Practical Bigraphs via Subgraph Isomorphism

Blair Archibald
School of Computing Science, University of Glasgow, Scotland

Kyle Burns
School of Computing Science, University of Glasgow, Scotland

Ciaran McCreesh
School of Computing Science, University of Glasgow, Scotland

Michele Sevegnani
School of Computing Science, University of Glasgow, Scotland

Abstract

Bigraphs simultaneously model the spatial and non-spatial relationships between entities, and have been used for systems modelling in areas including biology, networking, and sensors. Temporal evolution can be modelled through a rewriting system, driven by a matching algorithm that identifies instances of bigraphs to be rewritten. The previous state-of-the-art matching algorithm for bigraphs with sharing is based on Boolean satisfiability (SAT), and suffers from a large encoding that limits scalability and makes it hard to support extensions. This work instead adapts a subgraph isomorphism solver that is based upon constraint programming to solve the bigraph matching problem. This approach continues to support bigraphs with sharing, is more open to other extensions and side constraints, and improves performance by over two orders of magnitude on a range of problem instances drawn from real-world mixed-reality, protocol, and conference models.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Models of computation

Keywords and phrases bigraphs, subgraph isomorphism, constraint programming, rewriting systems

Funding *Blair Archibald*: supported by the EPSRC under grant S4: Science of Sensor Systems Software (EP/N007565/1).
Kyle Burns: supported by the EPSRC under a Doctoral Training Partnership (EP/R513222/1).
Ciaran McCreesh: supported by the EPSRC under grant Modelling and Optimisation with Graphs (EP/P026842/1).
Michele Sevegnani: supported by the EPSRC under PETRAS SRF grant MAGIC (EP/S035362/1).

1 Introduction

Bigraphs are a universal modelling formalism, used to represent both the spatial relationships of entities and their global interactions. Since their introduction by Milner [24], they have been used to model, amongst others: IoT/sensor systems [27, 6], Mixed-Reality systems [9], networking protocols [10, 11], security [1], and biological systems [18]. A bigraph consists of two graph-based structures over the same set of vertices: a *place graph* describing the nesting of entities, e.g. a device within a room, and a *link graph* describing non-local relationships through hyperedges, e.g. a device connected to (numerous) other devices regardless of location. A Bigraphical Reactive System (BRS) allows bigraphs to evolve over time through a set of reaction rules, of the form $L \longrightarrow R$, that *find*, through a matching algorithm, an instance of bigraph L in a bigraph B and replace it with bigraph R. With a BRS, model verification is performed either through reachability analysis over the transition system generated by the reaction rules, or by simulation.

Efficient matching and rewriting routines are essential for practical analysis of large models, and even small improvements per match can have significant impacts on the overall

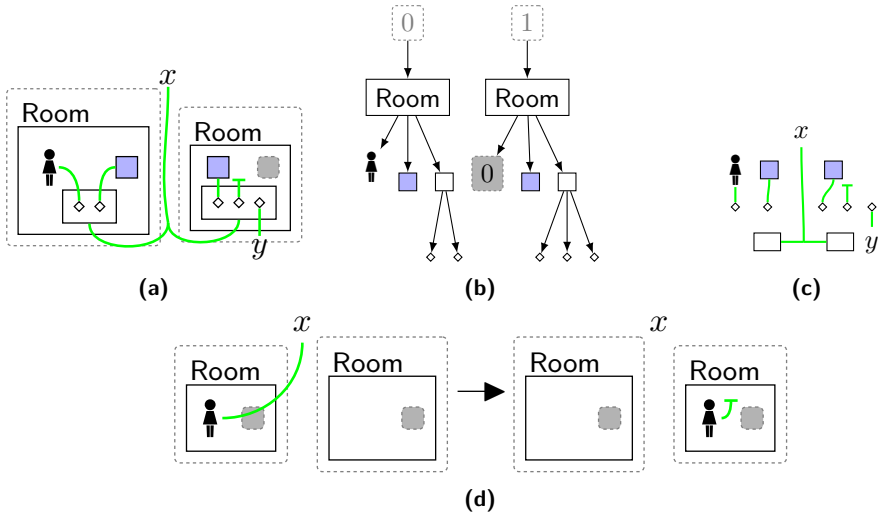


Figure 1 (a) Bigraph example with Rooms, People, Computers (blue squares), Routers (clear rectangles), and Sockets (diamonds); (b) Place graph for (a); (c) Link graph for (a), unlinked entities not drawn; (d) Reaction rule to move people between rooms.

analysis time, given the huge number of matches. In this work we show that a bigraph matching algorithm implemented on top of a constraint programming solver for the subgraph isomorphism problem provides a *performant* and *extensible* basis for bigraph matching. We provide an encoding of bigraphs to graphs, in such a way that we can solve the bigraph matching problem using a variant of the subgraph isomorphism problem (SIP) with additional constraints. This process supports both standard bigraphs, and the bigraphs with sharing extension. Using a set of real-world models, we show empirically that in *all cases* the SIP solver outperforms the previous state-of-the-art SAT solver found in the open-source BigraphER [26] toolkit, with a speedup of more than two orders of magnitude.

2 Background

We begin by giving the necessary background to present our contributions. In this section we describe the concepts and theory underlying bigraphs, and then explain bigraph matching and subgraph matching problems. The following section will explain how these matching problems can be related.

2.1 Bigraphs

Bigraphs simultaneously model systems based on both spatial and non-local relationships between entities. Throughout this paper we use bigraphs to refer to bigraphs with sharing, which allow entities to have multiple parents.

Bigraphs have equivalent algebraic and diagrammatic notation. Throughout this paper we use the diagrammatic notation when possible. An example bigraph is shown in Figure 1a. This simple model represents people and computers within rooms and their links to specific *sockets* (shown as diamonds) on a router. We use shapes and colour to denote different entity types. Nesting and adjacency of entities represents spatial relationships, e.g. the person is in the first room. The green hyperlinks represent non-spatial relationships, e.g. links between routers in different rooms. Importantly, entities have *fixed* arity (number of

links), e.g. sockets have arity 1, although links might not be linked elsewhere as shown by the second to last socket.

Spatial relationships are captured by the place graph, shown in Figure 1b, that forms a *directed-acyclic-graph (DAG)*[1] over entities. Top-level places, shown as dashed rectangles, are called *regions* that represent unknown (or empty) parent(s). The grey dashed rectangle is called a *site*; similar to regions, sites represent an unknown (or empty) child bigraph.

Non-spatial relationships are captured by the link graph, shown in Figure 1c, that forms a *hypergraph* over entities. Here, regions/sites are replaced by outer/inner names, i.e. they represent (potentially) other entities on the same link. We draw outer names above the diagrams and inner names below. A link is *open* if it connects to a name, and *closed* otherwise (i.e. it connects only the specified entities).

Regions/sites and outer/inner names—called *interfaces*—allow us to build bigraphs compositionally. That is, we can build larger bigraphs from smaller bigraphs. This is done by placing regions into sites and connecting on like-names. For example, the bigraph in Figure 1a accepts a bigraph with a single region and outer name y—adding it to the second room and linking up the incoming link to the rightmost socket—and can be composed with a bigraph with two sites (one for each region) that accepts a name x. Composition of bigraphs is shown in more detail in Section 2.3 to describe the bigraph matching problem. We denote the composition of bigraphs B_0 and B_1 as $B_0 \circ B_1$ (placing the regions of B_1 in the sites of B_0). Alternatively, we can build larger bigraphs through a tensor operation $B_0 \otimes B_1$ that places bigraphs side-by-side. In general, bigraphs form a specific type of symmetric monoidal category [24], although we do not need the full power of this fact in this paper.

2.2 Bigraph Definitions

We give enough definitions for bigraphs with sharing to explain our encoding and matching routines; full details are available elsewhere [25]. We use *concrete* bigraphs, where each entity and closed link is named. Models are usually defined over *abstract* bigraphs that represent an equivalence class of all bigraphs that have the same structure regardless of concrete names. We always perform *matching* on concrete bigraphs so this is sufficient for our purposes.

We assume a set $\mathcal{K}$ of entity types (e.g. Room), an arity function $ar : \mathcal{K} \to \mathbb{N}$, $\mathcal{V}$ a set of entity identifiers $v_0, \ldots, v_n$, $\mathcal{E}$ a set of link identifiers $e_0, \ldots, e_n$, and $\mathcal{X}$ a finite set of names $x, y, z, \ldots$, such that all names and identifiers are disjoint.

▶ **Definition 1** (concrete place graph with sharing). *A* concrete place graph with sharing

$$B = (V_B, ctrl_B, prnt_B) : m \to n$$

is a triple having m sites and n regions (treated as ordinals*)*[2]*. B has a finite set $V_B \subset \mathcal{V}$ of entities, a control map $ctrl_B : V_B \to \mathcal{K}$, and a* parent relation

$$prnt_B \subseteq (m \uplus V_B) \times (V_B \uplus n)$$

that is acyclic i.e. $(v, v) \notin prnt_B^+$ for any $v \in V_B$, with $prnt_B^+$ the transitive closure of $prnt$.

▶ **Definition 2** (concrete link graph). *A* concrete link graph

$$B = (V_B, E_B, ctrl_B, link_B) : X \to Y$$

[1] A forest in standard bigraphs, and a DAG for sharing.

[2] The function notation is used as place graphs (resp. link graphs, bigraphs) are arrows in a category with ordinals, e.g. m, n (resp. sets of names, ordinal/name pairs) as objects.

is a quadruple having (finite) inner name set $X \subset \mathcal{X}$ and an outer name set $Y \subset \mathcal{X}$. B has finite sets $V_B \subset \mathcal{V}$ of entities and $E_B \subset \mathcal{E}$ of links, a control map $ctrl_B : V_B \to \mathcal{K}$ and a link map

$$link_B : X \uplus P_B \to E_B \uplus Y$$

where $P_B \stackrel{def}{=} \{(v, i) \mid v \in V_B,\ i = ar(ctrl_B(v))\}$ is the set of ports of B.

Closed links are those where the domain is restricted to P_B and the image is in E_B. Otherwise they are *open*.

A concrete bigraph with sharing joins these two structures on V_B.

▶ **Definition 3** (concrete bigraph with sharing). *A concrete bigraph*

$$B = (V_B, E_B, ctrl_B, prnt_B, link_B) : \langle k, X\rangle \to \langle m, Y\rangle$$

consists of a concrete place graph with sharing $B^{\mathsf{P}} = (V_B, ctrl_B, prnt_B) : k \to m$ and a concrete link graph $B^{\mathsf{L}} = (V_B, E_B, ctrl_B, link_B) : X \to Y$. The inner and outer interfaces *of B are $\langle k, X\rangle$ and $\langle m, Y\rangle$, respectively.*

2.3 Bigraph Matching Problem

We denote the identity bigraph over an interface $I = \langle m, X\rangle$ by $\mathsf{id}_I : I \to I$. It maps names in X to themselves and places m sites in m regions.

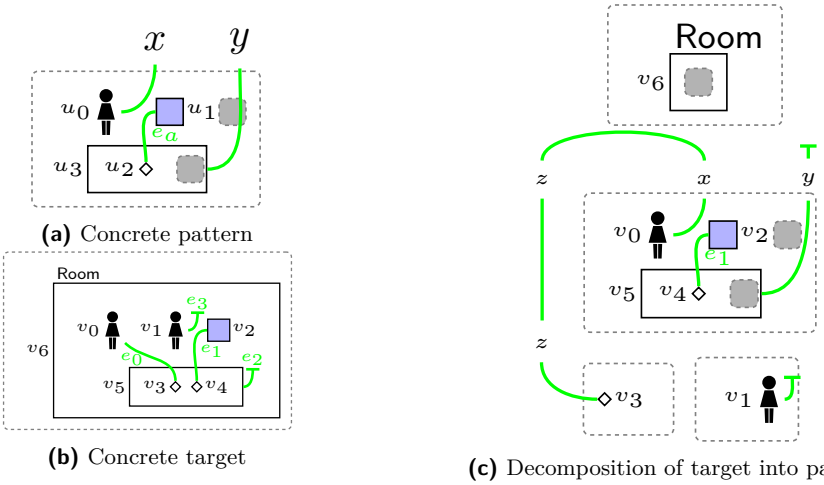
A Bigraphical Reactive System (BRS) allows bigraphs to evolve through a set of reaction rules of the form $L \longrightarrow\!\!\!\!\blacktriangleright R$. Intuitively a reaction rule replaces an occurrence of a bigraph L with R. An example reaction rule, allowing a person to move between rooms, is in Figure 1d. The use of sites within the rooms allows them to contain any other entities. Reaction rules can update both the place and link graph simultaneously as shown by the connection being severed.

The central operation when computing over a BRS is the ability to *match* the left-hand-side of a reaction rule. Matches are also used to define state predicates, i.e. as bigraph patterns [9], and multiple matches per rewrite step are required for conditional rewriting [4]. If stochastic semantics are required [18] then the *number* of matches is required to correctly normalise rates. Bigraph matching is defined formally in terms of occurrences:

▶ **Definition 4** (concrete occurrence). *Let P and T be two concrete bigraphs representing a pattern and target. We say there is a concrete occurrence of P in T if there is a valid decomposition $T = C \circ (\mathsf{id}_I \otimes P) \circ D$ for some interface I, and concrete bigraphs C and D. We call C the* context *and D the* parameter*. Two concrete occurrences are equal if they differ only by a permutation or a bijective renaming on the inner interface of C and the outer interface of D.*

It is always possible to determine an abstract occurrence starting from a concrete one. In other words, a bigraph P occurs in T (both abstract) only if an arbitrary concretisation of P occurs in an arbitrary concretisation of T, where *concretisation* means the assignment of distinct identifiers (drawn from $\mathcal{V}$ and $\mathcal{E}$) to all entities and closed links.

An example match through decomposition is in Figure 2. We use the same entities as before, but this time give a concrete pattern/target bigraph with identifiers v_n, u_n, e_n. The pattern is as given, while all additional entities are placed either in the context (i.e. the Room) or the parameter (the additional socket/person). The additional wiring of z is through the id_I component of the decomposition. Because of the ability to loop outer names back to the



(a) Concrete pattern

(b) Concrete target

(c) Decomposition of target into pattern

Figure 2 Example matching instance with $\{u_0 \to v_0, u_1 \to v_2, u_3 \to v_5, u_2 \to v_4, e_a \to e_1\}$. An alternative match with $u_0 \to v_1$ is possible.

parameter, for matching, we only need to consider open links and not distinguish between inner and outer.

Treating matching as a decomposition is essential since, while it is necessary to find an isomorphism between entities in the pattern and target graph, it is not sufficient. To have a *valid* match we *must* also be able to form a valid context/parameter. For example, we cannot have the same entity appear in both the context and the parameter.

Existing approaches.

The first bigraph matching algorithm [15] made use of structural induction on the algebraic representation of bigraphs in order to find a valid match through an inference system. The algorithm supports both standard bigraphs and binding bigraphs that allow names to have locality. A similar inductive approach was used to provide the matching algorithm for directed bigraphs [8] that allows directed link graphs.

Like the approach we outline, other algorithms encode the bigraph matching problem as an instance of a combinatorial search problem, allowing re-use of existing tools for efficiency. For example, `jLibBig` [12], which also supports directed bigraphs, formulates matching as a constraint satisfaction problem, while BigraphER [26] is the only implementation to support bigraphs with sharing through the use of SAT solvers.

The closest approach to ours encodes bigraphs as *ranked graphs* [14]. Ranked graphs can be seen a graphs-with-interfaces, mirroring the sites/regions/names of bigraphs. Rather than perform the encoding only for matching, ranked graphs are used to do the *rewriting* (as an instance of double-pushout graph transformation) and then the entire structure is converted back to a bigraph. Assuming negligible encoding/decoding time, the performance of this approach depends on the underlying graph transformation framework.

2.4 Subgraph Isomorphism

The Subgraph Isomorphism Problem (SIP) is a classic NP-complete decision problem that determines whether a pattern graph is a subgraph of (i.e. is present in) a target graph. Because of its broad applicability, many dedicated solving algorithms exist, with the current

state of the art being the *Glasgow Subgraph Solver* [22]. This solver adopts a constraint programming approach, combining inference and intelligent backtracking search, but with special data structures and algorithms designed specifically for subgraph problems. The solver supports variants of the problem, including non-induced and induced subgraph finding, graphs involving directed edges, and labelling schemes defining vertex and edge compatibilities. It can also explicitly enumerate all solutions, rather than just deciding whether one solution exists.

Formally, the problem we will be solving is as follows. Given a pattern directed graph $G = (V_G, E_G)$, a target directed graph $H = (V_H, E_H)$, and a vertex compatibility function[3] $\ell : V_G \times V_H \to \{t, f\}$, a *non-induced subgraph isomorphism with vertex compatibility constraints* from G to H is an injective mapping $i : V_G \to V_H$ such that edges are mapped to edges, $(u, v) \in E_G \implies (i(u), i(v)) \in E_H$, and where vertex compatibility is respected, $\ell(v, i(v)) = t \; \forall v \in V_G$. We wish to enumerate *all* such mappings.

3 Bigraph Matching as Subgraph Isomorphism

The key observation underlying our new approach is that bigraph matching instances can be seen as a SIP problem with additional constraints to handle abstractions (regions/sites, and names), and ensure valid decompositions (Section 2.3). Using a subgraph model rather than a SAT model is potentially beneficial for three reasons. Firstly, subgraph solvers can carry out stronger reasoning than SAT solvers, by using implied constraints based upon degrees and neighbourhood degree sequences [29], cardinality [28], distances [7], and path counts [20]. Secondly, we know how to design very good variable- and value-ordering heuristics for graph problems [21, 5]. And thirdly, subgraph solvers can potentially deal with much larger problem instances due to a compact representation of adjacency constraints.

However, most existing SIP solvers do not directly support hypergraphs, let alone multiple overlaid graph structures like bigraphs. In order to use existing tools we encode the pattern/target bigraphs into a traditional graph structure (with additional degree constraints). This is a two step process. First we encode place graphs by removing/replacing abstractions and then we *flatten* the link graphs into these encoded structures by replacing hyperedges with cliques resulting in a single *flattened* graph. This flattened graph representation is accepted by existing SIP tools, with small changes required to allow the vertex compatibility function (encoding labelling and additional degree constraints). These are required as the bigraph variant of SIP can be considered a special case somewhere between induced and non-induced SIP, that is, the use of sites/regions/names swaps the matching semantics from *must have all edges matching* to *must have at least n edges matching*. This encoding is slightly under-constrained, however: although every bigraph matching corresponds to a subgraph isomorphism, some subgraph isomorphisms do not give valid bigraph matchings, and there can be multiple subgraph isomorphisms for a given bigraph matching. Rather than attempting to handle these details through an awkward encoding, we will instead make use of additional constraints (Section 3.4) and projection nogoods (Section 3.5) on solutions to obtain the desired one-to-one correspondence between solver outputs and solutions.

Importantly there is not a single encoding function between bigraphs and flattened graphs. Instead, to allow matching constraints to be specified, we require different encoding functions

[3] Note that this kind of compatibility scheme is more general than the typical "oxygen atoms must be mapped to oxygen atoms, and fluorine atoms must be mapped to fluorine atoms" labelling scheme which occurs in many applications, but nearly all subgraph isomorphism solvers can easily be modified to support this in practice through simple unary constraints.

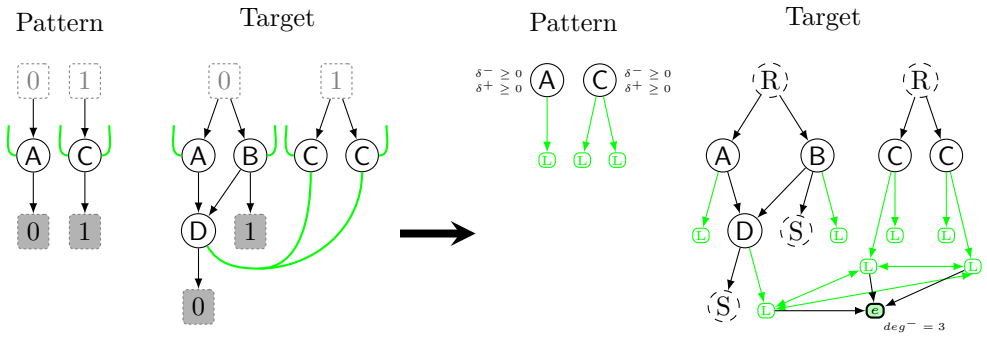


Figure 3 An illustration of a full encoding of a bigraph matching instance as a SIP instance. Unconnected links are open.

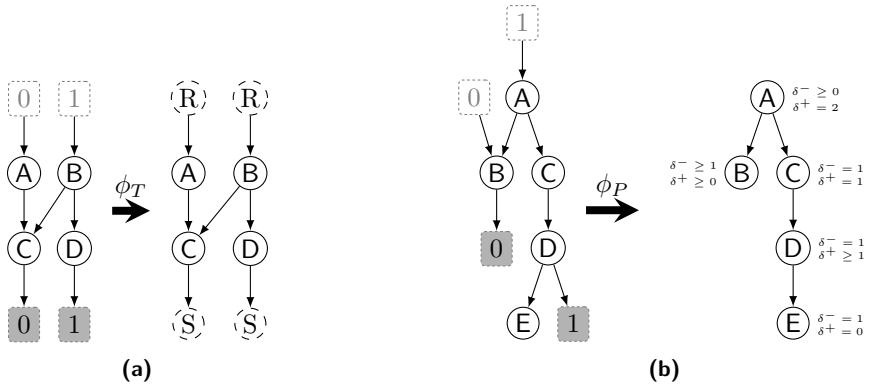


Figure 4 (a) Example target place graph encoding—regions/sites are replaced with unmatchable vertices (shown as R and S but have no label in the encoding). Vertices show control labels; (b) Example pattern place graph encoding—regions/sites removed and degree constraints introduced.

for the pattern and target graphs. A diagrammatic overview of the encodings is shown in Figure 3. In the following sections we detail how the encodings are constructed. The encodings are not total but are defined for all cases where matching is non-trivial—for example, they are not designed to be used for node-free bigraphs, i.e. bigraphs containing only hyperedges between names and/or roots/sites for example the identity bigraph.

3.1 Place Graph Encoding

The encodings take bigraphs and produce graphs (V, E) where V is a set of vertices and E a set of edges. Additional constraints are specified with a compatibility function ℓ_p that we define in Section 3.1.1.

The target place graph.

Let $T^{\mathsf{P}} = (V_T, ctrl_T, prnt_T) : m \to n$ be a concrete place graph representing the target of a matching instance. The target place graph encoding function $\phi_T : T^{\mathsf{P}} \mapsto (V, E)$ is shown diagrammatically in Figure 4a. Intuitively, we take the original place graph and extend it with additional vertices for the sites and regions. Formally, the encoding produces a graph with $V = V_T \uplus \{r_i \mid i \in n\} \uplus \{s_i \mid i \in m\}$ and $E = prnt_T^{-1}$ (the child relation). The site/region vertices are added to ensure the *structure* is maintained, i.e. parent/child entities have correct

in/out degrees, but regions/sites are never compatible and so do not appear in mappings.

The pattern place graph.

Let $P^{\mathsf{P}} = (V_P, ctrl_P, prnt_P) : i \to j$ be a concrete place graph representing the pattern of a matching instance. The pattern encoding function $\phi_P : P^{\mathsf{P}} \mapsto (V, E)$ is shown diagrammatically in Figure 4b. Intuitively we take the original place graph and remove sites/regions as we do not want to map abstract nodes into the target. The encoding produces a graph with vertices $V = V_P$, edges $E = \{(u, v) \in prnt_T^{-1} \mid v \notin i, u \notin j\}$. As we still need to remember the structure, we replace these with unary in/out degree constraints in the compatibility function (Section 3.1.1). For sites/regions we introduce $\geq$ constraints to allow additional incoming/outgoing edges, while all other entities must match in/out degrees exactly.

3.1.1 Place Compatibility Function

Additional bigraph specific constraints are handled by a place compatibility function $\ell_p : V_G \times V_H \to \{t, f\}$ that specifies when a pattern-target pair is allowed in a mapping. When defining ℓ_p we assume bigraph definitions such as V_T, $ctrl_P$ and $prnt_T$ are available for pattern/target bigraphs. For clarity we define ℓ_p over *bigraph* nodes, e.g. $u \in V_P$, although formally ℓ_p is over SIP graph vertices (e.g. V_G) and these are inverse-mapped into their bigraph representation for checking compatibility.

We define ℓ_p as the logical conjunction of two sub-functions, i.e. $\ell_p(u, v) = \ell_{p_1}(u, v) \wedge \ell_{p_2}(u, v)$. ℓ_{p_1} ensures the bigraph controls are maintained, while ℓ_{p_2} introduces cardinality constraints based on sites/regions.

$$\ell_{p_1}(u \in V_P, v \in V_T) = \begin{cases} t \text{ if } ctrl_P(u) = ctrl_T(v) \\ f \text{ otherwise} \end{cases}$$

Simply states that entities must maintain their controls.

For ℓ_{p_2}, we assume $\delta^- : V_p \to \mathbb{N}$ is the function determining the number of in-edges for an entity *ignoring* regions (ordinal n), and likewise $\delta^+ : V_p \to \mathbb{N}$ as the number of out-edges of an entity *ignoring* sites (ordinal m)[4].

We then define:

$$\ell_{p_2}(u \in V_P, v \in V_T) = \begin{cases} t \text{ if } prnt(u) \cap n \neq \emptyset \wedge \delta^-(v) \geq \delta^-(u) \\ t \text{ if } prnt(u) \cap n = \emptyset \wedge \delta^-(v) = \delta^-(u) \\ t \text{ if } prnt(u)^{-1} \cap m \neq \emptyset \wedge \delta^+(v) \geq \delta^+(u) \\ t \text{ if } prnt(u)^{-1} \cap m = \emptyset \wedge \delta^+(v) = \delta^+(u) \\ f \text{ otherwise} \end{cases}$$

Where m and n are the sites/regions of the pattern bigraph. The vertex compatibility function replaces the regions/sites with the semantics of how they should be matched, e.g. that entities connecting sites can have any number of additional children including none, but entities without sites must match out-degrees exactly.

[4] As described in Definition 1, ordinals are used to represent the place graph interfaces, such that, for example, $n = 2 = \{0, 1\}$ works as an (ordered) set with two points (roots) that can connect to the wider context.

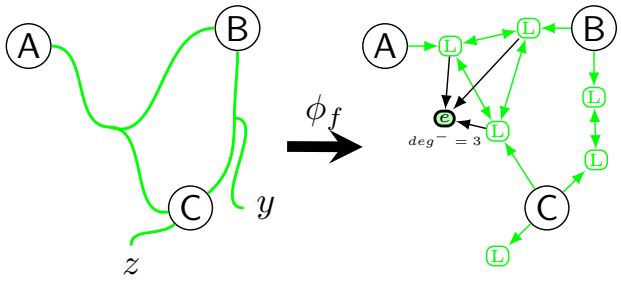


Figure 5 An example of flattening—links become cliques between port vertices, and closure nodes, shown as solid green, are added for closed links.

3.2 Link Graph Encoding

Once we have an encoded place graph we *flatten* the hyperedges in the link graph into it so we can treat links as vertices in our encoded graph. The key challenge is allowing non-injective matches for open links to cover the case where, for example, two open links are merged in the context. Unlike the place graph encoding, link graphs are always flattened the same way regardless of whether they are targets or patterns.

Let B^{L} be a concrete link graph: $(V_B, E_B, ctrl_B, link_B) : X \to Y$, and $\phi_{\{P,T\}}(D^{\mathsf{P}}) : (V_D, E_D)$ be a (pattern or target) encoding of a place graph D^{P}. We define the flattening function $\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$ that given an encoded place graph produces a new flattened graph.

Intuitively flattening creates a new vertex for every *port* (Definition 2) in the link graph, and connects these as a child of their corresponding entities. As arities are fixed, as is the number of port vertices to be added. The vertex compatibility function ensures that port nodes can only match with other port nodes in a subgraph isomorphism. As ports are treated separately to entity nodes, they do not contribute towards the δ^-, δ^+ values in the place graph encoding. Port vertices are wired based on the existing links, i.e. if they shared a link in the bigraph they share a link in the flattened representation. However as graphs do not support hyperedges the edges are encoded by forming a *clique* between the ports.

Converting links to cliques between port vertices is sufficient to encode open links. For closed links we have the following additional constraints:

1. Closed links in the pattern cannot map to open links in the target.
2. Closed to closed link mappings can only be one-to-one and have identical connected sets.
3. Closed links in the target can still be mapped to by many open links.

We implement these constraints by adding additional *closure* vertices e_n for each closed link clique encoding, to represent the "closing off" of these links. The closure vertex is linked to all port nodes in a closed link clique, and we add an equal-degree constraint to each closure node to enforce injectivity, i.e. each closed link in the pattern can only map to a single closed link with an identical adjacency set. The encoding for a link graph featuring both open and closed edges is shown diagrammatically in Figure 5.

Formalising flattening.

Given concrete link graph: $B^{\mathsf{L}} : (V_B, E_B, ctrl_B, link_B) : X \to Y$, and $\phi_{\{P,T\}}(D^{\mathsf{P}}) : (V_D, E_D)$ an encoding of a (pattern or target) place graph D (where $V_B = V_D$), we define $\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$. We let $\widehat{E_B} = \{e \in E_B \mid link_B(p) = e, p \notin X\}$ be the set of

closed links in B^{L}, and we have $V = V_D \uplus P_B \uplus \widehat{E_B}$, where P_B are the ports of B^{L} (defined in Definition 2), and one closure node is added for all closed links. We re-use the bigraph *edge* identifier as a vertex identifier in the flattened graph. For edges, $E = E_D \uplus \{(v,p) \mid p = (v,i) \in P_B\} \uplus \{(p_1,p_2) \mid p_1,p_2 \in P_B, link_B(p_1) = link_B(p_2)\} \uplus \{(p,e) \mid e \in \widehat{E_B}, link_B(p) = e\}$. As we build edges with $link_B(p_1) = link_B(p_2)$ we always get two directed edges e.g. (p_1,p_2) and (p_2,p_1) between two linked ports, and the clique structure is constructed automatically through this restriction to binary edges. Additional edges point port vertices to the closure nodes for the closed link representation.

Finally we extend the vertex capability function for place graphs ℓ_p (Section 3.1.1) to include extra link constraints:

$$\ell(u,v) = \ell_p(u,v) \wedge \begin{cases} t \text{ if } u \in P_P \wedge v \in P_T \\ t \text{ if } u \in \widehat{E_P} \wedge v \in \widehat{E_T} \wedge \deg^-(u) = \deg^-(v) \\ f \text{ otherwise} \end{cases}$$

This expresses that ports can only map to ports, and closure nodes can only map to closure nodes with the *exact* same in degree, where $\deg^-$ is the standard in-degree function.

3.3 Encoding Size

A key challenge with the existing SAT solver based algorithm is the large number of clauses required to encode the problem. On the other hand our SIP encoding requires a modest number of nodes and edges, with nodes growing linearly and edges quadratically (due to the clique representation). The number of nodes and edges for a pattern and target bigraph is as follows, where $|e| = |link_G^{-1}(e) \cap P_G|$ is the cardinality of a hyperedge $e \in E_G$ when counting only ports.

Pattern bigraph $P : \langle i, X \rangle \to \langle j, Y \rangle$

$$|V| = |V_P| + |P_P| + |\widehat{E_P}|$$
$$|E| = \sum_{v \in V_P} \delta^-(v) + |P_P| + \sum_{e \in E_P} |e| \cdot (|e| - 1) + \sum_{e \in \widehat{E_P}} |e|$$

Target bigraph $T : \langle n, X' \rangle \to \langle m, Y' \rangle$

$$|V| = |V_T| + n + m + |P_T| + |\widehat{E_T}|$$
$$|E| = \sum_{v \in V_P} \deg^-(v) + \sum_{s \in n} \deg^-(s) + |P_T| + \sum_{e \in E_T} |e| \cdot (|e| - 1) + \sum_{e \in \widehat{E_T}} |e|$$

3.4 Checking Constraints

To complete the encoding for standard bigraph matching, we require one additional constraint in the case where a region may have multiple children in the pattern graph—this introduces the rule that all of the child nodes must remain siblings relative to each resultant parent node that substitutes the abstract region during bigraph composition. We cannot validate this during the encoding stage due to the loss of this information from stripping regions in the pattern graph, and as a result the returned solution set may return a superset containing false solutions where the assigned target vertices do not share a parent. An encoding workaround

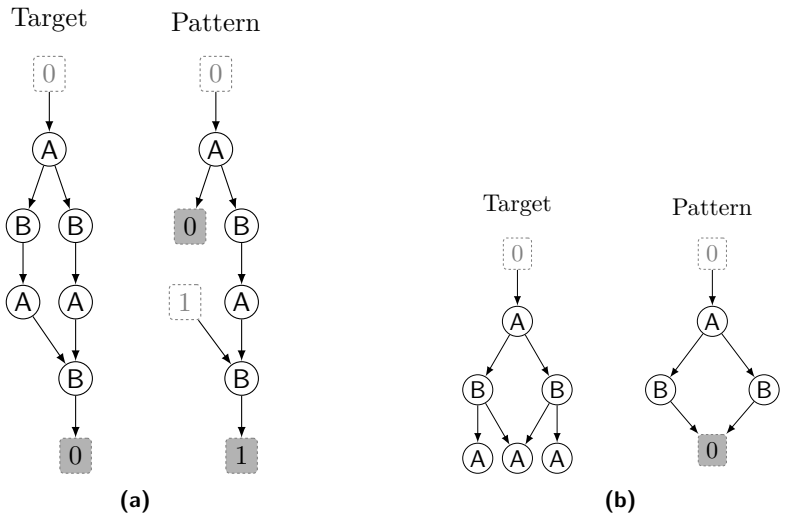


Figure 6 (a) Example instance showing need for transitive closure to avoid the parameter also appearing as a context; (b) Example instance with shared site in pattern. No matches are possible in either example.

would require the underlying SIP algorithm to be able to support pattern vertices that can encapsulate multiple target nodes in a matching assignment, however this would stray too far from the idea that any existing SIP solver that supports direction and labelling can be used for bigraph matching. We instead deal with this underconstrainedness by implementing a checking constraint on top of the constraint programming model through analysis of the input graphs and their vertex assignments whenever this case occurs—this enforces that a solution is only valid when the set of children of each region all share the same set of parent vertices once mapped to their corresponding target vertices. A similar constraint is also required in the case where we wish to support bigraphs with sharing, where we consider sites with multiple parents rather than regions with multiple children.

Similar constraints are required for the existing SAT encoding to handle the same conditions, so although these are not implemented directly in SIP, evaluating the two approaches against one another remains a fair comparison.

Constraints Imposed By Sharing

Our encoding in addition to the constraint described above is enough to perform matching for Milner's original bigraph formalism. However, bigraphs with sharing cannot be fully supported[5] without additional constraints that ensure firstly, that we do not try to form a bigraph where a site also ends up as a region i.e. the DAG property is violated from a cycle being introduced; and secondly that shared sites contain exactly the same elements.

Figure 6a shows an example where it seems there are valid matches, but sharing leads to an invalid context/parameter. Regardless of which of the two possible matches we choose, the remaining vertices are captured by site 0 in the *parameter*. However, this same path needs to rejoin the pattern through the *context* (region 1). As the same (concrete) bigraph cannot appear in both the context and parameter at the same time, both matches are invalid.

[5] Sharing within a pattern would be possible, but not on the interfaces.

To compute this constraint we check that no vertex in the match is *transitively* connected through *prnt* to any vertices in the parameter, that is, you never go upwards to reach the parameter.

Figure 6b shows the second sharing constraint, which is symmetric to the region constraint required for the standard bigraph formalism. As the site is shared it must include *only*, and all, entities shared by the two parents. In this example that means it must contain only a single A entity. This means there is nowhere in the parameter for the additional A children to go and so there is no match.

We implement both these constraints through analysis of the input graphs and their vertex assignments, ensuring all solutions returned are valid. These constraints always hold for standard bigraphs so it is safe to use them in all cases, although as an optimisation we detect sharing and only enable them for bigraphs with sharing.

The ease of implementing sharing constraints demonstrates a further advantage of our SIP encoding over the existing SAT algorithm: we can easily implement further variants of the bigraph matching problem by specifying additional high-level constraints instead of configuring the low-level set of clauses to support new conditions.

3.5 Nogood Recording

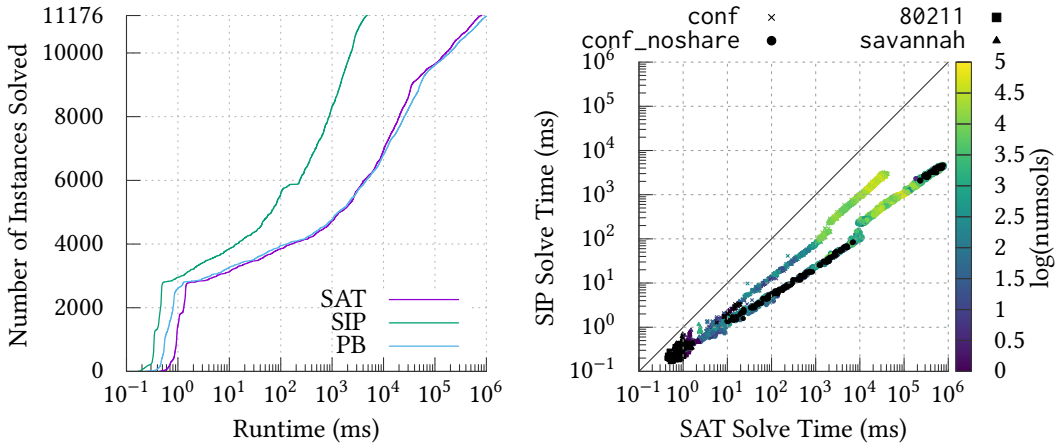
When enumerating solutions, the bigraph matching problem does not consider a permutation of open link assignments to be a new separate solution if there already exists a solution with the same place graph vertex and closed link assignments—this will result in the SIP solver returning "duplicate" solutions in its solution set which, whilst still technically valid solutions that differ in their vertex assignments, should not increment the total number of solutions found. We thus make use of the constraint solver's inbuilt nogood recording functionality where we insert a nogood upon finding a new valid solution, which records the current assignments of the place graph vertices and closure vertices such that any future solutions found with the same set of assignments are disregarded by the solver. This ensures that the set of solutions found by the SIP solver for an encoded bigraph matching instance will always bijectively match the set of solutions found by existing bigraph tools.

4 Implementation and Evaluation

We implemented [3] the encoding and SIP solving process within the Glasgow Subgraph Solver [22] due to it being the state of the art for subgraph solving. However, our approach could be implemented using any solver which supports solution enumeration, directed graphs, and a way of specifying vertex compatibilities.

We compare our SIP implementation to BigraphER's existing SAT approach on systems with dual Xeon E5-2687A v4 CPUs and 512GBytes RAM, running Ubuntu 18.04. To allow experimenting with a large number of instances, we perform up to 30 matching problems in parallel on the same machine. The SIP solver is compiled with GCC 7.5 while BigraphER is compiled with OCaml 4.10, statically linked to MiniSAT [13] compiled with GCC 9.3, and then copied to the benchmarking machine. For both solvers we measure the steady-clock time required to call the main solver function that includes generating clauses or constraints, but not the time taken to read input files from disk.[6]

[6] In an application context, calls to a solver would be made directly.



(a) Time to solve n instances.

(b) Instance solve time.

Figure 7 Comparing the performance of the SIP, SAT, and Pseudo-Boolean (PB) approaches. On the left, the cumulative number of instances solved over time for all three approaches. On the right, comparing SAT and SIP on an instance by instance basis; point colour indicates the number of solutions found, and shape the benchmark family.

As an additional point of comparison we have also implemented a pseudo-Boolean variant of the SAT algorithm, i.e. with direct cardinality constraints when encoding sites. The implementation is through BigraphER with MiniCARD [19] as the underlying solver.

Instances are drawn from two real-world models: `savannah` models a mixed-reality system [9] where children must work together in the physical world to hunt virtual impala, and `80211` that models the 802.11 MAC protocol [11]. In each case instances are recorded from steps to compute the full transition system of the model. Existing public datasets contain only relatively small bigraphs due to limitations of earlier solvers, and so to test scalability we additionally generate larger instances based on the conference call example of Milner [24, chapter 1]. In this case we generate larger instances by not only allowing the existing bigraph to reconfigure, e.g. for agents to join calls, but also rules that add additional agents, computers, rooms and buildings at random. We have two variants of the conference example `conf` and `conf_noshare` that allow/disallow sharing.

Instances use BigraphER's text based bigraph representation, which is essentially an adjacency matrix with additional entity type and link information. There are 11,176 matching instances in total and we are making these freely available, along with the results in this paper [2], to allow future comparative work. Of the 11,176 instances, 1,660 are unsatisfiable (i.e. no rewriting can be applied). To give us confidence in our implementation, we verified that the SAT and SIP approaches returned the same set of solutions for all of these 11,176 instances.

4.1 Performance

Figure 7a shows the cumulative number of instances that can be solved *individually* with a timeout of t (x-axis), and so the curve further to the top and left shows the better solver. The horizontal distance between the lines shows the increase in timeout required for both solvers to solve the same number of (but not necessarily the exact same set of) instances, and measures *aggregate speedup* [17]. In this case, no instance takes the SIP solver more

than 4,516ms to solve, while the SAT solver requires 820,117ms to solve its hardest instance, giving an aggregate speedup of 181. The pseudo-Boolean solver performs similarly to SAT especially for the harder instances, although there does seem to be a significant benefit for smaller instances. When solving many instances, i.e. when generating a transition system, we expect the pseudo-Boolean solver to be beneficial as it generates less clauses that would need to be garbage collected by OCaml. As expected given the encoding, the number of SAT clauses increases rapidly as the problem instances increase in size.

Figure 7b compares the per-instance runtime for both the SAT and SIP solver, where any point below the mid-line implies SIP outperforms SAT for that instance; the colour of each point gives the log of the number of solutions, with unsat instances shown in black. We see the SIP *always* outperforms the SAT solver for all instances. The consistency of the speedups seen is interesting, especially given the difference in techniques used by the two approaches. Further experiments with non-default configurations of the Glasgow Subgraph Solver show that for bigraph instances, neither neighbourhood degree sequence filtering nor supplemental graph constraints make much of a difference to performance. One might guess that perhaps all instances are computationally easy for any reasonable solver, and that long-running instances are due to either initialisation costs or the cost of enumerating large numbers of solutions. However, things are not this simple: some of the hardest instances are unsatisfiable, take hundreds of thousands of decisions to solve, and spend most of their runtime doing this search. In fact, close inspection of solver statistics suggests that the hard unsatisfiable instances contain very many *near*-solutions, that fail only on the checking constraints discussed in Section 3.4. This suggests that most of the performance gain comes down to the smaller encoding size and faster propagation speeds of the SIP solver, rather than any algorithmic cleverness—although there is still the risk that an insufficiently advanced solver will perform extremely badly on some instances [21]. As future work, we intend to investigate ways of speeding the solver up on these instances, either by using a propagating constraint, or through generation of small conflict clauses.

Finally, although the the difference in solve time for the real-world instances, i.e. `savannah` and `80211`, can seem modest, when generating transition systems for verification the solve routine can easily be called thousands of times. As such any speedup is likely to have a high impact on total model generation time.

5 Conclusion

We have shown that the bigraph (with sharing) matching problem can be considered a special case of the subgraph isomorphism problem with additional constraints to handle site/regions, open/closed links and sharing. Through an encoding from bigraphs to graphs with a vertex compatibility function, we can integrate with existing SIP solvers such as the Glasgow Subgraph Solver. We use this to improve on the state of the art SAT-based solver for bigraphs (with sharing), and show significant improvements to performance and scalability including an aggregate speedup of 181 when performance is compared for over 11,000 instances.

Future work

While the new solver already significantly outperforms the SAT approach, there is scope to optimise further, for example using symmetry breaking to reduce the impact of cliques in the encoding, adding further inference that exploits the structure of the labelling function, and using propagation rather than solution checking for sharing.

We expect to see our approach integrated into BigraphER as the new default matching algorithm, allowing a wider range of models to be efficiently manipulated.

Further afield, we wish to extend the algorithm to capture additional bigraph variants—something that was particularly difficult to do through the low-level CNF encoding of SAT. There are many extensions to the bigraph theory, such as local bigraphs [23] that support locality of names, e.g. to model restriction in the π-calculus, and directed bigraphs [16]. Just as sharing introduced a small number of additional constraints, e.g. transitive closure (Section 3.4), we believe that supporting additional bigraph extensions is possible, and requires significantly less effort than the SAT encoding, due to high-level constraint-based reasoning. Once extra variants are supported we will be able to perform a comparative study with existing solvers for these variants (e.g. [12]), to learn from and share new solving techniques.

References

1 Faeq Alrimawi, Liliana Pasquale, and Bashar Nuseibeh. On the automated management of security incidents in smart spaces. *IEEE Access*, 7:111513–111527, 2019. doi:10.1109/ACCESS.2019.2934221.

2 Blair Archibald, Kyle Burns, Ciaran McCreesh, and Michele Sevegnani. Practical Bigraphs via Subgraph Isomorphism – Benchmark Instances and Results. URL: http://dx.doi.org/10.5281/zenodo.4597074.

3 Blair Archibald, Kyle Burns, Ciaran McCreesh, and Michele Sevegnani. Practical Bigraphs via Subgraph Isomorphism – Glasgow Subgraph Solver Bigraph Source Code. URL: https://doi.org/10.5281/zenodo.5161185.

4 Blair Archibald, Muffy Calder, and Michele Sevegnani. Conditional bigraphs. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020*, volume 12150 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2020. doi:10.1007/978-3-030-51372-6_1.

5 Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. Sequential and parallel solution-biased search for subgraph algorithms. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Proceedings*, 2019. doi:10.1007/978-3-030-19212-9_2.

6 Blair Archibald, Min-Zheng Shieh, Yu-Hsuan Hu, Michele Sevegnani, and Yi-Bing Lin. BigraphTalk: Verified design of IoT applications. *IEEE Internet Things J.*, 7(4):2955–2967, 2020. doi:10.1109/JIOT.2020.2964026.

7 Gilles Audemard, Christophe Lecoutre, Mouny Samy Modeliar, Gilles Goncalves, and Daniel Cosmin Porumbel. Scoring-based neighborhood dominance for the subgraph isomorphism problem. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, 2014. doi:10.1007/978-3-319-10428-7_12.

8 Giorgio Bacci, Davide Grohmann, and Marino Miculan. DBtk: A toolkit for directed bigraphs. In *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009. Proceedings*, pages 413–422, 2009. doi:10.1007/978-3-642-03741-2_28.

9 Steve Benford, Muffy Calder, Tom Rodden, and Michele Sevegnani. On lions, impala, and bigraphs: Modelling interactions in physical/virtual spaces. *ACM Trans. Comput.-Hum. Interact.*, 23(2):9:1–9:56, 2016. doi:10.1145/2882784.

10 Muffy Calder, Alexandros Koliousis, Michele Sevegnani, and Joseph S. Sventek. Real-time verification of wireless home networks using bigraphs with sharing. *Science of Computer Programming*, 80:288–310, 2014. doi:10.1016/j.scico.2013.08.004.

11 Muffy Calder and Michele Sevegnani. Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing. *Formal Asp. Comput.*, 26(3):537–561, 2014. doi:10.1007/s00165-012-0270-3.

12 Alessio Chiapperini, Marino Miculan, and Marco Peressotti. Computing embeddings of directed bigraphs. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020*, volume 12150 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2020. doi:10.1007/978-3-030-51372-6_3.

13 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003.*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.

14 Amal Gassara, Ismael Bouassida Rodriguez, Mohamed Jmaiel, and Khalil Drira. Executing bigraphical reactive systems. *Discret. Appl. Math.*, 253:73–92, 2019. doi:10.1016/j.dam.2018.07.006.

15 Arne John Glenstrup, Troels Christoffer Damgaard, Lars Birkedal, and Espen Højsgaard. An implementation of bigraph matching. Technical Report TR-2010-135, IT University of Copenhagen, 2010.

16 Davide Grohmann and Marino Miculan. Directed bigraphs. In Marcelo Fiore, editor, *Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS 2007*, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 121–137. Elsevier, 2007. doi:10.1016/j.entcs.2007.02.031.

17 Ruth Hoffmann, Ciaran McCreesh, Samba Ndojh Ndiaye, Patrick Prosser, Craig Reilly, Christine Solnon, and James Trimble. Observations from parallelising three maximum common (connected) subgraph algorithms. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018*, pages 298–315, 2018. doi:10.1007/978-3-319-93031-2_22.

18 Jean Krivine, Robin Milner, and Angelo Troina. Stochastic bigraphs. *Electr. Notes Theor. Comput. Sci.*, 218:73–96, 2008. doi:10.1016/j.entcs.2008.10.006.

19 Mark H. Liffiton and Jordyn C. Maglalang. A cardinality solver: More expressive constraints for free - (poster presentation). In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 485–486. Springer, 2012. doi:10.1007/978-3-642-31612-8_47.

20 Ciaran McCreesh and Patrick Prosser. A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Proceedings*, 2015. doi:10.1007/978-3-319-23219-5_21.

21 Ciaran McCreesh, Patrick Prosser, Christine Solnon, and James Trimble. When subgraph isomorphism is really hard, and why this matters for graph databases. *J. Artif. Intell. Res.*, 61:723–759, 2018. doi:10.1613/jair.5768.

22 Ciaran McCreesh, Patrick Prosser, and James Trimble. The Glasgow Subgraph Solver: Using constraint programming to tackle hard subgraph isomorphism problem variants. In *Graph Transformation - 13th International Conference, ICGT 2020*, pages 316–324, 2020. doi:10.1007/978-3-030-51372-6_19.

23 Robin Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *Electr. Notes Theor. Comput. Sci.*, 175(3):65–73, 2007. doi:10.1016/j.entcs.2006.07.035.

24 Robin Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

25 Michele Sevegnani and Muffy Calder. Bigraphs with sharing. *Theor. Comput. Sci.*, 577:43–73, 2015. doi:10.1016/j.tcs.2015.02.011.

26 Michele Sevegnani and Muffy Calder. BigraphER: Rewriting and analysis engine for bigraphs. In *Computer Aided Verification - 28th International Conference, CAV 2016. Proceedings, Part II*, pages 494–501, 2016. doi:10.1007/978-3-319-41540-6_27.

27 Michele Sevegnani, Milan Kabác, Muffy Calder, and Julie A. McCann. Modelling and verification of large-scale sensor network infrastructures. In *23rd International Conference*

on Engineering of Complex Computer Systems, ICECCS 2018, pages 71–81. IEEE Computer Society, 2018. doi:10.1109/ICECCS2018.2018.00016.

28 Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artif. Intell.*, 174(12-13):850–864, 2010. doi:10.1016/j.artint.2010.05.002.

29 Stéphane Zampelli, Yves Deville, and Christine Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3):327–353, 2010. doi:10.1007/s10601-009-9074-3.