APPLICATION OF PROPOSITIONAL SATISFIABILITY TO SPECIAL CASES OF COOPERATIVE PATH-PLANNING

PAVEL SURYNEK[1,2]

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

Abstract:

A problem of cooperative path-planning is addressed from the perspective of propositional satisfiability in this paper. Two new encodings of the problem as SAT are proposed and evaluated. Together with the existent solution optimization method which locally improves a sub-optimal solution of the problem through SAT solving, one of the new encodings constitute a state-of-the-art method for cooperative path-planning in highly occupied environments.

Keywords:

Cooperative path-planning; Multi-robot path-planning; Propositional satisfiability; All-Different constraint

1. Introduction and Motivation

The problem of cooperative path-planning (CPP) [12] 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 CPP on a 4-connected grid is shown in Figure 1).

The problem attracts considerable attention as there are many real-life situations that can be modeled as CPFs. 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.

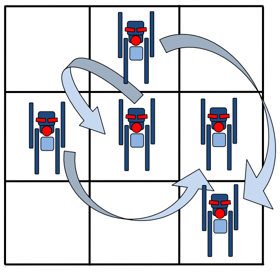
Figure 1. An instance of CPF. Three agents need to relocate in the 4-connected grid 3×3.

A real-time makespan sub-optimal incomplete algorithm WHCA* (A* based) was published in [12]. It actually became the standard in the computer entertainment industry (unit movement in RTS games). Several scalable complete algorithms for solving CPP sub-optimally have appeared recently – BIBOX [14] and PUSH-SWAP [7] represent two most important. Nevertheless, the problem has been addressed from the other side as well. A technique for solving CPP optimally in the case of very sparsely occupied environments has been described in [6]. Several other methods exploiting structural properties of the problem appeared in [11] and [18].

In our work we addressed the case of optimal or near optimal makespan and densely occupied 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. 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 [15], [16]. These (sub-optimal) sub-solutions are subsequently replaced by optimal ones found by the SAT solver.

Particularly in this paper we propose two novel propositional encodings for the CPP problem that can be used within the mentined solution optimization method. An experimental evaluation of encodings with respect to the state-of-the-art methods is presented.

2. Cooperative Path-Planning (CPP) 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 $A = \{a_1, a_2, \ldots, a_\mu\}$ 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 PLANNING). An instance of *cooperative path-planning* 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 planning 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 planning instance $\Sigma = [G, A, \alpha_0, \alpha_+]$ is a sequence of arrangements $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m]$ where $\alpha_m = \alpha_+$ and α_{i+1} is a result of valid transformation of α_i for every $= 1, 2, \ldots, m-1$. □

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 (bCPP)*. It is known that bCPP is NP-complete [8]. Notice that due to no-ops introduced in valid transitions it is equivalent to planning a solution of the makespan equal to the given bound.

3. CPP as Propositional Satisfiability: three encodings

Three different encodings for CPP as satisfiability are presented. All of them are based on the idea used in the domain-independent SAT-based planning represented by SATPLAN [5] and SASE [4] planning systems. That is, the snapshot of planning world called a *layer* is encoded for all the discrete time steps up to the given bound. Between the individual layers semantic constraints are encoded so that changes between layer respects the semantics of planning. Specifically in CPP layers are represented by arrangements of agents at individual time steps. Validity constraints describe allowed changes between layers.

For simplicity, encodings will be introduced on finite domain integer variables. The translation to propositional variables uses Tseitin's [17] hierarchical encoding and techniques from [10] to encode finite domain state variables by the logarithmic number of propositional variables is omitted.

3.1. INVERSE Encoding

The original version of the INVERSE encoding is described in [15]. In this encoding, layers – that is, arrangements of agents – are represented using inverse locations. To represent changes between layers we need to add finite domain integer 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 finite domain integer variable which domain contains one element per each primitive action 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 in the final translation to propositional satisfiability.

Let $\Sigma = [G = (V, E), A, \alpha_0, \alpha_+]$ be an instance of CPP and $k \in \mathbb{N}$ be a makespan bound. The INVERSE encoding has layers $0, 1, \ldots, k$. Suppose that neighboring vertices of a given vertex are ordered in the fixed order. That is, $\forall v \in V$ we have a function $\sigma_v: \{u | \{v, u\} \in E\} \longrightarrow \{1, 2, \ldots, \mathrm{dg}_G(v)\}$ 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**:

- $\mathcal{A}_i^v \in \{0, 1, 2, \ldots, \mu\}$ for all $v \in V$ such that $\mathcal{A}_i^v = j$ iff $\alpha_i(a_j) = v$
- $\mathcal{T}_i^v \in \{0, 1, 2, \ldots, 2\,\mathrm{dg}_G(v)\}$ 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;
  - $\mathcal{T}_i^v = \mathrm{dg}_G(v) + \sigma_v(u)$ iff an incoming primitive action 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 $v \in V$ (**no-op** case); (1)
- $0 < \mathcal{T}_i^v \leq \text{dg}_G(v) \;\; \Rightarrow \mathcal{A}_i^u = 0 \wedge \mathcal{A}_{i+1}^u = \mathcal{A}_i^v \wedge$
  $\mathcal{T}_i^u = \sigma_u(v) + \text{dg}_G(u)$ (2)
  where $u = \sigma_v^{-1}(\mathcal{T}_i^v)$ for all $v \in V$
  (**outgoing** agent case);
- $\text{deg}_G(v) < T_i^v \leq 2 * \text{dg}_G(v) \;\; \Rightarrow \mathcal{T}_i^u = \sigma_u(v)$ (3)
  where $u = \sigma_v^{-1}(\mathcal{T}_i^v - \text{dg}_G(v))$ for all $v \in V$
  (**incoming** agent case). □

Finite domain integer 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. 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 bCPP instance we need to encode the initial and the goal arrangement straightforwardly as follows:

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

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

3.2. Heuristic Pair-Wise All-Different Encoding

Originally, Pair-Wise All-Different (PWAD) encoding has been proposed in [16]. But here we will present a new improved version of it called Heuristic (HPWAD).

Let us first recall the original version. The main idea of this encoding is to use location function to represent arrangements. If the location function is chosen to represent the arrangement within layers 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 variables. The situation is very close to a *bi-clique* [10] 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 constraint [9] 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.

In the Heuristic version of the encoding, we are trying to overcome the difficulty with the mentioned extensive encoding of conditional equalities. We observed that in many cases lot of vertices are unreachable by agents supposed agents want to reach the goal. Those vertices can be ruled out and no conditional constraint is needed to be introduced for them. More precisely, a vertex, that is too far from the initial location of the agent so that the agent cannot reach it in the given number of steps or a vertex that is too far from the goal location so that there is not enough time steps remaining to reach it, can be ruled out from consideration.

The just introduced encoding with distance heuristic reasoning is summarized formally in the following definition.

Definition 4 (Regular Layer – Heuristic AllDifferent). 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 $l \in \{1,2,\ldots,n\}$ such that
  $\text{dist}_G(\alpha_0(a), v_l) \leq i$ and $\text{dist}_G(v_l, \alpha_+(a)) \leq k - i$ (4)
  $\mathcal{L}_i^a = l \Rightarrow \mathcal{L}_{i+1}^a = \ell \vee \bigvee \quad \mathcal{L}_{i+1}^a = \ell$
  (agents can move only **along edges** of G),
- for all $a \in A$ and $l \in \{1,2,\ldots,n\}$ such that
  $\text{dist}_G(\alpha_0(a), v_l) > i$ or $\text{dist}_G(v_l, \alpha_+(a)) > k - i$ (5)
  $\mathcal{L}_i^a \neq l$
  (agents cannot be located in **unreachable** vertices),
- for all $a \in A$
  $\bigwedge \quad \mathcal{L}_{i+1}^a \neq \mathcal{L}_i^b$ (6)
  (the **target vertex** of agent's move must be **empty**),
- and **at most one** agent resides in each vertex:
  $\text{AllDifferent}(\mathcal{L}_i^{a_1}, \mathcal{L}_i^{a_2}, \ldots, \mathcal{L}_i^{a_\mu})$ (7)

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

The last layer is irregular again; there is no propagation constraint to the next layer.

The propositional translation of the All-Different constraint in this encoding follows the scheme presented in [1]. That is, the constraint is encoded as pair-wise inequalities first as follows:

$$\text{AllDifferent}(\mathcal{L}_i^{a_1}, \mathcal{L}_i^{a_2}, \ldots, \mathcal{L}_i^{a_\mu}) \equiv \quad \bigwedge \quad \mathcal{L}_i^{a_i} \neq \mathcal{L}_i^{a_j}$$

Then inequalities over finite domain integer variables are translated to inequalities over bit-vectors.

3.3. Heuristic Bijection All-Different Encoding

The last presented encoding called a Heuristic Bijection All-Different (HBAD) differs from the previous one only in the encoding style of the All-Different constraint. Since encoding of the constraint as many inequalities may reduce its propagation strength in certain cases another encoding style worth considering. It maps variables that are required to be different to a set of linearly ordered variables through a bijection. The target variables have the same domain as the original variables. The linear ordering ensures that the original variables are eventually assigned different values. Intuitively, this encoding can break some symmetries that occur in the pair-wise encoding. Our preliminary experimental evaluation shows that this encoding of the ALL-DIFFERENT constraint performs better near the phase transition with respect to existence and nonexistence of the solution. However, it is a question whether instances of CPP also show such a phase transition. Illustration of the encoding is shown in Figure 2.

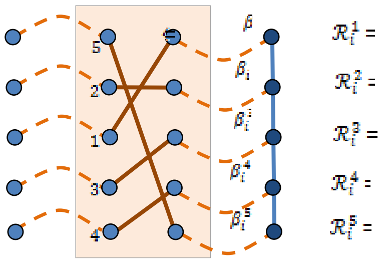

Figure 2. Illustration of the BIJECTION-BASED ALL-DIFFERENT encoding. A 1-to-1 mapping between the location variables $\mathcal{L}_i^{a_1}$, $\mathcal{L}_i^{a_2}$, …, $\mathcal{L}_i^{a_\mu}$ and auxiliary variables $\mathcal{R}_i^1, \mathcal{R}_i^2, \ldots, \mathcal{R}_i^\mu$ is found and linear ordering is imposed over the auxiliary variables.

The previous encoding is altered in the following way. For each occurrence of $\text{AllDifferent}(\mathcal{L}_i^{a_1}, \mathcal{L}_i^{a_2}, \ldots, \mathcal{L}_i^{a_\mu})$ in the model several fresh finite domain integer variables are introduced:

- $\alpha_i^j \in \{1,2,\ldots,\mu\}$, $\beta_i^j \in \{1,2,\ldots,\mu\}$, and $\mathcal{R}_i^j \in \{0,1,\ldots,n\}$ for $j = 1,2,\ldots,\mu$

Location variables $\mathcal{L}_i^{a_j}$ are 1-to-1 mapped to auxiliary variables $\mathcal{R}_i^j$. Other auxiliary variables α_i^j and β_i^j are used to encode the bijection mapping. The mapping itself is encoded by the following constraints:

- $\alpha_i^j = j \Rightarrow \beta_i^j = i$ for $i, j = 1,2,\ldots,\mu$
- $\beta_i^j = i \Rightarrow \mathcal{L}_i^{a_i} = \mathcal{R}_i^j$ for $i, j = 1,2,\ldots,\mu$

The first constraint may be changed to equivalence to improve the propagation strength (in our implementation equivalence is used). Finally, we need to ensure the linear ordering over the auxiliary variables by conjunction of pair-wise inequalities: $\bigwedge_{j=2}^{\mu} \mathcal{R}_i^{j-1} > \mathcal{R}_i^j$.

4. Solution Optimization Process

The proposed encodings are not used to model layers of a CPP instance from its initial state to its goal directly since typically many time-steps are needed to reach the goal in CPP. The requirement to model many time-steps may lead to a too big SAT model that cannot be solved by any SAT solver. Instead we first generate a solution to a given CPP by a suboptimal solving method (makespan sub-optimal solution is generated) like BIBOX [14] or PUSH-SWAP [7]. The obtained sub-optimal solution is subsequently improved so that relatively small sub-sequences of it are replaced by makespan optimal sub-plans.

To generate the optimal sub-plan a corresponding bounded CPP instance is modeled as a SAT using some of the proposed encodings and then it is solved by a SAT solver [3]. Various strategies can be used to find the optimal number of time steps for that the corresponding sub-goal can be reached. More details about the solution optimization method can be found in [15] and [16]. The method described in these works uses binary search to determine optimal number of time steps of the sub-plan. That is, if the SAT formulation is unsolvable for the given number of steps then the number of steps is doubled etc. This strategy allows us to reduce the size of instances that are modeled.

After replacing a given sub-sequence in the original solution, the next sub-sequence is processed. If the entire original solution is processed the optimization starts from the beginning of the solution until the fixed point is reached – the makespan of solution no longer changes. After reaching the fixed point the size of sub-sequences that are tried to be replaced is increased by 1 and whole process is repeated. The solution optimization continues until the time limit is reached or the makespan optimal solution is found. The latter case happens if the size of the sub-sequence to be replaced covers the whole original solution.

5. Experimental Evaluation

We did experimental evaluation in which we check what impact the proposed encodings have on the performance of the described solution optimization process. As the SAT solver we used MINISAT 2.2 [3]. To obtain initial sub-optimal solutions for CPPs we used the BIBOX algorithm [14].

Our benchmark set consists of two environments one small – a 4-connected grid of size 8×8 – and one large – a 4-connected grid of size 16×16. In both environments we tried random instances for the number of agents ranging from 1 to 50 in the case of 8×8 grid and 1 to 128 in the case of 16×16 grid. A random instance is constructed by placing initial and goal position randomly in the environment. This setup is known to produce hard to solve instances [6].

We compared the INVERSE, HPWAD, and HBAD encodings in terms of size of the encoding, quality of generated solutions, and the runtime of the solution optimization method that uses these encodings. To get more complete figure we also added comparison with SASE [4] which is a state-of-the-are domain independent makespan optimal planning system and with WHCA* [12] which is a de-facto standard algorithm for solving CPP.

5.1. Encoding Size Comparison

The comparison of size of encodings is shown in Table 1 and Table 2. It clearly shows that all the proposed domain dependent encodings has fewer propositional variables as well as clauses. The improvement ratio with respect to SASE is up to one order of magnitude in these terms. The most conserving seem to be HPWAD in the case with few agents while the INVERSE encoding dominates for larger number of agents. The HBAD encoding looses in all the cases.

The advantage of HPWAD with respect to the INVERSE encoding is even better in the case of 16×16 grid which can be accounted to the fact that the INVERSE encoding needs to encode the whole environment which is large in this case.

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

$\|A\|$ in the 4-connected grid **8×8**	Number of **layers**	**SASE** encoding		**INVERSE** encoding		**HBAD** encoding		**HPWAD** encoding	
		|Variables|	**|Clauses|**	**|Variables|**	**|Clauses|**	**|Variables|**	**|Clauses|**	**|Variables|**	**|Clauses|**
4	8	11386	53143	5400	38800	11128	54356	2528	10626
8	8	19097	105724	5920	48224	25136	114952	7942	27543
12	8	26857	168875	5920	46176	42024	181788	16026	49535
16	10	51662	372140	8122	76192	79008	326736	38304	119827
24	10	73101	588886	8122	71072	140400	537528	81000	235663
32	14	157083	1385010	12396	137120	309824	1120672	219059	659882

Table 2. Encoding sizes comparison on the grid 16×16. SASE failed to proceed to the goal level for large number of agents. That is why the number of layers is lower than the goal level*.

$\|A\|$ in the 4-connected grid **16×16**	Number of **layers**	**SASE** encoding		**INVERSE** encoding		**PBAD** encoding		**HPWAD** encoding	
		|Variables|	**|Clauses|**	**|Variables|**	**|Clauses|**	**|Variables|**	**|Clauses|**	**|Variables|**	**|Clauses|**
4	21	137406	677737	60755	478462	122368	827628	21980	147136
8	15	134482	712352	46904	412416	178816	1174616	29763	164052
16	18	342100	2347456	61154	611328	469888	2928336	125633	594618
32	4*	288498	2716096	13672	143104	197888	1101600	56725	144146
40	4*	357762	3783672	13672	134912	265280	1415080	88789	218010
64	4*	561210	5913320	14700	189440	510464	2446912	228186	532339

5.2. Performance Evaluation – Quality of Solutions

The quality of solutions – the makespan – is shown in Figure 3 and Figure 4. The solution optimization method based on proposed encodings is compared with WHCA*. We also made comparison with the SASE method. Due to space limitations the presentation is omitted. Nevertheless let us state, that SASE completely lost as it is unable to generate plan for more than 16 agents in the given time limit of 3600s.

WHCA* was able to solve instances with up to 16 agents in the case of 8×8 grid and up to 40 agents in the case of 16×16 grid. In both cases our method was able to find an optimal solution which is typically better than that found by WHCA*.

Again HBAD encoding seems to be the worst. However, the most interesting behavior appears in the case of 16×16 grid where we can observe point where the HPWAD encoding is better than the INVERSE encoding. For up to 80 agents in the environment which corresponds to the occupancy of 31.25% the HPWAD encoding dominates. Beyond the occupancy of 31.25% the INVERSE encoding starts to dominate.

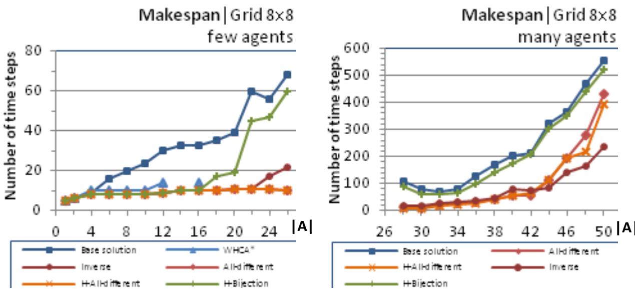

Figure 3. Makespan comparison on the 8×8 grid. Only up to 16 agents can be solved sub-optimally by WHCA*. HPWAD encoding provides best solutions for the number of agents ranging around 30. HBAD encoding has clearly no advantage.

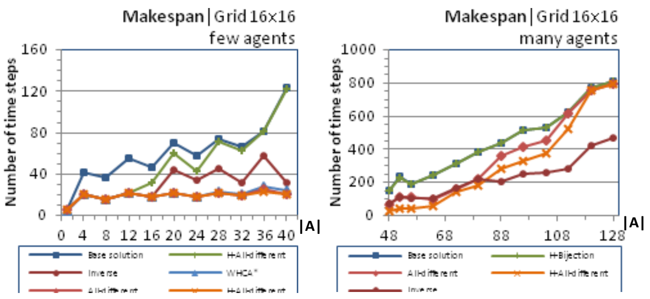

Figure 4. Makespan comparison on the 16×16 grid. WHCA* can solve up to 40 agents. HPWAD encoding shows its advantage here – in the range of 40-80 agents, it provides best solutions.

5.3. Performance Evaluation – Runtime

The runtime evaluation of the solution optimization method is shown in Figure 5. The runtime for WHCA* is omitted as it was always below one minute and runtime of SASE is also omitted as it was higher than that of presented method. Observably, the best runtime can be achieved with the HPWAD encoding. The HBAD encoding is the worst on the other hand. The sub-optimal method for producing the base sub-optimal is observably very fast. Hence the solution optimization process is an anytime method in fact as we can obtain a solution at every stage of the optimization process.

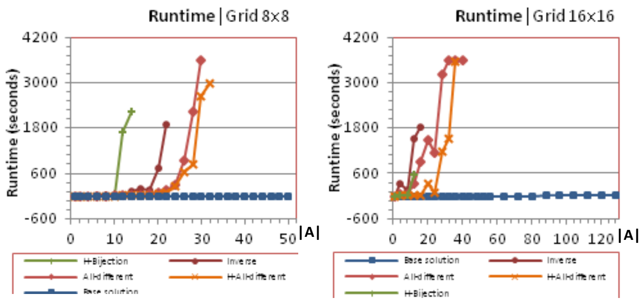

Figure 5. Runtime comparison. If the optimization finished in the time limit of 3600s then runtime is shown. In such a case optimal solution has been found. HPWAD encoding has slowest runtime degradatation while HBAD encoding degrades as fastest.

6. Conclusions and Future Works

We recalled an optimization method for improving solution of cooperative path-finding problem [15], [16]. The method employs formulation of the CPP instance as SAT. In this work we propose new encodings for the formulation of the task as SAT that can be alternatively used instead of those proposed in [15], [16]. One of the new encodings called HBAD turned out to be inferior in many cases however the other one called HPWAD outperformed encodings from [15], [16] in terms of speed and quality of generated solutions in the case of environment sparsely populated by agents. Thus, HPWAD become state-of-the-art encoding for the given case of the problem.

Regarding the HBAD encoding the hypothesis that CPP instances can take the advantage of the bijection-based encoding of the ALL-DIFFERENT constraint did not confirm.

For future work we plan to investigate the possibility of using different SAT solvers. Also we plan to formulate the CPP problem as CSP where global propagators for the ALL-DIFFERENT constraint based on network flows are available.

Acknowledgements

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

References

[1] Biere, A., Brummayer, R. "*Consistency Checking of All Different Constraints over Bit-Vectors within a SAT Solver*", Proceedings of FMCAD 2008, pp. 1-4, IEEE press, 2008.

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

[3] Eén, N., Sörensson, N. "*An Extensible SAT-solver*", Proceedings SAT 2003, pp. 502-518, LNCS 2919, Springer, 2004.

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

[5] Kautz, H., Selman, B. "*Unifying SAT-based and Graph-based Planning*", Proceedings of IJCAI 1999, pp. 318-325, Morgan Kaufmann, 1999.

[6] Korf, R. E., Taylor L. A. "*Finding Optimal Solutions to the 24-Puzzle*", Proceedings of AAAI 1996, pp. 1202-1207, AAAI Press, 1996.

[7] Luna, R., Berkis, K., E. "*Push-and-Swap: Fast Cooperative Path-Finding with Completeness Guarantees*", Proceedings of IJCAI 2011, pp. 294-300, IJCAI/AAAI Press, 2011.

[8] 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.

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

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

[11] Ryan, M. R. K. "*Exploiting Subgraph Structure in Multi-Robot Path Planning*", JAIR, Volume 31, pp. 497-542, AAA Press, 2008.

[12] Silver, D. "*Cooperative Pathfinding*", Proceedings of AIIDE 2005, pp. 117-122, AAAI Press, 2005.

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

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

[15] Surynek, P. "*Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving*", Submitted to PRICAI 2012.

[16] Surynek, P. "*A SAT-Based Approach to Cooperative Path-Finding Using All-Different Constraints*", Submitted to SoCS 2012.

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

[18] Wang, K. C., Botea, A., Kilby, P. "*Solution Quality Improvements for Massively Multi-Agent Pathfinding*". Proceedings of AAAI 2011, AAAI Press, 2011.