Lessons Learned from the Effort to Solve Cooperative Path-Finding Optimally

Reductions to Propositional Satisfiability

Pavel Surynek

Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics, Charles University in Prague

Malostranské náměstí 25, 118 00 Praha 1, Czech Republic

pavel.surynek@mff.cuni.cz

Abstract. This paper addresses makespan optimal solving of cooperative path-finding problem (CPF) by translating it to propositional satisfiability (SAT). The task in CPF is to relocate a set of agents to given goal locations so that they do not collide with each other. Recent findings indicate that a simple direct encoding outperforms the more elaborate encodings based on binary encodings of multi-value state variables. The direct encoding is further improved by a hierarchical build-up that uses auxiliary variables to reduce its size in this work. The conducted experimental evaluation shown that the simple design of the encoding together with new improvements which reduced its size significantly are key enablers for faster solving of the encoded CPFs than with existing encodings. It has been also shown that the SAT based methods dominates over A* based methods in environments with high occupancy by agents.

Keywords: cooperative path-finding (CPF), propositional satisfiability (SAT), SAT encodings, A*

1 Introduction, Motivation, and Related Works

The problem of *cooperative path-finding* (CPF) [13, 18, 20, 24] represents an abstraction for variety of problems where the task is to relocate some physical agents, robots, or other objects so that they do not collide with each other. Each agent is given its initial position in a certain environment and its task is to reach a

This work is supported by the Czech Science Foundation (contract no. GAP103/10/1287).

given goal position. It is assumed that all the agents are the same (same size and velocity) and are controlled centrally.

The major difficulty in CPF comes from possible interactions among relocated agents, which is imposed by the requirement that they must not collide with each other. The more agents appear in the instance the more complex interaction arises and consequently the instance is harder to solve.

The commonly adopted way to abstract the CPF problem is to model the environment in which agents are moving as an undirected graph where agents are placed in its vertices. Edges in the graph model the topology of the original environment – that is, an edge connects two neighboring places so that one can be visited immediately from the other. Physical constraints are captured by the requirement that at most one agent is located in each vertex together with constraints that determine how agents can move. An agent can move into a neighboring vertex provided it is unoccupied at the time of commencing the move. No two agents can enter the same target vertex simultaneously. These constraints together ensure that at most one agent is placed in each vertex in the next time step. Other conditions on how motions of agents are enabled are possible but the mentioned one is one of the most frequently used.

The notion of *makespan* in this abstraction corresponds to the number of time steps needed to relocate all the agents (equivalently the makespan is the maximum number of moves needed by an agent to relocate itself to its goal location over the set of all the agents).

One of the very successful approaches to solve CPF optimally is to translate makespan bounded CPF instance to propositional formula and use modern SAT solvers to solve the resulting formula. The advantage of this approach is that all the techniques implemented in SAT solvers such as learning and constraint propagation are employed almost for free in CPF solving. The key research question here is how to encode CPF instance as propositional formula so that the resulting formula can be solved fast.

Lessons learned from the design of various encodings of CPF as propositional are utilized in the design of a new encoding, which is surprisingly simple and efficient. It has been found that there is a correlation between the size of the formula and the speed of its solving in the domain dependent case of CPF (this is correlation however does not appear generally as there exist small formulae that are hard to solve). At the same time formulae modeling CPF that support *constraint propagation* (unit propagation) were found to be solvable more easily. Constraint propagation can be enabled by short clauses when the formula is expressed in conjunctive normal form. Combining these aspects in the design of CPF encodings as SAT is an interesting challenge.

Some propositional encodings of CPF will be discussed. Two most prominent encodings are based on the *all-different propagator* known from constraint programming and on *matchings in bipartite graphs*, which is based on the finding that a valid movement of agents corresponds exactly to the series of matchings in the time-expanded graph of the CPF instance. All the suggested encodings are however still far from the theoretical lower bound on the size of the formula, which offers room for further research.

There are many **motivations** for introducing CPF. Classical multi-robot relocation problems where agents are represented by actual mobile robots can be viewed as CPF. Planning movements of units in real-time strategy games is another application [24]. Even data relocation in a network can be regarded a CPF (agent is represented by a data packet and spatial occupancy turns into storage occupancy).

The indifference between agents in terms of their properties allows abstraction where the environment is modeled as an undirected graph and agents as items placed in vertices of this graph [20, 24]. At most one agent is placed in each vertex. The time is discrete and the move is possible only into a currently unoccupied vertex while no other agent is allowed to enter the same target vertex.

Contemporary approaches to solving CPF include polynomial time sub-optimal algorithms [13, 25] as well as methods that generate optimal solutions in certain sense [21, 22]. This work focuses on generating *makespan optimal* solutions to CPF where the makespan is the maximum of arrive times over all the agents. **Related** makespan optimal methods for CPF currently include methods employing translation of CPF to *propositional satisfiability* (SAT) [22, 23], methods based on *conflict resolution* between paths for individual agents [19], and classical A^* *based methods* equipped with powerful heuristics [21]. The first mentioned approach excels in relatively small environments with high density of agents while latter two approaches are better in large environments with few agents.

This work tries to contribute to SAT-based methods. It is inspired by our recent (unpublished) findings that quite complex and elaborate propositional encodings called INVERSE and ALL-DIFFERENT proposed in [22] and [23] can be easily outperformed by an encoding of a straightforward design. The direct encoding design is further simplified here by introducing auxiliary variables. The introduces simplifications reduced the size of the encoding significantly which in turn enabled faster solving of CPFs encoded using the proposed encoding. It is also shown how the SAT-based solving stands in comparison with A* based methods.

The **organization** of the paper is as follows. The CPF problem is introduced formally first. Then a theoretical study of sizes of CPF encodings is provided. A novel propositional encoding of CPF is described thereafter and its theoretical properties are summarized. Experimental evaluation in which existing encodings and the A* based method are compared with the novel encoding constitute the last part.

2 Cooperative Path Planning 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 $V = \{v_1, v_2, \ldots, v_n\}$ 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 $\alpha^{-1}: V \longrightarrow A \cup \{\bot\}$ will provide us an agent located in a given vertex or $\bot$ 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. □

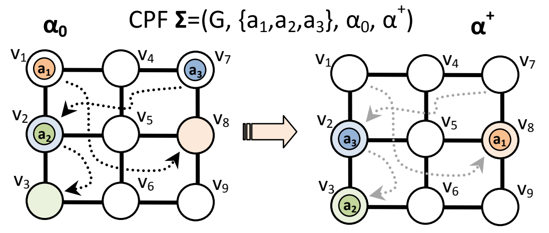
The dynamicity of the model supposes a discrete time divided into time steps. An arrangement α_i at the i-th time step can be transformed by a transition action which instantaneously moves agents in the non-colliding way to form a new arrangement α_{i+1}. The resulting arrangement α_{i+1} must satisfy 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 (1)
(agents move along edges or not move at all),

(ii) $\forall a \in A \;\; \alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow \alpha_i^{-1}(\alpha_{i+1}(a)) = \bot$ (2)
(agents move to vacant vertices only), and

(iii) $\forall a, b \in A \;\; a \neq b \Rightarrow \alpha_{i+1}(a) \neq \alpha_{i+1}(b)$ (3)
(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_+. An illustration of CPF and its solution is depicted in Figure 1.



$\vec{s}$	α_0	α_1	α_2	α_3	$\alpha_4 = \alpha^+$
a_1	v_1	v_1	v_2	v_5	v_8
a_2	v_2	v_3	v_3	v_3	v_3
a_3	v_7	v_4	v_4	v_1	v_2

Figure 1. ***Cooperative path-finding (CPF)*** *on a 4-connected grid.* The task is to relocate three agents a_1, a_2, and a_3 to their goal vertices so that they do not collide with each other. A solution $\vec{s}$ of makespan 4 is shown.

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 $= 1, 2, \dots, m - 1$. □

If it is a question whether there exists a solution of Σ of the makespan at most a given bound η we are speaking about a *bounded CPF (bCPF)*. It is known that bCPF is NP-complete and finding makespan optimal solution to CPF is NP-hard [15].

3 Theoretical Analysis of SAT Encodings of CPF

The goal is to build a propositional formula $F(\Sigma,\eta)$ for a given bCPF Σ and a makespan bound η so that $F(\Sigma,\eta)$ has a model (is satisfiable) if and only if Σ has a solution of makespan η. A sequence of arrangements of agents $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_\eta$ over the graph forming the solution should be readable from the model of $F(\Sigma,\eta)$. The idea of *time expansion graph* [1, 12] is adopted to construct such formula.

If it is known how many propositional variables are needed to express sequence of consecutive arrangements of agents over the graph, then it can seen how far from these bounds the suggested encoding is. Following propositions summarize estimations of the number of necessary propositional variables considering various approaches to express the arrangements. The presented estimations assume that almost every arrangement is possible at any time step.

Also, encodings need to be considered as sparse representations. Otherwise it would be possible to count the total number of distinct sequences of arrangements and take binary logarithm of this number as the estimation. Such an encoding is however impractical as it is hard to decode.

Proposition 1 (LOCATION-ESTIMATION). *Let η be a makespan bound. Then $\eta \cdot \mu \cdot \lceil \log_2 n \rceil$ propositional variables are sufficient to express consecutive arrangements of agents up to the time step η.* ■

Proof. A technique of expressing a multi-value state variable using vectors of propositional variables that encode individual values as binary numbers will be used. There are n possible states of an agent at every time – the agent can appear in any vertex of the input graph. To represent an n-state variable, $\lceil \log_2 n \rceil$ bits (propositional variables) are needed. Hence, we have $\eta \cdot \mu \cdot \lceil \log_2 n \rceil$ propositional variables in total to represent locations of all the agents (there are μ agents) at every time step. ■

Proposition 2 (INVERSE- ESTIMATION). *Consecutive arrangements of agents up to the time step η can be expressed by $\left\lceil \frac{\eta}{2} \right\rceil \cdot n \cdot \lceil \log_2 \mu \rceil + \eta \cdot n$ propositional variables.* ■

Proof. Instead of expressing where each agent is located, the content of vertices will be recorded. The crucial observation is that at most $\lceil \eta/2 \rceil$ changes of agents can occur in a single vertex (an agent must leave the vertex after which another agent can enter the vertex – the change consumes 2 time-steps). Information what agent entered the vertex is again multi-value state variable with μ states. Hence, $\lceil \log_2 \mu \rceil$ bits are needed to record it. Altogether, $\lceil \eta/2 \rceil \cdot n \cdot \lceil \log_2 \mu \rceil$ bits are needed to record possible changes for all the agents. Additional η bits per vertex indicate time-steps at which the change in the vertex occurred.■

The INVERSE encoding [22] partially use the idea presented in the proof. However, the content of vertices is recorded for all the time-steps (not only for half of them as here). An interesting way to represent arrangements of agents at all the time-steps is to record changes between consecutive arrangements while only the initial arrangement is recorded completely.

Proposition 3 (NEIGHBORHOOD-ESTIMATION). *Let $\delta = max_{v \in V}\ deg(v)$ and let the initial arrangement α_0 be expressible using $a \in \mathbb{N}$ propositional variables, then consecutive arrangements of agents up to the time step η can be expressed by $a + \eta \cdot \mu \cdot \lceil \log_2(\delta + 1) \rceil$ propositional variables.* ■

Proof. The idea is to record what move has been taken by each agent. Assume that neighbors of each vertex in G have a fixed order, then the move of an agent can be encoded as an order number of the neighbor into which it moved. Assuming that the degree (the number of edges incident with the given vertex) of all the vertices in G is at most δ, the move of an agent can be recorded by $\lceil \log_2(\delta + 1) \rceil$ bits (an extra state is needed to represent the no move action). Altogether, $\eta \cdot \mu \cdot \lceil \log_2(\delta + 1) \rceil$ bits are sufficient to record moves of all the agents at all the time steps. ■

The estimation gives good results in sparse graphs since each vertex has few neighbors in such a case. This property is partially used in the INVERSE encoding again. However, the estimation may degenerate up to the location-based estimation if the graph is highly connected.

Note that up to now two of three options of how to regard the 3-dimensional space of vertices, agents, and time-steps have been discussed. It remains to show an estimation in which we ask at what time-steps a given agent appears in given vertex. As a single agent may enter a vertex multiple times, the simple scheme in which multi-value state variable representing the third dimension is indexed by remaining two dimensions can no longer be used. Little of the structural properties of the CPF problem can be used in the estimation.

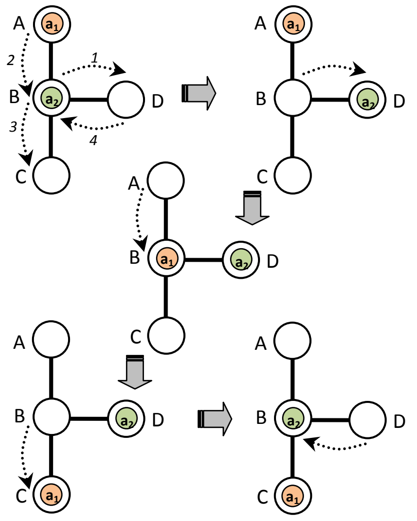


Figure 2. *Illustration of the **minimum number of time steps** before returning to the same vertex.* The next visit to the vertex can be separated by at least 4 time steps.

Proposition 4 (TIME-BASED ESTIMATION). *Consecutive arrangements of agents in an optimal solution up to the time step η can be expressed by $\left\lceil\frac{\eta}{4}\right\rceil \cdot n \cdot \mu \cdot \lceil\log_2 \eta\rceil$ propositional variables.* ■

Proof. The most important observation is that at least 4 time steps are allowed to elapse before an agent returns into a given vertex in a makespan optimal solution. Step 1 is for leaving the vertex, steps 2, 3 are for entering and leaving by the other agent, and step 4 is for returning to the vertex – the situation is illustrated in Figure 2. If the agent returns earlier, then the movement can be eliminated from the solution without compromising its optimality or correctness. Hence, a single agent can visit the given vertex at most $\lceil\eta/4\rceil$ times. Expressing the time step, at which the visit occurs, needs $\lceil\log_2 \eta\rceil$ bits. All these information are recorded for every vertex and every agent, which in total gives $\lceil\eta/4\rceil \cdot n \cdot \mu \cdot \lceil\log_2 \eta\rceil$ bits. ■

Other measures and characteristics than the size of the encoding are difficult to be captured as the dependence of behavior of SAT solvers on the structure of the formula is too complex.

4 A Simplification of Simple SAT Encoding

Let us recall a so called DIRECT encoding of bCPF $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ with makespan bound η where $V = \{v_1, v_2, \dots, v_n\}$ and $A = \{a_1, a_2, \dots, a_\mu\}$ with $n, \mu \in \mathbb{N}$. The DIRECT encoding is part of our unpublished work. As discussed in the previous section, arrangements of agents over the graphs at all the time steps from 1 to μ will be represented. The encoding will use a propositional variable for each vertex, agent, and a time step which will be assigned $TRUE$, if and only if the given agent appears in a given vertex at given time step.

Unlike representations of arrangements using binary encoding of multi-value state variable, this encoding has a propositional variable for every state. Although more propositional variables are needed to encode arrangements, we expect that the benefit of better Boolean constraint propagation outweighs the larger size of the encoding. The suggested encoding will be called DIRECT and is formally introduced in the following definition.

Definition 3 (DIRECT ENCODING). A DIRECT encoding of a given bCPF $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ with makespan bound η consists of propositional variables $\mathcal{X}_{j,k}^i$ for every $i = 0,1, \dots, \eta$, $j = 0,1, \dots, n$, $k = 0,1, \dots, \mu$. The interpretation is that $\mathcal{X}_{j,k}^i$ is assigned $TRUE$ if and only if a_k appears in v_j at time step i. The following constraints modeling validity conditions on consecutive arrangements are introduced:

(a) $\bigwedge_{j,l=1, j<l}^{n} \neg\mathcal{X}_{j,k}^i \vee \neg\mathcal{X}_{l,k}^i$ for every $i \in \{0,1, \dots, \eta\}$, (4)
$\bigvee_{j=1}^{n} \mathcal{X}_{j,k}^i$ and $k \in \{1,2, \dots, \mu\}$
(an agent is placed in exactly one vertex at each time step)

(b) $\bigwedge_{k,h=1,k<h}^{\mu} \neg\mathcal{X}_{j,k}^{i} \vee \neg\mathcal{X}_{j,h}^{i}$ for every $i \in \{0,1,\dots,\eta\}$, and $j \in \{1,2,\dots,n\}$ (5)

(<u>at most one agent is placed in each vertex at each time step</u>)

(c) $\mathcal{X}_{j,k}^{i} \Rightarrow \mathcal{X}_{j,k}^{i+1} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i+1}$ for every $i \in \{0,1,\dots,\eta-1\}$, (6)

$\mathcal{X}_{j,k}^{i+1} \Rightarrow \mathcal{X}_{j,k}^{i} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i}$ $j \in \{1,2,\dots,n\}$, and $k \in \{1,2,\dots,\mu\}$

(<u>an agent relocates to some of its neighbors or makes no move</u>)

(d) $\mathcal{X}_{j,k}^{i} \wedge \mathcal{X}_{l,k}^{i+1} \Rightarrow \bigwedge_{h=1}^{\mu} \neg\mathcal{X}_{l,h}^{i} \wedge \bigwedge_{h=1}^{\mu} \neg\mathcal{X}_{l,h}^{i+1}$ (7)

for every $i \in \{0,1,\dots,\eta-1\}$, $j,l \in \{1,2,\dots,n\}$
such that $\{v_j,v_l\} \in E$ and $k \in \{1,2,\dots,\mu\}$

(<u>target vertex of a move must be vacant and the source vertex will be vacant after the move is performed</u>). □

Observe that all the constraints are now written as *clauses* (disjunctions of *literals*, where literal is a variable or its negation) or can be easily rewritten as clauses. Thus, a *conjunctive normal form* (*CNF*) [5] can be easily obtained from Definition 3. The resulting formula modeling existence of solution of given bCPF Σ with makespan bound η in the CNF form will be denoted as $F_{DIR}(\Sigma,\eta)$.

The DIRECT encoding has a significant drawback, which is its size. Particularly (d) constraints produce too many clauses – (d) constraints stand for $2 \cdot \eta \cdot |E| \cdot |A|$ ternary clauses. Using auxiliary variables, the number can be reduced to $2 \cdot \eta \cdot |E|$.

The DIRECT encoding can be improved in another way as well. Constraints (a) can be eliminated without compromising equisatisfiability of bCPF Σ with η and $F_{DIR}(\Sigma,\eta)$. Omitting (a) constraints may cause that a single agent appears multiple times in the graph. Nevertheless, populating a single only makes it harder to find a solution thus does not matter if occurs. The same can be done with the second implication in (c) constraints. Again, it does not compromise equisatisfiability if it is omitted. The absence of constraints may cause appearance of an agent from nothing at a certain time step. Nevertheless, the first implication propagates all the agents towards the last time steps. Hence, extra-appeared agents just only make it harder to find a solution. Note that omitted constraints are not entailed by the rest at the logical level. The equisatisfiability after omitting mentioned constraints need to be seen at the abstract level of the solution existence in bCPF. The resulting encoding will be called SIMPLIFIED and is formally introduced in the following definition.

Definition 4 (SIMPLIFIED ENCODING). A SIMPLIFIED encoding of a given bCPF $\Sigma = [G = (V,E), A, \alpha_0, \alpha_+]$ with makespan bound η consists of propositional variables $\mathcal{X}_{j,k}^{i}$ and $\mathcal{E}_{j}^{i}$ (auxiliary for macros) for every $i = 0,1,\dots,\eta$, $j = 0,1,\dots,n$, $k = 0,1,\dots,\mu$. The interpretation is that $\mathcal{X}_{j,k}^{i}$ is assigned $TRUE$ if and only if a_k appears in v_j at time step i and $\mathcal{E}_{j}^{i}$ is $TRUE$ if and only if v_j is vacant at time step i. The following constraints modeling validity conditions on consecutive arrangements are introduced:

(A) $\bigwedge_{k,h=1,k<h}^{\mu} \neg\mathcal{X}_{j,k}^{i} \vee \neg\mathcal{X}_{j,h}^{i}$ for every $i \in \{0,1,\dots,\eta\}$, and $j \in \{1,2,\dots,n\}$ (8)

(at most one agent is placed in each vertex at each time step)

(B) $\mathcal{X}_{j,k}^{i} \Rightarrow \mathcal{X}_{j,k}^{i+1} \vee \bigvee_{l:\{v_j,v_l\}\in E} \mathcal{X}_{l,k}^{i+1}$ for every $i \in \{0,1,\dots,\eta-1\}$, $j \in \{1,2,\dots,n\}$, and $k \in \{1,2,\dots,\mu\}$ (9)

(an agent relocates to some of its neighbors or makes no move)

(C) $\mathcal{X}_{j,k}^{i} \wedge \mathcal{X}_{l,k}^{i+1} \Rightarrow \mathcal{E}_{l}^{i} \wedge \mathcal{E}_{j}^{i+1}$ (10)

for every $i \in \{0,1,\dots,\eta-1\}$, $j,l \in \{1,2,\dots,n\}$ such that $\{v_j,v_l\} \in E$ and $k \in \{1,2,\dots,\mu\}$

(target vertex of a move must be vacant and the source vertex will be vacant after the move is performed)

(D) $\mathcal{E}_{j}^{i} \Rightarrow \bigwedge_{h=1}^{\mu} \neg\mathcal{X}_{j,h}^{i}$ for every $i \in \{0,1,\dots,\eta\}$, $j \in \{1,2,\dots,n\}$ (11)

(empty vertex macro connected through auxiliary variable). □

Again, all the constraints of the SIMPLIFIED encoding can be written as clauses. Let the resulting formula for bCPF Σ with makespan bound η be denoted $F_{SIM}(\Sigma,\eta)$.

Properties of the DIRECT and SIMPLIFIED Encodings

Let us summarize basic properties of the proposed SIMPLIFIED encoding. The fact that the encoding does what is what designed for is summarized in the following proposition. Further, a discussion is devoted to the size of the encoding.

Proposition 5 (ENCODING SOUNDNESS). *Let $\Sigma = [G, A, \alpha_0, \alpha_+]$ be a bCPF and η a makespan bound, then $F_{DIR}(\Sigma,\eta)$ as well as $F_{SIM}(\Sigma,\eta)$ is satisfiable if and only if Σ has a solution of makespan η. Moreover, the solution of Σ can be reconstructed from the model of $F_{DIR}(\Sigma,\eta)$ or from $F_{SIM}(\Sigma,\eta)$.* ■

The proof is omitted for space limitations. Nevertheless, note that the second part of the proposition is important to be stated as it is possible to establish equi-satisfiability between a propositional formula and Σ even for trivial cases of the formula from which a solution of Σ cannot be reconstructed.

Proposition 6 (ENCODING SIZE). *Let $\Sigma = [G = (V,E), A, \alpha_0, \alpha_+]$, where with a bound η be an instance of bCPF. The DIRECT encoding $F_{DIR}(\Sigma,\eta)$ requires*

$$(\eta+1)\cdot|A|\cdot|V| \tag{12}$$

$$(\eta+1)\cdot\left(\binom{|V|}{2}\cdot(|A|+1)+\binom{|A|}{2}\cdot|V|\right)+\eta\cdot|A|\cdot(|V|+2\cdot|E|)$$

propositional variables *and* ***clauses*** *respectively.*

The SIMPLIFIED encoding $F_{SIM}(\Sigma,\eta)$ requires

$$(\eta+1)\cdot(|A|+1)\cdot|V| \tag{13}$$

$$(\eta+1)\cdot\left(|V|\cdot|A|+\binom{|A|}{2}\cdot|V|\right)+\eta\cdot|A|\cdot(|V|+2\cdot|E|)$$

propositional variables *and* ***clauses*** *respectively*. ■

Proof. Let us investigate the DIRECT encoding first. The number of propositional variables can immediately seen from the scope of indexes. The number of clauses appearing in (4), (5), and (6) can be calculated as a product of the size of index scopes. Clauses in (7) will develop into $2\cdot|A|$ ternary clauses. An analogical calculation can be done for the SIMPLIFIED encoding. ■

Note that most of clauses in the DIRECT encoding are binary or ternary which supports good performance of Boolean constraint propagation (unit propagation [5]). The same holds for the SIMPLIFIED encoding.

Asymptotically, the number of variables is $\mathcal{O}(\eta\cdot|V|\cdot|A|)$ and the number of clauses is $\mathcal{O}(\eta\cdot|V|^2\cdot|A|)$ in the DIRECT encoding. If this is compared with the INVERSE encoding [22] where the number of variables is reported to be $\mathcal{O}(\eta\cdot|V|^2\cdot\log_2|A|)$ and the number of clauses $\mathcal{O}(\eta\cdot|V|^2\cdot\log_2|A|)$ and with the ALL-DIFFERENT encoding [23] where the number of variables is $\mathcal{O}(\eta\cdot\log_2|V|\cdot|A|^2)$ and the number of clauses is $\mathcal{O}(\eta\cdot\log_2|V|\cdot|V|^2)$, then the size of the DIRECT encoding is larger approximately by the factor of $\mathcal{O}\left(\frac{|A|}{\log_2|A|}\right)$ or $\mathcal{O}\left(\frac{|V|}{\log_2|V|}\right)$ respectively (note that $|A|$ is dominated by $|V|$). However, the size of clauses is much smaller in case of DIRECT encoding.

The asymptotic number of variables is $\mathcal{O}(\eta\cdot|V|\cdot|A|)$ and the number of clauses is $\mathcal{O}(\eta\cdot|A|\cdot\max\{|V|\cdot|A|,|E|\})$ in the SIMPLIFIED encoding. Although factor $\max\{|V|\cdot|A|,|E|\}$ in the number of clauses may be up to $|V|^2$, it is smaller in typical CPF instances. Hence in theory, $F_{SIM}(\Sigma,\eta)$ will be smaller than $F_{DIR}(\Sigma,\eta)$ typically.

5 SAT-Based Optimal CPF Solving

The suggested SIMPLIFIED encoding is intended for makespan optimal CPF solving. As it is possible to solve bCPF with given makespan bound η by translating it to SAT, an optimal makespan and corresponding solution can be obtained using multiple queries to a SAT solver with encoded bCPF. Various strategies exist for getting the optimal makespan. The simplest one and very efficient one at the same time is to try sequentially makespan bounds $\eta=1,2,\ldots$ until η equal to the optimal makespan is encountered. This strategy will be further referred as *sequential increasing*. The sequential increasing strategy is also used in domain independent planners such as SATPLAN [12], SASE [11] and others. Pseudo-code of the strategy is listed as Algorithm 1.

The focus here is on SAT encoding while querying strategies are out of scope of the paper; though let us mention that in depth study of querying strategies is given in [17]. There is a great potential in querying strategies as they can bring speedup of planning process in orders of magnitude, especially when combined with parallel processing.

Any complete SAT solver [5] may be used as the external module of the suggested optimal CPF solving algorithm. Notice however that a CPF solver following the framework of Algorithm 1 is *incomplete*. If the given CPF instance Σ has no solution then the algorithm runs infinitely. The treatment of incompleteness is easy. The solvability of Σ can be checked by some of sub-optimal polynomial time solving algorithms such as that suggested in [13] or by PUSH-AND-ROTATE [25] (which corrects the previous algorithm [14]) before optimal SAT solving is started. The speed of the solving process is not compromised by solving the instance sub-optimally first since the runtime of solving encoded bCPFs by a SAT solver significantly dominates in the overall runtime.

Algorithm 1. SAT based optimal CPF solving.
input: a CPF instance Σ
output: a pair consisting of the optimal makespan and corresponding optimal solution

```
function Find-Optimal-Solution-Sequentially ($\Sigma = (G, A, \alpha_0, \alpha^+)$): pair
1:   $\eta \leftarrow 1$
2    loop
3:       $F(\Sigma,\eta) \leftarrow$ Encode-CPF-as-SAT ($\Sigma,\eta$)
4:       if Solve-SAT ($F(\Sigma,\eta)$) then
5:           $s \leftarrow$ Extract-Solution-from-Valuation($F(\Sigma,\eta)$)
6:           return $(\eta, s)$
7:       $\eta \leftarrow \eta + 1$
8:   return $(\infty, \emptyset)$
```

6 Experimental Evaluation

The proposed SIMPLIFIED encoding has been competitively evaluated with respect to other existing two propositional encodings of bCPF called INVERSE [22] and ALL-DIFFERENT [23] and with respect to the unpublished DIRECT encoding. Various static characteristics of encodings such as its size and runtime behavior, when the encoding is built-in to the SAT-based optimal CPF solving, were compared. The SAT-based solving with all the encodings has been compared with another state-of-the-art method developed around A* algorithm called OD+ID [21]. The comparison with OD+ID interestingly extends results shown in [22, 23] where SAT-based CPF solving was compared with domain independent SAT-based planners SASE [11] and SATPLAN [12] only.

The experimental setup employs random CPF instances over 4-connected grids with randomly placed obstacles. This is a standard benchmark for evaluating CPF solving methods suggested in [20]. Although it is not very general, it provides easy comparison with results in existing literature. Initial locations and goals of agents were distributed randomly over the grid. Grids of sizes 6×6, 8×8, and

12×12 were used in experiments; 10% were occupied by obstacles. All CPF the instances in the evaluation were solvable.

All the encodings were further augmented with *reachability heuristic* as proposed in [23]. That is, locations that are unreachable from the initial position or from the goal in the given number of time steps are forbidden and associated constraints (clauses) are omitted. This heuristic significantly reduces the size of all the encodings and speeds up the solving process.

`Glucose` version `3.0` [1] SAT solver has been used in the experimental evaluation. According to the 2013 SAT Competition [3] `Glucose` is one of few top SAT solvers in terms of performance in solving hard combinatorial problems. As CPF can be regarded as a combinatorial problem, this choice of SAT solver is justified.

Let us also briefly summarize properties of all the encodings and A* based method used in competitive comparison. The IVERSE encoding is build around the *inverse location function* α^{-1}, which is used to represent arrangements of agents. The primary motivation is to keep the encoding as small as possible, hence it uses multi-value state variables encoded by bit vectors, which eliminates additional constraints. However, no attention is paid to Boolean constraint propagation. Relatively long clauses appear in the encoding.

The ALL-DIFFERENT encoding on the other hand uses standard *location function* α to represent arrangements of agents. Again, locations are modeled as bit-vectors. The main idea is to express the requirement that each agent must occupy a unique location by the *all-different* relation over bit vectors [4]. Note that no propagator based on network flows, as it is known in constraint programming [16], is used here. Boolean constraint propagation is considered at the level of the all-different relation, which is designed in that sense [4]. However, the encoding of the all-different relation grows as quadratically which makes the encoding complicated for higher number of agents.

OD+ID is a search method. It is always trying to separate agents into groups for which shortest paths to their goals can be found independently on other agents. The best case occurs if each agent is in its own group. That is, the cooperative solution consists of shortest paths between initial positions and goals of agents. The worst case is on the other hand if all the agents are considered as a single group. If agents become too interdependent in the densely occupied environment, the performance of the method considerably degrades.

All the source codes used to conduct experiments are posted on website to allow full reproducibility of presented results: http://ktiml.mff.cuni.cz/~surynek/ research/cjs2014.

Static Evaluation of Encodings

There are several static characteristics of propositional formulae in CNF that are correlated with performance of their solving by most SAT solvers. Obviously, the size of the formula in terms of the *number of variables* and the *number of clauses* determines the time needed to find a solution significantly. Typically the larger the formula is the longer runtime of its solving should be expected though the solving runtime is also affected by other factors (easily solvable large formula is possible).

How the formula is *constrained* determines the difficulty of its solving as well. It is known that in random 3-SAT case (clauses has exactly three literals) the most difficult formulae have the *ratio of the number of clauses to the number of variables* around 4.24 [5] which is called a *phase transition*. Formulae that are over-constrained (tend to be unsatisfiable) or under-constrained (tend to be satisfiable), that is, have the clause to variable ratio far from the phase transition can be solved easily in most cases. Hence, encodings producing such formulae are preferred. Another important characteristic is the *length of clauses* while short clauses are preferred. This is implied by the *Boolean constraint propagation* represented by the *unit propagation* [5], which always assigns $TRUE$ to the last unassigned literal in a clause during search. Short clauses promote frequent use of unit propagation, which significantly speeds up the process of solving the formula [6].

Table 1. ***Static characteristics*** *of encodings over 8×8 grid.* INVERSE, ALL-DIFFERENT, DIRECT, SIMPLIFIED and encodings – all with compiled distance heuristics [23] – are compared. bCPF instances are generated over the 4-connected grid of size 8×8 with 10% of cells occupied by obstacles. Makespan bound η is always 16. The number of variables and clauses, the ratio of the number of clauses and the number of variables, and the average clause length are listed for different sizes of the of agents A. The DIRECT encoding is biggest in terms of the length of formula but has smallest clauses in average and is most constrained out of all the encodings, which suggests good behavior in Boolean constraint propagation (unit propagation) and pushes the formula far from the *phase transition*. The SIMPLIFIED encoding inherited a promising clause to variable ratio and short clauses from the DIRECT encoding.

Grid 8×8 / **|Agents|**			INVERSE		ALL-DIFFERENT		DIRECT		SIMPLIFIED	
1	#Variables	*Ratio*	8 358.7	*3.748*	1 489.3	*5.325*	**814.4**	***28.539***	1 628.8	*2.078*
	#Clauses	*Length*	31 327.9	*2.616*	7 930.4	*3.057*	23 241.9	***2.149***	**3 384.6**	*2.550*
4			10 019.5	*5.532*	7 834.5	*4.440*	**3 257.6**	***35.589***	4 072.0	*4.420*
			55 437.0	*2.641*	34 781.9	*3.103*	115 934.3	***2.272***	**17 997.8**	*2.374*
16			**11 680.3**	*7.820*	67 088.3	*3.231*	13 030.4	***64.506***	13 844.8	*10.853*
			91 344.5	*3.127*	216 745.4	*3.147*	840 540.6	***2.505***	150 259.2	*2.180*
32			**12 510.7**	*9.765*	230 753.0	*2.802*	26 060.8	***105.084***	26 875.2	*19.002*
			122 170.3	*3.733*	646 616.2	*3.168*	2 738 584.7	***2.621***	510 672.1	*2.111*

Note that preferences in the mentioned characteristics are sometimes contradictory. We want the formula to be small, hence the number of clauses should be small, but over-constraining by many clauses is desirable at the same time. Therefore, discussed preferences should not be considered literally – also because the considered behavior of SAT solvers appears in most cases but does not appear absolutely. They are rather a simple guidance that was kept in mind when an encoding is designed.

Static characteristics of the SIMPLIFIED encoding are compared with other three propositional encodings – INVERSE, ALL-DIFFERENT, and DIRECT – in Table 1. A 4-connected grid of size 8×8 various numbers of agents is shown. The winner according to discussed preferences in each characteristic is shown in bold (results over 6×6 and 12×12 indicate same conclusions and thus are omitted for space limitations).

It can be observed that the smallest encoding in terms of the number of variables and clauses is the INVERSE one while the biggest one is the DIRECT encoding with ALL-DIFFERENT and SIMPLIFIED encodings standing in the middle. This aspect seems to be disadvantageous for the proposed DIRECT encoding. However in terms of the clause to variable ratio and the size of clauses, the DIRECT encoding seems to be the best followed by the SIMPLIFIED one as they have highest number of shortest clauses. Whether these static advantages prevail over disadvantages of encodings, must be evaluated in runtime experiments.

Runtime Evaluation of Encodings

The speed of SAT-based optimal CPF solving with the three discussed encodings has been evaluated. Again, 4-connected grids of various sizes were used in experiments. The *runtime*[2] needed for finding an optimal solution has been measured for the number of agents ranging from 1 up to the number for which at least one method solves all the instances (the increasing number of agents makes the CPF instance more difficult). The timeout of 256 seconds has been used. For each number of agents, 10 random instances of bCPF have been generated and solved. All the testing instances were solvable. The average runtime and makespan is reported.

Evaluation of SAT-based CPF solving would be incomplete if it is not compared with other state-of-the-art solving methods. Therefore A*-based method OD+ID [17] is included into competitive comparison.

Runtime results are shown in Figure 3. Also average optimal makespans are shown for the sake of completeness. Results for the 8×8 grid indicate that SAT-based solving with the SIMPLIFIED encoding is the best option if the occupancy of the graph with agents is non-trivial, that is > 10%. Almost identical results can be observed in case of 6×6 and 12×12 grids.

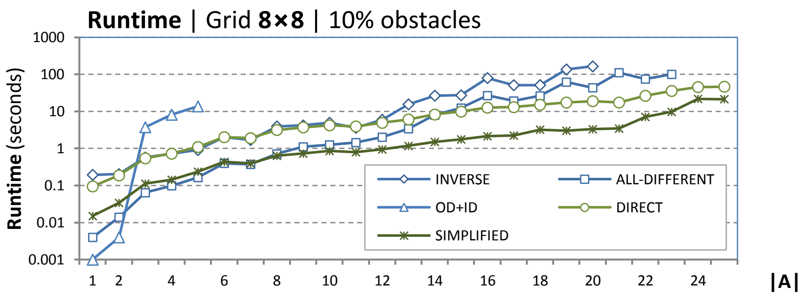
The DIRECT encodings follows the SIMPLIFIED one in term of solving speed. It outperforms remaining two encodings INVERSE and ALL-DIFFERENT if the occupancy of the graph with agents exceeds approximately 30%. The closest competitor to presented encodings seems to the ALL-DIFFERENT encoding which is a better option for less occupied graphs.

In very sparsely occupied graphs, OD+ID method is the best as lot of independence among agents can be found. However, OD+ID degrades dramatically if there is higher concentration of agents in the graph since agents become more interdependent and independence heuristics no longer work.

The INVERSE encoding was always the worst option out of all the tested methods. We consider that the reason for its weak performance is that relatively long clauses appear in it. On the other hand, short clauses of the DIRECT encoding and

[2] All the runtime measurements were done on an experimental server with the 4-core CPU Xeon 2.0GHz and 12GB RAM under Linux kernel 3.5.0-48.

their abundance promoting unit propagation are the main reasons for the good performance of this encoding. We observed that solving of formulae of the DIRECT encoding by the SAT solver is relatively fast while large portion of the time is consumed by generating the formula (the formula is generated into file, which is subsequently read by the SAT solver). Hence, there is still room to increase the speed of SAT-based solving if the solving process is better engineered.



Grid 8×8 |A|	1	2	4	8	12	16	20	24
Makespan	6.4	6.1	8.1	10.5	9.8	11.0	11.9	12.7

Figure 3. ***Runtime*** *of SAT-based CPF solving – grid 8×8.* `Glucose 3.0` is used as an external solver in SAT-based solving. For each number of agents, 10 random instances were solved and the average runtime is reported. The SIMPLIFIED encoding can be solved the fastest for the occupancy with agents greater than 10%. The SIMPLIFIED and the DIRECT encoding are the only encodings for which all the instances have been solved in a given timeout of 256 seconds. Note that the DIRECT encoding becomes faster than the IVERSE and ALL-DIFFERENT encodings when the occupancy exceeds 30%. The average optimal makespan for selected numbers of agents is shown in the table in the bottom (again it is the average makespan out of makespans of 10 instances). Note that OD+ID is fastest for sparsely populated graphs but its increases runtime quickly with higher number of agents.

Discussion, Conclusions, and Future Works

A new propositional encoding of the makespan bounded cooperative path-finding problem (bCPF) has been proposed. The idea of the work was to simplify further our unpublished encoding whose design was very simple with no elaborate technique behind. The next goal was to check how simple encodings stands with respect to existing relatively elaborate encodings for the problem. The new encoding has been called SIMPLIFIED as it simplifies our previous encoding called DIRECT.

The SIMPLIFIED encoding has been used within the SAT-based framework for solving CPF (unbounded version) optimally. The comparison of SIMPLIFIED and DIRECT encodings with existing two encodings INVERSE [22] and ALL-DIFFERENT [23] as well as with A* search based method OD+ID [21] on random CPF instances over 4-connected grids has been done and showed surprising results. The DIRECT encoding despite its relatively naive design performed better than the ALL-DIFFERENT encoding on instances with occupancy by agents $> 30\%$ and almost always better than the INVERSE encoding. The SIMPLIFIED encoding was even better and it prevailed over all the other encodings in case with occupancy higher than approximately 10%.

Generally, the SAT-based approach turned out to be better whatever encoding has been used than the A* based OD+ID whenever occupancy with agents has been higher than trivial. This can be explained by the fact that OD+ID's heuristic cannot detect independence among agents in cases with high occupancy. Note also that this method can be regarded as all-in-one while in the SAT-based approach the SAT solver itself is an external module. It is quite unrealistic to implement equivalent number of propagation, learning, and heuristic techniques in the all-in-one solution as they are in SAT solvers. Through an encoding of the problem in the SAT formalism, we can access all these elaborate techniques almost for free. We hope that lessons learned from the design of encodings for CPF can be applied in other areas as well.

It would be interesting to study models of CPF also in other formalisms than SAT. The promising candidate seems to be *constraint satisfaction problem* (CSP) [7] where *global constraints* can be used. *Integer programming* (IP) model are also worth studying. Recently an attempt to model the problem in the *answer set programming* (ASP) formalism has been made [9]. Also, it seems that existing encodings are still far from theoretically smallest possible sizes. Hence, we see an opportunity in further reductions of the size of SAT encodings.

Another interesting question is if finding *makespan sub-optimal* solution can be modeled as SAT. It is known that sub-optimal methods such as [13, 25] generate solutions that have excessively long makespans. Their shortening is a difficult process as shown in [22]. The optional way may be to propose a SAT encoding that prefers sub-optimal solution of short makespans.

We are also investigating the possibility of *automated generation* of encodings. Our first examination of the problem of finding an encoding automatically indicates that it belongs to Σ_1^P class. Hence, the encoding can be theoretically found by a QBF solver. The open question is if this process is practically feasible.

References

1. Ahuja, R. K., Magnanti, T. L., Orlin, J. B. *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.
2. Audemard, G., Simon, L. *The Glucose SAT Solver.* http://labri.fr/perso/lsimon/glucose/, 2013, [accessed in June 2014].
3. Balint, A., Belov, A., Heule, M., and Järvisalo, M. *SAT 2013 competition.* http://satcompetition.org/, 2013, [accessed in June 2014].

4. 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), pp. 1-4, IEEE Press, 2008.
5. Biere, A., Heule, M., van Maaren, H., Walsh, T. *Handbook of Satisfiability.* IOS Press, 2009.
6. Bjork, M. *Successful SAT Encoding Techniques.* Journal on Satisfiability, Boolean Modeling and Computation, Addendum, IOS Press, 2009.
7. Dechter, R. *Constraint Processing.* Morgan Kaufmann Publishers, 2003.
8. Eén, N., Sörensson, N. *An Extensible SAT-solver.* Proceedings of Theory and Applications of Satisfiability Testing (SAT 2003), pp. 502-518, LNCS 2919, Springer, 2004.
9. Erdem, E., Kisa, D. G., Oztok, U., Schüller, P. *Experimental Evaluation of Multi-Agent Pathfinding Problems using Answer Set Programming.* Proceedings of the 20th International Workshop on Knowledge Representation and Automated Reasoning (RCRA 2013), AI*IA, 2013.
10. Gent, I. P., Walsh, T. *The SAT Phase Transition.* Proceedings of the 11th European Conference on Artificial Intelligence (ECAI 1994), pp. 105–109, John Wiley and Sons, 1994.
11. Huang, R., Chen, Y., Zhang, W. *A Novel Transition Based Encoding Scheme for Planning as Satisfiability.* Proceedings AAAI 2010, AAAI Press, 2010.
12. Kautz, H., Selman, B. *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.
13. Kornhauser, D., Miller, G. L., Spirakis, P. G. *Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications*. Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241-250, IEEE Press, 1984.
14. 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), pp. 294-300, IJCAI/AAAI Press, 2011.
15. Ratner, D., Warmuth, M. K. *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.
16. Régin, J-C. *A Filtering Algorithm for Constraints of Difference in CSPs.* Proceedings of the 12th National Conference on Artificial In-telligence (AAAI 1994), pp. 362-367, AAAI Press, 1994.
17. Rintanen, J., Heljanko, K., and Niemelä, I. *Planning as satisfiability: parallel plans and algorithms for plan search.* Artificial Intelligence, Volume 170 (12-13), pp. 1031–1080, Elsevier, 2006
18. Ryan, M. R. K. *Exploiting Subgraph Structure in Multi-Robot Path Planning.* Journal of Artificial Intelligence Research (JAIR), Volume 31, pp. 497-542, AAA Press, 2008.
19. Sharon, G., Stern, R., Goldenberg, M., Felner, A. *The increasing cost tree search for optimal multi-agent pathfinding.* Artificial Intelligence, Volume 195, pp. 470-495, Elsevier, 2013.
20. Silver, D. *Cooperative Pathfinding.* Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117-122, AAAI Press, 2005.
21. Standley, T. S., Korf, R. E. *Complete Algorithms for Cooperative Pathfinding Problems.* Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), 668-673, IJCAI/AAAI Press, 2011.

22. Surynek, P. *Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving*. Proceedings of 12th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2012), LNCS 7458, pp. 564-576, Springer, 2012.
23. Surynek, P. *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.
24. Wang, K. C., Botea, A. *MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees*. JAIR, Volume 42, pp. 55-90, AAAI Press, 2011.
25. de Wilde, B., ter Mors, A., Witteveen, C. *Push and rotate: cooperative multi-agent path planning*. Proceedings of International conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), pp. 87-94, IFAAMAS, 2013.