Math Meth Oper Res (2009) 69:281–296
DOI 10.1007/s00186-008-0259-0

ORIGINAL ARTICLE

A minimization version of a directed subgraph homeomorphism problem

Janina A. Brenner · Sándor P. Fekete · Jan C. van der Veen

Received: 29 September 2006 / Accepted: 3 September 2007 / Published online: 20 September 2008

Abstract We consider a special case of the directed subgraph homeomorphism or topological minor problem, where the host graph has a specific regular structure. Given an acyclic directed pattern graph, we are looking for a host graph of minimal height which still allows for an embedding. This problem has applications in compiler design for certain coarse-grain reconfigurable architectures. In this application domain, the task is to *simultaneously* schedule, bind and route a so-called *data-flow* graph, where vertices represent operations and arcs stand for data dependencies between the operations, given an orthogonal grid structure of reconfigurable processing elements (PEs) that have restricted communication abilities. We show that the problem of simultaneously scheduling, binding and routing is NP-complete by describing a logic engine reduction from NAE-3-SAT. This result holds even when the input graph is a directed tree with maximum indegree two. We also give a $|V|^{3/2}$-approximation algorithm.

J. A. Brenner's research supported by the DFG Research Center MATHEON "Mathematics for key technologies".
J. C. van der Veen's research supported by DFG Focus Program 1148, "Reconfigurable Architectures", Grants FE 407/8-1 and FE 407/8-2.

J. A. Brenner (✉)
Institute of Mathematics, Berlin Institute of Technology,
Straße des 17. Juni 136, 10623 Berlin, Germany
e-mail: brenner@math.tu-berlin.de

S. P. Fekete · J. C. van der Veen
Algorithms Group, Department of Computer Science, Braunschweig University of Technology,
Mühlenpfordtstr. 23, 38106 Braunschweig, Germany
e-mail: s.fekete@tu-bs.de

J. C. van der Veen
e-mail: j.van-der-veen@tu-bs.de

Keywords Subgraph homeomorphism · Topological minor · NP-completeness · Approximation algorithms · Reconfigurable computing

1 Introduction

In the subgraph homeomorphism problem, we are given a large *host* graph H, and a smaller *pattern* graph G. The problem is to determine whether G is *homeomorphic* to a subgraph of H, i.e. to decide whether there exists an injective function ϕ that maps the vertices of G to a subset of vertices in H, and the edges $\{u, v\} \in G$ to vertex-disjoint paths from $\phi(u)$ to $\phi(v)$.

This problem has been studied under different names in the mathematical literature. Namely, G is called a *topological minor* of H if and only if there exists a subgraph homeomorphism from G to H. Sometimes, a topological minor of H is also called a *subdivision* of a subgraph of H. However, previous work has focused mainly on problems where the *pattern* graph is fixed. Examples include the Kuratowski graphs $K_{3,3}$ and K_5, cliques, or the set of two disjoint edges (Asano 1985, Kühn and Osthus 2002, LaPaugh and Rivest 1978, Mader 1998).

Fortune et al. (1980) proved that the *directed* version of the subgraph homeomorphism problem is NP-complete, even if the pattern graph is fixed and therefore not part of the input. Furthermore, they characterize the set of pattern graphs for which the *fixed* directed subgraph homeomorphism problem is solvable in polynomial time. However, to the best of our knowledge, the problem setting in which the *host* graph is fixed or of regular structure, while the pattern graph is given as the input graph, has not been studied so far.

In this paper, we study a special case of the directed subgraph homeomorphism problem. Driven by an application in compiler design, we assume a host graph with a fixed regular structure, while the *pattern* graph is an arbitrary acyclic input graph. In the application setting, we are given a so-called data-flow graph representing an application that is to be executed on a network of *processing elements* (PEs). The task of the compiler is to find a time-optimal mapping of the vertices of the data-flow graph to PEs and execution time steps, while respecting data dependencies in the graph and routing capabilities of the network. We target a specific *coarse-grain* reconfigurable architecture model proposed by Oppold, Schweizer, Kuhn, and Rosenstiel in the Configurable Reconfigurable Core (CRC) project (Oppold et al. 2004).

In our application domain, scheduling of tasks and routing of results are traditionally treated as two separate optimization problems. Intuitively, it is clear that only solving both problems simultaneously leads to optimal application performance. We will show that this *simultaneous* scheduling, binding and routing (SSBR) problem is a computationally hard mathematical optimization problem.

We review the architecture model and the problem definition in Sect. 2, where we also draw the connection to the directed subgraph homeomorphism characterization. In Sect. 3, we prove that the SSBR problem is NP-complete by a *logic engine* reduction from a variant of 3-satisfiability. We show that the problem remains NP-hard even for the class of directed trees. In Sect. 4, we describe a first $|V|^{3/2}$-approximation algorithm.

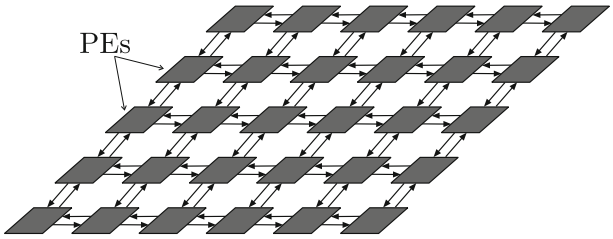


Fig. 1 PEs with communication arcs

2 Problem characterization

In this section, we elaborate on the *Simultaneous Scheduling, Binding and Routing* (*SSBR*) problem. We detail the application domain in Sect. 2.1, and draw the connection to the more mathematical characterization as a directed subgraph homeomorphism problem in Sect. 2.2.

2.1 Compiler design for coarse-grain reconfigurable architectures

Today, processors on the market vary from standard multi-purpose processors as in personal computers to custom-tailored VLSI chips used in all kinds of appliances. Originally designed to reduce the development cost of VLSI chips, modern *field-programmable gate arrays* (FPGAs) have emerged as new and very versatile computing devices. In *coarse-grain* reconfigurable FPGA architectures, the smallest reconfigurable unit is a so-called *processing element* that is capable of executing a certain reconfigurable arithmetic or logic operation. These PEs are connected by a typically fixed communication infrastructure.

In this work, we consider the highly-parameterizable coarse-grain reconfigurable architecture model developed by Oppold et al. in the Configurable Reconfigurable Core (CRC) project (Oppold et al. 2004). In one realization of their model, the PEs can either be reconfigured and execute an arbitrary unary or binary operation, all in one time step, or store an intermediate result. The PEs are arranged on the processor fabric to form a 2-dimensional orthogonal grid, and connected through a nearest neighbor interconnect network that provides connections between horizontally or vertically neighboring PEs (see Fig. 1). Data may be exchanged between directly connected PEs in both directions inbetween two time steps. Communication between non-neighboring PEs takes several time steps, because data can only be forwarded over one connection wire per time step. In the meantime, the information has to be buffered on the PEs it traverses.

On this reconfigurable architecture, we are to simultaneously schedule, bind and route an application given as a directed acyclic data-flow graph. The vertices of this graph correspond to the operations, while edges model data dependencies. An edge from operation u to operation v indicates that v depends on u, i.e. v needs the result value of operation u as an input value. As all operations are equal to the effect that

they can all be processed by any PE in any time step and require the same processing time, we can neglect the specific operation types.

A solution of the SSBR problem is a feasible mapping of the application to the architecture. Such a mapping, or *binding*, designates a PE and a time step for the execution of each operation. In order to be feasible, it must also identify the routing paths for results from their origins to the places in which they are needed as input values. Due to the technical restrictions of our architecture, a PE over which a result is routed is considered to be occupied with buffering during the respective time steps, and thus no other operation can be executed or stored on this PE at the same time. The routing aspect is therefore significant in the solving process of an SSBR instance. An *optimal* solution of the SSBR problem is a feasible mapping requiring a minimal number of time steps.

2.2 Directed subgraph homeomorphism problem

The SSBR problem is a special case of the *Directed Subgraph Homeomorphism (DSH)* problem. Here, the input data-flow graph of an SSBR instance constitutes the pattern graph G, whereas the host graph H is not explicitly given. It is a regular graph whose size is determined by the architecture dimensions and an upper bound on the number of time steps. The structure of the host graph is identical for every instance of the SSBR problem.

As described in Sect. 2.1, a feasible solution to the SSBR problem assigns two attributes to each vertex of the data-flow graph: The PE where and the time step when it is to be executed. This corresponds to a mapping to an integer point in a 3-dimensional grid with base $N \times M$ (the size of the PE architecture) and height S (the number of time steps). Hence, when defining the vertex set of H as this grid, the vertex image $\phi(V(G))$ corresponds to the PE and time step assignment for all jobs.

The routing of results between interdependent operations is modeled by paths connecting the respective vertex images. Recall that between two time steps, data can be transmitted from a PE to all of its direct neighbors or be saved in its own memory cell. Figure 2 shows the resulting outedges of a grid vertex in H.

Consider an edge (u, v) in the data-flow graph. The homeomorphic image of (u, v) in H is a path from $\phi(u)$ to $\phi(v)$ that does not intersect any other image path, except possibly at the start or end points. In terms of the SSBR problem, this respects the fact that every vertex on the image path is "occupied" with buffering the result of u. Since edges in H point from a vertex representing a PE exactly to those representing PEs reachable from it on the next time level, the image path constitutes a feasible routing path. Figure 3 shows the graph H for $N = 6$, $M = 3$ and $S = 4$.

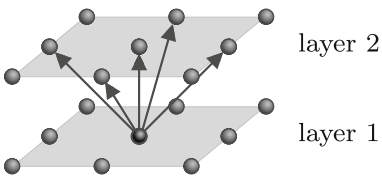


Fig. 2 The outedges of a grid vertex

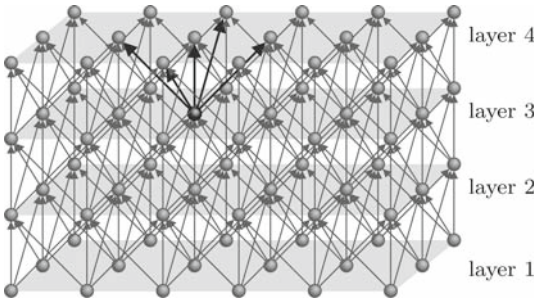


Fig. 3 The host graph H of size $6 \times 3 \times 4$

Due to the representation as a subgraph homeomorphism problem, we also speak of a solution to the SSBR problem as an *embedding* of the data-flow graph in the grid H.

The directed subgraph homeomorphism problem is shown to be NP-complete in general by Fortune et al. (1980). While this result does not imply that the problem is still NP-hard for the special regular host graph that we have in this case, we present a proof in the next section.

3 Computational complexity

In this section, we prove that the SSBR problem is NP-complete, even when restricting the input graph to directed trees. For the proof, we use the directed subgraph homeomorphism interpretation described above. Note that the DSH characterization states the *decision* problem for a fixed number of time steps. Nevertheless, proving that this problem is NP-hard suffices for showing NP-completeness of the associated minimization problem. Section 3.1 introduces the reduction technique we use for the proofs of both the general case, presented in Sect. 3.2, and the tree case, presented in Sect. 3.3.

3.1 A logic engine for proving NP-completeness

We employ a general tool for establishing NP-hardness of graph representation problems called the *logic engine*, which we briefly describe in this section. It was first used by Bhatt and Cosmadakis (1987) to show that it is NP-complete to determine whether a tree of maximum degree four can be embedded in a planar orthogonal grid, where tree vertices must be positioned at grid vertices and tree edges must occupy unit length grid edges.

The functionality of the logic engine is to offer a reduction from the Not-All-Equal-3-SAT (NAE-3SAT) problem, whose instances consist of c 3-element clauses on l literals. An instance is solvable iff there is a truth assignment such that every clause contains at least one true and one false literal. To determine whether such a truth assignment exists is known to be NP-hard (Garey and Johnson 1979).

For every instance of the NAE-3SAT problem, we define a gadget graph that can be embedded if and only if the NAE-3SAT instance is solvable. Very roughly, this gadget

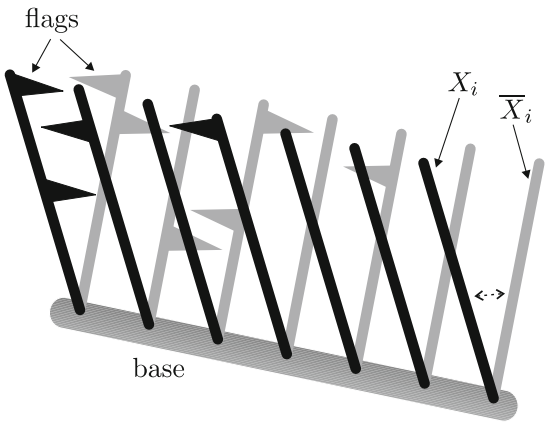


Fig. 4 The basic logic engine

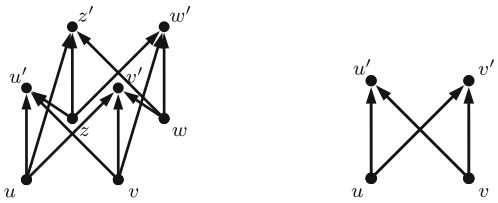
consists of a horizontal *bar* to which l pairs of rigid *flag poles* are attached, such that the only latitude in the embedding is the ability to exchange partner poles. Attached to the flag poles are small *flags* that can be directed parallel to the base. The small distance between neighboring poles forces flags to be positioned in distinct gaps. See Fig. 4 for an illustration.

The NAE-3SAT clauses are encoded by the flag positions as follows. Each literal is represented by a pair of flag poles, w.l.o.g. the true variable X_i by the front pole, and its complement $\overline{X}_i$ by the back pole. Each clause C_k, for $k = 1, \ldots, c$, is encoded on a distinct height level. First, a flag is attached to each pole and height level. Four auxiliary flags are added to the four outer poles (standing for variables X_1, $\overline{X}_1$, X_l, and $\overline{X}_l$) on each height level. For each clause C_k, we then remove exactly three flags from the corresponding height level. For each positive variable X_i contained in the clause, we remove a flag from the *front* pole standing for X_i, and for each negated variable $\overline{X}_j$, we remove a flag from the respective pole in the *back* row.

To find a feasible configuration of this gadget, i.e. one in which no flags overlap, partner poles can be exchanged and flags can be directed to the left or right. If we interpret a variable to be set TRUE if the pole standing for its true literal is placed in the front line, and FALSE otherwise, then it is easy to see that a feasible configuration of the logic engine directly leads to a valid truth assignment for the NAE-3SAT instance. Thus, NP-completeness is established for the geometric problem if it lies in NP. The following proposition has been shown by Bhatt and Cosmadakis (1987).

Proposition 3.1 *Each feasible configuration of the logic engine transfers to a YES assignment of the corresponding NAE-3SAT instance in polynomial time.*

We forbear from a formal proof, and give an intuition instead. Why is the variable assignment obtained from the pole positions feasible for the NAE-3SAT problem? Consider a clause C_k, which is encoded on a certain height level. Assume that the four outer flags are pointed outwards. Then, there are exactly $l - 1$ gaps between the l flag poles in each line. Remember that we deleted three out of the $2l$ flags on every height level. Since at most $l - 1$ flags fit in the gaps of each line without overlapping, at least one flag must be missing per line. A hole in the front line means that the corresponding



(a) An embedding of the *box* graph Q **(b)** The *flip* graph F

Fig. 5 Two basic modules

literal is true, whereas a hole in the back line means that a literal is false. Thus, for the truth assignment, this means that at least one literal must be true to have enough space in the front line, and at least one literal must be false in order to have enough room in the back line.

3.2 The gadget for general instances

Before proving that the SSBR problem is NP-complete even for trees with maximum indegree two, we shortly present the gadget for the general case in this section. A detailed description can be found in Brenner (2005).

In order to employ the technique described in the previous section, we need to define for every NAE-3SAT instance I a data-flow graph G_I whose embedding in H has the logic engine properties. Our basic idea is to construct G_I such that every vertex lies on a longest path. Then, we seek to embed G_I in a grid whose longest path has equal length, i.e. we use the number of vertices on a longest path in G_I as upper bound for the number of time steps. This fixes the schedule of all vertices beforehand. Intuitively, this reduces the freedom of the embedding. Assume that the remaining two dimensions of H are large enough; here, a grid of size $(3l + 1) \times 4$ suffices.

We define the two basic modules of which we compose the gadget graph through Fig. 5a, b. The following two lemmas are easily checked to be true.

Lemma 3.2 *There is at most one feasible embedding of the* box *graph in two time steps if the embedding of the layer 1 vertices* (*labeled* u, v, w, z *in Fig.* 5a) *is fixed.*

Lemma 3.3 *There are exactly two distinct embeddings of the* flip *graph in two time steps if the layer 1 vertices* (*labeled* u, v *in Fig.* 5b) *are fixed to two neighboring vertices on the first time layer.* u' *and* v' *have to be mapped to the same two vertices on the second time layer, with arbitrary allocation among the two.*

Very roughly, the graph G_I corresponding to the NAE-3SAT instance I with c clauses on l literals is constructed as follows. We build the *base* of the logic engine by merging a row of $3l - 2$ box graphs. We construct pairs of flag poles on top of the 1st, 4th, 7th, …, and $(3l - 2)$th base boxes. Each flag pole consists of a pile of boxes riddled with c double flips, such that every height segment between two flips corresponds to one clause. Flags are modeled by boxes attached to the sides of the

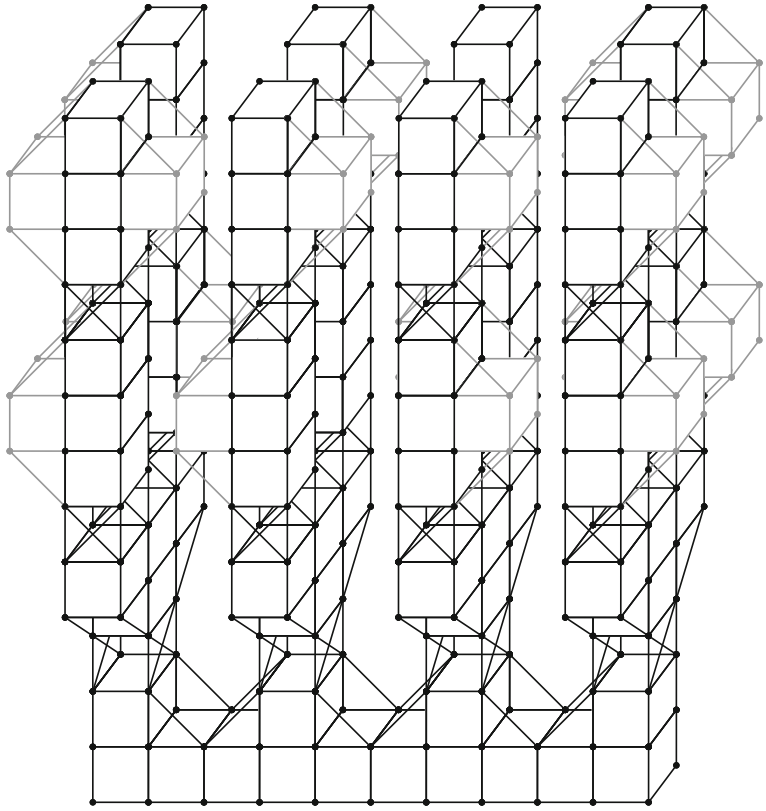


Fig. 6 The gadget graph corresponding to the NAE-3SAT instance $(X_1 \vee X_2 \vee X_4) \wedge (\overline{X}_1 \vee \overline{X}_3 \vee X_4)$

flag poles. Due to the built in flips, partner poles can be exchanged and flags can be flipped individually to the left or right.

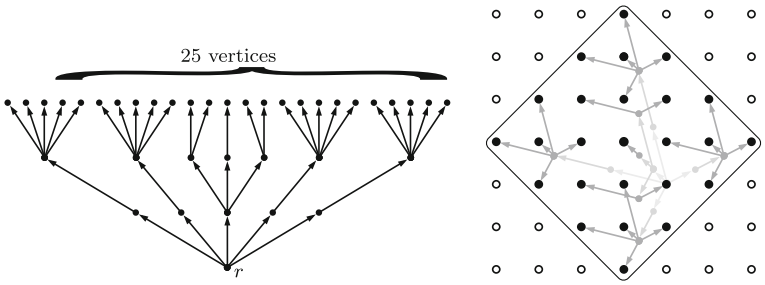
A feasible embedding of the resulting gadget graph for the instance $(X_1 \vee X_2 \vee X_4) \wedge (\overline{X}_1 \vee \overline{X}_3 \vee X_4)$ is shown in Fig. 6. It represents the solution $X_1 = X_3 = X_4 = \text{TRUE}$ and $X_2 = \text{FALSE}$, so the partner poles associated to X_2 have been exchanged. For a better visibility, boxes are represented by nontransparent cubes.

Theorem 3.4 *The directed subgraph homeomorphism problem remains NP-complete if we restrict the host graph to a regular graph H as defined in Sect. 2.2.*

We refer the reader to Brenner (2005) for a proof, and restrict ourselves to the stronger proof for the following special case in this work.

3.3 The gadget for tree instances

We prove in this section that the SSBR problem remains NP-complete when restricting the input graph to directed trees with maximum indegree two. Again, we design a logic engine graph T_I for every NAE-3SAT instance I. However, on account of T_I being



(a) The graph **(b)** An embedding

Fig. 7 The diamond D

a tree, the construction is remarkably different. Here, the flag poles are not oriented parallel to the time axis, but in two horizontal directions. We often use a projection to the grid base for visualization.

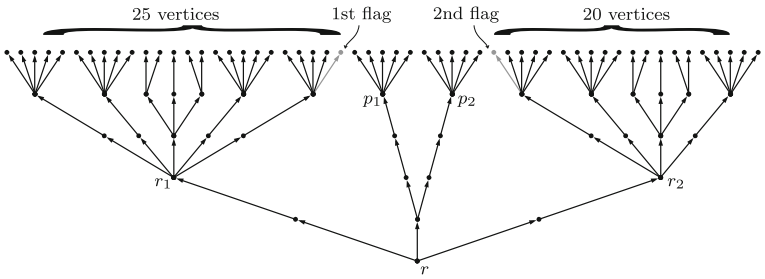
The main idea of the proof is the following. As before, we fix the schedule by placing every vertex on a longest path and setting the upper time bound respectively. Also, assume that the base grid dimensions are large enough. The main twist is the following simple observation. For any vertex $w \in H$, the number of vertices reachable from w within k time steps is at most $(k+1)^2 + k^2$. Thus, if a vertex $v \in T_I$ has $(k+1)^2+k^2$ children in the k-th generation, i.e. vertices that are connected to v through a path of length k, then we know exactly to which set of vertices in H they must be mapped. Of course, this uses the fact that we have fixed the schedule beforehand and therefore cannot prolongate paths. We use this observation frequently throughout the rest of this section, in which we describe the construction of T_I.

Flag Segments. We construct the flag poles with two different modules that we call *flag* segments and *auxiliary pole* segments. Before we define the flag segments, we detail our basic idea on the example of the *diamond* graph D defined in Fig. 7a. An embedding of D from the time perspective is depicted in Fig. 7b.

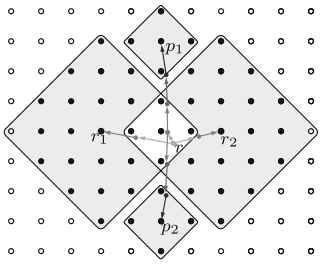
Lemma 3.5 *If the* diamond *graph D has to be embedded into H in four time steps, all layer 4 vertices in D have to be mapped to the circle centered at r with Manhattan radius 3.*

Proof D contains 25 vertices in layer 4, which are all connected to r by paths of length 3. Hence, they can only be mapped to vertices at a distance of ≤ 3 to the image of r. Since exactly 25 vertices satisfy this condition, all of them have to be used. □

The flag segment S_f is defined in Fig. 8a. It mainly consists of two diamond graphs connected by two paths of length 2. To enable a feasible embedding, five vertices from the union of both diamonds have been removed. The use of the additional two small subgraphs will become clear later. The number of flags f is encoded with the gray vertices. If the segment holds two flags ($f = 2$), we take the whole graph, while in case of one ($f = 1$) or zero ($f = 0$) flags, one or both of the gray vertices and their incident edges are cut off. Figure 8b shows an embedding of the 6th layer of S_0,



(a) The graph



(b) An embedding of S_0

Fig. 8 The flag segment S_f

pointing out the mapping of r and the connecting paths. The two empty vertices in the shaded area indicate possible positions for the missing flags.

Lemma 3.6 *If the flag segment graph S_f has to be embedded into H in six time steps, then the image of the set of layer 6 vertices is determined by the placement of r, apart from rotation and the positioning of the $2 - f$ spaces that result from missing flags.*

Proof It is not hard to verify that the layer 6 vertices of S_f have to be bound to the regions indicated in Fig. 8b: To map the two diamonds, we have to stretch out the two paths connecting them in order to achieve the minimum overlap of 5 vertices of the 6th-layer regions. Note that there is no other relative position for which the overlapping is ≤ 7. Hence, even though 7 vertices are missing in the case of the graph S_0, this is the only possible configuration. Then, the embedding of the remaining two small diamonds is clear. □

Poles. For the construction of the poles, we define auxiliary pole segments which are used to connect the resembling, but bigger flag segments. Figure 9a defines the auxiliary pole segment graph A.

Lemma 3.7 *If the auxiliary pole segment A has to be embedded into H in four time steps, then the usage of vertices in layer 4 is fully determined by the placement of the root r.*

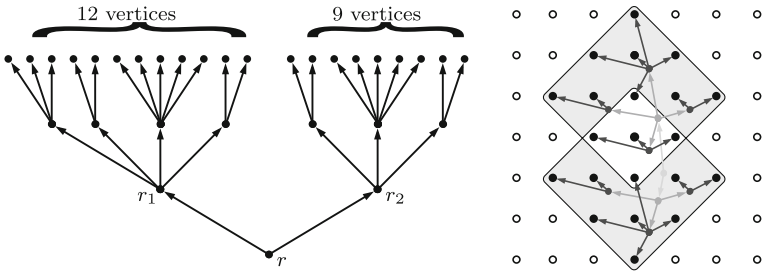


Fig. 9 The auxiliary pole segment A

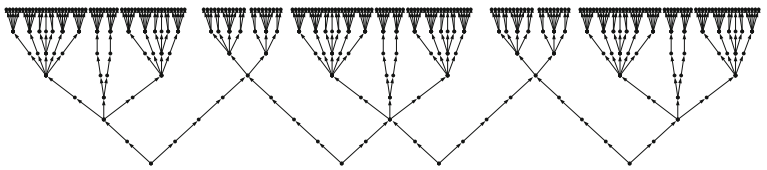


Fig. 10 A pole pair P_1

Proof Figure 9b shows the regions Q_1 and Q_2 reachable from r_1 and r_2 in two time steps. Each of them comprises 13 vertices, with a smallest possible overlap of 5 vertices. Hence, the whole region $Q_1 \cap Q_2$ has to be used by the 21 layer 4 vertices. □

A pole pair P_c is now created by connecting $2c+1$ flag segments S_f and $2c$ auxiliary pole segments A in alternating order; where c is the number of clauses in I. Each pair of neighboring segments is connected by two paths of length 2 and 4, respectively, both starting in the same vertex. The longer path ends in the root of the auxiliary pole segment, and the shorter ends in the root of the S_f graph. A resulting pole pair for $c = 1$ is depicted in Fig. 10.

Lemma 3.8 *If the pole pair P_c has to be embedded into H in 8 time steps, then the placement of the tilted squares in layer 8 is fixed for all segments S_f and A contained. The resulting embedding is a alternating line-up of the S_f and A graphs as in Fig.* 11.

Proof Consider the positioning of two neighboring segments A and S_f. Since the highest layer of A is completely rigid, and S_f leaves at most 2 vertices free within the tilted squares, the layer 8 regions of A and S_f can only overlap by at most two vertices. As the roots of neighboring segments have to lie at a distance of at most 6 from each other, Fig. 11 shows the only possible relative position. This configuration allows the neighboring segments to the other sides of S_f and A to be placed accordingly. □

Base. We complete the framework of the gadget by connecting l pole pairs P_c at their central flag modules; where l is the number of literals in I. Between each pair of neighboring pole pairs, we add two paths of length 5 that start in the same vertex and end in the roots of the respective central S_f graphs (see Fig. 11).

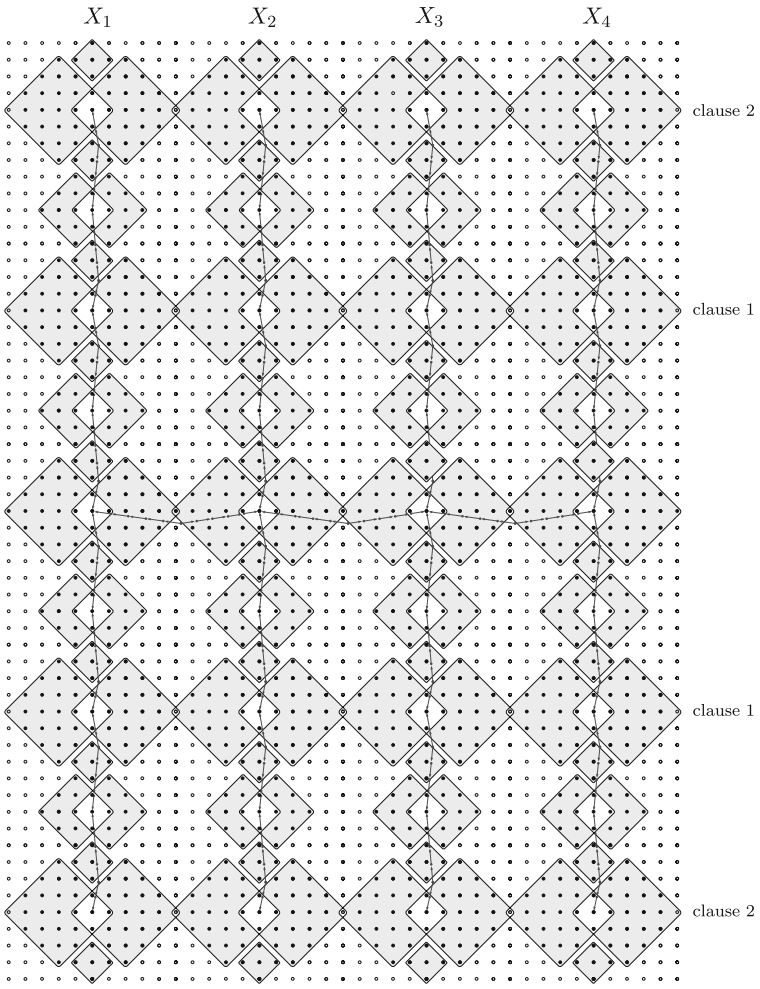


Fig. 11 An embedding of the layer 6 vertices of the logic engine corresponding to an NAE-3SAT instance with 2 clauses on 4 literals

Intuitively, the *base* of this logic engine graph is a number of connected paths of length 5. Here, the rigidness of the base comes from the poles attached to it. The resulting embedding is designed such that each pair of neighboring flag pole segments overlaps by one vertex. This vertex represents the gap in which one flag from either pole can be placed.

Finally, the gadget has to be encoded with the given NAE-3SAT clauses. In this gadget, the kth clause is encoded on the kth flag segments to both sides of the central flag segment S^* in each pole pair. We set the flags as described in Sect. 3.1, and w.l.o.g. treat the poles above S^* as the back line, and those plotted below as the front line. For technical reasons, the central flag segments each hold one flag, except for the first flag pole, in which S^* holds two flags.

We call the completed data-flow graph including base and v pole pairs consisting of $2c$ encoded flag pole segments each the *tree gadget* T_I.

Lemma 3.9 *If the tree gadget T_I has to be embedded into H in 11 time steps, then the flag pole pairs have to lie in parallel rows as in Fig. 11. Accordingly, there are exactly two distinct configurations for each flag pole, i.e. above or below the base.*

Proof Consider two neighboring flag poles k and $k+1$. The roots r^k and r^{k+1} of their central flag segments S^* are connected through 10 edges, so a feasible embedding enforces $\|r^k - r^{k+1}\|_1 \leq 10$. It is easy to see that relative positions other than in Fig. 11 result in an overlapping greater than 1. This is impossible, as the S^* use up all flag gaps in an overlapping of 1. Partner poles can be individually exchanged because they are connected through one vertex only and are identical except for their flag setting. □

Theorem 3.10 *The directed subgraph homeomorphism problem remains NP-complete if we restrict the host graph to a regular graph H as defined in Sect. 2.2, and the pattern graph to the class of directed trees with indegree at most two.*

Proof First, observe that the problem is in NP. To verify a solution, it suffices to check for every edge (u, v) of the pattern graph that its image in H is a path from $\phi(u)$ to $\phi(v)$ without intersections except possibly at starting and endpoints. This can easily be done by a polynomial-time algorithm.

From Lemmas 3.6 to 3.9 it follows that the tree gadget has the structure of a logic engine described in Sect. 3.1: (i) Rigid flag poles that can be exchanged partnerwise, (ii) flags that can be moved to the left or right side of a flag pole segment since $Q_1 \cup Q_2$ is symmetric, and finally, (iii) without overlapping, there can only be one flag on each clause level between two neighboring flag poles. Thus, Proposition 3.1 can be applied and we have NP-completeness. □

Remark 3.11 This result implies that the SSBR problem remains NP-complete for planar and series-parallel data-flow graphs.

4 A $|V|^{3/2}$-approximation

The NP-completeness results open the way for approximation or non-poly-nomial approaches. We proposed an integer linear programming approach capable of solving small to medium sized instances in Brenner et al. (2006). In this section, we present a first polynomial approximation algorithm. Here, we assume to have only binary operations, i.e. all vertices of the given data-flow graph $G = (V, E)$ have indegree smaller or equal to 2.

Unlike other NP-complete problems (as, e.g. the traveling salesman problem), it is very difficult to find an initial feasible solution to the SSBR problem; once a feasible solution is found, it is often very close to the optimum already. This may explain that we were only able to find an approximation with a non-constant guarantee of $|V|^{3/2}$ time steps. Indeed, the existence of a constant factor approximation seems very unlikely, especially one with a factor smaller than 2. An additional result that comes

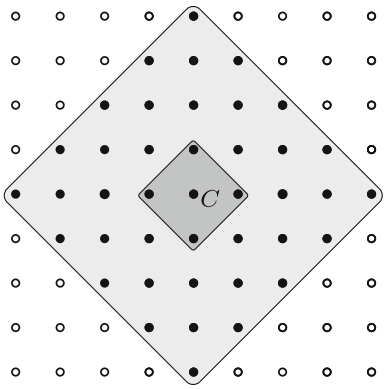


Fig. 12 The neighborhood region for $d = 4$

with the approximation is the proof that every acyclic digraph can be embedded in a feasible way in a large enough grid H.

The basic idea of our approximation algorithm is as follows. We sort the vertices of the input graph topologically, and execute all operations on a single central PE C in this superimposed topological order. For the storage of results, we provide a neighborhood region of maximal Manhattan distance d to C. Figure 12 illustrates such a d-neighborhood for a distance of $d = 4$. After execution, results are stored in this region, and routed back to the center when needed to process another operation.

The constant d depends on the maximal number of results k that have to be stored in any given time step. It is easy to see that the d-neighborhood of C contains exactly $(d+1)^2 + d^2 - 1$ PEs excluding C. This gives a lower bound of $(\sqrt{2k+1} - 1)/2$ for the width d of the neighborhood needed. Because $d \in \mathbb{N}$, we set

$$d = \left\lceil \frac{\sqrt{2k+1} - 1}{2} \right\rceil.$$

Note that k, and thus also d, depend on the input graph as well as the topological order chosen. Obviously, $k \leq \min\{|V|, |E|\}$.

Theorem 4.1 *The SSBR problem with binary operations can be approximated in polynomial time with a factor of*

$$\left\lceil \frac{\sqrt{2k+1} + 1}{2} \right\rceil \cdot |V|/opt,$$

where $k \leq \min\{|V|, |E|\}$. *In terms of* $|V|$, *we obtain an approximation factor of at most* $|V|^{3/2}$.

Proof First, observe that the first five operations can be executed in steps 1 through 5, simply by storing the first four relevant results on the four PEs connected to C.

For induction, assume that the first $l-1$ operations, for a fixed $l < |V|$, have been executed and all necessary results have been stored in the d-neighborhood. To execute operation l, we have to route at most two input values i_1, i_2 from arbitrary PEs in the d-neighborhood into the 1-neighborhood of C. We argue that this takes at most d time steps.

In the execution step of operation $l-1$, begin moving the result i_j, $j \in \{1,2\}$, that lies closer to C towards the center, thereby decreasing the distance by one in each step, until it reaches C. Start the same procedure with the second result two time steps later, until it has distance 1 to C. If the distance of the first result is smaller than d beforehand, wait until after the execution of operation $l-1$. Then, the first result reaches C at most $d-1$ time steps, and the second result reaches the 1-neighborhood at most $1+(d-1)=d$ time steps after operation $l-1$'s execution. Thus, operation l can be executed in the $(d+1)$th schedule step after $l-1$. Note that there may be results stored on the routing paths; however, we can use the double wires to simply exchange values between neighboring PEs.

Altogether, this gives an objective value of $(d+1)\cdot|V| = \lceil(\sqrt{2k+1}+1)/2\rceil\cdot|V|$ steps, resulting in the approximation factor in the theorem. By substituting k with $|V|$ and using the fact that the first five operations can be executed in the first five steps, we obtain a $|V|^{3/2}$-approximation for $|V| \geq 1$.

It remains to show that the algorithm requires only polynomial time. The time complexity of the algorithm is put together by those of the topological sort and the routing. We have the following time complexities: Topological sort can be achieved in $O(|V|+|E|)$ (Korte and Vygen 2002), while the routing and execution determination runs in $O(|V|\cdot d)$ (note that we can fix the routing paths from all PEs to C beforehand). Altogether, this yields a polynomial time complexity of $O(|V|+|E|+|V|\cdot d) \leq O(|V|^2)$. □

With a somewhat more complicated proof, Theorem 4.1 can be strengthened to yield a $|V|^{3/2}$-approximation for large enough graphs with maximum indegree at most 5. However, it should be stated that this is not a constant factor approximation and as such does not promise to be useful in practice.

5 Conclusion

Motivated by a problem in compiler design for a class of coarse-grain reconfigurable architectures, we proved NP-completeness for a special case of the directed subgraph homeomorphism problem. Furthermore, we presented a first polynomial approximation algorithm for this problem.

Subgraph homeomorphism problems where an arbitrary pattern graph has to be mapped to a smallest possible host graph of a given structure have not been studied before. It seems interesting to investigate further if and how our results carry over to other parameterizable host graphs with regular structure.

Another interesting open question is whether a constant factor approximation for the SSBR problem exists, or whether a corresponding hardness proof can be established.

References

Asano T (1985) An approach to the subgraph homeomorphism problem. Theoret Comput Sci 38(23):249–267

Bhatt S, Cosmadakis S (1987) The complexity of minimizing wire lengths in VLSI layouts. Inform Process Lett 25:263–267

Brenner J (2005) Simultaneous scheduling, binding and routing to processor-like reconfigurable architectures. Diploma thesis, Braunschweig University of Technology, Braunschweig, Germany

Brenner J, Fekete S, van der Veen J, Oliveira Filho J, Rosenstiel W (2006) Optimal simultaneous scheduling, binding and routing for processor-like reconfigurable architectures. International conference on field programmable logic and applications (FPL)

Fortune S, Hopcroft JE, Wyllie J (1980) The directed subgraph homeomorphism problem. Theor Comput Sci 10:111–121

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, New York

Korte B, Vygen J (2002) Combinatorial optimization: theory and algorithms. Algorithms and combinatorics 21, 2nd edn. Springer, Berlin

Kühn D, Osthus D (2002) Large topological cliques in graphs without a 4-cycle. Hamburger Beiträge zur Mathematik, Heft 144

LaPaugh A, Rivest R (1978) The subgraph homeomorphism problem. In: Proceedings of the 10th annual ACM symposium on theory of computing, association for computing machinery, New York, pp 40–50

Mader W (1998) Topological subgraphs in graphs of large girth. Combinatorica 18:405–412

Oppold T, Schweizer T, Kuhn T, Rosenstiel W (2004) A design environment for processor-like reconfigurable hardware. In: IEEE International conference on parallel computing in electrical engineering (PARELEC), Dresden, Germany, pp 171–176

Oppold T, Schweizer T, Kuhn T, Rosenstiel W (2004) CRC (Configurable Reconfigurable Core)—Bewertungs- und Entwurfsverfahren für prozessorartig rekonfigurierbare Architekturen. Zwischenkolloquium DFG-SPP 1148. http://www12.informatik.uni-erlangen.de/spprr/colloquium02/uni-tuebingen-rosenstiel.pdf

Schaefer TJ (1978) The complexity of satisfiability problems. In: Proceedings of the 10th annual ACM symposium on theory of computing, association for computing machinery, New York, pp 216–226