The Pair Completion algorithm for the Homogeneous Set Sandwich Problem*

Claudson Bornstein† Celina M. H. de Figueiredo*
Vinícius G. P. de Sá*

May 20, 2013

Abstract

A homogeneous set is a non-trivial module of a graph, i.e. a non-empty, non-unitary, proper vertex subset such that all its elements present the same outer neighborhood. Given two graphs $G_1(V, E_1)$ and $G_2(V, E_2)$, the Homogeneous Set Sandwich Problem (HSSP) asks whether there exists a graph $G_S(V, E_S)$, $E_1 \subseteq E_S \subseteq E_2$, which has a homogeneous set. This paper presents an algorithm that uses the concept of *bias graph* [14] to solve the problem in $O(n \min\{|E_1|, |\overline{E}_2|\} \log n)$ time, thus outperforming the other known HSSP deterministic algorithms for inputs where $\max\{|E_1|, |\overline{E}_2|\} = \Omega(n \log n)$.

1 Introduction

A graph $G_S(V, E_S)$ is said to be a *sandwich graph* of graphs $G_1(V, E_1)$, $G_2(V, E_2)$ if and only if $E_1 \subseteq E_S \subseteq E_2$. A *sandwich problem* for property Π asks whether there exists a sandwich graph (of a given pair of graphs) which has the desired property Π [8]. Graph sandwich problems were first defined in the context of Computational Biology and have arisen, ever since, as natural generalizations of recognition problems [3, 8, 9, 11].

A *homogeneous set* H for a graph $G(V, E)$ is a subset of V such that $1 < |H| < |V|$ and for all $v \in V \setminus H$, either $(v, h) \in E$ for all $h \in H$ or $(v, h) \notin E$ for all $h \in H$. The importance of homogeneous sets for graph decomposition is well known, specially in the perfect graphs field [12].

Given two graphs $G_1(V, E_1)$, $G_2(V, E_2)$, with $E_1 \subseteq E_2$, the *Homogeneous Set Sandwich Problem* (HSSP) asks whether there is a sandwich graph $G_S(V, E_S)$ of (G_1, G_2) which contains a homogeneous set. If so, such a homogeneous set is called a *sandwich homogeneous set* (SHS) of (G_1, G_2).

*This is a manuscript by the authors. The final article was published by Elsevier with DOI number 10.1016/j.ipl.2005.12.010.

†Universidade Federal do Rio de Janeiro, Brazil.

Sandwich versions for a number of other polynomially recognizable problems proved to be NP-complete [3, 5, 8, 9, 11, 16]. The HSSP, which belongs to the seemingly small subset of polynomial sandwich problems, has attracted attention [1, 4, 10, 14] as a challenging problem, since its known algorithms are considerably less efficient than the existing linear time algorithms to find homogeneous sets in a single graph [2, 13].

Throughout the paper, we denote the number of input vertices by n, the number of edges in G_1 (the so-called *mandatory* edges) by m_1 and the number of edges *not* in G_2 (or *forbidden* edges) by $\overline{m}_2$. The values $\min\{m_1, \overline{m_2}\}$ and $\max\{m_1, \overline{m_2}\}$ are represented by m and M, respectively. Also, we assume $m = \Omega(n)$, since a lesser number of either mandatory or forbidden edges disconnects G_1 or $\overline{G}_2$ and the problem is trivial. Non-directed edges between vertices a and b are denoted by (a, b), whereas directed edges from a to b are written $(a \to b)$. A *sink* is a subgraph with no outgoing edges.

The first polynomial-time algorithm for the HSSP was presented by Cerioli *et al.* [1], fixing the problem's time complexity at $O(n^4)$. A few years later, Tang *et al.* introduced the concept of *bias graph* as a tool to solve the HSSP efficiently [14]. A series of new algorithms with continuously improving complexities have been proposed [4, 7], culminating in the current $O(\min\{n^3 \log \frac{m}{n}, mM\})$ upper bound. Additionally, randomized approaches have also been proposed in the form of Monte Carlo [4] and Las Vegas [7] algorithms, both $O(n^3)$. All those previously known HSSP algorithms were based on some variation of a procedure called *bias envelopment* [1, 4, 6, 7]. This paper presents an $O(nm \log n)$ algorithm which does not employ any kind of bias envelopment calls but is totally based on Tang *et al.*'s bias graph idea instead—and which turns to be the fastest HSSP deterministic algorithm known to date for most instances. (There is a small range of inputs—namely, those where $M = O(n \log n)$—for which the Quick Fill algorithm given in [7] remains the fastest.)

In Section 2, we recall Tang *et al.*'s resourceful bias graph. Section 3 shows the proposed algorithm with its correctness proof and complexity analysis. Finally, Section 4 contains our concluding remarks.

2 Bias Graph

A *bias vertex* of a set $H \subset V$ is a vertex $b \in V \setminus H$ such that, for some $v_i, v_j \in H$, there hold $(b, v_i) \in E_1$ and $(b, v_j) \notin E_2$. The set $B(H)$ comprising all bias vertices of H defines the *bias set* of H [14].

Theorem 1. [1] *The set $H \subset V, |H| \geq 2$, is a sandwich homogeneous set of (G_1, G_2) if and only if its bias set $B(H)$ is the empty set.*

Due to Theorem 1, the search for a SHS becomes the search for a proper subset H of V with $|H| \geq 2$ and $B(H) = \varnothing$. Clearly, if b is a bias vertex of

set H, then b is also a bias vertex of every set H' containing H such that $b \notin H'$, which means that any set H' containing H might possibly be a SHS only if H' also contains $B(H)$.

The *bias graph* of $G_1(V, E_1), G_2(V, E_2)$, with node set $V_B = \{[x, y] \mid x, y \in V,\ x \neq y\}$ representing all distinct vertex pairs of the problem's instance. These nodes are interlinked so to represent their pairwise bias relationships, i.e. there are two outgoing edges from node $[x, y]$ to nodes $[x, b]$ and $[y, b]$ in G_B if and only if vertex b is a bias vertex of $\{x, y\} \subset V$. Notice that $[x, y] = [y, x]$.

We will write $L(X)$ to designate the subset of vertices $v \in V$ which appear in the label of some node in subgraph $X \subseteq G_B$, referring to it as the *labeling set* of X and to its elements as X's *labeling vertices*. In other words, $L(X) = \{v \mid [v, z] \in X,\ \text{for some } z\}$. A subgraph $X \subseteq G_B$ is said to be *pair-closed* if and only if $x, y \in L(X)$ implies $[x, y] \in X$.

Tang *et al.*'s algorithm starts by putting together the instance's bias graph $G_B(V_B, E_B)$. Then, it locates an *end strongly-connected component* (ESCC) $S \subset G_B$ and, in case $L(S) \neq V$, it returns *yes* and $L(S)$ as a SHS.

Notwithstanding the fact that (G_1, G_2) will indeed present no SHSs if G_B has no ESCCs, the assumption that every proper ESCC of G_B maps to a SHS is not sound. Indeed, an ESCC S might not contain *all* pairs formed by its labeling vertices, i.e. there might exist a *missing pair* $\{x, y\} \subset L(S)$ such that $[x, y] \in G_B \setminus S$. In case the pair $\{x, y\}$ presents a bias vertex b that is not contained in $L(S)$, b will also be a bias vertex of $L(S) \supset \{x, y\}$, therefore $L(S)$ will fail to be a SHS—despite being S a proper ESCC. This reasoning brought about Theorem 2 transcribed below:

Theorem 2. [7] *A set $H \subset V$, $|H| \geq 2$, is a SHS of (G_1, G_2) if and only if it is the labeling set of a pair-closed sink in that instance's bias graph.*

Supported by Theorem 2, we employ Tang *et al.*'s bias graph as the kernel of an efficient HSSP algorithm.

3 The Pair Completion algorithm

Theorem 2 suggests that a pair-closed sink search strategy is performed. The algorithm we propose, which we call the *Pair Completion* algorithm (*PC*, for short), starts by building the bias graph G_B. Then, as the beginning point of a pair-closed sink search strategy, it locates G_B's ESCCs. (If no proper ESCC exists, then the algorithm can safely stop with a *no* answer, since every sink is either itself an ESCC or properly contains an ESCC).

As a preparatory measure, the auxiliary routine *Locate_Reachable_Sinks* (LRS), given in Figure 1, gathers information about the set of ESCCs that can be reached by each node. Its details will be focused on later.

Locate_Reachable_Sinks $(G_B(V_B, E_B))$

```
1.          for each vertex [x,y] ∈ V_B do
1.1.          sink(x,y) ← undefined
2.          for each ESCC S do
2.1.          let [u,v] be some vertex in S
2.2.          sink(u,v) ← S
2.3.          let R be a list containing initially only [u,v]
2.4.          while R is not empty do
2.4.1.          let [c,d] be the first element in R
2.4.2.          for each edge ([x,y] → [c,d]) ∈ E_B do
2.4.2.1.          if sink(x,y) = undefined then
2.4.2.1.1.          sink(x,y) ← S; put [x,y] into R
2.4.2.2.          else if sink(x,y) ≠ S and sink(x,y) ≠ several then
2.4.2.2.1.          sink(x,y) ← several; put [x,y] into R
2.4.3.          remove [c,d] from R
```

Figure 1: The Locate_Reachable_Sinks routine

The algorithm proceeds by picking one of the ESCCs, say S, and submitting it to the *Perform_Pair_Completion* routine given in Figure 2. (We will refer to it simply as *pair completion*, from now on.)

The pair completion routine collects the labeling vertices of S in the set L (regarded as a SHS candidate). Then, for each pair $\{x, y\}$ in L, it checks whether $[x, y]$ reaches, in G_B, an ESCC *other than* S. If this is the case (not only S is reached by $[x, y]$), then one edge is added from S (i.e. from any of its vertices) to $[x, y]$ and S no longer constitutes an ESCC, hence the algorithm shall drop S and run the pair completion anew on a different ESCC of G_B. If, on the other hand, the only ESCC reached by $[x, y]$ is S itself, then no edges are added—and the algorithm just puts into L all labeling vertices of the out-neighbors of $[x, y]$ in G_B (in case they are not in L yet). If all pairs of vertices in L are investigated without the addition of any new edge, the algorithm will have found the pair-closed sink $S' = \{[u, v] \in V_B \mid u, v \in L\}$ and will therefore stop with a *yes* answer.

By the time all those former ESCCs have been submitted to pair completion, and assuming none of them yielded a *yes* answer, the algorithm is not yet able to stop, since new ESCCs might have arisen as a result of the edge additions. Hence, the whole process of locating ESCCs, running the LRS and performing the pair completion on all ESCCs has to be started over. (Please refer to Figure 3 for the PC algorithm's pseudo-code.) The algorithm goes forth with such successive pair completion *turns* (iterations of the algorithm's main loop) until it has successfully found a SHS, answering

Perform_Pair_Completion $(G_B(V_B, E_B), S)$

1. let L be the set $\{v \in V \mid [u,v] \in S$—or $[v,u] \in S$—for some $u\}$
2. let P be a list containing all $[x,y] \in V_B$ such that $\{x,y\} \subset L$
3. **while** P is not empty **do**
3.1. let $[x,y]$ be the first element in P
3.2. **if** $sink(x,y) \neq S$ **then** *// obtained by the LRS routine*
3.2.1. add edge $([u,v] \rightarrow [x,y])$ to E_B, for some $[u,v] \in S$
3.2.2. $P \leftarrow \varnothing$; $L \leftarrow \varnothing$
3.3. **else**
3.3.1. **for each** vertex $[z,w]$ such that $([x,y] \rightarrow [z,w]) \in E_B$ **do**
3.3.1.1. **if** $z \notin L$ **then**
3.3.1.1.1. **for each** element $h \in L$ **do** put $[z,h]$ into P
3.3.1.1.2. $L \leftarrow L \cup \{z\}$
3.3.1.2. **if** $w \notin L$ **then**
3.3.1.2.1. **for each** element $h \in L$ **do** put $[w,h]$ into P
3.3.1.2.2. $L \leftarrow L \cup \{w\}$
3.4. remove $[x,y]$ from P
4. **if** $1 < |L| < |V|$ **then return** *yes* **else return** *no*

Figure 2: The Perform_Pair_Completion routine

yes, or the current bias graph (the original one plus a number of additional edges) has become strongly connected, yielding a *no* answer.

3.1 Proof of correctness / completeness

The soundness of the PC algorithm results from the following Lemmas 3 and 4. (We call *extended bias graph* the bias graph with any number of extra edges added by the PC algorithm.)

Lemma 3. *If the PC algorithm answers* yes*, the input instance has a SHS.*

Proof. *Yes* answers always result from successful Perform_Pair_Completion runs. But that routine only returns *yes* if it finds a set $L \subset V$ such that, in some extended bias graph $G'_B(V_B, E'_B)$ of (G_1, G_2), *all* nodes labeled by two vertices in L only have out-neighbors whose labeling vertices also belong to L. Therefore L is the labeling set of a pair-closed sink in G'_B. Now, the nodes of any sink of G'_B induce a sink in the original bias set $G_B(V_B, E_B)$ as well, since $E'_B \supseteq E_B$. Thus, by Theorem 2, L is a SHS. □

Lemma 4. *If the input has a SHS, then the PC algorithm finds one.*

Proof. Let us suppose the input instance (G_1, G_2) has SHS L. By Theorem 2, its bias graph $G_B(V_B, E_B)$ must have a pair-closed sink P whose labeling

Pair_Completion_HSSP_Algorithm $(G_1(V, E_1), G_2(V, E_2), H_1)$
1. find the bias graph $G_B(V_B, E_B)$ of (G_1, G_2)
2. **repeat**
2.1. partition G_B into its strongly connected components
2.2. find G_B's proper end strongly connected components (ESCC)
2.3. **if** there is no ESCC **then return** *no*
2.4. Locate_Reachable_Sinks(G_B)
2.5. **for each** ESCC S **do**
2.5.1. **if** Perform_Pair_Completion(G_B, S) returns *yes* **then**
2.5.1.1. **return** *yes* *// else an edge will have been added to* G_B

Figure 3: The Pair Completion algorithm

set is $L(P) = L$. We want to show that the algorithm cannot answer *no*. In order to give a *no* answer, the algorithm must have added enough edges to strongly connect G_B. This means that, in particular, P has to cease being a sink. Now, every additional edge is such that it links an ESCC S to one of its missing pairs, i.e. to a node $[x, y] \notin S$ such that $x, y \in L(S)$. Thus, in order to leave P, the new edge has to link an ESCC $S \subseteq P$ to one of its missing pairs *not* in P. This is simply not possible, as P is pair-closed in G_B and remains so in every extended bias graph $G'_B(V_B, E'_B)$, since the extra edges in $E'_B \setminus E_B$ cannot change this fact. □

3.2 Complexity analysis

The PC algorithm comprises an $O(nm)$ step of building the bias graph [7, 14] plus some pair completion turns which consist in: (i) partitioning the current (extended) bias graph into its SCCs and locating the ESCCs among them; (ii) locating the reachable sinks of all nodes; (iii) visiting the (missing) pairs of each ESCC until one edge (per ESCC) is added.

Step (i) calls Tarjan's strongly connected components (SCC) partitioning method [15], whose time is linear in the number of the digraph's edges. Since the bias graph has formerly $O(nm)$ edges [7, 14], it will take $O(nm + d(k))$ time during the k-th turn, where $d(k)$ is the number of edges added prior to the start of the k-th turn. By Lemma 5, in [7], the number of ESCCs which might *not* be pair-closed is $O(m)$, therefore $\max\{d(k)\}$ is clearly bounded by $O(tm)$, where t is the maximum possible number of turns. We will soon show that t is $O(\log n)$, yielding an $O(nm)$ time bound for step (i).

The complexity of step (ii) is that of the LRS routine, whose mechanics is the following: a *reachable sink* attribute, initially empty, is created for each node. Then, starting by any one node of each ESCC S, the LRS

traverses the (extended) bias graph's edges backwards as if it were running an ordinary breadth-first search (BFS) on a similar graph with reversed edges. As each node is visited, its reachable sink attribute is set as either S, if no other reachable ESCC had yet been set for that node, or *several*, if S is the second reachable ESCC to be revealed for that node. It clearly suffices for the algorithm's purposes, as we actually only need to know if a certain ESCC happens to be the *only* ESCC reached by a node. The time-saving device here is that the search can be discontinued at nodes previously marked *several*, since its ancestors in the bias graph (i.e. descendants in the BFS tree) will already have been marked *several* as well.

Figure 4 illustrates a LRS call. The big ellipses labeled A, B and C, at the bottom, stand for ESCCs of the bias graph at that point. Each node's reachable sink attribute appears next to it. Plus signs indicate *several*.

All edges which leave nodes *not* marked *several* have been visited only once. On the other hand, an edge which leaves a node v marked *several* has been visited exactly twice, as further searches would be discontinued at v. Since all edges are visited a constant number of times, the overall time complexity of each LRS run is $O(|V_B| + |E_B|) = O(n^2 + nm) = O(nm)$.

Finally, step (iii) spends a constant time for each node it visits. It is easy to see that the maximum number of visited nodes per turn sum up to $O(n^2)$ nodes, since nodes toward which no edges were added have been investigated during at most one pair completion, whereas the nodes which did receive an extra incoming edge are not more than one per ESCC.

As for the worst-case number of pair completion turns, suppose $S(k) = \{S_i : i = 1, \ldots, s\}$ is the set of ESCCs in the k-th iteration. We know that, by the end of the k-th iteration, all s former ESCCs S_i will have ceased being an ESCC. We are interested in finding the maximum number of *new* ESCCs that might have been formed. Well, if a new ESCC S' is formed, it must contain at least one element of $S(k)$, no more a sink now. This is true because only edges whose origin node is contained inside some S_i have been added, during the k-th turn—therefore no sink can possibly have been formed containing only vertices which did not belong to any S_i. Nevertheless, we know that the outgoing edge added to former ESCC S_p during the k-th turn provided it with a path to another sink S_q, $q \neq p$, which will only allow the existence of a sink $S' \supset S_p$ if S' also contains S_q. Thus, the minimum number of elements of $S(k)$ contained in any newly formed ESCC is actually two. Since all ESCCs are obviously disjoint, the number of ESCCs in the $(k+1)$-th iteration is at most $|S(k)|/2$, so that $O(\log n)$ turns will be enough to strongly-connect G_B.

The overall time complexity of the Pair Completion algorithm is therefore $O(\log n) \cdot O(nm) = O(nm \log n)$, showing the algorithm's sensibility to the number of edges of G_1 and the number of non-edges in G_2, which is certainly appropriate for such a problem that is invariant under taking the complements of the input graphs.

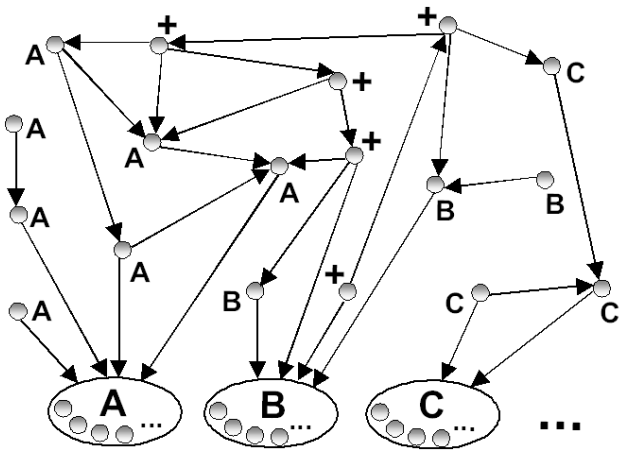

Figure 4: Result of a call to Locate_Reachable_Sinks

4 Conclusion

The algorithm proposed in this paper, which owes substantially to Tang *et al.*'s past efforts, establishes a new $O(nm \log n)$ upper bound for the HSSP. Actually, its performance is likely to be even better than that of the aforementioned randomized algorithms for instances with a limited number of mandatory or forbidden edges—namely, those where $m = O(n^2/\log n)$.

We believe this particular problem's research history reveals the usage of a number of arousing algorithmic tools, wherefrom plenty of interesting didactic resources might be harvested.

Acknowledgements We are grateful to the referees for their careful reading and the many suggestions which helped to improve the paper.

References

[1] M. R. Cerioli, H. Everett, C. M. H. Figueiredo, and S. Klein, *The homogeneous set sandwich problem*, Inf. Proc. Letters **67** (1998), 31–35.

[2] E. Dahlhaus, J. Gustedt, and R. M. McConnell, *Efficient and practical algorithms for sequential modular decomposition*, Journal of Algorithms **41** (2001), 360–387.

[3] S. Dantas, L. Faria, and C. M. H. Figueiredo, *On decision and optimization (k,l)-graph sandwich problems*, Disc. Appl. Math. **143** (2004), 155–165.

[4] C. M. H. Figueiredo, G. D. Fonseca, V. G. P. Sá, and J. Spinrad, *Faster deterministic and randomized algorithms on the homogeneous set sandwich problem*, 3rd Workshop on Efficient and Experimental Algorithms, Lecture Notes Comput. Sci., vol. 3059, Springer-Verlag, 2004, pp. 243–252.

[5] C. M. H. Figueiredo, S. Klein, and K. Vuskovic, *The graph sandwich problem for 1-join composition is NP-complete*, Disc. Appl. Math. **121** (2002), 73–82.

[6] C. M. H. Figueiredo and V. G. P. Sá, *A note on the homogeneous set sandwich problem*, Inf. Proc. Letters **93** (2005), 75–81.

[7] G. D. Fonseca, V. G. P. Sá, C. M. H. Figueiredo, and J. Spinrad, *The homogeneous set sandwich problem*, Tech. Report ES-680/05, Univ. Federal do Rio de Janeiro, 2005, available at http://www.cos.ufrj.br/publicacoes/reltec/es68005.pdf (to appear in Algorithmica).

[8] M. C. Golumbic, H. Kaplan, and R. Shamir, *Graph sandwich problems*, Journal of Algorithms **19** (1995), 449–473.

[9] M. C. Golumbic and A. Wassermann, *Complexity and algorithms for graph and hypergraph sandwich problems*, Graphs Combin. **14** (1998), 223–239.

[10] M. Habib, E. Lebhar, and C. Paul, *A note on finding all homogeneous set sandwiches*, Inf. Proc. Letters **87** (2003), 147–151.

[11] H. Kaplan and R. Shamir, *Bounded degree interval sandwich problems*, Algorithmica **24** (1999), 96–104.

[12] L. Lovász, *Normal hypergraphs and the perfect graph conjecture*, Disc. Math. (1972), no. 2, 253–267.

[13] R. M. McConnell and J. Spinrad, *Linear-time modular decomposition and efficient transitive orientation of comparability graphs*, Proc. of the 5th ACM-SIAM Symposium on Disc. Alg., 1994, pp. 536–545.

[14] S. Tang, F. Yeh, and Y. Wang, *An efficient algorithm for solving the homogeneous set sandwich problem*, Inf. Proc. Letters **77** (2001), 17–22.

[15] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. **1** (1972), no. 2, 146–160.

[16] R. B. Teixeira and C. M. H. Figueiredo, *The sandwich problem for cutsets*, Proc. of LACGA 2004, Electr. Notes in Disc. Math., vol. 18, Elsevier, 2004, pp. 219–225.