Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving

Pavel Surynek [1,2]

[1] Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic

[2] Kobe University, Graduate School of Maritime Sciences, Intelligent Informatics Laboratory
5-1-1 Fukae-minamimachi, Higashinada-ku, Kobe 658-0022, Japan

pavel.surynek@mff.cuni.cz

Abstract. A novel approach to cooperative path-planning is presented. A SAT solver is used not to solve the whole instance but for optimizing the makespan of a sub-optimal solution. This approach is trying to exploit the ability of state-of-the-art SAT solvers to give a solution to relatively small instance quickly. A sub-optimal solution to the instance is obtained by some existent method first. It is then submitted to the optimization process which decomposes it into small subsequences for which optimal solutions are found by a SAT solver. The new shorter solution is subsequently obtained as concatenation of optimal sub-solutions. The process is iterated until a fixed point is reached. This is the first method to produce near optimal solutions for densely populated environments; it can be also applied to domain-independent planning supposed that sub-optimal planner is available.

1 Introduction and Context

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 breakthrough in solving the task is represented by the WHCA* algorithm [8] which decouples the search for cooperative plan into searches for plans for individual agents. Recently, an optimal decoupled method appeared [10]. 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 [4, 11]. 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, too long solutions are usually generated for sparsely populated environments. Other methods are trying to exploit the structure of environment [7] or the structure of current arrangement of agents [12].

This work is supported by the by the Japan Society for the Promotion of Science (contract no. P11743) and by the Czech Science Foundation (contract no. GAP103/10/1287).

Again these methods require relatively unoccupied environment and in some cases they are not complete.

Here we are trying to contribute to a not yet addressed case with high occupancy and the requirement on solution to have short makespan. Our approach is basically complete. We use 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 – that is, 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 methods are introduced. An experimental comparison with several existent techniques is presented finally.

2 Cooperative Path Planning Formally

Arbitrary **undirected graph** can be used to model the environment. Let $G = (V, E)$ be such a graph where V 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 $A = \{a_1, a_2, \ldots, a_n\}$ 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 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 the Bounded Variant

Our goal was to devise a SAT encoding of bounded cooperative path planning suitable for relatively densely populated environments. At the same time we needed to keep the encoding compact. We followed the classical *Graphplan* inspired encodings [2, 3] as for we also encode each time step.

Using Multi-valued State Variables

We were primarily inspired by SATPLAN [3] and SASE [2] encodings in our design. 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 [6] each state variable can be encoded by 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. Moreover, we do not need any *mutex* constraints in the encoding. Notice also, 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 a cooperative instance and $k \in \mathbb{N}$ be a makespan bound. Our encoding has layers numbered $0,1, \ldots, k$. Suppose that neighboring vertices of a given vertex are ordered in the fixed order. That is, $\forall v \in V$ we have function σ_v: $\{u | \{v, u\} \in E\} \longrightarrow \{1,2, \ldots, \deg_G(v)\}$ and its inverse σ_v^{-1}.

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

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

$\mathcal{T}_i^v = 0$ iff no-op was selected in v;
$\mathcal{T}_i^v = \sigma_v(u)$ iff an outgoing primitive action with the target $u \in V$ was selected in v;
$\mathcal{T}_i^v = \deg_G(v) + \sigma_v(u)$ iff an incoming primitive 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);
- $0 < \mathcal{T}_i^v \leq \mathrm{dg}_G(v) \;\Rightarrow\; \mathcal{A}_i^u = 0 \wedge \mathcal{A}_{i+1}^u = \mathcal{A}_i^v \wedge \mathcal{T}_i^u = \sigma_u(v) + \mathrm{dg}_G(u)$ where $u = o_v^{-1}(\mathcal{T}_i^v)$ for all $v \in V$ (**outgoing** agent case);
- $\deg_G(v) < T_i^v \leq 2\mathrm{dg}_G(v) \;\Rightarrow \mathcal{T}_i^u = \sigma_u(v)$ where $u = \sigma_v^{-1}(\mathcal{T}_i^v - \mathrm{dg}_G(v))$ 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 of bounded cooperative instance we need to encode the initial and the goal arrangement straightforwardly as follows:

$\mathcal{A}_0^v = j$ iff $\alpha_0^{-1}(v) = a_j$,
$\mathcal{A}_0^v = 0$ iff $\alpha_0^{-1}(v) = \perp$,
$\mathcal{A}_k^v = j$ iff $\alpha_+^{-1}(v) = a_j$,
$\mathcal{A}_k^v = 0$ iff $\alpha_+^{-1}(v) = \perp$.

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.

Proposition 1 (ENCODING SIZE). *A regular layer of the propositional encoding of the bounded cooperative instance requires*

$|V|\lceil\log_2|A|\rceil + \sum_{v\in V}\lceil\log_2(2\,\mathrm{dg}_G(v)+1)\rceil$ (1)

propositional ***variables*** *for representing state variables,*

$2|E|\lceil\log_2|A|\rceil + |V|\lceil\log_2|A|\rceil$ (2)

auxiliary propositional ***variables*** *from Tseitin's translation*

$\sum_{v\in V}\mathrm{dg}_G(v)\,(2\lceil\log_2(2\,\mathrm{dg}_G(v)+1)\rceil + 7\lceil\log_2|A|\rceil) + |V|\lceil\log_2|A|\rceil$ (3)

clauses *for representing constraints, and*

$|V|(2^{\lceil\log_2|A|\rceil} - |A|)$, (4)

$\sum_{v\in V} 2^{\lceil\log_2(2\,\mathrm{dg}_G(v)+1)\rceil} - 2\sum_{v\in V}\mathrm{dg}_G(v) - |V|$ (5)

clauses *for excluding unused location and transition action states respectively.* ■

Proof. We just need to observe that $2\,\mathrm{dg}_G(v)+1$ cases (preconditions) need to be distinguished for each vertex $v \in V$ and for each of these cases a corresponding effect needs to be enforced. The cases with **outgoing** agent need each $\lceil\log_2|A|\rceil$ auxiliary propositional variables which come from Tseitin's encoding and $\lceil\log_2(2\,\mathrm{dg}_G(v)+1)\rceil + 2\lceil\log_2|A|\rceil + 5\lceil\log_2|A|\rceil$ clauses (1 equality between transition action and a constant + 2 equalities between inverse location and a constant, and 1 equality between two inverse locations). The cases with **incoming** agent do not require any auxiliary variable and only $\lceil\log_2(2\,\mathrm{dg}_G(v)+1)\rceil$ clauses are needed (1 equality between transition action and a constant). Finally, the single **no-op** case requires $\lceil\log_2|A|\rceil$

auxiliary variables and $5\lceil \log_2 |A| \rceil$ clauses (1 equality between two inverse locations). From this overview expressions (1)-(5) are straightforward. ■

Most of clauses generated in our encoding have arity of $\lceil \log_2 (2 \deg_G(v) + 1) \rceil + 1$ for some $v \in V$ or $\lceil \log_2 |A| \rceil + 1$. The comparison with the graph-plan based encoding used in SATPLAN is shown in Table 1. Our domain-specific encoding is clearly smaller while the difference is growing as the number of agents increases.

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: A New Approach to (Path) Planning

Our novel cooperative path planning technique called COBOPT exploits SAT solving technology [1] 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 cooperative instance first. Let this initial solution be called base solution. As we mentioned, many solving techniques for cooperative path planning are available at the present time [7], [8] (WHCA*); [11] (BIBOX); [4] (PUSH-SWAP); [10] - OD+ID; [12] (MAPP). 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*, MAPP) admit that not all the agents need to reach their destination [8, 12].

In our initial experiments, we found that it is becoming dramatically more difficult for SAT solvers to solve bounded cooperative 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 [2, 3] 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 a base solution, this is submitted to a SAT based optimization process. A maximum bound k^+ for encoding cooperative instances is specified. Then sub-sequences in the base solution are replaced with computed optimal sub-solution. Suppose that we are currently optimizing at time step t. 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 time step t^+ until the whole base solution is processed.

Algorithm 1. COBOPT: *SAT-based cooperative path planning 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*

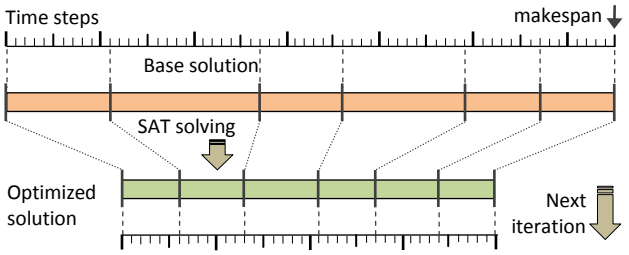


Figure 1. *Illustration of the optimization process.* A single iteration is shown – these are repeated until a fixed point is reached.

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. Notice that some extra care is needed to obtain optimal sub-solution at the end of the base solution sequence.

The process of optimization is illustrated in Figure 1. 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 optimizing placement of separation point by dynamic programming techniques. This approach generates slightly better base solution decomposition. However it is at the great expense in overall runtime as many more invocation of the SAT solver are necessary.

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 reference method as it is considered to be standard decoupled method for cooperative path planning 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 [10] who consider it together with the method of Ryan [7] to have memory and time requirements that limit their applicability. According to our findings, these algorithms have important theoretical guarantees and good practical performance. Particularly, BIBOX 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 and Berkis [4] 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 [1] was used.

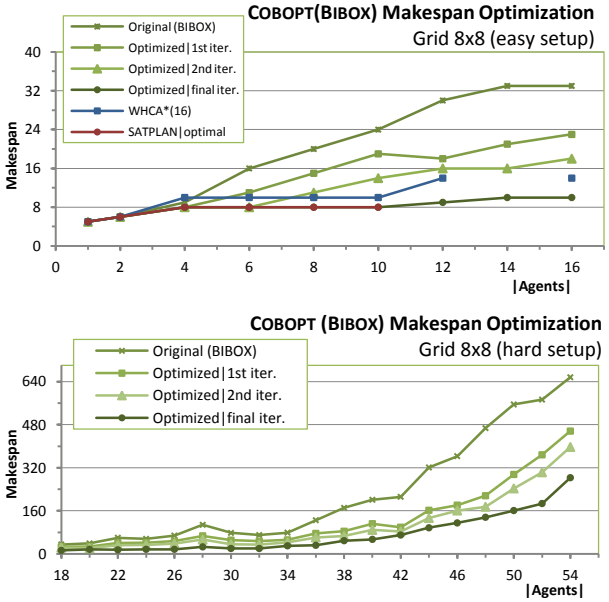


Figure 2. *Makespan optimization in the 4-connected grid 8×8.* A comparison with the optimal SATPLAN and near optimal WHCA* is shown.

Standard benchmark setups for cooperative path planning 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 of the increasing number of agents in the instance. Two setups were used: grids of size 8×8 and 16×16 with number of agents ranging from 1 to 54 and 1 to 128 respectively. The timeout of 240s and 120s per SAT solver invocation was used for these setups respectively. Makespan bounds of 8 and 6 were used respectively. Additionally there was an overall timeout of 7200s (2 hours) after which the optimization process was terminated.

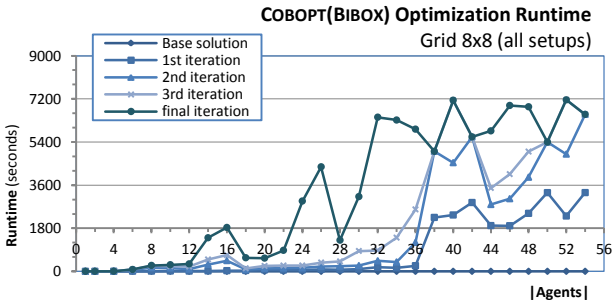


Figure 3. *Runtime measurements per optimization iteration in 8×8 grid.* The base solution can be produced in less than 0.1s.

Due to the bigger size of SAT encodings for the 16×16 grid the optimization method uses less aggressive setup with respect to the SAT solver. 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. This observation ruled out from our consideration the method OD+ID as it 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.

Table 3. MINISAT *statistics (8×8 grid).* Each invocation of MINISAT within COBOPT optimization has the timeout of 240s.

|**Agents**| in 4-connected grid 8x8	Number of MINISAT results in final iteration		
	SAT instances	**UNSAT** instances	**INDET** instances
4	13	2	0
8	44	4	0
12	79	5	0
16	96	15	0
24	253	28	0
32	194	27	2

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).

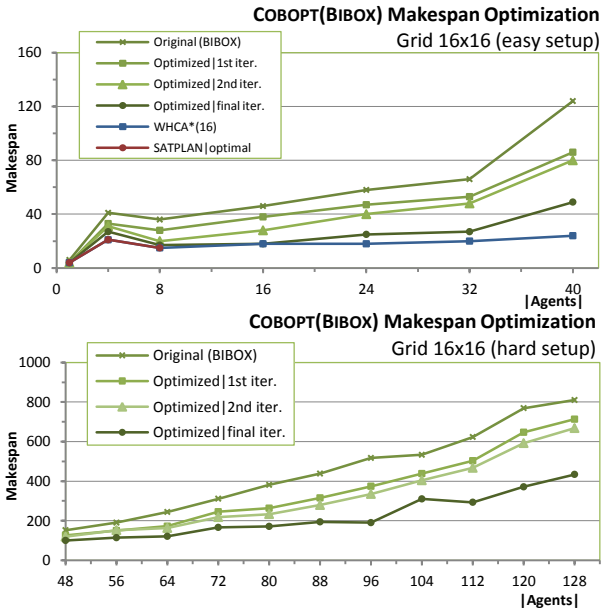
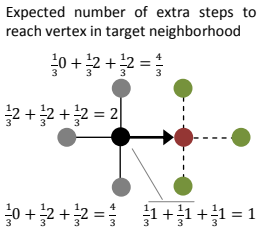


Figure 4. *Makespan optimization in the 4-connected grid 16×16.*

In the following experiments we exploited the decoupled WHCA* method. Expectably it is able to generate near optimal solutions (Figure 2, Figure 4) 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*.

Contrary to SATPLAN and WHCA* COBOPT is more successful; it is able to provide solution to every instance to which base solving method can do so – in case of the BIBOX algorithm these are all the instances in our test suite.



In case of the 8×8 grid COBOPT(BIBOX) generates very near optimal solutions for easy setups (same as SATPLAN; same as or better than WHCA*) - Figure 2. Nevertheless, the most interesting behavior is exhibited in the hard region where compression up to the ratio of $\frac{1}{3}$ with respect to the makespan of base solution can be achieved. Although it is not known if optimum was actually reached, this is a big qualitative leap from the base solution and it demonstrates efficiency of the COBOPT optimization process.

Supposed that certain simplification is accepted then we can calculate expected lower bound for the optimal makespan in the 4-connected grid environment. Let us suppose that if an agent is blocked on its path by another agent, it will either wait or go into unblocked neighborhood all with the same probability of $\frac{1}{4}$. This behavior deflects the agent from its original path and some extra steps are then necessary to continue in the right direction. It is supposed that original path continues to vertices in the neighborhood of blocked vertex with the same probability of $\frac{1}{3}$. Under these assumptions we obtain that the expected number of extra steps is $\frac{1}{4}(2\frac{4}{3}+2+1)=\frac{19}{12}$ per two original steps. Simply it means that two original steps require almost two extra steps. We will adopt quite strong assumption and round it up to exactly two extra steps which consequently implies that the agent actually does not reduce its distance from the destination.

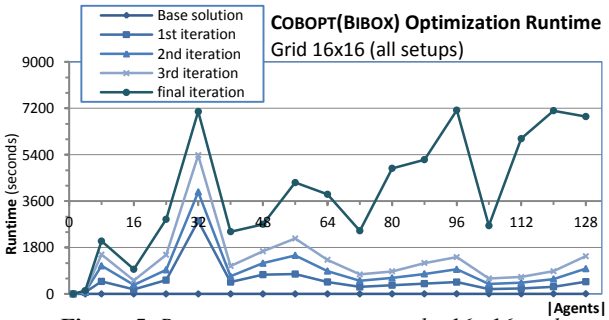


Figure 5. *Runtime measurements in the 16×16 grid.*

Proposition 2 (EXPECTED MAKESPAN). *The expected make-span required to travel distance* $d \in \mathbb{N}$ *in a 4-connected grid with occupancy ratio* c *under our assumptions is:*

$$\mu(d) = \frac{c(19-7d)-12d}{12(c-1)}. \blacksquare$$

Proof. The following recurrence holds under assumptions stated above: $\mu(d) \geq c(\mu(d) + \frac{19}{12}) + (1 - c)(\mu(d-1) + 1)$ where we can put $\mu(1) = 1$. From this we quickly obtain the required explicit form. ■

According to above calculations COBOPT(BIBOX) generates near optimal solutions that differs from the expected optimum by less than 25% in setups with up to occupancy of 60% in the grid 8×8. Expectably in the grid 16×16 the situation is not so optimistic, solutions differ here from the expected optimum by factor of 3.0 to 6.0 in hard setups with occupancy up to 50%. This worse performance is mainly because of the size of the grid which prevented us from using more aggressive optimization.

The number of iterations until the fixed point was reached ranged from 1 to 20 with median of 7 in case of 8×8 grid and from 2 to 31 with median of 11 for the grid 16×16. The number of SAT solver invocations is reported in Table 3. It is clear that in our approach the SAT solver is invoked many times with relatively easy instances.

Runtime[1] is reported in Figure 3 and Figure 5. Despite hundreds of SAT solver invocations the overall runtime is kept in acceptable bounds. Fortunately, the COBOPT method is very friendly to multithreaded implementation. Hence the scalability of the method is extremely good (provided that computational resources are available). Moreover, if the method for producing base solutions is fast enough then COBOPT represents the anytime method 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 6. 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.

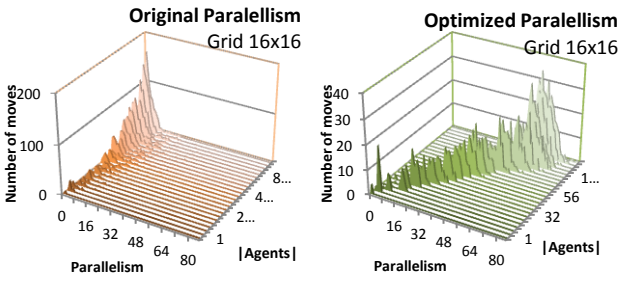


Figure 6. *Distribution of parallelism in the grid 16×16.* Almost all the free space is used for moving in the optimized solution.

6 Discussion, Conclusions, and Future Works

The new **SAT based** solving method for cooperative path planning called COBOPT has been presented. To be able to use a SAT solver for cooperative path-planning we

[1] All the runtime measurements were done on a machine with the 6 core CPU Intel Xeon 2.0GHz and 12GiB RAM under Linux kernel 2.6.24-19. All the 6 cores of the CPU were exploited in parallel.

also developed a new SAT **encoding** for cooperative instances. The encoding utilizes properties of cooperative planning in order 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 up to 16×16 with up to 50% 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). A possible future improvement is to reduce the size of the domain dependent encoding for sparsely populated instances. The application of binary search for solvable instance may be also revised as other types of search may be more efficient.

References

1. 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.
2. Huang, R., Chen, Y., Zhang, W. *A Novel Transition Based Encoding Scheme for Planning as Satisfiability.* Proceedings AAAI 2010, AAAI Press, 2010.
3. 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.
4. 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.
5. 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.
6. Rintanen, J. *Compact Representation of Sets of Binary Constraints.* Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), pp. 143-147, IOS Press, 2006.
7. 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.
8. Silver, D. *Cooperative Pathfinding.* Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117-122, AAAI Press, 2005.
9. Standley, T. S. *Finding Optimal Solutions to Cooperative Pathfinding Problems.* Proceedings of the 24th Conference on Artificial Intelligence (AAAI 2010), AAAI Press, 2010.
10. Standley, T. S., Korf, R. E. *Complete Algorithms for Cooperative Pathfinding Problems.* Proceedings of IJCAI 2011, 668-673, IJCAI/AAAI Press, 2011.
11. Surynek, P. *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, 2009.
12. 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.