Path Planning for Multiple Robots in Bi-connected Environments

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Department of Theoretical Computer Science and Mathematical Logic
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
pavel.surynek@mff.cuni.cz

Abstract

This paper addresses a problem of path planning for multiple robots. A class of the problem with bi-connected environments is defined and a new polynomial-time solving algorithm for this class is proposed. It is shown in the paper that the new algorithm significantly outperforms the existing state-of-the-art domain-independent approaches and it is able to solve problems of real-life size.

Introduction and Motivation

The problem of *path planning for multiple robots* ranks among the most challenging problems of artificial intelligence [5], [7], [10]. We need to plan a sequence of moves for each robot of a group of robots that can move in a certain environment and that need to reach certain positions. The major complication is that robots must not collide with each other that are especially prohibitive conditions in the environments with a limited free space.

This formal problem is motivated by many real-life tasks ranging from rearranging containers in storage yards to coordination of movements of a large group of automated agents [7], [8].

We would like to introduce a new solving algorithm called *BIBOX* for a class of multi-robot path planning problems with bi-connected environments and at least two free places in this paper. We experimentally show that our new algorithm outperforms existing state-of-the-art methods and it is suitable for problems of real-life size.

Problem of Path Planning for Multiple Robots

Consider a group of robots in a certain environment that need to move from their initial positions to the given goal positions. The robots are required to avoid collisions during their movements. Thus, the task is to find spatial-temporal paths from the initial to the goal position for each robot such that these paths do not intersect at the same time point.

We abstract from the individual properties of robots. Hence we assume that all the robots are the same. Furthermore, we abstract from the physical properties of the robots and the environment. The only important property is the topology of the environment in our abstraction. Hence the environment is modeled as an undirected graph where the robots are placed in the vertices of this graph.

The dynamicity of the model is defined by the notion of an allowed move for the robot. A robot can move from a vertex to a target neighboring vertex if there is no robot in the target vertex and no other robot is simultaneously entering the target vertex. The problem of *path planning for multiple robots* is formally described in the following definition.

Definition 1 *(path planning for multiple robots).* Let us have an undirected graph $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ that models the environment. Next, let us have a set of robots $R = \{r_1, r_2, \ldots, r_\mu\}$ where $\mu < n$. The initial positions of the robots are defined by a function $S_0 : R \rightarrow V$ where $S_0(r_i) \neq S_0(r_j)$ for $i, j = 1,2,\ldots,\mu$ such that $i \neq j$. The goal positions of the robots are defined by a function $S^+ : R \rightarrow V$ where $S^+(r_i) \neq S^+(r_j)$ for $i, j = 1,2,\ldots,\mu$ such that $i \neq j$. The problem of *path-planning for multiple robots* is a task to find a number m and a path $P_r = [p_1^r, p_2^r, \ldots, p_m^r]$ for every robot $r \in R$ where $p_i^r \in V$ for $i = 1,2,\ldots,m$, $p_1^r = S_0(r)$, $p_m^r = S^+(r)$, and either $\{p_i^r, p_{i+1}^r\} \in E$ or $p_i^r = p_{i+1}^r$ for $i = 1,2,\ldots,m-1$. Furthermore, paths $P_r = [p_1^r, p_2^r, \ldots, p_m^r]$ and $P_q = [p_1^q, p_2^q, \ldots, p_m^q]$ for every two robots $r \in R$ and $q \in R$ such that $r \neq q$ must satisfy that $p_{i+1}^r \neq p_i^q$ for $i = 1,2,\ldots,m-1$ (the target vertex is unoccupied) and $p_i^r \neq p_i^q$ for $i = 1,2,\ldots,m$ (no other robot is simultaneously entering the target vertex). □

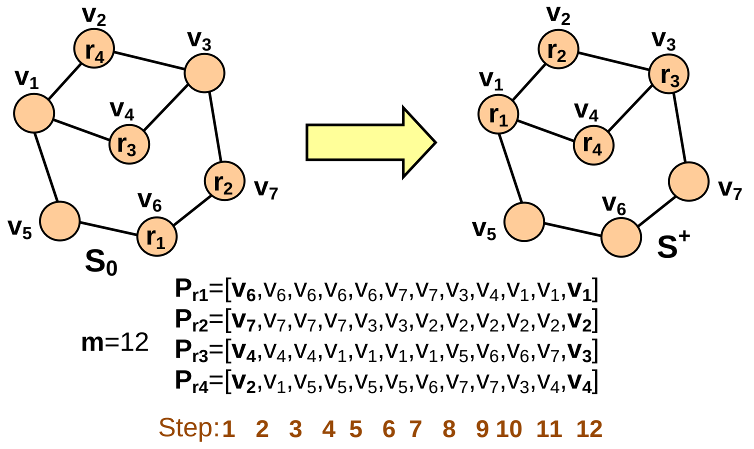
v2
v3
v1
v4
r4
r3
r2
v7
v6
v5
r1
S0
v2
v3
v1
v4
r2
r3
r1
r4
v7
v6
v5
S+
m=12
$P_{r1}=[\mathbf{v_6},v_6,v_6,v_6,v_6,v_7,v_7,v_3,v_4,v_1,v_1,\mathbf{v_1}]$
$P_{r2}=[\mathbf{v_7},v_7,v_7,v_7,v_3,v_3,v_2,v_2,v_2,v_2,v_2,\mathbf{v_2}]$
$P_{r3}=[\mathbf{v_4},v_4,v_4,v_1,v_1,v_1,v_1,v_5,v_6,v_6,v_7,\mathbf{v_3}]$
$P_{r4}=[\mathbf{v_2},v_1,v_5,v_5,v_5,v_5,v_6,v_7,v_7,v_3,v_4,\mathbf{v_4}]$
Step: 1 2 3 4 5 6 7 8 9 10 11 12

Figure 1. *A problem of path planning for multiple robots.* The task is to move robots from their initial positions denoted as S_0 to the goal positions denoted as S^+. A solution of length 12 is shown. Notice the parallelism at steps 7 and 8.

The problem of path planning for multiple robots is illustrated in figure 1. Notice that path for an individual robot may contain loops and the robot may stay in a vertex for more than a single time step. Another important property of the definition is that it intrinsically allows parallel movements of the robots (more than one robot can perform a move in a single time step). It is also possible to require the number m to be smallest as possible. However, this requirement makes the problem intractable (see the section Related Works).

Solving Algorithm for Bi-connected Environments

An algorithm for a class of problems of path planning for multiple robots is described in this section. We found that the problem is always solvable if the graph G is bi-connected and $\mu \leq n-2$, that is, at least two vertices in the graph G are unoccupied. Moreover, our solving algorithm for this class of problem runs in polynomial time. We called our algorithm *BIBOX*.

Graph Theoretical Preliminaries

Let us remind some basic graph theoretical notions [10] before we start with the description of the algorithm.

Definition 2 (graph connectivity). An undirected graph $G=(V,E)$ is *connected* if $|V| \geq 2$ and for every pair of vertices $u \in V$ and $v \in V$ such that $u \neq v$ there is a path connecting u and v consisting of edges from E. □

Definition 3 (graph bi-connectivity). An undirected graph $G=(V,E)$ is *bi-connected* if $|V| \geq 3$ and the graph $G' = (V - \{v\}, E \cap \{\{u,w\} \mid u,w \in V \wedge u \neq v \wedge w \neq v\})$ is connected for every $v \in V$. □

Bi-connected graphs have an important well known property which we exploit in our algorithm. Each bi-connected graph can be constructed from a cycle by an operation of *adding loops* to the graph [10].

Adding a loop which is a sequence of vertices $L=[u, x_1, x_2, ..., x_l, v]$ to an undirected graph $G=(V,E)$ where $u,v \in V$ and $x_i \notin V$ for $i=1,2,\ldots,l$ (x_i are new vertices) means to create a new graph $G'=(V',E')$; where $V' = V \cup \{x_1, x_2, \ldots, x_l\}$ and either $E' = E \cup \{\{u,v\}\}$ in the case when $l=0$ or $E' = E \cup \{\{u,x_1\},\{x_1,x_2\},\ldots,\{x_{l-1},x_l\}, \{x_l,v\}\}$ in the case when $l \geq 1$. As a preparation for the design of the algorithm the loop L is assigned a cycle $C(L)$ if the graph G is connected. The cycle $C(L)$ consists of vertices on a path between u and v in G followed by vertices $x_1, x_2, \ldots, x_l$.

Lemma 1 (loop decomposition) [9], [10]**.** Any bi-connected graph can be obtained from a cycle graph by the operation of adding a loop. ■

In addition to the above lemma the more holds. It is possible to decompose a given undirected graph $G=(V,E)$ to a sequence of loops in time $O(|V|+|E|)$ [9]. Moreover, the graph is bi-connected at any stage of the construction according to the decomposition.

Algorithm for Bi-connected Graphs

Let us have an instance of the multi-robot path planning problem with a bi-connected graph $G=(V,E)$ modeling the environment. Assume that a loop decomposition of the graph is constructed. That is we have a cycle C_0 and a sequence of loops $L_1, L_2, \ldots, L_k$ such that the graph G can be constructed from C_0 by adding loops $L_1, L_2, \ldots, L_k$ incrementally. Since the construction of the graph G starts with a cycle C_0 (which is a connected graph) $C(L_i)$ is defined for every $i=1,2,\ldots,k$. Specially, we define $C(C_0)=C_0$. Moreover, we assume that each vertex $v \in V$ has assigned a loop or initial cycle that it is part of. This assignment is formally expressed by a function $\Gamma: V \rightarrow \{C_0, L_1, L_2, \ldots, L_k\}$.

In order to reduce the complexity of the code we assume that $\mu=n-2$ and unoccupied vertices of the goal situation to be in the cycle C_0 (that is $(v \in V \wedge (\forall r \in R) S^+(r) \neq v) \ \Rightarrow v \in C_0$). Overcoming these assumptions is discussed in the next section.

The *BIBOX* algorithm itself is built upon several primitives. The pseudo-code of the *BIBOX* algorithm is presented as algorithm 1 (the code is illustrated with pictures for easier understanding). Except the functions S_0 and S^+ we further have a function $S: R \rightarrow V$ expressing current positions of robots. Next we have functions $\Phi_0: V \rightarrow R \cup \{\perp\}$, $\Phi^+: V \rightarrow R \cup \{\perp\}$, and $\Phi: V \rightarrow R \cup \{\perp\}$ which are generalized inverses of S_0, S^+, and S respectively where the symbol $\perp$ stands for unoccupied vertex (that is, $(\forall r \in R) \Phi(S(r)) = r$; $\Phi(v) = \perp$ if $(\forall r \in R) S(r) \neq v$). Next, we assume that we have a sequence of potentially infinite sequences representing the solution of the problem $[P_{r_1}, P_{r_2}, \ldots, P_{r_\mu}]$. For easier expressing of the algorithm we also have functions $next/V(C,v)$, $prev/V(C,v)$, $next/S(C,r)$, $prev/S(C,r)$, $next/S^+(C,r)$, $prev/S^+(C,r)$ that return the next or the previous vertex or robot in the given cycle with respect to the clock-wise orientation of the cycle (that is, for instance $next/S^+(C,r)$ returns a robot next to the robot r in the cycle C with respect to the goal positions of robots expressed by S^+) . Finally, we use operations lock(X) and unlock (X) that locks or unlocks a set of vertices X . Each vertex is either locked or unlocked. Robots in the locked vertices are restricted to move.

The algorithm works in two phases. Robots whose goal positions are within the regular loops of the loop decomposition are placed to their goal positions in the first phase (lines 2-4 of *BIBOX*-Solve). The second phase consists in placing the robots to the goal positions in the original cycle of the cycle decomposition (line 5 of *BIBOX*-Solve). This is due to the fact that original cycle needs a specialized approach.

The first phase proceeds from the last loop to the first loop of the loop decomposition. After the robots are placed to their goal positions within the current loop the algorithm proceeds with the previous loop. Notice that after finishing a loop we obtain a problem of the same type but smaller.

Within a loop, robots are placed to their goal positions in the stack manner (that is, a new robot comes at the beginning of the loop and the loop is rotated - stack pushes). The last rotation of the loop places the robots to their destinations. When placing the robots within the loop it is necessary to distinguish between the situation when the robot is outside the loop (lines 3-8 of SolveRegularCycle) and the situation when the robot is already within the current loop (lines 10-29 of SolveRegularCycle).

The whole first phase manages with only one unoccupied vertex. The second unoccupied vertex is necessary for placing the robots within the original cycle of the decomposition. A completely different approach is used here. Having two unoccupied vertices we are able to exchange two robots with respect to the clock-wise ordering in the original cycle (function ExchangeRobots). Using the operation of exchanging robots it is not difficult to obtain the goal permutation of robots in the cycle with respect to the clock-wise ordering (lines 3-6 of SolveOriginalCycle). Finally, it is necessary to rotate the cycle to place robots to their goal positions (lines 7-8 of SolveOriginalCycle) and free vertices that should be finally unoccupied (lines 9-13 of SolveOriginalCycle).

Algorithm 1. The *BIBOX* algorithm in the pseudo-code. The presented code solves the problem of multi-robot path planning in a bi-connected graph $G=(V,E)$ with exactly two vertices unoccupied. The algorithm assumes that goal positions of the robots preserves unoccupied vertices in the initial cycle of the loop decomposition of the graph G.

```
function BIBOX-Solve: pair
1:   $m \leftarrow 0$
2:   for $c = k, k-1, \ldots, 1$ do
3:   |    if $|L_c| > 2$ then
4:   |    |    SolveRegularCycle$(c)$
5:   SolveOriginalCycle
6:   return $(m, [P_{r_1}, P_{r_2}, \ldots, P_{r_\mu}])$

procedure SolveRegularCycle$(c)$
1:   let $[u, x_1, x_2, \ldots, x_l, v] = L_c$
2:   for $i = 1, 2, \ldots, l$ do
3:   |    if $\Gamma(S(\Phi^+(x_i))) \neq L_c$ then
4:   |    |    lock $(L_c)$
5:   |    |    MoveRobot$(\Phi^+(x_i), u)$
6:   |    |    MoveUnoccupied$(v)$
7:   |    |    unlock $(L_c)$
8:   |    |    RotateCycle$^+(C(L_c))$
9:   |    else
10:  |    |    lock $(L_c)$
11:  |    |    MoveUnoccupied$(u)$
```

```
12:                 unlock ($L_c$)
13:                 $\rho \leftarrow 0$
14:                 **while** $S(\Phi^+(x_i))) \neq v$ **do**
15:                     RotateCycle$^+$($C(L_c)$)
16:                     $\rho \leftarrow \rho + 1$
17:                 lock($L_c$)
18:                 **let** $o \in V - (\bigcup_{i=c}^{k} L_i \cup C(L_c))$
19:                 MoveRobot($\Phi^+(x_i), o$)
20:                 lock($\{o\}$)
21:                 MoveUnoccupied($u$)
22:                 unlock($L_c$)
23:                 **while** $\rho > 0$ **do**
24:                     RotateCycle$^-$($C(L_c)$)
25:                     $\rho \leftarrow \rho - 1$
26:                 unlock($\{o\}$)
27:                 MoveRobot($\Phi^+(x_i), u$)
28:                 MoveUnoccupied($v$)
29:                 RotateCycle$^+$($L_c$)
30:     lock($L_c$)
```

```
procedure SolveOriginalCycle
1:      let $u \in C_0$ and $v \in V - C_0$ such that $\{u,v\} \in E$
2:      let $[x_1, x_2, \ldots, x_l] = C_0$
3:      for $i = 1,2,\ldots,l-1$ do
4:          $r \leftarrow next/S^+(C_0, \Phi^+(x_i))$; $q \leftarrow next/S(C_0, \Phi^+(x_i))$
5:          if $r \neq q$ then
6:              ExchangeRobots($r,q,u,v$)
7:      while $S(r) \neq S^+(r)$ do
8:          RotateCycle$^+$($C_0$)
9:      let $x \in C_0$ such that $\Phi^+(x) = \perp$ and $x$ is not locked
10:     MoveUnoccupied($x$)
11:     lock($x$)
12:     let $y \in C_0$ such that $\Phi^+(y) = \perp$ and $y$ is not locked
13:     MoveUnoccupied($y$)
```

```
procedure ExchangeRobots($r,q,u,v$)
1:      $s \leftarrow \Phi(v)$
2:      MoveUnoccupied($u$)
3:      SwapRobotUnoccupied($v,u$)
4:      while $S(r) \neq u$ do
5:          RotateCycle$^+$($C_0$)
6:      SwapRobotUnoccupied($u,v$)
7:      lock($u$)
8:      $\rho \leftarrow 0$
9:      while $S(q) \neq prev/V(C_0,u)$ do
10:         RotateCycle$^+$($C_0$)
11:         $\rho \leftarrow \rho + 1$
12:     MoveUnoccupied($next/V(C_0,u)$)
13:     SwapRobotUnoccupied($prev/V(C_0,u),u$)
14:     SwapRobotUnoccupied($u,next/V(C_0,u)$)
15:     SwapRobotUnoccupied($v,u$)
16:     SwapRobotUnoccupied($u,prev/V(C_0,u)$)
17:     SwapRobotUnoccupied($next/V(C_0,u),u$)
```

```
18:     SwapRobotUnoccupied$(u,v)$
19:     SwapRobotUnoccupied$(u,v)$
20:     **while** $\rho > 0$ **do**
21:         RotateCycle$^{-}(C_0)$
22:         $\rho \leftarrow \rho - 1$
23:     SwapRobotUnoccupied$(v,u)$
24:     **while** $S(s) \neq u$ **do**
25:         RotateCycle$^{+}(C_0)$
26:     SwapRobotUnoccupied$(u,v)$
27:     unlock$(u)$

**procedure** RotateCycle$^{+}(C)$
1:      **let** $x \in C$ such that $\Phi(x) = \perp$ and $x$ is not locked
2:      **for** $i = 1,2,\ldots,|C|$ **do**
3:          SwapRobotUnoccupied$(prev/V(C,x),x)$
4:          $x \leftarrow prev/V(C,x)$

**procedure** RotateCycle$^{-}(C)$
1:      **let** $x \in C$ such that $\Phi(x) = \perp$ and $x$ is not locked
2:      **for** $i = 1,2,\ldots,|C|$ **do**
3:          SwapRobotUnoccupied$(next/V(C,x),x)$
4:          $x \leftarrow next/V(C,x)$

**procedure** MoveUnoccupied$(v)$
1:      **let** $x \in V$ such that $\Phi(x) = \perp$ and $x$ is not locked
2:      **let** $[x = p_1, p_2, \ldots, p_j = u]$ be a shortest path between
3:          $x$ and $v$ in $G$ not containing locked vertices
4:      **for** $i = 1,2,\ldots,j-1$ **do**
5:          SwapRobotUnoccupied$(p_{i+1}, p_i)$

**procedure** MoveRobot$(r,v)$
1:      **let** $[S(r) = p_1, p_2, \ldots, p_j = v]$ be a shortest path between
2:          $S(r)$ and $v$ in $G$ not containing locked vertices
3:      **for** $i = 1,2,\ldots,j-1$ **do**
4:          lock$(\{p_i\})$
5:          MoveUnoccupied$(p_{i+1})$
6:          unlock$(\{p_i\})$
7:          SwapRobotUnoccupied$(p_i, p_{i+1})$

**procedure** SwapRobotUnoccupied$(u,v)$
1:      $S(\Phi(u)) = v$ ; $\Phi(u) = \perp$ ; $\Phi(v) = r$
2:      **for** $i = 1,2,\ldots,\mu$ **do**
3:          $p_m^{r_i} = S(r_i)$
4:      $m \leftarrow m + 1$
```

Analysis of the *BIBOX* Algorithm

We briefly mention ideas supporting the correctness of the *BIBOX* algorithm in this section. Next, asymptotic time complexity of the algorithm is discussed. All the claims are presented as a series of propositions.

Lemma 2 (each vertex in a cycle). For any vertex $v \in V$ of a given bi-connected graph $G = (V, E)$ there exists a cycle in G containing v. ■

Proof. Let us have a loop decomposition of the graph G according to the lemma 1. Then $C(\Gamma(v))$ with respect to the given decomposition determines a cycle containing v. ■

Proposition 1 (correctness of the BIBOX algorithm). The *BIBOX* algorithm (algorithm 1) is correct. That is, it solves the problem of path planning for multiple robots under specified conditions. ■

Idea of proof. The crucial point where the algorithm may be suspected to fail is when a path between two vertices is searched under the condition that it does not contain any locked vertex (the failure may be caused by the non-existence of such a path - see lines 4-5 of MoveRobot). Without detailed proof observe that such path always exists since we lock at most one vertex in a cycle (lemma 2) in the not yet finished part of the graph. Hence a path using the not locked section of the cycle exists. ■

Proposition 2 (complexity of the BIBOX algorithm). The *BIBOX* algorithm (algorithm 1) solves the problem of path planning for multiple robots with the graph $G = (V, E)$ in $O(|V|^3)$ steps.

Proof. The initial loop decomposition can be found in $O(|V| + |E|)$ [9] which is $O(|V|^2)$. Placing a single robot (one iteration of the cycle on line 2 of SolveRegularCycle) requires $O(|V|^2)$ steps since a robot must be moved across the whole graph (must visit all the vertices) in the worst case and a moving through an edge takes $O(|V|)$ steps (one iteration of cycle on line 3 of MoveRobot). Hence, we need $O(|V|^3)$ steps to place robots to their goal positions in the loops except the original cycle of the decomposition.

For the original cycle we need $O(|V|)$ operations of exchanging robots (line 6 of SolveOriginalCycle). Each operation of exchange robots takes $O(|V|^2)$ steps since the cycle must be rotated $O(|V|)$ times and each rotation takes $O(|V|)$ steps. Again we have $O(|V|^3)$ steps for solving the original cycle. ■

Extensions and the Real Implementation

The presented pseudo-code of the *BIBOX* algorithm requires two special assumptions. The assumption that $\mu = n - 2$ can is easy to overcome since it is possible to use dummy robots instead of unoccupied vertices and to ignore their moves in the solution. The assumption that finally unoccupied vertices must be in the original cycle is little bit complicated. We need to modify the required solution given by the function S^+ so that unoccupied vertices are moved to the original cycle along two disjoint paths (that always exist in a bi-connected graph). After solving the problem by the presented algorithm we move unoccupied vertices back along these paths which finishes the solution of the original unmodified problem.

We implemented the proposed algorithm in C++. Our implementation uses additional techniques to increase speed and quality of solutions. Space limitations do not allow us to describe them in details. Nevertheless, let us mention the main ideas. First, the non-determinism of the code (for example line 1 of MoveUnoccupied) is replaced by the code that prefers shortest possible solutions. Second, the implementation performs additional analysis of the solution to increase parallelism (more than a single move is done in a time-step). And finally, the solving process for the original cycle of the decomposition is implemented in a more sophisticated way - again to shorten and to parallelize the solution.

Experimental Evaluation

We evaluated our new *BIBOX* algorithm by a collection of experiments. The experimental evaluation is targeted on the analysis of the quality of the resulting solutions as well as on the performance of the *BIBOX* algorithm.

The experimental evaluation is divided into two parts. A comparison of the *BIBOX* algorithm with two domain-independent planners is made in the first part. We selected two domain-independent planners for this evaluation - namely *SGPLAN 5.1* [3] and *LPG-td 1.0* [2] planners. This selection was guided by the fact that these two planners proved to perform well on the problem of multi-robot path planning. Both selected planners rank among the best in the International Planning Competition (*IPC*) [1]. We also considered some other planners from *IPC* for this evaluation - namely *IPP 4.1*, *MAXPLAN/miniSAT 2.0*, *SATPLAN/Siege 4*, and *STAN 3*. However, these planners failed to solve even the very small instances of the multi-robot path planning problem which makes them unsuitable for the experiments.

The second part of the experimental evaluation is targeted on the performance tests of the *BIBOX* algorithm on the large instances of the multi-robot path planning problem.

All the data necessary for reproducing the presented experiments including the source code of *BIBOX* are available at: http://ktiml.mff.cuni.cz/~surynek/research/icra2009/.

Competitive Comparison

We concentrate on comparison of the performance of the *BIBOX* algorithm with domain-independent planners *SGPLAN* and *LPG* in this part. These planners turned out to be the only systems publicly available that are able to solve the problem of path planning for multiple robots.

We generated several problems with random bi-connected graphs of small sizes modeling the environment. The graphs were generated by random adding of loops (connection vertices of the loop are randomly selected in the already existing graph using uniform distribution) of random sizes (the size of the loop is randomly selected from the interval $1,2,\ldots,4$ using uniform distribution) to the original cycle of random size (again the size of the cycle is randomly selected from the interval $3,4,\ldots,6$ using uniform distribution). Then the random initial positions S_0 and random goal positions S^+ of the robots were generated. In addition, two categories of problems were generated - problems with just 2 unoccupied vertices and problems with 10% of unoccupied vertices.

These problems were solved by using the *BIBOX* algorithm and by *SGPLAN* and *LPG* planners. The tests were run on a machine with AMD Opteron 1600MHz, 1GB of RAM, under Mandriva Linux 10.1. Along the process of solving several statistical data were collected. The comparison of duration of plans (lengths of solutions; m) is shown in figure 2. There are sizes of the graphs shown over the horizontal axis. Problems are ordered according to the increasing plan duration produced by the *BIBOX* algorithm. The new *BIBOX* algorithm produces plans that rank among those produces by the *SGPLAN* and *LPG* with respect to the length of solution. *SGPLAN* produces shortest solutions. The comparison of runtime is shown in figure 3. This result undoubtedly shows that the *BIBOX* algorithm is faster by orders of magnitude than *SGPLAN* and *LPG* on the problem of multi-robot path planning. Finally, the figure 4 shows comparison of parallelism of resulting solutions (the ratio of the number of movements to the length of the solution). The *BIBOX* algorithm has almost always the highest parallelism of solutions.

The above experimental results provide a justification for the claim that the *BIBOX* algorithm significantly outperforms *SGPLAN* and *LPG* in terms of speed on multi-robot problems. In other measured aspects *BIBOX* is competitive.

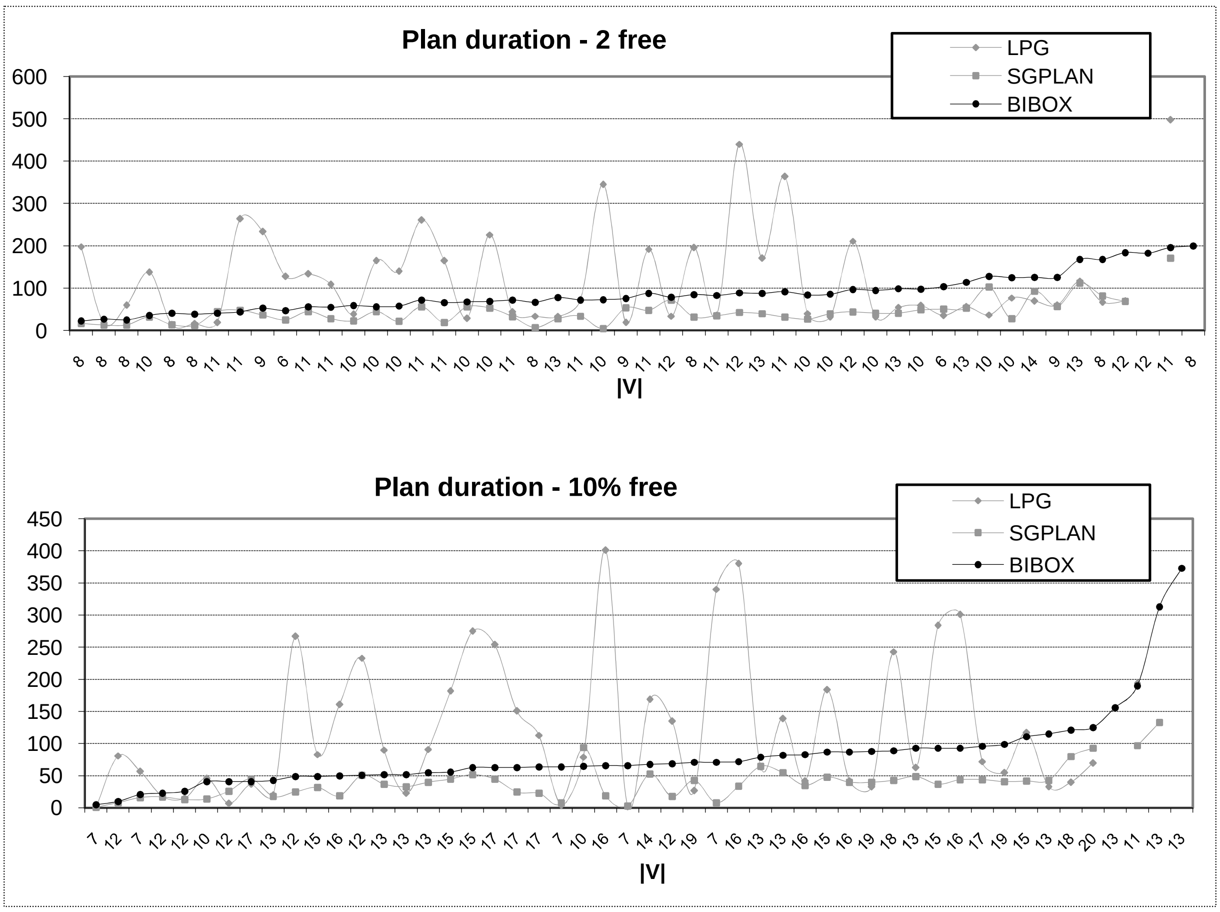

Figure 2. Comparison of plan durations of *LPG*, *SGPLAN*, and *BIBOX*. The number of time-steps is compared (parallel execution is allowed) depending on the size of the graph defining the environment. Two experiments are shown: a situation with 2 unoccupied vertices and with 10% of unoccupied vertices.

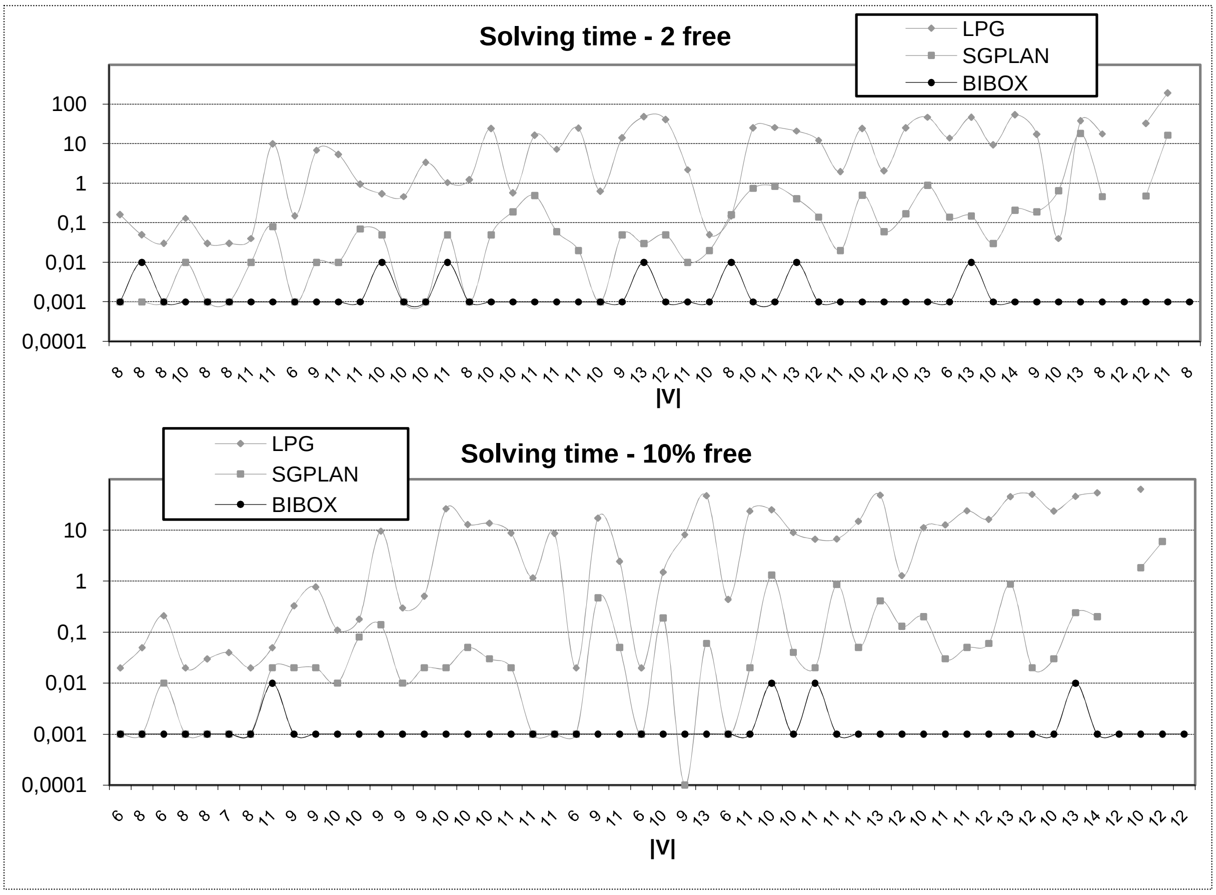

Figure 3. Comparison of solving times of *LPG*, *SGPLAN*, and *BIBOX*. The solving runtimes in seconds are compared depending on the size of the graph defining the environment (the vertical time axis uses the logarithmic scale). Two experiments are shown: a situation with 2 unoccupied vertices and with 10% of unoccupied vertices.

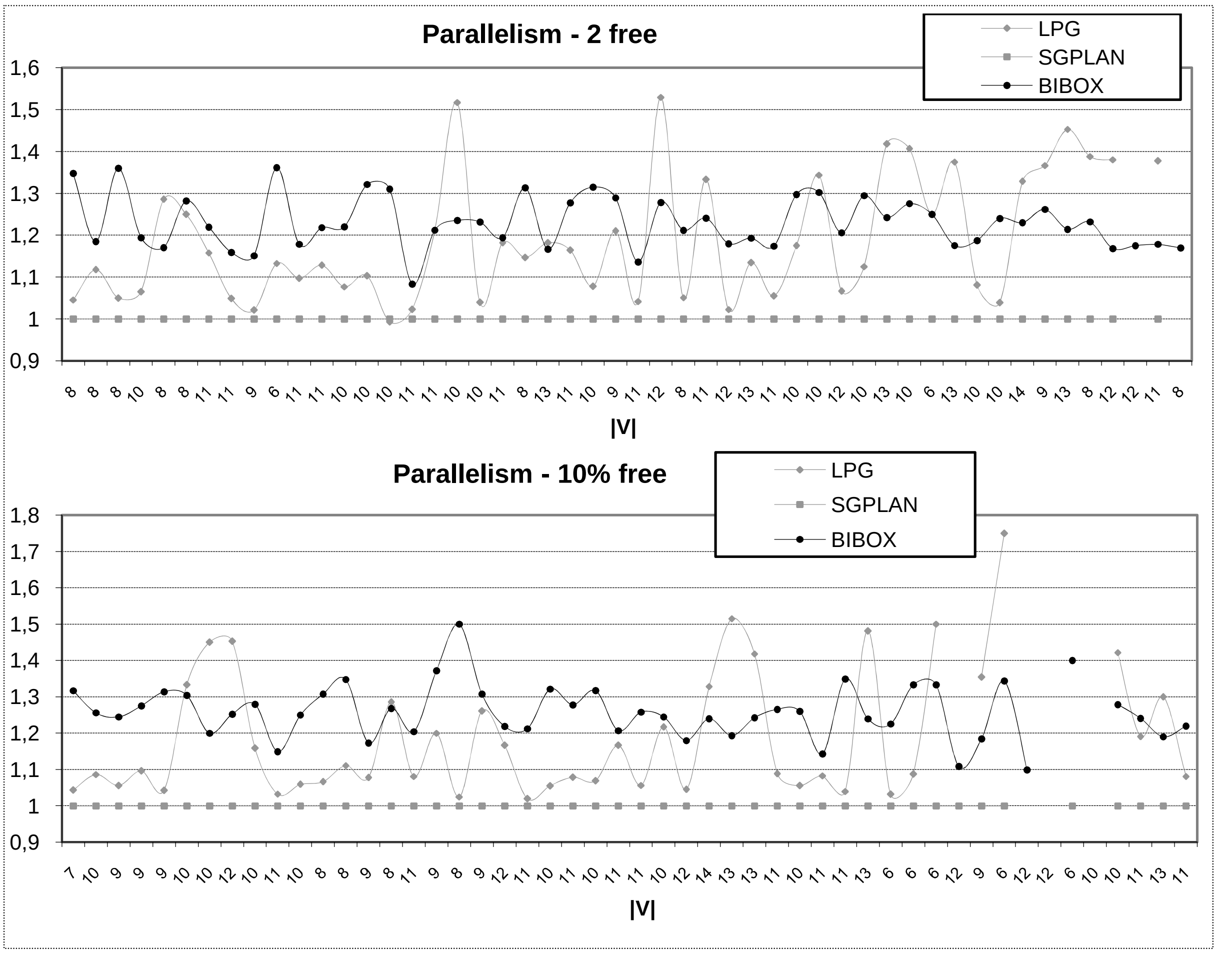

Figure 4. Comparison of parallelism of *LPG*, *SGPLAN*, and *BIBOX*. The average parallelism is compared depending on the size of the graph defining the environment. Average parallelism is defined as the ratio of the number of movements to the length of solution (m). Two experiments are shown: a situation with 2 unoccupied vertices and with 10% of unoccupied vertices.

Performance Analysis

This part of experimental evaluation is devoted to experimental evaluation of the *BIBOX* algorithm on large problems. These problems are unsolvable in reasonable time by domain-independent planners (this observation renders the tested planners useless for problems of real-life size).

We generated a collection of problems with random bi-connected graphs. In this case the size of graphs ranged up to almost 400 vertices. The graphs were generated by random adding of loops of random sizes to the original cycle of random size. The range of sizes of the loop was $1,2,\ldots,8$ and range of sizes of the original cycle was $3,4,\ldots,10$. Three categories of problems were generated - problems with just 2 unoccupied vertices, problems with 10% of unoccupied vertices, and problems with 50% unoccupied vertices.

The results regarding lengths of solution, runtime, and parallelism are shown in figure 5 (tests were run on the same machine). Problems are ordered along the horizontal axis according to the increasing size of the graph. The most importantly, the results show that by using the *BIBOX* algorithm we can simply solve a problem with solutions consisting of 10000s of moves which is far beyond what can be reached by domain-independent planners. This result shows that *BIBOX* is capable of solving problems of real-life size.

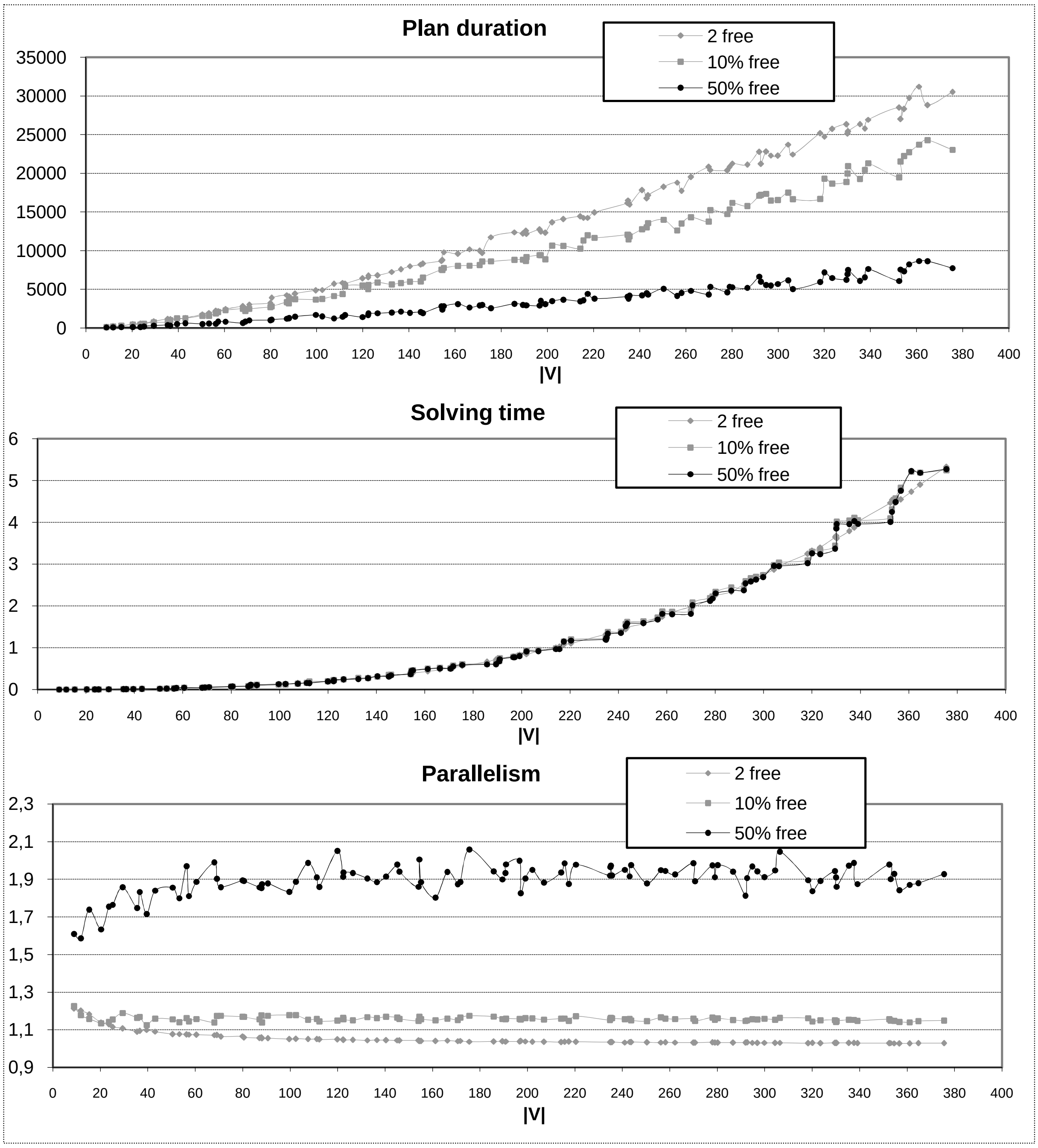

Figure 5. Performance analysis of the *BIBOX* algorithm on large problems. The plan duration, solving runtime, and average parallelism are compared depending on the size of the graph defining the environment. Three setups are used for every test: a setup with 2 unoccupied vertices, with 10% of unoccupied vertices, and with 50% unoccupied vertices.

Related Works and Concluding Discussion

The problem of multi-robot path planning has been already studied in the literature. It is sometimes referred as a problem of *pebble motion on graphs* or *sling box puzzle*. The non-constructive proof of that any problem with bi-connected non-bipartite graph can be solved when there is at least one unoccupied vertex is given in [10]. The constructive variant of this result is discussed in [5]; the authors present an algorithm of the worst case time complexity of $O(|V|^3)$. However, parallelism and experimentally tested performance of the algorithm is not addressed in the paper.

A modification of the problem with the requirement of the shortest possible solution has been also studied. However, the result is quite negative [6] since this requirement makes the problem NP-complete assuming that there is a constant number of unoccupied vertices.

A relatively modern approach to the problem of multi-robot path planning is presented in [7], [8]. The author's approach is to decompose the graph of the problem to well structured sub-graphs. The drawback is that this approach is based on (non-polynomial) search and only small numbers of robots were tested (up to 10 robots; notice that we simply manage 100s of robots).

Our contribution consists in complete solving of a relatively large class of the problem of multi-robot path planning (in [4] a solving process of this class of the problem is referred as an open question). In addition to related works, we present an experimental evaluation which shows that our algorithm is capable of solving order of magnitudes larger problems than other comparable algorithms. We plan to extend our technique to arbitrary graphs (not only bi-connected) and we plan to implement a more efficient algorithm for problems with only one unoccupied vertex than presented in [5]. Another issue worth studying is how to increase parallelism and how to shorten solutions.

References

[1] Gerevini, A., Bonet, B., Givan, B. (editors), "5th International Planning Competition", event in the context of ICAPS 2006 conference, United Kingdom, University of Brescia, Italy, 2006,
Available: http://ipc5.ing.unibs.it (January 2008).

[2] Gerevini, A., Serina, I., "Homepage of LPG", research web page, University of Brescia, Italy, 2008,
Available: http://zeus.ing.unibs.it/lpg/ (January 2008).

[3] Hsu, C. W., Wah, B. W., Huang, R., Chen, Y. X., "SGPlan 5: Subgoal Partitioning and Resolution in Planning", research web page, University of Illinois, USA, 2008,
Available: http://manip.crhc.uiuc.edu/programs/SGPlan/index.html (January 2008).

[4] Johnson, D. S., "The NP-Completeness Column: An Ongoing Guide", Journal of Algorithms, Volume 4 (4), pp. 397-411, Elsevier, 1983.

[5] Kornhauser, D., Miller, G. L., Spirakis, P. G., "Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications", in Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241-250, IEEE Press, 1984.

[6] Ratner, D., Warmuth, M. K., "Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE Is Intractable", in Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), pp. 168-172, Morgan Kaufmann Publishers, 1986.

[7] Ryan, M. R. K., "Multi-robot path planning with sub graphs", in Proceedings of the 19th Australasian Conference on Robotics and Automation, Auckland, New Zeland, Australian Robotics & Automation Association, 2006.

[8] Ryan, M. R. K., "Graph Decomposition for Efficient Multi-Robot Path Planning", in Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, pp. 2003-2008, IJCAI Conference, 2007.
Available: http://www.ijcai.org (February 2008).

[9] 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.

[10] West, D. B., "Introduction to Graph Theory, second edition", Prentice-Hall, 2000.

[11] Wilson, R. M., "Graph Puzzles, Homotopy, and the Alternating Group", Journal of Combinatorial Theory, Ser. B 16, pp. 86-96, Elsevier, 1974.