On Propositional Encodings of Cooperative Path-finding

Pavel Surynek [1,2]
[1] Charles University in Prague, Malostranské náměstí 25, Praha, 118 00, Czech Republic
[2] Kobe University, 5-1-1 Fukae-minamimachi, Higashinada-ku, Kobe 658-0022, Japan
e-mail: pavel.surynek@mff.cuni.cz

***Abstract*— The approach to solving cooperative-path finding (CPF) as propositional satisfiability (SAT) is revisited in this paper. An alternative encoding that exploits multi-valued state variables representing locations where a given agent resides is suggested. This encoding employs the ALL-DIFFERENT constraint to model the requirement that agents must not collide with each other. The use of suggested state variables also allowed us to incorporate certain heuristic reasoning to reduce the size of the propositional encoding of the problem. We show that our new domain-dependent encoding enables finding of optimal or near optimal solutions to CPFs in certain hard setups where A*-based techniques such as WHCA* fail to do so. Our finding is also that the ALL-DIFFERENT encoding can be solved faster than the existent encoding.**

Keywords-cooperative path-finding; propositional satisfiability (SAT); all-different constraint

I. INTRODUCTION, CONTEXT, AND MOTIVATION

THE problem of *cooperative path-finding* (CPF) [13] consists in finding non-colliding spatial-temporal paths for agents that need to relocate themselves from given *initial* locations to given *goal* locations. A generally adopted abstraction is that the environment is modeled as an undirected graph with agents placed in its vertices. At most one agent is placed in a vertex and at least one vertex remains unoccupied to allow agents to move. The move is possible along an edge into a currently unoccupied vertex (an example instance of CPF on a 4-connected grid is shown in Fig. 1).

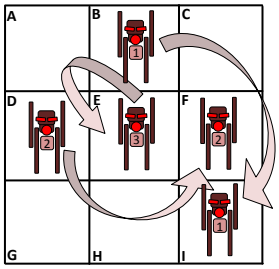

Fig. 1. *An instance of CPF.* Three agents need to relocate in the 4-connected grid 3×3.

The problem attracts considerable attention as there are many real-life situations that can be modeled as CPFs – *container relocation*, *traffic optimization*, or *ship avoidance* to name some. No less important are theoretical challenges that the problem offers. Although CPF has been studied for a long time, several important breakthroughs in its solving have been made recently. Here we are particularly interested in the quality of makespan of the resulting solution which is the total number of time steps needed for its execution. Thus, related works are referred with this regard. A real-time makespan sub-optimal incomplete algorithm WHCA* (A*-based) was published by Silver [13]. It actually became the reference algorithm in the computer entertainment industry (unit movement in RTS games). Several scalable complete algorithms for solving CPF sub-optimally have appeared recently – BIBOX [15] and PUSH-SWAP [7] represent two most important.

The problem has been attacked from the other side as well. A technique for solving CPF optimally in the case of sparsely populated environments called ID+OD has been described in [14]. Several other methods exploiting structural properties of the problem appeared in [12] and [18]. In the former case, graph modeling the environment is decomposed into simpler subgraphs; in the latter case, spatial properties of the current arrangement of agents are exploited.

In our work we addressed the case of **near optimal makespan** and **densely populated** environments, which has not yet been addressed. We employ the SAT solving technology [3] to optimize the makespan of solutions generated by existent fast sub-optimal techniques such as BIBOX or PUSH-SWAP. In contrast to the approach adopted in *domain independent* SAT-based planning [4], [5] we do not encode the whole problem as a SAT instance but only sub-problems represented by subsequences of the original solution are encoded. These (sub-optimal) sub-solutions are subsequently replaced by optimal ones found by the SAT solver. In addition, we propose two compact domain dependent encodings for CPFs – called INVERSE and ALL-DIFFERENT encoding – as alternatives to domain independent encodings used in SAT-based planning. The INVERSE encoding represents an improvement of the SAT encoding of CPFs introduced in [16]. The difference consists in more efficient translation of implication constraints over integer variables into propositional version. The ALL-DIFFERENT encoding together with its heuristic enhancement has been designed from scratch within this work. We also propose a new variant of the solution optimization process which first variant appeared in [16]. The new variant called iCOBOPT improves the previous one by adapting certain parameters.

The **organization** of the paper is that CPF is introduced formally first. The INVERSE and the ALL-DIFFERENT encodings of CPF are defined afterwards. Then a section is devot-

This research is work is supported by the Czech Science Foundation under the contract number GAP103/10/1287 and by the Japan Society for the Promotion of Science under the contract number P11743.

ed to the description of the iCOBOPT solution optimization process. Finally, we present an experimental analysis of our techniques against SAT-based planners SATPLAN and SASE. A comparison with WHCA* is also made.

II. COOPERATIVE PATH-FINDING (CPF) FORMALLY

An arbitrary **undirected graph** can be used to model the environment where agents are moving. Let $G = (V, E)$ be such a graph where $V = \{v_1, v_2, \dots, v_n\}$ is a finite set of vertices and $E \subseteq \binom{V}{2}$ is a set of edges. The placement of agents in the environment is modeled by assigning them vertices of the graph. Let $A = \{a_1, a_2, \dots, a_\mu\}$ be a finite set of *agents*. Then, an arrangement of agents in vertices of graph G will be fully described by a *location* function $\alpha: A \longrightarrow V$; the interpretation is that an agent $a \in A$ is located in a vertex $\alpha(a)$. At most **one agent** can be located in each vertex; that is α is uniquely invertible. A generalized inverse of α denoted as $\alpha^{-1}: V \longrightarrow A \cup \{\perp\}$ will provide us an agent located in a given vertex or $\perp$ if the vertex is empty.

Definition 1 (COOPERATIVE PATH FINDING). An instance of *cooperative path-finding* problem is a quadruple $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ where location functions α_0 and α_+ define the initial and the goal arrangement of a set of agents A in G respectively. □

The dynamicity of the model supposes a discrete time divided into time steps. An arrangement α_i at the i-th time step can be transformed by a transition action which instantaneously moves agents in the non-colliding way to form a new arrangement α_{i+1}. The resulting arrangement α_{i+1} must satisfy the following *validity conditions*:

(i) $\forall a \in A$ either $\alpha_i(a) = \alpha_{i+1}(a)$ or $\{\alpha_i(a), \alpha_{i+1}(a)\} \in E$ holds (agents move along edges or not move at all),

(ii) $\forall a \in A \;\; \alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow \alpha_i^{-1}(a) = \perp$ (agents move to vacant vertices only), and

(iii) $\forall a, b \in A \;\; a \neq b \Rightarrow \alpha_{i+1}(a) \neq \alpha_{i+1}(b)$ (no two agents enter the same target/unique invertibility of resulting arrangement).

The task in cooperative path finding is to transform α_0 using above valid transitions to α_+.

Definition 2 (SOLUTION, MAKESPAN). A *solution* of a *makespan* m to a cooperative path finding instance $\Sigma = [G, A, \alpha_0, \alpha_+]$ is a sequence of arrangements $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m]$ where $\alpha_m = \alpha_+$ and α_{i+1} is a result of valid transformation of α_i for every $= 1,2, \dots, m-1$. □

A notation $|\vec{s}|$ will be also used to denote the makespan. If it is a question whether there is a solution of Σ of the makespan at most a given bound we are speaking about a *bounded variant (bCPF)*. It is known that bCPF is NP-complete and finding makespan optimal solution to CPF is NP-hard [9]. Notice that due to no-ops introduced in valid transitions it is equivalent to finding a solution of the makespan equal to the given bound.

III. CPF AS PROPOSITIONAL SATISFIABILITY

To enable solving of CPF as satisfiability we needed to develop more compact SAT encodings than those represented by SAT-based domain-independent planning systems like SATPLAN [5] or SASE [4]. Our advantage here was the knowledge of the domain, which allowed us to make important design decisions.

In both encodings, we followed the classical *Graphplan* style [5] as for we also encode state of the planning world at each time step. The candidates for *multi-valued state variables* are clearly the location function and its inverse. It is known that each multi-valued state variable can be encoded by the logarithmic number of propositional variables with respect to the number of its values [11] which we also follow in our design. Location function tells us in what vertex a given agent is located while the inverse location tells us what agent is located in a given vertex. This is a substantial difference and the choice of state variables determines the overall design of constraints that will encode validity conditions of CPF.

A. INVERSE Encoding

If states of the planning world – that is, arrangements of agents – are represented using inverse locations we need to add other multi-valued state variables that will encode state transitions with regard on validity conditions. There are two *primitive actions* for each edge adjacent to the given vertex plus one no-op action. Half of the primitive actions corresponding to the vertex are reserved for incoming agents while the other half is for outgoing agents. If the outgoing primitive action is selected it is necessary to propagate the selection as corresponding selection of incoming primitive action in the target vertex; and vice versa. Representing the selection of the primitive action as a multi-values state variable automatically ensures that conditions (i) and (iii) are encoded. No other constraint is necessary. Notice also, that the degree of vertices in G is typically low for real-life environments, thus the action selection in the vertex can be captured by few propositional variables.

Let $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ be an instance of CPF and $k \in \mathbb{N}$ be a makespan bound (= bCPF altogether). The INVERSE encoding has layers $0,1, \dots, k$. Suppose that neighboring vertices of a given vertex are ordered in the fixed order. That is, $\forall v \in V$ we have a function $\sigma_v: \{u | \{v, u\} \in E\} \longrightarrow \{1,2, \dots, \deg_G(v)\}$ and its inverse σ_v^{-1}.

Definition 3 (REGULAR LAYER – INVERSE ENCODING). The i-th layer of the *INVERSE encoding* consists of the following finite domain integer **state variables**:

- $\mathcal{A}_i^v \in \{0,1,2, \dots, \mu\}$ for all $v \in V$ such that $\mathcal{A}_i^v = j$ iff $\alpha_i(a_j) = v$
- $\mathcal{T}_i^v \in \{0,1,2, \dots, 2\deg_G(v)\}$ for all $v \in V$ such that
  - $\mathcal{T}_i^v = 0$ iff no-op was selected in v;
  - $\mathcal{T}_i^v = \sigma_v(u)$ iff an outgoing primitive action with the target $u \in V$ was selected in v;
  - $\mathcal{T}_i^v = \deg_G(v) + \sigma_v(u)$ iff an incoming primitive ac-

tion with $u \in V$ as the source was selected in v.

and **constraints**:

- $\mathcal{T}_i^v = 0 \Rightarrow \mathcal{A}_{i+1}^v = \mathcal{A}_i^v$ for all $v \in V$ (1)
  (**no-op** case);
- $0 < \mathcal{T}_i^v \leq \deg_G(v) \Rightarrow \mathcal{A}_i^u = 0 \wedge \mathcal{A}_{i+1}^u = \mathcal{A}_i^v \wedge \mathcal{T}_i^u = \sigma_u(v) + \deg_G(u)$ (2)
  where $u = o_v^{-1}(\mathcal{T}_i^v)$ for all $v \in V$
  (**outgoing** agent case);
- $\deg_G(v) < T_i^v \leq 2 * \deg_G(v) \Rightarrow \mathcal{T}_i^u = \sigma_u(v)$ (3)
  where $u = \sigma_v^{-1}(\mathcal{T}_i^v - \deg_G(v))$ for all $v \in V$
  (**incoming** agent case). □

State variables $\mathcal{A}_i^v$ represent inverse locations; that is, $\mathcal{A}_i^v$ tells us what agent is located in v at the time step i. State variables $\mathcal{T}_i^v$ represent primitive transition actions selected in vertices. The last layer of the encoding is irregular as it has inverse location state variables only. To finish the encoding of the bCPF instance we need to encode the initial and the goal arrangement straightforwardly as follows:

$$\underline{\text{Initial}}: \begin{cases} \mathcal{A}_0^v = j & \text{iff } \alpha_0^{-1}(v) = a_j, \\ \mathcal{A}_0^v = 0 & \text{iff } \alpha_0^{-1}(v) = \perp, \end{cases}$$

$$\underline{\text{Goal}}: \begin{cases} \mathcal{A}_k^v = j & \text{iff } \alpha_+^{-1}(v) = a_j, \\ \mathcal{A}_k^v = 0 & \text{iff } \alpha_+^{-1}(v) = \perp. \end{cases}$$

Transformation of the encoding from the above integer representation to the propositional one exploits standard Tseitin's hierarchical encoding [17] with auxiliary variables. Basically we need to encode implications with equality between a state variable and a constant on the left side and one or more equalities between state variables or between a state variable and a constant on the right side.

For illustration, suppose we need to encode a constraint $\mathcal{T}_i^v = j \Rightarrow \mathcal{A}_{i+1}^u = \mathcal{A}_i^v$ with $u, v \in V$ and $i, j \in \mathbb{N}$ over vectors of propositional variables that encode $\mathcal{T}_i^v$, $\mathcal{A}_i^v$, and $\mathcal{A}_{i+1}^u$ (this is, actually part of the second constraint in the layer encoding; for simplicity we do not show the whole right side of the implication). Propositional variables will be referred to using array indices to the original integer state variables.

We introduce a fresh auxiliary propositional variable a. The original constraint will then split into conjunction as follows: $\mathcal{T}_i^v = j \Rightarrow a \quad \wedge \quad a \Rightarrow \mathcal{A}_{i+1}^u = \mathcal{A}_i^v$. The first constraint in the conjunction will be encoded simply as one clause:

$$a \vee \bigvee_{\mathbb{i}=1}^{\lceil \log_2(2 \deg_G(v)+1) \rceil} \neg \text{lit}(j[\mathbb{i}], \mathcal{T}_i^v[\mathbb{i}]) \text{ where}$$

$$\text{lit}(j[\mathbb{i}], \mathcal{T}_i^v[\mathbb{i}]) = \begin{cases} \neg \mathcal{T}_i^v[\mathbb{i}] & \text{iff } j[\mathbb{i}] = 0 \\ \mathcal{T}_i^v[\mathbb{i}] & \text{iff } j[\mathbb{i}] = 1 \end{cases}$$

The second constraint will be encoded as conjunction of several ternary clauses:

$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2(|A|+1) \rceil} \neg a \vee \neg \mathcal{A}_{i+1}^u[\mathbb{i}] \vee \mathcal{A}_i^v[\mathbb{i}] \quad \wedge$$

$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2(|A|+1) \rceil} \neg a \vee \mathcal{A}_{i+1}^u[\mathbb{i}] \vee \neg \mathcal{A}_i^v[\mathbb{i}]$$

Notice that we kept up with just one auxiliary variable in this case. Encoding equality on the right side is yet easier. Let us summarize the size of the encoding in the following proposition (the proof is omitted). It is easy to observe that most of the clauses in the INVERSE encoding are either binary or ternary.

Proposition 1 (INVERSE ENCODING SIZE). *A regular layer of the* INVERSE *encoding of the instance of bCPF requires:*

$|V|\lceil \log_2 |A| \rceil + \sum_{v \in V} \lceil \log_2(2 \deg_G(v) + 1) \rceil$
propositional variables for representing ***state variables****,*

$4|E| + |V|$
auxiliary *propositional* ***variables*** *from Tseitin's translation*

$2 \sum_{v \in V} \deg_G(v) \lceil \log_2(2 \deg_G(v) + 1) \rceil +$
$2(|V| + 2|E|) \lceil \log_2(|A| + 1) \rceil$
clauses for representing ***constraints****, and*

$|V|(2^{\lceil \log_2 |A|+1 \rceil} - |A| - 1)$,
$\sum_{v \in V} 2^{\lceil \log_2(2 \deg_G(v)+1) \rceil} - 2 \sum_{v \in V} \deg_G(v) - |V|$
clauses for excluding ***unused*** *location and transition action states respectively.* ■

B. ALL-DIFFERENT Encoding and Heuristic Estimation

If location function is chosen to represent the arrangement we need to take care of ensuring validity conditions (ii) and (iii) more explicitly. An agent must move into unoccupied vertex which in this representation means that it should avoid all the vertices occupied by other agents at the current time step. This condition is modeled by pair-wise differences between involved location state variables. The situation is very close to a *bi-clique* [11] of pair-wise differences but differences between locations for the same agent at consecutive time steps are missing here.

At the same time, it is necessary that no two agents occupy the same vertex (location). This requirement can be expressed through the ALL-DIFFERENT [10] constraint involving all the location state variables at the given time step. Finally, we need to encode the condition that agents can move along edges of G only. It requires quite extensive encoding as a conditional equality needs to be added for each vertex and agent. Briefly expressed, this tells that if an agent is located in a given vertex at a given time step then it must be located in some of the neighbors or in the same vertex at the next time step. The just introduced ALL-DIFFERENT-based encoding is summarized formally in the following definition.

Definition 4 (REGULAR LAYER – ALL-DIFFERENT). The i-th layer of the *ALL-DIFFERENT encoding* consists of the following finite domain integer **state variables**:

- $\mathcal{L}_i^a \in \{0,1,2,\ldots,n\}$ for all $a \in A$
  such that $\mathcal{L}_i^a = l$ iff $\alpha_i(a) = v_l$

and the **constraints** are as follows:

- for all $a \in A$ and $l \in \{1,2,\ldots,n\}$ (4)
  $\mathcal{L}_i^a = l \Rightarrow \mathcal{L}_{i+1}^a = \ell \vee \bigvee_{\ell \in \{1,\ldots,n\} | \{v_l, v_\ell\} \in E} \mathcal{L}_{i+1}^a = \ell$
  (agents can move only **along edges** of G),

- for all $a \in A$ (5)

$$\bigwedge_{b \in A | b \neq a} \mathcal{L}_{i+1}^{a} \neq \mathcal{L}_{i}^{b}$$

(the **target vertex** of agent's move must be **empty**),

- and **at most one** agent resides in each vertex:

$$\text{AllDifferent}(\mathcal{L}_{i}^{a_1}, \mathcal{L}_{i}^{a_2}, \ldots, \mathcal{L}_{i}^{a_\mu}) \quad (6)$$

which altogether directly encodes validity conditions (i), (ii), and (iii). □

The last layer is irregular again; there is no propagation constraint to the next layer. Location state variables $\mathcal{L}_{i}^{a}$ are encoded using $\lceil \log_2 |V| \rceil$ propositional variables each; let us again refer to them through indexing. The initial and the goal state are encoded trivially as several equalities between state variables and constants.

Let us fix $a \in A$ and $l \in \{1,2,\ldots,n\}$. Suppose further that v_l has $\nu \in \mathbb{N}_0$ neighbors. We need to introduce $\nu + 2$ fresh propositional variables – say $\mathcal{b}_1, \mathcal{b}_2, \ldots, \mathcal{b}_{\nu+2}$ – to encode the constraint (4). Each new auxiliary variable is put to stand instead of the equality in the original constraint. So we have an $(\nu + 2)$-ary clause $\neg\mathcal{b}_1 \vee \bigvee_{\mathbb{i}=2}^{\nu} \mathcal{b}_{\mathbb{i}}$. To ensure correct encoding, implications between the auxiliary variables and the original equalities need to be added. For simplicity let us show the second equality between the state variable and the constant only; it is encoded by $\lceil \log_2 |V| \rceil$ binary clauses:

$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2 |V| \rceil} \neg\mathcal{b}_2 \vee \text{lit}(\ell[\mathbb{i}], \mathcal{L}_{i+1}^{a}[\mathbb{i}])$$

The inequality between two state variables is encoded using the scheme introduced in [1]. Authors use the term *bit-vectors* in the same sense as we do use vectors of propositional variables encoding a state variable. Suppose that we need to encode $\mathcal{L}_{i+1}^{a} \neq \mathcal{L}_{i}^{b}$. Now, $\lceil \log_2 |V| \rceil$ fresh propositional variables $\mathcal{d}_1, \mathcal{d}_2, \ldots, \mathcal{d}_{\lceil \log_2 |V| \rceil}$ are introduced. Each encodes inequality between the corresponding propositional variables encoding $\mathcal{L}_{i+1}^{a}$ and $\mathcal{L}_{i}^{b}$ respectively. Hence, to express inequality between original state variables we can just put single clause: $\bigvee_{\mathbb{i}=1}^{\lceil \log_2 |V| \rceil} \mathcal{d}_{\mathbb{i}}$. Again the relation of new auxiliary variables to $\mathcal{L}_{i+1}^{a}$ and $\mathcal{L}_{i}^{b}$ as $2\lceil \log_2 |V| \rceil$ ternary clauses needs to be added:

$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2 |V| \rceil} \neg\mathcal{d}_{\mathbb{i}} \vee \neg\mathcal{L}_{i+1}^{a}[\mathbb{i}] \vee \mathcal{L}_{i}^{b}[\mathbb{i}] \quad \wedge$$
$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2 |V| \rceil} \neg\mathcal{d}_{\mathbb{i}} \vee \mathcal{L}_{i+1}^{a}[\mathbb{i}] \vee \neg\mathcal{L}_{i}^{b}[\mathbb{i}]$$

To encode the ALL-DIFFERENT constraint we again follow scheme presented in [1]. That is, inequalities between all-pairs of involved state variables are encoded in the same way as above which means to encode $\binom{|A|}{2}$ inequalities. Although the size of the encoding is now more than evident, let us summarize it in the following proposition. Again, most of the clauses are either binary or ternary.

Proposition 2 (ALL-DIFFERENT ENCODING SIZE). *A regular layer of the* ALL-DIFFERENT *encoding of bCPF instance requires:*

$|A|\lceil \log_2 |V| \rceil$
propositional variables for representing ***agent's locations****,*
$2(|V| + |E|) + |A|(|A| - 1)\lceil \log_2 |V| \rceil + \binom{|A|}{2}\lceil \log_2 |V| \rceil$
auxiliary *propositional variables,*
$2(|V| + |E|)\lceil \log_2 |V| \rceil +$
$|A|(1 + (|A| - 1)\lceil \log_2 |V| \rceil) + 2\binom{|A|}{2}\lceil \log_2 |V| \rceil$
clauses to represent ***validity conditions****, and*
$|A|(2^{\lceil \log_2 |V| \rceil} - |V|)$,
clauses for excluding ***unused*** *location states.* ■

As it is usually the case that $|A| < |V|$ the ALL-DIFFERENT encoding has fewer propositional variables needed to encode state variables than the INVERSE encoding (since then it holds, that $|A|\lceil \log_2 |V| \rceil < |V|\lceil \log_2 |A| \rceil$). This difference is becoming more prominent on sparsely populated environments. On the other hand, the ALL-DIFFERENT encoding has more constraints which add many auxiliary variables and most notably the representation of the ALL-DIFFERENT constraint grows quadratically as the number of agents increases. Notice also, that all the transition actions need to be chosen even for vertices not containing any agent in the INVERSE encoding. Hence, we should expect that INVERSE encoding will be better for densely populated environments while the ALL-DIFFERENT will be better for sparsely populated ones.

Unlike in the case of INVERSE encoding the ALL-DIFFERENT encoding can be enhanced by a certain heuristic reasoning. We observed in our preliminary experiments that the most space consuming constraint is constraint (4). Thus we made an enhancement in which we do not introduce this constraint if the given location l cannot be reached by the given agent a from its initial location at the i-th timestep. Similarly, the constraint is not introduced if there is no chance for the agent a to reach its goal location in the remaining number of steps. Formally the constraint (4) is introduced if and only if the following condition holds:

$$\text{dist}_{\text{G}}(\alpha_0(a), v_l) \leq i \wedge \text{dist}_{\text{G}}\big(v_l, \alpha_+(a)\big) \leq k - i$$

To ensure the correctness of the enhancement also we need to forbid occurrence of agents in **unreachable** locations. That is, following constraints are added to the model:

- for all $a \in A$ and $l \in \{1,2,\ldots,n\}$ such that $\text{dist}_{\text{G}}(\alpha_0(a), v_l) > i \vee \text{dist}_{\text{G}}\big(v_l, \alpha_+(a)\big) > k - i$ (7)
$\mathcal{L}_{i}^{a} \neq l$

IV. SAT-BASED OPTIMIZATION OF SOLUTIONS TO CPFS

The approach of our choice to obtain solutions to CPFs of short makespans (or even optimal makespans) is not to solve the CPF instance as SAT completely but to employ a SAT solver to optimize an existent sub-optimal solution. There already exist sub-optimal complete algorithms for solving CPF in polynomial time such as BIBOX [15] or PUSH-SWAP [7] which are ready to be used in this framework.

The basic idea of the optimization process is to take a relatively short subsequence of movements in the existent solution and replace it with an optimal sub-solution obtained from the SAT solver. Our new SAT-based solution optimi-

zation scheme is called iCOBOPT. The previous version called COBOPT is described in [16]. The iCOBOPT algorithm uses more intelligent adaptation of makespan bounds which limit the number of encoded time steps while in COBOPT it was fixed by the user.

Algorithm 1. iCOBOPT – *an iterative SAT-based optimization of solutions to CPFs.* The algorithm iteratively increases the makespan bound. The binary search for optimal sub-solutions to CPFs is shown. It finishes if the timeout is reached or the overall optimum is found.

function *iCOBOPT-Optimize-Cooperative-Plan* $(\Sigma, \vec{s}, \mathbb{t}^+)$: **solution**
1: $\mathbb{t}^0 \leftarrow$*Get-Current-Time*()
2: $\vec{s}_+ \leftarrow \vec{s}$; $\mathbb{k} \leftarrow 2$
3: **do**
4: **do**
5: $\vec{s}_- \leftarrow \vec{s}_+$
6: **let** $\vec{s}_- = [\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m]$
7: $\vec{s}_+ \leftarrow []$; $\mathbb{i} \leftarrow 0$
8: **while** $t < m$ **do**
9: $\mathbb{i}^+ \leftarrow$ *Find-Last-Reachable-Arrangement*$(\Sigma, \alpha_{\mathbb{i}}, \vec{s}_-, \mathbb{k})$
10: $\vec{s}_+ \leftarrow \vec{s}_+$.*Compute-Optimal-Solution*$(\Sigma, \alpha_{\mathbb{i}}, \alpha_{\mathbb{i}^+})$
11: $\mathbb{i} \leftarrow \mathbb{i}^+$
12: **while** $|\vec{s}_-| > |\vec{s}_+|$
13: $\mathbb{t} \leftarrow$*Get-Current-Time*()
14: $\mathbb{k} \leftarrow \mathbb{k} + 1$
15: **while** $\mathbb{t} - \mathbb{t}^0 < \mathbb{t}^+$ **and** $\mathbb{k} < |\vec{s}_+|$
16: **return** $\vec{s}_+$

function *Find-Last-Reachable-Arrangement* $(\Sigma, \alpha_{\mathbb{i}}, \vec{s}, \mathbb{k})$: **integer**
17: **let** $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m]$
18: $l \leftarrow \mathbb{i}$; $u \leftarrow m + 1$
19: **while** $u - l > 1$ **do**
20: $\mathbb{j} \leftarrow (u + l)/2$
21: $k \leftarrow \min(m - t, \mathbb{k})$
22: **if** *Check-Reachability*$(\Sigma, \alpha_{\mathbb{i}}, \alpha_{\mathbb{j}}, k)$ **then**
23: $\Xi \leftarrow$*Encode*$(\Sigma, \alpha_{\mathbb{i}}, \alpha_{\mathbb{j}}, \mathbb{k})$
24: **if** *Solve-SAT* (Ξ) **then** $l \leftarrow \mathbb{j}$
25: **else** $u \leftarrow \mathbb{j}$
26: **else**
27: $u \leftarrow \mathbb{j}$
28: **return** l

function *Check-Reachability* $(\Sigma, \alpha_{\mathbb{i}}, \alpha_{\mathbb{j}}, \mathbb{k})$: **boolean**
29: **let** $\Sigma = [G, A, \alpha_0, \alpha_+]$
30: **for each** $a \in A$ **do**
31: **if** $\mathrm{dist}_G(\alpha_{\mathbb{i}}(a), \alpha_{\mathbb{j}}(a)) > \mathbb{k}$ **then return** *FALSE*
32: **return** *TRUE*

The scheme of employing SAT solvers in iCOBOPT is much more scalable than the classical SATPLAN or SASE planning since incomparably smaller SAT instances need to be solved. Here we do not compare encoding style but the scheme in which encoded instances are submitted to the SAT solver; eventually all the time steps needed to cover the optimal makespan are encoded in the SATPLAN or SASE scheme (and it may be very large in the case of CPFs). Notice, that in our approach we encode few layers representing time steps of a given sub-solution. Observe also the important fact that the linear increase in the number of layers of the encoding may cause exponential increase in the solving runtime of the SAT solver. If we simplify the situation the time needed check if there is a solution to the encoded instance with x layers needs time of $\mathcal{O}(\mathbb{b}^x)$ where $\mathbb{b} \in \mathbb{N}$ while if we divide the makespan into two parts – say x_1 and x_2 layers where $x = x_1 + x_2$ are encoded separately then the time is $\mathcal{O}(\mathbb{b}^{x_1}) + \mathcal{O}(\mathbb{b}^{x_2})$ which is exponentially smaller than $\mathcal{O}(\mathbb{b}^x)$ (of course, the question in the latter case is not equivalent to the former one). On the other hand SATPLAN and SASE schemes guarantee to find makespan optimal solutions which iCOBOPT does not guarantee. Nevertheless, iCOBOPT is capable of optimizing much larger CPFs than SATPLAN or SASE can do (even if they would use our domain-dependent encodings).

The pseudo-code of the iCOBOPT optimization is shown as Algorithm 1. Throughout the algorithm the makespan bound of $\mathbb{k}$ is used which is gradually increased. At every time step $\mathbb{i}$ of the current solution to the CPF instance Σ (which is at the beginning that obtained from the suboptimal algorithm for CPF – called a *base solution*) a maximum step $\mathbb{i}^+$, such that $\mathbb{i} < \mathbb{i}^+$ and the state (arrangement) at $\mathbb{i}^+$ can be reached from the state at $\mathbb{i}$ in $\mathbb{k}$ steps, is found. The step $\mathbb{i}^+$ is found by the *binary search* which uses multiple queries to the SAT solver. The optimization process terminates if the given timeout of $\mathbb{t}^+$ is exceeded or the makespan optimal solution to Σ is found. The process is illustrated in Fig. 2.

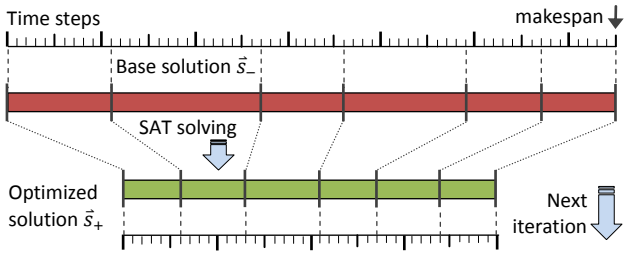

Fig. 2. *Illustration of the optimization process.* A single iteration is shown – these are repeated until a fixed point is reached.

V. EXPERIMENTAL EVALUATION

We have chosen the BIBOX algorithm to produce sub-optimal base solutions in our experimental evaluation. This choice was guided by the fact that authors of this algorithm do provide the source code and the benchmark generation suite. Additionally, according to our findings BIBOX algorithm was capable to generate the required sub-optimal solutions very quickly.

As benchmarks we have chosen environments consisting of 4-connected grids of size 8×8 and 16×16 respectively with random initial and goal arrangements of agents. Up to 78% occupancy was tested in case of the grid of size 8×8 and up to 50% occupancy was tested for the 16×16 grid.

As the SAT solver we used MINISAT 2.2 [3] as its pros and cons are well known. To evaluate the benefit of employing SAT technology in solving CPFs we also made a brief comparison with the incomplete solver WHCA* [13] which is de-facto considered to be the standard approach to solving CPFs (the window of size 16 was used in WHCA*). We also considered comparison with the ID+OD algorithm which is designed to generate optimal solutions to CPFs in sparsely populated environments. However, we found that it is not directly comparable as its main strength consists in separating agents into independent groups which do not intermix.

This contrasts with our approach where all the agents constitute one intermixing group.

To allow reproducing of all the results the source of iCOBOPT as well as experimental data is provided at: http://ktiml.mff.cuni.cz/~surynek/research/ictai2012.

A. Comparison of Encoding Sizes

We compared sizes of our INVERSE and ALL-DIFFERENT encodings with domain independent encodings of SATPLAN and SASE – results for the 8×8 grid are shown in Table 1; for the 16×16 grid in Table 2. Although it is a bit unfair comparison since our domain is fixed, it gives a nice picture of the situation.

Clearly the INVERSE encoding is very conservative regarding the number of variables and clauses – it dominates all the other encodings in this aspect. The small size is mainly due to the fact that lot of the domain knowledge is captured in the design of variables. On the other hand ALL-DIFFERENT encoding is very close to that of SASE (ALL-DIFFERENT tends to be smaller on larger instances especially in terms of the number of clauses).

Table 1. *Encoding sizes comparison on the grid* ***8×8***. The number of layers of encodings was determined as the goal level provided by SATPLAN (a step where the goal may be reachable).

|A| in the 4-connected grid **8×8**	Number of layers	SATPLAN encoding		SASE encoding		INVERSE encoding		ALL-DIFFERENT encoding	
		|Variables|	|Clauses|	|Variables|	|Clauses|	|Variables|	|Clauses|	|Variables|	|Clauses|
4	8	5864	55330	11386	53143	5400	38800	11128	54356
8	8	10022	165660	19097	105724	5920	48224	25136	114952
12	8	14471	356410	26857	168875	5920	46176	42024	181788
16	10	30157	1169198	51662	372140	8122	76192	79008	326736
24	10	43451	2473813	73101	588886	8122	71072	140400	537528
32	14	99398	8530312	157083	1385010	12396	137120	309824	1120672

The SATPLAN encoding seems to be conservative regarding the number of variables but the number of clauses is quickly blowing up (for larger number of agents in the 16×16 grid SATPLAN gave up with no memory left). Surprisingly we found, that the difference in size of the encoding generated by SATPLAN and SASE is not that huge as it is reported in [4] for other domains. Encoding sizes differ marginally for sparsely populated environments; while SATPLAN tends to have up to 6 times more clauses in densely populated environments (far from the reported 50-fold).

Table 2. *Encoding sizes comparison on the grid* ***16×16***. SATPLAN was unable to generate SAT instance for the larger number of agents, and SASE failed to proceed to the goal level. That is why the number of layers is lower than the goal level for larger numbers of agents*.

|A| in the 4-connected grid **16×16**	Number of layers	SATPLAN encoding		SASE encoding		INVERSE encoding		ALL-DIFFERENT encoding	
		|Variables|	|Clauses|	|Variables|	|Clauses|	|Variables|	|Clauses|	|Variables|	|Clauses|
4	21	69704	746562	137406	677737	60755	478462	122368	827628
8	15	65365	995507	134482	712352	46904	412416	178816	1174616
16	18	Out of memory		342100	2347456	61154	611328	469888	2928336
32	4*			288498	2716096	13672	143104	197888	1101600
40	4*			357762	3783672	13672	134912	265280	1415080
64	4*			561210	5913320	14700	189440	510464	2446912

Another important aspect which speaks in favor of domain dependent encodings like INVERSE and ALL-DIFFERENT is that a considerable time and space is necessary to generate SAT instance in SATPLAN and SASE (in case of SATPLAN huge memory consumption even hinders it from generating any output). In the case of our approach SAT, instances are generated faster than is the time to save them (no special computation is necessary while SATPLAN and SASE need to perform time consuming mutex reasoning). Notice, also that we typically use fewer layers, which is another important factor in reducing sizes of generated encodings.

B. Makespan and Runtime[1] Comparison

Experiments regarding makespan and runtime show that size of the encoding itself is not everything with regard on solving performance. Observe that each of our encodings is built in a different fashion – the INVERSE encoding is very flat while the ALL-DIFFERENT one is built more hierarchically (that is, many clauses may be switched off through auxiliary variables). Generally SAT solvers seem to be sensitive to such differences.

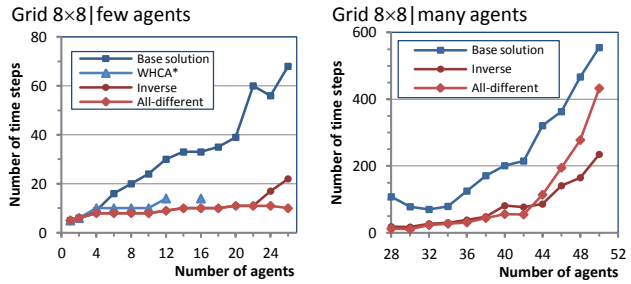

Fig. 3. ***Makespan*** *comparison on the 8×8 grid.* Optimal solutions for up to 22 and 30 agents can be found by iCOBOPT using the INVERSE and the ALL-DIFFERENT encoding respectively. Only up to 16 agents can be solved sub-optimally by WHCA*.

Makespan comparison reported in Fig. 3 and Fig. 4 shows that in the 8×8 grid, iCOBOPT is capable of generating optimal solutions for up to 30 agents. The ALL-DIFFERENT encoding tends to be better for fewer agents while it loses with respect to the INVERSE encoding on environments populated by many agents. A comparison with WHCA* shows that its incompleteness presents an unpleasant issue – it is unable to produce a solution for instances with the occupancy of environment exceeding 25% where iCOBOPT still produces optimal solutions (up to the occupancy of 47%).

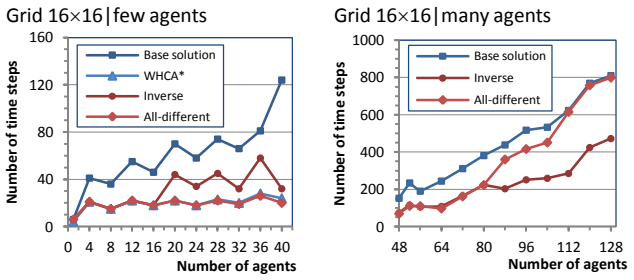

Fig. 4. ***Makespan*** *comparison on the 16×16 grid.* Optimal solutions for up to 40 agents can be found by iCOBOPT with the ALL-DIFFERENT encoding; in the same range WHCA* can find near optimal solution as well. The INVERSE encoding dominates in the range with more than 80 agents.

[1] All the runtime measurements were done on a machine with the 4-core CPU Intel i7 3.4GHz and 8GB RAM under Linux kernel 2.6.38-26.

The grid of size 16×16 represents the current limit of scalability of the iCOBOPT technique (see Fig. 5 for runtime comparison). Solutions to instances containing up to 128 agents were submitted to iCOBOPT for improvement. With the number of agents exceeding 100 the ALL-DIFFERENT encoding gives rise to the degradation so that almost no optimization gain can be obtained from it. The explanation is that if the number of variables involved in the ALL-DIFFERENT constraint exceeds certain limit the SAT solver starts to struggle over it. Again the ALL-DIFFERENT encoding dominates on cases with relatively fewer agents in the environment (optimal solutions are reached again).

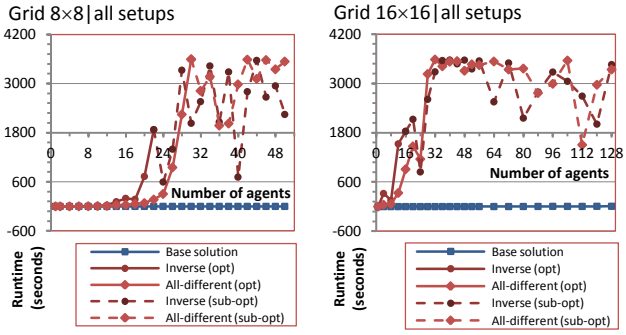

Fig. 5. ***Runtime*** *evaluation.* If timeout of 3600s is reached (indicated by dotted line) iCOBOPT starts to produce suboptimal solutions. Base solutions were obtained almost immediately.

However, WHCA* can generate near optimal solutions in the same range of the occupancy at much lower cost (all the invocations of WHCA* in our tests finished within 5 minutes). So, it would be more appropriate to use WHCA* instead of BIBOX to generate base solutions here and submit them to iCOBOPT for further improvement. Nonetheless, iCOBOPT is capable of shortening original base solutions by at least half in all the tested occupancy setups (for higher occupancies this refers to improvement on INVERSE encoding).

The domain-dependent approach turned out to be much better than SATPLAN and SASE if applied on CPFs (Table 3). SATPLAN and SASE encodings become quickly prohibitively large for the increasing number of agents. The size of the environment has also a great impact on the performance (as also does in case of iCOBOPT approach) – only up to 16 agents can be solved in 8×8 grid and 8 agents in 16×16 grid by SASE within the given time limit of 3600 seconds. For better comparison performance of iCOBOPT is on the same instances is summarized in Table 4.

Table 3. ***Runtime*** *of classical domain independent planners on CPFs.* Timeout of 3600s (1 hour) has been used. Only relatively small instances have been solved by SATPLAN and SASE. However they are not small in absolute terms as solutions consist of hundreds of actions.

$\|A\|$ in the 4-connected grid	4-connected grid **8x8**			4-connected grid **16x16**		
	Optimal makespan	**SATPLAN** Runtime (s)	**SASE** Runtime (s)	Optimal makespan	**SATPLAN** Runtime (s)	**SASE** Runtime (s)
1	5	0.0	0.45	4	0.68	1.66
4	6	0.15	2.57	21	195.5	17.98
8	8	19.85	4.73	15	1396.07	128.87
16	10	Timeout	253.55	N/A	Out of memory	Timeout

Very importantly, the iCOBOPT process is anytime provided that procedure for generating a base solution is fast enough (that is, at any time we have some solution).

Table 4. ***Runtime*** *of* iCOBOPT. Same instances as in the case of SATPLAN and SASE were used.

$\|A\|$ in the 4-connected grid	4-connected grid **8x8**			4-connected grid **16x16**		
	Computed makespan	**INVERSE** Runtime (s)	**ALL-DIFF** Runtime (s)	Computed makespan	**INVERSE** Runtime (s)	**ALL-DIFF** Runtime (s)
1	5/5	0.001	0.001	6/6	0.074	0.070
4	6/6	0.002	0.003	21/21	319.785	45.367
8	8/8	9.105	3.556	15/15	152.625	62.955
16	10/10	196.991	34.444	18/18	1833.080	910.391

A. *Enhancement via Heuristic Distance Estimation*

We evaluated the effect of using heuristic reasoning within the ALL-DIFFERENT encoding by comparing it with the version without the reasoning. Sizes of the enhanced encoding are shown in Table 5. The reduction of the size of the encoding is particularly significant if the number of layers is small since agents don't have enough time-steps to spread over the environment.

Table 5. ***Sizes*** *of the ALL-DIFFERENT encoding with* ***heuristic*** *reasoning on the grids* ***8×8*** *and* ***16×16***. Instances are the same as in Table 1, Table 2.

$\|A\|$ in the 4-connected grid **8x8**	Number of **layers**	**HEURISTIC ALL-DIFFERENT** encoding	
		|**Variables**|	|**Clauses**|
4	8	2528	10626
8	8	7942	27543
12	8	16026	49535
16	10	38304	119827
24	10	81000	235663
32	14	219059	659882

$\|A\|$ in the 4-connected grid **16x16**	Number of **layers**	**HEURISTIC ALL-DIFFERENT** encoding	
		21980	147136
4	21	29763	164052
8	15	125633	594618
16	18	56725	144146
32	4*	88789	218010
40	4*	228186	532339
64	4*	21980	147136

The smaller encoding can be solved faster according to our observations and hence more solution sub-sequence optimization attempts can be done within the given time limit by the iCOBOPT algorithm. As a result solutions with the shorter makespan can be generated. Particularly for the case of the grid 16×16 and the occupancy of 14% - 31% up to 50% time-steps can be saved with respect to the second best encoding (INVERSE). For more detailed results see Fig. 6 and Fig. 7.

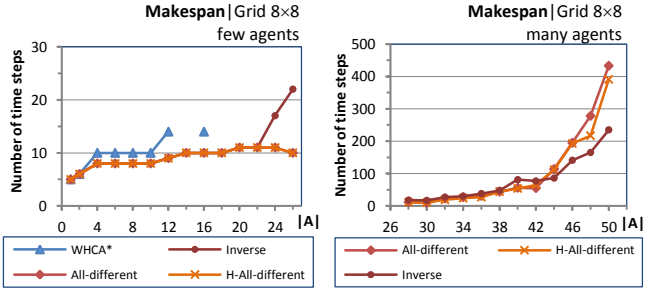

Fig. 6. ***Heuristic*** *improvement on the 8×8 grid.* A marginally better makespan can be achieved by using the ALL-DIFFERENT encoding with heuristic reasoning.

The result that the ALL-DIFFERENT encoding can be solved generally faster is also indicated in Fig. 8 where we show the runtime if the timeout has not been reached, that is when optimal solution can be generated. In this case the

optimal solution is generated as fastest by the use of the ALL-DIFFERENT with heuristic reasoning.

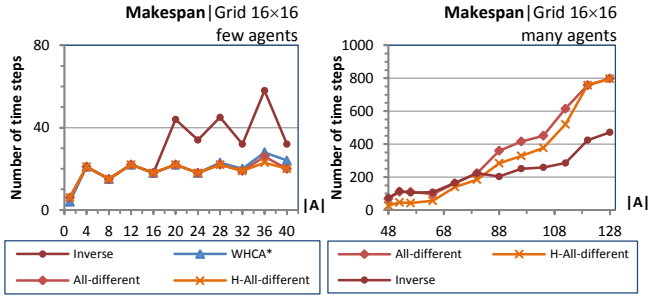

Fig. 7. ***Heuristic*** *improvement on the 16×16 grid.* The improvement achieved by using the heuristic reasoning in the ALL-DIFFERENT encoding is greater than in case of the 8×8 grid. Here, the heuristic variant is the best choice for the population of agents consisting of 36 to 80 agents (occupancy 14% - 31%).

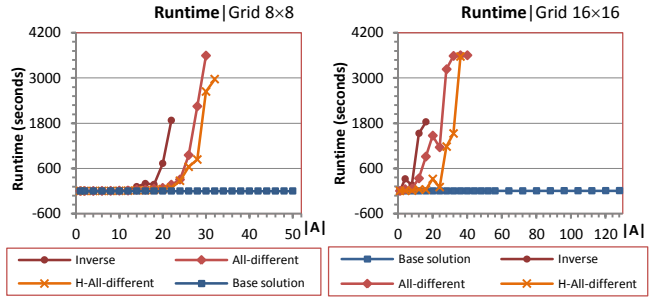

Fig. 8. ***Runtime vs. timeout*** *comparison.* If the optimization finished in the time limit of 3600s then the runtime is shown. In such a case optimal solution has been found. The ALL-DIFFERENT encoding with heuristic reasoning is fastest and degrades towards the timeout as the last.

VI. SUMMARY, CONCLUSIONS, AND FUTURE WORK

A novel SAT-based technique – called iCOBOPT – for solving cooperative path-finding optimally or near optimally has been introduced. Additionally two SAT encodings were proposed – called INVERSE and ALL-DIFFERENT. They represent CPF instances more compactly than existing domain independent encodings used in SAT-based planners like SATPLAN and SASE.

We found that the proposed approach performs especially well on highly constrained instances with many agents and small unoccupied space. These are exactly the cases where techniques from [14] do not scale well as authors report (grid of size 4×4 is reported). The INVERSE encoding is more suitable for densely populated environments while the ALL-DIFFERENT one is better in the sparse case. We also demonstrate the advantage of the ALL-DIFFERENT by enhancing it with the heuristic reasoning that can help to make it smaller by ruling out many states.

The important feature of our approach is also the fact that the iCOBOPT technique is very modular – a different SAT solver as well as arbitrary algorithm for generating base solutions can be used. Notice also, that the technique is friendly to multi-threaded implementation since optimizations of several isolated parts of the base solution can run in parallel.

We approached the classical $(N^2 - 1)$-puzzle in our experiments with 4-connected grid environments. Actually an attempt to solve $(64 - 14)$-puzzle has been made here. Although it is not expectable that for example 63-puzzle can be solved optimally even with elevated timeout by the iCOBOPT technique, some progress from [6] may be done in future. We are also considering investigating some alternative and more efficient encodings of the ALL-DIFFERENT constraint than that from [1]. For future work we also plan to switch the whole optimization process to CSP [2] where we could exploit the full strength of global propagators for the ALL-DIFFERENT constraint.

REFERENCES

[1] Biere, A., Brummayer, R. "Consistency Checking of All Different Constraints over Bit-Vectors within a SAT Solver", Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2008), IEEE press, 2008, pp. 1-4.

[2] Dechter, R.: "Constraint Processing", Morgan Kaufmann Publishers, 2003.

[3] Eén, N., Sörensson, N. "An Extensible SAT-solver", Proceedings of Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919, Springer, 2004, pp. 502-518.

[4] Huang, R., Chen, Y., Zhang, W. "A Novel Transition Based Encoding Scheme for Planning as Satisfiability", Proceedings AAAI 2010, AAAI Press, 2010.

[5] Kautz, H., Selman, B. "Unifying SAT-based and Graph-based Planning", Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999), Morgan Kaufmann, 1999, pp. 318-325.

[6] Korf, R. E., Taylor, L. A. "Finding Optimal Solutions to the 24-Puzzle", Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996), AAAI Press, 1996, pp. 1202-1207.

[7] Luna, R., Berkis, K., E. "Push-and-Swap: Fast Cooperative Path-Finding with Completeness Guarantees", Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), IJCAI/AAAI Press, 2011, pp. 294-300.

[8] Nightingale, P., Gent, I. "A New Encoding of AllDifferent into SAT", CP 2004 Workshop on Modelling and Reformulating CSPs, 2004.

[9] Ratner, D., Warmuth, M. K. "Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE Is Intractable", Proceedings of AAAI 1986, Morgan Kaufmann, 1986, pp. 168-172.

[10] Régin, J-C. "A Filtering Algorithm for Constraints of Difference in CSPs", Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994), AAAI Press, 1994, pp. 362-367.

[11] Rintanen, J. "Compact Representation of Sets of Binary Constraints", Proceedings of ECAI 2006, IOS Press, 2006, pp. 143-147

[12] Ryan, M. R. K. "Exploiting Subgraph Structure in Multi-Robot Path Planning", Journal of Artificial Intelligence Research (JAIR), Volume 31, AAA Press, 2008, pp. 497-542.

[13] Silver, D.. Cooperative Pathfinding. Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), AAAI Press, 2005, pp. 117-122.

[14] Standley, T. S., Korf, R. E. "Complete Algorithms for Cooperative Pathfinding Problems", Proceedings of IJCAI 2011, IJCAI/AAAI Press, 2011, pp. 668-673

[15] Surynek, P. "A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs", Proceedings of ICRA 2009, IEEE Press, 2009, pp. 3613-3619.

[16] Surynek, P. "Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving", Proceedings of PRICAI 2012, LNCS 7458, Springer, 2012, pp. 564-576.

[17] Tseitin, G. "On the complexity of derivation in propositional calculus", Studies in Constructive Mathematics and Mathematical Logic, 1968, pp. 115–125.

[18] Wang, K. C., Botea, A. "MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees", JAIR, Volume 42, AAAI Press, 2011, pp. 55-90.