Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 156 (2008) 55–75

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Node-searching problem on block graphs[☆]

Hsin-Hung Chou[a,*], Ming-Tat Ko[b], Chin-Wen Ho[c], Gen-Huey Chen[d]

[a]*Department of Information Management, Chang Jung Christian University, 396 Chang Jung Road, Section 1, Kway Jen, Tainan 711, Taiwan*
[b]*Institute of Information Science, Academia Sinica, Taiwan*
[c]*Department of Computer Science and Information Engineering, National Central University, Taiwan*
[d]*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan*

Received 16 September 2005; received in revised form 26 June 2007; accepted 12 August 2007
Available online 17 September 2007

Abstract

The node-searching problem, introduced by Kirousis and Papadimitriou, is equivalent to several important problems, such as the interval thickness problem, the path-width problem, the vertex separation problem, and so on. In this paper, we generalize the avenue concept, originally proposed for trees, to block graphs whereby we design an efficient algorithm for computing both the search numbers and optimal search strategies for block graphs. It answers the question proposed by Peng et al. of whether the node-searching problem on block graphs can be solved in polynomial time.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Avenue; Block graphs; Node-searching problem; Path-width; Vertex separation

1. Introduction

Node-searching problem, introduced by Kirousis and Papadimitriou [16], is a variant of the graph-searching problem. The allowable *search moves* in the node-searching problem are (1) placing a searcher on a vertex and (2) removing a searcher from a vertex. Initially, all edges are considered *contaminated*. A contaminated edge is *cleared* if both its endpoints are simultaneously guarded by searchers. The entire graph is cleared if all its edges are cleared. A *search strategy* is a sequence of search moves that will clear a graph with all edges contaminated. There are two subjects in the node-searching problem on a graph G. One is to compute the *search number* of G which is the minimum number of searchers needed to clear G. The other is to construct an *optimal search strategy* for G, which clears G using minimum number of searchers.

The node-searching problem is equivalent to several important problems, such as the interval thickness problem [15] with applications in combinatorics, the survivability problem [2] with applications in communication networks, the gate matrix layout problem [21], the path-width problem [26], the vertex separation problem [14] with applications in VLSI layout, and the narrowness problem [19] with applications in natural language processing.

☆ This work was supported in part by Taiwan NSC under Grant no. NSC 92-2213-E-008-030.
* Corresponding author. Tel.: +886 6 2785123x6073; fax: +886 6 2785657.
E-mail addresses: chouhh@mail.cjcu.edu.tw (H.-H. Chou), mtko@iis.sinica.edu.tw (M.-T. Ko), hocw@csie.ncu.edu.tw (C.-W. Ho), ghchen@csie.ntu.edu.tw (G.-H. Chen).

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2007.08.007

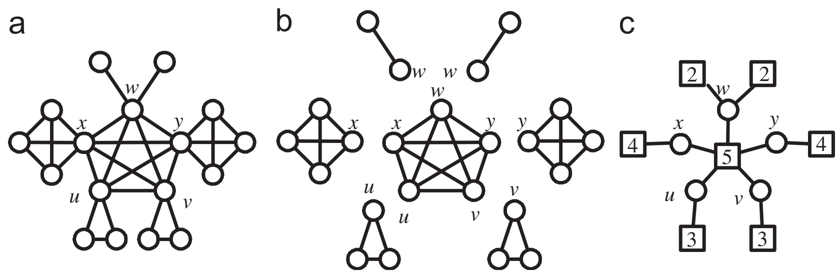


Fig. 1. An example of block graphs, blocks and block-cut-vertex graphs. (a) A block graph G. (b) Blocks of G. (c) The block-cut-vertex graph of G associated with block sizes in which square nodes represent blocks and circle nodes represent cut vertices.

The node-searching problem is **NP**-complete on planar graphs without vertex degree exceeding 3 [22], chordal graphs [9], starlike graphs [9], bipartite graphs [17], co-bipartite graphs [1] and bipartite distance-hereditary graphs [18]. For some special classes of graphs, it can be solved in polynomial time, such as trees [24,27], cographs [6], permutation graphs [5], k-starlike graphs [25] and partial k-trees [4] for a fixed $k \geqslant 1$.

For trees, the following results are known. In [8], Ellis et al. presented a linear-time algorithm to compute the vertex separation of a tree using the *labelling technique*. However, the algorithm needs $O(n \log n)$ time for computing optimal linear layouts of trees. In [24], Peng et al. presented a linear-time algorithm to construct optimal search strategies for trees based on the *avenue concept*. Independently, Skodinis [27] proposed another linear-time algorithm for trees in optimal linear layout formulation. Basically, these two algorithms are the same as designed by the technique of dynamic programming but different from their presentations. In this paper, the labelling technique and the avenue concept are adopted to design our algorithm and will be introduced later.

A vertex v in a graph G is called a *cut vertex* if the deletion of v and all edges incident to it increases the number of connected components in G. A vertex that is not a cut vertex is called a *non-cut vertex*. A maximal connected subgraph of G without cut vertex (i.e., a maximal 2-connected component of G) is called a *block* [28]. In [11], Harary and Prins defined the *block-cut-vertex graph* $T(G)$ of a graph G as the graph in which the blocks and cut vertices of G are the vertices and for all (v_K, v_u), v_K represents a block K, v_u represents a cut vertex u and $u \in K$, are the edges. It is known that $T(G)$ is a tree if G is connected.

A *block graph* is a graph and all its blocks are complete subgraphs [10]. A block graph can be represented by its block-cut-vertex graph with a block size on each vertex representing a block. An example of block graph, its blocks and block-cut-vertex graph are shown in Fig. 1. Various characterizations and optimization problems on block graphs have been studied in [13,12,7,29]. However, whether the node-searching problem on block graphs can be solved in polynomial time is still unknown so far [23]. In this paper, we answer the open question by providing an efficient polynomial-time algorithm for computing both the search numbers and an optimal search strategies of block graphs.

In Section 2, we introduce the definitions, notations and properties that will be used in this paper. In Section 3, we generalize the avenue concept on trees to block graphs. In Section 4, we define the data structures that will be used later. In Section 5, an efficient algorithm is introduced. In Section 6, time complexity of the algorithm is analyzed. The conclusion is given in the last section.

2. Preliminaries

In this paper, we consider undirected connected graphs. For a graph G, we use $V(G)$ and $E(G)$ to denote the *vertex set* and *edge set* of G, respectively. Given two sets X and Y, we use $X \cup Y$, $X \cap Y$ and $X \backslash Y$ to denote their union, intersection and difference, respectively. For two graphs G_1 and G_2, we use $G_1 \cup G_2$ to denote the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$. An *induced subgraph* H in G is a subgraph such that for all $u, v \in V(H)$, $(u, v) \in E(H)$ if and only if $(u, v) \in E(G)$. The subgraph of G induced by $S \subseteq V(G)$, denoted by $G[S]$, is the induced subgraph of G with S as the vertex set. For a subset $X \subseteq V(G)$, $G \backslash X$ is the subgraph of G induced by $V(G) \backslash X$. For a vertex $u \in V(G)$, a connected component of $G \backslash \{u\}$ is called a *branch* of G at u. The branch of G at u containing vertex v is denoted by $(G)_{u,v}$. We define the *fork* of G at u containing v, denoted by $[G]_{u,v}$, as the subgraph of G induced by $V((G)_{u,v}) \cup \{u\}$. Fig. 2 shows an example of branches and forks.

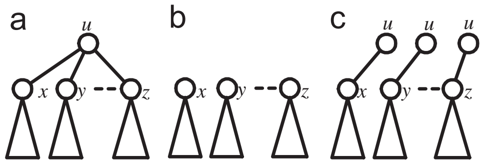


Fig. 2. Branches and forks of a graph G. (a) G. (b) Branches of G at u. (c) Forks of G at u.

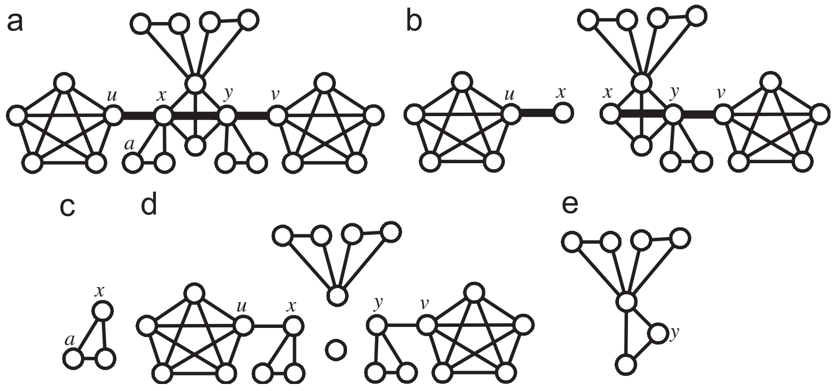


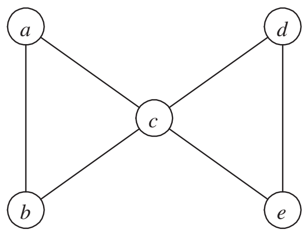
Fig. 3. An example of path forks, non-path forks, non-block forks and enclosure subgraphs, in which $P = \langle u, x, y, v\rangle$ and K is the block containing x and y. (a) A block graph G. (b) Path forks of P at x. (c) Non-path forks of P at x. (d) Non-block forks of K. (e) Enclosure subgraph of G at (x, y).

A *path* in a graph is a sequence of vertices, denoted by $\langle v_1, v_2, \ldots, v_r\rangle$, so that two vertices are adjacent if and only if they are consecutive in the sequence. A path in a graph is called an *induced path* if the subgraph induced by its vertex set is also a path. Given a path P of a block graph G and a cut vertex $u \in V(P)$, a branch (fork, resp.) at u is called a *path branch* (*path fork*, resp.) of P at u if it contains at least one vertex in $V(P)\backslash\{u\}$; otherwise it is called a *non-path branch* (*non-path fork*, resp.) of P at u. Given a block K of a block graph G and a vertex u in K, the subgraph $G\backslash V([G]_{u,v})$ $(G\backslash V((G)_{u,v})$, resp.) for any vertex v in K other than u, is called the *non-block branch* (*non-block fork*, resp.) of K at u and denoted by $(G)_{K,u}$ ($[G]_{K,u}$, resp.). Notice that $(G)_{K,u}$ may be a disconnected subgraph. For any edge $(x, y) \in E(K)$, the *enclosure subgraph* of G at (x, y) from x to y, denoted by $\{G\}_{x,y}$, is defined as $G\backslash V([G]_{K,x} \cup (G)_{K,y})$. Notice that $\{G\}_{x,y}$ and $\{G\}_{y,x}$ are isomorphic. Refer to Fig. 3 where path forks, non-path forks, non-block forks and enclosure subgraphs are shown.

2.1. Progressive search strategies

In a search strategy, a *guarded vertex* is a vertex with a searcher and a cleared edge is *recontaminated* if there exists a path connecting this edge to a contaminated one and none of the vertices along the path is guarded. A search strategy is *progressive* (also called *monotone* in [3]) if no edge recontamination occurs during the search strategy. In [16], Kirousis and Papadimitriou showed that there exists a progressive optimal search strategy for any graph. Therefore, we only consider progressive search strategies in the following.

In this paper, we adopt two representations of search strategies. One is the *operation representation*, in which a search strategy is represented by a sequence of search moves "place a searcher on a vertex" and "remove a searcher from a vertex". The other is the *set representation*, in which a search strategy S for a graph G is represented by a sequence of vertex subsets $(X_1, X_2, \ldots, X_r)$, where each $X_i \subseteq V(G)$ is the set of guarded vertices at stage i in S for $1 \leqslant i \leqslant r$. Since S is progressive, it satisfies (1) for any vertex $v \in V(G)$, the stages at which v is guarded are consecutive and (2) for any edge $(x, y) \in E(G)$, there exists a stage i such that $x, y \in X_i$ [15]. Obviously, conditions (1) and (2) are also satisfied by the *reverse strategy* of S, denoted by $S^{\text{rev}} = (X_r, X_{r-1}, \ldots, X_1)$. Thus, S^{rev} is also a progressive search strategy for G. For any vertex $v \in V(G)$, the consecutive stages at which v is guarded are denoted by an interval $[l_v, r_v]$, which is a



b

stage	search move	guarded set
1	Place searcher 1 on a	a
2	Place searcher 2 on b	a,b
3	Place searcher 3 on c	a,b,c
4	Remove searcher 1 from a	b,c
5	Remove searcher 2 from b	c
6	Place searcher 1 on d	c,d
7	Place searcher 2 on e	c,d,e
8	Remove searcher 3 from c	d,e
9	Remove searcher 1 from d	e
10	Remove searcher 2 from e	

Fig. 4. Two representations of a search strategy for a graph G. (a) G. (b) Two representations.

called the *guarded interval* of v in S. The number of searchers used in S, denoted by $\#(S)$, is $\max\{|X_i|\,|1\leqslant i\leqslant r\}$. We use $ns(G)$ to denote the search number of G, i.e., $ns(G)=\min\{\#(S)|S$ is a search strategy for $G\}$.

In the set representation, the corresponding search moves at stage i are removing searchers from vertices in $X_{i-1}\backslash X_i$ and placing searchers on vertices in $X_i\backslash X_{i-1}$, for $i=1,2,\ldots,r+1$ and $X_0=X_{r+1}=\emptyset$. Fig. 4 gives an example for the two representations.

Let Q be a block of G. There exists an index i such that $V(Q)\subseteq X_i$, which is referred to as the *clique containment property* [9]. Let H be a subgraph of G. The *induced search strategy* of S for H, denoted by $S[H]$, is $(X_1\cap V(H), X_2\cap V(H),\ldots,X_r\cap V(H))$. Notice that the guarded vertex sets in $S[H]$ may be empty sets, but it does not obstruct $S[H]$ satisfying the progressive properties and being a search strategy for H. It follows that $ns(H)\leqslant ns(G)$.

2.2. Oriented search strategies

In a search strategy, the *start vertex* is the first vertex a searcher is placed on, and the *terminal vertex* is the last vertex a searcher is removed from. Notice that in the set representation, any vertex in the first (last, resp.) guarded vertex set can be the start (terminal, resp.) vertex. Let u and v be two vertices of a graph G. An *oriented search strategy* for G from u to v is a search strategy with u as the start vertex and v as the terminal vertex. The *oriented search number* of G from u to v, denoted by $os(G,u,v)$, is the minimum number of searchers used over all oriented search strategies for G from u to v. An oriented search strategy for G from u to v is *optimal* if it uses $os(G,u,v)$ searchers. Notice that an optimal oriented strategy may not be an optimal search strategy. The reverse strategy of an oriented search strategy S for G from u to v is the search strategy from v to u. It follows that $os(G,u,v)=os(G,v,u)$. We define $os(G,u)=\min\{os(G,u,v)|v\in V(G)\}$. Then $ns(G)=\min\{os(G,u)|u\in V(G)\}$.

Lemma 1. *Let G be a graph and u, v two vertices of G. Then $ns(G)\leqslant os(G,u)\leqslant os(G,u,v)\leqslant ns(G)+1$.*

Proof. By definition, we have that $ns(G)\leqslant os(G,u)\leqslant os(G,u,v)$. The rest needs to be proved is $os(G,u,v)\leqslant ns(G)+1$.

Suppose $S=(X_1,X_2,\ldots,X_\alpha)$ is an optimal search strategy for G. Let $[l_u,r_u]$ and $[l_v,r_v]$ be the guarded intervals of u and v in S, respectively. Without loss of generality, we assume that $l_u\leqslant l_v$; otherwise we can consider S^{rev} instead. Let us consider $S'=(X'_1,X'_2,\ldots,X'_\alpha)$ where $X'_i=X_i\cup\{u\}$ for $1\leqslant i\leqslant l_u-1$, $X'_i=X_i\cup\{v\}$ for $r_v+1\leqslant i\leqslant\alpha$, and $X'_i=X_i$ for others. It is obvious that S' is an oriented search strategy for G from u to v with $\#(S')\leqslant\#(S)+1=ns(G)+1$. $\square$

Let S be an oriented search strategy for G from u to v. By the progressiveness of S, along any path from u to v at any stage, there is at least one guarded vertex on the path separating the cleared edges and contaminated edges. By this simple observation, we derive Lemma 2.

Lemma 2. *Let G be a graph and u, v two vertices of G. If there exists a subgraph H of G disjoint to a path connecting u and v, then $os(G,u,v)\geqslant ns(H)+1$.*

In Lemma 3, we prove two bounds for $os(G, u, v)$ that will be used later.

Lemma 3. *Let G be a block graph and u, v two vertices of G.*

(1) *Let P be the shortest path connecting u and v. For any edge (x, y) on P, $os(G, u, v) \geqslant ns(\{G\}_{x,y}) + 1$.*
(2) *If u is a non-cut vertex of G, then $os(G, u, v) \leqslant ns(G \backslash \{u\}) + 1$.*

Proof. (1) Let K be the block of G containing both x and y. Suppose $S = (X_1, X_2, \ldots, X_\alpha)$ is an optimal oriented search strategy for G from u to v. Let $S' = S[\{G\}_{x,y}] = (X'_1, X'_2, \ldots, X'_\alpha)$. By the clique containment property and the definition of induced search strategies, there exists a stage t at which $X_t \supseteq V(K)$ and $X'_t \supseteq V(K) \backslash \{x\}$. Suppose $[l_y, r_y]$ is the guarded interval of y in S'. Since $y \in V(K)$, it follows that $l_y \leqslant t \leqslant r_y$.

Let us consider $S'' = (X''_1, X''_2, \ldots, X''_\alpha)$ where $X''_i = X'_i \backslash \{y\}$ for all $i \neq t$, and $X''_t = X'_t$. Since y is a non-cut vertex of $\{G\}_{x,y}$ and all edges incident to y are cleared at stage t, S'' is a search strategy for $\{G\}_{x,y}$. Let us compare X_i with X''_i for all $1 \leqslant i \leqslant \alpha$. When $i \neq t$, X_i contains at least one vertex of $V(P)$. Since $X''_i \subseteq X_i$ and X''_i contains no vertex of $V(P)$ except $i = t$, $|X_i| \geqslant |X''_i| + 1$ for $i \neq t$. When $i = t$, $x \in X_t$ but $x \notin X''_t$. Thus, $|X_t| \geqslant |X''_t| + 1$. It follows that $os(G, u, v) = \#(S) \geqslant \#(S'') + 1 \geqslant ns(\{G\}_{x,y}) + 1$.

(2) Let Q be the block containing u in G. Suppose $T = (Y_1, Y_2, \ldots, Y_\beta)$ is an optimal search strategy for $G \backslash \{u\}$. Let $[l_v, r_v]$ be the guarded interval of v in T. By the clique containment property, there exists a stage t at which $Y_t \supseteq V(Q) \backslash \{u\}$. Without loss of generality, we assume that $l_v \leqslant t$; otherwise we consider T^{rev} instead.

Let us consider $T' = (Y'_1, Y'_2, \ldots, Y'_\beta)$ where $Y'_i = Y_i \cup \{v\}$ for $1 \leqslant i \leqslant l_v - 1$, $Y'_i = Y_i \cup \{u\}$ for $t \leqslant i \leqslant \beta$, and $Y'_i = Y_i$ for others. It is obvious that T' is an oriented search strategy for G from v to u with $\#(T') \leqslant \#(T) + 1 = ns(G \backslash \{u\}) + 1$. It follows that $os(G, u, v) \leqslant ns(G \backslash \{u\}) + 1$. □

3. Basic concepts

We generalize the avenue concept to block graphs whereby two data structures *label* and *strategy tree* are defined to keep track of the structure information of a rooted block graph.

For a block graph G and an integer $k \geqslant 2$, the *vertex condition with respect to k (VC_k)* is that for all $x \in V(G)$, there are at most two branches at x with search numbers at least k and the *block condition with respect to k (BC_k)* is that for all block K in G, there exists an edge $(x, y) \in E(K)$ such that $ns(\{G\}_{x,y}) < k$.

Theorem 4. *For a block graph G and an integer $k \geqslant 2$, G satisfies conditions VC_k and BC_k if and only if $ns(G) \leqslant k$.*

Before proving Theorem 4, we need the following lemmas. In the following Lemmas 5–7, G is a block graph satisfying conditions VC_k and BC_k for some integer $k \geqslant 2$, and A_k is the set of vertices x in $V(G)$ having exactly two branches of search number no less than k.

Lemma 5. *Let u be a vertex in A_k and v be an arbitrary vertex in G other than u. Then $ns((G)_{v,u}) \geqslant k$.*

Proof. Let B be the branch at u with $ns(B) \geqslant k$ which does not contain v. Obviously, $(G)_{v,u} \supsetneq B$. Thus, $(G)_{v,u}$ is a branch at v with $ns((G)_{v,u}) \geqslant k$. □

Lemma 6. *Assume that $|A_k| \geqslant 1$. Then*

(1) *A_k induces a path in G.*
(2) *For all blocks K in G, $|A_k \cap K| \leqslant 2$.*
(3) *Let K be a block such that $A_k \cap K = \{x\}$. Then there exists $u \in V(K)$ such that $ns([G]_{K,x}) < k$ and $ns(\{G\}_{x,u}) < k$.*
(4) *Let K be a block such that $A_k \cap K = \{x, y\}$. Then $ns(\{G\}_{x,y}) < k$.*

Proof. (1) The case when $|A_k| = 1$ is trivial. Let G_k denote the subgraph induced by A_k. It suffices to prove that G_k is connected and there is no vertex of degree greater than or equal to 3 in G_k. Suppose that $x, y \in A_k$ are two nonadjacent vertices of G. Let z be a vertex on the shortest path connecting x and y in G. Obviously, two branches $(G)_{z,x}$ and $(G)_{z,y}$

at z are disjoint. Since $x, y \in A_k$, by Lemma 5, we have that $ns((G)_{z,x}) \geqslant k$ and $ns((G)_{z,y}) \geqslant k$. It follows that $z \in A_k$. Thus, G_k is connected.

If there is a vertex v of degree no less than 3 in G_k, then v has at least three branches of G with search numbers no less than k. It contradicts to the condition VC_k. Thus, A_k induces a path in G.

(2) If there exists a block which contains three vertices in A_k, it implies that G_k contains a triangle, which contradicts to (1). Thus, $|A_k \cap V(K)| \leqslant 2$.

Let (x', y') be an edge of K such that there exists a vertex $v \in A_k \cap K$ but $v \notin \{x', y'\}$. For $v \in A_k$, $ns((G)_{K,v}) \geqslant k$. Since $G_{x',y'} \supset (G)_{K,v}$, we have that $ns(G_{x',y'}) \geqslant k$. Suppose that (x', y') be an edge in K such that $ns(G_{x',y'}) < k$. Then $\{x', y'\} \supset A_k \cap V(K)$.

(3) Without loss of generality, assume that $x' = x$. For $(G)_{y',x} \supset (G)_{K,x}$, $ns((G)_{y',x}) \geqslant k$. Since $y' \notin A_k$, y' has at most one branch with search number no less than k. Thus, $ns((G)_{K,y'}) < k$.

(4) In this case, $\{x', y'\} = A_k \cap V(K)$. Thus, $ns(G_{x,y}) < k$. □

Lemma 7. *Assume that $|A_k| = 0$, and for all $x \in V(G)$, there is exactly one branch at x of search number no less than k. Then there is an unique block K such that $ns((G)_{K,u}) < k$ and $ns(G \backslash V([G]_{K,u})) \geqslant k$ for all $u \in V(K)$.*

Proof. To prove the assertion, we construct a directed block-cut-vertex graph $\hat{T}$ of G from the block-cut-vertex graph T of G by assigning directions to edges of T as follows. For each edge (v_K, v_u) in T corresponding to block K and cut vertex u in G, (v_K, v_u) is directed from v_K to v_u in $\hat{T}$ if $ns((G)_{K,u}) \geqslant k$, and from v_u to v_K if $ns(G \backslash V([G]_{K,u})) \geqslant k$.

If v_u is directed to v_K in $\hat{T}$, all block vertices v_Q adjacent to v_u other than v_K are directed to v_u since $[G]_{Q,u} \supseteq G \backslash V([G]_{K,u})$ and $ns(G \backslash V([G]_{K,u})) \geqslant k$. Since each cut vertex has exactly one outgoing edge and each edge has one endpoint a cut vertex in $\hat{T}$, each edge has at least one way directed. If v_K is directed to v_u in $\hat{T}$, v_u must have one outgoing edge directed to a block vertex other than v_K. It follows that $\hat{T}$ has no cycle because if $\hat{T}$ has a bidirected edge (v_K, v_u), then u is a cut vertex with two branches having search numbers k, which is a contradiction. Thus, there exists at least one sink in $\hat{T}$ and the sink must be a block vertex.

Assume that $\hat{T}$ has two different block vertices v_X and v_Y which are sinks. Let x be the cut vertex of X such that $[G]_{X,x} \supseteq Y$. Then v_x has two outgoing edges, which is a contradiction. Therefore, G has a unique block K which satisfies the property that $ns(G \backslash V([G]_{K,u})) \geqslant k$ for all $u \in V(K)$. Since each vertex has exactly one branch with search number at least k, it follows that $ns((G)_{K,u}) < k$ for all $u \in V(K)$. □

Definition 8. Let $P = \langle v_1, \ldots, v_r \rangle$ be an induced path in G. Let G_i be the subgraph consisting of the non-path forks of P at v_i for all $1 \leqslant i \leqslant r-1$ and $G_{i,i+1}$ be the subgraph induced by $V(\{G\}_{v_i,v_{i+1}}) \cup \{v_i\}$. Let S_i be any optimal oriented search strategy for G_i from v_i to v_i for all $1 \leqslant i \leqslant r$, and $S_{i,j}$ be any optimal oriented search strategy for $G_{i,i+1}$ from v_i to v_{i+1} for all $1 \leqslant i < r$. The search strategy S composed of $S_1, S_{1,2}, S_2, S_{2,3}, \ldots, S_{r-1}, S_{r-1,r}, S_r$ is called a basic search strategy for G along P.

The number of searchers used in S is $\max\{ns(G_i) + 1, os(G_{i,i+1}, v_i, v_{i+1}) | 1 \leqslant i \leqslant r-1\}$.

Proof of Theorem 4. *Sufficiency*: It suffices to prove that if G does not satisfy VC_k or does not satisfy VB_k, then $ns(G) \geqslant k+1$. Assume that u and v are the start vertex and the terminal vertex of an optimal search strategy for G and let P be the shortest path connecting u and v.

First, let us consider the case when G does not satisfy VC_k. Then there exists a vertex x at which there are at least three branches with search numbers no less than k. Then u and v are contained in at most two of them. Thus, we can find a branch at x with search number no less than k which is disjoint to P. By Lemma 2, we have that $ns(G) \geqslant k+1$.

Next, let us consider the case when G does not satisfy BC_k. Then there exists a block K such that $ns(\{G\}_{x,y}) \geqslant k$ for all $(x, y) \in E(K)$. If P contains an edge (x', y') in K, by Lemma 3(1), we have that $ns(G) = os(G, u, v) \geqslant ns(\{G\}_{x',y'}) + 1 \geqslant k+1$. Otherwise, u and v must be both located in $[G]_{K,x''}$ for some $x'' \in V(K)$. Let y'' be a vertex other than x'' in K. Since G does not satisfy BC_k, we have that $ns(\{G\}_{x'',y''}) \geqslant k$. Since P contains no edge in K, $\{G\}_{x'',y''}$ is disjoint to P. By Lemma 2, we have that $ns(G) \geqslant k+1$.

Necessity: Let A_k be the set of vertices in which each vertex has exactly two branches with search numbers at least k.

First, let us consider the case when $|A_k| \geqslant 1$. By Lemma 6(1), we have that A_k induces a path in G. Suppose $\hat{P} = \langle v_2, v_3, \ldots, v_{r-1} \rangle$ for $r \geqslant 3$ is the path induced by A_k. By Lemma 6(4), there exists a vertex v_1 (v_r, resp.) in the non-path branch at v_2 (v_{r-1}, resp.) such that $ns(G \backslash V([G]_{v_1,v_2})) < k$ and $ns(\{G\}_{v_1,v_2}) < k$ ($ns(G \backslash V([G]_{v_r,v_{r-1}})) < k$ and $ns(\{G\}_{v_{r-1},v_r}) < k$, resp.). Let $P = \langle v_1, v_2, \ldots, v_{r-1}, v_r \rangle$. Consider the basic search strategy for G along P. Let G_i be the subgraph consisting of the non-path forks of P at v_i for all $1 \leqslant i \leqslant r-1$ and $G_{i,j}$ be the subgraph induced by $V(\{G\}_{v_i,v_{i+1}}) \cup \{v_i\}$. By Lemma 3(2), we have that $os(G_{i,i+1}, v_i, v_{i+1}) \leqslant ns(\{G\}_{v_i,vi+1}) + 1 \leqslant k$, and $os(G_i, v_i, v_i) \leqslant ns(G_i) + 1 \leqslant k$. It follows that the number of searchers used in the basic search strategy along P is no greater than k. Thus, $ns(G) \leqslant k$.

Next, let us consider the case when $|V_k| = 0$. If there exists a vertex u at which all branches have search numbers less than k, then the number of searchers used in the basic search strategy along $\langle u \rangle$ is no greater than k. Otherwise, every vertex has exactly one branch with search number at least k. By Lemma 7, we have that there is an unique block K such that $ns((G)_{K,u}) < k$ and $ns(G \backslash V([G]_{K,u})) \geqslant k$ for all $u \in V(K)$. Since G satisfy BC_k, there is an edge $(u, v) \in E(K)$ such that $G_{u,v} < k$, $ns((G)_{K,u}) < k$ and $ns((G)_{K,v}) < k$. It follows that the number of searchers used in the basic search strategy along $\langle u, v \rangle$ is no greater than k. Thus, $ns(G) \leqslant k$. $\square$

Based on the above characterization, we define an avenue of a block graph.

Definition 9. A path $P = \langle v_1, v_2, \ldots, v_r \rangle$ is an avenue of a block graph G, if the following conditions hold:

(1) If $r = 1$, then all branches of G at v_1 have search numbers smaller than $ns(G)$.
(2) If $r > 1$, then
  (a) each of v_1 and v_r has only one branch, $(G)_{v_1,v_2}$ and $(G)_{v_r,v_{r-1}}$ respectively, with search number $ns(G)$;
  (b) for $2 \leqslant i \leqslant r-1$, v_i has exactly two branches, $(G)_{v_i,v_{i-1}}$ and $(G)_{v_i,v_{i+1}}$, with search numbers $ns(G)$;
  (c) for $1 \leqslant j \leqslant r-1$, $ns(\{G\}_{v_j,v_{j+1}}) < ns(G)$.

Let G_i be the subgraph consisting of the non-path forks of P at v_i for all $1 \leqslant i \leqslant r-1$ and $G_{i,j}$ be the subgraph induced by $V(\{G\}_{v_i,v_{i+1}}) \cup \{v_i\}$. Since $os(G_{i,i+1}, v_i, v_{i+1}) \leqslant ns(\{G\}_{v_i,v_{i+1}} + 1 \leqslant ns(G)$, and $os(G_i, v_i, v_i) \leqslant ns(G_i) + 1 \leqslant ns(G)$, the number of searchers used in the basic search strategy along P is no greater than $ns(G)$, i.e., the basic search strategy along an avenue is an optimal search strategy.

When $r = 1$, the only vertex on P is called a *hub*, and in this case a block graph may have more than one hub. When $r > 1$, the vertices with exactly two branches having search numbers $ns(G)$ are called *critical vertices* and the vertices with only one branch having search number $ns(G)$ are called *outlet vertices*. They are all called *avenue vertices*. As shown in the proof of Theorem 4, all avenues share the same critical vertices and they can be different only on the outlet vertices. An edge in any avenue is called a *critical edge*. A block containing critical edges is called a *critical block*.

Lemma 10. *K is a critical block of a block graph G if and only if* $ns(G \backslash V([G]_{K,x})) = ns(G)$ *for all* $x \in V(K)$.

Proof. *Necessity*: If G contains no critical vertex, the assertion is true by Lemma 7. Consider the case when K contains at least one critical vertex. Let $u \in V(K)$ be the critical vertex and $(u, v) \in E(K)$ be the critical edge in G. Since the basic search strategy along an avenue is an optimal search strategy, it implies that $ns((G)_{u,v}) = ns((G)_{v,u}) = ns(G)$. Since $(G)_{u,v} = G \backslash V([G]_{K,u})$, it follows that $ns(G \backslash V([G]_{K,u})) = ns(G)$. By Lemma 5, we have that $ns((G)_{x,u}) \geqslant ns(G)$ for any $x \in V(K)$ other than u. Since $(G)_{x,u} = G \backslash V([G]_{K,x})$, it follows that $ns(G \backslash V([G]_{K,x})) = ns(G)$.

Sufficiency: If $ns(G \backslash V([G]_{K,x})) = ns(G)$ for all $x \in V(K)$, every vertex of G has at least one branch with search number $ns(G)$. Thus, G has no hub. Let P be an avenue of G with two different endpoints u and v. Notice that $os(G, u, v) = ns(G)$. Assume that P contains no edges in K. Then there exists a vertex $w \in V(K)$ such that $[G]_{K,w} \supseteq P$. Since $G \backslash V([G]_{K,w})$ has $ns(G \backslash V([G]_{K,w})) = ns(G)$ and is disjoint to P, by Lemma 2, we have that $os(G, u, v) \geqslant ns(G) + 1$, which is a contradiction. Thus, K contains an edge of P, i.e., a critical edge. It follows that K is a critical block of G. $\square$

4. Structure information

A *rooted block graph* $G[u]$ is a block graph G with a specified vertex u as its *root*. For any vertex v in $G[u]$, the *parent* of v is its neighbor on the shortest path connecting to u, the *children* of v are its neighbors which are not contained in

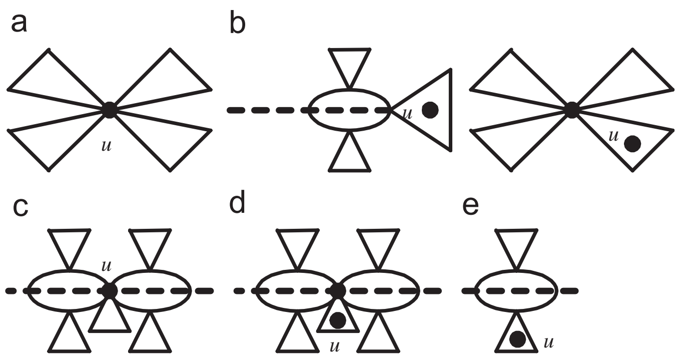


Fig. 5. Types of rooted block graphs. (a) Type H. (b) Type I. (c) Type E. (d) Type M_v. (e) Type M_b.

$(G)_{v,u}$ and the children of v which are cut-vertices in G are called the *cut-children* of v. For any vertex x other than u in $G[u]$, the *subgraph rooted at* x is the one consisting of all forks at x without u. The *structure information* for $G[u]$ consists of two data structures: *label* and *strategy tree*, which provides the search number of G and the skeleton of an optimal search strategy for G. Our approach is to compute the structure information of $G[u]$ recursively from that of the subgraphs rooted at the cut-children of u.

4.1. Labelling

According to the location of u relative to the avenue of G, $G[u]$ is classified into five types, H, E, I, M_v and M_b. The type of $G[u]$, denoted by $\tau(G[u])$, is defined as follows (see Fig. 5).

Definition 11. Let $G[u]$ be a rooted block graph.

(1) If u is a hub of G, then $\tau(G[u]) = H$.
(2) If (i) G has a hub but u is not a hub of G, or (ii)G has no hub and u is located in a non-block fork of a critical block at an outlet vertex, then $\tau(G[u]) = E$.
(3) If u is a critical vertex of G, then $\tau(G[u]) = I$.
(4) If u is located in a branch at a critical vertex without any avenue vertex, then $\tau(G[u]) = M_v$.
(5) If u is located in a non-block fork of a critical block without any avenue vertex, then $\tau(G[u]) = M_b$.

Lemma 12. *Let $G[u]$ be a rooted block graph. If $\tau(G[u]) = H$ or E, then $os(G,u) = ns(G)$; otherwise $os(G,u) = ns(G) + 1$.*

Proof. First, let us consider the case when $\tau(G[u]) = H$ or E. If u is a hub or an outlet vertex, then $os(G,u) = ns(G)$. Otherwise, let h be a hub or an outlet vertex such that u is contained in a branch at h without any avenue vertex. Notice that h is a non-cut vertex in $[G]_{h,u}$ and $ns((G)_{h,u}) < ns(G)$. By Lemma 3(2), we have that $os([G]_{h,u}, u, h) \leqslant ns(G)$. With $os(G\backslash V((G)_{h,u}), h) \leqslant ns(G)$, we obtain $os(G,u) \leqslant ns(G)$. Since $ns(G) \leqslant os(G,u)$, it follows that $os(G,u) = ns(G)$.

In the following two cases, let v be a vertex such that $os(G,u,v) = os(G,u)$ and P the shortest path connecting u and v.

Secondly, let us consider the case when $\tau(G[u]) = I$ or M_v. Let χ be the critical vertex of G closest to u. Since χ has two branches with search numbers $ns(G)$, we can find a branch at χ disjoint to P with search number $ns(G)$ no matter where v is located. By Lemma 2, we have that $os(G,u) = os(G,u,v) \geqslant ns(G) + 1$. By Lemma 1, we have that $os(G,u) \leqslant ns(G) + 1$. It follows that $os(G,u) = ns(G) + 1$.

Finally, let us consider the case when $\tau(G[u]) = M_b$. Let K be the critical block of G and x a vertex of K such that u is contained in $[G]_{K,x}$. Since x is not an avenue vertex, $ns(\{G\}_{x,w}) = ns(G)$ for any vertex $w \in V(K)$ other than x. If P contains no edge of K, then there exists a vertex z of K other than x such that $ns(\{G\}_{x,z}) = ns(G)$ and $\{G\}_{x,z}$ is disjoint to P. By Lemma 2, we have that $os(G,u,v) \geqslant ns(G) + 1$. Next, if P contains an edge (x, y) of K, by Lemma 3(1), we have that $os(G,u) = os(G,u,v) \geqslant ns(\{G\}_{x,y}) + 1 \geqslant ns(G) + 1$. It follows that $os(G,u) = ns(G) + 1$. $\square$

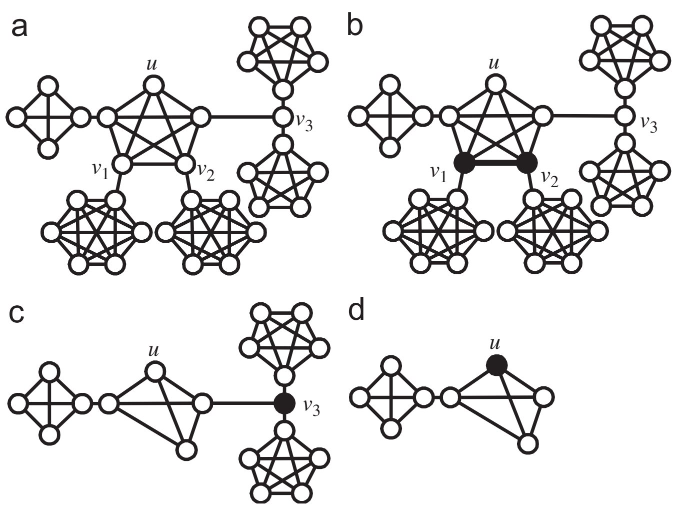


Fig. 6. Critical vertices, critical edges and outlet vertices of a rooted block graph $G[u]$ corresponding to its label $\lambda(G[u])$. (a) $G[u]$. (b) (v_1, v_2) is a critical edge of $G[u]$. (c) v_3 is a critical vertex of $G[u; (v_1, v_2)]$. (d) u is an outlet vertex of $G[u; (v_1, v_2), v_3]$.

Let $\chi_1, \chi_2, \ldots, \chi_i$ be a sequence of vertices or edges of a rooted block graph $G[u]$. We denote $G[u; \chi_1, \chi_2, \ldots, \chi_i]$ as the subgraph of G recursively defined as follows:

(1) if $i = 0$, then $G[u; \chi_1, \chi_2, \ldots, \chi_i] = G$;
(2) when $\chi_i = u$, $G[u; \chi_1, \chi_2, \ldots, \chi_i] = \emptyset$;
(3) when $\chi_i = v \in V(G)$ other than u, $G[u; \chi_1, \chi_2, \ldots, \chi_i] = (G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}])_{v,u}$;
(4) when $\chi_i = (x, y) \in E(K)$ for some block K in $G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}]$, if u is in $\{G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}]\}_{x,y}$, then $G[u; \chi_1, \chi_2, \ldots, \chi_i] = \{G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}]\}_{x,y}$, else $G[u; \chi_1, \chi_2, \ldots, \chi_i]$ is undefined.

Analogous to the label of a rooted tree defined in [8], we define the label of a rooted block graph as follows.

Definition 13. For any rooted block graph $G[u]$, a label of $G[u]$ consists of a list of structure records $(R_1, R_2, \ldots, R_p)$ for some integer $p \geqslant 1$, in which each R_i consists of three fields $[s_i, \tau_i, \chi_i]$ where s_i is a positive integer, τ_i is a type, and χ_i is either a vertex or an edge, such that

(1) $s_1 > s_2 > \cdots > s_p \geqslant 1$.
(2) For $0 \leqslant i < p$, $ns(G[u; \chi_1, \chi_2, \ldots, \chi_i]) = s_{i+1}$.
(3) For $1 \leqslant i < p$, $\tau_i = M_v$ or M_b.
  (a) When $\tau_i = M_v$, χ_i is the critical vertex of $G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}]$ such that u is located in a branch at χ_i containing no avenue vertex.
  (b) When $\tau_i = M_b$, χ_i is a critical edge of $G[u; \chi_1, \chi_2, \ldots, \chi_{i-1}]$ whose enclosure subgraph contains u.
(4) $\chi_p = u$ and $\tau_p = H, E$, or I.

Fig. 6 shows critical vertices, critical edges and outlet vertices of a rooted block graph corresponding to its label $([6, M_b, (v_1, v_2)], [5, M_v, v_3], [4, E, u])$. Fig. 7 gives an example that a rooted block graph $G[u]$ with neither critical vertices nor hubs may have more than one avenue. The labels of $G[u]$ are (a) $([7, M_b, (v_1, v_2)], [6, M_v, v_7], [5, E, u])$ and (b) $([7, M_b, (v_2, v_3)], [6, M_v, v_4], [3, H, u])$ for two different avenues in Fig. 7(a) and (b), respectively.

Let $\lambda_i = ([s^i_1, \tau^i_1, \chi^i_1], [s^i_2, \tau^i_2, \chi^i_2], \ldots, [s^i_{l_i}, \tau^i_{l_i}, \chi^i_{l_i}])$ for $i = 1, 2$ be two labels. We say labels $\lambda_1 > \lambda_2$ if the following conditions hold:

(1) $s^1_1 > s^2_1$, or

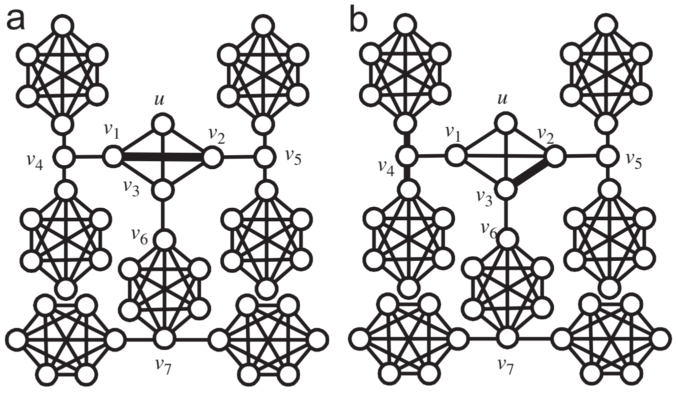


Fig. 7. Two different avenues of a rooted block graph $G[u]$. (a) (v_1, v_2). (b) (v_2, v_3).

(2) $s_1^1 = s_1^2$ and $(s_2^1, s_3^1, \ldots, s_{l_1}^1) > (s_2^2, s_3^2, \ldots, s_{l_2}^2)$, or
(3) $(s_1^1, s_2^1, \ldots, s_{l_1}^1) = (s_1^2, s_2^2, \ldots, s_{l_2}^2)$, i.e., $l_1 = l_2$ and $\tau_{l_1}^1 > \tau_{l_2}^2$ by the order of types $M_b = M_v > I > E > H$.

The labels of $G[u]$ with minimum order can be used to represent the structure information of $G[u]$ precisely for the merging routine, as shown later. In the following, we use $\lambda(G[u])$ to denote a minimum label of $G[u]$. With respect to $\lambda(G[u])$, we use $\chi(G[u])$ to denote χ_1.

4.2. Strategy path system

A path P is a *k-strategy path* of a block graph G if the basic search strategy along P uses k searchers. A k-strategy path of G with $k = ns(G)$ is called an *optimal strategy path* of G. Clearly, an avenue of G is an optimal strategy path of G. For convenience, k is omitted without ambiguity. In the following, we represent a strategy path by a sequence of alternately vertices and edges, e.g., $P = \langle v_1, e_1, v_2, e_2, v_3, \ldots, v_r \rangle$ where e_i denotes the edge (v_i, v_{i+1}) for $1 \leqslant i \leqslant r-1$.

Definition 14. For any strategy path $P = \langle v_1, e_1, v_2, \ldots, v_r \rangle$ of a block graph G which is not an isolated vertex, the decomposition of G w.r.t. P, denoted by $\mathscr{F}^P(G)$, is a sequence of proper subgraphs of G defined as follows:

(1) If $r = 1$ and v_1 is a non-cut vertex of G, then $\mathscr{F}^P(G)$ consists of the only branch of G at v_1.
(2) Otherwise, $\mathscr{F}^P(G)$ consists of all non-path forks at the vertices of P and all enclosure subgraphs at the edges of P, in which the enclosure subgraph at e_i is placed after the non-path forks at v_i and before the non-path forks at v_{i+1}.

For each element F in $\mathscr{F}^P(G)$ which is a branch, non-path fork or enclosure subgraph at x, we call F a *p-subgraph* of G at x of P for short. A *list* $\mathscr{L} = [L_1, L_2, \ldots, L_p]$ consists of a finite number of ordered elements, in which L_i is either a path or a list. In the following, we define the *strategy path system* of a block graph G to be a list of strategy paths to depict the skeleton of a search strategy for G.

Definition 15. For any block graph G and an integer $k \geqslant ns(G)$, a k-strategy path system $\mathscr{D}(G, k)$ is a list of strategy paths recursively defined as follows:

(1) If G is an isolated vertex v, then $\mathscr{D}(G, k) = [< v >, \emptyset]$.
(2) Otherwise, let P be a k-strategy path of G and $\mathscr{F}^P(G) = (\mathscr{F}_1, \mathscr{F}_2, \ldots, \mathscr{F}_t)$, then $\mathscr{D}(G, k) = [P, [\mathscr{D}(\mathscr{F}_1, k_1), \mathscr{D}(\mathscr{F}_2, k_2), \ldots, \mathscr{D}(\mathscr{F}_t, k_t)]]$ where $ns(\mathscr{F}_i) \leqslant k_i < k$ for $1 \leqslant i \leqslant t$.

In the definition of $\mathscr{D}(G, k)$, the k_i-strategy path P_i of $\mathscr{F}_i$ is called the *main strategy path* of $\mathscr{F}_i$. Relatively, $\mathscr{F}_i$ is called the *container* of P_i.

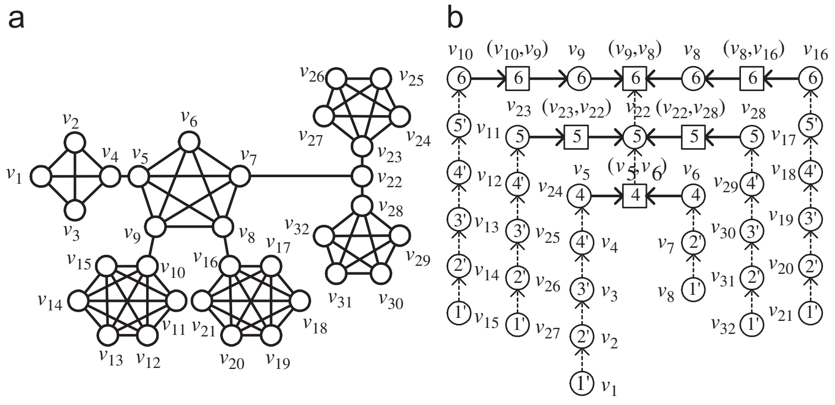


Fig. 8. A strategy tree for a rooted block graph $G[v_6]$. (a) $G[v_6]$. (b) A strategy tree.

4.3. Strategy tree

Given a k-strategy path system $\mathscr{D}(G,k)$ of a block graph G with $ns(G) \leqslant k$, we propose a data structure called *strategy tree* to store all the strategy paths in $\mathscr{D}(G,k)$. For distinction, we call the vertices (edges, resp.) of a strategy tree *nodes* (*arcs*, resp.). For any two adjacent nodes x and y in a strategy tree, we use the notation $x \rightarrow y$ or $y \leftarrow x$ to denote an arc directed from x to y. In such a case, y is called the *parent* of x and x is called the *child* of y.

Definition 16. Given a k-strategy path system $\mathscr{D}(G,k)$ for a rooted block graph $G[u]$ and an integer $k \geqslant ns(G)$, a k-strategy tree of $G[u]$, denoted by $\mathscr{T}(G[u],k)$, is a rooted tree constructed as follows.

(1) $V(\mathscr{T}(G[u],k))$ consists of two kinds of nodes: vertex-nodes and edge-nodes. There is a one-to-one mapping $\varphi(.)$ from the vertex-nodes (edge-nodes, resp.) in $\mathscr{T}(G[u],k)$ to the vertices (edges, resp.) in $\mathscr{D}(G,k)$.
(2) $E(\mathscr{T}(G[u],k))$ consists of two kinds of arcs: path-arcs and layer-arcs. For each strategy path $P = \langle v_1, e_1, v_2, \ldots, v_r\rangle$ in $\mathscr{D}(G,k)$, let $x_1, x_2, \ldots, x_{2r-1}$ be the sequence of nodes corresponding to the elements of P,
   (a) we arbitrarily choose a node x_p as the root, then $x_i \rightarrow x_{i+1}$ for $1 \leqslant i \leqslant p-1$ and $x_i \leftarrow x_{i+1}$ for $p \leqslant i \leqslant 2r-2$ are added as path-arcs;
   (b) for each p-subgraph $\mathscr{F}$ of G at x_j of P, $1 \leqslant j \leqslant 2r-1$, let y be the root of the main strategy path of $\mathscr{F}$, then $y \rightarrow x_j$ is added as a layer-arc.

Here, k is also omitted without ambiguity and a k-strategy tree of $G[u]$ with $k = ns(G)$ is called an *optimal strategy tree* of $G[u]$. For each node x in $\mathscr{T}(G[u],k)$, we associate x with an integer $id(x)$ to denote the number of searchers used for the strategy path containing vertex $\varphi(x)$ in $\mathscr{D}(G,k)$. Let $\pi(x)$ denote the sequence of nodes corresponding to the strategy path containing vertex $\varphi(x)$ in $\mathscr{D}(G,k)$. If $\pi(x)$ contains only one node x and x corresponds to a non-cut vertex, we mark $id(x)$ with a prime "$\,'\,$". Notice that $id(x)$ is not equal to $id(x)$ marked with a prime. Then we can identify whether an arc $x \leftarrow y$ is a path-arc or layer-arc according to $id(x)$ and $id(y)$. That is, if $id(x) = id(y)$, then it is a path-arc; otherwise it is a layer-arc. Fig. 8 shows a strategy tree of a rooted block graph $G[v_6]$, in which circle nodes denote vertices, square nodes denote edges, solid arrows denote path-arcs and dash arrows denote layer-arcs.

Given an optimal strategy tree T of $G[u]$, rooted at x, we design the following algorithm to construct an optimal search strategy for G. For any node y in T, we use T_y to denote the subtree rooted at y.

Algorithm. SEARCH(T, x)
/* Input: an optimal strategy tree T of $G[u]$, rooted at x. */
/* Output: an optimal search strategy. */

```
    find $\pi(x) = (x_1, x_2, \ldots, x_r)$;
    place a searcher on $\varphi(x_1)$;
```

```
        if $r = 1$ and $x_1$ corresponds to a non-cut vertex, then
        SEARCH($T_{x'}$, $x'$) where $x'$ is the only child of $x_1$;
        else
            for $i = 1$ to $r$ do
                if $i$ is an odd integer, then
                (+) for each child $x'$ of $x_i$ connected by layer-arcs do SEARCH($T_{x'}$, $x'$) in which the search
                moves "place a searcher on $\varphi(x_i)$" and "remove a searcher from $\varphi(x_i)$" are deleted;
                if $i$ is an even integer, then
                (*) for the only child $x'$ of $x_i$ connected by a layer-arc do
                    SEARCH($T_{x'}$, $x'$)in which if there exists any search move on
                    $\varphi(x_{i-1})$, then the search move "place a searcher on $\varphi(x_{i-1})$" is
                    replaced by "place a searcher on $\varphi(x_{i+1})$"; otherwise the search
                    move "remove a searcher from $\varphi(x_{i+1})$" is replaced by "remove
                    a searcher from $\varphi(x_{i-1})$";
            end for;
        remove a searcher from $\varphi(x_r)$;
    end SEARCH.
```

Notice that the statement (+) is a trick to clear all the non-path forks at $\varphi(x_i)$ during which $\varphi(x_i)$ is guarded at all stages, and the statement (*) is a trick to clear the enclosure subgraph at $(\varphi(x_{i-1}), \varphi(x_{i+1}))$ and to switch the searcher from $\varphi(x_{i-1})$ to $\varphi(x_{i+1})$. The statements (+) and (*) can be done in constant time by indexing all the search moves during the recursive call SEARCH$(T_{x'}, x')$ and returning the indices of the search moves that will be deleted or modified.

Lemma 17. *Given an optimal strategy tree T of $G[u]$, rooted at x, the algorithm SEARCH(T, x) constructs an optimal search strategy for G in* $O(|V(T)|)$ *time.*

Proof. It is trivial that the lemma follows when G is an isolated vertex. Assume $|V(G)| > 1$. Since every subgraph in $\mathscr{F}^P(G)$ is a proper subgraph of G, the search strategy constructed by the algorithm SEARCH(T, x) can be shown to be optimal by the similar argument in the proof of Theorem 4.

The algorithm traverses each node x of T at most three times: the first time occurs at deciding $\pi(y)$ where y is the parent of x, the second time occurs at determining $\pi(x)$, the third time occurs at clearing along $\pi(x)$. Therefore, the lemma follows. □

5. Merging routines

In this section, the algorithm computing the structure information and a strategy tree of a rooted block graph is presented. The computation is from the bottom of the rooted block graph to the root. In processing a node v, with the structure information of its cut-children's structure information available, we merge the information to obtain the structure information of the block graph rooted at v.

Let $G[u]$ be a rooted block graph. The *main* merging routine for computation of the structure information of $G[u]$ proceeds recursively as follows:

(1) If G is a complete graph with $V(G)=\{u_1, u_2, \ldots, u_n\}$ where $u=u_n$, we set $\lambda(G[u])=([n, H, u])$ and $\mathscr{T}(G[u], n)$ as a directed path $u_1 \to u_2 \to \cdots \to u_n$ with $id(u_i) = i'$ for $1 \leqslant i \leqslant n$.
(2) If u is a cut vertex of G, we first split G into two subgraphs G_1 and G_2 sharing only one vertex u. We recursively compute the structure information for each of $G_1[u]$ and $G_2[u]$, and then merge them by the *vertex merging* introduced later.
(3) If u is a non-cut vertex of G which is not a complete graph, we first compute the structure information for each rooted block subgraph rooted at a cut-child of u, and then sort them by the labels and merge them by the *block merging* introduced later.

In the following, we first introduce the rules for merging on labels. Then we insert some extra rules for constructing an optimal strategy tree.

5.1. Vertex merging

Let $G[u]$ be a rooted block graph which consists of two subgraphs G_1 and G_2 such that $G_1 \cup G_2 = G$ and $V(G_1) \cap V(G_2) = \{u\}$. Without loss of generality, we assume that $\lambda(G_1[u]) \geqslant \lambda(G_2[u])$. Let $k = ns(G_1)$ and & be a list concatenation operation to concatenate two labels into one. The vertex merging proceeds by the following rules which are similar to the merging rules on trees [20].

(1) If $\tau(G_1[u]) = H$, E or I and $\lambda(G_2[u]) \leqslant ([k, H, u])$, then $\lambda(G[u]) = \lambda(G_1[u])$.
(2) If $\tau(G_1[u]) = E$ and $\lambda(G_2[u]) = ([k, E, u])$, then $\lambda(G[u]) = ([k, I, u])$.
(3) If $\tau(G_1[u]) = I$ and $\lambda(G_2[u]) = ([k, E, u])$ or $([k, I, u])$, then $\lambda(G[u]) = ([k + 1, H, u])$.
(4) If $\tau(G_1[u]) > I$ and $ns(G_1[u; \chi(G_1[u])] \cup G_2) < k$, then $\lambda(G[u]) = ([k, \tau(G_1[u]), \chi(G_1[u])])$ & $\lambda((G_1[u; \chi(G_1[u])] \cup G_2)[u])$.
(5) If $\tau(G_1[u]) > I$ and $ns(G_1[u; \chi(G_1[u])] \cup G_2) \geqslant k$, then $\lambda(G[u]) = ([k + 1, H, u])$.

Lemma 18. $\lambda(G[u])$ *is computed correctly by the vertex merging rules.*

Proof. In this proof, graph pair (H_1, H_2) is an ordered pair of block subgraphs of G satisfying $V(H_1) \cap V(H_2) = \{u\}$ and $\lambda(H_1[u]) \geqslant \lambda(H_2[u])$. To prove this lemma, we prove the following two statements: (i) for any graph pair (G_1, G_2), $\lambda((G_1 \cup G_2)[u])$ is computed correctly by the vertex merging rules and (ii) for any two graph pairs (G_1, G_2) and (G'_1, G'_2) with $\lambda(G_1[u]) \geqslant \lambda(G'_1[u])$, $\lambda((G_1 \cup G'_1)[u]) \geqslant \lambda((G_2 \cup G'_2)[u])$.

Let $\lambda_i = \lambda(G_i[u]) = ([s^i_1, \tau^i_1, \chi^i_1], [s^i_2, \tau^i_2, \chi^i_2], \ldots, [s^i_{l_i}, \tau^i_{l_i}, \chi^i_{l_i}])$ for $i = 1, 2$. We will prove statements (i) and (ii) by induction on an ordered pair (s^1_1, l_1) in lexicographic order. Notice that $l_1 \leqslant s^1_1$. It is easy to verify the correctness of the two statements when $s^1_1 = 2$, which is omitted here.

Assume that the two statements are true when $s^1_1 < k$. In the following, we will prove the correctness of the two statements when $s^1_1 = k$. First of all, we show the cases when $l_1 = 1$.

Rule (1): Since $\lambda_2 \leqslant ([k, H, u])$, the search numbers of the branches of G_2 at u are all smaller than k. It follows that the avenue of G_1 is also an avenue of $G_1 \cup G_2$. Thus, $ns(G) = k$ and $\tau(G[u]) = \tau^1_1$.

Rule (2): Since $\tau^1_1 = \tau^2_1 = E$, we have that $os(G_1, u) = os(G_2, u) = k$. G thus can be cleared by k searchers, and has two branches at u with search numbers k. Hence, $\lambda(G[u]) = ([k, I, u])$.

Rule (3): Since $\tau^1_1 = I$ and $\lambda(G_2[u]) \geqslant ([k, E, u])$, u has at least three branches in G with search numbers k, two from G_1 and one from G_2. By Theorem 4, we have that $ns(G) \geqslant k + 1$. Since u has no branches with search numbers greater than $k + 1$, $\lambda(G[u]) = ([k + 1, H, u])$.

Let $s = \max\{ns(G_2), ns(G'_2)\}$. When $s < k$, $\lambda((G_2 \cup G'_2)[u]) \leqslant ([k, H, u]) \leqslant \lambda((G_1 \cup G'_1)[u])$. When $s = k$, we have either $ns(G_2) = k$ and the length of $\lambda(G_2[u])$ equals 1 or $ns(G'_2) = k$ and the length of $\lambda(G'_2[u])$ equals 1. From the monotonicity of merging rules (1)–(3), we can verify easily that $\lambda((G_1 \cup G'_1)[u]) \geqslant \lambda((G_2 \cup G'_2)[u])$.

Assume that the two statements are true when $l_1 < t$ for some integer $t > 1$. In the following, we prove the two statements when $l_1 = t$.

Rule (4): Let P be an avenue of G_1 containing χ^1_1. Since $ns(G_1[u; \chi^1_1] \cup G_2) < k$, $G_1 \cup G_2$ can be cleared along P using k searchers. It follows that $ns(G) = k$. From the definition of avenues, P is also an avenue of G. If χ^1_1 is a critical vertex of G_1, by the uniqueness of critical vertices, it is also a critical vertex of G closest to u. Thus, $\lambda(G[u]) = ([k, \tau^1_1, \chi^1_1])\&\lambda((G_1[u; \chi^1_1] \cup G_2)[u])$.

Next, if χ^1_1 is a critical edge of a block K in G_1, by the definition of labels, $\lambda(G_1[u, \chi^1_1][u]) \leqslant \lambda(G_1[u, \chi][u]) < \lambda_1$ for all critical edge χ of K. For any critical edge χ of K, $(G_1[u, \chi], G_1[u, \chi^1_1])$ is a graph pair. We consider two graph pairs $(G_1[u, \chi], G_1[u, \chi^1_1])$ and (G_2, G_2). Together with $ns(G_2) < k$, by the induction hypothesis of statement (ii), $\lambda((G_1[u; \chi^1_1] \cup G_2)[u]) \leqslant \lambda((G_1[u; \chi] \cup G_2)[u])$. It follows that χ^1_1 is also a critical edge of G such that $\lambda(G[u, \chi^1_1][u])$ is the minimum. Thus, $\lambda(G[u]) = ([k, \tau^1_1, \chi^1_1])$ & $\lambda((G_1[u; \chi^1_1] \cup G_2)[u])$.

Rule (5): When χ_1^1 is a critical vertex of G_1, it has two branches with search numbers k disjoint to $G_1[u;\chi_1^1]$. Since $ns(G_1[u;\chi_1^1]\cup G_2)=k$, χ_1^1 has three branches in G with search numbers k. By Theorem 4, we have that $ns(G)\geqslant k+1$. Hence, $\lambda(G[u])=([k+1,H,u])$.

Next, we consider the case when χ_1^1 is a critical edge of G_1 in a block K. If $ns(G_2)=k$, $ns(G_1[u;\chi']\cup G_2)\geqslant k$ for all edges χ' of K. Next, if $ns(G_2)<k$, similar to the arguments in the above proof of rule (4), then $\lambda(G_1[u;\chi_1^1]\cup G_2)[u])\leqslant\lambda(G_1[u;\chi]\cup G_2)[u])$ for all critical edges χ of K. Since $ns(G_1[u;\chi_1^1]\cup G_2)\geqslant k$, $ns(G_1[u;\chi']\cup G_2)\geqslant k$ for all edges χ' of K. By Theorem 4, we have that $ns(G)\geqslant k+1$. Hence, $\lambda(G[u])=([k+1,H,u])$.

In the following, we prove the correctness of statement (ii) when $l_1=t$. Since $l_1>1$, $\tau_1^1=M_v$ or M_b. From the monotonicity of merging rules (1)–(5), the correctness of statement (ii) can be verified easily for the cases when $\lambda_2\leqslant([k,I,u])$, and $\lambda((G_1\cup G_1')[u])=([k+1,H,u])$. Thus, the rest case to be proved is $ns(G_2)=k$, $l_2>1$ ($\tau_1^2=M_v$ or M_b), and $\lambda((G_1\cup G_1')[u])<([k+1,H,u])$.

Since $\lambda(G_2[u])\leqslant\lambda(G_1[u])$ and $l_1,l_2>1$, $\lambda(G_2[u;\chi_1^2][u])\leqslant\lambda(G_1[u;\chi_1^1][u])<([k,H,u])$. Since $\tau_1^1=M_v$ or M_b and $\lambda((G_1\cup G_1')[u])<([k+1,H,u])$, $\lambda(G_1'[u])<([k,H,u])$. Thus, $(G_1[u;\chi_1^1],G_2[u;\chi_1^2])$ and (G_1',G_2') are graph pairs with $ns(G_1[u;\chi_1^1])<k$ and $ns(G_1')<k$. By induction hypothesis, $\lambda((G_1[u;\chi_1^1]\cup G_1')[u])\geqslant\lambda((G_2[u;\chi_1^2]\cup G_2')[u])$. Together with merging rule (4), $\lambda((G_1\cup G_1')[u])\geqslant\lambda((G_2\cup G_2')[u])$. □

5.2. Block merging

Before introducing the block merging rules, we first prove the label property of re-rooting that will be used later.

Lemma 19. *Let G_1, G_2, and G_3 be three block graphs sharing a vertex u only and v be a non-cut vertex of G_3 adjacent to u. If $\lambda(G_1[u])\geqslant\lambda(G_2[u])$, then $\lambda((G_1\cup G_3)[v])\geqslant\lambda((G_2\cup G_3)[v])$.*

Proof. Let G_{i3} denote $G_i\cup G_3$ for $i=1,2$, and let $[k_{i3},\tau_{i3},\chi_{i3}]$ and $[k_{i3}',\tau_{i3}',\chi_{i3}']$ be the first records of $\lambda(G_{i3}[u])$ and $\lambda(G_{i3}[v])$, respectively. Notice that $k_{i3}=k_{i3}'$. Let K be the block of G_3 containing v. The following re-rooted labelling rules can be verified easily.

(1) $\tau_{i3}=H$.
  (1.1) if $ns(G_{i3}\backslash\{v\})<k_{i3}$, then $\lambda(G_{i3}[v])=([k_{i3},H,v])$;
  (1.2) if $ns(G_{i3}\backslash\{v\})=k_{i3}$, then $\lambda(G_{i3}[v])=([k_{i3},E,v])$.
(2) $\tau_{i3}=E$.
  (2.1) if K is critical block of G_{i3}, then $\lambda(G_{i3}[v])=([k_{i3},M_b,\chi_{i3}'])\&\lambda(G_{i3}[v;\chi_{i3}'][v])$ where χ_{i3}' is an edge of K;
  (2.2) otherwise, $\lambda(G_{i3}[v])=([k_{i3},E,v])$.
(3) $\tau_{i3}=I$.
  (3.1) if K is critical block of G_{i3}, then $\lambda(G_{i3}[v])=([k_{i3},M_b,\chi_{i3}'])\&\lambda(G_{i3}[v;\chi_{i3}'][v])$ where u is an endpoint of χ_{i3}' in K;
  (3.2) otherwise, $\chi_{i3}'=u$ and $\lambda(G_{i3}[v])=([k_{i3},M_v,u])\&\lambda(G_{i3}[v;u][v])$.
(4) $\tau_{i3}=M_v$ or M_b.
  (4.1) $\lambda(G_{i3}[v])=([k_{i3},\tau_{i3},\chi_{i3}'])\&\lambda(G_{i3}[v;\chi_{i3}'][v])$ where $\chi_{i3}'=\chi_{i3}$.

Graph triplet (G_1,G_2,G_3) is an ordered triplet of block graphs such that G_1, G_2 and G_3 share a vertex u only and $\lambda(G_1[u])\geqslant\lambda(G_2[u])$. We will prove the statement that all such graph triplets satisfy $\lambda(G_{13}[v])\geqslant\lambda(G_{23}[v])$.

By Lemma 18, we have that $\lambda(G_{13}[u])\geqslant\lambda(G_{23}[u])$. If $\lambda(G_{13}[u])>\lambda(G_{23}[u])$, the lemma can be verified easily by the above re-rooted labelling rules. The rest is to prove the case when $\lambda(G_{13}[u])=\lambda(G_{23}[u])$, i.e., $k_{13}=k_{23}$ and $\tau_{13}=\tau_{23}$.

When $\tau_{13}=H$, we have that $\lambda(G_{23}[v])\leqslant([k_{13},E,v])$. Thus, we only need to consider case (1.1). In case (1.1), $ns(G_{13}\backslash\{v\})<k_{13}$. By Lemma 18, we have that $\lambda(G_{23}\backslash\{v\})\leqslant\lambda(G_{13}\backslash\{v\})$. Thus, $\lambda(G_{23}[v])\leqslant\lambda(G_{13}[v])$.

When $\tau_{13}=E$, we have that $\lambda(G_{23}[v])\geqslant([k_{13},E,v])$. Thus, we only need to consider case (2.1). Since $\lambda(G_{13}[u])=\lambda(G_{23}[u])$, we have that if K is a critical block of G_{23}, then K is also a critical block of G_{13}. Thus, $\tau_{13}'=\tau_{23}'=M_b$.

When $\tau_{13}=I$, or M_v, M_b, we have that τ_{13}' is either M_v or M_b. Thus, $\tau_{13}'\geqslant\tau_{23}'$ for all cases.

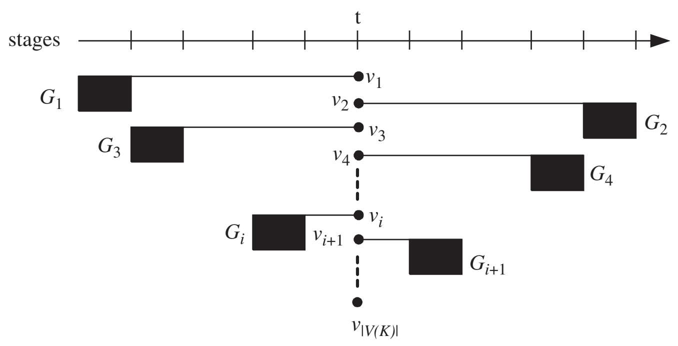


Fig. 9. The guarded interval diagram for a greedy search strategy.

For those cases that $\tau'_{13} = M_v$ or M_b, we need to prove that the tails of their labels satisfy the statement.

In cases (2.1), (3.1), and (4.1), the avenue of G_{i3} is on $[G_3]_{u,v}$. The tails of their labels are $\lambda(G_{i3}[v;\chi'_{i3}][v])$ where χ'_{i3} is an edge of K for case (2.1), $\lambda(G_{i3}[v;\chi'_{i3}][v])$ where u is an endpoint of χ'_{i3} in K for case (3.1), and $\lambda(G_{i3}[v;\chi'_{i3}][v])$ where $\chi'_{i3} = \chi_{i3}$ for case (4.1). Since the avenue of G_{i3} belongs to G_3, it can be verified easily by induction that the statement is true when $\tau_{13} = \tau_{23}$. Next, we consider the case when $\tau_{13} > \tau_{23}$. The only subcases are those $\tau_{13} = I$ and $\tau_{23} = E$. Since $\lambda(G_{23}[v;\chi'_{23}][v]) \leqslant \lambda(G_{23}[v;\chi'_{13}][v]) = \lambda(G_{13}[v;\chi'_{13}][v])$, the statement is true.

In cases 3.2 and 4.1, the avenue of G_{i3} belongs to $[G_{i3}]_{K,u}$. The tails of their labels are $\lambda(G_{i3}[v;u][v])$ for case (3.2), and $\lambda(G_{i3}[v;\chi'_{i3}][v])$ where $\chi'_{i3} = \chi_{i3}$ for case (4.1). Since the avenue of G_{i3} belongs to $[G_{i3}]_{K,u}$, it can be verified easily by induction that the statement is true when $\tau_{13} = \tau_{23}$. Next, we consider the case when $\tau_{13} > \tau_{23}$. Since $G_{23}[v;u] \subset G_{13}[v;\chi'_{13}]$, $\lambda(G_{23}[v;u][v]) \leqslant \lambda(G_{13}[v;\chi'_{13}][v])$, the statement is true. □

Lemma 20. *Let K be a block of a block graph G with* $|V(K)| \geqslant 3$ *and u a non-cut vertex of G in K. Let x and y be two vertices of K such that* $\lambda([G]_{K,x}[x]) \geqslant \lambda([G]_{K,z}[z])$ *and* $\lambda([G]_{K,y}[y]) \geqslant \lambda([G]_{K,z}[z])$ *for any* $z \in V(K)$ *other than* x, y. *Then* (1) (x, y) *is an edge of K whose enclosure subgraph rooted at u has the smallest label*; (2) $ns(G) \geqslant ns(\{G\}_{x,y}) + 1$.

Proof. (1) Assume that K has another vertex v other than x, y, u. Let $G' = \{G\}_{x,y} \cap \{G\}_{x,v}$. Then $\{G\}_{x,v} = [G]_{K,y} \cup G'$ and $\{G\}_{x,y} = [G]_{K,v} \cup G'$. Notice that $\lambda(G'[y]) = \lambda(G'[v])$ since y and v are non-cut vertices of G' in the same block. Since $\lambda([G]_{K,y}[y]) \geqslant \lambda([G]_{K,v}[v])$, by Lemma 19, we have that $\lambda(([G]_{K,y} \cup G')[u]) \geqslant \lambda(([G]_{K,v} \cup G')[u])$. Thus, $\lambda(\{G\}_{x,v}[u]) \geqslant \lambda(\{G\}_{x,y}[u])$.Moreover, assume that K has one another vertex w other than x, y, u, v. Since $\lambda([G]_{K,x}[x]) \geqslant \lambda([G]_{K,w}[w])$, by the same argument as above, we have that $\lambda(\{G\}_{w,v}[u]) \geqslant \lambda(\{G\}_{x,v}[u])$. It follows that $\lambda(\{G\}_{w,v}[u]) \geqslant \lambda(\{G\}_{x,y}[u])$.

(2) From the types of rooted block graphs (see Fig. 5), we obtain that there exists an optimal search strategy for G in which the start vertex u' and the terminal vertex v' are both in $V([G]_{K,x}) \cup V([G]_{K,y})$. By Lemma 3(1), we have that $ns(G) = os(G, u', v') \geqslant ns(\{G\}_{x,y}) + 1$. □

In the block merging, we compute the structure information of $G[u]$ from the ones of the subgraphs rooted at the cut-children of u, using the concept of the *greedy search strategy* described as follows to estimate $ns(G)$. Let G be a block graph with a non-cut vertex u, and K the block containing u. Suppose $v_1, v_2, \ldots, v_r$ are the cut vertices of G in K. In brief, we denote $[G]_{K,v_i}$ by G_i and $\lambda_i = \lambda(G_i[v_i]) = ([s^i_1, \tau^i_1, \chi^i_1], [s^i_2, \tau^i_2, \chi^i_2], \ldots, [s^i_{l_i}, \tau^i_{l_i}, \chi^i_{l_i}])$, for $i = 1, 2, \ldots, r$. Without loss of generality, we assume that $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_r$. Let $I_o = \{i \mid i$ is an odd integer for $1 \leqslant i \leqslant r\}$ and $I_e = \{i \mid i$ is an even integer for $1 \leqslant i \leqslant r\}$. A greedy search strategy S for G at K is conceptually described as follows (see Fig. 9):

```
(1) for each $i \in I_o$ in increasing order,
        clear $G_i$ by an optimal oriented strategy for it to $v_i$, and keep $v_i$ guarded after $G_i$ cleared;
(2) for each $i \notin I_o$, $1 \leqslant i \leqslant r$,
        place a searcher on $v_i$;
(3) for each $j \notin I_e$, $1 \leqslant j \leqslant r$,
        remove the searcher from $v_j$;
```

(4) for each $j \in I_e$ in decreasing order,
    clear G_j by an optimal oriented strategy for it from v_j.

Obviously, S can be divided into $r+1$ intervals, including r intervals in which one G_i is cleared per interval, and a special stage at which only all vertices of K are guarded. We define the *weight* of each rooted subgraph $G_i[v_i]$, denoted by ω_i, to be the number of searchers used in the ith interval, i.e., $\omega_i = os(G_i, v_i) + \lfloor (i-1)/2 \rfloor$. The maximum number of searchers used before stage t is called the *left weight* of G at K, denoted by ω_{L}, i.e., $\omega_{\mathrm{L}} = \max\{\omega_i | i \in I_o\}$. The maximum number of searchers used after stage t is called the *right weight* of G at K, denoted by ω_{R}, i.e., $\omega_{\mathrm{R}} = \max\{\omega_i | i \in I_e\}$. It is clear that $\omega_{\mathrm{L}} \geqslant \omega_{\mathrm{R}}$ and $\#(S) = \max\{\omega_{\mathrm{L}}, |V(K)|\}$. Any greedy search strategy for G is near optimal, which will be shown in Lemma 21 later. From the order among ω_{L}, $|V(K)|$ and ω_{R}, we divide greedy search strategies into three patterns:

(1) *centralized* : $|V(K)| \geqslant \omega_{\mathrm{L}}$ and $|V(K)| > \omega_{\mathrm{R}}$,
(2) *balanced* : $\omega_{\mathrm{L}} = \omega_{\mathrm{R}} \geqslant |V(K)|$,
(3) *skewed* : $\omega_{\mathrm{L}} > |V(K)|$ and $\omega_{\mathrm{L}} > \omega_{\mathrm{R}}$.

Lemma 21. *Let K be a block of a block graph G and S a greedy search strategy for G at K.* (1) *If S is centralized, then $ns(G) = |V(K)|$.* (2) *If S is balanced, then $ns(G) = \omega_{\mathrm{L}} = \omega_{\mathrm{R}}$.* (3) If S is skewed, then $ns(G) = \omega_{\mathrm{L}}$ or $\omega_{\mathrm{L}} - 1$.

Proof. *Centralized*: Since S is centralized, we have that $ns(G) \leqslant \#(S) \leqslant |V(K)|$. However, $ns(G) \geqslant |V(K)|$. Thus, we conclude that $ns(G) = |V(K)|$.

Balanced: Since S is balanced, $ns(G) \leqslant \#(S) \leqslant \omega_{\mathrm{L}}$. Suppose p is the smallest even index such that $\omega_p = \omega_{\mathrm{R}} = \omega_{\mathrm{L}}$. Let H_j denote the subgraph $G[\bigcup_{i=j}^{p} V(G_i)]$ for $j = 1, 2, \cdots, p$. By Lemma 20(2), we have that $ns(G) \geqslant ns(H_1) \geqslant ns(H_3) + 1 \geqslant \cdots \geqslant ns(H_{p-1}) + (p-2)/2$. Since $\omega_{p-1} \geqslant \omega_p = \omega_{\mathrm{L}}$ and $\omega_{p-1} \leqslant \omega_{\mathrm{L}}$, $\omega_{p-1} = \omega_p = \omega_{\mathrm{L}}$. Thus, $os(G_{p-1}, v_{p-1}) = os(G_p, v_p)$. It is easy to obtain $os(G_{p-1}, v_{p-1}) = os(G_{p-1} \cup \{(v_{p-1}, v_p)\}, v_p)$. From the vertex merge on $(G_{p-1} \cup \{(v_{p-1}, v_p)\})[v_p]$ and $G_p[v_p]$, we have that $ns(H_{p-1}) = os(G_p, v_p)$. It follows that $ns(G) \geqslant os(G_p, v_p) + (p-2)/2 = \omega_p = \omega_{\mathrm{L}}$. Hence, $ns(G) = \omega_{\mathrm{L}}$.

Skewed: Since S is skewed, $ns(G) \leqslant \#(S) \leqslant \omega_{\mathrm{L}}$. Suppose q is the smallest odd index such that $\omega_q = \omega_{\mathrm{L}}$. Let H'_j denote the subgraph $G[\bigcup_{i=j}^{q} V(G_i)]$ for $j = 1, 2, \ldots, q$. By Lemma 20(2), we have that $ns(G) \geqslant ns(H'_1) \geqslant ns(G_q) + (q-1)/2$. By Lemma 1, we have that $ns(G_q) \geqslant os(G_q, v_q) - 1$. It follows that $ns(G) \geqslant os(G_q, v_q) + (q-1)/2 - 1 = \omega_q - 1 = \omega_{\mathrm{L}} - 1$. Hence, $\omega_{\mathrm{L}} - 1 \leqslant ns(G) \leqslant \omega_{\mathrm{L}}$. $\square$

In the following, we use the notations $G' = G \backslash V(G_1)$, $G^- = G[V(G_1[v_1; \chi_1^1]) \cup V(G')]$ and $G'' = \{G\}_{v_1, v_2}$. The block merging proceeds by the rules described as follows.

(1) S is centralized,
  (a) if $ns(G \backslash \{u\}) < |V(K)|$, then $\lambda(G[u]) = ([|V(K)|, H, u])$.
  (b) if $ns(G \backslash \{u\}) = |V(K)|$, then $\lambda(G[u]) = ([|V(K)|, E, u])$.
(2) S is balanced,
  (a) if $ns(G') = \omega_{\mathrm{L}}$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, M_b, (v_1, v_2)])\&\lambda(G''[u])$.
  (b) if $ns(G') < \omega_{\mathrm{L}}$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$.
(3) S is skewed,
  (a) $\omega_1 = \omega_{\mathrm{L}}$ and $\tau_1^1 = H$ or E, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$.
  (b) $\omega_1 = \omega_{\mathrm{L}}$ and $\tau_1^1 = I$, M_v or M_b,
    (i) if $ns(G^-) = \omega_{\mathrm{L}} - 1$ and $\tau(G^-[u]) = H$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, H, u])$.
    (ii) if $ns(G^-) = \omega_{\mathrm{L}} - 1$ and $\tau(G^-[u]) \neq H$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$.
    (iii) if $ns(G^-) < \omega_{\mathrm{L}} - 1$ and $\tau_1^1 = I$ or M_v, then $\lambda(G[u]) = ([\omega_{\mathrm{L}} - 1, M_v, \chi_1^1])\&\lambda(G^-[u])$.
    (iv) if $ns(G^-) < \omega_{\mathrm{L}} - 1$ and $\tau_1^1 = M_b$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}} - 1, M_b, \chi_1^1])$ & $\lambda(G^-[u])$.
  (c) $\omega_1 < \omega_{\mathrm{L}}$,
    (i) if $ns(G'') = \omega_{\mathrm{L}} - 1$ and $\tau(G''[u]) = H$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, H, u])$.
    (ii) if $ns(G'') = \omega_{\mathrm{L}} - 1$ and $\tau(G''[u]) \neq H$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$.
    (iii) if $ns(G'') < \omega_{\mathrm{L}} - 1$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}} - 1, M_b, (v_1, v_2)])\&\lambda(G''[u])$.

Lemma 22. *The block merging rules correctly compute $\lambda(G[u])$.*

Proof. *Centralized*: Let $S=(X_1, X_2, \dots, X_\alpha)$ in which $X_t \supseteq V(K)$ and $[l_u, r_u]$ is the guarded interval of u. Obviously, $t < r_u$. Let us consider $S' = (X_1', X_2', \dots, X_\alpha')$ where $X_i' = X_i$ for $1 \leqslant i \leqslant r_u$, $X_i' = X_i \cup \{u\}$ for $r_u + 1 \leqslant i \leqslant \alpha$. Since S is centralized, $ns(G) = |V(K)| > \omega_{\mathrm{L}} = \max\{|X_i| \,|\, t+1 \leqslant i \leqslant \alpha\}$. It follows that $|X_i'| \leqslant |V(K)|$ for $1 \leqslant i \leqslant \alpha$. Thus, S' is an oriented search strategy for G to u using $ns(G)$ searchers. Then $os(G,u) = ns(G)$. By Lemma 12, we have that $\tau(G[u]) = H$ or E. From the definition of hubs, we have that if $ns(G\backslash\{u\}) < ns(G)$,then $\lambda(G[u]) = ([|V(K)|, H, u])$; otherwise $\lambda(G[u]) = ([|V(K)|, E, u])$.

Balanced: If $ns(G') = \omega_{\mathrm{L}}$, by Lemma 10, then K is a critical block of G. From Lemma 20(1), (v_1, v_2) is the edge in K whose enclosure subgraph rooted at u has the minimum label. Thus, $\lambda(G[u]) = ([\omega_{\mathrm{L}}, M_b, (v_1, v_2)])$ & $\lambda(G''[u])$. Next, if $ns(G') < \omega_{\mathrm{L}}$, by Lemma 3(2), then $os(G[V(G') \cup \{v_1\}], u, v_1) \leqslant \omega_{\mathrm{L}}$. Since $os(G_1, v_1) \leqslant \omega_{\mathrm{L}}$, $os(G,u) = \omega_{\mathrm{L}} = ns(G)$. Thus, $\tau(G[u]) = H$ or E. It is clear that any greedy search strategy for $G\backslash\{u\}$ at $K\backslash\{u\}$ is also balanced. It follows that $ns(G\backslash\{u\}) = ns(G)$. Thus, $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$.

Skewed: Since S is skewed, $ns(G) \leqslant \omega_{\mathrm{L}}$. In the case (a), $ns(G_1) = os(G_1, v_1) = \omega_{\mathrm{L}}$. It follows that $ns(G) \geqslant \omega_{\mathrm{L}}$. Thus, $ns(G) = \omega_{\mathrm{L}}$. In the case (b), $ns(G_1) = os(G_1, v_1) - 1 = \omega_{\mathrm{L}} - 1$. Thus, $\lambda(G_1[v_1]) \geqslant ([\omega_{\mathrm{L}} - 1, I, v_1])$. From a greedy search strategy for $G[V(G') \cup \{v_1\}]$ at K, we have that $ns(G[V(G') \cup \{v_1\}]) \leqslant \omega_{\mathrm{L}} - 1$. Then $ns(G)$ is determined by $ns(G^-)$ which is obtained by the vertex merge on $G_1[v_1]$ and $G[V(G') \cup \{v_1\}][v_1]$. By the vertex merging rules, if $ns(G^-) = \omega_{\mathrm{L}} - 1$, then $ns(G) = \omega_{\mathrm{L}}$; otherwise $ns(G) = \omega_{\mathrm{L}} - 1$.

In the case (c), $os(G_i, v_i) < \omega_{\mathrm{L}}$ for all i. If $ns(G'') = \omega_{\mathrm{L}} - 1$, by Lemma 20(2), then $ns(G) \geqslant ns(G'') + 1 = \omega_{\mathrm{L}}$. Thus, $ns(G) = \omega_{\mathrm{L}}$. Next, if $ns(G'') < \omega_{\mathrm{L}} - 1$, by Lemma 3(2), then $os(G[V(G'') \cup \{v_1\}], v_1, v_2) \leqslant \omega_{\mathrm{L}} - 1$. Since $os(G_1, v_1) \leqslant \omega_{\mathrm{L}} - 1$ and $os(G_2, v_2) \leqslant \omega_{\mathrm{L}} - 1$, there exists an oriented search strategy for G from a vertex in G_1 to a vertex in G_2 using at most $\omega_{\mathrm{L}} - 1$ searchers. It follows that $ns(G) \leqslant \omega_{\mathrm{L}} - 1$. Thus, $ns(G) = \omega_{\mathrm{L}} - 1$.

We now consider the case when $ns(G) = \omega_{\mathrm{L}}$. Since $\omega_{\mathrm{L}} > \omega_{\mathrm{R}}$, by the same argument used in the centralized case, we have that $os(G,u) = ns(G)$. From the definition of hubs, we have that if $ns(G\backslash\{u\}) < ns(G)$, then $\lambda(G[u]) = ([\omega_{\mathrm{L}}, H, u])$; otherwise $\lambda(G[u]) = ([\omega_{\mathrm{L}}, E, u])$. Next, we consider the case when $ns(G) = \omega_{\mathrm{L}} - 1$. If $\omega_1 = \omega_{\mathrm{L}}$, they are similar to the vertex merge rules (1), (2) and (5) on $G_1[v_1]$ and $G[V(G') \cup \{v_1\}][v_1]$. Next, if $\omega_1 < \omega_{\mathrm{L}}$, it is clear that any greedy search strategy for G' at $K\backslash\{v_1\}$ is balanced. It follows that $ns(G') = \omega_{\mathrm{L}} - 1 = ns(G)$. By Lemma 10, K is a critical block of G. Thus, $\lambda(G[u]) = ([\omega_{\mathrm{L}}, M_b, (v_1, v_2)])$&$\lambda(G''[u])$. □

Next, we show how to obtain the values $ns(G\backslash\{u\})$, $ns(G')$, $ns(G'')$ and $ns(G^-)$ described in the block merging rules.

(1) If S is centralized, we first compute $\lambda(G^*[v_1])$ by the block merging recursively where $G^* = G\backslash(V(G_1\backslash\{v_1\}) \cup \{u\})$. Next, we compute $\lambda((G\backslash\{u\})[v_1])$ by the vertex merging on $\lambda(G^*[v_1])$ and $\lambda(G_1[v_1])$.
(2) If S is balanced, we first compute $\lambda(G''[u])$ by the block merging recursively. Next, we transfer $\lambda(G''[u])$ to $\lambda(G''[v_2])$. At last, compute $\lambda(G'[v_2])$ by the vertex merging on $\lambda(G''[v_2])$ and $\lambda(G_2[v_2])$.
(3) If S is skewed, we recursively compute $\lambda(G^-[u])$ and $\lambda(G''[u])$ in (b) and (c), respectively.

In the above case (2), since u and v_2 are both non-cut vertices of G'' and have the same neighbors other than them, $\lambda(G''[u])$ can be transferred to $\lambda(G''[v_2])$ by replacing u by v_2 in the last record of $\lambda(G''[u])$ in constant time. Moreover, we show how to sort the labels efficiently in the block merging rules. Let $\lambda_i = (R_1^i, \dots, R_{l_i}^i)$ where $R_j^i = [s_j^i, \tau_j^i, \chi_j^i]$, for $i = 1, \dots, r$ and $j = 1, \dots, l_i$. Let $L = \sum_{i=1}^r l_i$. In the following, we propose a bottom-up algorithm to assign an index to each record of the labels. According to the indices, we can sort the labels in O(r) time.

```
Algorithm. INDEX($\lambda_1, \dots, \lambda_r$)
    let $\delta_j^i$ denote the index of the record $j$ in $\lambda_i$;
    let $\Gamma_i = l_i$ and $\delta_{l_i+1}^i = 0$, for $i = 1, \dots, r$;
    let $I = \{R_{\Gamma_i}^i \,|\, 1 \leqslant i \leqslant r\}$;
    for $i = 1$ to $L$ do
    (a)    select the smallest record $R_{\Gamma_t}^t$ in $I$ such that the ordered pair $(R_{\Gamma_t}^t, \delta_{\Gamma_t+1}^t)$
           is smallest in lexicographic order, and remove $R_{\Gamma_t}^t$ from $I$;
```

```
        (b)  $\delta^t_{\Gamma_t} = i$;
        (c)  $\Gamma_t = \Gamma_t - 1$;
        (d)  if $\Gamma_t \geqslant 1$, then insert $R^t_{\Gamma_t}$ to $I$;
    end for;
end INDEX.
```

The correctness of the algorithm INDEX is trivial. By comparing the indices $\delta^i_p > \delta^j_q$, we can obtain $(R^i_p, \dots, R^i_{l_i}) \geqslant (R^j_q, \dots, R^j_{l_j})$. That is, each comparison takes constant time. Therefore, the time complexity of the algorithm is $O(r \times L)$.

5.3. Construct an optimal strategy tree

In this section, we insert some rules into the vertex merging and the block merging for constructing an optimal strategy tree of $G[u]$. We first introduce the rules inserted into the vertex merging. Here, we use the same notation and assumption as Section 5.1. Let T_1 and T_2 be the given optimal strategy trees of $G_1[u]$ and $G_2[u]$ which are rooted at x_1 and x_2, respectively. In the following, we say "overlap x_1 with x_2" for redirecting all the children of x_2 to x_1 and then deleting x_2.

1. In the vertex merging rule (1), the main strategy path of G_1 can form an optimal strategy path of G. We insert the following rule after rule (1).
   (1′) if $id(x_2) = id(x_1)$, then overlap x_1 with x_2; otherwise add layer-arc $x_2 \rightarrow x_1$.
2. In the vertex merging rule (2), the concatenation of the main strategy paths of G_1 and G_2 can form an optimal strategy path of G. We insert the following rule after rule (2).
   (2′) overlap x_1 with x_2.
3. In the vertex merging rules (3) and (5), $\langle u \rangle$ forms an optimal strategy path of G. We insert the following rule after each of the rules (3) and (5).
   (3′) (a) create a new node x with $id(x) = ns(G)$ and set $\varphi(x) = u$; (b) for $i = 1, 2$, add layer-arcs $x_i \rightarrow x$.
4. In the vertex merging rule (4), the strategy path of G_1 forms an optimal strategy path of G. We insert the following rule after rule (4).
   (4′) (a) remove the subtree rooted at $\varphi^{-1}(\chi^1_2)$ from T_1, which results T'_1; (b) merge strategy trees T'_1 and T_2 recursively to a tree T'; (c) add a layer-arc $x' \rightarrow \varphi^{-1}(\chi^1_1)$ where x' is the root of T'.

In the following, we introduce the rules inserted into the block merging. Here, we use the same notation and assumption as the ones in Section 5.2. In addition, let T_i be the given optimal strategy tree of $G_i[v_i]$ rooted at x_i for $1 \leqslant i \leqslant r$.

1. In the block merging rules with $\tau(G[u]) = H$, $\langle u \rangle$ forms an optimal strategy path of G. We insert the following rule after each of those rules.
   (1″) (a) create a new node x with $id(x) = (ns(G))'$ and $\varphi(x) = u$; (b) construct an optimal strategy tree $\hat{T}$ of $(G \backslash \{u\})[v]$ where v is a vertex adjacent to u; (c) add a layer-arc $\hat{x} \rightarrow x$ where $\hat{x}$ is the root of $\hat{T}$.
2. In the block merging rules with $\tau(G[u]) = E$, we have that $ns(G \backslash V(G_1)) < ns(G)$. By Lemma 3(2), there exists an optimal search strategy for G from u to a vertex in G_1. Thus, (v_1, u) is a part of an optimal strategy path of G and the following rule is inserted after each of those rules.
   (2″) (a) create a directed path $y_1 \rightarrow y_2 \rightarrow y_3$ in which $\varphi(y_1) = v_1$, $\varphi(y_2) = (v_1, u)$, $\varphi(y_3) = u$, and $id(y_i) = ns(G)$ for $i = 1, 2, 3$; (b) if $id(x_1) < ns(G)$, then add a layer-arc $x_1 \rightarrow y_1$; otherwise overlap y_1 with x_1; (c) construct an optimal strategy tree T' of $G'[u]$; (d) add a layer-arc $x' \rightarrow y_2$ where x' is the root of T'.
3. In the block merging rules with $\tau(G[u]) = M_v$ or M_b, $\chi(G[u])$ is equal to either (v_1, v_2) or χ^1_1, which corresponds to an optimal strategy path of G.
   3.1 If K is a critical block, i.e., $\chi(G[u]) = (v_1, v_2)$, then the following rule is inserted.
   (3″) (a) create a directed path $z_1 \rightarrow z_3 \leftarrow z_2$ where $\varphi(z_1) = v_1$, $\varphi(z_2) = v_2$, $\varphi(z_3) = (v_1, v_2)$, and $id(z_i) = ns(G)$ for $i = 1, 2, 3$; (b) for $j = 1, 2$, if $id(x_j) < ns(G)$, then add a layer-arc $x_j \rightarrow z_j$, otherwise overlap z_j with x_j; (c) construct an optimal strategy tree T'' for $G''[u]$ by the block merging recursively; (d) add a layer-arc $x'' \rightarrow z_3$, where x'' is the root of T''.

3.2 If K is not a critical block, i.e., $\chi(G[u]) = \chi_1^1$, then the following rule is inserted.

(4″) (a) construct an optimal strategy tree T^- for $G^-[u]$; (b) remove the subtree rooted at $\varphi^{-1}(\chi_2^1)$ from T_1; (c) add a layer-arc $x^- \to \varphi^{-1}(\chi_1^1)$, where x^- is the root of T^-.

In the step (b) of the rule (1″), we construct $\hat{T}$ for various cases described as follows:

(1) If S is centralized, then (i) construct an optimal strategy tree T^* of $G^*[v_1]$ by the block merging recursively, where $G^* = G\backslash(V(G_1\backslash\{v_1\}) \cup \{u\})$; (ii) construct T' by the vertex merging on T^* and T_1 at v_1.
(2) If S is skewed and $\omega_1 < \omega_{\mathrm{L}}$, then $\hat{T}$ is constructed by the rule (3″).
(3) If S is skewed and $\omega_1 = \omega_{\mathrm{L}}$, then $\hat{T}$ is constructed by the rule (4″).

In step (c) of the rule (2″), we construct T' as the greedy strategy tree of $G'[u]$ introduced later for efficiency. Let $v_1, v_2, \ldots, v_r$ be the cut vertices of G in K where $r < |V(K)|$ and $v_{|V(K)|} = u$. Let T_i be an optimal strategy tree of $[G]_{K,v_i}[v_i]$ rooted at x_i for $i = 1, 2, \ldots, r$. Without loss of generality, we assume that $\lambda([G]_{K,v_1}[v_1]) \geqslant \lambda([G]_{K,v_2}[v_2]) \geqslant \cdots \geqslant \lambda([G]_{K,v_r}[v_r])$. Given T_i and $\lambda([G]_{K,v_i}[v_i])$ for $i = 1, 2, \ldots, r$, the greedy strategy tree of $G[u]$ at K is recursively constructed as follows.

```
**Function** GREEDY_TREE($G[u]$, $K$): rooted tree;
(1) **if** $G[u]$ is a rooted complete graph, **then**
    (a) create a directed path $z_1 \to z_2 \to \cdots \to z_{|V(K)|}$ with $\varphi(z_i) = v_i$ and $id(z_i) = i$ for $1 \leqslant i \leqslant |V(K)|$;
    (b) **return** the constructed directed path.
(2) **else**
    (a) $s = \max\{\omega_{\mathrm{L}}, |V(K)|\}$;
    (b) create a directed path $z_1 \to z_3 \leftarrow z_2$ in which $\varphi(z_1) = v_1$, $\varphi(z_2) = v_2$, $\varphi(z_3) = (v_1, v_2)$, and $id(z_i) = s$
        for $i = 1, 2, 3$;
    (c) for $j = 1, 2$, if $id(x_j) < s$ or $id(x_j) = (s)'$, then add a layer-arc $x_j \to z_j$; otherwise overlap $z_j$ with $x_j$;
    (d) $T'$= GREEDY_TREE($\{G\}_{v_1,v_2}$, $K\backslash\{v_1\}$, $s-1$);
    (e) add a layer-arc $x' \to z_3$ where$x'$ is the root of $T'$;
    (f) **return** the constructed strategy tree;
(3) **end if**;
**end** GREEDY_TREE.
```

Analogous to the proofs of Lemmas 18 and 22, the constructed strategy tree of $G[u]$ by the inserted rules can be shown to be optimal.

Lemma 23. *Let $G[u]$ be a rooted block graph. The number of nodes in the strategy tree of $G[u]$ constructed by the inserted rules is bounded by* $\mathrm{O}(|V(G)|)$.

Proof. Let $\mathscr{T}$ be the optimal strategy tree constructed by the inserted rules and $\mathscr{D}$ the strategy path system corresponding to $\mathscr{T}$. From the main merging routine, we have that any non-cut vertex of G appears only once in $\mathscr{D}$. Since the length of the strategy path containing a non-cut vertex is equal to one, the strategy paths of length greater than 1 are all induced by cut vertices. Note that the strategy paths in $\mathscr{D}$ are edge disjoint paths. For each block K with r cut vertices, there are at most r edges of K in $\mathscr{D}$. Let us consider the directed tree T rooted at u in which $V(T)$ is the set of all cut vertices of G and $E(T) = \{x \to y | y \text{ is the parent of } x \text{ in } G[u] \text{ for all } x, y \in V(T)\}$. Since T is a tree, $|E(T)| = |V(T)| - 1$. It follows that there are $\mathrm{O}(c)$ edges in $\mathscr{D}$ where c is the number of cut vertices of G. Thus, the total number of vertices and edges on the strategy paths of length greater than 1 in $\mathscr{D}$ is $\mathrm{O}(c)$.

For each cut vertex v of G, the number of the strategy paths of length 1 containing v is bounded by the number of forks at v not containing u. It follows that the number of the strategy paths of length 1 containing a cut vertex is also bounded by the number of arcs in T, $\mathrm{O}(c)$.

In total, the number of vertices and edges in $\mathscr{D}$ is bounded by $\mathrm{O}(|V(G)|)$. Thus, $|V(\mathscr{T})| = \mathrm{O}(|V(G)|)$. □

Together with Lemma 17, we obtain the following theorem.

Theorem 24. *Given an optimal strategy tree of $G[u]$. An optimal search strategy for G can be constructed in* $\mathrm{O}(|V(G)|)$ *time.*

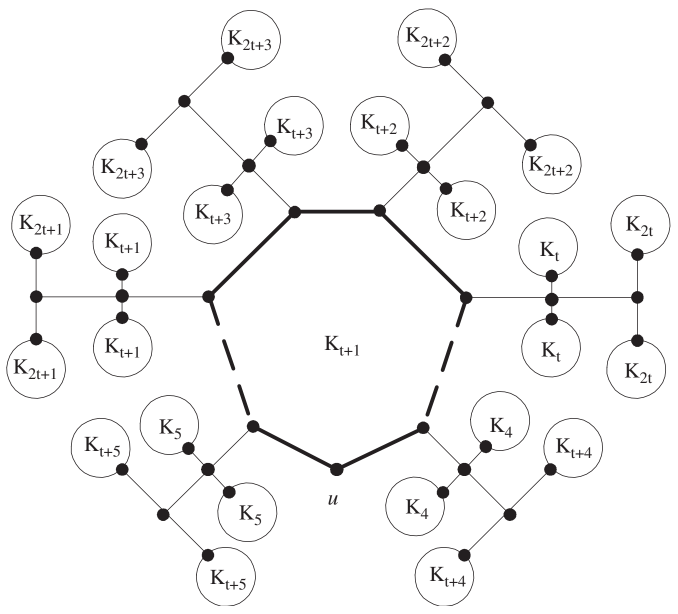


Fig. 10. A rooted block graph $G[u]$ in which K_i represents a block of size i.

6. Time complexity

By the definition of labels, each record of a label corresponds to at least one cut vertex. Thus, we obtain the following proposition.

Proposition 25. *For any rooted block graph $G[u]$ with c cut vertices, the length of $\lambda(G[u])$ is bounded by c.*

The time complexity of our algorithm is analyzed as follows.

Theorem 26. *Let $G[u]$ be a rooted block graph with $|V(G)| = n$. The time complexity for computing $\lambda(G[u])$ and constructing $\mathscr{T}(G[u], ns(G))$* is $O(bc + c^2 + n)$ *where b is the number of blocks and c is the number of cut vertices.*

Proof. First of all, we consider the vertex merging on rooted block graphs $G_1[u]$ and $G_2[u]$. Let l_1 and l_2 be the lengths of $\lambda(G_1[u])$ and $\lambda(G_2[u])$, respectively. From the vertex merging rules and the inserted rules, there are at most $l_1 + l_2$ recursive calls and each recursive call takes constant time. By Proposition 25, $l_1 + l_2$ is bounded by c. It is clear that the vertex merging routine totally proceeds $b - 1$ times. Thus, it takes $O(bc)$ time in total.

Next, we consider the block merging at a non-cut vertex u of a block K. Let $v_1, \ldots, v_r$ be the cut vertices in K. Let $G_i = [G]_{K,v_i}$ and l_i denote the length of $\lambda(G_i[v_i])$ for $1 \leqslant i \leqslant r$. Let $L = \sum_{i=1}^{r} l_i$. By Proposition 25, L is also bounded by c.

Before the block merging routine proceeding, we assign an index to each record by the algorithm INDEX and sorting r labels by the indices of their first records. It takes $O(r \times L)$ time. In a block merging, there are at most L recursive calls invoked by the block merging rule (3). In such a recursive call, it takes $O(r)$ time to resort the labels, to recompute the weights of the rooted subgraphs, and to decide the pattern of a greedy search strategy at K. On the other hand, there are at most r recursive calls invoked by the block merging rules other than (3). In such a recursive call, it takes $O(L)$ time to proceed an extra vertex merging and $O(r)$ time to recompute the weights of the rooted subgraphs and to decide the pattern of a greedy search strategy at K. At last, it takes $O(|V(K)|)$ time to construct a greedy strategy tree or to construct a directed path of the block containing all the non-cut vertices in K.

Suppose the block merging routine proceeds t times in total. Let $r_1, r_2, \ldots, r_t$ be the numbers of cut vertices in the respective blocks proceeding block merging. We have that $\sum_{i=1}^{t} r_i = O(c)$. Let $k_1, k_2, \ldots, k_t$ be the size of the respective blocks. We have that $\sum_{i=1}^{t} k_i = \mathrm{O}(n + bc)$. Thus, the total time for block merging is bounded by $\sum_{i=1}^{t}(c \times r_i) + \sum_{i=1}^{t}(c \times r_i) + \sum_{i=1}^{t}(r_i \times (c + r_i)) + \sum_{i=1}^{t} k_i = \mathrm{O}(c^2 + bc + n)$.

Therefore, the time complexity for computing $\lambda(G[u])$ and constructing $\mathscr{T}(G[u], ns(G))$ is $\mathrm{O}(c^2 + bc + n)$. □

Fig. 10 gives an example to show that the time complexity is tight. There are $\mathrm{O}(t)$ cut vertices and $\mathrm{O}(t)$ blocks in the graph and the time complexity of the algorithm for this instance is $\mathrm{O}(t^2 + n)$.

7. Conclusion

In this paper, we generalize the avenue concept to block graphs whereby we design a polynomial-time algorithm to solve the node-searching problem on block graphs. It answers the question proposed by Peng [23] of whether the node-searching problem on block graphs can be solved in polynomial time.

References

[1] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic and Discrete Methods 8 (2) (1987) 277–284.

[2] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), in: Reliability of Computer and Communication Networks (New Brunswick, NJ, 1989), American Mathamatical Society, Providence, RI, 1991, pp. 33–49.

[3] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (2) (1991) 239–245.

[4] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (2) (1996) 358–402.

[5] H.L. Bodlaender, T. Kloks, D. Kratsch, Treewidth and pathwidth of permutation graphs, SIAM J. Discrete Math. 8 (4) (1995) 606–616.

[6] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (2) (1993) 181–188.

[7] G.J. Chang, F.K. Hwang, Y.C. Yao, Localizing combinatorial properties for partitions on block graphs, J. Combin. Optim. 2 (4) (1999) 429–441.

[8] J.A. Ellis, I.H. Sudborough, J.S. Turner, The vertex separation and search number of a graph, Inform. and Comput. 113 (1) (1994) 50–79.

[9] J. Gustedt, On the pathwidth of chordal graphs, Discrete Appl. Math. 45 (3) (1993) 233–248.

[10] F. Harary, A characterization of block graphs, Canad. Math. Bull. 6 (1) (1963) 1–6.

[11] F. Harary, G. Prins, The block-cutpoint-tree of a graph, Publ. Math. Debrecen 13 (1966) 103–107.

[12] E. Howorka, On metric properties of certain clique graphs, J. Combin. Theory Ser. B 27 (1) (1979) 67–74.

[13] D.C. Kay, G. Chartrand, A characterization of certain ptolemaic graphs, Canad. J. Math. 17 (1965) 342–346.

[14] N.G. Kinnersley, The vertex separation number of a graph equals its path-width, Inform. Process. Lett. 42 (6) (1992) 345–350.

[15] L.M. Kirousis, C.H. Papadimitriou, Interval graphs and searching, Discrete Math. 55 (2) (1985) 181–184.

[16] L.M. Kirousis, C.H. Papadimitriou, Searching and pebbling, Theoret. Comput. Sci. 47 (2) (1986) 205–218.

[17] T. Kloks, Treewidth—Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer, Berlin, 1994.

[18] T. Kloks, H. Bodlaender, H. Muller, D. Kratsch, Computing treewidth and minimum fill-in: all you need are the minimal separators, Lecture Notes in Comput. Sci. 726 (1993) 260–271.

[19] A. Kornai, Z. Tuza, Narrowness, pathwidth, and their application in natural language processing, Discrete Appl. Math. 36 (1992) 87–92.

[20] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, C.H. Papadimitriou, The complexity of searching a graph, J. Assoc. Comput. Mach. 35 (1) (1988) 18–44.

[21] R.H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: Computational Graph Theory, Springer, Berlin, 1990, pp. 17–51.

[22] B. Monien, I.H. Sudborough, Min cut is NP-complete for edge weighted trees, Theoret. Comput. Sci. 58 (1–3) (1988) 209–229.

[23] S.L. Peng, A study of graph searching on special graphs, Ph.D. Thesis, National Tsing Hua University, Taiwan, 1999.

[24] S.L. Peng, C.W. Ho, T.S. Hsu, M.T. Ko, C.Y. Tang, A linear-time algorithm for constructing an optimal node-search strategy of a tree, Lecture Notes in Comput. Sci. 1449 (1998) 279–288.

[25] S.L. Peng, M.T. Ko, C.W. Ho, T.S. Hsu, C.Y. Tang, Graph searching on some subclasses of chordal graphs, Algorithmica 27 (3–4) (2000) 395–426.

[26] N. Robertson, P.D. Seymour. Graph minors. I. Excluding a forest. J. Combin. Theory Ser. B (1983) pp. 39–61.

[27] K. Skodinis, Computing optimal linear layouts of trees in linear time, Lecture Notes in Comput. Sci. 1879 (2000) 423–445.

[28] D. West, Introduction to Graph Theory, second ed, Prentice-Hall, Englewood Cliffs, NJ, 2001.

[29] P.K. Wong, Optimal path cover problem on block graphs, Theoret. Comput. Sci. 225 (1–2) (1999) 163–169.