International Journal on Artificial Intelligence Tools
Vol. XX, No. X (2014) 1–19

World Scientific
www.worldscientific.com

REDUNDANCY ELIMINATION IN HIGHLY PARALLEL SOLUTIONS OF MOTION COORDINATION PROBLEMS *

PAVEL SURYNEK

Faculty of Mathematics and Physics, Charles University,
Malostranské náměstí 25, Praha, 118 00, The Czech Republic
pavel.surynek@mff.cuni.cz

Received: November 17, 2012
Accepted: March 4, 2013
Revised: May 5, 2013

Problems of motion coordination of multiple entities are addressed in this paper. These problems are dealt on the abstract level where they can be viewed as tasks of constructing a spatial-temporal plan for a set of identical mobile entities. The entities reside in a certain environment where they can move. Each entity need to reach a given goal position supposed it is starting from some initial position. The most abstract formal representations of coordinated motion problems are known as “pebble motion on a graph” and “multi-robot path planning”. The existent algorithms for pebble motion and multi-robot problems were suspected of generating solutions containing redundancies. This hypothesis eventually confirmed in this work. We present several techniques for identifying and eliminating redundancies from solutions generated by existent algorithms. An extensive experimental evaluation was performed and it showed that the quality of generated solutions can be improved up to the order of magnitude. We also identify parameters characterizing instances of problems where a significant improvement is expectable.

Keywords: multi-robot path planning; pebble motion on a graph; redundancy elimination; parallel plans; SAT based optimization.

1. Introduction, Context, and Motivation

Problems of coordinated motion of multiple identical entities as they are introduced in [4, 8, 10, 16] (terms “multi-robot path planning” or “cooperative path-finding” are also used to denote the same or similar problem) represent a basic abstraction for many real-life and theoretical tasks. The classical task that can be abstracted as a problem of coordinated motion takes place in a certain physical environment where identical mobile entities are moving (typically represented by mobile robots). Each entity is given its initial and goal positions in the environment between which it should relocate. The task is to construct a spatial-temporal plan for all the entities such that they can reach their goal positions following the plan while the plan satisfies certain natural constraints. These constraints are constituted by a requirement that the entities must avoid obstacles in the environment and must not collide with each other.

* This work is supported by the PRVOUK project reference number P46 provided by the Charles University in Prague (Program rozvoje vědních oblastí na Univerzitě Karlově).

The standard abstraction adopted throughout this work uses an undirected graph to model the environment. Vertices of this graph represent positions in the environment and edges represent passable regions between two positions. An arrangement of entities in the environment is abstracted as a uniquely invertible assignment of entities to vertices. At least one vertex remains unoccupied in order to make the movement of entities possible – for example moving in a circle where each entity follows the preceding entity is not allowed. The time is discrete; it is an ordered set of time steps isomorphic to the structure of natural numbers. A way in which an arrangement of entities can be transformed into another can slightly differ in variants of the problem. The best known abstract formalizations of coordinated motion problems are represented by *pebble motion on a graph* (PMG) as defined in [4] and [16], and *multi-robot path planning* (MRPP) as defined in [8, 10, 11] while the latter allows higher parallelism.

Abstract problems of coordinated motion of multiple entities on a graph are motivated by many real-life problems. The most typical motivating example is motion planning of a group of mobile robots that are moving in 2-dimensional space. Generally, if there is enough free space in the environment, algorithms based on search for shortest paths in a graph with an eventual local repairs if collision occurs can be used [1]. However, if non-trivial amount of space is occupied different approaches must be adopted.

Many well known puzzles can be formulated as coordinated motion on a graph. The best known is so called Lloyd's 15-puzzle and its generalizations as described in [6, 7] and [16]. In practice, entities may be represented by various mobile or movable objects – for example rearranging containers in some storage area can be interpreted as a problem of coordinated motion where entities are represented by containers. Exactly this interpretation has been used for planning motions of automated straddle carriers in a storage area in Patrick port facility at Port Brisbane in Queensland as reported in [8]. Although the approach suggested in [8] is applied on few movable entities it clearly demonstrates the usefulness of discussed abstractions. Entities do not necessarily need to be physical objects. Virtual spaces of computer simulations and games contain many situations where motions of certain entities must be planned. A typical example is a coordination of groups of units in real-time strategic computer games (RTS) [14].

It is necessary to emphasize that contrary to multi-agent motion planning [3], the centralized approach is adopted in this work. This means that the environment is fully observable for the central planning mechanism and the individual entities merely execute the centrally created plan.

There exist several relatively efficient methods for solving problems of coordinated motion on a graph. This work is particularly targeted on solving methods described in [10, 11]. These methods represent algorithms for the class of problems where the graph modeling the environment is bi-connected [15] and the graph is densely occupied by entities. More precisely, the number of entities μ is comparable to the size of the set of vertices (that is, $\mu = \Theta(|V|)$). Despite the good performance of these methods, generated solutions are suspected of containing certain redundancies. This is a hypothesis whose examination is the main contribution of this paper. If it is the case that generated solutions contain redundancies, then a question how they can be removed to improve the solution arises.

The task was to analyze solutions of non-trivial size, which turned out to be infeasible to be done manually. Moreover, we were searching redundancies of a priori unknown nature. Therefore, a software tool GraphRec [5] allowing visual analysis of solutions of problems of motion on a graph has been developed and employed in this analysis. Several types of redundancies were observed using the GraphRec software in generated solutions. The most prominent three of them that we manage to formally capture are described in this paper. Methods for automated discovering and elimination of these three defined types of redundancies are suggested and analyzed theoretically as well as experimentally. We also suggest to model the problem of motion on a graph as propositional satisfiability (SAT) [1] which allows us to discover very generic redundancies automatically.

The top level organization of the paper has two parts. The first part explains a specific variant of the coordinated motion problem (section 2) and the basic solving algorithm (section 3); this part mostly recalls existing concepts. The second part contains the main contribution of this work; redundancy elimination methods are described (section 4), and the benefit of suggested methods is justified in the experimental section (section 5). Additionally a SAT based solution improvement technique is described in section 6.

2. Pebble Motion on a Graph (PMG)

In the rest of the paper, we restrict ourselves on the variant of the entity motion coordination problem known as *pebble motion on a graph* (PMG) defined in [7] and [16]. The work can be extended on other variants of the problem such as *multi-robot path planning* (MRPP) using minor modifications only.

The task in pebble motion on a graph is given by an undirected graph with an *initial* and a *goal arrangement* of pebbles in vertices of the graph. Each vertex contains at most one pebble (which represents a movable entity) and at least one vertex remains unoccupied. The task is to find a sequence of moves for each pebble such that all the pebbles reach their goal vertices. A pebble can move into a neighboring unoccupied vertex while no other pebble is entering the same target vertex simultaneously. The following definition formalizes the problem. An illustration of the problem is shown in Fig. 1.

Definition 1 *(pebble motion on a graph)*. Let $G = (V, E)$ be an undirected graph and let $P = \{p_1, p, \dots, p_\mu\}$ be a set of pebbles where $\mu < |V|$. The initial arrangement of pebbles is defined by an injective function $S_P^0: P \longrightarrow V$ (that is $S_P^0(p_i) \neq S_P^0(p_j)$ for $i, j = 1,2, \dots, \mu$ with $i \neq j$); the goal arrangement of pebbles is defined by another injective function $S_P^+: P \longrightarrow V$. A problem of PMG is the task to find a number ξ and a sequence $S_P = [S_P^0, S_P^1, \dots, S_P^\xi]$ where $S_P^k: P \longrightarrow V$ is an injective function for every $k = 1,2, \dots, \xi$. The following constraints must hold for S_P:

(i) $S_R^\zeta = S_R^+$, that is, pebbles eventually reach their destinations.

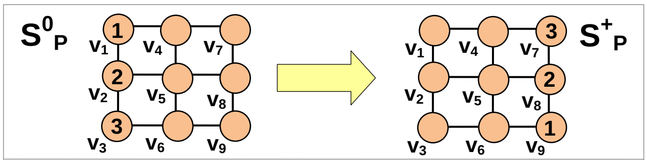
(ii) Either $S_P^k(p) = S_P^{k+1}(p)$ or $\{S_P^k(p), S_P^{k+1}(p)\} \in E$ for every $p \in P$ and $k = 1,2,\ldots,\xi - 1$.

(iii) $S_P^k(p) \neq S_P^{k+1}(p)$ and $S_P^k(q) \neq S_P^{k+1}(q)$ for $\forall q \in P$ such that $q \neq p$ must hold for every $p \in P$ and $k = 1,2,\ldots,\xi - 1$, that is no two pebbles can enter the same target vertex simultaneously.

The problem described above is formally a quadruple $\Pi = (G = (V,E), P, S_P^0, S_P^+)$. □

In practice, the *quality of solution* matters. The typical measures of the quality of solution are its length (the total number of moves) and the makespan (which corresponds to the number ξ). These numbers are required to be small. Unfortunately, requiring either the length of solution or its makespan to be as small as possible makes the problem intractable [7] (the decision variant of the problem is *NP*-complete). On the other hand, if there is no requirement on the quality, the question whether there exists a solution is in the *P* class as it shown in [4] and [16].

However, methods showing evidence that the problem belongs to the *P* class described in [4] and [16] generates excessively long solutions that are unsuitable for practice when each movement of an entity represented by a pebble has a nontrivial cost. Therefore, it was necessary to find a compromise between the quality of solution and computational effort of its construction. Methods following this compromise are described in [10] and [11]. Solutions produced by these methods were submitted for analysis into the visualization tool in order to find if and how they can be further improved.



Solution of the problem of pebble motion on a graph with P={1,2,3}

ξ=6

P	S^0_P	S^1_P	S^2_P	S^3_P	S^4_P	S^5_P	$S^6_P=S^+_P$
1	**v_1**	v_4	v_7	v_8	v_9	v_9	**v_9**
2	**v_2**	v_2	v_1	v_4	v_7	v_8	**v_8**
3	**v_3**	v_3	v_3	v_2	v_1	v_4	**v_7**

Fig 1. *An illustration of a **PMG problem***. The task is to move pebbles from their initial positions specified by S_P^0 to the goal positions specified by S_P^+. A solution of length 6 is shown.

3. Solving Coordinated Motion Problems

This section is devoted to a brief recall of algorithms described in [10] and [11]. Understanding how these algorithms work will provide us an insight into the structure of solutions produced by them. This theoretical insight founded the hypothesis that solutions can be further improved.

A very important class of pebble motion problems is formed by those whose graph is *bi-connected* which intuitively means that each pair of vertices is connected by two disjoint paths.

Definition 2 ***(connectivity, bi-connectivity).*** An undirected graph $G = (V, E)$ is *connected* if $|V| \geq 2$ and for every pair of distinct vertices $u, v \in V$ there exists a path connecting u and v in G. An undirected graph $G = (V, E)$ is *bi-connected* if $|V| \geq 3$ and for every vertex $u \in V$ the graph $G' = (V - \{u\}, E \cap \{\{v, w\} | v, w \in V \wedge v \neq u \wedge w \neq u\})$ is connected. □

The importance of this class of problems is assessed by the fact that they are almost always solvable. Moreover, spatial environments in real tasks are often abstracted as two dimensional grids which are bi-connected in most cases.

If the bi-connected graph contains at least two unoccupied vertices and it is not a cycle, then every goal arrangement of pebbles is reachable from every initial arrangement [10]. If the graph contains just one unoccupied vertex which can be without loss of generality fixed, then any arrangement of pebbles can be regarded as a *permutation* with respect to the initial arrangement.

A permutation is *even* if it can be composed of the even number of transpositions; otherwise it is *odd*. If the goal arrangement represents an even permutation, then the problem is always solvable. In case of an odd permutation, the problem is solvable if and only if the graph contains a cycle of the odd length [16].

An inductive construction of bi-connected graphs by adding *handles* is a pivotal concept in developing solving algorithms. Let $G = (V, E)$ be a graph, a *handle* with respect to G is a sequence of vertices $L = [u, x_1, x_2, \dots, x_l, v]$, where $u, v \in V$ and $x_i \notin V$ for $i = 1,2, \dots, l$ (it allowed that $l = 0$). The result of an *addition* of handle L to graph G is a new graph $G' = (V', E')$, where $V' = V \cup \{x_1, x_2, \dots, x_l\}$ and either $E' = E \cup \{\{u, v\}\}$ if $l = 0$ or $E' = E \cup \{\{u, x_1\}, \{x_1, x_2\}, \dots, \{x_{l-1}, x_l\}, \{x_l, v\}\}$ if $l \geq 1$. Every bi-connected graph $G = (V, E)$ can be constructed from a cycle by a sequence of handle additions.

3.1. *The BIBOX-θ Solving Algorithm*

The *BIBOX-θ* algorithm [11] solves a case of the PMG problem when the graph is bi-connected and there is single unoccupied vertex. The algorithm provides a good performance for the described class of problems in terms of speed and quality of generated solutions. This is the main reason why solutions produced by this algorithm are studied.

In the first phase, a handle decomposition is found; that is, a cycle - called *initial cycle* - and a sequence of handles is determined. Without loss of generality it is required that the unoccupied vertex within the goal arrangement of pebbles is located in the initial cycle. The algorithm then proceeds inductively according to the handle decomposition from the last handle to the initial cycle with the first handle.

Two properties of bi-connected graphs with at least one unoccupied vertex are exploited while pebbles are placed within handles: (a) every vertex can be made unoccupied (this is even true for a connected graph), (b) every pebble can be moved to an arbitrary vertex. A handle is processed in the following way. An orientation of the handle is chosen first – this orientation determines ordering of vertices within the handle. The first and the last vertex of the handle are the connection points to the remainder graph.

Then pebbles starting with the pebble whose goal position is in the second vertex of the handle are placed into the handle in the stack manner. The current pebble is moved to the last vertex of the handle.

Two cases are distinguished here. If the pebble is already somewhere in the handle it must be moved outside first. If the current pebble is outside the handle, then it can be moved into the last vertex of the handle using property (b).

After placing the pebble into the last vertex of the handle, the handle is rotated once in the direction to the first vertex. When all the pebbles within the handle are processed the task is to solve the problem of the same type on a smaller graph.

Nevertheless, the stack manner of placing pebbles cannot be applied for the initial cycle and the first handle of the decomposition. The algorithm uses a database containing pre-calculated optimal solutions for transpositions and rotation of pebbles along 3-cycles in graphs consisting of a cycle and a handle. A solution to any solvable instance on the initial cycle with the first handle is then composed of solutions from such a database.

3.2. *A Case with More Unoccupied Vertices*

If there are exactly two unoccupied vertices in the graph an alternative more efficient placing of pebbles in the initial cycle and the first handle can be used [10]. If there are more than two unoccupied vertices in the graph the approach proposed in [11] is to fill all the remaining unoccupied vertices except two with extra pebbles. The instance is then solved by the *BIBOX-θ* algorithm and the solution is post-processed by removing movements of extra pebbles out of the solution.

This approach is however suspected of generating unnecessary movements for original pebbles. Notice that original pebbles have to make quite complicated movements when an extra pebble is being placed into a handle. All these movements of the original pebbles are redundant in fact since movements of the extra pebble will be eventually filtered out.

4. Elimination of Redundancies

Several types of redundancies were discovered using the *GraphRec* software. A formal description of these redundancies and algorithms for their elimination are provided in the following sections. When reasoning about redundancies, it is convenient to assume solutions to be sequential; that is, a solution has just one movement between consecutive time steps. Fortunately, the *BIBOX-θ* algorithm can produce solutions in this form. A solution of this form can be viewed as a sequence of moves.

The notation $k_i: u_i \rightarrow v_i$ will denote a move of a pebble k_i from a vertex u_i to a vertex v_i commenced at time step i. The move is called *non-trivial* if $u_i \neq v_i$. From the formal point of view, the sequential solution is a sequence of non-trivial moves $\Phi = [k_i: u_i \rightarrow v_i | i = 1,2, \dots, \xi - 1]$ (consistency with Definition 1 is also assumed).

Definition 3 *(inverse moves)*. Two consecutive moves $k_i: u_i \rightarrow v_i$ and $k_{i+1}: u_{i+1} \rightarrow v_{i+1}$ with $i \in \{1,2,\ldots,\xi-2\}$ are called *inverse* if $k_i = k_{i+1}$, $u_i = v_{i+1}$, and $v_i = u_{i+1}$. □

Observe that a pair of inverse moves can be left out of the solution without affecting its *validity* – resulting sequence still solves the problem. However, elimination of an inverse pair may cause that another pair of inverse moves arises. Hence, it is necessary to remove inverse moves from the solution repeatedly until there are any.

Algorithm 1. *Elimination of inverse moves.*

function *Erase-Inverse-Moves* (Φ): **sequence**
1: **do**
2: $\eta \leftarrow \emptyset$
3: **let** $[k_1: u_1 \rightarrow v_1, k_2: u_2 \rightarrow v_2, \ldots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
4: **for** $i = 1,2,\ldots,\xi-1$ **do**
5: **if** $k_i: u_i \rightarrow v_i$ **and** $k_{i+1}: u_{i+1} \rightarrow v_{i+1}$ are inverse **then**
6: $\eta \leftarrow \eta \cup \{k_i: u_i \rightarrow v_i, k_{i+1}: u_{i+1} \rightarrow v_{i+1}\}$
7: $\Phi \leftarrow \Phi - \eta$
8: **while** $\eta \neq \emptyset$
9: **return** Φ

The process of elimination inverse moves is expressed as Algorithm 1. The worst case time complexity of the algorithm is $O(|\Phi|^2)$, the space complexity is $O(|\Phi|)$.

Definition 4 *(redundant moves)*. A sequence of moves $[k_{i_j}: u_{i_j} \rightarrow v_{i_j} | j = 1,2,\ldots,l]$, where $I = [i_j \in \{1,2,\ldots,\xi-2\} | j = 1,2,\ldots,l]$ is a an increasing sequence of indices, is called *redundant* if $|\{k_{i_j} | j = 1,2,\ldots,l\}| = 1$, $u_{i_1} = v_{i_l}$, and for each move $k_\iota: u_\iota \rightarrow v_\iota$ with $i_1 < \iota < i_l \wedge \iota \notin I$ it holds that $k_\iota \neq k_{i_1} \Rightarrow u_{i_1} \notin \{u_\iota, v_\iota\}$. □

Redundant moves represents generalization of inverse moves (a pair of inverse moves form a redundant sequence). It is a sequence of moves, which relocates a pebble into some vertex for the second time while the other pebbles do not enter this vertex at any time step between the beginning and the end of the sequence. Eliminating a redundant sequence of moves preserves validity of the solution.

Again, it is necessary to remove redundant sequences repeatedly since its removal may cause that another redundant sequence arises.

Algorithm 2 formalizes the process of removing redundant moves in the pseudo-code. The worst case time complexity is $O(|\Phi|^4)$, the space complexity is $O(|\Phi|)$.

Definition 5 *(long sequence)*. Let S_P^t be a set of vertices occupied by pebbles at time step t. A sequence of moves $[k_{i_j}: u_{i_j} \rightarrow v_{i_j} | j = 1,2,\ldots,l]$, where $I = [i_j \in \{1,2,\ldots,\xi-2\} | j = 1,2,\ldots,l]$ is an increasing sequence of indices, is called *long* if $|\{k_{i_j} | j = 1,2,\ldots,l\}| = 1$ and there exists a path $C = [c_1 = u_{i_1}, c_2, \ldots, c_n = v_{i_l}]$ in G such that $n < l$, $C \cap S_P^{i_1} = \emptyset$, and for all the moves $k_\iota: u_\iota \rightarrow v_\iota$ with $i_1 < \iota < i_l \wedge \iota \notin I$ it holds that $k_\iota \neq k_{i_1} \Rightarrow \{u_\iota, v_\iota\} \cap C = \emptyset$. □

The concept of long sequence is a generalization of redundant sequence (the path C is empty in the case of redundant sequence). Intuitively, the long sequence can be replaced by a sequence of moves along a shorter path (cutoff path) into which other pebbles do not enter between the beginning and the end of the sequence. Replacing a long sequence of moves by a sequence of moves along the path C again preserves validity of the solution. Again, the replacement of long sequences must be performed repeatedly since new long sequences may arise.

Algorithm 2. *Elimination of redundant moves.*

function *Erase-Redundant-Moves* (Φ): **sequence**
do
    $\eta \leftarrow$*Find-Redundant-Moves*(Φ)
    $\Phi \leftarrow \Phi - \eta$
while $\eta \neq \emptyset$
return Φ

function *Find-Redundant-Moves* (Φ): **sequence**
let $[k_1: u_1 \rightarrow v_1, \dots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
for $i = 1,2, \dots, \xi - 2$ **do** {beginning of redundant sequence}
    for $j = \xi - 1, \xi - 2, \dots, i + 1$ **do**
    {end of redundant sequence}
        if $k_i = k_j \wedge u_i = v_j$ **then**
            $\eta \leftarrow \emptyset$ {redundant sequence}
            for $\tau = i, i + 1, \dots, j$ **do**
                if $k_i = k_\tau$ **then** $\eta \leftarrow \eta \cup \{k_\tau: u_\tau \rightarrow v_\tau\}$
            if *Check-Redundant-Moves*(Φ, i, j) **then return** η
return $\emptyset$

function *Check-Redundant-Moves* (Φ, i, j): **boolean**
let $[k_1: u_1 \rightarrow v_1, \dots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
for $\iota = i + 1, i + 2, \dots, j - 1$ **do**
    if $k_\iota \neq k_i \wedge u_i \in \{u_\iota, v_\iota\}$ **then return** $False$
return $True$

The process of replacement is formally expressed as Algorithm 3. The worst case time complexity is $O(|\Phi|^4 + |\Phi|^3|V|^2)$; the space complexity is $O(|\Phi| + |V| + |E|)$.

Redundant moves and long sequences were described manually using the *GraphRec* software. Without the visualization software we would be unable to discover them.

Notice alos that the gradual generalization was adopted in the description of redundancies. Although long sequences subsume both less general redundancies, it is not advisable to apply their replacement directly. It is better to apply elimination of redundancies step-wise from the less general one to more general ones. The reason for this practice is the increasing time complexity of redundancy elimination algorithms. A sequence of moves submitted to the more complex algorithm is potentially shortened by eliminating less general redundancies by following this practice.

Algorithm 3. Replacement of long sequences.

function *Replace-Long-Moves* (Φ, G): **sequence**
do
  $(\eta, \pi) \leftarrow$*FindLongMoves*(Φ, G)
  $\Phi \leftarrow \Phi - \eta$; $\Phi \leftarrow \Phi \cup \pi$
while $(\eta, \pi) \neq (\emptyset, [])$
return Φ

function *Find-Long-Moves* (Φ, G): **pair**
let $\left[k_1\!: u_1 \rightarrow v_1, \ldots, k_{\xi-1}\!: u_{\xi-1} \rightarrow v_{\xi-1}\right] = \Phi$
for $i = 1,2, \ldots, \xi - 2$ **do**
  for $j = \xi - 1, \xi - 2, \ldots, i + 1$ **do**
    if $k_i = k_j$ **then**
      $\eta \leftarrow \emptyset$
      for $\tau = i, i + 1, \ldots, j$ **do**
        if $k_i = k_\tau$ **then** $\eta \leftarrow \eta \cup \{k_\tau\!: u_\tau \rightarrow v_\tau\}$
      $C \leftarrow$*Check-Long-Moves*$(\Phi, i, j, |\eta|, G)$
      if $C \neq []$ **then**
        let $[c_1, c_2, \ldots, c_n] = C$
        $\pi \leftarrow [k_i\!: c_1 \rightarrow c_2, \ldots, k_i\!: c_{n-1} \rightarrow c_n]$
        return (η, π)
return $(\emptyset, [])$

function *Check-Long-Moves* $(\Phi, i, j, l, G = (V, E))$: **sequence**
let $\left[k_1\!: u_1 \rightarrow v_1, \ldots, k_{\xi-1}\!: u_{\xi-1} \rightarrow v_{\xi-1}\right] = \Phi$
$(V', E') \leftarrow G$; $V' \leftarrow V' - S_P^i$; $E' \leftarrow E' \cap \{\{u, v\} | u, v \in V'\}$
for $\iota = i + 1, i + 2, \ldots, j - 1$ **do**
  if $k_\iota \neq k_i$ **then**
    $V' \leftarrow V' - \{u_\iota, v_\iota\}$; $E' \leftarrow E' \cap \{\{u, v\} | u, v \in V'\}$
let C be a shortest path between u_i and v_j in $G' = (V', E')$
if C is defined **and** $|C| < l$ **then return** C
return $[]$

5. Experimental Evaluation

An experimental evaluation was made with above three suggested methods for redundancy elimination. Algorithms 1, 2, and 3 were implemented in C++ and were tested on a set of benchmark instances of PMG. Solutions found by the *BIBOX-θ* [11] algorithm on these benchmark instances were submitted to redundancy elimination methods.

Several characteristics of redundancy elimination were evaluated: the reduction of the total number of moves within solutions, parallel makespan, average parallelism, and runtime were measured. The implementation of redundancy elimination algorithms almost exactly follows the pseudo-code given in the previous section.

It was always the case that solution was processed by the less general redundancy elimination before it was submitted to more general and more sophisticated one. This measure ensures that the more time consuming algorithms obtains already processed solution for which there is a chance to be significantly shorter. The complete source code to allow reproducibility of all the experiments presented in this paper and raw experimental data are provided at the website: http://ktiml.mff.cuni.cz/~surynek/research/j-redundancy-2012.

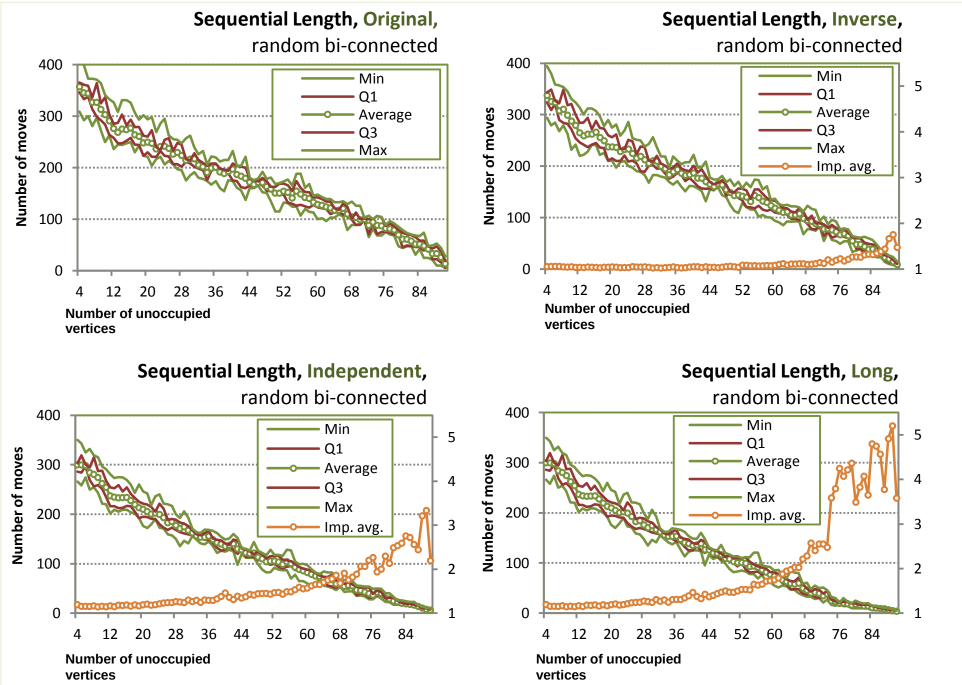


Fig. 2. *Sequential **length distribution** on random bi-connected graphs.* A collection of 10 graphs consisting of 90 vertices with length of handles ranging uniformly between 2 and 8 were generated for each number of unoccupied vertices. Minimum, maximum, average, first quartile, and third quartile out of sequential solution lengths of random instances over graphs from the collection are shown. The above characteristics of the solution length distribution are shown for original solutions as well as for solutions after removal of redundancies by the selected technique. The average improvement of solution is shown too in the same chart. It is possible to observe that solution lengths are distributed in a relatively narrow zone around the average length (approximately $\pm 10\%$ of the average length). The zone tends to narrow yet more for more sophisticated redundancy elimination.

Two structurally different sets of instances of the problem of PMG were tested. The first set of problems consists of randomly generated bi-connected graphs with approximately 90 vertices. The initial and the goal arrangement of pebbles were generated as a random permutation. The construction of the random bi-connected graphs exploits the construction that starts with a cycle followed by a gradual addition of handles to the currently constructed graph. Specifically, graphs were constructed by adding handles of random length (uniform distribution from interval $2..8$) to the initial cycle of length 7. Tests were done with a collection of 10 different random bi-connected graphs of the above setup.

The second set of testing instances consists of a grid of the size 8×8 where the initial and the goal arrangement of pebbles were again random permutations.

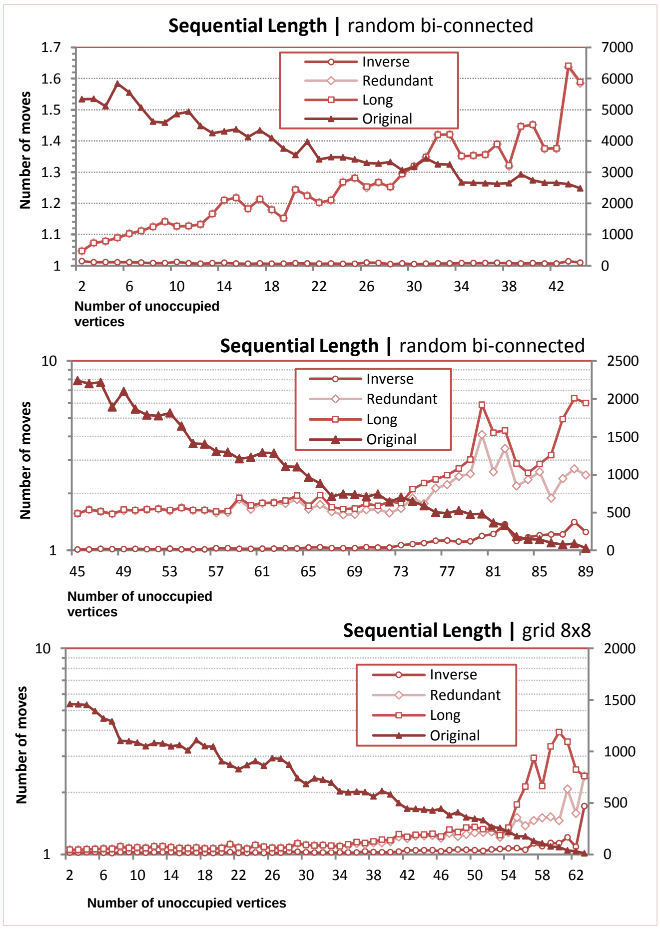


Fig. 3. ***Solution length*** *improvement on random bi-connected graph and 8×8 grid.* The total number of moves of the original solution and improvement ratio after applying redundancy elimination techniques are shown. As the number of unoccupied vertices grows the better improvements can be achieved. Up to 5 times smaller solutions can be obtained.

The series of results presented in Fig. 2 are devoted to an evaluation of the distribution of the total number of moves within the solution on random bi-connected graphs. All the three redundancy elimination methods were evaluated in this test. The solution length is shown in the dependence on the number of unoccupied vertices which ranged from 4 to 89. The following characteristics calculated out of solution lengths for instances over the mentioned collection of 10 graphs are shown for each number of unoccupied vertices: maximum, minimum, first quartile, third quartile, and average length.

It can be observed from results in Fig. 2 that the sequential solution lengths tend to be close to the average solution length; more precisely they are in the zone of approximately ±10% around the average length from which it can be concluded that the original *BIBOX-θ* and redundancy elimination techniques have a stable behavior.

To keep the results readable the remaining results are presented for a single bi-connected graph only – one of those 10 randomly generated bi-connected graphs was chosen.

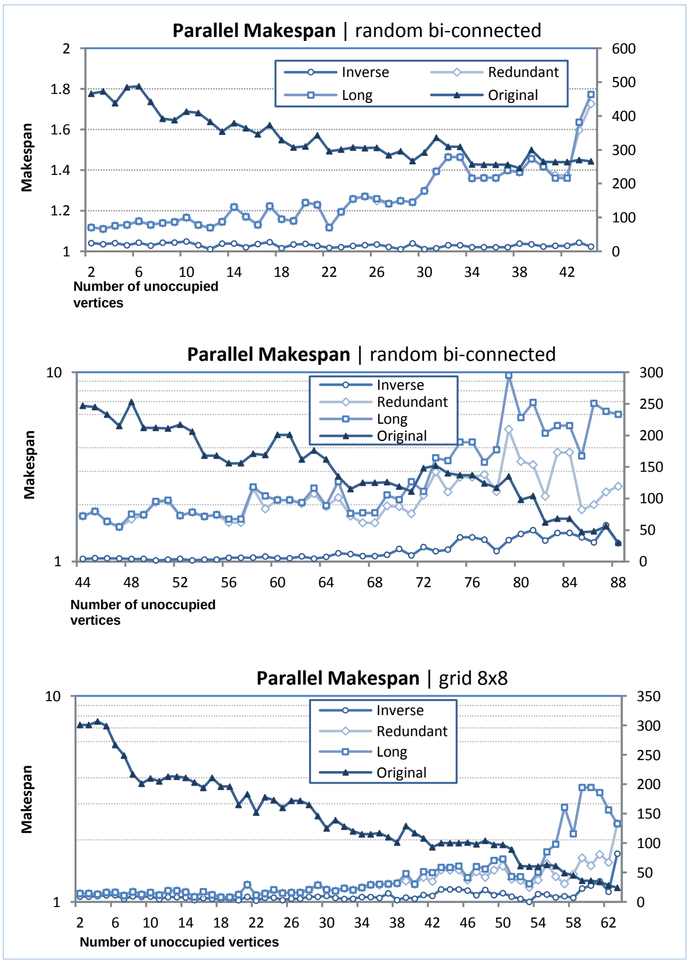


Fig. 4. *Parallel **makespan** improvement.* Redundancy elimination has even better effect on the makespan than on the size of the solution. Removal of redundancies allows more efficient increasing of the parallelism. Up to 10 times shorter solutions can be obtained on bi-connected graphs.

The reduction of the total number of moves within the solution depending on the increasing number of unoccupied vertices is shown in Fig. 3. It can be observed from Fig. 3 together with Fig. 2 that up to 5 times smaller solution can be obtained by applying redundancy elimination. The most expensive elimination of long sequences is beneficial when there is approximately 70% and more unoccupied vertices.

Results regarding the effect of redundancy elimination on parallel makespan are shown in Fig. 4. These results correlate well with the total number of moves while the improvement is slightly better for the makespan.

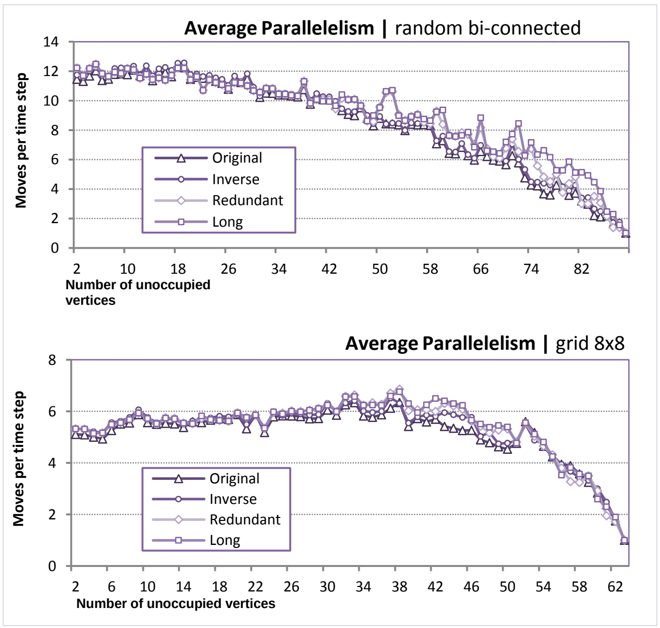


Fig. 5. *Average parallelism (average number of mover per time step).* The redundancy elimination leads to increasing of the parallelism most significantly when there is 50% to 90% of unoccupied vertices in the graph.

This observation is further quantified in Fig. 5. where the dependence of the average parallelism (which is defined as the total number of moves divided by the makespan) on the number of unoccupied vertices is shown. It can be observed that redundancy elimination typically leads to a slight increase in the average parallelism.

Results regarding runtime on a testing machine are summarized in Fig. 6. Expectably, the runtime consumed to eliminate long sequences is highest while it is still reasonable for an offline post-processing. Eliminating inverse moves and redundant sequences is relatively cheap so they can be used as an on-line post-processing tool.

The last part of the results presented in Fig. 7 is devoted to an investigation of step parallelism – that is, the number of moves performed simultaneously at the individual time steps. A single random bi-connected graph used in previous tests is presented here as well. There were 60 vertices out of 90 unoccupied. Although it is difficult to make any analysis of such results, one aspect is quite apparent from presented results – it can be observed that the qualitatively most significant change occurs when the elimination of redundant moves is used (this observation has been done also on other graphs and setups which are not presented here). On the other hand, the change obtained by applying elimination of inverse moves on the original solution as well as the change obtained by eliminating long sequences of moves from the solution which is already free of redundant moves is relatively little.

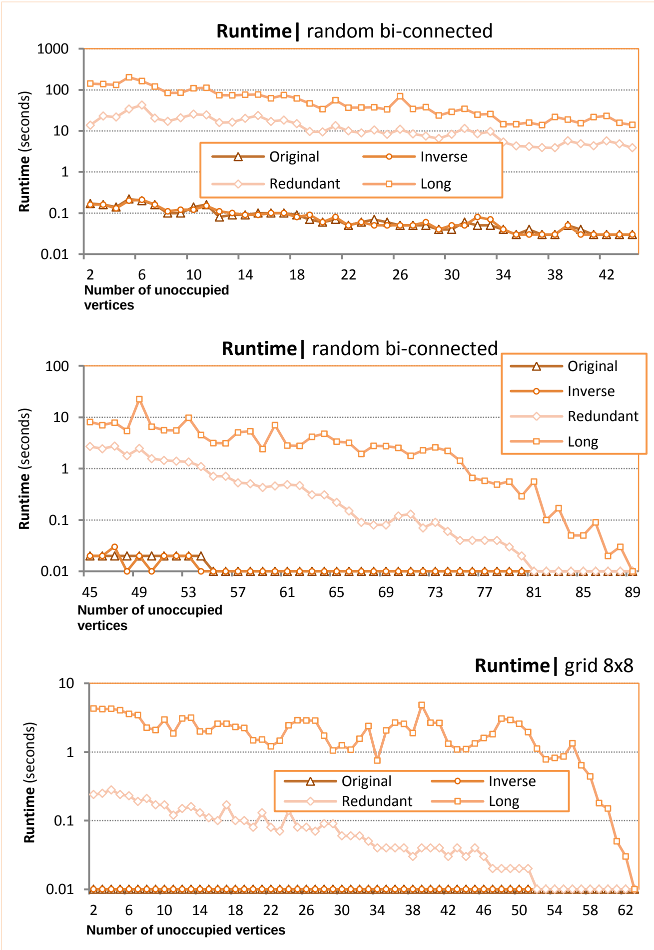


Fig. 6. *Runtime necessary for eliminating redundancies.* Eliminating long sequences is computationally the most costly (test were run on an Pentium 4, 2.4GHz, 512MB RAM, under Mandriva Linux 10.1, 32-bit edition).

It is possible to conclude that the solution can be improved by up to the order of magnitude in the measured characteristics for both types of tested graphs.

Removal of redundant sequences represents the best trade-off between detection cost and solution improvement according to performed experiments. Whereas eliminating inverse moves or long sequences feature extreme situations; the former brings almost no improvement; the latter seems to be computationally too costly for an on-line post-processing.

An expectable result is that the better improvement of solutions is gained when there are more unoccupied vertices in the input graph. Notice that definitions of redundancies are based on the mutual non-interfering of motions of pebbles. The more unoccupied space is available in the graph the less interference between moves of pebbles is possible.

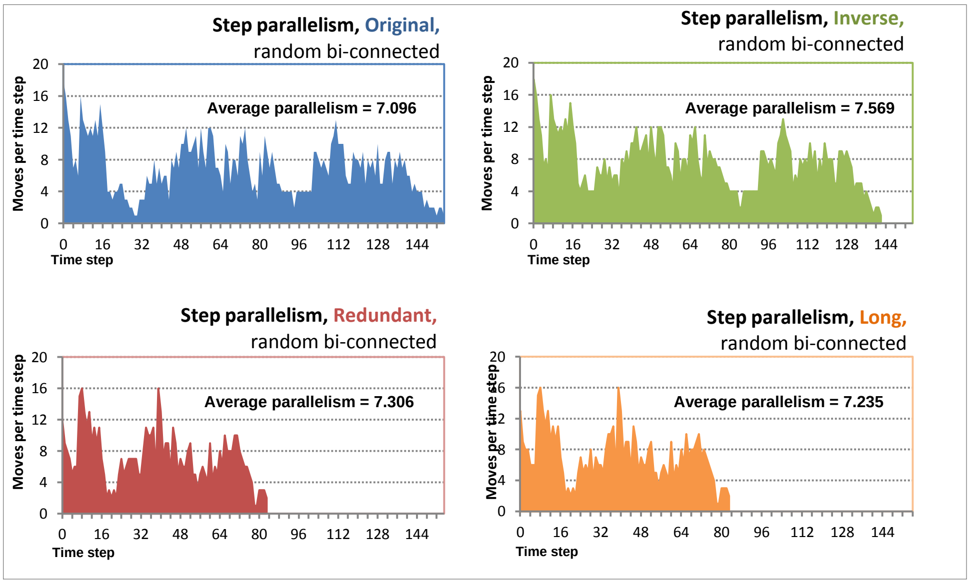


Fig. 7. *Step parallelism on random bi-connected graph.* The graph consists of 90 vertices and 60 of them are unoccupied. The length of handles was uniformly generated from the range $2..10$ - the same setup as in other experiments. Number of moves in the individual time steps is shown.

6. SAT Based Solution Improvements: An Overview

Our novel solution optimization technique called COBOPT employs SAT solving technology [1] to optimize the solution with respect to the makespan. The technique has been suggested in [12]. To be able to use SAT solvers in this way we need to obtain some (sub-optimal) solution to the PMG instance first. Let this initial solution be called a *base solution*. In this regard we used the same original solution as the base solution as in the case of redundancy elimination methods.

The crucial building block for using SAT solving technology is an encoding of motion coordination instance as an instance of propositional satisfiability. That is, we need to build a propositional formula such that it is satisfiable if and only if a solution of a certain makespan to the given motion coordination instance exists. Suppose that we are given makespan ξ. We model the arrangements of pebbles at every time step $1,2,\ldots,\xi$ where the arrangement at time step 1 is equal to the initial state and the arrangement at time step ξ is equal to the goal state. The individual arrangement consists of vectors of propositional variables for each vertex of G such that it tells us what pebble is located in the given vertex. Constraints to enforce valid transitions between consecutive time steps are also added. This encoding will be referred to as an *inverse encoding* in experiments.

Having such a propositional formula we are able to solve the given *solvable* PMG problem optimally with respect to the makespan. This is done by asking if a solution of some makespan ξ exists, where ξ is selected according to some search strategy. This asking strategy may be based for example on binary search – actually this is a strategy we use.

Notice that it is not possible to check that there is no solution to the PMG instance using this technique. However, as we use the technique to replace sub-optimal sub-solutions in the already constructed base solution we always know that the instance is solvable.

Algorithm 4. COBOPT: SAT-based PMG solution optimization – basic scheme based on binary search.

function *COBOPT-Optimize-Motion-Coordination-Plan* $(G, \vec{s}, k^+)$: **solution**
$\vec{s}_+ \leftarrow \vec{s}$
do
    $\vec{s}_- \leftarrow \vec{s}_+$
    let $\vec{s}_- = [S_P^1, \dots, S_P^m]$
    $t \leftarrow 0$; $\vec{s}_+ \leftarrow []$
    while $t < m$ **do**
        $t^+ \leftarrow$ *Find-Last-Reachable-Arrangement*$(G, S_P^t, \vec{s}_-, k^+)$
        $\vec{s}_+ \leftarrow \vec{s}_+$.*Compute-Optimal-Solution*$(G, S_P^t, S_P^{t^+})$
        $t \leftarrow t^+$
while $|\vec{s}_-| > |\vec{s}_+|$
return $\vec{s}_+$

function *Find-Last-Reachable-Arrangement* $(G, S_P^t, \vec{s}, k^+)$: **integer**
let $\vec{s} = [S_P^1, \dots, S_P^m]$
$l \leftarrow t$; $u \leftarrow m + 1$
while $u - l > 1$ **do**
$r \leftarrow (u + l)/2$
    $k \leftarrow \min(m - t, k^+)$
    $\Xi \leftarrow$ *Encode*(G, S_P^t, S_P^r, k)
    if *Solve-SAT* (Ξ) **then** $l \leftarrow r$
    else $u \leftarrow r$
return l

After producing a base solution, this is submitted to a SAT based optimization process. A maximum bound k^+ for encoding coordination instances is specified. Then subsequences in the base solution are replaced with computed optimal sub-solution. Suppose that we are currently optimizing at time step t. It is computed what is the largest $t^+ > t$ such that the time step t^+ can be reached from the time step t with no more than k^+ steps. Then sub-solution of the base solution from the time step t to t^+ is replaced by the optimal one obtained from the SAT solver. The process then continues with optimization at time step t^+ until the whole base solution is processed.

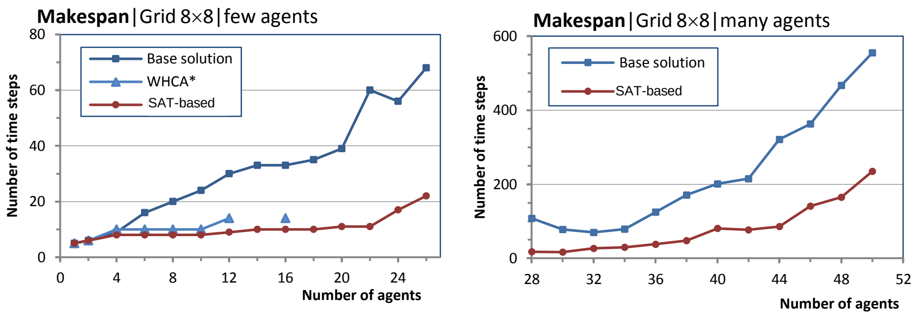


Fig. 8. *Makespan comparison on the 8×8 grid.* Optimal solutions for up to 22 and 30 agents can be found by SAT based optimization. Only up to 16 agents can be solved sub-optimally by WHCA*. The timeout for SAT based optimization was 3600 seconds.

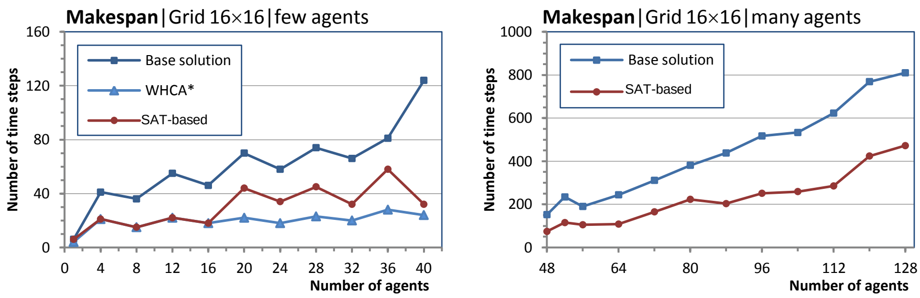


Fig. 9. *Makespan comparison on the 16×16 grid.* Optimal solutions for up to 40 agents can be found by SAT-based optimization; in the same range WHCA* can find near optimal solution as well. The timeout for SAT based optimization was 3600 seconds.

The optimization process can be iterated by taking new solution as the base one until a fixed point is reached. The binary search is employed to find t^+ and the optimal sub-solution in order to reduce the number of SAT solver invocations – see Algorithm 4**.** which summarizes basic COBOPT optimization method formally.

Notice that separation points in the base solution are selected on the greedy basis – optimization always continues on the first not yet processed time step. We also considered optimizing placement of separation point by dynamic programming techniques. This approach generates slightly better base solution decomposition. However it is at the great expense in overall runtime as many more invocation of the SAT solver are necessary.

In the experimental evaluation with SAT based optimization of solutions we also made comparison with the WHCA* algorithm [9] that is known to generate solutions that have makespan near to the optimum. WHCA* is however not able to tackle instances with environments densely occupied by agents.

Results showing comparison of the SAT-based optimization with respect to the base solution as well as with respect to WHCA* on 4-connected grids are shown in Fig. 8 and Fig. 9. The time limit for optimization was set to 3600 seconds. The process either found an optimal solution or the time limit was reached. It can be observed that SAT based optimization generates better solutions than WHCA*. Optimal solutions were obtained in cases with few agents.

If we compare SAT-based optimization with redundancy elimination methods it can be stated that SAT-based optimization is more general. It is able to discover a redundancy of a priori unknown type. On the other SAT based optimization is more time consuming which makes it suitable for off-line solving of the problem only while redundancy eliminations can be used on-line. Lot of improvements in the makespan when SAT based optimization is used comes from increasing parallelism – more moves are performed per single time step. It may happen that even though makespan of the solution has been improved the number of moves within the solution may increase.

7. Summary, Conclusions, and Future Work

This work addressed the quality (makespan) of solutions of problems motion coordination. Particularly, solutions generated by the existing algorithm *BIBOX-θ* for the given class of the problem were analyzed with respect to the presence of certain type of redundancies. Our hypothesis was that there exist certain types redundancies in generated solutions while we were not aware how do they look like.

A special visualization tool *GraphRec* was used for analyzing solutions produced by the *BIBOX-θ* algorithm. This tool allowed automating two tasks that cannot be made manually – proper drawing of a graph which a given instance consists of and visualizing moves of entities over this graph. The tool eventually confirmed that redundancies really exist and it was possible to propose their formal description.

Several types of redundancies were defined and methods for their elimination were proposed. To justify quality of our proposal an extensive experimental evaluation of proposed methods was performed on the number of different problem setups. It eventually confirmed that solutions can be improved by up to the order of magnitude using the suggested methods. The secondary finding is that the better improvement can be gained for problems with higher number of unoccupied vertices.

As a next step in solution improvements we suggest to employ SAT solving technology. A propositional formula satisfiable if and only if a given instance of motion coordination problem is solvable within the given makespan is constructed. Such a formula allows asking what is the makespan optimal replacement for a given sub-solution of an existing solution. The solution improvement process then repeatedly replaces sub-solutions by optimal ones until time limit is reached or the makespan optimal solution is found.

The SAT based technique generates high quality solutions with respect to the makespan however it is very time consuming. Thus it is more suitable for off-line improvements of solutions. On the other hand redundancy elimination methods are fast enough and can be used on-line.

References

1. N. Eén, N. Sörensson, "An Extensible SAT-solver," Proceedings of Theory and Applications of Satisfiability Testing (SAT 2003), pp. 502-518, LNCS 2919, Springer, 2004.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Introduction to Algorithms (Second edition)," MIT Press and McGraw-Hill, 2001, ISBN 0-262-03293-7.
3. A. Kishimoto, N. R. Sturtevant, "Optimized algorithms for multi-agent routing," Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Volume 3, IFAAMAS 2008, pp. 1585-1588.
4. D. Kornhauser, G. L. Miller, P. G. Spirakis, "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), IEEE Press, 1984, pp. 241-250.
5. P. Koupý, "GraphRec - a visualization tool for entity movement on graph," Student project web page, http://www.koupy.net/graphrec.php, 2011, (January 2011).
6. I. Parberry, "A real-time algorithm for the (nˆ2-1)-puzzle," Information Processing Letters, Volume 56 (1), pp. 23-28, Elsevier, 1995.
7. D. Ratner and M. K. Warmuth, "Finding a Shortest Solution for the N×N Extension of the 15-PUZZLE Is Intractable," Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), Morgan Kaufmann Publishers, 1986, pp. 168-172.
8. M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," Journal of Artificial Intelligence Research (JAIR), Volume 31, (January 2008), AAAI Press, 2008, pp. 497-542.
9. D. Silver, "Cooperative Pathfinding," Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), AAAI Press, 2005, pp. 117-122.
10. P. Surynek, "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), IEEE Press, 2009, pp. 3613-3619.
11. P. Surynek, "An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning," Proceedings of the 21st International Conference on Tools with Artificial Intelligence (ICTAI 2009), IEEE Press, 2009, pp. 151-158.
12. P. Surynek, "Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving", Proceedings of PRICAI 2012, LNCS 7458, Springer, 2012, pp. 564-576.
13. R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms," SIAM Journal on Computing, Volume 1 (2), pp. 146-160, Society for Industrial and Applied Mathematics, 1972.
14. K. C. Wang and A. Botea, "Tractable Multi-Agent Path Planning on Grid Maps," Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), IJCAI Conference, 2009, pp. 1870-1875.
15. J. Westbrook, R. E. Tarjan, "Maintaining bridge-connected and bi-connected components on-line," Algorithmica, Volume 7, Number 5&6, pp. 433–464, Springer, 1992.
16. R. M. Wilson, "Graph Puzzles, Homotopy, and the Alternating Group," Journal of Combinatorial Theory, Ser. B 16, Elsevier, 1974, pp. 86-96.