The Impact of a Bi-connected Graph Decomposition on Solving Cooperative Path-finding Problems

Pavel Surynek[*], **Petra Surynková**[**], and **Miloš Chromý**[*]

Charles University in Prague
Faculty of Mathematics and Physics

[*] *Malostranské náměstí 25, 118 00 Praha 1, Czech Republic*
[**] *Sokolovská 83, 186 75 Praha 8, Czech Republic*

pavel.surynek@mff.cuni.cz, petra.surynkova@mff.cuni.cz, miloschromy@gmail.com

Abstract. This paper proposes a framework for analyzing algorithms for inductive processing of bi-connected graphs. The *BIBOX* algorithm for solving cooperative path-finding problems over bi-connected graphs is submitted for the suggested analysis. The algorithm proceeds according to a decomposition of a given bi-connected graph into handles. After finishing a handle, the handle is ruled out of consideration and the processing task is reduced to a task of the same type on a smaller graph. The handle decomposition for which the *BIBOX* algorithm performs best is theoretically identified. The conducted experimental evaluation confirms that the suggested theoretical analysis well corresponds to the real situation.

Keywords: BIBOX algorithm, bi-connected graphs, cooperative path-finding, complexity

1. Introduction and Motivation

Many graph-processing algorithms proceed inductively. That is, after processing a part of the graph, the processing task is reduced to a task of the same type on a smaller graph where the finished part is no longer considered. Classical graph algorithms for *finding minimum spanning trees* (*Borůvka's* and *Jarník's* algorithms [1, 3, 5, 6, 7]) and *single source shortest paths* (*Dijkstra's* algorithm [2]) proceed exactly in this manner. In the former case, the induction step typically consumes an edge, while in the latter case, the induction step implies the processing of a vertex.

We would like to focus on another problem that takes place on a graph, namely the *cooperative path-finding problem* (CPF) [1, 8, 9, 14] where the task is to relocate certain objects (*agents*) along the edges of a given graph in a non-colliding way in order to reach the given goal vertices. An algorithm called *BIBOX* for solving CPF on *bi-connected* graphs was introduced in [10, 11]. It is another representative of an induction-based algorithm. The given bi-connected graph is processed inductively according to its decomposition into handles (a path connected by its two ends to the rest of the graph) by the algorithm. Here, the induction step is represented by handle processing. However, the analysis given in [11] does not address how the handle decomposition affects the overall performance of the algorithm. The objective of this paper is to fill this gap and determine the impact of the choice of a particular handle decomposition on the performance of the *BIBOX* algorithm.

Submitted to *Fundamenta Informaticae* on March 26, 2014. Decision received on May 12, 2014. Revision submitted on May 30, 2014.

A general framework for analyzing algorithms processing bi-connected graphs inductively is proposed. The *BIBOX* algorithm is subjected to the analysis with the view of finding handle decompositions on which the algorithm performs best. An experimental evaluation is conducted to verify the theoretical findings.

2. Background

Bi-connected graphs arise in many real-life scenarios dealing with navigation networks as they capture the important property that at least two alternative (disjoint) routes connect any two vertices. In other words, any two vertices lie on a cycle. This property can be well utilized in CPF, as it allows for relocations around cycles (rotations).

Definition 1 ***(bi-connected graph).*** An undirected graph $G = (V, E)$ is *bi-connected* if $|V| \geq 3$ and the graph after deletion of any vertex is connected; that is, the graph $G' = (V', E')$, where $V' = V \setminus \{v\}$ and $E' = \{\{u, w\} | u, w \in V \wedge u \neq v \wedge w \neq v\}$, is connected for every $v \in V$. □

Observe that a cycle itself is a bi-connected graph according to the aforementioned definition. The well known property that is utilized by algorithms for processing bi-connected graphs is that any bi-connected graph can be inductively build by adding handles [12, 13]. The operation of adding the handle $L = [u, w_1, w_2, \ldots, w_l, v]$ with $l \in \mathbb{N}_0$, which is a sequence of vertices, to the graph $G = (V, E)$ so that $\{u, w_1, w_2, \ldots, w_l, v\} \cap V = \{u, v\}$ (that is, the handle consists of already present vertices u, v and fresh vertices w_j for $j = 1,2, \ldots, l$) results in a new graph $G' = (V', E')$, where $V' = V \cup \{w_1, w_2, \ldots, w_l\}$ and $E' = E \cup \{\{u, w_1\}, \{w_2, w_3\}, \ldots, \{w_{l-1}, w_l\}, \{w_l, v\}\}$. The *size of the handle* L is defined as l, that is the number of internal vertices. The addition of a handle of size 0 corresponds to the addition of an edge.

Proposition 1 ***(handle decomposition)*** [12, 13]**.** Any bi-connected $G = (V, E)$ graph can be obtained from a cycle using a sequence of handle-adding operations. ■

The sequence of handles together with the initial cycle, from which the given bi-connected graph can be constructed, will be called a *handle decomposition*. The handle decomposition for a given bi-connected graph $G = (V, E)$ will be denoted as $C_0, L_1, L_2, \ldots, L_k$, where C_0 is a cycle (formally a sequence of vertices) and L_i is a handle for $i = 1,2, \ldots, k$ (for formal definition of sequences of vertices see above). G can be reconstructed from $C_0, L_1, L_2, \ldots, L_k$ as follows: first cycle C_0 is laid; then the handles $L_1, L_2, \ldots, L_k$ are added inductively. Consider that we have an intermediate graph G_i (as a result of adding the handles $L_1, L_2, \ldots, L_i$), the next intermediate graph G_{i+1} is obtained by adding L_{i+1} to G_i using the handle-adding operation. The final intermediate graph G_k is G. Notice that $V = C_0 \cup L_1 \cup L_2 \cup \ldots \cup L_k$ and the intermediate graph at any step of the process of reconstruction from handles is bi-connected.

3. Theoretical Analysis of Induction-based Algorithms and a Special Case

Assume that the algorithm processes handle decomposition handles one by one starting with the last one and moving towards the initial cycle. After the processing of a handle is finished, the algorithm continues inductively on a smaller bi-connected graph without the finished handle. Let $t(n,m)$ be the upper bound of the time consumed by the algorithm when processing a handle of size n in a bi-connected graph of size $n+m$.

Particularly in the case of the *BIBOX* algorithm that solves the cooperative path-finding problem it holds that $(n,m) = d_1 \cdot n^3 + d_2 \cdot n^2 m + d_3 \cdot nm^2$, where $d_1, d_2, d_3 \in \mathbb{R}$.

The upper bound of the time needed to process the given bi-connected graph $G = (V,E)$ according to the handle decomposition $V = C_0 \cup L_1 \cup L_2 \cup \ldots \cup L_k$ can thus be calculated as follows:

$$\sum_{i=1}^{k} t\left(|L_i|, |C_0| + \sum\nolimits_{j=1}^{i-1} |L_j|\right) \tag{1}$$

Obviously, the expression depends on the number of handles in the decomposition as well as on the sizes of the individual handles. Several scenarios with different distributions of handle sizes will be evaluated.

3.1. *Varying the Number and the Size of Handles*

The number of handles may vary in multiple different ways. In practice, however, cases with sizes of the handle L_i that can be expressed as $\mathcal{O}(1)$, $\mathcal{O}(i)$, $\mathcal{O}(i^2)$, $\mathcal{O}(\sqrt{i})$, or $\mathcal{O}(2^i)$ can be expected as a result of constructing a handle decomposition satisfying certain constraints. It is also natural to expect that the size of handles is a growing function with respect to their position in the handle decomposition (the larger part of the graph allows a larger handle to be present) with which the suggested handle sizes comply.

Assume that the size of the handle depends linearly on its position within the handle decomposition, that is, $|L_i| = \alpha i$; $\alpha \in \mathbb{R}$. For the sake of simplicity, the calculation will be done asymptotically where sums can be replaced with integrals. Also, the initial cycle is omitted in the analysis because it only adds a constant. Thus, asymptotically, the total number of handles is:

$$k = \sqrt{\frac{2|V|}{\alpha}} \tag{2}$$

which comes from the equation:

$$|V| = \int \alpha i \, di = \alpha \frac{i^2}{2} \tag{3}$$

The estimation of the time consumed by the algorithm over such a handle decomposition is:

$$\int t\left(\alpha i, \alpha \frac{i^2}{2}\right) di \tag{4}$$

In particular, for the *BIBOX* algorithm, the following estimation of time can be obtained as a simple integration over all the handles as follows:

$$\begin{aligned} &\int d_1 \cdot (\alpha i)^3 + d_2 \cdot (\alpha i)^2 \left(\alpha \frac{i^2}{2}\right) + d_3 \cdot (\alpha i) \left(\alpha \frac{i^2}{2}\right)^2 di = \\ &= \alpha^3 \left(\frac{1}{4} d_1 \cdot i^4 + \frac{1}{10} d_2 \cdot i^5 + \frac{1}{24} d_3 \cdot i^6\right) \end{aligned} \tag{5}$$

If the total number of handles is substituted into (5), the following running time of the *BIBOX* algorithm is obtained:

$$d_1 \alpha \cdot |V|^2 + \frac{2}{5} d_2 \sqrt{\alpha} \cdot |V|^2 \sqrt{|V|} + \frac{1}{3} d_3 \cdot |V|^3 \tag{6}$$

The asymptomatic running time is $\mathcal{O}(|V|^3)$, where the most important addend is $\frac{1}{3} d_3 \cdot |V|^3$. The sum of the remaining addends in the expression will be called a *residuum*. In the case of the *BIBOX* algorithm with linear sizes of handles, the residuum is as follows:

$$d_1 \alpha \cdot |V|^2 + \frac{2}{5} d_2 \sqrt{\alpha} \cdot |V|^2 \sqrt{|V|} \in \mathcal{O}\left(|V|^{2+\frac{1}{2}}\right) \tag{7}$$

Naturally, the smallest possible residuum is desirable for having the shortest possible running time of the algorithm. Fortunately, the residuum can be affected by the handle decomposition – particularly by handle sizes shown in the above calculation.

Proposition 1 ***(residuum in various handle decompositions).*** For standard cases of handle decompositions, where the size of the handle L_i is $\mathcal{O}(1)$, $\mathcal{O}(\sqrt{i})$, $\mathcal{O}(i^2)$, or $\mathcal{O}(2^i)$, respectively, the following estimates of the residuum are obtained:

- if $|L_i| = \beta; \beta \in \mathbb{R}$ $\Rightarrow$ $\mathcal{O}(|V|^2)$
- if $|L_i| = \zeta\sqrt{i}; \zeta \in \mathbb{R}$ $\Rightarrow$ $\mathcal{O}(|V|^{2+\frac{1}{3}})$
- if $|L_i| = \gamma i^2; \gamma \in \mathbb{R}$ $\Rightarrow$ $\mathcal{O}(|V|^{2+\frac{2}{3}})$
- if $|L_i| = \delta 2^i; \delta \in \mathbb{R}$ $\Rightarrow$ $\mathcal{O}(|V|^3)$. ■

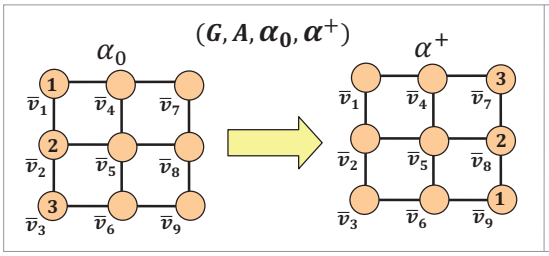
Since the calculations are analogous to the linear size of the handle, the proof is deferred to the Appendix.

According to the size of the residuum, it seems that, with respect to reducing the runtime, the best handle decomposition is that with a constant handle size while the worst one is that with an exponential handle size.

4. Experimental Evaluation in the Case of Cooperative Path-finding

It is interesting to explore whether theoretical estimations match the real situation. That is, whether a handle decomposition with handles of a constant size really is the best option for the *BIBOX* algorithm. To provide a complete picture, the problem is introduced in the following definition.

Definition 3 ***(cooperative path-finding)*** [1, 9]**.** Let $G = (V, E)$ be an undirected graph and $A = \{a_1, a_2, \ldots, a_n\}$, where $|A| < |V|$ be a set of agents. The *arrangement* of agents in the graph reflects the uniquely invertible function $\alpha: A \longrightarrow V$ (there is at most one agent in each vertex). The problem of *cooperative path-finding* (CPF) consists in finding a sequence of moves of agents so that the given initial arrangement α_0 is transformed to the given goal arrangement α^+. The move with an agent is possible along an edge. □



Solution of the cooperative path-finding problem with $A = \{1,2,3\}$

step	0	1	2	3	4	5	**6**
1	$\bar{v}_1$	$\bar{v}_4$	$\bar{v}_7$	$\bar{v}_8$	$\bar{v}_9$	$\bar{v}_9$	$\bar{v}_9$
2	$\bar{v}_2$	$\bar{v}_2$	$\bar{v}_1$	$\bar{v}_4$	$\bar{v}_7$	$\bar{v}_8$	$\bar{v}_8$
3	$\bar{v}_3$	$\bar{v}_3$	$\bar{v}_3$	$\bar{v}_2$	$\bar{v}_1$	$\bar{v}_4$	$\bar{v}_7$

Figure 1. *An example of the cooperative path-finding problem (CPF).* Three agents need to be rearranged in a 3 × 3 grid. A solution of length 6, where multiple agents move in a single step, is shown.

The *inverse arrangement* of agents is the mapping $\bar{\alpha}: V \longrightarrow A \cup \{\perp\}$ that provides information about which agent is located in a given vertex. The special value $\perp$ indicates that the given vertex is empty.

A number of variants of the problem need to be considered. In this case, a variant with an agent moving into an empty vertex through an edge is concerned (a variant where a sequence of agents can move at once like a train is discussed in [11]). An example instance and its solution are shown in Figure 1.

The *BIBOX* algorithm, for which the analysis has been designed, solves the problem over bi-connected graphs. It arranges agents into handles while the problem is inductively reduced on a smaller bi-connected graph whenever agents are arranged into the handle – agents in such a handle do not move any more.

Instead of measuring runtime, the number of generated moves will be evaluated. The number of generated moves corresponds exactly to the runtime. The reason for using a number of moves instead of runtime is that it is impossible to measure runtime as precisely as the number of moves, furthermore, it is less dependent on the implementation.

Here, the algorithm is recalled using a pseudo-code as Algorithm 1. It employs several auxiliary functions to solve subtasks. The pseudo-code of auxiliary functions is given in [11] – at this point, they are only briefly described.

Algorithm 1. *The **BIBOX** algorithm.* The algorithm solves the cooperative path-finding problem (CPF) over bi-connected graphs consisting of a cycle and at least one handle with two unoccupied vertices. The algorithm proceeds inductively according to a handle decomposition. The two unoccupied vertices are necessary for arranging agents within the initial cycle; in the rest of the graph only one unoccupied vertex is needed. A pseudo-code is built around several higher-level operations:

- Lock(U) — locks all the vertices from set U; each vertex is either *locked* or *unlocked*; an agent must not be moved out of the locked vertex, which is respected by other operations
- Unlock(U) — unlocks all the vertices from set U
- Make-Unoccupied(v) — vacates vertex v
- Move-Agent(a, v) — moves agent a from its current location to vertex v
- Rotate-Cycle$^{+}(C)$ — rotates cycle C in the positive direction; a vacant vertex must be present in the cycle
- Rotate-Cycle$^{-}(C)$ — rotates cycle C in the negative direction
- Transform-Goal(G, A, α^{+}) — transforms goal arrangement α^{+} to a new arrangement so that finally unoccupied vertices are located in the initial cycle of the handle decomposition; two disjoint paths along which empty vertices are relocated are returned
- Finish-Solution(φ, χ) — transforms the arrangement with two unoccupied vertices in the initial cycle to the original goal arrangement; φ and χ are two disjoint paths along which empty vertices shifted
- Solve-Original-Cycle — arranges agents within the initial cycle of the handle decomposition to comply with the transformed goal arrangement; two empty vertices are employed to arrange agents

---

procedure *BIBOX-Solve*$(G = (V, E), A, \alpha_0, \alpha^{+})$
  /* Top level function of the BIBOX algorithm; solves
  a given cooperative path-finding problem.
  Parameters: G – a graph modeling the environment,
  A – a set of agents,
  α_0 – the initial arrangement of agents,
  α^{+} – the goal arrangement of agents. */
1: **let** $\mathcal{D} = [C_0, L_1, L_2, \ldots, L_k]$ be a handle decomposition of G
2: $(\alpha^{+}, \varphi, \chi) \leftarrow$ Transform-Goal(G, A, α^{+})
3: $\alpha \leftarrow \alpha_0$
4: **for** $c = k, k-1, \ldots, 1$ **do**
5: ⎸ **if** $|L_c| > 2$ **then**
6: ⎸ ⎸ Solve-Regular-Handle(c)
7: Solve-Original-Cycle
8: Finish-Solution(φ, χ)

---

procedure *Solve-Regular-Handle*(c)
    /* Places agents the destinations of which are within
    The handle L_c; agents placed in the handle L_c are finally
    locked to prevent them from moving.
    Parameters: c – index of a handle */
let $[u, w_1, w_2, \dots, w_l, v] = L_c$
    /* Both unoccupied vertices must be located
    outside the currently solved handle. */
let $x, z \in V \setminus \bigcup_{c=j}^{k}(L_j \setminus \{u, v\})$ such that $x \neq z$
Make-Unoccupied(x)
Lock$(\{x\})$
Make-Unoccupied(z)
Unlock$(\{x\})$
for $i = l, l-1, \dots, 1$ **do**
    Lock$(L_c \setminus \{u, v\})$
    /* The agent to be placed is outside the handle L_c. */
    if $\alpha(\bar{\alpha}^+(w_i)) \notin (L_c \setminus \{u, v\})$ **then**
        Move-Agent$(\bar{\alpha}^+(w_i), u)$
        Lock$(\{u\})$
        Make-Unoccupied(v)
        Unlock(L_c)
        Rotate-Cycle$^+(C(L_c))$
    /* The agent to be placed is inside the handle L_c. */
    else
        Make-Unoccupied(u)
        Unlock(L_c)
        $\rho \leftarrow 0$
        while $\alpha(\bar{\alpha}^+(w_i)) \neq v$ **do**
            Rotate-Cycle$^+(C(L_c))$
            $\rho \leftarrow \rho + 1$
        Lock$(L_c \setminus \{u, v\})$
        let $y \in V \setminus (\bigcup_{j=c+1}^{d}(L_j \setminus \{u, v\}) \cup C(L_j))$
        Move-Agent$(\bar{\alpha}^+(w_i), y)$
        Lock $(\{y\})$
        Make-Unoccupied(u)
        Unlock(L_c)
        while $\rho > 0$ **do**
            Rotate-Cycle$^-(C(L_c))$
            $\rho \leftarrow \rho - 1$
        Unlock$(\{y\})$
        Lock$(L_c \setminus \{u, v\})$
        Move-Agent$(\bar{\alpha}^+(w_i), u)$
        Lock $(\{u\})$
        Make-Unoccupied(v)
        Unlock(L_c)
        Rotate-Cycle$^+(C(L_c))$
Lock$(L_c \setminus \{u, v\})$

The algorithm starts with the construction of a handle decomposition (line 1). This step is non-deterministic in the original algorithm and can be replaced with a handle decomposition where sizes of handles satisfy certain conditions. It is assumed that a cycle denoted as $C(L_i)$ is associated with each handle; $C(L_i)$ can be constructed by adding a path connecting the handle's connection vertices u and v. Thereafter, the goal arrangement of agents is transformed so that the vacant vertices are eventually located in the initial cycle of the decomposition (line 2). In fact, the algorithm solves this modified instance. The original instance is solved by relocating vacant vertices from the initial cycle to their original goal locations (line 8). This instance transformation is carried out using the auxiliary functions *Transform-Goal*, and *Finish-Solution* that relocate vacant vertices along two vertex disjoint paths. The main loop (lines 4-6) processes a handle from the last one towards the initial cycle. Agents are arranged by means of another auxiliary procedure, *Solve-Original-Cycle*, in the original cycle (line 7).

Individual handles are processed by the *Solve-Regular-Handle* procedure. It arranges agents into a handle in a stack-like manner. First, unoccupied vertices are moved out of the processed handle as they will be needed elsewhere (lines 10-14). Subsequently, agents, whose goal positions are in the handle, are processed. Two cases are distinguished depending on whether the processed agent is located outside the handle (lines 17-22) or within the handle (lines 23-45). The case is with the agent outside is easier to solve – in this case, the agent is moved to the connection vertex u using the *Move-Agent* auxiliary procedure. The other connection vertex v is vacated by the *Make-Unoccupied* procedure. If some vertex is free in the cycle $C(L_c)$ then the cycle can be rotated. This is performed once in the positive direction using the *Rotate-Cycle*$^+$ function. The rotation places the agent into the handle. Throughout the agent relocation process, vertex locking is used (functions *Lock* and *Unlock*) to fix the agent in a certain vertex while other agents or the vacant vertex are relocated.

A more difficult situation occurs when the agent is placed inside the handle. In such a case, the agent must be rotated out of the handle to the rest of the graph (lines 24-29). The number of positive rotations to get the agent out of the handle is counted (lines 27-29). The counted number of rotations is used to restore the situation with the corresponding number of negative rotations (lines 36 – 38). At this point, the situation is the same as in the previous case. Thus, the agent is stacked into the handle in the same way.

Consider that n agents need to be arranged into a handle of size n, which is connected to a bi-connected graph of size m. Processing a single agent requires rotating the handle no more than $2n$+1 times – at most n rotations are needed to get the agent out of the handle if it is originally located inside; at most n rotations are needed to rotate the handle back; and one rotation is required to push the agent inside the handle. Each rotation requires $n + m$ steps as, in the worst case, the cycle that is rotated can include the whole graph. The time required for the rotations can be thus estimated by $d \cdot (2n + 1) \cdot (n + m)$ with some $d \in \mathbb{R}$. Hence, the time needed for the rotations when processing a single handle can be estimated by $d_1 \cdot n^3 + d_2 \cdot n^2 m$ $d_1, d_2 \in \mathbb{R}$. To finish the estimation, it is needed to account for the time needed to relocate the agent towards the handle's connec-

tion vertex from inside the graph or between the handle's connection vertices. Both cases can be estimated using $d_3 \cdot m^2$ since the agent needs to be moved along a path of a length of at most m, where traversing a single edge requires at most m steps (a destination vertex must be vacated each time the edge is traversed). Altogether, the expression $t(n,m) = d_1 \cdot n^3 + d_2 \cdot n^2 m + d_3 \cdot n m^2$, where $d_1, d_2, d_3 \in \mathbb{R}$ estimates the time needed for processing the handle. The argumentation is based on the fact that performing a move with an agent corresponds to a constant time.

4.1. *Measurement of the Number of Moves*

An experimental evaluation has been made to check if the theoretical analysis matches the real-life situation. The number of moves generated by the *BIBOX* algorithm with different sizes of handles was measured. All the theoretically studied cases of the size of the handle L_i $\mathcal{O}(1)$, $\mathcal{O}(i)$, $\mathcal{O}(\sqrt{i})$, $\mathcal{O}(i^2)$ and $\mathcal{O}(2^i)$ were tested. The multiplication factor of 1 was used in all the cases to generate a sequence of handles of various sizes; that is, $\alpha, \beta, \gamma, \delta, \zeta = 1$. Ten instances of the CPF problem were generated for each size of the constructed bi-connected graph. Random initial and goal arrangements with exactly two unoccupied vertices were generated in each instance. The results are shown in Figure 2.

The experimental evaluation clearly matches the derived theoretical results; that is, constant size handles produce the lowest number of moves, while handles of the exponential size produce the most moves. The relative ordering of the number of moves in other handle sizes in the experimental evaluation is the same as in the theoretical analysis. It seems that differences in the number of moves in the experimental evaluation are even more pronounced than in the theoretical analysis. The explanation is that a graph with smaller handles exhibits higher connectivity that implies shorter paths along which agents are relocated.

5. Conclusions and Future Work

A simple but general framework for evaluating induction-based algorithms for processing bi-connected graphs has been introduced. It is assumed that algorithms process bi-connected graphs according to their decomposition into handles. The last handle is processed first, which reduces the processing task on a smaller bi-connected graph – the graph from which the last handle has been removed. The impact of the handle decomposition on the performance of the algorithm was studied.

A theoretical and experimental evaluation of a particular case of an algorithm solving the cooperative path-finding (CPF) problem over bi-connected graphs was performed. The theoretical evaluation showed that among several suggested handle decompositions the decomposition with the constant size of handles is the best option with respect to the number of moves solving the given CPF instance. The performed experimental evaluation confirmed the theoretical assumptions.

The described framework identifies the best decomposition among several known decompositions. It is worth further exploration whether a general mechanism for determining the best handle decomposition can be found.

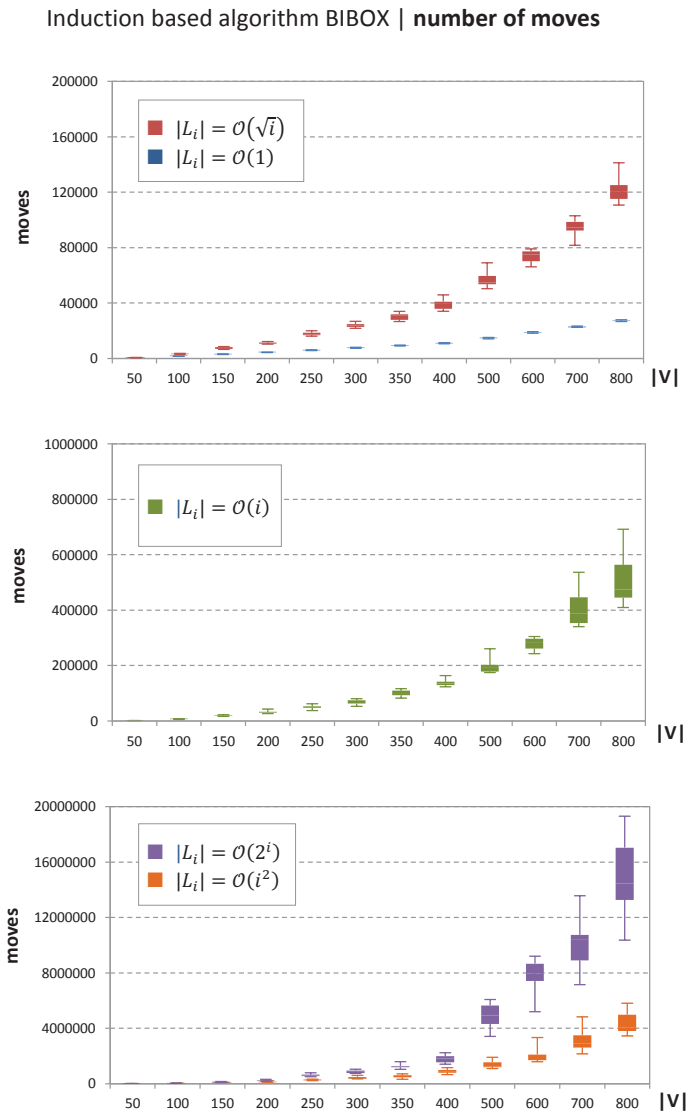


Figure 2. *Number of moves generated by the BIBOX algorithm.* The algorithm was tested on bi-connected graphs with the size of the i-th handle corresponding to the *constant*, $\sqrt{i}$, i, i^2, and 2^i respectively. The lowest number of moves is obtained for the constant handle size, and the highest number for handles, the size of which is exponential.

Acknowledgments

This work is supported by the Czech Science Foundation under the contract number GAP103/10/1287 and by Charles University in Prague within the PRVOUK project (section P46).

References

1. **Cheriton**, D., **Tarjan**, R. E.: *Finding Minimum Spanning Trees.* SIAM Journal on Computing, Volume 5, pp. 724–741, Society for Industrial and Applied Mathematics, 1976.
2. **Dijkstra**, E. W.: *A Note on Two Problems in Connection with Graphs.* Numerische Mathematik, Volume 1, pp. 269–271, Springer Verlag, 1959.
3. **Jarník**, V.: *O jistém problému minimálním (About a Certain Minimal Problem)*. Práce Moravské Přírodovědecké Společnosti, Volume 6, pp. 57–63, Moravská Přírodovědecká Společnost, 1930.
4. **Kornhauser**, D., **Miller**, G. L., and **Spirakis**, P. G.: *Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications.* Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241-250, IEEE Press, 1984.
5. **Kruskal**, J. B.: *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.* Proceedings of the American Mathematical Society, Volume 7, pp. 48–50, American Mathematical Society, 1956.
6. **Nešetřil**, J., **Milková**, E., **Nešetřilová**, H. *Otakar Borůvka on Minimum Spanning Tree Problem: Translation of both the 1926 Papers, Comments, History*. Discrete Mathematics, Volume 233 (1–3), pp. 3–36, Elsevier, 2001.
7. **Prim**, R. C.: *Shortest Connection Networks and Some Generalizations.* Bell System Technical Journal, Volume 36, pp. 1389–1401, American Telephone and Telegraph Company, 1957.
8. **Ryan**, M. R. K.: *Exploiting Subgraph Structure in Multi-Robot Path Planning*, Journal of Artificial Intelligence Research (JAIR), Volume 31, pp. 497-542, AAA Press, 2008.
9. **Silver**, D.: *Cooperative Pathfinding.* Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117-122, AAAI Press.
10. **Surynek**, P.: *A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs.* Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), pp. 3613-3619, IEEE Press, 2009.
11. **Surynek**, P.: *Solving Abstract Cooperative Path-Finding in Densely Populated Environments.* Computational Intelligence (COIN), Volume 30, Issue 2, pp. 402-450, Wiley, 2014.
12. **Tarjan**, R. E.: *Depth-First Search and Linear Graph Algorithms.* SIAM Journal on Computing, Volume 1 (2), pp. 146-160, Society for Industrial and Applied Mathematics, 1972.
13. **Westbrook**, J., **Tarjan**, R. E.: *Maintaining Bridge-connected and Biconnected Components On-line.* Algorithmica, Volume 7, Number 5&6, pp. 433–464, Springer Verlag, 1992.
14. **de Wilde**, B., **ter Mors**, A., **Witteveen**, C.: *Push and Rotate: Cooperative Multi-agent Path Planning.* Proceedings of International conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), IFAAMAS, 2013, pp. 87-94.

Appendix

Proof of **Proposition 1:** Assume that $|L_i| = \beta$; $\beta \in \mathbb{N}$; that is $|L_i| = \mathcal{O}(1)$. The number of handles, which will be used as a bound in the integration, is:

$$k = \frac{|V|}{\beta}$$

which can be obtained from the equation:

$$|V| = \int \beta \, di = \beta i$$

The number of moves generated by the algorithm in such a case can be obtained from the following integration:

$$\int d_1 \cdot \beta^3 + d_2 \cdot \beta^2(\beta i) + d_3 \cdot \beta(\beta i)^2 di = \\ = \beta^3 \left(d_1 \cdot i + \frac{1}{2} d_2 \cdot i^2 + \frac{1}{3} d_3 \cdot i^3 \right)$$

After substituting k into the expression, the following number of moves is obtained:

$$d_1 \beta^2 \cdot |V| + \frac{1}{2} d_2 \beta \cdot |V|^2 + \frac{1}{3} d_3 \cdot |V|^3$$

Hence, the residuum is:

$$d_1 \beta^2 \cdot |V| + \frac{1}{2} d_2 \beta \cdot |V|^2 \in \mathcal{O}(|V|^2)$$

Assume that $|L_i| = \gamma i^2$; $\gamma \in \mathbb{N}$; that is $|L_i| = \mathcal{O}(i^2)$. The number of handles, which will be used as a bound in the integration, is:

$$k = \sqrt[3]{\frac{3|V|}{\gamma}}$$

which can be obtained from the equation:

$$|V| = \int \gamma i^2 \, di = \frac{1}{3} \gamma i^3$$

In such a case, the number of moves generated by the algorithm can be obtained from the following integration:

$$\int d_1 \cdot (\gamma i^2)^3 + d_2 \cdot (\gamma i^2)^2 \left(\frac{1}{3} \gamma i^3 \right) + d_3 \cdot \gamma i^2 \left(\frac{1}{3} \gamma i^3 \right)^2 di = \\ = \gamma^3 \left(\frac{1}{7} d_1 \cdot i^7 + \frac{1}{24} d_2 \cdot i^8 + \frac{1}{81} d_3 \cdot i^8 \right)$$

After substituting k into the expression, the following number of moves is obtained:

$$\frac{9\sqrt[3]{3}}{7}d_1\left(\sqrt[3]{\gamma}\right)^2\cdot|V|^2\sqrt[3]{|V|}+\frac{3(\sqrt[3]{3})^2}{8}d_2\sqrt[3]{\gamma}\cdot|V|^2(\sqrt[3]{|V|})^2+\frac{1}{3}d_3\cdot|V|^3$$

Hence, the residuum is:

$$\frac{9\sqrt[3]{3}}{7}d_1\left(\sqrt[3]{\gamma}\right)^2\cdot|V|^2\sqrt[3]{|V|}+\frac{3(\sqrt[3]{3})^2}{8}d_2\sqrt[3]{\gamma}\cdot|V|^2(\sqrt[3]{|V|})^2\in\mathcal{O}(|V|^{2+\frac{2}{3}})$$

Assume that $|L_i|=\delta 2^i$; $\delta\in\mathbb{R}$; that is, $|L_i|=\mathcal{O}(2^i)$. The number of handles, which will be used as a bound in the integration, is:

$$k=\log_2|V|-\log_2\left(\frac{\delta}{\ln 2}\right)$$

which can be obtained from the equation:

$$|V|=\int\delta 2^i\,di=\frac{\delta}{\ln 2}2^i$$

The number of moves generated by the algorithm can be obtained from the following integration:

$$\int\left(d_1+\frac{d_2}{\ln 2}+\frac{d_3}{\ln^2 2}\right)\cdot(\delta 2^i)^3di=$$
$$=\frac{1}{3}\left(\frac{d_1}{\ln 2}+\frac{d_2}{\ln^2 2}+\frac{d_3}{\ln^3 2}\right)\delta^3\cdot 2^{3i}$$

After substituting k into the expression, the following number of moves is obtained:

$$\frac{1}{3}(d_1\ln^2 2+d_2\ln 2+d_3)\cdot|V|^3$$

Hence, the residuum is:

$$\frac{1}{3}(d_1\ln^2 2+d_2\ln 2)\cdot|V|^3\in\mathcal{O}(|V|^3)$$

Assume that $|L_i|=\zeta\sqrt{i}$; $\zeta\in\mathbb{R}$; that is, $|L_i|=\mathcal{O}(\sqrt{i})$. The number of handles, which will be used as a bound in the integration, is:

$$k=\sqrt[3]{\frac{9}{4}\left(\frac{|V|}{\zeta}\right)^2}$$

which can be obtained from the equation:

$$|V|=\int\zeta\sqrt{i}di=\zeta\frac{2}{3}i^{\frac{3}{2}}$$

The number of moves generated by the algorithm can be obtained from the following integration:

$$\int d_1 \cdot (\zeta\sqrt{i})^3 + d_2 \cdot (\zeta\sqrt{i})^2 \left(\zeta \frac{2}{3} i^{\frac{3}{2}}\right) + d_3 \cdot (\zeta\sqrt{i}) \left(\zeta \frac{2}{3} i^{\frac{3}{2}}\right)^2 di = \\ = \zeta^3 \left(\frac{2}{5} d_1 \cdot i^{\frac{5}{2}} + \frac{4}{21} d_2 \cdot i^{\frac{7}{2}} + \frac{1}{9} d_3 \cdot i^{\frac{9}{2}}\right)$$

After substituting k into the expression, the following number of moves is obtained:

$$\left(\frac{3}{2}\right)^{\frac{5}{3}} d_1 \zeta^{\frac{4}{3}} \cdot |V|^{\frac{5}{3}} + \frac{4}{21} \left(\frac{3}{2}\right)^{\frac{7}{3}} d_2 \zeta^{\frac{17}{6}} \cdot |V|^{\frac{7}{3}} + \frac{1}{3} d_3 \cdot |V|^3$$

Hence, the residuum is:

$$\left(\frac{3}{2}\right)^{\frac{5}{3}} d_1 \zeta^{\frac{4}{3}} \cdot |V|^{\frac{5}{3}} + \frac{4}{21} \left(\frac{3}{2}\right)^{\frac{7}{3}} d_2 \zeta^{\frac{17}{6}} \cdot |V|^{\frac{7}{3}} \in \mathcal{O}(|V|^{2+\frac{1}{3}})$$

■