1E3-OS-4-10

Near Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving

Pavel Surynek [*1,*2]

[*1] Charles University Prague
Czech Republic

[*2] Kobe University
Japan

A new approach to cooperative path-planning is presented. The makespan of a solution to a path-planning instance is optimized by a SAT solver. A sub-optimal solution to the instance is obtained by some existing method first. Then it is submitted to the optimization process which decomposes it into small subsequences. Each sub-sequence of the original solution is subsequently replaced by the optimal sub-solution found by the SAT solver. The process is repeated until a fixed point is reached. This is the first method capable of producing near optimal solutions for densely populated instances.

1. Introduction and Motivation

Cooperative path-planning recently attracted considerable interest of the AI community. This interest is motivated by the broad range of areas where cooperative path-planning can be applied (robotics, computer entertainment, traffic optimization, etc.) as well as by challenging aspects which it offers. The task consists in finding spatial temporal paths for agents which want to reach certain destinations without colliding with each other. One of the most important break-through in solving the task is represented by the WHCA* algorithm (Silver, 2005) which decouples the search for cooperative plan into searches for plans for individual agents. Recently, an optimal decoupled method appeared (Standley & Korf, 2011). The common drawback of decoupled approach is that it is applicable only on instances with small occupancy of the environment by agents.

The opposite of the spectrum of solving algorithms is represented by complete sub-optimal methods (Surynek, 2009; Luna & Berkis, 2011). These algorithms are able to provide solution irrespectively of the portion of space occupied by agents. Especially good performance is reported for highly occupied instances. On the other side, long solutions are usually generated for sparsely populated environments.

Here we are trying to contribute to a not yet addressed case with high occupancy and the requirement on solution to have short makespan. We use the SAT solving technology in a novel and unique way to address this case. First a sub-optimal solution is generated by some of the existent fast algorithms. The sub-optimal solution is then decomposed into small sub-sequences that are replaced by optimal sub-solutions generated by a SAT solver. The process is iterated until the makespan converges. This decomposition of the original problem allowed us to exploit the strongest aspect of SAT solvers – their ability to satisfy relatively small yet complex enough SAT instance very quickly.

The rest of the paper describes cooperative path planning formally first. Then our special domain dependent SAT encoding and the optimization method are introduced. An experimental comparison with several existent techniques is presented finally.

Contact: Pavel Surynek, Kobe University,
5-1-1 Fukae-minami-machi, Higashinada-ku, Kobe 658-0022, Japan. pavel.surynek@mff.cuni.cz, Phone: 078-4316262.

2. Cooperative Path-Planning Formally

An **undirected graph** is used to model the environment. Let $G = (V, E)$ be such a graph. The placement of agents in the environment is modeled by assigning them vertices of the graph. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set of *agents*. An arrangement of agents in vertices of graph G is 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* (CPP) 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. □

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).

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 $i = 1,2, \ldots, m-1$. □

If it is a question whether there is a solution of Σ of the makespan at most a given bound we are speaking about the *bounded variant*. Notice that due to no-ops introduced in valid transitions it is equivalent to finding a solution of the makespan equal to the given bound.

3. SAT Encoding of Bounded CPP

Our goal was to devise a SAT encoding of bounded CPP suitable for relatively densely populated environments. At the same time we needed to keep the encoding compact. We followed the classical *Graphplan* inspired encodings as for we also encode each time step. Our design is similar to that of SATPLAN (Kautz & Selman, 1999) and SASE (Huang et al., 2010) encodings. But unlike these generic encodings we were working with the specific domain so we could facilitate the domain knowledge in the design of the instance encoding.

We a priori know what the candidates for *multi-valued state variables* are in our domain – basically, these are represented by location function and its inverse. Using techniques proposed by Rintanen (2006) each state variable can be encoded by a logarithmic number of propositional variables with respect to the number its values. Another considerable aspect is how to encode transition actions together with validity conditions.

Representing arrangement of agents by **inverse locations** (that is, there is a state variable for each vertex) allowed us to encode transitions efficiently. 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 a vertex are 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. Representing the selection of the primitive action as a multi-values state variable automatically ensures that conditions (i) and (iii) are encoded. Notice that the degree of vertices in G is typically low for real-life environments, thus action selection in the vertex can be captured by few propositional variables.

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

Definition 3 (LAYER ENCODING). The i-th regular layer consists of the following integer interval **state variables**:

- $\mathcal{A}_i^v \in \{0,1,2,\dots,n\}$ for all $v \in V$ such that $\mathcal{A}_i^v = j$ iff $\alpha_i(a_j) = v$
- $\mathcal{T}_i^v \in \{0,1,2,\dots,2\deg_G(v)\}$ 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). □

State variables $\mathcal{A}_i^v$ for $v \in V$ represent inverse location function at the time step i. Analogically, state variables $\mathcal{T}_i^v$ for $v \in V$ represent transition actions selected in vertices at time step i. Constraints merely encode the validity conditions.

The last encoding layer is irregular as it has location state variables only. To finish the encoding we need to encode the initial and the goal arrangement straightforwardly as follows:

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

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

Transformation of the encoding from the above integer representation to the propositional one is also straightforward. To reduce size of clauses we should use standard Tseitin's hierarchical encoding with auxiliary variables.

Table 1. *Comparison of encoding sizes.* The smallest number of layers for which SATPLAN was unable to detect unreachability of the goal using mutex reasoning is indicated as *goal level* – it is used as the makespan bound.

|**Agents**| in 4-connected grid 8x8	**Goal level**	SATPLAN encoding		Our **domain specific** encoding	
		|**Variables**|	|**Clauses**|	|**Variables**|	|**Clauses**|
4	8	5864	55330	9432	55008
8	8	10022	165660	11968	70400
12	8	14471	356410	11968	68352
16	10	30157	1169198	18490	112580
24	10	43451	2473813	18490	107360
32	14	99398	8530312	32116	200768

4. COBOPT: Optimization Process

Our novel CPP technique called COBOPT exploits SAT solving technology (Eén & Sörensson, 2004) not to produce a solution but to **optimize** it with respect to the makespan. To be able to use SAT solvers in this way we need to obtain some (sub-optimal) solution to the CPP instance first. Let this initial solution be called *base solution*. As we mentioned, many solving techniques for CPP are available at the present time (Silver, 2005 – WHCA*; Ryan, 2008; Surynek, 2009 – BIBOX; Luna & Berkis, 2011 – PUSH-SWAP ; Standley & Korf, 2011 - OD+ID). Any of them can be used to produce base solution within our framework. Our approach is completely generic in this sense. Notice however, that particular solving technique is always designed for a specific class of the problem while outside this class it may provide worse performance. The typical weakness is for example that *decoupled* techniques (WHCA* - Silver, 2005) admit that not all the agents need to reach their destination.

In our initial experiments, we found that it is becoming dramatically more difficult for SAT solvers to solve bounded CPP instance as the bound is growing. To be more concrete, a SAT solver usually struggles with the instance consisting of the graph containing 100 vertices, 30 agents, and the bound of 10 for several minutes if the presented SAT encoding is used. In case of the SATPLAN encoding the situation is even worse – the solver even struggles with generating the formula for minutes. This finding renders possibility of using SAT solvers to solve a cooperative instance of considerable size in the SATPLAN style (Kautz & Selman, 1999; Huang et al., 2010) as infeasible at the current state-of-the-art since it may require hundreds of time steps. But using a SAT solver in the SATPLAN style has one undisputable advantage if we manage to get a solution from it – it is **makespan optimal**.

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

The binary search is exploited to find t^+ and the optimal sub-solution in order to reduce the number of SAT solver invocations – see Algorithm 1 which summarizes basic COBOPT optimization method formally.

Algorithm 1. COBOPT: *SAT-based CPP solution optimization – basic scheme based on binary search.*

function *COBOPT-Optimize-Cooperative-Plan* $(\Sigma, \vec{s}, k^+)$: **solution**
1: $\vec{s}_+ \leftarrow \vec{s}$
2: **do**
3: $\quad \vec{s}_- \leftarrow \vec{s}_+$
4: $\quad$ **let** $\vec{s}_- = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m]$
5: $\quad t \leftarrow 0; \vec{s}_+ \leftarrow []$
6: $\quad$ **while** $t < m$ **do**
7: $\quad\quad t^+ \leftarrow$ *Find-Last-Reachable-Arrangement*$(\Sigma, \alpha_t, \vec{s}_-, k^+)$
8: $\quad\quad \vec{s}_+ \leftarrow \vec{s}_+$.*Compute-Optimal-Solution*$(\Sigma, \alpha_t, \alpha_{t^+})$
9: $\quad\quad t \leftarrow t^+$
10: **while** $|\vec{s}_-| > |\vec{s}_+|$
11: **return** $\vec{s}_+$

function *Find-Last-Reachable-Arrangement* $(\Sigma, \alpha_t, \vec{s}, k^+)$: **integer**
1: **let** $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m]$
2: $l \leftarrow t; u \leftarrow m + 1$
3: **while** $u - l > 1$ **do**
4: $\quad r \leftarrow (u + l)/2$
5: $\quad k \leftarrow \min(m - t, k^+)$
6: $\quad$ **if** *Check-Reachability*$(\Sigma, \alpha_t, \alpha_r, k)$ **then**
7: $\quad\quad \Xi \leftarrow$*Encode*$(\Sigma, \alpha_t, \alpha_r, k)$
8: $\quad\quad$ **if** *Solve-SAT* (Ξ) **then** $l \leftarrow r$
9: $\quad\quad$ **else** $u \leftarrow r$
10: $\quad$ **else**
11: $\quad\quad u \leftarrow r$
12: **return** l

function *Check-Reachability* $(\Sigma, \alpha_t, \alpha_r, k)$: **boolean**
1: **let** $\Sigma = [G, A, \alpha_0, \alpha_+]$
2: **for each** $a \in A$ **do**
3: $\quad$ **if** $\mathrm{dist}_G(\alpha_t(a), \alpha_r(a)) > k$ **then return** *FALSE*
4: **return** *TRUE*

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

5. Experimental Evaluation

We implemented the proposed COBOPT optimization method in C++ to conduct an experimental evaluation. A competitive comparison against 3 existent methods was made – WHCA*, SATPLAN, and BIBOX. WHCA* was chosen as a reference method as it is considered to be standard decoupled method for CPP and its properties and performance are well known.

Table 2. *Optimal solutions obtained by* SATPLAN. No more agents can be solved by SATPLAN within the time limit of 7200s.

|Agents|	4-connected grid 8x8		4-connected grid 16x16	
	Optimal makespan	Runtime (s)	Optimal makespan	Runtime (s)
1	5	0.0	4	0.68
4	6	0.15	21	195.5
8	8	19.85	15	1396.07

As no implementation of WHCA* was available we re-implemented it in C++ by ourselves. SATPLAN is the most similar method to our approach and very importantly it produces optimal solutions – we used implementation provided by the authors. Finally, BIBOX was selected as major method for producing base solutions in hard setups. Our choice was not discouraged by the wrong statement of Standley and Korf (2011) who consider it to have memory and time requirements that limit its applicability.

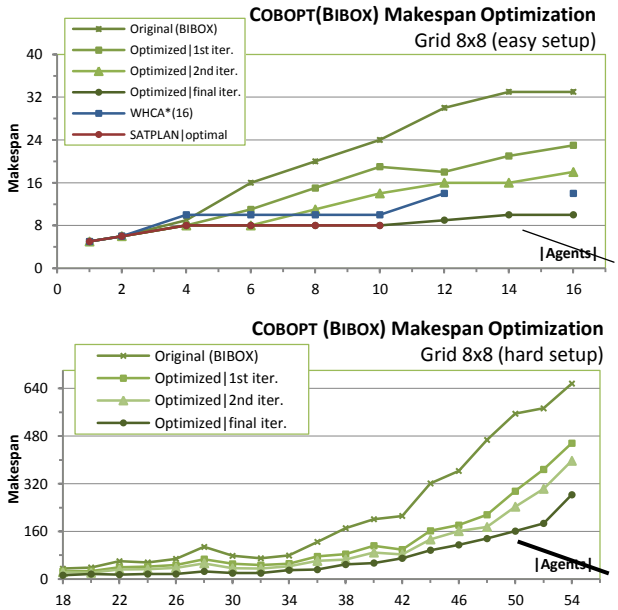

Figure 1. *Makespan optimization in the 4-connected grid 8×8.* A comparison with the optimal SATPLAN and near optimal WHCA* is shown.

The BIBOX algorithm has polynomial time complexity (solutions to all the benchmarks presented here were generated within less than 0.1 seconds) and generates good quality sub-optimal solutions irrespectively how many agents are contained in the instance – together with the algorithm PUSH-SWAP by Luna & Berkis (2011) it is the only algorithm able to generate base solution for hard setups. Authors provide working implementation of BIBOX which we exploited within our experiments. COBOPT using BIBOX as a base solver will be referred to as COBOPT(BIBOX). As a SAT solver within our method, MINISAT 2.2 (Eén & Sörensson, 2004) was used.

Standard benchmark setups for CPP which consists of a 4-connected grid graph and randomly arranged initial and goal locations for agents were used. Various parameters of the COBOPT(BIBOX) and other methods were observed in the dependence on the increasing number of agents in the instance.

A setup with the grid of size 8×8 ant the number of agents ranging from 1 to 54 was used. The timeout of 240s per SAT solver invocation and the makespan bounds of 8 were used. Additionally there was an overall timeout of 7200s (2 hours) after which the optimization process was terminated. The number of iterations until the fixed point was reached ranged from 1 to 20 with median of 7. Due to space limitations we present only a fraction of results here.

Using WHCA* we observed that setups with up to approximately 20% of occupied vertices are in fact easy as only very limited cooperation among agents is necessary. Notice, that the method OD+ID which also tries to generate good makespans is reported to be efficient only in the setups with less than 10% of occupied vertices. Here we are interested primarily in setups with occupancy in the range 20% - 50% which is increasingly harder as cooperation between agents gradually increases.

To learn what the optimal makespan for tested instances is we tried SATPLAN (Table 2). Unfortunately SATPLAN was able to generate solution only to instances with small number of agents. The reason is primarily inefficiency of domain-independent SAT encoding (Table 1).

Regarding the decoupled WHCA* method, we found that in sparse instances it is able to generate near optimal solutions (Figure 1) since near optimal path is tried to be found for each agent separately. However, this method is principally unable to solve instances where non-trivial cooperation among agents is necessary. WHCA* was used to classify instances on **easy** and **hard** – the easy ones are those solvable by WHCA*. Even on easy instances WHCA* was significantly outperformed by COBOPT(BIBOOX) according to our experiments.

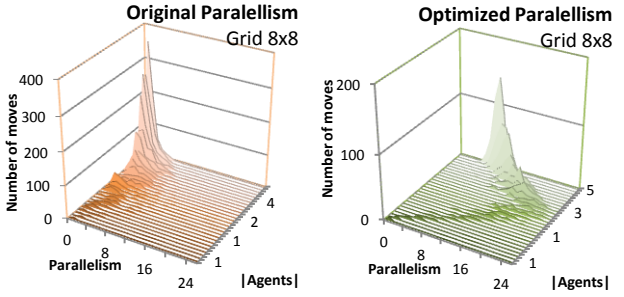

Figure 2. *Distribution of parallelism before and after* COBOPT *optimization.* Agents are using all the available freedom in the optimized variant – almost all the vacant vertices are used for movements while in un-optimized one there is lot of wait actions.

The COBOPT method is very friendly to multithreaded implementation. Hence the scalability of our is extremely good (provided that computational resources are available). Moreover, if the method for producing base solutions is fast enough then COBOPT is anytime in fact – at any time step the solving process can be terminated and feasible (sub-optimal) solution is returned.

To get insight what happen when a solver is used for optimization we investigated distribution of the number of actions executed in parallel – Figure 2. Base solutions seem to suffer from locked agents which are forced to wait until their path is freed. In optimized solutions, as many as possible agents are actively moving towards goals – it is possible to observe that agents utilize almost all the available unoccupied space.

6. Conclusion

The new SAT based solving method for CPP called COBOPT has been presented. To be able to use a SAT solver for cooperative path-planning we also developed a new SAT encoding for CPP instances. The encoding utilizes structural properties of CPP to reduce its size and increase efficiency.

The COBOPT method was shown that it is able to generate near optimal or good quality solutions in setups with high occupancy of the environment by agents. It is the first method capable of doing so. In our experiments we solved 4-connected grid instances of size 8×8 with up to 84% space occupied by agents with high quality makespans. One of the positive aspects of the new approach is also the fact that it can be easily parallelized for multi-core architectures which supports better scalability.

The COBOPT method has also quite strong implications for classical planning. Provided that efficient makespan sub-optimal planner is available, COBOPT can be immediately used to optimize its output (SASE and SATPLAN encodings are ready). Another possible future improvement is to reduce the size of the domain dependent encoding for sparsely populated instances.

References

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

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

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

[Luna, R., Berkis, K., E., 2011] 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.

[Ratner, D., Warmuth, M. K.,1986] Finding a Shortest Solu-tion for the N × N Extension of the 15-PUZZLE Is Intrac-table. Proceedings of AAAI 1986, pp. 168-172, Morgan Kaufmann.

[Rintanen, J., 2006] Compact Representation of Sets of Binary Constraints. Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), pp. 143-147, IOS Press 2006.

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

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

[Surynek, P., 2009] A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs. Proceedings of the International Conference on Robotics and Automation (ICRA 2009), pp. 3613-3619, IEEE Press.