Mutex Reasoning in Cooperative Path Finding Modeled as Propositional Satisfiability

Pavel Surynek

Abstract— **This paper addresses a problem of cooperative path finding (CPF) where the task is to find paths for agents of a group of agents. Each agent is given a starting and a goal position and its task is to reach the goal from the given start. When following the paths, agents must not collide with each other and must avoid obstacles. It is suggested to augment propositional encodings of CPF with a so called mutex reasoning. Mutex reasoning is trying to rule out unreachable situations to reduce the size of the search space. It is checked whether a given pair of locations is reachable by a given pair of agents cooperatively. If not occurrence of the pair of agents in the pair of vertices is forbidden. The performed experimental evaluation showed that mutex reasoning improves existent encodings by 2 to 5 times in terms of solving runtime when makespan optimal solutions are searched.**

I. Introduction and Related Works

Cooperative path finding (CPF) [7], [11], [12], [14] represents an important problem in robotics. Consider a group of agents or robots moving in a certain environment. Each agent is given its starting position and goal position in the environment. The task of each agent is to go from the starting position to the goal. Agents must not collide with each other and must avoid obstacles in the environment.

The environment where agents are moving is typically modeled as an *undirected graph* where agents are placed in vertices and edges represent passable regions [7].

In this work we are dealing with encodings of CPF as propositional satisfiability (SAT) [2]. This allows us to use the power of modern SAT solvers [13] in solving of CPF. Employing SAT in CPF solving has been already studied in [15], [16]. Several encodings of CPF as SAT were suggested in these works. T he most important of them are so called *inverse* and *all-different* encodings.

An improvement of existent encodings by a concept of mutex reasoning is proposed. Mutex reasoning is inspired by techniques from [5]. It check if a given pair of vertices is reachable by a given pair of agents in a given time. If this is not possible then simultaneous occurrence of the given pair of agents in the given pair of vertices is forbidden. This leads to reduction of the size of the search space and consequently in faster solving of the encoded instance.

Additionally it is proposed to solve CPF by translation to SAT optimally with respect to the makespan. This is done by trying to solve CPF encoding by SAT for iteratively longer and longer makespans until solvable makespan is found. This is the same approach as used in [4], [5]. This is however different from approach used in [15], [16] a makespan sub-optimal solution is locally improved by replacing sub-solutions by makespan optimal ones.

The approach employing SAT solving is targeted on instances that are highly crowded with agents. This is the case where other optimal solving methods such as [6] do not perform well since they need to have lot of independence among agents.

In section II. the problem of CPF is formally introduced and existent propositional encodings are recalled. Next, the concept of mutex reasoning is described (section III.) – this is the main contribution of the paper. In section IV. it is shown how to use SAT encodings of CPF to find makespan optimal solutions. An experimental part, where impact of mutex reasoning on overall runtime is evaluated, follows in section V. A competitive comparison with an alternative algorithm WHCA* [12] is also given. Finally, concluding remarks are given and future directions are mentioned.

Pavel Surynek is with Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic (phone: +420 221 914 245; fax: +420 221 914 323; e-mail: pavel.surynek@mff.cuni.cz).

II. Background

In this section we would like to recall definition of the problem of *cooperative path-finding* (CPF) [7], [11], [12], [14] and existent encodings of CPF as *propositional satisfiability* (SAT) [1], [2] as they were suggested in [15], [17]. Encodings of our interest were introduced as *inverse* and *all-different* in mentioned works.

A. Problem of Cooperative Path Finding (CPF)

The problem of cooperative path-finding which is also sometimes referred as *multi-agent path finding* is a task to find paths for agents of a group of agents. Each agent is given its starting position and its task is reach a given goal position without colliding with other agents and obstacles. Agents move in a certain environment which is typically modeled as undirected graph where agents are placed in vertices – at most one agent can be placed in a vertex and at least one vertex must remain vacant to allow agents to move. Edges model passable regions; that is, agents move between vertices through edges.

The dynamicity of the model is that we adopt discrete time. The arrangement of agents on the graph can change between consecutive time steps in the following way. An agent can move into neighboring unoccupied vertex supposed that no other agent is entering the same target vertex. Notice that multiple agents can move at a time.

Let $G = (V, E)$ be an undirected graph and let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of agents where $|A| < |V|$. The arrangement of agents in G will be a uniquely invertible function $\alpha: A \rightarrow V$ called a *location function*. The interpretation is that an agent $a \in A$ is located in a vertex $\alpha(a)$. A generalized inverse of α denoted as $\alpha^{-1}: V \rightarrow A \cup$

$\{\perp\}$ will provide us an agent located in a given vertex or $\perp$ if the vertex is empty.

An arrangement of agents at time step $i \in \mathbb{N}_0$ (natural numbers including 0) will be denoted as α_i. If we formally express above transition conditions in terms of location function then we have following *transition constraints*:

(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 initial arrangement is α_0 and α^+ will denote the goal arrangement. An instance of CPF is then given as quadruple $[G, A, \alpha_0, \alpha^+]$. The task is to transform α_0 to α^+ so that transition constraints are preserved between all consecutive time steps.

Definition 1 (solution, makespan). Let $\Sigma = [G, A, \alpha_0, \alpha^+]$ be an instance of CPF. A *solution* of Σ is a sequence of arrangements $\alpha_0, \alpha_1, \ldots, \alpha_\mu$ where $\alpha_\mu = \alpha^+$ and transition constraints are satisfied between α_{i-1} and α_i for every $i = 1, \ldots, \mu$. The number μ is called a *makespan* of the solution. The shortest possible makespan of Σ will be denoted as $\mu^*(\Sigma)$. □

This paper addresses a question what is $\mu^*(\Sigma)$ for a given CPF Σ and what is the corresponding makespan optimal solution. This question has been already addressed from the complexity point of view – it is known that the question is NP-hard [11].

B. Inverse Encoding

The *inverse encoding* introduced in [15] is used to translate a question whether there exists a solution to a given CPF of a given makespan into SAT. Having such an encoding it is then possible to find $\mu(\Sigma)$ for a given Σ by iterative trying of longer and longer makespans as it is done by SAT-based domain independent planners SASE [4] and SATPlan [5] (this approach however is not complete; unsolvable CPF cannot be answered in this way).

Basically we need to model arrangements of agents at individual time steps and to introduce transition constraints into the model. In the inverse encoding, the arrangement of agents at time step i is modeled by state variables $\mathcal{A}_i^v$ for $v \in V$ that represent inverse location function at the time step i. Next, there are state variables $\mathcal{T}_i^v$ for $v \in V$ that represent actions taken in vertices at time step i. An *outgoing action* into some of vertex neighbors or an *incoming action* from some of vertex neighbors or *noop* can be taken in each vertex. The domain of $\mathcal{T}_i^v$ must consist of $2\deg_G(v)$ values to represent all the possible actions. It is necessary to introduce some ordering on neighbors of each vertex to be able to assign concrete actions to elements of the domain of $\mathcal{T}_i^v$. Suppose that we have a function $\sigma_v: \{u | \{v, u\} \in E\} \longrightarrow \{1, 2, \ldots, \deg_G(v)\}$ and its inverse σ_v^{-1} that implements this ordering of neighbors. Now we are ready to introduce representation of the i-th arrangement formally.

Definition 2 (inverse encoding). The i-th level of *inverse encoding* consists of the following integer interval **state variables**:

- $\mathcal{A}_i^v \in \{0, 1, 2, \ldots, n\}$ for all $v \in V$ such that $\mathcal{A}_i^v = j$ iff $\alpha_i(a_j) = v$
- $\mathcal{T}_i^v \in \{0, 1, 2, \ldots, 2\deg_G(v)\}$ for all $v \in V$ such that
$$\begin{cases} \mathcal{T}_i^v = 0 & \text{iff no-op was selected in } v; \\ \mathcal{T}_i^v = \sigma_v(u) & \text{iff an outgoing primitive action with the target } u \in V \text{ was selected in } v; \\ \mathcal{T}_i^v = \deg_G(v) + \sigma_v(u) & \text{iff an incoming primitive action with } u \in V \text{ as the source was selected in } v. \end{cases}$$

and **constraints**:

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

Notice that the encoding is built upon integer state variables. We eventually use propositional encoding which is obtained by translating integer state variables into bit vectors. If the state variable has N states (N elements in its domain) then we need $\lceil \log_2 N \rceil$ propositional variables.

If we are asking whether there is a solution of makespan k to the given CPF Σ we need to build k levels according to the definition. The starting arrangement is encoded so that the $\mathcal{A}_0^v$ are set to reflect α_0. Analogically this is done for the goal arrangement with $\mathcal{A}_k^v$ and α^+.

C. All-Different Encoding

An alternative encoding to inverse has been presented in [17]. The alternative encoding is called *all-different* since it uses all-different constraints extensively [10]. It is designed to overcome the drawback of inverse encoding which size does not reduce if the number of agents is reduced. In the all-different encoding we encode positions of individual agents directly; that is, location function α is used as basis for the design of state variables. Suppose that $V = \{v_1, v_2, \ldots, v_m\}$ we have a state variable $\mathcal{L}_i^a \in \{1, 2, \ldots, m\}$ for every agent $a \in A$ and time step i representing in what vertex a is located at i. Such design of state variables however allows infeasible states where two or more agents share a vertex. Therefore all-different constraints need to be enforced on variables representing locations at every time step. Formally the encoding looks as follows.

Definition 3 (all-different encoding). The i-th level of the *all-different encoding* consists of the following finite domain integer **state variables**:

- $\mathcal{L}_i^a \in \{0, 1, 2, \ldots, m\}$ 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 $l \in \{1, 2, \ldots, m\}$
$$\mathcal{L}_i^a = l \Rightarrow \mathcal{L}_{i+1}^a = \ell \vee \bigvee_{\ell \in \{1,\ldots,m\} | \{v_l, v_\ell\} \in E} \mathcal{L}_{i+1}^a = \ell$$
(agents can move only **along edges** of G),
- for all $a \in A$
$$\bigwedge_{b \in A | b \neq a} \mathcal{L}_{i+1}^a \neq \mathcal{L}_i^b$$
(the **target vertex** of agent's move must be **empty**),

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

$$\text{AllDifferent}(\mathcal{L}_i^{a_1}, \mathcal{L}_i^{a_2}, \dots, \mathcal{L}_i^{a_n}). \quad \square$$

The encoding over integer variables is again translated into propositional vectors eventually.

Since encoding the constraint that agents can move along edges only is very space consuming and resulting in extremely large formulae a reduction heuristic has been introduced in [16]. If a vertex is unreachable by an agent from the starting position in the given time or cannot reach the goal from the vertex than constraints regarding this vertex can be omitted. Formally the first constraint is introduced if and only if the following condition holds:

$$\text{dist}_G(\alpha_0(a), v_l) \leq i \wedge \text{dist}_G\big(v_l, \alpha^+(a)\big) \leq k - i$$

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 $l \in \{1,2,\dots,n\}$ such that $\text{dist}_G(\alpha_0(a), v_l) > i \vee \text{dist}_G\big(v_l, \alpha^+(a)\big) > k - i$ **include** $\mathcal{L}_i^a \neq l$

Using above *distance heuristic* constraints to rule out certain situation let to a drastic improvement in size of the resulting formulae in all-different encoding as well as to significant reduction in solving runtime by a SAT solver [16]. These results inspired us to further develop this type of heuristic.

III. Mutex Reasoning

The application of the standard distance heuristic was originally motivated by reduction of the size of the resulting propositional formulae. Surprisingly it also let to improvements in the solving runtime. These results inspired us to use the distance heuristic also in the inverse encoding – here it does not reduce the size of the formula since no constraint cannot be omitted but the runtime may be reduced.

We also observed that the distance heuristic simplifies the situation very much. It does not account any interaction among agents which is an important feature of CPF. Hence we were considering some extension of the heuristic that also considers interaction among agents.

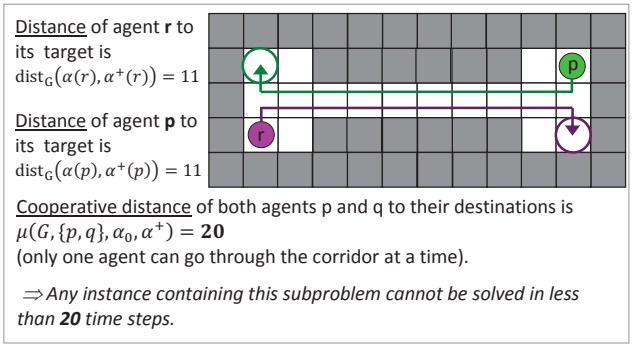

Figure 1. Illustration of filtration through **mutex reasoning**.

This resulted in our suggestion of *mutex reasoning* in encodings of CPF. Mutex reasoning takes into account pairs of agents and checks they can occur simultaneously in a given pair of vertices at a given time step. This is possible only if they have chance to reach cooperatively the given pair of vertices from starting positions and if they can cooperatively reach the target vertices from the given pair of vertices at the same time. Otherwise occurrence of the pair of agents in the given pair of vertices at the given time step will be forbidden.

The benefit of mutex reasoning is illustrated in Figure 1. Notice that it is stronger than the standard distance heuristic which considers distance to the target as the shortest path and if the remaining number of time steps is less than the distance then the situation is ruled out as infeasible. Such reasoning however omits collisions among agents and is thus quite inaccurate. Mutex reasoning on the other hand considers pairs of agents in the cooperative manner. The instance cannot be solved if the pair of agents cannot cooperatively (which means that collisions between them are considered) reach their targets in the remaining number of time steps. As it is summarized in the following proposition the cooperative distance to the targets is greater or equal to the shortest path distances and hence mutex reasoning is able to rule out more infeasible situations.

Proposition 1 (mutex reasoning). Let $G = (V, E)$ be a graph, $A = \{r, p\}$ be a set of agents, $\alpha_0 : A \longrightarrow V$ and $\alpha^+ : A \longrightarrow V$ be a starting and a goal arrangement of agents respectively. Then $\mu(G, A, \alpha_0, \alpha^+) \geq \mu\big(G, \{r\}, \alpha_0|_{\{r\}}, \alpha^+|_{\{r\}}\big)$ and $\mu(G, A, \alpha_0, \alpha^+) \geq \mu\big(G, \{p\}, \alpha_0|_{\{p\}}, \alpha^+|_{\{p\}}\big)$. If there exist shortest paths from $\alpha_0(r)$ to $\alpha^+(r)$ and from $\alpha_0(p)$ to $\alpha^+(p)$ that does not share any vertex then $\mu(G, A, \alpha_0, \alpha^+) = \max\big[\mu\big(G, \{r\}, \alpha_0|_{\{r\}}, \alpha^+|_{\{r\}}\big); \mu\big(G, \{p\}, \alpha_0|_{\{p\}}, \alpha^+|_{\{p\}}\big)\big]$. A case when strict inequalities appears also exists. ■

Proof. Clearly each agent has to reach its target vertex which means to travel at least the distance equal to the length of the shortest path from the starting position to the target vertex. Hence first two inequalities must hold. If shortest path connecting $\alpha_0(r)$ to $\alpha^+(r)$ and $\alpha_0(p)$ to $\alpha^+(p)$ are disjoint then agents r and p can travel along them independently and need maximum of both lengths to reach $\alpha^+(r)$ and $\alpha^+(p)$ respectively. The case with strict inequality is shown in Figure 1. ■

Notice that there is still inaccuracy in mutex reasoning since it does not account triples and larger groups of agents. Computing cooperative distance to targets for larger groups than pairs is computationally increasingly difficult. Hence we settle with pairs as it represents a good trade-off between filtering strength and computation time.

A. Extensions of Encoding with Mutex Reasoning

The inverse encoding can be extended with mutex reasoning in the following way. Again suppose that encoding is constructed for makespan of k. The distance heuristic is introduced by adding the following constrains into the i-th level of the encoding:

- for all $v \in V$ and $a \in \{1,2,\dots,m\}$ such that $\text{dist}_G(\alpha_0(a), v) > i \vee \text{dist}_G\big(v, \alpha^+(a)\big) > k - i$ **include** $\mathcal{A}_i^v \neq a$

Next suppose a pair of distinct agents $a, b \in A$ and a pair of distinct vertices $u, v \in V$. If the distance heuristic does not forbid occurrence of a in u at time step i nor occurrence

of b in v at time step i then mutex reasoning should be done and the possibility of simultaneous occurrence of a and b in u and v respectively at time step i should be checked. Formally it means to introduce the following constraint:

- for all $u, v \in V$ with $u \neq v$ and $a, b \in \{1,2,\dots,m\}$ with $a \neq b$ such that
  $\mathrm{dist}_{\mathrm{G}}(\alpha_0(a), u) \leq i \wedge \mathrm{dist}_{\mathrm{G}}(u, \alpha^+(a)) \leq k - i$ **and**
  $\mathrm{dist}_{\mathrm{G}}(\alpha_0(b), v) \leq i \wedge \mathrm{dist}_{\mathrm{G}}(v, \alpha^+(b)) \leq k - i$ **and**
  $\left[\begin{array}{l} \mu\big(G, \{a,b\}, \alpha_0|_{\{a,b\}}, (a \to u, b \to v)\big) > i \quad \textbf{or} \\ \mu\big(G, \{a,b\}, (a \to u, b \to v), \alpha^+|_{\{a,b\}}\big) > k - i \end{array}\right]$
  include $\mathcal{A}_i^u \neq a \vee \mathcal{A}_i^v \neq b$

The same augmentation can be made for the all-different encoding. We do not need to add distance heuristic as it is already present. Thus only mutex reasoning is added as follows:

- for all $a, b \in A$ with $a \neq b$ and $l, h \in \{1,2,\dots,n\}$ such that
  $\mathrm{dist}_{\mathrm{G}}(\alpha_0(a), v_l) \leq i \wedge \mathrm{dist}_{\mathrm{G}}(v_l, \alpha^+(a)) \leq k - i$ **and**
  $\mathrm{dist}_{\mathrm{G}}(\alpha_0(b), v_h) \leq i \wedge \mathrm{dist}_{\mathrm{G}}(v_h, \alpha^+(b)) \leq k - i$ **and**
  $\left[\begin{array}{l} \mu\big(G, \{a,b\}, \alpha_0|_{\{a,b\}}, (a \to u, b \to v)\big) > i \quad \textbf{or} \\ \mu\big(G, \{a,b\}, (a \to u, b \to v), \alpha^+|_{\{a,b\}}\big) > k - i \end{array}\right]$
  include $\mathcal{L}_i^a \neq l \vee \mathcal{L}_i^b \neq h$

B. Notes on the Implementation of Mutex Reasoning

To implement mutex reasoning it is necessary to quickly determine the optimal makespan μ for every pair of agents and every pair of vertices. The task is transformed into finding shortest paths in a derived graph. The graph say $G^2 = (V^2, E^2)$ will consists of pairs of vertices; that is, $V^2 = V \times V$. Edges will model allowed moves of pairs of agents; that is, $[(u,v);(w,z)] \in E^2$ if and only if $(u,v) = (w,z)$ or $|\{u,v,w,z\}| = 3 \wedge (u = v \vee w = z)$ or $|\{u,v,w,z\}| = 4$.

Construction of G^2 consumes time and space of $\mathcal{O}(|V|^4)$. Computing all pairs shortest paths in G^2 using matrix multiplication requires time of $\mathcal{O}(\log_2|V|^2 \cdot |V|^{2\omega}) = \mathcal{O}(\log_2|V| \cdot |V|^{2\omega})$ where $\mathcal{O}(N^\omega)$ is complexity of matrix multiplication of size $N \times N$. The well known bound for matrix multiplication is $\omega < 2.376$ [3].

The high complexity of this style of computing of optimal makespans prohibits application of the technique for larger groups of agents than pairs.

Let us note that in out implementation we compute optimal makespan for pairs of vertices only if it is needed (actually we need only single source shortest paths from pairs of starting positions and from goal positions respectively) which makes this computation negligible with respect to time consumption by the SAT solver.

IV. Makespan Optimal Solving of CPF

Modeling of CPF as propositional formula has been already studied in [15], [16], [17]. However, the approach adopted in these works was to generate solutions of makespan as short as possible but not necessarily the optimal one. This was done by generating sub-optimal solution first by some existing method – the polynomial time algorithm *BIBOX* [14] has been employed. Then subsequences of the computed sub-optimal solution were iteratively replaced by makespan optimal sub-solutions computed by the SAT solver until timeout was reached. Given a subsequence of the solution it can be used to form an instance of CPF. The arrangement at the beginning of the subsequence is taken as the starting arrangement and the state at the end of the subsequence is taken as the goal arrangement.

Here we would like to use SAT solving to compute makespan optimal solutions. We can to adopt the technique how makespan optimal replacements are computed in [15], [16], [17]. This is done by trying iteratively longer and longer makespans while for each makespan the encoding of the instance is constructed and the SAT solver is asked if it is satisfiable. If so the process terminates with answer that solution exists. If not longer makespan is tried. If the there is no solution to the given CPF instance then the process continues to the given makespan limit and terminates with no answer. In this sense the technique is incomplete. Notice that the same strategy is used in SAT-based domain independent planning as represented by SATPlan [5] and SASE [4] planners.

Notice however that it can be easily made complete by first checking if solution exists by some fast polynomial time algorithm such as *BIBOX* [14] or *PUSH-and-SWAP* [8]. If this initial check says that solution exists then we run iterative SAT solving with no makespan limit.

The process of generating makespan optimal solution to CPF is formalized as Algorithm 1.

Algorithm 1. SAT based CPF solving.
input: a CPF instance Σ and makespan limit $\mathbb{k}$.
output: a pair consisting of the optimal makespan and an optimal solution

```
function Find-Optimal-Solution (Σ = (G = (V, E), A, α0, α+), k): pair
1:  for k = 1,2,... , k do
2:      Ξ ← Encode-CPF-as-SAT (Σ, k)
3:      if Solve-SAT (Ξ) then
4:          s ← Extract-Solution-from-Valuation(Ξ)
5:          return (k, s)
6:  return (∞, ∅)
```

V. Experimental Evaluation

We implemented the presented mutex reasoning encoders and SAT-based optimal solution generator employing proposed encodings in C++. Our preliminary experiments showed that the most suitable SAT solver for CPF encodings is cryptominisat [13] hence we employed it in out solution generator. The code as well as all the experimental data will be made available on: http://ktiml.mff.cuni.cz/~surynek/research/iros2013 to allow reproducibility of all the presented results.

The experimental evaluation is targeted on measuring the runtime of the SAT-based optimal solving that employs proposed mutex reasoning in comparison with existent encodings without mutex reasoning and on comparison with WHCA* [12]. WHCA* is one of the most frequently used algorithm for CPF and it is known to generate near optimal solutions very quickly and thus it is suitable for comparison.

As it was suggested by Silver in [12] we used CPFs consisting of 4-connected grids with randomly placed obstacles. We the setting where 20% of randomly selected

vertices were occupied by obstacles (the same setting was used by Silver in [12]).

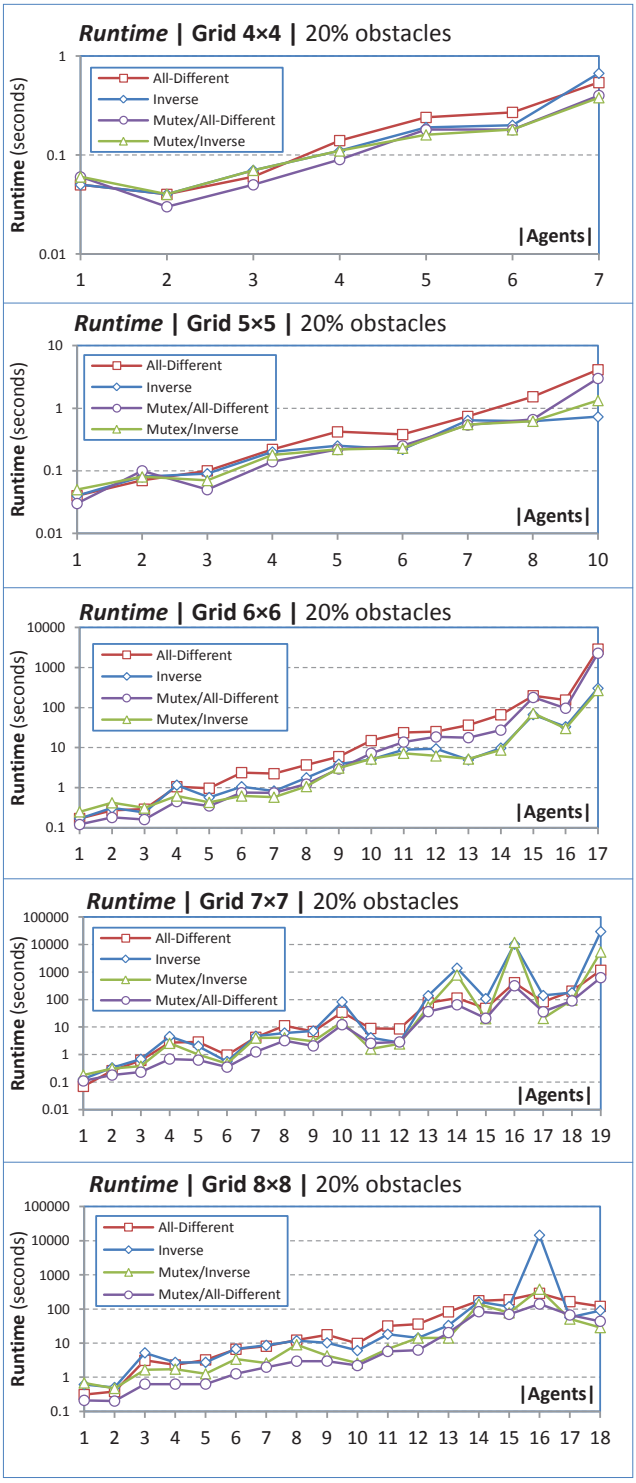

Figure 2. **Runtime of optimal** SAT-based solver with various encodings with respect to the incresing number of agents. In smaller grids *inverse encoding with mutex reasoning* performs marginally better than *all-different encoding with mutex resoning* which are both better than encodings without mutex reasoning - the improvement is around 2.0 times to 5.0 times. In larger grids all-different encoding with mutex resoning is the best.

Grid of sizes 4×4, 5×5, 6×6, 7×7, and 8×8 were tested. The number of agents ranged from 1 to $|V|/2$. The starting arrangement and the goal arrangement were generated randomly. An example of testing instance on 5×5 grid is shown in Figure 3.

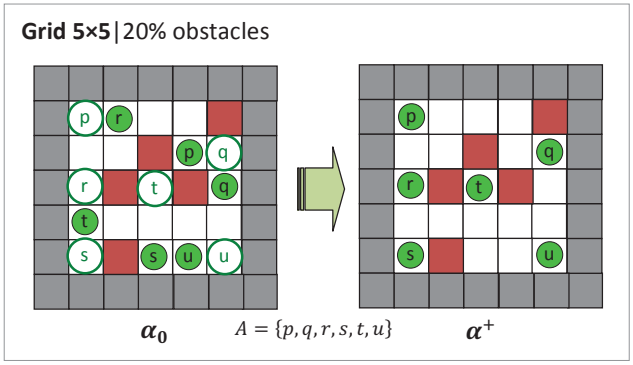

Figure 3. Example of CPF instance on a **grid of size 5×5** with 20% of positions occupied by obstacles.

Comparison of runtimes with *inverse* and all-different encodings and those augmented with mutex reasoning is shown in Figure 2. It can be read from results that mutex reasoning improves solving time approximately by 2.0 to 5.0 times. On grids of size 4×4, 5×5, and 6×6 *inverse encoding with mutex reasoning* dominates. On two larger cases – 7×7 and 8×8 – the best encoding is *all-different encoding with mutex reasoning*. This encoding turned out to be most suitable in cases with few agents in large environments. However, notice that SAT-based CPF solving is targeted on relatively crowded environments where other techniques such as WHCA* are failing. In crowded environments the *inverse* encoding is more suitable and mutex reasoning makes it yet faster solvable.

TABLE I. OPTIMAL MAKESPANS OF CPF ON GRIDS WITH 20% OBSTACLES.

Grid size |Agents|	4×4	5×5	6×6	7×7	8×8
1	4	3	6	4	7
2	3	4	7	6	6
3	4	4	6	7	12
4	5	6	10	11	9
5	6	7	8	10	9
6	6	6	10	6	11
7	7	7	9	10	11
8		8	10	13	11
9			11	10	11
10		9	13	13	9
11			13	10	11
12			12	9	11
13			12	13	12
14			13	13	13
15			14	11	13
16			14	15	14
17			15	11	12
18				13	12
19				16	

The computed optimal makespans are summarized in TABLE I. In some cases solution did not exists.

The comparison of optimal SAT-based solving with WHCA* regarding the makespan is shown in Figure 4. It can be observed that WHCA* really generates solution whose makespans are close to the optimal ones. However, the weakness of WHCA* is that on more crowded environments it is unable to generate any solution. In these cases SAT-based solving with improved encodings in more promising. Window size in WHCA* was set to 64 in all the presented test. All the invocations of WHCA* finished in less than 0.1 seconds.

If we summarize experimental results we can conclude that mutex reasoning brings a significant improvement into existent CPF encodings. This means further improvement of SAT-based CPF solving. Moreover we suggested to use translation of CPF to SAT for makespan optimal solving of CPF. As mutex reasoning improves SAT encodings of CPF it can be viewed as technique that makes optimal CPF solving through SAT more viable.

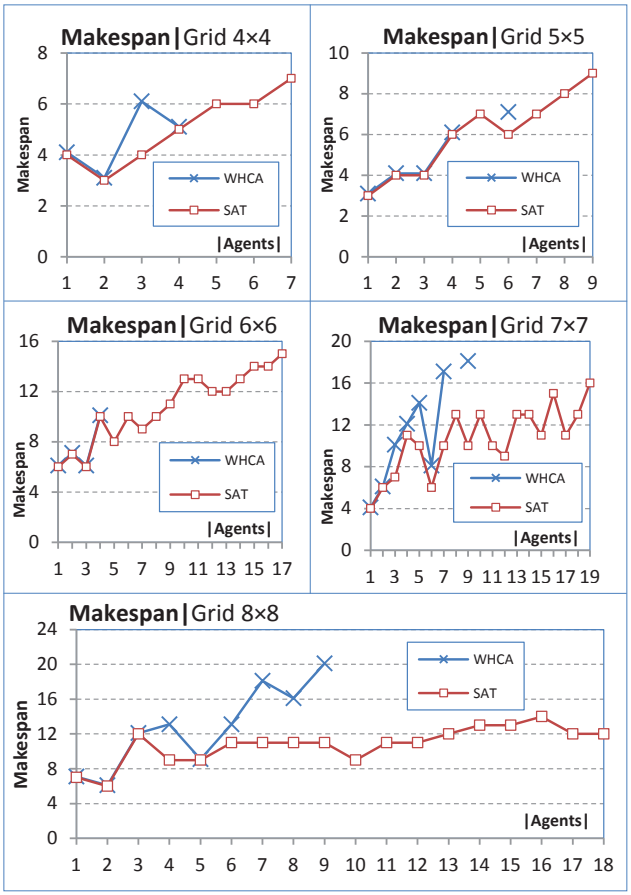

Figure 4. Comparison optimal makespans generated by SAT-based solving with those generated by WHCA*. The window size for WHCA* was 64. Again there is 20% of obstacles. In more crowded environments WHCA* is unable to generate any solution.

VI. Conclusion and Future Work

In this paper we suggested a so called *mutex reasoning* as an improvement of existent encodings of CPF as SAT. Namely existent inverse and all-different encodings are augmented with mutex reasoning. Mutex reasoning checks reachability of a pair of vertices by a pair of agents in cooperative manner. This allows to rule out certain important cases that can reduce the size the search space significantly – these cases include for example a situation where two agents need to exchange themselves through a corridor.

Additionally we suggested to employ translation of CPF to SAT for makespan optimal CPF solving. This is a first attempt to solve CPF optimally by SAT (in related works only near optimal solutions were found through SAT solving). Together with encodings augmented by mutex reasoning we have shown in comparison with WHCA* that SAT-based approach is a viable option to solve CPF optimally.

For future work we are considering to further improve SAT encodings of CPF. It seems to be promising to preprocess clauses expressing difference of certain state variable from a set of constants by resolution which can reduce the size of clauses expressing mutex reasoning.

Acknowledgment

This work is supported by The Ministry of Education, Youth and Sports, Czech Republic (Ministerstvo školství, mládeže a tělovýchovy ČR – MŠMT ČR) under the contract PRVOUK P46 and by the by The Czech Science Foundation (Grantová agentura České republiky - GAČR) under contract number GAP103/10/1287.

References

[1] A. Biere, M. Heule, H. van Maaren, T. Walsh, "Handbook of Satisfiability," IOS Press, 2009.

[2] S. A. Cook, "The Complexity of Theorem Proving Procedures," Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), pp. 151-158, ACM Press, 1971.

[3] D. Coppersmith, S. Winograd, "Matrix Multiplication via Arithmetic Progressions," Journal of Symbolic Computation, Volume 9(3), pp. 251-280, Elsevier, 1990.

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

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

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

[7] 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), pp. 241-250, IEEE Press, 1984.

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

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

[10] J.-C. Régin, "A Filtering Algorithm for Constraints of Difference in CSPs," Proceedings of AAAI 1994, pp. 362-367, AAAI Press, 1994.

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

[12] D. Silver, “Cooperative Pathfinding,” Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117-122, AAAI Press, 2005.
[13] M. Soos, K. Nohl, C. Castelluccia, “Extending SAT Solvers to Cryptographic Problems,” SAT 2009, 12th International Conference (SAT 2009), pp. 244-257, Lecture Notes in Computer Science 5584, Springer, 2009.
[14] 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), pp. 3613-3619, IEEE Press, 2009.
[15] P. Surynek, “Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving,” Proceedings of the 12th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2012), pp. 564-576, Lecture Notes in Computer Science 7458, Springer, 2012.
[16] P. Surynek, “Application of Propositional Satistiability to Special Cases of Cooperative Path-Planning,” Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC 2012), pp. 507-512, IEEE Association, 2012, ISBN 978-1-4673-1486-2.
[17] P. Surynek, “On Propositional Encodings of Cooperative Path-finding,” Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012), pp. 524-531, IEEE Press, 2012.