The Homogeneous Set Sandwich Problem

Celina M. H. de Figueiredo[*] Guilherme D. da Fonseca[†]
Vinícius G. P. de Sá[‡] Jeremy Spinrad[§]

December 7, 2004

Abstract

A homogeneous set is a non-trivial module of a graph, i.e. a non-empty, non-unitary, proper subset of a graph's vertices such that all its elements present exactly the same outer neighborhood. Given two graphs $G_1(V, E_1)$, $G_2(V, E_2)$, the Homogeneous Set Sandwich Problem (HSSP) asks whether there exists a sandwich graph $G_S(V, E_S)$, $E_1 \subseteq E_S \subseteq E_2$, which has a homogeneous set. In 2001, Tang *et al.* [15] published an all-fast $O(n^2 \triangle_2)$ algorithm which was recently proven wrong [5], so that the HSSP's known upper bound would have been reset thereafter at former $O(n^4)$ determined by Cerioli *et al.* [1] in 1998. We present, notwithstanding, new deterministic algorithms which have it established at $O(n^3 \log \frac{m}{n})$. We give as well two even faster $O(n^3)$ randomized algorithms, whose simplicity might lend them didactic usefulness. We believe that, besides providing efficient easy-to-implement procedures to solve it, the study of these new approaches allows a fairly thorough understanding of the problem.

Keywords: *graph, sandwich problem, homogeneous set*

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 [6].

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 $h \in H$.

[*]celina@cos.ufrj.br, Univ. Federal do Rio de Janeiro, Brazil
[†]fonseca@cs.umd.edu, Univ. of Maryland, USA
[‡]vgusmao@cos.ufrj.br, Univ. Federal do Rio de Janeiro, Brazil (corresponding author)
[§]spin@vuse.vanderbilt.edu, Vanderbilt Univ., USA

Given two graphs $G_1(V, E_1)$, $G_2(V, E_2)$ such that $E_1 \subseteq E_2$, the *Homogeneous Set Sandwich Problem* (HSSP) asks whether there exists 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 pair (G_1, G_2).

Graph sandwich problems were first defined in the context of Computational Biology and have attracted much attention ever since. They arise from many applications as a natural generalization of recognition problems [3, 6, 7, 8].

The importance of homogeneous sets in the context of graph decomposition has been well acknowledged, specially in the perfect graphs field [9].

It is not always straightforward to enable a standard recognition algorithm to handle efficiently the generalized sandwich version of a problem, as the number of sandwich graphs of a given pair $G_1(V, E_1), G_2(V, E_2)$ is exponential in the cardinality of $E_2 \setminus E_1$. This seems to be the case with the HSSP. Despite the existence of linear-time algorithms which find homogeneous sets in a single graph [2, 10, 11, 12, 14], the known HSSP algorithms are considerably less efficient.

The first polynomial-time algorithm for this problem was presented by Cerioli *et al.* [1], which set HSSP's upper bound at their algorithm's $O(n^4)$ time complexity. We refer to that algorithm as the *Exhaustive Bias Envelopment* algorithm (*EBE* algorithm, for short). A few years later, Tang *et al.* [15] tailored an interesting algorithm which would have largely diminished HSSP's upper bound. That algorithm, referred to as the *Bias Graph Components* algorithm (*BGC*, for short), was however proved incorrect in [5, 13]. As a consequence, the most efficient algorithm for the HSSP would turn back to be former EBE algorithm presented in [1], resetting HSSP's upper bound at $O(n^4)$. A careful study of the underlying ideas contained in both [1] and [15], though, has led us to the development of a series of faster algorithms.

Throughout this paper, we denote the number of vertices in the input graphs by n, the number of edges in graph G_i by m_i and the number of edges *not* in G_i by $\overline{m}_i$. $\triangle_i$ and $\overline{\triangle}_i$ stand for the maximum vertex degree of G_i and $\overline{G}_i$, respectively, where $\overline{G}_i$ is G_i's complement.

The terms *mandatory edges* and *forbidden edges* are used, in the context of sandwich problems, to designate the edges in G_1 and those not in G_2, respectively, reminding that every sandwich graph of (G_1, G_2) must contain the former but not the latter. Also, we denote $N_1(v) = \{x \in V \mid (x, v) \in E_1\}$ as the set of *mandatory neighbors* of vertex v, and $\overline{N}_2(v) = \{x \in V \mid (x, v) \notin E_2\}$ as the set of *forbidden neighbors* of v.

This paper is organized as follows:

Section 2 presents the common core of all HSSP algorithms known so far[1]: the $O(n^2)$ *Bias Envelopment* procedure, introduced in [1]. Actually, we show that its computing time can be bounded by $O(m_1 + \overline{m}_2)$, which is somewhat tighter. The section ends with a short description of Cerioli *et al.*'s $O(n^4)$ EBE algorithm [1], so to say the basis of all others.

In spite of the counterexamples given in [5, 13], which have shown the hopelessness of any simple corrective attempts, Tang *et al.*'s algorithm [15] holds the unquestionable merit of having introduced the so-called *bias graph*, an auxiliary digraph which has proved rather useful in fastening some of the new envelopment-based algorithms presented herein. For this reason, Section 3 briefly revisits the BGC algorithm, depicting the bias graph and its usefulness. Additionally, we disclose (in Appendix 2) a faster $O(n \cdot \min\{m_1, \overline{m}_2\})$ way of obtaining it.

We introduce, in Section 4, our first new algorithm, the $O(m_1\overline{m}_2)$ *Two-Phase* algorithm. It is not altogether original, it must be said, for it does employ Tang *et al.*'s fine bias graph concept — and still it bears essentially on the Bias Envelopment procedure.

Most algorithms introduced in this paper are actually based on a variation of the Bias Envelopment procedure which we call the *Incomplete Bias Envelopment*, to the study of which Section 5 is dedicated.

The Incomplete Bias Envelopment, along with the rather powerful *balancing* technique, gave rise to an $O(n^{3.5})$ algorithm referred to as the *Balanced Subsets* algorithm, which is introduced in Section 6.

Section 7 is devoted to a fast, randomized Monte Carlo algorithm, which solves the HSSP in $O(n^3)$ time within whichever desired error ratio. As often happens with randomized algorithms, its idea is very simple and so is its implementation — but its analysis is not quite so. Indeed, it is the only moment in the whole paper where a few, somewhat dull algebraic efforts were called for.

There follows Section 8, which brings about the *Harmonic Series* algorithm, whose $O(n^3 \log n)$ time complexity makes it more efficient than all previously known deterministic others.

In Section 9 we introduce the concept of *enemy vertices*, fundamental to the construction of all remaining algorithms.

Section 10 presents another $O(n^3 \log n)$ deterministic algorithm, the *Growing Cliques* algorithm. It is a little less simple than the Harmonic Series discussed in Section 8, but it is — by some constant factor — faster. Besides, it entails the basis for the understanding of the next-coming Las Vegas algorithm.

The algorithm that comes next is a randomized one which follows the Las Vegas standard of always giving the correct answer within some deterministically calculated expected time. Its simplicity, besides an expected

[1]As a matter of fact, Tang *et al.*'s BGC algorithm [15] was the only one which did not employ it anyhow.

time of $O(n^3)$, makes it almost unsurpassable, for the time being, as HSSP's best practical choice. Section 11 presents its description and analysis.

The last but not least contribution of this paper is the $O(n^3 \log \frac{m}{n})$ deterministic *Quick Fill* algorithm covered by Section 12, which sets the current upper bound for the HSSP.

Finally, Section 13 allows the practical comparison of all HSSP algorithms by reporting some relevant experimental results. It helps to deepen one's understanding of each algorithms' behavior, once it makes it possible to verify how each algorithm's theoretical time complexity matches with the practical computing time it demands to handle inputs of different sizes.

2 Bias Envelopment

Tang *et al.* [15] assigned the term *bias set* of a vertex subset $H \subset V$ to the set $B(H)$ of vertices $b \notin H$ such that, for some $v_i, v_j \in H$, there hold $(b, v_i) \in E_1$ (i.e. there is at least one mandatory edge between b and some $v_i \in H$) and $(b, v_j) \notin E_2$ (i.e. there is also at least one forbidden edge between b and some $v_j \in H$). Such vertices $b \in B(H)$ are called *bias vertices* of H.

The following theorem, whose proof is all the most intuitive, is the raison d'être of the procedure on which the current section is meant to focus on.

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.*

Proof. Suppose $B(H) \neq \varnothing$. Thus, in all possible sandwich graphs of (G_1, G_2), any vertex $b \in B(H)$ must be adjacent to at least one vertex $v_i \in H$ and also non-adjacent to at least one vertex $v_j \in H$. This clearly prevents H from being a SHS. If we suppose, on the other hand, that $B(H) = \varnothing$, it is certainly possible to build a sandwich graph $G_S(V, E_S)$ of (G_1, G_2) in such a way that H is a homogeneous set of G_S. This can be accomplished by adding all mandatory edges to an initially empty E_S. Then, for every vertex $x \in V \setminus H$ such that (x, v_i) is mandatory for some $v_i \in H$, we add to E_S the edges (x, v) from x to each and every vertex $v \in H$. Notice that this is always possible, once x, by hypothesis, cannot be a bias vertex of H. $\square$

It comes from Theorem 1 that any SHS containing $H \subset V$ must also contain $B(H)$. This requirement gave rise to a procedure, in [1], which we refer to as *Bias Envelopment*.

The Bias Envelopment aims at determining whether or not the input instance admits a SHS which contains a given set of vertices. Starting from a given initial SHS *candidate* $H_1 \subset V$, the procedure successively computes $H_q = H_{q-1} \cup B(H_{q-1})$ until either (i) $B(H_q) = \varnothing$, whereupon H_q is a SHS and it answers *yes*, or else (ii) $H_q \cup B(H_q) = V$, when its resulting *no* answer will mean there is no SHS containing H_1.

```
**Bias_Envelopment** $(G_1(V,E_1), G_2(V,E_2), H_1)$

1.      $H \leftarrow H_1$
2.      while $|H| < |V|$ do
2.1.        if $B(H) = \varnothing$
                return yes. //a SHS was found: $H$.
2.2.        else
                $H \leftarrow H \cup B(H)$
3.      return no. //there are no SHSs containing $H_1$.
```

Figure 1: The Bias Envelopment procedure

```
**Exhaustive_Bias_Envelopment_HSSP_Algorithm** $(G_1(V,E_1), G_2(V,E_2))$

1.      for each pair of vertices $\{x,y\} \subset V$ do
1.1.        if Bias_Envelopment $(G_1, G_2, \{x,y\})$ = yes
                return yes.
2.      return no.
```

Figure 2: The EBE algorithm [1]

Figure 1 shows the pseudo-code for the Bias Envelopment procedure.

The Bias Envelopment procedure is sufficient to show that the HSSP is polynomially solvable. Indeed, the idea of Cerioli *et al.*'s Exhaustive Bias Envelopment (EBE) algorithm [1] is that of simply running one Bias Envelopment for each of the $\Theta(n^2)$ pairs of the input vertices, as illustrated by Figure 2. Since, by definition, it is not possible that there be a SHS with less than two vertices, this strategy clearly works.

The trickiest contribution found in [1] is actually the way they achieve a quadratic running time for each Bias Envelopment. As the determination of a bias set $B(H_q)$ from scratch would demand $O(n^2)$ time, and because it is possible that the size of the candidate set grows slowly (even on a single increment basis), an entire Bias Envelopment would consume $O(n^3)$ time. However, granted some dynamically maintained auxiliary sets are used [1], each bias set $B(H_q)$ shall *not* be naively obtained from scratch, but from a constant number of unions, differences and intersections of sets, which update $B(H_{q-1})$ instead. This way, each Bias Envelopment procedure needs only $O(n^2)$ time to run in its entirety, yielding an overall $O(n^2 \cdot n^2) = O(n^4)$ time complexity for the whole EBE algorithm.

Actually, we show that there is still a faster way to run the Bias Envel-

```
**Bias_Graph_Components_HSSP_Algorithm** $(G_1(V, E_1), G_2(V, E_2))$

1. Construct the bias graph $G_B$ of $(G_1, G_2)$.
2. Find an end strongly connected component $P$ of $G_B$.
3. Let $H$ denote the set of vertices in $V$ that label the vertices in $P$.
4. if $H \neq V$
       return yes. //$H$ is a SHS.
5. return no.
```

Figure 3: The BGC algorithm [15]

opment procedure, whose $O(m_1 + \overline{m}_2)$ computing time not only will play an essential role in the complexity analysis of one of new algorithms studied herein but also gives a tighter bound for the EBE algorithm itself at $O\left(n^2(m_1 + \overline{m}_2)\right)$. Its technical details can be found in Appendix 1.

3 Bias Graph

Tang *et al.* [15] used the bias relations introduced in Section 2 in order to construct an auxiliary directed graph which turned out to be an important piece in the study of the HSSP. Unfortunately, the algorithm they created, based on the so-called *bias graph*, happened to be incorrect [5, 13].

The idea behind the bias graph is to exhibit at once the input vertices' bias relationships, allowing interdependent vertices to be quickly grouped in a number of disjoint sets, some of which likely to be associated with SHSs.

The bias graph $G_B(V_B, E_B)$ of a pair of graphs $G_1(V, E_1), G_2(V, E_2)$ has vertex set $V_B = \{[x, y] \mid x, y \in V, x \neq y\}$ and there are two outgoing edges from vertex $[x, y]$ to vertices $[x, b]$ and $[y, b]$ in G_B if and only if vertex b is a bias vertex of set $\{x, y\} \subset V$. Notice that vertices $[x, y]$ and $[y, x]$ in G_B are the same.

As pointed out in [15], the bias graph does not need more than $O(n^2 \triangle_2)$ time to be obtained. This bound — which dominates the complexity of the whole BGC algorithm — is, however, not tight. Please refer to Appendix 2 for an $O(n \cdot \min\{m_1, \overline{m}_2\})$ bias graph construction method.

The BGC algorithm proposed in [15] used the bias graph in the following way: once it had been constructed, the algorithm would run Tarjan's method [16] to find all its strongly connected components and then would look for an *end* strongly connected component (ESCC) among them, i.e. a strongly connected component with no outgoing edges. If only one ESCC was found and it embraced all input vertices (as part of its vertices' labels), the algorithm would return *no*. Otherwise, the algorithm would translate

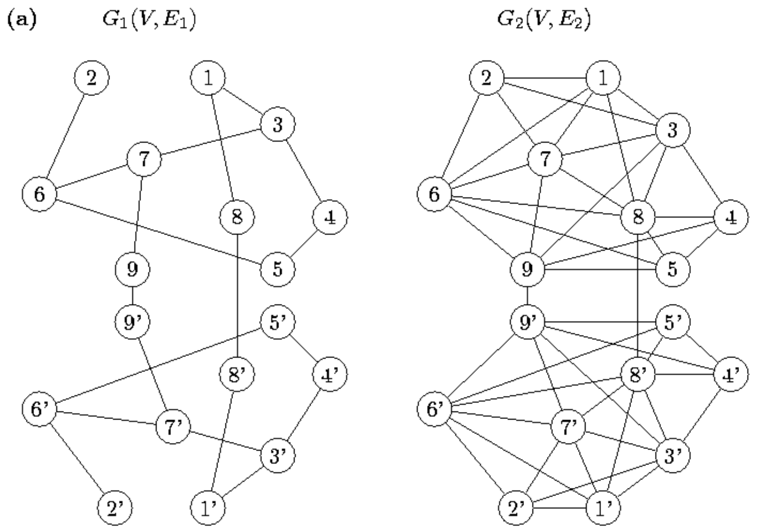

Figure 4: Counterexample to Claim 2 [15]

one of the bias graph's ESCCs, say component P, into the set $H \subset V$ of input vertices that appeared in the labels of P's vertices. Then it would return *yes*, for H would allegedly be a SHS.

The summarized steps of the BGC algorithm are shown in Figure 3.

Claim 2. [15] *The BGC algorithm correctly solves the HSSP.*

The above claim is not correct. Although Figure 4 recalls a sufficient counterexample instance thereof, one should refer to [5] for its detailed refutation.

Theorem 3 that follows employs Tang *et al.*'s bias graph in a correct SHS characterization, supporting two of the new HSSP algorithms in this paper: the Two-Phase, given in the following Section 4; and the Quick Fill, in Section 12 later on.

Let $G_B(V_B, E_B)$ be the bias graph of input graphs $G_1(V, E_1), G_2(V, E_2)$. A subset $K \subseteq V_B$ is said to be a *pair-closed* set if and only if there do not exist two vertices $x, y \in V$, among those which label K's vertices, such that vertex $[x, y]$ is not an element of K. The set $Q = \{[1, 2], [1, 3], [2, 3]\}$ is a pair-closed set. The set $Q' = \{[1, 2], [1, 3], [1, 4], [2, 3], [2, 4]\}$ is *not* pair-closed, for vertices 3 and 4 appear in the label of some vertices in Q' but $[3, 4] \notin Q'$.

Theorem 3. *A set $H \subset V$, $|H| \geq 2$, is a sandwich homogeneous set of graphs $G_1(V, E_1), G_2(V, E_2)$ if and only if the pair-closed set $K = \{[x, y] \mid x, y \in H\} \subset V_B$ induce an* end *subgraph in the bias graph G_B of (G_1, G_2).*

Proof. Let $K \subset V_B$ be the pair-closed set that holds all vertices $[v, w] \in V_B$ such that $v, w \in H \subset V$, and only these. Assume, by hypothesis, that K induces an end subgraph in G_B (not necessarily strongly connected). Now suppose, by contradiction, that H is not a SHS of (G_1, G_2). Then, H must have a bias vertex $b \in V \setminus H$, which means that there exists a mandatory edge between b and some vertex $h_1 \in H$ and also a forbidden edge between b and some other vertex $h_2 \in H$. But this implies that vertex $[h_1, h_2] \in K$ has outgoing edges to vertices $[h_1, b]$ and $[h_2, b]$, which cannot be in K. This is a contradiction, for K induces an end subgraph (i.e. one which does not have any outgoing edges). Conversely, if H is a SHS, then it does not have any bias vertices. Consequently, if a pair of vertices $h_1, h_2 \in H$ has a bias vertex b then b also belongs to H. (Otherwise, b would be a bias vertex of H.) Once K is pair-closed in the vertices of H, vertices $[h_1, b], [h_2, b] \in V_B$ must belong to K, so that the subgraph of G_B induced by K does not have any outgoing edges. □

4 The Two-Phase algorithm

Theorem 3 does not lead directly to an efficient algorithm for the HSSP, for it is not known of any quick means of finding pair-closed sets which induce end subgraphs in the bias graph. Corollary 4, however, brings about the central inspiration for the algorithm that follows.

Corollary 4. *If $H \subset V$ is a sandwich homogeneous set of graphs $G_1(V, E_1)$, $G_2(V, E_2)$, then either the subgraph $G_B\langle K\rangle$, induced by the pair-closed set $K = \{[x, y] \mid x, y \in H\} \subset V_B$ in the bias graph $G_B(V_B, E_B)$ of (G_1, G_2), is itself an end strongly connected component or else it contains, properly, some end strongly connected component of G_B.*

Proof. From Theorem 3, we know that $G_B\langle K\rangle$ is an end subgraph of G_B. If it is strongly connected, then the statement holds trivially. If it is not, then there must exist two vertices $x, y \in G_B\langle K\rangle$ such that there is no path from x to y. The set $R(x)$ of all vertices that are reachable from x certainly induces an end subgraph, for it cannot contain any outgoing edges to vertices $u \notin R(x)$, otherwise x would reach u. Also, $y \notin R(x)$, so that $R(x)$ is a proper subgraph of $G_B\langle K\rangle$. Thus, the fact that $G_B\langle K\rangle$ is not strongly connected implies that it contains some end, proper subgraph $G_B\langle K'\rangle$. This subgraph, in turn, must either be itself strongly connected (which would end the proof) or contain an end, proper subgraph $G_B\langle K''\rangle$, and so on and so forth. As G_B is finite, this ought to stop at some point,

```
Two_Phase_HSSP_Algorithm ($G_1(V, E_1), G_2(V, E_2)$)

1.     Construct the bias graph $G_B$ of $(G_1, G_2)$.
2.     for each end strongly connected component $P_i$ of $G_B$ do
2.1.      Let $H_i$ be the set of vertices in $V$ that label the vertices in $P_i$.
2.2.      if Bias_Envelopment $(G_1, G_2, H_i)$ = yes
             return yes.
3.     return no.
```

Figure 5: The 2-P algorithm

whereupon we will finally have found an end subgraph of $G_B\langle K\rangle$ which is strongly connected. □

The new algorithm we propose is rather simple. It can be regarded as either (i) an improved version of the EBE algorithm which just does not run the Bias Envelopment on all $\Theta(n^2)$ input vertices' pairs, but instead on a sufficient, shorter number of initial candidates; or (ii) an improved version of the BGC algorithm, which only does not hasten to associate the bias graph's ESCCs with (perhaps false) SHSs of the input graphs, but instead into subsets of the input vertices, among which at least one is meant to be *contained* (properly or not) in a SHS, in case there exists any.

As it comprises two totally distinct phases, and in the lack of a better name, we refer to it as the *Two-Phase* algorithm (*2-P*, for short).

The first phase of the 2-P algorithm builds the bias graph $G_B(V_B, E_B)$ of the input instance and locates all its end strongly connected components $P_i \subseteq G_B$. Each P_i is then used to determine the subset $H_i \subseteq V$ of the input vertices such that H_i contains all vertices which appear in some vertex label in P_i.

Its second phase simply runs the Bias Envelopment procedure taking each of the subsets H_i as the initial candidate.

The algorithm returns (i) *yes*, in case any a SHS $H \supseteq H_i$ exists, for some i; or (ii) *no* in case no subset H_i happens to be contained in a SHS of the input instance.

Figure 5 depicts the mechanics of the Two-Phase algorithm.

4.1 Proof of correctness / completeness

The soundness of the 2-P algorithm comes directly from Corollary 4 and from the fact that the Bias Envelopment procedure correctly determines whether there exists a SHS which contains a given subset of the input vertices. Let $G_1(V, E_1), G_2(V, E_2)$ be the input graphs for the 2-P algorithm, and $G_B(V_B, E_B)$ their bias graph. First of all, if the algorithm returns *yes*, then a SHS H must have been found by a Bias Envelopment call, which grants its validity. Moreover, if the pair (G_1, G_2) admits a SHS H, then, by Corollary 4, the subgraph $G_B\langle K\rangle$ which is induced in G_B by the pair-closed subset $K \subset V_B$ (comprising vertices $[x, y]$ labelled by all $x, y \in H$, and only these) contains an ESCC of G_B, say $G_B\langle P_i\rangle$. If $G_B\langle P_i\rangle = G_B\langle K\rangle$, then the set H itself is the initial candidate of some Bias Envelopment iteration (lines 2.1 and 2.2, in Figure 5). As its bias set is empty, the algorithm discovers that it is a SHS of (G_1, G_2) and stops. If, on the other hand, $G_B\langle P_i\rangle \subsetneq G_B\langle K\rangle$, then a subset $H_i \subseteq H$ is the initial candidate of some Bias Envelopment iteration. In this case, as there exists a SHS which contains H_i, namely H, the Bias Envelopment call which starts with candidate H_i certainly answers *yes*.

4.2 Complexity analysis

The time complexity of all steps in the first phase is bounded by the number of edges in the bias graph. Thus, it can be run in $O\left(n \cdot \min\{m_1, \overline{m}_2\}\right)$ time. (Please refer to Appendix 2 for details on the cardinalities and construction of the bias graph.)

The second phase of the 2-P algorithm runs several Bias Envelopments (one for each ESCC of the bias graph, in the worst case). The time demanded by each Bias Envelopment is $O(m_1 + \overline{m}_2)$, as one can find in Section 2 (with further details in Appendix 1). So the question is, how many times, in the worst case, does the Bias Envelopment procedure have to be run? That is, what is the maximum size of a sequence of ESCC-associated input vertices' subsets that fail to be contained in any SHSs of (G_1, G_2)?

The answer comes straight from Theorem 3, which grants that all end subgraphs of G_B whose vertices perform a pair-closed set are associated to a SHS. Consequently, the ESCC $G_B\langle P_i\rangle$, whose associated set $H_i \in V$ fails to be contained in any SHSs, must have been induced by a *non*-pair-closed vertex set $P_i \subset V_B$. The worst-case number of Bias Envelopments is therefore bounded by the maximum number of ESCCs, in the bias graph, that might be induced by non-pair-closed sets of vertices.

Lemma 5 establishes an upper bound to such ESCCs and allows us to determine the total time complexity of the 2-P algorithm.

Lemma 5. *In a bias graph, there are at most* $O(\min\{m_1, \overline{m}_2\})$ *end strongly connected components induced by non-pair-closed vertex sets.*

Proof. Let $G_B(V_B, E_B)$ be the bias graph of graphs $G_1(V, E_1), G_2(V, E_2)$ and let $P_i \subset V_B$ be a non-pair-closed vertex set that induces ESCC $G_B\langle P_i\rangle$ in G_B. Now let $[x, y]$ be a bias graph vertex that belongs to P_i. Vertex $[x, y]$ necessarily presents an outgoing edge, for $G_B\langle P_i\rangle$ is strongly connected. (Clearly,

$P_i \setminus \{[x, y]\}$ is nonempty, for $\{[x, y]\}$ *is* pair-closed, whereas P_i, by hypothesis, is not.) Let $([x, y], [x, t])$ be an edge in $G_B\langle P_i\rangle$. Vertex $t \in V$ is therefore a bias vertex of set $\{x, y\} \subset V \setminus \{t\}$, so that edge (x, t) is mandatory and edge (y, t) is forbidden (or vice-versa). Without loss of generality, let edge (x, t) be the mandatory one. We define a function $l_M(P)$ that associates ESCC $G_B\langle P\rangle$ with such a mandatory edge (i.e. a mandatory edge of the input instance which is necessary for the existence of some bias relationship that appears inside $G_B\langle P\rangle$). This chosen mandatory edge, returned by $l_M(P)$, will stand for $G_B\langle P\rangle$'s *label*. In the current example, "(x, t)" is a possible label for $G_B\langle P_i\rangle$. Notice that no other ESCC $G_B\langle P_j\rangle$ can possibly be assigned the same label "(x, t)". In other words, $j \neq i \Rightarrow l_M(P_j) \neq l_M(P_i)$. Otherwise, because of the way a label is chosen by l_M, there would necessarily be an edge $([x, w], [x, t])$ — or $([w, t], [x, t])$ — in $G_B\langle P_j\rangle$, for some $w \in V$. Because $G_B\langle P_j\rangle$ is an ESCC, $[x, t]$ must be in $G_B\langle P_j\rangle$. But this is not possible, for $[x, t]$ belongs to $G_B\langle P_i\rangle$ and any two strongly connected components in a digraph have an empty intersection. Thus, labelling function l_M is injective, which implies that the number of ESCCs that are induced by non-pair-closed vertex sets is bounded by the number m_1 of mandatory edges. We reason that an analogous injective labelling function l_F, which only assigns to each non-pair-closed ESCC some *forbidden* input edge instead, implies that the number of such non-pair-closed ESCCs is also bounded by the number $\overline{m}_2$ of forbidden edges. □

The time complexity of the second phase of the 2-P algorithm is therefore $O(\min\{m_1, \overline{m}_2\})$ times the $O(m_1 + \overline{m}_2)$ complexity of each Bias Envelopment, which yields $O\left((m_1 + \overline{m}_2) \cdot \min\{m_1, \overline{m}_2\}\right)$.

We remark that the time complexity of the first phase — $O(n \cdot \min\{m_1, \overline{m}_2\})$ — is dominated by that of the second's, since $(m_1 + \overline{m}_2)$ is clearly $\Omega(n)$. Otherwise, both G_1 and the complement of G_2 would be disconnected, characterizing trivial HSSP instances (for any vertex $v \in V$ which is isolated either in G_1 or in the complement of G_2 gives the SHS $V \setminus \{v\}$ effortlessly).

Finally, we rewrite the time complexity of the whole 2-P algorithm as

$$O\left((m_1 + \overline{m}_2) \cdot \min\{m_1, \overline{m}_2\}\right) =$$

$$O\left(\max\{m_1, \overline{m}_2\} \cdot \min\{m_1, \overline{m}_2\}\right) =$$

$$O\left(m_1 \overline{m}_2\right).$$

5 Incomplete Bias Envelopment

Before we step further into the next HSSP algorithms, we will introduce a variation of the Bias Envelopment procedure on which are based most of the forthcoming algorithms. We call it the *Incomplete Bias Envelopment*.

```
Incomplete_Bias_Envelopment ($G_1(V, E_1), G_2(V, E_2), H_1, k$)

1.    $H \leftarrow H_1$
2.    while $|H| \leq \min\{k, |V| - 1\}$ do
2.1.      if $B(H) = \varnothing$
              return yes. //a SHS was found: $H$.
2.2.      else
              $H \leftarrow H \cup B(H)$
3.    return no. //there are no SHSs with $k$ vertices
                 //or less which contain $H_1$.
```

Figure 6: The Incomplete Bias Envelopment procedure

The input of the Incomplete Bias Envelopment comprises not only a pair of graphs $G_1(V, E_1), G_2(V, E_2)$ and a SHS *candidate* $H_1 \subset V$ but also a *stop parameter* $k \leq n$. The only difference between this incomplete version and the former, complete one, is that now, whenever the size $|H_q|$ of the current candidate becomes greater than k, the envelopment stops prematurely with a *no* answer. Notice that such a *no* answer from the Incomplete Bias Envelopment with parameter k means that H_1 is not contained in any SHSs *with k vertices or less.*

The Incomplete Bias Envelopment clearly generalizes its complete version, as a normal (complete) Bias Envelopment is equivalent to an Incomplete Bias Envelopment with parameter $k = n - 1$.

The pseudo-code for the Incomplete Bias Envelopment is in Figure 6.

Using the same data structures as in [1], the Incomplete Bias Envelopment runs in $O(nk)$ time. If, on the other hand, the optimized Bias Envelopment method given in Appendix 1 is adapted to allow incomplete envelopments, an $O(\min\{m_1, k\triangle_1\} + \min\{m_2, k\overline{\triangle}_2\})$ bound can be achieved. Nevertheless, this latter bound is somewhat hard to work with and will not be taken into account in the analysis of the algorithms that employ the Incomplete Bias Envelopment in the remainder of this paper. The simpler — albeit less tight — $O(nk)$ suffices for our purposes and will be onwards used instead.

6 The Balanced Subsets Algorithm

The algorithm we propose in this section, which will be referred to as the *Balanced Subsets* algorithm (*BS*, for short), is quite similar to the EBE algorithm, in the sense that it submits each of the input vertices' pairs to the envelopment process. The only difference is that this new algorithm

```
Balanced_Subsets_HSSP_Algorithm $(G_1(V, E_1), G_2(V, E_2))$

1.    Label all vertices in $V$ from $v_1$ to $v_n$.
2.    Create $\lceil \sqrt{n}\, \rceil$ empty sets $C_i$.
3.    for each vertex $v_j \in V$ do
3.1.      $C_{j \text{ modulo} \lceil \sqrt{n}\, \rceil} \leftarrow C_{j \text{ modulo} \lceil \sqrt{n}\, \rceil} \cup \{v_j\}$.
4.    for each pair of vertices $\{x, y\}$ in the same subset $C_i$ do
4.1.      if Bias_Envelopment $(G_1, G_2, \{x, y\})$ = yes
              return yes.
5.    for each pair of vertices $\{x, y\}$ not in the same subset $C_i$ do
5.1.      if Incomplete_Bias_Envelopment $(G_1, G_2, \{x, y\}, \lceil \sqrt{n}\, \rceil)$ = yes
              return yes.
6.    return no.
```

Figure 7: The Balanced Subsets algorithm

establishes a particular order in which the vertex pairs are chosen, in such a way that it can benefit, at a certain point, from unsuccessful envelopments that took place prior to it. After some unsuccessful envelopments, a number of vertex pairs have been found not to be contained in any SHSs. This knowledge is then taken into consideration by the algorithm, which will stop further envelopments earlier by means of *Incomplete* Bias Envelopment calls, saving relevant time without loss of completeness.

Figure 7 illustrates the pseudo-code for the Balanced Subsets algorithm.

When the algorithm starts, it partitions all n vertices of the input graphs into $\lceil \sqrt{n}\, \rceil$ disjoint subsets C_i of size $O(\sqrt{n})$, each. Then all pairs of vertices will be submitted to Bias Envelopment in two distinct phases: in the first phase (lines 4 and 4.1, in Figure 7), all those pairs consisting of vertices from the same subset C_i are bias-enveloped (and only those); in the second phase (lines 5 and 5.1), all remaining pairs (i.e. those comprising vertices that are *not* from the same subset C_i) are then bias-enveloped. In the end, all possible vertex pairs will have been checked out as to belonging or not to some SHS from the input instance, just like in the EBE algorithm. The point is: if all Bias Envelopments in the first phase fail to find a SHS, then the input instance does not admit any SHSs which contain two vertices from the same subset C_i. Thence, the maximum size of any possibly existing SHS is $\lceil \sqrt{n}\, \rceil$ (the number of subsets into which the input vertices had been initially dispersed), which grants that all Bias Envelopments of the second phase need not search for SHSs any larger. That is why Incomplete Bias Envelopments with stop parameter $k = \lceil \sqrt{n}\, \rceil$ can be safely used.

6.1 Proof of correctness / completeness

Theorem 6. *The Balanced Subsets algorithm correctly answers whether or not the HSSP input instance has a sandwich homogeneous set.*

Proof. If the algorithm returns *yes*, then it has successfully found a set $H \subset V$, with $|H| \geq 2$, such that the bias set of H is empty. Thus, H is indeed a valid SHS.

Now, suppose the input has a SHS H. If $|H| > \lceil \sqrt{n} \rceil$ then there are more vertices in H than subsets into which all input vertices were spread, in the beginning of the algorithm (line 3.1). Thus, by the pigeon hole principle, there must be two vertices $x, y \in H$ which were assigned to the same subset C_i. So, whenever $\{x, y\}$ is submitted to Bias Envelopment (line 4.1), the algorithm is doomed to find a SHS. On the other hand, if $|H| \leq \lceil \sqrt{n} \rceil$, then it is possible that H does not contain any two vertices from the same subset C_i, which would cause all Bias Envelopments of the first phase to fail. In this case, however, when a pair $\{x, y\} \subseteq H$ happens to be bias-enveloped in line 5, the Incomplete Bias Envelopment is meant to be successful, for the size of H is, by hypothesis, less than or equal its stop parameter $k = \lceil \sqrt{n} \rceil$. $\square$

6.2 Complexity analysis

As each subset C_i has $O(n)$ pairs of vertices and there are $O(\sqrt{n})$ such subsets, the number of pairs that are bias-enveloped in the first phase of the algorithm is $O(n\sqrt{n})$. All Bias Envelopments, in this phase, are complete and take $O(n^2)$ time to be executed, which yields a subtotal of $O(n^{3.5})$ time in the whole first phase.

The number of pairs that are only submitted to Bias Envelopment in the second phase is $O(n^2) - O(n\sqrt{n}) = O(n^2)$ pairs. Each Bias Envelopment is, now, an *Incomplete* one with parameter $k = \lceil \sqrt{n} \rceil$. Because the time complexity of each Incomplete Bias Envelopment with parameter k is $O(nk)$, the total time complexity of the whole second phase of the algorithm is $O(n^3 k) = O(n^{3.5})$.

Thus, the overall time complexity of the Balanced Subsets algorithm is $O(n^{3.5}) + O(n^{3.5}) = O(n^{3.5})$.

7 The Monte Carlo HSSP Algorithm

A yes-biased Monte Carlo algorithm for a decision problem is one which always gives the right answer when it answers *yes* (a certificate is given), whereas a *no* answer may be wrongly given (with probability no greater than a fixed ϵ, however). In other words, it always answers *no* if the correct answer is *no* (for it cannot create a false *yes*-certificate) and answers *yes* with probability at least $p = 1 - \epsilon$ whenever the correct answer is *yes*.

In order to gather some intuition, let us suppose the input has a SHS H with h vertices or more.

What would be, in this case, the probability $\overline{p}_1$ that a random pair of vertices $\{x, y\} \in V$ is *not* contained in H? Clearly,

$$\overline{p}_1 \leq 1 - \frac{h(h-1)}{n(n-1)}.$$

What about the probability $\overline{p}_t$ that t random pairs of vertices fail to be contained in H? It is easy to see that

$$\overline{p}_t \leq \left(1 - \frac{h(h-1)}{n(n-1)}\right)^t.$$

Now, still on the hypothesis that there exists a SHS H with at least h vertices, what would be the probability p_t that, after t Bias Envelopment procedures have been run (starting from t randomly chosen pairs of vertices), a SHS have been found? Again, it is quite simple to reach the following expression, which will be vital to the forthcoming reasoning.

$$p_t \geq 1 - \left(1 - \frac{h(h-1)}{n(n-1)}\right)^t. \tag{1}$$

If, instead of obtaining the probability p_t from the expression above, we fix p_t at some desired value $p = 1 - \epsilon$, we will be able to calculate the minimum integer value of h_t (which, for simplicity, will denote h as a function of t) that satisfies the inequality 1. This value h_t is such that the execution of t independent Bias Envelopment procedures (on t random pairs) *is sufficient to find a SHS of the input instance with probability at least p, in case there exists any with h_t vertices or more* (see equation 2):

$$h_t = \left\lfloor \frac{1 + \sqrt{1 + 4(n^2 - n)(1 - (1-p)^{1/t})}}{2} \right\rfloor. \tag{2}$$

However, we want an algorithm that finds a SHS with some fixed probability p *in case there exists any, no matter its size.* As h_t decreases with the growth of t, the following question arises: how many random pairs do we need to submit to Bias Envelopment in order to achieve that? The answer is rather simple: the minimum integer t' such that $h_{t'} = 2$, for 2 is the shortest possible size of a SHS!

Determining t' comes straightforwardly from equation 2 (please refer to Subsection 7.2 for the detailed calculations):

$$t' = \frac{\ln(1-p)}{\ln\left(1 - \frac{2}{n(n-1)}\right)} = \Theta(n^2). \tag{3}$$

Once the number t' of Bias Envelopment procedures that need to be undertaken on randomly chosen pairs of vertices is $\Theta(n^2)$ and the time complexity of each Bias Envelopment is $O(n^2)$, so far we seem to have been lead to an $O(n^4)$ randomized algorithm, which is all the most undesirable, for we could already solve the problem *deterministically* with less asymptotical effort (see, for example, the previous Section 6)!

But now we have come to a point where the *incomplete* version of the Bias Envelopment procedure will play an important part as far as time saving goes. We show that, by the time the t-th Bias Envelopment is run, its incomplete version with stop parameter $k = h_{t-1}$ serves exactly the same purpose as its complete version would do.

Lemma 7. *In order to find a SHS with probability p, in case there exists at least one with h_t vertices or more, the t-th Bias Envelopment does not need to go further after the size of the candidate set has exceeded h_{t-1}.*

Proof. Two are the possibilities regarding the input: (i) there is a SHS with more than h_{t-1} vertices; or (ii) there is no SHS with more than h_{t-1} vertices.

If (i) is true, then no more than $t - 1$ Bias Envelopments would even be required to achieve that (by the definition of h_t). Hence the t-th Bias Envelopment can stop as early as it pleases.

If (ii) is the case, then an Incomplete Bias Envelopment with stop parameter $k = h_{t-1}$ is meant to give the exact same answer as the complete Bias Envelopment would, for there are no SHSs with more than h_{t-1} vertices to be found (by hypothesis).

Whichever the case, thus, such an Incomplete Bias Envelopment is perfectly sufficient. $\square$

Now we can describe an efficient Monte Carlo algorithm which gives the correct answer to the HSSP with probability at least p.

The algorithm's idea is to run several Incomplete Bias Envelopment procedures on randomly chosen initial candidate sets (pairs of vertices). At each iteration t of the algorithm we run an Incomplete Bias Envelopment with stop parameter $k = h_{t-1}$ and either it succeeds in finding a SHS (and the algorithm stops with an *yes* answer) or else it aborts the current envelopment whenever the number of vertices in the SHS candidate exceeds the h_{t-1} threshold. (In this case, Lemma 7 grants its applicability.) For the first iteration, the stop parameter k is set to $h_0 = n - 1$, as it will correspond to a complete Bias Envelopment. At the end of each iteration, the value of h_t is then updated (see equation 2), which makes it progressively decrease throughout the iterations until it reaches 2 (the minimum size allowed for a homogeneous set), which necessarily happens after $\Theta(n^2)$ iterations (see equation 3).

The pseudo-code for this algorithm is in Figure 8.

Monte_Carlo_HSSP_Algorithm $(G_1(V, E_1), G_2(V, E_2), p)$

```
1.    $h \leftarrow |V| - 1$
2.    $t \leftarrow 1$
3.    while $h \geq 2$
3.1.      Let $(v_1, v_2)$ be a random pair of distinct vertices of $V$.
3.2.      if Incomplete_Bias_Envelopment$(G_1, G_2, \{v_1, v_2\}, h)$ = yes
              return yes.
3.3.      $h \leftarrow \lfloor (1 + \sqrt{1 + 4(|V|^2 - |V|)(1 - (1-p)^{1/t})})/2 \rfloor$
3.4.      $t \leftarrow t + 1$
4.    return no.
```

Figure 8: A yes-biased Monte Carlo algorithm for the HSSP

7.1 Proof of correctness / completeness

Theorem 8. *The Monte Carlo HSSP algorithm correctly answers whether there exists a sandwich homogeneous set in the input graphs with probability at least p.*

Proof. If the algorithm returns *yes*, then it is the consequence of having found a set $H \subset V$, with $|H| \geq 2$, such that the bias set of H is empty, which makes a valid SHS out of it. In other words, if the correct answer is *no* then the algorithm gives a correct *no* answer with probability 1.

If the correct answer is *yes*, we want to show that it gives a correct *yes* answer with probability at least p. Let h^* be the size of the largest SHS of the input instance. As $h_0 = n - 1$ and the algorithm only answers *no* after h_t has lowered down to 2, there must exist an index d such that $h_d \leq h^* \leq h_{d-1}$. From the definition of h_t we know that, on the hypothesis that the input has a SHS with h_t vertices or more, t Bias Envelopments are sufficient to find one, with probability at least p. As, by hypothesis, there *is* a SHS with $h^* \geq h_d$ vertices, then d independent Bias Envelopments are sufficient to find a SHS with probability p. So, it is enough to show that this quota of d Bias Envelopments is achieved. It is true that Incomplete Bias Envelopments that stop *before* the candidate set has reached the size of h^* cannot find a SHS with h^* vertices. Nevertheless, the first d iterations alone perform this minimum quota of Bias Envelopments. Because h^* is the size of the largest SHS, the fact of being *incomplete* simply does not matter for these first d Bias Envelopments, none among which being allowed to stop before the size of the candidate set has become larger than $h_{d-1} \geq h^*$. □

7.2 Complexity analysis

The first iteration of the algorithm runs the complete Bias Envelopment in $O(n^2)$ time [1]. (Actually, a more precise bound is $O(m_1 + \overline{m}_2)$, as seen in Section 2. However, as the complexity analysis of the Incomplete Bias Envelopment does not benefit at all from allowing edge quantities, we opted, for the sake of simplicity, to write time bounds only as functions of n.) The remaining iterations take $O(nh_t)$ time each. To analyze the time complexity of the algorithm, we have to calculate

$$\sum_{t=1}^{t'} O\left(nh_{t-1}\right),$$

where t' is the number of iterations in the worst case.

The value of h_t, obtained at the end of iteration t, is defined by equation 2. To calculate t', we replace $h_{t'}$ for 2 and have

$$\left(1 - \frac{2}{n(n-1)}\right)^{t'} = 1 - p, \text{ finally reaching}$$

$$t' = \frac{\ln(1-p)}{\ln\left(1 - \frac{2}{n(n-1)}\right)}.$$

For $0 < x < 1$, it is known that

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \cdots .$$

Consequently,

$$t' = \frac{\ln(1-p)}{-\frac{2}{n(n-1)} - \frac{1}{\Theta(n^4)}} = \Theta(n^2).$$

Now, we will show that $h(h-1)/n(n-1) \geq h^2/2n^2$. This result is useful to simplify some calculations. We have

$$\frac{n}{n-1} \cdot \frac{h-1}{h} \cdot \frac{h^2}{n^2} = \frac{h(h-1)}{n(n-1)}, \text{ and}$$

$$\frac{h-1}{h} \cdot \frac{h^2}{n^2} \leq \frac{h(h-1)}{n(n-1)}.$$

Since $h \geq 2$,

$$\frac{h^2}{2n^2} \leq \frac{h(h-1)}{n(n-1)}.$$

To calculate the total time complexity, we replace $h(h-1)/n(n-1)$ for $h^2/2n^2$ and p_t for the fixed value p in equation 1, and have

$$\left(1 - \frac{h^2}{2n^2}\right)^t \geq 1 - p,$$

$$\frac{h^2}{2n^2} \leq 1 - (1-p)^{1/t}, \text{ and}$$

$$h \leq \Theta(n)\sqrt{1 - (1-p)^{1/t}}.$$

It is well known that

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots .$$

Consequently, for $x > 1$,

$$e^{1/x} = 1 + 1/\Theta(x).$$

Using this approximation, we have

$$h \leq \Theta(n)\sqrt{1 - (1 + 1/\Theta(t))} = \Theta(n)/\Theta(\sqrt{t}).$$

The total time complexity of the algorithm is

$$\sum_{t=1}^{\Theta(n^2)} O\big(nh_t\big) = \sum_{t=1}^{\Theta(n^2)} \frac{O(n^2)}{O(\sqrt{t})} = O(n^2) \sum_{t=1}^{\Theta(n^2)} 1/O(\sqrt{t}).$$

Using elementary calculus, we have

$$O(n^2) \sum_{t=1}^{\Theta(n^2)} 1/O(\sqrt{t}) = O(n^3),$$

which stands for the overall total time complexity of the Monte Carlo HSSP algorithm.

8 The Harmonic Series algorithm

This section introduces a new algorithm for the HSSP (see Figure 9), named the *Harmonic Series* (*HS*) algorithm due to its peculiar complexity analysis.

The main idea of the algorithm presented herein is pretty much the same as in the previously seen EBE (Section 2) and BS (Section 6) algorithms: to submit each and every pair of input vertices $\{x, y\} \subset V$ to Bias Envelopment procedures in order to find out whether they are contained in a SHS of the input graphs.

The common feature of both the HS algorithm which follows and the Balanced Subsets algorithm given in Section 6 is the ability to benefit from unsuccessful Bias Envelopments, differently from what happened in other

```
**Harmonic_Series_HSSP_Algorithm** $(G_1(V,E_1), G_2(V,E_2))$

1.      for $t = 1$ to $\lfloor n/2 \rfloor$ do
1.1.        for each $\{v_i, v_j\} \subset V$ such that $\mathrm{cd}_n(i,j) = t$ do
1.1.1.          if Incomplete_Bias_Envelopment $(G_1, G_2, \{v_i, v_j\}, \lfloor n/t \rfloor)$ = yes
                    return yes.
2.      return no.
```

Figure 9: The Harmonic Series algorithm

algorithms (e.g. EBE and 2-P algorithms), where the envelopments were totally independent from one another.

The HS algorithm establishes a tricky order in which the vertex pairs are examined, so that the cumulative knowledge brought up by earlier envelopments can be used to speed up later ones.

We define the *n-circular distance* $\mathrm{cd}_n(x,y)$ between two numbers $x, y \in \{1, ..., n\}$ as the number of edges in the minimum path from vertex v_x to vertex v_y over an ordered cycle $v_1, v_2, \ldots, v_n, v_1$ (e.g. $\mathrm{cd}_8(1,4) = 3$, $\mathrm{cd}_8(1,7) = 2$). The n-circular distance between x and y can be easily obtained by the following expression:

$$\mathrm{cd}_n(x,y) = \min\{x - y \text{ (modulo } n), \ y - x \text{ (modulo } n)\}. \tag{4}$$

Having indexed all input vertices as v_i, with i ranging from 1 to n, we say two vertices v_i and v_j are *k-distant-labelled* if and only if $\mathrm{cd}_n(i,j) = k$. Clearly, two vertices can be at most $\lfloor n/2 \rfloor$-distant labelled.

The HS algorithm proceeds as follows: the $n(n-1)/2$ vertex pairs will be submitted to Bias Envelopment in $\lfloor n/2 \rfloor$ turns with n pairs each. The first turn submits to Bias Envelopment all pairs of 1-distant-labelled vertices $(\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_n, v_1\})$, the second turn submits all pairs of 2-distant-labelled vertices $(\{v_1, v_3\}, \{v_2, v_4\}, \ldots, \{v_n, v_2\})$ and so on, with further t-th turns always submitting to Bias Envelopment all n pairs of t-distant-labelled input vertices.

The point is, by the time the n Bias Envelopments of the t-th turn will take place, it is known already that no SHSs sets exist containing a pair of vertices $\{v_i, v_j\}$ such that $\mathrm{cd}_n(i,j) < t$ (once all such pairs will have already been unsuccessfully bias-enveloped in earlier turns). This knowledge bounds the size of possibly existing SHSs in this turn at $\lfloor n/t \rfloor$. Consequently, all Bias Envelopments in the t-th turn need not continue whenever the size of the current candidate set has exceeded this $\lfloor n/t \rfloor$ bound, which calls for Incomplete Bias Envelopments with stop parameter $k = \lfloor n/t \rfloor$.

8.1 Proof of correctness / completeness

Theorem 9. *The Harmonic Series algorithm correctly solves the HSSP.*

Proof. Once again, if the algorithm returns *yes*, then it has successfully found a set $H \subset V$, with $2 \leq |H| < |V|$, such that the bias set of H is empty (Bias Envelopment yes-answer condition). Thus, H is a valid SHS.

Now, suppose the input has a SHS H. Let $d = \min\{\mathrm{cd}_n(i,j) \mid v_i, v_j \in H\}$ stand for the minimum n-circular distance between any two vertex indexes in H and let $v_r, v_s \in H$ be two d-distant-labelled vertices. Now, suppose H has more than $\lfloor n/d \rfloor$ vertices. Then, by the pigeon hole principle, there must be two vertices v_x and v_y such that $\mathrm{cd}_n(x,y) < d$. This is a contradiction, for d is minimum. Hence, the cardinality of H is no larger than $\lfloor n/d \rfloor$. If the algorithm still has not found any SHSs by the time $\{v_r, v_s\}$ turns out to be the initial candidate set (which must happen during the d-th turn of Bias Envelopments), the Incomplete Bias Envelopment with stop parameter $k = \lfloor n/d \rfloor$ which will then take place is destined to find a SHS containing v_r and v_s (for there is one, by hypothesis — H — with k vertices or less). The algorithm, in this case, answers *yes*, as it was supposed to. $\square$

8.2 Complexity analysis

In the worst case, the algorithm performs $\lfloor n/2 \rfloor$ turns of n Incomplete Bias Envelopments each. In the t-th turn, the stop parameter k of each Bias Envelopment is set to $\lfloor n/t \rfloor$. As the time complexity of an Incomplete Bias Envelopment with stop parameter k is $O(nk)$, the complexity of the whole t-th turn is $n \cdot O(n\lfloor n/t \rfloor) = O(n^3/t)$.

Thus, the overall time complexity of the whole Harmonic Series algorithm is:

$$\sum_{t=1}^{\lfloor n/2 \rfloor} O\left(\frac{n^3}{t}\right) = O\left(n^3 \sum_{t=1}^{O(n)} O\left(\frac{1}{t}\right)\right) = O\left(n^3 \log n\right).$$

9 Enemies

The next new algorithm, presented in Section 10, is so to say smarter than the Harmonic Series one (and in practice reasonably faster, despite sharing the same asymptotical time complexity).

Before we focus on it, we introduce the definition of *enemy vertices* (or simply *enemies*), for this very useful concept will be the ground basis of the algorithm to come.

Let $G_1(V, E_1), G_2(V, E_2)$ be the input instance of a HSSP. Two vertices x, y are said to be *enemies* of each other if no sandwich homogeneous sets exist containing both x and y.

We define the *enmity graphs* of (G_1, G_2) as graphs $G_N(V, E_N)$ which have the same vertices of the input graphs and where an edge (x, y) might exist only if x and y are enemies.

A *trivial* enmity graph is one that is empty. It reveals a complete lack of information about enmity relationships among the HSSP input vertices. A *final* enmity graph, on the other hand, is one that reflects *all* enmity relationships of the input graphs' vertices. It clearly answers the HSSP, once any two vertices that are *not* adjacent (in the final enmity graph) must be non-enemies, i.e. must be contained in some SHS of (G_1, G_2).

An enmity graph of a HSSP instance, thus, can be any sandwich graph of its trivial and final enmity graphs; it is one which shows some (but perhaps not all) of the existing enmity relationships among the instance's pairs of vertices. Lemma 10, which is given next, highlights the extreme usefulness of enmity graphs.

(We recall that an *independent set* is a subset S of V which contains only pairwise non-adjacent vertices.)

Lemma 10. *Given an enmity graph G_N of (G_1, G_2), the maximum size of any SHS of (G_1, G_2) is bounded by the size of the maximum independent set of G_N.*

Proof. Let d be the size of the maximum independent set of the enmity graph G_N at hand. Now suppose the pair (G_1, G_2) admits a sandwich homogeneous set H with more than d vertices. The subgraph of G_N induced by H cannot be totally disconnected (or the vertices of H would constitute an independent set of G_N with more than d vertices), so there must be at least two vertices $x, y \in H$ which are connected by an edge in G_N, which is only the case if x and y are enemies. But this is a contradiction, for there is a sandwich homogeneous set — H, by hypothesis — which contains both x and y. $\square$

To illustrate the usefulness of the enmity graphs, we shall take a glance back at the Harmonic Series algorithm from Section 8. The main difference between the HS algorithm and the former Cerioli *et al*'s EBE algorithm, for instance, is that, whereas the EBE neglects the fact that two already bias-enveloped vertices are enemies (and will never belong both to a same SHS), the HS continuously uses this knowledge to constraint the length of further envelopments (once, by Lemma 10, the size of possibly existing SHSs is more and more restricted by the increasing number of known enmity relationships).

In other words, what all algorithms based on the Bias Envelopment do is to start from a trivial enmity graph and try to answer *yes* along the way or *no* after a complete enmity graph has been reached. Whereas the EBE algorithm simply did not care about enmity graphs, the knowledge they bring allows the HS algorithm to save precious time.

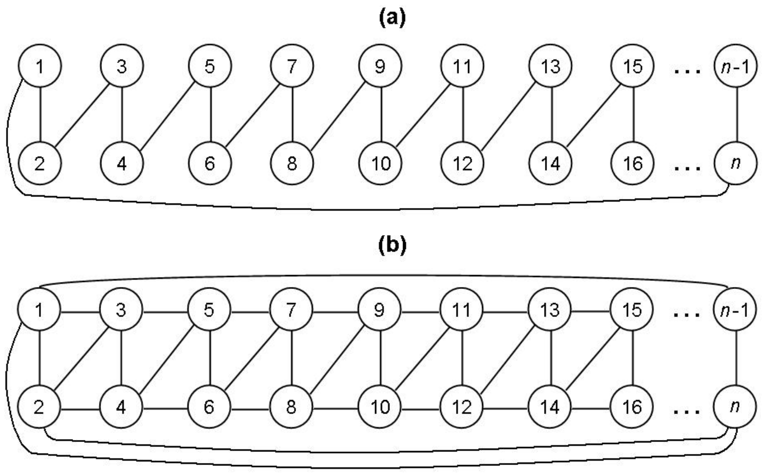

Figure 10: Two enmity graphs during the HS algorithm

Figures 10(a) and 10(b), respectively, show the enmity graphs at hand by the end of the first and the second Bias Envelopment turns during the execution of the HS algorithm on some input instance whatsoever (under the hypothesis that no SHS had yet been found up to that point).

In the HS algorithm, n new enmity relationships are brought to light (i.e. n new edges are added to G_N) at the end of each turn of Bias Envelopments, so that the number of vertices in possibly existing (albeit still undiscovered) SHSs is limited to the size of the maximum independent set in the current G_N (which, as seen in Section 8, is easy to calculate *given that particular envelopment order*).

The following Growing Cliques algorithm, given in Section 10, differs from the HS algorithm in that it applies the *best* such order, i.e. the order (in which edges are added to the subjacent enmity graph) which minimizes the average Bias Envelopment length.

10 The Growing Cliques algorithm

It is clear that, in all HSSP algorithms based on the Bias Envelopment procedure, the tighter the known upper bound for the size of searched-for SHSs at a certain point, the earlier all remaining envelopments will be allowed to stop. In this sense, the development of an optimized envelopment-based algorithm has to find the answer to the question: which is the best order in which vertex pairs can be submitted to Bias Envelopment?

Given that unsuccessful Bias Envelopments reveal enmity relationships (in other words, add edges to the subjacent enmity graph G_N) likely to tighten the upper bound to the cardinality of possibly existing SHSs (according to Lemma 10), the aforementioned question reads: what is the envelopment order that continuously minimizes the size of the maximum independent set of G_N?

The answer is quite simple and leads directly to the algorithm depicted by Figure 11: the one in which disjoint cliques of increasing size are assembled — so that the size of the minimum *clique cover* is as small as possible. (We recall that a clique cover is a collection of cliques such that each and every vertex of the graph belongs to precisely one clique. Clearly, the size of the minimum clique cover is an upper bound for the size of the maximum independent set).

The *Growing Cliques* (*GC*) algorithm, as well as the HS algorithm, submits each and every pair of input vertices to Bias Envelopment.

In the GC algorithm, still the Bias Envelopments are organized in turns. The first turn, which starts with a trivial (empty) enmity graph G_N, aims to join G_N's current maximal cliques of size 1, pairwisely, in order to assemble $\lfloor n/2 \rfloor$ cliques of size 2 (and possibly a remainder clique of size 1, in case n is odd). Figure 12(a) shows the partial enmity graph G_N after all Bias Envelopments of the first GC algorithm's turn have been run. It is easy to see that the size of the maximum independent set of G_N is now $O(n/2)$. The second turn of Bias Envelopments will join G_N's cliques of size 2, pairwisely, ending up with $O(n/4)$ cliques of size 4 (plus at most one remainder clique of lesser size). All further envelopment turns continue the same way. Figures 12(b) and 12(c), respectively, show the partial enmity graph G_N at the end of the second and third turns of Bias Envelopments undertaken by the GC algorithm. This clique-assembling process goes on and forth until either a Bias Envelopment answers *yes* or the last turn ends up with all vertices tied together in a big, single clique of the enmity graph, which yields a *no* answer to the HSSP instance at hand.

The point is, all Bias Envelopments in the t-th turn need not go beyond the threshold of $\lceil n/2^{t-1} \rceil$, which bounds the size of the maximum independent set of the enmity graph during this turn (and, as a consequence, the size of any SHS to be possibly found thenceforth).

10.1 Proof of correctness / completeness

Theorem 11. *The Growing Cliques algorithm correctly solves the HSSP.*

Proof. As usual, an *yes* answer is only given if a Bias Envelopment procedure has successfully found a valid SHS.

Now, suppose the input has a sandwich homogeneous set H with size h. In order to prove that, under this hypothesis, the algorithm does answer *yes*, it is enough to show that: (i) the algorithm cannot answer *no* before each and every pair $\{x, y\} \subseteq H$ have been submitted to an Incomplete

Growing_Cliques_HSSP_Algorithm $(G_1(V,E_1), G_2(V,E_2))$

```
1.          $C \leftarrow \{\{v_i\} \mid v_i \in V\}$   //initializes the cover with singletons.
2.          while $|C| > 1$
2.1.            Index all cliques in $C$ from 1 to $|C|$.
2.2.            for each clique $C_j \in C$ such that $j$ is even do
2.2.1.              for each pair $\{x,y\}$ such that $x \in C_j,\ y \in C_{j-1}$ do
2.2.1.1.                if Incomplete_Bias_Envelopment $(G_1, G_2, \{x,y\}, |C|) =$ yes
                            return yes.
2.2.2.              $C \leftarrow (C \cup \{C_j \cup C_{j-1}\}) \setminus \{C_j, C_{j-1}\}$  //joins $C_j$ and $C_{j-1}$.
3.          return no.
```

Figure 11: The Growing Cliques algorithm

Bias Envelopment; and (ii) whenever the first such pair $\{x,y\} \subseteq H$ is bias-enveloped, the envelopment does not stop before a SHS has successfully been found.

To show (i), we argue that the algorithm can only stop with a *no* answer if the condition of the main loop (line 2) is not anymore satisfied, which means that a single clique, covering all vertices of G_N, has been reached. In particular, each $x, y \in H$ must have been connected by an edge in this final G_N, but such an edge can only be added to G_N after the set $\{x,y\}$ has been bias-enveloped.

To show (ii), let us suppose, by contradiction, that the Incomplete Bias Envelopment which started with initial candidate $\{x,y\} \subseteq H$ stopped without having found any SHSs. Once its stop parameter had been set (see line 2.2.1.1 in Figure 11) at the current cardinality of the set C (i.e. the number of cliques, all maximal by construction, in the current enmity graph) and it has not found H, which has h vertices, it comes that the number of maximal cliques in the partial enmity graph at that moment was less than h. But this is a contradiction, for Lemma 10 states that the size of the maximum independent set (which cannot be greater than the number of cliques in a clique cover) is a ceiling for the size of any SHSs of that input instance. $\square$

10.2 Complexity analysis

As each turn of Bias Envelopments divides the size of the clique cover C by 2, the GC algorithm stops, in the worst case, after $O(\log n)$ turns. This is the maximum number of iterations of the algorithm's main loop (line 2).

The time complexity of each turn of Bias Envelopments (lines 2.1 to 2.2.2) is given by the number of Incomplete Bias Envelopments that are undertaken multiplied by the complexity of each Incomplete Bias Envelop-

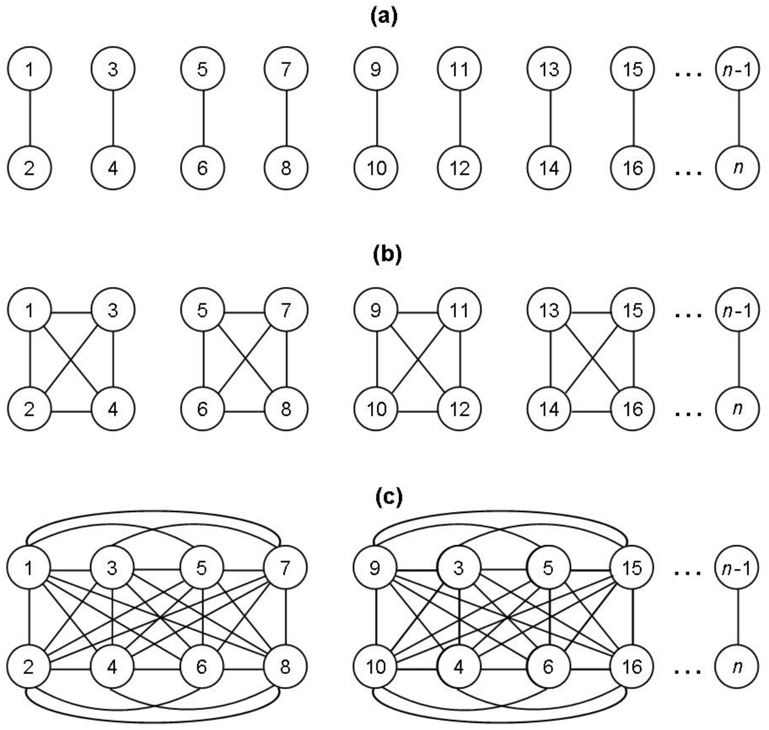

Figure 12: Enmity graphs during a GC algorithm execution

ment. In the t-th turn, the number of envelopments is $O(2^{t-2}n)$ and the stop parameter is $n/2^{t-1}$. As the time complexity of an Incomplete Bias Envelopment with stop parameter k is $O(nk)$, it comes that each Incomplete Bias Envelopment in the t-th turn takes $O(n \cdot n/2^{t-1}) = O(n^2/2^{t-1})$ time. Hence, the total time complexity of the t-th turn is $O(2^{t-2}n) \cdot O(n^2/2^{t-1}) = O(n^3/2) = O(n^3)$, regardless of t.

Thus, the time complexity of the GC algorithm is $\sum_{t=1}^{O(\log n)} O(n^3) = O(n^3 \log n)$.

A different way of benefiting from the enmity graphs is exemplified in the following Section 11, where we present a Las Vegas randomized algorithm that solves the HSSP in $O(n^3)$ expected time, saving a $\log \frac{m}{n}$ factor compared to the best deterministic algorithms' worst-case time complexities. It achieves this by using a data structure that explicitly keeps track of one evolving, dynamically built enmity graph.

11 The Las Vegas HSSP algorithm

A Las Vegas algorithm for a decision problem is one which always gives the correct answer, but whose running time is a random variable (it somehow depends on random choices). Its efficiency is therefore evaluated in terms of its *expected* running time.

While studying both the Harmonic Series and the Growing Cliques algorithms, in Sections 8 and 10, it became clear that the performance of such algorithms depends largely on the chosen sequence of vertex pairs to be put under Bias Envelopment. Even if the best deterministic such sequence is opted for, though, it has been seen that a worst-case complexity no better than $O(n^3 \log n)$ could be achieved. In this section, we show how it is possible to save some relevant worst-case running time by making use of a randomized order of envelopment submittance.

The Las Vegas algorithm introduced in the present section also belongs to the family of envelopment-based HSSP algorithms. Once again, input vertex pairs are submitted to Bias Envelopment in order to find out whether any of them belongs to some SHS, and still the enmity relationships disclosed by unsuccessful Bias Envelopments will play a fundamental role.

In the HS and GC algorithms, the subjacent "enmity graph" only existed virtually (no edge or enmity information was stored anywhere), as a means of providing theoretically-calculated bounds for the length of further envelopments. This time, however, the enmity graph will actually *exist.* Instead of obtaining decreasing upper bounds to the size of searched-for SHSs (which would constitute appropriate stop parameters to Incomplete Bias Envelopments), the enmity graph will be now randomly built (and its adjacency matrix explicitly stored), providing an efficient way of achieving early envelopment stops.

A variation of the Incomplete Bias Envelopment, which is referred to as *Enmity-Constrained Bias Envelopment*, is shown in Figure 13.

The Enmity Constrained Bias Envelopment is also *incomplete*, in the sense that it may, as well, answer *no* before the candidate set has embodied the whole input vertex set V. Instead of being provided with a stop parameter k which would limit the size of the candidate set, the envelopment will stop whenever the candidate set contains two vertices that are known enemies. (Notice that, now, the known enmity relationships between vertices are explicitly upheld by G_N, which is passed as a parameter).

The Las Vegas algorithm we present (see Figure 14) is similar to Cerioli *et al.*'s EBE algorithm [1], except that (i) Enmity-Constrained Bias Envelopments will replace the normal (complete) Bias Envelopments thereof; and (ii) by having the input vertex pairs follow a *random* envelopment submittance sequence, an efficient worst-case expected running time will be achieved.

```
Enmity_Constrained_Bias_Envelopment
($G_1(V, E_1), G_2(V, E_2), H_1, G_N(V, E_N)$)

1.      $H \leftarrow H_1$
2.      while $|H| < |V|$ do
2.1.      if $B(H) = \varnothing$
            return yes.
2.2.      else
            for each vertex pair $\{x, y \mid x \in B(H),\ y \in H \cup B(H)\}$ do
2.2.1         if $(x, y) \in E_N$
                return no.
2.2.2.        $H \leftarrow H \cup B(H)$
3.      return no.
```

Figure 13: The Enmity-Constrained Bias Envelopment procedure

```
Las_Vegas_HSSP_Algorithm ($G_1(V, E_1), G_2(V, E_2)$)

1.    Initialize $G_N(V, E_N)$ with an empty edge set $E_N$.
2.    Choose a random order $\Gamma$ of all pairs of vertices $\{x, y\} \subset V$.
3.    for each vertex pair $\{x, y\}$ in $\Gamma$ do
3.1.    if Enmity_Constrained_Bias_Envelopment($G_1, G_2, \{x, y\}, G_N$) = yes
          return yes.
3.2.    else $E_N \leftarrow E_N \cup \{(x, y)\}$
4.    return no.
```

Figure 14: A Las Vegas algorithm for the HSSP

11.1 Proof of correctness / completeness

Theorem 12. *The Las Vegas algorithm given in Figure 14 correctly solves the HSSP.*

Proof. As it is an envelopment-based algorithm, it is needless to reiterate that an *yes* answer is only possible if a SHS has been correctly found. On the other hand, if the algorithm answers *no*, it is the consequence of having already submitted all input vertex pairs to Bias Envelopments, none of which leading to a positive result. Each Enmity-Constrained Bias Envelopment, in its turn, only aborts with a *no* answer after a pair of enemy vertices has become part of the candidate set, which, by the definition of enemies, witnesses the non-existence of SHSs containing that candidate. □

11.2 Complexity analysis

The number of envelopments in the worst case is $n(n-1)/2 = O(n^2)$. The time complexity of the t-th Enmity Constrained Bias Envelopment is $O(nr_t)$, where r_t is the number of vertices in the candidate set H by the first time two enemies appear among its elements. We want to calculate the expected value of r_t.

By the time the t-th Enmity Constrained Bias Envelopment starts, $t-1$ pairs of vertices will have already been *no*-answered by preceding envelopments, which means that the current partial enmity graph will have exactly $t-1$ edges. As these $t-1$ edges have been added to E_N *following a random order*, an edge between two vertices $x, y \in H$ has the probability $p_t = 2(t-1)/n(n-1)$ of belonging to E_N. The number q_t of pairs of vertices that will be contained in H, by the time the first pair of enemies shall belong to H in the t-th envelopment, is a geometric random variable with success probability $p_t = O(t/n^2)$. It is well known that such a variable has an expected value of: $E[q_t] = 1/p_t = O(n^2/t)$.

As the number r_t of *vertices* in H that corresponds to q_t *pairs of vertices* in H is $r_t = O(\sqrt{q_t})$, the searched-for expected value of r_t is $E[r_t] = O(\sqrt{E[q_t]}) = O(\frac{n}{\sqrt{t}})$.

The expected running time of the t-th Enmity Constrained Bias Envelopment will thus be $O(nr_t) = O(n^2/\sqrt{t})$, giving rise to the following expected time for the whole algorithm:

$$\sum_{t=1}^{O(n^2)} O\left(\frac{n^2}{\sqrt{t}}\right) = O(n^2) \sum_{t=1}^{O(n^2)} O\left(\frac{1}{\sqrt{t}}\right) = O(n^3).$$

12 The Quick Fill algorithm

The algorithm presented in this section, which is referred to as the *Quick Fill* (*QF*) algorithm, is so to say the sum of three previous algorithms' important achievements: (i) the progressive decrease in the length of the envelopments, made possible by some well-chosen enmity-checking sequence — as in the Growing Cliques algorithm (Section 10); (ii) the lesser number of $O(\min\{m_1, \overline{m}_2\})$ envelopments that need to be run — as in the Two-Phase algorithm (Section 4) — instead of other algorithms' worst-case $\Theta(n^2)$, which was made possible by the proper use of the bias graph supported by Corollary 4; and (iii) the ability of saving time by stopping an envelopment as early as there are two enemies belonging to the current candidate set, which can be kept track of by means of actually having an appropriate data structure to store the continuously evolving enmity graph data — as in the Las Vegas algorithm (Section 11).

The central inspiration for the QF algorithm arises from the following lemma:

Lemma 13. *Let $G_1(V, E_1), G_2(V, E_2)$ be an input instance for the HSSP and let $u, v \in V$ be two enemies. Then, every two vertices $x, y \in V$ such that there is a path, in the bias graph G_B, from vertex $[x, y]$ to vertex $[u, v]$, are also enemies of each other.*

Proof. Lemma 13 states that, if there is no SHS which contains $\{u, v\} \subset V$ and vertex $[x, y] \in V_B$ reaches vertex $[u, v] \in V_B$, then there is no SHS which contains $\{x, y\} \subset V$ either. From the construction of the bias graph, it is known that all edges in E_B are of the form $([x, y], [x, b])$ [2] and one such edge only belongs to E_B if b is a bias vertex of $\{x, y\}$. Now, it comes from Theorem 1 that all SHSs that contain $\{x, y\}$ must also contain $\{b\}$. Hence, if there is no SHS containing $\{x, b\}$, there cannot be any containing $\{x, y\}$ either. In other words, if $\{x, b\}$ are enemies then $\{x, y\}$ are enemies as well.

It is easy to see that this reasoning holds true for paths, in the bias graph, of whichever length. Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be a path in G_B, where α_i stands for a vertex $[u_i, v_i] \in V_B$. If there exists no SHS which contains $\{u_k, v_k\}$ (i.e. the labelling vertices of α_k), then the above reasoning shows there cannot exist any containing $\{u_{k-1}, v_{k-1}\}$, which prevents, in its turn, the existence of any SHSs containing u_{k-2}, v_{k-2} and so on and so forth. This is enough to show that the two labelling vertices u_i, v_i of $\alpha_i, 1 \leq i \leq k$, are indeed enemies of one another. □

In order to introduce the Quick Fill algorithm, let us think about the Las Vegas algorithm (see Section 11) for a while. There is an enmity graph, initially empty, whereto an edge is added every time an unsuccessful envelopment is run (disclosing a new enmity relationship between two input vertices). The storage of these enmity relationships' data allows each Enmity-Constrained Bias Envelopment procedure to stop as early as a pair of enemies takes part in the candidate set. Thus, the more edges have been added to the enmity graph, the sooner an envelopment is expected to stop.

The first, simplest improvement brought about by Lemma 13 would therefore be the ability to add to the enmity graph *more than one edge* per unsuccessful envelopment. Actually, after a pair $\{u, v\} \subset V$ has been discovered not to be contained in any SHSs, one edge could be added, in the enmity graph, between any two vertices $x, y \in V$ such that $[x, y]$ reaches $[u, v]$ in the bias graph associated to that input instance.

The Quick Fill algorithm, however, does not even need to worry about these extra edge additions, for it will accomplish this intended faster completion of the enmity graph as a simple consequence of its envelopment strategy, as follows.

[2]or $([x, y], [y, b])$, as well.

Quick_Fill_HSSP_Algorithm $(G_1(V, E_1), G_2(V, E_2))$

```
1.          Construct the bias graph $G_B$ of $(G_1, G_2)$.
2.          $S \leftarrow \varnothing$   //list of sink representatives.
3.          for each end strongly connected component $P_i$ of $G_B$ do
3.1.           Add to $S$ the first element of $P_i$.
4.          Let $G'_B$ be a copy of $G_B$ with all edges reversed.
5.          Obtain a depth-first search forest $F$ of $G'_B$,
                      with all trees rooted on elements of $S$.
6.          for each vertex $[x, y] \in G_B$ do
6.1.           $s(x, y) \leftarrow [u, v]$, where $[u, v]$ is the root of the tree
                         whereto $[x, y]$ belongs, in $F$.
7.          Initialize $G_N(V, E_N)$ with an empty edge set $E_N$.
8.          $C \leftarrow \{\{v_i\} \mid v_i \in V\}$   //initializes the clique cover.
9.          while $|C| > 1$
9.1.           Index all cliques in $C$ from 1 to $|C|$.
9.2.           for each clique $C_j \in C$ such that $j$ is even do
9.2.1.            for each pair $\{x, y\}$ such that $x \in C_j,\ y \in C_{j-1}$ do
9.2.1.1.             Let $[u, v]$ be the reachable sink $s(x, y)$.
9.2.1.2.             if $(u, v) \in E_N$
                        go to 9.2.1.5.
9.2.1.3.             else if Enmity_Constrained_Bias_Envelopment
                              $(G_1, G_2, \{u, v\}, G_N) =$ yes
                        return yes.
9.2.1.4.             $E_N \leftarrow E_N \cup \{(u, v)\}$
9.2.1.5.             $E_N \leftarrow E_N \cup \{(x, y)\}$
9.2.2.            $C \leftarrow (C \cup \{C_j \cup C_{j-1}\}) \setminus \{C_j, C_{j-1}\}$   //joins $C_j$ and $C_{j-1}$.
10.         return no.
```

Figure 15: The Quick Fill algorithm

It starts by generating the bias graph $G_B(V_B, E_B)$ of (G_1, G_2) and then, mostly as in the BGC algorithm, it finds G_B's end strongly connected components (ESCC). For each ESCC $P \subseteq G_B$ it assigns a *sink representative*, which can be any vertex $[u, v] \in P$. It is clear that all vertices in the bias graph reach at least one sink representative.

Every vertex $[x, y] \in V_B$ will then be assigned a *reachable sink* $s(x, y) = [u, v] \in V_B$, which is nothing but one of the sink representatives reached by $[x, y]$. This is accomplished by a simple depth-first search run on a copy of G_B with reversed edges, having the sink representatives of G_B as roots of the resulting search forest's trees.

A trivial enmity graph $G_N(V, E_N = \varnothing)$ is created.

The idea, from then on, will be that of following the exact edge-addition order as in the Growing Cliques algorithm, so as to continuously minimize the size of the maximum independent set of the enmity graph at hand. The only difference is, whenever it is time for a pair $\{x, y\} \subset V$ to be enveloped (in order to add edge (x, y) to E_N), the labelling vertex pair $\{u, v\}$ of the reachable sink $s(x, y) = [u, v] \in G_B$ will be enveloped instead, by the end of which the algorithm will have stopped with an *yes* answer or else two edges are added to the enmity graph, namely edges (x, y) and (u, v). If there had already been an edge $(u, v) \in E_N$ by the time $\{x, y\}$ happened to be the next pair in the addition sequence, then the edge (x, y) would be added immediately and effortlessly to E_N, without running any Bias Envelopments at all — wherefrom came the algorithm's name.

No doubt such an envelopment-free $O(1)$ edge addition would mean, in practice, a welcome saving of the algorithm's running time, but is it that the theoretical worst-case time complexity could benefit from this at all?

The answer is yes, and it bears entirely on Lemma 5. As long as only the sink representatives will ever be actually submitted to Bias Envelopment, the number of envelopments that will take place, in the worst case, is $O(\min\{m_1, \overline{m}_2\})$.

In short, the QF can be regarded as either (i) a GC which only runs $O(\min\{m_1, \overline{m}_2\})$ envelopments, in the worst case, instead of the latter's $\Theta(n^2)$; or (ii) a 2-P which only uses early-stops envelopments instead of complete $O(n^2)$ ones.

Figure 15 makes things clearer by depicting QF's pseudo-code.

12.1 Proof of correctness / completeness

Theorem 14. *The Quick Fill algorithm correctly solves the HSSP.*

Proof. The proof is identical to that of the Growing Cliques algorithm (Section 10) plus the fact, supported by Lemma 13, that two vertices $u, v \in V$ need not be put under Bias Envelopment in order to find out they are enemies if it is already known there is a pair of enemies $x, y \in V$ such that $[u, v]$ reach $[x, y]$ in the bias graph of that HSSP instance. $\square$

12.2 Complexity analysis

Each Incomplete Bias Envelopment, in the Growing Cliques algorithm, is provided with a stop parameter which is equal to a known upper bound for the size of the maximum independent set of the subjacent enmity graph during a certain envelopment *turn*.

In this Quick Fill algorithm, Enmity-Constrained Bias Envelopments are used, just as in the Las Vegas algorithm (Section 11), as a means of benefitting as largely as possible from the known enmity relationships among input vertices. Its worst-case analysis, though, is similar to that of the GC algorithm, since, in the worst case, each of its envelopments will only be allowed to stop when the size of the candidate set has reached the very same theoretical bound the GC would use as stop parameter to its corresponding Incomplete Bias Envelopment. Thus, given that each QF envelopment will have exactly the same length of those of the GC, their analysis are, to a great extent, the same.

In both GC and QF algorithms, the envelopments are divided into turns, each one joining cliques of the enmity graph pairwisely together into bigger ones of doubled size. Also, in the t-th turn, the number of envelopments is $O(2^{t-2}n)$ in both cases. What distinguishes both algorithms is, actually, the number of turns that have to be run in the worst case. There, in the GC, the number of turns had to be enough for all $\Theta(n^2)$ input vertex pairs to be bias-enveloped, which yielded a total of $O(\log n)$ turns. Here, the Quick Fill algorithm stops after the first $O(\min\{m_1, \overline{m}_2\})$ envelopments have been run. This allows us to obtain an upper bound t' for the number of envelopment turns it undertakes:

$$\begin{aligned} \sum_{t=1}^{t'} O(2^{t-2}n) &= O(\min\{m_1, \overline{m}_2\}) \\ t' &= O\left(\log \frac{m}{n}\right) \end{aligned}$$

where m here stands for $O(\min\{m_1, \overline{m}_2\})$, i.e. the minimum between the number of mandatory edges and the number of forbidden edges.

Just as in the GC algorithm, the total time complexity of the t-th turn is $O(2^{t-2}n) \cdot O(n^2/2^{t-1}) = O(n^3/2) = O(n^3)$, regardless of t.

Thus, the time complexity of the QF algorithm is

$$\sum_{t=1}^{t'} O(n^3) = \sum_{t=1}^{O\left(\log \frac{m}{n}\right)} O(n^3) = O\left(n^3 \log \frac{m}{n}\right).$$

It is noticeable that the QF algorithm is never slower than the Two-Phase from Section 4, since both run the exact same number of envelopments and still the QF might have them stop earlier than that other algorithm would. A more accurate upper bound for the Quick Fill algorithm is therefore $O(\min\{m_1\overline{m}_2, n^3 \log \frac{m}{n}\})$, which is however far more complicate.

Instance	**EBE** $O(n^4)$	**2-P** $O(m_1\overline{m}_2)$	**BS** $O(n^{3.5})$	**MC** $O(n^3)$	**HS** $O(n^3 \log n)$	**GC** $O(n^3 \log n)$	**LV** $O(n^3)$	**QF** $O(n^3 \log \frac{m}{n})$
$R_{100,10\%M,10\%F}$	1”620	0”265	0”234	0”094	0”125	0”110	0”032	0”268
$R_{200,10\%M,10\%F}$	18”580	0”875	2”985	0”828	1”219	0”859	0”343	0”888
$R_{400,10\%M,10\%F}$	5’56”670	3”813	38”703	6”437	12”610	8”563	3”160	4”105
$R_{100,10\%M,70\%F}$	1”109	0”609	0”234	0”094	0”141	0”125	0”062	0”620
$R_{200,10\%M,70\%F}$	18”407	5”470	2”844	0”750	1”219	0”859	0”375	6”212
$R_{400,10\%M,70\%F}$	6’06”921	out mem	37”531	6”687	12”719	9”297	2”985	out mem
C_{100}	1”470	0”141	0”203	0”063	0”094	0”062	0”031	0”130
C_{200}	19”468	0”500	2”627	0”563	1”320	0”688	0”297	0”510
C_{400}	6’18”631	2”330	36”227	6”399	11”970	7”148	2”700	2”344
C_{800}	1h45’39”121	8”423	7’12”790	45”107	1’43”499	1’04”659	19”735	8”820
C_{1600}	28h02’22”424	35”532	1h22’25”139	5’45”349	15’16”156	9’27”998	2’37”126	34”294

Figure 16: Table of experimental results

13 Experimental Results

All algorithms presented in this paper have been implemented on a 3.2 GHz Pentium 4 with 2 GB of RAM. Figure 16 lists the time (in seconds) it took for each of the algorithms to run.

The random input instances $\mathrm{R}_{n,pm\%M,pf\%F}$ have been obtained randomly, given the number of vertices n and the percentages of mandatory (pm) and forbidden (pf) edges. The Monte Carlo algorithm has been demanded an error ratio $\epsilon \leq 0.05$.

The obtained results very clearly show that:

- all algorithms' behaviors match the theoretical analysis' predictions;
- for small instances, the algorithms which carry the overhead of having to build the bias graph (2-P and QF) are understandably slower;
- even though the worst-case time complexity of the 2-P algorithm is not better than that of the GC for dense input graphs, large inputs for which the latter runs faster than the former are very rare and difficult to generate, for their bias graph must own plenty of those most peculiar structures which are end strongly connected components not associated with any sandwich homogeneous sets.
- the same difficulty, as in the previous item, arises when one tries to generate instances which lead to a better performance by the QF algorithm when compared to the 2-P, although the former is certainly never slower than the latter.
- the main drawback of the algorithms based on the bias graph is their huge demand of space.

14 Conclusion

This paper provided the HSSP with a handful of new ideas, to wit: an $O(n^{3.5})$ algorithm, based on the *balancing* technique; two $O(n^3 \log n)$ easy-to-implement algorithms; the best HSSP deterministic algorithm to date, which set the problem's current upper bound at $O(n^3 \log \frac{m}{n})$; and two didactic-like $O(n^3)$ randomized algorithms: a one-side error Monte Carlo and a Las Vegas one. This latter algorithm probably turns out to be our recommended practical choice on the HSSP, not only due to its all-fast $O(n^3)$ expected running time but also for its being altogether simple.

An open question, concerning the complexity analysis of both the 2-P and the QF algorithms, is whether or not $O(\min\{m_1, \overline{m}_2\})$ is a tight enough bound for the number of non-pair-closed end strongly connected components in a bias graph. So far we do not know of any HSSP instance whose bias graph presents a number of non-pair-closed ESCCs that is not $O(n \log n)$, clearly a better (albeit unproven) bound. For the time being, either a proof that $O(n \log n)$ is indeed an upper bound or the exhibition of a bias graph with a greater number of non-pair-closed ESCCs shall be equally welcome.

15 Acknowledgements

This text was intentionally organized in a (hopefully) self-contained fashion, so that most readers, who might be unacquainted with HSSP's past research, would be provided with a complete panel of its history and state-of-the-art techniques.

The counterexamples to Tang *et al.*'s BGC algorithm and the 2-P algorithm first appeared in [13], although Counterexample 3 has been slightly improved to a more economic version since, as in [5]. Also, the complexity analysis of the 2-P has benefited from the sped-up implementation of the Bias Envelopment procedure and appears, in this paper, in its refined version. An extended abstract, containing both the Balanced Subsets and the Monte Carlo algorithms, appeared in [4]. All other algorithms in this text were yet unpublished.

We would like to thank Professor Carlos Alberto Martinhon[3] for his valuable, friendly support.

Appendix 1: Quick Bias Envelopment

The pseudo-code in Figure 17 shows a very simple way to implement the Bias Envelopment procedure, so much dealt with along the text. Such a sped-up Bias Envelopment implementation turned out to be an important piece in the analysis of some of the algorithms presented herein.

The idea is to provide each vertex $v \in V$ with two boolean flags $M[v]$ and $F[v]$ which will signal, respectively, whether there exist mandatory or forbidden edges between v and any one vertex in the current candidate set H. This speeds up the envelopment process, as former $O(n)$ set operations [1] can be replaced by constant-time read-and-write flag handling on the number of forbidden and mandatory neighbors of a given vertex.

There is an initialization step (lines 3 to 4.1.2) which flags all vertices that are not contained in the initial candidate set H_1 as to being mandatory or forbidden neighbors of some vertex $x \in H_1$. Vertices $b \in V \setminus H_1$ which have both flags set to *true* are put into the initial bias set B.

Then, there takes place a loop which only breaks when B is empty. Although all vertices $b \in B$ prevent H from being contained in a SHS which

[3]Univ. Federal Fluminense, Brazil.

Quick_Bias_Envelopment $(G_1(V, E_1), G_2(V, E_2), H_1)$

```
1.        $H \leftarrow H_1$
1.        $B \leftarrow \varnothing$
2.        for each vertex $v \in V$ do
2.1.          $M[v] \leftarrow$ false
2.2.          $F[v] \leftarrow$ false
          //Initialization.
3.        for each vertex $v \in H$ do
3.1.          for each vertex $x \in N_1(v)$ do
3.1.1.            $M[x] \leftarrow$ true
4.        for each vertex $v \in H$ do
4.1.          for each vertex $x \in \overline{N}_2(v)$ do
4.1.1.            $F[x] =$ true
4.1.2.            if $M[x] =$ true and $x \notin H$
                      $B \leftarrow B \cup \{x\}$
          // Main envelopment loop.
5.        while $|B| > 0$ do
5.1.          $b \leftarrow$ an element of $B$
5.2.          $B \leftarrow B \setminus \{b\}$
5.3.          $H \leftarrow H \cup \{b\}$
5.4.          for each vertex $x \in N_1(b)$ do
5.4.1.            $M[x] \leftarrow$ true
5.4.2.            if $F[x] =$ true and $x \notin H$
                      $B \leftarrow B \cup \{x\}$
5.5.          for each vertex $x \in \overline{N}_2(b)$ do
5.5.1.            $F[x] \leftarrow$ true
5.5.2.            if $M[x] =$ true and $x \notin H$
                      $B \leftarrow B \cup \{x\}$
6.        if $H \neq V$
              return yes.
7.        else return no.
```

Figure 17: Implementation of a fast Bias Envelopment

does not contain b, each loop iteration will have exactly *one* vertex b moved from B to H. The only vertices which need to be looked at, after such a single move, are b's mandatory and forbidden neighbors in $V \setminus (H \cup B)$ (in order to be either appropriately flagged or else added to B).

The correctness of the method is very easily verified, for Theorem 1 makes it sufficient to show that (i) every vertex in $H_{k+1} \setminus H_k$ is indeed a bias vertex of H_k and (ii) every bias vertex of H_k ought to become an element of H_{k+t}, for some t before the algorithm stops. It is easy to see that (i) holds, since a vertex enters the candidate set only if both its flags (M and F) are *true*; as for (ii), the non-emptiness of B simply prevents the algorithm from stopping.

The running time of the Quick Bias Envelopment procedure is clearly $O(m_1 + \overline{m}_2)$, as each edge — either mandatory or forbidden — triggers a constant-time operation for a constant number of times.

Appendix 2: Quick Bias Graph

According to [15], the assembling of the bias graph $G_B(V_B, E_B)$ of a given HSSP instance (G_1, G_2) would take $O(n^2\triangle_2)$ time. Actually, it can be done in $O(n \cdot \min\{m_1, \overline{m}_2\})$ time, provided an adequate data structure is used, as follows: (i) adjacency matrices for G_1 and G_2 (or one single three-state matrix, as well), in order to allow constant-time adjacency-type verification; and (ii) two adjacency lists per vertex $v \in V$, in order to allow $O(|N_1(v)|)$ and $O(|\overline{N}_2(v)|)$ access to all mandatory and, respectively, forbidden neighbors of v — instead of the $O(n)$ time it takes for traversing an entire matrix row.

Figure 18 shows a simple method to achieve a fast bias graph creation.

The insertion, in the bias graph, of all due outgoing edges from each vertex $[x, y] \in V_B$ requires the bias set of $\{x, y\}$, with respect to (G_1, G_2), to be determined. Now, the bias vertices of $\{x, y\}$ are those which are at the same time mandatory neighbors of x and forbidden neighbors of y, or vice-versa.

In order to efficiently determine the intersection of, say, the mandatory neighbors list of x and the forbidden neighbors list of y, the shortest between these two lists is traversed, having each of its members checked as to being an element of the longest one (this check is achieved in constant time by means of a lookup in the adjacency matrix).

Let us suppose that, instead of choosing the shortest list, the algorithm always opted for the traversal in the *mandatory* neighbors list. This would certainly not make a faster algorithm out of it. Now, it is clear that such a less efficient algorithm would take $\Theta(n \cdot m_1)$ time to obtain the whole edge set E_B, once, for all $x \in V$, each mandatory neighbor of x would be accessed $\Theta(n)$ times (one for each $[x, w] \in V_B$), giving rise to a matrix lookup and (possibly) an edge addition, both $O(1)$.

Analogously, a $\Theta(n \cdot \overline{m}_2)$ bound would arise if the method always traversed the forbidden neighbors list instead.

Thus, the procedure shown in Figure 18, which is no slower than these modified attempts, is certainly $O(\min\{n \cdot m_1, n \cdot \overline{m}_2\}) = O(n \cdot \min\{m_1, \overline{m}_2\})$.

```
Quick_Bias_Graph $(G_1(V, E_1), G_2(V, E_2))$

1.        $V_B \leftarrow \{[x, y] \mid x, y \in V, x \neq y\}$
2.        $E_B \leftarrow \varnothing$
3.        for each vertex $[x, y] \in V_B$ do
              // mandatory to $x$, forbidden to $y$.
3.1.          if $|N_1(x)| \leq |\overline{N}_2(y)|$
3.1.1.            for each vertex $w \in N_1(x)$ do
3.1.1.1               if $(y, w)$ is forbidden
                          $E_B \leftarrow E_B \cup \{([x, y], [x, w]), ([x, y], [y, w])\}$
3.2           else
3.2.1.            for each vertex $w \in \overline{N}_2(y)$ do
3.2.1.1               if $(x, w)$ is mandatory
                          $E_B \leftarrow E_B \cup \{([x, y], [x, w]), ([x, y], [y, w])\}$
              // mandatory to $y$, forbidden to $x$.
3.3.          if $|N_1(y)| \leq |\overline{N}_2(x)|$
3.3.1.            for each vertex $w \in N_1(y)$ do
3.3.1.1               if $(x, w)$ is forbidden
                          $E_B \leftarrow E_B \cup \{([x, y], [x, w]), ([x, y], [y, w])\}$
3.4           else
3.4.1.            for each vertex $w \in \overline{N}_2(x)$ do
3.4.1.1               if $(y, w)$ is mandatory
                          $E_B \leftarrow E_B \cup \{([x, y], [x, w]), ([x, y], [y, w])\}$
4         return $G_B(V_B, E_B)$.
```

Figure 18: A fast bias graph creation method

References

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

[2] A. Cournier, *Sur quelques algorithmes de décomposition de graphes*, Ph.D. thesis, Université Montpellier II, 1993.

[3] S. Dantas, L. Faria, and C. M. H. Figueiredo, *On decision and optimization (k, l)-graph sandwich problems*, Discrete 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 and V. G. P. Sá, *A note on the homogeneous set sandwich problem*, [to appear in] Information Processing Letters.

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

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

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

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

[10] R. M. McConnell and J. Spinrad, *Modular decomposition and transitive orientations*, Discrete Math. **201** (1999), 189–241.

[11] J. H. Muller and J. Spinrad, *Incremental modular decomposition*, J. ACM **36** (1989), 1–19.

[12] B. Reed, *A semi-strong perfect graph theorem*, Ph.D. thesis, School of Computer Science, McGill University, Montreal, 1986.

[13] V. G. P. Sá, *O problema-sanduíche para conjuntos homogêneos em grafos*, Master's thesis, COPPE / Universidade Federal do Rio de Janeiro, May 2003.

[14] J. Spinrad, *P_4-trees and substitution decomposition*, Discrete Appl. Math. **39** (1992), 263–291.

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

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