Ann Math Artif Intell (2017) 81:329–375
DOI 10.1007/s10472-017-9560-z

Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems

Pavel Surynek[1,2]

Published online: 2 August 2017

Abstract This paper deals with solving cooperative path finding (CPF) problems in a makespan-optimal way. A feasible solution to the CPF problem lies in the moving of mobile agents where each agent has unique initial and goal positions. The abstraction adopted in CPF assumes that agents are discrete units that move over an undirected graph by traversing its edges. We focus specifically on makespan-optimal solutions to the CPF problem where the task is to generate solutions that are as short as possible in terms of the total number of time steps required for all agents to reach their goal positions. We demonstrate that reducing CPF to propositional satisfiability (SAT) represents a viable way to obtain makespan-optimal solutions. Several ways of encoding CPFs into propositional formulae are proposed and evaluated both theoretically and experimentally. Encodings based on the log and direct representations of decision variables are compared. The evaluation indicates that SAT-based solutions to CPF outperform the makespan-optimal versions of such search-based CPF solvers such as OD+ID, CBS, and ICTS in highly constrained scenarios (i.e., environments that are densely occupied by agents and where interactions among the agents are frequent). Moreover, the experiments clearly show that CPF encodings based on the direct representation of variables can be solved faster, although they are less space-efficient than log encodings.

Submitted to *Annals on Mathematics and Artificial Intelligence* on April 15, 2016. Reviews received on June 26, 2016. Major revision submitted on October 28, 2016. Reviews received on March 21, 2017. Minor revision submitted on May 20, 2017.

✉ Pavel Surynek
pavel.surynek@mff.cuni.cz

[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC, 2-3-26, Aomi, Koto-ku, Tokyo 135-0064, Japan

[2] Department of Theoretical Computer Science and Mathematical Logic, Charles University, Malostranské náměstí 25, Praha 1, 118 00, Czechia

Keywords Cooperative path-finding (CPF) · Propositional satisfiability (SAT) · Time-expanded graphs · Makespan-optimality · Multi-robot path planning · Multi-agent pathfinding · Pebble motion on graphs · Operator decomposition/independence detection – OD+ID · Conflict-based search – CBS · Increasing cost tree search – ICTS

Mathematics Subject Classification (2010) 68T20 · 68T27 · 68T30

1 Introduction

Cooperative path-finding - CPF [31, 51, 53] (also known as *multi-agent path-finding* - MAPF [49, 50, 68, 69] or *multi-robot path planning* - MRPP [42, 43] or *pebble motion on graphs* - PMG [31, 39]) is an abstraction of many real-life tasks, in which the goal is to move certain objects that spatially interact with each other. In the context of CPF, we speak about mobile *agents* (or robots) that can be moved in a certain environment. Each agent starts at a given initial position in the environment and is assigned a unique goal position which it has to reach.

A feasible solution to the CPF problem is a spatial-temporal path for each agent. The agents move from their given initial positions to their goal positions by following their paths without colliding with each other.

The applications of CPF range from unit navigation in computer games [52] to item repositioning in automated storage systems (see KIVA robots [30]). Further interesting applications can be found in traffic control, for example in collision avoidance at sea [29]. Similar challenges extend to aviation, as the presence of drones necessitates cooperation between them [33].

A graph-theoretical abstraction where the environment in which the agents move is modeled as an undirected graph is often adopted [42, 47]. In this abstraction, agents are represented as discrete units placed in the vertices of the graph. Several constraints are assumed in the abstract model, so that it reflects the physical properties of the agents and the environment. We classify these constraints into **static** and **dynamic**:

- **Static constraints** represent spatial occupancies caused by the presence of agents. These constraints are usually modeled based on the requirement that no more than one agent can be present in each vertex.
- **Dynamic constraints** impose restrictions on the ways the agents can move and define what is considered as a collision. We adopt a model commonly used in literature [43, 49], in which an agent can instantaneously move to the neighboring vertex, if the target vertex is unoccupied and no other agent is trying to enter the same target vertex at the same time.

There are other types of dynamic constraints in CPF including chain-like relocations [56], during which only the leading agent of a chain of agents moves into an unoccupied vertex and all other agents in the chain follow it, entering the vertices that are simultaneously vacated by the agents that precede them. Some models also allow the rotations of agents over cycles with at least 3 vertices with no free vertex inside the cycle [71] (swapping the agents over edges is usually forbidden, as such a transition corresponds to a head-on collision).

It is important to note that methods presented in this paper are fully applicable across various versions of static and dynamic constraints.

1.1 Introducing the objective function

In this paper, we are interested in solutions with *makespans* as short as possible [51, 57]. Makespan refers to the number of time steps required for all agents to reach their goal positions. In other words, makespan is the number of time steps elapsed until the last agent reaches its goal vertex.

By introducing the makespan-objective function to CPF we have also created a significant challenge to solving this problem. From the theoretical point of view, finding makespan-optimal solutions to CPF is known to be a complex problem. Namely, the problem is NPhard (the decision version is NPcomplete) [39, 60, 71].

Note that the so-called *sum-of-costs* [49, 50] is also studied as an objective function in the context of CPF. The sum-of-costs corresponds to the total number of moves carried out by the agents. In practice, the sum-of-costs can be understood as the amount of energy consumed by the agents. Hence, sum-of-costs-optimal solutions are energy-saving, whereas makespan-optimal solutions are time-saving.

These two objective functions are correlated but distinct – there are CPF instances where none of the makespan-optimal solutions is sum-of-costs-optimal and vice versa. This naturally reflects the real-life situations where we can save time at the cost of expending more energy.

In this paper, we also focus specifically on CPF instances with *high agent densities* and frequent spatial interactions between the agents. These high-density scenarios are widely applicable in connection with makespan-optimality in practical domains such as navigating cars automatically through intersections (agents represented by cars), navigating robots in automated production/storage facilities (agents represented by robots), or the evacuation of people from danger (agents are represented by the evacuating people). In these cases, *makespan-optimality* is particularly desirable because it translates into the quickest possible clearance of an intersection, the fastest production lines in manufacturing, and the most efficient evacuation, respectively.

1.2 Contribution

The main contribution of this paper lies in the evaluation of makespan-optimal CPF solving methods based on the reduction of CPF to *propositional satisfiability* (SAT) [11]. The main reason for the reduction of CPF to SAT is the ability to use modern off-the-shelf SAT solvers [3, 5, 8] to answer the resulting propositional formulae. The reduction to SAT enables us to use the efficient *propagation*, *heuristics*, and *learning* techniques implemented in SAT solvers to solve CPF. Several encodings of CPF into propositional formulae are discussed.

The fundamental approach common to the various encodings presented here is the use of the technique of *time expansion* for the underlying graph [28, 54]. Time expansion results in a*time-expanded graph* that consists of a copy of the underlying graph for each time step. This structure enables us to represent all possible configurations of agents in vertices in all time steps. In an analogical approach common in SAT-based domain independent planning, states in all time steps are representable as a graph-like structure. This approach was introduced in the GRAPHPLAN planner [14] and then modified in the SATPLAN [28] planner and its successors [7, 24, 41, 66] for SAT-based planning.

We aim to evaluate various encodings theoretically in terms of size and experimentally as part of a makespan-optimal CPF solver. We compare our SAT-based solver with search-based techniques for solving CPF in a makespan-optimal way such as *operator*

decomposition/independence detection method (OD+ID) [51], *conflict-based search* (CBS) [50], and *increasing cost tree search* (ICTS) [49].

Multiple hand-tailored encodings for solving CPF makespan-optimally are available. They differ mainly in the way the placement of the agents – which are the major decision variables – in the time-expanded graph is represented. Two distinct representations are used in propositional encodings – **log** and **direct** encodings:

- **Log representation** (*log encoding)* uses a vector of $\lceil \log_2 n \rceil$ bits/propositional variables to represent an n-state variable [9, 25, 36]. The representation is based on the same idea as the binary representation of numbers in a computer memory.
- **Direct representation** (*direct encoding*) [9, 61, 65] uses a vector of n bits/propositional variables to represent an n-state variable. In contrast to log encoding, where, in principle, any valuation of propositional variables corresponds to the valuation of the relevant n-state variable,[1] only those valuations that have exactly one propositional variable in the vector set to $TRUE$ are valid in the direct encoding. This requires the use of additional constraints to ensure that invalid valuations are excluded.

Both representations have their pros and cons. The log representation is more space-efficient than the direct representation. The direct representation, on the other hand, provides better support for *constraint propagation* in the SAT solver [37].

Several representative encodings of CPF will be described and analyzed in the next section – two encodings that are based purely on log representation, one encoding based on mixed representation (some variables are log variables, some are direct), and one based purely on directly represented variables.

The rest of the paper is structured as follows. After a survey of the literature on CPF we will describe in detail the concept of the time-expanded graph. Next, we will introduce in detail the encodings based on the concept of the *time-expanded graph*. A theoretical analysis with a focus on the size of the encodings will follow. Finally, we will provide an experimental evaluation and comparison of the encodings and will compare them with selected search-based approaches.

2 Related work

While this paper is primarily focused on solving CPF problems and comparing various dedicated CPF solvers, we cannot omit a related stream of work dealing with propositional encodings from the modeling point of view.

These publications look at the ways to translate a problem formulated as the satisfaction of a conjunction of constraints – a *constraint satisfaction problem* (CSP) [21] – into propositional formulae.

CSP-to-SAT reformulations must be approached from the perspective of the needs of the existing CPF solution techniques. For this reason, we will first survey the related work on dedicated CPF solvers. Next, we will discuss publications dealing with CSP-to-SAT encodings and analyze those with respect to CPF needs.

[1] If n is not a power of 2, then some valuations of propositional variables are not used.

2.1 Survey of related work in cooperative path finding

There are several dedicated alternatives to solving CPF by reducing it to SAT:

(i) **Search-based methods related to A*.** Some of these methods are based on the A* algorithm and include both optimal and suboptimal cases. A*-based algorithms compute all configurations of agents that can be obtained by putting together the valid moves of all agents while at the same time pruning out unpromising configurations. A prominent example of these algorithms is *operator decomposition/independence detection* (OD+ID) [51], which improves on the standard A* by ordering the agents, reducing the branching factor and enabling better search space pruning. Additionally, *independence detection* divides the set of agents into independent groups that can be dealt with separately, thus further decreasing the size of the composite search space by reducing its dimensionality.

Another example of A*-based methods is Silver's suboptimal WHCA* algorithm [47], in which the cooperation between agents is integrated via a conflict avoidance table that keeps track of the agents' trajectories over time. Yet another example is M* [64], which changes the dimensionality of the composite search space dynamically.

(ii) **Search-based methods over a transformed search space.** Algorithms operating with a different view of the search space include sum-of-costs optimal *conflict-based search* (CBS) [50] and *increasing cost tree search* (ICTS) [49]. CBS searches for a consistent path for each agent. The consistency is defined with respect to the conflict avoidance constraint. If a collision between two agents occurs in a certain vertex and a certain time step, a new disjunctive constraint forbidding one or the other agent from showing up in the particular vertex/time is introduced and the search branches into two new directions.

The ICTS algorithm operates very similarly to the SAT-based optimal CPF solution methods. ICTS performs a top-level search, testing all possible distributions of a given cost among the individual agents. Once each agent has been assigned its costs, a low-level search will attempt to find non-conflicting paths for the agents within the individual cost limits. The search for the individual paths occurs in a *multi-value decision diagram* similar to a time-expanded graph – the vertices of the underlying graph are copied in every time step, while the vertices unreachable within the cost limits are omitted to reduce the size of the search space. If the search for non-conflicting paths fails, the total cost is incremented and the process is repeated.

(iii) **Polynomial-time complete algorithms.** Fast polynomial time methods are a very important category of algorithms that generate makespan-suboptimal solutions to the CPF problem. As well as the PUSH-AND-SWAP [32], PUSHAND-ROTATE [68, 69], and BIBOX algorithms [56], this category includes algorithms originally designed for graph pebbling (PMG) [31, 70]. Although these algorithms have the common drawback of producing solutions that are very far from the optimum, they are complete and scalable with the size of the input instance. The completeness ensures that these algorithms can produce an answer even if the input instance has no solution.

Thanks to their completeness, these algorithms can be used to check whether or not a given instance is solvable before an incomplete method assuming a solvable input is applied. This is also the case of the present SAT-based approach, which cannot, by design, detect whether the input instance is unsolvable.

(iv) **Other algorithms.** Trajectory conflict resolution methods are often used in a continuous space [19, 20]. The main advantage of these methods is their scalability in

dealing with distributed variants. On the other hand, there's usually no guarantee of the completeness and optimality of the solutions.

2.2 Survey of related work in modeling paradigms

Literature on SAT encodings has focused on the translation of CSP into propositional formulae [37, 38, 61, 62] in *conjunctive normal form* (CNF) [11] (conjunction of *clauses* where a clause is a disjunction of *literals* – positive or negative *Boolean variables*). One of the key differentiating features of the existing encodings is the way they model finite domain variables [9, 25, 65]. The choice of variable encodings further determines the way the problem-specific constraints are expressed as clauses. It is important to note that in CPFs, we need to encode very specific constraints that are imposed by *static* and *dynamic* constraints.

Various design approaches to encoding problems in SAT have been proposed. In [38] Petke analyzes several characteristics that a good SAT-encoding design should have:

(a) **Compactness.** The encoding should be as small as possible. The two common measures of the size of a formula in CNF are the total number of propositional variables and the number of clauses. The number of propositional variables determines the size of the search space. However, the total number of propositional variables does not often correspond to the true dimensionality of the formula, as the values of some variables may be directly deduced from other valuations.

(b) **High solution density.** Solution density is calculated as the total number of solutions divided by 2^N where N is the total number of propositional variables in the formula. In [17] some evidence is given that with an increasing solution density, the search algorithm might find the solution to such a formula more easily. It is usually difficult to achieve a high solution density without an in-depth understanding of the problem being encoded.

(c) **Suitability for deductions.** One of the very powerful techniques implemented in SAT solvers is *Boolean constraint propagation*, particularly *unit propagation* [11] (once all but one of the literals in a clause are set to $FALSE$, the last remaining literal must be set to $TRUE$ to satisfy the clause). In an encoding that lends itself to unit propagation, many variables may be assigned by deduction only, i.e., without a search [23]. It can be intuitively observed that having short clauses in a formula increases the probability of deductions.

The **direct representation** of n-state finite domain variables in CPF, that is, having a propositional variable for each of n states, has the advantage of supporting *(c) deductions*, but is less advantageous when it comes to *(a) compactness* and *(b) solution density*. Note that the static and dynamic constraints in CPF are rather symbolic implications such as "if agent a_i is in vertex v in time step t, then it is either in v or the neighbors of v in $t + 1$" and trivial arithmetic constraints like *at-most-one/at-least-one* – "at most one agent is in vertex v in time step t". These simple constraints can be expressed in short clauses involving directly represented variables [61]. On the other hand, many possible assignments to propositional variables form invalid patterns that need to be excluded by additional constraints to ensure assignment consistency (exactly one propositional variable in the vector encoding a given finite domain variable can be set to $TRUE$). Therefore, the number of clauses is large and the solution density is low.

A **log encoding** is more compact because it uses only $\lceil \log_2 n \rceil$ propositional variables to represent an n-state finite domain variable via a binary encoding. Moreover, there is no

need to introduce any consistency constraints since all assignments to the vector of $\lceil \log_2 n \rceil$ propositional variables form a valid pattern. Hence, log encodings should have the advantage of *(a) compactness* and *(b) solution density*. However, that cannot be positively said of CPF because constraints such as the aforementioned implication need to select a concrete pattern of assignment to the vector of $\lceil \log_2 n \rceil$ propositional variables in order to check whether the LHS of a given implication holds. Such pattern matching requires complex encodings of the log representation where the individual bits are queried. Sometimes a log encoding also requires the introduction of fresh auxiliary propositional variables and this may compromise the compactness of the encoding [15].

There are several other encodings apart from direct and log encodings that can be used to translate a CSP to SAT. ***Order encoding*** is an important representative [18, 37, 48, 61] that attempts to preserve the deduction-related advantages of the direct encoding, but offers better support for the representation of linear arithmetic constraints. It uses n propositional variables for representing relations $x \leq i$ for $i \in \{1, 2, \ldots n\}$. Again, consistency constraints need to be encoded, but linear relations are represented easily. As there are no linear relations in CPF, an order encoding would be very similar to a direct encoding when applied to CPF.

2.3 Reduction of CPF to ASP, CSP, and IP

The applicability of various other formalisms in a compilation-based approach to CPF has been demonstrated in literature. The reductions of CPF to *answer set programming* (ASP) [22] show that the various static and dynamic constraints in CPF can be represented declaratively. *Integer programming* (IP) was used as a target formalism for CPF reduction in [71]. IP models provide important insights into the relationship between CPF and multi-commodity flows – a version of a time-expanded graph is proposed, in which a multi-commodity flow directly corresponds to a CPF solution.

We conducted our own preliminary study of the usability of CSP and IP in solving CPF problems [27].[2] Our CSP model of CPF extensively uses *global all-different constraints* [40] to express static CPF constraints. We implemented a CSP-based solver on top of the GECODE system [44, 45]. In addition, we implemented an *integer programming* model of CPF on top of GUROBI [12, 13] that exactly corresponded to the INVERSE SAT encoding that will be discussed later.

The findings of these experiments with CSP and IP models indicate that the scalability of both the CSP and IP models is limited. Despite the use of powerful global constraints, the CPS only performed better than IP and SAT on very small CPF instances. With an increasing number of agents in an instance the performance of the CSP model rapidly declined. The IP model delivered better performance than CSP and did not exhibit any rapid decline in performance; however, it was ultimately outperformed by the SAT approach on larger instances.

3 Background

In this section we will introduce some basic definitions relating to CPF. The environment in which agents move is represented by an undirected graph. 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.

[2] The implementation was carried out by a student supervised by the author as a part of his bachelor thesis.

A *configuration* (*arrangement*) of agents in an environment is modeled by assigning the vertices of the graph to the agents. Let $A = \{a_1, a_2, \ldots, a_\mu\}$ be a finite set of *agents*. Then, a configuration of agents in the vertices of graph G is fully described by the *position* function $\alpha : A \longrightarrow V$ which means that agent $a \in A$ is located in vertex $\alpha(a)$.

The standard *static constraints* in CPF can be easily expressed using α: having at most one agent in a vertex – when translated into a position function – means that α is a *uniquely invertible function*. The generalized inverse of α denoted as $\alpha^{-1} : V \longrightarrow A \cup \{\perp\}$ will return the agent located in a given vertex or $\perp$ if the vertex is empty.

Definition 1 (COOPERATIVE PATHFINDING). An instance of the *cooperative pathfinding* problem (CPF) is a quadruple $\Sigma = [G = (V, E)\, A, \alpha_0, \alpha_+]$ where the position functions α_0 and α_+ define the initial and goal configurations, respectively, of a set of agents A in G.

Dynamic constraints are expressed in discrete time steps. Configuration α_i in time step i can be instantaneously transformed by the agents' movements to configuration α_{i+1} in time step $i + 1$, provided that the following constraints are satisfied:

$$\forall a \in A \text{ either } \alpha_i(a) = \alpha_{i+1}(a) \text{ or } \{\alpha_i(a), \alpha_{i+1}(a)\} \in E \text{ holds} \tag{1}$$

(agents move only by traversing the edges of the graph or do not move at all)

$$\forall a \in A \alpha_i(a) \neq \alpha_{i+1}(a) \Rightarrow \alpha_i^{-1}(\alpha_{i+1}(a)) = \perp \tag{2}$$

(the target vertex of an agent's move must be empty), and

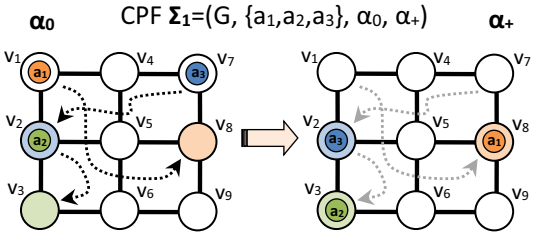
$$\forall ab \in A a \neq b \Rightarrow \alpha_{i+1}(a) \neq \alpha_{i+1}(b) \tag{3}$$

(no two agents can enter the same target vertex or else they would collide).

It is important to note that constraint (3), together with the assumption that α_i is uniquely invertible, ensures that α_{i+1} is uniquely invertible as well.

The task in CPF is to transform α_0 using the aforementioned valid transitions into α^+. An illustration of CPF and valid transitions is given in Fig. 1.

Definition 2 (SOLUTION, MAKESPAN). A *solution* with a *makespan* η to a CPF instance $\Sigma = [G A \alpha_0 \alpha^+]$ is a sequence of configurations of agents $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_\eta]$ where $\alpha_\eta = \alpha^+$ and α_{i+1} is the result of a valid transition from α_i for every $i = 1, 2, \ldots, \eta - 1$.



$\vec{s}$	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

Fig. 1 *An example of the cooperative path-finding problem (CPF).* Three agents a_1, a_2, and a_3 need to move from their initial positions represented by α_0 to their goal positions represented by α_+. A solution with a makespan of 4 is shown.

The number $\left|\vec{s}\right| = \eta$ is the *makespan* of solution $\vec{s}$. An often asked question is whether there is a solution of Σ with a given makespan $\eta \in \mathbb{N}$. This problem is referred to as the (*makespan*) *decision variant of CPF*. The decision variant of CPF is known to be NP-complete [39]. For this reason, finding a makespan-optimal solution to CPF presents a significant challenge.

Let $\xi_0(a_i)$ be the length of the shortest path connecting $\alpha_0(a_i)$ to $\alpha_+(a_i)$. Next, let $\eta_0 = \max_{i=1}^{\mu} \xi_0(a_i)$ denote the length of the longest of the shortest paths. Note that η_0 is the lower bound for the makespan, as even in the best-case scenario each agent a_i needs to travel the distance $\xi_0(a_i)$.

4 Solving CPF optimally via propositional satisfiability

The question we address is how to obtain *makespan-optimal solutions* to CPF problems. We suggest using *propositional satisfiability* (SAT) [11] as the key technique and integrating off-the-shelf SAT solver [3] into the CPF solving process. Considering that the decision variant of CPF is NP-complete, SAT represents a formalism of just the power we need.

A propositional formula $F(\Sigma, \eta)$ satisfiable if and only if a given CPF Σ with makespan η is solvable can be constructed – $F(\Sigma, \eta)$ encodes the decision variant of CPF Σ. If we are able to construct such a formula $F(\Sigma, \eta)$, we will be able obtain the optimal makespan for CPF Σ by submitting multiple queries $F(\Sigma, \eta)$ with different makespan bounds η to the SAT solver.

There are various strategies for optimizing queries aimed at obtaining optimal makespans. Many have been developed for the *maximum satisfiability* SAT variant [34]. During our preliminary experimentation, we discovered that the most efficient optimization strategy was simply to test sequentially for makespans $\eta = \eta_0, \eta_0 + 1, \ldots$ until the η equal to the optimal makespan was reached. This strategy will hereinafter be referred to as the *incremental strategy*. This strategy is made possible by the monotonicity of the satisfiability of $F(\Sigma, \eta)$ with respect to parameter η (the function is non-decreasing in η).

The use of this type of queries is justified by the non-uniform cost of the queries with respect to parameter η. The query runtime is exponential in the size of $F(\Sigma, \eta)$, while the length of the formula is linear in η. Hence, the total runtime of all queries $\eta = \eta_0, \eta_0 + 1, \ldots, \eta^* - 1$ is comparable to the runtime of a query about $\eta = \eta^*$ and the use of the incremental strategy is thus justified. The incremental strategy is also implemented into such domain-independent planners as SATPLAN [28], SASE [24] and others [41]. The pseudo-code of a makespan-optimal SAT-based MAPF solution based on the incremental strategy is given as Algorithm 1.

An important property of the propositional encoding $F(\Sigma, \eta)$ is that a solution to CPF Σ of makespan η can be unambiguously extracted from the satisfying valuation of $F(\Sigma, \eta)$ (otherwise, an equivalence between the solvability of CPF Σ and $F(\Sigma, \eta)$ could be trivially established by setting $F(\Sigma, \eta) \equiv TRUE$ if Σ is solvable in η time steps and $F(\Sigma, \eta) \equiv FALSE$ in all other cases). Prior to using a SAT-based optimization we can use one of the fast polynomial time algorithms described in [31, 56, 69] to check if the instance in line 1 is solvable.

One of the important advantages of solving CPF as SAT is that there are many powerful SAT solvers [2, 3] that use various techniques such as intelligent search space pruning and learning. Considering that a SAT solver has been included as a module in the proposed framework, any progress in solving SAT directly translates into progress in solving CPF. However, the design of CPF-to-SAT encodings has a significant impact on the efficiency of SAT solution techniques in solving CPF problems.

Algorithm 1. *SAT-based optimal CPF solving – incremental strategy.* The algorithm finds the shortest possible makespan η provided that CPF $\Sigma = (G, A, \alpha_0, \alpha_+)$ is solvable. Questions as to whether CPF Σ has a solution are constructed with respect to increases in makespan and submitted to the SAT solver.

input: Σ – a CPF instance
output: a pair consisting of the optimal makespan and the corresponding optimal solution

function *Find-Optimal-Solution-Sequentially* ($\Sigma = (\mathrm{G}, A, \alpha_0, \alpha_+)$): **pair**
1: **if** Σ is solvable **then**
2: $\quad \eta \leftarrow \eta_0$
3: $\quad$ **loop**
4: $\quad\quad F(\Sigma, \eta) \leftarrow$*Encode-CPF-as-SAT* (Σ, η)
5: $\quad\quad$ **if** *Solve-SAT* $(F(\Sigma, \eta))$ **then**
6: $\quad\quad\quad \vec{s} \leftarrow$ *Extract-Solution-from-Valuation*$(F(\Sigma, \eta))$
7: $\quad\quad\quad$ **return** $(\eta, \vec{s})$
8: $\quad\quad \eta \leftarrow \eta + 1$
9: **else**
10: $\quad$ **return** $(\infty, \emptyset)$

This paper focuses on SAT encodings; querying strategies are out of the scope of this work. Nevertheless, let us mention that an in-depth study of querying strategies is given in [41]. There is great potential in querying strategies, as these can substantially speed up the planning process, especially when they are combined with parallel processing.

4.1 Time-expanded graphs

The *trajectory* of an agent over G in time may not necessarily be a *simple path*. Generally, a single vertex may be visited multiple times. For this reason, we use a graph that is derived from G by expanding it in time. The trajectory of each agent over G will correspond to a simple path that touches each vertex at most once in the *expanded version* of G. A detailed description of the expanded graph is given in the following definition and illustrated in Fig. 3.

Definition 3 (TIME-EXPANDED GRAPH - $\mathrm{Exp}_\mathrm{T}(G, \eta)$). Let $G = (V, E)$ be an undirected graph and $\eta \in \mathbb{N}$. A *time-expanded graph* with $\eta + 1$ time layers (indexed from to η) associated with G is a directed graph $\mathrm{Exp}_\mathrm{T}(G, \eta) = (V \times \{0, 1, \ldots, \eta\}, E')$ where $E' = \{([u, t][v, t+1]) \mid \{u, v\} \in E; t = 0, 1, \ldots, \eta - 1\} \cup \{v \in V; t = 0, 1, \ldots, \eta - 1\}$.

The notation u^t will sometimes be used instead of $[u, t]$ in figures. The directed edges that connect the vertices corresponding to the distinct vertices in G are called *regular* – these will correspond to an agent's movement; the remaining edges correspond to wait actions.

The search for a solution with a makespan bound η can be understood as a search for a collection of *non-overlapping vertex-disjoint paths* in the corresponding time-expanded graph consisting of η layers $\mathrm{Exp}_\mathrm{T}(G, \eta)$. Non-overlapping vertex-disjoint paths must have a disjoint set of the endpoints of *regular edges* in the consecutive time layers of $\mathrm{Exp}_\mathrm{T}(G, \eta)$, as described in the following definition.

Definition 4 (NON-OVERLAPPING VERTEX-DISJOINT PATHS IN $\mathrm{Exp}_\mathrm{T}(G, \eta)$). A set of paths $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$ in $\mathrm{Exp}_\mathrm{T}(G, \eta)$ where π_i connects $[x_i]$ with $[y_i, \eta]$ for

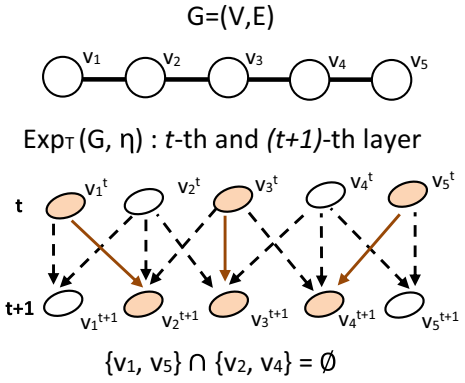


Fig. 2 *An illustration of non-overlapping vertex-disjoint paths*. Parts of three non-overlapping paths between time layers t and $t+1$ of $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$ are shown

$x_i, y_i \in V$ and $i = 1, 2, \ldots, \mu$ form *non-overlapping vertex-disjoint paths* if and only if $\pi_i \cap \pi_j = \emptyset$ and $\{\pi_i[t, 2] | \pi_i[t, 2] \neq \pi_i[t+1, 2] \wedge i = 1, 2, \ldots, \mu\} \cap \{\pi_i[t+1, 2] | \pi_i[t, 2] \neq \pi_i[t+1, 2] \wedge i = 1, 2, \ldots, \mu\} = \emptyset$[3] for $t = 0, 1, \ldots, \eta - 1$ for any two $i, j \in \{1, 2, \ldots, \mu\}$ with $i \neq j$.

Non-overlapping vertex-disjoint paths crossing between two consecutive time layers of $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$ are shown in Fig. 2. The correspondence between the existence of a CPF solution and non-overlapping vertex-disjoint paths is established in the next proposition (Fig. 3).

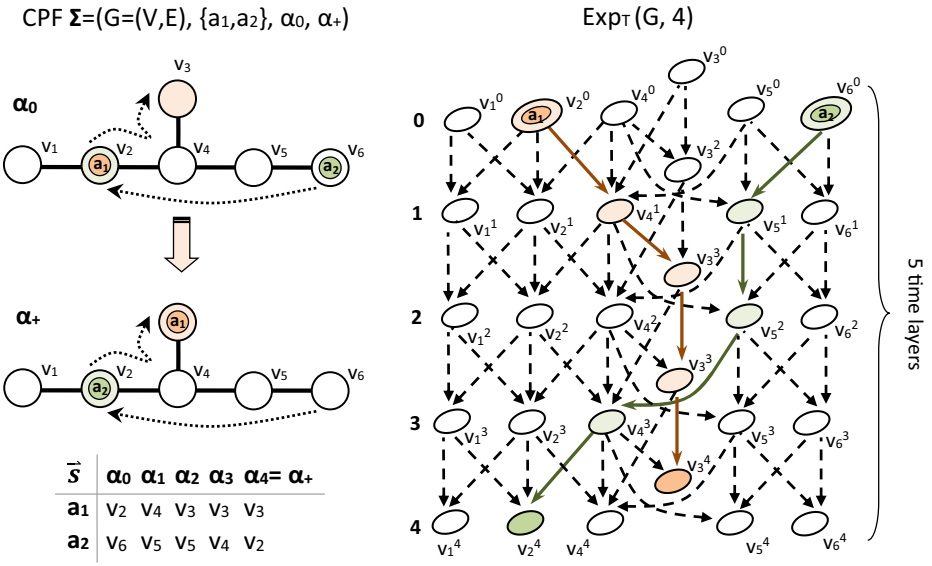
Proposition 1 (Non-Overlapping Vertex-Disjoint Paths in Exp$_T$). *A solution with makespan $\eta \in \mathbb{N}$ of a CPF $\Sigma = (G, A, \alpha_0, \alpha_+)$ with $A = \{a_1, a_2, \ldots, a_\mu\}$ exists if and only if there is a set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$ of non-overlapping vertex-disjoint paths in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$ so that π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i), \eta]$ for $i = 1, 2, \ldots, \mu$.*

Proof Assume that a given CPF Σ has a solution $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_\eta]$ with makespan η. Then vertex-disjoint paths $\pi_1, \pi_2, \ldots, \pi_\mu$ in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$ can be constructed from $\vec{s}$. Path π_i will correspond to the trajectory of agent a_i; that is, $\pi_i = ([\alpha_0(a_i), 0], [\alpha_1(a_i), 1], \ldots, [\alpha_\eta(a_i), \eta])$. The path thus constructed is a correct path in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$, since $\{\alpha_t(a_i), \alpha_{t+1}(a_i)\} \in E$ or $\alpha_t(a_i) = \alpha_{l+1}(a_i)$ for $t = 0, 1, \ldots, \eta - 1$; that is, $([\alpha_t(a_i), t], [\alpha_{t+1}(a_i), t+1]) \in E'$ which holds by construction of $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$. Obviously π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i) = \alpha_\eta(a_i), \eta]$ in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$.

It only remains to check that no two constructed paths intersect and that these paths are non-overlapping. Condition (3) together with the unique invertibility of α_0 ensures that no two paths share a vertex, or else the agents would collide. Conditions (1) and (2) together ensure that the endpoints of the edges of the paths overlap between consecutive time layers only if there are non-regular edges – that is, edges that continue towards the same vertex in the next time layer.

Let us discuss the opposite implication. Assume that there are non-overlapping vertex-disjoint paths $\pi_1, \pi_2, \ldots, \pi_\mu$ in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$. We will construct a solution to CPF Σ with makespan η. Let $\pi_i = ([u_0, 0], [u_1, 1], [u_2, 2], \ldots, [u_\eta, \eta])$, $u_t \in V$ for $t =$

[3]The notation $\pi_i[t, 2]$ refers to the second component of the t-th element of π_i.



$\vec{s}$	α_0	α_1	α_2	α_3	$\alpha_4 = \alpha_+$
a_1	v_2	v_4	v_3	v_3	v_3
a_2	v_6	v_5	v_5	v_4	v_2

Fig. 3 *An example of CPF and the related time-expanded graph.* The time-expanded graph $\mathrm{Exp}_T(G, 4)$ consisting of 5 time layers is built for a given CPF Σ. A 5-step solution to Σ can be represented as a search for a set of non-overlapping vertex-disjoint paths connecting the initial positions of the agents in layer 0 to their goal positions in the last layer of $\mathrm{Exp}_T(G, 4)$

$0, 1, \ldots, \eta$. The trajectory of agent a_i is as follows: $\alpha_0(a_i) = u_0, \alpha_1(a_i) = u_1, \alpha_2(a_i) = u_2, \ldots, \alpha_\eta(a_i) = u_\eta$.

We can now verify that conditions (1) – (3) plus the unique invertibility of α_t are satisfied by this construction. The paths are vertex-disjoint, therefore agents do not collide when they follow these paths – condition (3) is satisfied and the unique invertibility of α_t is ensured. As paths do not overlap, agents either wait in a vertex or move into a vertex that was unoccupied in the previous time step. That satisfies (1) and (2). □

5 Propositional encodings of CPF

Time-expanded graphs represent a conceptual step towards the design of a propositional formula that is satisfiable if and only if a CPF has a solution with a given makespan. Moreover, we need to be able to derive CPF solutions from a satisfying valuation of a formula. A time-expanded graph can be used as a basis for such a formula, as it can capture all configurations of agents in G in all time steps up until the final step.

We will describe several propositional encodings, starting with those based on the **log** representation of finite domain state variables, i.e., INVERSE [53] and ALL-DIFFERENT [54] encodings. We will then proceed to discuss **directly** encoded variables in an encoding called MATCHING [57] (a *mixture of* **direct and log** encoded variables). Next, we will look at a purely direct encoding called DIRECT/SIMPLIFIED [58, 59]. In contrast to the original papers in which these encodings were proposed we introduce the encodings using a unified notation.

5.1 INVERSE propositional encoding

Let $\deg_G(v)$ denote the degree of vertex v in G; that is, $\deg_G(v)$ is the number of edges from E incident with v. It is further assumed that the neighbors of each vertex v in G are ordered. The sequential number of each neighbor u of v is determined by the assignment $\sigma_v : \{u | \{v, u\} \in E\} \longrightarrow \{1, 2, \ldots, \deg_G(v)\}$. In other words, u is the $\sigma_v(u)$-th neighbor of v. An inverse σ_v^{-1} is naturally defined. That is, $\sigma_v^{-1}(i)$ returns the i-th neighbor of v for $i \in \{1, 2, \ldots, \deg_G(v)\}$.

The following definition introduces the INVERSE encoding over finite domain state variables which will be further refined into bit-vectors using a *log* representation.

Definition 5 (INVESE ENCODING – $F_{INV}(\eta, \Sigma)$). Assume CPF $\Sigma = [G, A, \alpha_0, \alpha_+]$ with $G = (V, E)$. An *INVERSEencoding* for CPF Σ consists of the following finite domain **variables**:

- $\mathcal{A}_v^t \in \{0, 1, \ldots, \mu\}$ for every $v \in V$ and time step $t \in \{0, 1, \ldots, \eta\}$ to model agent *occurrences* in vertices and
- $\mathcal{T}_v^t \in \{0, 1, \ldots, 2 \cdot \deg_G(v)\}$ for every $v \in V$ and $t \in \{0, 1, \ldots, \eta - 1\}$ to represent the agents' movement between vertices.

The**constraints** of the INVERSE encoding are as follows:

$$\mathcal{T}_v^t = 0 \Rightarrow \mathcal{A}_v^{t+1} = \mathcal{A}_v^t \qquad \text{for every } v \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\} \tag{4}$$

(if no movement occurs in v then v retains the same agent in time step t+1)

$$0 < T_v^t \leq \deg_G(v) \Rightarrow \mathcal{A}_u^t = 0 \wedge \mathcal{A}_u^{t+1} = \mathcal{A}_v^t \wedge \mathcal{T}_u^t = \sigma_u(v) + \deg_G(u), \\ \text{for every } v \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \text{where } u = o_v^{-1}\left(\mathcal{T}_v^t\right) \tag{5}$$

(an agent leaves v for its $\mathcal{T}_v^t$ -th neighbor u)

$$\deg_G(v) < T_v^t \leq 2 \cdot \deg_G(v) \Rightarrow \mathcal{T}_u^t = \sigma_u(v), \\ \text{for every } v \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \text{where } u = \sigma_v^{-1}\left(\mathcal{T}_v^t - \deg_G(v)\right) \tag{6}$$

(an agent arrives in v from its $(\mathcal{T}_v^t - \deg_G(v))$ -th neighbor u).

Initial and **goal** configurations will be expressed using the following constraints:

$$\left.\begin{array}{ll} \mathcal{A}_u^0 = i & \text{for } u \in V \text{ if there is } i \in \{1,2,\ldots,\mu\} \\ & \text{such that } \alpha_0(a_i) = u \\ \mathcal{A}_u^0 = 0 & \text{for } u \in V \text{ if } (\forall a \in A)\alpha_0(a) \neq u \end{array}\right\} \text{Initial positions} \tag{7, 8}$$

$$\left.\begin{array}{ll} \mathcal{A}_u^\eta = i & \text{for } u \in V \text{ if there is } i \in \{1,2,\ldots,\mu\} \\ & \text{such that } \alpha_+(a_i) = u \\ \mathcal{A}_u^\eta = 0 & \text{for } u \in V \text{ if } (\forall a \in A)\alpha_+(a) \neq u \end{array}\right\} \text{Goal positions} \tag{9, 10}$$

The resulting propositional formula in CNF where $\mathcal{A}_v^t$ and $\mathcal{T}_v^t$ variables are replaced with bit vectors with a binary encoding and where constraints are replaced accordingly will be denoted as $F_{INV}(\eta, \Sigma)$.

The meaning of $\mathcal{A}_v^t$ variables corresponds to an inverse position function in time step t. Specifically, $\mathcal{A}_v^t = j$ iff $\alpha_t^{-1}(v) = a_j$ and $\mathcal{A}_v^t = 0$ iff $\alpha_t^{-1}(v) = \perp$. Variables $\mathcal{T}_v^t$ represent the agents' movements between vertices. A zero value is reserved for no movement. One

half of the remaining positive values from 1 to $\deg_G(v)$ represent outgoing movements from v to any neighbor indicated by $\mathcal{T}_v^l$; the other half of the positive values represent incoming movements to v from any of its neighbors indicated by $\mathcal{T}_v^l - \deg_G(v)$.

Variables $\mathcal{A}_v^t$ are modeled by a vector of $\lceil \log_2(\mu+1) \rceil$ propositional variables where individual (propositional) bits will be accessed by a bit index $\mathbb{i} \in \{0,1,\ldots,[\log_2(\mu+1)]-1\}$; the notation $\mathcal{A}_v^t[\mathbb{i}]$ stands for $\mathbb{i}$-th bit of $\mathcal{A}_v^t$. Similarly, the variables $\mathcal{T}_v^t$ are modeled by vectors of $\lceil \log_2(2{\cdot}\deg_G(v)+1) \rceil$ propositional variables. Note that the topology of typical environments only exhibits a local interconnection, that is, $\deg_G(v)$ is usually a small value. This means that $\deg_G(v) \ll \mu$ and $\deg_G(v) \ll n$. If the number of states of the represented finite domain variable is different from a power of 2, then extra states are forbidden.

We need to distinguish between all $2 \cdot \deg_G(v) + 1$ states of the $\mathcal{T}_v^t$ variables to be able to posit constrains like "if $\mathcal{T}_v^t$ equals to 5" assuming that $\deg_G(v) = 2$ "an agent will arrive in v from v's $5 - \deg_G(v) = 3$rd neighbor in time step $t+1$". On the other hand, dealing with $\mathcal{A}_v^t$ variables is easier, as we only have to model the equality between them and the equality to zero and we do not need to distinguish between so many cases.

Let $\mathbb{b}\colon \mathbb{N}_0 \times \mathbb{N}_0 \to \{0,1\}$ be a binary representation of positive integers where $\mathbb{b}(x,\mathbb{i})$ represents the value of the $\mathbb{i}$-th bit in a binary encoding of x; that is $x = \sum_{\mathbb{i}=0}^{b-1} \mathbb{b}(x,\mathbb{i}) \cdot 2^{\mathbb{i}}$. The equality of variable $\mathcal{T}_v^t$ to constant $c \in \{0,1,\ldots,2{\cdot}\deg_G(v)\}$ will be expressed as the following conjunction:

$$\begin{aligned} \mathrm{con}_{=}(\mathcal{T}_v^t, c) &= \bigwedge_{\mathbb{i}=0}^{\lceil \log_2(2\cdot\deg_G(v)+1)\rceil-1} \mathrm{lit}(\mathcal{T}_v^t, c, \mathbb{i}) \\ \text{where } \mathrm{lit}(\mathcal{T}_v^t, c, \mathbb{i}) &= \begin{cases} \mathcal{T}_v^t[\mathbb{i}] & \text{iff } \mathbb{b}(c,\mathbb{i}) = 1 \\ \neg\mathcal{T}_v^t[\mathbb{i}] & \text{iff } \mathbb{b}(c,\mathbb{i}) = 0 \end{cases} \end{aligned} \tag{11}$$

The equality between variables $\mathcal{A}_v^t$ and $\mathcal{A}_u^{t+1}$ is expressed as the equalities between bits at corresponding positions in the following conjunction of equivalences:

$$\mathrm{var}_{=}(\mathcal{A}_v^t, \mathcal{A}_u^{t+1}) = \bigwedge_{\mathbb{i}=0}^{\lceil \log_2(\mu+1)\rceil-1} (\neg\mathcal{A}_v^t[\mathbb{i}] \vee \mathcal{A}_u^{t+1}[\mathbb{i}]) \wedge (\mathcal{A}_v^t[\mathbb{i}] \vee \neg\mathcal{A}_u^{t+1}[\mathbb{i}]) \tag{12}$$

The elementary constructions above are put together to represent constraints (4)–(6) using Tseitin's encoding [63], which introduces *auxiliary propositional variables*; in this context any other than auxiliary variables will be called *visible*. The following auxiliary propositional variables are introduced: $a_{v,t}^{\mathrm{zero}}$ for representing an empty vertex v in time step t, $a_{u,v,t}^{=}$ for representing equality between $\mathcal{A}_v^t$ and $\mathcal{A}_u^{t+1}$, and $a_{v,t,c}^{\mathrm{tran}}$ for representing equality

$\mathcal{T}_v^t = c$. The connection of auxiliary variables to their exact meaning with respect to the main $\mathcal{A}_v^t$ and $\mathcal{T}_v^t$ variables is ensured by the following constraints:

$$a_{v,t}^{\text{zero}} \Rightarrow \text{con}_{=}\left(\mathcal{A}_v^t, 0\right) \tag{13}$$

$$a_{u,v,t}^{=} \Rightarrow \text{var}_{=}\left(\mathcal{A}_v^t, \mathcal{A}_u^{t+1}\right) \tag{14}$$

$$a_{v,t,c}^{\text{tran}} \Leftrightarrow \text{con}_{=}\left(\mathcal{T}_v^t, c\right) \tag{15}$$

Considering that the $\mathcal{A}_v^t$ variables appear only on the right-hand side of the implications in constraints (4)–(6) of the INVERSE encoding, it is enough to connect their auxiliary variables by implications. On the other hand, the $\mathcal{T}_v^t$ variables appear on both sides of the implications in (4)–(6) and therefore need to be connected to their auxiliary variables by equivalences.

Having introduced the auxiliary variables above, the INVERSE encoding constraints on top of them can be expressed as follows:

$$a_{v,t,0}^{\text{tran}} \Rightarrow a_{v,v,t}^{=} \tag{16}$$

for every $v \in V$ and $t \in \{0, 1, \ldots, \eta - 1\}$

$$a_{v,t,c}^{\text{tran}} \Rightarrow a_{u,t}^{\text{zero}} \wedge a_{u,v,t}^{=} \wedge a_{u,t,\sigma_u(v)+\deg_G(u)}^{\text{tran}} \tag{17}$$

for each $0 < c \leq \deg_G(v)$, $v \in V$ and $t \in \{0, 1, \ldots, \eta - 1\}$, where $u = o_v^{-1}(c)$

$$a_{v,t,c}^{\text{tran}} \Rightarrow a_{u,t,\sigma_u(v)}^{\text{tran}} \tag{18}$$

for each $\deg_G(v) < c \leq 2{\cdot}deg_G(v)$, $v{\in}V$ and $t \in \{0, 1, \ldots, \eta{-}1\}$, where $u = \sigma_v^{-1}\left(\text{T}_v^t - \deg_G(v)\right)$

The space complexity of the INVERSE encoding is summarized in the following proposition. Only the regular time layers are counted; the asymptotical space requirements of the initial and final time layers are dominated by the rest.

Proposition 2 *(*INVERSE ENCODING SIZE*). The number of visible propositional variables in* $F_{INV}(\eta, \Sigma)$ *is* $O\left(\eta \cdot \left(|V| \cdot \lceil \log_2(\mu) \rceil + \sum_{v \in V} \lceil \log_2(\deg_G(v)) \rceil\right)\right)$ *and there are* $O(\eta \cdot (|V| + |E|))$ *auxiliary variables; that is,* $O\left(\eta \cdot \left(|V| \cdot \lceil \log_2(\mu) \rceil + \sum_{v \in V} \lceil \log_2(\deg_G(v)) \rceil + |E|\right)\right)$ *propositional variables in total. The total number of clauses is* $O\left(\eta \cdot \left(|V| \cdot \lceil \log_2(\mu) \rceil + |E| \cdot \lceil \log_2(\mu) \rceil + \sum_{v \in V} \deg_G(v) \cdot \left(\lceil \log_2\left(\deg_G(v)\right) \rceil\right)\right)\right)$.

Proof To obtain the result we need to add up all variables and clauses. The visible variables, that is, the propositional variables representing $\mathcal{A}_v^t$ and $\mathcal{T}_v^t$ add up $(\eta + 1) \cdot |V| \cdot \lceil \log_2(\mu + 1) \rceil$ and $\eta \cdot \sum_{v \in V} \lceil \log_2(2{\cdot}deg_G(v) + 1) \rceil$, respectively. The number of auxiliary variables $a_{v,t}^{\text{zero}}$ is $(\eta{+}1){\cdot}|V|$; the number of $a_{u,v,t}^{=}$ variables is $(\eta{+}1){\cdot}|E|$; and the number of $a_{v,t,c}^{\text{tran}}$ variables is $2{\cdot}\eta{\cdot}\sum_{v \in V} \deg_G(v)$ which is $4{\cdot}\eta{\cdot}|E|$. Hence, the total number of propositional variables is $(\eta + 1) \cdot \left(|V| \cdot \lceil \log_2(\mu + 1) \rceil + |V| + |E|\right) + \eta \cdot \left(\sum_{v \in V} \lceil \log_2(2{\cdot}deg_G(v) + 1) \rceil + 4 \cdot |E|\right)$ which is asymptotically $O\left(\eta \cdot \left(|V| \cdot \lceil \log_2(\mu) \rceil + \sum_{v \in V} \lceil \log_2(\deg_G(v)) \rceil + |E|\right)\right)$.

Let us calculate the number of clauses. A single constraint (13) develops into $\lceil \log_2(\mu + 1) \rceil$ binary clauses; a single constraint (14) develops into $2 \cdot \lceil \log_2(\mu + 1) \rceil$ ternary clauses;

and a single constraint (15) develops into $\lceil \log_2(2 \cdot deg_G(v) + 1) \rceil$ binary clauses and one clause of arity $\lceil \log_2(2 \cdot deg_G(v) + 1) \rceil + 1$. There are as many as $\eta \cdot |V|$ constraints (13); $\eta \cdot |E|$ constraints (14); and $\eta \cdot \sum_{v \in V} \deg_G(v)$ constraints (15) which in total gives $\eta \cdot \left((|V| + 2 \cdot |E|) \cdot \lceil \log_2(\mu + 1) \rceil + \sum_{v \in V} \deg_G(v) \cdot (\lceil \log_2 (2 \cdot deg_G(v) + 1) \rceil + 1) \right)$ clauses (binary, ternary, and one multi-arity).

Constraints (16) count for $\eta \cdot |V|$ binary clauses, constraints (17) together with (18) count for $4 \cdot \eta \cdot \sum_{v \in V} \deg_G(v)$ binary clauses which is clearly dominated by the already calculated number of clauses. Hence, we have $\eta \cdot \left(|V| \cdot \lceil \log_2(\mu) \rceil + |E| \cdot \lceil \log_2(\mu) \rceil + \sum_{v \in V} \deg_G(v) \cdot (\lceil \log_2 (\deg_G(v)) \rceil) \right)$ clauses. □

Proposition 3 *(PATHS AND $F_{INV}(\eta, \Sigma)$ SATISFACTION). Aset $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$ of non-overlapping vertex-disjoint paths in $\mathrm{Exp}_T(G, \eta)$ where π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i), \eta]$ for $i = 1, 2, \ldots, \mu$ exists if and only if $F_{INV}(\eta, \Sigma)$ is satisfiable. Moreover, paths $\pi_1, \pi_2, \ldots, \pi_\mu$ can be unambiguously constructed from a satisfying valuation of $F_{INV}(\eta, \Sigma)$ and vice versa.*

Proof For sake of simplicity, we will show the proposition using finite domain variables rather than bit-vectors. The equivalence between bit vectors and finite domain variables can be seen directly in the translation of the finite domain constraints into equivalent constraints on bit vectors.

Assume a set of vertex-disjoint paths $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$, where $\pi_i$connects $[\alpha_0(a_i), 0]$with $[\alpha_+(a_i), \eta]$. Let $\pi_i = ([u_0, 0], [u_1, 1], [u_2, 2], \ldots, [u_\eta, \eta])$, $u_t \in V$ for $t = 0, 1, \ldots, \eta$ where $u_0 = \alpha_0(a_i)$ and $u_\eta = \alpha_+(a_i)$. We can set $\mathcal{A}^0_{u_0} = i \mathcal{A}^1_{u_1} = i, \ldots, \mathcal{A}^\eta_{u_\eta} = i$. Transition variables are set based on the edges traversed; that is, $\mathcal{T}^0_{u_0} = \sigma_{u_0}(u_1), \mathcal{T}^0_{u_1} = \sigma_{u_1}(u_0) + \deg_G(u_1), \mathcal{T}^1_{u_1} = \sigma_{u_1}(u_2), \mathcal{T}^1_{u_2} = \sigma_{u_2}(u_1) + \deg_G(u_2), \ldots, \mathcal{T}^t_{u_t} = \sigma_{u_{t1+}}(u_{t+1}), \mathcal{T}^t_{u_{t+1}} = \sigma_{u_{t+}}(u_t) + \deg_G(u_{t+1}), \ldots, \mathcal{T}^{\eta-1}_{u_{\eta-1}} = \sigma_{u_{\eta-10}}(u_\eta), \mathcal{T}^{\eta-1}_{u_\eta} = \sigma_{u_{\eta 1}}(u_{\eta-1}) + \deg_G(u_\eta)$. Other paths in Π are processed in the same way. Note that there is no conflict in setting the variables; that is, each variable is set at most once by the assignment. This is because the paths are vertex-disjoint. The variables $\mathcal{A}^t_v$ and $\mathcal{T}^t_v$ which have not yet been set are set to. Checking that constraints (4)–(6) and (7)–(10) are satisfied is not difficult.

Conversely, if there is a satisfying valuation of $F_{INV}(\eta, \Sigma)$, we are able to reconstruct the vertex-disjoint paths from it. Let $\pi_i = ([u_0, 0], [u_1, 1], [u_2, 2], \ldots, [u_\eta, \eta])$ where $u_0 = \alpha_0(a_i)$, and $u_{t+1} = \sigma^{-1}_{u_t}(\mathcal{T}^t_{u_t})$ for every $t = 0, 1, \ldots, \eta - 1$ (it also holds that $u_t = \sigma^{-1}_{u_{t+1}}\left(\mathcal{T}^{t+1}_{u_{t+1}}\right) - \deg_G(u_{t+1})$). The transition state variables $\mathcal{T}^t_v$ that take exactly one value ensure that each vertex in each time step has to determine whether it is connected to a neighbor or accepting a connection from a neighbor (or is connected to itself). This ensures that no selected paths intersect because otherwise a vertex would accept connections from at least two sources or extend connections to at least two neighbors. Both of those cases are forbidden. A value of the $\mathcal{A}^t_v$ variable is only propagated to the next time layer via a connection of the corresponding transition state variable $\mathcal{T}^t_v$. The fact that agents are propagated to their goal positions means that the transition state variables have established paths leading from the agents' initial to their goal positions. □

The following theorem can be directly obtained by applying Proposition 1 and 3 which together justify the approach to solving CPF via a translation into SAT.

Theorem 1 *(*SOLUTION OF Σ AND $F_{INV}(\eta, \Sigma)$ SATISFACTION*). A solution of CPF* $\Sigma = (G, A, \alpha_0, \alpha_+)$ *with* $A = \{a_1, a_2, \ldots, a_\mu\}$ *exists if and only if there is* $\eta \in \mathbb{N}$*, for which the formula* $F_{INV}(\eta, \Sigma)$ *is satisfiable.*

5.2 ALL-DIFFERENT propositional encoding

The choice of a position function rather than its inverse to represent agent configurations in individual time steps has led to another encoding called ALL-DIFFERENT – the name derives from the necessity to explicitly express the requirement that each vertex be occupied by no more than one agent, which is modeled by pair-wise differences between the variables representing the configuration [10]. Again, it is easier to express the encoding using finite domain state variables prior to transforming it to a propositional formula.

Definition 6 (ALL-DIFFERENT ENCODING – $F_{DIFF}(\eta, \Sigma)$). Assume that a CPF $\Sigma = [G, A, \alpha_0, \alpha_+]$ with $G = (V, E)$ is given. An *ALL−DIFFERENT encoding* for CPF Σ consists of finite domain **variables** $\mathcal{L}_a^t \in \{1, \ldots, n\}$ for every $a \in A$ and each time step $t \in \{0, 1, \ldots, \eta\}$ to model the positions of agents in time. The **constraints** are as follows:

$$\mathcal{L}_a^t = j \Rightarrow \mathcal{L}_a^{t+1} = j \vee \bigvee\nolimits_{j \in \{1,\ldots,n\} | \{v_j, v_j\} \in E} \mathcal{L}_a^{t+1} = j \tag{19}$$

for every $a \in A, j \in \{1, 2, \ldots, n\}$ and $t \in \{0, 1, \ldots, \eta - 1\}$ (agent a only moves along the edges or remains in a vertex)

$$\bigwedge\nolimits_{b \in A | b \neq a} \mathcal{L}_a^{t+1} \neq \mathcal{L}_b^t \text{ for every } a \in A \text{ and } t \in \{0, 1, \ldots, \eta - 1\} \tag{20}$$

(the target vertex for agent a must be empty)

$$\text{AllDifferent}(\mathcal{L}_{a_1}^t, \mathcal{L}_{a_2}^t, \ldots, \mathcal{L}_{a_\mu}^t) \text{ for every } t \in \{0, 1, \ldots, \eta\} \tag{21}$$

(each vertex can be occupied by at most one agent in each time step).

The initial and goal configurations will be expressed using the following constraints:

$$\mathcal{L}_a^0 = j \text{ for } a \in A \text{ with } \alpha_0(a) = v_j\} \text{ Initial positions} \tag{22}$$

$$\mathcal{L}_a^\eta = j \text{ for } a \in A \text{ with } \alpha_+(a) = v_j\} \text{ Goal positions} \tag{23}$$

Again, the finite domain state variables $\mathcal{L}_a^t$ are represented as bit vectors (vectors of propositional variables) using a **log** encoding. That is, $\lceil \log_2 |V| \rceil$ propositional variables are introduced for each $\mathcal{L}_a^t$ variable. The resulting formula in CNF will be denoted as $F_{DIFF}(\eta, \Sigma)$.

The AllDifferent$(\mathcal{L}_{a_1}^t, \mathcal{L}_{a_2}^t, \ldots, \mathcal{L}_{a_\mu}^t)$ constraint requires that all variables involved be assigned different values; that is, $\bigwedge_{j,k \in \{1,2,\ldots,\mu\} | j<k} \mathcal{L}_{a_j}^t \neq L_{a_k}^t$. The differences between the finite domain state variables are encoded using the scheme given in [1]. The scheme is used to encode constraints (20) and (21). The inequalities between variables $\mathcal{L}_{a_j}^t$ and $\mathcal{L}_{a_k}^t$

are expressed by the following clauses. Auxiliary variables $d^t_{j,k}$ representing the difference between individual bits are introduced.

$$\mathrm{var}_{\neq}\left(\mathcal{L}^t_{a_j}, \mathcal{L}^t_{a_k}\right) = \bigvee_{\mathbb{i}=0}^{\lceil \log_2 n \rceil - 1} d^t_{j,k}$$

$$\text{where} \quad \bigwedge_{\mathbb{i}=0}^{\lceil \log_2 n \rceil - 1} \left(\neg d^t_{j,k} \vee \neg \mathcal{L}^t_{a_j}[\mathbb{i}] \vee \mathcal{L}^t_{a_k}[\mathbb{i}]\right) \wedge \left(\neg d^t_{j,k} \vee \mathcal{L}^t_{a_j}[\mathbb{i}] \vee \neg \mathcal{L}^t_{a_k}[\mathbb{i}]\right) \tag{24}$$

Conditional equality disjunction (19) is encoded using auxiliary propositional variables to represent the equalities between bit vectors. For each $j \in \{1, 2, \ldots, n\}$ (that is, for each vertex), agent $a \in A$ and time layer $t \in \{0, 1, \ldots, \eta\}$, an auxiliary variable $e^t_{a,j}$ which stands for equality $\mathcal{L}^t_a = j$ is introduced. The link between auxiliary variables $e^t_{a,j}$ and actual equalities is established by the following constraint:

$$e^t_{a,j} \Leftrightarrow \mathrm{con}_{=}\left(\mathcal{L}^t_a, j\right) \tag{25}$$

Then, moving along the edges – constraints (19) – can be encoded as a single clause using auxiliary variables:

$$e^t_{a,j} \Rightarrow \neg e^{t+1}_{a,j} \vee \bigvee_{j \in \{1,\ldots,n\} | \{v_j, v_j\} \in E} e^{t+1}_{a,j} \tag{26}$$

Again, the space consumption of the ALL-DIFFERENT encoding will be calculated for regular time layers only.

Proposition 4 *(*ALL-DIFFERENT ENCODING SIZE*). The number of visible propositional variables in $F_{DIFF}(\eta, \Sigma)$ is $O(\eta \cdot \mu \cdot \lceil \log_2 |V| \rceil)$ and there are $O(\eta \cdot \mu \cdot |V|)$ auxiliary variables; that is, the number of variables is $O(\eta \cdot \mu \cdot |V|)$. The number of clauses is $O\left(\eta \cdot \lceil \log_2 |V| \rceil \cdot \left(\binom{\mu}{2} + \mu \cdot |V|\right)\right)$.*

Proof Each variable $\mathcal{L}^t_a$ is represented by $\lceil \log_2 |V| \rceil$ propositional variables and the number of $\mathcal{L}^t_a$ variables is $(\eta + 1) \cdot \mu$. For each $\mathcal{L}^t_a$ variable and its value, an auxiliary variable is introduced. As $\mathcal{L}^t_a$ can take $|V|$ values, we conclude that there are $(\eta + 1) \cdot \mu \cdot (\log_2 |V| + |V|)$ variables, which is $O(\eta \cdot \mu \cdot |V|)$.

A single time layer requires as many as $\binom{\mu}{2}$ inequalities between all pairs of the $\mathcal{L}^t_a$ variables corresponding to the distinct agents in order to express the AllDifferent constraint given in (21). Each inequality is modeled by $2 \cdot \lceil \log_2 |V| \rceil$ ternary clauses plus one clause of arity $\lceil \log_2 |V| \rceil$. This is $(\eta + 1) \cdot \binom{\mu}{2} \cdot (2 \cdot \lceil \log_2 |V| \rceil + 1)$ clauses in total.

Next, we need as many as $\binom{\mu}{2}$ inequalities between the $\mathcal{L}^t_a$ variables from two successive time layers (constraint (20). This results in the addition of the same number of $(\eta + 1) \cdot \binom{\mu}{2} \cdot (2 \cdot \lceil \log_2 |V| \rceil + 1)$ clauses.

The links between auxiliary variables $e^t_{a,j}$ and the actual equalities (25) they represent require $\lceil \log_2 |V| \rceil$ binary clauses plus one clause of arity $\lceil \log_2 |V| \rceil + 1$, which is $(\eta + 1) \cdot \mu \cdot |V| \cdot (\lceil \log_2 |V| \rceil + 1)$ in total.

Finally, the constraints requiring that agents only move along the edges (26) contribute μ clauses of arity $\deg_G(v_j) + 2$ to each vertex v_j in $\mathrm{Exp}_{\mathrm{T}}(G, \eta)$ in any given time layer except the last one which is $\eta \cdot \mu \cdot |V|$ clauses in total.

In all, we have $(\eta+1)\cdot\left(\left(\frac{\mu}{2}\right)\cdot\left(2\cdot\left\lceil\log_2|V|\right\rceil+1\right)+\mu\cdot|V|\cdot(\left\lceil\log_2|V|\right\rceil+1)\right)+\eta\cdot\mu\cdot|V|$ clauses in a $F_{DIFF}(\eta,\Sigma)$ encoding, which is $O\left(\eta\cdot\left\lceil\log_2|V|\right\rceil\cdot\left(\left(\frac{\mu}{2}\right)+\mu\cdot|V|\right)\right)$. □

Proposition 5 *(*PATHS AND $F_{DIFF}(\eta,\Sigma)$ SATISFACTION*). Aset* $\Pi=\{\pi_1,\pi_2,\ldots,\pi_\mu\}$ *of non-overlapping vertex-disjoint paths in* $\mathrm{Exp}_\mathrm{T}(G,\eta)$ *where* π_i *connects* $[\alpha_0(a_i),0]$ *with* $[\alpha_+(a_i),\eta]$ *for* $i=1,2,\ldots,\mu$ *exists if and only if* $F_{DIFF}(\eta,\Sigma)$ *is satisfiable. Moreover, paths* $\pi_1,\pi_2,\ldots,\pi_\mu$ *can be unambiguously constructed from a satisfying valuation of* $F_{DIFF}(\eta,\Sigma)$ *and vice versa.*

Proof For simplicity's sake, we will operate at the level of finite domain state variables. Assume that non-overlapping vertex-disjoint paths $\pi_1,\pi_2,\ldots,\pi_\mu$ exist in $\mathrm{Exp}_\mathrm{T}(G,\eta)$. A satisfying valuation of $F_{DIFF}(\eta,\Sigma)$ can be directly constructed from these paths. Let $\pi_i=([u_0,0],\ [u_1,1],[u_2,2]\ \ldots,[u_\eta,\eta])$, $u_t\in V$ for $t=0,1,\ldots,\eta$ where $u_0=\alpha_0(a_i)$ and $u_\eta=\alpha_+(a_i)$. Then, the finite domain state variables will be set as follows: $\mathcal{L}^0_{a_i}=u_0$ $\mathcal{L}^1_{a_i}=u_1,\ldots,\mathcal{L}^\eta_{a_i}=u_\eta$ for every $i=1,2,\ldots,\mu$. The assumptions that paths were vertex-disjoint and were non-overlapping ensure that constraints (20) and (21), respectively, are satisfied. Any consecutive vertices within paths are connected by the directed edges that correspond to the edges in G. Hence, constraints (19) are satisfied.

Assume, on the other hand, that we have a satisfying valuation of $F_{DIFF}(\eta,\Sigma)$. We can immediately set $\pi_i=([\mathcal{L}^0_{a_i},0],\ [\mathcal{L}^1_{a_i},1],\ [\mathcal{L}^2_{a_i},2],\ldots,\ [\mathcal{L}^\eta_{a_i},\eta])$ for every $i=1,2,\ldots,\mu$. The satisfaction of constraints (19) ensures that any constructed sequences of vertices are paths in $\mathrm{Exp}_\mathrm{T}(G,\eta)$. In addition, paths π_i are non-overlapping and vertex-disjoint due to constraints (20) and (21), respectively. □

The approach to solving CPF by way of satisfying $F_{DIFF}(\eta,\Sigma)$ is justified by the following theorem which combines Proposition 1 and 5. From $F_{DIFF}(\eta,\Sigma)$, non-overlapping vertex-disjoint paths corresponding to the solution to the CPF problem can be obtained.

Theorem 2 *(*SOLUTION OF Σ AND $F_{DIFF}(\eta,\Sigma)$ SATISFACTION*). A solution of CPF* $\Sigma=(G,A,\alpha_0,\alpha_+)$ *exists if and only if there is*$\eta\in\mathbb{N}$*, for which the formula* $F_{DIFF}(\eta,\Sigma)$ *satisfiable.*

5.3 MATCHING propositional encoding

We have observed that vertex-disjoint non-overlapping paths in a time-expanded graph resemble *single-commodity flow* [1] in a time-expanded graphs, in which vertices and edges are assigned unit capacities. Intuition tells us that the edges included in paths should be saturated by one unit of flow. Such a setting would convey the commodity from all initial to all goal vertices (Figs. 4 and 5).

However, a correspondence between paths of the required properties and flow is present in one direction only. The flow reflects well the requirement that paths should be vertex-disjoint, but it cannot simulate the non-overlapping between the paths or the correct linkage between the initial and goal vertices for the same agent (the flow can, however, connect the initial and goal vertices for two distinct agents because it treats them as anonymous commodity units).

The design of the MATCHING encoding will be based on intuitive insights concerning single commodity flows. It will be divided into two parts – the first part called the FLOW part will check for the existence of a flow that generates non-overlapping vertex-disjoint

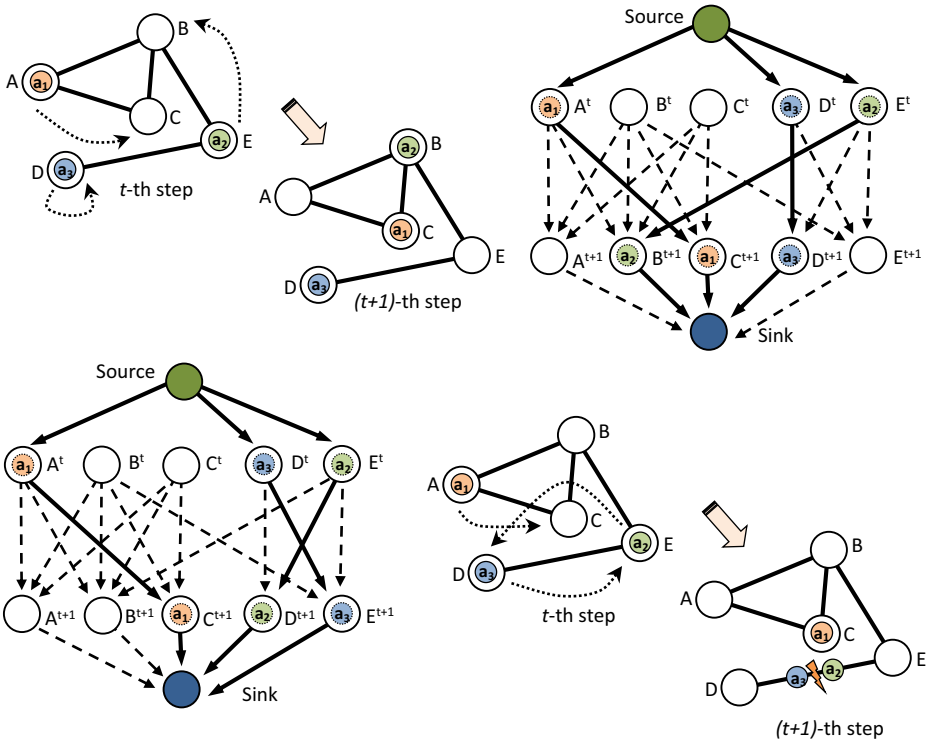


Fig. 4 *The correspondence between agent movement and flow in a bipartite graph.* The movement between time steps t and $t+1$ and the corresponding flow is shown in a bipartite graph consisting of t-th and $(t+1)$-th time step where vertices and edges are assigned unit capacities. A valid movement induces a flow, in which saturated edges are non-overlapping; that is, $\{A, E\} \cap \{B, C\} = \emptyset$ (*upper part*). On the other hand, a standard network flow does not necessarily produce non-overlapping edges, which may result in invalid agent movement (*lower part*)

paths. This part can be understood as an encoding of a CPF problem with relaxed constraints, in which the agents are anonymous. When dealing with anonymous agents, we are looking at moving a group of agents to a set of goal vertices, but are not interested in having control over which agent arrives in which goal vertex (the paths generated may connect the initial and goal vertices of different agents). As we will see later, the propositional encoding of this part will be based on a **direct** representation.

The second part, called the MAPPING part, maps different agents to paths marked out by the flow. This part of the encoding eventually strengthens requirements imposed by the MATCHING part.

The proposed design is based on the assumption that a SAT solver should be able to quickly check for the existence of non-overlapping flows, which is a prerequisite of the existence of a true solution.

Definition 7 (MATCHING ENCODING – $F^{FLOW}_{MATCH}(\eta, \Sigma)$). The FLOW part of the MATCHING encoding of CPF $\Sigma = [G, A, \alpha_0, \alpha_+]$ with $G = (V, E)$ consists of a propositional variable for each vertex and edge in the time-expanded graph used to model network flows. In other words, a propositional variable $\mathcal{M}^t_v$ is introduced for every $t = 0, 1, \ldots, \eta$ and

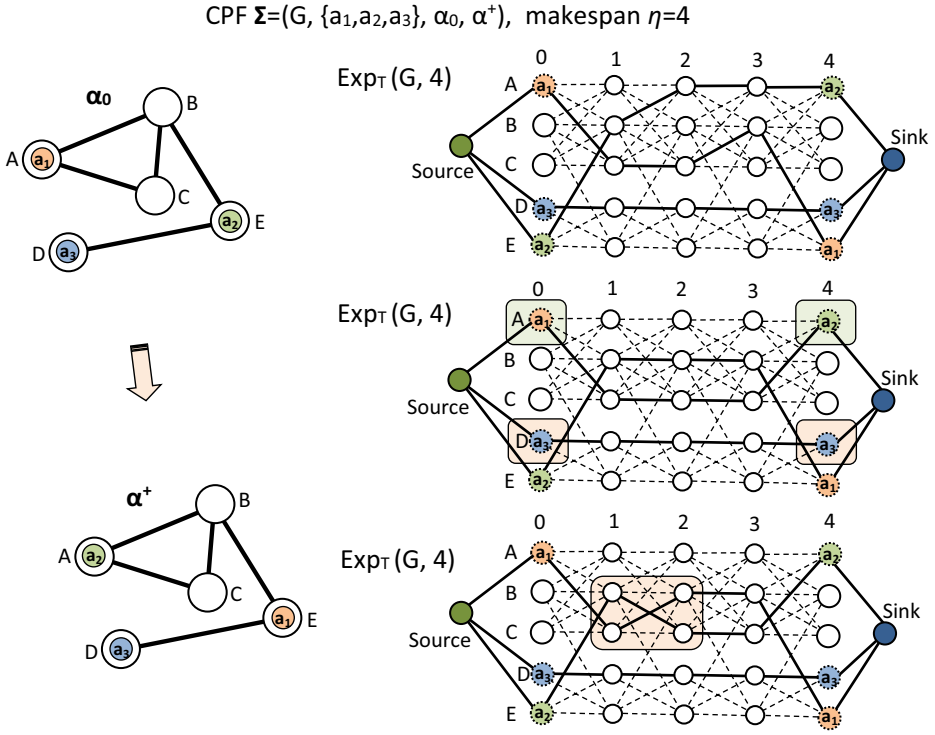


Fig. 5 *Non-overlapping vertex-disjoint paths in a time-expanded graph and correspondence with single commodity flow.* The edges and vertices in a time-expanded graph are assumed to have unit capacities. A correct flow can be reconstructed from vertex-disjoint non-overlapping paths (*upper right part*). On the other hand, a flow does not necessarily correspond to paths of the required properties (the *middle right part* shows the connection between the initial and goal vertices of different agents and the lower right part shows the overlapping paths between time steps 1 and 2)

$v \in V$ and propositional variables $\mathcal{E}^t_{u,v}$ and $\mathcal{E}^t_u$ are introduced for every $t = 0, 1, \ldots, \eta$ and $\{u, v\} \in E$ and $u \in V$, respectively. The constraints ensure that variables set to $TRUE$ form a non-overlapping flow:

$$\begin{aligned} &\mathcal{E}^t_{u,v} \Rightarrow \mathcal{M}^t_u \wedge \mathcal{M}^{t+1}_v \quad \text{for every} \{u, v\} \in E \\ &\qquad\qquad\qquad\quad \text{and } t \in \{0, 1, \ldots, \eta - 1\}, \\ &\mathcal{E}^t_u \Rightarrow \mathcal{M}^t_u \wedge \mathcal{M}^{t+1}_u \quad \text{for every } u \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\} \end{aligned} \tag{27}$$

(if an edge is included in a flow, then its endpoints are included as well)

$$\begin{aligned} &\mathcal{E}^t_u + \textstyle\sum_{v|\{u,v\}\in E} \mathcal{E}^t_{u,v} \leq 1 \quad \text{for every } u \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \\ &\mathcal{E}^t_v + \textstyle\sum_{u|\{u,v\}\in E} \mathcal{E}^t_{u,v} \leq 1 \quad \text{for every } v \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \end{aligned} \tag{28}$$

(there can be at most one incoming and one outgoing edge in a flow)

$$\begin{aligned} &\mathcal{M}^t_u \Rightarrow E^t_u \vee \textstyle\bigvee_{v|\{u,v\}\in E} \mathcal{E}^t_{u,v} \quad \text{for every } u \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \\ &\mathcal{M}^{t+1}_v \Rightarrow E^t_u \vee \textstyle\bigvee_{u|\{u,v\}\in E} \mathcal{E}^t_{u,v} \quad \text{for every } v \in V \text{ and } t \in \{0, 1, \ldots, \eta - 1\}, \end{aligned} \tag{29}$$

(if a vertex is included in a flow, then at least one outgoing and one incoming edge must be included as well)

$$\mathcal{E}^t_{u,v} \Rightarrow M^t_v \quad \begin{array}{l}\text{for every } \{u, v\} \in E \\ \text{and } t \in \{0, 1, \ldots, \eta - 1\},\end{array} \tag{30}$$

(the source and target vertices of regular moves must be disjoint).

The second part of the encoding, in which the distinct agents manifest themselves individually, is introduced in the following definition.

Definition 8 (MATCHING ENCODING – $F^{MAP}_{MATCH}(\eta, \Sigma)$). The *MAPPINGpart of the* MATCHING encoding of a given CPF $\Sigma = [G, A, \alpha_0, \alpha_+]$ with $G = (V, E)$ consists of a finite domain variable $\mathcal{A}^t_v \in \{0, 1, \ldots, \mu\}$ for each vertex $v \in V$ and layer $t = 0, 1, \ldots, \eta$ to model the appearance of an agent in a vertex. The constraints connect the MAPPING part to the FLOW part, so that the actual distinct agents follow the paths indicated by the flow:

$$\mathcal{E}^t_{u,v} \Rightarrow \mathcal{A}^t_u = \mathcal{A}^{t+1}_v \quad \begin{array}{l}\text{for every } \{u, v\} \in E \\ \text{and } t \in \{0, 1, \ldots, \eta - 1\},\end{array} \tag{31}$$

(if an edge is saturated by the flow, then the same agent appears at its both ends)

$$\mathcal{A}^t_u \neq 0 \Rightarrow \mathcal{M}^t_u \quad \begin{array}{l}\text{for every } u \in V \\ \text{and } t \in \{0, 1, \ldots, \eta\}\end{array} \tag{32}$$

(if an agent appears in a vertex, then the vertex is saturated by the flow)

As in the previous encodings, $\mathcal{A}^t_v$ variables with $\mu + 1$ states are represented by $\lceil \log_2(\mu + 1) \rceil$ propositional variables using a binary encoding. The initial and goal configurations will be expressed using the following constraints:

$$\left.\begin{array}{ll} \mathcal{A}^0_u = i \wedge \mathcal{M}^0_u & \text{for } u \in V \text{ if there is } i \in \{1,2,\ldots,\mu\} \\ & \text{such that } \alpha_0(a_i) = u \\ \mathcal{A}^0_u = 0 \wedge \neg\mathcal{M}^0_u & \text{for } u \in V \text{ if } (\forall a \in A)\alpha_0(a) \neq u \end{array}\right\} \text{Initial positions} \tag{33, 34}$$

$$\left.\begin{array}{ll} \mathcal{A}^\eta_u = i \wedge \mathcal{M}^\eta_u & \text{for } u \in V \text{ if there is } i \in \{1,2,\ldots,\mu\} \\ & \text{such that } \alpha_+(a_i) = u \\ \mathcal{A}^\eta_u = 0 \wedge \neg\mathcal{M}^\eta_u & \text{for } u \in V \text{ if } (\forall a \in A)\alpha_+(a) \neq u \end{array}\right\} \text{Goal positions} \tag{35, 36}$$

The resulting formula denoted as $F_{MATCH}(\eta, \Sigma)$ in CNF is a conjunction of the FLOW part, MAPPING part, and the constraints on the initial/goal position. To obtain CNF, the *at-most-one constraints* (28) need to be rewritten as clauses. That is, for example $\mathcal{E}^t_u + \sum_{v|\{u,v\}\in E} \mathcal{E}^t_{u,v} \leq 1$ can be rewritten as a conjunction of clauses that forbids all pairs of the variables involved to be set to $TRUE$:

$$\begin{array}{c} \bigwedge_{v,w|\{u,v\}\in E \wedge \{u,w\}\in E \wedge v\neq w} \neg\mathcal{E}^t_{u,v} \vee \neg\mathcal{E}^t_{u,w} \\ \bigwedge_{v|\{u,v\}\in E} \neg\mathcal{E}^t_{u,v} \vee \neg\mathcal{E}^t_u \end{array} \tag{37}$$

Note that more sophisticated encodings of the *at-most-one* constraint that are more scalable with the number of variables constrained are available [35, 46]. However, this basic encoding is very powerful when the number of variables constrained is small because it

greatly supports unit propagation thanks to binary clauses. Moreover, the number of variables here depends on $\deg_G(v)$, which is low in typical locally interconnected (planar) graphs used in CPF applications.

The log encoded variables $\mathcal{A}_v^t$ are not involved in any complex relation – only the conditional equality between these variables is introduced, while all other modeling issues concerning static and dynamic CPF constraints occur in the FLOW part of the encoding.

The conditional equality between $\mathcal{A}_u^t$ and $\mathcal{A}_v^{t+1}$ in (31) can be expressed as follows:

$$\mathcal{E}_{u,v}^t \Rightarrow \mathrm{var}_= \left(\mathcal{A}_u^t, \mathcal{A}_v^{t+1}\right) \tag{38}$$

Constraint (32) can be rewritten as follows:

$$\bigwedge_{\mathbb{i}=0}^{\lceil \log_2(\mu+1)\rceil - 1} \neg \mathcal{A}_u^t[\mathbb{i}] \vee \mathcal{M}_u^t \tag{39}$$

Proposition 6 *(*MATCHING ENCODING SIZE*). The number of propositional variables in $F_{MATCH}(\eta, \Sigma)$ is $\mathcal{O}\left(\eta \cdot \left(|E| + |V| \cdot \lceil \log_2(\mu)\rceil\right)\right)$. The number of clauses is $\mathcal{O}\left(\eta \cdot \left((|V| + |E|) \cdot \lceil \log_2(\mu)\rceil + \sum_{v \in V}\left(\frac{\deg_G(v)}{2}\right)\right)\right)$.*

Proof The FLOW part of the MATCHING encoding has a propositional variable $\mathcal{M}_v^t$ for each vertex $v \in V$ and time layer $t \in \{0, 1, \ldots, \eta\}$ and $\mathcal{E}_{u,v}^t$ for each edge $\{u, v\} \in E$ and time layer, which makes $(\eta + 1) \cdot (|V| + |E|)$ propositional variables in total. Further, we have a vector of $\lceil \log_2(\mu + 1)\rceil$ propositional variables representing $\mathcal{A}_v^t$ for each vertex and time layer in the MAPPING part. This makes another $(\eta+1) \cdot |V| \cdot \lceil \log_2(\mu + 1)\rceil$ variables in total. In all, there are $(\eta + 1) \cdot \left(|E| + |V| + |V| \cdot \lceil \log_2(\mu + 1)\rceil\right)$ variables, which is $O\left(\eta \cdot \left(|E| + |V| \cdot \lceil \log_2(\mu)\rceil\right)\right)$.

Constraints (27) develop into $\eta \cdot (|V| + |E|)$ ternary clauses. Constraints (28) develop into $2 \cdot \eta \cdot \sum_{v \in V}\left(\frac{\deg_G(v)+1}{2}\right)$ binary clauses as indicated by (31). Constraints (29) introduce two clauses of a length of $\deg_G(v) + 1$ for each vertex and time layer; that is, $2 \cdot \eta \cdot |V|$ clauses are added. Finally, constraints (30) add a binary clause for each vertex and time layer, which is, again, dominated by the previous expressions. The conditional equality between the two bit vectors in (28) develops into $2 \cdot \lceil \log_2(\mu + 1)\rceil$ ternary clauses, while the equality is introduced for each edge and time layer; that is, $2 \cdot \eta \cdot |E| \cdot \lceil \log_2(\mu + 1)\rceil$ ternary clauses are added. One can observe that the expression (32) represents $(\eta + 1) \cdot |E| \cdot \lceil \log_2(\mu + 1)\rceil$ binary clauses. In all, there are $\eta \cdot (3 \cdot |V| + |E| + 2 \cdot \sum_{v \in V}\left(\frac{\deg_G(v)+1}{2}\right) + 2 \cdot |E| \cdot \lceil \log_2(\mu + 1)\rceil) + (\eta + 1) \cdot |E| \cdot \lceil \log_2(\mu + 1)\rceil$ clauses, which is $O\left(\eta \cdot \left((|V| + |E|) \cdot \lceil \log_2(\mu)\rceil + \sum_{v \in V}\left(\frac{\deg_G(v)}{2}\right)\right)\right)$. □

Proposition 7 *(*PATHS AND $F_{MATCH}(\eta, \Sigma)$ SATISFACTION*). A set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$ of non-overlapping vertex-disjoint paths in $\mathrm{Exp}_\mathrm{T}(G, \eta)$ where π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i), \eta]$ for $i = 1, 2, \ldots, \mu$ exists if and only if $F_{MATCH}(\eta, \Sigma)$ is satisfiable. Moreover, the paths $\pi_1, \pi_2, \ldots, \pi_\mu$ can be unambiguously constructed from a satisfying valuation of $F_{MATCH}(\eta, \Sigma)$ and vice versa.*

Proof We will operate at the level of finite domain state variables $\mathcal{A}_v^t$ as opposed to bit vectors to simplify the proof.

Assume that there are non-overlapping vertex-disjoint paths $\pi_1, \pi_2, \ldots, \pi_\mu$ where π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i), \eta]$ in $\mathrm{Exp}_\mathrm{T}(G, \eta)$. Let $\pi_i = ([u_0, 0], [u_1, 1], [u_2, 2], \ldots, [u_\eta, \eta])$, with $u_t \in V$ for $t = 0, 1, \ldots, \eta$ where $u_0 = \alpha_0(a_i)$ and $u_\eta = \alpha_+(a_i)$. A satisfying valuation of $F_{MATCH}(\eta, \Sigma)$ can be constructed by setting $\mathcal{A}^0_{u_0} = i$, $\mathcal{A}^1_{u_1} = i, \ldots, \mathcal{A}^\eta_{u_\eta} = i$. Next, the variables $\mathcal{M}^0_{u_0}, \mathcal{M}^1_{u_1}, \ldots, \mathcal{M}^\eta_{u_\eta}$ representing flow are set to $TRUE$ and $\mathcal{E}^0_{u_0,u_1}, \mathcal{E}^1_{u_1,u_2}, \ldots, \mathcal{E}^{\eta-1}_{u_{\eta-1},u_\eta}$ are set to $TRUE$ as well (the convention that $\mathcal{E}^t_{u_l,u_{l+1}} \equiv \mathcal{E}^t_{u_l}$ if $u_t = u_{t+1}$ is used here). Now, all constraints are satisfied.

The connection between the FLOW and MAPPING parts is satisfied by the construction, so we just need to check the constraints in the FLOW part of the encoding. The propagation of the flow from the edges to vertices is also ensured by the construction.

The fact that the original paths are vertex-disjoint ensures the validity of constraints (27) and (28), which together ensure the inclusion of exactly one incoming and one outgoing edge by setting the $\mathcal{E}^t_{u,v}$ variables for each vertex saturated by the flow, as indicated by the $\mathcal{M}^t_v$ variable set to $TRUE$.

Finally, the fact that the paths are non-overlapping directly translates into the satisfaction of constraints (30). The constraints on the initial and goal positions are trivially satisfied. All in all, $F_{MATCH}(\eta, \Sigma)$ is satisfied by the constructed valuation of its variables.

Let us now test the opposite implication. Assume that $F_{MATCH}(\eta, \Sigma)$ is satisfiable. Let $\pi_i = ([u_0, 0], [u_1, 1], [u_2, 2], \ldots, [u_\eta, \eta])$ such that $u_0 = \alpha_0(a_i)$ and $\mathcal{E}^l_{u_l,u_{l+1}}$ is $TRUE$ for each $t = 0, 1, \ldots, \eta - 1$. This is feasible thanks to constraints (27) - (29) which propagate the flow from the initial positions in the first time layer to the final layer. We will verify that the paths constructed this way have the required properties – they are vertex-disjoint, non-overlapping and connect the initial and goal positions of the agents.

The FLOW part of the encoding ensures that the constructed paths are vertex-disjoint and non-overlapping. We just need to add the non-overlapping property to the already checked flow propagation. The non-overlapping property is indeed established by constraints (30).

However, the FLOW part does not ensure that $u_\eta = \alpha_+(a_i)$; the satisfaction of the FLOW part alone may result in a path connecting the initial and goal positions of two distinct agents. This is corrected by the constraints included in the MAPPING part of the encoding. These constraints propagate agent a_i along the edges $\{u_t u_{t+1}\}$ and eventually make it show up in u_η and the goal constraints (35) and (36) ensure that agent a_i arrives in the correct vertex. □

By combining Proposition 7, which has just been proven, with a correspondence between non-overlapping vertex-disjoint paths in $\mathrm{Exp}_\mathrm{T}(G, \eta)$ we can immediately obtain the following theorem.

Theorem 3 SOLUTION OF Σ AND $F_{MATCH}(\eta, \Sigma)$ SATISFACTION*). A solution of a CPF* $\Sigma = (G, A, \alpha_0, \alpha_+)$ *exists if and only if there is* $\eta \in \mathbb{N}$*, for which the formula* $F_{MATCH}(\eta, \Sigma)$ *is satisfiable.*

5.4 DIRECT/SIMPLIFIED propositional encoding

The previous CPF encodings used a **log** representation of an agent's presence in a vertex in one form or another. We will now introduce a CPF encoding based purely on the **direct** representation of state variables. We will use a single propositional variable to encode the presence of an agent in a vertex in a specific time-step. The resulting CPF encoding will be called DIRECT [58].

Definition 9 (DIRECT ENCODING – $F_{DIR}(\eta, \Sigma)$). Assume a CPF $\Sigma = [G, A, \alpha_0, \alpha_+]$ with $G = (V, E)$. A *DIRECTencoding* for CPF Σ consists of propositional variables $\mathcal{X}^t_{a,v}$ for every $a \in A$, $v \in V$, and time step $t \in \{0, 1, \ldots, \eta\}$ to model the appearance of agents in vertices over time. $\mathcal{X}^t_{a,v}$ is assigned $TRUE$ if and only if agent a shows up in vertex v in time step t. The following constrains ensure the satisfaction of *static* and *dynamic* constraints:

$$\begin{array}{l} \bigwedge_{u,v \in V, u \neq v} \neg \mathcal{X}^t_{a,u} \vee \neg \mathcal{X}^t_{a,v} \\ \bigvee_{v \in V} \mathcal{X}^t_{a,v} \end{array} \quad \begin{array}{l} \text{for every } t \in \{0, 1, \ldots, \eta\} \\ \text{and } a \in A \end{array} \tag{40}$$

(an agent is placed in exactly one vertex in each time step)

$$\bigwedge_{a,b \in A, a \neq b} \neg \mathcal{X}^t_{a,v} \vee \neg \mathcal{X}^t_{b,v} \quad \begin{array}{l} \text{for every } t \in \{0, 1, \ldots, \eta\} \\ \text{and } v \in V \end{array} \tag{41}$$

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

$$\begin{array}{l} \mathcal{X}^t_{a,v} \Rightarrow \mathcal{X}^{t+1}_{a,v} \vee \bigvee_{u \in V, \{v,u\} \in E} \mathcal{X}^{t+1}_{a,u} \\ \mathcal{X}^{t+1}_{a,v} \Rightarrow \mathcal{X}^t_{a,v} \vee \bigvee_{u \in V, \{v,u\} \in E} \mathcal{X}^t_{a,u} \end{array} \quad \begin{array}{l} \text{for every } t \in \{0, 1, \ldots, \eta - 1\}, \\ v \in V, \text{ and } a \in A \end{array} \tag{42}$$

(an agent moves to a neighboring position or does not move)

$$\mathcal{X}^t_{a,v} \wedge X^{t+1}_{a,u} \Rightarrow \bigwedge_{b \in A} \neg \mathcal{X}^t_{b,u} \wedge \bigwedge_{b \in A} \neg \mathcal{X}^{t+1}_{b,v} \quad \begin{array}{l} \text{for every } t \in \{0, 1, \ldots, \eta - 1\}, \{u, v\} \in V \\ \text{such that } \{u, v\} \in E \text{ and } a \in A \end{array} \tag{43}$$

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

The initial and goal configurations will be expressed using the following constraints:

$$\left.\begin{array}{ll} \mathcal{X}^0_{a,v} & \text{for } v \in V \text{ if there is } a \in A \\ & \text{such that } \alpha_0(a) = v \\ \neg \mathcal{X}^0_{a,v} & \text{otherwise} \end{array}\right\} \text{Initial positions} \quad \begin{array}{l} (44) \\ (45) \end{array}$$

$$\left.\begin{array}{ll} \mathcal{X}^\eta_{a,v} & \text{for } v \in V \text{ if there is } a \in A \\ & \text{such that } \alpha_+(a) = v \\ \neg \mathcal{X}^\eta_{a,v} & \text{otherwise} \end{array}\right\} \text{Goal positions} \quad \begin{array}{l} (46) \\ (47) \end{array}$$

The resulting DIRECT encoding formula in CNF will be denoted as $F_{DIR}(\eta, \Sigma)$. We can observe the repetitive evacuation of the target and source vertices before and after a move (constraint (43), as the right hand-side of the implication is independent of agent a. Therefore, the encoding can be enhanced by introducing auxiliary variables $\mathcal{E}^t_u$ for each vertex $u \in V$ and time step $t \in \{0, 1, \ldots, \eta\}$. These represent the evacuation of vertex u in time step t. The semantics of the $\mathcal{E}^t_u$ variables is represented by the following constraint:

$$\mathcal{E}^t_u \Rightarrow \bigwedge_{a \in A} \neg \mathcal{X}^t_{a,u} \quad \text{for every } t \in \{0, 1, \ldots, \eta\} \text{ and } u \in V \tag{48}$$

(in an empty vertex no agent can appear at a given time)

The repetitive part in constraint (43) can now be replaced as follows by a version with auxiliary variables:

$$\mathcal{X}^t_{a,v} \wedge X^{t+1}_{a,u} \Rightarrow \mathcal{E}^t_u \wedge \mathcal{E}^{t+1}_v \quad \begin{array}{l}\text{for every } t \in \{0, 1, \ldots, \eta - 1\}, u, v \in V \\ \text{such that } \{u, v\} \in E \text{ and } a \in A.\end{array} \tag{49}$$

The resulting encoding with auxiliary variables is called SIMPLIFIED in [59] and the corresponding CNF formula will be denoted as $F_{SIM}(\eta, \Sigma)$.

Proposition 8 *(*DIRECT/SIMPLIFIED ENCODING SIZE*). The number of propositional variables in $F_{DIR}(\eta, \Sigma)$ is $O(\eta \cdot \mu \cdot |V|)$. The number of clauses is $O\left(\eta \cdot \left(\mu \cdot |V|^2 + \mu^2 \cdot |V| + \mu^2 \cdot |E|\right)\right)$. $F_{SIM}(\eta, \Sigma)$ contains additional $O(\eta \cdot |V|)$ propositional variables, while the total number of clauses is only $O\left(\eta \cdot \left(\mu \cdot |V|^2 + \mu^2 \cdot |V| + \mu \cdot |E|\right)\right)$.*

Proof By adding up the index scopes, we establish that there are exactly $(\eta + 1) \cdot \mu \cdot |V|$ variables $\mathcal{X}^t_{a,v}$ and $\eta \cdot |V|$ $\mathcal{E}^t_u$ variables

Each time layer and agent adds $\binom{|V|}{2}$ binary clauses and one $|V|$-ary clause within constraints (40). Thus, we have altogether $(\eta+1) \cdot \mu \cdot \binom{|V|}{2}$ binary clauses and $(\eta+1) \cdot \mu |V|$-ary clauses resulting from rewriting constraints (40) as clauses. A similar calculation can be done for constraints (41); we have $\binom{\mu}{2}$ binary clauses for each time layer and vertex; that is, $(\eta + 1) \cdot |V| \cdot \binom{\mu}{2}$ binary clauses in total.

There are two $(deg_G(v) + 2)$-ary clauses for each vertex v in each time step except the last one and for each agent resulting from constraints (42), which altogether means $\eta \cdot \mu$ $(deg_G(v) + 2)$-ary clauses for each vertex $v \in V$. That is, we have $\eta \cdot \mu \cdot |V|$ clauses in total.

Each implication in constraint (43) develops into $2 \cdot \mu$ ternary clauses. There are $|E|$ such groups of clauses for every agent and time step except the last one. Thus, $2 \cdot \eta \cdot \mu^2 \cdot |E|$ ternary clauses in total are needed to express constraints (43).

In all, the total number of clauses in $F_{DIR}(\eta, \Sigma)$ is $(\eta + 1) \cdot \left(\mu \cdot \left(\binom{|V|}{2} + 1\right) + |V| \cdot \binom{\mu}{2}\right) + 2 \cdot \eta \cdot \mu \cdot (|V| + \mu \cdot |E|)$, which is $O\left(\eta \cdot \left(\mu \cdot |V|^2 + \mu^2 \cdot |V| + \mu^2 \cdot |E|\right)\right)$.

Constraints (49) develop into fewer clauses compared to the original constraints (43) which they replace because of the shorter right-hand side of the implication in $F_{SIM}(\eta, \Sigma)$. In particular, each implication from (49) develops into exactly 2 ternary clauses, which gives $2 \cdot \eta \cdot \mu \cdot |E|$ ternary clauses in total.

The connection of the auxiliary variables $\mathcal{E}^t_u$ to their meaning requires μ binary clauses for each implication resulting from constraint (48). There are as many as $(\eta + 1) \cdot |V|$ such connections, which results in $(\eta + 1) \cdot \mu \cdot |V|$ in total. Hence, the total number of clauses in $F_{SIM}(\eta, \Sigma)$ is $(\eta + 1) \cdot \left(\mu \cdot \left(\binom{|V|}{2} + 1\right) + |V| \cdot \binom{\mu}{2}\right) + \eta \cdot \mu \cdot (|V| + 2 \cdot |E|)$, which is $O\left(\eta \cdot \left(\mu \cdot |V|^2 + \mu^2 \cdot |V| + \mu \cdot |E|\right)\right)$. □

Proposition 9 *(*PATHS AND $F_{DIR}(\eta, \Sigma)/F_{SIM}(\eta, \Sigma)$ SATISFACTION*). A set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_\mu\}$ of non-overlapping vertex-disjoint paths in $\mathrm{Exp_T}(G, \eta)$ where π_i connects $[\alpha_0(a_i), 0]$ to $[\alpha_+(a_i), \eta]$ for $i = 1, 2, \ldots, \mu$ exists if and only if $F_{DIR}(\eta, \Sigma)$ is satisfiable. Moreover, paths $\pi_1, \pi_2, \ldots, \pi_\mu$ can be unambiguously constructed from a satisfying valuation of $F_{DIR}(\eta, \Sigma)$ and vice versa. The same holds for $F_{SIM}(\eta, \Sigma)$.*

Proof Assume that there are non-overlapping vertex-disjoint paths $\pi_1, \pi_2, \ldots, \pi_\mu$ where π_i connects $[\alpha_0(a_i), 0]$ with $[\alpha_+(a_i), \eta]$ in $\text{Exp}_\text{T}(G, \eta)$. We will construct a satisfying valuation of $F_{DIR}(\eta, \Sigma)$ from $\pi_1, \pi_2, \ldots, \pi_\mu$.

Let $\pi_i = ([u_0, 0],[u_1, 1], [u_2, 2],\ldots,[u_\eta, \eta])$, with $u_t \in V$ for $t = 0, 1, \ldots, \eta$ where $u_0 = \alpha_0(a_i)$ and $u_\eta = \alpha_+(a_i)$, then the variables $\mathcal{X}^0_{a_i,u_0}$, $\mathcal{X}^1_{a_i,u_1}$, ..., $\mathcal{X}^\eta_{a_i,u_\eta}$ will be set to $TRUE$. The $\mathcal{X}^t_{a,v}$ variables will set this way for every $i = 1, 2, \ldots, \mu$.

We can verify that all constraints resulting from the DIRECT encoding hold. Constraints (40) hold because each directed path π_i intersects the time layer in exactly one vertex. Constraints (41) hold because the directed paths are vertex-disjoint. Because the paths extend from one time layer to the next, constraints (42) hold as well. Finally, because the paths are non-overlapping, constraints (43) hold as well.

A satisfying valuation of $F_{SIM}(\eta, \Sigma)$ requires that truth values also be assigned to the $\mathcal{E}^t_u$ variables. The truth values of $\mathcal{E}^t_u$ are directly derived from the assignment of truth values to $\mathcal{X}^t_{a,v}$ through constraints (48). The satisfaction of constraints (47) is ensured by the satisfaction of constraints (43) and the transitivity of the implication through the auxiliary $\mathcal{E}^t_u$. The existence of the paths connecting the agents' initial and goal positions is ensured by the satisfaction of constraints (44)-(47), which ensure that the initial and final time layers correspond to the agents' initial and goal configurations.

If, on the other hand, we have a satisfying valuation of $F_{DIR}(\eta, \Sigma)$, then non-overlapping vertex-disjoint paths can be constructed from it. Paths $\pi_1, \pi_2, \ldots, \pi_\mu$ will be constructed according to the truth values of the variables $\mathcal{X}^t_{a,v}$. Let $\pi_i = ([u_0, 0], [u_1, 1],[u_2, 2], \ldots,[u_\eta, \eta])$ where $\mathcal{X}^0_{a_i,u_0}$, $\mathcal{X}^1_{a_i,u_1}$, ..., $\mathcal{X}^\eta_{a_i,u_\eta}$ are $TRUE$. A single path is correctly defined because for each time step $t \in \{0, 1, \ldots, \eta\}$ and agent $a_i \in A$ there is exactly one $\mathcal{X}^t_{a_i,v}$ with $v \in V$ set to $TRUE$, as ensured by constraints (40). Any successive vertices of the path are connected by arcs in $\text{Exp}_\text{T}(G, \eta)$, as ensured by constraints (42). If we consider all paths together, then constraints (41) ensure that the paths never intersect because two distinct agents cannot share a vertex. Finally, the non-overlapping property is ensured by constraints (43) because whenever a regular traversal between two successive time layers is made, no other agent can enter the affected vertices. □

Again, note that non-overlapping vertex-disjoint paths correspond to CPF solutions (Proposition 1). This, together with the proposition just proven, gives us the following theorem.

Theorem 4 *(*SOLUTION OF Σ AND $F_{DIR}(\eta, \Sigma)/F_{SIM}(\eta, \Sigma)$ SATISFACTION*). A solution of CPF $\Sigma = (G, A, \alpha_0, \alpha_+)$ exists if and only if there is$\eta \in \mathbb{N}$, for which the formula $F_{DIR}(\eta, \Sigma)$ is satisfiable. The same holds for the simplified formula $F_{SIM}(\eta, \Sigma)$.*

5.5 Summary of the space complexity of propositional encodings

The theoretical study of the size of encodings has been fine-grained. It is therefore not immediately clear how the individual encodings compare with each other. For this reason, several specific cases of CPFs with very **sparse/dense graphs** or **sparse/dense agent populations** are studied to provide a more complete picture.

A *dense/sparse graph* will be defined using an asymptotic comparison between the degree of vertices and the total number of vertices. Similarly, *sparse/dense agent populations* will be defined using an asymptotic comparison between the number of agents and the total number of vertices.

The following 4 scenarios will be distinguished (2 cases for each of the 2 parameters):

- **Scenario (i) - *dense graph/dense agent population*:**
  The number of agents μ and the size of the vertex neighborhood in G are asymptotically equal to the number of vertices (that is, $\mu \in \Theta(|V|)$ and $\deg_G(v) \in \Theta(|V|)$ for $\forall v \in V$).

The assumptions are that the graph is densely occupied by agents and that it contains many edges. As an implication of the second assumption the number of edges in the graph is asymptotically quadratic in the number of vertices; that is, $|E| \in \Theta(|V|^2)$.

The space complexities for this scenario in terms of the number of variables and clauses based upon the assumptions above are shown in Table 1.

- **Scenario (ii) - *sparse graph/dense agent population*:**
  The number of agents μ is asymptotically equal to the number of vertices, while the size of the vertex neighborhood in G is asymptotically constant (that is, $\mu \in \Theta(|V|)$ and $\deg_G(v) \in \Theta(1)$ for $\forall v \in V$).

The second assumption is that the graph is sparse and intuitively comparable to the *planar graphs* [67] commonly used in practice (for example, the grid graph studied in the experimental evaluation section satisfies this property). The assumption also means that the number of edges is asymptotically equal to the number of vertices; that is, $|E| \in \Theta(|V|)$.

The space complexities for this scenario are shown in Table 2.

- **Scenario (iii) - *sparse graph/dense agent population*:**
  The number of agents μ is asymptotically constant, while the size of the vertex neighborhood in G is asymptotically equal to the number of vertices (that is, $\mu \in \Theta(1)$ and $\deg_G(v) \in \Theta(|V|)$ for $\forall v \in V$).

This scenario can be intuitively approached as a planar graph densely occupied by agents. The space complexities for this scenario are shown in Table 3.

- **Scenario (iv) - *sparse graph/sparse agent population*:**
  The number of agents μ and the size of the vertex neighborhood in G are both asymptotically constant (that is, $\mu \in \Theta(1)$ and $\deg_G(v) \in \Theta(1)$ for $\forall v \in V$).

Again, this scenario can be intuitively approached as a planar graph with few agents inside. The space complexities for this scenario are shown in Table 4.

In this comparison of encodings, preference is given to encodings that produce the smallest number of variables or clauses. This is usually a realistic measure, as search space often correlates with the number of (decision) variables when we are trying to solve a propositional formula satisfiability problem using standard search procedures [38].

Similarly, a small number of clauses mean shorter formulas that are easier to process. Nevertheless, the preference for short formulas should be considered a mere intuitive measure since sometimes many variables can be derived from the values of other variables; that is, they do not increase the size of the search space. Similarly, a larger number of clauses may improve unit propagation in the SAT solver.

Several conclusions can be made based on the asymptotic numbers of variables and clauses in the individual encodings presented in Table 1.

In cases involving a large number of agents and dense graphs (corresponding to scenario (i)), INVERSE and ALL-DIFFERENT encodings excel in producing a small number of clauses. That was the primary design goal for these encodings, pursued through a log representation.

Table 1 *Comparison of the size complexities of CPF encodings –* ***Scenario (i) – dense graph/dense agent population***

	#Variables fine-grained/scenario (i)	#Clauses fine-grained/scenario (i)
INVERSE $F_{INV}(\eta, \Sigma)$	$O\left(\eta \cdot \left(\lvert V\rvert \cdot \lceil \log_2(\mu)\rceil + \sum_{v\in V} \lceil \log_2(\deg_G(v))\rceil + \lvert E\rvert\right)\right)$	$O\left(\eta \cdot \left(\lvert V\rvert \cdot \lceil \log_2(\mu)\rceil + \lvert E\rvert \cdot \lceil \log_2(\mu)\rceil + \sum_{v\in V} \deg_G(v) \cdot \left(\lceil \log_2\left(\deg_G(v)\right)\rceil\right)\right)\right)$
	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^2 \cdot \lceil \log_2 \lvert V\rvert\rceil\right)$
ALL-DIFFERENT $F_{DIFF}(\eta, \Sigma)$	$O(\eta \cdot \mu \cdot \lvert V\rvert)$	$O\left(\eta \cdot \lceil \log_2 \lvert V\rvert\rceil \cdot \left(\binom{\mu}{2} + \mu \cdot \lvert V\rvert\right)\right)$
	$O(\eta \cdot \lvert V\rvert^2)$	$O\left(\eta \cdot \lvert V\rvert^2 \cdot \lceil \log_2 \lvert V\rvert\rceil\right)$
MATCHING $F_{MATCH}(\eta, \Sigma)$	$O\left(\eta \cdot \left(\lvert E\rvert + \lvert V\rvert \cdot \lceil \log_2(\mu)\rceil\right)\right)$	$O\left(\eta \cdot \left(\left(\lvert V\rvert + \lvert E\rvert\right) \cdot \lceil \log_2(\mu)\rceil + \sum_{v\in V} \left(\frac{\deg_G(v)}{2}\right)\right)\right)$
	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^3\right)$
DIRECT $F_{DIR}(\eta, \Sigma)$	$O(\eta \cdot \mu \cdot \lvert V\rvert)$	$O\left(\eta \cdot \left(\mu \cdot \lvert V\rvert^2 + \mu^2 \cdot \lvert V\rvert + \mu^2 \cdot \lvert E\rvert\right)\right)$
	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^4\right)$
SIMPLIFIED $F_{SIM}(\eta, \Sigma)$	$O(\eta \cdot \mu \cdot \lvert V\rvert)$	$O\left(\eta \cdot \left(\mu \cdot \lvert V\rvert^2 + \mu^2 \cdot \lvert V\rvert + \mu \cdot \lvert E\rvert\right)\right)$
	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^3\right)$

The number of agents μ in this scenario is asymptotically equal to the number of vertices in G (that is, $\mu \in \Theta(\lvert V\rvert)$) and the size of the vertex neighborhood in G is also asymptotically equal to the number of vertices (that is, $\deg_G(v) \in \Theta(\lvert V\rvert)$ for $\forall v \in V$). For an easier reference, fine-grained complexity expressions are shown as well

Table 2 *Comparison of the size complexities of CPF encodings –* ***Scenario (ii) – sparse graph/dense agent population***

	#Variables scenario (ii)	#Clauses scenario (ii)
INVERSE $F_{INV}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$	$O\left(\eta \cdot \lvert V\rvert \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$
ALL-DIFFERENT $F_{DIFF}(\eta, \Sigma)$	$O(\eta \cdot \lvert V\rvert^2)$	$O\left(\eta \cdot \lvert V\rvert^2 \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$
MATCHING $F_{MATCH}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$	$O\left(\eta \cdot \lvert V\rvert \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$
DIRECT $F_{DIR}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^3\right)$
SIMPLIFIED $F_{SIM}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^3\right)$

The number of agents μ in this scenario is asymptotically equal to the number of vertices in G (that is, $\mu \in \Theta(|V|)$), while the size of the vertex neighborhood in G is asymptotically constant (that is, $\deg_G(v) \in \Theta(1)$ for $\forall v \in V$)

With a large number of agents and relatively sparse graphs (scenario (ii)), which is the most common case encountered in practice with planar graphs, the INVERSE and MATCHING encodings produce the smallest number of variables and clauses.

Note that the DIRECT and SIMPLIFIED encodings do not produce the shortest formulae in any of the suggested scenarios. This is due to the nature of the **direct** representation of multi-value state variables in these encodings, in which a redundant number of propositional variables is used. This redundancy also necessitates additional constraints that are needed to exclude unused valuation patterns.

The remaining two scenarios (scenarios (iii) and (iv)) can be considered as non-cooperative since the number of agents is constant and there are only limited interactions between them.

When it comes to space efficiency, the ALL-DIFFERENT encoding excels in cases involving dense graphs (scenario (iii)), whereas the INVERSE and MATCHING encodings excel on sparse graphs (scenario (iv)).

Note that most clauses correspond to the *at-most-one constraint* encoded by excluding all pairs that are fully binary and do not represent a practical performance burden.

5.6 Knowledge compilation – distance heuristics

Encodings based on time-expanded graphs can be further enhanced with a so-called *distance heuristic* [55]. Intuitively speaking, a path indicating the trajectory of an agent cannot touch

Table 3 *Comparison of the size complexities of CPF encodings –* ***Scenario (iii) – dense graph/sparse agent population***

	#Variables scenario (iii)	#Clauses scenario (iii)
INVERSE $F_{INV}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^2 \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$
ALL-DIFFERENT $F_{DIFF}(\eta, \Sigma)$	$O(\eta \cdot \lvert V\rvert)$	$O\left(\eta \cdot \lvert V\rvert \cdot \lceil \log_2 \lvert V\rvert \rceil\right)$
MATCHING $F_{MATCH}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$	$O\left(\eta \cdot \lvert V\rvert^3\right)$
DIRECT $F_{DIR}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert\right)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$
SIMPLIFIED $F_{SIM}(\eta, \Sigma)$	$O\left(\eta \cdot \lvert V\rvert\right)$	$O\left(\eta \cdot \lvert V\rvert^2\right)$

The number of agents μ in this scenario is constant (that is, $\mu \in \Theta(1)$), while the size of the vertex neighborhood is asymptotically equal to the number of vertices in G (that is, $\deg_G(v) \in \Theta(|V|)$ for $\forall v \in V$)

Table 4 *Comparison of the size complexities of CPF encodings –* ***Scenario (iv) – sparse graph/sparse agent population***

	#Variables scenario (iv)	#Clauses scenario (iv)
INVERSE $F_{INV}(\eta, \Sigma)$	$O(\eta \cdot \lvert V \rvert)$	$O(\eta \cdot \lvert V \rvert)$
ALL-DIFFERENT $F_{DIFF}(\eta, \Sigma)$	$O(\eta \cdot \lvert V \rvert)$	$O\left(\eta \cdot \lvert V \rvert \cdot \lceil \log_2 \lvert V \rvert \rceil\right)$
MATCHING $F_{MATCH}(\eta, \Sigma)$	$O(\eta \cdot \lvert V \rvert)$	$O(\eta \cdot \lvert V \rvert)$
DIRECT $F_{DIR}(\eta, \Sigma)$	$O(\eta \cdot \lvert V \rvert)$	$O\left(\eta \cdot \lvert V \rvert^2\right)$
SIMPLIFIED $F_{SIM}(\eta, \Sigma)$	$O(\eta \cdot \lvert V \rvert)$	$O\left(\eta \cdot \lvert V \rvert^2\right)$

In this scenario, both the number of agents μ and the size of the vertex neighborhood are asymptotically constant (that is, $\mu \in \Theta(1)$ and $\deg_G(v) \in \Theta(1)$ for $\forall v \in V$)

any vertices that are too far from the initial or goal vertices within the particular makespan limit η.

An agent can never show up in vertices in the time layer, for which the distance from the agent's initial position is greater than the duration of that time layer (which is equal to the position of the time layer in the time-expanded graph) or for which the distance from the goal vertex is greater than the time needed to reach the makespan limit η (which is equal to the position of the time layer in the time-expanded graph when counted from the end). Excluding unreachable vertices from further consideration can reduce the search space.

Assume a time-expanded graph $\mathrm{Exp_T}(G, \eta)$ for CPF $\Sigma = (G, A, \alpha_0, \alpha_+)$; let $\mathrm{dist}_G(u, v)$ denote the length of the shortest path connecting u to v in G; let $\mathrm{dist}_G(u, v) = \infty$ if there is no path connecting u and v. Similarly, let $\mathrm{dist}^{\rightarrow}_{\mathrm{D}}(u, v)$ denote the length of the shortest directed path connecting u to v in digraph D; let $\mathrm{dist}^{\rightarrow}_{\mathrm{D}}(u, v) = \infty$ if there is no path connecting u to v in$D$.

The following equivalence holds by the construction of a time-expanded graph $\mathrm{Exp_T}(G, \eta)$: $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}([u, t_1], [v, t_2]) < \infty$ iff $\mathrm{dist}_G(u, v) \leq t_2 - t_1$. Having defined this equivalence, we can now express the aforementioned intuitive distance reasoning in G over $\mathrm{Exp_T}(G, \eta)$.

Proposition 10 *(*DISTANCE HEURISTIC*). Any solution* $\vec{s} = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_\eta]$ *to* Σ *satisfies that* $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}([\alpha_0(a_i), 0], [\alpha_t(a_i), t]) < \infty$ *and* $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}([\alpha_t(a_i), t]\left[\alpha_\eta(a_i), \eta\right]) < \infty$ *for every* $i \in \{1, 2, \ldots, \mu\}$ *and* $t \in \{0, 1, \ldots, \eta\}$.

Proof If $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}([\alpha_0(a_i), 0], [v, t]) = \infty$ or $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}\left([v, t], \left[\alpha_\eta(a_i), \eta\right]\right) = \infty$ for any $v \in V$ and $t \in \{0, 1, \ldots, \eta\}$, then there is no directed path connecting $[\alpha_0(a_i), 0]$ and $\left[\alpha_\eta(a_i), \eta\right]$ that passes through $[v, t]$. A fortiori, there is no path connecting $[\alpha_0(a_i), 0]$ and $\left[\alpha_\eta(a_i), \eta\right]$ that passes through $[v, t]$ and does not overlap with or intersect other paths. Hence, $\alpha_t(a_i) \neq v$. □

The proposition above can be used to design a heuristic. All vertices $\left[v_j, t\right]$ with $v_j \in V$ and $t \in \{0, 1, \ldots, \eta\}$ in $\mathrm{Exp_T}(G, \eta)$, for which $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}\left([\alpha_0(a_i), 0], \left[v_j, t\right]\right) = \infty$ or $\mathrm{dist}^{\rightarrow}_{\mathrm{Exp_T}(G,\eta)}(\left[v_j, t\right]\left[\alpha_\eta(a_i), \eta\right]) = \infty$, can be excluded from the trajectories corresponding to agent a_i (in the original graph this translates into the requirement that agent

a_i should not enter v_j in time step t). In terms of encodings, this can be formulated as follows:

$$\mathcal{A}^t_{v_j} \neq i \quad \text{in the INVERSE and MATCHING encoding} \tag{50}$$

$$\mathcal{L}^t_{a_i} \neq j \quad \text{in the ALL-DIFFERENT encoding} \tag{51}$$

$$\neg\mathcal{X}^t_{a_i,v_j} \quad \text{in the DIRECT/SIMPLIFIED encoding} \tag{52}$$

The inequality between a bit vector and constant is encoded as a single clause that forbids the bit vector from taking that constant. That is, at least one bit must disagree with the binary representation of the constant. For example, the inequality $\mathcal{A}^t_v \neq c$ is encoded as follows:

$$\text{con}_{\neq}(\mathcal{A}^t_v, c) = \bigvee_{\mathbb{i}=0}^{\lceil \log_2 \mu \rceil - 1} \neg\text{lit}(\mathcal{A}^t_v, c, \mathbb{i}) \tag{53}$$

The added inequalities are the logical consequence of the encoded propositional formula (that is, for example $F_{INV}(\eta, \Sigma) \Rightarrow \mathcal{A}^t_{v_j} \neq i$ is a valid formula). Thus, in theory, the SAT solver should be able to infer the unreachability of vertices in certain time steps. However, such inferences may be costly to make for the SAT solver. It is more efficient to inject into the SAT solver the outside knowledge of vertex unreachability gained by understanding the problem structure.

6 Experimental evaluation

In the experimental evaluation section we focus on measuring the actual **size** of the proposed encodings as well as the **runtime** when encodings are used in makespan-optimal solving CPF in a makespan-optimal way using SAT.

The solution procedure presented as Algorithm 1 formed the core framework of our makespan-optimal CPF solution technique. In other words, we adopted the **incremental strategy** of querying the SAT solver. In this framework, the individual propositional encodings can be seen as exchangeable modules.

All encodings we implemented use the built-in distance heuristic discussed in Section 5.6.

The SAT-based CPF solution procedure was implemented in C++ along with procedures aimed at generating propositional formulae (the solution procedure and formulae generation were compiled into a single executable program).

In our tests we used the `glucose 3.0` SAT solver [2, 3], which was ranked among the best SAT solvers in recent *SAT Competitions* [5, 6, 26] in the category of hard combinatorial problems, of which CPF can be considered a sub-class.

The SAT solver was a separate module called externally by the CPF solution procedure. The solution procedure generated a formula and saved it to a file in the textual DIMACS format [5], which was then submitted as input to the SAT solver; the answer from the SAT solver was saved to another file which the procedure read and processed further.

6.1 Benchmark setup

We followed the benchmark setup originally suggested by Silver in [47]. Four-connected grids of various sizes were used to model the environments in the test instances. The grid sizes ranged from 6×6 to 12×12, with 20% of randomly selected vertices occupied by obstacles (an obstacle was represented by a missing vertex in the grid – see Fig. 6).

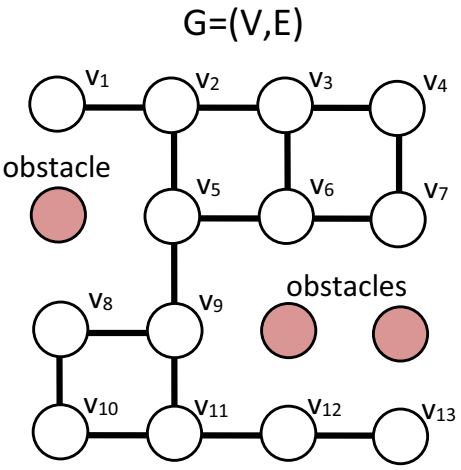


Fig. 6 *4×4 four-connected grid with 3 obstacles.* The positions of obstacles in the grid are depicted, although they are actually not present in the graph

The agents' initial and goal configurations were random, obtained by placing the agents one by one onto the grid. The positions were randomly selected from the remaining unoccupied vertices. Only solvable instances were included in runtime tests (solvability was tested by a sub-optimal complete algorithm).

Although rather small synthetic instances, the test cases were very computationally intensive when there were many agents. We predicted that this class of instances would lend themselves to the SAT-based approach particularly well.

We did not consider the SAT-based approach to be a universal strategy. In instances involving few agents and large underlying graphs, we did not expect to see any interesting results due to the long formulae generated. Tests involving large graphs were therefore omitted from the experimental evaluation.

To allow the full reproducibility of the presented results, all source codes and experimental data were published online on: http://ktiml.mff.cuni.cz/∼surynek/research/j-encoding-2015.

6.2 Encoding size evaluation

The size of propositional formulae was tested using 4-connected grids with an incremental number of agents. The number of agents ranged from 1 to one half of all vertices in the graph.

For each number of agents, 10 random CPF instances were generated and their characteristics measured. For each number of agents we generated formulae corresponding to all encodings presented. The number of time steps in time-expanded graphs was fixed and relative to the size of the instance – 12 for a 6×6 grid; 16 for a 8×8 grid; and 24 for a 12×12 grid.

The *average number of propositional variables*, *average number of clauses*, and *average clause length* were calculated for each encoding and number of agents from 10 randomly generated instances. Selected results are shown in Tables 5, 6, and 7 – the preferred values for the individual characteristics are listed in bold.

The number of variables and clauses directly correspond to the length of the formulae. Preference is given to shorter formulae, as these are expected to be easier to solve and process.

Average clause length is an important measure. Short clauses are preferred because they support *unit propagation* [11], which allows us to derive the values of other variables without searching.

Table 5 *Size comparison of the propositional encodings of CPF in a 6×6 grid*

Grid 6×6 |Agents|			**INVERSE**		**ALL-DIFFERENT**		**MATCHING**		**DIRECT**		**SIMPLIFIED**	
1	#Variables	*Clause*	3 384.3	*2.622*	701.4	*2.979*	1 841.1	*2.436*	342.0	*2.261*	684.0	*2.587*
	#Clauses	*length*	12 494.3		3 160.7		10 300.6		6 048.2		1 499.6	
2			3 738.3	*2.599*	1 723.5	*2.980*	2 195.1	*2.512*	684.0	*2.353*	1 026.0	*2.562*
	17 012.0		7 191.7		13 497.1		14 176.0		3 441.4			
4			4 092.3	*2.642*	4 127.5	*3.026*	2 549.1	*2.632*	1 368.0	*2.427*	1 710.0	*2.423*
	22 110.2		15 392.6		17 274.1		34 962.7		7 956.1			
8			4 446.3	*2.794*	12 066.7	*3.060*	2 903.1	*2.867*	2 736.0	*2.543*	3 078.0	*2.319*
	28 225.0		39 216.1		22 067.7		99 381.3		21 436.3			
16			4 800.3	*3.133*	38 791.0	*3.104*	3 257.1	*3.313*	5 472.0	*2.633*	5 814.0	*2.201*
	36 527.1		109 781.6		29 048.6		308 484.7		64 314.8			

CPF instances are generated in a 6×6 4connected grid with 20% of vertices occupied by obstacles. The number of time steps in time-expanded graph η is 12. The number of *variables* and *clauses*, and the *average clause length* are listed for different sizes of a set of agents A. Short formulas and clauses (i.e, such that support unit propagation) are preferred – the best values achieved are shown in bold. DIRECT and SIMPLIFIED encodings excel in a number of measures on the 6×6 grid

Table 6 *Size comparison of the propositional encodings of CPF over a 8×8 grid*

Grid 8×8 |**Agents**|			**INVERSE**		**ALL-DIFFERENT**		**MATCHING**		**DIRECT**		**SIMPLIFIED**	
1	#Variables	*Clause*	4 520.3	*2.616*	1 489.3	*3.057*	4 520.3	2.441	814.4	*2.149*	1 628.8	*2.550*
	#Clauses	*length*	25 881.1		7 930.4		25 881.1		23 241.9		3 384.6	
4			10 019.5	*2.641*	7 834.5	*3.103*	6 181.1	2.640	3 257.6	*2.272*	4 072.0	*2.374*
	55 437.0		34 781.9		43 171.0		115 934.3		17 997.8			
8			10 849.9	*2.792*	21 875.4	*3.113*	7 011.5	2.874	6 515.2	*2.390*	7 329.6	*2.694*
	70 725.9		83 794.2		55 050.3		297 319.9		49 381.3			
16			11 680.3	*3.127*	67 088.3	*3.147*	7 841.9	3.315	13 030.4	*2.505*	13 844.8	*2.180*
	91 344.5		216 745.4		72 259.3		840 540.6		150 259.2			
32			12 510.7	*3.733*	230 753.0	*3.168*	8 672.3	4.045	26 060.8	*2.621*	26 875.2	*2.111*
	122 170.3		646 616.2		99 675.5		2 738 584.7		510 672.1			

The number of time layers in the corresponding time-expanded graph is 16. DIRECT and SIMPLIFIED encodings have fewer variables and clauses for smaller numbers of agents, whereas the MATCHING encoding performs better in these measures when there are many agents in an instance

Table 7 *Size comparison of the propositional encodings of CPF in a 12×12 grid*

Grid 12×12 |**Agents**|			**INVERSE**		**ALL-DIFFERENT**		**MATCHING**		**DIRECT**		**SIMPLIFIED**	
1	#Variables	*Clause*	29 798.7	*2.635*	4 973.9	*3.031*	15 961.3	*2.443*	2 767.2	*2.073*	5 534.4	*2.578*
	#Clauses	*length*	116 302.8		30 928.8		94 603.2		168 027.8		11 587.0	
8			38 172.3	*2.793*	55 602.1	*3.088*	24 334.9	*2.871*	22 137.6	*2.230*	24 904.8	*2.289*
	257 739.9		271 730.3		197 835.9		1 722 059.3		167 026.1			
16			40 963.5	*3.115*	153 047.5	*2.999*	27 126.1	*3.300*	44 275.2	*2.343*	47 042.4	*2.059*
	330 249.1		656 615.4		257 974.6		4 310 137.7		542 862.4			
32			43 754.7	*3.701*	475 135.0	*3.148*	29 917.3	*4.021*	88 550.4	*2.475*	91 317.6	*2.112*
	439 680.0		1 628 634.8		354 306.4		12 121 528.6		1 730 745.7			
64			46 545.9	*4.632*	1 626 205.9	*3.183*	32 708.5	*5.065*	177 100.8	*2.594*	179 868.0	*2.062*
	620 942.7		4 713 520.1		522 834.3		38 361 723.7		6 297 660.9			

The number of time layers in the time-expanded graph is 24. The MATCHING encoding is clearly the smallest for larger numbers of agents

The results indicate that DIRECT and SIMPLIFIED encodings have the best size characteristics with respect to the preference for small size in cases involving small numbers of agents in an instance. The same result can be observed all for all grid sizes.

Because neighborhood connectivity in 4-connected grids can be considered constant; that is, $\deg_G(v) \in \Theta(1)$ for $\forall v \in V$, the cases where DIRECT and SIMPLIFIED encoding exhibit the best size characteristics roughly correspond to scenario (iv).

However, theoretical asymptotic formula size estimations suggest different results – The performance of DIRECT and SIMPLIFIED encodings should be the same as that of the other encodings when it comes to the number of variables and worse than the other encodings when it comes to the number of variables. The experimental evaluation yielded a surprising result in this respect.

If the number of agents is larger, the MATCHING encoding excels in size characteristics for all grid sizes. It produces the fewest propositional variables and the fewest number of clauses. Considering that this case roughly corresponds to scenario (ii), these observations correspond to theoretical asymptotic estimations which indicate that the MATCHING and INVERSE encodings should be the smallest (note that according to the experimental results the INVERSE encoding is the second smallest).

The SIMPLIFIED encoding has the shortest average clause length. As the number of agents increase, the average clause length approaches 2 for all grid sizes (that is, most of the clauses in the SIMPLIFIED encoding are binary).

The aforementioned observations regarding the static characteristics of the encodings indicate that the MATCHING encoding and even more so the SIMPLIFIED encoding should perform better than the other encodings.

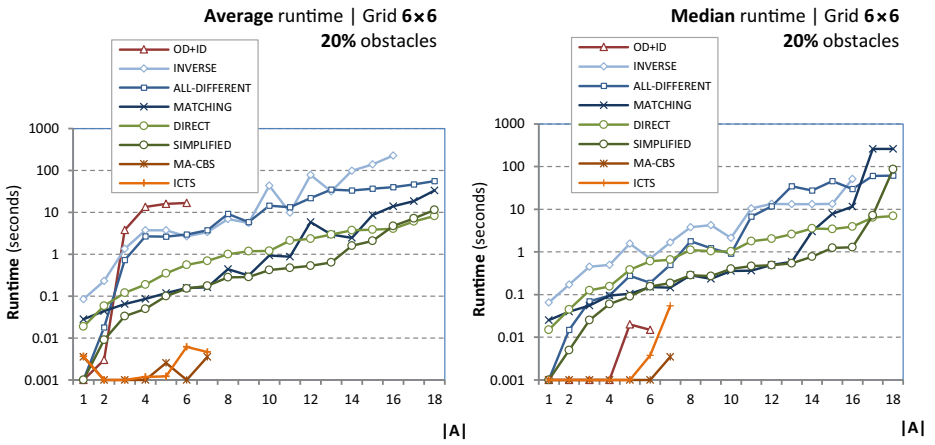
6.3 Runtime evaluation

We re-implemented the A*-based operator decomposition/independence detection method OD+ID [51] in C++, adding an objective function for minimizing the *makespan*; the original version of OD+ID was designed to minimize the *sum-of-costs*. Analogical modifications of *conflict-based search* CBS [50] and *increasing cost tree search* ICTS [49] were developed. In this case, the existing implementations in C# were modified to optimize makespan as opposed to total cost. The makespan-optimal versions of OD+ID, CBS, and ICTS were compared with the SAT-based solution method with various encodings.

CBS and ICTS solvers are available in multiple-parameter configurations suitable for different scenarios. We adopted the best-known configurations that are used with grids containing obstacles. It is also important to note that these solvers are, by design, focused on solving CPF in a sum-of-costs-optimal way and the makespan-optimal modifications to some extent compromise their design.

While techniques like MDD (*multi-value decision diagram* – a cost-optimal analogy to our distance heuristics) in ICTS significantly reduce the search space in the case of a sum-of-costs objective, they are equally efficient in the makespan-optimal case. This is mostly due to the different nature of the makespan objective, which produces a larger search space (the optimal cost is the lower bound for cost in the makespan-optimal solution).

Again, CPFs in 6×6, 8×8, and 12×12 4-connected grids with 20% of vertices occupied by randomly placed obstacles were used. The agents' initial and goal configurations were generated randomly. We evaluated the runtime for an incremental number of agents in instances. For each number of agents, 10 random instances were generated and solved. All instances used for the evaluation were solvable.



Grid 6×6 |A|	1	2	3	4	5	6	7	8	9	10	12	14	16	18
Makespan	4.2	4.9	5.2	5.6	6.5	7.0	7.0	7.4	7.1	6.9	7.9	8.6	8.2	9.1

Fig. 7 ***Runtime evaluation****in a 6×6 4-connected grid with 20% of vertices occupied by obstacles.* The search-based methods OD+ID, CBS, ICTS and the SAT-based method with INVERSE, ALL-DIFFERENT, MATCHING, DIRECT, and SIMPLIFIED encodings are compared on random CPF instances in the grid. We show the average and median runtimes calculated from the runtimes achieved on 10 random instances; the average optimal makespans are also shown. The OD+ID, CBS, and ICTS methods do not scale with the number of agents, while the SAT-based method performs better with many agents. The SIMPLIFIED encoding yields particularly good results. When it comes to runtime, the best and the worst encoding differ by up to two orders of magnitude

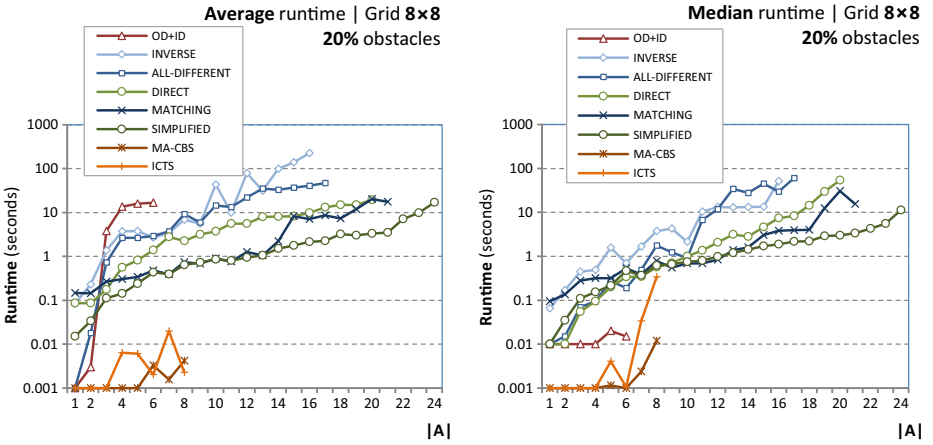
The timeout for solving a single CPF instance was set to 256 seconds (approximately 4 minutes). The number of agents was gradually increased until all 10 random instances could be solved before the timeout was reached – in other words, each solution method (encoding) was characterized by the maximum number of agents, for which it was able to solve all 10 random instances within the timeout period.

The average and median runtimes were calculated from these 10 instances for all tested methods. In the case of the SAT-based CPF solving methods, the runtime was a sum of the runtime of the core CPF solution procedure (corresponding to Algorithm 1) and the runtimes of all runs of the SAT solver invoked by the core procedure.

The runtime results together with average optimal makespans are shown in Figs. 7, 8, and 9[4] (note that all methods generate solutions with the same optimal makespan).

Although offering superior performance for smaller numbers of agents, OD+ID, CBS, and ICTS do not scale up, as their runtime increases quickly when more agents are introduced. It seems that whenever the interaction between the agents becomes more intense, these methods cannot fully utilize their independence detection and conflict resolution heuristics. The SAT-based solution method with all its encodings performs better and scales

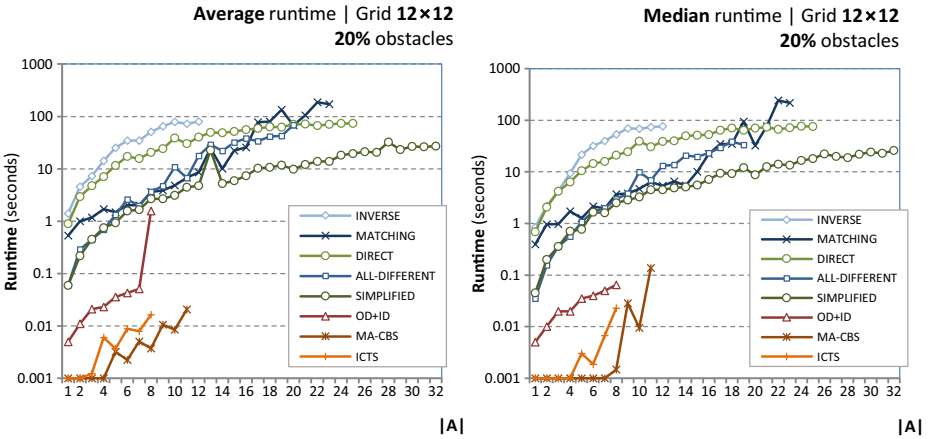
[4]All runtime measurements were performed using a computer with 4-core CPU Xeon 2.0GHz and 12GB RAM under Linux kernel 3.5.0-48. Although multiple cores were utilized to run the experiments in parallel, the individual instances were solved in a single thread (that is, the core solution procedure and all its calls to the SAT solver were run in a single thread).



Grid 8×8 |A|	1	2	3	4	5	6	8	10	12	14	16	18	20	22	24
Makespan	5.6	5.3	7.8	8.0	8.8	10.7	9.7	10.4	10.8	10.0	11.0	10.5	11.5	11.2	11.5

Fig. 8 ***Runtime evaluation****in a 8×8 4-connected grid with 20% of vertices occupied by obstacles.* Again, the SAT-based method with the SIMPLIFIED encoding exhibited the best performance for larger numbers of agents. The performance of the MATCHING encoding is initially promising, but it quickly deteriorates when there are more than approximately 14 agents

up with the number of agents. Namely, the SIMPLIFIED encoding delivers the best performance regardless of grid size, followed by the MATCHING, DIRECT, ALL-DIFFERENT, and INVERSE encodings.



Grid 12×12 |A|	1	2	3	4	6	8	10	12	16	18	20	24	28	32
Makespan	7.5	10.6	11.4	12.7	13.9	15.3	13.8	14.9	15.5	16.3	14.6	17.3	15.9	16.4

Fig. 9 ***Runtime evaluation****in a 12×12 4-connected grid with 20% of vertices occupied by obstacles.* The SIMPLIFIED encoding delivers the best performance again. Within the timeout period, the MATCHING encoding is able to solve instances with the second largest number of agents

It is important to note that the superior performance of the SIMPLIFIED encoding was predicted in the theoretical analysis of the encoding. In particular, we expected it to support unit propagation well. The competitive MATCHING encoding was also predicted to deliver good performance thanks to its small size in test instances.

An interesting behavior of the MATCHING encoding can be observed. Its initial performance is as promising as that of the SIMPLIFIED encoding for a small number of agents; however, it quickly deteriorates and the encoding is eventually outperformed by the DIRECT encoding on 6×6 and 12×12 grids involving larger numbers of agents (in a 8×8 grid, a deterioration in the performance of the MATCHING encoding can be observed as well, but is less significant – the DIRECT encoding reaches the timeout before it can overtake the MATCHING encoding).

Instances of 6×6 grids with agent occupancy rates of up to 62% can be solved within the timeout period by using the SIMPLIFIED encoding. The maximum agent occupancy rates in 8×8 and 12×12 grids are 46% and 28%, respectively, when using the SIMPLIFIED encoding. The OD+ID, CBS, and ICTS methods can solve instances characterized by agent occupancy rates of up to 24%, 13%, and 7% in 6×6, 8×8, 12×12 grids, respectively.

Another general conclusion from the experimental evaluation above is that the **log encoding** used to encode the finite domain state variables in the INVERSE, ALL-DIFFERENT, and MATCHING encodings contributes to small size, but hurts the overall performance, as these encodings clearly performed worse than the DIRECT and SIMPLIFIED encodings which use a **direct encoding**.

On the other hand, the simple design of the DIRECT and SIMPLIFIED encodings is not detrimental to solution performance. The simple design of the variables allowed us to model the constraints using short clauses that greatly support unit propagation, which is the key to the good performance of both these encodings.

6.4 Solution quality evaluation

Although all solutions generated by the proposed SAT-based solution techniques are makespan-optimal, that is, optimal with respect to the objective function selected, they may differ in other aspects. Of particular importance is the *total number of moves* executed by the agents, which is referred to as the *sum-of-costs* objective. Sum-of-costs can be understood as the **total energy** required for the agents to execute their movements. Recall that the original versions of OD+ID, CBS, and ICTS were designed to optimize sum-of-costs [49–51].

Hence, it would be interesting to look at the solutions generated by the makespan-optimal SAT-based methods in terms of sum-of-costs, although this was not taken into consideration in the design of the propositional encodings.

The way a particular problem is encoded into a propositional formula significantly influences the heuristics that the SAT solver uses to select variables and their values. The values selected to satisfy the formula are then reflected in the CPF solution reconstructed from its satisfying valuation. Although not a rule, SAT solvers are by default configured to assign $FALSE$ values, unless it is more advantageous to assign a $TRUE$ value.

Note that only the values assigned to visible propositional variables are directly reflected in the resulting CPF solution. The visible propositional variables in the proposed encodings are either part of a directly encoded state (DIRECT and SIMPLIFIED encodings) or a binary encoded bit vector (INVERSE, ALL-DIFFERENT, and MATCHING encodings).

The value patterns in bit vectors representing state variables behave differently with respect to the preference for assigning $FALSE$ in the direct and log encoding. Propositional

variables within a directly encoded state directly correspond to the occupancy of a vertex or edge by a fixed agent. The assignment of a $FALSE$ value to the propositional variable of a directly encoded state corresponds to non-occupancy by the particular fixed agent. A complete non-occupancy occurs if and only if all propositional variables directly encoding the state are set to $FALSE$ for all agents.

For bit vector propositional variables, the occupancy of a corresponding vertex or edge occurs if any of the propositional variables within the bit vector are set to $TRUE$. Non-occupancy corresponds to the assignment of the integer zero to the bit vector. This means the assignment of $FALSE$ to all propositional variables that make up the bit vector.

Assuming that the SAT solver uses a conservative approach to finding a solution, that is, it prefers to assign $FALSE$ values, the use of encodings with visible variables that directly encode states seems to result in lower vertex and edge occupancy rates. This corresponds to CPF solutions consisting of fewer moves.

The reasoning behind this hypothesis assumes that we have a set of agents A and a position (vertex/edge) that is to be occupied by at most one agent from A. The occupancy of the position is modeled by a directly encoded state in one scenario and as a log encoded bit vector in another scenario.

The first scenario yields $|A|$ propositional variables with $|A| + 1$ allowed assignments - one of these assignments corresponds to non-occupancy of the position assigns $FALSE$ to all propositional variables; other allowed assignments have just one propositional variable set to $TRUE$. The second scenario yields $\log_2 \lceil |A| + 1 \rceil$ propositional variables - all possible combinations of Boolean values are allowed as assignments, while all propositional variables set to $FALSE$ correspond to non-occupancy of the position. If the preference for assigning $FALSE$ actually results in setting strictly fewer variables to $TRUE$, then non-occupancy immediately occurs in the first scenario, while there is little chance that non-occupancy will occur in the second scenario (setting strictly fewer variables to $TRUE$ may lead to another assignment of the bit vector with some propositional variables set to $TRUE$ - that is, representing some occupancy).

The results of measuring the total number of moves generated by the SAT-based CPF solution method using the encodings tested are presented in Figs. 10, 11, and 12.

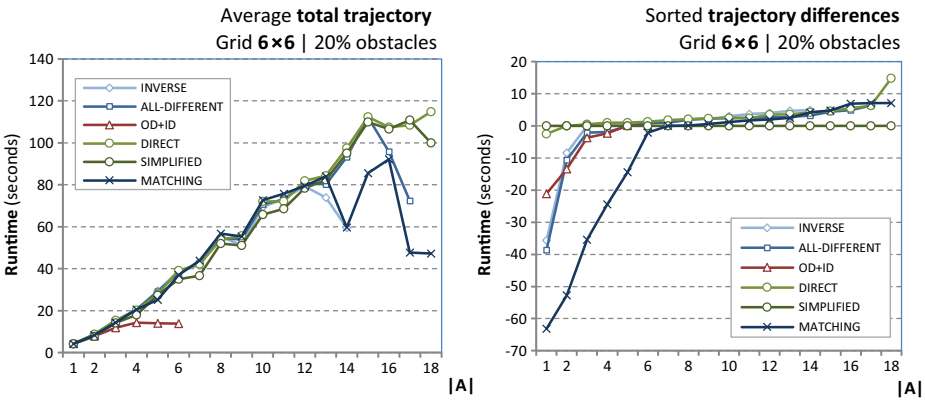


Fig. 10 ***Comparison of the quality of solutions*** *in a 6×6 4-connected grid.* We compare the total number of moves in optimal solutions obtained by each method under evaluation for an incremental number of agents in the grid (*left*). The sorted differences in the total number of moves from the number of moves generated by the SIMPLIFIED encoding are also shown (*right*). The SIMPLIFIED encoding yields a solution with a smaller number of moves, compared to the other methods, in about one third of all solutions generated

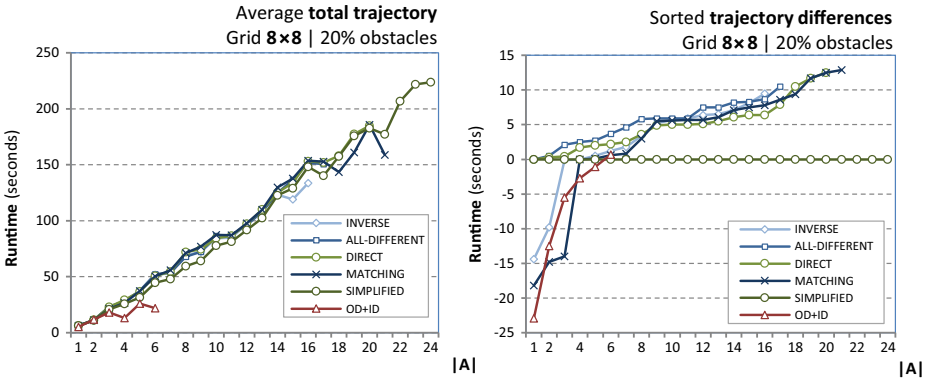


Fig. 11 ***Comparison of the quality of solutions*** *in a 8×8 4-connected grid.* The SIMPLIFIED encoding yields solutions with the fewest moves in approximately 75% of the cases. Note that the A*-based OD+ID generates solutions with even fewer moves, but it does not scale up well with agent density

We used the same set of test instances of 4-connected grids as in the runtime measurement. The total number of moves generated by OD+ID was also included in the measurement.

The SIMPLIFIED encoding produced the smallest number of moves in most test instances. Thus, we also present the sorted differences between the total numbers of moves produced by the SIMPLIFIED encoding and the other methods. It can be also observed that OD+ID generates solutions with the smallest number of moves for instances containing few agents. Unfortunately, OD+ID does not scale up well enough with the number of agents.

There is barely any significant difference between the total numbers of moves generated by the other methods except for a marginal tendency of the DIRECT encoding to produce better solutions compared to methods using binary encoded bit vectors, which is observable especially in the 8×8 grid.

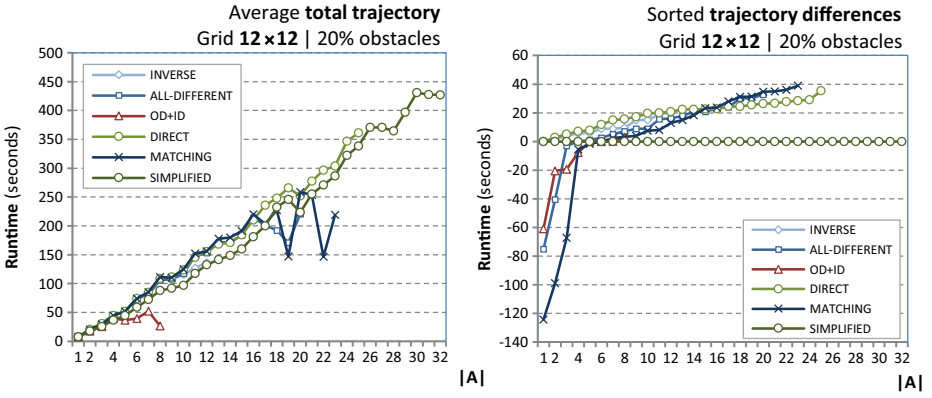


Fig. 12 ***Comparison of the quality of solutions*** *in a 12×12 4-connected grid.* The SIMPLIFIED encoding almost invariably yields a solution with fewer moves compared to the other methods within this larger scenario. Again, solutions with the fewest moves are generated by OD+ID, but the comparison was run for instances with only a few agents due to the insufficient scalability of OD+ID

All in all, we can conclude that the hypothesis stating that the encodings using directly encoded states would be better in terms of the total number of moves than encodings with binary encoded bit vectors has proven to be correct.

7 Discussion

The superior performance of the least sophisticated encodings – DIRECT and SIMPLIFIED, which are based on the direct representation of state variables, was relatively surprising. A definite drawback of these encodings is their size which mostly corresponds to the representation of the *at-most-one* constraints (constraints (40) and (41) for instance) that exclude any forbidden pairs.

We did further experiments with more space-efficient representations of the at-most-one constraint based on *sequential adder* circuits [4] and *Cartesian products* [16]. The results indicate a negative performance gain against the present basic encoding, although the formulae were generated faster due to their smaller size.

It is important to note that the ICTS and CBS solvers used in the experimental evaluation were implemented in C#, whereas the SAT-based solver was written in C++. This may give the SAT-based solver a certain performance advantage (though we are not able to quantify it). Note that the engineering aspects of the present SAT-based solver can in many ways be improved. For example, communication with the SAT solver occurs through text files generated and saved to a disk. While generating disk data does not affect performance, as everything takes places in a cache memory on high-spec hardware, the textual form represents an overhead – the CPF solver first translates its internal representation of the time-expanded graph into a textual formula and then the SAT solver translates the text back into its internal representation of the formula.

8 Conclusions

We have described several propositional encodings of the cooperative path-finding problem (CPF) - INVERSE, ALL-DIFFERENT, MATCHING, DIRECT, and SIMPLIFIED. The present encodings are based on the concept of a time-expanded graph. This graph expands the graphical model of the environment in time, so that the agent configurations in all time steps up to a certain final time step can be represented.

Time-expanded graphs are an essential step towards building propositional formulae that include an encoded query for checking whether there is a solution of a given CPF with the specified number of time steps. A makespan-optimal solution is then arrived at by submitting multiple queries to a SAT solver. The CPF-to-SAT reduction allow us to access the entire collection of advanced search, pruning, and learning techniques that are available in the SAT solver, which can thus be used to solve CPF problems.

The encodings proposed here use either log encoded bit vectors (INVERSE, ALL-DIFFERENT, and MATCHING encodings) or directly encoded states (DIRECT and SIMPLIFIED encodings) to model the agent configurations in individual time steps. The use of log encoded bit vectors results in formulae that are shorter in terms of the number of variables and clauses. On the other hand, encodings with directly encoded states offer better support for *Boolean constraint propagation* (*unit propagation*), which is made possible by the presence of many short clauses.

Experimental evaluation indicates that the use of SAT in solving CPF problems is generally the best option in highly constrained situations (environments densely occupied by agents). SAT-based makespan-optimal methods scale up better with the number of agents than alternative state-of-the-art search-based techniques, which were, however, originally designed to produce sum-of-costs-optimal solutions (the adaptation from sum-of-costs to makespan may impair the performance).

If we compare SAT encodings alone, the SIMPLIFIED encoding delivered the best performance, although it was one of the least sophisticated encodings in the present collection. Only the SIMPLIFIED encoding solved the instances with the highest agent occupancy rates within the specified timeout period. Moreover, a comparison of the quality (defined in terms of the total number of moves generated) of the solutions produced by the SAT-based methods also indicated that the SIMPLIFIED encoding was the most efficient.

One possible future direction of the SAT-based approach to makespan-optimal CPF solving is a more sophisticated treatment of cases when encoded formula is unsatisfiable. Modern SAT solvers are able to provide *unsatisfiable core* [11] that can be further processed in the CPF solving process.

Acknowledgments This work has been supported by an AIRC project commissioned by the New Energy and Industrial Technology Development Organization Japan (NEDO) and the Czech-Israeli cooperation project number 8G15027. The author would like to thank the research team at Ben Gurion University, Israel led by Ariel Felner and Roni Stern and the student members of the team, Guni Sharon and Eli Boyarski, for providing the source code needed to conduct the present experiments.

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/. [accessed in September 2014] (2013)
3. Audemard, G., Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 399–404 (2009)
4. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: Principles and Practice of Constraint Programming, 9th International Conference (CP 2003), pp. 108–122, LNCS 2833, Springer (2003)
5. Balint, A., Belov, A., Heule, M., Järvisalo, M.: SAT 2013 competition. http://www.satcompetition.org/. [accessed in April 2015] (2013)
6. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Overview and analysis of the SAT Challenge 2012 solver competition. In: Artificial Intelligence, vol. 223, pp. 120-155. Elsevier (2015)
7. Balyo, T.: Relaxing the relaxed exist-step parallel planning semantics. In: Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2013), pp. 865–871. IEEE Computer Society (2013)
8. Balyo, T., Biere, A., Iser, M., Sinz, C.: SAT Race 2015. Artificial Intelligence. Vol. 241, pp. 45–65, Elsevier (2016)
9. Barahona, P., Hölldobler, S., Nguyen, V.-H.: Representative encodings to translate finite CSPs into SAT. In: Proceedings of the Integration of AI and OR Techniques in Constraint Programming - 11th International Conference (CPAIOR 2014), pp. 251–267. Springer (2014)
10. Biere, A., Brummayer, R.: Consistency checking of all different constraints over bit-vectors within a SAT solver. In: Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2008), pp. 1–4. IEEE Computer Society (2008)
11. Biere, A., Heule, M., Van Maaren, H., Walsh, T.: Handbook of satisfiability. IOS Press (2009)
12. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and Practice - Closing the Gap. Proceedings of the System Modelling and Optimization 1999. In: 19th IFIP TC7 Conference on System Modelling and Optimization, pp. 19–50. Kluwer (2000)

13. Bixby R. E., Gu, Z., Rothberg, E.: The Gurobi Team: Gurobi Optimization - The Best Mathematical Programming Solver. http://www.gurobi.com/. [accessed in September 2016] (2016)
14. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995), pp. 1636–1642. Morgan Kaufmann (1995)
15. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, (TACAS 2009), pp. 174–177, LNCS 5505, Springer (2009)
16. Chen, J.: A new SAT encoding of the at-most-one constraint. In: Proceedings of the 9th International Workshop of Constraint Modeling and Reformulation (ModRef 2010), the 16th International Conference on the Principles and Practice of Constraint Programming (CP 2010), Uppsala Universitet (2010)
17. Clark, D.A., Frank, J., Gent, I.P., MacIntyre, E., Tomov, N., Walsh, T.: Local search and the number of solutions (1996)
18. Crawford, J.M., Baker, A.B.: Experimental results on the application of satisfiability algorithms to scheduling problems. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994), pp. 1092–1097. AAAI Press (1994)
19. Čáp, M., Novák, P., Vokřínek, J., Pěchouček, M.: Multi-agent RRT: sampling-based cooperative pathfinding. In: International conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), pp. 1263–1264. IFAAMAS (2013)
20. Čáp, M., Novák, P., Kleiner, A., Selecký, M.: Prioritized planning algorithms for trajectory coordination of multiple mobile robots. In: IEEE Transaction Automation Science and Engineering, Volume 12, Number 3, pp. 835–849, IEEE Computer Society (2015)
21. Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)
22. Erdem, E., Kisa, D.G., Öztok, U., Schüller, P.: A general formal framework for pathfinding problems with multiple agents. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013). AAAI Press (2013)
23. Gent, I.P.: Arc consistency in SAT. In: Proceedings of the 15th Eureopean Conference on Artificial Intelligence (ECAI 2002), pp. 121–125. IOS Press (2002)
24. Huang, R., Chen, Y., Zhang, W.: A novel transition based encoding scheme for planning as satisfiability. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010). AAAI Press (2010)
25. Iwama, K., Miyazaki, S.: SAT-variable complexity of hard combinatorial problems. In: Proceedings of Technology and Foundations - Information Processing '94, Volume 1, Proceedings of the IFIP 13th World Computer Congress, pp. 253–258, North-Holland (1994)
26. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine, Vol. 33, Number 1. AAAI Press (2012)
27. Ježek, M.: Modeling of Cooperative Pathfinding. Bachelor thesis, Charles University (2015)
28. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999), pp. 318–325. Morgan Kaufmann (1999)
29. Kim, D., Hirayama, K., Park, G.-K.: Collision Avoidance in Multiple-Ship Situations by Distributed Local Search. J. Adv. Comput. Intell. Intell. Inform. (JACIII) **18**(5), 839–848 (2014). Fujipress
30. KIVA Systems. Official web site. http://www.kivasystems.com/, 2015 [accessed in April 2015]
31. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241–250. IEEE Computer Society (1984)
32. Luna, R., Bekris, K.E.: Push and Swap: Fast cooperative path-finding with completeness guarantees. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 294–300. IJCAI/AAAI (2011)
33. Michael, N., Fink, J., Kumar, V.: Cooperative manipulation and transportation with aerial robots. Autonom. Robot. **30**(1), 73–86 (2011). Springer
34. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and coreguided MaxSAT solving: A survey and assessment. Constraints **18**(4), R8–534 (2013). Springer
35. Nguyen, V.-H., Mai, S.T.: A new method to encode the at-most-one constraint into SAT. In: Proceedings of the 6th International Symposium on Information and Communication Technology (SoICT 2015), pp. 46–53. ACM (2015)
36. Nguyen, V.-H., Velev, M.N., Barahona, P.: Application of hierarchical hybrid encodings to efficient translation of CSPs to SAT. In: Proceedings of the 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013), pp. 1028–103. IEEE Computer Society (2013)
37. Petke, J., Jeavons, P.: The order encoding: From tractable CSP to tractable SAT. In: Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011), LNCS 6695, pp. 371–372. Springer (2011)

38. Petke, J.: Bridging Constraint Satisfaction and Boolean Satisfiability. Artificial Intelligence: Foundations, Theory, and Algorithms, Springer (2015)
39. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N × N extension of the 15- PUZZLE Is intractable. In: Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), pp. 168–172. Morgan Kaufmann (1986)
40. Régin, J.-C.: A filtering algorithm for constraints of difference in CSPs. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994), pp. 362–367. AAAI Press (1994)
41. Rintanen, J., Heljanko, K., Niemela, I.: Planning as satisfiability: parallel plans and algorithms for plan search. Artif. Intell. **170**(12-13), 1031–1080 (2006). Elsevier
42. Ryan, M.R.K.: Graph decomposition for efficient multi-robot path planning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 2003–2008. IJCAI Conference (2007)
43. Ryan, M.R.K.: Subgraph structure in multi-robot path planning. J. Artif. Intell. Res. (JAIR) **31**, 497–542 (2008)
44. Schulte, C., Stuckey, P.J.: Speeding up constraint propagation. In: Principles and Practice of Constraint Programming, 10th International Conference (CP 2004), pp. 619–633, LNCS 3258 (2004)
45. Schulte, C.: The Gecode Team: Gecode – generic constraint development environment. http://www.gecode.org/. [accessed in September 2016] (2016)
46. Silva, J.P.M., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: Principles and Practice of Constraint Programming, 13th International Conference (CP 2007), pp. 483–497. Springer (2007)
47. Silver, D.: Cooperative pathfinding. In: Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117–122. AAAI Press (2005)
48. Soh, T., Banbara, M., Tamura, N.: A hybrid encoding of CSP to SAT integrating order and log encodings. In: Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015), pp. 421–428. IEEE Computer Society (2015)
49. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013). Elsevier
50. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
51. Standley, T.S., Korf, R.E.: Complete algorithms for cooperative pathfinding problems. In: Proceedings of Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 668-673. IJCAI/AAAI Press (2011)
52. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. IEEE Trans. Comput. Intell. AI Games **4**(2), 144–148 (2012)
53. Surynek, P.: Towards optimal cooperative path planning in hard setups through satisfiability solving. In: Proceedings of 12th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2012), pp. 564–576, LNCS 7458. Springer (2012)
54. Surynek, P.: On propositional encodings of cooperative path-finding. In: Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012), pp. 524–531. IEEE Computer Society (2012)
55. Surynek, P.: Mutex reasoning in cooperative pathfinding modeled as propositional satisfiability. In: Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), pp. 4326–4331. IEEE Computer Society (2013)
56. Surynek, P.: Solving abstract cooperative path-finding in densely populated environments. Comput. Intell. (COIN) **30**(2), 402-450 (2014). Wiley
57. Surynek, P.: Compact representations of cooperative path-finding as sat based on matchings in bipartite graphs. In: Proceedings of the 26th International Conference on Tools with Artificial Intelligence (ICTAI 2014), pp. 875–882. IEEE Computer Society (2014)
58. Surynek, P.: A simple approach to solving cooperative path-finding as propositional satisfiability works well. In: Proceedings of the 13th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2014), pp. 827–833, Lecture Notes in Computer Science, Vol. 8862, Springer (2014)
59. Surynek, P.: Simple direct propositional encoding of cooperative pathfinding simplified yet more. In: Proceedings of the 13th Mexican International Conference on Artificial Intelligence (MICAI 2014), LNCS, vol. 8857, pp. 410–425. Springer (2014)
60. Surynek, P.: On the complexity of optimal parallel cooperative path-finding. Fund. Inform. **137**(4), 517–548 (2015). IOS Press
61. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints **14**(2), 254–272 (2009)

62. Tanjo, T., Tamura, N., Banbara, M.: Azucar: A SAT-Based CSP solver using compact order encoding - (tool presentation). In: Proceedings of the Theory and Applications of Satisfiability Testing - 15th International Conference (SAT 2012), pp. 456–462. Springer (2012)
63. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Structures in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics (translated from Russian), pp. 115–125. Steklov Mathematical Institute (1968)
64. Wagner, G., Choset, H.: Subdimensional expansion for multirobot path planning. Artif. Intell. **219**, 1–24 (2015)
65. Walsh, T.: SAT v CSP. Principles and practice of constraint programming - CP 2000. In: 6th International Conference (CP 2000), pp. 441–456, LNCS 1894. Springer (2000)
66. Wehrle, M., Rintanen, J.: Planning as satisfiability with relaxed exist-step plans. In: Proceedings of the 20th Australian Joint Conference on Artificial Intelligence, pp. 244–253. Springer (2007)
67. West, D.B.: Introduction to graph theory. Prentice Hall (2000)
68. De Wilde, B., ter Mors, A., Witteveen, C.: Push and rotate: cooperative multi-agent path planning. In: Proceedings of the International conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), pp. 87–94. IFAAMAS (2013)
69. de Wilde, B., ter Mors, A.W., Witteveen, C.: Push and rotate: a complete multi-agent pathfinding algorithm. J. Artif. Intell. Res. (JAIR) **51**, 443–492 (2014). AAAI Press
70. Wilson, R.M.: Graph Puzzles, Homotopy, and the Alternating Group. J. Comb. Theory, Ser. B **16**, 86–96 (1974). Elsevier
71. Yu, J., LaValle, S.M.: Structure and intractability of optimal multi-robot path planning on graphs. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013). AAAI Press (2013)