha') = S(M,\alpha)\Delta\{e_{n+1}\}$. It is required, however, that $e_{n+1} \neq e_n$, unless $e_n \in S(M,\alpha)$ is a loop.

It follows from the above definition that $S(M,\alpha)$ is a perfect internal matching iff the endpoint v_n of α is external, too. In this case we say that α is a *soliton walk*. By *traversing* a soliton walk α we mean switching on α, that is, creating the perfect internal matching $S(M,\alpha)$ in an interactive way. Intuitively, a walk starting and ending at an external vertex can be traversed as a soliton walk if, dynamically, its edges follow an alternating pattern with respect to the given perfect internal matching M.

Example 2.1. Consider the graph G of Figure 1, and let $M = \{e, h_1, h_2\}$. A possible soliton walk from u to v with respect to M is $\alpha = uewgz_1h_1z_2l_2z_3h_2z_4l_1z_1gwfv$. Traversing α then results in $S(M,\alpha) = \{f, l_1, l_2\}$.

Every soliton graph G gives rise to a *soliton automaton* $\mathcal{A}_G$, the states of which are the perfect internal matchings of G. The input alphabet for $\mathcal{A}_G$ is the set of all (ordered) pairs of external vertices in G, and the state transition function δ is defined by

$$\delta(M,(v,w)) = \{S(M,\alpha) \mid \alpha \text{ is a soliton walk from } v \text{ to } w\}$$

if there exists a soliton walk from v to w, and

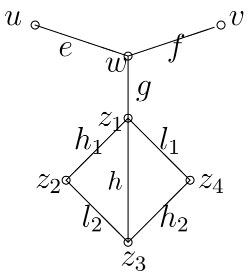

Figure 1: Example soliton graph G

$\delta(M, (v, w)) = \{M\}$ if there is none.

As a new feature compared to previous definitions of soliton automata, we extend $\mathcal{A}_G$ with a binary output defined by the function

$\zeta(M, (v, w)) = 1$ iff there exists a soliton walk from v to w.

The rationale for this extension is the following. We do not want the automaton $\mathcal{A}_G$ to crash upon receiving an input (v, w) for which there exists no soliton walk from v to w in a given state M. This is reflected by the above definition of δ. On the other hand, if $v = w$, then there might as well exist a soliton walk α from v to v, the traversal of which does not change the current state M. If all soliton walks from v to v are of this nature, then the only way to distinguish between having or not having a soliton walk is to introduce a binary output. The practical importance of this observation is clear: we can actually build a variety of interesting flip-flops using simple soliton automata.

Example 2.2. (Binary flip-flop) Consider the top graph G in Figure 2a with two external vertices. The automaton $\mathcal{A}_G$ has two states, which are shown below G in Figure 2a. (Double lines indicate positive edges in the figure.) Without the output feature $\mathcal{A}_G$ would be isomorphic to $\mathcal{A}_{G'}$, where G' is the graph appearing on the top of Figure 2b. In $\mathcal{A}_G$, the inputs $(1, 1)$ and $(2, 2)$ can be used to test the current state through the output provided by the automaton. The same test is not possible in $\mathcal{A}_{G'}$.

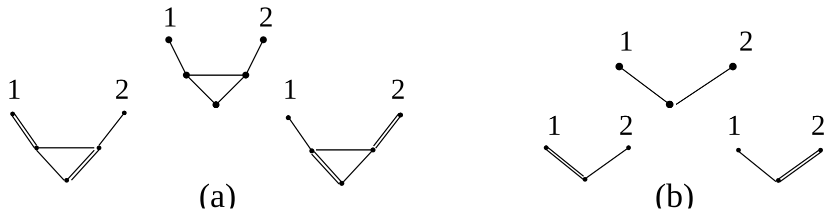

Figure 2: A binary flip-flop

Example 2.3. (Ternary flip-flop) The underlying graph G, and the four states of the soliton automaton $\mathcal{A}_G$ are shown in Figure 3. This automaton comes with a distinguished "initial" state, in which all three external vertices are covered by the corresponding matching. The initial state is recognized by getting a negative output on each of the inputs $(1, 1)$, $(2, 2)$, and $(3, 3)$. The transition function and monoid of $\mathcal{A}_G$ is described in [8].

Following the pattern of Examples 2.2 and 2.3 one can easily design flip-flops with an arbitrary number of states as soliton automata. In order for this idea to work in practice, one needs an interface sophisticated enough to pick up the output signals provided by soliton automata.

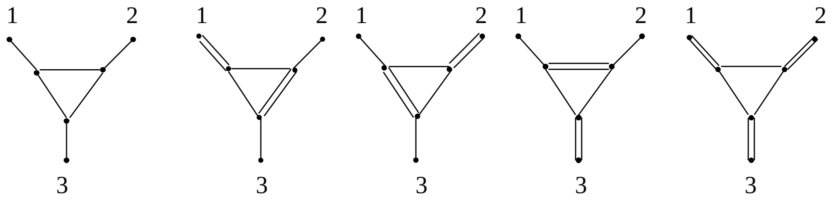

Figure 3: A ternary flip-flop

Notice that both the binary and ternary flip-flop graphs are 3-graphs in the sense that they do not have loops or multiple edges, and the degree of each vertex is at most 3. Soliton graphs and automata have been introduced originally for such graphs in [8]. Even though our generalization from 3-graphs to all undirected graphs regarding the underlying structure seems substantial, it turns out that this generalization is merely technical. Every soliton automaton is isomorphic to one having a 3-graph as its underlying graph [17].

A soliton graph G is called *deterministic* if $\mathcal{A}_G$ is such in the usual sense, that is, for every state M and input (v, w), $|\delta(M, (v, w))| = 1$.

The binary and ternary flip-flop graphs are both deterministic. The soliton graph of Figure 1 is, however, non-deterministic, as $\alpha' = uewfv$ is another soliton walk from u to v with respect to the state M of Example 2.1 such that $S(M, \alpha) \neq S(M, \alpha')$. In the sequel, for brevity, all perfect internal matchings of a soliton graph G will simply be called *states*.

Recall from [8] that an edge $e \in E(G)$ (vertex $v \in V(G)$) of a soliton graph G is *impervious* if there is no soliton walk passing through e (respectively, v) in any state of G. Edge e (vertex v) is *viable* if it is not impervious. Graph G is called *viable* if all of its vertices are such. See edge h in Figure 1 for an example of an impervious edge. The importance of viability in soliton graphs is self-explanatory: only the viable vertices and edges affect the behavior of the corresponding soliton automata. All impervious vertices and edges can be deleted without any impact on these automata.

Following the terminology introduced in [3], an internal vertex $v \in V(G)$ is called *accessible* in state M if there exists a positive external M-alternating path leading to v. It is easy to see, cf.[3], that v is accessible in some state M iff v is accessible in all states of G. As a consequence, the following simple characterization of impervious edges and vertices is obtained.

Proposition 2.4. [4] *For every edge $e \in E(G)$, e is viable iff at least one endpoint of e is accessible. Vertex $v \in V(G)$ is viable iff there exists an external alternating path (positive or negative) leading to v in any state of G, or, equivalently, if v is the endpoint of at least one viable edge in $E(G)$.*

3 Elementary decomposition of soliton graphs

In this section we review the main results obtained in [3] on the structure of soliton graphs. We begin with a short summary of the matching theoretic concepts involved. The reader can obtain a good understanding of these concepts by following the definitions to come on Figure 4 below.

Let us fix a graph G (open or closed) having a perfect internal matching for the entire section. An edge $e \in E(G)$ is called *allowed (mandatory)* if e is contained in some (respectively, all) state(s) of G. *Forbidden edges* are those that are not allowed. We shall also use the term *constant edge* to identify an edge that is either forbidden or mandatory. One of the earliest results on soliton automata [1] is the simple fact that, for every state M of G, every other state M' can be obtained from M by switching on a number of pairwise disjoint M-alternating units. By this statement it is obvious that an edge $e \in E(G)$ is not constant iff there exists an alternating unit passing through e in every state of G. We shall use this observation in the proof of Proposition 5.4 below.

By the standard definition [19], a closed graph G is *elementary* if its allowed edges form a connected subgraph. This definition is extended to open graphs by requiring that the connected subgraph spanned by the allowed edges contain all the external vertices, or G itself be an isolated external vertex. Observe that if G is elementary, then it cannot contain a mandatory edge, unless G is a mandatory edge by itself with a number of loops incident with one of its endpoints.

The concept of *canonical equivalence*[19] in elementary graphs is also well-known. To define this equivalence we choose the simplest wording here that is also meaningful for open graphs with perfect internal matchings. Consider the relation $\sim$ on the set of internal vertices of an elementary graph G for which $v_1 \sim v_2$ iff the pair $e = (v_1, v_2)$ becomes a forbidden edge in $G + e$. It is known, cf. [19], [3], that $\sim$ is an equivalence relation, which determines the so called *canonical partition* of (the internal vertices of) G. The reader is referred to [19] for more information on canonical equivalence.

In general, the subgraph of G determined by its allowed edges has several connected components, which are called the *elementary components* of G. Notice that an elementary component can be as small as a single external vertex of G. Such elementary components are called *degenerate*. Elementary components are classified as *external* or *internal*, depending on whether or not they contain an external vertex. A *mandatory* elementary component is a single mandatory edge $e \in E(G)$, which, as a subgraph of G, will have a loop around one or both of its endpoints. An elementary component C is *viable* if all vertices in C are such. By Proposition 2.4, a non-viable elementary component can only contain impervious vertices, and is therefore called impervious itself. In Figure 4, every elementary component, with the exception of C_7, is viable.

Recall from [19] that an *ear* of an arbitrary graph G relative to a subgraph G' is a path in G having both endpoints – but no interior vertices – in G', or a cycle in G having exactly one vertex in G'. In the former case the ear is called *open*, while in the latter it is *closed*. All ears considered in this paper will be of odd lenght. If M is a state of G, the restriction of which to $E(G')$ defines a state of G', too, then an *M-alternating G'-ear* α is an ear of G relative to G' which alternates on positive and negative edges with respect to M. Clearly, the edges on α incident with G' must then be negative. In the same vein, an *M-alternating negative G'-fork* is a pair of edge-disjoint negative external M-alternating paths having their two endpoints – but no other vertices – in G'.

An *ear decomposition* of G starting from a subgraph G' is a representation of G in the form $G = G' + \alpha_1 + \cdots + \alpha_k$, where α_1 is an ear of $G' + \alpha_1$ relative to G' and α_i is is an ear of $G' + \alpha_1 + \cdots + \alpha_i$ relative to $G' + \alpha_1 + \cdots + \alpha_{i-1}$ for $2 \le i \le k$. This decomposition is M-alternating if all ears α_i are such with respect to some fixed state M of G. The *connected components* of the decomposition are those of the subgraph $\alpha_1 + \cdots + \alpha_k$.

For each non-degenerate elementary component C of G, let C_h denote the elementary graph obtained from C by adding the following edges. A pair (u, v) is added to $E(C)$ if there exists an M-alternating C-ear or negative C-fork α with respect to some state M such that the internal endpoints of α are u and v. The newly added edges are called *hidden*, and graph G, enriched with all of its hidden edges, is denoted by G_h. See Figure 4 for a number

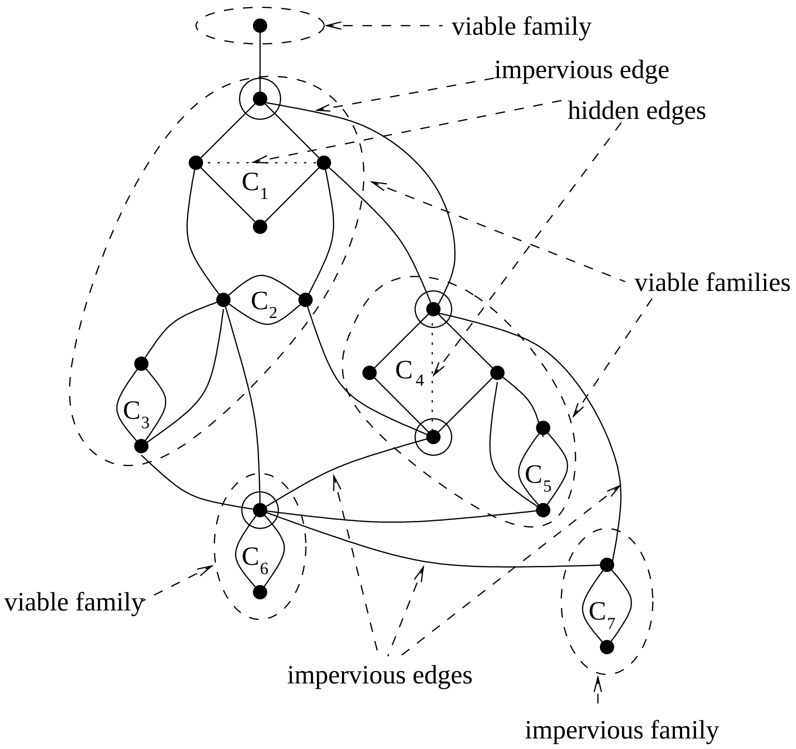

Figure 4: The structure of elementary components in a soliton graph

of hidden edges. Observe that each hidden edge must connect two vertices belonging to the same canonical class of its elementary component, even when this component has already been endowed with all hidden edges. As a consequence [3], the elementary decomposition of G_h, with components C_h, is the same as that of G. Moreover, the soliton automata $\mathcal{A}_G$ and $\mathcal{A}_{G_h}$ are isomorphic [17].

A viable internal elementary component C of G is called *one-way* if all external alternating paths (with respect to any state M) enter C at vertices belonging to the same canonical class of C_h. This unique class, as well as the vertices in this class, are called *principal*. Also as part of this definition, every external elementary component is considered a priori one-way (with no principal canonical class, of course). A viable elementary component is *two-way* if it is not one-way. In Figure 4, elementary components C_1, C_4, and C_6 are one-way internal, with their principal vertices encircled.

Let C and C' be two distinct elementary components of G. We say that C' is *two-way accessible* from C with respect to any (or all) state(s) M, in notation $C\rho C'$, if there exists an M-alternating C-ear passing through C'. It is required, though, that if C is one-way and internal, then the endpoints of this ear are not in the principal canonical class of C_h. As it was shown in [3], the two-way accessible relationship ρ is antisymmetric and state invariant. In Figure 4, C_2 is two-way accessible from C_1, C_3 from C_2, and C_5 from C_4. (But C_3 is not two-way accessible from C_1, and C_2, C_3, C_4, C_5 are not two-way accessible from C_6, even though there exists a negative closed ear originating from the principal vertex of C_6 that covers all four of these components.)

A *family* of elementary components in G is a block of the partition induced by the smallest equivalence relation containing ρ. A family $\mathcal{F}$ is called external if it contains an external elementary component, otherwise $\mathcal{F}$ is internal. Family $\mathcal{F}$ is *viable* if every elementary component in $\mathcal{F}$ is such, and impervious if this is not the case. A *degenerate* family consists of a single isolated external vertex (elementary component), and G is degenerate if its single family is such. The graph of Figure 4 has five families, four of which are viable. The only external family is a stand-alone degenerate external elementary component.

The first group of results obtained in [3] on the structure of elementary components of G can now be stated as follows.

Theorem 3.1. *Each viable family $\mathcal{F}$ of G contains a unique one-way elementary component, called the root of the family. Each vertex in every member of the family $\mathcal{F}$, except for the principal vertices of the root, is accessible. The principal vertices themselves are inaccessible, but all other vertices can only be reached through them by external alternating paths.*

Corollary 3.2. *Let $e \in E(G)$ be incident with at least one viable vertex. Then e is impervious iff the viable endpoints of e are principal.*

Along the lines of Theorem 3.1, a *mandatory* family is a viable family having a mandatory edge as its root. For two distinct viable families $\mathcal{F}_1$ and $\mathcal{F}_2$ of G, $\mathcal{F}_2$ is said to *follow* $\mathcal{F}_1$, in notation $\mathcal{F}_1 \mapsto \mathcal{F}_2$, if there exists an edge in G connecting a non-principal vertex in $\mathcal{F}_1$ with a principal vertex of the root of $\mathcal{F}_2$. The reflexive and transitive closure of $\mapsto$ is denoted by $\overset{*}{\mapsto}$. With a slight ambiguity, family $\mathcal{F}$ will be identified with the subgraph of G induced by the vertices belonging to $\cup\mathcal{F}$. The second group of results found in [3] can now be summarized as follows.

Theorem 3.3. *The following four statements hold for the families of G.*

1. *Every viable family $\mathcal{F}$ has an M-alternating ear decomposition relative to its root R with respect to any state M. The connected components of this decomposition are state invariant, and for each such component P, the vertices common to P and R belong to the same non-principal canonical class of R_h.*
2. *For every edge e connecting a viable family $\mathcal{F}_1$ to any other family $\mathcal{F}_2$ (viable or not), at least one endpoint of e is principal in $\mathcal{F}_1$ or $\mathcal{F}_2$. If the endpoint of e in $\mathcal{F}_1$ is not principal, then $\mathcal{F}_2$ is viable and it follows $\mathcal{F}_1$.*
3. *The relation $\overset{*}{\mapsto}$ is a partial order between viable families, by which the external families are minimal elements.*

See again Figure 4, and identify the partial order $\overset{*}{\mapsto}$ of families. For convenience, the inverse of the partial order $\overset{*}{\mapsto}$ between G's families will be denoted by $\leq_G$, or simply $\leq$ if G is understood. Referring to the Hasse diagram of $\leq$, we say that family $\mathcal{G}$ is *below* family $\mathcal{F}$ if $\mathcal{G} \leq \mathcal{F}$, that is, $\mathcal{G}$ follows $\mathcal{F}$.

4 Deterministic soliton automata

Deterministic soliton automata play a distinguished role in the design of soliton circuits. It is therefore crucial that we have a simple characterization of such automata, which also allows efficient algorithms to decide the deterministic property. Partial results towards this goal have been obtained in [4] and [18]. In this section we present a complete characterization of soliton automata, and provide a linear-time decision algorithm for two kinds of deterministic soliton automata: chestnuts and elementary graphs.

Definition 4.1. [8] A connected graph G is a *chestnut* if it has a representation in the form $G = \gamma + \alpha_1 + \dots + \alpha_k$ with $k \geq 1$, where γ is a cycle of even length and each α_i $(1 \leq i \leq k)$ is a tree subject to the following conditions:

(i) $V(\alpha_i) \cap V(\alpha_j) = \emptyset$ for $1 \leq i \neq j \leq k$;
(ii) $V(\alpha_i) \cap V(\gamma)$ consists of a unique vertex – denoted by v_i – for each $1 \leq i \leq k$;
(iii) v_i and v_j are at even distance on γ for any distinct $1 \leq i, j \leq k$;
(iv) any vertex $w_i \in V(\alpha_i)$ with $d(w_i) > 2$ is at even distance from v_i in α_i for each $1 \leq i \leq k$.

See Figure 5 for an example chestnut.

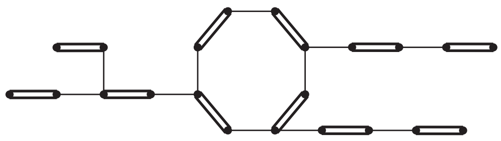
Figure 5: A chestnut

Clearly, every chestnut is a bipartite graph. A *baby chestnut* [15] is a special instance of a chestnut graph by which γ is a pair of parallel edges, and each α_i is a single edge or 2-length path incident with the same (principal) vertex of γ.

Theorem 4.2. ([4], see also [8]) *Let G be a connected deterministic soliton graph having no impervious edges. If G has a non-mandatory internal elementary component, then G is a chestnut.*

A *redex* r in an arbitrary graph G consists of two adjacent edges $e = (u, z)$ and $f = (z, v)$ such that $u \neq v$ are both internal and $d(z) = 2$. The vertex z is called the *center* of r, while u and v are the two *focal* vertices of r. A *secondary loop* in G is one, the removal of which does not introduce a vertex with degree 1.

Let r be a redex in G. *Shrinking* r means creating a new graph G_r from G by deleting the center of r and merging the two focal vertices of r into one "sink" vertex s. As it was shown in [4], the automata $\mathcal{A}_G$ and $\mathcal{A}_{G_r}$ are isomorphic for every soliton graph G. *Reducing* G entails shrinking all of its redexes in a recursive manner, and, at the same time, deleting all secondary loops emerging in this process. Reduction yields a graph $r(G)$, which is unique up to graph isomorphism. For an example, every chestnut graph reduces to a baby chestnut with no chance of deleting a secondary loop in this process, since chestnuts are bipartite.

In general, if G is a soliton graph, then the automata $\mathcal{A}_{r(G)}$ and $\mathcal{A}_G$ need not be isomorphic. The reason is that the deletion of a secondary loop may also eliminate some existing soliton walks in G. See again Figure 2, where the automata $\mathcal{A}_G$ and $\mathcal{A}_{G'}$ are not isomorphic. It is still true, however, that $\mathcal{A}_{r(G)}$ is deterministic iff $\mathcal{A}_G$ is such, unless $r(G)$ is a baby chestnut, in which case G need not be deterministic. See [15] for details. Reduction is well-defined for all graphs G, and, as proved in [6], if G has n vertices and m edges, then the graph $r(G)$ can be constructed in $\mathcal{O}(m)$ time. See also [15] for a simpler $\mathcal{O}(n^2)$ algorithm. The following characterization of viable deterministic soliton graphs was obtained in [4]. Using the terminology introduced there, a *generalized tree* is a connected graph not containing even-length cycles.

Theorem 4.3. *A connected viable soliton graph G is deterministic iff it satisfies one of the following two conditions.*
1. G is a chestnut augmented by a number of impervious edges.
2. Each external component of G reduces to a generalized tree, and the subgraph of G induced by the union of its internal components has a unique perfect matching.

Corollary 4.4. *It is decidable in $\mathcal{O}(m)$ time if an arbitrary graph G with n vertices and m edges is a chestnut or an elementary deterministic soliton graph.*

Proof. Chestnuts allow a simple $\mathcal{O}(m)$ decision algorithm [15], even when they are augmented by any number of impervious edges connecting their principal vertices. As to elementary graphs, reduce G first, and see if $r(G)$ is a generalized tree. By the depth-first algorithm given in [6], this can be done in $\mathcal{O}(m)$ time. Then reverse the shrinking procedure, and check if it preserves the elementary property. For more details of this procedure, see [6] or [15]. □

According to Theorems 3.3 and 4.3, every non-chestnut deterministic soliton automaton is isomorphic to an elementary one. The internal elementary components of the underlying graph G of such an automaton can be cut without affecting the behavior of the automaton. To preserve isomorphism, however, all hidden edges must be introduced in every external elementary component of G. Moreover, at each vertex v of every external elementary component such that v is adjacent to an internal family $\mathcal{F}$, a loop must be introduced around v in order not to lose soliton walks that go down to $\mathcal{F}$ and return, leaving no change behind.

Let G' denote the resulting graph, which is the union of the augmented external elementary components of G. To effectively construct G', one must first isolate and detach the internal elementary components of G, check their union for the unique perfect matching property, and produce that matching. At the same time, all hidden edges must be located and added to the remaining graph, together with the loops specified above. The graph obtained in this way is G', which still must be tested for the deterministic elementary property using reduction. If the test is successful, then it will also produce a state for the graph G', so that the whole graph G will have a state. Graph G' may then be re-reduced without eliminating any loops in it, which will preserve isomorphism, yet simplify G' to some extent.

5 The general algorithm

In this section we present an algorithm to decide if an arbitrary graph G is a viable deterministic soliton graph. We start out by generalizing an old theorem of Kotzig [16], which will be the fundamental instrument in our algorithm. Through this generalization we also provide a new and elegant proof of the old result.

Recall that a *bridge* (or cut edge) of a connected graph G is an edge $e \in E(G)$, the deletion of which cuts G into two connected components G_1 and G_2. A bridge e is called *odd* if G_1 and G_2 are both closed, having an odd number of vertices, *semi-odd* if G_1 is odd closed and G_2 is open, and *open* if both G_1 and G_2 are open. A bridge of an arbitrary graph G is one of a connected component of G.

Theorem 5.1. *Every non-chestnut viable deterministic soliton graph G having at least one internal family of elementary components contains a semi-odd bridge which belongs to every state M of G.*

Proof. Consider the partial order $\leq$ of G's families by Theorem 3.3, and let $\mathcal{F}$ be a minimal element in this partial order. It is evident that the root of $\mathcal{F}$, as a mandatory edge in G, is a semi-odd bridge. □

Intuitively, for every internal family $\mathcal{F}$ of G, only the mandatory root r of $\mathcal{F}$ can be a bridge. (See statement 1 in Theorem 3.3 regarding the ear decomposition of $\mathcal{F}$.) This may happen only if every family below $\mathcal{F}$ is only reachable through $\mathcal{F}$ from above $\mathcal{F}$ in the Hasse diagram of $\leq$. Even this fact may not guarantee that r is a bridge, because G might have impervious edges coming from above $\mathcal{F}$ to below $\mathcal{F}$. But if $\mathcal{F}$ is at the bottom of the diagram, then such impervious edges do not exist, so that $\mathcal{F}$ becomes isolated when removing its root.

Corollary 5.2. (Kotzig [16], see also [19]) *If a closed graph G has a unique perfect matching M, then G contains an odd bridge belonging to M.*

Proof. Add a single external edge e to any vertex of G to obtain an open graph G_e. Clearly, G_e is a deterministic soliton graph, which is not a chestnut. Moreover, M is a state of G_e by which $e \notin M$. Separate the viable part of G_e, and apply Theorem 5.1 to obtain a semi-odd bridge f. The edge f belongs to M, so that $e \neq f$. After deleting e, f becomes a bridge of G, which must be odd since G has an even number of vertices. □

Gabow *et al.* [11] gave the following algorithm to decide if a closed graph G has a unique perfect matching (UPM, for short), and specify that matching M in case of a positive answer. Using the terminology of [11], a *2-edge component* of graph G is a connected component of the graph remaining from G after the deletion of all bridges.

Algorithm A

```
Initialize $M = \emptyset$ and $R$ to be the set of all bridges of $G$.
While $R \neq \emptyset$ repeat the following steps:
    1. Delete an edge $(x, y)$ from $R$.
    2. If $(x, y)$ is an odd bridge, delete $(x, y)$ from $G$, add $(x, y)$ to $M$, and repeat the
       following steps for each edge $(v, w)$ incident with $x$ or $y$:
           a) Delete $(v, w)$ from $G$, and from $R$ if it is in $R$.
           b) If $v$ and $w$ are still connected but are in different 2-edge components (in
              the current graph), then:
              Find a path $p(v, w)$ connecting $v$ and $w$, and add every bridge on $p(v, w)$
              to $R$.
           c) Delete vertices $x$ and $y$.
```

Graph G has a unique perfect matching M iff the final graph becomes empty after running the above procedure. As it was shown in [11], Algorithm A runs in $\mathcal{O}(m \log^4 n)$ time.

The correctness of Algorithm A is based on Kotzig's theorem. The algorithm will remove all odd bridges from G in a recursive manner. Every time such a bridge is deleted in Step 2, it must be the case that the resulting two connected components still have a UPM.

The bridges in these components are located in Step 2b, and the efficiency of the whole algorithm hinges upon the speed of implementing this particular step. The $\mathcal{O}(m \log^4 n)$ complexity is achieved by using the dynamic 2-edge connectivity algorithm given in [14]. From our point of view this is only a technical issue, because the minor changes that we are going to apply have little impact on how the algorithm works under the modified circumstances. It is an important point, however, that the order in which odd bridges are picked for deletion in Step 2 is irrelevant.

The changes to Algorithm A are summarized as follows.

(i) In the initialization, R is set to the collection of all non-open bridges. (Note that a non-open bridge need not be closed, e.g. it can be semi-odd.)

(ii) In Step 1, bridge (x, y) is considered only if it is odd or semi-odd, and in Step 2b only non-open bridges are added to R.

In the light of Theorem 5.1, if G is a non-chestnut viable deterministic soliton graph, then the modified algorithm will remove every internal family of G after checking each of these families for the UPM property. Conversely, if, after running the modified algorithm, the remainder of G consists of a number of open components with open bridges only (if any), then the closed subgraph induced by the deleted vertices has a UPM. We should not forget, however, to come back at the end and check for the viability of the deleted part. In Step 2 we can also easily mark those vertices in the remainder graph that are adjacent to the semi-odd bridges deleted last. These are potentially the vertices where a loop must be inserted at the end to preserve isomorphism of automata. Observe that the remainder graph at this point is still a collection of external families, not components.

There is an easy way to implement the changes (i) and (ii) above without actually modifying Algorithm A. The trick is to augment G by the following vertices and edges.

– Introduce a new vertex c, and connect c to each external vertex in G.

– If $|V(G)|$ is even, then add another vertex c' and connect it to c.

In this way we have added a new star graph S with center c to G. The edges of S are called *auxiliary*. Let G^* denote the resulting graph, which is turned closed by adding a loop around each vertex with degree 1. The role of the star S is to keep the external components of G together, while killing all open bridges. Running Algorithm A on G^* will preserve this status quo, provided that the auxiliary edges are not considered for deletion in Step 1. Notice that an originally open bridge e might become semi-odd during the modified algorithm, but in this case e will become odd when Algorithm A is run on G^*. See Figure 6 for a trivial example. In general, a non-auxiliary edge (x, y) is deleted in Step 1 of Algorithm A run on G^* iff (x, y) is deleted in the same step of the modified algorithm when run on G. At the same time, of course, all edges incident with x or y are also removed by both versions of the algorithm, regardless of these edges being auxiliary or not.

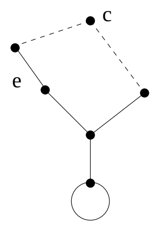

Figure 6: An open bridge e becoming close

After Algorithm A has successfully removed all potential external families of G, it will pause for a while before it resumes on a modified version of the remainder graph G'. The pause will last $\mathcal{O}(m)$ time, during which a simplification of G' takes place. We now describe the actions to be taken during the pause. Remember that we still have entire external families to cope with, which families might contain internal elementary components that we must get rid of after checking them for being mandatory.

Recall from [19] that a *factor-critical* graph is a closed graph G for which $G - v$ has a perfect matching for each $v \in V(G)$. Such a matching is called a *near-perfect matching*. Let G be a connected non-degenerate viable deterministic soliton graph different from a chestnut. By *pruning* G we mean deleting all of its external vertices and edges. Let $pr(G)$ denote the resulting closed graph. Furthermore, for each external $e \in E(G)$, let $pr_e(G)$ be the open graph obtained from $pr(G)$ by putting back the edge e only. Adding a loop around the external endpoint of e then results in a closed graph, denoted $cpr_e(G)$.

Theorem 5.3. *If $pr(G)$ does not contain bridges, then it is factor-critical. Moreover, for every external edge e, $cpr_e(G)$ has a unique perfect matching.*

Proof. It is sufficient to prove the first statement only. Indeed, if $pr(G)$ is factor-critical, then $cpr_e(G)$ does have a perfect matching, but the existence of an alternating cycle in $cpr_e(G)$ would contradict the assumption that G does not contain such a cycle.

By Theorems 3.3 and 5.1, G must be a single external family of elementary components. If this family is mandatory, then the statement follows from Theorem 3.3/1. Indeed, by Theorem 3.1 every internal vertex v of G is accessible by a positive alternating path p starting from the single external vertex of G. Switching along p then gives rise to a near-perfect matching of $pr(G)$ that misses v. If G is not a mandatory family, then introduce all hidden edges in the external elementary component C of G to obtain an elementary deterministic soliton graph C_h. By construction, C_h does not contain bridges. Using Theorem 4.3, reduce C_h to a generalized tree T. Since reduction does not introduce bridges, $pr(T)$ is a collection of pairwise edge-disjoint odd-length cycles. It follows immediately that $pr(T)$ is factor-critical. Consequently, for every external $e \in E(G)$, G has a state M_e for which e is the unique external edge present in M_e. (Remember that reduction preserves, while inverse reduction creates perfect internal matchings without adding any external edges to them.) Thus, again by Theorems 3.3 and 5.1, $pr_e(G)$ is a deterministic soliton graph consisting of a single mandatory family. The result now follows from Theorem 3.3/1 as above. □

Now let G be an arbitrary soliton graph consisting of a single external family, and assume that G does contain internal bridges. By Theorem 3.3, all these bridges must be open. Let $c \in E(G)$ be an internal bridge. By *cutting* c we mean deleting c from G first, then putting it back separately in both of the resulting two connected components as an external edge. Let G_1 and G_2 denote the two connected graphs obtained. Clearly, G_1 and G_2 are still soliton graphs consisting of a single external family, and G is deterministic iff both G_1 and G_2 are such. Cutting all internal bridges of G will then result in a number of soliton graphs $G_1, \ldots, G_k$, each of which is a single external family, so that G is deterministic iff all of $G_1, \ldots, G_k$ are such.

Combining the above argument with Theorem 5.3, we decide if a connected open graph G containing open internal bridges only is a deterministic soliton graph as follows.

Algorithm B

Step 1. Cut all open internal bridges in G to obtain the open graphs $G_1, \dots, G_k$ as described above.
Step 2. In each G_i still containing internal edges, keep one external edge e_i arbitrarily and delete the rest to obtain a graph $\bar{G}_i$.
Step 3. For each $1 \le i \le k$, check if $\bar{G}_i$ has a unique perfect matching, and if so, find that matching M_i by applying Algorithm A.
Step 4. Taking M_i as a state of G_i, find those vertices $X_i \subseteq V(G_i)$ that lie on some crossing with respect to M_i. At the same time, locate all open M-alternating ears attached to $G_i[X_i]$ and augment this graph by edges connecting the two endpoints of such ears.
Step 5. Check if $G_i[X_i]$, augmented by the hidden edges, reduces to a generalized tree. Graph G is a deterministic soliton graph iff a positive answer is obtained for each i.

Proposition 5.4. *Algorithm B correctly decides in $\mathcal{O}(m \log^4 n)$ time if a connected open graph G containing open internal bridges only is a deterministic soliton graph.*

Proof. If G is deterministic, then by Theorem 5.1 it must be a single external family, so that the argument after Theorem 5.3 applies. Conversely, assume that M_i is a UPM of each $\bar{G}_i$, and $G_i[X_i]$ reduces to a generalized tree. As we have seen in the proof of Theorem 5.3, $\bar{G}_i$ is a single external family. But then G_i, too, is a single external family, since every internal family of G_i would be one of $\bar{G}_i$ as well. (See again Theorems 3.1 and 3.3.) Let C_i denote the external elementary component of G_i, and observe that each allowed edge f in C_i must lie on some crossing of C_i with respect to to the restriction M_i' of M_i to C_i. Indeed, since f is not constant, there exists an M_i'-alternating unit passing through f. This unit cannot be an alternating cycle, because M_i is a UPM of $\bar{G}_i$. Consequently, $V(C_i) = X_i$, so that G_i is deterministic by Theorem 4.3. Thus, G is deterministic.

Regarding the complexity of Algorithm B, it is clear that Steps 1 and 2 require $\mathcal{O}(m)$ time. We also know that the cost of Step 3 is $\mathcal{O}(m \log^4 n)$. In the presence of the matchings M_i Step 4, too, can be implemented in $\mathcal{O}(m)$ time using an alternating depth-first tree rooted at the distinguished external vertex of G_i. Finally, Step 5 also requires $\mathcal{O}(m)$ time only, according to Corollary 4.4. Thus, the overall time complexity of the algorithm is $\mathcal{O}(m \log^4 n)$. □

We are now ready to combine Algorithms A and B, and state our main result.

Theorem 5.5. *It is decidable in $\mathcal{O}(m \log^4 n)$ time if an arbitrary graph G with n vertices and m edges is a viable deterministic soliton graph.*

Proof. Without loss of generality we can assume that G is open, connected, non-degenerate, and is not a chestnut. We first apply Algorithm A on G^* to recursively identify all odd and semi-odd bridges f, and decide if f, together with either of the odd internal components incident with it, is indeed a graph having a UPM with f belonging to that matching. When this algorithm stops, we remove the star graph S (or whatever is left of it) from the remainder graph and pause. During the pause we apply Steps 1 and 2 of Algorithm B to each connected component of the current graph. We also "freeze" (i.e., memorize) the graphs G_i obtained in Step 1. The pause lasts $\mathcal{O}(m)$ time, during which the graph has simplified.

Then we add the edges e_i specified in Step 2 to the set R of bridges, and continue running the main loop of Algorithm A all the way until the graph becomes empty or an error occurs. Since the size of the graph after the pause is not greater than that of the original G, the overall complexity of Algorithm A is still $\mathcal{O}(m \log^4 n)$.

At the end, with the matchings M_i at our disposal, we perform the required checks described in Steps 4 and 5 of Algorithm B on the graphs G_i.

At this point we have succesfully decided if G is a deterministic soliton graph with all of its internal elementary components being mandatory. In case of a positive answer, we have also found a state M for G. Moreover, we have isolated the external elementary components, added all of the hidden edges, and marked the vertices in these components where a loop must be inserted to preserve isomorphism of automata. In total, we have used $\mathcal{O}(m \log^4 n)$ time only, but we still need to check if G is viable. Knowing state M, this check can easily be performed in $\mathcal{O}(m)$ time by the algorithm given in [2] to find the accessible vertices and isolate the families of an arbitrary soliton graph. The proof of Theorem 5.5 is now complete. $\square$

6 Conclusions

We have presented an $\mathcal{O}(m \log^4 n)$-time algorithm to decide if a graph with n vertices and m edges is a viable soliton graph defining a deterministic automaton. The basis of our algorithm was the characterization of deterministic soliton graphs and automata given in Theorem 4.3. This characterization shows that deterministic soliton graphs different from a chestnut are essentially the open counterparts of graphs having a unique perfect matching. We have considered the currently fastest UPM algorithm, and modified it to accommodate deterministic soliton graphs. The resulting algorithm is capable of constructing a state for the automaton found, and can also be used to simplify the automaton to an isomorphic elementary one. We have also given yet more efficient $\mathcal{O}(m)$-time algorithms for chestnuts and elementary soliton graphs.

References

[1] M. Bartha and E. Gombás, On graphs with perfect internal matchings, *Acta Cybernetica* **12** (1995), 111–124.

[2] M. Bartha and M. Krész, Isolating the families of soliton graphs, *Pure Mathematics and Applications* **13** (2002), 49–62.

[3] M. Bartha and M. Krész, Structuring the elementary components of graphs having a perfect internal matching, *Theoretical Computer Science* **299** (2003), 179–210.

[4] M. Bartha and M. Krész, Deterministic soliton graphs, *Informatica* **30** (2006), 281–288.

[5] M. Bartha and M. Krész, Flexible matchings, *Lecture Notes in Computer Science* **4271** (2006), 313–324.

[6] M. Bartha and M. Krész, A depth-first algorithm to reduce graphs in linear time, submitted for publication.

[7] F. L. Carter, Comformational switching at the molecular level, in: F. L. Carter (ed.), *Molecular Electronic Devices*, Marcel Dekker, Inc., New York, 1982, 51–72.

[8] J. Dassow and H. Jürgensen, Soliton automata, *J. Comput. System Sci.* **40** (1990), 154–181.

[9] J. Dassow and H. Jürgensen, Soliton automata with a single exterior node, *Theoretical Computer Science* **84** (1991), 281–292.

[10] J. Dassow and H. Jürgensen, Soliton automata with at most one cycle, *Journal of Computer and System Sciences* **46** (1993), 155–197.

[11] H. N. Gabow, H. Kaplan and R. E. Tarjan, Unique maximum matching algorithms, *Journal of Algorithms* **40** (2001), 159–183.

[12] M. C. Golumbic, T. Hirst and M. Lewenstein, Uniquely restricted matchings, *Algorithmica* **31** (2001), 139–154.

[13] M. P. Groves, C. F. Carvalho, and R. H. Prager, Switching the polyacetylene soliton, *Materials Science and Engineering* **C3** (1995), 181–185.

[14] J. Holm, K. de Lichtemberg, and M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity, in: *Proc. 30th Annual ACM Symposium on Theory of Computing*, 1998, 79–89.

[15] M. M. Hossain, Some Graph Algorithms of Molecular Switching Devices. Master's thesis, Memorial University of Newfoundland, Canada, 2007.

[16] A. Kotzig, On the theory of finite graphs with a linear factor I, *Mat.-Fyz. Casopis Slovensk. Akad. Vied* **9** (1959), 73–91.

[17] M. Krész, Soliton Automata: A Computational Model on the Principle of Graph Matchings, PhD thesis, University of Szeged, Hungary, 2004.

[18] M. Krész, Simulation of soliton circuits, *Lecture Notes in Computer Science* **3845** (2005), 347–348.

[19] L. Lovász and M. D. Plummer, Matching Theory, North Holland, Amsterdam, 1986.

[20] J. E. Tabaska, R. B. Cary, H. N. Gabow, and G. D. Stormo, An RNA folding method capable of identifying pseudoknots and base triples, *Bioinformatics* **14** (1998), 691–699.