On multirobot system operation plan formalization

Alexey S. Kruglikov
Ural Federal University
Yekaterinburg, Russia
a.s.kruglikov@urfu.ru

Sergey V. Kruglikov
UrFU, IMM UrO RAN
Yekaterinburg, Russia
svk@imm.uran.ru

Abstract

The paper considers a discrete event algorithm of information image forming in the process of exchanging structured data and coordinating the information spaces of participants. The task under consideration is inspired by a multirobot system that could be used for unmanned goods transportation. A formal definition of operation plan is introduced as a directed graph. Such definition is close to notions used in critical path analysis , but allows for higher level of abstraction in expressed relations. An algorithm is proposed to form a schedule from plan for automatic execution.

Keywords: multirobot system, multi-agent approach, operation plan, algorithm, schedule.

1 Introduction

The development of advanced technologies for cognitive robotics requires research into the mathematical foundations of a combination of autonomous, centralized, distributed and decentralized management processes, reflecting differences in the concepts of distributed artificial intelligence and artificial life. The development of effective algorithms for controlling the operation of intelligent robotic complexes in solving problems arising, in particular, during transport operations, monitoring and protection of objects and zones of special attention, is typical. A convenient means of formalizing meaningfully technical productions is a multi-agent approach that provides comparable opportunities for coordinated description of real and virtual agents, including intellectual ones. This feature allows us to consider the concept of a common information space of intellectual subsystems of a robotic complex based on methods of guaranteed control theory under uncertainty, as a mathematical model for tasks of interpretive navigation and autonomous management; coordination of actors' behavior [1]. Practical use of multirobot systems (MRS) implies coordination of jobs done by its subsystems and components. One of the ways to achieve it is based on explicit use of plan.

The paper considers a discrete event algorithm of information image forming in the process of exchanging structured data and coordinating the information spaces of participants. The task under consideration is inspired by the following MRS: several UAVs, e.g. quadcopters, used for aerial surveillance, and several SUVs. Such system could be used for unmanned goods transportation. A formal definition of operation plan is introduced as a directed graph [2]. Such definition is close to notions used in critical path analysis (CPA)[3], but allows for higher level of abstraction in expressed relations. Before work is commenced the operation plan is transformed into a schedule.

In: M.Yu. Filimonov, S.V. Kruglikov M.S. Blizorukova (eds.): Proceedings of the International Workshop on Information Technologies and Mathematical Modeling for Efficient Development of Arctic Zone (IT&MathAZ2018), Yekaterinburg, Russia, 19-21-April-2018, published at http://ceur-ws.org

CPA is hard to apply directly to planning multirobot system operation because of its peculiarities. Firstly the composition of MRS it subject to change with time as some robots break down and others coming from reserve. Secondly a goal once achieved cannot be assumed completed forever since, for example, after taking a prescribed position a quadcopter could be blown away by the wind, it could also leave position according to a new order. Thirdly time estimate for an action depends on both the type of executor and its state at the moment order acquisition. A slow robot located close to the prescribed position could be a preferable choice over quick one located far. And the last peculiarity to mention is a possibility of MRS repeated use for the same job on different terrains, which calls for unified planning with adaptation to terrain features.

For these reasons a two-phase planning process is proposed where both phases essentially implement a transformation of plan form from less detailed into more detailed. Three forms of plan are described as follows:

- **Plan template** describes general sense of a set of similar MRS operations. In general it is a procedure that forms operation plan according to the given terrain features. Its invocation is the essence of the first planning phase.

- **Operation plan** describes an idea of an operation on a terrain with specific features but is not tied to resources available at any moment. This allows for multiple subsequent operations on same terrain, for example if the initial resources proved insufficient and we try again after better preparation.

- **Schedule** is the most detailed form of plan with respect to environment and available resources. It is required to give an operation time estimate.

This paper addresses operation plan formalization and proposes an algorithm of its transformation into a schedule.

A plan is essentially a set of jobs harmonized by their goals, time, place and executors [4]. An executor could be a robot or a group of them; we assume any executor has a control system of its own, possibly including human operator. Operation plan only minds executor control method which could be one of the following three:

- **command-based control** implies that an executor can only perform actions from a pre-formed action set, which is characteristic for centralized control;

- **plan-based control** assumes an executor can execute not just an action but a plan of actions though it is unable to make a plan of its own which is often the case in decentralized control;

- **directive-based control** implies that an executor can on its own plan actions to achieve a set goal and carry them out which is characteristic for autonomous control.

Therefore, an operation plan can theoretically be used with an arbitrary choice of approach to control and even with combination of latter.

We assume that

- for any job there is a defined *goal*, i.e. formally described set of states each of which is called *target state*;

- several different executors are intended to *execute* schedule; their interoperation is the reason for plan existence;

- for any job there is an executor able to execute it.

2 Mathematical Model of Operation Plan

A plan is defined as a digraph $P = (J, L)$, where J is a set of jobs, L is a set of links, i.e. job dependencies. We will stick to the rule that arrow direction is away from the dependent job.

Set J is divided into subsets M, A and B to denote milestones, auxiliary and basic jobs correspondingly.

$$J = M \cup A \cup B.$$

We thus assume jobs can be of three different kinds, namely

- **Milestone** $m \in M$ is a fictive job not to be executed by an executor. It is used to represent a moment in time when a complex, composite goal is achieved. That means that a goal of a milestone is an intersection of several simpler goals.

- **Auxiliary job** $a \in A$ is executed to *sustain* a goal, i.e. at any moment in time an executor doing this job must be in one of target states. In other words, for an executor with this kind of job notions of target state and current state are same. Precision of goal sustainment is described as its cardinality as a set. Duration of auxiliary jobs is undefined; it should be calculated according to time dependence on other jobs.

- **Basic job** $b \in B$ is a job that is intended to *achieve* a goal and we assume that an executor will expend some time and resources to do that. Duration of such a job depends on the moment when an executor first achieves one of target states.

As operations plan describes purposeful action we assume that a main goal is defined, and it describes formally what is supposed to achieve. This goal corresponds to certain milestone in a plan called *main milestone* and denoted

$$m^* \in M.$$

Set of links is also divided into subsets according to kinds of job dependencies. We call them G, T, P, S to denote dependencies by goal, time, position and executor correspondingly.

$$L = G \cup T \cup P \cup S.$$

Dependency by goal is one the following situations:

1. When achieving a goal i.e. successful completion of a job is a condition for start of some other job.

2. When achieving a goal is decomposed into simultaneous achieving a set of subgoals. Such a goal is a milestone and is to be achieved by parallel completion of several jobs.

Conceptual difference between those situations can be discarded since we assume milestones to be fictive jobs, so we use a single link type for both variants. We will use arrows from depending job to the ones it depends on to model such link. It should be noted that dependency by goal expresses not just logical but also time dependence — the dependent job must not start before all the jobs it depends on are finished.

Thus we suppose that $G \subseteq \{(x, y)|, x, y \in J\}$.

Any basic job and milestone should drive the system to achievement of main goal which is formally expressed by a requirement that main milestone must be connected to any non-auxiliary job.

We use following notation for a path connecting x to y with arrows from subset Z

$$path_z(x, y) \equiv \exists j_0, \ldots, j_n \in J : \forall i \in \overline{1, n}\ (j_{i-1}, j_i) \in Z \wedge j_0 = x \wedge j_n = y.$$

to write down this requirement as

$$\forall j \in M \cup A\ (j \neq m^* \rightarrow path_G(m^*, j)).$$

No job can be its own condition, so we forbid existence of cycles in goals

$$\forall j \in J(\neg path_G(j, j)).$$

Lastly, any milestone must have at least one basic job as its condition

$$\forall m \in M\ \exists b \in B : (m, b) \in G.$$

It is practical to demand that no milestone was dependent directly on another milestone. Actually, such situation would simply express more than one layer of goal decomposition which is redundant. That is why

$$\forall x, y \in M\ (x, y) \notin G.$$

Once all requirements are met it is possible to partially order jobs in the sense of their logical sequence; it is enough to compare jobs by maximal path length from main milestone. It should be noted that such partial order is of limited use, and so we introduce additional dependencies. Since operation plan is unaware of present MRS resources it is impossible to define precise timing of execution. Thus we only use two kinds of **time dependencies:**

- Simultaneous job starts;
- Simultaneous job end.

More kinds of time links between jobs are imaginable, but they will appear in a schedule, on later stages of planning when jobs will be distributed among executors. Time links set is divided into two subsets for links of simultaneous start TStart and simultaneous end TEnd accordingly. We will denote time links with arrows between jobs. Milestones cannot be linked by time.

$$T \subseteq \{(x, y)|, x, y \in A \cup B\},$$

$$T = T_{Start} \cup T_{End}.$$

If an auxiliary job is time-linked to a basic one, we will only put an arrow from auxiliary to basic job; otherwise we put two counter-directed arrows between jobs. In essence this means that in a pair of auxiliary and basic

jobs time link distinguishes latter as determinative while in all other cases it can't be decided which job in a pair defines time.

$$\forall x, y \in A\ \forall T^* \in \{T_{Start}, T_{End}\}\ (x, y) \in T^* \to (y, x) \in T^*,$$
$$\forall x, y \in B\ \forall T^* \in \{T_{Start}, T_{End}\}\ (x, y) \in T^* \to (y, x) \in T^*,$$
$$\forall a \in A,\ b \in B\ (b, a) \notin T^*.$$

Any auxiliary job must be linked by start to exactly one basic job though maybe through other auxiliary jobs. The same is required for link by end, but it should be noted that auxiliary job could start with one basic job and end with another.

$$\forall a \in A,\ \forall T^* \in \{T_{Start},\ \exists! b \in B : path_{T^*}(a, b).$$

If finishing a job is required to start some other, then such jobs cannot be linked by time with simultaneous start or simultaneous ending.

$$\forall x, y \in B\ path_G(x, y) \to \neg path_T(x, y).$$

Dependency by position is a situation when geographical attributes of one job are defined by geographical attributes of another. To formalize this, we use an arrow from job whose attributes are dependent to the job they depend on. We will also put dependency-formalizing expression on the arrow. For example, assume job A has its goal to be a position inside an area which is the result of job B. Then we draw an arrow from A to B and mark it with an expression like *Inside(Goal.Position(A), Goal.Position(B))*.

$P \subseteq \{(x, y, f) | x, y \in J \wedge f \in F\}$, where F is a set of expressions.

Dependency by executor allows to formally describe situations when a job must be carried out by the executor who performed some known other job before. This is a direct prohibition of parallel execution of jobs; if two jobs are not bound by such dependency, we suppose their parallel execution by different robots normal. Most often such dependencies happen with jobs already bound by goal. Dependency by executor can be expressed with counter-directed arrows with executor name assigned to them. To that we add a requirement that all arrows incident to same job were marked with the same executor name. Note that since milestones are not actually jobs to be carried out by any distinct executor, they do not have dependencies by executor. Denote set of executor names as N and write down

$$S \subseteq \{(x, y, f) | x, y \in A \cup B, n \in N\},$$
$$\forall x, y \in J,\ n \in N\ (x, y, n) \in S \to (y, x, n) \in S,$$
$$\forall x, y, z \in J\ [(x, y, n_1) \in S \wedge (x, z, n_2) \in S \to n_1 = n_2].$$

Another way to express the same would be to require that each job had an executor name, maybe undefined. This is more convenient in programming implementation but even more practical way would be to use a dictionary with executor name as key and set of corresponding jobs as its value

3 Plan-to-schedule Conversion Algorithm

RTS ability to execute operation plan and amount of time it will take can only be determined with respect to available executors and their states.

Take an operation plan $P = (j, L)$, set $Sub = \{s_i\}$ of executors, set $States = \{state_i\}$ of their respective start states. Take set of jobs in a plan and append a special element denoting no job $J^* = (A \cup B) \cup \{none\}$, and a set of time intervals $T_\Delta = \{(t_{start}, t_{end}) | t_{start} \leq t_{end}\}$. Assume a planning function $f_t : Sub \times J^* \times J^* \to R^+$ is known, used to estimate time it takes an executor to perform a job. Actually, $f_t(s, j_1, j_2) = t$ means that an executor s who took job j_2 when being in some target state of job j_1, will finish j_2 expending approximately t of time. We assume that $j_2 \neq none$, and if $j_1 = none$, then the only target state of j_1 is the start state of an executor.

Our goal is to construct map $Scen : J \to Sub \times T_\Delta$ such that:

$$\forall j \in (A \cup B)\ \exists! (s, t) \in Sub \times T_\Delta : (s, t) = Scen(j),$$

$$\forall s \in Sub,\ \ \forall t_1, t_2 \in T_\Delta\ (\exists j_1, j_2 : (s, t_1) = Scen(j_1) \wedge (s, t_2) = Scen(j_2)) \to (t_1 \cap t_2 = \emptyset),$$

$$\forall j_1, j_2 \in (A \cup B) \quad \begin{cases} ((j_1, j_2) \in T_{start}) \to [Scen(j_1) = (s_1, (t, t_1)) \wedge Scen(j_2) = (s_2, (t, t_2))], \\ ((j_1, j_2) \in T_{end}) \to [Scen(j_1) = (s_1, (t_1, t)) \wedge Scen(j_2) = (s_2, (t_2, t))], \\ ((j_1, j_2) \in G) \to [Scen(j_1) = (s_1, (t_1, t_2)) \wedge Scen(j_2) = (s_2, (t_3, t_4)) \wedge t_3 \leq t_4], \\ ((j_1, j_2) \in S) \to [Scen(j_1) = (s_1, (t_1, t_2)) \wedge Scen(j_2) = (s, (t_3, t_4))]. \end{cases}$$

Essentially, these requirements mean that

- for any job except milestones an executor is found;
- no executor is busy with two jobs simultaneously;
- if two jobs are bound by time constraints, this dependency is supported by their start or end times;
- if two jobs are dependent by goal, the dependent job won't start before its prerequisite job ends;
- if two jobs are dependent by executor, same executor is assigned to both.

Schedule construction is done in two steps

(A1) Jobs are assigned to executors so that each job gets an executor, and stories of executors, i.e. sequences of jobs are formed.

(A2) Schedule is formed according to a set of stories with respect to dependencies by goal and time.

For **step A1** we first make a family of digraphs $G = \{G_i = (j^*, T_i)\}$ where G_i describes all possible jobs and their time estimates for executor number i. This is done using function f_t as follows: form all possible pairs of jobs $(j_1, j_2) : j_1 \in J^*$, $j_2 \in (A \cup B)$. A pair is checked for possible sequence, i.e. we evaluate a hypothesis that there exists an executor which could carry out j_2 after j_1. For each pair:

- if jobs are bound by time, pair is discarded;
- if jobs are bound by goal in inverse order, i.e. j_2 is prerequisite for j_1, pair is discarded;
- pair (j, j) is discarded;
- all possible executors are checked, and if executor s_i can carry out both jobs, then we add an arrow (j_2, j_1) into graph G_i with time estimate $f_t(s_i, j_1, j_2)$.

This is how we describe all possible distributions of jobs between executors with respect to their execution sequence and also all sequences forbidden by time and goal dependencies are filtered out. Any path $P = j_1, t_1, j_2, t_2, \ldots, j_n$ in graph G_i can be interpreted as a fragment of story for executor s_i. A story of executor s_i is defined as such path $P_i = j_1^i, t_1^i, j_2^i, t_2^i, \ldots, j_{n(i)}^i$ in graph G_i, that it ends with absence of job, i.e. $\forall i\ j_{n(i)}^i = none$. It should be noted that jobs in a path are put in reverse order — from last executed to first executed.

Denoting as J_i set of jobs in path P_i, we call a *solution* of first phase such set of stories that

- all jobs are distributed among executors;
- any job other than none is assigned exactly to one executor.

$$J = \bigcup j_i, \qquad \forall j \in (A \cup B)\ \exists! i : j \in J_i. \tag{$*$}$$

We call a *partial solution* one executor story, i.e. one path ending with job none.

$$P_{partial}^i = j_1, t_1, j_2, t_2, \ldots, j_n :\ j_n = none.$$

When a partial solution is found, in order to satisfy (*) we must exclude possibility of duplicate job distribution. It is enough to take set of jobs J_i in the partial solution and exclude all of its jobs from remaining graphs. A solution finding algorithm can be then built as graph path searching. For example, assume we are given a square area to be viewed when searching some object. It is divided into areas A, B, C, D (Fig. 1a). RTS has two identical executors S_1, S_2, able to move and view areas. We will examine two alternative start positions: first (Fig. 1b) assumes executors are located in areas A and B, while second (Fig. 1c) locates them in areas A and C.

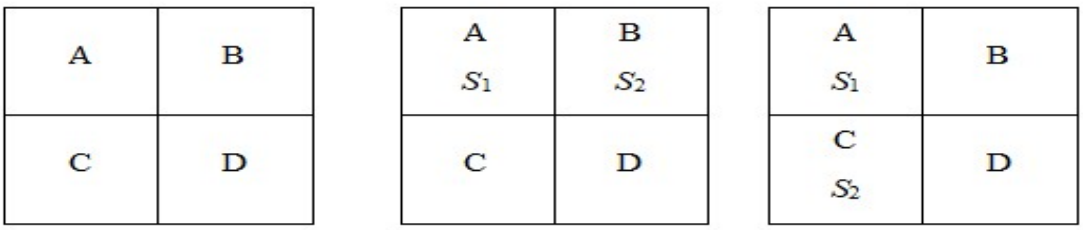

Figure 1: Examples: a. Area and its subareas; b. first alternative start position; c. second alternative start position.

Assume we have operation plan of eight jobs and a milestone (Fig. 2).

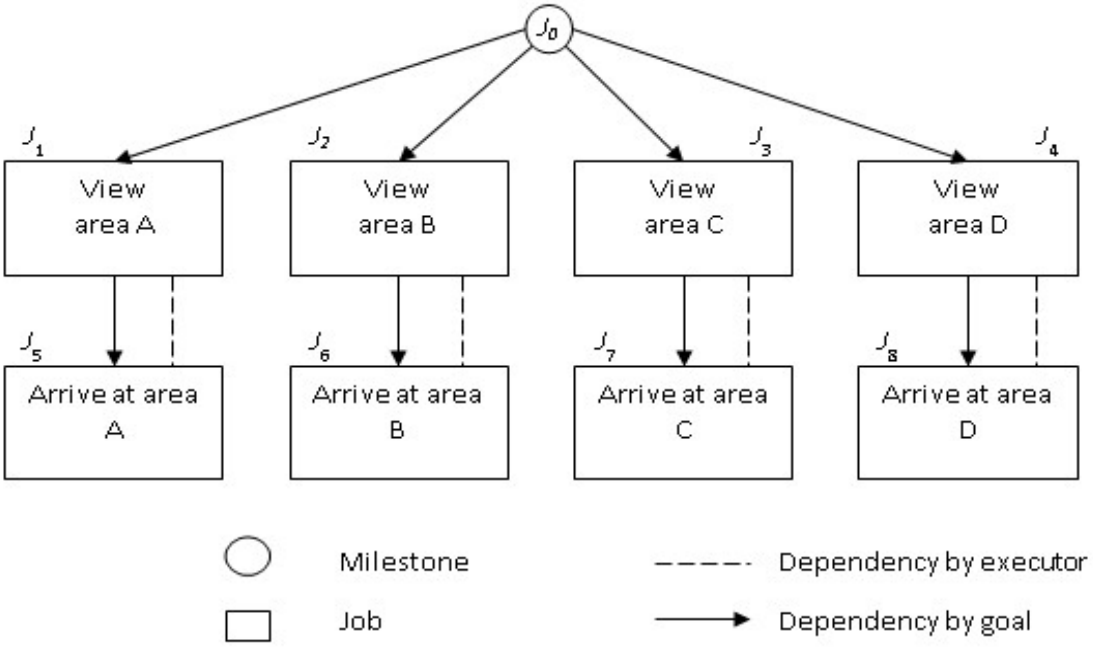

Figure 2. Plan

Graphs G_1 and G_2 are isomorphic since executors are identical but weights of arrows, i.e. time estimates are different because of different start positions. After algorithm is run executor stories are formed (Fig.3).

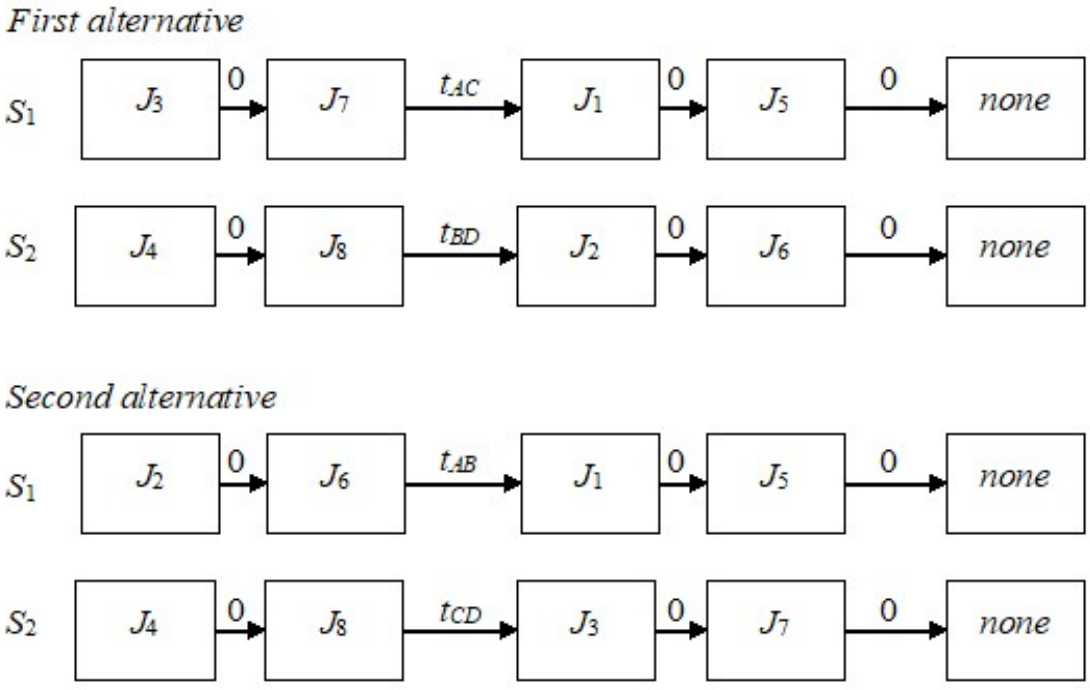

Figure 3. Executor stories

The essence of **step A2** is building a schedule on the base of story set if latter was constructed successfully. For that we must define a planned time interval for each job with respect to dependencies by goal and time.

First we introduce an auxiliary **algorithm of conflict node search** for executor s_i. It moves jobs one by one from the story into a scenario until it finds a job with a possible conflict. Possibility of conflict emerges when a job is dependent by time, and also when a job is a prerequisite for a job from another executor story.

To describe **conflict resolution algorithm**, it is convenient to use notion of *latest end moment* of job. It is a moment in time such that a job cannot end after it because of some restriction. Algorithm uses the following ideas

- job j_1, succeeding job j_2 in an executor story cannot end after the moment of j_2 start;
- if job j_1 is prerequisite for job j_2 from other executor story, then it cannot end after moment of j_2 start;
- jobs $j_1, \ldots, j_n$ bound by common end time cannot end after moment $t = \min[t_{i0}]$, where t_{i0} is latest end moment of job j_i;
- jobs $j_1, \ldots, j_n$ bound by common start time cannot start after moment $t = \min[t_{i0} - t_i]$, where t_{i0} is latest end moment of job j_i, while t_i is its time estimate;
- job bound by several restrictions must start at the earliest moment of those determined by restrictions.

Algorithm finds such jobs in a conflict set that their restrictions are all defined and moves them into schedule, and forms a list of executors whose jobs were removed from conflict set. It is possible to move a job into scenario when we have enough information to calculate its latest end moment and start moment. It should be noted that in case of a job bound by time from both sides i.e. bound by start time and also bound by end time, we knowingly change its time estimate to satisfy both conditions.

Schedule construction algorithm simply initializes a conflict set of jobs, and then iteratively resolves conflicts and finds new ones until all jobs in all histories are moved into schedule.

4 Conclusion

A formalization of operation plan based on graph theory is introduced, it is close to CPA notions. An algorithm is proposed to form a schedule from plan. A schedule can be carried out by MRS control system automatically.

Program implementation of plan model and schedule construction algorithm is done and tested on several examples.

References

[1] S.V.Kruglikov, A.S.Kruglikov. An A Priori Planning of Joint Motions for USV as a Problem of Guaranteed Control/Estimation *Applied Mechanics and Materials.* 494-495:1110-1113, 2014.

[2] B. Bollobas. *Modern Graph Theory.* Springer Science & Business Media, 2013.

[3] J.E. Kelley. Critical-path planning and scheduling - mathematical basis. *Operations research.* 9(3):296-320, 1961.

[4] M. Hadad, S. Kraus, I. Ben-Arroyo Hartman, A. Rosenfeld. Group planning with time constraints. *Annals of mathematics and artificial intelligence.* 69(3):243-291, 2013.