Efficient and Scalable Algorithms for Bigraph Matching

Kyle Burns

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computing Science

College of Science and Engineering
University of Glasgow

January 2024

Abstract

Bigraph reactive systems provide an established approach to modelling large and dynamic systems which contain both spatial and non-spatial relationships between entities, and have been used for modelling in a wide variety of research areas such as biology, networking, sensors and security. Bigraph state rewriting operations which represent temporal evolution rely upon an underlying NP-complete matching algorithm to identify pattern components to substitute, but this limits the scale and scope of bigraphs due to the computational cost and frequency of rewriting. The bigraph matching tool BigraphER relies on a Boolean satisfiability (SAT) encoding to do this, which generates an impractically large number of clauses for larger models which limits scalability and is difficult to adapt for extensions to the bigraph structure.

We propose a novel and efficient algorithm for solving bigraph matching which encodes the problem as a subgraph isomorphism constraint satisfaction problem, applying additional constraints where required to model the added complexity of bigraph composition logic. This approach can be supported by any constraint programming toolkit as well as any graph solving tool which supports additional side-constraints. This approach also grants more flexibility in regards to modelling extensions to the bigraph formalism such as bigraphs with sharing and directed bigraphs.

We adapt the state of the art constraint-based Glasgow Subgraph Solver tool to implement the encoded matching problem, where we observe a greater solve time of over two orders of magnitude on a variety of different real-world matching instances performed within mixed-reality, protocol and conference models. We also integrate the subgraph solver into the BigraphER framework and provide further evaluation metrics when used as a component for building full-scale models.

We build further upon this idea by proposing an adaptation for the McSplit algorithm for finding largest common subgraphs in order to obtain a maximum common bigraph between two agents, a novel definition which provides the means for supporting more rich and complex types of modelling in a bigraph toolkit such as performing contextual transitions for

labelled transition systems and identifying bisimulations. This is implemented using a prototype solver to demonstrate the promise of this approach. These contributions substantially expand the scope of bigraph modelling tools and their applications for modeling large-scale systems.

Acknowledgements

I would like to thank my supervisors, Michele Sevegnani and Ciaran McCreesh, for their continued and reliable support, guidance and feedback throughout my Ph.D journey, and helping me see things through to the very end—I very much look forward to continuing my work with you in the future. I would also like to particularly extend my gratitude to Blair Archibald, who was a huge help and a pillar of support at many crucial points during the project, and Phil Trinder for helping me get started in the early stages.

Thanks also to my friends Stephen, Vivian, Paul, Lochlann, Shaun, Benas and Emma, for helping me stay sane and providing a good laugh whenever I was in need of moral support, especially during the Covid pandemic years.

I would also like to thank Darin for keeping me motivated and making sure I stayed on pace, Jack for broadening my horizons with his sincerity and wisdom and his collaboration with me on my many crazy ideas, J.P. for his infinite patience and helpful advice during tough times, as well as Nick and Mike for always being there and sticking by me for all these years.

Finally, I am forever grateful to my family, Mum, Dad, Lewis and Kiara; without your unconditional love and support, I would not be where I am today.

[This project was supported by the EPSRC under a Doctoral Training Partnership (EP/R513222/1).]

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

Kyle Burns

Table of Contents

Chapter 1

Introduction

1.1 Motivation: Scope of Bigraph Systems

Bigraphs are a universal mathematical model which are used to represent both the spatial relationships of entities and their global interactions. First proposed by Milner [1], a bigraph G consists of two graph-based structures over the same set of vertices that we refer to as *entities*: the *place graph* G_P, a directed forest graph which describes the physical nesting of each entity relative to one other e.g. a person inside a room, and the *link graph* G_L, a hypergraph which defines non-spatial connections between entities e.g. a device connected to numerous other devices regardless of location. A bigraph is capable of modeling a variety of different systems, such as sensors in IoT devices [2, 3], network protocols [4, 5], molecular and biological reactions [6] and security for smart buildings [7, 8].

A pair of bigraphs can define a *reaction rule* $R : r \to r'$, which represents a valid substitution from a sub-bigraph $r \subseteq G$ to r' within the larger bigraph structure to represent a change in state in a process known as rewriting, i.e. moving a device from one room to another. An initial bigraph state and set of reaction rules make up the key components of a *Bigraph Reactive System* (BRS), which provides the means for modelling interactive and dynamic systems that change state as time progresses. In particular, the BRS is able to repeatedly apply rewriting operations to produce a *transition system*—a graph of unbounded size which represents all possible branching states of the initial bigraph which can be reached through the rewriting process.

In order to apply a reaction rule $R : r \to r'$ to a bigraph, the BRS must first find, through a matching algorithm, all instances of r that exist in G in order to perform the rewriting operation—this is known as the *bigraph matching problem*. This is an NP-complete operation which must be performed every time a reaction rule is applied in a BRS, which quickly becomes computationally expensive when modelling larger and more complex systems. Ef-

ficient matching and rewriting routines are essential for practical analysis of large models, such as traffic systems or power grids that may require hundreds or thousands of defined entities. The current state of the art bigraph modelling tool *BigraphER* [9] performs bigraph matching through a Boolean satisfiability (SAT) encoding [10], which generates an impractically large number of clauses as the scale of matching problems increase, and the inflexible nature of SAT also makes it difficult to build upon an existing encoding implementation to add support for extensions to the bigraph structure such as *bigraphs with sharing* [11] or *directed bigraphs* [12], which allow support for more complex model paradigms.

We make the observation that the bigraph matching problem can be viewed as a bigraph equivalent to the more well-known *subgraph isomorphism problem* (SIP), an NP-complete graph problem which determines whether a given pattern graph P exists as a subgraph within a larger target graph T—both problems provide two graph structures as an input, and return a mapping or set of mappings which represent any or all found instances of the smaller component within the larger structure. There exist many publicly available dedicated state of the art tools such as the constraint programming-based *Glasgow Subgraph Solver* (GSS) [13] which can efficiently solve SIP instances. We thus propose that a method can be developed to encode instances of bigraph matching into instances of SIP, where a dedicated subgraph solver will be able to solve the encoded version of the problem. Following from this, we propose that by employing state-of-the-art optimized graph tools to solve these encoded instances, we will see substantial improvements in performance compared to the existing SAT encoding method. The more complex bigraph-specific logic could either be encoded into the graph structures themselves, or as additional constraints on top of SIP.

In this dissertation, we present a novel bigraph matching algorithm implemented on top of the constraints-based GSS which is both performant and extensible. The link and place graphs of both input bigraphs are encoded and "flattened" together into a more conventional directed graph format, and passed to a modified version of the GSS which imposes additional constraints to handle the complexities introduced by *regions* and *sites* of bigraphs - abstract nodes which can represent any (including none) connected parent/child bigraph respectively, and define the rules for composing bigraph structures together. Via this method, we are also able to support matching instances for the directed and sharing extensions of bigraphs through the addition of further constraints without modifying the existing encoding structure, demonstrating the flexibility of this approach. This new version of the solver is integrated into the BigraphER framework and evaluated runtime-wise against the previously used SAT-based solver, on both isolated instances of the matching problem as well as when used for building the full transition systems of models for a variety of real-world use cases, which include both bigraphs and the bigraphs with sharing extension.

We also extend this research further to define a bigraph equivalent to the maximum common induced subgraph (MCIS) problem—a generalized variant of SIP which finds the most

optimal solution(s) rather than a fully satisfiable one—which we call the *maximum common bigraph* (MCB) problem. This finds the largest possible shared structure(s) that can exist as a component inside of two input bigraphs G_1 and G_2, constructed in such a way that there does not exist any non-trivial composition onto any solution which also produces a valid solution. This can be further extended to find the *minimal contextual transition* of a reaction rule upon an agent—the smallest composition required to turn a non-match into a match in a BRS. We achieve this through an adaptation of the *McSplit* algorithm [14], a partitioning backtracking algorithm for MCIS. A prototype implementation and preliminary performance metrics for both maximum common bigraph and finding a minimal contextual transition are provided.

1.2 Thesis Statement

I assert that better algorithms for performing search on bigraphs can be devised via encoding and flattening them into a more standard graph format and adapting reliable subgraph solving tools to operate on these encodings. There exists a way to perform this which captures the compositional constraints of bigraphs, and ensures a bijective mapping of solutions. This alleviates the current limitations of bigraph reactive system implementations caused by frequent necessary matching operations, and expands the scale and scope for modelling large and dynamic systems in a BRS.

1.3 Contributions

This dissertation makes the following research contributions:

1. **The design of a new algorithm for bigraph matching based on subgraph isomorphism and constraint programming** (Chapter 3). A critical review of previously established tools for solving bigraph matching was conducted (Chapter 2.5), evaluating their strengths and drawbacks of their approaches, taking note of the use of graph encoding techniques and constraint modelling toolkits. We then define a novel encoding (Section 3.2) and flattening (Section 3.3) function for bigraphs, which allows instances of the bigraph matching problem to be reduced to an instance of subgraph isomorphism (with extra constraints) which can then be given to and solved by a state-of-the-art subgraph isomorphism solver which will identify all (and only) correct matches corresponding to the original matching instance. We provide proofs of soundness and correctness of this algorithm (Section 3.5).

2. **The adaption of the bigraph matching algorithm to support extensions of bigraphs** (Chapter 4). We demonstrate the extensibility of our encoding by adding

support three variants of bigraph matching through optional extra constraints: bigraph equality checking which identifies when two bigraphs are functionally identical in a BRS (Section 4.1), bigraphs with sharing which generalizes the place graph's forest structure into a DAG (Section 4.2), and directed bigraphs which assign a direction property to hyperedge adjacencies (Section 4.3). We also provide the necessary proofs for these extensions.

3. **An implementation and extensive performance evaluation and analysis of the matching algorithm** (Chapter 5). We adapt the Glasgow Subgraph Solver to accept a bigraph format as input and apply the additional constraints required to reflect our encoding (Section 5.1), and demonstrate its performance to be be over two orders of magnitude more efficient when compared against BigraphER's MSAT algorithm, as well as outperforming MSAT on all 11,176 provided test instances (Section 5.2). This is then integrated into the BigraphER framework as the new primary solver and further evaluations are performed for when we use the solver to build a full model transition system (Section 5.3).

4. **The definition of, design and implementation of an algorithm for maximum common bigraph and finding minimal contextual transitions** (Chapter 6). We identify the use-cases for where a MCB algorithm is necessary (Section 6.1), such as the optimization, simplification and cost-reduction of processes through identifying bisimilar components. We then define what it means for a shared bigraph structure to be a maximum (Section 6.3), in a way which is consistent with both a conventional understanding of largest common graphs (Section 6.2) as well as the definition of maximality in the context of pushouts of bigraphs. We adapt the McSplit MCIS algorithm to support MCB through additional constraints and modification of the underlying data structures (Section 6.4), and provide proofs of soundness and correctness of this algorithm (Section 6.5). We then adapt this further to identify all partial mappings that can be matched through the addition of a minimal context (Section 6.6), and engineer a prototype solver (Section 6.7) and compare the relative performance of the two algorithm variants (Section 6.8).

1.4 Publications and Authorship

A preliminary version of the research, design, implementation and evaluation of the SIP-based algorithm for bigraph matching as well as bigraphs with sharing as described in Sections 3.1 to 3.4, Section 4.2 and Sections 5.1 and 5.2.1 have been previously reported in the publication:

- B Archibald, K Burns, C McCreesh, M Sevegnani, **Practical Bigraphs via Subgraph Isomorphism**, 27th International Conference on Principles and Practice of Constraint Programming, DOI:10.4230/LIPIcs.CP.2021.15, Published 2021 (Reference [15]).

The work described in this dissertation is primarily my own, with the exceptions of the following:

- The initial implementation of the Glasgow Subgraph Solver was the work of Ciaran McCreesh.
- The integration of the matching solver as a component in BigraphER as described in Section 5.3.1 was the work of Blair Archibald.
- The initial implementation of the McSplit Python script as described in Sections 6.7 and 6.8 was the work of James Trimble.

Chapter 2

Background

In this dissertation, we present a new constraint-based matching algorithm for bigraphs in order to perform efficient rewriting operations at scale. To give context to this research, this chapter introduces the background material on the fields of which our research builds upon by presenting the theory of general term rewriting models and graph transformation systems, as well as the fundamentals of bigraph reactive systems, constraint programming models and existing tools for bigraph matching.

In Section 2.1, we introduce term rewriting systems in the general case. Section 2.2 covers graph transformation systems, a specific type of rewrite system built upon graph transformations, as well as the underlying subgraph isomorphism algorithm. Section 2.3 introduces bigraphs and bigraph reactive systems, a more complex graph-based modelling paradigm which also builds upon rewriting systems, and the bigraph matching problem that must be solved to execute BRS operations. Section 2.4 defines the constraint programming paradigm, a popular method for solving complex NP-complete problems and how this can be applied to efficiently solve graph problems. Section 2.5 presents and discusses a selection of existing bigraph tools which aim to solve bigraph matching via various methods. Section 2.6 summarizes the chapter.

2.1 Term Rewriting

Rewriting broadly describes a fundamental type of operation in computing science and mathematics, in which a formula, sequence or structure S is transformed into another expression S' by removing and substituting a sub-expression l within S with a different sub-expression r, most commonly to simplify the expression or model a change in state [16]. A rewriting operation is performed according to a *rewrite rule* $R : l \longrightarrow r$, which defines a pair of sub-expressions—the left-hand side *redex* $l \subseteq S$, the expression within the larger structure

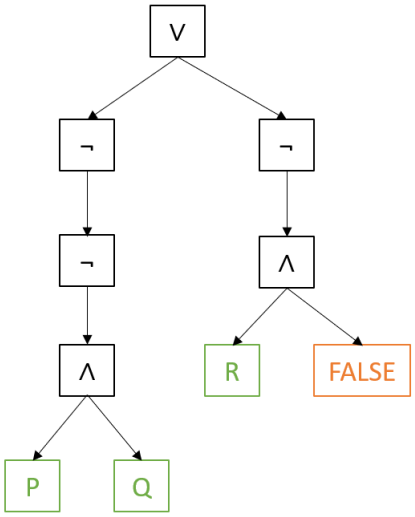


Figure 2.1: The boolean algebra equation $\neg(\neg(P \wedge Q)) \vee \neg(R \wedge \mathbf{false})$ represented as a term within a term rewriting system, where leaf nodes consist of variables and constants and non-leaf nodes represent logic operators which encapsulate their descendants.

to both find and then substitute, and the right-hand side *reactum* $r \subseteq S'$ which defines the new sub-expression that will be substituted in place of l when this operation is performed [17]. The generalizability of rewriting allows it to be applied to a variety of different contexts which involve sequences of substitutions, ranging from reducing and simplifying mathematical equations or logical expressions [18] to compiler optimization for programming languages [19].

More specifically, we are interested in *term rewriting*. To understand this, we initially define a *term* as an expression which satisfies either of the following two conditions:

- The expression is a single elementary variable or constant.
- The expression is a function, which takes an arbitrary number of terms as input.

Under this definition, a term can be treated as a tree-like structure where leaf nodes are variables and constants, and non-leaf nodes are nested combinations of functions which are recursively applied to the resultant values of their children. As a familiar example, both arithmetic equations and boolean algebra expressions can be represented in this form—an example is provided in Figure 2.1, where we demonstrate the visual representation of a boolean algebra equation as a term. Rewrite operations then substitute subgraphs in this form which match their corresponding redex term's tree representation, without altering the rest of the structure.

2.1.1 Rewriting Systems

Rewriting systems consist of an initial term or expression, and a set of rewrite rules which specify what possible sequences of transformations can be made for each state transition step. These are most commonly used for the purpose of either simplifying the expression, or allowing it to gradually evolve in structure over time i.e. modelling a simulation [20].

We are only concerned with *term rewriting systems* (TRS) specifically, and thus assume that we are only operating on terms going forward. In arithmetic contexts, equivalences such as $x * 0 = 0$ or $x * 1 = x$ would be represented as rewrite rules, whereas for boolean algebra these would represent algebraic laws such as the double negation law or De Morgan's law. Rewrite rules are commutative in many instances, particularly in the given examples where they represent logical or mathematical equivalences i.e. $(R : l \longrightarrow r) \implies (R^{-1} : r \longrightarrow l)$, however equivalence is not always guaranteed in the general case. As a trivial example, there can be a rule that allows for the substitution of one element with another within a data structure or graph (Section 2.2) where there is no equivalent inverse rule.

A simplified example of the TRS process is shown in Figure 2.2. Here, the TRS attempts to find whether the redex of each defined rewrite rule l_k exists within S. If it does, it will then rewrite S by substituting the found instance with r_k. This process then repeats until either no redex of any of the rewrite rules exist within S i.e. no more rewriting operations can be performed, or a user-defined stopping condition has been met, such as after a certain number of total rewrites or the structure reaching a certain size or normal form. It is crucial to note that the TRS does not actually define the specific algorithm used for finding the required redex matches and altering the structure: the TRS itself only defines the rules for when a substitution from one subterm to another can be allowed to take place. The underlying necessary method for performing redex searching and term rewriting is instead treated as a separate context-dependent component to be supplied by the developer and combined with the TRS framework. The TRS treats this given algorithm as a "black box", where the term S and rewrite rule R_K are provided as input to the matcher/rewriter, and the rewritten term S' is provided as output if l_k occurs in S. Algorithms for performing matching in more specific contexts are further explored for applications of TRSs in Section 2.2 and Section 2.3.

TRSs can also be non-deterministic: whereas the given TRS example iterates through each rewrite rule in turn, it is also possible for rewrite operations to be applied in any order to S as long as the corresponding rule is valid in its current state. The same rewrite rule can also potentially be applied in a multitude of different ways to the same term if there exist multiple non-isomorphic valid matches of l_k within S for some R_k—that is, they produce distinct non-isomorphic variations of S'.

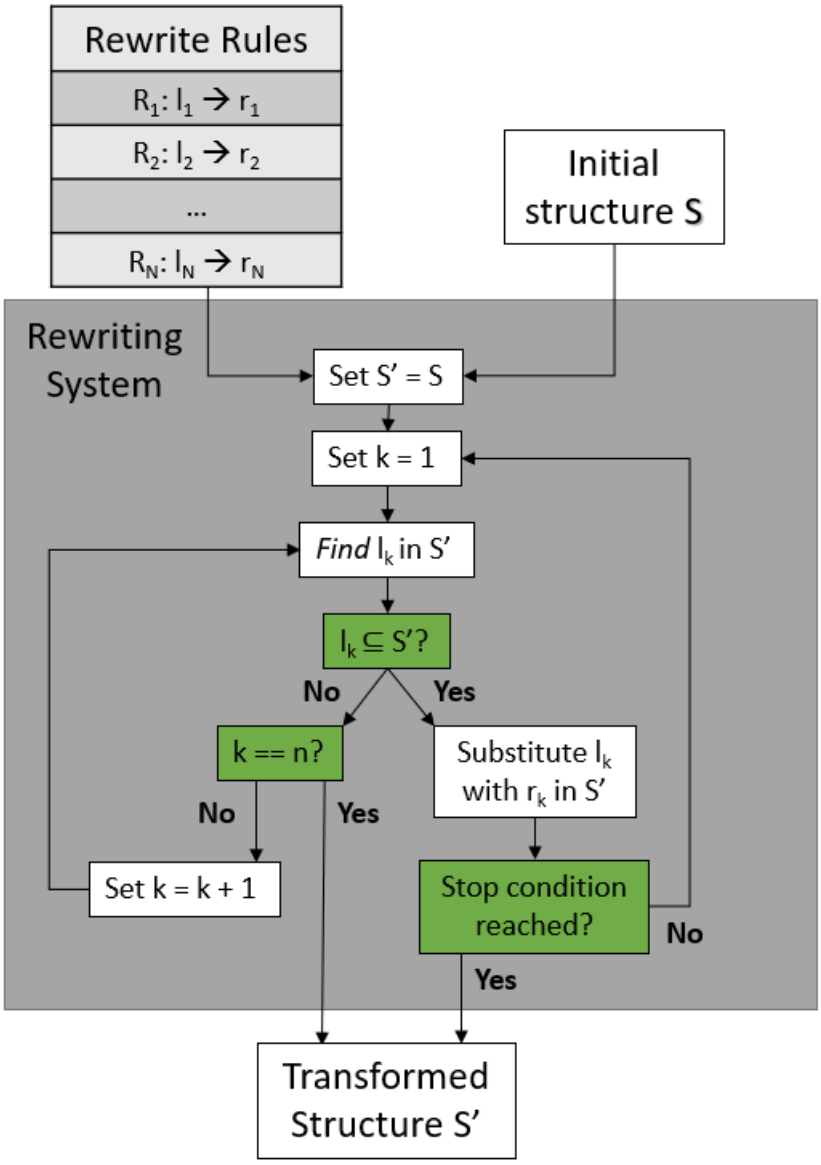


Figure 2.2: A simple implementation of a term rewriting system [21]. The set of rewrite rules are repeatedly applied to S until either no more rewrites are possible or a stopping condition has been reached.

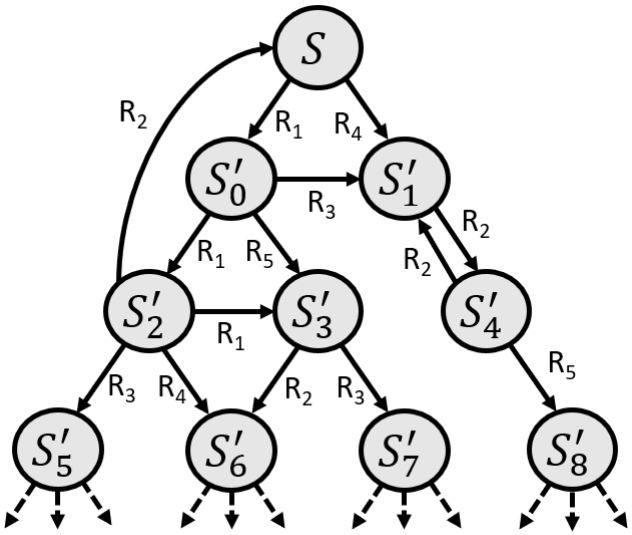


Figure 2.3: Example of a transition system with initial state S, rewrite rules $R = \{R_1, ..., R_5\}$ and a bounded depth of 3. To further simplify this example for demonstration purposes, it is assumed no cases exist here where the same rule can be applied more than once to the same state to produce different transformations.

2.1.2 Transition Systems

While a TRS—alongside a supplementary matching and substitution algorithm—is mainly concerned with transforming a given term into another through the application of rewrite rules, it is also possible to use a similar paradigm with the same initial state and set of rules to map out all the different possible reachable states of S in what we define as a *transition system*. This is primarily used in model checking and verification contexts to simulate all the different ways a structure can potentially change over time [22].

A transition system can be represented as an arbitrary directed graph of term states as shown in Figure 2.3, where edges represent the transformation of a state via an application of a valid rewrite rule. A transition system is expanded from a given vertex by applying each rewrite rule to its corresponding term and adding a new child vertex for every discovered valid match and its resultant transformation. Because of this, a transition system has a theoretically infinite size as long as the rewrite rules allow it to grow indefinitely, and thus in practice we normally bind these by only generating up to a specified depth or total number of states found.

Two key observations can be made: firstly, graph cycles often exist in transition systems due to the potential for a sequence of rewrite rules to generate a state which is isomorphic to one which already exists in the graph—i.e. a formally equivalent structure, such as applying the sequence $R_1 \rightarrow R_1 \rightarrow R_3$ to S in the given example which results in returning back to the initial state. Secondly, different rewriting sequences can also potentially lead to the same

state, such as $R_1 \rightarrow R_1 \rightarrow R_4$ and $R_1 \rightarrow R_5 \rightarrow R_2$ both leading to state S'_6. These cases both introduce the need to perform a term isomorphism check on a new state against all other current term states that exist in the transition system whenever attempting to expand it, i.e. determining if a state already exists which is isomorphic to a resultant candidate state after rewriting and substitution. If so, a new edge is drawn from the current state to the previous state instead of adding a new vertex to the graph. This however has the potential to introduce scaling issues when expanding the transition system to large depths, since the number of isomorphism checks required to add a new candidate state grows proportionally to the size of the graph, particularly in model checking contexts where determining equality between two terms is non-trivial due to the data structures used [23].

2.2 Graph Transformation Systems

We now look at a more specific type of application of this paradigm, which operates on graph based structures rather than terms. A graph transformation system (GTS) is a generalization of TRSs which substitutes terms for graphs; that is, a GTS defines an initial graph G and a set of rewriting rules, each in the form of a pattern graph and replacement graph pair $R : l \longrightarrow r$, which specify how G can be transformed in discrete sequences of steps [24]. Similarly to TRSs, GTS rewrite operations are performed by first finding an occurrence of the pattern graph l as a subgraph within G—known as pattern matching in this context—before then substituting l with an instance of r inside G. A *subgraph isomorphism* algorithm is required to perform the preliminary step of pattern graph matching. GTSs can also support labelled and directed graphs in both the initial state G and rewrite rule pair l and r as long as the underlying matching algorithm also supports these additional features [25]. As graphs are a widely used formalism for modelling a variety of systems and relationships between entities, many dedicated GTS building tools have been developed such as *GROOVE* [26] and *AGG* [27], and there also exist many dedicated GTS-based tools for specific applications such as software engineering tooling [28], biological models [29] and natural language processing [30].

The algebraic method of formalizing graph rewriting is built upon *category theory* [31]—a mathematical paradigm which relates to sets of objects and the relationships between them, which are referred to as *morphisms*. Through this, the process of subgraph substitution is performed using *pushout* operations such as the single pushout (SPO) or double pushout (DPO) approach. We are more concerned with the application of the necessary matching algorithm for GTSs rather than the rewriting operation itself, so we do not cover category theory for graphs for the remainder of this dissertation.

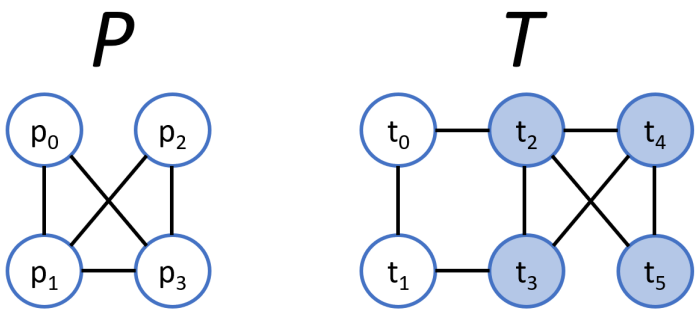


Figure 2.4: A simple instance of SIP. The colored vertices show visually the single match of P inside T, subject to the mapping $\{(p_0, t_5), (p_1, t_4), (p_2, t_3), (p_3, t_2)\}$.

2.2.1 Subgraph Isomorphism

So far, we have taken it for granted that the underlying matching algorithms for performing rewriting are already provided for us "under the hood". However, as mentioned previously, rewriting system models do not actually supply this algorithm to the developer, and these can be potentially difficult and computationally complex operations which can cause scaling issues for larger models. Here we take a closer look at subgraph isomorphism, the algorithm for determining whether a rewrite rule can be applied to a GTS state.

The Subgraph Isomorphism Problem (SIP) is a classic NP-complete decision problem that determines whether some pattern graph $P = (V_P, E_P)$ is a subgraph of (i.e. is present in) some target graph $T = (V_T, E_T)$. More specifically, we aim to find an injective *mapping* from P to T, where each vertex $p \in P$ is assigned to a vertex $t \in T$ such that for all edges $e_p = (p_1, p_2) \in E_P$, there is a corresponding matching edge $t_p = (t_1, t_2) \in T_P$. Whilst we are mainly concerned with SIP in a model checking context, SIP is a very generalizable algorithm which is used in a wide variety of contexts such as bioinformatics [32], computer vision [33], compilers [34] and program code similarity [35]. A simple example of SIP is shown in Figure 2.4.

When performing SIP as part of a GTS, it is insufficient to halt upon finding a single match of P within T: instead, we want to enumerate *all* instances of P in T, as it is possible more than one exists, and we wish to compute all possible changes in state for the application of a rewrite rule or the building of a transition system. Formally, the problem that SIP wishes to solve is as follows.

Definition 2.2.1 (Subgraph isomorphism problem)**.** Given an input pattern graph $P = (V_P, E_P)$ and target graph $T = (V_T, E_T)$, a valid subgraph isomorphism solution is an injectively mapping function $f : P \to T$ where edges are mapped to edges, such that $(u, v) \in E_P \implies (f(u), f(v)) \in E_T$. We wish to enumerate *all* such mappings.

There also exist many extensions and variations of SIP to accommodate more complex use-

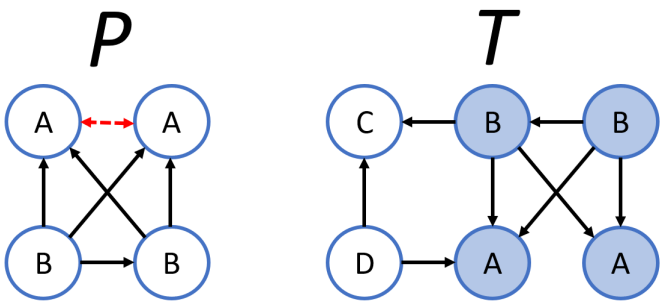


Figure 2.5: A simple instance of induced SIP which mirrors that of Figure 2.10, which also incorporates directed edges and vertex labelling. The non-edge in P is shown as a red dashed bidirectional edge.

cases. For example, *induced* SIP enforces that non-edges between vertices in P must also map to non-edges in T, as depicted as a red dashed edge in Figure 2.5 [36]. SIP can also support directed graphs, where directed edges adjacent to each pattern-to-target mapping pair must both be facing the same direction. We are also interested in *labelled* SIP, where each vertex is assigned a type similarly to bigraph entities (Section 2.3), and each vertex mapping $p \in P$ to $t \in T$ must have $label(p) = label(t)$. Figure 2.5 shows a simple example of SIP which incorporates these additional properties.

We are interested in combining these properties to establish a common behavior between SIP and bigraph matching as described later in Chapter 3, and thus we formally define this variant of SIP as follows:

Definition 2.2.2 (Induced, labelled, directed subgraph isomorphism problem)**.** Given an input pattern graph $P = (V_P, E_P)$, target graph $T = (V_T, E_T)$, and a vertex compatibility function $\ell : V_G \times V_H \to \{t, f\}$, an *induced subgraph isomorphism with vertex compatibility constraints* from G to H is an injective mapping $f : P \to T$ such that edges are mapped to edges and non-edges are mapped to non-edges i.e. $(u, v) \in E_G \iff (f(u), f(v)) \in E_H$, and where vertex compatibility is respected, i.e. $\forall v \in V_P \to \ell(v, f(v)) = t$. We wish to enumerate *all* such mappings.

It can also be observed that when $|P| = |T|$, the desired mapping becomes bijective, and thus becomes the *graph isomorphism* problem. In this case, which of the two given graphs are designated as the pattern graph P and target graph T becomes insignificant; their designations can be switched and SIP will still find the same solution. When implementing a transition system for graphs, isomorphism checking on each new candidate graph state is an additional required operation that must be performing against each existing state in the system, in order to determine whether we wish to create a new transition system node (the candidate graph state is unique) or simply draw a new edge to a previous node (isomorphic to a previously found state).

Because of its broad applicability, many dedicated solving tools for SIP exist. Solnon [37] provides an experimental evaluation of four well-known subgraph solvers, where it is found that the *Glasgow Subgraph Solver* [13] (GSS) provides the best runtime performance for solving "harder" instances of SIP - that is, instances which are computationally challenging even with smaller numbers of nodes in the pattern and target graphs [38]. This solver models SIP as a *constraint satisfaction problem*, and uses an informed backtracking search heuristic with bit-parallel data structures and propagation algorithms which are optimized specifically for SIP to ensure optimal performance. GSS also supports our required extensions of SIP including induced SIP, directed SIP and a labelling function for defining vertex and edge compatibilities, and is also able to explicitly enumerate all solutions or count the total occurrences of a solution rather than deciding whether a single solution exists. Subgraph isomorphism in the context of constraint programming is covered further in Section 2.4.

2.3 Bigraph Reactive Systems

Bigraph reactive systems (BRSs) are a universal mathematical model first proposed by Milner [1], which simultaneously model systems based on both spatial and non-local relationships between entities. Bigraph systems are constructed similarly to TRSs and GTSs, with the exception that the underlying structure and rewrite rules are composed of *bigraphs*—a two-tier graph like structure containing both directed edges and *hyperedges* between vertices. Unlike standard graphs, bigraphs are also subject to unique rules for construction, composition and matching which adds further complexity to rewriting operations and transition system building. To fully understand BRSs, we first begin by sufficiently defining bigraphs, their components and terms, as well as relevant operations that can be performed on bigraphs.

2.3.1 Overview

Instances of bigraphs can be represented both algebraically and diagrammatically, however in this dissertation we will be primarily describing them in their diagrammatic form as presenting them in a visual format allows them to be described more intuitively. A bigraph instance in this form can be understood as the overlapping of two graph based structures that share the same vertex set, as demonstrated in Figure 2.6. The three main components which make up a bigraph are described as follows:

- **Entities**: Vertices of the bigraph, representing agents, objects or devices in the model.

Each entity has an associated type known as a *control*, represented as the A, B and C values in Figure 2.6. The set of controls in a BRS is known as the *signature*.

- **The place graph**: A directed forest graph as shown in Figure 2.6 (b), which captures spatial and physical relationships between entities e.g. a device inside a room. In the standard bigraph diagrammatic form as shown in Figure 2.6 (a), the place graph is represented as the encapsulation of parent entities around their children. An entity is considered *atomic* if it is a leaf node in the place graph. Top-level places, shown as dashed rectangles, are called *regions*. Similarly, the grey rectangles inside entities are called *sites*. These abstract region and site nodes, each distinguished by integer ordinals incrementing from zero, represent abstractions of unknown (or empty) parent and child place graph structures respectively, and denote where the composition of another place graph can be allowed to take place.

- **The link graph**: An undirected hypergraph as shown in Figure 2.6 (c), which captures non-spatial relationships between entities e.g. a wireless network of devices. Each control has an *arity*, a non-negative integer value which defines its number of *ports*, which can be viewed as "sockets" for the link graph. An entity with a control of arity value n thus must always have n link graph adjacencies. A shared connection between a set of entities is called a *link*. Equivalently to sites and regions, the link graph contains *outer names* and *inner names*, represented in the bigraph diagrammatic form as links extending above and beneath the bigraph structure. These represent adjacencies to unknown (or empty) parent and child link graphs. A link is *open* if it connects to an outer name, and *closed* otherwise i.e. it connects exclusively between entities and inner names. Closed links in a concrete bigraph are given unique identifiers similarly to sites and regions. In the example link graph, it can be observed that the arity of controls A and B are 1, and the arity of C is 2. There exists a single open link adjacent to the outer name x, and there are three closed links: e_0 and e_1 which connect between entities and e_2 which connects to inner name y.

The set of sites m and inner names X, when taken together make up the *inner face* $\langle m, X \rangle$ of a bigraph. Similarly, the set of regions and n and outer names Y denote the bigraph's *outer face* $\langle n, Y \rangle$. The inner and outer faces together describe the *interface* $B : \langle m, X \rangle \to \langle n, Y \rangle$ of a bigraph B. The sets of sites and regions are labelled using an ordered set of non-negative integers, i.e. $m = 3$ indicates that the set of regions is $\{0, 1, 2\}$. The interface of the example bigraph in Figure 2.6 is $B : \langle 2, \{y\} \rangle \to \langle 2, \{x\} \rangle$

Sites, regions and names act like "sockets" which allow other bigraph structures to "join" together to build a larger combined bigraph structure, by connecting the sites and inner names of the above bigraph with the regions and outer names of the below structure to produce

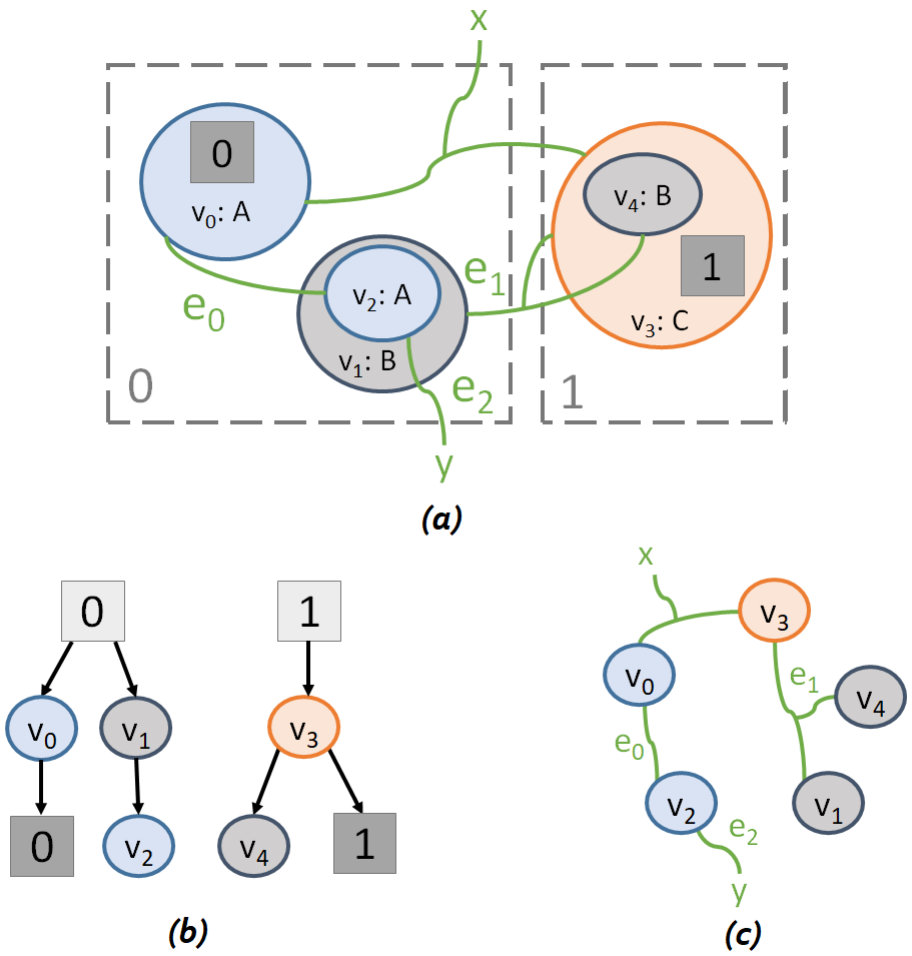


Figure 2.6: (a) Example of a *concrete* bigraph with entities, their controls and their spatial and non-spatial relationships. (b) Place graph for (a); (c) Link graph for (a). Colors denote control type.

edges and links between the two. This is only possible however when the inner and outer faces being linked together are compatible with one another, i.e. the number of inner names and sites on the above bigraph's inner face matches the number of outer names and regions of another bigraph's outer face respectively. A bigraph is considered *idle* in the place graph if there exist no nodes or sites, and idle in the link graph if there exist no ports or inner names. The compositional behavior of bigraphs is covered in further detail in Section 2.3.3.

The bigraph shown can also be described as a *concrete* bigraph. This term is used to describe bigraphs where all their component entities are assigned unique identifiers $v_0, v_1, ..., v_n$, and each closed link is also given an identifier $e_0, e_1, ..., e_m$. Taken together, these vertex and edge identifiers constitute the *support* of a concrete bigraph. We denote bigraphs lacking a support as *abstract* bigraphs. Two bigraphs are considered *support equivalent* as long as their abstract forms are isomorphic, i.e. have the same structure and only differ by name assignment.

2.3.2 Bigraph Definitions

Now that we have informally described the core components of a bigraph, we now proceed to define them via more formal means as provided by Milner [1]. We are mostly concerned with concrete bigraphs as part of this dissertation, and thus we begin by providing the definition of bigraphs in their concrete form.

Components

Initially, we establish some conventional notations used when defining bigraphs: firstly, the symbol $A \uplus B$ is used to denote a union of sets that are known to be disjoint. Secondly, the labels used for control names, entity identifiers and closed-link identifiers all belong to the disjoint infinite sets $\mathcal{X}$, $\mathcal{V}$, and $\mathcal{E}$ respectively [11]. In order to formally define each component of a bigraph, we first define the signature.

Definition 2.3.1 (Signature)**.** A *signature* $(\mathcal{K} : ar)$ defines the set of controls $\mathcal{K}$ in a bigraph and each of their corresponding mappings $ar : \mathcal{K} \to \mathbb{N}$ to a non-negative arity value. A bigraph over $\mathcal{K}$ assigns every entity a control $k \in \mathcal{K}$ which in turn assigns that entity $ar(k)$ link ports.

We then build upon this definition to define the concrete place and link graphs, and subsequently the full concrete bigraph structure.

Definition 2.3.2 (Concrete Place Graph)**.** A *concrete place graph*

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

is a triple which has the inner face m and outer face n, indicating m sites and n regions. 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 map*

$$prnt_B : m \uplus V_B \to V_B \uplus n$$

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 2.3.3 (Concrete link graph)**.** A *concrete link graph*

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

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 $P_B \stackrel{\text{def}}{=} \{(v, i) \mid v \in V_B,\ i = ar(ctrl_B(v))\}$ 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*. In addition, *idle edges* are links where the domain is restricted to $\emptyset$, i.e. have no source to point from.

A concrete bigraph is the combination of a concrete place graph and concrete link graph that each share the same entity set V_B.

Definition 2.3.4 (concrete bigraph)**.** A concrete bigraph

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

consists of a concrete place graph $B^{\mathsf{P}} = (V_B, ctrl_B, prnt_B) : m \to n$ 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 $\langle m, X \rangle$ and $\langle n, Y \rangle$, respectively. The support size $|B|$ of the concrete bigraph is $V_B \uplus E_B$.

A full example of the deconstruction of the given bigraph example in Figure 2.6 into this form is provided in Appendix A.

Equivalence

We also formally define the notion of support equivalence for bigraphs. This is key for understanding what constitutes as "equality" between bigraphs (Section 4.1), particularly for when we wish to discern equivalent bigraphs in a transition system, as well as for retaining compositional information between the two resultant parts of a decomposed bigraph. We

are only interested in support equivalence between bigraphs with no idle edges however, as these cannot be sufficiently mapped without having a source. We hence define *lean* bigraphs as those with no idle edges, and define *lean-support equivalence* as follows.

Definition 2.3.5 (lean-support equivalence)**.** Two concrete bigraphs A and B are lean-support equivalent, denoted as $A \simeq B$, when there exists a valid *support translation* $\rho : |A| \to |B|$ from A to B after discarding idle edges. A support translation ρ consists of a pair of bijections $\rho_V : V_A \to V_B$ and $\rho_E : E_A \to E_B$ from A to B's vertex and closed edge sets respectively, such that the structure of both bigraphs are maintained as follows:

- Controls are maintained, i.e. $ctrl_B \circ \rho_V = ctrl_A$—this also produces a bijection $\rho_P : P_A \to P_B$ on the ports of the corresponding entities.
- Parent and link relations are maintained, i.e.

  $prnt_B \circ (\mathtt{Id}_m \uplus \rho_V) = (\mathtt{Id}_n \uplus \rho_V) \circ prnt_A$,

  $link_B \circ (\mathtt{Id}_X \uplus \rho_P) = (\mathtt{Id}_Y \uplus \rho_E) \circ link_A$,

  where $\mathtt{Id}_m$ is the identity function of m, and so on for the other interface components.

More informally, we can imagine two bigraphs to be lean-support equivalent if they have the same structure and only differ via the component identifiers of their supports. In other words, if $A \simeq B$, then we can simply produce B through renaming the vertices and edges of A via a support translation. An important observation is that support equivalence does not permit bijections between differently named interface components, as this would no longer retain the original bigraph structure between interfaces after a decomposition—that is, attempting to recompose two component bigraphs after a reordering of the interface assignments would cause them to join different vertices together and result in a bigraph which is not equivalent to the original state, and hence the support translation itself must be producing a bigraph which is not equivalent to its pre-image.

An *abstraction* of a concrete bigraph describes the result of discarding its support in order to produce an abstract bigraph with the same structure. Conversely, the *concretization* of an abstract bigraph describes when a support is added to produce a concrete form of the bigraph. It can be easily deduced that the abstraction of two support equivalent bigraphs will produce an isomorphic structure.

Symmetries

We are also interested in distinguishing between different types of *symmetry* that can occur in bigraphs. Two place graphs (and hence bigraphs) are considered *symmetrical* if they

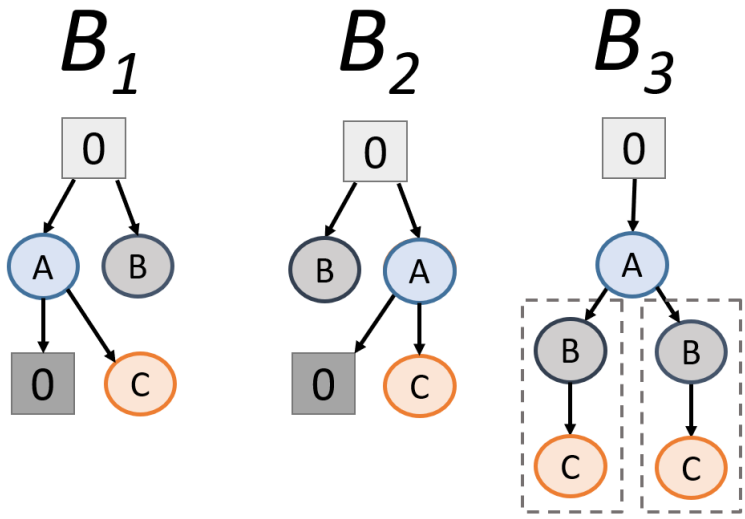


Figure 2.7: Instances of symmetry in place graphs. Bigraphs B_1 and B_2 are symmetrical, as they only differ in structure from the swapped ordering of entities A and B beneath the region. Bigraph B_3 contains an intra-symmetry between the descendants of A.

are support equivalent, but differ in the left-to-right orderings of children for a selection of regions and entities, as shown between bigraphs B_1 and B_2 in Figure 2.7. Symmetrical bigraphs when described algebraically are identical, as entity relations are only defined via $prnt$ and do not take into account the diagrammatic ordering of siblings.

A different categorization of symmetry can also occur between entities within a single place graph, when there exists a support translation between sibling vertices and their descendants as shown in place graph B_3 in Figure 2.5. We denote these occurrences as *intra-symmetries* going forward, to disambiguate them from instances of symmetry between independent bigraphs. Bigraphs containing intra-symmetries will have multiple possible valid support translations for mapping to a support equivalent bigraph, as the subset of vertex assignments between intra-symmetric entities can be swapped while preserving structure.

2.3.3 Bigraph Construction

Bigraphs follow a specific formalism for allowing composition onto one another through their interfaces. There are two key methods of combining two bigraphs A and B in order to create a larger merged structure G: one is to simply place them side-by-side to produce what we call the *tensor product* of two bigraphs $G = A \otimes B$, as shown in Figure 2.8. The other method involves placing B under A, and merging the regions of B with the sites of A in their place graphs to produce connecting edges between the two, and similarly connecting the inner links of A with the outer links of B in their link graphs to produce the *composition* $G = A \circ B$ of the two bigraphs A and B as shown in Figure 2.9.

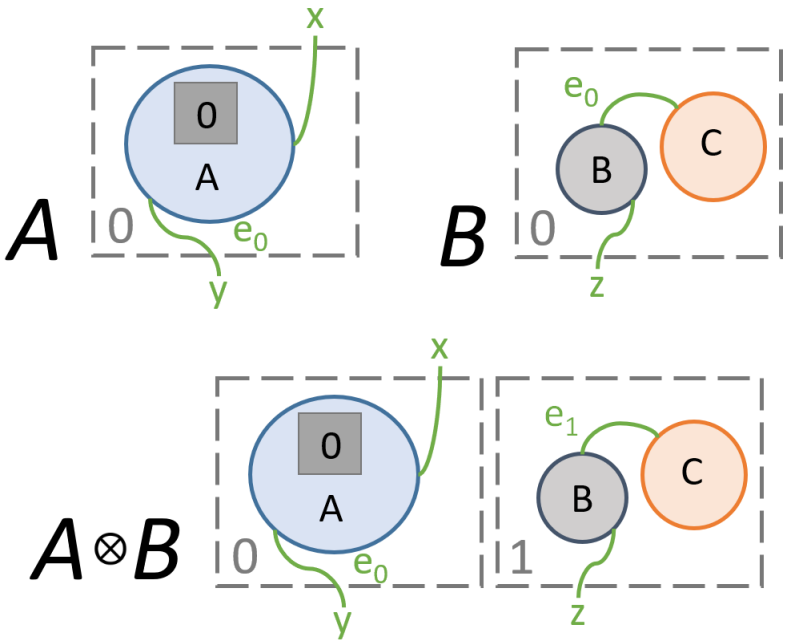


Figure 2.8: A diagrammatic example of the tensor product $A \otimes B$ of two bigraphs A and B.

Tensor Product

The tensor product operation on two bigraphs is the simultaneous independent application of the tensor product on their underlying place and link graphs, and hence we begin by defining these separately.

Definition 2.3.6 (Tensor product of concrete place graphs)**.** Given two place graphs $A = (V_A, ctrl_A, prnt_A) : k \to l$ and $B = (V_B, ctrl_B, prnt_B) : m \to n$, the tensor product

$$G = A \otimes B : k + m \to l + n$$

is defined as

$$G \stackrel{\text{def}}{=} (V_A \uplus V_B, ctrl_A \uplus ctrl_B, prnt_A \uplus prnt'_B),$$

where $prnt'_B(k + i) = n + j$ whenever $prnt_B(i) = j$.

Definition 2.3.7 (Tensor product of concrete link graphs)**.** Given two disjoint link graphs $A = (V_A, E_A, ctrl_A, link_A) : X \to Y$ and $B = (V_B, E_B, ctrl_B, link_B) : Z \to W$, the tensor product

$$G = A \otimes B : X \uplus Z \to Y \uplus W$$

is defined as

$$G \stackrel{\text{def}}{=} (V_A \uplus V_B, E_A \uplus E_B, ctrl_A \uplus ctrl_B, link_A \uplus link_B)$$

Using these separate definitions, we can now define the full tensor product operation as follows.

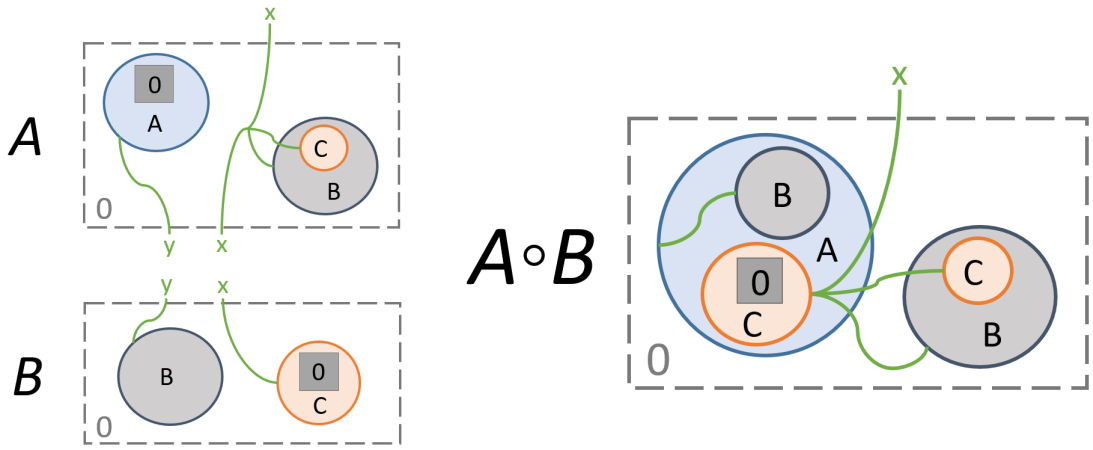


Figure 2.9: Example bigraphs $A : \langle 1, \{x, y\}\rangle \to \langle 1, \{x\}\rangle$, $B : \langle 1, \{\}\rangle \to \langle 1, \{x, y\}\rangle$, and the resultant bigraph $A \circ B : \langle 1, \{\}\rangle \to \langle 1, \{x\}\rangle$ when composed together. It can be observed in this diagrammatic form that the entirety of B now exists where the lone site of A used to reside.

Definition 2.3.8 (Tensor product of concrete bigraphs)**.** Given two disjoint bigraphs $A : \langle k, X\rangle \to \langle l, Y\rangle$ and $B : \langle m, Z\rangle \to \langle n, W\rangle$, the tensor product

$$G = A \otimes B : \langle k + m, X \uplus Z\rangle \to \langle l + n, Y \uplus W\rangle$$

is defined as

$$G \stackrel{\text{def}}{=} \langle A_P \otimes B_P, A_L \otimes B_L\rangle$$

for the place graphs A_P, B_P and link graphs A_L, B_L of A and B respectively.

Importantly, the tensor product operation is only applicable when there is no overlap in the names of vertices or interfaces of the two component concrete bigraphs A and B—that is, the interfaces and supports of A and B are fully disjoint.

Composition

The composition of two bigraphs is only possible if the inner face of A is equal to the outer face of B, i.e. $A : \langle m, X\rangle \to \langle n, Y\rangle$ and $B : \langle o, Z\rangle \to \langle m, X\rangle$. When composed, the resultant bigraph's interface will be $A \circ B : \langle o, Z\rangle \to \langle n, Y\rangle$. An example of a composition being performed is shown in Figure 2.9. Decompositions are also possible within the context of bigraphs, where a single bigraph can be decomposed into two smaller bigraphs by applying this same operation in reverse, splitting apart links and place graph edges and replacing them with inner/outer names and region/site pairs accordingly.

The composition operation for bigraphs as a whole can be considered as the simultaneous compositions of each bigraph's place and link graph components. As such, to formally define

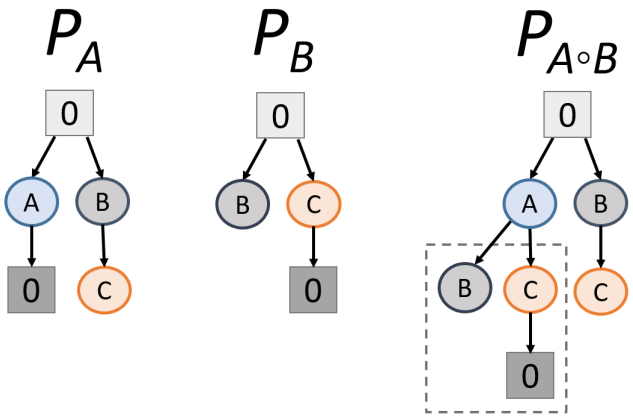


Figure 2.10: A composition example of two place graphs. The highlighted region in $P_{A\circ B}$ shows where P_B was joined into P_A in the resultant composition.

composition, we first define these separately.

Definition 2.3.9 (Concrete place graph composition)**.** Given two concrete place graphs $A : m \to n$ and $B : o \to m$ with disjoint supports, we define the composite place graph as

$$A \circ B = (V, ctrl, prnt) : o \to n$$

Where $V = V_A \uplus V_B$, $ctrl = ctrl_A \uplus ctrl_B$, and the $prnt$ value of each site and entity $v \in o \uplus V$ depends on the following conditions.

$$prnt(v) \stackrel{\text{def}}{=} \begin{cases} prnt_B(v) \text{ if } v \in o \uplus V_B \text{ and } prnt_B(v) \in V_B \\ prnt_A(r) \text{ if } v \in o \uplus V_B \text{ and } prnt_B(v) = r \in n \\ prnt_A(v) \text{ if } v \in V_A \end{cases}$$

The composition for the place graph components of the given bigraph examples are shown in Figure 2.10.

Definition 2.3.10 (Concrete link graph composition)**.** Given two concrete link graphs $A : X \to Y$ and $B : Z \to X$ with disjoint supports, we define the composite link graph as

$$A \circ B = (V, E, ctrl, link) : Z \to Y$$

where $V = V_A \uplus V_B$, $E = E_A \uplus E_B$, $ctrl = ctrl_A \uplus ctrl_B$, and the $link$ map value for each

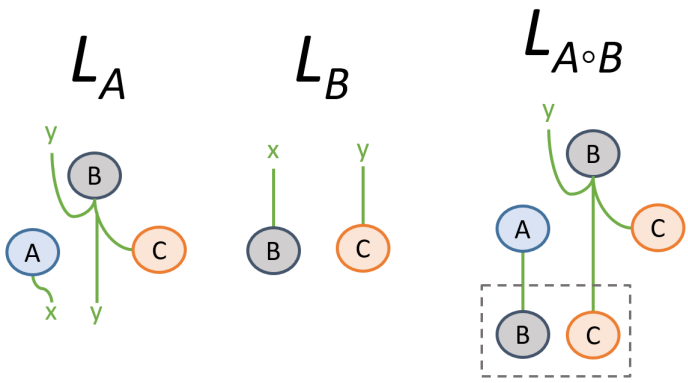


Figure 2.11: A composition example of two link graphs. The highlighted region in $L_{A\circ B}$ shows where L_B was joined into L_A in the resultant composition.

port and inner name $p \in Z \uplus P_A \uplus P_B$ depends on the following conditions.

$$link(p) \stackrel{\text{def}}{=} \begin{cases} link_B(p) \text{ if } p \in Z \uplus P_B \text{ and } link_B(p) \in E_B \\ link_A(o) \text{ if } p \in Z \uplus P_B \text{ and } link_B(p) = o \in Y \\ link_A(p) \text{ if } p \in P_A \end{cases}$$

The composition for the link graph components of the given bigraph examples are shown in Figure 2.11.

2.3.4 Bigraph Rewriting

Now that we understand how bigraphs can be composed and decomposed, we now look toward defining the rewriting rules of bigraphs, which act as the fundamental underlying operation of bigraph reactive systems. Similarly to rewriting rules for term rewriting and GTSs, bigraphs can have *reaction rules* $R : r \to r'$, a bigraph pair which defines the potential for a substitution of a bigraph r with the bigraph r' inside some larger bigraph structure G. The bigraphs which define a reaction rule also have both underlying place and link graph components, which are applied simultaneously in one step during the substitution operation. In order to be able to apply the reaction rule $r \to r'$ to G however, it must first be determined whether r *occurs* in G. This introduces the *bigraph matching* problem, a computational task that aims to find either any or all occurrences of a smaller bigraph inside a larger one, which is covered in further detail in Section 2.3.5. To begin with, we assume that an algorithm to solve this is provided for us.

We formally specify an *occurrence* as follows.

Definition 2.3.11 (Concrete occurrence)**.** Given two concrete bigraphs A and B, it is said

that B occurs in A if there exists some *context* bigraph C and *parameter* bigraph D such that

$$A = C \circ (B \otimes id) \circ D,$$

where id is the identity bigraph—an entity-free bigraph instance which only contains abstractions and faces that directly connect from $\langle m, X \rangle$ to $\langle n, Y \rangle$, allowing for place graph edges and link graph hyperedges to connect directly between C and D where necessary to ensure interface compatibility.

To first be able to define a reaction rule, we first restrict the connections to/from interfaces that can occur in our formal definition of a bigraph. This is because specific abstract configurations may allow for theoretically infinite occurrences in an instance, i.e. a region pointing to a site in the smaller bigraph. We thus initially define solid bigraphs.

Definition 2.3.12 (Solid bigraph)**.** A bigraph G is solid if it meets the following six criteria, which prevent idle interface components and connections between them:

- For all regions $o \in n$, there exists an entity $v \in V_G$ such that $prnt(v) = o$.
- For all outer faces $y \in Y$, there exists an entity port $\{p_v | v \in V_G\}$ such that $(p_v, y) \in link_G$.
- For all pairs of sites $s, t \in m$, $prnt(s) \cap prnt(t) = \emptyset$.
- There are no two inner names $x, y \in X$ such that $(x, e), (y, e) \in link_G$ for some source e.
- There is no $o \in n$, $s \in m$ such that $prnt(s) = o$.
- There are no two names $x \in X$, $y \in Y$ such that $(x, y) \in link_G$.

For the application of a reaction rule, r and G are always solid. Following on from this, we can formally define a bigraph reaction rule as follows.

Definition 2.3.13 (Bigraph reaction rule)**.** A reaction rule $R : r \to r'$ consists of a pair of bigraphs—the solid redex bigraph r and reactum bigraph r', which permits the substitution of an occurrence of r with r' in some host bigraph $G = C \circ (r \otimes id) \circ D$ to produce the new bigraph state $G' = C \circ (r' \otimes id) \circ D$, for some context bigraph C and parameter bigraph D.

A *ground* reaction rule $R_g : r_g \to r'_g$ is one which specifically applies to a *ground bigraph* state, which we define as a bigraph with no inner face i.e. $B_g : \langle \{\}, \{\} \rangle \to \langle n, Y \rangle$. In this case, it follows that the parameter D must also be ground.

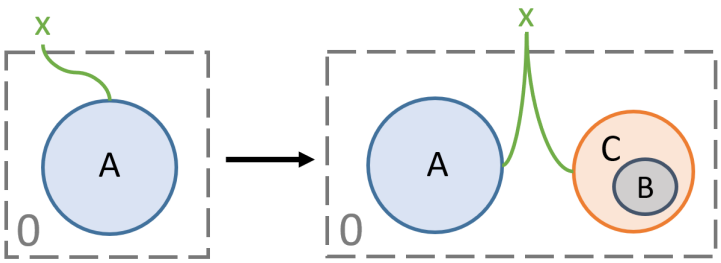


Figure 2.12: A reaction rule $r \rightarrow r'$ within a BRS.

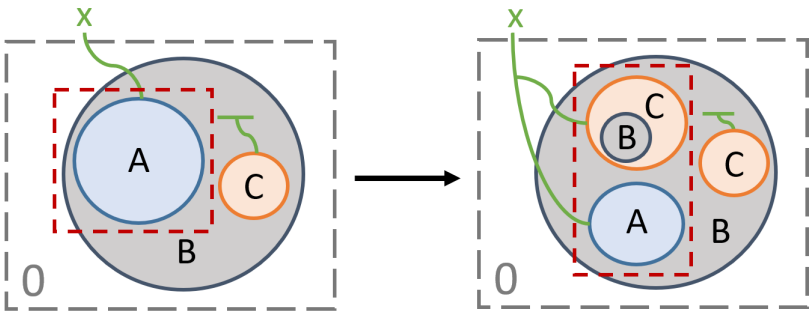


Figure 2.13: The reaction rule in Figure 2.12 applied to a bigraph in order to perform a rewrite operation. The highlighted region indicates the change in structure.

A simple example bigraph reaction rule is provided in Figure 2.12, where applying this rule will introduce two additional entities to the structure, with the port of entity C joining onto the open link of A. An application of this rule is then demonstrated in Figure 2.13, where the state of the bigraph evolves and adds the additional entities to the larger bigraph upon finding the valid occurrence of A via some bigraph matching algorithm.

With reaction rules and bigraph rewriting defined, we can now finally provide the definition of the bigraph reactive system (BRS), a rewriting system based upon the bigraph formalism.

Definition 2.3.14 (Bigraph reactive system)**.** A bigraph reactive system (G, R) consists of an initial ground bigraph state G and set of ground reaction rules $R_1, ..., R_n$, where the application and corresponding rewriting operations of valid reaction rules can be performed on G and its resultant states at each transition step in order to build a transition system of all reachable states.

BRSs can be understood as the bigraph equivalent to a GTS—a key difference between the two formalisms is that the complex and abstract behavior of bigraphs can more intuitively model scenarios in a BRS where we want to define explicit abstraction, locality, instantiation maps and relationships between entities represented by hypergraphs in a way that GTSs do not support [24]. Model verification can be performed on the transition system produced by a BRS in order to determine the reachability of each state, which can be enhanced with the use of *stochastic bigraphs*, which assigns a weighted probability to each reaction rule in the

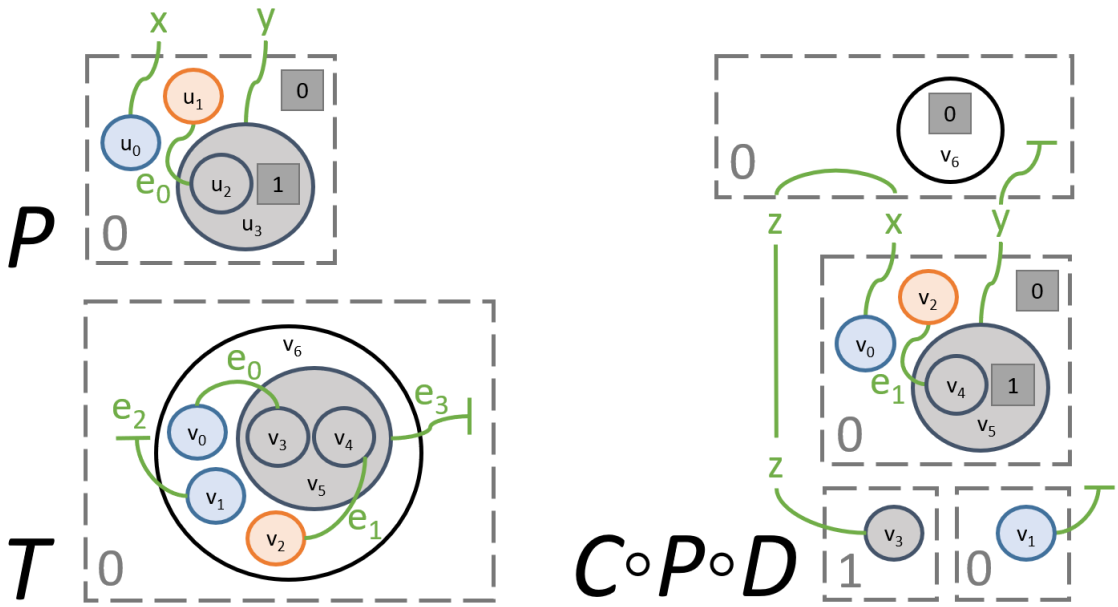


Figure 2.14: An example bigraph matching instance where vertex color denotes entity control type. The target bigraph T can be decomposed from above and beneath to produce pattern bigraph P, producing the vertex mapping $\{(u_0, v_0), (u_1, v_2), (u_2, v_4), (u_3, v_5)\}$ and edge mapping $\{(e_0, e_1)\}$.

BRS in order to calculate the overall chance of each possible reachable state being produced when a simulation is run [6].

2.3.5 Bigraph Matching

The bigraph matching problem can be described as the bigraph equivalent to SIP, a similarly NP-complete decision problem where we want to determine whether a pattern bigraph P occurs inside a host target bigraph T. However, unlike SIP which is defined via a mapping from pattern vertices to target vertices, we recall that an occurrence in bigraphs is defined in terms of bigraph composition—where P can only be considered occurring in T if the target bigraph can be obtained through composing some context C and parameter bigraph D onto the top and bottom of the pattern bigraph respectively (Definition 2.3.11). Hence, a bigraph matching algorithm must determine if $T = C \circ (P \otimes id) \circ D$, which will confirm whether the subsequent rewriting operation is then possible via performing the decomposition, substituting P with the supplied reactum bigraph, and recomposing the context and parameters to obtain the transformed bigraph.

We formally define the problem as follows.

Definition 2.3.15 (Bigraph matching problem)**.** Given an input pattern bigraph $P : \langle m, X \rangle \to \langle n, Y \rangle$ and target bigraph $T : \langle k, W \rangle \to \langle l, Z \rangle$, a valid matching solution is one where T can

be decomposed to achieve an isomorphism P such that

$$T = C \circ (P \otimes id) \circ D$$

for the identity bigraph id and context and parameter bigraphs C and D.

In order to build the full transition system of a BRS, we wish to enumerate *all* such possible matches, and thus we aim to find $T = C_n \circ (P \otimes id) \circ D_n$ for all n possible matches in the problem instance.

There are clear parallels to SIP that can be observed—both are subgraph algorithms where a matching solution can be represented in the form of an injective mapped assignment from each pattern element $p \in P$ to a target element $t \in T$ [39]. As described previously however, bigraph matching unlike SIP is a fundamentally compositional problem, where it must also hold that T can be decomposed into P via bigraph composition rules. This introduces additional constraints as to what can be considered a valid match, and hence solely finding an isomorphism of P inside T is not enough to assume it is a valid solution. For example, any unmapped entity in T must be able to reside in either C and D for a candidate solution. Another key difference is that bigraph matching seeks to map all elements of the support (V_P, E_P) of P to T rather than just its vertex set, so all closed edges in E_P must also have a corresponding match.

An example matching instance is provided in Figure 2.14, where it can be observed that any target entity which isn't mapped to must be placed in either the context or parameter. The hyperedge connection from v_0 to v_3 is made through id—since the identity bigraph provides a route for a link from an outer face that loops back around to D, we do not need to distinguish between inner and outer links for matching. Open links can also be closed in the parameter as seen through y, demonstrating that open links can match with closed links but not vice versa.

In addition to identifying occurrences for rewriting in a BRS, bigraph matching is also required multiple times in order to perform a rewrite step for conditional bigraph reaction rules [40]. It is also used to determine state predicates i.e. patterns in bigraphs [41]. When working with stochastic bigraphs [6], the number of isomorphic/symmetric solutions found for a reaction rule applied to a given state will impact the overall probability of the resultant state being reached upon rewrite.

2.4 Constraint Programming

The constraint programming (CP) paradigm provides an efficient method of solving complex NP-hard satisfiability problems, in a manner where the user is not required to manually

perform any solving or calculations but rather simply model the problem in the form of a *constraint satisfaction problem* (CSP). A CSP consists of a set of variables, their range of possible values and the relationships between variables in the form of *constraints*—a list of conditions which forbid certain combinations of value assignments, as otherwise the solution would be rendered invalid (i.e. no rows or columns in a Sudoku puzzle can contain the same number more than once). A CP language or toolkit allows the user to construct a CSP, which is then passed into an "under the hood" solver which aims to find a combination of values for all variables such that none of the user-defined constraints are violated: this result is then returned as a *satisfiable* solution [42]. The underlying solver relies upon a variety of different methods to solve CSPs such as intelligent *backtracking search* (Section 2.4.1) as well as inference and propagation, i.e. ruling out possible values for variables by considering the list of constraints in combination with values already assigned to other variables. Specialized CP solvers designed for specific subclasses of problems (such as the Glasgow Subgraph Solver for graph matching) may also apply domain-specific *heuristics* and *search strategies* which are known to be the most optimal for that type of problem. One example is the "fail-first" method, which prioritizes the assigning of values to the most constrained variables — which for some problems directly correlates with the difficulty of finding a non-conflicting value for them [43].

At a high level, CP can be seen as a form of declarative programming, where the programmer specifies what needs to be done, rather than how it needs to be done. Whilst there is some level of control in how the solving can be performed via variable and value ordering heuristics and other customizable parameters, the underlying complexity of searching for solutions is mostly abstracted away from the user. Modelling combinatorial problems as CSPs, in addition to simply solving logic puzzles like Sudoku, can be used to model NP-complete software algorithms like type system inference in programming languages [44] and graph coloring, as well as a variety of industrial real-world applications such as resource allocation for performing factory equipment maintenance [45] and job shop scheduling [46].

Formally, we define a CSP as follows [47].

Definition 2.4.1 (Constraint satisfaction problem)**.** A CSP $\{V, D, C\}$ is a triple, made up of:

- **Variables** $V = \{v_1, v_2, ..., v_n\}$: The model's set of variables to assign values to, with cardinality n.
- **Domains** $D = \{d_1, d_2, ..., d_n\}$: A set which has a bijective mapping from V, where each $d_k \in D$ is a list of potential values that the corresponding variable $v_k \in V$ can be assigned.
- **Constraints** $C = \{c_1, c_2, ..., c_m\}$: The set of constraints which are imposed on permutations of the set of variables.

Domains can be a variety of data types, such as booleans (SAT), integers or real values. As an example, in a 9x9 Sudoku puzzle where tiles can only be filled with integers from 1-9, each variable $v_k \in V = \{v_{(1,1)}, v_{(1,2)}, ..., v_{(9,9)}\}$ would have the corresponding domain value set $d_k = \{1, 2, ..., 9\}$.

Constraints can take many forms—types of constraints include binary constraints such as the less-than constraint, which involve the relationship between the values of two variables, e.g. $3v_1 < 4v_2$. Ternary constraints similarly involve three variables e.g. the arithmetic constraint $v_1 + v_2 = v_3$. *Global constraints* meanwhile can enforce a specific relation between an arbitrarily-sized permutation of variables in the CSP—an often-used example of a global constraint is the `alldifferent` constraint, which declares that for all possible pairs $(v_a \in U, v_b \in U)$ in a given subset of variables $U \subseteq V$, $d_a \neq d_b$. Described informally, this enforces that no values in the subset can share the same value assignment. The set of constraints in a Sudoku puzzle modelled as a CSP simply consists of 27 `alldifferent` constraints, for each of the nine rows, columns and 3x3 boxes.

A solution is complete when all variables $v_k \in V$ have been assigned a value $d_k \in D$. However, a solution is only considered *satisfiable* if it is both complete and all constraints $c_k \in C$ are satisfied. The user can determine whether the solver immediately terminates upon finding a satisfiable solution, or continues to search for all possible solutions in a CSP. A CSP is called unsatisfiable if it does not contain any satisfiable solution. For both retrieving all solutions and proving that a CSP is unsatisfiable, the entire underlying search space in a backtracking solver must be fully exhausted in order to rule out all other possibilities.

There also exists a variant to CSPs known as *constraint optimization problems* (COPs)—these are modelled similarly to a CSP, but with the objective of optimizing a user-defined variable known as the *reward function*. Optimization can take the form of either maximizing the reward function e.g. the most packages delivered in a given time limit in an instance of the travelling salesman, or minimizing it e.g. the minimum total distance to travel to deliver all packages. These are typically solved by proving that any instance where the reward function is fixed to a more optimal value than the current best solution is unsatisfiable.

2.4.1 Backtracking Search

A typical underlying combinatorial search process can be represented as a depth-first search of a decision tree of value assignments, which make up a partial/candidate solution. Each node at depth k represents a value selection from the domain Dv_k to assign to v_k. Upon each step, the solver will *propagate* the constraints of the problem, which will remove values from the sets of each domain which would cause a constraint violation to occur if assigned to the current candidate solution—narrowing down the available search paths as it traverses.

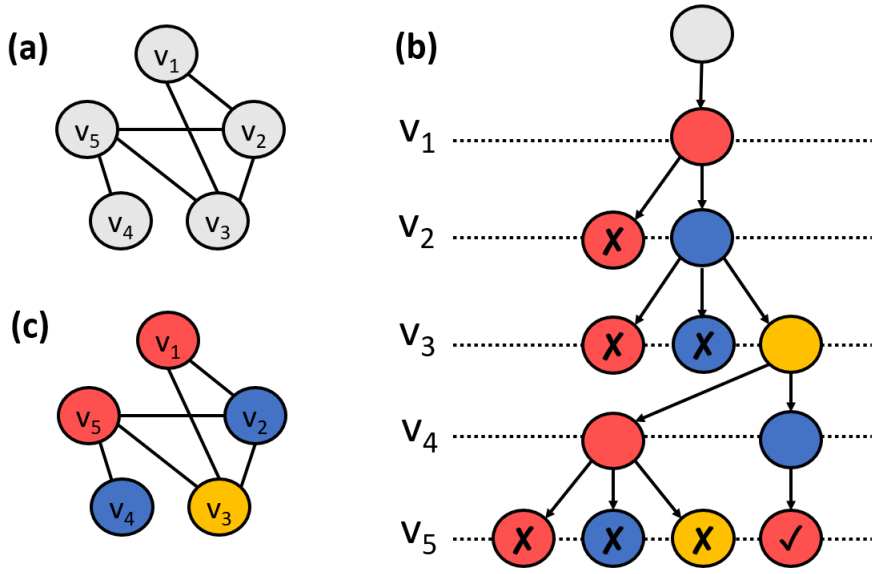


Figure 2.15: (a) An instance of the NP-complete 3-coloring problem. (b) An example of DFS backtracking search, where the value-ordering heuristic selects in the order $\mathbf{red} \rightarrow \mathbf{blue} \rightarrow \mathbf{yellow}$. (c) A satisfiable solution found through search.

If any single domain is completely exhausted mid-traversal, this indicates that the partial solution is unsatisfiable, and it will backtrack to a previous state and attempt an alternate search path. A simple demonstration of the CP search and backtracking process is given in Figure 2.15, where it is used to solve a trivial instance of the 3-color problem—that is, we want to color each vertex in a graph with one of three possible colors such that no connected vertices share the same color. Here, each variable in the CSP represents each vertex the graph, each variable's initial possible domain of values are $\{\mathbf{red}, \mathbf{blue}, \mathbf{yellow}\}$, and there exists a constraint for each edge $e = (u, v)$ which enforces that $u \neq v$. The solver continues to assign values and backtrack upon hitting a constraint conflict, i.e. two connected vertices sharing a color, until all vertices have been colored, which is returned as a satisfiable solution.

2.4.2 Optimizing Search

The efficiency of a CP solver can be improved upon through various context-dependent configurations. One such primary method is making use of an effective *heuristic*, which specifies the order in which variables and values are selected and propagated during search. These come in the form of *variable ordering* and *value ordering* heuristics, which define the ordering selection of variables and the ordering selection of values for each variable respectively [48]. For example, it is often beneficial to prioritize the selection of variables and values which are more heavily constrained in the model, under the assumption that finding a satisfiable assignment for these will be a more difficult task. This causes the solver to "fail early" during search and backtrack early in the search tree, rather than get stuck searching large

subtrees where there exist little or no solutions due to a poor assignment made early on, in a phenomenon known as *thrashing*.

Other such heuristic techniques include search *restarting*, where the solver will periodically reset all current assignments and start again from the top of the search tree as an alternative strategy for preventing thrashing. This is combined with introducing a slight amount of randomness during selection to ensure that the same search paths are unlikely to be redundantly explored multiple times. This can also be combined with parallelism to explore multiple branches of the search space simultaneously [49].

Restarts are typically combined with a special type of dynamically posted constraint called *nogoods* [50]. Nogoods define a specific subset of variable-value pairs which can never appear in any satisfiable solution, and can be represented via the formula $\neg(v_1 = d_1 \vee v_2 = d_2 \vee ... \vee v_k = d_k)$ for a subset of variables v_1 to v_k. Nogoods can be posted to a solver during search to retain information on which subtrees yield no solutions (and thus should not be redundantly searched again) after a restart [51], or preliminarily invalidate solutions which can be inferred to be incorrect based on what has already been discovered during search [52]. Another potential use-case of nogoods is to employ them as a *symmetry breaking* constraint, which prevents the solver from considering candidate solutions which are isomorphic to (and thus can be treated as a duplicate of) a previously discovered solution.

2.4.3 Subgraph Isomorphism Modelled as a CSP

As a relevant example, we demonstrate how a general CP model can be utilized to solve the subgraph isomorphism problem by constructing a representation of SIP in the form of a CSP. Given a pattern $P = (V_P, E_P)$ and target $T = (V_T, E_T)$ input graphs, we model the set of variables V as the set of pattern vertices V_P since we seek to find an assignment for all $v \in V_P$. Correspondingly, the set of domains D for each variable is the set of target vertices V_T since we aim to find an injective mapping from V_P to V_T.

We now consider the constraints of the model. We firstly apply the `alldifferent` constraint across all $v \in V_P$ as no two pattern vertices can map to the same target vertex. We then consider the set of edges in E_P—each edge can be modelled as its own constraint applied to its pair of pattern vertices, which enforces that their mapped assignments must also exist as a pair in E_T in order to be considered a valid assignment. We thus present our basic SIP model as follows.

$$CSP_{SIP} = \{V_P, V_T, \{\texttt{alldifferent}(V_P), \{\forall\, (p_1, p_2) \in E_P \mid (f(p_1), f(p_2)) \in E_T\}\}$$

For the induced variant of SIP, we can simply add the following additional constraint to

enforce non-edges between pairs.

$$\{\forall\, p_1 \in V_P, p_2 \in V_P, p_1 \neq p_2 \mid (p_1, p_2) \notin E_P \rightarrow (f(p_1), f(p_2)) \notin E_T\}$$

Labelled SIP can also be intuitively modelled through an additional equality constraint between candidate mappings.

While the constraints described are sufficient to accurately model SIP, we can also apply additional constraints and inference techniques in order to more quickly reduce the domains. For example, it can be deduced that a pattern vertex of degree n can never map to a target vertex of degree less than n. Therefore for all pattern vertices we can preliminarily reduce their domains to only target vertices with a compatible degree, which can be performed by checking all $(p \in V_P, t \in V_T)$ pairs in polynomial time before running the main search loop [53]. In the case of directed SIP, we can optimize this further by ensuring that both the in-degrees and out-degrees of each (p, t) pair are compatible as separate vertex compatibility constraints. Labelled SIP can also preliminarily eliminate pairs with non-matching types using this pair-checking method. The Glasgow Subgraph Solver extends the idea of degree filtering further through the use of *neighborhood degree sequencing* [54], where whenever a value assignment is made during search, the degrees of neighbors of the currently assigned set of vertices are also compared and incompatible target nodes eliminated from the domain during that propagation step.

GSS also makes use of a special all-different filtering technique based on vertex degree values. During search, if there exist any combination of remaining pattern nodes of size n still awaiting assignment where there are only $< n$ target nodes when taking the union of all their current domains, this preliminarily detects that no possible solution exists for the current state of value assignments. Conversely, if the union of target nodes is instead equal to n, then it can be inferred that no additional pattern vertex outside of that set can occupy any of those target nodes [55].

Finally, it also possible to reason on distances and paths between vertices rather than simply considering direct adjacencies. For example, for any pair of matches $(p_1, t_1), (p_2, t_2)$, the number of paths between t_1 and t_2 of length k must be greater than or equal to the number of paths between p_1 and p_2 of the same length in a valid solution. GSS identifies and propagates these additional path constraints through building additional *supplemental graphs* [56]. A supplemental graph S contains two specially-labelled distinguished vertices v_1 and v_2, and is used to build a modified variant of the pattern and target denoted as P_S and T_S which share the same vertex set as P and T respectively. However there can only exist an edge (g_1, g_2) between two vertices in $G \in \{P_S, T_S\}$ if there also exists an isomorphism from S to G where g_1 and g_2 map to v_1 and v_2 respectively. This produces a new SIP instance which reflects the desired path constraints. The set of supplemental graphs which GSS relies on

have been manually selected based on performance evaluations on a variety of benchmark instances [13].

2.5 Related Work

There already exist multiple available implementations for executing BRSs which handle matching through a variety of methods, such as the use of general-purpose solving tools like SAT encodings and constraint programming models, as well as a full reduction from BRS to a GTS where SIP can be used. In this section, we discuss and compare previously developed frameworks which aim to solve matching problems relating to bigraphs.

2.5.1 BPL

The first known implementation of a BRS was introduced by Højsgaard et al. [57], presenting the BPL Tool; an experimental toolkit that allows for the building and manipulation of bigraphs. BPL, through a domain specific language, allows a user to define bigraph reaction rules and agent states, run model simulations, and visualize bigraphs in either SVG or TikZ format. The underlying matching algorithm [58] required for rewriting reduces the graph-like structure of bigraphs into a term representation of bigraphs, where axiomatic matching rules can be applied to the transformed place and link graphs to give a non-deterministic set of embedded solution mappings for each subterm of the pattern. The non-deterministic property is then limited through the introduction of additional inference and congruence rules between terms. When BPL is run on a matching instance, the tool uses a lazy list to asynchronously return sets of matches, as the computational complexity involved causes BPL to be unreliable at finding all solutions quickly. Højsgaard et al. explicitly note in the conclusion of their BPL tooling paper that this implementation struggles to perform efficiently and is primarily suited for experimenting with smaller instances, recommending that a SAT solving approach will be likely to yield a better performance [57].

2.5.2 BigraphER

Sevegnani and Calder propose BigraphER [9], a command-line bigraph toolkit written in OCaml which allows for the modelling of BRSs through a user-provided algebraic representation of an initial state and its set of reaction rules. From there, BigraphER is able to perform matching, rewriting, simulation and visualisation on the BRS in order to reflect the evolving behavior of an interacting system of entities. BigraphER supports these operations on both bigraphs and the bigraphs with sharing extension, and additionally supports the building of

stochastic BRSs [6] as well as probabilistic BRSs [59], a discrete extension of stochastic BRSs.

The underlying matchings are primarily performed by encoding the problem into a SAT algorithm for subgraph isomorphism [10]. This is built upon MiniSAT [60], although there also exists an alternate Pseudo-Boolean implementation of this algorithm which relies on MiniCARD [61]. However, the number of CNF clauses required to perform matching grows rapidly in relation to the scale of the problem: $O(m^2n^2)$ clauses are required for a matching problem with m and n target and pattern entities respectively. This results in poor scaling for larger more difficult problems, limiting the overall scope of what the tool is capable of modelling. This SAT-based approach is also inherently low-level, and requires direct and manual encoding of constraints into conjunctive normal form (CNF) by the user which makes it less adaptable to building alternative formulations or extensions. This can be particularly tricky from a user perspective for encoding high-level graph constraints such as "at most k edges must belong to this vertex" in a way that is both correct and efficient [62]. Conversely, a more high-level approach such as a CSP solver, which can model the problem simply through expressions of variables and constraints, is able to provide a more adaptable framework without sacrificing solver performance. As the solver component runs independently from the rest of the BigraphER framework, it is feasible to swap these out with a hypothetical more efficient matching tool without affecting the rest of the toolkit's functionality, making BigraphER an optimal framework for adapting a new algorithm for with minimal overhead.

2.5.3 jLibBig

Peressotti et al. introduce jLibBig [63], a BRS toolkit in the form of a Java library which provides an interface for the modelling and manipulation of bigraph reactive systems as a component within a Java program. Similarly to our proposed approach, the underlying matching is performed by a constraint programming model. This is built upon *Choco*, a general constraint modelling toolkit which is also provided as a Java library, where a developer can define the variables and constraints of a CSP/COP for the underlying solver to search [64]. jLibBig has been integrated as the matching and rewriting component within wider Bigraph projects, such as the Bigraph Framework tool as part of the Bigraph Toolkit Suite [65].

The Choco model for bigraph matching is able to support pure bigraphs as well as the directed bigraphs extension (Section 4.3) through additional optional constraints, which preliminarily demonstrates the extensibility of a constraint-based approach to the problem. The matching of the place and link graphs are modelled as two distinct problems which are solved separately; the place graph matching is performed through attempting to build a bipartite graph relation from the set of entities and sites/regions in P to those in T, with constraints

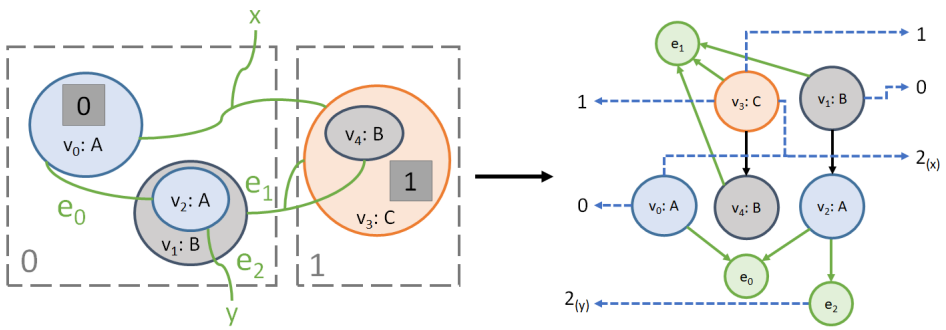


Figure 2.16: An example of the encoding of a bigraph to a ranked graph.

which ensure that the graph is injective for all P entities and all T nodes which are assigned a region/site as their pre-image follow the rules of bigraph composition. Meanwhile, the pattern and target link graphs are encoded and solved as a *multi-flux problem* where ports and faces are each encoded as vertices, each inner name and port represented by a source node and each outer name and closure a sink node, with adjacencies drawn between them to match the structure of the original hyperlink structure. The solver then aims to find a solution in the form of an additional set of edges connecting each pattern node to a target node in a way which satisfies the rules of hyperedges. Finally, a set of "gluing" constraints are introduced which enforce consistency between the solutions of the two sub-models and return a full solution set. A proof of soundness and completeness for this method is also provided. While Choco was chosen as the CSP tool for this algorithm, this has been designed to be implementable for any general CSP modelling toolkit.

While an algorithm built upon a general-purpose CSP solver such as Choco allows for flexibility in regard to implementation, the ability to add more complex heuristics and techniques that would be more optimal for graph solving problems is restricted to only those which can be supported by the interface of the solver of choice: for example, Choco does not support parallel solution biased search, which is the heuristic GSS relies on. In addition, generalized solvers are designed to support a wide variety of possible CSPs and are not engineered specifically for modelling and solving graph-based problems, and thus do not offer the additional under the hood benefits that GSS provides such as supplemental graphs, bit-parallelism and other state of the art features and techniques that promise optimal performance. We also propose that a SIP based modelling for bigraph matching will be able to solve matching for both the place graph and link graph as one single encoded graph pair, rather than requiring the matching of the place and link graph components to be split into their own sub-problems.

2.5.4 BiGMTE

Gassara et al. propose an approach similar to ours for the BRS toolkit BiGMTE, where bigraphs are encoded as *ranked graphs* [66]. Ranked graphs $G = \langle r, d, v \rangle \rightarrow: i \rightarrow j$ are a triple consisting of a directed graph d, in addition to a root mapping function $r : j \rightarrow V_d$ and variable mapping function $v : V_d$, which define interfaces similarly to the inner and outer faces of bigraphs. Also similarly to bigraphs, two ranked graphs can be composed by merging these interfaces whenever $i_{G_1} = j_{G_2}$.

This structure is extended to include an additional labelling function $l : N \rightarrow L$ to support the encoding of control types. During the encoding process, each entity is represented by a labelled vertex, each region/outer face and site/inner face is represented by a root mapping and variable mapping respectively, and each hyperedge is encoded as an additional sink node in the graph which is a child of all of its encoded entity nodes. Rather than perform the encoding only for matching, ranked graphs are used to reduce the entire BRS into a GTS instance, which then performs the graph rewriting process (as an instance of DPO graph transformation) and then converts the resultant state back into a BRS format in a way which preserves the expected behavior of applying a bigraph reaction rule.

Assuming negligible encoding/decoding time, the performance of this ranked graph approach depends on the underlying graph transformation framework. BiGMTE employs GMTE [67], a graph matching and rewriting toolkit to perform these underlying SIP and rewriting instances, and performance metrics are provided in comparison with jLibBig. It is shown that BiGTME generally solves instances faster than jLibBig, and scales far more efficiently for instances with a large number of entities ($|V_P| > 50$) in the pattern graph. However for instances where there are increasingly many matchable nodes in the target graph ($|V_T| > 100$), BiGMTE scales more poorly compared to jLibBig. Another observation is that BiGMTE only supports matching on the default definition of bigraphs, unlike jLibBig supporting directed bigraphs and BigraphER supporting sharing.

Whilst this method aims to find a halfway point between bigraphs and standard graph structure which a graph matching tool can then solve through ranked graphs, we believe that this idea can be explored further through a new encoding which resembles and behaves closely enough to a subgraph isomorphism instance in isolation that any SIP solver (GSS in particular) can be capable of solving it, with the additional help of preliminary and checking constraints to preserve the compositional integrity of all found solutions and filter false positives.

2.6 Summary

In this section, we provided a background reading of the elements of research that our work later builds upon. Section 2.1 introduced term rewriting and transition systems. Section 2.2 provides a specification of rewriting where this is applied to graphs, alongside the underlying required graph matching problem (SIP). Section 2.3 introduces bigraphs, the main focus of this project, as well as the bigraph matching problem, which is the key problem we wish to solve. Section 2.4 introduces constraint programming, the paradigm we wish to use to tackle the complexities of bigraph matching. Section 2.5 provides a review of currently available BRS tools and their corresponding solvers for bigraph matching.

In the following chapter, we introduce a novel algorithm for bigraph matching which incorporates various elements used by the solvers covered in Section 2.5, including a reduction to SIP, encoding the bigraph as a graph with additional properties to model interfaces, and an implementation in the form of a CSP model. This later replaces the SAT matching component within BigraphER.

Chapter 3

A New Algorithm For Bigraph Matching

In this chapter we introduce the formal definition of our matching algorithm, which encodes problem instances into a format which can be understood and solved by a conventional SIP solver. A preliminary version of this work has presented in our CP2021 publication [15].

In Section 3.1, we discuss our hypothesis of the feasibility of a "bigraph matching to SIP" strategy for efficient solving, and give a brief high-level overview of the full proposed encoding. Section 3.2 provides a formal definition of the encoding of the place graph component of a bigraph, as well as the handling of an identified edge case where our encoding is slightly under-constrained. Section 3.3 covers the encoding of the link graph component and the flattening function which converts the hypergraph elements into a conventional graph format, as well as the handling and removal of symmetries introduced by our algorithm. Section 3.4 provides the size of the resultant encoding compared to that of the input pattern-target bigraph pair. Section 3.5 gives a proof of soundness and completeness of our encoding. Section 3.6 concludes this chapter.

3.1 Motivation

The key hypothesis which motivates this proposed algorithm is that bigraph matching instances share a strong similarity in properties and behavior to graph matching problems such as SIP, as long as extra constraints to ensure that the additional bigraph rules introduced by interfaces and composition/decomposition rules still hold can also be enforced. As outlined in Section 2.4.3, using a dedicated high-level subgraph solving tool like GSS has numerous potential advantages over existing bigraph tools through being able to perform additional reasoning on degrees, as well as supporting neighbourhood degree sequencing, all-different fil-

tering and supplemental graphs. In addition, a high-level constraints-based approach allows for the application of efficient heuristics which are optimized for graph problems [38, 49]. There exist some obstacles to this approach: graph matching tools generally do not support the complexities introduced by the bigraph formalism such as hyperedges or a two-tier graph model.

While Gassara et al.'s ranked graph approach (Section 2.5.4) involves a similar strategy of encoding bigraphs into graphs (with interfaces), this is performed as part of a wider reduction from the full BRS to a GTS rewriting system, and hence does not provide the underlying dedicated matching algorithm itself (Section 2.1.1) but rather the framework to execute a BRS using a graph transformation toolkit. This method also requires tight coupling from the BRS to the GTS to be able solve bigraph matching in this manner, rather than being able to treat the solver as an abstract "black box" matching component within a wider toolkit — retaining the existing wider bigraph-focused framework in a state of the art BRS such as BigraphER. Hence, we are interested in devising a dedicated optimized SIP-based algorithm *exclusively* for performing matching, which is able to take the pattern and target bigraph as input and return the solution without any wider dependencies. This would the solver to support any existing BRS tool, where it can "swap out" any previously relied-upon SAT, PB or other types of solvers typically used for bigraph matching without affecting how the higher-level bigraph framework will function. Thus, the main contribution of this proposed approach is an extensible and portable *standalone* algorithm for bigraph matching, based on the strategy of encoding bigraphs to graphs.

In order for a SIP solver to be able to solve bigraph matching, we propose that the problem can be reduced into an instance of SIP through an *encoding* from bigraphs to a more standard graph format in such a way that the extra complexity and behavior of bigraphs are still retained in the transformed structures. This encoding is performed via two main steps—firstly, an initial encoding function is defined which transforms a place graph into a forest graph (removing/replacing interface components), and additional conditional degree constraints are added to each vertex which enforce either "must have an in/out degree of at least n" if connected to the inner/outer face respectively and "must have in/out degree equal to n" otherwise, which models only allowing additional edge if that entity can potentially join to another entity through composition. This demonstrates that matching of place graphs can be treated as a hybrid of induced and non-induced SIP, evaluated upon a vertex by vertex case. Secondly, the hyperedges of the link graph can then be *flattened* into the encoded graph through representing them as cliques of edges between adjacent entities, producing a *flattened* graph which is now in a sufficient graph format to give to a SIP solver as input.

Additional labelling constraints and degree constraints are handled through a vertex compatibility function, which eliminates invalid pattern-target vertex pair matches before search. While this is sufficient to ensure that every solution in a bigraph matching instance corre-

sponds to a SIP solution, some false solutions may also potentially occur due to an edge case related to regions in the pattern place graph, and duplicate solutions may occur from symmetries which are introduced by our the cliques-as-hyperedges strategy. The former is handled by adding one additional constraint which ensures that for all pattern vertices that share a parent region, their set of matched target vertices must also share the same (or no) parent. The latter can handled by either nogood constraints or symmetry breaking constraints; we provide a strategy (and later working implementation) using both methods. Dealing with these is enough to guarantee a bijection of solutions between an instance of bigraph matching and its encoded SIP instance.

We now go on to more formally describe these encoding and flattening functions. These assume that the input bigraph is both solid and that the instance is non-trivial—that is, there exists at least one entity in both the pattern and target.

3.2 Place Graph Encoding

The place graph encoding functions take an input bigraph and produce the graph (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 which we define in Section 3.2.3.

Importantly, two separate functions must be specified for the encoding process of the pattern and target place graphs in order to allow matching constraints conditional upon region and site adjacencies in the pattern to be sufficiently modelled.

3.2.1 Pattern Place Graph

To identify the necessary constraints, we begin by considering the most basic case of bigraph matching—where neither the pattern nor target contain any regions or sites, i.e. the place graphs solely consist of entities in a directed forest graph structure. In this case, when passing the pattern and target place graph directly into a SIP solver, it will identify all instances of the structure of P occurring in T, one of the necessary requirements of a valid match. However, when no abstractions exist, the only way $T = C \circ (P \otimes id) \circ D$ can hold is if $T = P \otimes A$ for some disjoint bigraph A, as there exists no interface for composition onto P and thus the context and parameter must be left empty. We can deduce from this that when modelled as SIP, all trees in the forest of P must match to an isomorphic tree in T—and more specifically, each vertex $p \in P$ can thus only match to a vertex $t \in T$ if they share the same number of both incoming and outgoing adjacencies. From this, we can reason that this additional degree equality constraint must hold for all p which does not have an edge to or from an interface node.

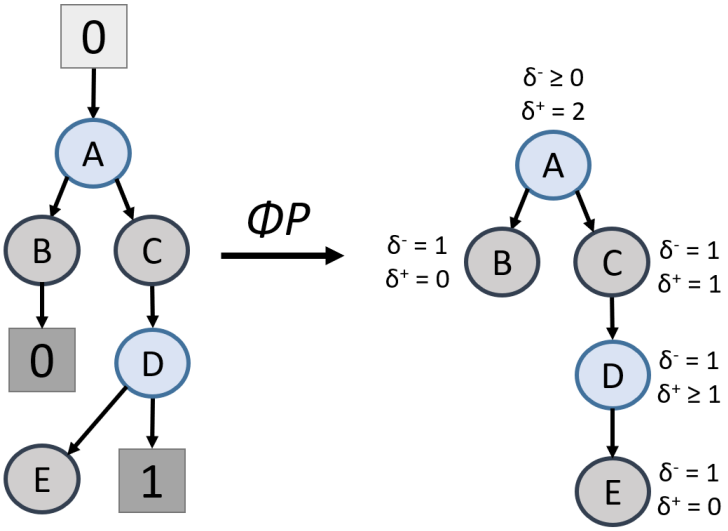


Figure 3.1: Example pattern place graph encoding—regions/sites removed and degree constraints introduced.

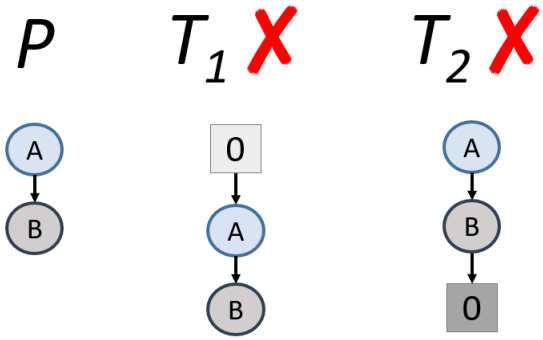


Figure 3.2: Example matching instance with the introduction of abstractions in the target. Neither match is possible here without the corresponding abstraction existing in the pattern.

Now we consider the introduction of regions and sites into the pattern graph. While we want to remove these abstractions from the structure as they cannot be encoded into a tangible vertex to be matched, their positions and adjacencies in the graph are still key to be able to determine what can be considered a valid composition. When an entity has a region as its parent, this indicates that there exists an opening that permits composition with an entity adjacent to the inner face of the context C. Conversely, when an entity is pointing to a site (recalling that solid bigraph entities may only have up to one child site), this permits composition with any number of entities (including none) via a region on the outer face of the parameter D. We can therefore reason that in cases where $prnt(p) \cap n \neq \emptyset$ or $prnt^{-1}(p) \cap m \neq \emptyset$ for some $p \in P$, the proposed degree constraints on entities should be more relaxed to accurately model the bigraph composition logic.

Let $P^{\mathsf{P}} = (V_P, ctrl_P, prnt_P) : i \to j$ be a concrete place graph representing the pattern of a

bigraph matching instance. We define our pattern encoding function as

$$\phi_P : P^{\mathsf{P}} \mapsto (V, E)$$

The place graph is stripped of its sites and regions in addition to any adjacencies to/from them, as they do not represent a concrete component to be matched onto the target Formally, the encoding produces the following graph:

$$V = V_P$$

$$E = \{(u, v) \in prnt_P^{-1} \mid v \notin i, u \notin j\}$$

An example application of this encoding is provided in Figure 3.1.

Each node in the pattern is assigned up to two unary degree constraints in the compatibility function (Section 3.2.3), which restrict their in/out degrees respectively. All entities which have no direct adjacencies to any abstract nodes must match in/out degrees exactly, and thus are assigned equality constraints for each.

Nodes adjacent to a region are left unconstrained, to allow an additional potential incoming edge. Similarly, nodes adjacent to a site are instead assigned a more lax $\geq$ constraint on the out-degree count of the node to allow additional potential outgoing edges. This method of encoding allows us to model the behaviour of abstractions on connecting entities without requiring them to physically exist in the graph structure, making it compatible as an input pattern graph for SIP.

3.2.2 Target Place Graph

Now we consider the addition of regions and sites into the target graph. Similarly to the place graph encoding, these abstractions will never correspond to a mapping as they represent interfaces for composition rather than nodes that we are interested in directly matching. However, their presence still impacts what can be considered a valid match, as seen in Figure 3.2—particularly, an entity adjacent to an interface node cannot match to an otherwise identical entity which lacks an adjacency to the same interface, as these must be preserved to produce T after composition. To handle this case, we encode regions and sites as "dummy" vertices that will never be compatible with any nodes in the pattern and so will never appear in mappings for any solution—however, their edges will still contribute to the in and out-degree values of their children and parent respectively, which is sufficient to filter any false positive matches and model the expected behavior of bigraph matching whenever this occurs.

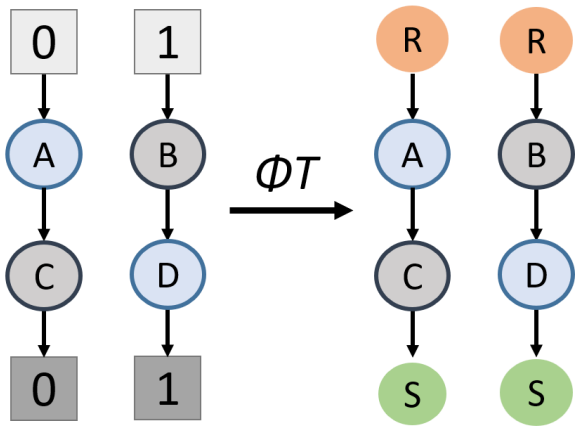


Figure 3.3: Example target place graph encoding—regions/sites are replaced with unmatchable vertices (shown as R and S but have no label in the encoding). Vertices show control labels.

Let $T^{\mathsf{P}} = (V_T, ctrl_T, prnt_T) : m \to n$ be a concrete place graph representing the target of a bigraph matching instance. We then define our target bigraph encoding function as

$$\phi_T : T^{\mathsf{P}} \mapsto (V, E)$$

The abstract components of the place graph are replaced with an additional vertex with a "dummy" label to ensure they will never be matched to, while retaining their adjacency relations. Formally, the encoding produces the following graph:

$$V = V_T \uplus \{r_i \mid i \in n\} \uplus \{s_i \mid i \in m\}$$

$$E = prnt_T^{-1}$$

where $prnt_T^{-1}$ is the child relation. We encode them in this manner in order to preserve the in/out degree values of their parent/child entities, which is required for preserving the compositional property of bigraph matching in the encoded graph and its corresponding constraints. An example application of this encoding is provided in Figure 3.3.

3.2.3 Place Compatibility Function

Additional bigraph specific constraints are handled by the place compatibility function:

$$\ell_p : V_G \times V_H \to \{t, f\}$$

This identifies prior to search whether a specific pair of vertices (p, t) belonging to the pattern and target respectively can potentially—or never—be matched in a solution, based on their

controls and adjacencies.

In the definition of this function, we refer to some bigraph definitions specified in Section 2.3 for both the pattern and target bigraphs. For clarity we define ℓ_p over the original bigraph entities e.g. $u \in V_P$, although formally ℓ_p is over their encoded SIP graph vertices e.g. $v \in V_G$, where $v = \phi_P(u)$.

ℓ_p can be defined as a combination of two sub-functions $(\ell_{p_1} \wedge \ell_{p_2})$, for checking control/label compatibility and conditional degree compatibility based on the presence of sites/regions respectively.

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

Simply states that entities must maintain their controls, which can be enforced by encoding each control to a corresponding vertex label.

For ℓ_{p_2}, we define $\delta^- : V_p \to \mathbb{N}$ as the number of vertices $u \in E_P$ where $(u, v) \in E_P$ such that $v \in V_P$. Likewise, we define $\delta^+ : V_p \to \mathbb{N}$ as the number of vertices $u \in E_P$ such that $(v, u) \in E_P$. As described in Chapter 2, ordinals are used to represent the place graph interfaces.

We then define

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

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.

Within the context of a constraint programming model, this function is called as a polynomial pre-process step before the main search loop is performed. All $(u \in V_P, v \in V_T)$ pattern/target pairs are compared, and v is removed from the domain of u if f is returned.

3.2.4 Dealing With Multi-Child Regions

This encoding is almost sufficient to fully model bigraph matching, however there remains one edge case where the encoded SIP instance might find a false solution that does not respect

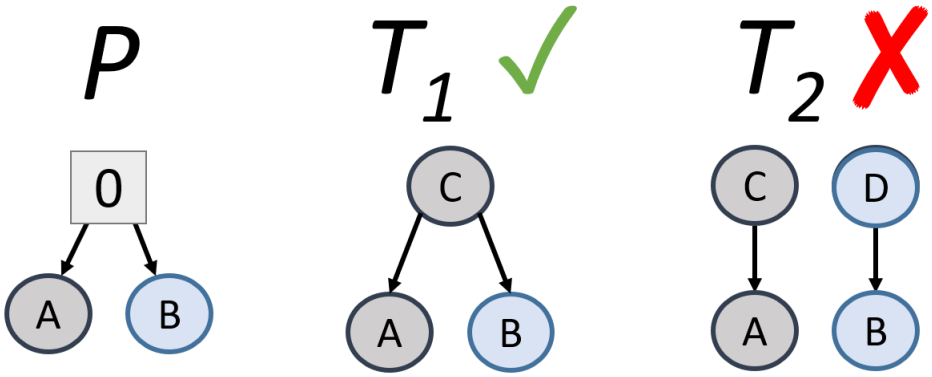


Figure 3.4: (P, T_1) is a matching instance which includes a region with multiple children. A valid mapping exists as node C can be encapsulated by the region. (P, T_2) is a similar matching instance with no valid solution, where our current encoding will return a false positive without further constraining all children of a region to have the same value of $prnt(p)$.

bigraph composition. Whenever a subset of entities $U_P \subseteq P$, $|U| \geq 2$ all share the same region value $prnt(u \in U_P) = k$, then this indicates that the value of $prnt(\texttt{match}(u)) \in T$ for all matches of U_P must also be the same, in order to ensure that the region merges with a single site in the context or identity during composition. This cannot be enforced by the encoding itself as we lose the necessary information from discarding regions. In Figure 3.4, a valid match can be found for matching instance (P, T_1) as node C can simply be joined onto the region of P in the context a s part of the composition, respecting its incoming edges. For instance (P, T_2) however, neither C nor D can be encapsulated by the region, as neither entity's set of children contains the full child set of R - therefore no valid match exists.

It may be possible to build a domain-specific solver to allow conditional matching in this special case, i.e. encoding the shared region as a vertex that can match to any (or no) target, however resorting to this method would contradict our initial hypothesis that matching can be solved by using an existing SIP solver (which supports direction and labelling) with additional constraints. We instead handle this necessary edge case by enforcing the following extra constraint on all pattern entities belonging to a shared region:

$$\{\forall v_a, v_b \in V_P \mid prnt(v_a) = prnt(v_b) = r \in n\} \rightarrow prnt(\texttt{match}(v_a)) = prnt(\texttt{match}(v_b))$$

where we denote a mapping of $v \in V_P$ to a target vertex as $\texttt{match}(v) \in V_T$ for a candidate/partial solution. This enforces that a solution is only valid when the set of children of each region's corresponding set of matches are either also siblings or orphans together.

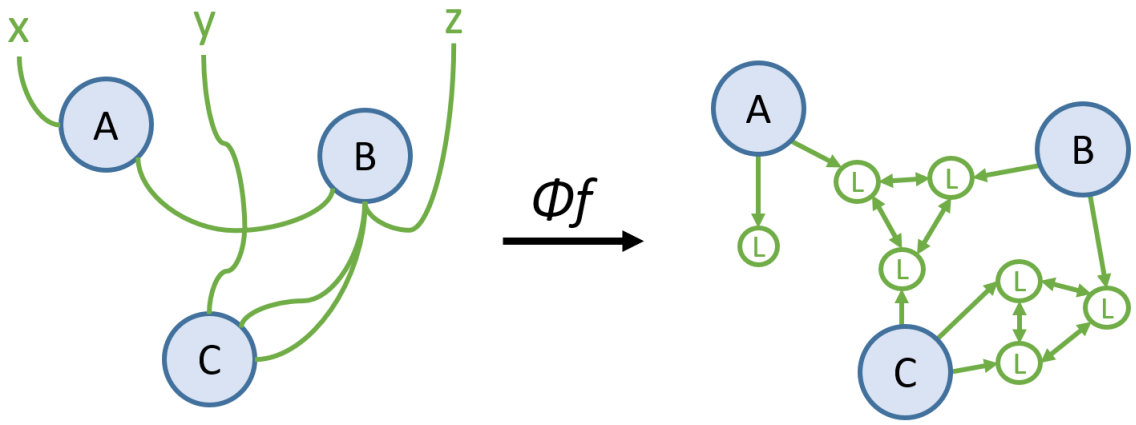


Figure 3.5: An example of flattening of open links—links become cliques between port vertices. Bidirectional arrows indicate a pair of directed edges between the vertices.

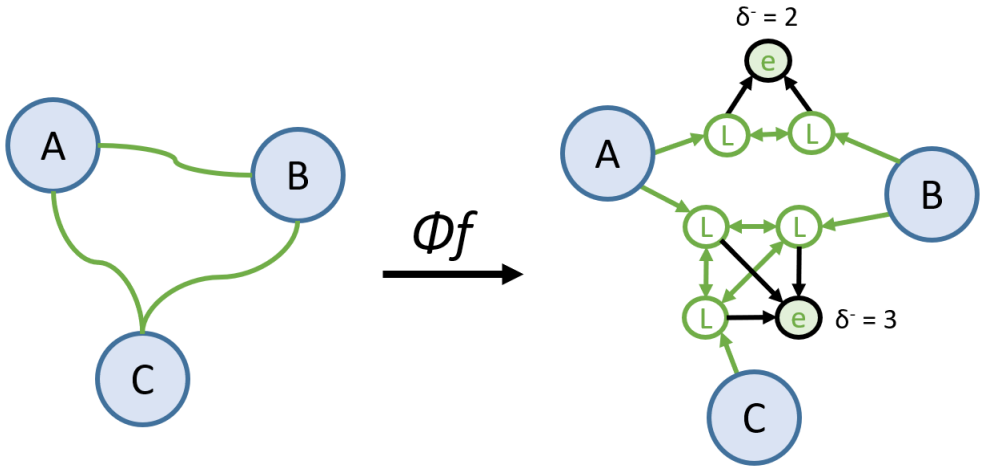


Figure 3.6: An example of flattening of closed links—edges are explicitly encoded as additional vertices to be matched, with a degree constraint to preserve the closure.

3.3 Link Graph Encoding

Once a place graph has been produced, we can then *flatten* the hyperedges in the link graph into our encoding so we can treat links as vertices in the resultant graph. In the case of bigraph matching, matching of open links is non-injective—that is, multiple open links in the pattern can "merge" and map to the same target link, but most methods for solving SIP do not support non-injective matching of vertices. This thus denotes the key challenge of modelling link graphs in a standard graph format.

A known method of reducing graph problems which include hyperedges into a more standard graph format is to flatten them into some form of clique format between adjacent vertices of each hyperedge [68]. The most simple way of achieving this is to draw labelled undirected edges between all pairs of vertices that share a hyperedge adjacency, however this is not sufficient to model the added complexity of the different behaviors of open and closed links, as well as encoding hyperedges of cardinality 1 which can exist in a bigraph matching instance. Instead, we look toward explicitly encoding the ports of each entity as specially typed child vertices where hyperedges are encoded as cliques between all ports, i.e. if a pair of ports share a link in the bigraph they share a link in the flattened representation. Because controls define the arity (number of ports) of each entity, a matching $(p \in P, t \in T)$ pair of the same type will always have a matching set of child link nodes, so we can represent ports this way without ever ruling out any potentially valid solutions. This flattening of hyperedges sufficiently models link graph matching for open links without the need for any additional constraints, as it encodes the expected behavior which only allows any pair of component pattern entities within a link to match to a pair of target vertices if there exists a similar shared link adjacency. This encoding is sufficient to model the non-injective property of open link matching; many links in the pattern can match to the one link in the target as they can be merged in the context, and thus our encoding similarly allows many of these link node cliques to match to the nodes that make up one larger clique as long as there is sufficient space between ports. While these nodes are explicitly matched during SIP, their assignments are omitted from any resultant solutions found as they do not count toward the support of a bigraph. It can also be noted that through this encoding, links in the pattern and the target can this be flattened using the same function, unlike place graph edges.

This method of encoding links also helps to more easily distinguish between place and link graph edges and allow us to model their behavior separately without requiring support for edge labelling. Edges pointing toward link vertices formally do not count toward the vertex degree constraints required to model the matching of place graphs in our encoding, however this conditional check is not required in practice as this encoding method simply increases the total out-degree of all matching vertices between P and T by the same amount and thus will always still permit a match.

3.3.1 Open Link Flattening

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

$$\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$$

where given an encoded place graph, a new flattened graph is produced.

We wish to explicitly model ports of entities, thus for every entity with an arity value of k, the flattening function will add k specially labelled $link$ vertices as a child of the encoded entity vertex. The label compatibility constraints enforced by the vertex compatibility function ensure that these vertices can only ever to each another. Each link is then represented by building a bidirectional clique between the encodings of all the ports it occupies, to model the property of each port in the hyperedge being adjacent to one another.

An example of the flattening function is shown in Figure 3.5, where all links of cardinality n are represented by an n-clique of specially-typed link nodes. It can be observed that this allows hyperedges of cardinality 1 to be represented as a single child link node with no additional adjacencies.

3.3.2 Closed Link Flattening

Whilst converting links to cliques between port vertices is sufficient to encode open links, closed links are subject to additional constraints.

Since closed edges count toward the support of a bigraph, we wish to model these in a way that we can explicitly match them to other closed edges in the target and include these assignments in an output solution. Closed links cannot match to open links as they cannot be "opened" again during composition. Intuitively, this requires the introduction of at least one additional specially typed vertex into our encoding to represent the mapping of closed links. A closed edge in the pattern can also only match to target closed edges with isomorphic adjacency sets—that is, for every n ports of $p \in P$ connected to a closed edge $e \in P$, $\texttt{match}(e) \in T$ must be adjacent to n ports of $\texttt{match}(p) \in T$. This can be modelled by constraining the matching of closed links to only those of equal degree, and their associated adjacencies will then match via conventional SIP rules. In addition, open links in the pattern can still match to closed edges similarly to they would a target open link, provided that it is not already occupied by a closed pattern edge, so our method of flattening these cannot diverge from our current clique encoding format.

This additional property of links can be modelled in the encoding itself by adding a new uniquely-labelled type of vertex—which we denote as a *closure* vertex— as a member of the

encoded clique of each closed link representation. Closure vertices are given an additional equal-degree constraint to enforce the constraint that only closed-to-closed matches are possible if their cardinality is the same size (ensuring isomorphism through matching of their adjacent entities), whilst ensuring the matching of open links in the pattern to closed links in the target remains otherwise unaffected. The encoding for a link graph featuring closed edges is shown diagrammatically in Figure 3.6.

3.3.3 Formalisation of Flattening

Given the concrete link graph

$B^{\mathsf{L}} : (V_B, E_B, ctrl_B, link_B) : X \to Y$, and the encoding of a (pattern or target) place graph $D\ \phi_{\{P,T\}}(D^{\mathsf{P}}) : (V_D, E_D)$, where (where $V_B = V_D$), we define the flattening function as follows:

$$\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$$

The vertices of the resultant flattened graph can be described as:

$$V = V_D \uplus P_B \uplus \widehat{E_B}$$

$$\widehat{E_B} = \{e \in E_B \mid link_B(p) = e, p \notin X\}$$

where $\widehat{E_B}$ is the set of closed links in B^{L}, P_B are the ports of B^{L} (defined in Definition X), 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.

We describe the resultant edge set as follows:

$$E = E_D \uplus \{(v, p) \mid p = (v, i) \in P_B\} \uplus \{(p_1, p_2) \mid p_1, p_2 \in P_B,$$

$$link_B(p_1) = link_B(p_2)\} \uplus \{(p, e) \mid e \in \widehat{E_B}, link_B(p) = e\}$$

For each pair of ports that share a link i.e. $(p_1, x), (p_2, x) \in link_B$ for some $x \in \{\widehat{E_B} \uplus Y\}$, this adds two additional directed edges (p_1, p_2) and (p_2, p_1) between their encodings which builds the resultant clique structure between all ports adjacent to x. For closed links in particular, one additional edge (p, e) is added for each encoded $e \in \widehat{E}$ where $(p, e) \in link_B$.

Finally we extend the vertex capability function for place graphs ℓ_p (3.2.3) 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.

3.3.4 Full encoding

We present a full encoding of a bigraph matching instance as SIP in Figure 3.7, which includes regions and sites in the place graph and open/closed hyperedges in the link graph. It can be observed that the same two solutions exist here for both the original matching problem and our SIP representation, from A to A and C to either target C entity. The open links of C in P can also be assigned to either the open links of the target C nodes or the closed link e_0, but we do not consider this difference to constitute as a separate solution in bigraph matching, which is only concerned with mapping the support constituents of P.

3.3.5 Removing Clique Symmetries

In the bigraph matching problem, while all open pattern links must be non-injectively assigned to target links, these mappings are not included in the embedded form of a solution—thus, it can be inferred that uniqueness of solutions is determined by the support mapping (entities and closed links) from P to T. Therefore for any two solutions identified by SIP which only differ in matches between encoded port nodes, one must be treated as a "duplicate" and discarded rather than incrementing the solution count when enumerating all solutions of an instance to ensure the bijective relation between matching and SIP. An example instance is provided in Figure 3.8, where it can be observed that this can occur whenever multiple ports of an entity share an adjacency with the same hyperedge, and thus produce symmetrical link nodes in the corresponding clique. To distinguish these new symmetries introduced by our link graph encoding from our definition of symmetrical bigraphs and bigraph intra-symmetries given in Chapter 2, we refer to these going forward as *clique symmetries.*

While these duplicate solutions are easy to detect and remove after search has completed for any SIP solver (filter any additional solutions where all support vertices are identical to a previous solution), these new symmetries can also be identified and eliminated during search in a constraint programming context via two methods; either through the use of nogood recording constraints to disallow solutions that only differ in their link node assignments,

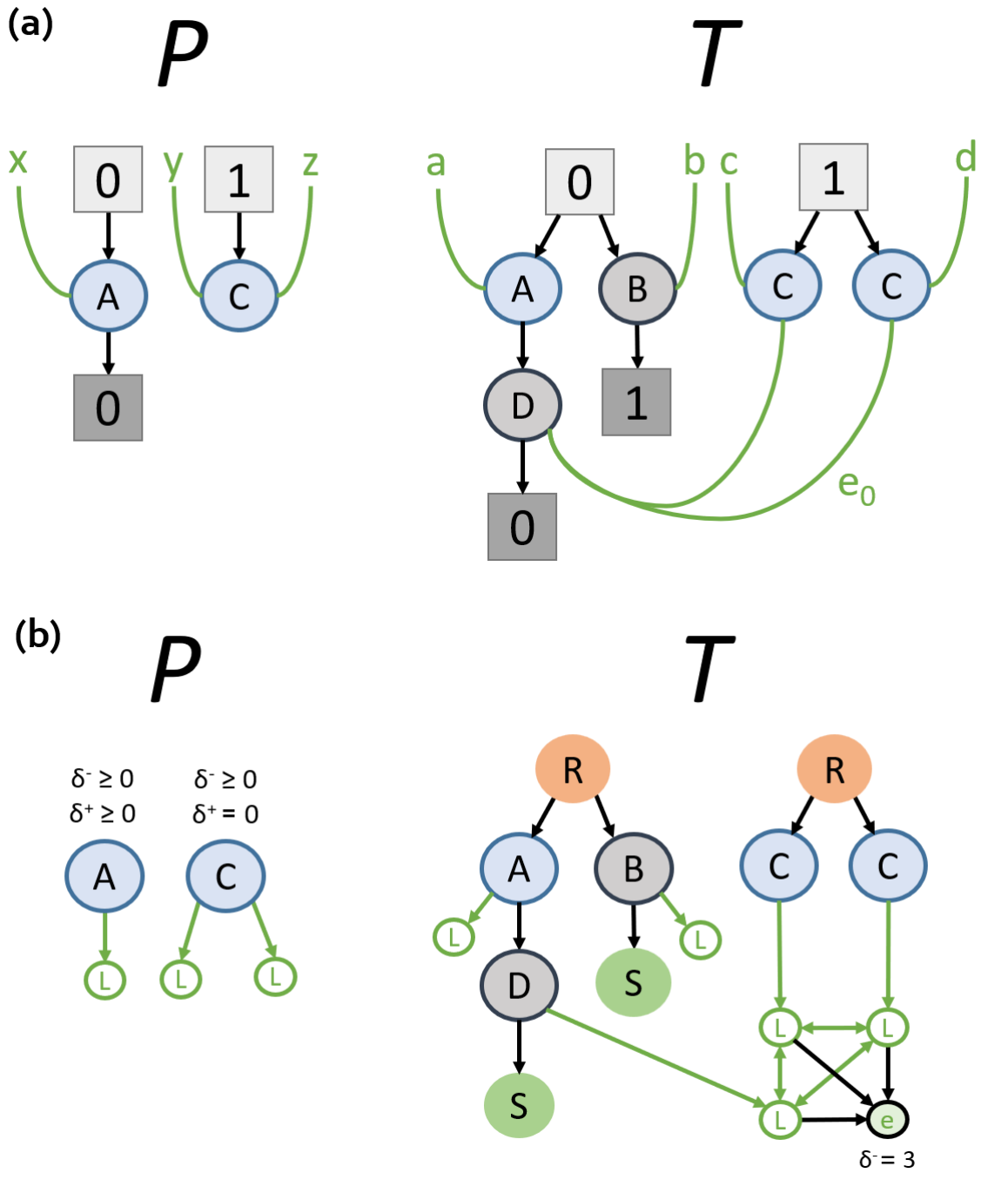


Figure 3.7: (a) An instance of bigraph matching, where P and T are shown as directed forests with additional hyperedges. (b) The fully encoded and flattened instance represented as a SIP with extra degree constraints.

or by introducing additional *symmetry breaking* constraints between pairs of link nodes to enforce that only one solution in each set of symmetrical assignments will ever be found. We define the corresponding new constraints using each method as fellows.

Nogood Recording

In our initial algorithm, we call upon exploiting nogood constraints to remove symmetries on link cliques by dynamically inserting the following new constraint into the model for every newly discovered valid solution.

Given the subset of a complete SIP solution in the form of the set of assignments:

$$\{(v_1, \texttt{match}(v_1)), (v_2, \texttt{match}(v_2)), ..., (v_n, \texttt{match}(v_n))\}$$

for encoded pattern nodes $v \in \phi^P(V_P \uplus E_P)$ that correspond to a support element, all subsequent solutions must adhere to:

$$((v_1 \neq \texttt{match}(v_1)) \vee (v_1 \neq \texttt{match}(p_n)) \vee ... \vee (v_n \neq \texttt{match}(v_n)))$$

This records the assignments of the place graph vertices and closure vertices such that any future solutions found during search with an identical set of assignments will be 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. A disadvantage of this method however is that the solver can only detect clique symmetries using nogoods when all support vertices have already been assigned deep in the search tree when this could be potentially detected earlier. This can also only be applied when the solver already supports dynamic constraints and/or nogood recording. Thus, we look toward a more intuitive way of eliminating clique symmetries using constraints statically defined before search.

Symmetry Breaking Constraints

Symmetry breaking constraints are a type of optimization constraint which disallow any occurrences of symmetrical solutions (i.e. graph symmetries/isomorphisms, matrix rotations) in the returned solution set of a CSP by identifying and preliminarily removing them at propagation time [42]. These are expected to be more performance efficient when compared to a nogood constraint filtering technique in this case, where we require a full candidate matching of support nodes to be found before they can be enforced as opposed to eliminating symmetries earlier in search and reduce the overall search space traversed.

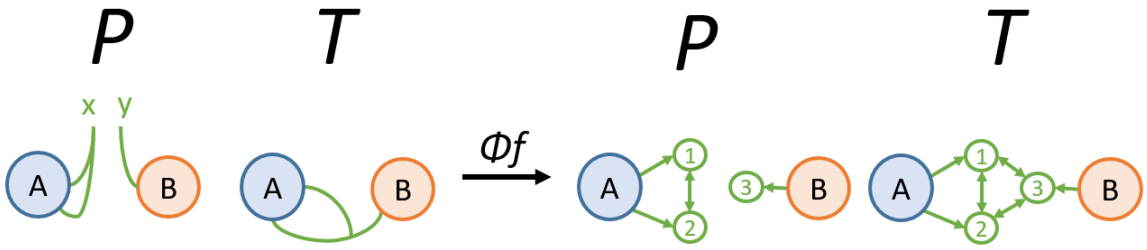


Figure 3.8: A simple example flattened link graph matching instance with explicit identifiers shown on link nodes. There exists one bigraph matching solution $\{(A, A), (B, B)\}$, but two SIP solutions due to both $(L_1, L_1), (L_1, L_1)$ and $(L_1, L_2), (L_2, L_1)$ being possible mappings on link nodes.

Rather than relying on an underlying constraint solver having to support nogoods to remove duplicate solutions introduced by our clique encoding, we employ the more intuitive symmetry breaking approach by introducing additional constraints between pairs of link nodes in the flattened pattern bigraph. We make use of "less than" constraints, which enforce that the value of some pattern node p_1's mapped target id must be strictly less than that of p_2's mapping, to ensure that only one possible permutation of link node assignments within a clique will be considered a valid match.

As seen in Figure 3.8, while one matching solution exists, two SIP solutions will be found due to the symmetry between link nodes L_1 and L_2 in P. It can be observed that these symmetries are introduced whenever multiple ports (and thus encoded link nodes) of the same entity occupies the same hyperedge, so we wish to define a symmetry constraint between all pairs of link nodes that meet this criteria. It is not sufficient to consider only link vertices inside cliques however, as any pair of link nodes which share the same parent entity but do not belong to any clique at all (hanging links where its source has no other connections) will also introduce a symmetry and must be similarly broken. It can be observed in Figure 3.8. that even if the (L_1, L_2) and (L_2, L_1) edges were removed from the flattened pattern, the SIP instance will still return the same two symmetrical solutions. These particular pairs of link node siblings can be considered to be making up a clique of non-edges in this case. We formally introduce our new constraint as

$$\{\forall p = (v, a), q = (v, b) \in P_P \mid (a < b) \wedge ((link(p) = link(q)) \vee$$

$$(|link^{-1}(link(p))| = |link^{-1}(link(q))| = 1))\} \rightarrow$$

$$\texttt{match}(p) = (t, i),\ \texttt{match}(q) = (t, j),\ i < j$$

where $\texttt{match}(p \in P_P)$ indicates the the mapping of the encoded vertex $v_p \in V_P$ modelling p to a target link node $v_t \in V_T$ which models a port $t \in P_T$.

In practice however, this does not need to be applied to all possible pairs of link nodes due

to the transitive property of less-than constraints—i.e. if $p_a < p_b$ and $p_b < p_c$ hold for some clique nodes in a solution then a separate constraint to enforce $a < c$ is unnecessary. Thus for each set of n symmetrical link nodes, only $n-1$ less-than constraints applied in sequence to pairs $\sum_{k=0}^{n-1}(p_k, p_{k+1})$ are sufficient to fully break all possible clique symmetries that occur in the encoding [69].

An additional benefit to using symmetry breaking over the dynamic nogood recording method is that a fully static constraint model would be more compatible with uniform sampling algorithms for approximating solution counts on very large instances, as a potential avenue for future work (Chapter 7).

3.4 Encoding Size

Overall, upon being provided with a pattern and target bigraph, our SIP encoding produces a pattern and target graph with the following number of nodes and edges, where $|e| = |link_G^{-1}(e) \cap P_G|$ is the number of ports that a hyperedge $e \in E_G$ occupies, i.e. their cardinality when excluding faces.

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

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

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

$$
\begin{aligned}
|V| &= |V_T| + n + m + |P_T| + |\widehat{E_T}| \\
|E| &= |V_T| + n + |P_T| + \sum_{e \in E_T} |e| \cdot (|e| - 1) + \sum_{e \in \widehat{E_T}} |e|
\end{aligned}
$$

The SIP encoding hence requires only a number of nodes and edges, scaling linearly for nodes and in the order of $O(|e|^2)$ for edges through our clique representation of hyperedges. This demonstrates an improvement upon the clause generation scalability of $O(|V_P|^2|V_T|^2)$ provided by BigraphER's SAT solver, although this does not yet consider that SAT additionally supports the bigraphs with sharing extension—we later demonstrate however that SIP with sharing does not impact the linear scaling of vertices/quadratic scaling of edges in the encoding and thus this remains a fair comparison (Section 4.2.4).

3.5 Soundness and Completeness

We now provide a proof of soundness and completeness of our algorithm. Soundness is proven by demonstrating that any solution identified by the SIP model corresponds to an instance of a match within bigraph matching. Conversely, completeness is proven by showing that when a solution exists in an instance of bigraph matching, our algorithm will also find a corresponding match in the SIP encoding.

We describe each instance of a solution for bigraph matching and SIP as an injective set of (p, t) pairs denoting the mapping of each pattern support component and vertex to those in the target bigraph and graph respectively. Thus we wish to prove that for any matching solution

$$S_{big} = \{(p_1, t_1), ..., (p_n, t_n)\}$$

the corresponding encoding will produce a solution

$$S_{SIP} = \{(p'_1, t'_1), ..., (p'_n, t'_n)\}^*$$

with a bijective relation between (p_k, t_k) and (p'_k, t'_k), and vice-versa.

We begin by making the following observations about the properties of bigraph composition, which are subsequently used to build our proofs.

Proposition 1. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $v \in V_B$ and $prnt_B(v) \notin k$ then $prnt_B(v) = prnt_G(v)$.

This states that if an entity in B has a non-region parent then it will have the same parent in G, and conversely if it has no parent it will still have no parent in G. This can be simply proven by referring to the definition of place graph composition (Definition 2.3.9) where $prnt_G(v) = prnt_B(v) \iff prnt_B(v) \in V_B$. As a corollary, we deduce that if $prnt_B(v) \notin k$ then $|prnt_B(v)| = |prnt_G(v)|$.

Proposition 2. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

for $v_1, v_2 \in V_B$ if $prnt_B(v_1) = prnt_B(v_2)$ then $prnt_G(v_1) = prnt_G(v_2)$.

*Excluding mappings of port nodes, which can only have up to one permitted combination (Section 3.3.5)

This states that sibling entities in B remain siblings in G. By Definition 2.3.9, if $prnt_B(v_1) = prnt_B(v_2) = v' \in V_B$, then $prnt_G(v_1) = prnt_G(v_2) = v'$. Alternatively, if $prnt_B(v_1) = prnt_B(v_2) = k \in r$ then $prnt_G(v_1) = prnt_G(v_2) = prnt_A(r)$.

Proposition 3. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $v \in V_A$ then $prnt_A^{-1}(v) \subseteq prnt_G^{-1}(v)$.

This states that every child of a vertex in A will also be a child of the same vertex in G. This can again be proven by referring to Definition 2.3.9 where for any entity, $prnt_G(v) = prnt_A(v) \iff v \in A$. As a corollary, we deduce that $|prnt_A^{-1}(v)| \leq |prnt_G^{-1}(v)|$.

Proposition 4. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $v \in V_A$ and $prnt_A^{-1}(v) \cap k = \emptyset$ then $prnt_A^{-1}(v) = prnt_G^{-1}(v)$.

This states that if a vertex in A has no site, then its set of children will be isomorphic to its set of children in G. We know by Proposition 3 that $prnt_A^{-1}(v) \subseteq prnt_G^{-1}(v)$, thus to prove this we hypothesize the presence of an additional entity v' where $prnt_G(v') = v$. If $v' \in V_A$ and $prnt_A^{-1}(v') \neq v$, then by Definition 2.3.9 it follows that $prnt_G^{-1}(v') \neq v$ which is a contradiction. Alternatively, if $v' \in V_B$, then by Definition 2.3.9, $prnt_G(v') \in A$ only if $prnt_B(v') = o \in k$ and $prnt_A(o) = prnt_G(v')$ to allow the bridging of place graphs during composition. Our initial proposition states that v has no site adjacency, therefore $prnt_G(v') \neq v$ always holds, contradicting our hypothesis and proving our proposition by contradiction. As a corollary, we deduce that $|prnt_A^{-1}(v)| = |prnt_G^{-1}(v)|$ when $prnt_A^{-1}(v) \cap k = \emptyset$.

Proposition 5. Given the composition of two link graphs

$$G : X \to Y = (A : Z \to Y) \circ (B : X \to Z)$$

for any $p_1, p_2 \in P_B$, if $link_B(p_1) = link_B(p_2)$ then $link_G(p_1) = link_G(p_2)$.

This states that if two ports share a link in B then they will also share a link in G. This is proven by the definition of link graph composition (Definition 2.3.10), where if $link_B(p_1) = link_B(p_2) = e \in P_B$, then $link_G(p_1) = link_G(p_2) = e$. Alternatively, if $link_B(p_1) = link_B(p_2) = z \in Z$, then $link_G(p_1) = link_G(p_2) = link_A(z)$.

Proposition 6. Given the composition of two link graphs

$$G : X \to Y = (A : Z \to Y) \circ (B : X \to Z)$$

for any $p \in P_B$ and $e \in E_B$, $link_G(p) = e$ if and only if $link_B(p) = e$.

This states that a port in B is linked to an edge if and only if they are also linked in G. Firstly, from Definition 2.3.10, $link_B(p) = e \in E_B \implies link_G(p) = link_B(p)$, therefore $link_G(p) = e$. Secondly, we prove that $link_B(p) = e \impliedby link_G(p) = e$ by hypothesizing a $p \in P_B$, $e \in E_B$ such that $link_G(p) = e$ but $link_B(p) \neq e$. This would instead mean $link_B(p) = x \in (Z \uplus E_B)/\{e\}$. If this were the case, then by link graph construction, either $link_G(p) = x \neq e$ if $x \in E_B$ which is a contradiction, or $link_G(p) = link_A(x)$ if $x \in Z$. However as $link_A(x)$ exists in A, then it cannot be $e \in E_B$ which contradicts our hypothesis and proves our proposition by contradiction. As a corollary, we deduce that $|link_G^{-1}(e) \cap P_G| = |link_B^{-1}(e) \cap P_B|$.

3.5.1 Soundness

Given an instance of bigraph matching (P, T) with a SIP encoding $(\phi_f(\phi_P(P)), \phi_f(\phi_T(T)))$ for which there exists a solution in the form of the injective mapping $S_{SIP} = \{(p'_1, t'_1), ..., (p'_n, t'_n)\}$, we wish to prove that this corresponds to a matching solution $T = C \circ (P \otimes id) \circ D$ in the form of an injective embedded mapping $S_{big} = \{(p_1, t_1), ..., (p_n, t_n)\}$ from all support elements $p \in P$ to a support element $t \in T$, where v'_k is the encoded form of a support element v_k. We prove this by construction, beginning with a SIP encoding with the valid solution S_{SIP}.

We first consider the place graph encodings. The entities of the target place graph $T = (V_T, ctrl_T, prnt_T) : k \to l$ and relations between them can be constructed from $\phi_T(T)$ as follows:

$$\begin{aligned}
&V_T = \{t'_i \in V_{\phi_T(T)} \mid \ell(t'_i) \notin \{\mathbf{root}, \mathbf{site}\}\},\\
&\{\forall t_i \in V_T \mid ctrl_T(t_i) = \ell(t'_i)\}\\
&\{\forall (t_a, t_b) \in V_T \mid (t'_a, t'_b) \in E_{\phi_T(T)} \to prnt(t_b) = t_a\}
\end{aligned}$$

The interface of the target place graph can be constructed by collecting all vertices in $\phi_T(T)$ labelled **root** or **site** into respective numbered lists $R = \{r_1, ..., r_l\}$ and $S = \{s_1, ..., s_k\}$ and adding a corresponding abstraction to T for each such that $(r_i, v') \in E_{\phi_T(T)} \to prnt(v) = i$ and $(v', s_i) \in E_{\phi_T(T)} \to prnt(i) = v$.

The entities of the pattern place graph $P = (V_P, ctrl_P, prnt_P) : m \to n$ and their relations

can then be constructed from $\phi_P(P)$ as follows:

$$V_P = V_{\phi_P(P)}$$
$$\{\forall p_i \in V_P \mid ctrl_T(p_i) = \ell(p'_i)\}$$
$$\{\forall (p_a, p_b) \in V_P \mid (p'_a, p'_b) \in E_{\phi_P(P)} \to prnt(p_b) = p_a\}$$

The interface of P can then be constructed by looking at degree constraints on vertices. Any vertex p' which has no degree constraint on its in-degree indicates $prnt(p) = r \in n$, and if it has a $\geq$ constraint on its out-degree then $prnt(o \in m) = p$. Additionally, if for any pair $(p'_a, p'_b) \in V_{\phi_P(P)}$ which both vertices have no in-degree constraint and $prnt(t_a) = prnt(t_b)$, then $prnt(p_a) = prnt(p_b) = r \in n$. This is sufficient to construct the n regions and m sites of P.

We now build C and D in adherence to the compositional property of place graphs in Definition 2.3.9 as follows:

1. $\{\forall p_i \in V_P \mid prnt_P(p_i) = r \in n\} \to prnt_T(t_i) \in V_C, prnt_C(r) = t_i$
2. $\{\forall p_i \in V_P, t_j \in V_T \mid prnt_P^{-1}(p_i) \cap m = s, prnt(t_j) = t_i, t'_j \notin S_{SIP}\} \to$
   $t_j \in V_D, prnt_D(t_j) = m$
3. $\{\forall t_i \in V_T \mid prnt_T(t_i) \in V_D\} \to t_i \in V_D, prnt_D(t_i) = prnt_T(t_i)$
4. $\{\forall t_i \in V_T \mid t_i \notin V_P, t_i \notin V_D\} \to t_i \in V_C$
5. $\{\forall t_i \in V_C\} \to prnt_C(t_i) = prnt_T(t_i)$

This is sufficient to build the $T = C \circ (P \otimes id) \circ D$ decomposition on the place graph, retaining all parent relations.

We now consider the flattened link graphs. A link graph $G = (V_G, E_G, ctrl_G, link_G) : X \to Y \in \{P, T\}$ can be constructed from $\phi_f(G)$ as follows.

$$V_G = \{g' \in \phi_f(G) \mid \ell(g') \notin \{\mathbf{link}, \mathbf{closure}\}\}$$
$$P_G = \{g' \in \phi_f(G) \mid \ell(g') = \mathbf{link}\}$$
$$E_G = \{g' \in \phi_f(G) \mid \ell(g') = \mathbf{closure}\}$$
$$\{\forall g \in V_G \mid ctrl_G(g) = \ell(g')\}$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G)} \mid \ell(g'_a) = \ell(g'_b) = \mathbf{link}\} \to link_G(g_a \in P_G) = link_G(g_b \in P_G)$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G)} \mid \ell(g'_a) = \mathbf{link}, \ell(g'_b) = \mathbf{closure}\} \to link_G(g_a \in P_G) = g_b \in E_G$$

As we discard idle edges for matching, we only need to rebuild the outer face of G, which can be constructed by adding a new outer name as a sink for each $g \in P_G$ where $link_G(g) \notin E_G$,

respecting the property that ports share a name when their flattened nodes share a clique. This is sufficient to build the link graphs of P and T.

As the place graph construction already assigns all target entities to either C, P or D, we assume this has been already performed. We now build the links of C and D in adherence to the compositional property of link graphs in Definition 2.3.10 as follows:

- $\{\forall p \in P_P \mid link_P(p) = y \in Y\} \rightarrow link_C(y) = link_T(p)$
- $\{\forall p \in P_D \mid link_T(p) \notin D\} \rightarrow link_D(p) = x \in X, link_{id}(x) = x' \in Y,$ $link_C(x') = link_T(p)$
- $\{\forall p \in P_C\} \rightarrow link_C(p) = link_T(p)$
- $\{\forall e \in E_D\} \rightarrow link_D^{-1}(e) = link_T^{-1}(e)$

This thus builds the $T = C \circ (P \otimes id) \circ D$ decomposition on the ports and edges of the link graph. This concludes our proof by construction.

3.5.2 Completeness

Given an instance of bigraph matching (P, T) where $T = C \circ (P \otimes id) \circ D$ for solid bigraphs P and T where $|P| > 0$ and $|T| > 0$, and there exists a solution in the form of an injective embedded mapping $S_{big} = \{(p_1, t_1), ..., (p_n, t_n)\}$ from all support elements $p \in P$ to a support element $t \in T$, we wish to prove that a parallel solution $S_{SIP} = \{(p'_1, t'_1), ..., (p'_n, t'_n)\}$ exists in the SIP instance $(\phi_f(\phi_P(P)), \phi_f\phi_T(T))$, where v'_k is the encoded form of a support element v_k.

Assume that there exists a valid composition $T = C \circ (P \otimes id) \circ D$ with an embedding $S_{big} = \{(p_1, t_1), ..., (p_n, t_n)\}$, where the corresponding SIP solution $S_{SIP} = \{(p'_1, t'_1), ..., (p'_n, t'_n)\}$ is not a valid solution. This suggests that at least one of our defined model constraints are being violated.

We first consider place graph constraints. By construction, every parent relation between entities $prnt(v_b) = v_a$ in P and T is maintained through building the edge (v'_a, v'_b) in E_P and E_T, thus conventional SIP rules hold. By trivial inspection, the label compatibility function (p'_k, t'_k) will always return **true** for any valid embedding (p_k, t_k) as controls are preserved. Propositions 1, 3 and 4 accordingly prove that the three cases considered by the degree compatibility function will always return **true** for the encodings of all possible $(p_k, t_k) \in S_{big}$. Proposition 2 additionally proves that the extra constraint introduced to deal with shared regions can never be violated. Thus we conclude S_{SIP} must satisfy all constraints on the place graph.

We next consider the link graph flattening. Via conventional SIP rules, if there is an edge between flattened port nodes $(v'_a, v'_b) \in E_P$ in a clique encoding then there must also exist the edge $(\texttt{match}(v'_a), \texttt{match}(v'_b)) \in E_T$. Proposition 5 proves that all ports which share a hyperedge in P will always share a hyperedge in T, adequately reflecting this property and thus this will always hold true. Trivially, the label compatibility function (p'_k, t'_k) will always return **true** on all flattened edge nodes as $p_k \in E_P \iff t_k \in E_P$. Finally, Proposition 6 proves that the degree constraint on closure nodes will always return **true** for any valid $(p_k, t_k) \in S_{big}$ bigraph embedding on edges. This thus exhausts all extra constraints in the SIP model.

Our original hypothesis that a constraint violation occurs is shown to be a contradiction, and therefore S_{SIP} must be a valid SIP solution. This concludes the proof.

3.6 Summary

In this chapter we have presented our encoding for the bigraph matching problem into a format which can be described as SIP with additional constraints, to allow for efficient solving via a dedicated subgraph solving framework. In Section 3.1, we discussed our reasoning as to why this would be an intuitive and efficient way to ease the bottlenecks introduced by the current state-of-the-art SAT encoding. In Section 3.2 we demonstrate our proposed encoding of the place graph components of the pattern-target bigraph pair, providing algebraic expressions for each, as well as identifying the single edge case in our encoding which results in false positives and introduce an extra constraint to eliminate these. In Section 3.3 we demonstrate the same for the pattern and target link graphs, observe that our encoding introduces new clique symmetries, and provide two methods for handling and preliminarily removing these from the solution set. Section 3.4 gives the size of the resultant encoding in relation to the size of the input. In Section 3.5, we provide a formal proof of soundness and completeness of our algorithm.

Up until now, we have only covered matching for pure bigraphs—that is, bigraphs with no modification to the original specification of the data structure as described by Milner. The following chapter covers three extensions to the bigraph matching problem, which we can adapt our algorithm to solve with the addition of optional further constraints to handle new potential cases introduced by their added complexity.

Chapter 4

Bigraph Matching on Extensions

This chapter introduces two commonly used variations of the bigraph structure— *bigraphs with sharing*, which allow entities and sites to have multiple parents in the place graph, and *directed bigraphs* which assign polarity to hyperedges and interfaces in the link graph. We then demonstrate how we are able to add further optional constraints onto our original SIP model to allow support for these extensions without the need to modify our existing encoding, demonstrating the extensibility and flexibility of our approach. We also add support for checking *equality* between bigraphs using similar methods. To distinguish between the original bigraph formalism and these variants, we denote bigraphs without extensions as *pure bigraphs.*

In Section 4.1, we describe how we can extend our encoding method to support checking for equality (support equivalence) between two bigraphs. Section 4.2 introduces bigraphs with sharing, and the additional edge cases it introduces which leave our current encoding under-constrained. We then describe how we add additional constraints to cover these cases, and provide a formal proof of such. Section 4.3 similarly covers directed bigraphs, and the additional constraints required to model the increased complexity introduces by direction in the hyperedges of the link graph, also with an accompanying formal proof. Concluding remarks are provided in Section 4.4.

4.1 Bigraph Support Equivalence

When two bigraphs G_1 and G_2 are support-equivalent, then the set of solutions returned upon performing matching on them can be considered as analogous to the set of valid support translations between them, provided that both interfaces of each graph are also matching i.e. the same orderings of regions/sites and naming of open links are adjacent to support-equivalent entities. Following on from this, they can be considered functionality identical

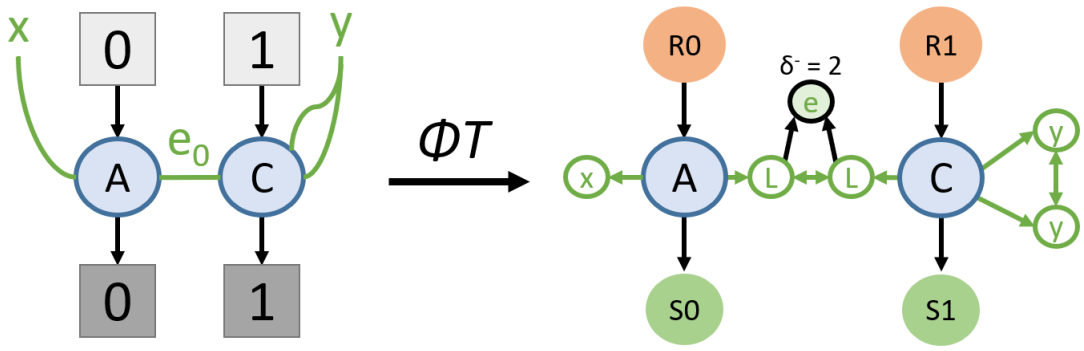


Figure 4.1: Example instance of an equivalence check encoding, where each open link node and abstraction are assigned unique types to ensure matching interfaces.

in any composition—for any pair of compositions on a bigraph $F \circ G_1$ and $F \circ G_2$ (or vice versa if composing above F), both resultant bigraph states will also share this equivalence property. The notion of equality is important to define as transition systems must perform this check for all rewritten states against each bigraph already present in the system whenever a match is found, to create loop-backs and ensure no duplicate nodes are added.

Checking for equivalence in this manner can be considered the bigraph version of *graph isomorphism*. In our current SIP encoding, two support equivalent bigraphs will produce a pair of isomorphic graphs, however further constraints are required to guarantee the equality property upon their interfaces.

4.1.1 Encoding

To model equality checking between two bigraphs G_1 and G_2, their place graphs are encoded and flattened using the target encoding function ϕ_T as described in Section 3.2.1, since we now wish to explicitly match all regions and sites. To ensure that equality holds, we must introduce the following six additional constraints to their encoded forms that must all be true:

1. $|V_{G_1}| = |V_{G_2}|$
2. $|E_{G_1}| = |E_{G_2}|$
3. $\{\forall i \in n \mid i = \texttt{match}(i)\}$
4. $\{\forall j \in m \mid j = \texttt{match}(j)\}$
5. $\{\forall x \in X \mid x = \texttt{match}(x)\}$
6. $\{\forall y \in Y \mid y = \texttt{match}(y)\}$

Constraints 1 and 2 enforce that a bijective mapping must exist between the two bigraphs by ensuring that the same number of support components exist in both. This can be trivially determined before running the main search process by comparing the number of non-link vertices and closure vertices in each encoding. Constraints 3 to 6 enforce that the identifiers of each mapping of interface components must also be the same, e.g. region 1 in P_{G_1}

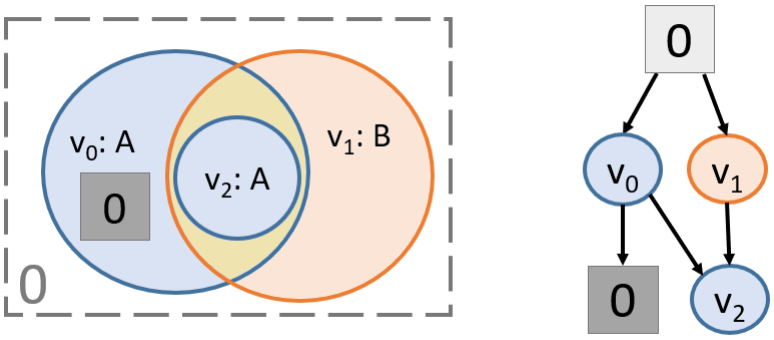


Figure 4.2: Example instance of a concrete place graph with sharing represented as (a) its diagrammatic form, where the shared area is shaded yellow and (b) as a DAG. Colors denote control type.

matches to region 1 in P_{G_2} and open link name x in L_{G_1} matches to x in L_{G_2}. For sites and regions, we can encode this by giving a unique label/control to each region and site vertex based on its ordinal value. Similarly, for each open link, we can also assign each vertex in their corresponding flattened clique its own label that denotes which face they point toward. Making use of labels in this way ensures that all of the required constraints can be enforced as a pre-process. As long as these constraints hold then the SIP solver can then search for a match in order to determine whether the support equivalence is preserved. An example encoding is shown in Figure 4.1.

When we search for all solutions using this encoding, the solver will also identify all intra-symmetries that exist between sets of entities in the shared structure in the form of distinct solution mappings. When doing bigraph matching, we can remove all symmetries/isomorphic solutions by running this equality check on the pattern against itself to identify all intra-symmetries that exist, then check these against the list of matches, removing any that have identical mappings aside from the sets of assignments included in the identified intra-symmetry. This is useful in contexts where we only wish to enumerate all *unique* solutions.

4.2 Bigraphs with Sharing

4.2.1 Definition

The bigraphs with sharing formalism was first introduced by Sevegnani et al. [11] as a generalization of Milner's original formalization, where the $prnt$ component of the place graph is modified to a be a set of unique pairs that denote child-parent relations between entities and interfaces, rather than a single mapping for each entity and abstract region. This allows for entities and sites to belong to multiple parents, thus meaning they can now spatially overlap

as shown in Figure 4.2. The place graph as a result is now a directed acyclic graph (DAG) rather than a forest.

We formally define this generalization in the concrete context as follows:

Definition 4.2.1 (Concrete place graph with sharing)**.** A *concrete place graph with sharing*

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

is a triple which has the inner face m and outer face n, indicating m sites and n regions. 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 $prnt$.

Composition for directed place graphs is performed in a process similar to pure bigraphs (Definition 2.3.9), however as $prnt(v)$ now defines a set of values rather than a single element, $prnt_G(v)$ for each $v \in o \uplus V_B$ can be obtained by performing the conditional check on *each* $v' \in prnt_B(v)$ in turn and adding the result to $prnt_G(v)$. As sites can now also have multiple parents, *all* parent entities of a site $prnt_A(r \in k) \subseteq V_A$ will be added to the parent set of the entity with $prnt_G(v \in V_B)$ during composition when it has the corresponding region parent $prnt_B(v) = r$. We formally define this as follows.

Definition 4.2.2 (Concrete place graph with sharing composition)**.** Given two concrete place graphs with sharing $A : m \to n$ and $B : o \to m$ with disjoint supports, we define the composite place graph with sharing as

$$A \circ B = (V, ctrl, prnt) : o \to n$$

Where $V = V_A \uplus V_B$, $ctrl = ctrl_A \uplus ctrl_B$, and the $prnt$ value of each site and entity $v \in o \uplus V$ depends on the following conditions.

$$prnt(v) \stackrel{\text{def}}{=} \begin{cases} prnt_B(v) \text{ if } v \in o \uplus V_B \text{ and } prnt_B(v) \cap n = \emptyset \\ prnt_B(v) \uplus prnt_A(r) \text{ if } v \in o \uplus V_B \text{ and } prnt_B(v) \cap n = r \\ prnt_A(v) \text{ if } v \in V_A \end{cases}$$

The sharing extension modifies only the definition of the place graph component of a bigraph, so we can re-use the definition of the link graph provided in Chapter 2 to build the full definition of a concrete bigraph. This also means we do not have to take into account the link graph or our flattened representation of it when later extending our model to support sharing.

Definition 4.2.3 (Concrete bigraph with sharing)**.** A concrete bigraph with sharing

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

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 $B^{\mathsf{L}} = (V_B, E_B, ctrl_B, link_B) : X \to Y$.

As an addendum, our definition of a solid bigraph G in Chapter 2 is also further restricted to account for the new symmetrical behavior of the place graph interface. The following criteria must now also be met:

- For any site $s \in m$, $prnt(s) \neq \emptyset$.
- For any two regions $r, o \in n$, $prnt^{-1}(r) \cap prnt^{-1}(o) = \emptyset$.

Allowing space to be shared crucially expands the scope of what can be modeled using bigraphs, as many potential scenarios may involve spatial overlap—for example, modelling a device which is connected to multiple local area networks or in physical range of multiple wireless signals is less intuitive and more difficult when the spatial hierarchy restricts each entity or location to only a single container. Bigraphs with sharing in particular have been utilized to model Mixed-Reality systems [41], wireless networking protocols [5] and real-time verification of wireless sensor networks [2].

Another key motivation for supporting bigraphs with sharing in our algorithm is that BigraphER's solver also supports the sharing property as mentioned in Section 2.7.1. Thus this must be solved in order to fully integrate the SIP solver and evaluate the performance of our approach against the previously used solver as described in Chapter 5.

Our current method of encoding the pattern and target place graphs does not require any modification to be able to support instances of bigraphs with sharing: as it stands, any possible match in a matching instance with sharing will already be correctly identified by the SIP solver. However, sharing introduces new vertex patterns and interface interactions which will result in the solver returning a superset of the correct solution set of a matching problem—that is, further constraints must be enforced to remove the false positives introduced by the sharing property.

An observation that can be made is that almost all of the complexities introduced by sharing are caused by sharing involving regions and sites. For sharing that exists exclusively between entities, there is no need to make any changes or additions to our logic as we simply encode the place graph pair as DAGs rather than forests which is sufficient to allow the SIP solver to find a match.

A preliminary version of our algorithm for sharing has been presented in our CP2021 publication [15].

4.2.2 New Place Compatibility Function

With the introduction of sharing, it is now possible for an entity or site to have multiple parents. We thus update our place compatibility function to reflect this, by enforcing an additional degree constraint on the in-degree of entities similarly to how we already encode children of regions. This causes the compatibility function to now be fully symmetric between regions and sites.

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

4.2.3 Additional Constraints

To support sharing we must also ensure that we consider and enforce the following new composition constraints identified by Sevegnani [70]. Instances involving sharing introduce the new risk that even if a match seemingly exists that respects all region and site constraints individually, the returned solution does not respect the form $T = C \circ P \circ D$ since there may be adjacencies that loop from the parameter back into the context, i.e. produce cycles in the graph which violate the DAG property of the place graph. We denote this phenomenon as a *transitive closure* violation, and is caused whenever for a candidate solution, an entity in the target abstracted to a site can transitively reach an entity abstracted to a region through its descendants.

In addition to transitive closure, we must also consider that sites can now potentially have multiple parents. This introduces two complications: a region in P may now encapsulate any number of entities in the context rather than just zero or one, and a new constraint must also be applied to shared sites which is symmetrical to our adapted region constraint. Albeit redundant, since sharing is a generalization of the standard bigraph formalism, these constraints can also be safely applied to standard bigraph matching as well as bigraphs with sharing without potentially invalidating any matches that would otherwise be found.

Transitive Closure

Figure 4.3 shows an example instance that is now possible due to sharing where it seems there should be a valid match via conventional SIP rules, but where it is impossible to decompose T to produce P in a manner which allows the remaining entities to be split into a valid context

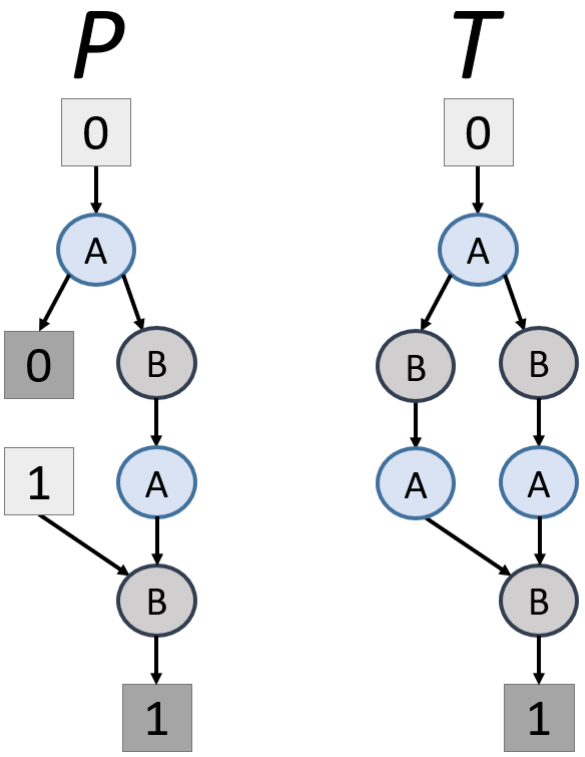


Figure 4.3: Example instance showing need for transitive closure to avoid the parameter also appearing as a context. No match is possible in this instance.

and parameter. Regardless of which of the two possible matches we choose, the remaining two 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. Therefore, our model must disallow candidate solutions that produce a transitive path which leads from a an entity abstracted to a site (parameter) back to an entity abstracted to a region (context) in the set of unmapped entities of T.

To enforce this constraint, we preliminarily compute a descendants map $\tau(u, v) : V_T \times V_T \to \{t, f\}$ between all possible pairs of entities in the target, which is true if and only if $(u, v) \in prnt_T^+$—that is, whether v can be reached when beginning from u. Whenever a candidate solution is then found in the matching instance, the following post-processing constraint is applied:

$$\{\forall(t \in V_T, p \in V_P) \,|\, \texttt{match}^{-1}(t) = \emptyset, prnt_P(s \in m) = prnt_T(t)\} \to \tau(t, \texttt{match}(p)) = f$$

where $\texttt{match}^{-1}(t) \in V_P$ is the pre-image mapping of t in the potential solution. This enforces that no top-level unmatched target entities assigned to a site (and thus the parameter) are allowed to be able to transitively reach any target entities that occur in the pattern, ensuring all solutions returned respect bigraph composition rules.

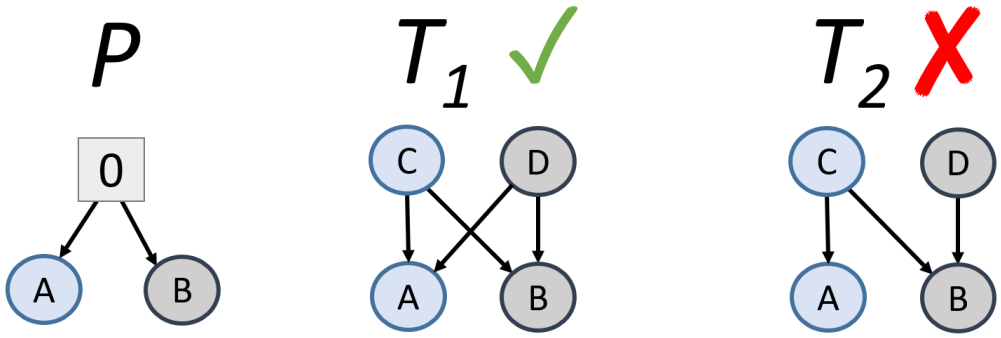


Figure 4.4: An updated example matching instance where sharing between entities in T is now possible. A match can only exist if every entity encapsulated by the shared region individually connects to the full set of its mapped children.

Shared Regions

We must update our region constraint defined in Section 3.2.4 to account for two new possible configurations: as shown visually in Figure 4.4, an entity may now have multiple parent entities in addition to a region simultaneously, and a region can now potentially encapsulate more than one entity in the context. We therefore specify that this constraint only concerns unmatched parents of matched nodes in the target, and that the shared region must capture *only*, and all of these parents.

$$\{\forall p_a, p_b \in V_P \mid (r \in n) \in prnt_P(p_a) \cap prnt_P(p_b)\} \rightarrow$$

$$prnt_T(\texttt{match}(p_a))/prnt_P(p_a) = prnt_T(\texttt{match}(p_b))/prnt_P(p_b)$$

where $prnt_T(\texttt{match}(v))/prnt_P(v)$ is the set of unmapped parents of the mapping of an encoded entity $v \in V_P$.

Similarly to the non-sharing case, this can be checked and verified in polynomial time as a post-processing constraint by inspecting all top-level nodes and whether they can suitably slot into a region.

Shared Sites

As there can now also be occurrences of shared sites, these must be dealt with in a similar fashion. Figure 4.5 shows the second required sharing constraint, which is symmetric to the newly defined required region constraint for sharing. As the site is shared it must include *only*, and all, entities shared by the two parents. In this example that means there cannot exist any entities which are a child of one B entity but not the other. We define this symmetric

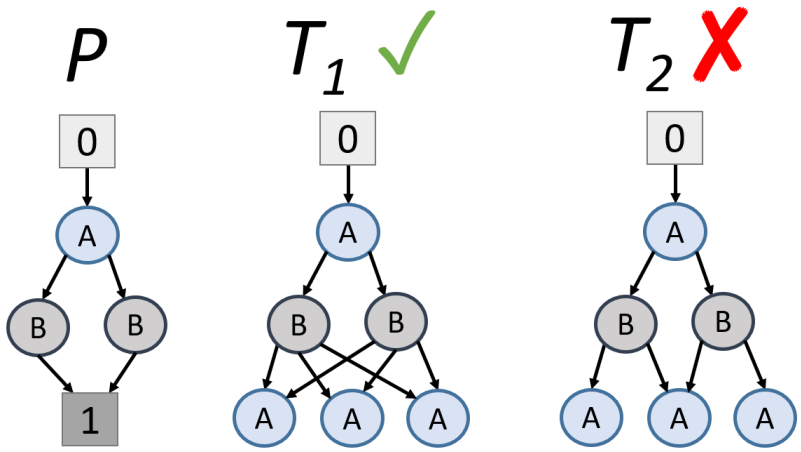


Figure 4.5: (P, T_1) is a matching instance which includes a shared site. A valid mapping exists as all three A leaf nodes and their edges can be encapsulated by the site. (P, T_2) is a similar matching instance with no valid solution, where our encoding will return a false positive without a further checking constraint to identify and remove the candidate solution.

interface constraint as

$$\{\forall p_a, p_b \in V_P \mid (s \in m) \in prnt_P^{-1}(p_a) \cap prnt_P^{-1}(p_b)\} \rightarrow$$

$$prnt_T^{-1}(\texttt{match}(p_a))/prnt_P^{-1}(p_a) = prnt_T^{-1}(\texttt{match}(p_b))/prnt_P^{-1}(p_b)$$

where $prnt_T^{-1}(\texttt{match}(v))/prnt_P^{-1}(v)$ is the set of unmapped children of the mapping of an encoded entity $v \in V_P$.

4.2.4 Sharing Encoding Size

The pattern graph size remains the same with the introduction of sharing, as the value of $\delta^-(v)$ for any $v \in V_P$ will still capture any instance of a vertex's in-degree being above one. The total number of edges in the target graph encoding is slightly modified as follows.

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

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

We no longer assume that the in-degree of each site is only one, thus we instead sum the value of $\{\delta^-(i) \mid i \in V_T \uplus n\}$ to obtain the total number of encoded place graph edges. This permits more potential adjacencies in the SIP encoding, however this remains a linear

relation between the number of parent relations in the bigraph and encoded edges. Therefore edges will remain scaling quadratically in the order of $O(|e|^2)$ when sharing is possible, improving upon the scalability of BigraphER's SAT clause generation.

4.2.5 Soundness and Completeness

We now demonstrate a formal proof of soundness and completeness for our adapted encoding. This builds upon our proof for pure bigraphs as provided in Section 3.5. We observe that our six initial propositions still hold with the introduction of sharing, and can be reused in the context of sharing which we now demonstrate.

Consider the composition of two place graphs with sharing:

$$G : m \in n = (A : k \in n) \circ (B : m \to k)$$

Proposition 1 can be proven for *each* parent $v' \in prnt_B(v)$ of an entity $v \in V_B$ by Definition 4.2.2, and thus the corollary $|prnt_B(v)| = |prnt_G(v)|$ remains true overall in this case. Proposition 2 remains true for equal parent sets by evaluating each relation separately, but can now be generalized in the case where $prnt_B(v_1) \neq prnt_B(v_2)$ but $prnt_B(v_1) \cap prnt_B(v_2) \neq \emptyset$ (Proposition 8). Proposition 3 still holds as $v \in V_A \to prnt_A(v) = prnt_G(v)$ remains true for sharing. Proposition 4 can be proven for sharing through the same method of contradiction by demonstrating there can be no hypothetical entity v' where $(v', v) \in prnt_G$. Finally, Propositions 5 and 6 still hold as they relate to the link graph which is unaffected by sharing.

In addition, we propose some new observations specifically for proving correctness in the context of sharing as follows.

Proposition 7. Given the composition of two place graphs with sharing

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $v \in V_B$ then $prnt_B(v) \subseteq prnt_G(v)$.

This states that every parent of an entity in V_B will always remain a parent in V_G. We prove this by hypothesizing a potential pair of entities $v, v' \in V_B$ where $v' \in prnt_B(v)$ but $v' \notin prnt_G(v)$, thus $prnt_B(v) \subseteq prnt_G(v)$ cannot be true. By Definition 4.2.2, when $prnt_B(v) \subseteq prnt_G(v)$ is false then $v \in V_A$, which contradicts our original assumption. As a corollary we thus deduce that $|prnt_G(v)| \geq |prnt_B(v)|$.

Proposition 8. Given the composition of two place graphs with sharing

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

for any two entities $v_1, v_2 \in V_B$, if $(r \in k) \in prnt_B(v_1) \cap prnt_B(v_2)$ then $prnt_G(v_1)/prnt_B(v_1) = prnt_G(v_2)/prnt_B(v_2)$.

This states that if any two entities share a region r in B then both entities will share the same set of A parents in G. By Definition 4.2.2, we can determine that $prnt_G(v_1) = prnt_B(v_1) \uplus prnt_A(r)$ and $prnt_G(v_2) = prnt_B(v_2) \uplus prnt_A(r)$, thus $prnt_G(v_1)/prnt_B(v_1) = prnt_G(v_2)/prnt_B(v_2) = prnt_A(r)$, and therefore this holds by inspection.

Proposition 9. Given the composition of two place graphs with sharing

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

for any two entities $v_1, v_2 \in V_A$, if $(s \in k) \in prnt_A^{-1}(v_1) \cap prnt_A^{-1}(v_2)$ then $prnt_G^{-1}(v_1) /prnt_A^{-1}(v_1) = prnt_G^{-1}(v_2)/prnt_A^{-1}(v_2)$.

This states that if any two entities share a site s in A then both entities will share the same set of B children in G. We prove this by hypothesizing an entity $v' \in B$ such that $s \in prnt_B(v')$ but $\{v_1, v_2\} \notin prnt_G(v')$. By Definition 4.2.2, since $\{v_1, v_2\} \in prnt_A(s)$, then this can only be true if either $prnt_B(v') \cap k = \emptyset$ or $v' \in A$ which both contradict our initial hypothesis. Thus this provides a proof by contradiction.

Soundness

We prove soundness by construction in a similar manner to our method for proving soundness for pure bigraphs (Section 3.5.1). The entities of P and T are rebuilt in the same way, with the minor adjustment that their parent relations are now constructed as follows:

$$\{\forall (v_a, v_b) \in (G \in \{V_P, V_T\}) \mid (v'_a, v'_b) \in E_{\phi_G(G)} \to (v_b, v_a) \in prnt_G\}$$

The interface of T is also rebuilt using the same method for pure bigraphs, with the minor adjustment that $(v, s_i) \in E_{\phi_T(T)} \to (s_i, v) \in prnt_T$.

The interface of P is again reconstructed by examining the presence of degree constraints on vertices. When a $\geq$ exists on the in-degree it indicates that a vertex has a region parent, while a $\geq$ on the out-degree indicates the presence of a child site. Shared abstractions are added to the structure by inspecting these vertices and their mappings in the solutions via the following:

$$\{\forall p_a, p_b \in V_P \mid prnt(t_a)/prnt(p_a) = prnt(t_b)/prnt(p_b)\} \to$$

$$(p_a, (r \in n)) \in prnt_P, (p_b, r) \in prnt_P$$

$$\{\forall p_a, p_b \in V_P \mid prnt^{-1}(t_a)/prnt^{-1}(p_a) = prnt(t_b)^{-1}/prnt^{-1}(p_b)\} \to$$

$$((s \in m), p_a) \in prnt_P, (s, p_b) \in prnt_P$$

Any remaining vertex with a $\geq$ constraint is given a lone respective region/site. This is sufficient to construct the n regions and m sites of P, and thus builds the full place graphs with sharing for P and T.

We now build C and D in adherence to the compositional property of place graphs with sharing in Definition 4.2.2 as follows, taking into account that sharing now potentially allows context and parameter entities to connect through the identity.

1. $\{\forall p_i \in V_P, t_j \in V_T \mid prnt_P(p_i) \cap n = r, t_j \in prnt(t_i), t'_j \notin S_{SIP}\} \rightarrow$
   $t_j \in V_C, (n, t_j) \in prnt_C$
2. $\{\forall p_i \in V_P, t_j \in V_T \mid prnt_P^{-1}(p_i) \cap m = s, t_i \in prnt(t_j), t'_j \notin S_{SIP}\} \rightarrow$
   $t_j \in V_D, (t_j, m) \in prnt_D$
3. $\{\forall t_i \in V_T \mid prnt_T(t_i) \cap V_D \neq \emptyset\} \rightarrow t_i \in V_D$
4. $\{\forall t_i \in V_T \mid t_i \notin V_P, t_i \notin V_D\} \rightarrow t_i \in V_C$
5. $\{\forall t_i, t_j \in V_T \mid (t_j, t_i) \in prnt_T, \{t_i, t_j\} \subseteq V_D\} \rightarrow (t_j, t_i) \in prnt_D$
6. $\{\forall t_i, t_j \in V_T \mid (t_j, t_i) \in prnt_T, \{t_i, t_j\} \subseteq V_C\} \rightarrow (t_j, t_i) \in prnt_C$
7. $\{\forall t_i, t_j \in V_T \mid (t_j, t_i) \in prnt_T, t_i \in T_C, t_j \in V_D\} \rightarrow (t_j, x \in n_{id}) \in prnt_D,$
   $(x, x' \in m_{id}) \in prnt_{id}, (x', t_i) \in prnt_C$

Thus this builds the $T = C \circ (P \otimes id) \circ D$ decomposition on the place graph with sharing. As link graphs are unaffected by sharing, the reconstruction for these remain the same, and this concludes our proof by construction.

Completeness

Completeness similarly to pure bigraphs in Section 3.5.2 is proven by contradiction, by observing that if S_{SIP} is not a valid solution then at least one constraint in the SIP model must be violated. As all parent relations are preserved between entities by building the edge (v'_a, v'_b) in E_G for each $(v_b, v_a) \in prnt_G$, $G \in \{P, T\}$, and Propositions 1 to 6 remain true, all constraints from the original SIP model must hold—therefore we must consider each new sharing constraint in the adapted model.

Proposition 7 proves that the new $\geq$ in-degree constraint on vertices connected to a region will always return **true** for all possible embeddings $(p, t) \in S_{big}$ where this occurs. Proposition 8 and Proposition 9 additionally prove that the adapted shared region constraint and

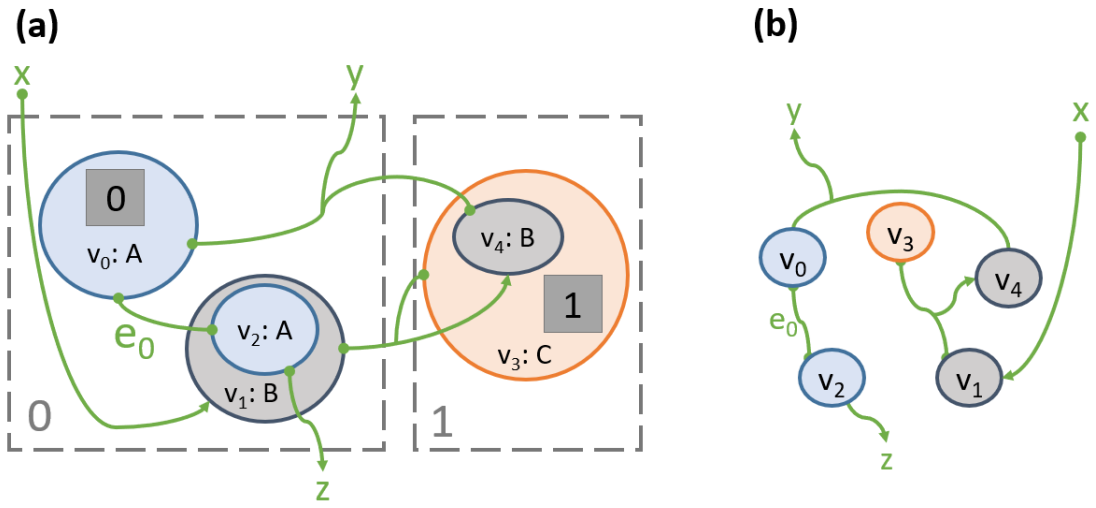


Figure 4.6: (a) An example of a directed bigraph. (b) The separated link graph which shows that each port and face now has an additional inward/outward direction property, and negative ports can act as sinks in addition to outgoing faces and closed edges.

new shared site constraint respectively can never be violated. Finally, for any valid S_{big} embedding, our transitive closure constraint must always return **true** as the definition of place graphs with sharing (Definition 2.3.15) specifies that the structure is always acyclic. This demonstrates that no constraint can be violated, and thus if there exists a solution S_{big} then the encoding will find the responding S_{SIP} solution in a match by contradiction. This concludes the proof.

4.3 Directed Bigraphs

Directed bigraphs, first proposed by Grohmann et al. [12] are a generalization of the link graph component of bigraphs, in a similar manner to how sharing generalizes the place graph. Directed bigraphs introduce direction to a bigraph's hyperedges by designating an additional inward or outward polarity property to each port of an entity and name, as shown in Figure 4.6. These permutations of directions, when constructed into a hyperedge, can be represented as a set of directed edges pointing either to or from a shared central non-existent node for each link.

Introducing polarity to the link graph can allow for more expressive and intuitive modelling compared to standard bigraphs, particularly in cases where it is important to model the flow of data between entities and interfaces. Polarity also allows for the representation of asymmetric dependencies between entities, which would not be possible in pure bigraphs without the use of additional "dummy" nodes or edges [63]. Directed bigraphs have been employed to build models for cybersecurity protocols [71], molecular biology [72] and cloud comput-

ing [73].

Formally, direction is introduced through two key changes in our formalization of bigraphs. Firstly, the interfaces of a link graph $B_L : X \to Y$ are partitioned into two disjoint sets (X^+, X^-) and (Y^+, Y^-): X^+ and Y^+ represent names where their link adjacency points "upward" (outgoing from X and incoming from Y respectively), and vice versa for X^- and Y^- which represent "downward" facing names. Secondly, the signature of the bigraph is substituted with a *polarized signature* as follows.

Definition 4.3.1 (Polarized signature)**.** A *polarized signature* $(\mathcal{K}_P : ar)$ defines the set of controls $\mathcal{K}_P$ in a directed bigraph and each of their corresponding mappings $ar : \mathcal{K}_P \to \mathbb{N} \times \mathbb{N}$ to a pair of non-negative arity values. $ar^+ : \mathcal{K}_P \to N$ and $ar^- : \mathcal{K} \to N$ define the set of positive and negative ports of controls respectively in ar. A bigraph over $\mathcal{K}_P$ assigns every entity a control $(m, n) \in \mathcal{K}_P$ which in turn assigns that entity $ar^+(m)$ outgoing (positive) link ports and $ar^-(n)$ incoming (negative) link ports.

We then use these to formalize the concrete instance of a directed link graph as described by Grohmann et al. [63].

Definition 4.3.2 (Concrete directed link graph)**.** A *concrete directed link graph*

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

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

$$\begin{aligned} link_B : X^+ \uplus Y^- \uplus P_B^+ &\to E_B \uplus X^- \uplus Y^+ \uplus P_B^-, \\ P_B^+ &\stackrel{\text{def}}{=} \{(v, i) \mid v \in V_B,\ i = ar^+(ctrl_B(v))\}, \\ P_B^- &\stackrel{\text{def}}{=} \{(v, i) \mid v \in V_B,\ i = ar^-(ctrl_B(v))\} \end{aligned}$$

where P_B^- and P_B^+ are the sets of outgoing and incoming ports of B respectively. The link map is also subject to the constraints

$$\begin{aligned} \{\forall x \in X^-, \forall y \in X^+ \mid link(y) = x\} &\to link^{-1}(x) = y \\ \{\forall x \in Y^+, \forall y \in Y^- \mid link(x) = y\} &\to link^{-1}(y) = x \end{aligned}$$

These enforce that only "U-turn" mappings are permitted when directly connecting incoming and outgoing names on the same inner or outer face respectively, with no other incoming adjacencies allowed to join the link. This is to preserve a consistent definition of the link

graph during compositions, as otherwise this could potentially introduce invalid "loop" patterns in the composite link and hence an invalid link graph. In the initial definition of directed bigraphs provided in [12], a more strict version of this constraint which disallows any link mapping to the same face at all was applied to prevent this from occurring, however this has later evolved to the pair of constraints we describe as presented in [74], which is the definition that we will be using for this dissertation.

Composition for link graphs is defined as follows.

Definition 4.3.3 (Concrete directed link graph composition)**.** Given two concrete link graphs $A : (Z^+, Z^-) \to (Y^+, Y^-)$ and $B : (X^+, X^-) \to (Z^+, Z^-)$ with disjoint supports, we define the composite link graph as

$$A \circ B = (V, E, ctrl, link) : (X^+, X^-) \to (Y^+, Y^-)$$

where $V = V_A \uplus V_B$, $E = E_A \uplus E_B$, $ctrl = ctrl_A \uplus ctrl_B$, and the $link$ map value for each port and inner name $p \in X^+ \uplus Y^- \uplus P_A^+ \uplus P_B^+$ depends on the following conditions.

$$link(p) \stackrel{\text{def}}{=} \begin{cases} link_A(p) \text{ if } p \in Y^- \uplus P_A^+ \text{ and } link_A(p) \in E_A \uplus P_A^- \uplus Y^+ \\ link_B(p) \text{ if } p \in X^+ \uplus P_B^+ \text{ and } link_B(p) \in E_B \uplus P_B^- \uplus X^- \\ link_A(o) \text{ if } p \in X^+ \uplus P_B^+ \text{ and } link_B(p) = o \in Z^- \\ link_B(o) \text{ if } p \in Y^- \uplus P_A^+ \text{ and } link_A(p) = o \in Z^- \end{cases}$$

Directed bigraphs exclusively modify the definition of the link graph similarly to how sharing exclusively extends the place graph, so we can re-use the definition of the place graph in Chapter 2 to arrive at our full definition of a concrete directed bigraph. This also means we do not have to take into account the place graph's edges or interface in our encoding when extending our model to support directed hypergraphs.

Definition 4.3.4 (Concrete directed bigraph)**.** A concrete directed bigraph

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

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

When performing composition on two directed link graphs $A_L : X \to Y$ and $B_L : Z \to W$ to produce $A \circ B : Z \to Y$, names in W^+ can only merge with names in X^+ to preserve the direction/flow of upward flowing links in the composite bigraph. Similarly, names of W^-

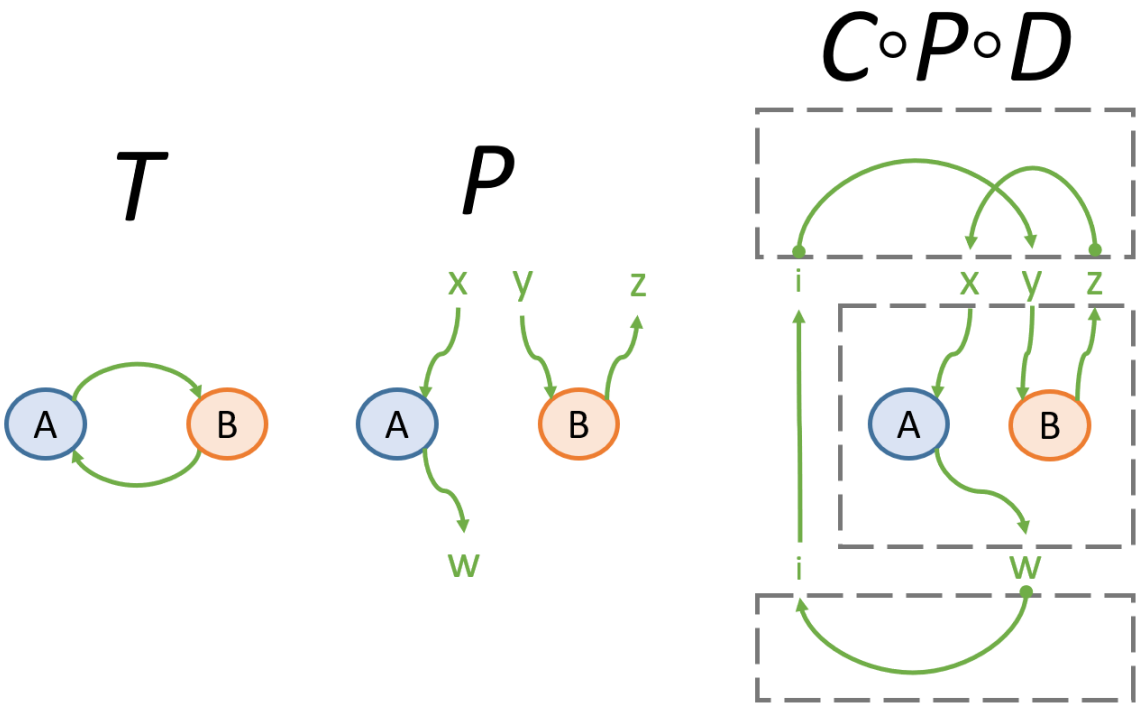


Figure 4.7: An example of composition and closure of directed links. Names on the inner face are able to be rewired onto the outer face by looping them through the identity bigraph back into the context.

can only merge with those of X^- for preserving downward flowing links. When an inner and outer name join in this context, it acts as a closure between the two faces—removing the two names and their corresponding directions in a similar fashion to the behavior of edges in the place graph when regions join with sites during composition.

As directed bigraphs are a generalization of the initial bigraph formalism, an instance of a pure bigraph can still be modelled using the directed definition by ensuring that the directions on each port all flow outward i.e. $ar^-(c) = 0$ for all $c \in ctrl$, and also that the directions of all names in both interfaces all flow upward—in the context of directed bigraphs, these are called *output-linear* link graphs. Conversely, a link graph with only negative ports and downward flowing faces is considered *input-linear*. Any directed link graph can hence be considered the composition of an input-linear and output-linear link graph. [63]

4.3.1 Directed Bigraph Matching

While there already exists a tool for solving the bigraph matching problem for directed bigraphs [74], this tackles the problem by abstracting the matching of places and links into two modular sub-problems which are then separately solved through the use of a general constraint toolkit (Section 2.5.2). In comparison, an optimized domain-specific graph solving tool like GSS has demonstrated efficient performance times for instances where constraint

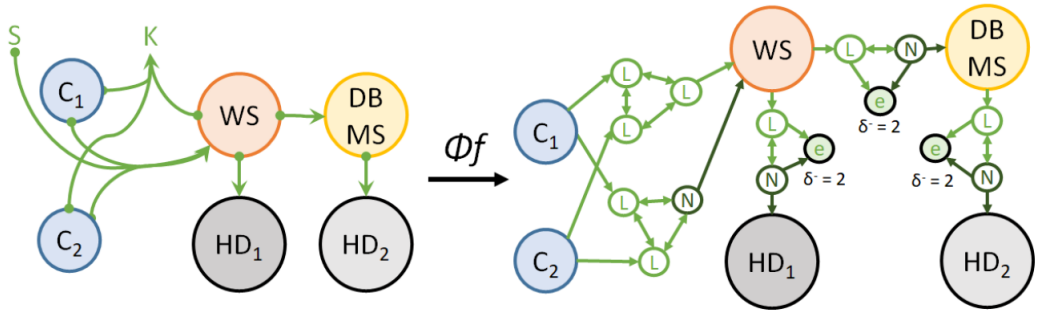


Figure 4.8: An encoding of a directed link bigraph which models a client-server architecture data flow.

toolkits suffer scaling issues, due to their underlying data structures not being built specifically to handle large graph states [38]. Because our approach instead encodes the link and place graph components into a single flattened structure, we first establish the new possible edge cases and matching rules that we now have to consider in this format due to the polarization property of links, before proposing an adapted encoding to appropriately model each of these.

Due to the U-turn constraint, we do not have to consider instances where a hyperlink flows both outward to and inward from the same face, as this can only occur when they have no adjacency to an entity, i.e. in non-solid bigraphs. To preserve symmetry, we extend our definition of a solid bigraph to disallow any occurrence of two downward outer names to be siblings in order to reflect the behavior of upward inner names. We also remain unconcerned in regards to distinguishing whether a link exists on the outer or inner face, as the allowance of U-turn links in the parameter provides a path for connecting a link through the identity and onto the inner face of the context. This allows us to treat downward inner faces as analogous to upward outer faces for matching—this is demonstrated in Figure 4.7, from w to i. Similarly, upward inner faces can become downward outer faces via the same wiring. Therefore, this reduces the scope of what we have to consider when encoding this extension down to simply whether an open link is flowing outward or inward. This leaves two new possible types of links which are introduced by direction; positive ports to negative ports, and incoming names to negative ports. Going forward, we denote links which flow into a negative port as a direct link. We consider direct links as open if an incoming inner face flows into its port, and closed otherwise. We hence adapt our encoding to support these as follows.

4.3.2 Encoding Directed Bigraphs

Open Direct Links

Firstly, we consider the flattening of negative ports which remain open in the pattern or target. Each hyperedge can only have at most one negative port as an alternative mapping for open links instead of to a closed edge or outgoing name, so we can intuitively modify our encoding to support these through the addition of a new uniquely-labelled "negative port vertex" N which joins in with the encoded clique. As we already assign an arbitrary outgoing direction property from entities to their port nodes in our flattened pure link graph representation, we are trivially able to "flip" their direction in our existing encoding to reflect that these new vertices point toward an entity rather than outward. This is sufficient to model all possible isolated instances mapping involving open direct links; many open links are able to match to the direct link through the clique as required and the injective property of direct link matching is preserved, similarly to how closed edges for pure link graphs are already encoded. A difference between closed links and open direct links however is that no degree constraint is necessary on N, as additional links and subsequently entities are permitted to join the link by adding and merging them through the context.

Closed Direct Links

Now we consider how to constrain direct links with no available incoming face for connectivity. Another parallel can be drawn here between closed direct links and closed links for pure bigraphs, as closed direct links can only match to closed direct links of the same cardinality and cannot match to open direct links, but still allow many open links to match to it. Hence we encode this similarly through adding an additional *negative closure* vertex to the clique, ensuring that this behavior is sufficiently encoded. A visual demonstration of this encoding is provided in Figure 4.8, showing the encoded form of the DBMS model example provided in [63] which includes both open and closed direct links.

Respecting the U-Turn Constraint

Our provided encoding ensures that any possible solution for directed bigraphs will be discovered in any matching instance. However, whenever a standard open link and direct link match to the same edge in T, this requires that a U-turn must be made in the context of P to recompose the target, which opens up the possibility for the U-turn constraint from Definition 4.3.2 to be violated during this process and rendering the candidate solution invalid despite matching in SIP—Figure 4.9 shows an example instance which returns a false solution without handling this extra case. Therefore we must identify and define the following

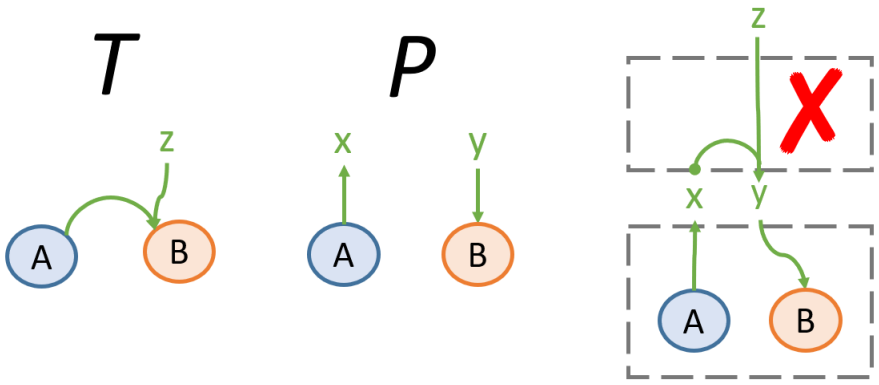


Figure 4.9: An example directed matching instance where there is no valid solution due to an open link joining to an open direct link, which produces an invalid link graph in the pattern due to the U-turn constraint.

two further constraints to ensure complete bijectivity between directed bigraph matching and our SIP encoding.

Firstly, whenever an open direct link in P incoming from face d_o is assigned to an open direct link with face d'_o in T, the required preservation of the adjacency from d_o in the context prevents any open link adjacent to h_0 in P from also mapping to d'_o without causing a violation, as both d'_o and h_o would then exist in the pre-image of d_o while the (h_o, d_o) link must make a U-turn. Thus for any candidate solution, there cannot exist any negative port nodes

$$\{\exists n \in V_P \mid \ell(n) = \textbf{neg\_link}\}$$

for the encoded graph pair (P, T) where the following two conditions are met:

- $\{\exists v \in V_P \mid \ell(v) = \textbf{link}\} \rightarrow (v, n) \notin E_P, (\texttt{match}(v), \texttt{match}(n)) \in E_T$
- $\neg\{\exists v \in V_T \mid \ell(v) = \textbf{neg\_closure}\} \rightarrow (\texttt{match}(n), v) \in E_T$

The second case occurs whenever a standard open link with face h_o and open direct link with face d_o matches to the same closed direct link d_c in T. In this case, the matching can potentially be permitted as long as the recomposition in the context produces a perfect U-turn with no other ports or faces in the context also joining d_c. An example instance is given in Figure 4.10 where this does not hold due to multiple open links attempting to join to one direct link, requiring them to both U-turn on the same name which creates an invalid directed link graph. A valid U-turn is only produced when every encoded vertex in the clique of the link in T belongs to a match, and is mapped to by only vertices representing the port adjacencies of h_o or d_o exclusively. This can also be reflected in the form of a constraint based on degree cardinality, where $\delta^-(h_0) + \delta^-(d_0) = \delta^-(d_c)$ must hold to allow the pair of pattern links to share the same target link.

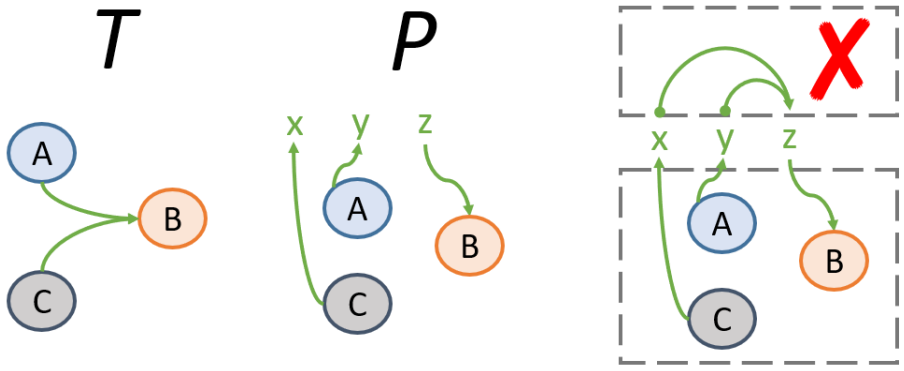


Figure 4.10: An example directed matching instance where there is no valid solution due to multiple links joining to a face which connects an incoming-outgoing link pair in the pattern, violating the U-turn constraint.

Similarly to the additional constraints for sharing, these conditions can potentially be implemented either as an additional constraint during propagation in a constraint model, or as a post-process on returned solutions.

4.3.3 Formalizing Directed Bigraphs

Now that our adapted encoding and additional constraints have been defined, we present the new flattening function as follows.

Flattening Function

Given the concrete directed link graph

$$B^{\mathsf{L}} : (V_B, E_B, ctrl_B, link_B) : (X^+, X^-) \to (Y^+, Y^-)$$

and the encoding of a (pattern or target) place graph D $\phi_{\{P,T\}}(D^{\mathsf{P}}) : (V_D, E_D)$, where (where $V_B = V_D$), we define a new flattening function as follows:

$$\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$$

where the vertices of the resultant flattened graph can be described as

$$V = V_D \uplus P_B^+ \uplus P_B^- \uplus \widehat{E_B} \uplus \widehat{N_B}$$

$$\widehat{E_B} = \{e \in E_B \mid link_B(p) = e, p \notin (X^+ \uplus Y^-)\}$$

$$\widehat{N_B} = \{n \in P_B^- \mid link_B(p) = n, p \notin (X^+ \uplus Y^-)\}$$

where $\widehat{E_B}$ is the set of closed links in B^{L}, P_B^+ and P_B^- are the positive and negative ports of B^{L}, a closure node is added for each closed edge and a negative closure node is added for each closed direct link.

We describe the resultant edge set as

$$E = E_D \uplus \{(v,p) \mid p = (v,i) \in P_B^+\} \uplus \{(p,v) \mid p = (i,v) \in P_B^-\} \uplus$$

$$\{(p_1,p_2) \mid p_1,p_2 \in (P_B^+ \uplus P_B^-), link_B(p_1) = link_B(p_2)\} \uplus$$
$$\{(p,e) \mid e \in \widehat{E_B}, link_B(p) = e\} \uplus \{(p,n) \mid n \in \widehat{N_B}, link_B(p) = n\}$$

Finally the vertex compatibility function for link graphs is adapted to support directed links as

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

subject to the following additional conditional constraint to preserve the structural integrity of links during composition, for cases involving open and closed direct links respectively:

$$\{\forall p_1 \in P_P^+, p_2 \in P_P^- \mid (\texttt{match}(p_1), \texttt{match}(p_2)) \in E_T\} \rightarrow$$
$$\begin{cases} \delta^-(n) = \delta^+(p_1) + \delta^+(p_2) \text{ if } \exists(\texttt{match}(p_2), n \in \widehat{N_T}) \in E_T \\ (p_1,p_2) \in E_P \text{ otherwise} \end{cases}$$

This encoding is sufficient for finding all matches that exist in a directed link graph whilst ensuring no false matches are found.

4.3.4 Working Example: Travelling Bus

To further demonstrate this encoding, we provide a full working example via application of the matching instance for the roaming bus model as described in [63]. In this model, the reaction rule allows a `bus` entity to move from one `zone` entity to another, as long as the origin zone has a `road` which leads to the destination. It can be observed that the origin-destination relation between zones is modelled as a closed direct link, and that the addition of the direction property allows the necessary one-way movement between roads to be modelled. The matching problem for this model aims to find all instances where a bus is

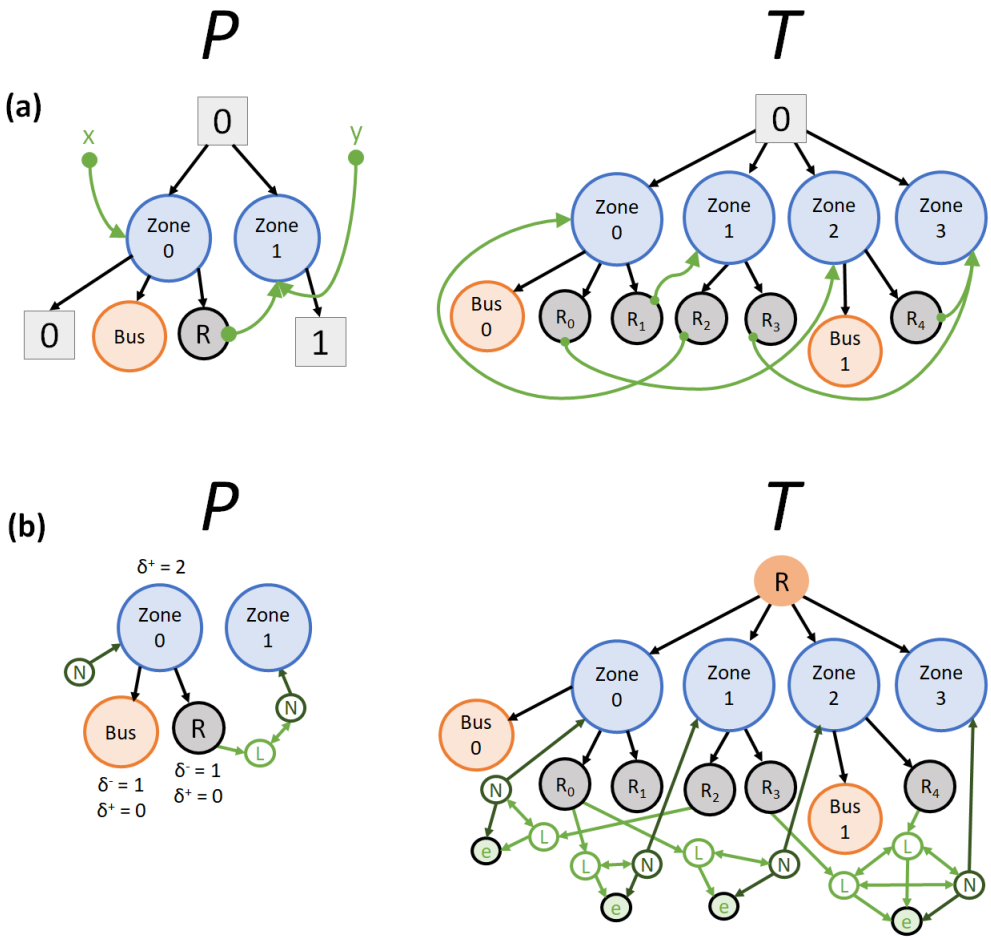


Figure 4.11: An encoding for a directed bigraph matching instance which models the travel of a bus entity between zones. In both the original problem and our SIP encoding, three possible bijective matches exist.

by a road, and the road connects to a potential destination for the bus.

An example instance is provided in Figure 4.11 (a). It can be observed that for the given target bigraph, three possibilities for movement exist which correspond to three matches; Bus 0 can travel from Zone 0 to either Zone 2 via Road 0 or Zone 1 via Road 1, and Bus 1 can travel from Zone 2 to Zone 3 via Road 4. The same matches can also be observed in the encoded SIP instance as shown in Figure 4.11 (b) through representing negative ports as parent vertices of their respective entity.

4.3.5 Directed Encoding Size

The introduction of open and closed negative links modifies the size of a flattened link graph as follows, ignoring any encoded place graph components, where $|e| = |link_G^{-1}(e) \cap P_G^+|$ for a standard hyperedge $e \in E_G$, and $|n| = |link_G^{-1}(n) \cap P_G^+|$ for a negative port $n \in P_G^-$.

Directed Link Graph $G_L : (X^+, X^-) \to (Y^-, Y^+)$

$$|V_L| = |V_G| + |P_G^+| + |P_G^-| + |\widehat{E_G}| + |\widehat{N_G}|$$
$$|E_L| = |P_T^+| + |P_T^-| + \sum_{e \in E_G} |e| \cdot (|e| - 1) + \sum_{e \in \widehat{E_G}} |e| + \sum_{n \in N_G} |n| \cdot (|n| - 1) + \sum_{n \in \widehat{N_G}} |n|$$

The scalability of our encoding hence remains unchanged by this adaptation, as open and closed negative links will each introduce the same number of vertices and edges to the graph in their clique representation compared to a standard (positive) open/closed link.

4.3.6 Soundness and Completeness

We now demonstrate a formal proof of soundness and completeness for our adaptated flattening function. As this modifies the encoding of link graphs only, we can infer that Propositions 1 to 4 (Section 3.5.1) will still remain true. Proposition 5 holds by the definition of directed link graph composition (Definition 4.3.3) it remains that $link_B(p_1) = link_B(p_2) \to link_G(p_1) = link_G(p_2)$ when $G = A \circ B$. Proposition 6 also holds for any $p \in P_B^+$ as the properties of labelled edges in the pattern are unaffected by direction on solid bigraphs (can only be linked to by positive ports).

In addition, we propose some new observations specifically for proving correctness in the context of directed bigraphs as follows.

Proposition 10. Given the composition of two directed link graphs

$$G : (X^+, X^-) \to (Y^+, Y^-) = (A : (Z^+, Z^-) \to (Y^+, Y^-)) \circ (B : (X^+, X^-) \to (Z^+, Z^-))$$

for any $p \in P_B^+$ and $n \in P_B^-$ where $\neg\exists x \in (Z^+ \uplus X^+)$ such that $link(x) = n$, $link_G(p) = n$ if and only if $link_B(p) = n$.

This states that a positive port in B links to a negative port if and only if they are also linked in G when the negative port is unconnected to the interface. Firstly, from Definition 4.3.3, $link_B(p) = n \in P_B^- \implies link_G(p) = link_B(p)$, therefore $link_G(p) = n$. Secondly, we prove that $link_B(p) = n \impliedby link_G(p) = n$ by hypothesizing a $p \in P_B^+, n \in P_B^-$ such that $link_G(p) = n$ but $link_B(p) \neq n$. This would instead mean $link_B(p) = x \in (Z^- \uplus Y^- \uplus P_B^- \uplus E_B)/\{n\}$. Assume the case that $x \in \{Y^- \uplus P_B^- \uplus E_B\}/\{n\}$: by Definition 4.3.3, this implies $link_G(p) = x \neq n$ which violates our initial conditions, leaving only the possibility that $x \in Z^-$ and therefore $link_G(p) = link_A(x) \in A$. By our initial assumption, $link_A(x) = n$ which is only possible if there is some mediating link $y \in Y$ such that $link_A(x) = y$ and $link_B(y) = n$, but this contradicts our initial condition that no such link pointing to n from the interface exists. Therefore we prove our proposition by contradiction. As a corollary, we deduce that $\{link_B^{-1}(n) \cap (Z^+ \uplus X^+) = \emptyset\} \rightarrow |link_G^{-1}(n) \cap P_G^+| = |link_B^{-1}(n) \cap P_B^+|$.

Proposition 11. Given the composition of two directed link graphs

$$G : (X^+, X^-) \rightarrow (Y^+, Y^-) = (A : (Z^+, Z^-) \rightarrow (Y^+, Y^-)) \circ (B : (X^+, X^-) \rightarrow (Z^+, Z^-))$$

for any $n \in P_B^-$ where $link_B(z \in Z^+) = n$ and $link_A(x \in X^+) = z$ then $link_G^{-1}(n) = link_B^{-1}(n) \uplus link_A^{-1}(z)$.

This states that when an open direct link in B composes with another open direct link in A then its resultant set of adjacencies will include those belonging to both links exclusively. By Definition 4.3.3, if $link_B(p \in (P^+ \uplus Z^+)) = n$ then $link_G(p) = link_A(a) = n$ and if $link_A(p \in (P^+ \uplus X^+)) = z$ then $link_G(p) = link_B(z) = n$, therefore $link_B^{-1}(n) \uplus link_A^{-1}(z) \subseteq link_G^{-1}(n)$ at least. Now we hypothesise a source $p \notin link_B^{-1}(n) \uplus link_A^{-1}(z)$ where $link_G(p) = n$. If $p \in A$ then the only way $link_G(p) = n$ is if $link_A(p) = z$, meaning $p \in link_A^{-1}(z)$ which is a contradiction. If $p \in B$ then this can be achieved if $link_B(p) = z' \in Z^-$ and there exists a U-turn $link_A(z') = z$. However this would mean $\{z', x\} \subseteq link_B^{-1}(z)$ and cannot be a valid link graph by the U-turn constraint (Definition 4.3.2). Therefore all possibilities lead to a contradiction which proves our proposition. As a corollary, we deduce from this that if $link_B(p) \notin n$ then $link_G(p) \notin n$ when $link_G(y \in Y^+) = n$.

Proposition 12. Given the composition of two directed link graphs

$$G : (X^+, X^-) \rightarrow (Y^+, Y^-) = (A : (Z^+, Z^-) \rightarrow (Y^+, Y^-)) \circ (B : (X^+, X^-) \rightarrow (Z^+, Z^-))$$

for any $p, \in P_B^+, n \in P_B^-$ where $link_B^{-1}(n) \cap Z^+ = z', link_B(p) = z \in Z^-$ and $link_G(p) = n$ then $link_G^{-1}(n) = link_B^{-1}(n) \uplus link_B^{-1}(z)$

This states that if a open direct link and open standard link in B are ever wired together during composition, then the resultant direct link must include all, and only, ports belonging to the two links. We know from Proposition 5 that if any two ports share a link in B then they share a link in G, therefore $link_B^{-1}(n) \subseteq link_G^{-1}(n)$. Since $link_B(p) \neq n$ but $link_G(P) = n$, then there must exist a U-turn such that $link_A(z) = link_B(n)$, thus $link_A(z) = z'$, transitively connecting z to n. From Proposition 5, we thus deduce $link_B^{-1}(z) \subseteq link_G^{-1}(n)$. Now we hypothesize a source p' such that $link_G(p') = n$ but $p' \notin link_B^{-1}(n) \uplus link_B^{-1}(z)$. We assume $p' \in A$ as if $p' \in B$ then it must link to a name in Z that loops back from A into n regardless. $link_G(p') = n$ can then hold if $link_A(p') = z$, but then $link_A^{-1}(z') = \{p', z\}$ where a U-turn exists and therefore cannot be a valid directed link graph (Definition 4.3.2) and this contradicts our hypothesis, concluding our proof. As a corollary, we deduce $|link_G^{-1}(n)| = |link_B^{-1}(n)| + |link_B^{-1}(z)|$ when $link_B(p) = z$, $link_G(p) = n$.

Soundness

We prove soundness by construction in a similar manner to our method for proving soundness for pure bigraphs. We initially assume that the place graphs for $T = C \circ (P \otimes id) \circ D$ have already been rebuilt and each target entity has been assigned to either C, P or D by following the steps outlined in Section 3.5.1.

A directed link graph $G = (V_G, E_G, ctrl_G, link_G) : (X^+, X^-), (Y^+, Y^-) \in \{P, T\}$ can then be reconstructed from $\phi_f(G)$ as follows.

$$V_G = \{g' \in \phi_f(G) \mid \ell(g') \notin \{\mathbf{link}, \mathbf{closure}, \mathbf{neg\_link}, \mathbf{neg\_closure}\}\}$$
$$P_G^+ = \{g' \in \phi_f(G) \mid \ell(g') = \mathbf{link}\}$$
$$P_G^- = \{g' \in \phi_f(G) \mid \ell(g') = \mathbf{neg\_link}\}$$
$$E_G = \{g' \in \phi_f(G) \mid \ell(g') = \mathbf{closure}\}$$
$$\{\forall g \in V_G \mid ctrl_G(g) = \ell(g')\}$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G)} \mid \ell(g'_a) = \ell(g'_b) = \mathbf{link}\} \rightarrow link_G(g_a \in P_G^+) = link_G(g_b \in P_G^+)$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G)} \mid \ell(g'_a) = \mathbf{link}, \ell(g'_b) = \mathbf{closure}\} \rightarrow link_G(g_a \in P_G^+) = g_b \in E_G$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G)} \mid \ell(g'_a) = \mathbf{link}, \ell(g'_b) = \mathbf{neg\_link}\} \rightarrow link_G(g_a \in P_G^+) = g_b \in P_G^-$$

The interface of Y^- for P and T can then be rebuilt using the same method as pure bigraphs on positive ports (Section 3.5.1). Y^+ is constructed by adding a new outer name y as a source for each $g \in P_G^-$ where there is no $\{v' \in V_{\phi_f(G)} \mid \ell(v') = \mathbf{neg\_closure}\} \rightarrow (g', v') \in E_{\phi_f(G)}$ and adding $link_P(y) = g$.

We now rewire the links of C and D in adherence to the compositional property of directed link graphs (Definition 4.3.3) for the nine combinations of directed link pair positions as

follows:

- $\{\forall p \in P_P^+ \mid link_P(p) = y \in Y^-,\ link_T(p) \in C\} \rightarrow link_C(y \in Y^+) = link_T(p)$
- $\{\forall p \in P_P^+ \mid link_P(p) = y \in Y^-,\ link_T(p) \in P\} \rightarrow link_C(y) = y' \in Y^-$ where $link_P(y' \in Y^+) = link_T(p)$
- $\{\forall p \in P_P^+ \mid link_P(p) = y \in Y^-,\ link_T(p) \in D\} \rightarrow link_C(y) = y' \in Y^-, link_{id}(y' \in Y^+) = x \in X^-, link_D(x \in X^+) = link_T(p)$
- $\{\forall p \in P_C^+ \mid link_T(p) \in C\} \rightarrow link_C(p) = link_T(p)$
- $\{\forall p \in P_C^+ \mid link_T(p) \in P\} \rightarrow link_C(p) = y \in Y^-$ where $link_P(y \in Y^+) = link_T(p)$
- $\{\forall p \in P_C^+ \mid link_T(p) \in D\} \rightarrow link_C(p) = y \in Y^-, link_{id}(y \in Y^+) = x \in X^-, link_D(x \in X^+) = link_T(p)$
- $\{\forall p \in P_D^+ \mid link_T(p) \in C\} \rightarrow link_D(p) = x \in X^-, link_{id}(x) = y \in Y^-, link_C(y \in Y^+) = link_T(p)$
- $\{\forall p \in P_D^+ \mid link_T(p) \in P\} \rightarrow link_D(p) = x \in X^-, link_{id}(x) = y \in Y^-, link_C(y \in Y^+) = y' \in Y^-$ where $link_P(y' \in Y^+) = link_T(p)$
- $\{\forall p \in P_D^+ \mid link_T(p) \in D\} \rightarrow link_D(p) = link_T(p)$

This thus builds the $T = C \circ (P \otimes id) \circ D$ decomposition on the ports and edges of the directed link graph. This concludes our proof by construction.

Completeness

We again prove completeness with a proof by contradiction, observing that if S_{big} is a valid embedding while S_{SIP} is not a valid solution then at least one constraint in the adapted SIP model for directed link graphs must be violated. Propositions 1 to 6 remain true, so all constraints from the original SIP model must hold and we consider the newly defined constraints on links.

Proposition 10 proves that closed direct hyperedges in the pattern must always match on a closed target hyperedge of the same cardinality, and thus the possible degree constraint on newly introduced negative closure vertices will always be true for a valid composition. Proposition 11 shows that an open standard hyperedge and open direct hyperedge cannot map to the same open target link due to the additional U-turn constraint on faces (Definition 4.3.2), therefore the new constraint to handle this will always return true in this case. Proposition 12 proves the case where an open standard hyperedge and open target hyperedge can only match

to a closed target link together if the target's cardinality is the sum of the two pattern links (producing a valid U-turn), satisfying the other case of the new constraint. This demonstrates that no constraint in the adapted model and flattening can be violated, and thus if there exists a solution S_{big} then the encoding will find the responding S_{SIP} solution in a match by contradiction. This concludes the proof.

4.4 Summary

The ease of implementing support for sharing in the place graph and direction in the link graph, as well as adapting our encoding for finding support equivalences, 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. As none of the adapted encodings for these extensions conflict with one another, it is easily possible to support any combination of equality checking, shared places and directed links in a single matching instance.

Support for directed bigraphs was not implemented for this project as BigraphER itself does not yet support the extension, however this could be achieved through the following steps:

- Adapt the core bigraph data structure in BigraphER such that entities can also store a negative arity value, and subsequently possess negative ports.
- Revise all existing bigraph manipulation, composition and rewriting functions to account for the extended format, ensuring that they model the expected behavior for directed links as described in Definition 4.3.3.
- Modify the syntax and language of BigraphER to allow the user to optionally define agents and reaction rules which contain directed links when creating a BRS.
- Define a new textual representation of bigraphs (Figure 5.1) which allow for direction in links, replacing the current format, and modify BigraphER to both read in and write out bigraphs in this form.
- Thoroughly evaluate the updated version of BigraphER for correctness (e.g. test cases).
- Modify BigraphER's graph visualization tools to include direction in their graphical output.
- Adapt our current GSS encoding to support directed links as described in Section 4.3.

As this concerns a large amount of engineering and refactoring work which strays from the focus of improving the matching algorithm, we leave the adaptation of BigraphER and our adapted GSS solver to support direction as a potential avenue for future work, which would then allow us to compare the runtime metrics obtained against those achieved by jLibBig [74].

We now describe in the following chapter our method of implementing our constraint models for matching and equivalence on bigraphs and bigraphs with sharing within GSS, and evaluating its performance as the solver component of BigraphER.

Chapter 5

Implementation and Evaluation of Matching

In this chapter, we describe our working adaptation of the Glasgow Subgraph Solver to support our algorithm for bigraph matching, and provide evaluation metrics on both solving isolated instances of matching as well as computing full transition systems for a variety of BRS models when integrated as the new solver for BigraphER. We find that GSS comfortably outperforms the two other available solvers on all individual matching problems while achieving an aggregate speedup of over two orders of magnitude when compared to BigraphER's MSAT and MCARD solvers, scaling increasingly well on instances containing many solutions. Upon computing full transition systems, our integrated GSS implementation achieves the best performance on 10 out of 13 evaluated BRSs.

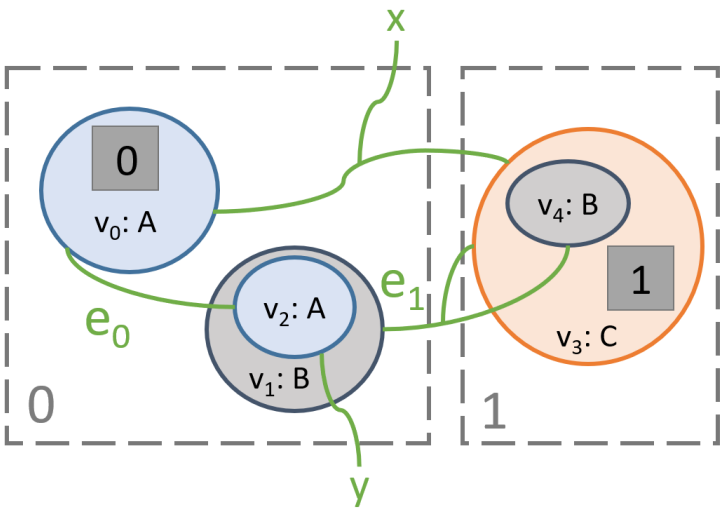
Section 5.1 describes the adaptation of GSS to allow for the preliminary encoding and flattening of an input bigraph and application of the constraints identified in Chapter 3, as well as the constraints to support sharing in Section 4.2. Section 5.2 showcases the performance run times of our initial version of our adaptation compared against those returned by BigraphER's MSAT and MCARD solvers, with additional comparisons for instances of equality and evaluating the optimal configuration of GSS. Section 5.3 compares our GSS implementation against BigraphER's current solvers when integrated as the matching tool for building transition systems for both simpler and more hard to solve BRS models. Section 5.4 summarizes the chapter.

5.1 Implementation

We implemented the encoding and SIP solving process for bigraphs and bigraphs with sharing within the Glasgow Subgraph Solver 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 extra vertex compatibilities and post-processing on solutions. We denote this adaptation as the *Glasgow Bigraph Solver* (GBS). The tool supporting our approach has been made publicly available [75].

5.1.1 Glasgow Bigraph Solver




```
{(0, A:2), (1, B:1), (2, A:1), (3, C:2), (4, B:1)}
2 5 2
1100000
0001000
0000010
0010000
0000000
0000101
0000000
({}, {x}, {(0, 1), (3, 1)})
({}, {y}, {(2, 1)})
({}, {}, {(0, 1), (2, 1)})
({}, {}, {(1, 1), (3, 1), (4, 1)})
```

Bigraph.big

Figure 5.1: An example bigraph and its corresponding text representation in the BigraphER `.big` format.

GSS provides a command line interface for solving single instances of SIP through reading and parsing an input pattern and target graph file for a variety of standard graph formats. Examples of graph formats that GSS can support include DIMACs, LAD and CSV representations of graphs, which are supplied to the input parsing component of the solver in the form of text files. We adapt GSS to support bigraphs as input on the command line through the implementation of an additional parsing option, where a text representation of a bigraph can

be taken as an input and built directly as an encoded graph according to our algorithm, in the bigraph file format used by BigraphER as provided in Figure 5.1. Therefore, the conversion from a bigraph to an encoded graph is being performed directly by the existing input parser while reading in the bigraph text file—as though it were simply a new graph format—rather than requiring a separate process to convert a bigraph data structure into an encoded graph that the solver can understand. The first line defines the control and arity value of each entity in the structure, and the second line specifies that the bigraph contains n regions, $|V_P|$ entities and m sites respectively. The following set of lines then provide a $(n + |V_P|) \times (|V_P| + m)$ adjacency matrix for each pair of components in the place graph. Finally, each remaining line defines a shared link in the link graph via its connected face name (field left empty for closed links) and a set of pairs which each denote its adjacency to an entity and the number of ports on the entity it occupies. Regions, sites and closed links are internally labelled in order of their definition in this format, incrementing from zero, and all flattened link nodes are added after the encoded place graph vertices.

Two separate processes are defined for the reading and encoding of the pattern and target: while the encoded target graph is treated as a standard SIP graph beyond this process, the pattern encoding function in addition to building the graph produces two additional vectors called **pattern_site_edges** and **pattern_root_edges** which retain the information regarding edges to/from the interface which is otherwise lost in the encoding process, and are retained during and after search for the application of preliminary and checking constraints. On top of these, we also define a list of pair boolean values of length $|V_P|$ called **pattern_big_constraints**, where for each $b_i = (b_{i1}, b_{i2})$ in this list, $b_{i1} = true \iff (a, v_i) \in$ **pattern_root_edges** for some region a and $b_{i2} = true \iff (v_i, a) \in$ **pattern_site_edges** for some site a. This allows us to more easily define the required degree constraints later on.

Preliminary Constraints

Before search, GSS runs a set of functions to compare the labels and degrees of all pattern-target pairs to preliminarily eliminate any immediately detectable incompatibilities from the domain. We add our own additional function to this set in order to perform this check for our place and link compatibility functions in $O(|V_P||V_T|)$ time as shown in Algorithm 1, which takes an input integer pair (i, j) and returns **true** if encoded pattern vertex p_i can potentially map to pattern vertex t_j and **false** otherwise. We denote $|link_G|$ as the number of link nodes in an encoded graph G, and $l(e)$ as the label of a vertex e.

For equality checking, the number of entities and link edges are also trivially counted during input and will immediately return false instead of calling the solver if there is a mismatch detected.

Algorithm 1 Check Bigraph Degree Compatibility (int i, int j)

Ensure: $0 \leq i \leq |V_P|, 0 \leq j \leq |V_T|$
{Link graph compatibility}
if $(i \geq (|V_P| - |link_P|))$ **and** $(j < (|V_T| - |link_T|))$ **then**
return false;
else if $(i < (|V_P| - |link_P|))$ **and** $(j \geq (|V_T| - |link_T|))$ **then**
return false;
else if $l(p_i) \rightarrow$ CLOSURE **and** $l(t_j) \rightarrow$ CLOSURE **and** $deg^-(p_i) \neq deg^-(t_j)$ **then**
return false;
else if $(i \geq (|V_P| - |link_P|))$ **and** $(j \geq (|V_T| - |link_T|))$ **then**
return true;
end if
{Place graph compatibility}
if pattern_big_constraints$(i)_1$ **and** $deg^-(p_i) > deg^-(t_j)$ **then**
return false;
else if not pattern_big_constraints$(i)_1$ **and** $deg^-(p_i) \neq deg^-(t_j)$ **then**
return false;
else if pattern_big_constraints$(i)_2$ **and** $deg^+(p_i) > deg^+(t_j)$ **then**
return false;
else if not pattern_big_constraints$(i)_2$ **and** $deg^+(p_i) \neq deg^+(t_j)$ **then**
return false;
else
return true;
end if

Checking Constraints

We implement the remainder of constraints (shared regions, shared sites, transitive closure) as checking constraints. The sharing constraints will always hold for standard bigraphs, however for optimization purposes we only enforce these if it is detected during parsing that they are required—that is, transitive closure checking is only performed when sharing exists i.e. a site or entity is given multiple parents, and only pattern entities adjacent to exclusively shared regions or sites are checked for a valid abstraction of parents and sites in the target.

In the event sharing exists in an instance, the additional $(|V_T| - |link_T|) \times (|V_T| - |link_T|)$ bitset **reachability_matrix**(t_1, t_2) structure is constructed during the target graph encoding which returns **true** iff. t_2 is a transitive descendant of t_1. We then check for transitive closure using this via Algorithm 2, for all matches of $p_i, p_j \in P$ where p_1 and p_2 are adjacent to a site and region respectively to ensure no cycles are produced. If **false** is returned, then the candidate solution is discarded. It can be deduced that this check is performed on a solution in the order of $O(|V_P|^2|V_T|^2)$ in the worst case where all pattern vertices connect to a region and site, although this would be unlikely to occur in practice for larger numbers of entities.

The final necessary constraints required are those which deal with shared regions and sites for

Algorithm 2 Transitive Closure Check (int i, int j)

Ensure: $0 \leq i, j \leq (|V_P| - |link_P|), i \neq j$
Ensure: **pattern_site_edges**(i) **and pattern_root_edges**(j)
  for all $\{k \mid (\texttt{match}(p_i),\ t_k) \in E_T$ **and** $\texttt{match}^{-1}(t_k) = \emptyset\}$ **do**
    for all $\{l \mid (t_k,\ \texttt{match}(p_j)) \in E_T$ **and** $\texttt{match}^{-1}(t_l) = \emptyset\}$ **do**
      if reachability_matrix(t_k, t_l) **then**
        return false;
      end if
    end for
  end for
  return true;

Algorithm 3 Shared Region Check (int r)

Ensure: $(\sum_{n=0}^{|V_P|-|link_P|}(r, n) \in$ **pattern_root_edges**$) \geq 2$
  encapsulated_nodes $= \{\}$;
  for all $\{i \mid (r, \texttt{match}^{-1}(i)) \in$ **pattern_root_edges**$)\}$ **do**
    for all $\{j \mid (t_j, t_i) \in E_T$ **and** $\texttt{match}^{-1}(t_j) = \emptyset$ **and** $j \notin$ encapsulated_nodes$\}$ **do**
      encapsulated_nodes $\leftarrow j$;
    end for
  end for
  for all $\{i \mid (r, \texttt{match}^{-1}(i)) \in$ **pattern_root_edges**$)\}$ **do**
    for all $\{j \mid j \in$ encapsulated_nodes$\}$ **do**
      if $(t_j, t_i) \notin E_T$ **then**
        return false;
      end if
    end for
  end for
  return true;

bigraphs with sharing. While we implement these as checking constraints which is sufficient to build a fully working model, as a future optimization these can potentially be enforced at propagation time via an additional GSS supplemental graph encoding. This is performed using Algorithm 3 and Algorithm 4 for each shared region and shared site in the pattern respectively. These are each applied to a solution in $O(|V_P||V_T|)$ time in the worst case, where all vertices in the pattern are adjacent to a shared region or shared site respectively: this upper bound is due to the solid rule for sharing, where entities can only have up to one region parent or one child site.

Eliminating Symmetries

Both methods of removing symmetries as defined in Section 3.3.5 were able to be implemented using existing functionality supplied by GSS. Support for adding nogood constraints is required for the implementation of the solution biased search heuristic, thus we

Algorithm 4 Shared Site Check (int s)

Ensure: $(\sum_{n=0}^{|V_P|-|link_P|}(n, s) \in$ **pattern_site_edges**$) \geq 2$
  encapsulated_nodes $= \{\}$;
  for all $\{i \mid (\texttt{match}^{-1}(i), s) \in$ **pattern_site_edges**$)\}$ **do**
    for all $\{j \mid (t_i, t_j) \in E_T$ **and** $\texttt{match}^{-1}(t_j) = \emptyset$ **and** $j \notin$ encapsulated_nodes$\}$ **do**
      encapsulated_nodes $\leftarrow j$;
    end for
  end for
  for all $\{i \mid (\texttt{match}^{-1}(i), s) \in$ **pattern_site_edges**$)\}$ **do**
    for all $\{j \mid j \in$ encapsulated_nodes$\}$ **do**
      if $(t_i, t_j) \notin E_T$ **then**
        return false;
      end if
    end for
  end for
  return true;

can simply apply a new nogood constraint containing the set of variable-value assignments of each support component each time a new solution is found. GSS also supplies a **apply_pattern_less_than** constraint for symmetry breaking in conventional SIP, thus we are able to apply this to all pairs of support vertices sharing a clique (or both have no clique) as required before search. Which of the two strategies used to break clique symmetries can be specified in a command line argument when calling the matcher.

5.2 Evaluation: Single Instances

We now demonstrate the performance runtimes obtains by GBS on individual matching problems compared to those of BigraphER's MiniSAT (MSAT) solver, for both bigraphs and bigraphs with sharing. As an additional comparison, we also consider the results of BigraphER's alternative psuedo-boolean MiniCARD (MCARD) solver [61] which is able to encode additional cardinality constraints.

All instances in this section that we evaluate are made available in BigraphER's text format as well as the results we obtain [76].

5.2.1 Initial Performance vs MSAT/MCARD

A preliminary evaluation and discussion of the results found in this subsection is featured in our CP2021 publication [15]. The experiments were run on systems with dual Xeon E5-2687A v4 CPUs and 512GBytes RAM, running Ubuntu 18.04. The SIP solver is compiled

with GCC 7.5 while BigraphER is compiled with OCaml 4.10, statically linked to MiniSAT [60] compiled with GCC 9.3, and then copied to the benchmarking machine. For each of the three solvers, the time we specifically seek to measure is the time elapsed between calling the main matching function and retrieving the returned solution set, which includes the main search and backtracking process as well as time spent preliminarily generating clauses, constraints and any supplementary data structures prior to running the search loop, and any necessary post-process checking on solutions. We do not wish to measure any time spent performing additional handling of input/output for the time being, as in a practical context this would be handled through an interface within the integration of GBS as a component in BigraphER, rather than running the tool directly through the command line for experimental purposes. To allow experimenting with a large number of instances, we perform up to 30 matching problems in parallel on the same machine.

The set of instances used for this evaluation have been derived from three BRSs with sharing, two of which having been previously used for modelling real-world systems. These are described as follows:

- **Savannah:** A Mixed-Reality [41] game developed for children where each player takes the role of a lion hunting prey, and where their real-world locations are mapped to a virtual savannah environment using their GPS location from their mobile devices to determine their relative position in the game world. As players explore the area, they may encounter prey (and are notified of such on their devices) where they can choose whether to attack or not. An attack is successful if the correct number of close-by players required for defeating that type of prey co-operate to attack simultaneously. The aim of the game is thus to figure out this correct value for each type of prey in the environment. The savannah BRS with sharing models interactions between players, their environment and their devices as well as the virtual lions and prey [41]. There are 2528 savannah instances in our matching dataset.

- **802.11:** A stochastic [6] BRS with sharing which models the 802.11 MAC protocol. The BRS models WLANs and their connections, and the reaction rules allow for transmissions of packets between connected devices [5]. There are 288 of these instances in our matching dataset.

- **Conference:** An adaptation of the conference BRS as defined by Milner [1], where agents can move around between rooms in a building and connect to, or disconnect from a conference call if there is a computer in the room. The matching instances of the Savannah and 802.11 models are relatively small due to the limitations of previous solvers (no instance takes more than 3ms to solve on MSAT), so to test scalability we make use of harder generated instances of this BRS obtained through modifying the

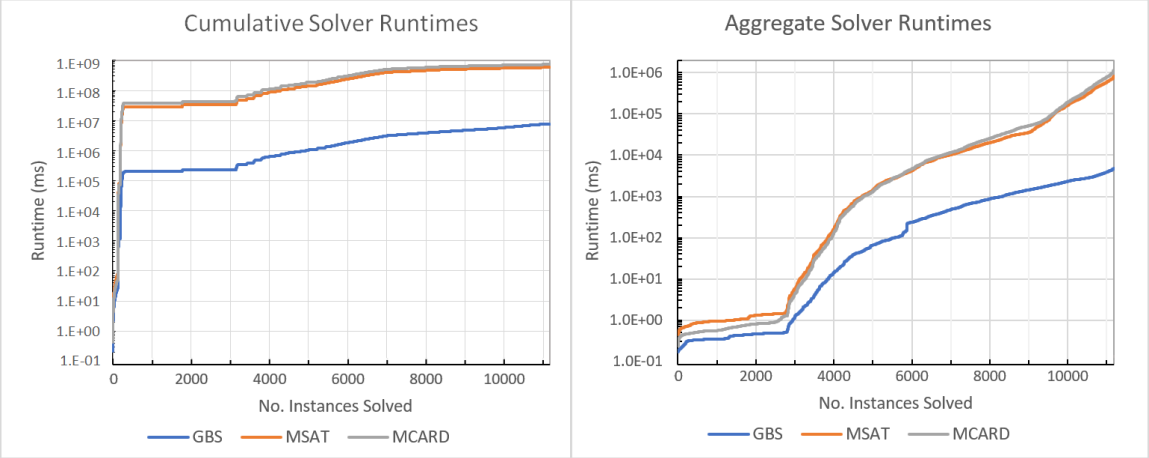


Figure 5.2: The solver runtimes achieved by GBS, MSAT and MCARD on the matching dataset. Cumulative runtime (left) measures the total time taken to solve n instances. Aggregate runtime (right) measures that the corresponding solver was able to solve n instances within a runtime $\leq t$.

reaction rules to add extra entities at random. There are 8360 conference instances in our matching dataset, which we partition into 4128 instances with sharing and 4232 instances without sharing for evaluating any differences in runtime performance.

In total, there are 11,176 matching instances in this dataset. To ensure confidence in our working implementation, we verify that the set of solutions obtained for GBS, MSAT and MCARD are always the same for every instance.

Cumulative and Aggregate Performance

We first evaluate a high-level view of the performances of each solver. For this initial run, GBS uses the nogoods implementation for clique symmetry breaking, and each instance is run once against each of the three evaluated solvers. We seek to determine the relative performance of GBS against MSAT and MCARD via two different measurements: the conventional *cumulative* runtime achieved upon solving the full dataset of instances, as well as the *aggregate* runtime [77] achieved by each solver. The time ratio between the runtimes of GBS and MSAT/MCARD can then determine the cumulative and aggregate *speedup* respectively, achieved by GBS compared to the currently used solvers.

The cumulative runtime measures the *total* amount of time taken for each solver to solve all instances, where instances are ordered by the size of the solution set from smallest to largest. Figure 5.2 (left) shows the total time elapsed after instance n is solved for each solver, when ordered from 1 to n in sequence. The vertical distance between lines indicates the cumulative speedup between solvers for solving each instance. It can be observed that MSAT and MCARD as a whole achieve similar overall timings, while GBS demonstrates

a significantly improved performance upon both other solvers—GBS achieves an overall cumulative speedup of 76 over MSAT and a speedup of 95 over MCARD.

The aggregate runtime meanwhile measures the total time taken to solve all instances if each were run *simultaneously in parallel* to one another, as opposed to in sequence when we measure cumulative runtime. The overall aggregate runtime for a solver can thus be determined as the time taken to solve the single most difficult instance in the dataset for that solver (the most difficult instance may not necessarily be the same one across all solvers). We also measure performance by this metric for three main reasons: firstly, when it comes to comparing easier instances in the dataset i.e. the Savannah and 802.11 protocol problems, the cumulative speedup observed will appear less significant due to the fact that the existing solvers can already solve them near-instantly ($\approx 1ms$) which physically limits how much their runtimes can be improved upon using a more efficient solve method—and thus we may wish to measure relative performance in a way that reduces the significance of these instances. Secondly, when large speedups are observed on a logarithmic scale, it becomes more difficult to display differences in speedup value that appear small but are significant in practice, i.e. a speedup ratio of 10 vs 30 on a chart [77]. Thirdly, it demonstrates the hypothetical maximum potential for performance gains in a hypothetical parallelized system where instances are split up between worker threads and solved simultaneously. Figure 5.2 (right) shows the aggregate runtimes for each solver as the number of *individual* instances each solver was able to solve within a time limit of t milliseconds — the horizontal distance between lines hence indicates the difference in the number of instances that were able to be solved in t ms. The vertical distance between a pair of lines indicates the increase in timeout required to allow the two solvers to solve any set of n instances, and hence measures their *aggregate* speedup up to that point.

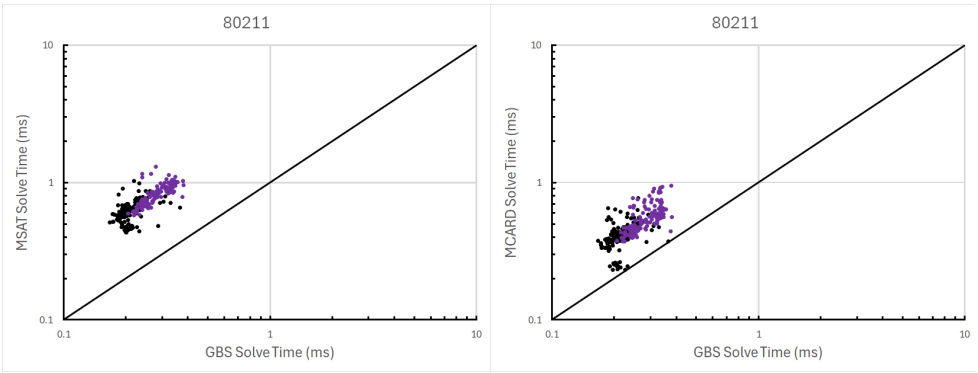
An initial observation is that all solvers hold a flat consistent aggregate runtime for the easiest ≈ 3000 instances before beginning to scale upward for the harder remaining instances. This can be explained by the real-world instances being significantly more trivial to solve (2528 Savannah instances plus 288 802.11 MAC instances) before the transition to the larger and more difficult 8360 generated Conference instances (also later shown in Figure 5.3) — hence demonstrating the initial need to create harder problems in order to more comprehensively evaluate the scalability of each solver. It can be observed that MCARD tends to solve more problems very quickly ($< 1ms$ solve time) compared to MSAT, but performs slightly worse than MSAT in aggregate where it also solves more problems more slowly ($> 1s$ solve time). While this is an unexpected result as MCARD is able to natively encode more high-level constraints, we hypothesize that while these additional constraints trigger on a subset of problems which result in very fast solve times, this does not occur on the majority of instances and we hence observe slight overhead from handling the constraints compared to MSAT overall. The key observation to be made is that GBS again significantly outper-

forms both other solvers on both easier and harder problems, scaling particularly well on hard instances; its most difficult instance took 4,516ms to solve, while MSAT and MCARD required 820,117ms and 1,135,314ms respectively to solve their hardest instances. Therefore GBS achieves an aggregate speedup of 181 over MSAT and 251 over MCARD.

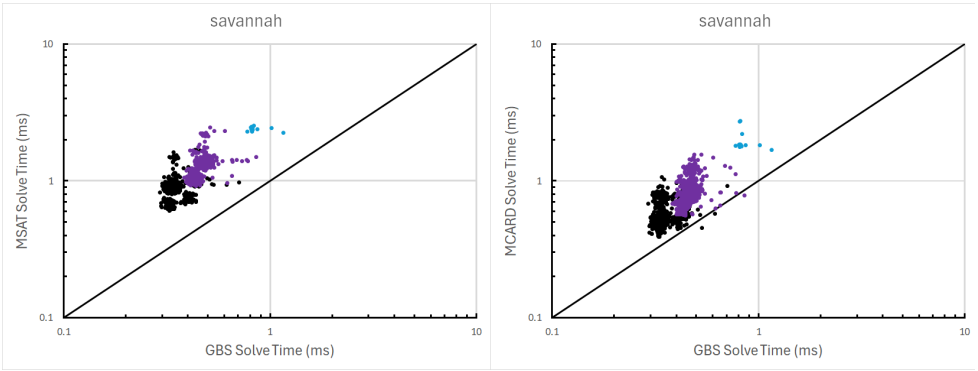
Repeating this experiment for GBS demonstrated little variance (roughly $\pm 3\%$ in both cumulative and aggregate runtimes from ten runs)—this is because when enumerating all solutions, the solver is required to fully exhaust the search tree which ends up being a fairly deterministic process, unlike a decision problem where simply finding one single solution is sufficient (where the solver might get "lucky" and find a valid combination of assignments earlier than expected by happenstance, due to the slightly-random search heuristic). This in combination with the dataset containing a large amount of instances causes variation to be further smoothed out in aggregate, thus a single run is sufficient to deduce the performance of the algorithm. While not a primary focus of the evaluation, it was also observed that memory usage was negligible, the maximum resident set size recorded for the largest instances never surpassing a maximum of 64MB — suggesting that running multiple GBS instances in parallel could be a promising feasible approach to future performance gains.

Performance Per Instance

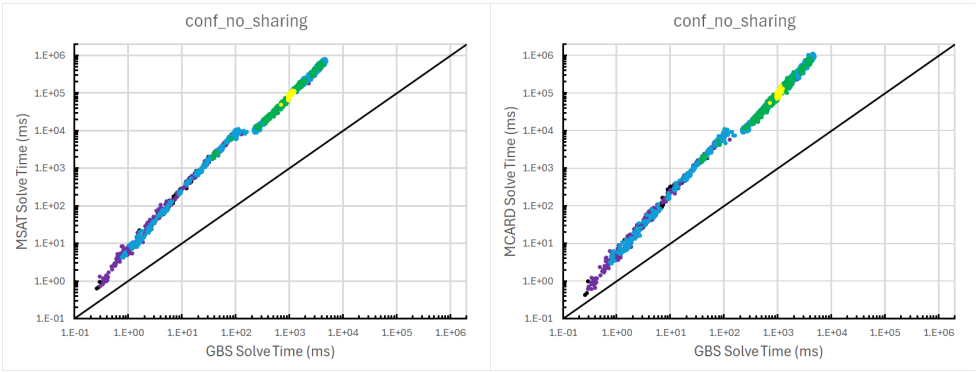
We now evaluate the relative performance of GBS against MSAT and MCARD on a case by case basis, analyzing the same data retrieved from Section 5.2.1. Figure 5.3 (left) shows the GBS runtimes of each problem for each BRS on the x-axis, while the y-axis plots the MSAT runtime for that instance—thus each point which appears above each $x = y$ diagonal line indicates an instance where GBS achieved a better time than MSAT. It can be observed that GBS outperforms MSAT on *all* instances, with relative performance also generally increasing as the scale of the problems increase—this is particularly noticeable for the set of conference BRS instances containing many solutions. As both solvers employ very different methods and techniques (clauses vs constraints), it was not initially expected that different instances of similar sizes would consistently demonstrate similar speedups. Speedups on instances with fewer/no solutions are more modest (which can be particularly observed in 80211 and savannah), however this would still be expected to have a substantial impact on performance for generating full transition systems as the solver would be likely be getting called thousands of times. Another key observation in regards to the conference instances is that matches with sharing scale significantly more linearly than those without sharing, which can be explained by the extra computation time devoted to filtering false candidate solutions through the additional checking constraints, therefore further performance improvements on larger instances may be possible through an improved implementation where these are instead enforced upon value assignment and detected earlier in search. As expected, it was



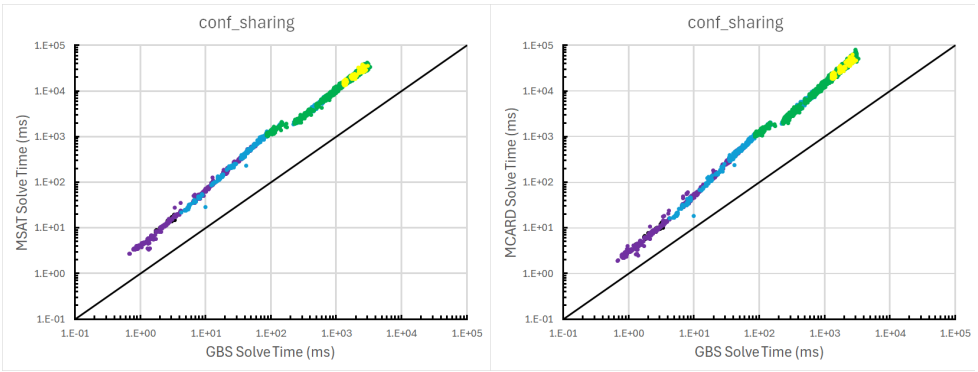
(a)



(b)



(c)



(d)

Figure 5.3: Comparing GBS against (left) MSAT and (right) MCARD for each instance of each family of matching problems. Instances are color coded as follows—Black: no solutions, Purple: 1-9 solutions, Blue: 10-49 solutions, Green: 50-99 solutions, Yellow: $\geq$ 100 solutions.

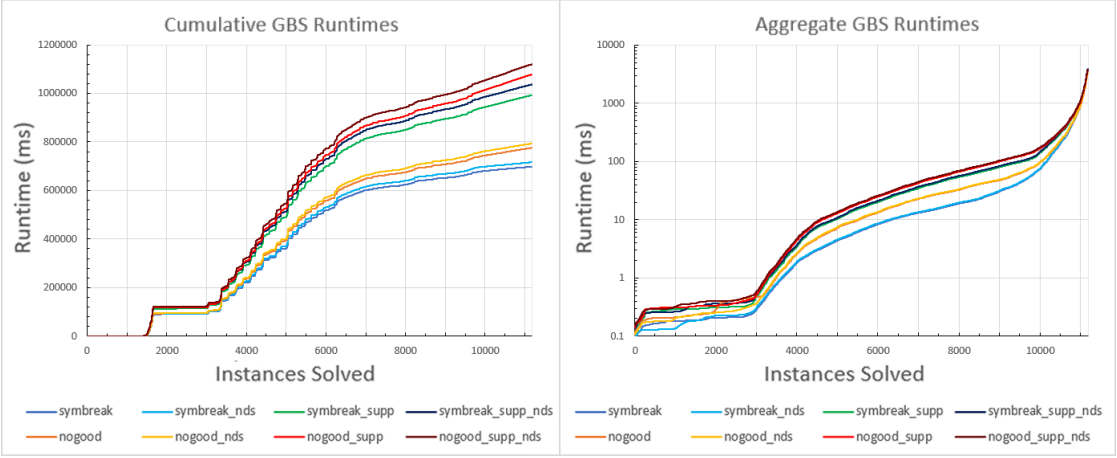


Figure 5.4: The solver runtimes achieved by GBS on the matching dataset for eight different parameter configurations. Cumulative runtime (left) measures the total time taken to solve n instances. Aggregate runtime (right) measures that the corresponding solver was able to solve n instances within a runtime $\leq t$.

observed that the number of SAT clauses increased rapidly as instances grew in size and complexity, but this is not evaluated further as this was prior known behavior for MSAT.

Figure 5.3 (right) shows the same GBS runtimes, except now plotted against the runtimes obtained by MCARD. Across all four BRSs, it can be observed in this case that GBS's performance gains are overall more moderate for smaller and easier instances. In total, GBS outperforms MCARD on all problems, except for 3 instances of savannah that which took both solvers $<$ 1ms to solve, however as noted earlier, this may suggest that potential runtime improvements for building transition systems which involve thousands of these more easy/trivial matches may be more limited.

5.2.2 Determining Optimal Configuration

We perform further evaluation on GBS, comparing its performance against itself for the cross product of all toggle-able parameters supplied by GSS in order to determine the most performant configuration. These experiments (and all subsequent experiments for this chapter) were run on systems with dual Xeon E5-2697A v4 CPUs and 512GBytes RAM, running Ubuntu 22.04. We primarily wish to determine whether our nogoods or symmetry breaking approach is more efficient for dealing with flattened hyperedge cliques, however additional features of GSS—particularly the building of supplemental graphs and performing NDS—could potentially have a negative effect on performance if more time is spent doing preliminary setup than time is saved from solving a more constrained version of the problem.

Figure 5.4 (left) shows the cumulative GBS runtimes on the 11,176 problem instances (ordered by no. solutions) for the cross product of three toggle-able parameters, totalling eight

different combinations: symmetry breaking vs nogood constraints, NDS vs no NDS, and supplemental graphs vs no supplementals. The first instances in the dataset are all UNSAT when ordered in this way, and it can be observed that the solver reliably instantly solves the majority of this subset of problems regardless of configuration used—most likely due to detecting failure early due to a domain wipeout produced by the preliminary degree constraints before ever reaching the main search loop. As expected, it can be observed that all four configurations which employ symmetry breaking constraints outperform their nogood counterparts. Interestingly, the largest negative impact on performance occurs for settings where the solver builds supplemental graphs—each of the four configurations where supplementals are turned on perform significantly worse than any configuration where this process is skipped. It can be observed in the aggregate runtime results in Figure 5.4 (right) that roughly 11,000 out of the 11,176 instances are solved within 1 second for all eight versions of GBS, suggesting that the instances become trivial for the underlying SIP backtracking algorithm to solve once encoded, i.e. requiring few search node traversals, regardless of whether additional pre-processing is performed. Preliminarily building supplemental graphs is hence likely to introduce unnecessary overhead on these easier instances.

It can also be observed in Figure 5.4 (right) that the aggregate runtimes begin to converge for instances that take longer to solve, suggesting supplementals may have some application for use-cases that involve really large and difficult instances, but is generally not needed for bigraph matching where many instances will be trivial. Similarly to our evaluation in Section 5.2.1, we again observe that the ≈ 3000 non-conference instances were trivial to solve for all configurations while the conference instances take an increasingly long time to solve. We deduce that while performing NDS has a less substantial impact on performance overall, skipping this step will result in a slightly better runtime when considering the total cumulative runtimes for all instances. On the other hand, there are some observable instances where NDS performs significantly better than no NDS despite being slightly slower overall—closer examination of the data shows that there are 850 total instances where **symbreak_nds** returns a runtime which is at least 20% faster than that obtained by **symbreak**, while there is no instance where **symbreak** performs more than 9.5% faster than **symbreak_nds**. This can be further seen in Figure 5.4 (right) where **symbreak_nds** is able to solve much more instances near-instantly compared to its non-NDS counterpart, resulting in a trade-off between the two options.

For subsequent experiments, we use GBS with symmetry breaking and NDS, and with supplemental graphs turned off.

Multithreaded Configurations

GSS can also be configured to allow for the parallelization of its underlying solution-biased search algorithm, where a small amount of randomness is introduced to the variable selection heuristic to ensure that multiple threads explore different branches of the search tree. This is combined with periodic restarts where the threads will share information regarding fully exhausted branches (nogoods) before resuming search [49]. The default implementation of solution-biased search in GSS also only begins parallel search after the first restart, as to avoid unnecessary time loss from handling overhead and thread creation when solving trivial instances.

When running GBS using these parallelization options, we observed no significant change in performance time compared to that of single-threaded search. In all real-world instances (Savannah, 80211), GBS was able to find all solutions before ever reaching the threshold for triggering a restart, meaning it never reaches the point of multiple thread creation regardless of whether it is toggled on. Even for the more difficult Conference model instances, no observable difference in performance could be discerned, suggesting that any potential advantages that parallel search provides is being offset by the overhead introduced through handling threads and nogood sharing, particularly when already, no instance ever takes less than 4 seconds to solve on any single-threaded configuration (the single worst instance/configuration pair measured using this hardware was 3856ms, using the **symbreak_supp_nds** set of parameters).

While we observe that use of parallelism within individual instances does not meaningfully impact the efficiency of search at this scale, the substantial aggregate speedup that GBS provides compared to MSAT/MCARD suggests that a more fruitful application of parallel processing in this case would be to solve batches of instances in parallel instead, where each thread runs a separate instance of GBS. In a BRS toolkit like BigraphER, this would likely be used to match multiple reaction rule patterns against a target agent simultaneously and allocating each of the derived pattern/target match instances amongst the available threads.

5.2.3 Equality Instances

As BigraphER performs frequent isomorphism checks in addition to matching, we also evaluate GBS's performance for finding equivalences between agent states when compared to MSAT and MCARD. This is performed on a set of 1265 equality instances derived from a BRS which models nurses, doctors and computer systems within a hospital. Similarly to matching, it was verified that GBS achieves the same solutions as BigraphER on all instances. Of the 1265 instances, 294 were found to be trivially unsatisfiable, i.e. were found

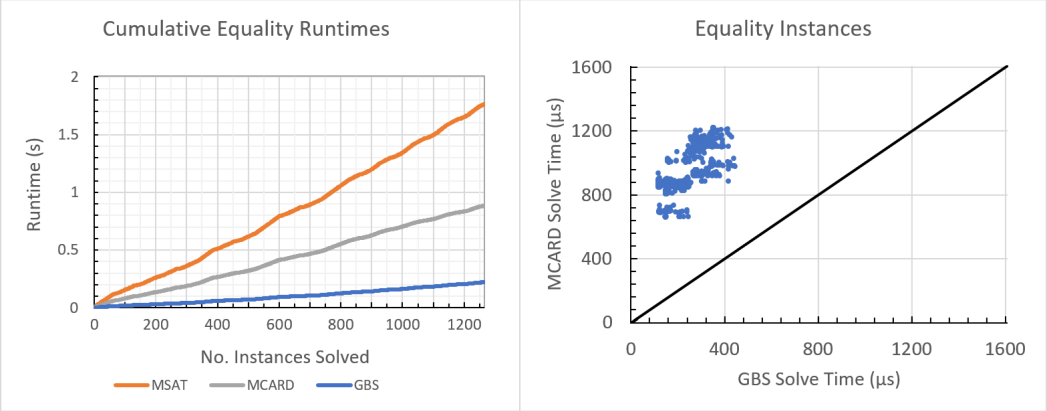


Figure 5.5: The solver runtimes achieved by GBS, MSAT and MCARD on the equality checking dataset. Cumulative runtime (left) measures the total time taken to solve n instances. Per-instance runtimes (right) measure the performance achieved by GBS against MCARD for each non-trivial equality instance.

to be unsatisfiable during input before search was called and returned a solution in less than 0.1ms for all solvers.

Figure 5.5 (left) shows the cumulative runtimes achieved by MSAT, MCARD and GBS on the set of equality instances. It can be observed that no solver took more than 2 seconds to solve the total set of 1265 instances—it is expected that equality is an easier problem to solve when compared to matching, due to the more constrained domains and only being required to find one solution rather than all solutions for an instance. GBS again demonstrates significantly better performance than the existing BigraphER solve tools in this context, achieving a cumulative speedup of 8 against MSAT and 4 against MCARD. This is roughly the performance improvement which would be expected for these easier problems based on the results for matching, as it was previously seen in Figure 5.3 that speedups are generally more modest for easy problems and more significant as the problems grow larger both in size and in difficulty.

Figure 5.5 (right) shows the GBS solve times for each non-trivial equality instance on the x-axis plotted against MCARD's time on the y-axis. GBS consistently outperforms MCARD on all instances of the dataset, demonstrating that GBS is the more promising solver for both matching and equality operations in a BRS based on these evaluations.

5.3 Evaluation: Full BRS Models

We now look toward employing GBS as an integrated matching component within BigraphER itself for running full scale model simulations and generating transition systems on a variety of BRSs, and compare the performance times against those achieved by MCARD

and MSAT. Whilst GBS demonstrates promising results for standalone instances of matching, this does not yet guarantee that the same performance gains will be observed in practice; firstly, BigraphER will also be spending time performing compilation and rewrite operations in addition to matching [9], where the choice of solver will not have an impact on performance. Secondly, depending on the method used to bind GBS to BigraphER, potential performance gains may be dampened through time lost encoding, sending and parsing data back and forth between them, which we take into account and examine the impact of on a subset of the evaluated BRS models (Section 5.3.3).

5.3.1 BigraphER Integration

GBS was integrated into BigraphER by initially adding functionality for encoding and flattening a pattern and target bigraph pair as required inside BigraphER's main solver function. This is used whenever BigraphER's solver module is called to perform a match on any input (P, T) pair, and **GBS** is specified as the solver to use in the command line input for running a BRS model. The resultant flattened bigraphs are then reconstructed in GBS through an interface module which connects to an API for GBS through a set of OCaml bindings, where BigraphER is able to call functions such as **add_node** and **add_edge** on a pair of graph instances in GBS to reproduce the required matching instance. The set of assignments in each solution are then returned in the standard format for BigraphER after solving.

This method of integration through OCaml binding is similarly employed for the previous MSAT and MCARD integrations, albeit with the additional required preliminary process of flattening the pair of bigraphs beforehand.

5.3.2 BRS Performance

The set of BRS models that were chosen for evaluation consist of the 37 BRS examples provided by the source code of BigraphER, which we make available [76]. Out of the 37, we discard any that were found to be too easy to solve for meaningful discussion of results, i.e. BRSs where all three solvers exhaust the full transition system in < 1 second on the benchmark hardware. The 13 remaining BRSs and the properties of their resultant transition system are provided in Table 5.1. **States** denotes how many agent states (and hence target instances) were discovered during generation. **Rules** provides the number of reaction rules (and hence the number of pattern redexes) of the BRS. **Transitions** denotes the total number of rewrite operations performed. **Occurs** corresponds to the total sum of solutions found across all matching operations. **Average State Size** and **Average Match Size** describe the average number of support elements of the target and pattern respectfully across all matching operations.

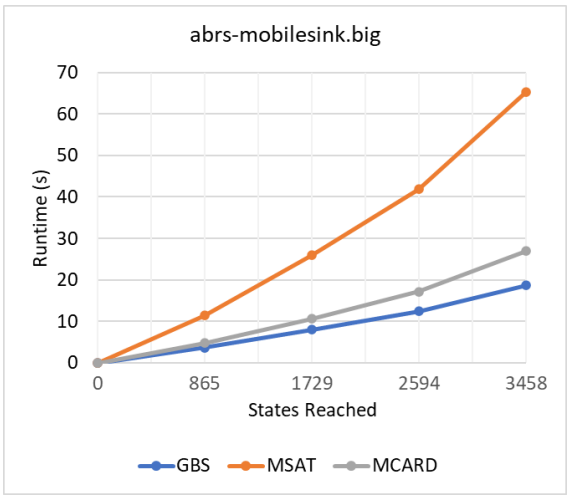
BRS	States	Rules	Transitions	Occurs	Average State Size	Average Match Size
abrs-mobilesink	3458	85	8413	17595	15.0931	7.12903
plato-graphical-loc	10000	9	49395	60715	28.3853	5.55556
virus-simpl	809	3	3972	4694	45	2.33333
hospital	2226	10	11794	12824	34.9704	7.8
plato-graphical	10000	9	59652	130360	34.8278	4.55556
savannah-general	10000	150	161342	187341	33.5633	7.96
dining_philosophers	815	4	3023	3193	21	9
virus-multifirewall	809	4	3972	4694	54	2.5
AutoBigraphER_127	1000	125	2783	5024	167.672	6.43562
AutoBigraphER_52	1000	125	2952	5203	145.777	6.43562
AutoBigraphER_83	1000	125	2859	5062	165.661	6.43562
floor_security_robot10_1	20	338	22	324	882.5	3.60997
link_inst_map	500	1	501	501	760.5	2

Table 5.1: The set of 13 example BRS models provided by BigraphER used for measuring GBS performance against MCARD and MSAT, split into 8 "standard" models (exhausts all possible states OR reaches 10,000 states in $<$ 30 minutes) and 5 harder models with significantly larger average agent state sizes.

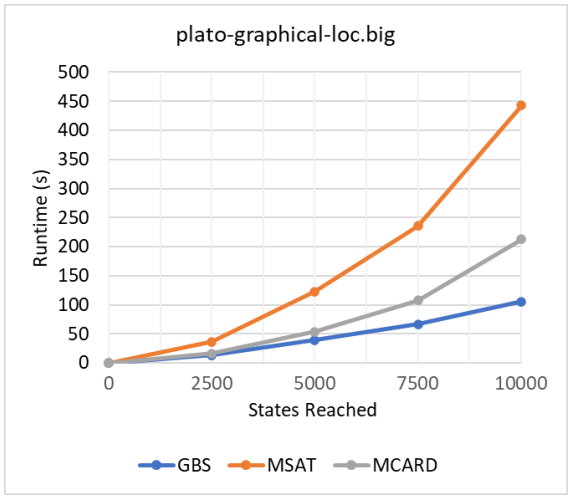
Out of the 13 chosen BRSs, we partition these into two categories based on their difficulty to solve, and evaluate them separately. 8 BRSs in the set were found to either fully exhaust their transition system or reach 10,000 unique states within 30 minutes for all solvers, which we denote as the standard set of models. 5 BRSs were found to be significantly harder to solve, taking all solvers at minimum several hours to reach the same threshold. We denote these 5 as the "hard" set of models, where the amount of states they must reach before terminating has been set to a substantially lower manually chosen value to ensure that all solvers complete each generation within 30 minutes. It can be observed in Table 5.1 that these models produce substantially larger average agent states (≥ 100) compared to the standard set. The full data tables for this evaluation are provided in Appendix C.

Standard Models

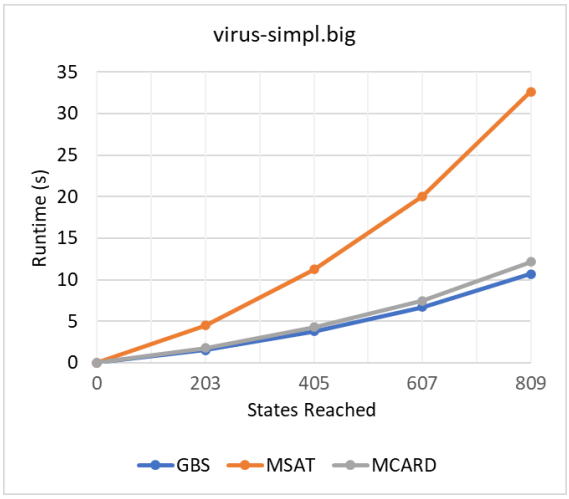
Figures 5.6(a) through (h) show the runtime performances achieved by GBS, MSAT and MCARD for each of the 8 standard-difficulty BRS models. The times taken to reach 25%, 50%, 75% and 100% of total states are plotted to show the relative scalability of each solver as the transition system grows. For all 8 BRSs in this dataset, GBS reliably provides a better overall performance gain over MSAT—with runtimes that range from 72% faster for the **dining_philosophers** BRS in Figure 5.6(h) up to 421% faster on **plato-graphical-loc** in Figure 5.6(e), for reaching all states. However, performance relative to MCARD is shown to be more competitive; while significant gains greater than 65% relative to MCARD are shown



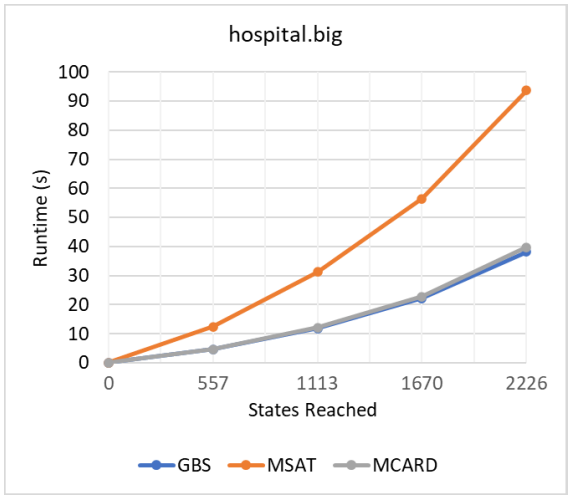
(a)



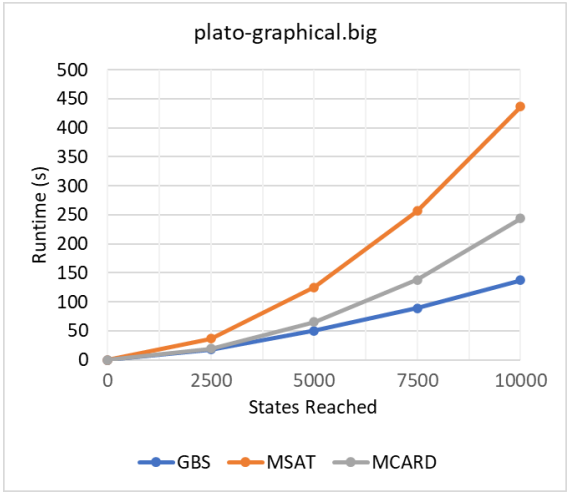
(b)



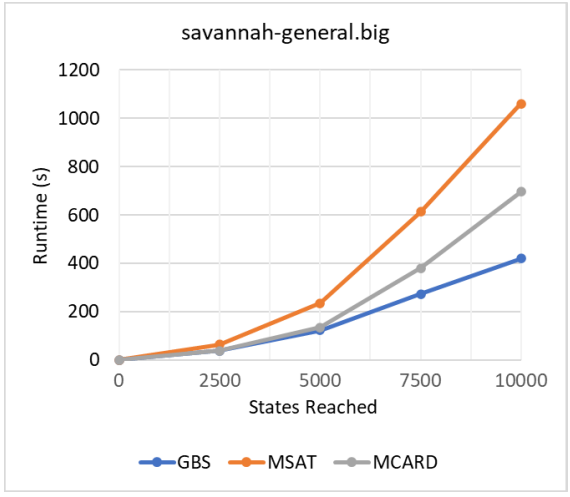
(c)



(d)

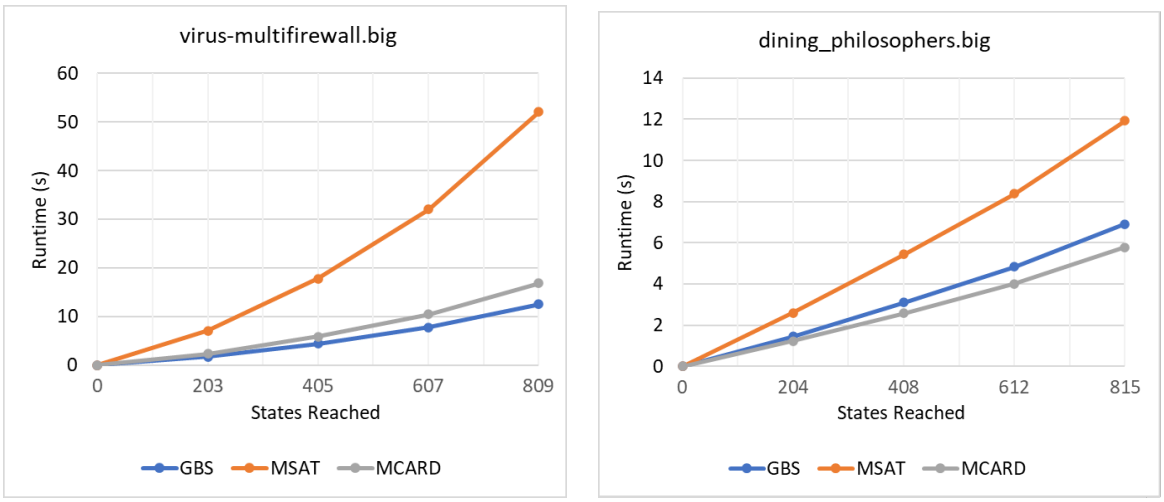


(e)



(f)

Figure 5.6



(g) (h)

Figure 5.6: Comparing the performance of MSAT, MCARD and GBS for building the transition system on the selection of standard-difficulty BRS instances, showing the number of unique states reached over time for each solver.

in Figures 5.6(b), (e) and (f), MCARD's runtimes are otherwise more comparable to, and for one BRS faster overall than as shown in Figure 5.6(h), those obtained by the integrated GBS. This is investigated further in Section 5.3.3.

Harder Models

Figures 5.7(a) through (e) show the runtime performances achieved by the three BigraphER solvers for each of the 5 harder-difficulty BRS models. For the three **AutoBigraphER** models as shown in Figures 5.7(a), (b) and (c), which simulate a routing protocol for low-power and lossy WSNs, GBS provides significant improvements in overall solve time, gaining speedups of over 5 against MSAT and a gain over 75% against MCARD. On the other two hard BRSs however, GBS underperforms both MCARD and MSAT.

We can understand this result for Figure 5.7(e) by closer analysis of the model structure. As shown in Figure 5.8, the **link_inst_map** BRS defines only a single reaction rule containing two unconnected entities C and Dup, which rewrites a state by adding an additional Alg entity as a child of C. For this particular instance (not shown in the figure), BigraphER also allows the rewrite to duplicate the subtree of site 0 in r and compose it as a subtree onto site 2 in r'. It is also shown that the reactum bigraph r' and initial agent state A each only contain one instance of C and Dup, meaning that one single matching solution will always exist and can be trivially found by comparing control types of entities regardless of how large the agent may grow, while each subsequent state adds an additional A, B and Alg entity, and the new A and B join their respective hyperedges. As the match will always remain trivial, we deduce (and later demonstrate in Figures 5.9(c) and (d)) that the bulk of time doing matching

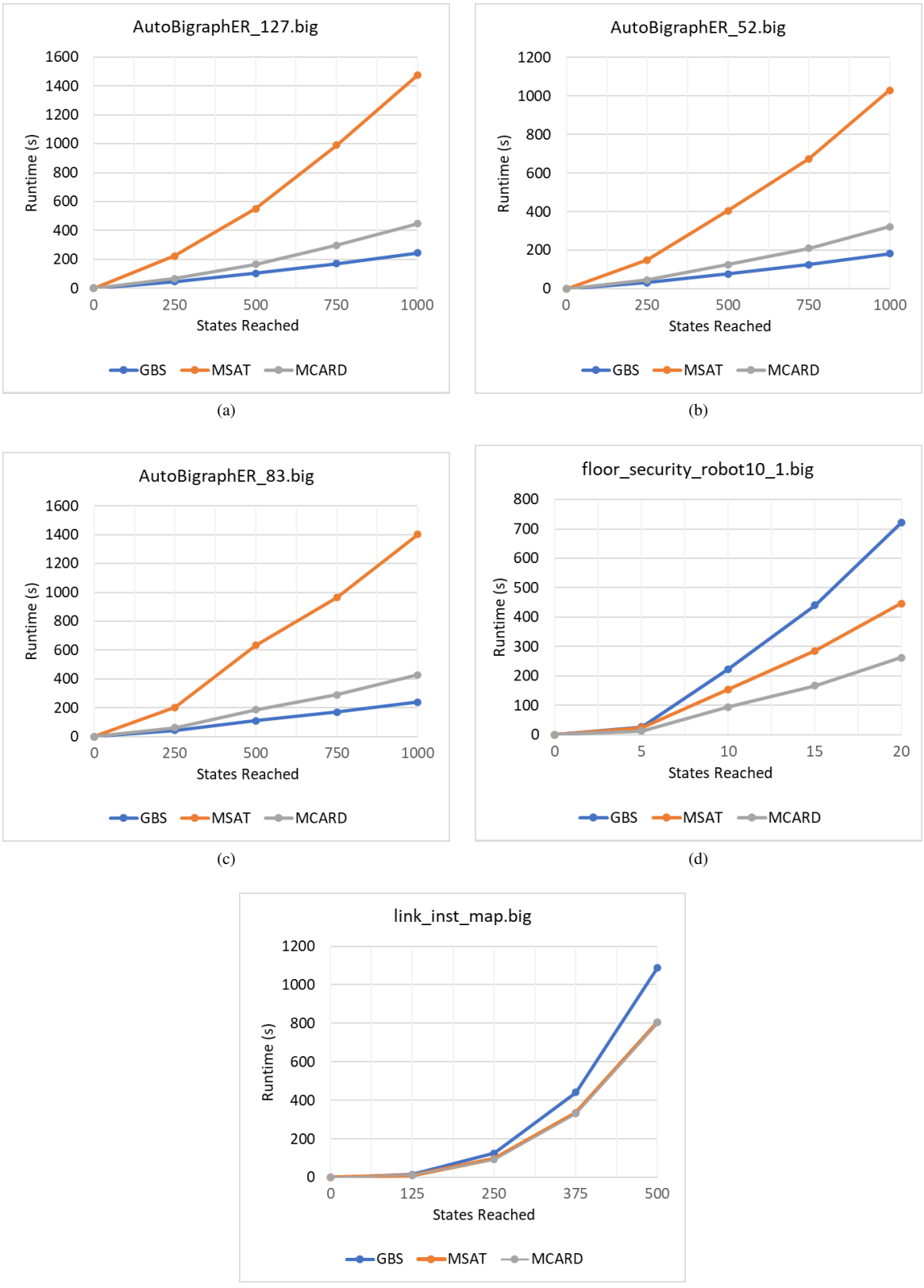


Figure 5.7: Comparing the performance of MSAT, MCARD and GBS for building the transition system on a selection of BRS instances identified to be particularly difficult to solve.

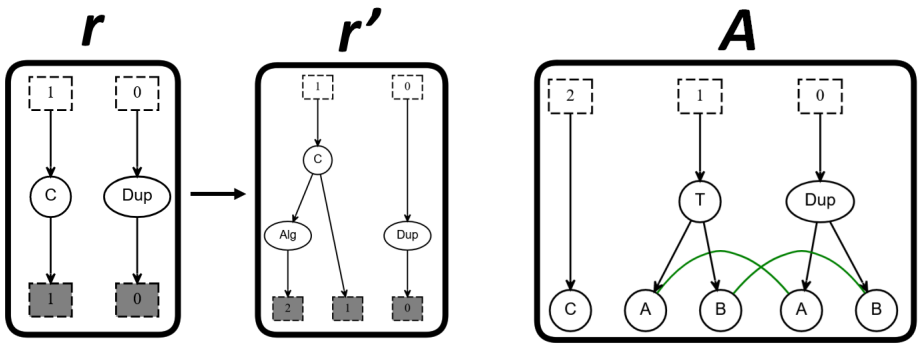


Figure 5.8: The sole rewrite rule and initial agent state of the **link_inst_map** BRS model provided by BigraphER.

for GBS will be spent in the flattening and construction process, which becomes increasingly computationally expensive as the target states and their hyperedge cliques continue to grow, which reflects the relative worsening of performance over time that can be observed in Figure 5.7(e). While **link_inst_map** is a more abstract BRS which does not model a real-world use case, our reason for including this in our suite of tests are twofold; firstly, we wish to evaluate *all* example models provided by the BigraphER repository to ensure that our evaluation is fair and exhaustive. Secondly, it may still highlight a weakness of our current approach that could also potentially occur for more practical usage, and thus we wish to record this as part of our evaluation. Specifically, this highlights that instances involving large target states and easy matches may present a challenge for BigraphER with GBS relative to the old solvers' performances without a more efficient method for building the flattened bigraph encodings in GBS.

We analyze GBS's performance for **floor_security_robot** further in Section 5.3.3 by examining individual instances of matching in the BRS.

5.3.3 Investigating Underperforming BRSs

To measurably determine the impact our GBS integration method has on overall performance, we extract the sets of pattern redexes and target states from BigraphER for the two BRSs where we observe an unexpected underperformance from GBS, and evaluate these separately. For **dining_philosophers**, the combination of its 815 possible states with its 4 redexes were retrieved to give 3260 instances of matching, while the 338 redexes of **floor_security_robot** were retrieved along with its first 10 reached states to produce 3380 total matching instances. For completeness, we also evaluate the lone pattern against the 100 target states of **link_inst_map**.

For each extracted set of instances, we measure the cumulative runtimes achieved by the matching engine of BigraphER with GBS, including time spent encoding, flattening and

constructing the instance. We compare these to the cumulative search time reported by the main solver function of GBS when called directly, including constraint generation time but ignoring input/output time in a similar fashion to our initial matching evaluations (Section 5.1). We compare these results against those achieved by MCARD in order to provide additional context to the difference in performance observed.

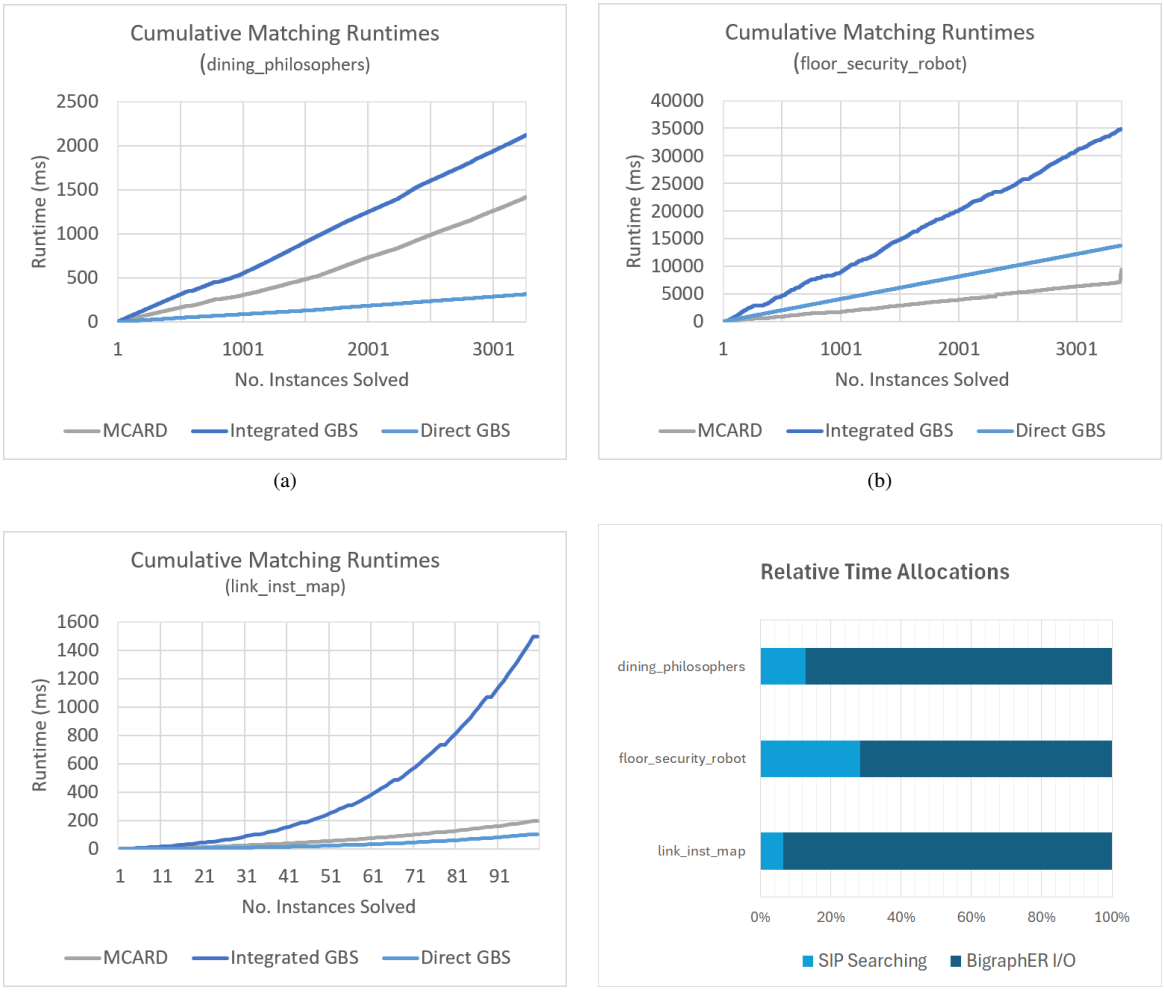


Figure 5.9: The cumulative runtimes for the sets of extracted matching instances for the (a) dining philosopher, (b) floor security robot BRSs and (c) link inst map BRS, comparing the total runtime of the integrated GBS against GBS search time achieved when called directly—the corresponding runtimes of MCARD are included as an additional point of comparison. (d) Time spent performing search relative to BigraphER flattening and I/O operations for each BRS.

Figures 5.9(a), (b) and (c) show the cumulative runtimes achieved by the integrated GBS compared to calling the GBS solve function directly, on the set of matches for each of the three identified underperformant models. Figure 5.9(c) demonstrates that our hypothesis explaining the slow runtimes reported in **link_inst_map** (Section 5.3.2) is accurate, spending 93% of reported time inside the BigraphER matching module handling the flattening and

input/output between GBS for the bigraph pair. This is compared to 85% of total runtime for the **dining_philosopher** instances and 61% for **floor_security_robot**, as shown in Figure 5.9(d).

We also observe for the **link_inst_map** runtimes in Figure 5.9(c) that integrated GBS appears to scale geometrically, despite GBS showing more linear scaling for the other two underperformant models. This is because as previously observed in Figure 5.8, the states of **link_inst_map** in particular scale in direct proportion to the number of transitions made, where each consecutive state update is adding a new bigraph where one additional entity is added onto the previously largest state. This causes the I/O processing time to scale polynomially, where it will encode each state with support size k to $(k + n)$ at each update step n when starting with an initial state size k. This is in contrast to more practical models like **dining_philosophers** where the average state size will not observe a substantial increase until at least after thousands of transitions (Table 5.1).

Across the three BRSs, it was observed that each model mainly consisted of relatively easy matches for GBS; the most difficult single link inst map matching instance was solved by GBS in 2.73ms, whilst this was 0.24ms for dining philosophers and 7.08ms for floor security. This suggests that while the integrated GBS in its current state shows promise for solving BRSs involving harder matches, an efficient method of passing data between the two components is necessary in order to take full advantage of the faster solve times that the SIP encoding offers, for models that mostly consist of trivial matches—particularly on large target states. One obvious avenue to explore for mitigating input/output bottlenecks would be to avoid performing flattened graph construction on any bigraph which has already been processed in a previous instance of matching; this could be achieved by modifying the matching engine (and by extension, GBS) to accept *several* patterns and a single target whenever attempting to branch from a given transition system state, ensuring that the same target isn't encoded multiple times, which is particularly beneficial if a BRS contains many reaction rules. A method of caching the set of reaction redexes inside GBS would also ensure that the same pattern graphs are not redundantly encoded multiple times in a simulation, and caching all received targets would similarly allow for more efficient checking of support equivalence between agents.

One important observation that we additionally observe appears in the runtimes for floor security in Figure 5.9(b); it is shown that regardless of integration performance, the actual solving time achieved by GBS underperforms that of MCARD; overall, direct GBS took 47% more time to solve the set of matches. Investigation on a selection of floor security instances where GBS performs particularly poorly compared to MCARD showed that in each of these cases, over 50% of solve time was spent building the transitive reachability matrix of V_T as a preliminary process prior to running the main search loop. We hence propose that some method of caching this structure in addition to the target graph itself inside GBS, avoid-

ing further redundancy, can potentially allow for faster overall solve times in the context of transition system building. The floor security BRS is a particularly good example of where this would be a useful potential improvement due to its 338 reaction rules, as this will currently be unnecessarily performed upon the same target graph for each $(r_1, T), ..., (r_{338}, T)$ instance.

5.3.4 Scalability Assessment

Finally, we now explore the potential of GBS for expanding the scale of larger simulations which BigraphER can support in real-time, in cases where the transition system grows to the point where MCARD and MSAT begin to suffer scaling issues. It can be observed in Figures 5.6 and 5.7 that different models may scale at different rates over time, and this depends on the complexity of the underlying bigraphs in each state and how they evolve as the transition system grows through numerous rewriting operations. For example, a model may exhibit almost-linear scaling if many of its reaction rules do not significantly change the underlying state agents, or in some cases may even reduce their size or complexity if the reactum bigraph is smaller than the redux bigraph — this almost-linear behavior can be observed in the **AutoBigraphER** model runs (Figures 5.7(a) through (c)). Conversely, a model will likely demonstrate more exponential scaling if the iterative application of reaction rules produce increasingly more complex agent states as the system grows, resulting in more difficult matching instances as the simulation progresses. Another important factor affecting this phenomenon is that each new candidate state also needs to be isomorphism checked against all previous states to ensure no two equivalent states are mistakenly added to the transition system as separate nodes, which can quickly grow computationally expensive when dealing with systems involving tens of thousands of possible states when the underlying support equivalence solver is not optimized.

For this assessment, we specifically focus on the **plato-graphical** and **plato-graphical-loc** models, as they are BRSs which produce infinite transition systems and exhibit the previously described behavior. It can be observed from Figures 5.6(b) and (e) that the time required to add more states demonstrates a more geometric pattern for this pair of models. Informally, it was also observed through preliminary test simulations that while BigraphER is able to simulate these models at hundreds of states per second initially, they begin to bottleneck as their systems grow beyond tens of thousands of states when relying on the MCARD and MSAT solvers. We do not consider the other large model **savannah-general**, as this BRS simulation eventually completed at 20666 states and therefore was found to be finite.

To measure the scale at which each matching/isomorphism solver can support these models maintaining acceptable performance, we add a timeout clause to BigraphER's `scan` function. Calling the `scan` function on a target agent state T performs the following process:

Solver	plato-graphical	plato-graphical-loc
MSAT #1	48657	19294
MSAT #2	48515	19054
MSAT #3	48638	19333
MSAT Avg.	**48603**	**19227**
MCARD #1	61645	30065
MCARD #2	61118	30065
MCARD #3	61639	30065
MCARD Avg.	**61467**	**30065**
GBS #1	1107427	1031491
GBS #2	1107343	1031491
GBS #3	1107414	1031491
GBS Avg.	**1107394**	**1031491**

Table 5.2: The number of states BigraphER was able to reach before the state update function timed out ($\geq 1s$), per solver.

- Calls the underlying solver to solve $\texttt{match}(r, T)$ for each reaction rule $R : r \to r'$ in the BRS.
- For each match found for each r, performs the corresponding substitution in T to produce a set of new candidate states $C = \{C_1, ..., C_m\}$.
- For each $C_i \in C$ and each agent state $A_j \in \{A_1, ..., A_N\}$ already in the transition system, calls the underlying solver to solve $\texttt{equality}(C_i, A_j)$.
- Adds a new transition system node for each C_i where $\texttt{equality}(C_i, A_j) =$ **false** for all $A_j \in \{A_1, ..., A_N\}$, and adds a directed edge from T to C_i.
- Otherwise, if C_i is an isomorphism of an existing state A_j, simply adds a directed edge from T to A_j.

Overall, this function handles the branching of a given agent state to all states it can evolve into through pattern matching and then rewriting of all reaction rules in the BRS. For each of the three available solvers, we record the size that the transition system is able to reach before a call to the `scan` function takes more than 1 second to complete for a state (in which the timeout will then trigger and halt the program) as the simulation will be too slow for practical modelling beyond that point; this allows us to measure the impact that swapping MCARD/MSAT with GBS will have on the scalability of BigraphER as a whole. We perform this three times for each model/solver pair — in doing so, variance in results between runs was observed to be very low, likely due to the transition system building process being deterministic (it will always simulate states and perform the same matching/equality operations in the same order each time).

Table 5.2 displays the sizes of the transition systems reached for each model/solver pair before timeout. We immediately see that while MCARD does slightly better than MSAT for this benchmark, GBS heavily outperforms both other solvers, allowing BigraphER to grow the system demonstrating acceptable performance to just beyond one million nodes on both evaluated BRSs. As a ratio, GBS was able to increase the scale of the simulation under this criteria by a factor of 34.3 over MCARD and 53.7 over MSAT on the **plato-graphical-loc** model, and 18.0 over MCARD and 22.8 over MSAT on the **plato-graphical** model.

Deeper analysis of the underlying bigraph states and matching operations being performed reveal that in this instance, while the agents tended to grow larger as they evolved, they never reach a drastically large size — within the first 50,000 steps, the largest agent in the **plato-graphical** system was found to have a support size of 75 (from an initial agent size of 6), and for **plato-graphical-loc** this was 40 (from an initial size of 9). This suggests that while the underlying matches are indeed getting gradually more difficult as the system grows, another likely contributor to MSAT and MCARD's scaling issues in this instance upon reaching tens of thousands of nodes is the increasingly large number of required equality checking operations required per state update. When determining whether a candidate state can be added as the nth node of the system, a maximum of $n - 1$ equality operations against all other existing states must first be performed to ensure its uniqueness, which — at the point where the toolkit must support a transition system containing hundreds of thousands of nodes — threatens to take up the vast majority of processing time in order to grow further. This highlights the importance of having an efficient equivalence solver under the hood in addition to our optimized pattern matching algorithm, when it comes to supporting large-scale simulations. Overall, these results are a promising sign that through our adapted subgraph solving algorithm, we are now able to support significantly larger and more complex models where a SAT-based approach would previously struggle to perform at scale.

5.4 Summary

In this chapter, we have demonstrated that through adapting an efficient subgraph solving tool to solve bigraph matching, we can achieve a significantly better performance over the current state of the art SAT tools, including an aggregate speedup of over two orders of magnitude against both previously used solvers on a large suite of pattern/target pair instances. This can be integrated into the BigraphER rewriting engine, where our adapted solver provides the best runtime performance on a majority of 13 evaluated BRSs which is sufficient to showcase the usefulness of this approach, as well as demonstrating promising resilience on BRSs which grow to the size of hundreds of thousands of states. However there exist possible engineering improvements that may allow for further, more substantial gains in a practical

context.

Currently, BigraphER with GBS considers every (P, T) matching operation in isolation whenever required, encoding and flattening the pair of bigraphs in BigraphER before passing function calls to rebuild and subsequently solve them in GBS. However, this will ultimately result in time wasted repeatedly flattening the same graphs; for a BRS model with r reaction rules and a possible agent states, each target state will be redundantly flattened $r - 1$ extra times to apply each reaction rule on it, and each redex of a reaction rule will also be flattened up to a times for every attempted branching of an agent. This particularly negatively impacts relative performance against MCARD/MSAT on BRSs with a combination of lots of reaction rules, large target states and easy matches, where BigraphER/GBS will end up spending the vast majority of its matching time stuck in the flattening and setup functions as seen with the **floor_security_robot** model (Section 5.3.3). Hence, the following adaptations can potentially be made to alleviate these bottlenecks:

- Adapt the BigraphER solver module to accept the *full set* of reaction redexes $\{r_1, ..., r_n\}$ in the BRS when attempting to match them to an agent A, instead of performing every match (r_1, A) to (r_n, A) as independent instances. This would also involve adapting GBS to accept an arbitrary number of pattern graphs as input and apply them all to one given target as one collective operation, ensuring necessary setup processes like building the reachability matrix for the target for applying sharing constraints is only done once.

- Investigate methods of caching the set of reaction rules inside GBS, removing the need to flatten and rebuild the set of pattern bigraphs more than once for a full model simulation.

- Investigate methods of caching any received target state inside GBS during a simulation, ensuring faster equivalence checking operations.

- Implement the shared site/region and transitive closure constraints (Section 5.1.1) as constraints propagated during value assignment rather than checking constraints applied on candidate solutions.

Whilst we identify these specific areas for improvement, additional effort spent on optimizing I/O and implementing caching was deemed outside of the scope of this dissertation's research, as this wholly concerns further software engineering tasks to more tightly integrate GBS as a component into BigraphER rather than relating to the logic of the underlying matching algorithm itself, i.e. the main primary scope of this work. Hence, now that we have sufficiently demonstrated that it is feasible to use our adapted SIP algorithm as the back-end

solver in a BRS toolkit to achieve faster transition system building in the majority of evaluated models with our current method of integration, we leave these extra optimization efforts as a promising future exercise for refactoring BigraphER's handling of data transfer and improving further upon performance and applicability.

It is known to be difficult to determine in advance whether a SIP instance will be hard, as even small instances can sometimes be computationally intensive depending on their structure [38]. This means that proposing a "hybrid" approach which automatically selects the most optimal solver for each type of model would be a difficult endeavor. However, because of the portability of our approach to GBS, this allowed us to integrate it into BigraphER without removing any existing infrastructure, and thus support giving the user the option of which solver to use as an input command line parameter (either MCARD, MSAT or GBS) without any difference in how the rest of the tool will operate. We thus provide the user the ability to determine and select the most ideal solver for whichever context they may be working in, e.g. **floor_security_robot** where it can be determined via brief simulation testing that MCARD is the fastest available solver.

In the following chapter, we build upon our bigraph matching algorithm to consider a possible optimization variant of the problem based on the relation between SIP and maximum common induced subgraph (MCIS), and its potential to allow for support for *labelled* transition systems and identifying minimal contextual transitions, if also added to BigraphER or other bigraph rewriting engines. We thus define the maximum common bigraph (MCB) problem, identify its similarities to both bigraph matching and MCIS, and introduce an algorithm for MCB based upon the state of the art McSplit partitioning and backtracking MCIS algorithm with additional constraints [14].

Chapter 6

Introducing Maximum Common Bigraph

In this chapter, we build upon our previous work to introduce an algorithm for solving the *maximum common bigraph* (MCB) problem, which aims to find the largest shared area (support-equivalent substructures) between two bigraphs. This allows us to identify the minimal contexts required to add to an agent that allows a *contextual* reaction rule to apply in a transition system, which then allows for identifying bisimulations, a powerful utility for optimization and verfication — which cannot currently be achieved through existing BRS tools.

Section 6.1 discusses the motivation and use-cases that would benefit from a maximum common bigraph algorithm. Section 6.2 evaluates the graph-equivalent large common subgraph problem, as well as commonly used algorithms for solving instances of MCIS. Section 6.3 proposes our definition of a maximum common bigraph and its expected observable properties. Section 6.4 describes the adaptation of McSplit to retrieve the MCB of two agent states. Section 6.5 provides a proof of soundness and completeness of this approach. Section 6.6 extends this adaptation further to identify all occurrences where the composition of a minimal context would permit a full match to occur in an agent bigraph. Section 6.7 describes the implementation of our prototype McSplit solver. Section 6.8 provides some preliminary performance metrics and observations for our prototype implementation. Section 6.9 summarizes the chapter, where we note some observations and avenues for further research on MCB.

6.1 Motivation

Our primary motivation is as follows: currently, there is no known algorithm which is concerned with defining, nor finding a maximum component between two abstract bigraphs. A similar concept exists in the form of identifying *relative pushouts* (RPOs) of bigraphs—that is, the shared decomposition G of the *span* and *cospan* A_1 and A_2 for one greater bigraph structure A, where no further composition onto G is possible (other than the identity bigraph) without preventing it from occurring in both A_1 and A_2—in other words, the composition is in a saturated, maximal state. Pushouts however are only concerned with concrete bigraphs where there already exists a support mapping between A_1 and A_2 supplied by A, rather than searching for this mapping to begin with between two abstract bigraphs in a BRS. Further reading into bigraph pushouts is described in further detail in Appendix B, but for the purposes of this dissertation, we only need to understand that they define a minimally bounded triple relation between A_1, A_2 (decompositions of A) and a shared decomposition G between the two.

An algorithm which is able to identify a maximum mapping between two abstract bigraphs as a generalization of bigraph matching, i.e. finding the largest possible set of valid support assignments if a full match does not exist, would open up an avenue toward supporting *minimal contextual transitions* in a BRS, where the smallest possible bigraph is initially composed onto the agent as a context to allow an otherwise invalid match to appear. This is because such an algorithm could identify the largest shared region between the reaction rule redux r and the agent A to determine exactly what needs to be provided in the minimal context (Section 6.1.1). This would then allow for establishing properties such as *bisimulation* between agents in a BRS — the guarantee that two agents will always behave in the same way within any possible environment — through mapping out and comparing their respective minimal contextual transition systems, which cannot be guaranteed using a standard transition system which relies upon full matches (Section 6.1.2). Hence, if a maximum common bigraph algorithm were devised and implemented into a tool such as BigraphER to allow for simulating more flexible and sophisticated transition systems, this opens up the potential for introducing more rich, efficient and complex modelling techniques for real-world models.

6.1.1 Labelled Transition Systems

Up until this point, we have been wholly concerned with modelling transition systems where a reaction rule must already exist within an agent to be able to perform its corresponding rewrite operation, as described in Section 2.1.2 — this can be described as mapping out the *raw* transition system of a BRS. However, this presumes that the agents we wish to simulate must exist in a vacuum without any additional context, instead of as a component

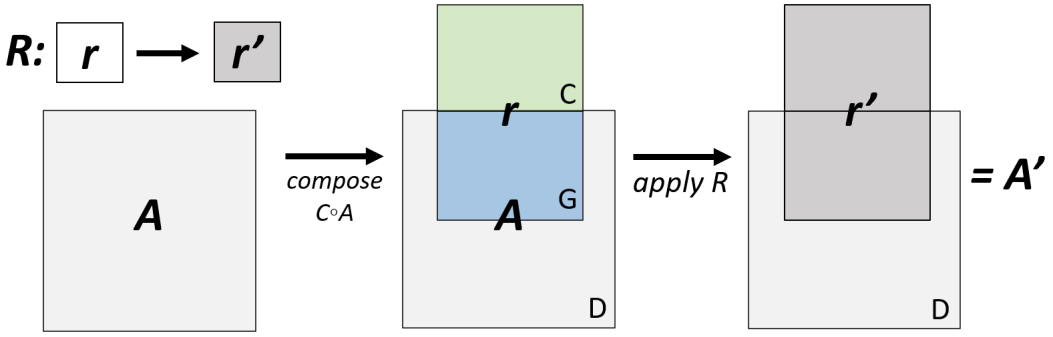


Figure 6.1: A high level representation of a transition in an LTS, where a context C from the environment of agent A is first composed to produce $C \circ A$, such that $r = C \circ G$ (for some remainder bigraph G within A) now occurs in the agent and the reaction rule $R : r \to r'$ can be applied.

within a wider surrounding environment. In reality, there may be unknown entities and links connected to the agent we wish to account for, which can impact the behavior of how the state could potentially evolve over time. A hypothetical more powerful modelling paradigm would be able to account for this and allow for transitions which take into account the possibility that the wider context connected to the agent could supply the missing structure needed for a match to occur, allowing a reaction rule to apply that wouldn't otherwise. A transition system of this form is known as a contextual *labelled* transition system (LTS) [78].

In an LTS, a transition between states can be denoted as $A \xrightarrow{f} A'$: where A and A' describe the agent states before and after applying the transition, and f denotes the *label* of the transition. In a *full* transition system for a BRS as defined by Milner [1], the label f takes the form of a bigraph which represents a piece of the wider environment, and is composed onto the agent A as a context as a preliminary step, to produce $f \circ A$ before applying the reaction rule to it. This means that effectively any reaction rule can apply to any agent state in a bigraph LTS, as long as f provides the appropriate context to act as a dependency. Figure 6.1 provides a simplified visual example of this process, where f first provides the necessary context bigraph C before the state transition, allowing the reaction rule redux r to appear. Raw transition systems can be seen as a simplified specification of LTSs as a whole, where f is restricted to $\emptyset$ and cannot change whether a reaction rule can apply or not, as the structure of A remains unchanged.

The implementation of an LTSs opens up the ability to simulate not only the standalone behavior of an agent, but also its potential behavior in *any* possible context, where its above environment could influence its behavior and therefore the possible states it is able to reach — and thus provides a more expansive and powerful modelling paradigm than the raw transition systems that we have been working with thus far. This would allow us to verify the bisimulation of agents, i.e. given two different agents that behave similarly, we may wish

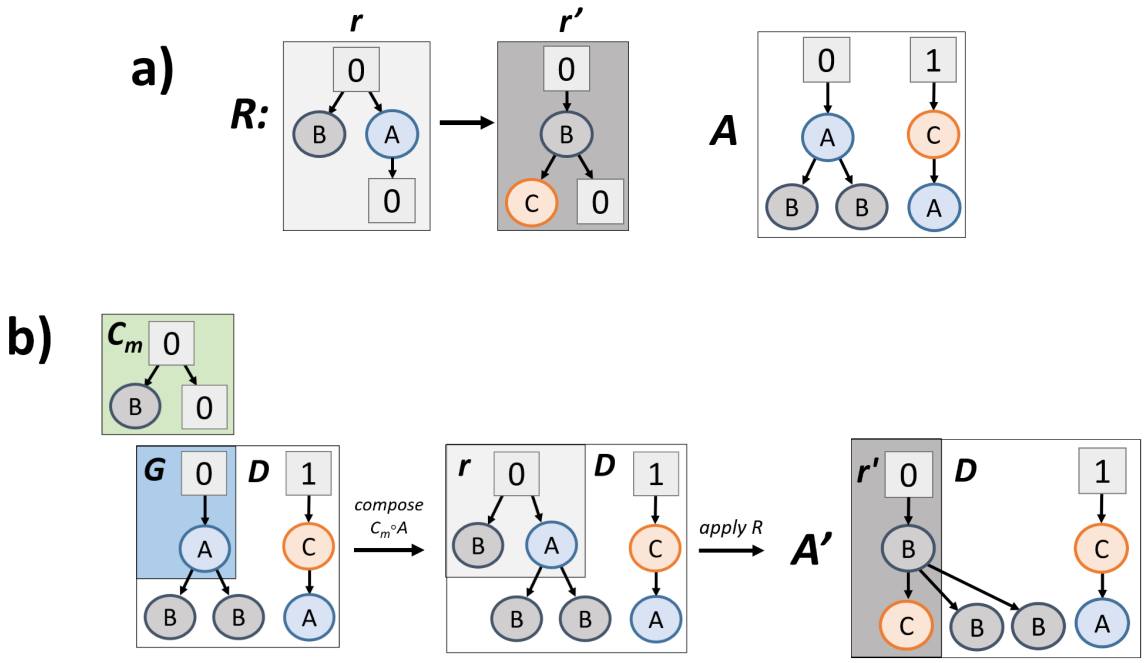


Figure 6.2: A high-level view of how a BRS can apply a minimal context C_m to agent A to allow the application of reaction rule $R : r \to r'$, in order to rewrite A to produce the successor agent A'. The blue area denotes the overlapping area G between r and A, and D is the parameter of the constructed matching that can now occur.

to determine whether they always exhibit the same behavior regardless of any environment they may possibly be placed in (Section 6.1.2). However, given the current lenient definition of an LTS, there are hypothetically infinitely many contexts which could be supplied to any reaction rule, which is not practical to model as a transition system. In addition, *where* the contextual bigraph f is composed onto A is not specified, and it is possible that the same f could be composed onto more than one combination of regions in A that allow the reaction rule to apply, but where the resultant agent states will differ in structure. Thus, we wish to define a specification of LTSs that both supplies information about where the additional context is placed, as well as restricting the possible range of contexts to only the minimal structure necessary to allow the reaction rule to fire.

Milner defines a *contextual* transition system (CTS) as an LTS that executes transitions in the form $A \xrightarrow{(f,j)} A'$, where the context bigraph is paired with a contextual location j which defines the mapping from the inner faces of f to the outer faces of A. Each contextual transition is based upon an underlying reaction rule $R : r \to r'$, such that $f \circ A$ is equivalent to $r \circ D$ for some remainder bigraph D, and A' is thus equivalent to $r' \circ D$.

A transition within a CTS is considered a *minimal transition* if the produced triple relation $((f, j), r, A)$ is an RPO (Appendix B) — that is, it is minimally bounded, and there exists no other contextual label (f', j') where f' occurs within f, and the alternative transition relation

$((f', j'), r, A)$ can also be applied. In other words, for the transition to be minimal, f must provide to A *only* what is missing from r and nothing more. A minimal transition system can hence be defined as a CTS which is restricted to only minimal transitions, which is sufficient for capturing any possible state the agent may reach through influence from its environment. An example application of a minimal context for an LTS is shown in Figure 6.2.

The challenge of algorithmically identifying the minimal context C_m can be viewed as equivalent to identifying a maximal overlapping area G between r and A, such that $r = C_m \circ G$ and $A = C_r \circ G \circ D_r$ where the resultant bigraph $C_m \circ A = (C_m \otimes C_r) \circ (r \otimes id) \circ D_r$ is produced. It logically follows as shown in Figure 6.2 that C_m can then be obtained by decomposing all elements of G from R. In other words, we would first require a method of finding the *largest possible* instances of a common bigraph G between r and A where we can take the complement of G from r to obtain C. As the lower face of C_m must mirror the upper face of G where the interfaces of C_m are subject to the minimal bound constraint, we can infer that the interfaces of G must also be minimal to match and similarly be closed/merged. Hence, in order to support the building of minimal transition systems, the need for a maximum common bigraph algorithm arises as an initial necessary step and must be addressed. This proposed strategy would follow a similar principle to using maximum common subgraph to solve graph edit distance problems, but within the context of bigraphs [79].

6.1.2 Bisimulations for Bigraphs

A *bisimulation* between two agent states A and B in a transition system describes a symmetric relation such that A and B will always exhibit the same behavior, and hence can be considered functionally equivalent regardless of any internal difference in the composition of their structures — in essence, they simulate one another. A pair of agents are called *bisimilar* (labelled $A \sim B$) if there exists a bisimulation between them [80], and the transition systems produced by setting either A or B as the initial state will be indistinguishable from one another. Being able to verify whether two states are bisimilar is a powerful tool for model checking, as this can allow for the simplification, optimization and the reducing of cost of models and systems through replacing more complex components with smaller or cheaper components which are verified to be bisimilar. Existing real-world examples of where identifying bisimulations has been utilized include ensuring equivalence after refactoring databases [81], optimization of large-scale graph processing [82] and simplifying complex environments within deep reinforcement learning [83].

Within the domain of bigraphs specifically, engineering a BRS toolkit such as BigraphER to be able to model the minimal CTS of a model, and hence be able to determine whether two bigraph states are bisimilar by comparing their resultant minimal CTSs, which can be performed in polynomial time [84]. This would hence meaningfully expand upon the exten-

sibility and practicality of BRSs as a modelling paradigm. Potential real-world applications include the following:

- **Simplification of Models:** Bisimilar components are interchangeable within a model without changing the overall state behavior. Hence whenever a bisimulation $A \sim B$ is identified where the bigraph A is smaller or simpler in structure compared to B, all instances of B can be substituted with A to result in a more refined system and reduce redundancy.
- **Optimization of Models:** Bigraphs which represent components that are more expensive, resource intensive or more difficult to obtain in a real-world context can be substituted with a cheaper, more efficient or more available bigraph component if they are bisimilar, without affecting the overall model behavior.
- **System Verification:** An modified bigraph agent A' representating a system state can be checked against a previous version of itself A to verify that $A \sim A'$, to ensure that no unexpected alteration in behavior is observed from the change. This is useful for applications such as ensuring backward compatibility between programs and safe refactoring of code.
- **Fault Tolerance:** Bigraph agents that are identified as bisimilar could be used as backups for one another, if one becomes unavailable or encounters a fault. This would be useful in contexts such as networking or distributed systems where servers running equivalent protocols can safely replace each other to avoid downtime.

As discussed previously, bisimulations between agents cannot be identified with only a raw transition system due to wider environmental influences potentially disrupting their congruence, and can only be guaranteed through comparing minimal CTSs which exhaustively account for all potential wider contexts. We have also established that building minimal CTSs requires the finding, through some algorithm, of maximal shared areas between agents and reaction rule reduxes. Hence, a maximum common bigraph algorithm would provide the foundational missing step that would then allow for the supporting of minimal CTS building and subsequently the introduction of bisimulation verification in a BRS.

Now that the key motivations for devising such an algorithm have been established, we move forward to discuss the MCIS problem and the observed logical parallels to finding a hypothetical maximum common bigraph, as well as reviewing existing MCIS tools and their potential for being adapted to solve MCB.

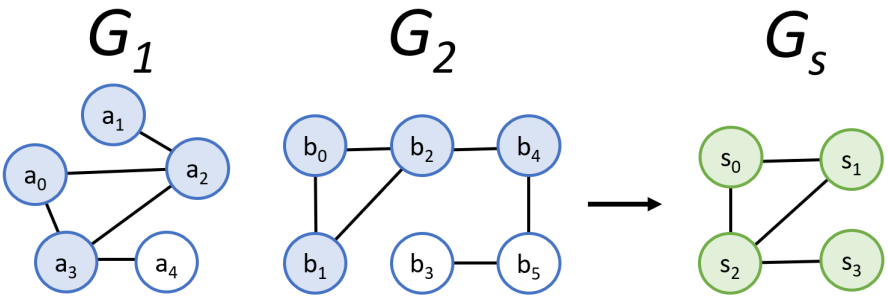


Figure 6.3: (a) An instance of the maximum common subgraph problem, with G_S as a solution subgraph between G_1 and G_2. The solution mapping takes the form $\{(s_0, a_3, b_0), (s_1, a_0, b_1), (s_2, a_2, b_2), (s_3, a_1, b_4)\}$.

6.2 Maximum Common Induced Subgraph

The *maximum common subgraph* (MCS) problem is an NP-complete optimization problem, which seeks to find a graph structure G_S which exists as an isomorphism to a subgraph inside two given input graphs G_1 and G_2, where there does not exist any larger (by some metric) subgraph which meets the same criteria [85]. There exist two main forms of MCS: maximum common *induced* subgraph (MCIS), which is concerned with maximizing the number of vertices that can be mapped between G_1 and G_2, and the *maximum common edge subgraph* problem (MCES) which instead uses the number of edges of G_S as the measure of size to maximize. Going forward, we are primarily concerned with the induced variant of MCS, as this more accurately reflects the criteria which we want to later define for a maximum common bigraph (Section 6.3). We formally define this as follows.

Definition 6.2.1 (Maximum Common Induced Subgraph)**.** Given two input graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a common induced subgraph $G_S = (V_S, E_S)$ is one such that there exists a pair of injective mappings $f_1 : V_S \to V_1$ and $f_2 : V_S \to V_2$, where vertices and edges and mapped such that $(u, v) \in E_S \iff (f_1(u), f_1(v)) \in E_1$ and $(u, v) \in E_S \iff (f_2(u), f_2(v)) \in E_2$.

G_S is *maximum* when there exists no alternate solution $G'_S = (V'_S, E'_S)$ which satisfies the above conditions and also satisfies $|V'_S| > |V_S|$.

MCIS can be seen as a generalization of subgraph isomorphism, where instead of finding only solutions that are complete and satisfiable, it instead searches for the set of solutions which come as close as possible to a full matching, i.e. minimizing nodes with no assignment [86]. A solution to MCIS can be represented by a set of triples (v_1, v_2, v_s) representing the bijective mapping from a subset of vertices $v_1 \in F_1 \subseteq V_1$ to a subset of vertices $v_2 \in F_2 \subseteq V_2$, where G_S is isomorphic to F_1 and F_2 and v_s indicates the mapping from each to a vertex in V_S. A solution in this format can also be treated as a solution to both $\texttt{SIP}(G_S, G_1)$ and

$\texttt{SIP}(G_S, G_2)$, highlighting another relation between SIP and MCIS. As G_S is a maximum by definition, then for any other possible graph G'_S where $|V'_S| > |V_S|$, G'_S no valid match will be found for either (or both) of SIP(G'_S, G_1) or SIP(G'_S, G_2), otherwise G'_S would be the more optimal MCIS solution.

Despite their similarities, MCIS is a much more difficult problem to solve compared to SIP, due to no longer being able to perform powerful reasoning on vertex degrees and path distances when moving away from the domain of exact match searching. However, the fixed forest structure and introduction of labels to bigraphs may allow for larger instances to be solved using MCIS, as the domains of vertices will be more restricted before search, potentially cutting down the overall search space. We provide a brief analysis of MCIS algorithms and their potential for adaptation for bigraphs.

6.2.1 Existing MCIS Algorithms

Max Clique Reduction

Multiple methods of solving MCIS rely upon a reduction to the *maximum clique* problem [87], which seeks to find the largest subset of vertices $S \subseteq G$ in a graph such that all pairs of vertices in S are connected via an edge — this is achieved by building an *association graph* A from the input pair of graphs G_1 and G_2. The vertex set V_A is made up of the cross product $V_{G_1} \times V_{G_2}$, where each vertex represents the shared mappings $u \in G_1 \to s$ and $v \in G_2 \to s$ to the solution graph vertex $s \in G_S$ in a MCIS solution. Edges between association graph vertices represent *compatible* assignments - that is, the pair of $G \times H$ assignments can appear in a solution together because the edges between the pairs of vertices do not conflict with one another. From there, it can be derived that the maximum clique of compatible assignments in the association graph corresponds to the largest common structure between G_1 and G_2.

McCreesh et al. propose such an adaption of an optimized max clique algorithm MCSa1 [85], which utilizes greedy coloring of vertices to determine the upper bound of any solution in order to cut down on redundant computation. This adaptation also uses a degree-based order of assignments, and similarly to the Glasgow Subgraph Solver, employs bit-parallel data structures to ensure optimal performance. This could hence potentially provide an ideal foundation for re-engineering this approach to support an encoded bigraph input. However, unlike a CSP/COP-based format where the extra complexities of bigraphs can be simply abstracted away from the base model and handled through additional constraint rules, a max clique-based algorithm would have to be substantially overhauled from the ground up in order to deal with factors that cannot be accounted for when considering only pairs of assignments in a vacuum, such as ensuring that a solution respects the compositional rules of bigraphs, or the handling of sites and regions. Hence, a constraints-based adaptation which

resembles our encoding for bigraph matching and retains the core underlying MCIS logic would be more ideal as an approach for devising an algorithm for MCB.

MCIS as a Constraint Optimization Problem

MCIS in a constraints-based context can be considered as the optimization problem variant of the SIP satisfaction problem. This is reflected in the building of the COP model — the variables, values and constraints all remain the same as SIP (Section 2.4.3), with the key difference being that the mapping of each vertex from G_1 to G_2 can now optionally take a *null value* $\emptyset$ to represent no assignment, and the solver aims to find a set of assignments such that the occurrences of $\emptyset$ in the solution are minimized. Because of this congruency between the two problems, this suggests that there can also feasibly exist an optimization-based generalization of our bigraph matching encoding (Chapter 3) that must adhere to similar conditions, and which aims to find the largest possible support mapping between two bigraphs G_1 and G_2 such that the resultant solution bigraph G_S must occur as a pattern in both G_1 and G_2.

Employing a general constraint toolkit such as Choco [64] to build an MCIS model would be a feasible approach to engineering and evaluating a MCB algorithm, as this could be used as the foundation for building the necessary additional constraints to handle bigraph logic in a similar manner to that of our matching algorithm. However, a simple constraint model implementation would lack the performance benefits of a state-of-the-art subgraph solver with domain-specific optimizations and efficient data structures, such as those provided by GSS or MCIS-adapted MCSa1. Ideally, we should again be able to make use of adapting of an existing optimized subgraph solver, provided that it is built using a constraint programming-like paradigm and thus suitably extensible for the requirements of a bigraph-based model. We thus look toward the *McSplit* MCIS algorithm.

The McSplit Algorithm

McSplit is a state-of-the-art branch and bound MCIS algorithm first introduced by Trimble [88, 14], which operates in a very similar fashion to a constraint solver, where vertices from V_{G_1} are assigned to those in V_{G_2}, and the search tree backtracks when no more assignments are possible. Where McSplit differs from the typical constraint framework is in its *partitioning* of node variables into sets of *label classes* (not to be confused with vertex labels in labelled graphs), where all unassigned $(u \in V_{G_1}, v \in V_{G_2})$ pairs that belong to the same label class, i.e. $lc(u) = lc(v)$, can be assigned to one another from a given search state. McSplit utilizes sets of label classes to keep track of possible assignments instead of maintaining separate $D \subseteq V_{G_2}$ domains for each $v \in V_{G_1}$.

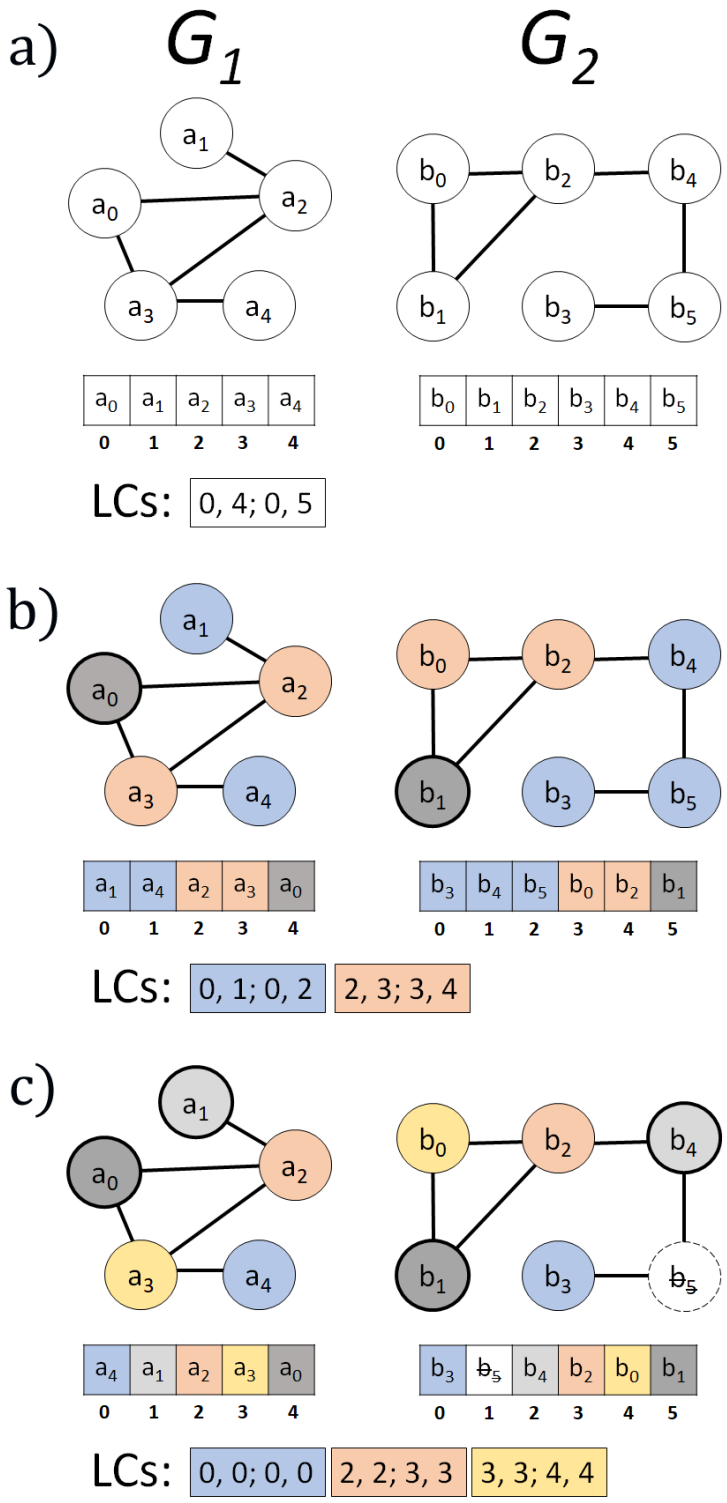


Figure 6.4: The McSplit algorithm when applied to the MCIS instance in Figure 6.3, (a) in its initial state, (b) after assigning a_0 to b_1, and (c) after assigning a_1 to b_4. The V_1 and V_2 arrays are positioned below their respective graphs, and label classes are color coded. Vertex b_5 is eliminated from selection when there is no G_1 vertex which has the same label class/set of neighbors - that is, non-adjacent to the first assigned vertex and adjacent to the second assignment.

Each unique label class can be identified using a bitset of length n, where n is the number of assignments made at that point during search. Each kth bit of the set indicates whether each unassigned vertex belonging to that label class was adjacent to the vertex belonging to the kth assignment mapping $(u_k \in V_{G_1} \rightarrow v_k \in V_{G_2})$. From this, it can be deduced that all vertices belonging to the same label class will share the same neighborhoods in relation to all nodes currently mapped to the common subgraph V_{G_M}, which is where the compatibility of label class members can be inferred. It thus follows that with each subsequent assignment, all label classes are then partitioned further (into up to two new subsets) based upon whether their member vertices were adjacent to the vertex which was assigned at that step.

Maintaining the variables and domains of the MCIS instance as label classes in this way can be achieved using only three array structures, representing a permutation of V_1 vertex ids (the V_1 array), a permutation of G_2 vertex ids (the V_2 array), and a set of records that each contain four pointer values (the LC array). All vertices in the V_1 and the V_2 arrays which share a label class are positioned next to one another, and each record $lc \in LC, lc = \{Start(V_1), End(V_1), Start(V_2), End(V_2)\}$ indicates the bounds of each grouped label class in the V_1 and V_2 arrays. At each assignment step, it takes only $O(|V_1| + |V_2|)$ time to refine these data structures using this compact setup through the shifting of V_1 and V_2 elements — the label class bitsets themselves do not need to be stored at any point. Upon backtracking, only the LC array needs to be reverted to an earlier state, as the element position of the V_1 and V_2 arrays at a given step in the search tree will remain correct for earlier states as they are only reordered within their current group at each assignment step and subsequent partition. A visual representation of the McSplit assignment process is provided in Figure 6.4.

Additionally, McSplit further ensures optimal performance by calculating an *upper-bound* at each search state as follows:

$$bound = |M| + \sum_{l \in LC} min(|(u \in G_1 \wedge LC(u) = l)|, |(v \in G_2 \wedge LC(v) = l)|)$$

where LC is the set of label classes and $|M|$ is the number of currently assigned vertices. This bound calculates the maximum hypothetical number of mappings that can be achieved at that point based on the sizes of each label class group in both G_1 and G_2. If at any point during search the bound value is equal to or less then a previously known maximum, the solver will prune the full branch without searching further as it logically cannot contain a more optimal solution (when enumerating all solutions, this is modified to strictly less than). By relying upon its time and space optimized data structures, in addition to its bound constraint, McSplit was found to outperform McSa1 by over an order of magnitude [14]. There also exists an even further optimized implementation of McSplit introduced by Calabrese et al. which combines McSplit with efficient heuristic strategies which are based upon the

PageRank algorithm [89].

Labelled, Directed and Connected McSplit

Trimble [88] also introduces extensions to McSplit which can support vertex labels and directed edges, which is a promising feature that would enable us to potentially add also support for encodings of bigraphs.

Support for vertex labels can be accomplished through preliminarily partitioning all vertices which share the same vertex label into separate label class before search, ensuring that only vertices with the same vertex label can ever be assigned to each another. Support for directed edges is more complex, and requires the introduction of two new 2D arrays A_{V_1} and A_{V_2}, representing the adjacency matrices of V_1 and V_2 respectively, where $A_{V_k}(u, v)$ is 0 if there no adjacency, 1 if there is an outgoing adjacency and 2 if there is an incoming adjacency between vertices u and v. At each assignment step where vertex u is assigned, label classes are now partitioned into up to three subsets rather than two, based on the value of $A_{V_k}(u, v)$ for each remaining unassigned vertex $v \in V_k$*.

Another interesting feature of McSplit is that it can also be adapted to support the maximum common *connected* induced subgraph (MCCIS) problem, which introduces the extra constraint that all vertex pairs in a solution must be transitively adjacent to one another (i.e. no disjoint subgraphs). This is performed through only permitting the assignment of a vertex if it is adjacent to a vertex which was previously assigned earlier in the search tree, after first assignment has been made. This is accomplished in practice by storing an additional boolean flag in each label class, which is equal to **false** if and only if its bitset representation is "all zeroes" (disjoint from all current assignments), and these label classes are ignored during variable selection beyond the initial assignment. This extension is of particular interest, as an encoded representation of MCB must still respect the compositional logic of bigraphs; the structure of a common bigraph G_M that adheres to $B_1 = C_1 \circ G_M \circ D_1, B_2 = C_2 \circ G_M \circ D_2$ must be either connected, or its disjoint parts belonging to a valid tensor product of one another, upon rebuilding the initial input bigraphs. Hence, a bigraph adaptation for McSplit will likely require a hybrid algorithm between MCIS and MCCIS solving, similarly to how GSS required a hybrid of induced and non-induced subgraph matching to model the same property of bigraphs for bigraph matching (Chapter 3).

*This method can also be used to support labelled edges and graphs where edges can point in both directions, by adding further integer values that represent each possible edge type.

6.3 Proposing Maximum Common Bigraph

We now formally introduce our notion of the maximum common bigraph (MCB) problem as follows.

Definition 6.3.1 (Maximum Common Bigraph)**.** Given two input solid bigraphs G_1 and G_2, an instance of a maximum common bigraph $\texttt{MCB}(G_1, G_2) = \{G_M, M\}$ of G_1 and G_2 is a tuple, containing the solid bigraph G_M which satisfies

$$G_1 = C_1 \circ (id \otimes G_M) \circ D_1,\ G_2 = C_2 \circ (id \otimes G_M) \circ D_2$$

such that there does not exist some other bigraph G'_M which satisfies

$$G_1 = C_3 \circ (id \otimes G'_M) \circ D_3,\ G_2 = C_4 \circ (id \otimes G'_M) \circ D_4,$$
$$(|G'_M| > |G_M|)\ \vee\ (G'_M = C_5 \circ G_M \circ D_5)$$

for some arbitrary context and parameter bigraphs C_{1-5} and D_{1-5} respectively.

$M = \{(m_1, m_2, ..., m_n)\}$ is a set of triples $m_k = \{(g_k, a_k, b_k)\}$, representing the embedded mapping between the entities, ports and closed links of $g \in G$ to those in $a \in G_1$ and $b \in G_2$ respectively. We wish to enumerate all such instances and their corresponding mappings.

It can be observed that from this definition, a candidate solution must meet two separate requirements to be considered a maximum. Firstly, there cannot exist any other solution bigraph G' with a greater support size as this would mean it cannot reasonably be considered the largest shared region between G_1 and G_2. Secondly, each solution must also meet the criterion of being maximal in the compositional sense—that is, we must close or merge any interface components if it is possible to do so while retaining G_M as a sub-component of both G_1 and G_2. This ensures that the solution bigraph G_M, using the support mapping supplied by M can be considered an RPO between G_1 and G_2. This extra step is necessary for two reasons—firstly, without restricting the interfaces in this manner, there would be a theoretically infinite number of additional variants for each solution bigraph containing extra arbitrary regions/sites/links. Secondly, if any further valid composition onto G_M is possible, this violates the conceptual notion of G_M being maximal—we wish to "saturate" each discovered solution such that any non-identity composition onto G at all would no longer cause it to be a valid solution. Figure 6.5 demonstrates an instance of the MCB problem, and the corresponding decompositions of G_1 and G_2 to retrieve the common bigraph.

Similarly to how MCIS can be considered an optimization problem variant of the SIP decision problem, we propose MCB as the optimization variant of bigraph matching. We also propose that in addition to our SIP encoding for bigraph matching, MCB can be solved

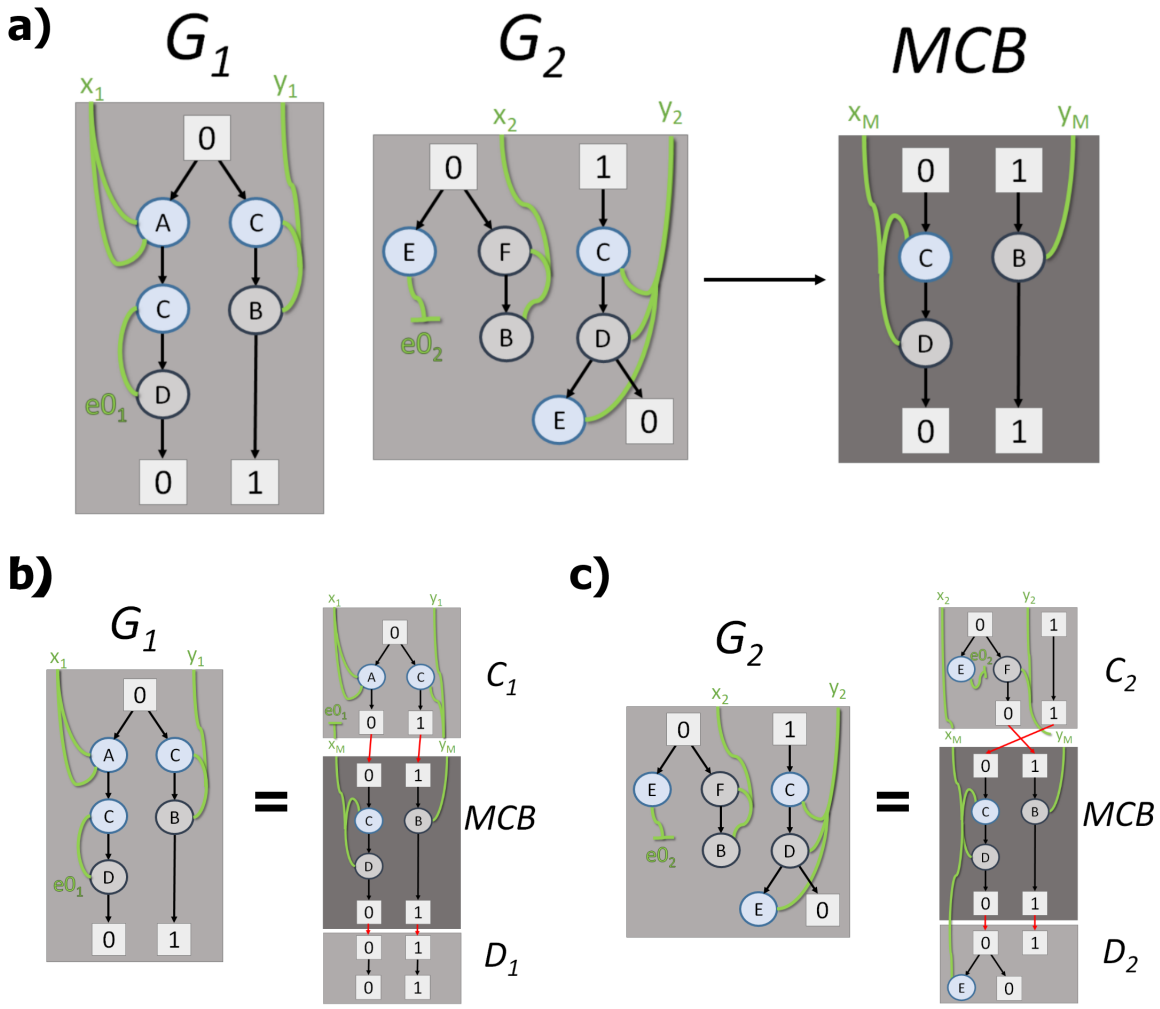


Figure 6.5: (a) An instance of the maximum common bigraph problem, with G_1 and G_2 as the input bigraphs and MCB as the solution. (b) A decomposition of G_1 to show that MCB exists as a component in the bigraph. (c) A similar decomposition of G_2 to show MCB also exists. The ports of C and D must be joined in MCB as this still allows for a valid solution.

by treating a problem instance as MCIS with additional constraints and pre/post processing to handle the complexities introduced by hyperedges, interfaces and RPO reconstruction to ensure that the compositional criteria still hold in addition to the shared graph structure.

6.3.1 Properties

Through this definition, we can intuitively infer some properties of MCB which must always hold for any instance, which can be used to preliminary verify the correctness of a prototype MCB solver.

- **Identity**: $G \in \texttt{MCB}(G, G)$ for any bigraph G. The maximum common bigraph between two equivalent bigraphs will also be an equivalence. There can also exist additional solution mappings when intra-symmetries exist within G, similar to bigraph matching.
- **Inverse Rule**: $\{G_M, M\} \in \texttt{MCB}(G_1, G_2) \longleftrightarrow \{G_M, M^{-1}\} \in \texttt{MCB}(G_2, G_1)$ for any bigraph pair G_1 and G_2. Swapping the order of input graphs should still produce the same set of solutions, but with all mappings between G_1 and G_2 inverted.
- **Succession**: The solution $\{G_M, M\} \in \texttt{MCB}(G_3,\ \texttt{MCB}(G_1, G_2))$ represents the largest shared area between three bigraphs G_{1-3}, regardless of order of operations (transitive).
- **Matching**: $(S_{big} \neq \emptyset) \in \texttt{MATCH}(P, T) \longrightarrow \{P, S_{big}\} \in \texttt{MCB}(P, T)$–if a full match of pattern bigraph P exists in the target T, it follows that P will also be a maximum common bigraph between the two with the same embedded mapping.

We now go on to describe how McSplit can be adapted to support a modified variant of our encoded and flattened bigraph structure from Chapter 3. Once again, we assume that the input bigraph pairs are both solid and that instances are non-trivial.

6.4 Encoding McSplit to Solve MCB

Firstly, we consider only the encoding of place graphs in isolation, and the changes we make to the McSplit array structures which enable us to enforce valid solutions which adhere to bigraphical composition rules. We then introduce our link graph flattening function, and the further adaptations made to the refinement process and reward function to support these. For each of places and links, we also describe their respective post-search processes that ensure that all solution bigraphs are RPOs in relation to the input pair. A high level view of this process is demonstrated visually in Figure 6.6.

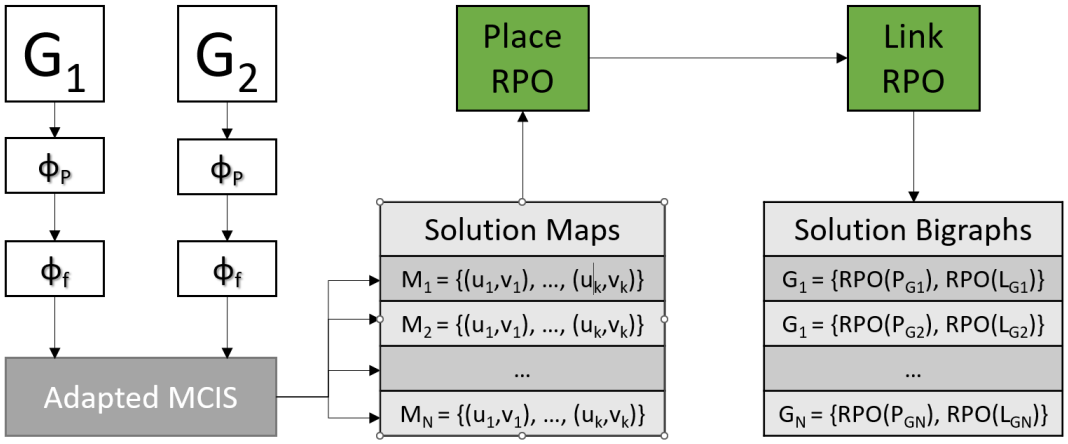


Figure 6.6: The MCB solving process, where the largest common areas between two encoded bigraphs are found using a MCIS algorithm before their interfaces are refined to produce a set of minimally bound solution bigraphs.

6.4.1 Place Graph Encoding

We make use of the *pattern* place graph encoding function from Section 3.2.1 to encode both MCB place graphs, where the sites and regions of G_1 and G_2 are discarded, alongside all parent relations involving them. We are not immediately concerned with the site and region placements of G_1 and G_2 when performing the underlying MCIS process, because a site/region placement (or lack of thereof) cannot ever cause an otherwise valid set of entity mappings for MCB to be invalidated; we can simply assign a site as a child to all entities $v \in V_{G_M}$ and a region as a parent to every top-level entity in the building of the solution bigraph(s), in order to bypass any of the degree constraints which were identified in the full matching case, guaranteeing that the result will occur in G_1 and G_2. Hence, we first assume as a baseline that this relaxed structure — where all entities are adjacent to both the inner and outer interface — will be the form of the returned solution bigraph(s) upon completing the main MCIS search loop, and then later proceed to *refine* the interfaces using a post-search function by closing and merging all regions and sites where possible, while ensuring G_M remains a sub-bigraph of both G_1 and G_2 (Figure 6.6, Section 6.4.3).

After applying the function, the encoded versions of $G_1 = (V_{G_1}, ctrl_{G_1}, prnt_{G_1}) : i \to j$ and $G_2 = (V_{G_2}, ctrl_{G_2}, prnt_{G_2}) : k \to l$ take the form of the following graph pair:

$$\phi_P(G_1) = \{V_{G_1}, (u, v) \in prnt_{G_1}^{-1} \mid v \neq i, u \neq j\}$$

$$\phi_P(G_2) = \{V_{G_2}, (u, v) \in prnt_{G_2}^{-1} \mid v \neq k, u \neq l\}$$

Ensuring Valid Compositions

The next step to implementing MCB for the place graph is modifying McSplit to ensure that a solution bigraph G_M will always respect the compositional property of bigraphs, and thus exist as a component in G_1 and G_2 in the forms $G_1 = C_1 \circ G_M \circ D_1$ and $G_2 = C_2 \circ G_M \circ D_2$ respectively. To ensure this, all pairs of assigned entities must either be directly adjacent to one another (i.e. producing a connected subgraph), or fully transitively disjoint where neither are descendants of the other (i.e. producing a tensor product). We begin by considering McSplit for MCCIS, which enforces connected solutions through only selecting vertices from label classes if that class has at least one adjacency to the current solution subgraph. However, we want to relax this restriction in a way which still allows disjoint vertices to be selected, as long as they cannot be transitively reached by or from any current assigned vertex. In a CSP format, this would be added to the list of constraints as follows:

$$\{\forall u, v \in G_1 \mid \{\forall v' \in prnt^{-1}_{G_1}(v) \mid \texttt{match}(v') = \emptyset\} \wedge (u, v) \in prnt^{+}_{G_1} \wedge \texttt{match}(u) \neq \emptyset\}$$

$$\rightarrow \texttt{match}(v) = \emptyset$$

The same constraint applies to G_2, with the difference that all calls to the `match` function are replaced with $\texttt{match}^{-1}$ to ensure symmetry.

Similarly to our matching implementation for bigraphs with sharing, we initially construct two descendant maps for all pairs of vertices within each of G_1 and G_2 prior to beginning search — which we define as τ_1 and τ_2 respectively — where $\tau_{\{1,2\}}(u, v)$ is true if and only if entities u and v are transitively adjacent (Section 4.2.3). This allows the solver to determine whether a non-adjacent (belongs to an "all-zeroes" label class) entity should still be allowed to be selected for assignment.

Connected McSplit can be adjusted to accommodate this at the variable selection step by replacing the boolean flag within each label class with a bitset variable which we define as $A(l)$, which indicates entity selection visibility for its member entities. $A(l)$ simply contains a 1 (true) value if the label class l is adjacent to the current solution bigraph as it can be inferred that all member vertices will also be adjacent. If the label class is disjoint from the solution bigraph however, then the bitset will begin with a zero followed by $(|\{u \in G_1 \wedge LC(u) = l\}| + |\{v \in G_2 \wedge LC(v) = l\})$ bits, where $A(l)[k+1]$ indicates whether the kth entity in the class is a current valid selection. During variable selection, a vertex v is prohibited from being assigned to if $A(LC(v))[0] = 0$ and $A(LC(v))[v+1] = 0$, indicating that it is currently invisible to the propagator. Figure 6.7 provides an example instance of this adapted approach for a given state, where each LC is shown alongside their $A(l)$ values. For simplicity, this conditional visibility constraint does not apply to the ports of an entity for this approach, and thus an entity's port vertices can only be assigned once the entity itself

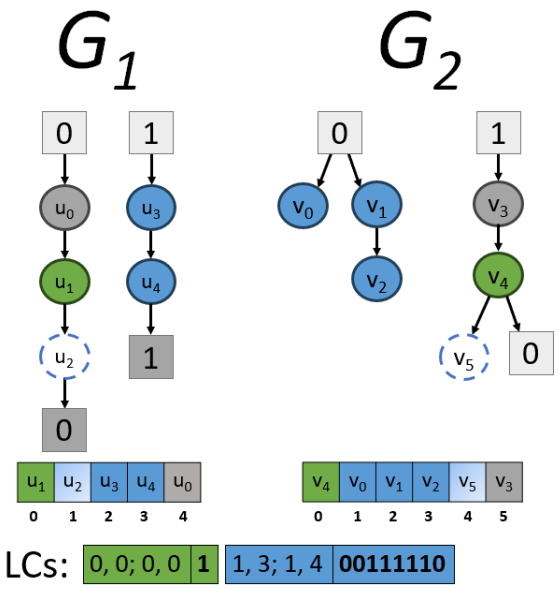


Figure 6.7: The adapted McSplit algorithm for a pair of place graphs after the mapping $u_0 \rightarrow v_3$ is made. Entities u_2 and v_5 are restricted from selection at the next assignment step, as they are transitively but not directly adjacent to u_0 and v_3 respectively. The blue label class vertices, which would be unselectable in default connected McSplit, are allowed to be selected here as they can exist as a tensor product of u_1 and v_4 respectively.

has been assigned.

We also define $A'(l)$ as the value of the label class l's bitset at the previous assignment step. Algorithm 5.1 demonstrates the label class bitset refinement process upon making the assignment $(u \in V_{G_1} \rightarrow v \in V_{G_2})$, which is performed prior to the partitioning process. It can be observed that the bitset structure can be refined at an upper bound of $O(n + m)$ time similarly to the main label class refinement process, by checking each unassigned entities' relation to the entity in each bigraph assigned at that step. The partitioning process itself is also modified such that each entity's corresponding label class bit value is repositioned alongside them when appropriate, to maintain congruency between states.

Our encoding function, in addition to this additional refinement process to enforce compositional integrity, is sufficient to find all valid assignments between the maximum common bigraph(s) between the entities of a pair of place graphs. The regions and sites of the graphs are later further restricted as a post-process to guarantee compositional saturation for each solution, as described in Section 6.4.3.

6.4.2 Link Graph Encoding

We now consider the addition of the link graph into our encoding. As with our matching algorithm, we proceed by constructing a link flattening function.

Algorithm 5 Refine LC Visibility (int u, int v)

for all $l \in LC$ **do**
  $\mathbf{S_1} \leftarrow \{w \in V_{G_1} \wedge lc(w) = l\}$
  $\mathbf{S_2} \leftarrow \{w \in V_{G_2} \wedge lc(w) = l\}$
  if $A'(l)[0] = 1$ **or** $(|S_1| > 0$ **and** $A_{V_1}[S_1[0]][u] > 0)$ **or** $(|S_2| > 0$ **and** $A_{V_2}[S_2[0]][v] > 0)$ **then**
    $A(l) \leftarrow 1$
    continue
  end if
  $A(l)[0] \leftarrow 0$
  if $\ell(l) \neq$ **link then**
    for all $u' \in S_1$ **do**
      if $A'(l)[u'+1] = 1$ **and** $R[u][u'] = 1$ **then**
        $A(l)[u'+1] \leftarrow 0$
      end if
    end for
    for all $v' \in S_1$ **do**
      if $A'(l)[v'+1] = 1$ **and** $R[v][v'] = 1$ **then**
        $A(l)[v'+1] \leftarrow 0$
      end if
    end for
  end if
end for

Given a bigraph $B^{\mathsf{L}} : (V_B, E_B, ctrl_B, link_B) : X \to Y$, and the encoding of its place graph $D\ \phi_{\{P,T\}}(D^{\mathsf{P}}) : (V_D, E_D)$, where (where $V_B = V_D$), we define the flattening function as follows:

$$\phi_f : \phi_{\{P,T\}}(D^{\mathsf{P}}) \times B^{\mathsf{L}} \mapsto (V, E)$$

The vertices of the resultant flattened graph can be described as:

$$V = V_D \uplus P_B \uplus \widehat{E_B}$$

$$\widehat{E_B} = \{e \in E_B \mid link_B(p) = e, p \notin X\}$$

where $\widehat{E_B}$ is the set of closed links in B^{L}, P_B are the ports of B^{L} (defined in Definition X), 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.

We describe the resultant edge set as follows:

$$E = E_D \uplus \{(v, p) \mid p = (v, i) \in P_B\} \uplus \{(p, e) \mid e \in \widehat{E_B}, link_B(p) = e\}$$

This is a less constrained variant of our flattening function from Section 3.3.3, where we no

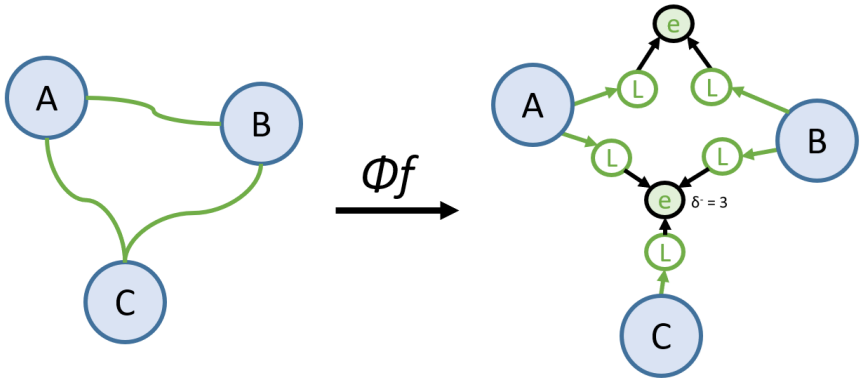


Figure 6.8: An example of link flattening for the MCB algorithm — hyperedge connections are not considered for the main search loop, and are instead re-added and merged upon finding a solution.

longer build cliques between ports which share a hyperedge. This allows the flattened graph to represent a version of the input link graph pair where all ports have been freed up and each have a lone connection to the bigraph interface, which is the link graph's equivalent to our process of freeing all sites and regions in the place graph. This produces a link graph where there are no constraints between interface components, representing the minimal possible structure that can be a substructure of G_1 and G_2 when a valid set of mappings are found. Similarly to the place graph, then later refine and merge all possible sets of links (retaining G_M as a sub-bigraph of G_1 and G_2) as part of a post-search function to ensure the composition is fully maximized. A visual application of the modified flattening function is provided in Figure 6.8.

We retain the degree constraint from our matching encoding on closure nodes to ensure that any common hyperedge will have isomorphic adjacency sets in G_1 and G_2. In McSplit, this can be preliminarily enforced using the existing label class structures by further partitioning the $\ell =$ **closure** label class's vertices by in-degree prior to search. To preserve the structure of the solution bigraph's hyperedges, we also constrain closure nodes such that they are only available for variable selection once *all* of their parent port nodes have been assigned. This prevents the case where a solution may contain a "closed" link edge that is still connected to the interface, which we seek to prohibit. This can be reflected by enforcing the additional constraint:

$$\{\forall e \in E_{G_1},\ p = (v, i) \in P_{G_1} \mid (p, e) \in link_{G_1} \wedge \texttt{match}(p) = \emptyset\} \rightarrow \texttt{match}(p) = \emptyset$$

This can be implemented by checking neighboring closures upon port assignment and toggling them to visible if all their parent ports have been assigned, using the existing bitarray structure in the label class to distinguish between visible and invisible closure vertices. An

additional adjustment to achieve this is that closure label class bitarrays will always have their first bit set to zero, as adjacency to the currently selected substructure alone no longer guarantees visibility for selection. Enforcing this on G_1, in combination with the degree constraint to ensure like-like matching, is sufficient to also ensure this holds for closures in G_2 as sharing a label class guarantees that their respective neighborhoods will match.

Modifying the Reward Function

This method of representing ports as flattened vertices introduces a discrepancy between the size of a candidate solution's encoded form and its support size. An example of where this arises is shown in Figure 6.9, which shows a MCB instance in the form of a pair of bigraphs, and its corresponding MCIS instance after encoding and flattening G_1 and G_2. The MCB instance has two solutions of equal size 1, that is, $(A \rightarrow A)$ and $(B \rightarrow B)$. However, in the encoded MCIS instance, only the $(B \rightarrow B)$ solution will be returned, because its ports will also contribute toward the size of the mapping, meaning that the MCIS algorithm will consider the mapping of $(B \rightarrow B)$ and its ports a solution of size 3, compared to mapping $(A \rightarrow A)$ and its lone port to produce a solution of size 2. Hence, we wish to modify McSplit to ignore the assignment of port vertices, specifically when enumerating the current score of a candidate solution. However, an occurrence of an assigned entity with an unassigned port can never appear in a valid solution, as all components of the entity must be matched alongside the entity itself (as matching labels always have the same arity value, all ports should always be available for matching). Taking this into account, we propose a new scoring function to determine the optimality of a solution as follows:

$$score = \begin{cases} -1 & \text{if } \exists\{\ p = (v,i) \in P_{G_1} \mid \texttt{match}(p) = \emptyset\ \wedge \texttt{match}(v) \neq \emptyset\} \\ |G_M| & \text{otherwise} \end{cases}$$

Where P_{G_1} is the set of flattened ports of bigraph G_1, and $|G_M|$ is the support size of the common bigraph. The symmetry of assigned entities and arities means only one of the input bigraphs needs to be checked to verify validity. In addition, we also modify the bound function to ignore all label classes containing port nodes as follows, since they do not contribute to the score of a solution:

$$bound = |G_M| + \sum_{l \in LC} min(|(u \in \{V_{G_1} \uplus E_{G_1}\} \wedge LC(u') = l\ \wedge\ u \neq (p,i) \in P_{G_1})|,$$

$$|(v \in \{V_{G_2} \uplus E_{G_2}\} \wedge LC(v') = l\ \wedge\ v \neq (q,j) \in P_{G_2})|)$$

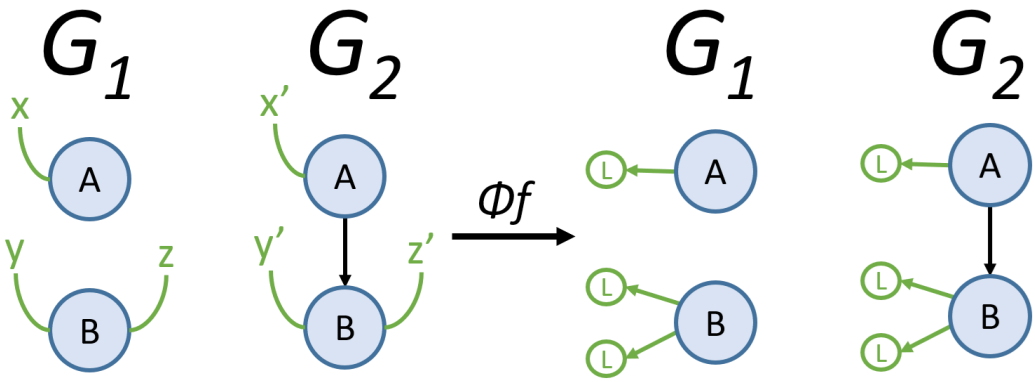


Figure 6.9: An example of a flattened MCB instance where the returned MCIS solution set does not match the expected solution set of MCB due to counting port vertices, without first modifying the score function.

Where u' and v' are the encoded vertices representing support elements u and v. As vertex label types are partitioned into separate label classes during initialization, it is trivial to determine whether a label class l is a "port" class by checking the vertex label value of any member element, and skip l during summation if the label function of the vertex $\ell(v) =$ **link**. While restricting the upper bound is not a necessary step to ensure correctness of solutions, it is in our interest to do so whenever possible to ensure optimal performance and minimize time spent performing redundant search tree traversal, as long as no valid solution is ever incorrectly filtered as a result.

6.4.3 RPO Construction

After retrieving the set of each largest partial mapping $M = \{(u_1, v_1), (u_2, v_2), ..., (u_k, v_k)\}$ produced by the modified McSplit algorithm, where $u_i \in G_1$ and $v_i \in G_2$, this is sufficient to then build each solution bigraph G_M. These solutions exist as a pattern in both input bigraphs G_1 and G_2, and each solution vertex $w \in G_M$ is common to an assigned vertex $(u \in G_1 \rightarrow v \in G_2)$. In order for each G_M to be maximal in terms of compositional saturation however, we must ensure that the triple relation (G_M, G_1, G_2) is a minimal bound. We can achieve this for each solution by first assuming each top-level entity of G_M has a parent region and each entity also has a child site. For the link graph, we also assume at first that all ports unconnected to a closure are fully open and unconnected. This reflects the least constrained possible state of G_M, which ensures it will always match to both G_1 and G_2 to begin with. We then refine each solution by closing all unused interface components and merging all compatible pairs of interfaces in G_M where possible, until any further merging or closing of interface components will result in either $\texttt{match}(G_M, G_1) =$ **false** or $\texttt{match}(G_M, G_2) =$ **false**. This can be achieved for both the

place graph and the link graph as follows, by considering each $w \in G_M$ in accordance with its triple relation $(u \in G_1, v \in G_2, w \in G_M)$, as originally defined by Milner [1, 90].

Place Graph

Algorithm 6 Construct Place Graph RPO (G_1, G_2, M)

$V_{G_M} \leftarrow \{u \in G_1 \mid M(u) \neq \emptyset\}$
$ctrl_{G_M} \leftarrow \{ctrl_{G_1}(u) \mid M(u) \neq \emptyset\}$
$prnt_{G_M} \leftarrow \{(u, u') \in prnt_{G_1} \mid M(u) \neq \emptyset \wedge M(u') \neq \emptyset\} \uplus \{(s \in m, u) \mid (s' \in m, u) \notin prnt_{G_1}\} \uplus \{(u, r \in n) \mid prnt_{G_1}(u) = \emptyset\}$
$G_M \leftarrow \{V_{G_M}, ctrl_{G_M}, prnt_{G_M}\}$
for all $m = (u, v) \in M$ **do**
  if $prnt_{G_M}(u) = r \in n$ **and** $prnt_{G_1}(u) = prnt_{G_2}(v) = \emptyset$ **then**
    $prnt_{G_M}(u) \leftarrow \emptyset$
  end if
  if $|prnt^{-1}_{G_M}(u) \cap V_{G_M}| = |prnt^{-1}_{G_1}(u)| = |prnt^{-1}_{G_2}(v)|$ **then**
    if $prnt^{-1}_{G_1}(u) \cap V_{G_1} = prnt^{-1}_{G_1}(u)$ **and** $prnt^{-1}_{G_2}(v) \cap V_{G_2} = prnt^{-1}_{G_2}(v)$ **then**
      $prnt^{-1}_{G_M}(u) \leftarrow prnt^{-1}_{G_M}(u) \cap V_{G_M}$
    end if
  end if
  if $prnt_{G_M}(u) = r \in n$ **then**
    for all $\{m' = (u', v') \in M \mid m' > m\}$ **do**
      if $prnt_{G_M}(u') = r' \in n$ **then**
        if $prnt_{G_1}(u) = prnt_{G_1}(u')$ **and** $prnt_{G_2}(v) = prnt_{G_2}(v')$ **then**
          $prnt_{G_M}(u') \leftarrow prnt_{G_M}(u)$
        end if
      end if
    end for
  end if
end for
return G_M;

The following three operations are sufficient to ensure that the solution place graph G_M produces a minimal bound.

- **Close unused sites**: For each $(u \in V_{G_1}, v \in V_{G_2}, w \in V_{G_M})$ triple relation, if $|prnt^{-1}_{G_M}(w) \cap V_{G_M}| = |prnt^{-1}_{G_1}(u)| = |prnt^{-1}_{G_2}(v)|$ and neither u and v are adjacent to a site, remove the site of w.
- **Close unused regions**: For each $(u \in V_{G_1}, v \in V_{G_2}, w \in V_{G_M})$ triple relation, if w is top-level and $prnt(u) = prnt(v) = \emptyset$, remove the region of w.
- **Merge regions**: For each pair of triple relations (u_1, v_1, w_1) and (u_2, v_2, w_2), if both w_1 and w_2 have a parent region and $prnt(u_1) = prnt(v_1)$ and $prnt(u_2) = prnt(v_2)$, merge the regions of w_1 and w_2.

The process of closing a region in bigraph terms consists of removing the region $r \in n$ from the interface of G_M and decrementing n and all ordinals $\{r' \in n \mid r' > r\}$ by 1. Merging two regions $r_1, r_2 \in n, r_2 > r_1$ involves removing r_2 in a similar fashion, but also changes all parent relations $(v \in V_{G_M}, r_2)$ to (v, r_1). This is sufficient to guarantee a minimal bound in the place graph of G_M as no further non-identity composition is possible while retaining G_M as a common component of G_1 and G_2. We define this process algorithmically in Algorithm 6. As this iterates over all pairs of top-level entity mappings to determine maximality, this requires a maximum of $O(|M|^2)$ time for each solution — although in practice, only a fraction of nodes in a solution bigraph would be expected to be top-level.

Link Graph

Algorithm 7 Construct Link Graph RPO (G_1, G_2, M)

$V_{G_M} \leftarrow \{u \in G_1 \mid M(u) \neq \emptyset\}$
$ctrl_{G_M} \leftarrow \{ctrl_{G_1}(u) \mid M(u) \neq \emptyset\}$
$E_{G_M} \leftarrow \{e \in E_{G_1} \mid M(e) \neq \emptyset\}$
$link_{G_M} \leftarrow \{(p, e) \in link_{G_1} \mid e \in E_{G_1}\} \uplus \{(p, y \in Y) \mid (p, e \in E_{G_1}) \notin link_{G_1}\}$
$G_M \leftarrow \{V_{G_M}, ctrl_{G_M}, E_{G_M}, link_{G_M}\}$
for all $m = (p, q) \in M$ **do**
  for all $\{m' = (p', q') \in M \mid m' > m\}$ **do**
    if $link_{G_M}(p) = y \in Y$ **and** $link_{G_M}(p') = y' \in Y$ **then**
      if $link_{G_1}(p) = link_{G_2}(p')$ **and** $link_{G_1}(q) = link_{G_2}(q')$ **then**
        $link_{G_M}(p') \leftarrow link_{G_M}(p)$
      end if
    end if
  end for
end for
return G_M;

To ensure that the solution link graph G_M produces a minimal bound, we consider each pair of triple relations $(p \in G_1, q \in G_2, r \in G_M)$ and $(p' \in G_1, q' \in G_2, r' \in G_M)$ in the solution. We wish to merge each pair of ports (r, r') in G_M such that $link(r) = link(r')$ as long as $link(p) = link(p')$ and $link(q) = link(q')$ both hold, as we can infer that this will still allow for a match to both G_1 and G_2 while ensuring that the number of faces in G_M is minimized. We define this process algorithmically in Algorithm 7. Similarly to the place graph RPO function, this requires a maximum of $O(|M|^2)$ time to calculate each solution's RPO as we are comparing all pairs of ports.

We observe here that the link graph RPO function may produce non isomorphic bigraphs for a seemingly symmetrical pair of solutions, as the relations of ports to the original link graphs may differ. This indicates that we cannot perform symmetry breaking between open

ports when performing search, although symmetry breaking between closed hyperedge ports should remain unaffected as those are not modified as a part of this refinement.

6.5 Soundness and Completeness

We now provide a proof of soundness and completeness of our adaptation. Soundness is proven by demonstrating that any solution identified by the MCIS model corresponds to an instance of a solution which adheres to our definition of a maximum common bigraph. Conversely, completeness is proven by showing that when a solution exists in an instance of MCB, the adapted algorithm will also find a corresponding match in the MCIS encoding.

As it is already established in the literature of bigraphs how to construct an RPO for a pair of bigraphs and a valid set of entity/closure mappings between them [1], we do not repeat the proof of this here — instead, it is satisfactory to prove that the set of assignments returned by the adapted McSplit solver is sufficient for then passing to the RPO building functions to retrieve the maximum common structure. With that in mind, we represent each solution M returned by modified McSplit as an injective set of (u, v) pairs denoting the mapping from each vertex u in G_1 to v in G_2, ignoring the vertices of the later constructed common bigraph (handled by the RPO functions). Thus we wish to prove that for any maximum common bigraph embedding of size m:

$$S_{MCB} = \{(u_1, v_1), ..., (u_m, v_m)\}$$

The corresponding encoding will produce a solution:

$$M = \{(u'_1, v'_1), ..., (u'_m, v'_m)\}$$

with a bijective relation between (u_k, v_k) and (u'_k, v'_k) and vice versa. We propose some new observations related to bigraph composition in order to supplement our proofs.

Proposition 13. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $u \in V_B, v \in V_G$ and $u \in prnt^+_G(v)$ then $v \in V_B$.

This states that any descendant of an entity in V_B must also be in V_B. We prove this through the recursive application of the definition of bigraph composition (Definition 2.3.9), where for some $v' \in V_G$, if $prnt_G(v') \in V_B$ then $v' \in o \uplus V_B$, and we know v' is an entity and therefore $v' \in V_B$. Applying this to u means that $prnt^{-1}(u) \subseteq V_B$, and this can be

recursively applied to all children of u to prove that all grandchildren of u exist in V_B and so on, and this can be repeated recursively in order to reach all descendants of u. Hence, for any v such that $prnt^+_G(v) = u \in V_B$, we show that $v \in V_B$ as required.

Proposition 14. Given the composition of two place graphs

$$G : m \to n = (A : k \to n) \circ (B : m \to k)$$

if $u \in V_A, v \in V_G$ and $v \in prnt^+_G(u)$ then $v \in V_A$.

This states that any ancestor of an entity in V_A must also be in V_A. We again prove this through the recursive application of the definition of bigraph composition (Definition 2.3.9), where for some $v' \in V_A$, $prnt_G(v') = prnt_A(v')$ and hence $prnt_G(v') \in V_A$. Applying this to $prnt_G(u)$ means that the grandparent of u will be in V_A, and this can be recursively applied at each upper depth to prove that all ancestors of u must exist in V_A. Hence, for any v such that $prnt^+_G(u \in V_A) = v$, we show that $v \in V_A$ as required.

Proposition 15. Given the composition of three place graphs

$$G : m \to n = (A : k \to n) \circ (B : l \to k) \circ (C : m \to l)$$

if $u \in V_B, v \in V_B, w \in V_G$ and $w \in prnt^+_G(u), v \in prnt^+_G(w)$ then $w \in V_B$.

This states that any entity that exists between a pair of entities in V_B must also be in V_B. We prove this by applying Propositions 13 and 14 together as follows: by Proposition 13, as u is an ancestor of w and $u \in (B \circ C)$, then we know $w \in (B \circ C)$. By Proposition 14, as v is a descendant of w and $v \in (A \circ B)$, then we know $w \in (A \circ B)$. Taken together, we can conclude that w can only exist in B, and therefore $w \in V_B$. As a corollary, we can informally deduce that for any two entities $(u \in V_G, v \in V_G)$, if $u \in V_B$ and $v \in V_B$ and $prnt^+_G(u) = v$, then *all* entities between them must also exist in B.

6.5.1 Soundness

Given an instance of MCB (G_1, G_2) with a McSplit encoding $(\phi_f(\phi_P(G_1)), \phi_f(\phi_P(G_2)))$ for which there exists a solution of size m in the form of the injective mapping $M = \{(u'_1, v'_1), ..., (u'_m, v'_m)\}$, we wish to prove that this corresponds to a MCB solution $G_1 = C_1 \circ (G_M \otimes id) \circ D_1$, $G_2 = C_2 \circ (G_M \otimes id) \circ D_2$, in the form of an injective embedded mapping $S_{MCB} = \{(u_1, v_1), ..., (u_m, v_m)\}$ from a subset of support elements $u \in G_1$ to a support element $v \in G_2$, where $u'_k \to v'_k$ is the encoded form of support elements $u_k \to v_k$. We wish to prove two key attributes: firstly, that the bigraph G_M constructed from M is common to G_1 and G_2. Secondly, that there are no alternate set of assignments M' of size $> m$ which is also common to G_1 and G_2.

Commonality

We begin our proof of commonality by constructing the input bigraphs $G_K,\ K = \{1, 2\}$ from their encoded forms. We perform this for the place graphs $P = (V_K, ctrl_K, prnt_K)$ as follows:

$$
\begin{aligned}
&V_K = V_{\phi_P(G_K)} \\
&\{\forall v_i \in V_K \mid ctrl_K(v_i) = \ell(v_i')\} \\
&\{\forall (v_i, v_j) \in V_K \mid (v_i', v_j') \in E_{\phi_P(G_K)} \rightarrow prnt(v_j) = v_i\}
\end{aligned}
$$

The link graphs $L = (V_K, E_K, ctrl_K, link_K)$ can then be constructed as follows:

$$
\begin{aligned}
&V_K = \{g' \in \phi_f(G_K) \mid \ell(g') \notin \{\mathbf{link}, \mathbf{closure}\}\} \\
&P_K = \{g' \in \phi_f(G_K) \mid \ell(g') = \mathbf{link}\} \\
&E_K = \{g' \in \phi_f(G_K) \mid \ell(g') = \mathbf{closure}\} \\
&\{\forall g \in V_K \mid ctrl_K(g) = \ell(g')\} \\
&\{\forall (g_a', g_b') \in E_{\phi_f(G_K)} \mid \ell(g_a') = \mathbf{link}, \ell(g_b') = \mathbf{closure}\} \rightarrow link_G(g_a \in P_G) = g_b \in E_G
\end{aligned}
$$

We note that the structural information regarding the interfaces of G_1 and G_2 are lost as part of the initial encoding process. However, as previously discussed, these interface components cannot impact whether or not the set of assignments M produce a valid MCB since the interface of G_M is initially assumed to be in a fully open state, where the RPO functions handle the constraining of interface components later on — hence, they can only affect the structure of the common bigraph RPO, and therefore we do not require them as part of proving the soundness of M. Using M and either of our reconstructed input bigraphs (we use G_1 for this proof), the common bigraph G_M with the least constrained possible interface (fully open from above and below by assigning sites and regions where possible) can thus be built by disregarding all elements which are not part of the solution mapping (and thus cannot appear in G_M).

We perform this for the place graph $P = (V_M, ctrl_M, prnt_M)$ as follows:

$$V_M = \{u \in V_{G_1} \mid M(u) \neq \emptyset\}$$
$$\{\forall v \in V_M \mid ctrl_M(v) = ctrl_{G_1}(v)\}$$
$$\{\forall (v_i, v_j) \in V_M \mid prnt_{G_1}(v_j) = v_i \rightarrow prnt_M(v_j) = v_i\}$$
$$\{\forall v \in V_M \mid prnt_M(v) \notin V_M \rightarrow prnt_M(v) = r \in n\}$$
$$\{\forall v \in V_M \mid (s \in m) \in prnt_M^{-1}(v)\}$$

As with our matching proof for links (Section 3.5.1), we add a new outer name as a sink for each $g \in P_M$ where $link(g) \notin E_M$. The link graph $L = (V_M, E_M, ctrl_M, link_M)$ can then be constructed as follows:

$$V_M = \{u \in V_{G_1} \mid M(u') \neq \emptyset\}$$
$$P_M = \{p \in P_{G_1} \mid M(p') \neq \emptyset\}$$
$$E_K = \{e \in E_{G_1} \mid M(e') \neq \emptyset\}$$
$$\{\forall v \in V_M \mid ctrl_M(v) = ctrl_{G_1}(v)\}$$
$$\{\forall (g'_a, g'_b) \in E_{\phi_f(G_K)} \mid \ell(g'_a) = \mathbf{link}, \ell(g'_b) = \mathbf{closure} \rightarrow link_G(g_a \in P_G) = g_b \in E_G\}$$
$$\{\forall p \in G_M \mid link_M(p) \notin E_M \rightarrow link_M(g) = y \in Y\}$$

Thus, we are able to build the common bigraph G_M. In doing so, this provides us the required set of triple relations $(u \in G_1, v \in G_2, w \in G_M)$, where each w corresponds to the mapping of common elements $(u, v) \in M$.

From here, we can effectively reduce the problem to a pair of bigraph matching instances, where G_M must occur as a pattern in both G_1 and G_2 in order to demonstrate that $G_1 = C_1 \circ (G_M \otimes id) \circ D_1$ and $G_2 = C_2 \circ (G_M \otimes id) \circ G_2$. As we have already proven how to retrieve these decompositions by construction for bigraph matching instances once P and T have been decoded, we refer to our matching proof for soundness (Section 3.5.1) which demonstrates how C_1 and D_1 can be built where $\texttt{match}(P \leftarrow G_M, T \leftarrow G_1)$, and similarly C_2 and D_2 where $\texttt{match}(P \leftarrow G_M, T \leftarrow G_2)$. As this now gives us our initial pair of MCB compositions and the common bigraph G_M from its MCIS encoding and set of vertex assignments M, this concludes our proof by construction.

Maximality

Maximality can also be proven by construction through analysis of the modified scoring function as follows. The score of M will either be -1 if the current set of assignments decode

into an incomplete bigraph, or conversely if the solution is structurally sound, the score of M will be $\sum_{(u',v')\in M} \{\ell(u') \neq \textbf{link}\}$. In any MCB instance, the minimum possible score of any instance will always be 0, where $M = \{\}$ and $G_M = \emptyset$, therefore an incomplete bigraph will never be a solution and we can assume $score(M) >= 0$ is always true for a final result.

From this, we can infer that the score of the MCIS solution M will always equal the support size of G_M in the original MCB instance, as ports do not count toward support size, and this is reflected in our MCIS adaptation by enforcing that their corresponding flattened link nodes do not count toward the scoring function. Therefore we can conclude through this bijection that the maximum scoring subgraph(s) returned by MCIS must always correspond to the maximum common structure(s) G_M between G_1 and G_2. This concludes our proof.

6.5.2 Completeness

Given an instance of MCB (G_1, G_2) where $G_1 = C_1 \circ (G_M \otimes id) \circ D_1$ and $G_2 = C_2 \circ (G_M \otimes id) \circ D_2$, for solid bigraphs G_1 and G_2 and the shared bigraph G_M, and there exists a solution in the form of an injective embedded mapping of size m, $S_{MCB} = \{(u_1, v_1), ..., (u_m, v_m)\}$ from a subset of support elements $u \in G_1$ to a support element $v \in G_2$, we wish to prove that a parallel solution of the same size $M = \{(u'_1, v'_1), ..., (u'_n, v'_n)\}$ exists in the modified MCIS instance $(\phi_f(\phi_P(G_1)), \phi_f\phi_P(G_2))$, where $u'_k \rightarrow v'_k$ is the encoded form of support elements $u_k \rightarrow v_k$. Similarly to our soundness proof, we wish to prove two key attributes: firstly, that no valid MCB solution will be incorrectly filtered by the encoding or constraints. Secondly, that no optimal MCB solution will be incorrectly pruned by the bound function.

Validity

Assume that there exists a valid pair of compositions $G_1 = C_1 \circ (G_M \otimes id) \circ D_1$ and $G_2 = C_2 \circ (G_M \otimes id) \circ D_2$, with a corresponding embedding of G_M in both bigraphs $S_{MCB} = \{(u_1, v_1), ..., (u_m, v_m)\}$ of size m, where the corresponding MCIS solution $M = \{(u'_1, v'_1), ..., (u'_m, v'_m)\}$ is not a valid solution. This suggests that at least one of our newly added constraints are being violated.

We first consider our method of encoding the bigraphs. By construction, all parent relations between entities in $G_K, K = \{1, 2\}$ are preserved through the encoding of $prnt_{G_K}(v) = u$ as the edge (u', v') in E_K, and thus conventional MCIS holds. Trivially, as controls are preserved through graph labelling, McSplit will split any incompatible entities apart before search, ensuring only compatible entities can be mapped to one another. This also sufficiently handles flattened edge nodes as for any (u', v'), $u \in E_K \iff v \in E_K$. From Proposition 6 (Section 3.5), we know that the degree and solidity constraints on closure nodes will never

be violated for a valid bigraph composition, as $link_{G_M}(p \in P_{G_M}) = e \in E_{G_M}$ if and only if $link_{G_K}(p) = e$.

Finally, we now consider our connectedness constraint on entities. By Proposition 15, we have proven that for all pairs of entities $(u, v) \in G_M$ which are transitively adjacent, then all entities between u and v must also be in G_M and cannot be assigned to either of C_1, C_2, D_1, D_2. Therefore, this constraint will always return **true** for any valid common bigraph. This thus exhausts all extra constraints in the McSplit MCIS model.

Our original hypothesis that a constraint violation occurs is shown to be a contradiction, and therefore S_{MCB} must be a valid MCIS solution. This concludes the proof.

Bound Consistency

We prove bound consistency by contradiction by setting up the following scenario. Given a MCB instance and solution (G_1, G_2, G_M), our MCIS encoding should return a corresponding solution $M = \{(u'_1, v'_1), ..., (u'_m, v'_m)\}$ where $\sum_{(u',v') \in M} \{\ell(u') \neq \textbf{link}\} = |G_M|$. However, let us assume that for some partial solution $K = \{(u'_1, v'_1), ..., (u'_k, v'_k)\}, k < m$ which eventually reaches M, the bound check fails and the remaining search tree is pruned, incorrectly preventing M from being found. We set up the bound check so that it is at its strictest, and so we assume that the current best score at this stage is already $|G_M|$ and that M is a valid solution of equal size to the known best maximum.

For the bound to fail, the following must be true:

$$|G_M| > |G_K| + \sum_{l \in LC_K} min(|(u \in G_1 \wedge LC_K(u) = l \ \wedge \ u \neq (p, i) \in P_{G_1})|,$$

$$|(v \in G_2 \wedge LC_K(v) = l \ \wedge \ v \neq (q, j) \in P_{G_2})|)$$

Where all label classes $l \in LC_K$ contain disjoint subsets of (but do not necessarily together make up the whole sets of) vertices from G_1 and G_2.

We can infer by our encoding that $|G_K| = \sum_{(u',v') \in K} \{\ell(u) \neq \textbf{link}\}$. We now define $R = \{(u', v') \in \{\{M \setminus K\} \mid \ell(u) \neq \textbf{link}\}$, which consists of all non-port element pairs (and therefore all support elements) in $\{M \setminus K\} = \{(u'_{k+1}, v'_{k+1}), ..., (u'_m, v'_m)\}$, the set of assignments which have yet to be added to K in order to reach M. By inspection, $|R| = |G_M| - |G_K|$. We substitute this back into the violated bound function as follows:

$$|\{(u', v') \in \{\{M \setminus K\} \mid \ell(u) \neq \textbf{link}\}| >$$

$$\sum_{l \in LC_K} min(|(u \in G_1 \wedge LC_K(u) = l \ \wedge \ u \neq (p, i) \in P_{G_1})|,$$

$$|(v \in G_2 \wedge LC_K(v) = l \ \wedge \ v \neq (q, j) \in P_{G_2})|)$$

To simplify, this now states that the number of non-port vertices yet to be assigned for K to reach M must exceed the number of combined pairs of vertices across all non-port label classes which are available for selection. Decoding this back to our original MCB instance, this suggests that adding all remaining compatible support elements to G_K in the pair of compositions $G_1 = C_1 \circ (G_K \otimes id) \circ D_1$ and $G_2 = C_2 \circ (G_K \otimes id) \circ D_2$ is not enough for it to reach a support size of $|G_M|$, and therefore G_M itself cannot be an optimal solution. This contradicts our original hypothesis, and can only occur if M is not an optimal solution to MCIS in the first place. Therefore the bound function will only prune non-optimal solutions, and thus concludes our proof.

6.6 Finding Minimal Contextual Transitions

We now consider our approach toward adapting MCB to find all partial matches where the composition of a minimal context would allow for a full match to occur — that is, instead of finding all largest instances of G_M such that $G_1 = C_1 \circ (G_M \otimes id) \circ D_1$ and $G_2 = C_2 \circ (G_M \otimes id) \circ D_2$, we now wish to find all instances of G_M where $r = C_M \circ G_M$ and $A = C_r \circ G_M \circ D$, where the necessary interface components exist to allow for the composing of C_M onto the occurrence of G_M in A. From being provided a list of solutions by the solver, a BRS toolkit such as BigraphER can then simply rely upon existing rewriting capabilities to obtain the corresponding minimal context of each solution by taking the complement of each G_M against r. As this is performed in order to build a minimal CTS, we describe this as the MCTS matching problem going forward. For simplicity, we assume that the input redex r and agent A assume the place of the G_1 and G_2 inputs of MCB respectively.

The main modifications we make to allow our MCB algorithm to perform MCTS are as follows: firstly, we constrain solutions to only occur from the "bottom" of the pattern bigraph r, to ensure that the parameter of G_M is always empty (as agents are grounded, composition on the inner face of A to reconstruct r is forbidden). We then conversely constrain solutions to only occur on the "top" of the agent bigraph A, verifying for each candidate solution that r can be constructed inside A through composing the remainder C_M onto the available regions and faces of G_M inside A. Finally, we modify our main search loop to find all *maximal* solutions, rather than just all *maximum* solutions — that is, any valid partial solution where no further solutions can be retrieved through further vertex assignments is considered a maximal common bigraph. Figure 6.10 demonstrates an instance of MCTS which can be solved in this manner.

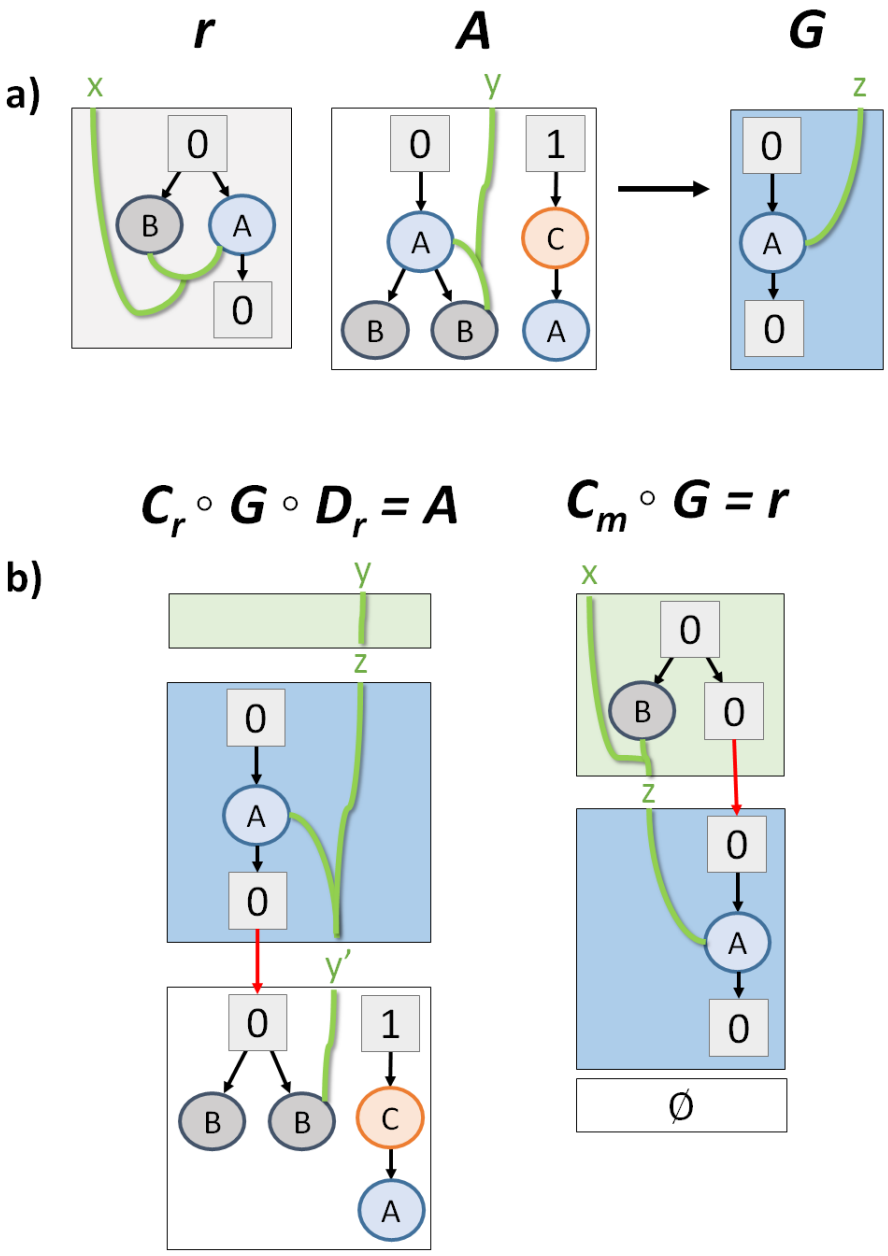


Figure 6.10: An example instance demonstrating that finding the minimal context for a MCTS can be treated as MCB with the extra constraint that the parameter beneath G must be empty when constructing r, and that $C_m \circ A$ is a valid composition. C_m denotes the minimal context needed to produce an occurrence of r in A.

6.6.1 Ensuring Null Parameter

The first step to ensuring that $r = C_M \circ G_M \circ D_M$ where $D_M = \emptyset$ holds is forbidding any entity $u \in r$ from appearing in D_M. This can be enforced at variable selection time, where we add further constraints to prohibit any vertex from being selected if it has any child vertices that have yet to be assigned. We modify MCB to handle this in a similar fashion to how closure nodes are currently constrained to ensure solid links, where whenever a vertex assignment is made, its parent is checked to determine whether all of its children have now been assigned, and toggled to visible in their label class's bitarray structure if so. Additionally, this means that at the beginning of search, only leaf nodes in the place graph will initially be allowed to be selected. As a constraint, this can be represented as follows:

$$\{\forall u, v \in V_r \mid prnt_r(u) = v \ \wedge \ \texttt{match}(u) = \emptyset\} \rightarrow \texttt{match}(v) = \emptyset$$

The second step to ensuring that nothing exists below the occurrence of G_M in r is that we no longer add a site to an entity in G_M if its corresponding assignment in r does not have one — this is necessary to prevent any G_M occurrence where the additional site would have to be closed in the parameter, rendering D_M a non-empty bigraph. As a consequence, this also requires the re-introduction of conditional degree constraints on encoded vertices, where if an entity in the pattern rule r has no site, then its mapped assignment in A must share the same out-degree. (Section 3.2.1). Similarly to our matching algorithm, we construct a label compatibility function to enforce this as follows:

$$\ell(u \in V_r, v \in V_A) = \begin{cases} t \text{ if } prnt(u)^{-1} \cap m = \emptyset \wedge \delta^+(v) = \delta^+(u) \\ f \text{ otherwise} \end{cases}$$

This can be applied to a modified McSplit implementation in practice through the introduction of a matrix of booleans **forbid_pairs**, where **forbid_pairs**$[u][v] =$ **true** if a preliminary process has deemed that the mapping $(u \in G_1 \rightarrow v \in G_2)$ can never be valid. The algorithm can then conditionally skip over the assigning of u to v during the vertex assignment stage. Before the main search loop, all $(u \in V_r, v \in V_A)$ pairs of entities are compared, and **forbid_pairs**$[u][v] =$ **true** if u lacks a site and their out-degree values don't match. Because the agent A is always grounded in this context, we do not need to handle the inverse case where a site is adjacent to v. In addition, at the RPO construction step, we no longer attempt to modify/close the sites of G_M.

The two additional constraints described restrict D_M from containing any entities and regions respectively, and as all resultant link graph composition is already handled in the context for non-directed bigraph matching (thus never requiring wiring through the lower face of G_M),

these are sufficient to always ensure that $D_M = \emptyset$ and therefore guarantee all solutions will occur from the bottom of r.

6.6.2 Checking For Valid Compositions

We now consider how to constrain the algorithm further to only permit the recording of solutions if the remainder of r (the minimal context) is able to be composed onto the occurrence of G_M in A, allowing r to be reconstructed as a pattern of A. This can be checked for a given candidate set of assignments $M = \{(u_1, v_1), ..., (u_k, v_k)\}$, where we consider the $prnt_r(u)$ and $link_r(u)$ relations of each top-level entity and port in G_M respectively, and compare their values to that of their mappings $prnt_A(v)$ and $link_A(v)$ to verify whether composition is possible. The four key cases that we wish to check are the following:

- For each triple relation $(u \in r, v \in A, w \in G_M)$ where w is a top-level entity, if u has a parent, does v have a corresponding region such that $prnt(v) = prnt(u)$ is possible in $C_M \circ A$?
- For each triple relation $(u \in r, v \in A, w \in G_M)$ where w is a port, if u is linked to a closed edge e not in G_M, is v an open port such that $link(v) = e$ is possible in $C_M \circ A$?
- For each shared region of A participating in the composition, is its structure preserved?
- For each shared outer face of A participating in the composition, is its structure preserved?

The first two cases are trivial to determine given a current partial solution — it simply requires checking if the required interface components are adjacent to the occurrence of G_M in A in order for composition to be possible. The third and fourth cases are more complex, and must be checked due to the possibility that a given candidate solution expects a region or face to be occupied by two conflicting components at once. We demonstrate the third and fourth cases through Figures 6.11 and 6.12 respectively, where the solution $\{(A, A), (B, B)\}$ is checked for each reaction rule redux against the agent.

In Figure 6.11, r_1 is not compatible as the A entity requires the shared region to be closed in the context while simultaneously the B entity requires the region to stay open in order to permit a match occurrence. Similarly, r_2 would require that the shared region both remain open and also be occupied by entity C in the context. r_3 requires that both the C and D entities compose onto the same region to permit a match, which is prohibited in pure bigraphs without sharing. r_4 is compatible with an empty minimal context as r_4 already matches in A as is, where their separate regions can be merged in the matching context to reproduce the

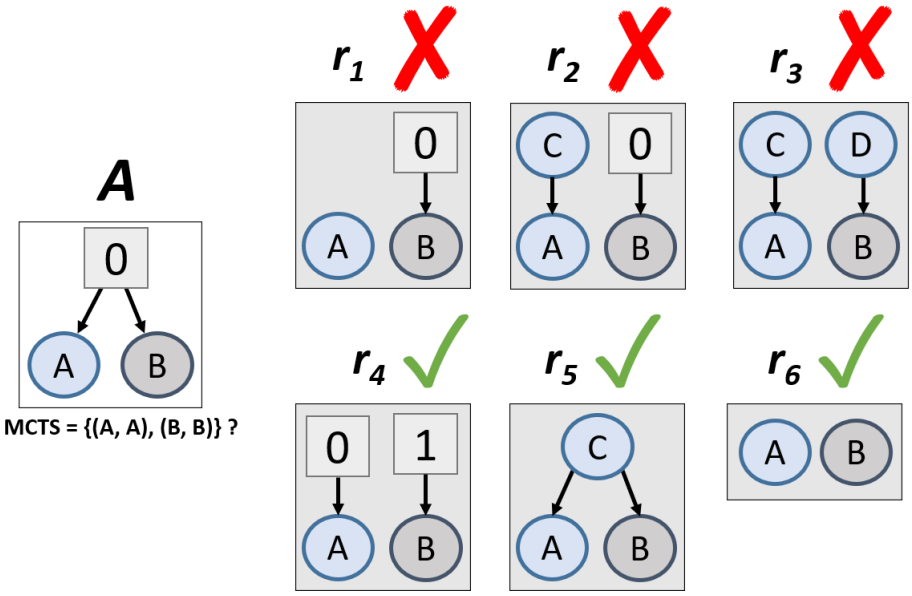


Figure 6.11: An example place graph agent A with a shared region and six reaction rule reduxes r_1 to r_6. For rules r_1 to r_3, the MCTS solution $\{(A, A), (B, B)\}$ is not possible as the shared region cannot encapsulate the required parent relations of A and B in the minimal context. For rules r_4 to r_6, the parent relations in r are compatible and the solution is possible.

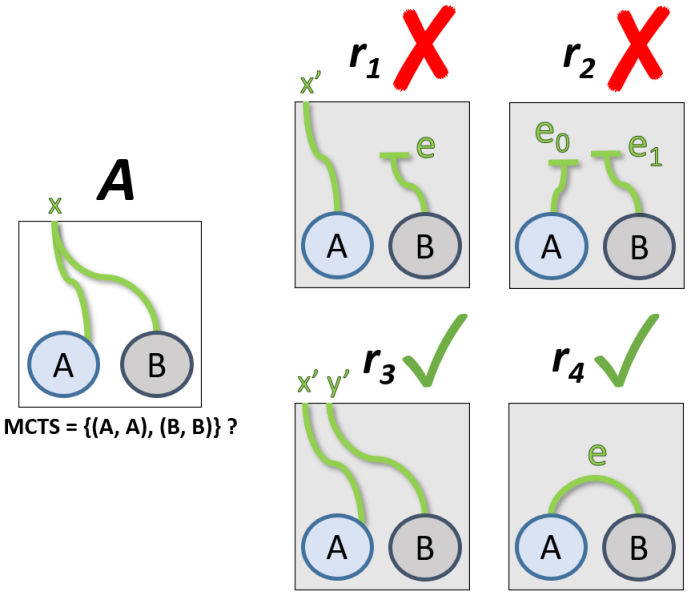


Figure 6.12: An example link graph agent A with a shared open hyperedge and four reaction rule reduxes r_1 to r_4. For rules r_1 and r_2, the MCTS solution $\{(A, A), (B, B)\}$ is not possible as the hyperedge cannot encapsulate the required link relations of A and B in the minimal context. For rules r_3 and r_4, the link relations in r are compatible and the solution is possible.

agent A. r_5 allows the match by composing the entity C onto the shared region. Finally, r_6 is compatible as it can simply close the shared region in C_M.

In Figure 6.12, r_1 is not compatible as the A entity requires the face x to remain open while B simultaneously forces a closure. r_2 is similarly incompatible as it would require composing more than one closed edge onto the same face. r_3 is compatible as it already occurs in the agent, and x' and y' can be merged in the matching context. Finally, r_4 is compatible as the lone edge e can close x in the minimal context.

From these examples, we can derive that a minimal context composition is valid for a partial solution when the following occurs.

For each region in A, they can only retain one of the following states:

- Remain open, allowing an unbounded number of open regions in r to occupy it, where they will merge in the matching context.
- Close in the minimal context, forbidding any region or entity from occupying it.
- Compose an entity in the minimal context, only permitting that entity to occupy it.

For each outer link in A, they can only retain one of the following states:

- Remain open, allowing an unbounded number of links in r to occupy it, where they will merge in the matching context.
- Close with an edge in the minimal context, only permitting that edge to occupy it.

Thus, any candidate solution that requires any region or link in A to be in more than one of these states at once will not have a valid minimal context that would allow the reaction rule to occur in the agent. All four of the identified requirements can be validated by checking each top-level entity and port within a given candidate set of assignments and comparing their respective parents in G_1 (r) and G_2 (A), keeping track of what each interface component of A encapsulates in r, and ensuring that there no conflicts exist. Algorithm 8 (Appendix D) demonstrates the full algorithm for performing this, using the $region\_lock$ and $face\_lock$ map structures to retain the states of each A region and face respectively. This can hence be validated in $O(|M|)$ time for any given candidate solution.

6.6.3 From Maximum to Maximal

Finally, in order to find *all* minimal contexts, we wish to change our criteria from finding all *maximum* common bigraphs to finding all *maximal* common bigraphs — that is, all valid

mappings where no other valid solution can be obtained through further vertex assigning. While this does not require us to change the underlying McSplit partitioning logic for the most part, this effectively means that our scoring function and bound functions are no longer functional, as we do not track the size of the partial solution anymore. Instead, the validity of candidate solutions are verified upon backtracking to their respective state if the solver did not find a solution deeper in the search tree, after fully exhausting all branches from the solution's position. If a solution is then found, it will return a **True** value to all earlier states to signal that a deeper solution was indeed discovered and that they will not be a maximal. This behavior can be observed through our prototype implementation's main search loop in Algorithm 9 (Appendix D), where whether a deeper state contains a solution is tracked by utilizing the $future$ boolean variable. While removing the bound function may potentially result in less optimal solve times compared to MCB, we theorize that the more constrained nature of the MCTS problem and limited number of variables that can be selected at each step due to the "bottom-up" rule will sufficiently narrow the search space as a trade-off.

A Note on Maximality

We observe that this method of finding maximal solutions, while sufficient to find all valid maximal mappings, will sometimes also return non-maximal solutions due to how McSplit handles null assignments — e.g. a solution of the form $\{(a, 1), (b, 2), (c, 3)\}$ may also return $\{(a, \emptyset), (b, 2), (c, 3)\}$ as a solution if the a vertex is not structurally integral to whether the mapping is compositionally valid. While this potentially introduces false positives into the set of valid maximal bigraphs, these could be handled in a more optimized implementation by representing each solution as a $|G_1| \times |G_2|$ bitmask structure S, where $S[u][v] = 1$ if and only if vertex $u \in r$ is assigned to $v \in A$. False positives can then be filtered out by checking if they are dominated by another solution — that is, if for a candidate solution C and existing solution S, $C \vee S = S$. While this step is not necessary to validate bisimulations when passing the solution set back to a bigraph toolkit like BigraphER, since non-minimal contexts simply model the behavior of their existing minimal counterpart, it is preferred to do this to keep the transition system as small as possible for optimal performance.

6.7 Prototype Implementation

We demonstrate a prototype of our adapted MCB algorithm, which was implemented by modifying a variant of McSplit provided by Trimble, written in Python and originally created as a contribution to the *NetworkX* Python graph library [91]. This variant implements a simplified and unoptimized version of McSplit which does not make use of the efficient label

class structure described in Section 6.2.1, but instead simulates their behavior through creating new Python list structures which store the label class membership of each vertex at each refinement step. Whilst not as efficient as an optimized implementation of McSplit engineered using a more efficient language, experimentation with this simplified implementation allowed for easier engineering and prototyping of the necessary constraints to model MCB and MCTS, which could then be evaluated for correctness using a mix of manually crafted MCB instances and a suite of bigraph matching instances later used for benchmarking.

This implementation also makes use of the McSplit↓ version of the algorithm, where the solver treats the instance as a sequence of decision problems. Beginning with $n = min(|G_1|, |G_2|)$, the search loop attempts to find a maximum common subgraph of size n, then iteratively decrements n by 1 when an UNSAT is returned until at least one solution is found. The bound function is also modified to trigger a backtrack when the bound is strictly less than the goal, rather than less than or equal. It was found that McSplit↓ performs narrowly better overall than default McSplit on a variety of evaluated benchmark instances [88].

We make our code publicly available [92]. During experimentation, it was found that restricting the variable selection heuristic to only allow assignments to ports for n steps after assigning to an entity of arity n in order to enforce the "no entities can have unassigned ports" constraint resulted in improved solve times. This is achieved through keeping track of the $port\_lock$ variable, which is set to n whenever selecting an entity of arity n, and then decremented by 1 every time a port vertex is selected. Whenever $port\_lock > 0$, the score of the current set of assignments is set to zero and considered an invalid solution. Because of the connected-ness constraint on ports, an entity's port vertices only become visible for selection once the vertex itself has been selected, ensuring that this logic is sufficient to guide the variable selection order appropriately.

Algorithm 9 (Appendix D) demonstrates the logic of our adapted main MCTS search function for our modified solver. The functions `Select Vertex` and `Select Label Class` use the in-built vertex selection heuristic employed by the existing McSplit solver logic, while the `Select Port Class` function restricts the label class selection to only those containing port vertices. `Refine Label Classes` uses our adapted partitioning logic as described in Sections 6.4 and 6.6 to preserve the structure of bigraphs when building a solution. When a valid solution is discovered, it is added to the global **solution_list** array.

6.8 Prototype Evaluation

We provide a preliminary evaluation of our prototype tool, using a subset of the non-sharing **Conference** matching instances as a benchmark (Section 5.2.1). Since MCTS takes a reaction rule redux and agent state as input similarly to our bigraph matching context, and both

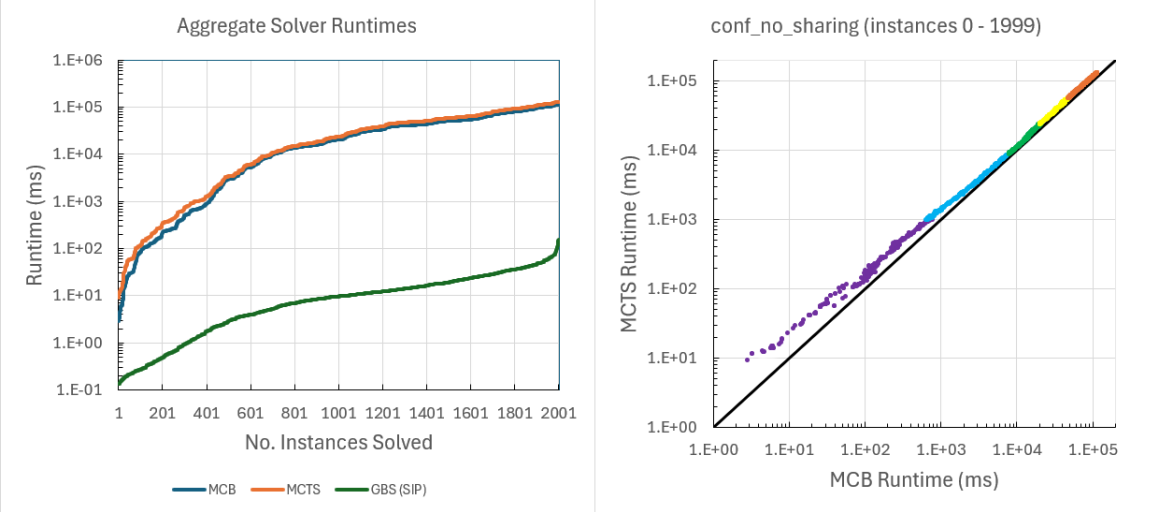


Figure 6.13: (left) The aggregate runtimes achieved by the prototype solver when solving for MCB and MCTS on the set of 2000 conference instances, and that of GBS when performing SIP on the same test suite. (right) Comparing MCB solve time against MCTS for each instance. Colors indicate encoded graph size of the target. Purple: < 300, Blue: 300-599, Green: 600-899, Yellow: 900-1199, Orange: ≥ 1200

search for occurrences of the pattern in order to perform a rewrite and grow their respective transition systems, these test instances are suitable for re-use in this new context. We take from the non-sharing conference instances exclusively for now, as the other model instances include sharing which is not yet supported by this adaptation. We select 2000 easier to solve instances from this dataset, where the pattern rules (an encoded graph size of 6) are matched against target agents of varying sizes, ranging from encoded sizes of 59 up to 1581. Similarly to our matching evaluation, we record the time elapsed upon calling the underlying MCIS search function on the encoded graphs ignoring input/output time, and also including the time taken to build each solution mapping's RPO bigraph after search. When solving MCTS, we do not perform the optional step of filtering non-maximal solutions for this prototype, although this could be achieved by simply comparing pairs of solutions in a more optimized implementation as discussed in Section 6.6.3.

We ensure the confidence of our MCB implementation by verifying that for each instance with at least one full occurrence in bigraph matching, that each of its occurrences also exist in the solution set of MCB — a required property which we identify in Section 6.3.1. This property also holds for MCTS, where the corresponding minimal contexts of each occurrence will be an empty bigraph. We also verify the symmetry of MCB by ensuring that when the order of the input graphs are reversed, that the same set of solutions (with inverted mappings) are always returned for the full suite of instances. We make the full raw dataset available at [76].

Figure 6.13 (left) shows the aggregate solve times achieved by our prototype tool. For in-

terest, we also include the aggregate solve times achieved by GBS when solving bigraph matching on the same set of instances. It can be observed that our maximum common solver does not yet achieve the efficient solve times that GBS is able to obtain, although this was expected, as MCIS is a much harder combinatorial problem to solve than SIP (Section 6.2) — so it follows that a similar jump in difficulty would also be demonstrated here. In addition, this prototype does not yet implement the optimized McSplit data structure for storing label classes, which will have a further impact on performance. Finally, without the use of an optimized compiler under the hood, A solver written in Python will underperform a more low-level language implementation (e.g. C++ for GBS) for optimization problems, without the use of specialized external libraries or an optimized compiler under the hood to support this shortcoming [93]. Even when accounting for these drawbacks however, both MCB and MCTS were able to solve roughly 400 of the problem instances within one second, suggesting that this prototype may still be feasible to use for solving smaller scale models.

Figure 6.13 (right) compares the performance of MCB directly against MCTS on each individual instance. We can observe that the additional constraints and checks required to ensure the structural integrity of the solution bigraph and corresponding minimal context has a mild impact on overall solve time for this benchmark, and the proportional difference in performance decreases further as the instances grow more difficult. On average, it was found that MCTS takes 24.95% more time to solve an instance compared to MCB, which is inconsequential enough to showcase that building minimal contextual transition systems through a maximum common bigraph algorithm is indeed a promising approach. From this comparison, we can also determine that both MCB and MCTS were able to solve encoded target states with a graph size of up to roughly 300 within one second, showing promise for being able to solve bisimulations for small to moderately sized agents using this implementation. Notably, bisimulations for bigraphs cannot be verified for models which produce infinite transition systems, as this signals that there are infinite possible states that each agent can evolve into — at most, a model will only be able to determine "bisimiliar up to N steps" for a pair of agents in this case. Thus, it would not be expected that the agents grow to absurdly large sizes in practice when performing this check.

While we record the additional time spent building the RPO bigraphs of each solution, this was found to take $< 1\%$ of the total solve time across all 2000 instances for both MCB and MCTS, and thus was found to be trivial in this context. This was primarily due to the small pattern sizes of each instance, although in a practical context these pattern rules would be expected to be small regardless, e.g. where the largest average match size across all evaluated BigraphER models was 9 (Table 5.1). However, we note that it may be of interest to record the impact on performance as the provided pattern bigraph increases in size, up to (or even greater than) the size of the target, as part of a full evaluation for a future optimized/parallelized solver.

Opportunities For Parallelization

As with our approach to bigraph matching, this approach shows potential to be improved upon through parallelization. As MCTS also aims to grow the transition system of a given model similarly to our context for performing full matches in BigraphER, the difficulties of performing MCIS could also be alleviated through solving multiple instances simultaneously where possible by sending them to separate threads. In practice, if a MCB solver were integrated into BigraphER similarly to GSS, support for parallel transition system growth could be integrated in a solver-agnostic manner without being concerned what type of transition system is being built or solver is being relied upon under the hood.

In addition to this, this new context presents a further motivation for parallelization from the increased difficulty to solve MCIS. While this was found to be a redundant optimization in GBS as the underlying matches already solved quickly enough that running with multiple threads only introduced overhead (Section 5.2.2), we may wish to re-explore this for harder instances of MCB/MCTS that would otherwise take several seconds up to several minutes to enumerate all solutions for. Archibald et al. have demonstrated that McSplit↓ can be adapted to support parallel solution-biased search similarly to GSS's implementation of SIP, where it was found that this offers a performance gain of up to an order of magnitude on the vast majority of evaluated instances [49].

6.9 Summary

In this chapter, we have provided a definition for the maximum common bigraph problem, which we identify as an optimization-based generalization of bigraph matching where solutions adhere to the RPO property of bigraphs in order to guarantee maximality on compositions. We also describe how this can be used to find the inverse of the smallest possible composition that allows a reaction rule to be applicable, which is the key problem that must be solved within minimal contextual transition systems, and identify additional constraints to guarantee interface compatibility between the context and agent. We then build a prototype backtracking solver based upon the McSplit algorithm to solve both the MCB and MCTS variants of the problem. While this implementation is presently an unoptimized prototype, its evaluated performance on a selection of rule/agent input pairs found that it is theoretically feasible for use on smaller scale models.

The clear avenue presented for further research — now that we have introduced the problem, its use cases and how it can be solved — is to improve upon our approach and build upon a more efficient McSplit solver which implements the optimized label class array structure, such as that provided by Calabrese et al. [89]. In addition to this, further substantial perfor-

mance gains can be retrieved through parallelization of the McSplit search loop as described by Archibald et al. [49]. For MCTS specifically, a more optimized solver may also wish to explore a potential alternate variable selection method which ensures maximality of solutions without filtering.

This adaptation also presently only considers solving MCB/MCTS for pure bigraphs. When considering adapting this approach to extensions of bigraphs, it is important to note that bigraphs with sharing do not have RPOs in the general case, unless the sharing exclusively exists between pairs of entities [94]. Even so, it is possible that a MCB instance where this criteria is met for both G_1 and G_2 may still produce a common bigraph which contains sharing in the regions or sites, which means that MCB for sharing is most likely not be possible via our current definition. Conversely, RPOs exist for directed bigraphs [12], and thus it may be feasible to adapt our encoding to support these as we did for our matching algorithm.

Chapter 7

Conclusion

In this dissertation, we have successfully demonstrated that bigraph matching can be modelled as a CSP model of subgraph isomorphism with additional constraints, and that this strategy in combination with employing a state of the art subgraph CP solver produces efficient and scalable models for bigraph matching that can outperform previous methods of solving by several orders of magnitude on harder instances. This method of encoding is also shown to be extensible and able to support several generalizations and modifications to the bigraph formalism, as shown by the implementations of the bigraphs with sharing and directed bigraphs extensions, as well as bigraph equality, through a small number of extra constraints applied on top of the base model. We have also demonstrated that this can be integrated with the BigraphER BRS toolkit, where for a selection of practical models, our solver substantially improves upon both performance time and the scale of which BigraphER can perform real-time simulations. We also deduce several additional engineering strategies to minimize input/output time based on early results when simulating full models. Building upon this, we also propose the definition of — and provide a novel algorithm for solving using the McSplit algorithm — the maximum common bigraph problem, opening the door to further extensions and executions of BRSs defined by Milner [1] that have yet to be implemented in a BRS, such as building labelled transition systems and identifying bisimulations.

7.1 Future Work

Throughout this dissertation, we have identified various ways to potentially build upon our research on subgraph implementations of bigraph algorithms. These include:

- Add support for building directed BRSs and solving directed matching (Section 4.3) in GBS and BigraphER, and compare the runtime performance of GBS against jLibBig [74] on a suite of directed matching instances.

- Implement the various optimizations and opportunities for parallelization in BigraphER with GBS as identified in Section 5.4 in order to make full use of GBS's promising efficiency in a practical context.
- Implement an optimized variant of the maximum common bigraph algorithm using McSplit (Chapter 6), applying the potential of parallelizing this approach and including the extended functionality for finding all minimal contexts, and integrate this into BigraphER for experimenting with support for identifying bisimilar agents.

7.1.1 Further Extensions

A promising potential for further research would be to extend the SIP solver (and hence BigraphER) to be able to support additional established bigraph extensions through further extra optional constraints. Some established examples of these include:

- **Local bigraphs**: bigraphs which preserve the locality of inner and outer faces of the link graph, which introduces the notion of confluence to bigraph theory and extends the scope of scenarios that can be modelled, particularly in biological settings. [90]
- **Binding bigraphs**: an extension which introduces an alternate possible relation between entity ports called a *binding* in place of a closed edge, which also introduces a locality property to hyperlinks relative to the place graph's structure. [95]
- **Conditional bigraphs**: a conditional BRS which allows reaction rules to specify extra conditions that must be met before being allowed to perform a state transition, such as NOT constraints in the form of additional pattern bigraphs that must not exist in the target. This can be used for guaranteeing uniqueness of specific patterns in an agent. [96]

7.1.2 Proposing Parameterised Bigraphs

While bigraphs with sharing and directed bigraphs extend the place graph and link graphs of the standard bigraph formalism respectively, it is also feasible to extend the properties of the entities themselves. BigraphER using the integrated SIP encoding is currently able to support basic entity types in the form of controls, where only entities of the same $ctrl$ can be mapped to one another during a matching instance, however there is currently no implementation for assigning entities with an additional value independent of their control type, i.e. an integer value representing the current capacity of a room or a recording of how many consecutive states the entity has been present in. Reaction rules would then be able to alter or apply new

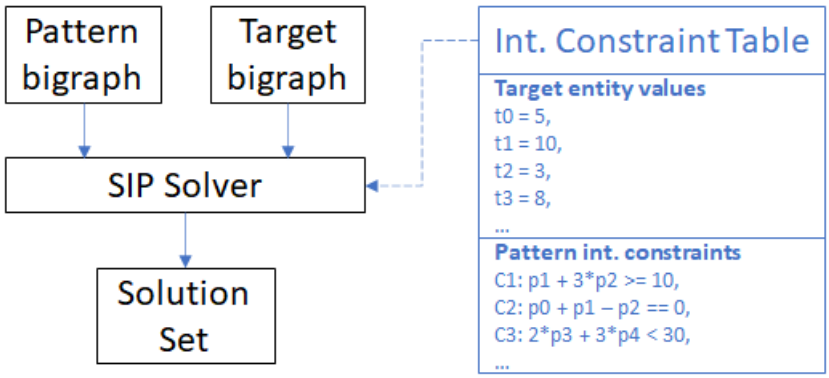


Figure 7.1: Modified SIP solver input to allow an optional third file, defining integer values of the target entity and the integer constraints imposed on the pattern entities during matching of parameterised bigraphs.

values during a rewrite, e.g. an entity's value incrementing from n to $n+1$ when applying $R : r \to r'$. Additionally, r could also specify further constraints on these values that must be met for a reaction rule to be valid, e.g. $n \leq 3$ to enforce a maximum capacity constraint on a given entity type.

The motivation for this is twofold: firstly, it allows for more rich and complex modelling of scenarios that require further constraints than what the current bigraph formalism provides. One such example is a model requiring a clock system where some reaction rules can only apply to a set of entities if a certain amount of time has elapsed, i.e. a traffic light turning from red to green after n steps in a model of a traffic system network. Secondly, this extension would allow for the optimization of existing models where an entity's value could represent information that would previously require the addition of multiple child entities, such as a stadium entity with a variable that represents its current capacity, where the number of people currently in the room can be modelled by this new value rather than having to add a new atomic entity for every person in the room - potentially significantly reducing the overall size of the model and the work required to perform operations in the BRS. One other use-case could be a load balancing problem, where the sum of values for a set of entities must stay within a certain threshold between state changes.

As an initial proof of concept of parameterised bigraphs, this could initially be implemented for only a single integer parameter per control type. However, this could be extended further to allow for several parameters for each $ctrl$ as well as the option of several different data types, where the set of controls would begin to resemble a class system in an object orientated programming language.

To support this extension for bigraph matching instances, it can be assumed that each entity of the target bigraph is already given a concrete value. Conversely, the values of the pattern bigraph (representing the LHS of a reaction rule) will be represented in the form of a set of

integer linear inequality constraints $C_1...C_n$ between entities, where all constraints can be represented by one of the following equations, for each vertex v in the set of pattern vertices and some integer values $a_0...a_k$ and s:

1. $\sum_{n=0}^{k} a_n v_n >= s$ 2. $\sum_{n=0}^{k} a_n v_n == s$ 3. $\sum_{n=0}^{k} a_n v_n <= s$

These integer programming constraints allow for arithmetic constraints to be defined between any combination of pattern graph entities in order to further constrain the set of possible solutions for a matching instance. This can be implemented by allowing the SIP solver to accept an optional third file as an input, which defines the target entity values and pattern constraints as shown in figure B.1. An additional interesting feature of representing integer constraints in this format is that it also allows for CNF encoding between entities, which effectively grants functionality for boolean logic in addition to standard numerical summing: for example, the CNF clause $(x \vee y \vee \neg z)$ can be represented in the form $x + y + (1 - z) >= 1$ where $x, y, z \in \{0, 1\}$, and other clause variations can be represented similarly.

7.1.3 Counting and Sampling

There can potentially exist instances of bigraph matching where the graph states are either so large, or the instance contains so many solutions, that it is impractical to exhaustively search for and return every valid set of vertex assignments. Another possibility is that simply storing and printing out every solution ends up being very slow e.g., for a very small pattern bigraph matched against a huge target bigraph. Large scale traffic networks [2] or power grids may require up to tens of thousands of entities to accurately model, and due to the NP-hard nature of matching, the solution set will not be returned in a reasonable time. In these cases, it may be more useful to instead count the number of solutions in a matching instance without listing all the vertex mappings themselves, or even approximate the number of solutions which exist in cases where a complete search is not feasible.

As a path forward for further research, methods of adding support for counting and sampling could be evaluated, then adapted and implemented into the SIP solver to partially support very large matching problems which would otherwise be completely unsolvable, which opens the door further for applying BRSs for modelling real-world problems. Approximate model counting algorithms have already been created in conjunction with SAT solvers [97], which shows promise for developing an approximated solver for bigraph matching as a component of the SIP solver.

There also exist algorithms for achieving practically uniform sampling for really large problems in the domain of CP in particular [98], which are implemented using configurable linear modular constraints to obtain an appropriate sampling of solutions. These can also

be particularly useful in the context of stochastic bigraphs [6] and probabilistic bigraphs [59], where symmetric solutions that would otherwise be discarded (due to inefficiency introduced by enumerating them) can potentially alter the overall probabilities of reaching each specific agent state; this could be alleviated by instead exploring a method for sampling these symmetries to obtain an acceptable approximation of overall probability for each set of non-isomorphic solutions.

Appendix A

Formal Concrete Bigraph Example

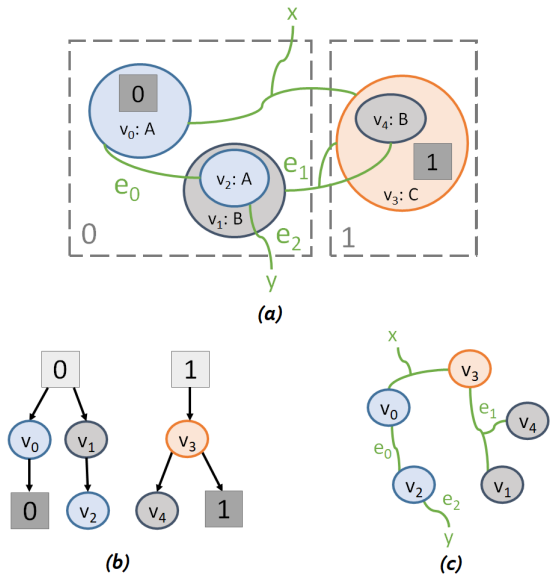


Figure A.1: Concrete bigraph example as provided in Section 2.3.

The provided concrete bigraph example can be mathematically described as follows.

$$
\begin{aligned}
&B = (V_B, E_B, ctrl_B, prnt_B, link_B) : \langle 2, \{y\}\rangle \to \langle 2, \{x\}\rangle,\\
&\mathcal{K}_B = \{(A, 2), (B, 1), (C, 2)\},\\
&V_B = \{v_0, v_1, v_2, v_3, v_4\},\\
&E_B = \{e_0, e_1, e_2\},\\
&ctrl_B = \{(v_0, A), (v_1, B), (v_2, A), (v_3, C), (v_4, B)\},\\
&prnt_B = \{(0, v_0), (1, v_3), (v_0, 0), (v_1, 0), (v_2, v_1), (v_3, 1), (v_4, v_3)\},\\
&link_B = \{(y, e_2), ((v_0, 0), e_0), ((v_0, 1), x), ((v_1, 0), e_1), ((v_2, 0), e_0), ((v_2, 1), e_2),\\
&\qquad ((v_3, 0), e_1), ((v_3, 1), x), ((v_4, 0), e_1)\}
\end{aligned}
$$

Appendix B

Pushouts on Bigraphs

We provide a brief overview of spans, bounds and pushouts in the context of bigraphs as described by Milner [1], to give additional context for the necessity for constructing IPOs as part of the maximum common bigraph problem described in Chapter 6.

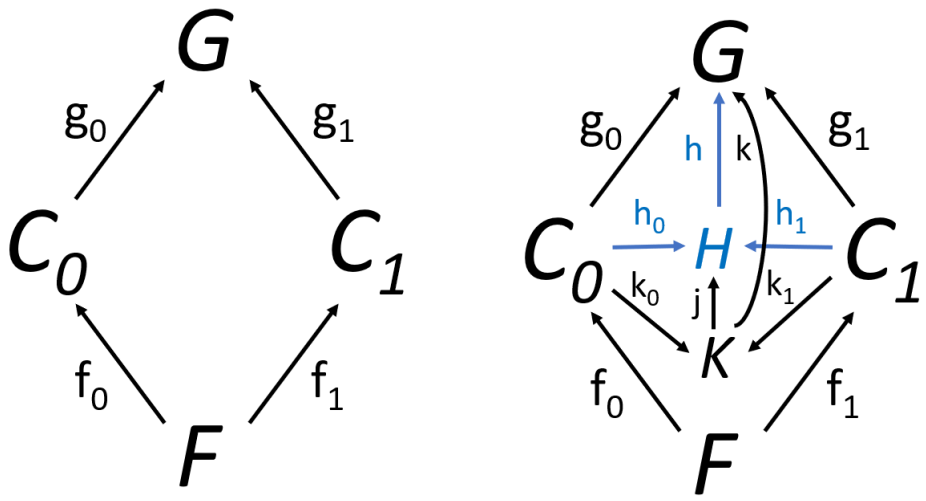


Figure B.1: A visual demonstration of pushouts for bigraphs. On the left, a cospan $\vec{g}$ bounding a span $\vec{f}$. On the right, the RPO $\vec{h}$ for this span, where any other bound $\vec{k}$ will produce a larger bigraph K such that $K = H \circ j$ for some j. If the bound itself is an RPO then the cospan $\vec{g}$ is an IPO.

Definition B.0.1 (span)**.** A *span* $\vec{f} : (f_0, f_1)$ describes a pair of concrete bigraphs that can be obtained through two different decompositions of a bigraph F, i.e. $G = C_0 \circ f_0 = C_1 \circ f_1$.

Definition B.0.2 (cospan)**.** A *cospan* $\vec{g} : (g_0, g_1)$ describes a pair of concrete bigraphs that when decomposed from two bigraphs C_1 and C_2 respectively, produce the same resultant bigraph G, i.e. $C_1 = g_0 \circ G$ and $C_2 = g_1 \circ G$.

Definition B.0.3 (bound)**.** A cospan *bounds* a span if $g_0 \circ f_0 = g_1 \circ f_1$, and hence this bound $\vec{g}$ for $\vec{f}$ denotes two separate paths for decomposing a bigraph F to reach the same result G when this occurs.

Definition B.0.4 (pushout)**.** A *pushout* $(\vec{h}, h)$ for $\vec{f}$ is a triple (h_0, h_1, h) which describes a bound for $\vec{f}$ such that for any other possible bound $\vec{g}$ for $\vec{f}$, there is a unique bigraph h such that $h \circ h_0 = g_0$ and $h \circ h_1 = g_1$.

Definition B.0.5 (relative pushout)**.** A pushout $(\vec{h}, h)$ is considered a *relative pushout* (RPO) for $\vec{f}$ bounded by $\vec{g}$ if for any other relative bound $(\vec{k}, k)$, there exists a unique bigraph j such that $j \circ \vec{h} = \vec{k}$ and $k \circ j = h$, and thus introduces a notion of a minimal relative bound—by this definition there can not exist any other bound where the H can be decomposed further to produce the resultant bigraph.

Definition B.0.6 (idem pushout)**.** The cospan $\vec{g}$ for a span $\vec{f}$ with resultant bigraph G is considered an *idem pushout* (IPO) of $\vec{f}$ if $(\vec{g}, id)$ itself is an RPO for $\vec{f}$ bounded by $\vec{g}$, defining the minimal bound for the triple (f_0, f_1, G).

Appendix C

BigraphER Data Tables

BRS	GBS 25%	GBS 50%	GBS 75%	GBS 100%
abrs-mobilesink	3.661	7.993	12.4723	18.726
plato-graphical-loc	13.729	39.17	66.819	105.234
virus-simpl	1.5391	3.813	6.7079	10.6844
hospital	4.584	11.79	22.163	38.165
plato-graphical	18.218	50.0736	89.398	137.139
savannah-general	39.006	122.599	273.81	420.095
dining_philosophers	1.4556	3.1127	4.8409	6.9157
virus-multifirewall	1.74	4.3861	7.827	12.5212
AutoBigraphER_127	46.1696	103.092	171.665	245.32
AutoBigraphER_52	31.5901	76.8971	125.052	182.542
AutoBigraphER_83	43.297	111.602	170.981	239.707
floor_security_robot10_1	26.53	223.059	440.713	721.169
link_inst_map	15.1791	125.401	442.047	1088.64

Table C.1: GBS runtimes for each BRS, in seconds

BRS	MSAT 25%	MSAT 50%	MSAT 75%	MSAT 100%
abrs-mobilesink	11.429	25.908	41.903	65.218
plato-graphical-loc	36.657	122.473	235.843	443.007
virus-simpl	4.5189	11.2515	20.0451	32.623
hospital	12.459	31.209	56.434	93.68
plato-graphical	36.7858	124.898	256.451	436.846
savannah-general	64.775	235.36	614.127	1,061.27
dining_philosophers	2.59421	5.4347	8.3754	11.9226
virus-multifirewall	7.0933	17.8304	32.0598	52.0769
AutoBigraphER_127	224.099	551.03	990.508	1474.207
AutoBigraphER_52	149.384	405.506	673.67	1,029.13
AutoBigraphER_83	203.793	632.538	964.225	1402.153
floor_security_robot10_1	24.29	153.876	284.676	446.288
link_inst_map	11.973	95.779	333.884	805.739

Table C.2: MSAT runtimes for each BRS, in seconds

BRS	CARD 25%	CARD 50%	CARD 75%	CARD 100%
abrs-mobilesink	4.782	10.667	17.153	26.923
plato-graphical-loc	16.353	53.853	107.666	212.718
virus-simpl	1.7519	4.3171	7.4482	12.17
hospital	4.7023	12.089	22.72	39.8
plato-graphical	19.5867	65.4058	137.917	243.826
dining_philosophers	1.2286	2.5788	4.01	5.7816
virus-multifirewall	2.3487	5.854	10.429	16.8707
AutoBigraphER_127	67.723	165.276	296.42	446.22
AutoBigraphER_52	46.261	124.422	209.093	322.203
AutoBigraphER_83	61.672	187.642	289.862	427.254
floor_security_robot10_1	12.2862	93.591	167.124	262.749
link_inst_map	11.9207	92.42	331.398	804.103

Table C.3: MCARD runtimes for each BRS, in seconds

Appendix D

MCTS Algorithms

Algorithm 8 `Check Valid Composition` (G_1, G_2, M)

$region\_lock \leftarrow \emptyset$
$face\_lock \leftarrow \emptyset$
for all $m = (u, v) \in M$ **do**
  if $l(u) = link$ **then**
    if $link_{G_1}(u) = e \in E_{G_1}$ **and** $M(e) = \emptyset$ **and** $link_{G_2}(v) = e' \in E_{G_2}$ **then**
      return False
    end if
    if $link_{G_2}(v) = y \in Y$ **then**
      $compose \leftarrow \emptyset$
      if $link_{G_1}(u) = e \in E_{G_1}$ **then**
        $compose \leftarrow e$
      else if $link_{G_1}(u) = y' \in Y$ **then**
        $compose \leftarrow open$
      end if
      if $face\_lock[compose] = \emptyset$ **then**
        $face\_lock[y] \leftarrow compose$
      else if $face\_lock[compose] \neq y$ **then**
        return False
      end if
    end if
  else if $l(u) \neq closure$ **then**
    if $prnt_{G_1}(u) = u' \in V_{G_1}$ **and** $M(u') = \emptyset$ **and** $prnt_{G_2}(v) \neq r \in n$ **then**
      return False
    end if
    if $prnt_{G_2}(v) = r \in n$ **then**
      $compose \leftarrow \emptyset$
      if $prnt_{G_1}(u) = u' \in V_{G_1}$ **then**
        $compose \leftarrow u'$
      else if $prnt_{G_1}(u) = r' \in n$ **then**
        $compose \leftarrow open$
      else if $prnt_{G_1}(u) = \emptyset$ **then**
        $compose \leftarrow close$
      end if
      if $region\_lock[compose] = \emptyset$ **then**
        $region\_lock[r] \leftarrow compose$
      else if $region\_lock[compose] \neq r$ **then**
        return False
      end if
    end if
  end if
end for
return True

Algorithm 9 `MCTS Search`$(G_1, G_2, M, LCs, port\_lock)$

if $port\_lock > 0$ **then**
  $label\_class = [LC_{G_1}, LC_{G_2}] \leftarrow$ `Select Port Class`(LCs)
else
  $label\_class = [LC_{G_1}, LC_{G_2}] \leftarrow$ `Select Label Class`(LCs)
end if
if $label\_class = \emptyset$ **then**
  if $port\_lock = 0$ **and** `Check Valid Composition`(G_1, G_2, M) **then**
    solution_list $\leftarrow$ (**solution_list** $\cup M$)
    return True
  end if
  return False
end if
$u \leftarrow$ `Select Vertex`$(label\_class)$
if $l(u) = port$ **then**
  $new\_port\_lock \leftarrow port\_lock - 1$
else if $l(u) \neq closure$ **then**
  $new\_port\_lock \leftarrow ar(l(u))$
end if
$future \leftarrow$ **False**
for all $\{v \in LC_{G_2}\} \uplus \{v = \emptyset\}$ **do**
  if not forbid_pairs$[u][v]$ **then**
    $LC_{G_2} \leftarrow LC_{G_2} \setminus v$
    $M \leftarrow M \uplus (u, v)$
    $new\_LCs \leftarrow$ `Refine Label Classes`(LCs, M, u, v)
    $future \leftarrow$ `MCTS Search`$(G_1, G_2, M, new\_LCs, new\_port\_lock)$ **or** $future$
    $M \leftarrow M \setminus (u, v)$
  end if
end for
if future $= false$ **then**
  if $port\_lock = 0$ **and** `Check Valid Composition`(G_1, G_2, M) **then**
    solution_list $\leftarrow$ (**solution_list** $\cup M$)
    return True
  end if
  return False
end if
return $future$

Bibliography

[1] R. Milner, *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[2] M. Sevegnani, M. Kabác, M. Calder, and J. A. McCann, "Modelling and verification of large-scale sensor network infrastructures," in *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018*. IEEE Computer Society, 2018, pp. 71–81. [Online]. Available: https://doi.org/10.1109/ICECCS2018.2018.00016

[3] B. Archibald, M. Shieh, Y. Hu, M. Sevegnani, and Y. Lin, "BigraphTalk: Verified design of IoT applications," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2955–2967, 2020.

[4] M. Calder, A. Koliousis, M. Sevegnani, and J. S. Sventek, "Real-time verification of wireless home networks using bigraphs with sharing," *Science of Computer Programming*, vol. 80, pp. 288–310, 2014.

[5] M. Calder and M. Sevegnani, "Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing," *Formal Asp. Comput.*, vol. 26, no. 3, pp. 537–561, 2014.

[6] J. Krivine, R. Milner, and A. Troina, "Stochastic bigraphs," *Electr. Notes Theor. Comput. Sci.*, vol. 218, pp. 73–96, 2008.

[7] F. Alrimawi, L. Pasquale, and B. Nuseibeh, "On the automated management of security incidents in smart spaces," *IEEE Access*, vol. 7, pp. 111 513–111 527, 2019.

[8] F. Alrimawi, L. Pasquale, D. Mehta, N. Yoshioka, and B. Nuseibeh, "Incidents are meant for learning, not repeating: Sharing knowledge about security incidents in cyber-physical systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 120–134, 2022.

[9] M. Sevegnani and M. Calder, "BigraphER: Rewriting and analysis engine for bigraphs," in *Computer Aided Verification - 28th International Conference, CAV 2016. Proceedings, Part II*, 2016, pp. 494–501.

[10] M. Sevegnani, C. Unsworth, and M. Calder, "A SAT based algorithm for the matching problem in bigraphs with sharing," Department of Computing Science, University of Glasgow, Tech. Rep. TR-2010-311, 2010.

[11] M. Sevegnani and M. Calder, "Bigraphs with sharing," *Theor. Comput. Sci.*, vol. 577, pp. 43–73, 2015.

[12] D. Grohmann and M. Miculan, "Directed bigraphs," in *Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS 2007*, ser. Electronic Notes in Theoretical Computer Science, M. Fiore, Ed., vol. 173. Elsevier, 2007, pp. 121–137.

[13] C. McCreesh, P. Prosser, and J. Trimble, "The Glasgow Subgraph Solver: Using constraint programming to tackle hard subgraph isomorphism problem variants," in *Graph Transformation - 13th International Conference, ICGT 2020*, 2020, pp. 316–324.

[14] ——, "A partitioning algorithm for maximum common subgraph problems," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 712–719. [Online]. Available: https://doi.org/10.24963/ijcai.2017/99

[15] B. Archibald, K. Burns, C. McCreesh, and M. Sevegnani, "Practical bigraphs via subgraph isomorphism," in *Principles and Practice of Constraint Programming - 27th International Conference, CP 2021*, 08 2021.

[16] F. Baader and T. Nipkow, *Term rewriting and all that.* Cambridge University Press, 1998, pp. 1–301.

[17] J. W. Klop, "Chapter 1: Term rewriting systems," in *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaurn, Eds. Oxford University Press, 1992, pp. 1–116.

[18] J. Avenhaus and K. Madlener, "Term rewriting and equational reasoning," in *Formal Techniques in Artificial Intelligence*. Elsevier, 1990.

[19] J. M. Li and A. W. Appel, "Deriving efficient program transformations from rewrite rules," *Proc. ACM Program. Lang.*, vol. 5, no. ICFP, aug 2021. [Online]. Available: https://doi.org/10.1145/3473579

[20] J. Meseguer, "Twenty years of rewriting logic," *The Journal of Logic and Algebraic Programming*, vol. 81, no. 7, pp. 721–781, 2012, rewriting Logic and its Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1567832612000707

[21] P. Klint, "Quick introduction to term rewriting," Centrum Wiskunde & Informatica, Tech. Rep., 2007. [Online]. Available: https://homepages.cwi.nl/~daybuild/daily-books/extraction-transformation/term-rewriting/term-rewriting.html

[22] E. M. Clarke, *The Birth of Model Checking*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–26. [Online]. Available: https://doi.org/10.1007/978-3-540-69850-0_1

[23] U. Schöning, "Graph isomorphism is in the low hierarchy," *Journal of Computer and System Sciences*, vol. 37, no. 3, pp. 312–323, 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022000088900104

[24] B. Archibald, G. Kulcsár, and M. Sevegnani, "A tale of two graph models: a case study in wireless sensor networks," *Formal Aspects of Computing*, vol. 33, 08 2021.

[25] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds., *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*. USA: World Scientific Publishing Co., Inc., 1999.

[26] A. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova, *Modelling and Analysis Using GROOVE*, ser. CTIT Technical Report Series. Netherlands: Centre for Telematics and Information Technology (CTIT), Apr. 2010, no. TR-CTIT-10-18.

[27] G. Taentzer, "AGG: A graph transformation environment for modeling and validation of software," in *Applications of Graph Transformations with Industrial Relevance*, J. L. Pfaltz, M. Nagl, and B. Böhlen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 446–453.

[28] D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehrer, M. Ohrndorf, and M. Tichy, "Henshin: A usability-focused framework for EMF model transformation development," in *Graph Transformation*, J. de Lara and D. Plump, Eds. Cham: Springer International Publishing, 2017, pp. 196–208.

[29] O. Kniemeyer, "Design and implementation of a graph grammar based language for functional-structural plant modelling," 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:33527212

[30] D. Hart and B. Goertzel, "Opencog: A software framework for integrative artificial general intelligence," in *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*. NLD: IOS Press, 2008, p. 468–472.

[31] H. Ehrig, K. Ehrig, U. Golas, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation*, 01 2006, vol. XIV.

[32] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC Bioinformatics*, vol. 14, no. SUPPL7, Apr. 2013.

[33] G. Damiand, C. Solnon, C. de la Higuera, J.-C. Janodet, and Émilie Samuel, "Polynomial algorithms for subisomorphism of nd open combinatorial maps," *Computer Vision and Image Understanding*, vol. 115, no. 7, pp. 996–1010, 2011, special issue on Graph-Based Representations in Computer Vision. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314211000816

[34] A. C. Murray and B. Franke, "Compiling for automatically generated instruction set extensions," in *IEEE/ACM International Symposium on Code Generation and Optimization*, 2012.

[35] M. Dalla Preda and V. Vidali, "Abstract similarity analysis," *Electronic Notes in Theoretical Computer Science*, vol. 331, pp. 87–99, 2017, proceedings of the Sixth Workshop on Numerical and Symbolic Abstract Domains (NSAD 2016). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066117300087

[36] P. Damaschke, "Induced subgraph isomorphism for cographs in np-complete," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:11499871

[37] C. Solnon, "Experimental evaluation of subgraph isomorphism solvers," in *Graph-Based Representations in Pattern Recognition*, D. Conte, J.-Y. Ramel, and P. Foggia, Eds. Cham: Springer International Publishing, 2019, pp. 1–13.

[38] C. McCreesh, P. Prosser, C. Solnon, and J. Trimble, "When subgraph isomorphism is really hard, and why this matters for graph databases," *J. Artif. Intell. Res.*, vol. 61, pp. 723–759, 2018.

[39] M. Miculan and M. Peressotti, "A CSP implementation of the bigraph embedding problem," *CoRR*, vol. abs/1412.1042, 2014.

[40] B. Archibald, M. Calder, and M. Sevegnani, "Conditional bigraphs," in *Graph Transformation - 13th International Conference, ICGT 2020*, ser. Lecture Notes in Computer Science, F. Gadducci and T. Kehrer, Eds., vol. 12150. Springer, 2020, pp. 3–19.

[41] S. Benford, M. Calder, T. Rodden, and M. Sevegnani, "On lions, impala, and bigraphs: Modelling interactions in physical/virtual spaces," *ACM Trans. Comput.-Hum. Interact.*, vol. 23, no. 2, pp. 9:1–9:56, 2016.

[42] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. USA: Elsevier Science Inc., 2006.

[43] J. Beck, P. Prosser, and R. Wallace, "Toward understanding variable ordering heuristics for constraint satisfaction problems," 01 2003.

[44] S. Chandra, C. S. Gordon, J.-B. Jeannin, C. Schlesinger, M. Sridharan, F. Tip, and Y. Choi, "Type inference for static compilation of javascript," ser. OOPSLA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 410–429. [Online]. Available: https://doi.org/10.1145/2983990.2984017

[45] L. Popovic, A. Côté, M. Gaha, F. Nguewouo, and Q. Cappart, "Scheduling the Equipment Maintenance of an Electric Power Transmission Network Using Constraint Programming," in *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), C. Solnon, Ed., vol. 235. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 34:1–34:15. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2022.34

[46] G. Da Col and E. C. Teppan, "Industrial-size job shop scheduling with constraint programming," *Operations Research Perspectives*, vol. 9, p. 100249, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716022000215

[47] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[48] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000437028090051X

[49] B. Archibald, F. Dunlop, R. Hoffmann, C. McCreesh, P. Prosser, and J. 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. [Online]. Available: https://doi.org/10.1007/978-3-030-19212-9_2

[50] G. Katsirelos and F. Bacchus, "Generalized nogoods in csps," in *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'05. AAAI Press, 2005, p. 390–396.

[51] J. Lee, C. Schulte, and Z. Zhu, "Increasing nogoods in restart-based search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10437

[52] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal, "Recording and minimizing nogoods from restarts," *JSAT*, vol. 1, pp. 147–167, 05 2007.

[53] S. Zampelli, Y. Deville, and C. Solnon, "Solving subgraph isomorphism problems with constraint programming," *Constraints*, vol. 15, no. 3, pp. 327–353, 2010. [Online]. Available: https://doi.org/10.1007/s10601-009-9074-3

[54] G. Audemard, C. Lecoutre, M. S. Modeliar, G. Goncalves, and D. C. Porumbel, "Scoring-based neighborhood dominance for the subgraph isomorphism problem," in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, 2014. [Online]. Available: https://doi.org/10.1007/978-3-319-10428-7_12

[55] C. Solnon, "Alldifferent-based filtering for subgraph isomorphism," *Artificial Intelligence*, vol. 174, no. 12, pp. 850–864, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370210000718

[56] C. McCreesh and P. Prosser, "A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs," in *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Proceedings*, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-23219-5_21

[57] E. Højsgaard and A. J. Glenstrup, "The BPL tool: A tool for experimenting with bigraphical reactive systems," The IT University of Copenhagen, Tech. Rep., 2011.

[58] A. J. Glenstrup, T. C. Damgaard, L. Birkedal, and E. Højsgaard, "An implementation of bigraph matching," IT University of Copenhagen, Tech. Rep. TR-2010-135, 2010.

[59] B. Archibald, M. Calder, and M. Sevegnani, "Probabilistic bigraphs," *Form. Asp. Comput.*, vol. 34, no. 2, sep 2022. [Online]. Available: https://doi.org/10.1145/3545180

[60] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003.*, ser. Lecture Notes in

Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518.

[61] M. H. Liffiton and J. C. Maglalang, "A cardinality solver: More expressive constraints for free - (poster presentation)," in *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, ser. Lecture Notes in Computer Science, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Springer, 2012, pp. 485–486. [Online]. Available: https://doi.org/10.1007/978-3-642-31612-8_47

[62] C. R. Codel, J. Avigad, and M. Heule, "Verified encodings for sat solvers," *2023 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 141–151, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:265859217

[63] A. Chiapperini, M. Miculan, and M. Peressotti, "Computing (optimal) embeddings of directed bigraphs," vol. 221, 2022, p. 102842. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642322000752

[64] C. Prud'homme and J.-G. Fages, "Choco-solver: A java library for constraint programming," *Journal of Open Source Software*, vol. 7, no. 78, p. 4708, 2022. [Online]. Available: https://doi.org/10.21105/joss.04708

[65] D. Grzelak, "Bigraph framework user manual," Technische Universität Dresden, Germany, Tech. Rep., 2023. [Online]. Available: https://bigraphs.org/products/bigraph-framework/docs/

[66] A. Gassara, I. B. Rodriguez, M. Jmaiel, and K. Drira, "Executing bigraphical reactive systems," *Discret. Appl. Math.*, vol. 253, pp. 73–92, 2019.

[67] M. A. Hannachi, I. Bouassida Rodriguez, K. Drira, and S. E. Pomares Hernandez, "Gmte: A tool for graph transformation and exact/inexact graph matching," in *Graph-Based Representations in Pattern Recognition*, W. G. Kropatsch, N. M. Artner, Y. Haxhimusa, and X. Jiang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–80.

[68] S. Szabo and B. Zavalnij, "Reducing hypergraph coloring to clique search," *Discrete Applied Mathematics*, vol. 264, pp. 196–207, 2019, combinatorial Optimization: between Practice and Theory. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X1830475X

[69] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy, "Symmetry-breaking predicates for search problems," in *Proceedings of the Fifth International Conference on*

Principles of Knowledge Representation and Reasoning, ser. KR'96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, p. 148–159.

[70] M. Sevegnani, "Bigraphs with sharing and applications in wireless networks," Ph.D. dissertation, University of Glasgow, 2012.

[71] D. Grohmann, "Security, cryptography and directed bigraphs," in *Graph Transformations*, H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 487–489.

[72] G. Bacci, D. Grohmann, and M. Miculan, "Bigraphical models for protein and membrane interactions," *Electronic Proceedings in Theoretical Computer Science*, vol. 11, pp. 3–18, nov 2009. [Online]. Available: https://doi.org/10.4204%2Feptcs.11.1

[73] F. Burco, M. Miculan, and M. Peressotti, "Towards a formal model for composable container systems," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 173–175. [Online]. Available: https://doi.org/10.1145/3341105.3374121

[74] A. Chiapperini, M. Miculan, and M. Peressotti, "Computing embeddings of directed bigraphs," in *Graph Transformation - 13th International Conference, ICGT 2020*, ser. Lecture Notes in Computer Science, F. Gadducci and T. Kehrer, Eds., vol. 12150. Springer, 2020, pp. 38–56.

[75] B. Archibald, K. Burns, C. McCreesh, and M. Sevegnani, "Practical Bigraphs via Subgraph Isomorphism – Glasgow Subgraph Solver Bigraph Source Code." [Online]. Available: https://github.com/ciaranm/glasgow-subgraph-solver/tree/gbs

[76] K. Burns, "Test Instances and Results: Efficient and Scalable Algorithms for Bigraph Matching (Ph.D Thesis)," Jan. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10578633

[77] R. Hoffmann, C. McCreesh, S. N. Ndiaye, P. Prosser, C. Reilly, C. Solnon, and J. 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*, 2018, pp. 298–315.

[78] O. H. Jensen and R. Milner, "Bigraphs and transitions," *SIGPLAN Not.*, vol. 38, no. 1, p. 38–49, jan 2003. [Online]. Available: https://doi.org/10.1145/640128.604135

[79] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865597000603

[80] R. Milner, "Bigraphs: a model for mobile agents," University Lecture, 2008.

[81] Y. Wang, I. Dillig, S. K. Lahiri, and W. R. Cook, "Verifying equivalence of database-driven applications," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3158144

[82] J. Rau, D. Richerby, and A. Scherp, "Computing k-bisimulations for large graphs: A comparison and efficiency analysis," in *Graph Transformation*, M. Fernández and C. M. Poskitt, Eds. Cham: Springer Nature Switzerland, 2023, pp. 223–242.

[83] M. Kemertas and T. Aumentado-Armstrong, "Towards robust bisimulation metric learning," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., 2024.

[84] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 01 2008, vol. 26202649.

[85] C. Mccreesh, S. N. Ndiaye, P. Prosser, and C. Solnon, "Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems," in *22nd International Conference on Principles and Practice of Constraint Programming (CP)*, ser. Lecture Notes in Computer Science, vol. 9892. Toulouse, France: Springer, Sep. 2016, pp. 350–368. [Online]. Available: https://hal.science/hal-01331298

[86] R. Hoffmann, C. McCreesh, and C. Reilly, "Between subgraph isomorphism and maximum common subgraph," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11137

[87] J. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures," *Journal of computer-aided molecular design*, vol. 16, pp. 521–33, 08 2002.

[88] J. Trimble, *Partitioning Algorithms for Induced Subgraph Problems*. University of Glasgow, 2023. [Online]. Available: https://books.google.co.uk/books?id=1nnOzwEACAAJ

[89] A. Calabrese, L. Cardone, S. Licata, M. Porro, and S. Quer, "A web scraping algorithm to improve the computation of the maximum common subgraph," in *Proceedings of the 18th International Conference on Software Technologies - ICSOFT*, INSTICC. SciTePress, 2023, pp. 197–206.

[90] R. Milner, "Local bigraphs and confluence: Two conjectures: (extended abstract)," *Electr. Notes Theor. Comput. Sci.*, vol. 175, no. 3, pp. 65–73, 2007.

[91] J. Trimble, "NetworkX — "Add max common induced subgraph" branch." [Online]. Available: https://github.com/networkx/networkx/pull/4221

[92] K. Burns and J. Trimble, "Maximum Common Bigraph Source Code." [Online]. Available: https://github.com/KyleBurns/MaximumCommonBigraph/tree/main

[93] P. Fua and K. Lis, "Comparing python, go, and C++ on the n-queens problem," *CoRR*, vol. abs/2001.02491, 2020. [Online]. Available: http://arxiv.org/abs/2001.02491

[94] W. D. Frohlingsdorf, "RPOs in place graphs of epimorphic bigraphs with sharing," 2016. [Online]. Available: https://www.jacktex.eu/research/material/rpo_bigraph.pdf

[95] T. Damgaard, A. Glenstrup, L. Birkedal, and R. Milner, "An inductive characterization of matching in binding bigraphs," *Formal Aspects of Computing - FAC*, vol. 25, 01 2010.

[96] B. Archibald, M. Calder, and M. Sevegnani, "Conditional bigraphs," in *Graph Transformation: 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25–26, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 3–19. [Online]. Available: https://doi.org/10.1007/978-3-030-51372-6_1

[97] M. Soos and K. S. Meel, "Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting," in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 1 2019.

[98] G. Pesant, C.-G. Quimper, and H. Verhaeghe, "Practically uniform solution sampling in constraint programming," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 19th International Conference, CPAIOR 2022, Los Angeles, CA, USA, June 20-23, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 335–344. [Online]. Available: https://doi.org/10.1007/978-3-031-08011-1_22